



# Shake 3



- Table of Contents
- Customizing Shake
- Useful Functions
- Hot Keys
- Index

## Apple Computer, Inc.

© 2003 Apple Computer, Inc. All rights reserved.

Under the copyright laws, this manual may not be copied, in whole or in part, without the written consent of Apple. Your rights to the software are governed by the accompanying software license agreement.

The Apple logo is a trademark of Apple Computer, Inc., registered in the U.S. and other countries. Use of the keyboard Apple logo (**Opt+Shift+K**) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

Every effort has been made to ensure that the information in this manual is accurate. Apple Computer, Inc. is not responsible for printing or clerical errors.

Apple Computer, Inc.

1 Infinite Loop

Cupertino, CA 95014-2084

408-996-1010

[www.apple.com](http://www.apple.com)

Apple, the Apple logo, Final Cut, Final Cut Pro, FireWire, Mac, Macintosh, Nothing Real, QuickTime, Shake, and TrueType are trademarks of Apple Computer, Inc., registered in the U.S. and other countries.

Adobe is a trademark of Adobe Systems Inc.

Cineon is a trademark of Eastman Kodak Company.

Maya, Alias, Alias/Wavefront, IRIX, and O2 are trademarks of SGI Inc.

3ds max is a trademark of Autodesk Inc.

Softimage and Matador are registered trademarks of Avid Technology, Inc.

Times is a registered trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Other company and product names mentioned herein are trademarks of their respective companies. Mention of third-party products is for informational purposes only and constitutes neither an endorsement nor a recommendation. Apple assumes no responsibility with regard to the performance or use of these products.



# Contents

## **1 Introduction 7**

- Chapter Summary 7
- The Shake 3 Reference Guide 7
- Learning Shake 3 8
- Platform Considerations and the Reference Guide 9
- What's New in Shake 3 11

## **2 The Shake User Interface 15**

- Chapter Summary 15
- The Shake User Interface 15
- Menus and the Title Bar 16
- Viewers 21
- Buttons 39
- The Parameters View 43
- The Domain of Definition (DOD) 50
- The Flipbook 57
- Creating a Disk-Based Flipbook 60
- Using a Broadcast Monitor 63
- The Time Bar 64
- General Window Interactivity 65

## **3 Basics of Building a Script 69**

- Chapter Summary 69
- The File Browser 69
- About Image Input and Output 75
- About Time 84

About Audio	89
Global Parameters	96
Working With High-Resolution Images	101
About Proxies	103

## **4 Compositing and the Node View 125**

Chapter Summary	125
About Node-Based Compositing	125
About Channels	129
About the Node View and the Tool Tabs	132
About the Infinite Workspace	150

## **5 Compositing: The Layer Nodes 153**

Chapter Summary	153
About Compositing	153
Compositing Math	154
Photoshop Layering Modes	156
The Layer Functions	161
Other Compositing Functions	197

## **6 File Formats and Footage 201**

Chapter Summary	201
File Formats	201
About Resolution	208
Functions Affecting Image Resolution	211
Working With Video	214
Video Functions	221
About Aspect Ratio and Non-Square Pixels	226
Cropping Functions	233

## **7 Keying 237**

Chapter Summary	237
About Keying and Spill Suppression	237
Edge Treatment	247
Keying DV Footage	251
Keying Functions	254

## **8 Color Correction: Part I 273**

- Chapter Summary 273
- Color Space 273
- Using the Color Picker 278
- Color-Correction Tools 285
- Atomic-Level Functions 288
- Utility Correctors 300
- Consolidated Color Correctors 316
- Color Correction With the Infinite Workspace 340

## **9 Color Correction: Part II 349**

- Chapter Summary 349
- The Logarithmic Cineon File 349
- Bit Depth 363
- About Premultiplication and Compositing 368

## **10 Using Masks 383**

- Chapter Summary 383
- About Masks 383
- Masking an Effect 384
- Masking a Layer 391
- Masking Filters 394
- Masking Using the Constraint Node 397

## **11 Tracking 401**

- Chapter Summary 401
- About Tracking, Stabilizing, and MatchMoving 401
- How a Tracker Works 412
- Tracking Strategies 413
- Modifying Tracks 418
- Saving Tracks 423
- Tracking Functions 424

## **12 Transforms, Motion Blur, and Warping 435**

- Chapter Summary 435
- About Transformations 435

	Concatenation of Transformations	436
	Inverting Transformations	436
	Onscreen Controls	437
	Setting Output Resolution and Scaling Images	442
	About Motion Blur	445
	Applying a Smear Effect	448
	The Transform Functions	450
	About Warps	468
	The Warp Functions	468
<b>13</b>	<b>Using the Curve Editor</b>	<b>483</b>
	Chapter Summary	483
	About the Curve Editor	483
	Navigating the Curve Editor	485
	Working With Keyframes	488
	About Splines	502
<b>14</b>	<b>Time View</b>	<b>511</b>
	Chapter Summary	511
	About the Time View	511
	The Transition Function	520
<b>15</b>	<b>Painting, Rotoscoping, and Other Image Functions</b>	<b>527</b>
	Chapter Summary	527
	Using the QuickPaint Node	527
	Using the RotoShape Node	542
	Other Image Functions	555
	Using the QuickShape Node	562
<b>16</b>	<b>Filters</b>	<b>581</b>
	Chapter Summary	581
	About Filters	581
	Masking Filters	581
	The Filter Functions	583
<b>17</b>	<b>Customizing Shake</b>	<b>623</b>
	Chapter Summary	623

Setting Preferences and Customizing Shake	623
Locations for .h Files and Custom Icons	624
General Settings	626
The Curve Editor and Time Bar	640
File Path and Browser Controls	641
Function Tabs	645
Node View	649
Parameters Tab	649
Viewers	658
Template Preference Files	664
Environment Variables for Shake	664
Interface Devices and Styles	671
<b>18 Macros, Expressions, and Scripting</b>	<b>673</b>
Chapter Summary	673
Making Macros	673
Variables, Linking, and Expressions	681
Functions, Variables, and Expressions	684
Script Manual	692
<b>19 Rendering</b>	<b>725</b>
Chapter Summary	725
The Render Parameters Window	725
The Render File Menu	727
The FileOut Function	727
<b>Appendix A</b>	
<b>Keyboard Shortcuts and Hot Keys</b>	<b>729</b>
Hot Key List	729
<b>Appendix B</b>	
<b>The Command-Line Manual</b>	<b>737</b>
Viewing, Converting, and Writing Images	738
<b>Index</b>	<b>757</b>



# Introduction

## Chapter Summary

- The Shake 3 Reference Guide
- Learning Shake 3
- Platform Considerations and the Reference Guide
- What's New in Shake 3

## The Shake 3 Reference Guide

Shake is a node-based compositing and visual effects application for compositing or modifying clips of images. Shake includes two industry-standard keys for pulling blue screens and green screens, a complete suite of color-correction tools, tracking/stabilization capabilities, integrated procedural paint, rotoscoping tools, support for third-party plug-ins, and compatibility across Mac OS X, Linux, and IRIX platforms.

Shake is also an image-processing tool to be used as glue between different image-oriented applications or, for large facilities, different departments, such as film recording/output, 3D technical directors, and compositors. It can be easily used in the interface, in a Terminal, or in scripts created by more technically-oriented users.

Because Shake's scripting language is similar to C, it is also an extensive image-manipulation library to be used by programmers in custom applications.

The Shake Reference Guide chapters are divided by Shake's major features and functionality. Functions are discussed in their relative chapter sections. For example, the *DeInterlace* and *SwapField* functions are covered in the "Working With Video" section of Chapter 6, "File Formats and Footage." As another example, although the nodes *Fit*, *Resize*, and *Zoom* are located in the Transform tab, the nodes are discussed in the "Functions Affecting Image Resolution" section of Chapter 6, "File Formats and Footage," rather than in the more general "Transforms, Motion Blur, and Warping" chapter (Chapter 12).

There is also limited online help that includes information on customizing Shake and functions.

## Learning Shake 3

The available Shake documentation includes the *Shake 3 Reference Guide* (print and PDF versions), *Shake 3 Tutorials* (print and PDF versions), *Shake 3 Installation Guide*, the *Shake 3 Read Me*, and limited online help. The online help includes information on customizing Shake and function information (organized by name and by tool tab). The *Shake 3 Late-Breaking News* contains last-minute release information.

If you are new to Shake, you are encouraged to go through “Lesson One: Basic Tutorial” and “Lesson Two: Intermediate Tutorial” in the *Shake 3 Tutorials* for an introduction to Shake functionality and workflow (or, at least say you did when you call in for support). To understand specific topics, do lessons three through nine (the advanced tutorials).

To view the available documentation, do one of the following:

- To view the PDF version of the *Shake 3 Reference Guide*, choose Help > Reference Guide.

**Note:** You can also open the PDF version of the reference guide from the *Shake3/doc* folder.

- To view the PDF version of the *Shake 3 Tutorials*, choose Help > Tutorials.

**Note:** You can also open the PDF version of the tutorials from the *Shake3/doc* folder.

- To view the online help, choose Help > Customizing Shake.

### Viewing Shake PDF Documentation

The Shake online reference guide and tutorials are in Adobe® Portable Document Format. To view the documentation, please use the following guidelines.

#### Macintosh OS X Systems

On a Macintosh OS X system, the Shake PDF documentation opens in your specified PDF reader. To take advantage of the links and cross references in the PDF, use Adobe® Acrobat® Reader® to view the documentation.

To view the documentation, choose Help and select the Reference Guide or Tutorials.

#### Linux and IRIX Systems

To view the Shake online documentation on a Linux or IRIX system, you need to configure your PDF browser path in the Shake application.

**Note:** Adobe Acrobat Reader needs to be installed on your system.

To configure the PDF browser path:

- Click the Globals tab.
- Expand the guiSettings parameter (click the + [plus] sign).
- Click the folder icon next to pdfBrowserPath.
- In the Choose Application window, browse to the Adobe Acrobat Reader application.

To view the documentation, choose Help and select the Reference Guide or Tutorials.

To save your PDF browser settings in Shake:

- Choose File > Save Interface Settings.

In the “Save preferences to” window, save your settings to a *defaultui.b* file.

For information on Shake training, resources, and “Tips and Tricks,” visit the Apple Web site at <http://www.apple.com/shake>. There is an additional listserver, archive, and extensive macro collection at [www.highend2d.com/shake](http://www.highend2d.com/shake), the unofficial Shake user community site.

### Platform Considerations and the Reference Guide

Shake can be used on the Macintosh OS X, Linux, and IRIX platforms. Functions or commands that are platform-specific are documented whenever possible.

**Keyboard:** Hot keys or keyboard commands that vary between the Macintosh and Linux/IRIX platforms are documented when possible. In most cases, the **Command** and **Ctrl** keys are interchangeable. The Macintosh **Delete** key (the key located below **F12**) is the equivalent of the Linux/IRIX **Backspace** key; the Macintosh **Del** key (grouped with **Home**, **Page Up**, **Page Down**, etc.) is the Linux/IRIX **Delete** key.

**Mouse:** A three-button mouse is required to use Shake. A three-button mouse provides quick access to contextual menus and navigational shortcuts. Shake also supports the middle scroll wheel of a three-button mouse.

The documentation refers to the three mouse buttons as follows:

Mouse Button	Documentation Reference
Left-mouse button	Click
Middle-mouse button	Middle-mouse button
Right-mouse button	Right-click

**Note:** This reference guide uses “right-click” to specify contextual menu commands and navigational shortcuts.

The following table lists the Shake Reference Guide notation system.

Notation	Example
Hot keys/keyboard commands appear in bold.	To break a tangent handle in the Curve Editor, <b>Ctrl</b> -click the handle.
Some hot keys/keyboard commands vary depending on the platform. The Macintosh command appears first, followed by the Linux/IRIX command. The two hot keys/commands are separated by a forward slash.  In general, the <b>Command</b> and <b>Ctrl</b> keys are interchangeable.	In the Node View, you can press <b>Ctrl+Opt</b> -click / <b>Ctrl+Alt</b> -click to zoom in and out.
Menu selections are indicated by dividing angle brackets.	To open a script, choose File > Open Script.
File paths and filenames appear in italics. Also, directories and file paths are divided by forward slashes.	In <i>Shake3/doc/pix/vp</i> , load the <i>vanilla1.sbk</i> script. Or, load the script in <i>Shake3/doc/pix/vp/vanilla1.sbk</i> .

Notation	Example
Node groups (Tool Tabs) appear in the default font, followed by the name of the node in italics. A dash appears between the tab and node names.	In the Node View, select the <i>Cloud</i> node, and insert a Transform- <i>CornerPin</i> node.
Command-line functions appear in italics.	<i>shake -exec my_script -t 1-240</i>
Modifications to preferences files appear in italics.	Add the following lines to a .h file in your startup directory: <i>script.cineonTopDown = 1;</i> <i>script.tiffTopDown = 1;</i>

## What's New in Shake 3

This section discusses the new functionality in Shake 3.

- *PixelAnalyzer* Node

The *PixelAnalyzer* node allows you to draw one or more boxes on an image and then calculate the average, minimum, and maximum values inside the box over a series of frames.

- *FilmGrain* Node

With improved grain algorithms, the *FilmGrain* node allows you to apply grain from selected film stocks, or to sample grain from an image.

- *MultiLayer* Node

The *MultiLayer* node allows you to input unlimited layers. Each layer has unique channel, opacity, and compositing operation settings.

- Photoshop Image Support

Adobe Photoshop files can be imported into Shake. When a Photoshop file is imported with a *FileIn*, you can read in a merged image, or select an individual layer. These files can also be imported as a script using the File menu. Each layer then becomes a unique *FileIn* fed into a *MultiLayer* node.

- *Select* Node

The *Select* node now supports any number of inputs.

- Opening a Script with a Missing Macro

If you open or load a Shake script that contains macros that you do not have on your system, you have the option to load the script using a substitute node, or to not load the script at all. These controls are located in the Globals tab, in the guiSettings subtree.

- DPX File Format Support

The DPX file format is now supported.

- **YUV Enhancements**

10-bit YUV is now supported. On YUV import, there is a “decodeAsFloat” toggle in the *FileIn* parameters to enable floating-point math to avoid clipping. This allows YUV colors outside the 0-1 range to be imported.
- **Broadcast Monitor Support**

On a Macintosh OS X system, you can preview your work on a broadcast video monitor.
- **New *RotoShape* Behavior**
  - **Closing Splines:** To close a spline, click the first point of the current spline.
  - **Drawing Splines:** When drawing a spline, click the point down to create a linear point; click and drag to draw the tangents out.
  - **Tangent Modification:** Tangents can be stretched without affecting the opposite tangent control. To affect both tangents, press **Shift** and drag on a tangent.
  - **Attach a Tracker:** When you right-click a shape’s transform control, the option to attach a preexisting tracker appears.
  - **Duplication of a Shape:** When you right-click a shape, the option to duplicate the shape appears.
- **New *QuickPaint* Behavior**
  - **Changing Brush Types:** The *QuickPaint* Edit subtab contains brush mode buttons (in the Tool row) to change the current brush stroke between the different brush modes.
  - **Converting a Stroke Type:** The *QuickPaint* Edit subtab contains a Convert Current Stroke button that allows you to convert a paint stroke(s) to different modes. For example, you can convert a Frame mode stroke to a Persistent mode stroke.
  - **Attach a Tracker to a Stroke:** When you right-click a paint stroke, the option to attach a preexisting tracker appears.
  - **Straight Lines:** To draw a straight line, press **Shift** when creating the paint stroke.
- **Tracking Enhancements**
  - **Track Preprocessing:** The preprocessing blurs the tracking area to help reduce inaccuracies from noise.
  - **Two-Point Scaling and Rotation Curve Baking:** You can now bake your curves into a single rotation or scaling curve.
  - **Window Pans to Display Tracker:** A zoomed-in plate pans in the Viewer as a tracker moves outside of the Viewer window.
- **Audio Support**
  - You can load AIFF and WAV files into Shake via the audio panel. Several files can be mixed down to create a single file, with several controls. The audio waveforms can be displayed inside the Curve Editor. Sound playback can be activated in the Time Bar controls (Macintosh OS X only).

**Note:** Because audio playback is handled through the use of Macintosh-specific QuickTime libraries, you can only hear audio playback on Macintosh OS X systems. You can still analyze and visualize audio in Linux and IRIX.

- Curve Editor Enhancements
  - Speed/Flicker Improvements
  - Manipulator Box: There is now a transform box to move and scale your points. The box persists if you hold **B** while drawing.
  - Overlapping Keys Controls: When moving several keyframes left or right, there are controls to determine the behavior of the keyframes when they collide with other keyframes. They are either Bounded by the stationary points, the moving points Interleave with the stationary points, the moving points Push the stationary points, or the moving points Replace the stationary points.
  - Key/Value Text Fields: If multiple points are selected, you can enter a Key or Value for all points in the group. If multiple points are selected, Mult appears in the text field to indicate multiple keys are selected. After you enter a value, you can use the virtual slider in the Value text field to raise or lower your points.
  - Parameters List Acts as Sliders: The list of curves now acts as duplicates of the Parameters list. For example, you can adjust the value in the text field, automatically entering a key. Expressions open below the curve entry, allowing for more editing room.
  - Copy/Paste of Keyframes: You can copy and paste keyframes on a curve. The keyframes are pasted at the frame that the cursor is positioned.
  - Split Editor Windows: To split the Editor into two windows, click on the top of the Curve Editor and drag it down. Each window is sensitive to its own visibility toggles.
  - Source/Target Color Pickers: There are Source/Target color pickers for nodes with embedded Curve Editors (*Lookup*, for example). These allow you to pick color from the screen to set your key points. Only the visible curves receive keys.
  - Resample Function/Expression Baking: You can resample a curve using the Curve Function list (smooth, jitter, etc.). Enter a time range you want to resample over, with a step to represent the keys. For example, 1-100x10 creates keys every 10 frames, removing previously existing keys. This can also be used to bake expressions.
- Curve Editor Hot Keys
  - **Q** – Moves selected points.
  - **W** – Scales points. This is a two-step process. You first click on the scale center, and then a second point to pull toward or away from the first point you clicked.
  - **X/Y** – Allow movement on only the X or Y axis. Press the hot keys again to free that axis.
  - **H** – Horizontal tangents. This flattens the tangents of a Hermite curve.
  - **V** – Toggles visibility of the selected curves.
  - **K** – Inserts a key on the current cursor position on the curve.

- Node Groups and Clusters
  - Groups have been entirely redone in the Node View. You can now edit open groups and link them to nodes outside of the group. You can also consolidate two groups together (**Shift+G**). Additionally, if you expand a group, it draws a background for that set of nodes, allowing you to color-code areas of your Node View. These clusters can also have a limited set of exposed parameters, so you can easily tweak several nodes simultaneously without having to jump between the nodes.
- Node View Improvements
  - Grid Alignment: Grid alignment is controlled in the Globals tab in the guiSettings parameters (gridEnabled), or in the right-click menu in the Node View (Snap to Grid). Nodes snap to a grid when enabled.
  - Layout Tightness: The layoutTightness parameter sets how closely the nodes are placed to each other in the Node View. This is controlled in the Globals tab in the guiSettings.
  - Squeezing Nodes: To vertically stack nodes together, press **Shift+L**.
  - Focus: To frame the selected nodes but maintain the zoom level, press **Shift+F**.
- Disk-Based Playback
  - On Macintosh OS X systems, you have an additional rendering option to render to a temporary QuickTime file. This allows you to play extremely long clips. Once this is rendered, you can save the clip to disk using the QuickTime built-in features.
- Color Picker Enhancement
  - Right-click on a Color Picker to access the Pop-Up Color Palette.
- Recover Script
  - To recall the last autoSave script, select File > Recover Script (**Command+Shift+O** / **Ctrl+Shift+O**). When starting Shake, you can also type *shake -recover* to automatically launch the last autoSave.
- Long Lists Support
  - Long lists now have a new scrolling widget. This is particularly helpful for selecting fonts.

# The Shake User Interface

## Chapter Summary

- The Shake User Interface
- Menus and the Title Bar
- Viewers
- Buttons
- The Parameters View
- The Domain of Definition (DOD)
- The Flipbook
- Creating a Disk-Based Flipbook
- Using a Broadcast Monitor
- The Time Bar
- General Window Interactivity

## The Shake User Interface

The Shake user interface is divided into four main areas: The Viewer, the Tool Tabs, the Parameters/Globals, and the Node View (which can also display the Curve Editor, Color Picker, or Pixel Analyzer).

- The Viewer(s) displays your images and allows you to view those images in different ways. (For example, you can view just the red channel of an image.)
- The Tool Tabs contain groups of nodes by function type. Nodes selected in the tabs are added to the node tree. For example, to add a *Keylight* node, click the Key tool tab, and click *Keylight*. The *Keylight* node appears in the node tree.

The Tool Tabs can also display the Curve Editor, Node View, or Time View.

- The parameters and Globals display all of the parameters associated with a selected node and the parameters that affect the behavior of the entire script (such as proxy information), respectively.
- The Node View displays the network of linked functions that comprise the node tree. Every effect in Shake is a distinct node that can be inserted into a node tree. Use the Node View to select, view, navigate, and organize your composite.  
For more information on the Node View, see Chapter 4, “Compositing and the Node View,” on page 125.

To expand any area of the interface, position the cursor in the quadrant and press the **Space bar**. For example:

- In the Tool Tab, click the Curve Editor tab to show the Curve Editor.
- Position the cursor in the Curve Editor and press the **Space bar**.  
The Curve Editor window is expanded to the full size of the interface.
- Press the **Space bar** again to reset the window.

To resize any of the four main windows, position the cursor over a quadrant border, and then click and drag on the border.

## Menus and the Title Bar

This section discusses the Shake title bar and the Shake, File, Edit, Tools, Viewers, Render, and Help menus.

### Title Bar Information

The title bar of the full Shake window displays current version data, the current script name, and the current proxy resolution.

### Shake Menu (Mac OS X Only)

The following table shows the Shake menu options. The Shake menu only appears in the Macintosh version of Shake.

Function	Notes
About Shake	Displays the Shake version number and copyright information.
Services	Services provide a quick way to perform tasks with several applications.
Hide Shake ( <b>Command+H</b> )	Hides Shake. To show Shake again, click the Shake icon in the Dock.

Function	Notes
Hide Others ( <b>Opt+Command+H</b> )	Hides all running applications other than Shake. To show the applications again, select Show All from the Shake menu.
Quit Shake	Quits the Shake application.

## File Menu


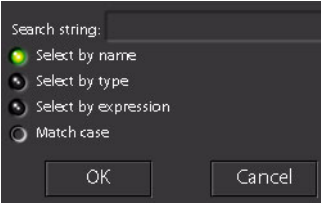
The following table shows the File menu options.

Function	Notes
New Script ( <b>Command+N</b> / <b>Ctrl+N</b> )	Deletes all nodes currently in the Node View. (You can also press <b>Command+A</b> / <b>Ctrl+A</b> in the Node View to select all nodes and then press <b>Del.</b> )
Add Script	Opens the Load Script window. Adds a second set of nodes to those currently in the Node View. The added nodes are renamed if a naming conflict arises. (For example, <i>FileIn1</i> becomes <i>FileIn2</i> if <i>FileIn1</i> already exists.) Global settings are taken from the added script, as is the new script name.
Open Script ( <b>Command+O</b> / <b>Ctrl+O</b> )	Opens the Load Script window. The script selected in the Browser replaces what is already in the Node View. You can also use the Load button in the title bar.
Import Photoshop File	Imports a Photoshop file. If the Photoshop file contains multiple layers, you can import the layers as separate <i>FileIns</i> that are fed into a <i>MultiLayer</i> node, or as a single, composited image by using a normal <i>FileIn</i> node.
Reload Script	Reloads the script listed in the top of the title bar.
Save Script ( <b>Command+S</b> / <b>Ctrl+S</b> )	Saves the script without prompting you for a script name (if you have already saved). You can also use the Save button on the title bar.
Save Script As ( <b>Shift+Command+S</b> / <b>Shift+Ctrl+S</b> )	Opens the Save Script window. Enter the new script name, and click OK to save the script.

Function	Notes
Recover Script ( <b>Shift+Command+O</b> / <b>Shift+Ctrl+O</b> )	<p>Loads the last autoSave script and is usually done when the user has forgotten to save a script and quits Shake, or when Shake has unexpectedly quit. The script is found under <i>\$HOME/nreal/autoSave</i>. (The <i>\$HOME</i> directory is your personal Home directory, for example, the <i>/Users/jobn</i> directory.)</p> <p>If you have environment variables set, you can launch Shake on the command line with the same option using <i>-recover</i>:</p> <p><i>shake -recover</i></p> <p>For more information on environment variables, see Chapter 17, "Customizing Shake."</p>
Save Interface Settings	<p>Opens the Save Preferences To window. Saves your window layout to a file in your <i>&lt;UserDirectory&gt;/settings</i> file. If you call it <i>defaultui.b</i>, it is automatically read next time you launch Shake. You can save the settings file anywhere, but it is not read automatically unless the file is in the settings directory.</p>
Load Interface Settings	<p>Opens the Load Preferences From window. Select an interface settings file from disk, and click OK to load the file.</p>
Flush Cache	<p>Pushes the images in the memory cache out to the disk cache. Similar to what is done upon exit.</p>
Purge Memory Cache	<p>Purge cache is similar to Flush cache, but then continues to free the allocated memory. Useful if you have most of your RAM filled and want to run a flipbook without actually exiting Shake. The delay when you exit is Shake flushing the memory cache to disk. The cacheMode controls whether or not images in the cache are used (regardless of whether they are coming from the disk or memory).</p>
Exit (Linux and IRIX only)	<p>Exits the program. No kidding. You can also use the standard OS exit buttons in the upper corner of the interface.</p>

### Edit Menu

The following table shows the Edit menu options.

Function	Notes
Undo ( <b>Command+Z</b> / <b>Ctrl +Z</b> )	Undo previous commands; up to 100 levels of undo. Layout, viewing, and parameter changes are saved in the Undo list. You can also click Undo/Redo  . (You can change the amount of levels in your <i>ui.b</i> file. See "Menus and the Title Bar" on page 16 for more information.)
Redo ( <b>Command+Y</b> / <b>Ctrl+Y</b> )	If you do an Undo and you have not changed anything, click Redo to go back to your previous settings.
Find Nodes ( <b>Command+F</b> / <b>Ctrl+F</b> )	Opens the Select Nodes by Name window that allows you to dynamically select nodes that match your criteria in the search string.  <ul style="list-style-type: none"><li>■ Select by name. Nodes that match the search string are immediately activated. For example, if you enter <i>f</i>, <i>FileIn1</i> and <i>Fade1</i> are selected. If you enter <i>fi</i>, just the <i>FileIn1</i> is selected.</li><li>■ Select by type. Select nodes by type. For example, enter <i>Move</i>, and all <i>Move2D</i> and <i>Move3D</i> nodes are selected.</li><li>■ Select by expression. Allows you to enter an expression. For example, to find all nodes with an angle parameter greater than 180, enter: <i>angle&gt;180</i></li><li>■ Match case. Sets case sensitivity.</li></ul>

### Tools Menu

The Tools menu provides a menu listing for each of the nodes in the Tool Tabs (for example, Image, Color, Filter, etc.). You can also right-click a tab to display the tools list.

## Viewers Menu

The following table shows the Viewers menu options.

Function	Notes
Create Viewer	Creates a new Viewer in the Viewer area, and automatically stretches it to fill the Viewer area. While in a Viewer, you can also right-click and select New Viewer, or press <b>N</b> .
Spawn Viewer Desktop	Launches a floating Viewer window that can be moved independently of the interface. The Viewer Desktop is ideal for dual-monitor setups.

## Render Menu

The following table shows the Render menu options. For more information on Rendering, see Chapter 19, "Rendering," on page 725.

Function	Notes
Render Flipbook	Renders a flipbook of the current Viewer. Opens the Flipbook Render Parameters window, which allows you to override the Global parameters (if necessary). To cancel the render, press <b>Esc</b> when in the Flipbook window.
Render Disk Flipbook	Macintosh OS X only. Launches a disk-based Flipbook into QuickTime. This has several advantages over normal Flipbooks. It allows for extremely long clips, allows you to attach audio (loaded with the audio tab in the main interface), and you can write out the sequence as a QuickTime file after viewing, bypassing the need to render the sequence again. For more information, see "Creating a Disk-Based Flipbook" on page 60.
Render FileOut Nodes	Renders <i>FileOut</i> nodes in the Node View. In the Node View, press <b>F</b> to frame all active nodes. You have the option to render only the active <i>FileOut</i> nodes, or all <i>FileOut</i> nodes.
Render Proxies	Renders your proxy files for your <i>FileIn</i> nodes, and leaves your <i>FileOuts</i> untouched. For more information on proxies, see "About Proxies" on page 103.

## Help Menu

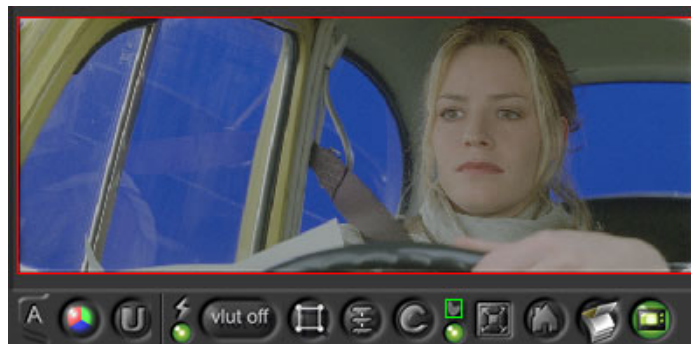
The following table shows the Help menu options.

Function	Notes
Reference Guide	Opens the PDF version of the <i>Shake 3 Reference Guide</i> . The documentation is located in <i>Shake3/doc</i> folder.
Tutorials	Opens the PDF version of the <i>Shake 3 Tutorials</i> . The documentation is usually located in <i>Shake3/doc</i> folder.
Customizing Shake	Opens HTML help that includes information on customizing Shake and function information (organized by name and by tool tab).
Getting Help	Provides instructions for obtaining help with Shake issues.
Late-Breaking News	Opens the latest release notes for the current version of Shake.
Shake Home Page	Opens a browser with <a href="http://www.apple.com/shake">www.apple.com/shake</a> .

## Viewers

Use the Viewers to display your images and control how you see those images. For example, you can view the individual channels of an image. Other tools such as the Histogram or PlotScanline also appear in the Viewers, as well as onscreen controls that allow you to transform or animate images.

The following images from “The Saint” are provided courtesy of Framestore CFC and Paramount British Pictures Ltd.



You can create as many Viewers as you want, and each Viewer updates dynamically (in any channel you want from any node you select). So, you can view the results downstream of a node you modify upstream. For example, you can refine a key in one Viewer while viewing the composite in another.

Each Viewer has a set of two buffers to compare images, or you can create multiple Viewers in the Viewer area.

**To create an additional Viewer:**

- 1 Position the cursor in the Viewer1 area.
- 2 Press **N**, or right-click and select New Viewer.

**Note:** You can also choose Viewers > New Viewer to create a new Viewer.

A new Viewer called “Viewer2” is created above Viewer1.

To select a Viewer, double-click the Viewer window. The title bar is highlighted when the Viewer is selected and the currently viewed node is displayed. Selected nodes are displayed in the active Viewer.

If a Viewer moves behind another Viewer, do one of the following:

- Click the Iconify Viewer button.



Iconify Viewer button

The Viewer is minimized.

- Press the **Space bar**.

The Viewer window expands. Press the **Space bar** again to reset the windows.

**Note:** If you get strange Viewer behavior, delete the Viewer (right-click and select **Delete Viewer**), and create a new Viewer.

The Viewers use memory, so close your higher-resolution Viewers when rendering. Also, the more Viewers open, the slower the display rate.

**To create a separate Viewer workspace for use on a second monitor:**

- Choose Viewers > Spawn Viewer Desktop. The second monitor must be run from the same graphics card as the primary monitor.

To help prevent accidentally rendering an enormous image (for example, you enter 200 into your zoom parameter instead of 20), the Viewer’s resolution is limited to 4K. This limit is configurable. The Viewer resolution has no effect on rendered images—it only crops the view in the GUI to the set resolution.

**Looking at Images in a Viewer**

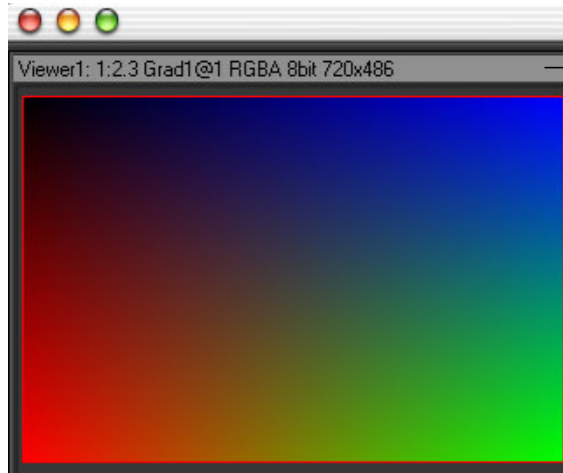
To load a node into the current Viewer, click the button on the left side of the node. Double-click the button to load the node’s parameters in the Parameters tab.

In the following image, the *Grad* node is loaded into Viewer 1, buffer A.



Click once to display node in Viewer.  
Double-click to load node parameters.

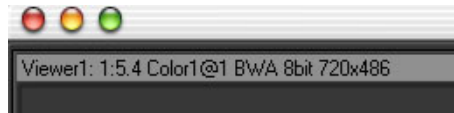
The information (node name, channels, bit depth, size, etc.) for the selected node appears in the Viewer title bar. When the *Grad* node is loaded into the Viewer, the following appears in the Viewer title bar.



**Note:** The channels displayed in the Viewer are the non-zero channels. Non-zero channels are not the same as active channels. For example, a *Color* node set to black (black and white values are zero) displays the alpha channel (A) in the Viewer title bar:



When the Color node is adjusted toward grey (the black and white values are no longer zero), the black, white, and alpha channels (BWA) are displayed:



Click and scrub with the mouse in the Viewer to print the X, Y, R, G, B, and alpha values in the Viewer title bar. These values are also displayed in the help window in the bottom-right corner of the interface.


The Pixel Analyzer and Color Picker windows can help you analyze this data.

**Note:** To display the values in the Terminal window that launched Shake, press **Ctrl** and scrub in the Viewer.

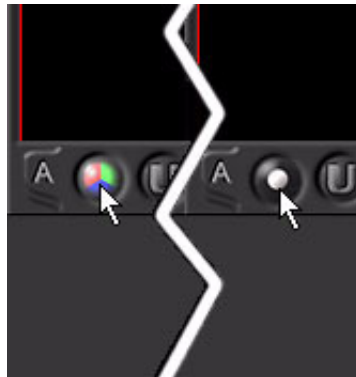
### The Viewer Buttons

Many Viewer buttons correspond to a right-click (contextual) menu item with the same function. You can also select the setting from the menu.

There are different click and hold button behaviors, for example:

- Click the Viewer Channel button  in the Viewer to toggle between the RGB and the Alpha Views.
- Click and hold the Viewer Channel button to select a specific channel view.

#### Click:



#### Click and hold:











You can override the default channel display progression when the Viewer Channel button is clicked. For example, clicking the button in RGB View displays the Alpha View. When you click again, the RGB View is displayed.




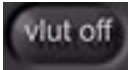





### To go from RGB to Red to alpha:










- 1 Go to RGB View, **Command**-click / **Ctrl**-click and hold the Viewer Channel button, and select the Red View button.
- 2 **Command**-click / **Ctrl**-click and hold the Viewer Channel button, and select Alpha View.  
When you click the Viewer Channel button, you now toggle through RGB, Red, alpha, and back to RGB.
- 3 To save this behavior, choose File > Save Interface Settings.





**Note:** You can cycle through some Viewer functions using number keys. Press **2** to cycle forward through the channel views, and press **Shift+2** to toggle backward through the channel views. Press **1** to toggle between the A and B compare buffers.

The following table shows the Viewer buttons, the keyboard or hot key shortcuts, and describes the button functions.

Button	Shortcut	Button Name and Function
		Cursor—Click and drag the cursor in the Viewer to display the X and Y position values, and the RGBA color values in the Viewer title bar. The values are also displayed in the help window.
		Iconify Viewer—Stows the current Viewer.
	<b>Ctrl+F</b>	Fit Viewer to image—Fits the Viewer to the image.
	<b>Shift+F</b>	Grip to desktop—Fits the frame to the Viewer workspace. When enabled, the Viewer "sticks" to the workspace. You can then resize the workspace and the Viewer expands to match.
	Right-click menu > <b>Delete</b>	Close window—Deletes the Viewer. (A good strategy if a Viewer is misbehaving. Press <b>N</b> to create a new Viewer.)
	<b>1</b>	Buffer tabs. You can have two different buffers in a Viewer to compare images. See "Using the Compare Buffers" on page 28.
	<b>R, G, B, A, C;</b> <b>2/Shift+2</b> cycle	View RGBA channels—Toggles through the color channels.
	Right-click menu > Update On; <b>3/Shift+3</b>	Change update mode—Update On. Update mode that displays a rendered image only after it is finished rendering. This is for relatively fast renders.

Button	Shortcut	Button Name and Function
	Right-click menu > Progress; <b>3/ Shift+3</b>	Change update mode—Progress. Scrolling update mode that displays each line (starting from the bottom) as the image renders. Used for slower renders.
	Right-click menu > Update Off; <b>3/Shift+3</b>	Change update mode—Update Off. The Viewer does not update. Use this to load an image into a Viewer, then switch to the second buffer (see below) and do some changes. You can then compare it with the original.
		Toggle Incremental Viewer—Updates the changing portion of the image. For example, if Toggle Incremental Viewer is disabled and you composite a 10 x 10 pixel element on a 6K plate and pan the element, the entire 6K plate updates. When enabled, only the 10 x 10 pixel area is updated. To fix this, turn off the Incremental Update and adjust again—the glitches are corrected. This button has no effect on the output file or batch rendering speed, only on the image in the Viewer.
		Change/Activate Viewer Lookup Table—vlut off. VLUTs differ from Viewer Scripts in that you can scrub from the unmodified plate. See "Viewer Lookups, Viewer Scripts, and the Viewer DOD" on page 30.
		Change/Activate Viewer Lookup Table—g/o/log. Gain/Offset/LogLin allows you to apply different quick lookups to your image. See "Viewer Lookups, Viewer Scripts, and the Viewer DOD" on page 30.
		Viewer DOD—Turns on Region of Interest (ROI) rendering (limits your rendering area). See "Viewer Lookups, Viewer Scripts, and the Viewer DOD" on page 30.
	<b>4/Shift+4</b>	Change Viewer script—See "Viewer Lookups, Viewer Scripts, and the Viewer DOD" on page 30.
	Click and hold the Change Viewer script button.	Applies aperture markings. You can also right-click the Change Viewer script button and select Aperture Markings. See "Viewer Lookups, Viewer Scripts, and the Viewer DOD" on page 30.
	Click and hold the Change Viewer script button.	Applies a PlotScanline. You can also right-click the Change Viewer script button and select Plot Scanline. See "Viewer Lookups, Viewer Scripts, and the Viewer DOD" on page 30.


Button	Shortcut	Button Name and Function
	Click and hold the Change Viewer script button.	Applies a Histogram. You can also right-click the Change Viewer script button and select Histogram. See "Viewer Lookups, Viewer Scripts, and the Viewer DOD" on page 30.
	Click and hold the Change Viewer script button.	Views the Z channel. You can also right-click the Change Viewer script button and select ViewZ. See "Viewer Lookups, Viewer Scripts, and the Viewer DOD" on page 30.
	Click and hold the Change Viewer script button.	Displays superwhite and subzero pixels. You can also right-click the Change Viewer script button and select Float View. See "Viewer Lookups, Viewer Scripts, and the Viewer DOD" on page 30.
	Click and hold the Change Viewer script button.	Displays frames or timecode in the active Viewer. See "Viewer Lookups, Viewer Scripts, and the Viewer DOD" on page 30.
	<b>5/Shift+5</b>	Compare Mode—No Compare. Only one buffer is displayed. See "Using the Compare Buffers" on page 28.
	<b>5/Shift+5</b> ; click and hold the Compare mode button.	Compare Mode—Y Wipe (Horizontal Compare) mode. You can also right-click the Compare mode button and select Y Wipe. See "Using the Compare Buffers" on page 28.
	<b>5/Shift+5</b> ; click and hold the Compare mode button.	Compare Mode—X Wipe (Vertical) compare mode. You can also right-click the Compare mode button and select X Wipe. See "Using the Compare Buffers" on page 28.
	<b>5/Shift+5</b> ; click and hold the Compare mode button.	Compare Mode—Blend (Fade) compare mode. You can also right-click the Compare mode button and select Blend. See "Using the Compare Buffers" on page 28.
	Click to toggle; or right-click and select Show Bounding Boxes or Hide Bounding Boxes.	Display DOD and image border—Displays the green Domain of Definition (DOD; the DOD is either created automatically, or manually with SetDOD) and the red frame border. It has no effect on processing or the rendered image.

Button	Shortcut	Button Name and Function
	<b>Home</b>	Reset Viewer—Centers the image and sets the zoom level to 1:1.
	<b>F</b>	Fit image to Viewer—Fits the image to the frame. Be careful, since you may get a non-integer zoom (for example, instead of 2:1, you get 2.355:1), which may result in display artifacts. Do not use this option when "massaging" pixels.
		Launch Flipbook—Renders a RAM-based image player. Left-mouse click: Render with the current settings. Right-mouse click: Displays the Render Parameters window.
		Broadcast Monitor—Mirrors the selected node in the Viewer on a video broadcast monitor. The broadcast monitor option is only available on the Macintosh OS X version of Shake. For more information, see "Using a Broadcast Monitor" on page 63.

For a table of additional common buttons related to onscreen controls, see “Onscreen Control Buttons” on page 41.

### Using the Compare Buffers

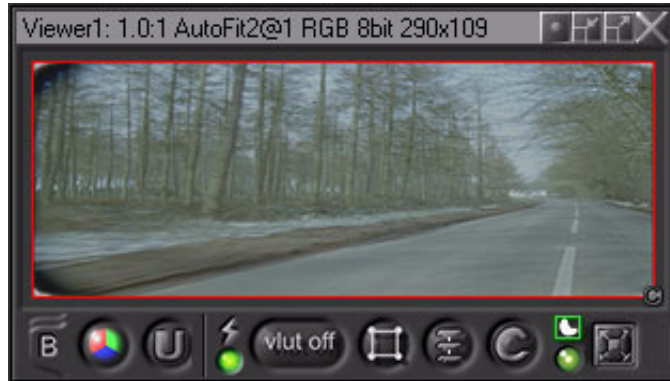
Use the A and B buffers to load two images at once into a Viewer. The following example uses two images from the Keying tutorial. (The images can be found in *doc/pix/Keylight*.)

- 1 In the Viewer, ensure buffer  is open, and load one of the Keylight tutorial images in the Viewer.

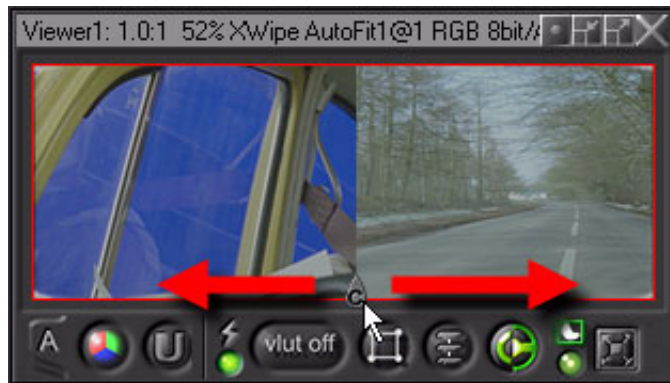
**Note:** To load an image into the Viewer, click the button on the left side of the node.




- 2 To switch to buffer B, click the A tab, or press **1** (above the Tab key, not on the NumPad). The A tab switches to B when clicked.
- 3 Load the second image into buffer B.
- 4 Press **1** to toggle between buffers. You can also click the A and B tabs.



- 5 In the lower-right corner of the Viewer, click and drag the small grey C button left and right.



The Compare Mode button  at the bottom of the Viewer indicates that you are in vertical mode.

- 6 To remove the compare, click Compare Mode.  
The compare is removed and the button is no longer highlighted .

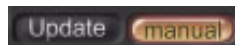
- 7 To enable compare again, click and drag the small C button in the corner of the Viewer. For a horizontal compare mode, click and drag the C button up on the right highlighted edge. For a vertical compare mode, click and drag the C button to the left on the lower highlighted edge.



No updates in the Viewer? A common mistake is to move the Compare Slider all the way to the left or right (or top or bottom)—one image disappears and only the second image is revealed. So, when you change a node parameter, you do not see any updates, which makes you angry at us, which makes you go to the Shake list and rant for several messages before you settle down. To avoid this regrettable path, turn off the Compare Mode to ensure you are looking at the current image.




If it is not the Compare Mode, ensure the Change Update Mode is set to Update On .

If the Update Mode is not the problem, check the manual update mode at the top of the interface.



OK, now you can rant on the Shake list.

### Viewer Lookups, Viewer Scripts, and the Viewer DOD

There are three similar ways that affect how your images are viewed—the Viewer Lookups (VLUTs) , the Viewer DOD , and Viewer Scripts . These functions modify the image for efficiency or previsualization purposes, and do not affect the output image. You can, however, optionally apply these to a render that is launched from the interface.

The following is an example of using a VLUT with a log image.



When LogLin conversion is enabled in g/o/log (Gain/Offset/LogLin), you still work on the log image in the process tree, but you see the linearized plate.



**To activate the VLUT or Viewer Script Controls:**

- 1 Apply your VLUT or script.
- 2 Right-click the Viewer Lookup Table button  and select one of the three Load Viewer Lookup Controls location options.
- 3 In the designated window or tab (selected in step 2), enable the specific controls you want. For example, in the above illustration, activate Log2Lin.

VLUTs have an additional right-click function to specify whether pixel values are scrubbed from before or after the VLUT. Right-click and hold the VLUT button, and enable (or disable) “Scrub before lookup.”

The following table includes the current default scripts and VLUTs.

Function	Notes
	Gamma/Offset/LogLin—Allows Gamma, Add, and LogLin operators to be applied.
	A field chart with many chart controls. 
	Displays a PlotScanline of your image. For more information, see "Using the PlotScanline to Understand Color Correction Functions" on page 342. 
	Displays the values along the horizontal axis (where the light grey line is). The sample image blue screen is evenly lit. You can view RGB, A, or RGBA, and calculate the value based on color, luminance, or value.

**Function****Notes**

Displays a Histogram of your image.

Viewer Script Controls (right-click the Viewer Script button to select Load options):

- ignore: Ignores pixels with a 0 or 1 value.
- maxPerChannel: Pushes the values up on a per-channel basis.
- fade: Fades the display of the Histogram.



The colors are squeezed down in a limited range, an indication that this is probably a logarithmic image.

Notice the big healthy chunk of blue near the high end. That is good.





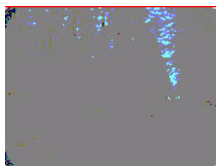
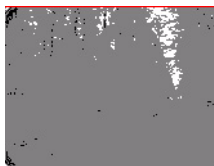




Displays the Z depth of an image either normalized or between a set range. A very important note: Closer pixels are white, so the image can fade to infinity (black) without a visual discontinuity.

Viewer Script Controls (right-click Viewer Script button to select Load options):

- floatZinA: Puts the Z values in the alpha channel to scrub and retrieve these values. The values are either Off, the Original values, or Distance (normalized between 0 and 1). If you have an object that moves from far away toward the screen over several frames, Original returns your Z values relative to each other; Normalized only indicates the Z values within that frame.
- zNormalize: Indicates if the render came from Maya or 3ds max. The subparameter zInfinity sets the limit at which point pixels are considered infinite, and are therefore clipped.
- zRangeSource: Evaluates the original values, or the near/far Input values.



Function	Notes	
	<p>Displays pixel values above 1 or below 0 for float images. The alpha channel is also tested.</p> <p>Viewer Script Controls (right-click the Viewer Script button to select Load options):</p> <ul style="list-style-type: none"><li>view: This parameter controls how the pixels are displayed:<ul style="list-style-type: none"><li>per Channel: Sets subzero pixels to 0, sets pixels between 0 and 1 to .5, and pixels above 1 to 1. This is applied on a per-channel basis.</li><li>per Image: Turns subzero pixels black, pixels between 0 and 1 grey, and pixels above 1 to 1. This is applied across the entire image, so if any channel is beyond 0 or 1, it is indicated.</li><li>on Image: Mixes the subzero and superwhite pixels back onto the image. The colors are controlled with the two color pickers.</li></ul></li><li>subzero color: Only active when view equals on Image, it indicates the subzero pixels.</li><li>superwhite color: Only active when view equals on Image, it indicates the superwhite pixels.</li></ul> <p>In this example, do the following:</p> <ul style="list-style-type: none"><li>Read in <i>doc/pix/keylight/saint_bg.@.jpg</i>.</li><li>Apply an Other-Bytes node and a Color-LogLin node.</li><li>Toggle Bytes from 8 to float.</li><li>Apply the Float View Viewer Script. Since LogLin pushes values above 1, the sky loses its punch when you go back out to Log and you process the image in only 16 bits.</li></ul>	
<b>Tree</b>	<div><div>Input Log image</div><div>LogLin (linear) image</div></div>	
	 	
<b>per-channel float view</b>	<b>per-image float view</b>	<b>on-image float view</b>
		
<p>The per-channel view indicates that most of the superwhite values are in the blue channel. The per-image view indicates the dark areas more clearly. The on-image view codes the highlights yellow and the darks blue.</p>		

Function	Notes
	<p>Timecode—Displays frames or timecode in the active Viewer.</p> <p>To show and modify the frames/timecode display:</p> <ul style="list-style-type: none"> <li>■ Right-click the Viewer Script button and select TimeCode, or click and hold the Viewer Script button and select the Timecode button. By default, timecode is displayed in the Viewer.</li> <li>■ Right-click the Viewer Script button and select Load Viewer Script into Parameters2 Tab. The timecode parameters are loaded into the tab.</li> <li>■ Click the modes pop-up list to select Frames, Padded Frames, Timecode, or Timecode Dropped Frame.</li> <li>■ Use the TimeOffset subtree to offset by hours, minutes, seconds, or frames.</li> <li>■ Color: Click the Color Picker box to change the color of the text display.</li> <li>■ BgColor: Click the Color Picker box to change the color of the timecode display background box.</li> <li>■ BgOpacity: Controls the opacity of the timecode display background box.</li> <li>■ size: Controls the size of the frames/timecode display.</li> <li>■ xPos: Controls the X position of the frames/timecode display. You can also use the onscreen controls to reposition the display.</li> <li>■ yPos: Controls the Y position of the frames/timecode display. You can also use the onscreen controls to reposition the display.</li> </ul>

The VLUTs and the Viewer Scripts are similar in that they apply an arbitrary set of functions that modify the image. A typical example is a color lookup table to compensate for the display properties of the monitor. The key difference is that VLUTs allow you to scrub pixel values from the unmodified image. You can disable this behavior for VLUTs. You always scrub the modified pixel values with Viewer Scripts. As an example, you may want to work on Cineon plates in logarithmic space without converting the plates to linear space. However, you want a rough idea of what the images look like in linear color space. Apply a VLUT to convert the images to linear space. Your color scrubs still derive from the input logarithmic plates, and ensure accurate processing for your output images.

Therefore, VLUTs are typically used for color correction, and Viewer Scripts are typically used for odd operations like something for stereoscopic viewing. Both allow you to enter any series of precreated functions.


Although Shake includes only one VLUT, you can have as many VLUTs as you want. You can only turn on one VLUT and one Viewer Script at the same time, but both can be activated simultaneously. To apply multiple color corrections, build your VLUTs and scripts to have multiple controls.

**Note:** Currently there is no version control in the Viewer Script. If you extend functionality of existing Viewer Scripts that may have been saved in existing Shake scripts, you should rename the new version of the Viewer Script to something other than the original name.

The Viewer Domain of Definition (DOD) limits your rendering region to a box to optimize processing. For example, if you are doing head replacement, you may want to activate the Viewer DOD and box off the head, saving your processing time for the rest of the image.

Keep in mind:

The Viewer DOD  limits the rendering area on the Viewer (does not affect the output image).

Display DOD  displays the green internal DOD box associated with each node and the red frame boundary (does not affect processing).

The following image has the VLUT 1 and the Viewer DOD applied.



Right-click the Viewer DOD button to access the DOD control options. For example, using Frame DOD to Viewer (sets the DOD to the Viewer frame), you can zoom in on an area you want to focus on and limit your DOD to that area. Note that it is not dynamic, as it would need to constantly recalculate as you pan. For more information, see “The Domain of Definition (DOD)” on page 50.

### Creating Your Own VLUTs and Viewer Scripts

The preset examples are stored in the end of the *nreal.b* file. To roll your own, you first declare them in a *startup* directory following the same guidelines for macros. The following functions do absolutely nothing:

```
image ViewerLookup1_(image img)
{
    return img;
}
```

```
image ViewerScript1_(image img)
{
    return img;
}
```

You then, also in a *startup* file, hook them into the Viewers:

```
nfxDefViewerLookup("Lookup1", "ViewerLookup1_()", "default");  
nfxDefViewerScript("Script1", "ViewerScript1_()", "default");
```

The first argument ("Lookup1", "Script1") is the name of the VLUT/Script as it appears in the list in the interface. The second arguments ("ViewerLookup1\_()", "ViewerScript1\_()") are the actual functions they call when activated. These must be declared in a *startup* .h file. The third arguments are the optional icon files, relative to the *icons/viewer* directory. It is assumed there is an .nri extension and that you also have a focused version called *[icon].focus.nri*.

Therefore, if you want to load a button called *icons/viewer/vluts/dufus.nri*, you also create a focused version called *icons/viewer/vluts/dufus.focus.nri*. You then use "vluts/dufus" as your icon name. "default" means it is looking for *vlut.@.nri*, *vlut.@.focus.nri*, *vscrip.@.nri* and *vscrip.@.focus.nri* (@ = 1, 2, 3, etc.). All paths are relative to *icons/viewer*. The icons for Viewer scripts are 30 x 30 pixels, no alpha. The standard VLUT buttons are 51 x 30 pixels, no alpha. There is a macro in the "Cookbook" section of the *Shake 3 Tutorials*, "Other Macros—VLUT Button," to make your own. Other macros required to run the VLUT Button macro can be found in *doc/html/cook/macros*.

### The Viewer Hot Keys

The following table contains additional Viewer hot keys.

Keyboard	Notes
<b>N</b>	Create/Copy New Viewer.
<b>F</b>	Fit Image to Viewer.
<b>Ctrl+F</b>	Fit Viewer to Image.
<b>Shift+F</b>	Fit Viewer to Desktop.
<b>Alt</b> -drag	Pan image.
<b>+</b> / <b>-</b>	Zoom image in Viewer.
<b>Home</b>	Reset view.
<b>R, G, B, A, C</b>	Toggle Red, Green, Blue, alpha, and Color views.

Also, see the table on page 25 for keyboard equivalents to Viewer buttons.

### The Viewer Right-Click (Contextual) Menu

Contextual menus differ depending on the location of the cursor in the interface, or what function/button the cursor is on. The following table shows the right-click (contextual) menu options for the Viewer.

Tree	Function	Keyboard	Notes
Edit	Undo	<b>Command +Z / Ctrl+Z</b>	Undo the last operation. Does not work with <i>RotoShape</i> or <i>QuickPaint</i> .
	Redo	<b>Command +Y / Ctrl+Y</b>	Redo the last undo command.
View	Zoom In/Out	<b>+ / -</b> (next to the <b>Delete</b> (Mac) / <b>Backspace</b> (Linux / IRIX) key	Zooms in and out by increments. You can also <b>Ctrl</b> -middle drag or <b>Ctrl+Alt</b> -drag to zoom in or out with non-integer increments.
	Reset View	<b>Home</b>	Sets the Viewer ratio to 1:1. The Viewer ratio is listed in the upper-left corner of the title bar.
	Fit Image To Viewer	<b>F</b>	Resizes the image to the Viewer boundaries.
	Fit Viewer To Desktop	<b>Shift+F</b>	Fits the Viewer window to the larger desktop window. Does not change the Viewer zoom; it just helps you when resizing the larger Desktop pane.
	Fit Viewer To Image	<b>Ctrl+F</b>	Snaps the Viewer to the image size.
Render	Render Flipbook	<b>.</b> (period)	Renders a non-permanent flipbook.
	Render FileOut Nodes		Renders <i>FileOut</i> nodes to disk.
	Render Proxies		Renders proxy images.
Clear Buffer A/B			Clears buffer A or B.

Tree	Function	Keyboard	Notes
New Viewer		<b>N</b>	Creates a new Viewer. If the mouse is over a Viewer, it clones that Viewer.
Delete Viewer			Deletes that Viewer. Helps to clear up graphic/refresh problems.
Minimize / Restore Viewer			Stores the Viewer as a small bar.
Load Viewer Script Controls			Loads the VLUT/Viewscript controls into one of three options.
View Channel			Like the View Channels button, views the channel you select.

## Buttons

The following section discusses the buttons in the upper-right corner of the Shake interface.

### Load/Save



- Click Load or Save to open the Load Script window, or to save the current script with the same name. You can also press **Command+S** / **Ctrl+S** to save the script quickly. If the script is not yet named, the Save Script window opens.
- To save a script with a new name, choose File > Save Script As, and enter a new filename in the Save Script window.
- To reload the same script, choose File > Reload.  
The script that appears in the Shake title bar is reloaded.

### Backup Interval Times

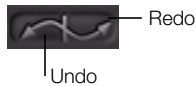
A backup script is stored automatically every 60 seconds in your `<UserDirectory>/nreal/autoSave` directory. The last saved script can be accessed with the File > Recover option (*shake -recover* in the Terminal), or browsed to under the Directories pull-down menu in the Browser.

The backup time interval can be changed in your *ui.b* files in *include/startup/ui/something.b*. Enter the following line (with the desired time interval in place of the “60”):

```
script.autoSaveDelay = 60;
```

The “60” is time in seconds between saves. For more information, see Chapter 18, “Macros, Expressions, and Scripting,” on page 673.

### Undo/Redo



There are 100 steps of undo and redo in Shake. To undo or redo a previous step, click Undo or Redo.

**Note:** You can also press **Command+Z** / **Ctrl+Z** to undo, or press **Command+Y** / **Ctrl+Y** to redo.

### Changing the Level of Undos

To change the level of undos, enter the following line (with your desired number of undos in place of the “100”) in one of your *ui.b* files:

```
gui.numUndoLevels = 100;
```

For more information, see Chapter 18, “Macros, Expressions, and Scripting,” on page 673.

## Update



The Update button controls what is updated in the scene. The Update button has three modes:

- Always: Updates the scene when you change a parameter, including time.
- Manual: The scene is not updated, including time, until you click Update, or click the left side of a node in the Node View, or press the **U** key.
- Release: Updates the scene when you release the mouse after changing any parameter, including time.

By default, click Manual once to toggle to Always. Click and hold Manual to select Always, Manual, or Release.

When Update mode is set to Manual, click Update to update the scene.

## Proxy












The proxy button, labeled “Base,” allows you to quickly get to one of your four proxy settings. Click Base once to toggle to P1. Click and hold the Base button for other proxy options. For more information on proxies, see “About Proxies” on page 103.




## Onscreen Control Buttons

Some functions, mainly transformations, have onscreen controls to help you interactively control your images in the Viewer. These controls appear whenever the node’s parameters are loaded.

When you use an active function with onscreen controls, a second shelf of buttons appears in that Viewer, and disappears when you load a different node’s parameters.

The following table shows the common onscreen control buttons.

Button	Shortcut	Button Name and Function
	Click Autokey.	Autokey—Auto keyframing is on. A keyframe is automatically created each time an onscreen control is moved. To enable, you can also right-click and select On-Screen Ctrl Auto Key On.  To manually add a keyframe without moving an onscreen control, click Autokey off and on.
		On-Screen controls—Show On-Screen Ctrl. Displays the onscreen controls. Click to toggle between Show On-Screen Ctrl and Hide On-Screen Ctrl.
	Click and hold the On-Screen controls button, or right-click.	On-Screen controls—Show On-Screen Ctrl On Release. The onscreen controls are displayed when modifying the image, and they return when you release the mouse.
	Click and hold the On-Screen controls button, or right-click.	On-Screen controls—Hide On-Screen Ctrl. Turns off the onscreen controls.
		Delete Keyframe—Deletes the keyframe at the current frame. This is used because controls for functions such as <i>Move2D</i> , keyframes for xPan, yPan, xScale, yScale, and angle are created simultaneously. Delete Key deletes the keyframes from all the associated parameters at the current frame.  To delete all keyframes for a parameter, such as <i>Move2D</i> on all frames, right-click the Delete Keyframe button and select Delete All Keys.
		Lock Direction—A control can be moved in both the X and Y directions.
	Click and hold Lock Direction.	Lock Direction—A control can be moved in only the X direction.
	Click and hold Lock Direction.	Lock Direction—A control can be moved in only the Y direction.
		On-Screen Control Color—Click the onscreen control color button to change the color of the onscreen controls.

Button	Shortcut	Button Name and Function
		Toggle Path Display—Displays the motion path and the keyframe positions in the Viewer. You can select and move the keyframes onscreen.
	Click and hold Toggle Path Display.	Displays only the keyframe positions in the Viewer.
	Click and hold Toggle Path Display.	The motion path and keyframes are not displayed in the Viewer.

For more information on onscreen controls and transformations, see “Onscreen Control Buttons” on page 41.

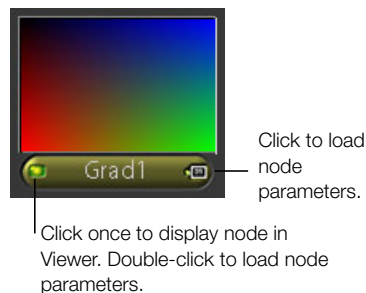
## The Parameters View

The controls for the nodes and the global script parameters are located in the Parameters tabs.

### Loading the Parameters Tabs

#### To view a node’s parameters:

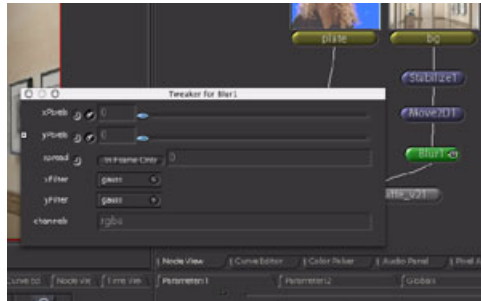
- 1 Click the icon on the right side of a node to display its parameters in a tab.



- 2 Double-click the left side of a node to load the parameters, and load that node in the active Viewer.
- 3 To load a second set of parameters in the Parameters2 tab, **Shift**-click on the right side of a node.

## Tweaker Windows

You can also tune a parameter in a floating “Tweaker” window. To open a Tweaker window, select a node and press **Ctrl+T**.



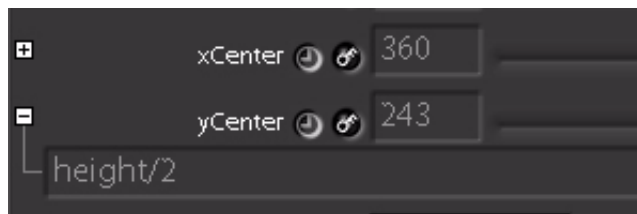
## Global Parameters

Double-click in an empty area in the Node View to load the Global Parameters into the Control workspace, or click the Globals tab in the Parameters workspace. For more information on Global Parameters, see “Global Parameters” on page 96.

## Editing Parameters

You can change the value of a parameter in several ways:

- Move the slider (if available).
- If a parameter has a plus sign next to it, it has either an expression or animation curve (a special expression subset). Click the plus sign to open the expression field. An expression can be an animation curve, a link to a different parameter, or a function. To clear a non-curve expression, move the slider, enter a new value in the text field, or right-click and select Clear Expression.

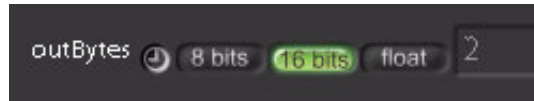



- To use virtual sliders to go beyond the slider limits, **Ctrl**-drag in the text field.

**Note:** If you are using a Wacom tablet, go to the Global Parameters and enable virtualSliderMode and set virtualSliderSpeed to 0.



- Some parameters have associated toggle buttons. You can enter a value, or click the toggle button. The numeric value allows you to enter expressions.



- Click the clock icon  to load parameters into the Curve Editor. When enabled (highlighted), the parameter is displayed in the Curve Editor. When disabled, the parameter does not appear in the Curve Editor.
- Press **Tab** or **Shift+Tab** to advance or retreat into adjacent text fields.

### Color Pickers

Some parameters have associated Color Pickers. There are several ways to use the Color Picker buttons:



- Click the Color Picker button (swatch)—the Color Picker opens, and you select your color from the Color Picker or Viewer.

- Click the + sign to expand the parameters and use the subparameters. The first row contains a slider to modify one channel (that you select)—(R)ed, (G)reen, (B)lue, (O)ffset, (H)ue, (S)aturation, (V)alue, (T)emperature, (M)agenta-Cyan, or (L)uminance. Move the slider to calculate in that channel, but convert the numbers back to RGB.



- Edit the individual channels or add expressions in the subtree.
- You can also keep the subtree closed, and use the Color button (color swatch) itself as a virtual slider. Using the channel buttons as the keyboard guide, press and hold the key (**R**, **G**, **B**, **H**, **S**, **V**, etc.) and drag left or right on the Color button. In the following illustration, **G** is pressed and the cursor dragged, and the green channel increases or decreases.



**Note:** To gang your color sliders together, press **O** and drag (**O** represents Offset).

### Animating Parameters

- To animate a parameter, enable Autokey  for that parameter. When enabled, keyframes are created when you modify the parameter.
- To enter a new keyframe, move the time slider to a different frame and change the value. You can also edit the curve in the Curve Editor by clicking on the parameter's clock icon.

When a parameter is animated, markers appear in the Time Bar to indicate that you have a keyframe(s). If the keyframe is on the parameter the cursor is over, the marker is green. If the marker is grey, other parameters from that node have keyframes.



To delete a keyframe in the Parameter View, go to that frame, right-click the parameter that you want to delete a keyframe from, and select Delete Current Key.

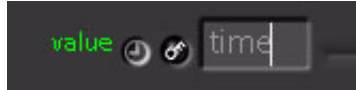
**Note:** You can also delete keyframes in the Viewer and Curve Editor.

Parameters linked to onscreen controls can be animated with the Viewer Autokey button. The Delete Key button also affects those parameters.

### Using Expressions

Any parameter can use an expression. An expression is any non-numerical entry. Some expressions are extremely simple, such as time.

- The time expression is a very simple example. The following returns the current frame value:



- Entering an expression or any letter (whether valid or not) creates a plus sign by the parameter name. To edit your expression, click the plus sign to show the text field.



- If the parameter has an expression and you adjust the slider, the expression is removed. If the parameter is animated, however, these special expressions are recognized by Shake and are not removed when the slider is adjusted.

**Note:** You can also remove an expression by right-clicking on the field and selecting Clear Expression.

- To load or save an expression, use the right-click menu.
- To create extra sliders to build complex expressions (and still allow interactive input), right-click the field and select Create Local Variable. To remove a local variable, right-click and select Delete Local Variable.

For a tutorial on using local variables and expressions, see “Lesson Four: Using Local Variables With Expressions” in the *Shake 3 Tutorials*.

For a list of mathematical expressions and variables, see Chapter 18, “Macros, Expressions, and Scripting,” on page 673.

### Linking Parameters

You can link any parameter to any other parameter.

- To interactively copy a parameter from one field to another, click the parameter name you want to copy, and drag it to the parameter name (a value or expression) that you want to copy the value to. This copies the value from the first field to the second.

**Note:** This drag and drop behavior also works when you drag color from one Color Picker button to another.

- You can also select the text in the text field and press **Command+C** / **Ctrl+C** to copy the information. Go to the second text field and press **Command+V** / **Ctrl+V** to paste.
- To interactively link two parameters together, **Shift**-drag on the parameter name you want to link to, and drag down to the parameter you want to link to the first parameter. This creates an expression. It links back to the first parameter by listing its name.
- You can also type the name of the parameter. In a *Move2D* node, for example, link yPan to xPan by typing xPan in the yPan parameter. The default argument for yScale is a link to xScale.

If the parameter is in a different node, preface the link with *nodeName.parameter*. For example, to link the red parameter *Add1* to *Add2*, enter the following expression in the *Add1* red channel:

*Add2.red*

Links can be used in expressions as well. For example, to double the value, enter *Add2.red\*2*

- If you want to link interactively using the drag and drop technique, load the one parameter into the Parameter2 tab, and then use the middle mouse button to drag the Parameter2 tab into a different window pane. (This is more trouble than it's worth, however.)
- To link to another time, use the @@ signs. For example:  
*Multi1.red@@(time-2)*  
links to *Multi1*'s red parameter from two frames earlier.
- Print values with the *Text* and *AddText* functions. To differentiate a parameter from regular text in the field, surround it with a pair of curly brackets.

For example:

*The current frame is: {time}*

prints:

*The current frame is: x*

where x automatically updates as each frame.

In another example, to print out a value of a parameter from a color correction node using a *Text* node:

*My red value = {Gamma1.rGamma}*

prints

*My red value is 1.7*

assuming there is a node called *Gamma1*, and its rGamma value is 1.7.

There is a macro called *Wedge* in the “Cookbook” section of the *Shake 3 Tutorials* book to print out wedging values for color timing Cineon files.

For a tutorial on linking parameters, see “Lesson Four: Using Local Variables With Expressions” in the *Shake 3 Tutorials*.

### Using Pop-Up Menus

Some parameters have associated pop-up menus, such as the filters on a transform or *Blur*, or the font type in *Text*.

Click the button to show the pop-up menu, then do the following:

- Right-click a menu item to select that item and remain in the menu. When you first click on the menu, hold down the left mouse button and move the cursor off of the menu. Then return the cursor to the menu and right-click. This allows you to quickly test different parameters.
- Left-click on a menu item to select that item and then close the menu.

### The Parameters Tab Right-Click Menu

The following table lists the options that appear when you right-click the top portion of the Parameter View.

Function	Notes
Clear Tab	Unloads the current parameters from the tab.
Create Local Variable	Allows you to create a variable specific to that node. Use this when you want to drive one or more parameters off of other parameters. See "Lesson Four: Using Local Variables With Expressions" in the <i>Shake 3 Tutorials</i> .
Delete Local Variable	Deletes the local variable for the selected parameter.
Add Notes	A dedicated local variable in string format. Allows you to add notes to any node (to help you remember what you were thinking at the time).
Reset Values	Resets all values in the node to their default state.

The following table lists the options that are available when you right-click a parameter.


Function	Keyboard	Notes
Copy	<b>Command</b> <b>+C</b> / <b>Ctrl+C</b>	Copies the selected nodes into the paste buffer.
Paste	<b>Command</b> <b>+V</b> / <b>Ctrl+V</b>	Pastes the buffer into the Node View. You can also copy nodes from the Node View and paste the nodes into a text document, and copy the text and paste it into the Node View.
Load Expression		Loads an expression from disk. The expression should be in Shake format. You can use this if you have a translator for another package's curve types.
Save Expression		Saves the current expression as a text file to disk.
Clear Expression		Clears the current expression.
Clear Tab		Unloads the current parameters from the tab.
Create Local Variable		Allows you to create a variable specific to that node. Use this when you want to drive one or more parameters off of other parameters. See "Lesson Four: Using Local Variables With Expressions" in the <i>Shake 3 Tutorials</i> .
Delete Local Variable		Deletes the local variable for the selected parameter.
Add Notes		A dedicated local variable in string format. Allows you to add notes to any node.

## The Domain of Definition (DOD)

The Domain of Definition (DOD) is a rectangular zone that Shake uses to bind the significant pixels in an image in order to optimize rendering speed. Everything outside of the DOD is considered as background (black by default), and is therefore ignored in most computations. Proper handling of the DOD is an extremely powerful way to speed your render times.

### To examine the efficiency of the DOD node:

- 1 Create an Image-*Text* node.

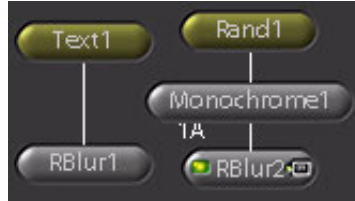
- 2 Ensure the Display DOD button in the Viewer is on . The green box around the letters is the DOD.



Because of the DOD, nodes attached to this image process in a fraction of the time it takes to calculate the same nodes with an image that fills the frame.

#### To test rendering times with DOD:

- 1 Attach a Filter-*RBlur* to the Text node, and set the *RBlur* oRadius parameter to 360.
- 2 In a separate branch, create an Image-*Rand* (to create an entire frame of pixels).
- 3 Attach a Color-*Monochrome* to the *Rand* node to turn it into a two-channel image (the *Text* node creates a BWA (black and white, with alpha) image by default, so you must match the channels to compare rendering speeds).
- 4 Copy the *RBlur1* node and attach the copied node (*RBlur2*) to the *Monochrome1* node.

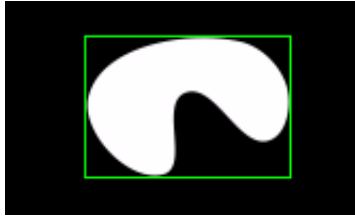


There is a significant difference in rendering speed, even though both images are the same resolution.

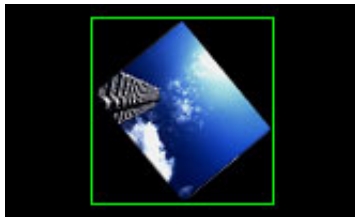
#### Assigning a DOD

All images from disk are automatically assigned a DOD that is equal to the resolution of the image. There are five ways to alter the DOD:

- Images generated in Shake have a DOD. For example, nodes from the Image tab such as *RGrad*, *Text*, and *RotoShape* automatically have an assigned DOD.

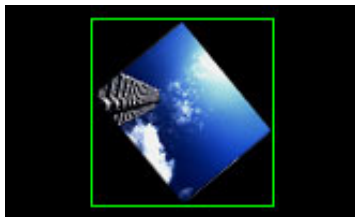


- The DOD of an image from disk that is transformed or filtered is automatically recalculated. For example, the following image is read in and scaled down and/or rotated with a *Move2D*.

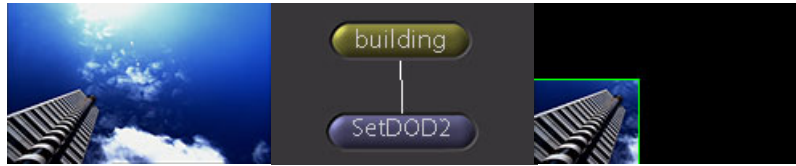


Also, if an image is blurred, the DOD expands accordingly.

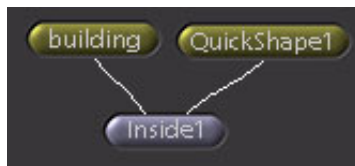
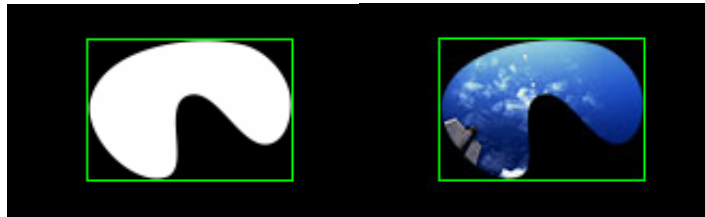
- A rendered .iff file from Shake is embedded with a DOD. When Shake writes an .iff file, it automatically saves the DOD information. Only the .iff format embeds the DOD. In the following example, the image that was written out in the previous (above) example is read back into Shake.



- The *SetDOD* node, located in the Transform tab, allows you to manually assign a DOD to an image. In the following illustration, a *SetDOD* is attached to the building image to limit the effects to the tower. This node is the bee's knees. Use it.

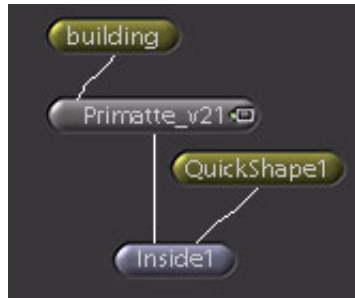


- You can combine multiple images using a DOD. When you combine two images, the DODs combine to form a larger rectangle. If, however, you use a node like *Inside* or *IMult*, it takes the DOD of the second node. If the building image is placed *Inside* of the *QuickShape* image from above, it inherits the DOD of the *QuickShape* node.



**Note:** When using onscreen controls to edit a shape (for example, a *Rotoshape* or *QuickPaint* object) that has control points within the boundaries of the DOD, move the cursor over the shape inside of the DOD, and then drag-select the points.

Combining images with a DOD is an excellent way to optimize green screen or blue screen images that need garbage mattes, because it simultaneously removes the garbage areas and assigns an efficient DOD to the image. The following is an example tree.



**Building**



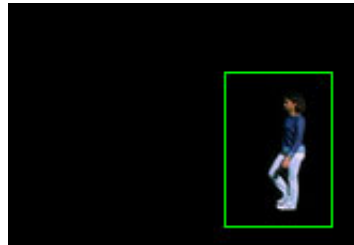
**QuickShape1**



**Primatte1**



**Inside1**



With a good understanding of the role of the DOD, you can optimize the tree before and after the node in question. The above example not only optimizes any nodes you attach to *Inside1*, but executes the *Primatte* and reads in the part of the image that is inside of the DOD, reducing processing and I/O activity.

### Keying, Color Correcting, and the Background Color

This section discusses the area outside of the DOD, which is called the Background Color (BGColor).

The two main keyers in Shake, *Keylight* and *Primatte*, recognize the background color, and have a toggle to key the background color in or out. By default, the background area is left black in the alpha by the keyer. To turn the background completely white, toggle BGCOLOR on.

Shake processes color correction of the BGCOLOR very quickly, as it recognizes there is a pure correction applied to previously black pixels. If the color correction does not change black, such as *Gamma* or *Mult*, it is ignored. If it does affect the black areas, such as *Add* or *Compress*, it processes these areas, but understands that they are still the result of a lookup process. Therefore, the DOD does not get reasserted to the resolution frame. This is the same process that is used when the Infinite Workspace kicks in. So, even though the pixels outside of the DOD are not visibly different from the pixels inside, the DOD remains in place. (For more information, see Chapter 4, “Compositing and the Node View,” on page 125.)



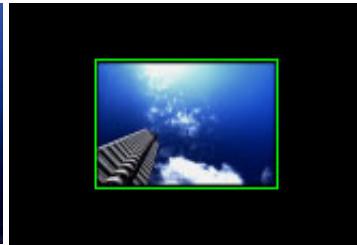
There may be cases, however, where you want to take advantage of the DOD for masking purposes. In this tree, an image is scaled down, and the brightness raised up with an *Add* node. This, however, turns the area outside of the image a medium grey. Since this area is recognized as outside of the DOD, it can be returned to black with a *Color-SetBGColor* node, which sets the color for the area outside of the DOD.



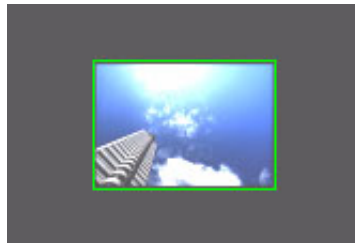
**Building**



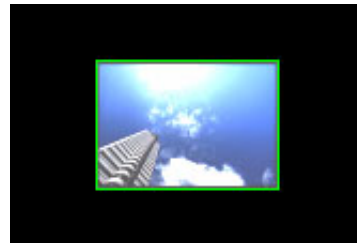
**Move2D1**



**Add1**



**SetBGColor1**



The *Layer-Constraint* node also limits a process. For more information on masking using the *Constraint* node, see Chapter 10, “Using Masks,” on page 383.

## The Flipbook

You can render a temporary Flipbook to preview your work. Once the Flipbook has rendered into RAM, use the playback controls (described below) to play back the Flipbook. The Flipbook is available on Macintosh OS X, Linux, and IRIX systems.

On a Macintosh OS X system, you can create a disk-based QuickTime Flipbook. For more information, see “Creating a Disk-Based Flipbook” on page 60.

### To launch the Flipbook from the interface:

- In the Globals tab, set the timeRange, for example, 1-50, 1-50x2.
- Do one of the following:
  - Load the node in the main Viewer and click Flipbook .
  - Load the node in the main Viewer, and then right-click in the Node View and select Render > Render Flipbook. Enter your settings in the Flipbook Render Parameters window and click OK. The images are rendered into RAM.
- To play the images, press . (period, or > key).
- To play backward, press , (comma, or < key).
- To stop playback, press the **Space bar**.
- To step through the animation, use the **Left Arrow** and **Right Arrow** keys.
- To scrub through the animation, press **Shift** and click.  
**Note:** On Linux systems, press **Shift+Ctrl** and click.
- To ping-pong the playback, press **Shift++**.
- To play through once, press **Ctrl++**.
- To increase or decrease the frame rate, press **+** or **-** (on the numeric pad).  
Note that the frame rate is displayed on the title bar. If you are getting real-time playback, it says “Locked.” Otherwise, it slows down the playback.  
**Note:** On Linux systems, press **O** to display information on the image itself.
- For real-time playback, press **T**.  
Real-time playback drops frames if it cannot maintain the desired speed. If dropping frames, it tells you what percent is being dropped.  
**Note:** On IRIX systems, press **D** to get double buffering. This reduces flickering.

In the Flipbook, you still have access to the same viewing functions that are available in the Viewer (view planes, coordinates and values, zoom in and out, etc.).

**Launching the Flipbook from the command line:**

Call up the files by relative or absolute paths. In the command line, indicate a time range and a frame placeholder—either `#` for padded numbers or `@` for unpadded numbers. For multiple padding that is not four-places, use the `@` or `#` signs multiple times; for example, `##### = 00001, 00002`, and so on.

For example, in `doc/pix/vp/vanilla`:

```
shake vanilla.#.iff -t 1-56
shake vanilla.#.iff -t 1-56x2
```

**Viewer Controls**

Function	Key	Notes
View R, G, B, alpha, or lum channel	<b>R, G, B, L, A</b>	
View RGB channels	<b>C</b>	
Get RGBA and x, y values of a pixel	Left-mouse scrub	The values appear in the title bar.
Linux: overlay information	<b>O</b>	
Change color values between 0-1, 0-255, Hex	<b>I</b>	
Zoom in/out	<b>+ / - by Delete</b> (Macintosh) / <b>Backspace</b> (Linux/ IRIX)	
Pan image	Middle-mouse button (Some mouse button behavior may vary, depending on the manufacturer. If the middle-mouse button does not pan, try right-clicking.)	
Re-center image	<b>Home</b>	
Close Window	<b>Esc</b>	

## Animation Controls

Function	Key	Notes
Play	. (period)	Think of it as the > key.
Play Backward	, (comma)	Think of it as the < key.
Stop Playing/Rendering	<b>Space bar</b>	
Continue Rendering	<b>/</b>	
Step Through Animation	<b>Left Arrow / Right Arrow</b>	
Scrub	<b>Shift</b> -click and scrub left and right.	
Ping-Pong	<b>Shift++&gt;</b>	
Play Once	<b>Ctrl++&gt;</b>	
Increase/decrease frame rate	<b>+ / 0</b> on numeric keypad	The rate is displayed in the title bar; the left number is the actual fps, and the right is the target fps.
Real-time toggle	<b>T</b>	Drops frames.
Double buffer (SGI only)	<b>D</b>	

## Memory Requirements

Real-time playback is a function of RAM, processor, image size, clip length, and graphics card. In Shake, images are loaded into memory and then played back. Current systems cannot achieve real-time playback with 2K-resolution images. With sufficient RAM and a good graphics card, files of up to 1K resolution should play back in real time.

Use the following formula to determine the amount of required memory:

$\text{width} * \text{height} * \text{channels} * \text{bytes per channel} * \text{images} = \text{bytes}$

For example, a single 1024 x 768 RGB 8-bit (1 byte) per channel image is:

$1024 * 768 * 3 * 1 = 2359296 \text{ bytes}$

Or, it is approximately 2.4 MB per frame.

To convert from bytes to megabytes (MB), divide by 1024 two times (1024 equals the number of bytes per kilobyte). Thankfully, all operating systems come with calculators. For a rough approximation, drop the last 6 digits.

**Note:** An 8-bit image is 1 byte, a 10- or 16-bit image is 2 bytes, and a float image is 4 bytes.

## Creating a Disk-Based Flipbook

Available on Macintosh OS X systems only, the Render Disk Flipbook command launches a disk-based Flipbook into QuickTime. This has several advantages over normal Flipbooks. For example, the Disk Flipbook allows you to view very long clips and to attach audio (loaded with the audio tab in the main interface).

**Note:** Real-time playback performance varies depending on your system hardware.

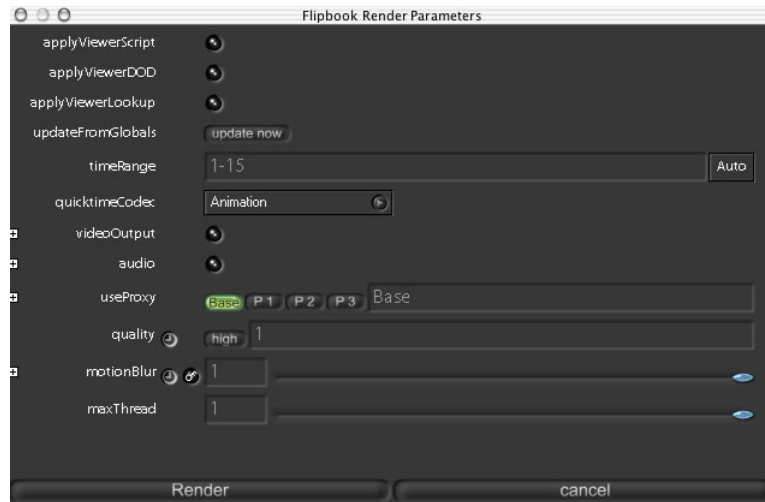
After viewing the Flipbook, you can write out the sequence as a QuickTime file and bypass the need to render the sequence again.

### To render a Disk Flipbook:

**Note:** It is recommended to select a format for the Flipbook in the Globals tab before rendering the Disk Flipbook.

- 1 Choose Render > Render Disk Flipbook.

The Flipbook Render Parameters window appears.



- 2 In the Flipbook Render Parameters window, set your Disk Flipbook parameters:
  - *Viewer Scripts, DODs, and Lookups* – To apply an active Viewer Script, Viewer DOD, or Viewer Lookup to the flipbook render, enable the relative parameter. For example, to render the flipbook with the DOD currently set in the Viewer, enable `applyViewerDOD`.
  - *updateFromGlobals* and *timeRange* – By default, `updateFromGlobals` is enabled (indicated by a green 'updated' button). When enabled, the time range and other Global settings (such as aspect ratio, proxy setting, motion blur, etc.) for the flipbook render are constantly updated in the Flipbook Render Parameters from the Globals tab.

**Note:** To disable `updateFromGlobals`, toggle “updated” to “update now” (indicated by a button labeled 'update now').

To override the `updateFromGlobals`, enter a frame range in the time range field and press Enter. The `updateFromGlobals` parameter is disabled.

To automatically set the frame range based on the *FileIn*, click Auto.

- *quicktimeCodec* – Select a quicktimeCodec from the pop-up list.

By default, the Animation codec is selected.

- *videoOutput* – To enable playback on a broadcast monitor, enable the videoOutput parameter.

When videoOutput is enabled, the quicktimeCodec is automatically set to your installed monitor card.

**Note:** When using a broadcast monitor, ensure that the quicktimeCodec parameter and the device parameter (in the videoOutput subtree) are the same.

The videoOutput subtree contains three conform options:

- Scale to Fit – Whatever is set in the DOD is fit to the broadcast monitor.
- Fit Maintaining Aspect Ratio – Fits to DOD to the full window while maintaining the aspect ratio.
- Crop To Fit – Crops the image to the DOD, and centers the image in the window.
- Scale To Fit – Whatever is set in the DOD is scaled to fit the window.
- aspectRatio – Set according to broadcastViewerAspectRatio in the Globals tab.

- *audio* – To render the flipbook with audio, enable audio.

In the audio subtree, you can set the sample rate and audio bits.

- *useProxy* – You can use proxies in the disk-based flipbook. For more information on proxies, see “About Proxies” on page 103.
- *quality* – Set your quality for the flipbook. By default, high (1) is enabled. Click the “high” button to toggle to “lo” quality.
- *motionBlur* – Enables and disables motion blur.
- *maxThread* – If you are working on a multiprocessor system, use the maxThread parameter to specify how many processors are devoted to the render.


### 3 Click Render.

The Shake QuickTime Viewer starts and the Pre-Rendering bar displays the render progress of the flipbook.

**Note:** The Shake QuickTime Viewer is a separate application—when launched, the viewer application icon appears in the Dock.

The rendered QuickTime clip appears in the Shake Preview window.

### To view and save the Disk Flipbook:

- 1 In the Shake Preview (Shake QuickTime Viewer) window, click Play  .  
**Note:** You can also press the **Space bar** to start and stop playback.
- 2 To loop the playback, choose Movie > Loop.  
**Note:** You can also choose Loop Back And Forth to ping-pong the playback.  
If using a broadcast monitor, the Movie menu includes the following additional options:
  - Video Output – Enables and disables the Flipbook display in the broadcast monitor.
  - Echo Port – Enables and disables the Shake Preview window in the interface. When disabled, only the playback bar of the Shake Preview window is displayed.**Note:** If audio is rendered with the flipbook and Play Every Frame is enabled, you will likely lose audio in the playback.
- 3 To save the QuickTime render, choose File > Save Movie, and specify the name and location for the saved file.
- 4 To quit the QuickTime viewer, do one of the following:
  - Press **Command+Q**.
  - Choose Shake QuickTime Viewer > Quit Shake QuickTime Viewer.
  - Press **Esc**.
  - Click the Quit button on the top of the Shake Preview window.

### Disk-Based Flipbook Temporary Files

You can specify a location (other than the default) for the temporary disk-based flipbooks. For example, if you have an Xserve RAID or other setup, you can store the temporary disk flipbooks on the array for real-time playback. The syntax for the default location for the temporary disk-based flipbooks (in the *nreal.b* file) is:

```
sys.qtMediaPath = "/var/tmp/Shake/cache";
```

To change the location for the temporary files, create a .h file and put the .h file in your home directory in the */nreal/include/startup/* file. For example, create a .h file similar to the following:

```
sys.qtMediaPath = "/Volumes/Scene12/QTtemp";
```

**Note:** You must first create the folder to store the files—Shake does not create a folder based on the information in the .h file.

You do not need to comment out the default path in the *nreal.b* file. Any .h file in the startup folder overrides the *nreal.b* file.

## Using a Broadcast Monitor

You can use a broadcast video monitor to preview your work with the Mac OS X version of Shake.

**Note:** The broadcast viewer option is not available with the IRIX or Linux versions of Shake.

### To enable the broadcast viewer:

- 1 Click the Globals tab.

By default, the format parameter is set to Custom.

- 2 In the format parameter, select your footage format from the format pop-up menu. For example, if you are working with NTSC D1 (4:3) non-drop frame footage, select NTSC ND (D1 4:3) from the format menu.

- 3 In the Viewer Toolbar, click the Broadcast Monitor button .

The broadcast video monitor mirrors the selected node (the node displayed in the Viewer), as well as the viewer scripts, VLUts, etc. In the Node View, the viewers displaying the node are printed under the node (for example, 1A, 2A).

**Note:** If a broadcast viewer is spawned prior to setting the correct format, the image may appear incorrect if the wrong aspect ratio is assigned. Go to the Globals tab and select the correct ratio from the format menu.

## Monitor Aspect Ratio

The guiSettings subtree in the Globals tab displays the broadcast viewer aspect ratio. When launched, Shake looks at the system's monitor card and creates the proper aspect ratio based on the format you select in the Globals. For example, if you have a D1 card and you select NTSC D1 from the format parameter, Shake displays non-square pixels in the Viewer and sends square pixels to the video monitor.

**Note:** If you change the value of the broadcastViewerAspectRatio using the slider or the text field, the default script is removed. As with all Shake parameters, you can enter an expression in the broadcastViewerAspectRatio parameter.

## High-Quality Mode

When the broadcastHighQuality parameter is enabled in the Globals tab in the guiSettings subtree), the image is fit to the size of the broadcast monitor in software mode (rather than hardware mode). The broadcastHighQuality parameter applies a scale node and a resize node, instead of using OpenGL. The broadcastHighQuality parameter is enabled by default.

**Note:** To adjust the gamma of your broadcast monitor (for example, when you are sending an NTSC signal to an HD monitor), use the broadcastGammaAdjust parameter in the broadcastHighQuality subtree.

## Navigating the Broadcast Monitor

You can use the standard Viewer navigation keys, such as pan (middle-mouse button), zoom (+ or -), and **Home** in the broadcast viewer.

To remove the broadcast viewer, do one of the following:


- Click the Broadcast Monitor button in the Viewer toolbar.
- Position the cursor in the broadcast Viewer, right-click, and select Delete Viewer.

## The Time Bar




The Time Bar is a display of a time range. It does not limit or control the actual parameters saved into the script. (To set the frame range to render, go to the Globals tab and enter the frame range in the timeRange parameter.)

### Setting the Frame Range

- The number in the field to the left of the bar is the start frame of the Time Bar, and the number in the field to the right is the end frame. In the above example, frames 1 to 21 are displayed. The Current frame is 4, with an Increment of 1. When you move the cursor in the Time Bar, the frame number that you will jump to is displayed under the cursor.
- If you have already set the timeRange in the Globals, click Home  in the Time Bar controls; the timeRange is used as the Time Bar frame range.
- Increment controls how far you advance when you press the **Left Arrow** or **Right Arrow** key.
- To set the current time, click or drag the time slider, or enter it numerically in the Current frame text field. As with any text field, you can use the virtual sliders—press **Ctrl** and drag the cursor left and right in the text field.
- To pan the Time Bar, press the middle-mouse button and drag, or press **Opt-click** / **Alt-click** and drag.
- To scale the Time Bar, press **Ctrl** and the middle-mouse button, or press **Ctrl+Opt-click** / **Ctrl+Alt-click**.

The controls illustrated below play through the script according to the Time Bar frame range, not the global timeRange.



- To stop playback, click Stop . You can also click the left mouse anywhere.
- There is no fps control for the Time Bar playback. If you **Shift**-click on a playback button, it renders all frames and stores the frames in memory. The second time you play back is then much faster.
- Click the keyframe buttons to jump to the previous or next keyframes.

The following table lists additional keyboard shortcuts.

Functions	Notes
<b>Left Arrow / Right Arrow</b>	Retreat/advance a frame based on the frame Increment setting (works in any window).
<b>Up Arrow / Down Arrow</b>	Jump to next/previous keyframe.
<b>Home</b>	Fit the current time range into the Time Bar.
<b>T</b>	Toggle timecode/frame display.

For more information, see Chapter 14, “Time View,” on page 511.

## General Window Interactivity

This section discusses Shake’s general window behavior.

### The Escape Button

To stop any processing at any time, press **Esc**.

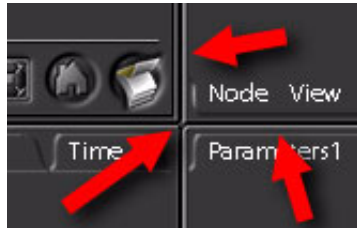
### Help Information

There are two ways to get information about the interface:

- As you pass the cursor (no need to click) over a node or Viewer, information for the node appears either in the title bar of the Viewer, or in the bottom-right text window. The displayed information includes node name, type, resolution, bit depth, and channels.
- Right-click most buttons to display a pop-up menu that lists the button options. You can use this to select a function or to find out what a button does.

## Resizing Windows

To resize any window, grab its border and drag. If you drag an intersection, you can move in two directions at once.



## Expanding a Window

To enlarge a window, press the **Space bar**. To zoom the window back down, press the **Space bar** again.

**Note:** Use of the **Space bar** is especially helpful when in the Curve Editor, working with high-resolution elements, or large scripts.

## Panning a Window

To pan a window, press **Opt-click** / **Alt-click** and drag, or press the middle-mouse button and drag.

## Zooming a Window

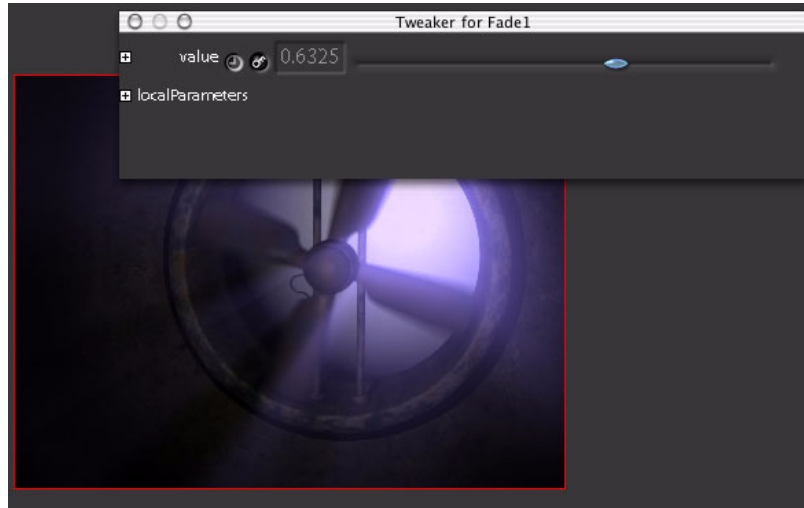
To zoom the Node View and Viewers, press **Ctrl** + middle-mouse button, or **Ctrl+Opt-click** / **Ctrl+Alt-click**. You can also use the **+** / **-** keys to zoom in or out based on the position of the cursor.

To reset the display to 1:1 viewing, press **Home**.

## Arranging Tabs

To rearrange the tabs, drag the tabs with the middle-mouse button or **Opt-click** / **Alt-click**, and then drag the tab to a new window pane. To detach a tab as a floating window, **Shift** + middle-mouse button or **Shift+Opt-click** / **Shift+Alt-click** on the tab. A good use for this is to detach a Parameter tab, then press the **Space bar** on the Viewer. You can then tune your image full screen.

To open a floating Tweaker window, select the node you want to tune and press **Ctrl+T**. A movable, floating Tweaker window for the node appears.



Close the detached tab to return it to its original position.

**Note:** To save your window settings for later use, choose File > Save Interface Settings.

### OS Window Functions

Shake responds to OS windowing, so you can resize the entire window, expand it to full screen, or stow it as an icon by clicking on the buttons in the upper-right of the Shake window title bar.



# Basics of Building a Script

## Chapter Summary

- The File Browser
- About Image Input and Output
- The *FileIn/SFileIn* Node
- About Time
- About Audio
- Global Parameters
- About Proxies

## The File Browser

The File Browser is an interactive browser to navigate through the network to load or write scripts, images, lookup files, expressions, or to track files. With the Browser, images can be listed as a long list of individual files or as a sequence. You can create directories, and delete files and directories directly in the Browser. You can also bookmark favorite directories.

## Opening the Browser

### To open the File Browser:

- Create or edit a *FileIn* or a *FileOut* node.
- Click the Load Script and Save Script buttons.

To open the Browser (“Load image or sequence” window) from an existing *FileIn* or *FileOut* node, click the folder icon next to the file path.



The Browser opens.



## Navigating in the Browser

You can access a directory in several ways:

- Type the entire path, with or without the filename itself, in the File name field at the bottom of the Browser.

Absolute file paths can be one of several styles:

*/my\_proj/pix/my\_file.iff*

*/d4/my\_proj/pix/my\_file.iff*

*//Biggo/D/my\_proj/pix/my\_file.iff*


*Local file paths may also be used:*

*../pix/my\_file.iff*

- Use the Directories pull-down menu at the top of the Browser.  
The pull-down menu browses up the entire directory tree, including your root directory, the directory you launched Shake from, the \$HOME directory, the Shake installation, and any favorite directories you have entered. It also automatically adds recently visited directories to the list.

## Adding Directories to the Favorites List

You can explicitly add directories to the list in two ways:

- To temporarily add an entry to the favorites list, click Bookmark .  
To save the settings in your script, make sure “include interface settings” is enabled when you save your script. When loading a script, make sure “include interface settings” is on. Otherwise, choose File > Save Interface Settings, and they load by default.
- You can also permanently add to your favorites list. Add an entry in a *ui.h* file similar to the following:

```
nuiFileBrowserAddFavorite("/Documents/icons/scr");
nuiFileBrowserAddFavorite("/shots/shot1/pix");
```





- You can also instruct Shake to look in certain directories when you start the software with the following *ui.h* settings. Each listing is for a type of file—images, scripts, expressions, and so on. Note the slash at the end of the path:

```
gui.fileBrowser.lastImageDir= "/Documents/pix/";
gui.fileBrowser.lastScriptDir= "$MYPROJ/shakeScripts/";
gui.fileBrowser.lastExprDir= "//Server/shakeExpressions/";
gui.fileBrowser.lastTrackerDir= "$MYPROJ/tracks/";
gui.fileBrowser.lastAnyDir= "/";
```

For more information on a *ui.h* file, see Chapter 17, “Customizing Shake,” on page 623.

## Navigating in the File Listings Area

Click a folder or drive to enter that directory or drive.

Function	Notes
	Indicates a folder.
	Indicates a drive.
	Takes you to the last viewed directory.
	Takes you up one directory. You can also press <b>Delete</b> (Macintosh) / <b>Backspace</b> (Linux/IRIX).

Function	Notes
<b>Up Arrow / Down Arrow</b>	Moves up and down in the list.
Any letter	Once you have clicked in the file listings, press a letter on the keyboard to jump to the next occurrence of a file or directory that starts with that letter.

### Selecting Files

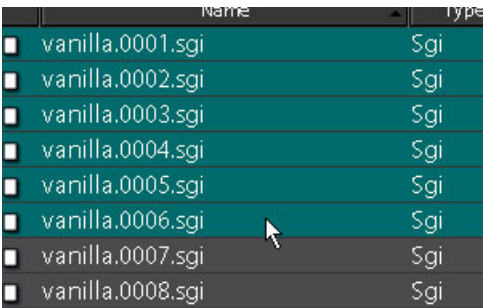
You can select files in several ways.

**To select single files:**

- Double-click the file.
- Press the **Up Arrow** / **Down Arrow**, and click OK (or press **Enter**).
- Press the first letter of the file you want. Press it again to jump to the next file that starts with that letter. Click OK (or press **Enter**).

**To Select multiple files in the same directory:**

- To select multiple files, drag-select the files and click OK.



- To select multiple individual files, press **Shift** and select the files.



**To select multiple images in different directories:**



Click Next in the Browser to load the current image(s) and keep the Browser open to continue to add files. When you have reached the last file, click OK. At any time in this process, the Node View may be accessed to examine *FileIn* nodes.

You can also press the **Space bar** to load in another file.

**Viewing Controls**

There are several tools to help you in the Browser.

- **The File List**  
Click the title of a column to arrange the list according to that type of information. For example, click Modified to list files by creation date. Click Modified again to reverse the order of information.
- **Toggle Buttons**  
The following buttons also change what is listed in the Browser:

Function	Notes
Short Listing	Lists only filenames, type, and size.
Sequence Listing	Toggles the listing of an image sequence as one listing or as several. To read in the entire sequence, ensure this Sequence Listing is enabled. These icons signify single or sequence files:  Indicates a single file.  Indicates an image sequence.
Images Only	Lists only recognized image types.
Show exact sizes	Shows the exact file size in kilobytes, rather than rounded off in megabytes.
Show full path	Lists the entire path of the selected file.
Filter	Filters out information. Use * and ? as your wildcards:

Function	Notes	
Filter (continued)	<b>Wildcard</b>	<b>Means</b>
	*	Any set of characters for any length.
	<b>Example</b>	<b>Lists</b>
	*.cin	a.cin, image.cin, image.0001.cin, ...
	*.cin.*	a.cin.0001, image.cin.hr
	*.cin*	a.cin, image.0001.cin, image.cin.0001
	image.*.tga	image.1.tga, image.10.tga, image.0100.tga
	<b>Wildcard</b>	<b>Means</b>
	?	Any character in that single position.
	<b>Example</b>	<b>Lists</b>
	?.cin	a.cin, 1.cin
	?.iff	ab.iff, 01.iff
	a.???	a.cin, a.iff, a.tga
	image.????.iff	image.0001.iff, image.9999.iff
	image.???1.iff	image.0001.iff, image.0011.iff, image.1111.iff

Click Update  to refresh the listing of the current directory in case files have been added or deleted outside of Shake.

### Reading or Writing Sequences

To select a file, click the filename, or click in the main file display area and use the hot keys as described above. You can also type the name in the File Name box at the bottom. Click OK to validate. If you are reading in an image sequence, enable Sequence Listing.

To write out a file with a *FileOut* node, select the directory, and enter the filename in the File Name window. If you are writing an image sequence, be sure to insert a # or an @ sign where you want the frame number to go in the name. Once you are finished, click OK to validate.

The following is a table of examples.

Files	Shake Notation
<i>image.0001.cin, image.0002.cin</i>	<i>image.#.cin</i>
<i>image.1.tif, image.2.tif</i>	<i>image.@.tif</i>
<i>image.iff0001, image.iff0002</i>	<i>image.iff.#</i>
<i>image1.tga, image2.tga</i>	<i>image@.tga</i>
<i>image.001.tga, image.002.tga</i>	<i>image.@@.tga</i>
<i>image.01, image.02</i>	<i>image.@@</i>

## About Image Input and Output

This section discusses importing images into Shake and their associated file paths, temporary files and disk space, basic time shifting, and retiming footage. For more information on supported file formats and other footage issues, see Chapter 6, “File Formats and Footage,” on page 201.

### Reading and Writing Images

Images are usually read into Shake with the *Image-FileIn* function. Importing images is therefore like any other function, since the *FileIn* node appears in the node tree.

Shake also includes image generation tools to create circles, gradations, text, shapes, etc., so you can create many images from within the application.

To save an image, attach a *FileOut* node to the image you want to write to disk. You can have unlimited *FileOuts* that can be placed anywhere along the node tree. Therefore, you can have outputs of different resolutions, bit depths, or color spaces. For example, you can simultaneously write a 10-bit 2K log Cineon image and an 8-bit video resolution linear gamma-adjusted frame for a video preview of your composite before the filmout images return from a film processing lab.

If you write an image without an extension (for example, *image\_name* instead of *image\_name.cin*), and you haven't explicitly set an output format, Shake writes the image to its native .iff format.

For the batch system, you can use the *-fileout* option, or the abbreviation *-fo* to write your image. For example:

```
shake my_image.cin -fo my_image.iff
```

copies *my\_image.cin* as a new image file in .iff format.

The interface allows you to view frames anywhere along the node tree using multiple Viewers. In the script or the command-line mode, however, you may need to explicitly call intermediate nodes with either *-view* or *-monitor*. For example:

```
shake my_image.rla -rotate 45 -view -flop
```

shows two viewers, one image rotated 45 degrees, and the second image rotated and flopped.

If you append a .gz to the end of the filename, Shake further compresses the file. Shake recognizes the file format and all of its channels when reading or writing one of these images:

```
shake uboat.iff -fo uboat.iff.gz
```

This further compresses *uboat.iff*, maintains it in .iff format, and retains the Z channel.

### Image Sequences

You can refer to an image sequence by replacing the frame number with a # sign or an @ sign.

- The # sign signifies a 4-place padded number.
- The @ sign signifies an unpadded number.

You can also use several @ signs to indicate padding to a different number. (For example, @@@ signifies 001.)

The following table lists some formatting examples.

Shake Format	Reads/Writes
<i>image.#.iff</i>	<i>image.0001.iff, image.0002.iff</i>
<i>image.%04d.iff</i>	<i>image.0001.iff, image.0002.iff</i>
<i>image.@.iff</i>	<i>image.1.iff, image.2.iff</i>
<i>image.%d.iff</i>	<i>image.1.iff, image.2.iff</i>
<i>image.@@@.iff</i>	<i>image.001.iff, image.002.iff</i>
<i>image.%03d.iff</i>	<i>image.001.iff, image.002.iff</i>

The above examples assume an exact relation between the current frame processed in Shake, and the frame read in. For example, at frame 1, *image.1* is read in. If you are reading the images in from the interface with Sequence Listing enabled in the Browser, you see the actual sequence in the fileName field. For example:

Image	Shake Syntax With Sequence Listing
<i>image.4.iff, image.5.iff ... image.10.iff</i>	<i>image.4-10@.iff</i>
<i>image.4, image.5.iff, image.6.iff, image.10.iff</i>	<i>image.4-6,10@.iff</i>

Unlike the previous examples, these offset the clip timing by placing *image.4.iff* at frame 1 and *image.5.iff* at frame 2. In the first example, *image.10.iff* is placed at frame 7. In the second example, *image.10.iff* is placed at frame 5. All sequence gaps are ignored. To offset or retime a clip, use the Timing subtab in the *FileIn* parameters or the Time View tab.

When reading in an image, the Browser allows you to specify if the first sequence image (suppose the sequence starts at frame 20) is placed at frame 1, start frame (for example, 20), or the current frame.

## File Paths

Shake can read local or absolute file paths. For example, with a machine named “Biggo,” take a directory structure like the following:

```
/shots/pix/my_image.iff
```

```
/shots/scr/my_script.shk
```

The script can access *my\_image.iff* in the following ways:

```
FileIn1 = FileIn("../pix/my_image.iff");
```

```
FileIn2 = FileIn("/shots/pix/my_image.iff");
```

```
FileIn3 = FileIn("//Biggo/shots/pix/my_image.iff");
```

Local file paths in a script are local to where the script is located, not from where Shake is started.

When Shake reads in an image, it converts the file path of the image to the UNC naming convention. This labels the machine name first, and then the file path. The third listing above is an example of this convention. This behavior can be turned off in a preferences file. For more information, see “File Path and Browser Controls” on page 641.

Shake looks for its files in the User directory (\$HOME) when launched from the icon, or the current directory if launched from the Terminal. This affects how manually-entered paths for *FileIns* are read. If the image paths are local (for example, “little\_man\_with\_pointy\_red\_hat/image.#.iff”), images are read relative to where the script is saved. If paths are global (for example, “//MyMachine/DiskWithStuffOnIt/little\_man\_with\_pointy\_red\_hat/image.#.iff”), then images have no relation to where the script is saved, and thus the script may be easily moved into different directories.

If the script and the image are on different disks, you must specify the proper disk—local file paths do not work.

## Temporary Files and Disk Space

Shake creates temporary files (“*tmp* files”) when writing certain formats of images, or when running out of memory. These temporary files are written like swap files, but are used before onerous activity occurs to save your machine from the crush of swapping. Normally, Shake reads in only the portion, either a group of scanlines or a tile of the image, that it can fit into memory. This means that the image is read not once, but many times, with only a bit read each time. Additionally, only the contributing portion of the image is loaded in. The ideal format to support this behavior is the native Shake .iff format (also the format licensed to Alias/Wavefront for their Maya software).

Certain other formats, however, do not support the ability to efficiently read a random portion of the image, and can take significantly longer to load.

Create temporary files:	Do not create temporary files:
Alias	AVI
BMP (depending on orientation)	DXP
Cineon (depending on orientation)	GIF
JPEG	IFF
PBM	Mental Images
Softimage	.mov/QuickTime
Targa (depending on orientation)	PNG
TIFF (depending on orientation)	RLA
YUV	SGI
	Side FX

To calculate the maximum temporary space needed during I/O, use the following formula:

$$tmp\ file = width * height * number\ of\ channels * bytes$$

where bytes = 1 for 8 bit, 2 for 10 or 16 bits, 4 for float.

Additionally, certain nodes also create *tmp* files during processing. This is required when pixels drastically change their x, y position during rendering. For example, if you rotate an image 90 degrees, the pixel in the lower right now moves to the upper right. In order to process this, Shake creates a *tmp* file that includes as much information as necessary to calculate the image. A tiling system is used, so these *tmp* files are typically much smaller than those created during I/O. The nodes prone to this behavior are:

- *Move2D*
- *Move3D*
- *Rotate*
- *Orient*
- *Flip*
- All Warps (*WarpX*, *DisplaceX*, *Turbulate*, etc.)

By default, your temporary files are written to: */var/tmp*

To relocate the temporary directory, set the environment variable TMPDIR:

```
setenv TMPDIR /more_disk_space/tmp
```

## The FileIn/SFileIn Function

*FIBlend/FINearest/FIPullUpDown/FISpeed/IRetime*

The *FileIn* function reads in images from disk. The *SFileIn* is an advanced version of *FileIn* with more functionality, and can be additionally modified by the invisible functions *FIBlend*, *FINearest*, *FIPullUpDown*, and *IRetime*. These are “invisible” because they do not appear in the Node View, but are saved into the script to modify the *FileIn* and *SFileIn* functions.

- *FileIn*: A pre-v2.5 function, convenient for scripting, given its brevity. Can only be shifted in time with *IRetime*.
- *SFileIn*: From v2.5 and later; can be hooked up to have preset proxies, shifted with *IRetime*, or modified by *FIBlend*, *FINearest*, or *FIPullUpDown*.
- *FIBlend*: This invisible function does non-linear retiming of a sequence, blending frames together. Modifies *SFileIn* only.
- *FINearest*: This invisible function does non-linear retiming of a sequence with no frame blending. Modifies *SFileIn* only.
- *FIPullUpDown*: Does pullup or pulldown operations on an *SFileIn*.
- *FISpeed*: Similar to *Blend*, except instead of a curve, you control speed with a slider, for example, 2x, .5 speed, etc.
- *IRetime*: Sets the start/stop frame of a clip, can slip sync, and controls how the clip behaves for frames outside of the frame range. Works for *FileIn* and *SFileIn*.

## Paths

Both *FileIn* and *SFileIn* recognize local, absolute, variables, or URL paths:






- Absolute Path: */usr/people/dufus/myimage.iff*
- Local Path: *./myimage.iff*
- Environment Variables: *\$myimages/myimage.iff*
- URL Address: *//Monster/usr/people/dufus/myimage.iff*

For more information on variables, see “Environment Variables for Shake” on page 664.

If a frame cannot be found, a black frame is used. The size of the frame is set according to the *defaultWidth* and *defaultHeight* settings in the Format subtree in the Global Parameters.

## Basic Time Shifting

With a *FileIn*, you can shift your clip in time and set in/out frames. As of Shake version 2.5, the *FileIn* creates *SFileIn* functions. You typically shift your sequences using the Time View. However, the *SFileIn* controls can also set the behavior of your frames before and after the sequence range.

Icon	Name	Notes	Example: Assumes a 5-Frame Sequence
	<b>Black</b>	The frames are black.	1, 2, 3, 4, 5, black, black, black...
	<b>Freeze</b>	The first and last frames are repeated before and after the clip.	1, 2, 3, 4, 5, 5, 5, 5...
	<b>Repeat</b>	The sequence is repeated.	1, 2, 3, 4, 5, 1, 2, 3...
	<b>Mirror</b>	The sequence is repeated, flipping the order each time. The first and last frames are not doubled up.	1, 2, 3, 4, 5, 4, 3, 2...
	<b>Inclusive Mirror</b>	The sequence is repeated, flipping the order each time. The first and last frames are shown twice in a row.	1, 2, 3, 4, 5, 5, 4, 3...

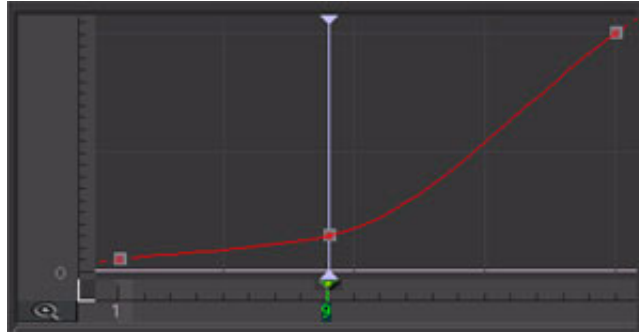
For more information on shifting clips, see “Shifting Clips and In/Out Points” on page 512.

## Retiming

You can squeeze, stretch, or non-linearly retime your clip when you activate reTiming in *SFileIn* parameters. You can use Speed or Remap.

- Speed: Gives you a multiplier on the speed of the clip, so .5 returns a clip twice the length of the original.
- Remap: Allows you to draw a curve, with the X axis as the input and the Y as the output.

This creates an ease-in on your clip, slowing the first part and accelerating the clip as it nears the end.



Both of these can blend frames with either Blend or Nearest mode. Blend averages frames together and Nearest takes the frame right below the called for frame. For example, if frame 5.7 is called, frame 5 is used.

Parameter	Function
<i>retimeMode</i>	By default, you are given two options for frame blending—the dramatic "None," and the dazzling "Blending" which averages frames together. It's written so that others can input their own algorithms, so write your Congressman.
<i>weight</i>	A gamma curve is applied to the contribution of the source frames blended to create the destination frame. Gain of 0 means each source frame needed to create the destination contributes equally, while a higher gain, such as 2, causes the center frame to give the greatest contribution and frames farther away proportionately less.
<i>range</i>	Controls how many frames beyond the normal averaging should be considered. For example, if you want to extend a source clip of 20 frames to 40 frames, each source frame is considered in 2 output frames. With a range of 2, it is considered in 4 output frames, resulting in more blending. If you only apply this value with no other modifications, you get repetitions of neighboring frames to help you with degrading.
<i>start/endFrame</i>	Specifies what frame range is considered for retiming purposes.
<i>retime</i>	Indicates the actual frame pulled in. Although the number is a float, the resulting frame is rounded.
<i>speed</i>	Only available in Speed, this is a scaling factor to the clip length.

### Understanding the Retiming Parameters

If you are having difficulty understanding the multitude of Retiming parameters, copy this string, paste it into the Node View, and then render out 100 frames.

```
Text1 = Text(300, 100, 1, "%f", "Courier", 44.3, xFontScale, 1,  
Hermite(0,[10,76.08,76.08]@1,[261,76.08,76.08]@100), Hermite(0,[90,-34.69,-  
34.69]@1,[30,-34.69,-34.69]@100), 0, 2, 2, 1, 1, 1, 1, 0, 0, 0, 45, 0, 1);
```

Then, read in the rendered clip and test the retiming. Have fun.

### The TimeX Function

The *TimeX* function in the Other tab applies a rule to change timing on the input clip. By default, the value is *time*—at the current frame, it uses the current frame, or no change. However, you can modify the formula for interesting effects.

Like *Lookup* and *ColorX*, you can duplicate most other Time functions with *TimeX*. These other functions are, generally, just macros that include *TimeX*. The other functions are included because at times *TimeX* is counter-intuitive. The following are some examples that mimic the other functions.

TimeX expression	time-10	101-time	time%10+1	time>10?10:time
Explanation	Shifts the clip 10 frames forward. While processing frame 50, it reads input frame 40.	Assumes frame 100 is the last frame. At frame 1, 100 used.	Loops every 10 frames. Takes the remainder of time/10, and adds 1 (otherwise frame 10 = 0).	Freezes the clip at frame 10. Any frame before that is processed normally.
Other functions	Do nothing.	Pick 1 frame.	Double the rate.	Speed the clip up and down.

<b>TimeX equivalent</b>	time	100	time*2	"CSpline(0, 1@1, 30@25, 40@50, 90@75, 100@100)"
<b>Explanation</b>		At every frame, the node returns 100, so only input frame 100 is used.	At frame 10, frame 20 is used.	This arbitrary curve returns different frame values. You can use any spline type, with as many keyframes as you want.

Another more complex example is to animate 360 3D frames with an animated light. The light is from the right at frame 0, from the top at frame 90, from the left at frame 180, etc. You then position a fake light source in the composite. By figuring out the angle of the light to the 3D element (using trigonometry), you can pick one of the 360 input frames to fake the lighting change.

**Multiple Branches**

You can only have one branch traced up to the *FileIn* with a *TimeX* in it. To get multiple time shifts on the same clip, copy the *FileIn*. Note also that *FileIn* has timing controls.

Parameters	Type	Default	Function
<i>newTime</i>	float	time	These set the first and last frames as the clip length.

**Synopsis**

```
image TimeX(image, float time );
```

**Script**

```
image = TimeX( image, time );
```

**Command Line**

```
shake -timex time
```

## About Time

This section explains the notation Shake uses for a *FileIn* node, and the available *FileIn* options. It also discusses the notation for the *timeRange* parameter in the Global Parameters, or the *-t* option on the command line. For a discussion of the interactive controls of time, see Chapter 14, “Time View,” on page 511, and “The *FileIn*/*SFileIn* Function” on page 79.

### Time Notation for a FileIn

This section focuses on manual manipulation of time. For most interactive time manipulation, Shake relies on the Time View and its associated timing subtree in the *FileIn* parameters. You can manipulate time in other ways, specifically on the command line.

When Shake reads in a clip, it inserts the start and end frame of the clip in the clip name, and gives an indication of the padding style, denoted here with the number sign *#*:

*image.1-50#.iff*

Therefore, this indicates that only frames 1 through 50 are loaded, even though there may be more files. The other frames are black when read in with the default settings.

Shake puts the start of the range at frame 1. If you have:

*image.20-50#.iff*

at frame 1, *image.0020.iff* is read.

You can also shift the clip to frame 20 in the Time View of the interface.

Shake can recognize a series of frames when reading in a file without using the clip range. When looking at a sequential series of files, use a placeholder in the filename to represent the frame number. This placeholder is either a *#* (padded images, *image.0001.iff*, *image.0002.iff*, etc.) or an *@* (unpadded images, *image.1.iff*, *image.2.iff*, etc.). If your numbers are padded to a number other than 4, you can substitute multiple *@* signs. The following are some examples:

Shake Format	Reads/Writes
<i>image.#.iff</i>	<i>image.0001.iff</i> , <i>image.0002.iff</i>
<i>image.%04d.iff</i>	<i>image.0001.iff</i> , <i>image.0002.iff</i>
<i>image.@.iff</i>	<i>image.1.iff</i> , <i>image.2.iff</i>
<i>image.%d.iff</i>	<i>image.1.iff</i> , <i>image.2.iff</i>
<i>image.@@@.iff</i>	<i>image.001.iff</i> , <i>image.002.iff</i>
<i>image.%03d.iff</i>	<i>image.001.iff</i> , <i>image.002.iff</i>

Time Notation Setting the Script Range

The script range can be set in the Global parameters in the timeRange settings, or on the batch command line with the -t option, which overrides the script.

The range description is extremely flexible. The following are some examples:

Time Range	Number of Frames	Frames Rendered
1-100	100	1, 2, 3... 100
1-100x2	50	1, 3, 5... 99
1-100x20	5	1, 21, 41... 81
1-20, 30-40	31	1, 2, 3... 20, and 30, 31, 32... 40
1-10x2, 15, 18, 20-25	13	1, 3, 5... 9, 15, 18, 20, 21, 22 ... 25
100-1	100	100, 99, 98... 2

To set this in the command line when rendering a script, use the -t option:

```
shake -exec my_script.shk -t 50-60 -v
```

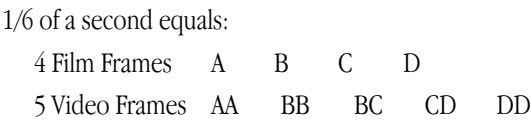
For command line examples of time manipulation, see “Frequently Used Functions” on page 746 of Appendix B, “The Command-Line Manual.”

For information on using the Time View, see Chapter 14, “Time View.”

Pulldown / Pullup

The 3:2 switch in SFileIn allows you to manage your pulldown/pullup of a sequence. A 30 fps to 24 fps (pullup) indicates a film sequence that has been telecined to 30 fps. You now want to return it to 24 fps. A 24 fps to 30 fps (pulldown) means you want to take 24 fps film footage and convert it to 30 fps. Both allow you to select the dominant field. Typically, PAL is odd and NTSC is even.

What in blazes does “pulldown” mean? This is a technique to temporally convert (resolution not considered) film footage to video footage and back again. Because film uses solid frames and video uses interlaced fields, and film runs at 24 fps and NTSC runs at 30 fps, you split the film footage into fields and double up 2 out of 5 frames to increase your frames to fill the 30 fps. To use the classic graph:



The third and fourth frames have fields that blend to stretch time. It's therefore called 3:2 because you have three solid frames and two mixed frames.

To fully reconstruct the original four film frames (in time—the resolution is lost), extract the field data from the 5 video frames. Here comes the odd bit: When you receive your footage, it has probably been edited, so it is not necessarily the case that frames 3 and 4 are the mixed frames (because all of the clips have been shifted in the edit). You therefore need to determine the first frame. Go to your first 5 frames in the sequence. If the first frame with field blending is frame 3, you know your firstFrame should be set to AA. If the first frame with field blending is frame 2, then the firstFrame is BB. Set your firstFrame parameter accordingly. If your first frames are a solid color and you are unable to determine the first mixed frame, jump to a time range of frames that displays the blending, and start guessing what firstFrame is until the fields go away.

**FileIn Parameters**

Parameters	Type	Defaults	Function
<i>imageName</i>	string		The local or absolute path to the image. See above for format. Note: You can get away with just supplying the imageName and omitting the others in the script.
<i>fileFormat</i>	string	"auto"	Tells Shake what the format is if the title is ambiguous. Generally, you do not need to set this, as "Auto" automatically detects the format. If you have a problem reading an image, try setting the format to correct this.
<i>autoAlpha</i>	int	0	If this is on (1), then Shake creates a solid alpha channel for images not containing an alpha channel. If the image already has an alpha channel, this parameter is ignored.
<i>deInterlacing</i>	int	0	This parameter is to be used when importing interlaced images and you intend to render with fieldRendering on. It takes the odd or even field of the image (counted from the top) and copies it over the other remaining field. It then does the same thing half a frame forward. You are therefore left with two images the same height as your input image, but squeezed into the same time frame. For more information, see Chapter 6, "File Formats and Footage," on page 201.

## Synopsis

```
image FileIn (  
    const char * imageName,  
    const char * fileFormat,  
    int autoAlpha,  
    int deInterlacing,  
  
    ...  
);  
  
image SFileIn (  
    const char * imageName,  
    const char * fileFormat,  
    int autoAlpha,  
    int deInterlacing,  
    const char * version,  
    int numberOfProxies,  
    const char * proxy1DefaultFile,  
    float proxy1DefaultScale,  
    float proxy1DefaultAspect,  
    const char * proxy1DefaultFormat,  
    const char * proxy2DefaultFile,  
    float proxy2DefaultScale,  
    float proxy2DefaultAspect,  
    const char * proxy2DefaultFormat,  
    etc  
  
    ...  
);  
  
image FIBlend (  
    image image,  
    char * retime,  
    char * retimeStartFrame,  
    char * retimeEndFrame,  
    float range,  
    flat weight,  
    int retimeBytes,  
    int reTiming,  
    float speed
```

```

)

image FINearest (
    image image,
    char * retime,
    char * retimeStartFrame,
    char * retimeEndFrame
    int reTiming,
    float speed
)

```

```

image FIPullUpDown (
    image image,
    char * type,

    int dominance,
    int firstFrame
)

```

```

image IRetime (
    image image,
    float timeSlide,
    float inPoint,
    float outPoint,
    string inMode,
    string outMode
)

```

### **Script**

```

image = FileIn(
    "imageName",
    "fileFormat",
    autoAlpha,
    deInterlacing
);

```

```

image = SFileIn(
    "imageName",
    "fileFormat",
    autoAlpha,
    deInterlacing,
    numberOfProxies,
    "proxy1DefaultFile",
    proxy1DefaultScale,
    proxy1DefaultAspect,
    "proxy1DefaultFormat",
    "proxy2DefaultFile",
    proxy2DefaultScale,
    proxy2DefaultAspect,
    "proxy2DefaultFormat",
    etc
);

```

### Command Line

*shake imageName imageName2 imageName3 etc...*

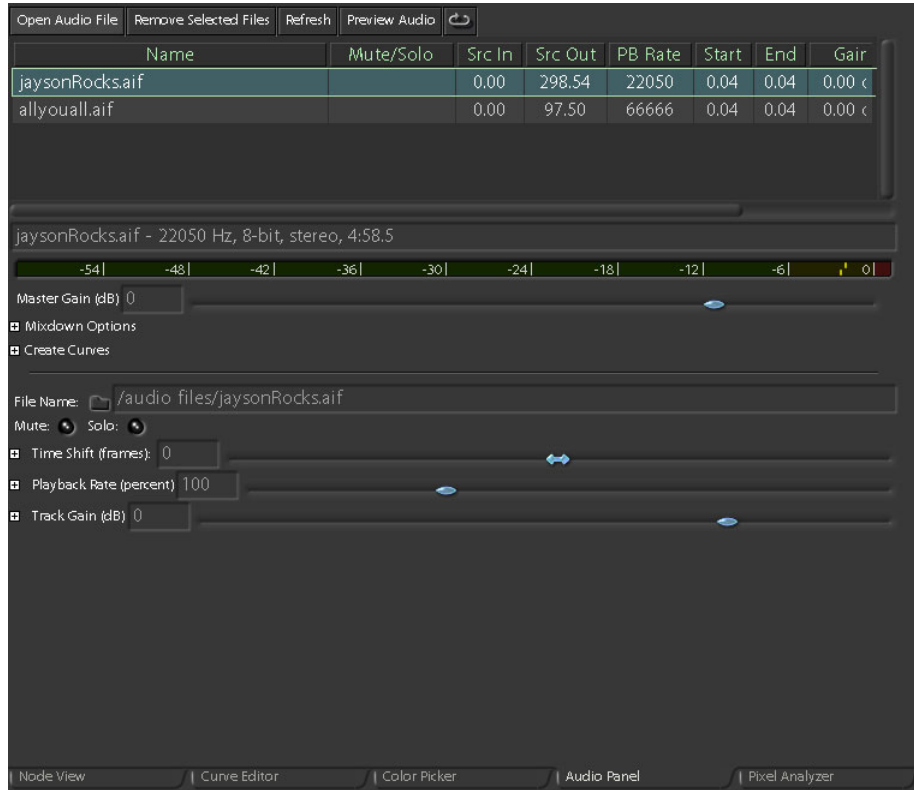
## About Audio

Shake supports PCM AIFF and PCM Wave files. You can import the audio files, mix them together, extract curves based on frequency, manipulate the timing of the sound, and save the files out again. These audio curves can be visualized in the Curve Editor. Because audio playback is handled through the use of Macintosh-specific QuickTime libraries, you can only hear audio playback on Macintosh OS X systems. You can still analyze and visualize audio in Linux and IRIX. On systems supporting QuickTime, audio may also be played with video.

Although multiple frequencies and bit-depth importation is supported, playback is always 44.1 kHz, 16 bit at Medium quality. Export is always done with the Highest quality, corresponding to a 27-point symmetric Kaiser-windowed sinc interpolation.

To access the audio controls, click the Audio Panel tab (located in between the Color Picker and Pixel Analyzer tabs).

## The Audio Panel

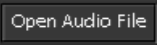


The first row of buttons on the top of the audio panel controls loading, removing, refreshing, and previewing .aif (.aiff) and .wav files.

### Loading and Refreshing Audio Files

You can import AIFF and Wave files into Shake. Also, if changes are made to the audio file outside of Shake, you can refresh the sound file listing.

#### To load an audio file:

- 1 Click the Audio Panel tab.
- 2 In the Audio Panel, click Open Audio File .
- 3 In the Select Audio File window, select the audio file(s) to import, and click OK.

**Note:** You can also double-click an audio file to import the file.

The audio file is loaded as a track.



### To remove an audio file:

- 1 In the Audio Panel, select the audio file.  
**Note:** You can drag-select or **Shift**-select multiple files.
- 2 Click Remove Selected Files  , or press **Delete** / **Backspace**.  
The audio track is removed.


### Previewing Audio

You can use the Preview Audio feature to listen to and edit the audio tracks in real time. You can also loop an audio track in a designated time range. The preview tool is independent of any footage. To view or sync an image sequence with audio, use the Audio Playback button on the Time Bar. For more information, see “Enabling, Viewing, and Editing Audio” on page 92.

### To preview an audio file:

- 1 To load the audio file, follow steps 1 through 3 in the “To load an audio file” section, above.
- 2 Click Preview Audio  .  
The Audio track plays in real time.  
To stop the preview playback, click Stop Preview  .

### To loop the clip:

- 1 In the Globals tab, enter the time range for the loop in the timeRange parameter.  
**Note:** If the time range is not specified in the Globals, the preview continues to play (beyond the end of the actual audio track) until you click Stop Preview.
- 2 Toggle Loop to enabled  .
- 3 Click Preview Audio.

The track is repeated in the designated time range.

The information for a selected audio file appears in the text field below the audio track list. The peak meter is located below the audio file list and information text field.

### Secondary Peak Meter

You can enable a small peak meter next to the Load/Save script buttons by entering the following line in a *ui.b* file:



```
gui.showTopPeakMeter = 1;
```

## Enabling, Viewing, and Editing Audio

This section discusses the tools associated with enabling audio playback with footage, viewing audio waveforms, and editing audio tracks.


**Note:** Depending on your hardware configuration, audio playback does not necessarily sync with the frame indicator in the Time Bar.


To enable audio playback (associated with footage):

- Click Audio Playback  (on the Time Bar) and click Play .

**Note:** You can scrub the audio at any time by holding **Command** / **Ctrl** and dragging on the Time Bar.


To view an audio waveform in the Curve Editor:

- In the Curve Editor tab, click the Draw Waveform toggle .

When enabled , the waveform is displayed in the Curve Editor background.



To slip a single audio track in time, do one of the following:


- In the Audio Panel, select the track in the sound file list and enable solo . The waveform for the track is redrawn in the Curve Editor. Press **Shift+Opt** / **Shift+Alt** and drag the mouse in the Curve Editor.
- In the Audio Panel, enter the slip amount in the Time Shift text field.

To slip all audio tracks in time:

- Press **Shift+Opt** / **Shift+Alt** and drag the mouse in the Curve Editor.

**Note:** Although multiple tracks can be highlighted in the sound file list (and deleted), you cannot selected multiple tracks in the sound file list for slipping, muting, or setting to solo. Only the highlighted track with the line around it is actually selected.

### Refreshing Audio

If an audio file is on an offline server or drive when the script is loaded, the audio file appears red in the sound file list. Once the server or drive is online, click Refresh  to reload the audio file.

## Audio File Parameters

The following table lists the set of parameters associated with every audio track.

Name	Notes
<i>Mute/Solo toggles</i>	Disable track, or disable all other tracks except the current track.
<i>Time Shift (frames)</i>	Sync slip control in frames. To interactively drag sound in the Curve Editor, press <b>Shift+Opt</b> / <b>Shift+Alt</b> and drag in the Editor.
<i>Time Shift (seconds)</i>	Sync slip control in seconds.
<i>Source In (seconds)</i>	Beginning point of the clip, listed as seconds.
<i>Source Out (seconds)</i>	End point of the clip, listed as seconds.
<i>Start Time (seconds)</i>	Beginning point of the clip, listed as frames.
<i>End Time (seconds)</i>	End point of the clip, listed as frames.
<i>Playback Rate (percent)</i>	A scaling control for time to speed up or slow down playback.
<i>Playback Rate (Hz)</i>	Same as playback rate, but allows you to enter a specific cycle rate.
<i>Track Gain (dB)</i>	A decibel gain.
<i>Track Pan</i>	Adjusts the stereo effect by panning a track toward or away from one ear.
<i>Track Wide</i>	Controls how much uniqueness exists in the left and right channels, giving a feeling for wider stereo.

## Mixing and Exporting Sound

You can control how sound files are mixed together in the Mixdown Options subtree. Once finished, you can export the result to disk.

### Master Gain

Use the Master Gain parameter (located below the peak meter) to control the overall gain of the final mixdown.



**Note:** To adjust gain on individual tracks, use the TrackGain parameter.

## Mixdown Options

Name	Notes
<i>Update from Globals</i>	Indicates if timing for the audio export should come from script Globals or from the audio tab. Pressing update now reads settings from the Globals.
<i>Time Range</i>	The amount of frames to be exported.
<i>Mixdown File Name</i>	The output filename.
<i>Sample Rate, Bit Depth</i>	The output sample rate and bit depth of the output file.
<i>Resampling quality</i>	When input clips are adjusted and scaled in time, their sound samples must be interpolated to the output sampling rate. How closely this is done is determined by the quality parameter. "Highest" should be used if intended for broadcast. For the technically minded, this corresponds to a 27-point symmetric Kaiser-windowed sinc interpolation.
<i>Clipping Info</i>	When lighted, indicates that the right or left audio channel is clipping.
<i>Save Mixdown</i>	In effect, the Render button to save the file to disk.
<i>Track Wide</i>	Controls how much uniqueness exists in the left and right channels, giving a feeling for wider stereo.

## Extracting Curves From Sound Files

The Create Curves subtree gives you control over sampling of the audio to be converted into a curve. These curves appear in the Globals tab as a Local Parameter, and can then be referenced by other functions in Shake using standard expressions.

### Create Curves Options

Name	Notes
<i>Update from Globals</i>	Indicates if timing for the audio export should come from script Globals or from the audio tab. Click "update now" to read settings from the Globals.
<i>Time Range</i>	The frame range of the audio to be extracted.
<i>Interval (frames)</i>	A key is entered every N frames, controlled by the Interval parameter.

Name	Notes
<i>Type:</i>	"Peak" means the generated value is the highest absolute value the waveform reaches during that interval. RMS ("Root Mean Square"), on the other hand, uses the square root of the mean of the squares on the absolute values of the waveform over the interval. In other words, each absolute displacement value in the interval is squared, the average of all those squared values is calculated, and its square root is taken as the representative value for that interval. The RMS method is said to be more representative of the actual perceived volume of an interval of digital audio.
<i>Logarithmic (dB)</i>	<p>When deactivated, the returned value is the actual value from the audio file. The sample values are normalized from -1.0 to 1.0, with (1.0 = 0 dBFS). Therefore, the usual range in this mode is 0.0 to 1.0. Values may exceed 1.0 if the audio is clipping (exceeding 1.0, or 0 dBFS).</p> <p>If "logarithmic" mode is on, the value generated is converted to a dB scale, normalized linearly over 90 decibels from 0.0 to 1.0. A value of 0.0 would thus represent -90 dBFS (or lower), and 1.0 would represent 0 dBFS.</p> <p>For example, if in peak mode, and the peak audio value over an interval is 0.5 (approx. -6 dBFS), the value entered with logarithmic mode off would be 0.5. With logarithmic mode on, it would be <math>(-6 + 90) / 90 = 0.9333</math>.</p>
<i>Create separate left/right curves</i>	Either one curve is created, or left and right channels are sampled and two curves are created.
<i>Lowpass Filter</i>	Activates the LPF, as described below.
<i>Lowpass Filter Freq (Hz)</i>	This determines the cutoff frequency for the lowpass filter. All frequencies at or above this frequency are cut off completely when the filter is activated. This setting is useful for analyzing strictly low-frequency content, such as bass drums, rumble, earthquakes, and other subwoofer-shaking phenomena. When the filter is active, its effect may be heard during playback. (Use "Preview Audio" to hear the effect of sliding the filter freq in real time.) The lowpass filter setting does not affect mixdown files or QuickTime renders.

## Global Parameters

The Global Parameters affect the behavior of the entire script. These parameters include setting proxy information and global motion blur controls. Many of these parameters can be set in the command line, so you do not necessarily have to reset them each time you write out a script. For example, your timeRange may be 1-10, but you can modify that when you render on the command line with the *-t* option:

```
shake -exec my_script -t 1-240
```

## Accessing the Global Parameters

To access the Global Parameters, click the Globals tab or double-click an empty area in the Node View. The Global Parameters appear in the Parameter View. The Globals also appear when Shake is started.

## Default Values for Global Parameters

The following are the default values of the Global Parameters, as listed out in the saved script. See Chapter 17, “Customizing Shake,” for information on how to set these.

```
SetTimeRange("1");  
SetFieldRendering(0);  
SetFps(24);  
SetMotionBlur(1, 1, 0);  
SetQuality(1);  
SetProxyScale(1, 1);  
SetProxyFilter("default");  
SetPixelScale(1, 1);  
SetUseProxyOnMissing(1);  
SetDefaultWidth(720);  
SetDefaultHeight(486);  
SetDefaultBytes(1);  
SetDefaultAspect(1);  
SetDefaultViewerAspect(1);  
SetTimecodeMode("24 FPS");  
SetDisplayThumbnails(1);
```

## The Global Parameters

Name	Notes
<i>cacheMode</i>	<p>Sets the caching mode. No kidding. The cache is a directory or precalculated images with script information attached. When Shake evaluates a node tree, it compares the tree to the cache to see if it has done this before. If so, by default, it calls up the cached image rather than recalculate. For more information on caching, see "Memory and the Cache" on page 637.</p> <p>none: No cache is read or written.</p> <p>read-only: Cache is read from, but not written to.</p> <p>regular: Cache is used completely, but it only writes non-animated values.</p> <p>aggressive: As regular, but animated values are also cached.</p>
<i>ColorSettings</i>	<p>Controls the color for the text and noodle colors, as well as setting the drawing tension of the noodles.</p>
<i>defaultAspect</i> <i>default Bytes</i> <i>defaultHeight</i> <i>defaultViewerAspectRatio</i> <i>defaultWidth</i>	<p>See "Default Format Choices" on page 101 for more information.</p>
<i>defaultZoom</i>	<p>When you change the proxyScale, you are changing the actual output resolution. Since the proxyScale is used as a tool to speed up testing, and not as a resizing tool, defaultZoom is set equal to proxyScale to keep the visualized image in the Viewer at the same screen size. This is simply a hardware zoom. If you want to disable this behavior, set it to 1.</p>
<i>displayThumbnails</i>	<p>See Chapter 4, "Compositing and the Node View."</p>
<i>dropFrame</i>	<p>Drops 30 frames to 29.97 frames for timecode purposes when activated.</p>
<i>fieldRendering</i>	<p>Activates field-based rendering. When 0, full frames are rendered. When set to 1, the odd field takes precedence, meaning it is the first line at the top. This is usually for PAL images. When set to 2, the even field is dominant. This is usually for NTSC images. See Chapter 6, "File Formats and Footage," for more information.</p>

Name	Notes
<i>format</i>	<p>Calls up a preset format with this list to set your defaultWidth, Height, etc. These settings are only for Shake-generated image nodes that you create in the future—setting this does not change any nodes that already exist. They also do not set the output resolution. Typically, you set this when you first start your project. Every time thereafter you create something like a <i>RotoShape</i>, it inherits this resolution.</p> <p>See the table of "Default Format Choices" on page 101.</p>
<i>framesPerSecond</i>	<p>Sets the default playback rate of the launched flipbook. To change the rate in the Flipbook itself, press + and – (on the number pad). The frame rate is displayed at the top of the Flipbook.</p>
<i>gridEnabled,</i> <i>gridWidth, Height</i> <i>gridVisible</i>	<p>Places nodes on a grid in the Node View, creating an organized tree layout. The size and visibility of the grid are controlled with the other grid parameters, gridWidth, gridHeight, and gridVisible. See also layoutTightness.</p>
<i>interactiveScale</i>	<p>Similar to proxyScale, but applied only when you are doing an onscreen transform. The image uses lower-res samples, set to the interactiveScale (for example, .5 is half resolution), and then pops back to your normal proxyScale when you complete a move. Note that this value is a factor of the proxyScale, so a value of .5 on the interactiveScale with a proxyScale value of .5 means .25 of the total resolution. See "About Proxies" on page 103 for more information.</p>
<i>layoutTightness</i>	<p>Controls the vertical distance between nodes when new nodes are created, or when you select a group of nodes and press <b>L</b> to clean up the layout.</p>

Name	Notes
<i>macroCheck</i>	<p>If you open or load a script on your system, and the script does not appear, a potential problem is that the script contains macros that are not on your system. If this is the problem, a message similar to the following appears in the Console:</p> <p><i>line 43: unknown function MissingMacro</i>  <i>MissingMacro1=MissingMacro(Keylight_1v41);</i></p> <p>To open or load a script that contains a missing macro(s):</p> <ul style="list-style-type: none"> <li>■ Click the Globals tab.</li> <li>■ Expand the guiSettings subtree.</li> <li>■ Select one of the following options: <ul style="list-style-type: none"> <li>abort load – does not load the script</li> <li>sub. with text – substitutes a <i>Text</i> node in place of the missing macro</li> <li>sub. no text – substitutes a <i>MissingMacro</i> node</li> </ul> </li> <li>■ Open/Load the script.</li> </ul> <p>To set the default macroCheck behavior to substitute a <i>MissingMacro</i> node, include the following in a .h file:</p> <pre>sys.useAltOnMissingFunc = 2</pre> <p>For more information on .h files, see "Locations for .h Files and Custom Icons" on page 624.</p>
<i>maxThread</i>	Tells Shake how many processors to use, and is set by default to the maximum amount on your system.
<i>motionBlur</i>	<p>A global control for all transform node motionBlur parameters. Each node has its motionBlur parameter multiplied by this number. Therefore, if this number is 1, all transform parameters are left alone. If the number is 0, all motion blur is effectively turned off.</p> <p>By setting it to a number between 0 and 1, you control the amount of samples taken, speeding up the process.</p>
<i>noodleTension</i>	Sets the drawing tension of the noodles in the Node View.
<i>pixelRatio</i> <i>pixelScale</i> <i>proxyFilter</i> <i>proxyRatio</i> <i>proxyScale</i>	See "About Proxies" on page 103 for more information.

Name	Notes
<i>quality</i>	When this is set to 0, anti-aliasing is disabled, a poor-grade filter is used, and transformations are set to integer values to avoid subpixel sampling. This is the same as using -fast. This considerably speeds up your processing. Once you are finished with the script, boost quality back to 1 and save your script.
<i>shutterOffset</i>	This value is added, not multiplied, to each individual transform node. It sets the beginning frame, relative to the current frame, that motion is calculated for blurs.
<i>shutterTiming</i>	Similar to the global motionBlur, this modifies all individual transform node shutterTiming parameters. A shutterTiming of 1 equals 360 degrees, .5 equals 180 degrees, etc. A value of 0 turns off motion blur.
<i>thumbnail parameters</i>	See Chapter 4, "Compositing and the Node View."
<i>time</i>	Linked to the Time Bar to list the current frame. It is only used for the interface, since the script and the command line both depend on the timeRange parameter to determine the current frame.
<i>timecodeMode</i>	Sets how timecode is calculated, either as 24 fps, 25 fps, 30 fps drop frame, or 30 fps non-drop frame.
<i>timeRange</i>	<p>Sets the range of frames for the script. The Flipbook reads this parameter by default when rendering out to see how many frames to render, and is saved with the script as well. This parameter can be overridden in the command line. See "About Time" on page 84 for more information on the syntax.</p> <p>Click the Auto button to set the bounding box of all of your clips, for example, from the earliest frame in any clip to the last frame in any clip.</p>
<i>useProxy</i> <i>useProxyOnMissingFile</i>	See "About Proxies" on page 103 for more information.
<i>viewerAspectRatio</i>	When set to formatDefault, scales the X-size of the Viewer by format's defaultViewerAspectRatio. When set to custom, it is set to whatever value you set. This is usually used to compensate for video distortion. For cinema frames, you typically use the proxyRatio to scale down in Y.
<i>viewerZoom</i>	The zoom level applied to the Viewer. Does not affect output resolution.

Name	Notes
<i>virtualSliderMode</i>	When set to zero, it assumes you are using the mouse. When set to 1, it assumes you are using a stylus. It also allows you to use the virtual sliders in the text fields simply by dragging the left-mouse button (without pressing <b>Ctrl</b> ).
<i>virtualSliderSpeed</i>	Adjusts the speed of the virtual slider. When using a stylus, set it to 0.

## Default Format Choices

Name	default Width	default Height	default Aspect	default ViewerAspect
Academy	1828	1332	1	1
CinemaScope	1828	1556	.5	2
Full	2048	1556	1	1
1.85	1828	1332	1	1
NTSC (D1 4:3)	720	486	1.1111	.9
NTSC (16:9)	720	486	.83333	1.2
PAL (D1 4:3)	720	576	.9380	1.066
PAL (16:9)	720	576	.7032	1.422
PAL (square)	768	576	1	1

Changing any parameter in format sets the format to Custom.

You can create your own formats in a *startup* .h file. In *HOME/nreal/include/startup*, add a line in the following format:

```
DefFormatType("Name", defaultWidth, defaultHeight, defaultAspect,
defaultViewerAspectRatio, framesPerSecond, fieldRendering)
```

For example:

```
DefFormatType("NTSC (D1 4:3)", 720, 486, 1.1111, 0.9, 29.97, 0);
```

## Working With High-Resolution Images

These guidelines are specifically for high-resolution images of 4K and 6K. The following discussion is based on the premise that you have a massive amount of RAM for your interactive workstation (at least 1 GB).

Although Shake works with any resolution, there is a default crop on Viewers in the interface of 4096 x 4096 pixels. This protects the user in case a zoom of 1000 is applied to a 2K plate. Instead of trying to render an enormous image, only the lower-left corner up to 4096 pixels is rendered in the interface. This is fine for normal HD or film production, but the cropping takes effect if you read in 6K IMAX plates. This limitation is only in the GUI—images rendered to disk are at the uncropped full resolution.

You can get around this in two ways. The first is to work with proxies of less than 1. With a proxyScale of .5, you can potentially look at images up to 8K x 8K resolution. The other workaround is to change the default Viewer limits by customizing a *ui* preference file. Add the following lines:

```
gui.viewer.maxWidth = 4096;
```

```
gui.viewer.maxHeight = 4096;
```

These lines set the maximum resolution to 4K. If you want a larger resolution, enter it here.

For more information on these preference files, see Chapter 17, “Customizing Shake.”

Another change you must implement is the cache settings. By default, only images under 2K resolution are cached. This is to keep the cache open enough to add more files by not caching large files. You can override this with the following two lines, which indicate the default values. The first line tells the maximum size by listing the X resolution, the Y resolution, number of channels, and amount of bytes. The second line gives the maximum amount of disk space for the cache directory. You can assume that if you are working on 6K plates, you can allow for more than 512 MB of disk space for your cache. These lines go in your *startup* preference files. You modify the numbers to suit your production situation:

```
diskCache.cacheMaxFileSize = 2048*2048*4*2;
```

```
diskCache.cacheSize = 512;
```

Keep in mind that if you set your maximum file size to 6K x 6K x 4 channels x float, you are saving massive files. The return you have on swapping this in and out of cache is extremely limited, at best. It is recommended you use proxies when interactively working with 4K and 6K images.

If you need to work at full resolution, try putting a *Crop* at the end of the chain to focus on an area of interest or using the Viewer DOD. This retains full pixel resolution, but keeps your image resolution within the framework of your machine.

Finally, you need to tune the amount of RAM used by Shake. By default, 96 MB are assigned to the nodes and 64 MB to the images themselves. You need to increase the second setting. A setting of one-third of your memory dedicated to each of the two following settings to reserve memory for other applications and flipbooks is recommended. However, the first setting rarely needs to go above 96 MB. For example, if you have 1 GB of RAM, you might want to have memory settings like the following:

```
cache.cacheMemory = 96;  
diskCache.cacheMemory = 500;
```

The first line is associated for nodes, and is not affected by image resolution. The second setting is associated with the images themselves, so you want to increase it as your images get larger. The default setting is 64 MB—not useful for large resolutions. These settings also go in your *startup* preference file.

For more information about resolution, see Chapter 6, “File Formats and Footage.” For more information about caching, see “Memory and the Cache” on page 637.

## About Proxies

This section discusses the use of proxies to speed your workflow, including using interactive scale, setting proxies, creating custom proxies, working with offline full-resolution elements, and pregenerating proxies.

### What Are Proxies?

A proxy is a lower-resolution image that you substitute for your high-resolution plates for fast tests. Because the images are smaller, you drastically decrease your I/O time, memory consumption, and processing time. Naturally, your quality suffers as well, which is why proxies are generally for testing purposes. Once you assemble your script with proxies, return your script to full resolution to render your final output. You also use proxies for temporarily viewing anamorphic images in flattened space. For more information, see “About Aspect Ratio and Non-Square Pixels” on page 226.

#### Proxies and Final Low-Resolution Output Renders

When you work with film plates and you need to generate a high-quality video output, you have better (but slower) results if you render your plates at full resolution and then size them down, instead of using the proxies to generate low-resolution images. The proxies can be used to generate lower-resolution output files, but the quality is not as high as with the full resolution.

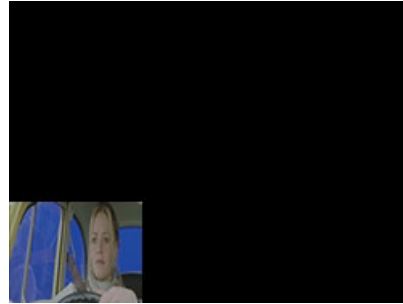
The following example shows a full-resolution image and a 1/3 proxy image. You can see that the proxy uses 1/9 of the space, or only potentially 11 percent, of the processing time, memory required, and I/O activity.

“The Saint” images are used courtesy of Framestore CFC and Paramount British Pictures Ltd.

**Full Resolution**



**1/3 Proxy**



As shown in the following images, there is a dramatic quality difference when the result is viewed at the same resolution.

**Full Resolution**



**1/3 Proxy**



Shake automatically adjusts pixel-based values to compensate for the lower resolution, so a *Pan* of 100 pixels is calculated to be only 33.333 pixels at 1/3 proxy. The actual *Pan* parameters in the interactive text field are not modified.

There are three basic approaches for using proxies that are controlled in the Globals section of the interface:

- **interactiveScale:** Your general speed is fine, but you want to tune a heavy operation a little more quickly. Once it is done, you want to see the full resolution result immediately. This does not affect your flipbooks or *FileOut* renders.
- **useProxy:** You want to speed up processing, but you don't anticipate working on the project for a very long time or with a lot of networked machines.
- **useProxy + prerendering your proxy files:** You are working on a project for a long time, using networked machines. The project is extremely heavy, and your high-resolution files are probably on a remote disk. You are doing many flipbook tests.

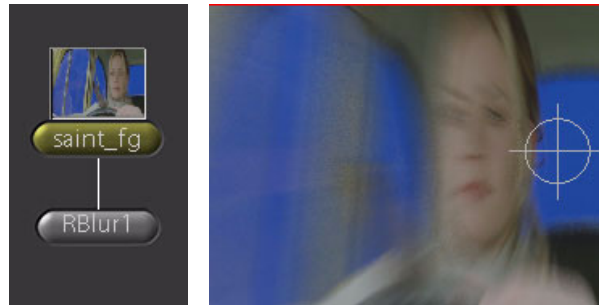
**Note:** The `pixelScale` and `pixelRatio` parameters are generally obsolete due to the new (v2.5) proxy functions in *SFileIn*. They are kept for compatibility purposes.

### Using interactiveScale

This Global Parameter temporarily drops you down to a lower resolution when you are tuning a parameter. As soon as the mouse is released, the rendering snaps back to full resolution. It does not affect your output render or flipbooks. You can combine this setting with the other proxy systems.

The following is an example of using the `interactiveScale` parameter:

- Read in `pix/keylight/saint_fg.1-5#.jpg`.
- Apply a Filter—*RBlur* (not *Blur* or *IBLur*).
- Set your `oRadius` to approximately 300 (note the painfully pokey *RBlur*).



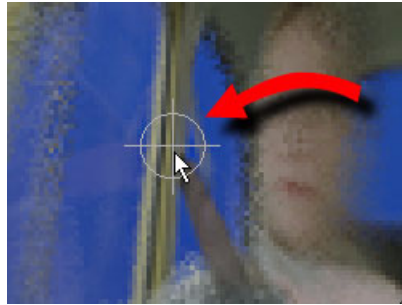
**Note:** For very fast radial blur plug-ins, contact The Foundry and GenArts.

Barring the plug-ins, one way to accelerate your workflow is to go to the Globals, and drop the `interactiveScale` to 1/3.



As you drag the center control around, it drops to a temporary lower resolution. When the mouse is released, it returns to full resolution.

#### Modify a Parameter...



#### ...Release the Mouse



#### Faster RBlur

Another way to speed up interactivity is to drop the *RBlur* quality.

### Setting Proxies

These are the normal proxies (also called on-the-fly proxies) that are generated only when needed and discarded when your disk cache is full. This set of parameters reads your input images and scales the images down, placing them first into memory and then into the cache as memory runs out. It then recomputes the script at the lower values, but, unlike *interactiveScale*, leaves it at that resolution until you return it to full resolution. This setting affects flipbooks and *FileOuts*.

To change to a lower proxy with preset numbers:

- In the Globals, switch *useProxy* from Base to P1, P2, or P3, or use the pull-down button menu in the title bar.

By default, P1 is set to .5, 1 for its scale/aspect ratio; P2 is set to .25, 1; and P3 is set to .1, 1.

- You can alternatively select from the predefined proxy sets in the *useProxy* subtree. These are common proxy settings that automatically modify all proxy values. These can also be customized.

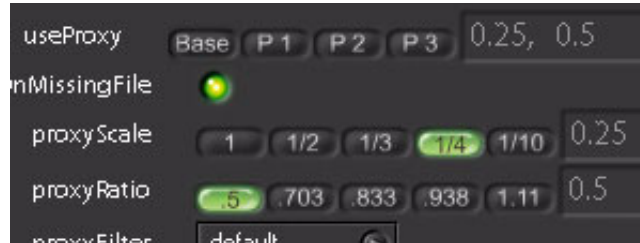
The proxy setting appears in the associated text field.



- When activated, the proxy button in the title bar is highlighted. You can also use this to change your proxy level.

#### To temporarily set a custom proxy:

- 1 Modify the numbers in the useProxy text field, or open the useProxy subtree and modify proxyScale or proxyRatio.
- 2 Click Base/P1/P2/P3 (or Other in the title bar) to return to your presets.

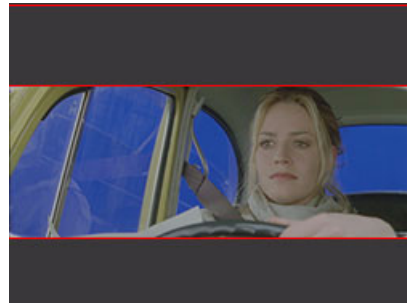


In the following example, the proxyRatio is set to .5. As soon as proxyScale or proxyRatio is modified, the useProxy Base/P1/P2/P3 buttons turn off, since you no longer have a match to a preset proxy setting.

#### Full Resolution



#### .5 proxyRatio



#### Setting the Aspect Ratio to .5

To ensure that your nodes understand the aspect ratio as .5 (for example, *Rotate*), go to the Globals and set your defaultAspectRatio format to .5. This does not modify your image, it just affects functions such as *Twirl* and *Move2D* created after you set the ratio. For more information, see “About Aspect Ratio and Non-Square Pixels” on page 226.

## Customizing P1/P2/P3 for a Script or Session

- Open proxy1DefaultFile (or 2 or 3...).
- Modify the Scale and Ratio parameters.

For example, you want to have the anamorphic *Saint* image as the full resolution. You want P1 the same width as the base file (the full resolution image) but flattened, and P2 1/4 scale and also flattened. Therefore, open proxy1DefaultFile and proxy2DefaultFile and set the parameters as follows:

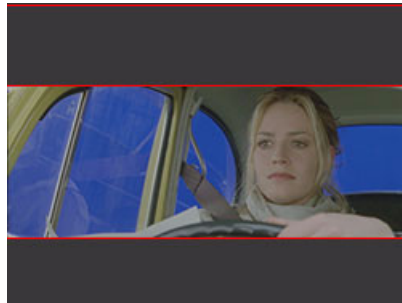
```
proxy1DefaultScale    1
proxy1DefaultRatio    .5
proxy2DefaultScale    1/4
proxy2DefaultRatio    .5
```

**Note:** You can now close the useProxy subtree to clean up the Globals page.

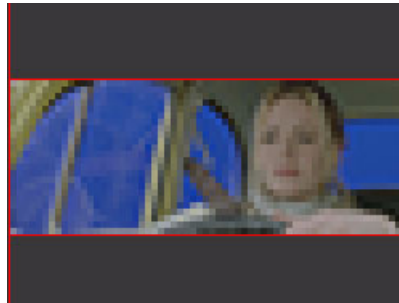


Click P1 or P2 to toggle to the flattened versions of the full-resolution image.

**Modified P1**

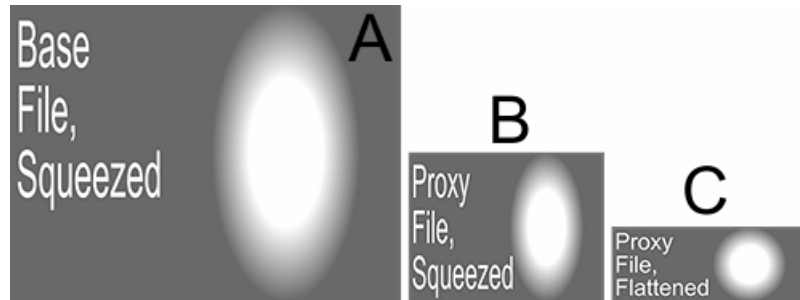


**Modified P2**



## Anamorphic Images and Pregenerated Proxies

Do not use the ratio parameter to change your aspect ratio on the fly if working with pre-rendered proxies. This parameter dictates the relationship of the height-to-width ratio between the proxy file and the base file as they exist on disk. Therefore, either ensure you are using flattened pre-generated proxies (that can be a second proxy set), or use the Global parameter `viewerAspectRatio` to flatten the anamorphic proxies. In the following example, image A is the full-resolution anamorphic frame, so it looks squeezed. Image B is a half-size anamorphic-resolution proxy. Image C is a half-size, flattened image. This is how the raw images appear on disk.



Therefore, your proxy settings should be:

P1 – Image B, scale of .5, ratio of 1; as the ratio of the height to the width is the same as in image A.

P2 – Image C, scale of .5, ratio of .5. The width is the same as image B, the ratio is half of the ratio of the height-to-width of image A.

## Permanently Setting P1/P2/P3

Your custom settings are saved into a script. However, if you always want to use the same settings for new scripts, you can set your own P1/P2/P3 settings by creating a `.h` file in your *startup* directory.

When an *SFileIn* node is created, three pieces of information are taken from the Browser: The filename, the proxy level corresponding to this filename (Base, P1, P2, or P3), and the proxy set to use. The chosen filename is stuffed into the node's `proxy1File` parameter, and the settings for the selected proxy level from the selected proxy set are used to set the other `proxy1Whatever` fields. Next, the remaining proxy set paths defined in the proxy set are filled into the remaining `proxyNWhatever` fields in the *SFileIn* node, with path modifications and substitutions made as defined in the `DefProxyPath` or `DefBasePath` statements for the proxy set. In much the same way that a *Grad* node takes its initial width and height parameter values from the current `defaultWidth` and `defaultHeight` values in the Globals, the *SFileIn* node's parameters are set via the values, modifications, and substitutions defined in the proxy set.

The syntax for defining proxy sets is as follows:

```
DefProxyGroup("proxySet",
DefBasePath("baseDefaultFile", "baseDefaultFileType", defaultAlwaysAdd,
"baseDefaultReplace"),
DefineProxyPath(
const char *proxyPath = "../proxy.50/<base>.<format>",
float scale = .5,
float aspect = 1,
int bytes = GetDefaultBytes(),
const char *fileFormat = "Auto",
int render = 0,
int alwaysAdd = 1,
int index = 1,
string substitutionString
);
```

- *proxyPath*: Default location for pregenerated proxies. (See the example below.) Note that you can use variables to grab strings from the baseName:
  - <base> = image name + frame range
  - <format> = format extension
  - <name> = image name (no frame range)
  - <range> = the frame range
  - <dir1>, <dir2>, etc. = the name of the parent directory, two directories up, etc.
- *scale*: The proxy scale.
- *aspect*: The aspect ratio (the Y-axis scale).
- *bytes*: The default bytes for pregenerated proxies. This does not affect on-the-fly generation of proxies, so you maintain your bit depth if you do not prerender your proxies.
- *fileFormat*: The file format for prerendered proxies. (See below.)
- *render*: Turns on or off the render lights on the Render Proxies menu. When on (1), the files are rendered with the Render Proxies menu. (See below.)
- *alwaysAdd*: When set to 1, an entry is added to a *SFileIn* node at the time of its creation.
- *index*: Sets whether you are P1, P2, or P3 (1, 2, 3). This replaces Shake's default settings, unless you set the index to 0, in which case it is appended.

- *substitutionString*: When the first string is found in the base filename, it substitutes the second string; for example, from the first line, "4096x3112" is substituted by "2048x1556." This string is not always necessary—the proxyPath may already be taking care of differentiating the proxy files if all of the names of the files are the same except for a root directory stating the size of the proxies.

### Example

This example sets a proxy of .25 with an aspect ratio of .5. It takes the default bytes setting, turns on the render light for the Render Proxies menu, adds an entry into an *SFileIn*, and is set as P1:

```
DefineProxyPath("../proxy.25.5/<base>.<format>", .25, .5, GetDefaultBytes(), "Auto",
1,1,1, "substitutionStrings");
```

You can also create and use predefined proxy sets in the useProxy subtree (in the Globals). Selecting it sets values for P1, P2, and P3. The following example assumes filenames such as "4096x3112/name\_4k.#.cin," "2048x1556/name\_2k.#.cin," "1024x778/name\_1k.#.cin," and "410x366/name\_sm.#.cin." To set a group, use the following code in a *startup* .h file:

```
DefProxyGroup("4K Fullap",
DefBasePath("../4096x3112/<base>.<format>", "Auto", 1, "2k | 4k;1k | 4k;sm | 4k"),
DefProxyPath("../2048x1556/.", .50, 1., GetDefaultBytes(), "Auto", 0, 1, 0,
"4k | 2k;1k | 2k;sm | 2k"),
DefProxyPath("../1024x778/.", .25, 1., GetDefaultBytes(), "Auto", 0, 1, 0,
"4k | 1k;2k | 1k;sm | 1k"),
DefProxyPath("../410x366/.", .10, 1., GetDefaultBytes(), "Auto", 0, 1, 0,
"4k | sm;2k | sm;1k | sm")
);
```

*//This sets the default set at startup, so obviously you only set it once.*

```
script.proxySet = "4k Fullap"
```

**Note:** It is best to copy an example from the <ShakeDir>/shake.app/Contents/Resources/nreal.h file and to paste it into a new file.

The first line names the group as "4k Fullap." The next line describes the base filename. The next three lines describe the subproxies using the DefineProxyPath definition, except the substitutionStrings. When the first string is found in the base filename, it substitutes the second string. For example, from the first line, "4096x3112" is substituted with "2048x1556."

## Working With Offline Full-Resolution Elements

It is sometimes convenient to only have prerendered proxies (typically generated during the scanning process) online, and to substitute your full-resolution elements at a later time. Shake's proxy structure allows you to do this. It is assumed that all users in your production pipeline are using a standardized naming convention.

- Open the useProxy subtree in the Globals.
- Select a proxySet, or construct your own (as described in the previous section). This automatically enters names for the file path names for the base file and the proxy sets.
- If you want to substitute a string in the filename, use the Replace parameters. These allow you to scan and replace text in the first set loaded in for the particular set you are working with. For example, your P1 proxies are already online, but your full-resolution elements are not. You therefore start compositing with the proxies and work on the full-resolution elements later. Additionally, the proxies are on the computer "MyMachine" and the full resolution elements are on "Server1." If the proxy was named:

```
//MyMachine/project1/shot1/plate1/proxy1/myfile__proxy1
```

and the full resolution elements are:

```
//Server1/project1/shot1/plate1/full/myfile__full
```

you enter *MyMachine* | *Server1;proxy1* | *full* as your baseDefaultReplace. Note the semicolon to split the entries.

- Create a *FileIn* that is intended to be a prerendered proxy, specify its proxy set in the bottom of the *FileIn* Browser, in the above example, P1.
- When the full-resolution elements are later brought online, you may toggle over to full-resolution mode and the images are pointed to properly.
- Use caution when working with anamorphic elements. The proxy ratio parameter determines the relationship of the proxy to the base file. Therefore, if you load in low-resolution flattened images, ensure your ratio reflects the proper ratio of the height to the width in the base files. See "Anamorphic Images and Pregenerated Proxies" on page 109.

## Pregenerating Your Proxies Inside the Interface

When Shake creates proxies with the useProxy method, the proxies are held in memory. Eventually, memory fills up, so the proxy images are dumped to disk into the cache. However, the cache is a closed set of files viewable only by Shake. Additionally, remote renders do not recognize the cache directory if not explicitly told. Finally, you may have so many files that the cache mechanism becomes overworked. In these cases, it makes sense to pregenerate your proxies when you start the project with an initial rendering process. The proxy files are then pulled from these precalculated images rather than generated on the fly.

You can either pregenerate the files inside the interface, or you can load them after they are created by an external process.

### To pregenerate files in the interface:

- 1 Open the Globals tab and view the useProxy subtree.
- 2 Determine how many proxies you want to use, and what values you want.
- 3 Set the paths for the proxies with the File path. By default, you have:

*../proxy.50/<base>.<format>*

*<base>* = image name + frame range

*<format>* = format extension

*<name>* = image name (no frame range)

*<range>* = the frame range

*<dir1>*, *<dir2>*, etc = the name of the parent directory, two directories up, etc.

Therefore, for *MyAmazingFootage.1-100#.cin*,

*<base>* = *MyAmazingFootage.1-100#*

*<format>* = *cin*

*<name>* = *MyAmazingFootage*

*<range>* = *1-100#*

The above indicates that the files go in a directory at the same level as the source images, are named the same name, have the same frame range, and are in the same format. For example, if the source images are:

*<MyLongPath>/MyHiResImages/plate.#.cin*

you get:

*<MyLongPath>/proxy.50/plate.#.cin*

*<MyLongPath>/proxy.25/plate.#.cin*

You can alternatively specify a directory as:

*../<name>\_half/<base>.<format>*

### Proxies and YUV Files

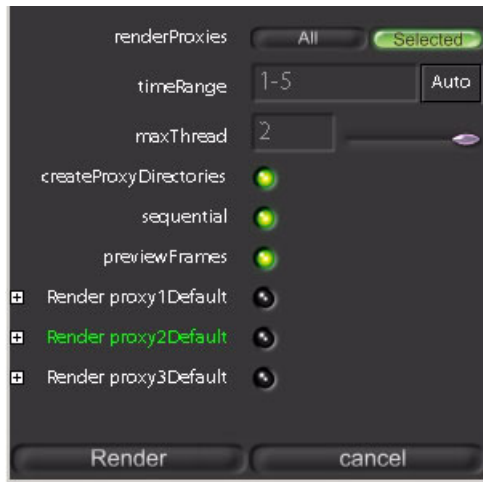
Use caution when making proxies of YUV files, as they are always at a set resolution. Be sure to toggle your FileType to a different format. By default, the proxies are switched to .iff format.

- In the useProxy parameters, open the File subtree and set the scale, ratio, format, and bit depth parameters for the proxy. For example, if you are working with Cineon plates, you may decide to use 8-bit SGI files as your proxies to boost your testing speed.
- When AlwaysAdd is enabled, the file setting is added to a *FileIn* at the time of creation.

- Read in the images with *FileIn*. The proxy paths are automatically loaded for every proxy with AlwaysAdd enabled.
- Select the *FileIns* you want to generate as proxies. If proxies are already rendered, they are rendered again. Shake has no way of checking to see if a proxy is present and/or valid.
- Choose Render > Render Proxies.

The Render Proxy Parameters window appears.

- Make sure the proxies you want to generate are enabled. In the following example, no proxies are rendered since all three sets are turned off:



- *renderProxies*: Specifies whether all or just active *FileIns* are rendered.
- *timeRange*: Sets the range to generate proxies. When Auto is clicked, the entire frame range is calculated. Note proxies are not generated beyond the clip's frame range.
- *maxThread*: The amount of processors devoted to the process.
- *createProxyDirectories*: Creates appropriate directories.
- *sequential*: When generating many files, it may be more efficient to process each file individually, one after the other, rather than simultaneously. This is equivalent to the *-sequential* flag on the command line.
- *previewFrames*: Displays the thumbnails of the frames as they render.
- *Render proxy Defaults*: Must be enabled for the proxy to render. You can also open the subtree to modify any parameters. If you create your own custom proxy setting with a .h file (see above), you can specify if this light is on or off by default.

- Click Render.

### Rendering Proxies on the Command Line

Use *-renderproxies*. You specify your settings, for example, *-renderproxies p1 p3*.

When the render is done, you are loading the pregenerated files rather than generating them on the fly when you activate P1, P2, or P3. In a *FileIn*, the *imageName* indicates the image that is actually loaded. Open the *imageName* subtree to display *baseFile*, *proxy1File*, *proxy2File*, etc. As you toggle the Globals from Base to P1 and P2, the *imageName* changes to reflect the loaded proxy file.

If you have not yet rendered a proxy file, the proxy is generated on the fly using the standard proxy technique. The *imageName* parameter in the *FileIn* turns an odd green color to indicate that it is not valid. In the Globals, the *useProxy* subtree has a parameter called *useProxyOnMissingFile* (on by default) that generates a proxy from the closest available proxy image. This proxy is not saved to disk until you activate Render Proxies.

Note that proxy1, 2, etc., do not necessarily mean they are automatically loaded into P1, P2, or P3. The proxy settings look for the files that are the closest match, so P1 may load files that are created by proxy3.

When specifying your proxy directories, you generally have one of two approaches for where to put your proxies:

- All your images are at the same level, for example:

```
images/bluescreen1/bs1.#.cin,  
images/bluescreen2/bs2.#.cin,  
images/cg_plate/cg_plate.#.iff
```

In the following case, the default path puts all proxies into the same subdirectory:

```
images/bluescreen1/bs1.#.cin,  
images/bluescreen2/bs2.#.cin,  
images/cg_plate/cg_plate.#.iff  
images/proxy.50/bs1.#.cin  
images/proxy.50/bs2.#.cin  
images/proxy.50/cg_plate.#.iff
```

If you have many plates and a high frame count, you may want to provide a file path such as:

```
../<name>_p.50/<base>.<format>
```

This gives you:

```
images/bluescreen1/bs1.#.cin,  
images/bs1_p.50/bs1.#.cin  
images/bluescreen2/bs2.#.cin,  
images/bs2_p.50/bs2.#.cin  
images/cg_plate/cg_plate.#.iff  
images/cg_plate_p.50/cg_plate.#.iff
```

The above approach keeps your file count down in each directory, but increases the amount of directories.

- All of your images are in subdirectories based on resolution, for example:

```
images/bluescreen1/2048x1556/bs1.#.cin  
images/bluescreen2/2048x1556/bs2.#.cin  
images/cg_plate/2048x1556/cg_plate.#.iff
```

In this case, the default naming scheme works fine:

```
../proxy.50/<base>.<format>
```

This gives you:

```
images/bluescreen1/2048x1556/bs1.#.cin  
images/bluescreen1/proxy.50/bs1.#.cin  
images/bluescreen2/2048x1556/bs2.#.cin  
images/bluescreen2/proxy.50/bs2.#.cin  
images/cg_plate/2048x1556/cg_plate.#.iff  
images/cg_plate/proxy.50/cg_plate.#.iff
```

You can of course use

```
../1024x778/<base>.<format>
```

as your path to return numerical values for a half-proxy.

You can also use the `<dirN>` variable to access the name of any parent directory. For example:

```
<dir2>/1024x778/<dir2>.<range>.<format>
```

creates:

```
images/bluescreen1/1024x778/bluescreen1.#.cin
```

### Full-Resolution Proxies and Network Rendering

You can use local copies of your full-resolution images by setting P1 (or whatever) to 1, 1. When you jump to P1, the local copies are used. When Base is used, the network copies can be used. This helps you do faster network renders without drawing from your machine, and also allows you to have faster local renders on your own machine. To specify this on the command line, use either

```
shake -proxyscale Base
```

to use the original files or

```
shake -proxyscale 1 1
```

to use the local full-resolution copies.

### Example:

This example also uses the images in `doc/pix/keylight`. Do not read the images in yet.

- In the Globals, expand the useProxy subtree.
- For proxy1, set defaultScale to 1 and defaultRatio to .5.
- For proxy2, set defaultScale to .25 and defaultRatio to .5.
- Since these are the tutorial images, it is likely (if you are at a large facility) that you do not have permissions to create files and directories in the install directory. Therefore, set the proxy directories to be in an open user area. (Typically, you do not save your proxies into the TEMP directory, however.)

Set your proxy1File to:

```
/TEMP/saint_p.1x.5
```

Set your proxy2File to:

```
/TEMP/saint_p.25x.5
```

- Open the proxy1 and 2 subtrees and set scale to 1, ratio to .5, and format to .iff. Set proxy 2, to 25 and .5, also .iff.
- Open proxy3 and disable AlwaysAdd (you only need two proxy sets).
- Load in images `doc/pix/keylight/saint_fg.1-5#.jpg` and `saint_bg.1-5#.jpg`.
- Choose Render > Render Proxies.

- Set your frame range to 1-5.
- Enable proxy 1 and proxy 2 for render.
- Click Render.

Your proxies are rendered and available for use. When you click P1, you switch to the half-height images. When you click P2, you switch to the quarter-resolution, half-height images. When you click P3, you are using 1/10 resolution images, but these are generated on the fly, as they have not been prerendered.

## Pregenerating Your Proxies Outside the Interface

There are two methods to generate proxies outside the interface.

- If the base-resolution images are already loaded into a script and you have ensured the proxy paths are correct (see above), you can launch a proxy-only render on the command line with the *-renderproxies* command:

```
shake -exec myscript.shk -renderproxies p1 p2 p3 -t 1-100 -v -createdirs
```

This automatically creates the appropriate subdirectories when *-createdirs* is activated. There is no checking for file status, so all images are still rerendered, even if they already exist. Also, each sequence is only rendered for its actual length, so a five-frame sequence is not rendered out to 100 frames.

The command to specify your proxies looks like the following example, and can be in a *startup*.h file or in a script. Its format is identical to what is listed above:

```
DefineProxyPath("../proxy.25.5/<base>. <format>", .25, .5, GetDefaultBytes(), "Auto", 1,1,1);
```

- If you only have the raw image files, you can use the *-z* or *-zoom* functions to render your images:

```
shake fullres.#.cin -z .5 -fo halfres.#.cin -t 1-100 -v
```

```
shake fullres.#.cin -zoom .5 .5 -fo halfres_halfheight.#.cin -t 1-100 -v
```

```
shake fullres.#.cin -zoom .5 .5 -bytes 1 -fo halfres_halfheight_8bit.#.sgi -t 1-100 -v
```

The first command renders half-resolution Cineon files. The second renders half-resolution flat files (to squeeze scope images). The third command does the same, but writes out 8-bit SGI files.

### Once generated, load your proxies into the interface:

- If you followed the first choice from above, no further work is needed, as all paths are determined by the default proxy settings and are already saved into the script.

- If you followed the second choice, read in your full-resolution image with a *FileIn*. There is a line at the bottom of the Browser to indicate if this image is Base, P1, P2, or P3. In this case, the Base is fine.




### Keeping High-Resolution Elements Offline

If you have not yet loaded your full-resolution images onto a disk and are loading a proxy into the interface, Cancel out of the *FileIn* and set the format to the base resolution in the Globals tab. This helps to automatically calculate the proxy size. Next, return to the *FileIn* and indicate the proxy set of your image with the Browser. Later, when the full-resolution elements go online, you can load in your elements with the *FileIn*. If your layer does not have a full-resolution element the same size as Format, you must manually adjust the scale and ratio parameters for the proxy set of that *FileIn*.

- In the *FileIn* parameters, expand the *imageName* subtree. The *baseName* is already supplied. The *proxy1File* probably has an incorrect default name. Therefore, click on the folder to browse to the proxy location. You do not need to indicate if it is P1, P2, or P3 in the Browser.
- There are settings for the scale and ratio of the proxy in the *proxy1File* subtree. These should be automatically set in relation to the full-resolution image (as long as the full image is already loaded). Otherwise, it is calculated according to the Format width and height.
- When you toggle the Globals proxy from Base to P1, P2, or P3, you do not necessarily load a *FileIn*'s corresponding proxy1, 2, or 3. The proxy mechanism loads the set that is closest to the Global settings, doing a further scale if necessary. For example, if you haven't loaded a 10 percent pregenerated proxy and you jump to P3 on the Globals, a 10 percent file is generated from the generated P2 proxy.

### Compatibility of Proxies With Other Compositing Functions

Proxies are fine for gauging color and relative position. However, proxies do not work well for pixel-sensitive nodes such as *DilateErode* (where you may chew just one pixel in or out), or for tracking due to round-off errors. This is not true for all filters. *Blur*, for example, works fine at a proxy resolution.

To increase your refresh speed while working on fine detail, activate the Viewer DOD button . For more information, see “The Domain of Definition (DOD)” on page 50.

Do not use proxies for tracking. (If you want really bad tracks, then of course feel free to use them.)

Proxies also interfere with Z-depth compositing, as there is no anti-aliasing with the Z channel. The left image is a full-resolution Z composite. The right image is the same composite at a proxy resolution. The anti-aliasing of the depth channel significantly alters the image quality.

**Full Resolution**

**Proxy Image**



This quality loss can be somewhat minimized by using a Box filter for your proxies, but then the rest of your image quality suffers in the lower resolutions.

**Proxy Parameters List**

**Globals Tab**

Parameter	Notes
<i>useProxy</i>	Specifies the proxy set to be used. The sizes are determined by opening proxy1File, 2, etc., and setting the scale/ratio parameters. The two numbers are the scale and ratio of the proxy.
<i>useProxyOnMissingFile</i>	When active, substitutes a proxy generated from an associated proxy file when a missing image is encountered.
<i>proxyScale</i>	A temporary setting (though it is saved into a script) for the proxy resolution that is overwritten when useProxy is set. If setting this parameter matches up with a proxy set, the proxy set is automatically activated.
<i>proxyRatio</i>	A temporary setting (though it is saved into a script) for the squeeze on the Y axis to compensate for non-square pixel images that are overwritten when useProxy is set. If setting this parameter matches up with a proxy set, the proxy set is automatically activated.

Parameter	Notes
<i>proxyFilter</i>	The filter used in the sampled images. The default is generally fine, although you may want to switch to Box when working with Z-depth files.
<i>pixelScale</i>	An obsolete function to scale all pixel values.
<i>pixelRatio</i>	An obsolete function to scale all Y pixel values.
<i>proxyXDefaultFile</i>	The default file path for that proxy set. Relative paths are relative to the image read in with a <i>FileIn</i> .
<i>baseDefaultFile</i>	This is used when you bring in prerendered proxies before loading in the full resolution elements. It is assumed you are using a standardized naming convention. Therefore, by using this naming convention, Shake can establish the name of the full resolution elements, based on the name of the proxy you supplied.
<i>baseDefaultFileType</i>	The anticipated file type of the full resolution element.
<i>baseDefaultAlwaysAdd</i>	Whether to add this into the <i>FileIn</i> .
<i>baseDefaultReplace</i>	This cryptic little devil allows you to replace strings in the first loaded proxy set to be replaced by a second string, taking the format: <i>source   replace;source   replace</i> . For example, if you always have <i>_lr</i> at the end of a low-resolution filename and <i>_br</i> at the end of a high res, you could use <i>_br   _lr</i> to automatically change <i>myfile_br</i> to <i>myfile_lr</i> for that proxy set.
<i>proxyXDefaultFile</i>	The default file path for that proxy set. Relative paths are relative to the image read in with a <i>FileIn</i> .
<i>proxyXDefaultScale</i>	The setting for that proxy set, also setting P1, P2, etc. For example, proxy1 is P1.
<i>proxyXDefaultRatio</i>	The Y scaling for that proxy set. If working with pre-rendered elements and with anamorphic elements, ensure this setting reflects the height-to-width relationship between the proxy and base files as they actually exist on disk. See "Anamorphic Images and Pregenerated Proxies" on page 109.
<i>proxyXDefaultType</i>	When prerendering your proxy files, they are stored in this format. This has no effect with on-the-fly proxies.
<i>proxyXDefaultBytes</i>	The bit depth for prerendered proxies. This has no effect with on-the-fly proxies.

Parameter	Notes
<i>proxyXDefaultAlwaysAdd</i>	When enabled, this proxy set is added to a <i>FileIn</i> node when created.
<i>proxyXDefaultReplace</i>	See <i>baseDefaultReplace</i> .
<i>interactiveScale</i>	Sets a temporary proxy setting that is active only when modifying a parameter. Fully compatible with the other proxy files.

## FileIn

Parameter	Notes
<i>baseFile</i>	The full-resolution image on which other images are based.
<i>proxyXFile</i>	The directory for pregenerated proxies. This is ignored for on-the-fly proxies.
<i>proxyXScale</i>	The scaling factor for that proxy set.
<i>proxyXRatio</i>	The Y scaling factor for that proxy set, used for anamorphic files.
<i>proxyXFileType</i>	The file type for pregenerated proxies. Ignored for on-the-fly proxies.

## Render > Render Proxies Window

Parameter	Notes
<i>renderProxies</i>	Determines whether all <i>FileIns</i> or just active <i>FileIns</i> are rendered.
<i>timeRange</i>	Sets the frame range to generate proxies. When Auto is clicked, calculates the entire frame range of the script.
<i>maxThread</i>	The amount of processors devoted to the process.
<i>createProxyDirectories</i>	Creates appropriate directories when rendering.
<i>sequential</i>	When generating many files, it may be more efficient to process each file individually, one after the other, rather than simultaneously. This is equivalent to the <i>-sequential</i> flag on the command line.
<i>previewFrames</i>	Displays the thumbnails of the frames as they render.
<i>Render proxy Defaults</i>	Must be enabled for the proxy to render. You can also open the subtree to modify any parameters. If you create your own custom proxy setting with an .h file (see above), you can specify if this light is on or off by default.

## Command-Line Options

Parameter	Notes
<i>-renderproxies</i>	Only renders the proxy subsets of the <i>FileIn</i> nodes. If no subsets are specified, what is saved in the script is rendered. Otherwise, you can use <i>-renderproxies p1 p2 p3</i> .
<i>-proxyscale</i>	You can specify numerical scale and ratio parameters, or use the key words <i>Base</i> , <i>p1</i> , <i>p2</i> , <i>p3</i> .
<i>-createdirs</i>	Creates directories for the <i>-renderproxies</i> command. Does nothing for normal <i>FileOut</i> nodes.



# Compositing and the Node View

## Chapter Summary

- About Node-Based Compositing
- About Channels
- About the Node View and the Tool Tabs
- About the Infinite Workspace

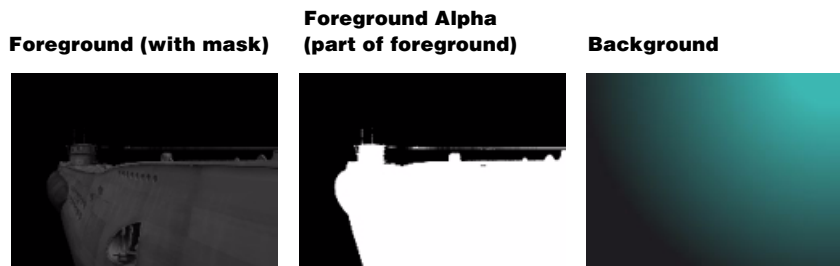
## About Node-Based Compositing

The Node View in Shake displays all of the nodes in a script. Each node has a specific function, and is connected to other nodes by lines referred to as “noodles.” The noodles between each node represent the flow of information between the functions. The output of one node is the input of the following node in the node tree. The functions are listed (and selected) in their respective groups in the Tool Tabs in the lower-left area of the interface.

## Compositing and the Alpha Channel

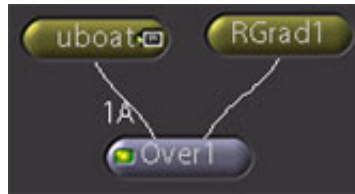
Shake’s compositing nodes are located in the Layer tab. The primary compositing nodes are *Over*, *Layer*, and *KeyMix*. For more information on the specific layer functions, see Chapter 5, “Compositing: The Layer Nodes,” on page 153.

The following images are used in this discussion to demonstrate the primary compositing nodes:

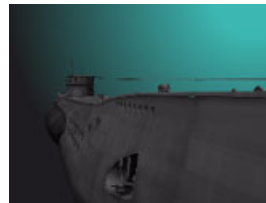


The *Over* node places a foreground image with an alpha channel over a background image. The foreground RGB channels should be premultiplied by the alpha channel. In a premultiplied image, the RGB channels are multiplied by the alpha channel. Premultiplication plays a very important role in compositing, and Shake gives you explicit control over premultiplying and unpremultiplying your images. For more information on premultiplication, see Chapter 9, “Color Correction: Part II,” on page 349.

#### Tree With Over Node



#### Result



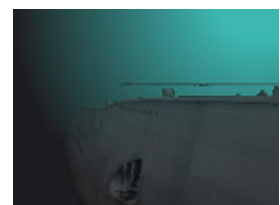
If the image is not premultiplied, the premultiplication can be done in two ways:

- Add a Color-*MMult* before the *Over* node in the process tree.
- Use the preMultiply toggle in the *Over* node's parameters.

The *Layer* node is identical to the other nodes with one exception—you can toggle between different compositing modes within the *Layer* node. Because the node is automatically named “*Layer*” by default, you cannot quickly scan your tree to determine the operation type.

**Note:** You can rename a node in the Parameters1 tab with the nodeName text field.

*KeyMix*, the last primary node, mixes a foreground input image and a background input image through a third input image—the mask. You can select what channel of the third image works as the mask. You can also invert the mask and control its intensity.



As mentioned, a successful *Over* composite requires an alpha channel for the foreground and foreground RGB channels that are premultiplied by that alpha channel. 3D-rendered elements are almost always premultiplied. Scanned elements or other 2D-generated plates require an added alpha channel (also called the matte or mask channel), that is then used to premultiply that image with the Color-*MMult* node. To get the necessary alpha channel, you have several options:

- Pull a key with a Shake keying node (or combination of nodes).

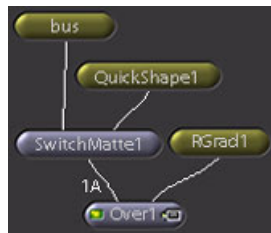
- Pull a key in a different software package and read the images into Shake. Copy the key into the alpha channel of the foreground image (with *Layer-Copy* or *Layer-SwitchMatte*). Apply an *MMult*, and then composite.
- Draw a mask with an *Image-RotoShape* node.
- Paint a mask with an *Image-QuickPaint* node.

**Note:** The *QuickShape* node is used here for illustration purposes only. *QuickPaint* is an older node that has been replaced by the *RotoShape* node.

- All of the above, combining masks with *Layer-Add*, *Layer-Max*, *Layer-Inside*, or *Layer-Outside*.

In the following example, the mask is drawn using the *QuickShape* node and copied in as the alpha channel for the bus with the *SwitchMatte* node. Because no color corrections have been made, the *preMultiply* toggle is used in *SwitchMatte* to premultiply the foreground.

**Tree With SwitchMatte**



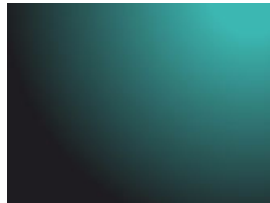
**Bus (no alpha)**



**QuickShape**



**Background**



**Premultiplied Result of SwitchMatte**

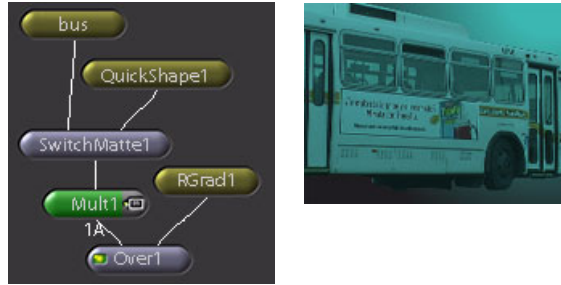


**Over Result**



The bus is color corrected in the next example, so *preMultiply* is disabled in the *SwitchMatte* node, and enabled in the *Over* node.

**Note:** You can also insert a Color-*MMult* node between *Mult1* and *Over2*.



In the following example, *MDiv* and *MMult* are added because a 3D-rendered element is color corrected. Again, you can alternatively omit the *MMult*, and enable preMultiply in the *Over* node.

#### Color Correcting



#### Result



For an example of color correcting premultiplied elements, see “Lesson Three: Depth Compositing” in the *Shake 3 Tutorials*.

#### Viewing an Alpha Channel

To view the alpha channel of an image, position the cursor in the Viewer (or the Flipbook image) and press **A**. To return to RGB, press **C**.

You can also toggle the View Channel button at the bottom of the Viewer from Color View  to Alpha View .

Shake also has a set of mathematical and Boolean layering operators. The *IAdd*, *IDiv*, *IMult*, *ISub*, and *ISubA* nodes add, divide, multiply, or subtract two images together. The second subtracting node, *ISubA*, returns the absolute value of the image. If you place a dark grey image in the first input (value .2, .2, .2), and then a white image (1, 1, 1), it returns a light grey image (.2 - 1 = -.8, take the absolute value of, returning .8, .8, .8). This is a quick way to compare two images.

The more flexible tool for generating difference mattes is the *Layer-Common* node, used to isolate common or different elements between two images. The other mathematical operators are *Min* and *Max*, which return the lower or higher values of two images, respectively. The Boolean operators *Inside*, *Outside*, and *Xor* are also useful for masking images:

- *Inside* places the foreground image only inside of the background alpha.
- *Outside* places the foreground outside of the background alpha.
- *Xor* only reveals areas without a common alpha area.

Shake contains several effect operators:

- *ZCompose* for Z-based compositing.
- *Screen*, to composite light elements and preserve highlights.
- *Atop*, to place an element only on the foreground. *Atop* is used, for example, to put a smoke element on a CG character that matches smoke in the background plate.

You can easily composite elements of any resolution. To set the output resolution of a composite that contains images of multiple resolutions, go to the *Layer* parameters and use *clipMode* to toggle between foreground or background (as the output resolution). This applies to all layering commands. An element composited over a differently-sized background is one way to set your output resolution. For more information on setting resolution, see Chapter 3, “Basics of Building a Script,” on page 69.

As outlined in the “About Channels” section below, you can easily combine 1-, 2-, 3-, 4-, or 5-channel images. For example, you can combine a luminance image with an alpha channel (2 channels) over a 5-channel image with an *Over* command.

For more information on compositing math and the individual layer functions, see Chapter 5, “Compositing: The Layer Nodes,” on page 153.

## About Channels

Shake supports and tracks different numbers of channels in an image in your composition, giving you channel independence as well as bit-depth and resolution independence.

For information on displaying channels in the Viewer, see “Viewers” on page 21.

Code	Description
BW (Black and White)	1-channel grayscale image.
A (alpha)	1-channel matte image.
BWA	2-channel grayscale image with matte channel.
RGB	3-channel color image.
RGBA	4-channel color image with matte channel.

An additional Z channel can be added to any of the above.

Therefore, the maximum amount of channels in an image is 5: RGBAZ. Unfortunately, the Z channel does not show up in the Viewer unless you use the Z ViewerScript, but it notifies you that there is a Z channel available on the top of the Viewer.

In Shake, you can combine different channel images. For example, you can composite a 2-channel image over a 4-channel image. Shake is also optimized to work on a per-channel basis—a 1-channel image usually calculates about 3 times faster than a 3-channel image. For this reason, if you read in masks from a different package, feel free to make them 1-channel images to save on disk space and processing time. If you apply an operation that changes channel information, Shake automatically updates that information. For example, if you place a Color–*Monochrome* or a Filter–*Emboss* node on an RGB image, that image becomes a BW image at that point, making following nodes faster. If you then composite the image over an RGB image or change its color (for example, *Mult* with values of 1.01, 1, 1), it becomes an RGB image again.


In certain situations, this behavior may seem a bit non-intuitive. A 3-channel image composited with a Layer–*Inside* node to a matte image still results in a 3-channel image—no matte channel is added to the result. This eliminates the need to add an alpha channel to the 3-channel image just to combine it. If, however, you want to add or remove channels at some point, you can use Layer–*Copy*, Layer–*SwitchMatte*, Color–*Set*, or Color–*Reorder*.

## Viewing the Number of Image Channels

To determine the number of channels in an image, do one of the following:

- Load the image in a Viewer, and read the Viewer title bar display.
- Position the cursor over the node in the Node View and look at the help field in the bottom-right of the interface.
- On the command line, type:

```
shake my_image -info
```

- To view the Z channel, use the Viewer's Z ViewerScript .

For information on displaying channels in the Viewer, see “Viewers” on page 21.

## Changing the Number of Image Channels

As stated above, certain operations automatically add or remove channels. For example, *Emboss* and *Monochrome* change an RGB to a BW, and a non-uniform *Add* changes a BW to an RGB. You can also explicitly change the number of channels in an image with specific functions.

The following functions also potentially modify image channels:

Function	Changes	Operation
<i>Add</i>	RGBAZ	A value raised above 0 creates that channel.
<i>Brightness</i>	RGB	A brightness value set to 0 removes the RGB channels.
<i>Copy</i>	Adds R, G, B, A, or Z	Copies a channel from the second input to the example. If you copy Z, then the second image must have the Z channel.
<i>Emboss</i>	Turns an RGB image to BW	This, of course, radically alters your image.
<i>Monochrome</i>	Turns an RGB image to a BW	Uses a luminance balance, but you can adjust this to push specific channels.
<i>Mult</i>	R, G, B, A, Z	Setting the RGBA or Z to 0 removes those channels.
<i>Reorder</i>	Adds or removes R, G, B, A, or Z	By using <i>n</i> or 0, you remove that channel: <i>rgbn</i> or <i>rgb0</i> removes the alpha channel. <i>rgb0a</i> adds the luminance into the Z channel, thereby creating a Z channel. <i>rrra</i> creates a 2-channel image, assuming a is not black. <i>000a</i> or <i>nmna</i> turns the image into a 1-channel alpha image.
<i>Set</i>	Adds or removes R, G, B, A, or Z	A value set to 0 removes that channel. A channel set to something other than 0 adds that channel.
<i>SwitchMatte</i>	Adds A	Copies in any channel from the second input to be used as the new alpha channel for the first input.

Many operations allow you to select what channel is used as the modifying channel. For example, *SwitchMatte*, *KeyMix*, *IBlur*, etc., all give you the option to select the R,G,B, or A channel as your control or alpha channel. This often removes the need to swap your channels before you do many operations. Two exceptions to this are *Layer-Inside* and *Layer-Outside*, which always depend on the second image's alpha channel.

To convert a BW (or BWA) image into an RGB (or RGBA) image without changing its values, specify the following in the *FileOut* in the interface. This is not necessary in your node tree as Shake figures it out. For batch operations, this can be somewhat annoying, so *ForceRGB* is included for the command line:

```
shake myBlackAndWhiteImage.iff -forcergb -fo myRGBImage
```

For information on channel/compositing functions, see Chapter 5, “Compositing: The Layer Nodes,” on page 153.

## About the Node View and the Tool Tabs

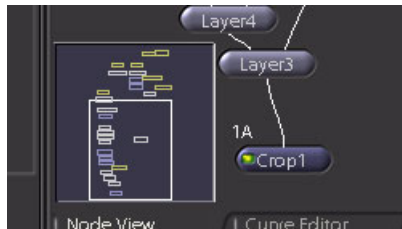
Every effect in Shake is a distinct node that can be inserted into a network of linked functions, or a node tree. Each node has its own specific function and specific parameters. To create a node, select the node in the Tool Tabs. The selected nodes appear in the Node View, which can be used to select, view, and organize your composite.

In both the Tool Tabs and the Node View, the right-click menu is available.

As with other windows, press the middle-mouse button or press **Opt-click** / **Alt-click** to pan. In the Node View, you can also press **Ctrl+Opt-click** / **Ctrl+Alt-click** to zoom in and out.

## Overview

The Overview in the Node View is a helpful navigation tool, especially for complex node trees. To enable the Overview, press **O** in the Node View. Click and drag in the Overview to pan the node tree. Press **O** again to remove the Overview.



## Creating Nodes

There are five ways to add nodes to a tree:

- Select the node you want in the Tools menu at the top of the screen.
- Click the node in the Tool Tabs in the lower-left window of the Shake interface.
- Right-click a Tool Tab and select a node from the pop-up menu.
- Right-click in the Node View and select Nodes > (the node group) > (node).  
For example, to add an *Atop* node, select Nodes > Layer > *Atop*.
- Copy a previously existing node and paste the node into the Node View.

To display a pop-up menu of the functions, right-click the tab name. The modifier keys (see below) work as well. Additionally, if you lower the Tool Tabs so only the tabs are visible, you can also access the pop-up menus with the left-mouse button (a cool trick).



To add multiple nodes within a tab, right-click the tab, and then right-click to select the nodes. To close the menu, left-click in the menu.

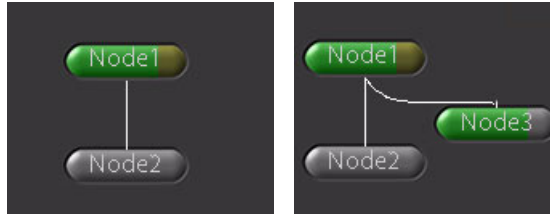
## Inserting, Replacing, and Deleting Nodes

There are several ways to insert nodes into a tree:

- To insert a node between two nodes, pick the parent node and click the new node from the function tabs.



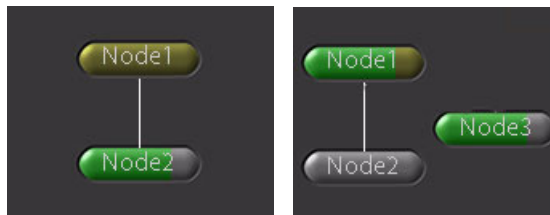
- To create a new branch, select the parent node, and **Shift**-click the new node.



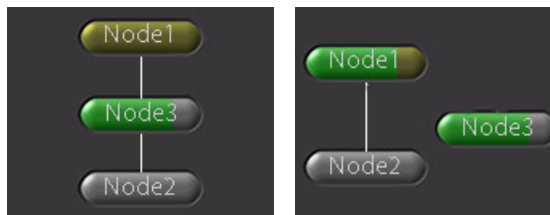
- To replace a node, select the node to replace, and **Ctrl**-click the new node.



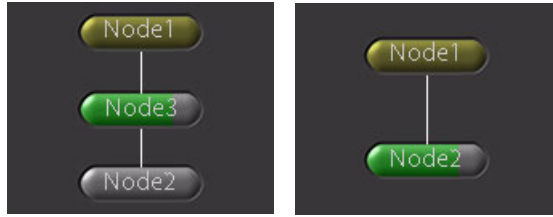
- To create a floating node, **Ctrl+Shift**-click the node you want:



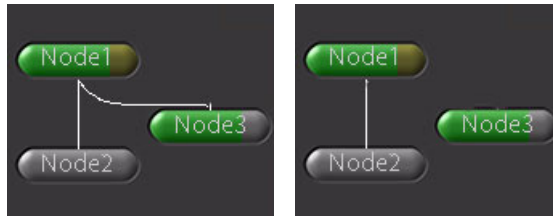
- To extract a node from a tree, select the node and press **E** (for Extract).



- To delete a node, select the node and press **Del** (Macintosh) / **Backspace** (Linux/IRIX).



- To disconnect a noodle, **Ctrl**-click the noodle, or put the cursor over the noodle (it turns red) and press **Del** / **Backspace**.

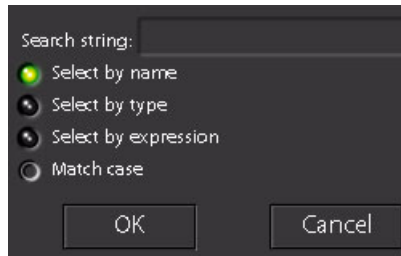


### Selecting and Deselecting Nodes

There are several ways to select and deselect nodes:

- **Shift**-drag nodes in the Node View to select the nodes.
- **Ctrl**-drag nodes in the Node View to deselect the nodes.
- Click in an empty area of the background to deselect all nodes.
- Press **Command+A** / **Ctrl+A** to select all nodes.
- To select all nodes attached to a node, select one node and press **Shift+A**.
- To select a node by name, press **Command+F** / **Ctrl+F** in the Node View.

The Select Nodes by Name window appears. Nodes are selected in real time as you type. There are several options to select nodes:



Function	Notes
<i>Select by name</i>	Enter the search string, and matching nodes are immediately activated. For example, if you enter just <i>f</i> , <i>FileIn1</i> and <i>Fade</i> are selected. If you enter <i>fi</i> , just the <i>FileIn1</i> is selected.
<i>Select by type</i>	Selects by node type. For example, enter <i>Move</i> , and all <i>Move2D</i> and <i>Move3D</i> nodes are selected.
<i>Select by expression</i>	Allows you to enter an expression. For example, you want to find all nodes with an angle parameter greater than 180: <i>angle &gt; 180</i> .
<i>Match case</i>	Sets case sensitivity.

## Copying and Pasting Nodes

To copy and paste a node or group of nodes:

- In the Node View, select the node(s).
- Press **Command+C** / **Ctrl+C**, and then **Command+V** / **Ctrl+V**.

**Note:** You can also right-click a selected node(s) and select Edit > Copy, and then select Edit > Paste.

## Moving Nodes

To move a node, select the node and drag it in the Node View.

## Connecting Nodes

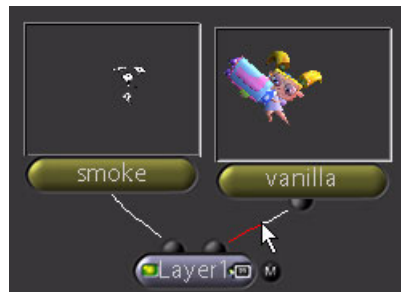
To connect nodes, drag the output of one node to the input of another node. If a different node is already connected, it is replaced by the new node connection.

You can also drag an input of a node to the output of a different node, for example, from one of the *Layer1* node inputs to the output of the *vanilla* node.



You can drag as many connections from the output or to the output of a node as you want. Each node input can have only one noodle connection.

- To delete a connection, put the cursor over the noodle so that it turns red, and press **Del** / **Backspace**.



- To switch inputs, drag the bottom half of a noodle (it turns red) to the other input. The inputs are switched.



## Loading a Node Into a Viewer

Click the left side of a node to load that node into the current Viewer. Double-click the left side of a node to load that node's parameters into the Parameter View.

When a node is displayed, a small button on the left side is highlighted. Additionally, a number and a letter appear with it to indicate the Viewer and compare buffer that the image occupies.



## Loading Node Parameters

Click the icon on the right side of the node to load its parameters into the Parameters1 tab. In the following image, the parameters for *vanilla* are loaded.

To load the parameters into the Parameter2 tab, **Shift**-click the right side of the node.



## Ignoring Nodes

To turn off the processing of a node, select the node(s) and press **I**. This puts a red diagonal line over the node indicating it is ignored. To activate a node again, press **I**.

**Note:** You can also load a node in the Parameters, and enable the ignoreNode button in the Parameters tab.

For scripting purposes, this places the *Ignore* function after an ignored node, and is not visible in the Node View.



## Renaming Nodes

Click the icon on the right side of the node to load its parameters into the Parameters1 tab. In the parameters, enter the new name in the nodeName field.

**Note:** When naming nodes, avoid spaces or funny characters (', .!, etc.). Also, to avoid confusion, do not name a node another function name, such as *Fade*.

*FileIn* and *FileOut* nodes are automatically renamed based on their associated file.

## Organizing Nodes, Groups, and Clusters

To assist in the organization and navigation of your more complex node trees, Shake has several features to clean up nodes in the Node View.

### Grid Snap

To activate a grid in the Node View, do one of the following:

- In the Node View, right-click and select Snap to Grid.
- In the guiSettings subtree of the Globals tab, enable the gridEnabled parameter.

When the Node View grid is enabled, nodes that are moved are automatically aligned.

- To temporarily activate the grid snap when Snap to Grid is disabled, press **Shift** as you drag a node.
- To temporarily deactivate the grid snap when Snap to Grid is enabled, press **Shift** as you drag a node.

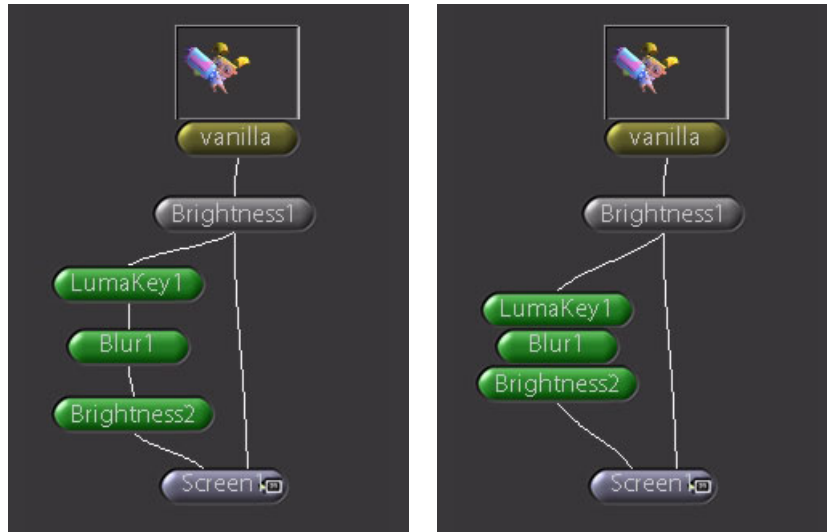
The grid spacing and visibility parameters are under the guiSettings subtree (gridEnabled) in the Globals tab.

## Layout Controls

To clean up nodes, select the nodes and press **L**. The nodes are automatically laid out in a somewhat organized manner. Use Layout with caution, however, as complicated trees with multiple cross-branching may have odd results. (Entire college theses have been written on this subject, so be generous.)

When nodes are created or organized with the Layout function, they are spaced according to the layoutTightness parameter in the guiSettings subtree of the Globals tab. Newly created nodes are spaced this distance from their parents, so a smaller number creates tighter trees.

To tightly align nodes vertically, select the nodes and press **Shift+L**.



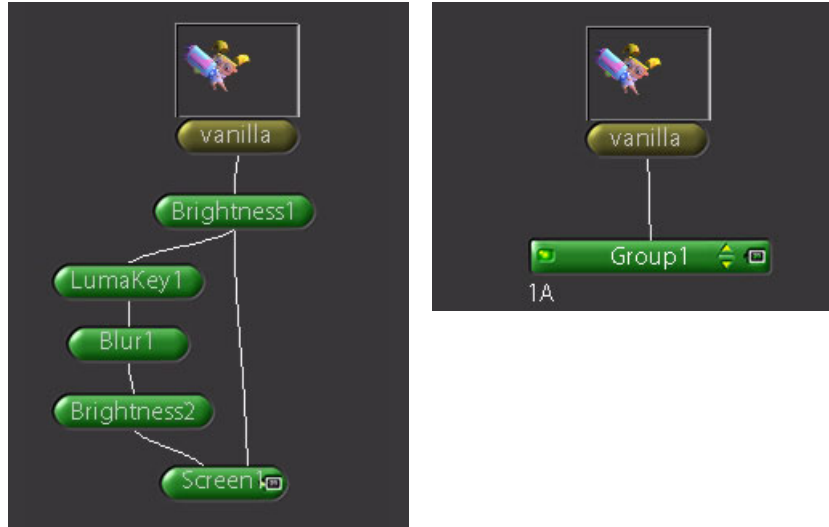
To align the nodes vertically (on the same X axis), press **X**. To align the nodes horizontally (on the same Y axis), press **Y**.

## Groups

You can also collapse a selection of nodes into a small group. The noodles (connections) feeding in and out of the group are preserved, and you can also view and edit the group's contents.

To create a group, select the nodes and press **G**.

**Note:** You can also right-click in the Node View and select Groups > Group Selected Nodes.



To ungroup the grouped nodes, press **Ctrl+G**.

To expand the node group into a cluster, click the Expand Group button  on the group node (see “Clusters” on page 142).

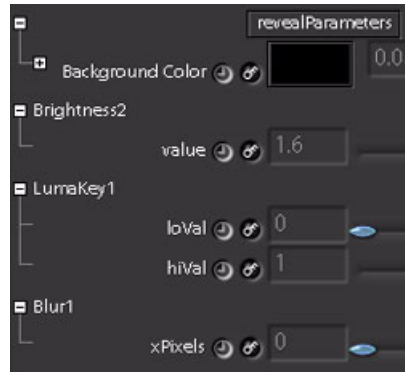
Loading the parameters of the group allows you to change the color of the cluster background (see below), add notes, and expose selected specific parameters. This allows you to isolate important sliders from multiple nodes and load them into a single parameter tab, saving you the trouble of jumping between multiple nodes.

#### **To expose parameters in a Group:**

- 1** Load the parameters of the Group (click the right side of the Group node).
- 2** In the Group parameters, click Reveal Parameters.  
The Expose Group Parameters window appears.
- 3** Select the nodes:
  - To expose all parameters of a node, click Select All.
  - To expose only certain parameters of a node, expand the node’s Select All subtree, and enable the desired parameters.
- 4** Click OK.

The selected nodes and parameters appear in the Group parameters.

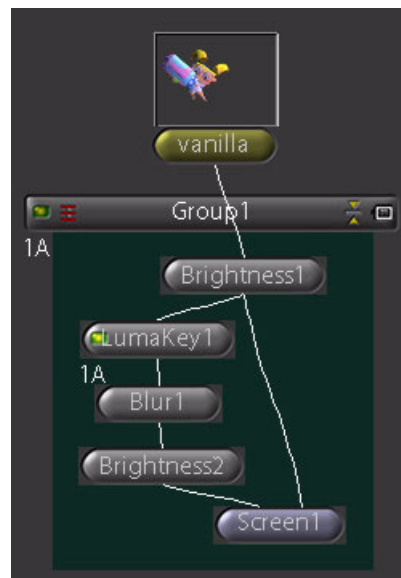
### Exposed Parameters From a Group





### Clusters

Clusters are nodes that have been grouped and then expanded. Clusters allow you to color-code a large area of nodes for visual navigation and provide a handle for node organization. The advantage over a group is that you have access to each node internally for rewiring or for tuning that node. If you ungroup a node, you lose all group information. By using the cluster, you keep both the group information as well as the flexibility of free nodes.

To expand a group into a cluster, click the Expand button  on the right side of the Group node.



Once a Group is expanded into a cluster, the buttons on the cluster bar include the standard View and Load Parameters buttons, and two additional buttons:

- The Ungroup button (on the left side of the cluster)  removes all grouping/cluster information. A Warning window appears that states, “You are about to ungroup the selected group. Continue?” Click Yes to ungroup the selected group.
- The Collapse button (on the right side of the cluster)  closes the cluster back into a normal group.

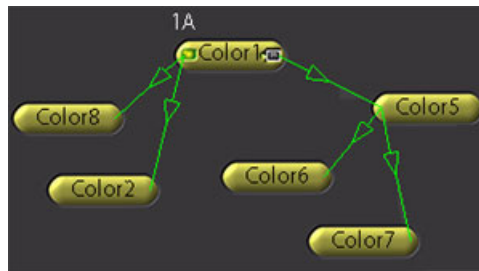
## Opening Macros

To examine the contents of a macro, press **B**. The macro opens in the same way as if you were to look at a group of nodes. To close the macro, click the macro, position the cursor on an empty area, and press **Opt+B** / **Alt+B**. When a macro is open, you can view any parameter or stage of the macro, but you cannot edit parameters or rewired nodes.

## Linking Nodes

To link one node to another, copy and paste the node using **Command+Shift+C** and **Command+Shift+V** / **Ctrl+Shift+C** and **Ctrl+Shift+V**. This links the pasted node to the original node. If you modify the pasted node, the link disappears for that particular parameter. Otherwise, you typically tune the master node.

To view the links, press **Ctrl+E** in the Node View. Lines with directional arrows display the connections between nodes.



## Thumbnails

Thumbnails are automatically generated in the Node View for all image nodes. The thumbnails are meant to be visual roadmarks to help navigate the Node View. They do not constantly update, and so therefore do not slow down your frame-to-frame processing.



The thumbnails are stored in the cache and, since they are rarely modified, are relatively painless to retrieve at later script loadings.

Thumbnail Keyboard Shortcuts/Commands	Function
<b>R</b> (on active nodes)	Refresh/Create Thumbnail.
<b>T</b> (on active nodes)	Turns thumbnail display on or off.
<b>A/C</b> (over thumbnail)	Displays alpha/color channels.
Globals—displayThumbnails	Turns all thumbnails on and off.
Globals—thumbSizeRelative	Makes thumbnails similar size or relative to their actual sizes.
Globals—thumbAlphaBlend	Turns thumbnail transparency on and off.
Globals—thumbSize	Controls the display size of the thumbnails.

If you have made changes to thumbnail settings, you can save your changes.

**To save the new settings:**

- Choose File > Save Interface Settings.

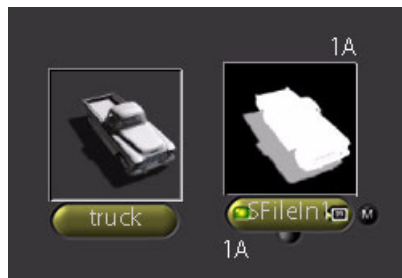
See the section, “Default Values for Global Parameters” on page 96.

To generate a thumbnail of a non-image node, select the node and press **R**. This is also used to refresh a thumbnail to the current frame.



**Note:** Before you whip out an email to ask why thumbnails do not always update, here is a preemptive strike: Dynamic thumbnails would be inefficient or inaccurate. If you worked on high-resolution plates, you would constantly be shrinking down 2K plates for the purposes of a thumbnail. The alternative is to use tiny images, but as soon as you put a *DilateErode* of 5 on it, you have no idea if what you see is accurate. When working with a sequence, your problems just compound, especially if you have 500 or more nodes.

When the cursor is placed over a thumbnail, it indicates what frame is loaded as a thumbnail and if you are looking at Color View (C) or the Alpha View (A). Press **A** or **C** over the thumbnail to display the Alpha or Color channels Views.

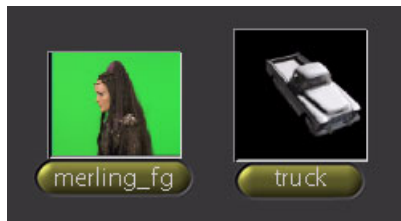


When a thumbnail is moved over another thumbnail, you can get a glimpse of how the nodes might appear when composited. (Ain't it swell? Probably useless, but it plays well in Peoria.) To disable the transparency, toggle `thumbAlphaBlend` in the `displayThumbnails` subtree of the Globals.

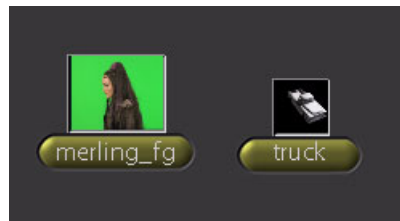


To modify thumbnail size, adjust the `thumbSize` slider in the Globals. Thumbnails are displayed at the same width. To have the thumbnails display their true relative sizes, enable `thumbSizeRelative` in the Globals. In the following images, the green screen image is PAL and the truck image is 410 x 410.

**thumbSizeRelative Deactivated**



**thumbSizeRelative Activated**



You can declare any specific type of node to always contain thumbnails. For example, to add *FileOuts* into the list of nodes receiving a thumbnail, set the following *ui.b* code:

```
nuiNodeViewEnableThumbnail("FileOut");
```

You can also disable an enabled node with:

```
nuiNodeViewDisableThumbnail("FileOut");
```

For all nodes, use *NRiFx* as your class. Note, however, that specifying downstream nodes (especially *FileOut* nodes) can cause pauses at script load time, as the entire tree must be calculated to derive the proper thumbnail. Use with caution.

To have thumbnails always off, disable the thumbnails in the `guiSettings` of the Globals tab, and choose File > Save Interface Settings, or add the following *ui.b* code:

```
script.displayThumbnails = 0;
```

## The Node View Right-Click Menu

Submenu	Function	Keyboard	Notes
Nodes			Create nodes directly in the Node View from the node list.
Edit	Cut	<b>Command+X / Ctrl+X</b>	Removes selected nodes and places them into the paste buffer.
	Copy	<b>Command+C / Ctrl+C</b>	Copies the selected nodes into the paste buffer.
	Paste	<b>Command+V / Ctrl+V</b>	Pastes the buffer into the Node View. You can also copy nodes from the Node View and paste them into a text document. This is really swank, by the way. It won't impress your mother, but it might impress her if she is a programmer.
	Delete	<b>Del / Backspace</b>	Deletes the selected nodes. If the branching is not complicated, the noodles between the parent(s) and children are automatically reattached to each other.
	Undo	<b>Command+Z / Ctrl+Z</b>	Undo up to 100 steps. Rearranging nodes counts as a step.
	Redo	<b>Command+Y / Ctrl+Y</b>	Redo your steps unless you have changed values after several undos.
View	Zoom In	<b>+</b>	Zooms into the Node View (also use <b>Ctrl+Opt-click / Ctrl+Alt-click</b> ).
	Zoom Out	<b>-</b>	Zooms out of the Node View (also use <b>Ctrl+Opt-click / Ctrl+Alt-click</b> ).
	Reset View	<b>Home</b>	Centers all nodes.
	Frame Selection	<b>F</b>	Frames all selected nodes into the Node View.
Render	Render Flipbook		Renders a flipbook of the node visualized in the active Viewer.

Submenu	Function	Keyboard	Notes
	Render Disk Flipbook		Macintosh OS X only. This option launches a disk-based flipbook into QuickTime. This has several advantages over normal flipbooks. It allows for extremely long clips and allows you to attach audio (loaded in with the audio tab in the main interface). You can also choose to write out the sequence as a QuickTime file after viewing, bypassing the need to rerender the sequence.
	Render Selected FileOuts		Renders all selected <i>FileOut</i> nodes, and writes them to disk.
	Render All FileOuts		Renders all <i>FileOut</i> nodes, and writes them to disk.
Overview On/Off		<b>O</b>	Turns on the Overview window to help navigate in the Node View.
Expr Links On/Off		<b>Ctrl+E</b>	Displays links between nodes.
Select	Find Nodes	<b>Command+F / Ctrl + F</b>	<p>Activates nodes according to what you enter in the Search string field in the Select Nodes by Name window.</p> <p><i>Select by name.</i> Enter the search string, and matching nodes are immediately activated. For example, if you enter just <i>f</i>, <i>FileIn1</i> and <i>Fade</i> are selected. If you enter <i>fi</i>, just the <i>FileIn1</i> is selected.</p> <p><i>Select by type.</i> Selects by node type. For example, enter <i>Transform</i>, and all <i>Move2D</i> and <i>Move3D</i> nodes are selected.</p> <p><i>Select by expression.</i> Allows you to enter an expression. For example, to find all nodes with an angle parameter greater than 180: angle&gt;180</p> <p><i>Match case.</i> Sets case sensitivity.</p>
	All	<b>Command+A / Ctrl+A</b>	Selects all nodes.
	Associated Nodes	<b>Shift+A</b>	Selects all nodes attached to the current group.

Submenu	Function	Keyboard	Notes
	Invert Selection	<b>!</b>	All selected nodes are deactivated; all deactivated nodes are activated.
	Select Upstream	<b>Shift+U</b>	Adds all nodes upstream from the currently active nodes to the active group.
	Select Downstream	<b>Shift+D</b>	Adds all nodes downstream from the currently active nodes to the active group.
	Select Upstream 1 Level	<b>Shift+Up Arrow</b>	Adds one upstream node to the current selection.
	Add Downstream 1 Level	<b>Shift+Down Arrow</b>	Adds one downstream node to the current selection.
Node Layout	Layout Selected	<b>L</b>	Automated layout on the selected nodes.
	Align Selected Vertically	<b>X</b>	Snaps all selected nodes into the same column.
	Align Selected Horizontally	<b>Y</b>	Snaps all selected nodes into the same row.
Thumbnails	Refresh Selected Thumbnails	<b>R</b>	Activates/refreshes the thumbnails for selected nodes.
	Show/Hide Selected Thumbnails	<b>T</b>	Turns on/off selected nodes. If you haven't yet created a thumbnail (R), this does nothing.
	View RGB Channels	<b>C</b>	Displays the RGB channels.
	View Alpha Channels	<b>A</b>	Displays the alpha channel.
Group/ Ungroup Selected Nodes		<b>G</b>	Visually collapses selected nodes into one node. When saved out again, they are remembered as several nodes. To ungroup, press <b>G</b> again.

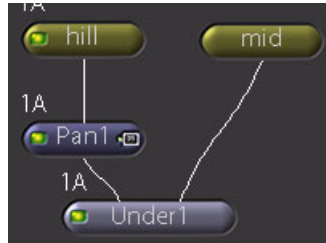
Submenu	Function	Keyboard	Notes
Maximize/ Minimize Selected Groups		<b>M</b>	Opens a group into a subwindow.
Ignore/ UnIgnore Selected Nodes		<b>I</b>	Turns off selected nodes when activated. Select them again and press <b>I</b> to reactivate the nodes. You can also load the parameters into the Parameter View and enable ignoreNode.
Extract Selected Nodes		<b>E</b>	Pulls the active nodes from the tree, and reconnects the remaining nodes to each other.
Save Selection as Script		<b>S</b>	Saves the selected nodes as a script.
Force Selected FileIn Reload  Force All FileIn Reload			When an image is changed on disk and you have already looked at that image in the interface, Shake does not recognize that the image has changed. Selecting these two functions forces the checking of the date stamp for the images on disk.
Macro	Make Macro	<b>Shift+M</b>	Launches the MacroMaker with the selected nodes as the macro body.
	Show Macro Internals	<b>B</b>	Opens a macro into a subwindow so you can review wiring and parameters. You cannot change the nodes inside the subwindow.
	Hide Macro Internals	<b>Opt+B</b> / <b>Alt+B</b>	Closes up the macro subwindow when the cursor is placed outside of the open macro.

## About the Infinite Workspace

Another powerful feature is the Infinite Workspace. Shake's rendering is similar to a camera; whatever is exposed in the frame is rendered (and only that), no matter what its original resolution. This means that if you have a very small element such as 100 x 100 pixels, and you pan it 50 pixels in X and 50 pixels in Y, three-fourths of the image is outside of the 100 x 100 pixel frame. However, if you then place that over a 400 x 400 pixel frame, everything previously outside of the frame is “rediscovered”—you never lose data due to transformations or resolution changes.

In the following example, the *hill* and *mid* images are used from the *pix/vp/bg* directory. The *hill* image is panned and then composited with the larger *mid* image. However, the *hill* image is restored in the composite.

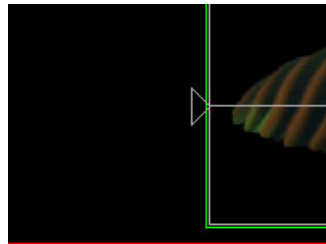
**Tree**



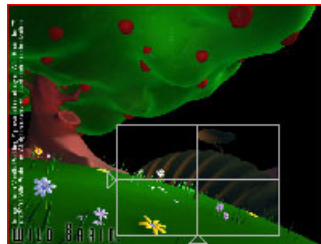
**hill image**



**Pan1**



**Under1**



This powerful feature has almost no memory or calculation cost, and significantly eases the handling of typical resolution difficulties.

Another optimizing aspect of this camera analogy is that Shake only renders what is in the current frame. This means if you have a 20,000 x 20,000 pixel image but have a *Crop* window of 100 x 100 pixels, only those 100 x 100 pixels are considered, even if you have other nodes before the *Crop*. Again, only what is inside of the *Crop* window is calculated. This makes Shake ideally suited for higher-resolution functions such as scrolling a large background plate under lower-resolution foreground elements.

The Infinite Workspace has several important workflow advantages:

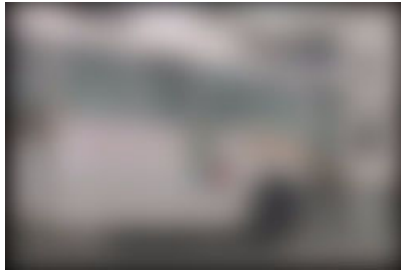
- You do not need to crop a small image before it is composited with a larger image when you are panning the image. Simply read in your image, apply the pan, and composite it over/under the larger image.

- Shake's *Blur* node gives you the option to blur pixels outside of the frame with the *spread* parameter. When set to 0, only pixels inside the frame are considered. When set to 1, outside pixels are calculated into the blur as well. When you read in an image and then blur the image, set the *spread* to 0. Otherwise black ringing occurs around the edge. If you read the image, then scale it up, and then blur, you want to set *spread* to 1, since there are now non-black pixels outside of the frame.

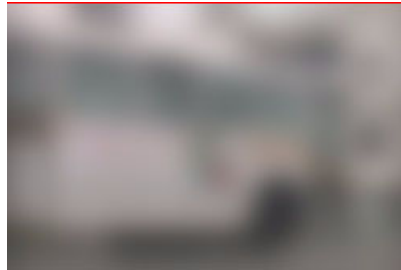
### Clipped Images

If an image is clipped, it is usually because a *Crop* node has been applied, or because you have applied a *Blur* with a *spread* set to 0. Set *spread* to 1 in the *Blur* node's parameters.

**spread = 1**



**spread = 0**



- If you transform an object, apply a color correction, and then transform the object back, the entire image has the result of the color correction, not just what was in the frame at the time you applied the correction. You must also be careful when pulling a blue screen matte with *ChromaKey*. The outside black pixels are counted as invisible because it is keying off of a non-black color.

To disable the effect of the Infinite Workspace, insert a *Crop* node and leave it at the default values (which does not change the resolution). This cuts off everything outside of the frame, and replaces it with black pixels. The *Viewport* command is similar to the *Crop*, except it does not disable the Infinite Workspace.

Be very careful about scaling elements up, applying an operation, and then zooming back down. When you apply an operation on the scaled element, even though your frame is small, it still calculates everything outside of the frame *if* you scale it back down to fit in the frame. For more information on the Infinite Workspace, see “Color Correction With the Infinite Workspace” on page 340. You can also see “The Domain of Definition (DOD)” on page 50.

# Compositing: The Layer Nodes

## Chapter Summary

- About Compositing
- Compositing Math
- Photoshop Layering Modes
- The Layer Functions

## About Compositing

This chapter discusses each of the Layer nodes and demonstrates the compositing math for you geeks in the audience.

There are three types of layering nodes:

- Atomic Nodes

The atomic nodes (*Over*, *IAdd*, *Atop*, etc.) do one thing—combine two images according to a fixed mathematical algorithm. They are useful for command-line compositing, scripting, and are also nice for the Node View in that the user can quickly see which type of operation is occurring.
- More Flexible Nodes

The second node type is the more flexible *Layer*, *MultiLayer*, and *Select* nodes. *Layer* and *MultiLayer* allow you to duplicate most of the atomic nodes (with the exception of the *AddMix*, *AddText*, *Interlace*, and *KeyMix* nodes). For a description of each node, see the *Layer* or *MultiLayer* sections, below. *MultiLayer* and *Select* have the additional benefit of adding unlimited inputs to the node.
- LayerX Node

The third layering node group is the unique node *LayerX*, which allows you to enter your own compositing math.

## Masking Layer Nodes

Basically, don't. The side-input masks for layer nodes behave counter-intuitively and inaccurately from what you want. If you want to mask a layering node, mask the input nodes or use the *KeyMix* node.

## Compositing Math

If you were stuck on a desert island and had only one *Layer* node, it would have to be *LayerX*, since it can be used to mimic all of the other functions. The math for most of the operators is included in the function, both in general notation and in *LayerX* syntax. The *LayerX* syntax only has the expression for the red channel. You can easily extrapolate the additional equations for the other channels (for example, change R to G for the green channel).

A = foreground RGBA

B = background RGBA

Aa = foreground Alpha

Ba = background Alpha

The following table provides a quick reference to the Shake layer functions and their common uses, math, and *LayerX* syntax. For specific function descriptions, see “The Layer Functions” on page 161.

Shake Layer Function	Common Uses	Math	LayerX Syntax
<i>Atop</i>	Add effects to foreground elements, like smoke over a CG character to match the background.	$A * B_a + (B * (1 - A_a))$	$r2 + (r * a * a2)$ or $(a2 == 0 \mid \mid (r2 == 0 \ \&\& \ g2 == 0 \ \&\& \ b2 == 0 \ \&\& \ a2 == 0) ? r2 : (a2 * r + (1 - a2 * a) * r2))$
<i>Common</i>	Difference masks.		
<i>IAdd</i>	Fire effects, adding mattes together.	$A + B$	$r + r2$
<i>IDiv</i>		$A / B$	$r2 == 0 ? 1 : r / r2$
<i>IMult</i>	Mask elements.	$A * B$	$r * r2$
<i>Inside</i>	Mask elements.	$A * B_a$	$r * a2$

Shake Layer Function	Common Uses	Math	LayerX Syntax
<i>Interlace</i>	Interlaces two images, pulling one field from one image, and the second field from the other image.		
<i>ISub</i>		$A - B$	$r - r2$
<i>ISubA</i>	Find the difference between elements.	$\text{absolute}(A - B)$	$\text{fabs}(r - r2)$
<i>KeyMix</i>	Mix two images through a third mask.	$A * (1 - M * C) + (B * M * C)$	M represents the percentage mix.
<i>Layer</i>	Duplicate most other <i>Layer</i> operations.		
<i>Max</i>	Combine masks.	If $(A > B)$ then A, else B	$r > r2 ? r : r2$ or $\text{max}(r, r2)$
<i>Min</i>		If $(A < B)$ then A, else B	$r < r2 ? r : r2$ or $\text{min}(r, r2)$
<i>Mix</i>	Cross-fade.	$A * (1 - \text{Mix}) + (B * \text{Mix})$	$r * (1 - \text{mix}) + r2 * \text{mix}$
<i>Outside</i>	Mask elements.	$A * (1 - B_a)$	$r * (1 - a2)$
<i>Over</i>	Primary compositor.	$A + (B * (1 - A_a))$	$r + (1 - a) * r2$
<i>Screen</i>	Add reflections and light elements without losing highlight detail.	$1 - (1 - A) * (1 - B)$	$1 - (1 - r) * (1 - r2)$
<i>Under</i>	A reverse of <i>Over</i> .	$B + (A * (1 - B_a))$	$r2 + r * (1 - a2)$

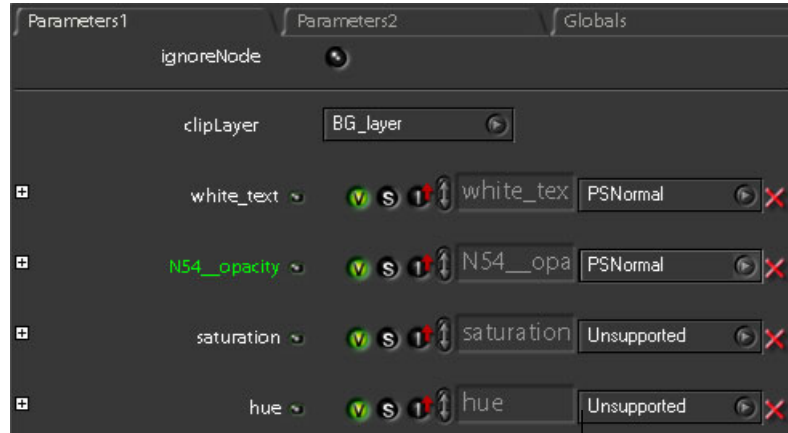
Shake Layer Function	Common Uses	Math	LayerX Syntax
<i>Xor</i>	Find areas that are mutually exclusive.	$A*(1-Ba) + B*(1-Aa)$	$r*(1-a2) + r2*(1-a)$
<i>ZCompose</i>	Z-depth composite.	<pre>f(Aa == 1) then   If (Az == Bz) then     ((A + (B*(1-Aa))) +      B + (A*(1-Ba))) / 2   else     If (Az &lt; Bz) then A     else B   else A + (B*(1-Aa))</pre> <p>If foreground alpha does not equal 1, an <i>Over</i> is performed. Otherwise, if the Z values are not equal, the lower Z is taken. If the Z values are the same, the result is a 50 percent mix of an <i>Over</i> and an <i>Under</i>.</p>	

## Photoshop Layering Modes

You can import a multilayer Photoshop file into Shake with a *FileIn* node or using File > Import Photoshop File. Shake supports most of the Photoshop transfer modes, with the exception of Color, Hue, Saturation, Dissolve, and Luminosity.

When you import a Photoshop file that contains a layer (or layers) set to an unsupported transfer mode, a grumpy “Unsupported Transfer Mode (Mode Name)” message appears in the Viewer, and the mode for that layer in the Parameters tab reads “Unsupported.”

You can set the layer to a different mode in the Parameters tab.



Click and hold to select a different compositing operation.

**Note:** Shake does not support Photoshop layer masks, or alpha channels created in the Photoshop Channels tab.

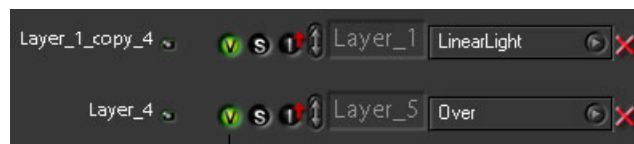
#### To import a Photoshop file using the File menu:

- Choose File > Import Photoshop File.

The file is imported as a script—each layer in the Photoshop file is imported as a *FileIn* node that is then fed into a *MultiLayer* node. The *MultiLayer* node is named *Composite* by default.

Double-click the *Composite* node to display the Photoshop image (the composite) in the Viewer and load the *MultiLayer* node parameters.

**Note:** When you import a Photoshop file that contains invisible layers (the eye icon is disabled in Photoshop), the visibility for that layer (the *FileIn* node) is disabled in Shake. To show the layer, toggle the layer's visibility button in the *MultiLayer* node parameters.



Layer Visibility button

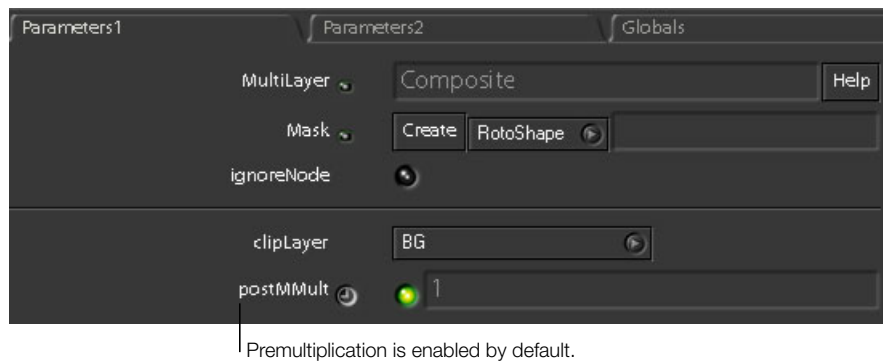
To change the opacity of a layer, expand the subtree for the layer and set the opacity parameter to a value between 0 and 1. By default, the layer opacity is set to 1. A layer that is set to a Photoshop transfer mode contains an additional opacity control—the PSOpacity parameter.



This additional opacity control is necessary because Photoshop and Shake handle transparency differently. Photoshop’s transparency setting works by varying the intensity of the alpha channel on the foreground image (prior to the blending operation). In Shake, the opacity setting on each layer of the *MultiLayer* node varies the intensity of all channels (RGBA). The results can differ between the two methods, depending on the selected blend mode and the way it uses color. The default PSOpacity setting is the opacity of the layer as set in Photoshop.

**Note:** To view the name of the Photoshop file, click the right side of a *FileIn* (Photoshop layer) node to display the *FileIn* parameters. In the Source tab, the name of the imported Photoshop file is displayed in the imageName parameter.

In the *MultiLayer* parameters, postMMult is enabled by default. When enabled, the postMMult parameter is premultiplying the *MultiLayer* node (in the same manner as a composite in the Photoshop application).



For more information on the other buttons in the *MultiLayer* parameters, see “MultiLayer” on page 186.

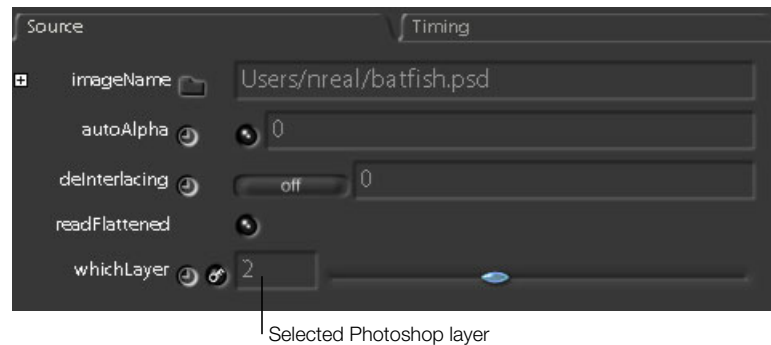
### To import a Photoshop file using a FileIn:

- In the Image tab, click *FileIn* and navigate to the Photoshop file in the “Load image or sequence window.” Select the file(s) and click OK.

By default, the image is imported as merged. To select an individual layer:

- Load the Photoshop file’s *FileIn* parameters (click the right side of the node).
- In the Source subtab, disable readMerged.
- In the whichLayer parameter, select the layer you want to display. In the following example, the third layer of the Photoshop file is selected.

**Note:** When Shake imports a multilayer Photoshop file, the first layer in the file is numbered 0 in the whichLayer parameter. In the following image, the whichLayer parameter is set to 2—the third layer of the Photoshop file (since the first layer is imported as 0, the second layer imported as 1, and so on).



The layer modes listed in the following table are accessed in the *Layer* or *MultiLayer* nodes.

**Note:** The true math of the Photoshop transfer modes is the proprietary information of Adobe Systems Incorporated. Descriptions listed are therefore not necessarily technically accurate.

Shake Layer Function	Explanation
<i>ColorBurn</i>	Takes the color of the second image and darkens the first image by increasing contrast. White in the second image has no effect on the first image.
<i>ColorDodge</i>	The opposite of ColorBurn, this lightens the image. Black in the second image does not affect the first image.
<i>Darken</i>	Identical to Min, taking the lower pixel value.
<i>Exclusion</i>	The second image acts as a mask for inverting the first image. White areas of the second image completely invert the first image, black areas of the second image leave the first image unmodified.
<i>HardLight</i>	Screens or multiplies the first image, depending on the strength of the second image. Values below 50 percent in the second image multiply the first image, making it darker. Values above 50 percent in the second image act as a Screen effect. Values of pure black or white in image 2 replace image 1.
<i>Lighten</i>	Identical to Max. Takes the maximum value when comparing two pixels. Good for adding mattes together.
<i>LinearBurn</i>	Similar to ColorBurn, except it decreases brightness, not contrast, to darken the first image. Whites in the second image do not modify the first image.
<i>LinearDodge</i>	The opposite of LinearBurn, brightens the first image. Black areas in image 2 leave the first image unaffected.
<i>LinearLight</i>	A combination of LinearBurn and LinearDodge. Values above 50 percent in image 2 increase brightness of the first image. Values below 50 percent darken the first image. Values of pure black or white in image 2 replace image 1.
<i>Overlay</i>	To shade an image. 50 percent in the second image keeps the first image the same. Brighter pixels brighten the first image, darker pixels darken it.
<i>PinLight</i>	Performs Min or Max, depending on the strength of the second image. If the second image is brighter than 50 percent, a Max (Lighten) is performed. If the second image is darker than 50 percent, a Min (Darken) is performed. Values of pure black or white in image 2 replace image 1.

Shake Layer Function	Explanation
<i>SoftLight</i>	Raises or lowers brightness, depending on the strength of the second image. Values above 50 percent in the second image decrease the brightness of the first image, values below 50 percent increase the brightness.
<i>VividLight</i>	Raises or lowers contrast, depending on the strength of the second image. Values above 50 percent in the second image decrease the contrast of the first image, values below 50 percent increase the contrast.

## The Layer Functions

This section provides a detailed description of each of the layer functions, including parameters, defaults, scripts, and references to related functions.

### AddMix

The *AddMix* function is similar to *Over*, except you have control over curves to help blend the edges together.

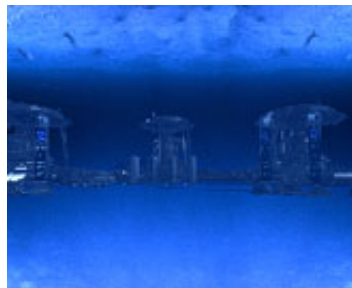
#### AddMix Example

In the following tree, a key is pulled on the foreground with a *KeyLight* node. The foreground is then attached to the background tree. The first “Closeup” illustration (on the left) shows the default result, a result identical to an *Over* node. Ringing problems occur along the hair (a bright ring), and along the shoulder (a dark ring). When the curves are adjusted, the ringing is dramatically reduced, resulting in a much cleaner composite.

**Tree**



**Over1**



**Blue Screen**



**AddMix1**



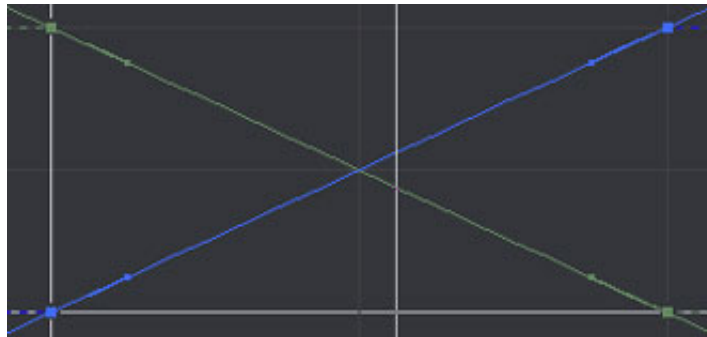
**Closeup, Default Curves**



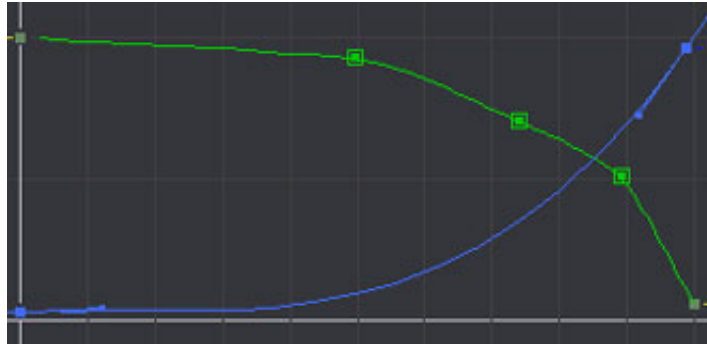
**Closeup, Modified Curves**



**Default Curves**



## Modified Curves



Because the curves can dramatically alter opacity of the foreground, you may need to ensure the interior of the matte is 100 percent opaque through the use of additional masking, garbage mattes, etc.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between the foreground (0) or the background resolution (1).
<i>preMultiplied</i>	int	1	Tells Shake if the foreground is premultiplied. This is the opposite of the standard <i>Over</i> . If enabled, the foreground curve does nothing.
<i>fgExpr</i>	float	HermiteV(x,0,[1,-50,-50]@0,[0,-50,-50]@1)	The controlling curve of the foreground. To use it, load it into the Curve Editor by clicking the small Load Curve button.
<i>bgExpr</i>	float	HermiteV(x,0,[0,50,50]@0,[1,50,50]@1)	The controlling curve of the foreground. To use it, load it into the Curve Editor by clicking the small Load Curve button.

## Synopsis

```
image AddMix(
    image Foreground,
    image Background,
    int clipMode,
    int preMultiplied,
    float fgExpr,
    float bgExpr
);
```

### Script

```
image AddMix(  
    Foreground,  
    Background,  
    clipMode,  
    preMultiplied,  
    fgExpr,  
    bgExpr  
);
```

### See Also

“Over” on page 191

### AddText

The *AddText* function differs from the *Text* image function only in that it places text over a preexisting background. See “Text” on page 574 for more information on this extremely flexible feature. *AddText* is also a bit nicer than *Text* for the command line because you do not have to actually enter all of the parameters for the image width and height, etc.

Parameters	Type	Defaults	Function
<i>text</i>	string	"Text"	The text you want to print.
<i>font</i>	string	"Utopia Regular"	The font you want to use.
<i>x,yFontScale</i>	float	100, 100	Size of the font.
<i>leading</i>	float	1	Leading to control the space between the lines.
<i>x/yPos</i>	float	width/2, height/2	The center of the text. Keep in mind this does not represent the geometric center of the text, but only in relation to the characteristics of the particular font you are using. It changes from font to font, and also by your <i>x/yAlign</i> values.
<i>x/yAlign</i>	int	2, 2	0 = left/top justified 1 = right/bottom justified 2 = centered
<i>red, green, blue, alpha</i>	float	1, 1, 1, 1	Your color values, ranging (usually) from 0 to 1.
<i>x,y,zAngle</i>	float	0, 0, 0	Rotations on the image.

Parameters	Type	Defaults	Function
<i>fieldOfView</i>	float	0	Degrees of view. If this is the value, your x- and yAngle values appear to have no effect.
<i>kerning</i>	float	0	The spacing between each letter. You can also have negative values.
<i>fontQuality</i>	float	1	The polygonalization factor of the font splines. This is conservatively set to a high value. For flat artwork, you can probably get away with a value of 0. When you have extreme perspective, you want to keep it high.

### Synopsis

```
image AddText(
    image img,
    const char * text,
    const char * font,
    float xScale,
    float yScale,
    float leading,
    float xPos,
    float yPos,
    float zPos,
    int xAlign,
    int yAlign,
    float red,
    float green,
    float blue,
    float alpha,
    float xAngle,
    float yAngle,
    float zAngle,
    float fieldOfView,
    float kerning,
    float fontQuality
);
```

## Script

```
image AddText(  
    image,  
    "text",  
    "font",  
    xScale,  
  
    Scale,  
    leading,  
    xPos,  
    yPos,  
    zPos,  
    xAlign,  
    yAlign,  
    red,  
    green,  
    blue,  
    alpha,  
    xAngle,  
    yAngle,  
    zAngle,  
    fieldOfView,  
    kerning,  
    fontQuality  
);
```

## Command Line

*-addtext text font xScale yScale etc...*

## Examples

*shake lisa.iff -addt "Hello World" "Courier" 50*

*shake -addtex "Today is %M %d, %D" "Courier" 30*

*shake lisa.iff -addtex "frame = %" "Courier" 60 -t 1-10*

## See Also

*"Text" on page 574*

### Atop

The *Atop* function is similar to *Over*, except that the background matte is used; the foreground only appears where there is background matte. Unlike *Inside*, it also keeps the background information. In other words, *Atop* is the equivalent of applying an *Inside* followed by an *Over*. Use *Atop* for applying atmosphere effects to a foreground element, such as compositing smoke over a foreground character.

Parameter	Type	Default	Function
<i>clipMode</i>	int	1	Toggles between the foreground (0) or the background resolution (1).

### Synopsis

```
image Atop(  
    image Foreground,  
    image Background,  
    int clipMode  
);
```

### Script

```
image = Atop( Foreground, Background, clipMode );
```

### Command Line

```
shake -atop image clipMode
```

### Example

```
shake lisa.iff -atop uboat.iff
```

### See Also

“*Over*” on page 191, “*Screen*” on page 192, “*Outside*” on page 190, “*Inside*” on page 176

### Common

The *Common* function compares two images and extracts or hides common elements. Use *Common* for difference mattes. The extracted elements are taken if the difference between the two images is less than the set tolerances. A similar function is *ISubA*, which subtracts two images and returns the absolute value of these images.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	0	Toggles between foreground (0) and background resolution (1).
<i>mode</i>	int	0 0 - 5	Tells Shake what to do with common elements. 0: "show" keeps common elements only, turning the rest black. 1: "hide" hides common elements, leaving different elements untouched. 2: "white" turns the image to black and white, keeping common elements white. 3: "black" is the inverse of "white." 4: "proportion" takes the difference between the values, averages the values, and then inverts the values.
<i>Tol</i>	float	0, rTol, rTol, rTol	The Tolerance values. If pixels between the two images are less than the Tolerance value, they are considered common.

### Synopsis

```
image Common(  
    image Foreground,  
    image Background,  
    int clipMode,  
    int mode,  
    float redTol,  
    float greenTol,  
    float blueTol,  
    float alphaTol  
);
```

## Script

```
image = Common(  
    Foreground,  
    Background,  
    clipMode,  
    mode,  
    redTol, greenTol,  
    blueTol, alphaTol  
);
```

## Command Line

*sbake -common image clipMode "mode" redTol etc...*

## Examples

*sbake lisa.iff -solarize .5 -common lisa.iff 1 4 .1*

*sbake lisa.iff -mmult -common lisa.iff 1 2*

## See Also

*"ISub"* on page 177, *"ISubA"* on page 178

## Constraint

*Constraint* is a multifunctional node that restricts the effect of nodes to limited areas, channels, tolerances, or fields. Toggle the type switch to select the constraint type. Certain parameters then become active; others no longer have any effect. This is similar to the *KeyMix* node in that you mix two images according to a third constraint. *KeyMix* expects an image to be the constraint. *Constraint* allows you to set other types of constraints.

The *Constraint* node also speeds calculation times considerably in many cases. The speed increase always occurs when using the ROI or field mode, and for many functions when using channel mode. Channel mode decreases calculation time when the output is a result of examining channels, such as layer operations. Calculation time is not decreased, however, when it must examine pixels, such as warps and many filters. The tolerance mode may in fact increase calculation times, as it must resolve both input images to calculate the difference between the images.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground and background resolution. 0 = foreground resolution 1 = background resolution
<i>type</i>	int	0	Selects the type of constraint you use. AOI - Area of Interest (1) - Draws a mixing box. Threshold - (2) - Only changes within a tolerance are passed on. Channel - (4) Only specific channels are modified. Field - (8) - Only a selected field is modified. Because of the labeling, you can do multiple types of constraining in the script by adding the numbers together, for example, 7 equals Area of Interest, Threshold, and Channel are all active.
<i>left, right, bottom, top</i>	float	0, width, 0, height	These are active only if type is set to 1. (See type, above.) They describe a cropping box for the effect.
<i>rTol, gTol, bTol, aTol</i>	float	0, 0, 0, 0	Tolerance values to be used if type is set to 2. (See type, above.)
<i>tLevel</i>	int	0	Active only when type is equal to 2. This sets the Tolerance to "lo" or "hi." 0 = "lo." Changes are made only if the difference between image1 and image2 is less than the Tolerance values you set. 1 = "hi." Changes are made only if the difference between image1 and image2 is greater than the Tolerance values.
<i>tReplace</i>	int	0	Active when type is set to 2. Toggles whether the entire pixel is replaced, or just the channel meeting the Tolerance criteria.
<i>channels</i>	string	rgba	If type is set to 4 (see type, above), the operation only applies to these channels.

Parameters	Type	Defaults	Function
<i>field</i>	int	0	If type is set to 8 (see type, above), effect only applies to 1 field.  0 = even field 1 = odd field
<i>invert</i>	int	0	Inverts the selection. For example, everything beyond a color tolerance is included, rather than below, etc.

### Synopsis

```
image Constraint(
    image Foreground,
    image Background,
    int clipMode,
    int type,
    int left,
    int right,
    int bottom,
    int top,
    float rTol,
    float gTol,
    float bTol,
    float aTol,
    int tLevel,
    int tReplace,
    const char * channels,
    int field
);
```

### Script

```
image Constraint(
    Foreground,
    Background,
    clipMode,
    type,
    left,
    right,
    bottom,
    top,
    rTol,
    gTol,
```

```

bTol,
aTol,
tLevel,
tReplace,
"channels",
field
);

```

### Command Line

*shake -constraint image clipMode etc...*

### Examples

*shake lisa.iff-blur 30-const lisa.iff 0 1 150 350 200 400*

*shake lisa.iff-solariz-const lisa.iff 0 2 0 0 0 0 "Linear(0,0@1,1@20)" -t 1-20*

### See Also

“KeyMix” on page 179, “Field” on page 224, “Interlace” on page 222, “SwapFields” on page 224, “SetDOD” on page 465, “Mix” on page 185

### Copy

The *Copy* function copies selected channels from Image B to Image A (replacing the images entirely). It is common to copy over the alpha channel. To copy channels within the same image, use *Reorder*. *SwitchMatte* depends on *Copy*, but is a macro with *MMult* and *Invert* placed inside of the node.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>channels</i>	string		Tells Shake what channels to copy from image B to image A. Possible channels are: R, G, B, A, Z.

### Synopsis

```

image Copy(
    image Foreground,
    image Background,
    int clipMode,
    const char * channels
);

```

**Script**

```
image = Copy(  
    Foreground, Background,  
    clipMode,  
    "channels"  
);
```

**Command Line**

```
shake -copy image clipMode channels
```

**Examples**

```
shake lisa.iff -copy parrot.jpg 0 rg  
shake lisa.iff -pan 100 100 -copy lisa.iff 1 r
```

**See Also**

“Reorder” on page 312, “SwitchMatte” on page 193, “IAdd” on page 173

**IAdd**

The *IAdd* image math function adds one image to another. The strength of the second image can be raised and lowered with the percent parameter.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>percent</i>	float	100	A built-in fade parameter. This is the amount of the second image that is taken into consideration. Full strength is 100. If set to 50, the added amount is reduced by 50 percent.

**Synopsis**

```
image IAdd(  
    image Foreground,  
    image Background,  
    int clipMode,  
    float percent  
);
```

### Script

```
image = IAdd(  
    Foreground,  
    Background,  
    clipMode,  
    percent  
);
```

### Command Line

*shake -iadd image clipMode percent*

### See Also

“*IMult*” on page 175, “*ISubA*” on page 178, “*ISub*” on page 177, “*IDiv*” on page 174

### IDiv

The *IDiv* image math function divides one image by another. The strength of the second image can be raised and lowered with the percent parameter. To divide an image by its own matte channel, use *MDiv*.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>percent</i>	float	100	A built-in fade parameter. This is the amount of the second image that is taken into consideration. Full strength is 100. If set to 50, the added amount is reduced by 50 percent.
<i>ignoreZero</i>	int	0	Tells Shake to ignore black (zero) values.

### Synopsis

```
image IDiv(  
    image Foreground,  
    image Background,  
    int clipMode,  
    float percent,  
    int ignoreZero  
);
```

**Script**

```
image = IDiv(  
    Foreground,  
    Background,  
    clipMode,  
    percent,  
    ignoreZero  
);
```

**Command Line**

```
shake -idiv image clipMode percent ignoreZero
```

**See Also**

“*IMult*” on page 175, “*ISubA*” on page 178, “*ISub*” on page 177, “*IAdd*” on page 173, “*MDiv*” on page 311

**IMult**

The *IMult* image math function multiplies one image by another. The second image’s strength can be raised and lowered with the percent parameter. To multiply an image by its own matte channel, use *MMult*.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>percent</i>	float	100	A built-in fade parameter. This is the amount of the second image that is taken into consideration. Full strength is 100. If set to 50, the added amount is reduced by 50 percent.
<i>ignoreZero</i>	int	0	Tells Shake to ignore black (zero) values.

**Synopsis**

```
image IMult(  
    image Foreground,  
    image Background,  
    int clipMode,  
    float percent,  
    int ignoreZero  
);
```

### Script

```
image = IMult(  
    Foreground,  
    Background,  
    clipMode,  
    percent,  
    ignoreZero  
);
```

### Command Line

*shake -imult image clipMode percent ignoreZero*

### See Also

“IDiv” on page 174, “ISubA” on page 178, “ISub” on page 177, “IAdd” on page 173, “MMult” on page 311

### Inside

The *Inside* function places one image only inside the mask of a second image, so only the mask of the second image is considered in the composite; the rest comes from the color of the foreground image. This is an extremely useful function for masking images.

Parameter	Type	Default	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).

### Synopsis

```
image Inside(  
    image Foreground,  
    image Background,  
    int clipMode  
);
```

### Script

```
image = Inside( Foreground, Background, clipMode );
```

### Command Line

*shake -inside image clipMode*

### See Also

“Over” on page 191, “Screen” on page 192, “Outside” on page 190, “Atop” on page 167

### Interlace

For information on the *Interlace* function, see “*Interlace*” on page 222.

### ISub

The *ISub* image math function subtracts one image from another. The strength of the second image can be raised and lowered with the percent parameter. If you are using this to detect the difference between two images, use *ISubA* instead, since it takes the absolute value of the difference.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>percent</i>	float	100	A built-in fade parameter. This is the amount of the second image that is taken into consideration. Full strength is 100. If set to 50, the added amount is reduced by 50 percent.

### Synopsis

```
image ISub(  
    image Foreground,  
    image Background,  
    int clipMode,  
    float percent  
);
```

### Script

```
image = ISub(  
    Foreground,  
    Background,  
    clipMode,  
    percent  
);
```

### Command Line

*sbake -isub image clipMode percent*

### See Also

“*IMult*” on page 175, “*IAdd*” on page 173, “*ISubA*” on page 178, “*IDiv*” on page 174

## ISubA

The *ISubA* function is identical to *ISub*, except that it takes the absolute value of the result. This is to see if similar images are different by subtracting one image from another. If the second image is brighter, *ISub* does not reveal anything, but *ISubA* does (for example,  $.5 - .8 = -.3$ , absolute value of  $-.3 = .3$ ).

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>percent</i>	float	100	A built-in fade parameter. This is the amount of the second image that is taken into consideration. Full strength is 100. If set to 50, the added amount is reduced by 50 percent.

### Synopsis

```
image ISubA(  
    image Foreground,  
    image Background,  
    int clipMode,  
    float percent  
);
```

### Script

```
image = ISubA(  
    Foreground,  
    Background,  
    clipMode,  
    percent  
);
```

### Command Line

*shake -isuba image clipMode percent*

### See Also

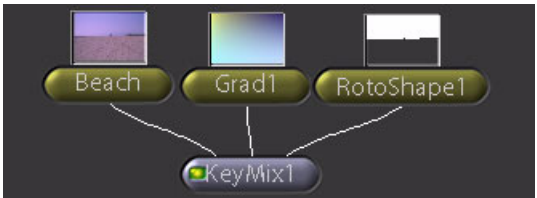
*IMult* on page 175, *IAdd* on page 173, *ISub* on page 177, *IDiv* on page 174

### KeyMix

The *KeyMix* function mixes two images together through the specified channel of a third image. You can control the mix percentage, and also invert the masking image. *KeyMix* and *Over* are the two primary compositing nodes—*KeyMix* for unpremultiplied images and *Over* for premultiplied images. For more information on premultiplication, see “The Definition of Premultiplied” on page 368.

### Example

In the following node tree, by mixing through the *RotoShape* node, you can color correct the beach and position the sky separately.



**Grad1**



**RotoShape1**



**Beach**



**KeyMix1**



Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>channel</i>	string	"a"	The masking channel from the third image.

Parameters	Type	Defaults	Function
<i>mixPercent</i>	float	100	The percentage of the second image mixed in.
<i>invert</i>	int	0	Inverts the masking channel of the third image.

### Synopsis

```
image KeyMix(
    image Foreground,
    image Background,
    int clipMode,
    const char * channel,
    float mixPercent
    invert
);
```

### Script

```
image = KeyMix(
    Foreground,
    Background,
    clipMode,
    "channel",
    mixPercent,
    invert
);
```

### Command Line

*shake -keymix Background clipMode channel mixPercent invert*

### See Also

“*Mix*” on page 185, “*IMult*” on page 175, “*IAdd*” on page 173

### Layer

The *Layer* function incorporates most of the other layer operations, with the exception of *KeyMix* and *AddText*. There is no particular advantage to using *Layer* instead of one of the specific layer operations, except that you can test different modes quickly. For a list of Photoshop layer modes, see “Photoshop Layering Modes” on page 156.

Parameters	Type	Defaults	Function
<i>operation</i>	string	"Over"	<p>The expression used for the composite.</p> <p>Possible values are:</p> <p>Atop—Places foreground (fg) over background (bg) only inside of bg's mask.</p> <p>Common—Returns the differences or commonalities between two images.</p> <p>Constraint—Limits one image and another by area, tolerance, channel, or field.</p> <p>Copy—Copies channels from bg to fg.</p> <p>IAdd—Adds fg to bg.</p> <p>IDiv—Divides fg by bg.</p> <p>IMult—Multiplies fg by bg.</p> <p>Inside—Places fg inside of bg's mask.</p> <p>ISub—fg minus bg.</p> <p>ISubA—Absolute value of fg minus bg.</p> <p>Max—Takes greater pixel value of fg or bg.</p> <p>Min—Takes lesser pixel value of fg or bg.</p> <p>Mix—Mixes fg and bg by bgPercent. 0 is all fg, 100 is all bg.</p> <p>Outside—Places fg outside of bg's mask.</p> <p>Over—Places fg over bg. Default mode.</p> <p>Screen—Inverts both fg and bg, multiplies them together, and inverts back. This is good for reflections and glows.</p> <p>SwitchMatte—Copies a channel from bg to the alpha channel of fg.</p> <p>Under—Places fg under bg.</p> <p>Xor—Performs Boolean to keep only non-common elements.</p> <p>ZCompose—Evaluates the Z information in the two images.</p>

### Synopsis

```
image Layer(
    image Foreground,
    image Background,
```

```

        string operation,
        ...
    );

```

### Script

```

image = Layer(
    Foreground,
    Background,
    "operation",
    ...
);

```

### Command Line

Don't use *Layer*. Instead, use the compositing operator you want. For example, don't use:

*layer bg "over"*

Instead use:

*enter -over bg*

### See Also

“*Atop*” on page 167, “*IAdd*” on page 173, “*IDiv*” on page 174, “*IMult*” on page 175, “*Inside*” on page 176, “*ISub*” on page 177, “*ISubA*” on page 178, “*Max*” on page 183, “*Min*” on page 184, “*Mix*” on page 185, “*Outside*” on page 190, “*Over*” on page 191, “*Screen*” on page 192

### LayerX

The *LayerX* function is exactly like *ColorX*, except that you also have access to a second image. This allows you to composite with expressions. The values of the second image are accessed with the variables *r2*, *g2*, *b2*, *a2*, and *z2*. For more information, see “*ColorX*” on page 302.

Parameter	Type	Default	Function
<i>r</i> , <i>g</i> , <i>b</i> , <i>a</i> , <i>zExpr</i>	float	<i>r</i> , <i>g</i> , <i>b</i> , <i>a</i> , <i>z</i>	The expression used for each individual channel.

### Synopsis

```

image LayerX(
    image Foreground,
    image Background,
    float expression rExpr,
    float expression gExpr,
    float expression bExpr,

```

```

float expression aExpr,
float expression zExpr
);

```

### Script

```

image = LayerX(
    Foreground,
    Background,
    rExpr,
    gExpr,
    bExpr,
    aExpr,
    zExpr
);

```

### Command Line

*shake -layerx image expr*

### See Also

*“ColorX” on page 302*

### Max

The *Max* function compares two images and takes the pixel with the higher value. This is a good tool to merge alpha masks.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>percent</i>	float	100	A gain on the second image.

### Synopsis

```

image Max(
    image Foreground,
    image Background,
    int clipMode,
    float percent
);

```

### Script

```

image = Max(
    Foreground,
    Background,

```

```
clipMode,  
percent  
);
```

**Command Line**

```
shake -max image clipMode percent
```

**See Also**

“Min” on page 184

**Min**

The *Min* function compares two images and takes the pixel with the lower value. Like the *Max* function, this is a good tool to merge alpha masks.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>percent</i>	float	100	A gain on the second image.

**Synopsis**

```
image Min(  
    image Foreground,  
    image Background,  
    int clipMode,  
    float percent  
);
```

**Script**

```
image = Min(  
    Foreground,  
    Background,  
    clipMode,  
    percent  
);
```

**Command Line**

```
shake -min image clipMode percent
```

**See Also**

“Max” on page 183

### Mix

The *Mix* function mixes two images together according to a percentage. Animate the *mixPercent* parameter to create a dissolve between two clips.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>mixPercent</i>	float	50	The percentage of each image. The default is 50. 0 = nothing of image 2 100 = nothing of image 1
<i>channels</i>	string	"rgba"	What channels are mixed. Default is "rgba."

### Synopsis

```
image Mix(  
    image Foreground,  
    image Background,  
    int clipMode,  
    float mixPercent,  
    const char * channels  
);
```

### Script

```
image = Mix(  
    Foreground,  
    Background,  
    clipMode,  
    mixPercent,  
    "channels"  
);
```

### Command Line

*shake -mix Background clipMode mixPercent etc...*

### See Also

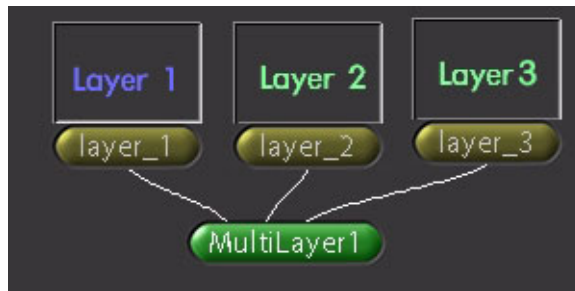
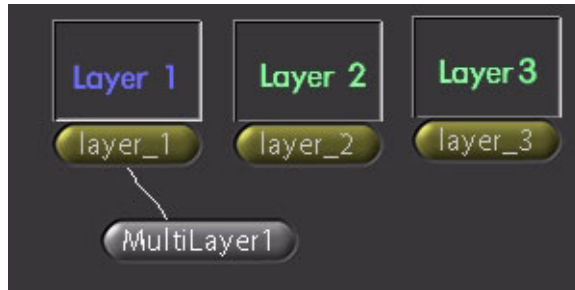
*"KeyMix" on page 179*

## MultiLayer

This node accepts an infinite amount of input images, each layer containing its own unique settings to control compositing mode, opacity, and channels. You can rearrange the layers via drag and drop in the Parameters tab, and it allows you to work using a more layer-based, rather than node-based, logic. It also helps clean up the Node View.

### Using the MultiLayer Node

The *MultiLayer* node accepts a variable amount of inputs. Drag the inputs to the + sign on the top of the *MultiLayer* node that appears when the cursor passes over it.

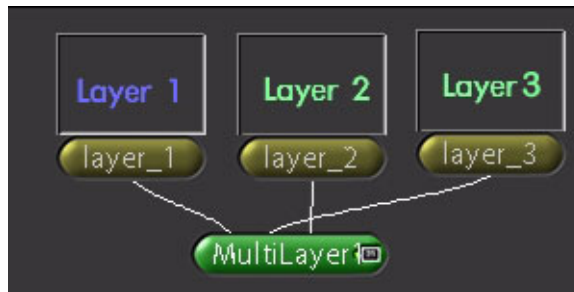


Unlike the other Layer nodes, the first input represents the background layer, the second node is the next deepest, etc., until you reach the input furthest to the right, representing the foreground. When the nodes are inserted, they are stacked in the Parameters tab as well, with the background at the bottom, foreground at the top.

This compositing order may be rearranged by rewiring the node in the Node View, or by dragging the Reposition button in the Parameters tab .







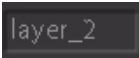
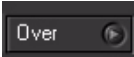

Drag the Reposition button up and down between other layers until a horizontal line appears. The compositing order is rearranged, and the nodes rewired in the Node View. Therefore, in the following illustration, *Layer\_1* is the background, and *Layer\_2* is the most prominent foreground layer.



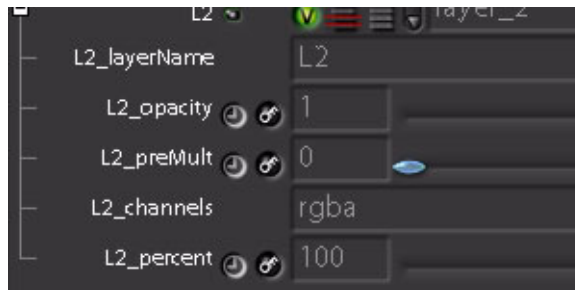
Each layer has associated parameters and controls. In the following illustration, several controls are visible on each line.



These controls are, in order:

Shake Layer Function	Explanation
	Shows the input image for the layer in the Viewer.
	Toggles the visibility for the layer.
	Solos the layer, making all other layers invisible.
	Ignore Above turns off all layers above the current layer, keeping only the current layer and those behind it visible.
	The name of the input image node. By blanking it out, the node is disconnected, but the layer information remains.
	The compositing operation for the layer.
	Deletes the current layer.

In the subtree of a layer, you can controls its parameters. Note the parameter names are all prefixed by *layerName\_* (*L2\_* in the following image). The only tricky parameter is channels—it determines what channels get bubbled up from below. For example, to add several files together to fill the matte in a *Keylight* node, insert all the mattes first, then the *Keylight* node on top of the list, using “a” as your channel for that layer.



To rename a layer, expand the subtree and enter the new name in the *layerName* text field.

Parameter	Type	Default	Function
<i>version</i>	string	1	The version number for version compatibility purposes. Should not be changed by the user.
<i>clipLayer</i>	string	"L1"	You select the input layer that determines the output resolution.
<i>postMMult</i>	int	1	Premultiplies the node.
<i>img</i>	image	0	The input image for that layer.
<i>compParamCount</i>	int	-	This parameter appears only in the script. It tells Shake how many extra parameters to add, based on unique options for each mode.
<i>layerName</i>	string	"L1"	The name of the layer. All associated parameters for that layer are prefixed by the <i>layerName</i> .
<i>opacity</i>	float	1	Opacity of the input layer. With an imported Photoshop file, there is an additional PSOpacity parameter. See "Photoshop Layering Modes" on page 156 for more information.
<i>preMult</i>	int	0	Indicates if the layer is to be premultiplied.
<i>compType</i>	string	"Over"	The compositing mode. All modes from the Layer node are accepted, including most Photoshop modes.
<i>visible</i>	int	1	Can deactivate that layer.
<i>solo</i>	int	0	Deactivates all other layers.
<i>ignoreAbove</i>	int	0	Ignores all layers above (in front of) this layer.
<i>compChannels</i>	string	"rgba"	What channels are passed up from below (behind) this layer.
[...]			Repeats the last 10 parameters to make an extra layer.

### Synopsis

```
image MultiLayer(  
    const char * version,  
    const char * clipLayer,  
    image img,  
    int compParamCount,
```

```

const char * layerName,
float opacity,
int preMult,
const char * compType,
int visible,
int solo,
int ignoreAbove,
const char * compChannels,
[...] (repeat last 10 parameters)
);

```

### Script

```

image MultiLayer(
    "version",
    "clipLayer",
    img,
    compParamCount,
    "layerName",
    opacity,
    preMult,
    "compType",
    visible,
    solo,
    ignoreAbove,
    "compChannels",
    [...] (repeat last 10 parameters)
);

```

### Command Line

Not really a command-line thing.

### Outside

The *Outside* function places only one image outside of the mask of a second image. Therefore, only the mask of the second image is considered in the composite; the color comes from the foreground image. This is a great tool to mask layers.

Parameter	Type	Default	Function
<i>clipMode</i>	int	0	Toggles between foreground (0) and background resolution (1).

### Synopsis

```
image Outside(  
    image Foreground,  
    image Background,  
    int clipMode  
);
```

### Script

```
image = Outside( Foreground, Background, clipMode );
```

### Command Line

*shake -outside image clipMode*

### See Also

“Over” on page 191, “Screen” on page 192, “Inside” on page 176, “Atop” on page 167

### Over

The *Over* function is the main compositing node of Shake. This function places one image over another, according to the matte of the foreground image. Images are assumed to be premultiplied. You can use a *MMult* on an input node if it is not premultiplied, or you can toggle the *preMultiply* parameter to 1.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>preMultiply</i>	int	0	When this is on (1), the foreground image is multiplied by its alpha mask. If it is off (0), it assumes the foreground image is premultiplied.
<i>addMattes</i>	int	0	When enabled (1), the mattes are added together for the composite.

### Synopsis

```
image Over(  
    image Foreground,  
    image Background,  
    int clipMode,  
    int preMultiply,  
    int addMattes  
);
```

### Script

```
image = Over(  
    Foreground,  
    Background,  
    clipMode,  
    preMultiply,  
    addMattes  
);
```

### Command Line

*shake -over image clipMode preMulti*

### See Also

“Under” on page 194, “Screen” on page 192, “Inside” on page 176, “Atop” on page 167

### Screen

The *Screen* function mimics the effect of exposing two film negatives together. Technically, it inverts both layers, multiplies the two layers together, and inverts the result. It is particularly handy for reflections and luminescent elements, as it preserves the highlights.

Parameter	Type	Default	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).

### Synopsis

```
image Screen(  
    image Foreground,  
    image Background,  
    int clipMode  
);
```

### Script

```
image = Screen(Foreground, Background, clipMode);
```

### Command Line

*shake -screen image clipMode*

### See Also

“Over” on page 191, “Atop” on page 167, “Outside” on page 190, “Inside” on page 176

## SwitchMatte

The *SwitchMatte* function is a *Copy* command dedicated to copying a channel from the second image into the matte channel of the first image. You can choose the source channel, if the channel is inverted, and if the result is premultiplied.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>channelMatte</i>	string	"a"	Tells Shake what channels to copy from image B to the image A alpha channel.
<i>matteMult</i>	int	1	Enables premultiplication of the resulting image by the new alpha. 0 = no premultiply 1 = premultiply matte
<i>invertMatte</i>	int	0	Inverts the result matte. 0 = do not invert the matte 1 = invert the matte

### Synopsis

```
image SwitchMatte(  
    image Foreground,  
    image Background,  
    int clipMode,  
    const char * matteChannel  
    int matteMult,  
    int invertMatte  
);
```

### Script

```
image = SwitchMatte(  
    Foreground,  
    Background,  
    clipMode,  
    "channelMatte",  
    matteMult,  
    invertMatte  
);
```

**Command Line**

*shake -switchmatte image clipMode "channelMatte" etc...*

**See Also**

“Reorder” on page 312, “Copy” on page 172, “Add” on page 173

**Under**

The *Under* function is the same as the *Over* command, except that it reverses the two input images. This is largely trivial for the script and the interface, but is extremely handy when working in command-line mode, since you can easily work on a background image.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>preMultiply</i>	int	0	When this is on (1), the foreground image is multiplied by its alpha mask. If it is off (0), it assumes the foreground image is premultiplied.
<i>addMattes</i>	int	0	When enabled (1), the mattes are added together for the composite.

**Synopsis**

```
image Under(  
    image Background,  
    image Foreground,  
    int clipMode,  
    int preMultiply,  
    int addMattes  
);
```

**Script**

```
image = Under(  
    Background,  
    Foreground,  
    clipMode,  
    preMultiply,  
    addMattes  
);
```

**Command Line**

*shake -under image clipMode preMulti*

**See Also**

“Over” on page 191, “Screen” on page 192, “Inside” on page 176, “Atop” on page 167

**Xor**

The *Xor* function is somewhat like a combination of the *Inside* and *Outside* nodes. It is used to remove overlapping mask areas.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>preMultiply</i>	int	0	When this is on (1), the foreground image is multiplied by its alpha mask. If it is off (0), it assumes the foreground image is premultiplied.
<i>addMattes</i>	int	0	When enabled (1), the mattes are added together for the composite.

**Synopsis**

```
image Xor(  
    image Foreground,  
    image Background,  
    int clipMode,  
    int preMultiply,  
    int addMattes  
);
```

**Script**

```
image = Xor(  
    Foreground,  
    Background,  
    clipMode,  
    preMultiply,  
    addMattes  
);
```

**Command Line**

```
shake -xor image clipMode preMult
```

**See Also**

“Under” on page 194, “Screen” on page 192, “Inside” on page 176, “Atop” on page 167

### ZCompose

The *ZCompose* function composes the input image over the background image using the Z values of both images to determine what image's pixel is placed on top.

Z values are depth measurements from the camera to each object. A premultiplied image is assumed since Z values are normally produced by a render. With a *ZCompose*, one of the image's pixels is either on top or not. This is sometimes a problem if you have subpixel objects. You might have ten dust particles in a single pixel, but if only one is in front of the corresponding object in the other image, then the pixel containing all ten particles is placed in front of that object. The problem is that the pixel does not contain the subpixel objects, just a representative color. For similar reasons, object edges can also be a problem.

For a tutorial on Z-compositing, see “Lesson Three: Depth Compositing” in the *Shake 3 Tutorials*.

Parameter	Type	Default	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).

#### Synopsis

```
image ZCompose(  
    image Foreground,  
    image Background,  
    int clipMode  
);
```

#### Script

```
image = ZCompose(  
    Foreground,  
    Background,  
    clipMode  
);
```

#### Command Line

*shake -zcompose image clipMode*

#### See Also

*“LayerX” on page 182*

## Other Compositing Functions

Shake contains other useful compositing functions that are located in the Other Tool Tab.

**Note:** The *DepthKey* and *DepthSlice* functions modify the alpha channel of an image based on Z-depth values. For more information, see Chapter 7, “Keying,” on page 237.

### AddShadow

The *AddShadow* function takes the alpha channel of an image, and then colors, blurs, fades, and moves the alpha channel. Next, the alpha is composited under the input, creating a cheap shadow effect. It differs from *DropShadow* in that it already composites the shadow under the element; *DropShadow* creates an entirely new shadow element that can be manipulated separately.

Parameters	Type	Defaults	Function
<i>x, yOffset</i>	float	40, -40	The amount of pixels the shadow moves away from its origin.
<i>fuzziness</i>	float	60	The amount of, um, fuzziness on the shadow.
<i>r, g, b Shadow</i>	float	.01, .02, .05	Color of the shadow.
<i>opacity</i>	float	.6	A fade value on the shadow, between 0 and 1.

### Synopsis

```
image AddShadow( image,  
    float xOffset,  
    float yOffset,  
    float fuzzyness,  
    float rShadow,  
    float gShadow,  
    float bShadow,  
    float opacity  
);
```

### Script

```
image = AddShadow( image, xOffset, yOffset,  
    fuzzyness, rShadow, gShadow, bShadow,  
    opacity );
```

### Command Line

*shake -addshadow xOffset yOffset fuzzyness etc...*

### Example

*shake man.iff -addsb 20 -20 10 .05 0 .1 .7 -over hallway.jpg*

### DropShadow

The *DropShadow* function is similar to *AddShadow*, except that it gives you a separate shadow element that can be moved and manipulated independently from the original source.

Parameters	Type	Defaults	Function
<i>fuzziness</i>	float	60	The amount of fuzziness on the shadow.
<i>r, g, b Shadow</i>	float	.01, .02, .05	Color of the shadow.
<i>opacity</i>	float	.6	A fade value on the shadow.

### Synopsis

```
image DropShadow( image,  
    float fuzzyness,  
    float rShadow,  
    float gShadow,  
    float bShadow,  
    float opacity  
);
```

### Script

```
image = DropShadow( image,  
    fuzzyness,  
    rShadow, gShadow, bShadow,  
    opacity  
);
```

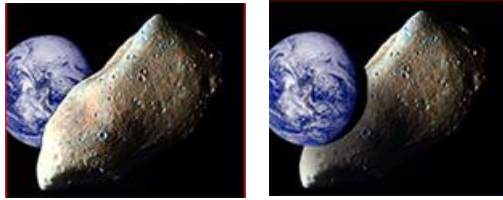
### Command Line

*shake -dropshadow fuzzyness rShadow gShadow etc...*

### Select

The *Select* function, located in the Other Tool Tab, allows you to select between multiple inputs. This function can be useful when you want to switch between different compositing logic over time, or when you have “doubles” that you want to insert. For example, you can switch a blue screen character with a CG replacement using the *Select* node.

In the following simple example, you can specify if the asteroid is in front or behind the Earth by putting two different *Over* operations into *Select* and toggling between the two operations.



Select has an infinite amount of potential inputs.

### **Synopsis**

```
image Select(  
    int branch,  
    image,  
    image,  
    ...  
);
```

### **Script**

```
image = Select( branch, image, image,...);
```

### **Command Line**

```
shake -select branch image image image image
```



# File Formats and Footage

## Chapter Summary

- File Formats
- About Resolution
- Functions Affecting Image Resolution
- Working With Video
- Video Functions
- About Aspect Ratio and Non-Square Pixels
- Cropping Functions

## File Formats

Shake can read in and write out a variety of image file formats. These formats all assume one image file per frame. Streaming formats such as QuickTime are also available, but only on Macintosh OS X systems.

These individual images can be numbered with padding (0001, 0002, etc.) or without padding (1, 2, etc.), with a file extension or without one. If Shake is unable to find a frame, it assumes you are using a 720 x 486 black frame (by default) and continues the composite with that image instead.

It is good practice to include the file extension (for example, .iff, .cin, .tif, etc.), but Shake does not necessarily need it. In general, you use the extension to define the input or output format, but you can also explicitly set it to a certain type if your files are not using extensions.

Shake is a hybrid renderer—it adapts its rendering from either scanlines or a group of tiles. This means it never has to load the entire image, just a single piece of the image, making a much smaller memory footprint than other compositors. Sometimes you cannot load just a single line, for example, during a *Rotate*, in which case Shake internally breaks the image down into small tiles to work with more manageable bits.

For a complete list of file formats and their characteristics, see the following section, “Supported File Formats.” Also, see the “Table of File Sizes” on page 207.

### Supported File Formats

BW[A] is either BW or BWA. BW[A][Z] is any combination of BW, alpha, and Z. RGB[A] and RGB[A,Z] are optional additions of alpha or Z channels.

**Note:** Targa and SGI have different input/output options for Channels. When you write a BWA image, it is converted to RGBA. Also, many options must be explicitly stated when in command-line mode. For example, Cineon and JPEG files always write in RGB unless you specify every argument found in the *FileOut* node for Cineon or JPEG in the interface.

Compression Controls indicate any special compression techniques. Note that Cineon and YUV have no compression.

An asterisk indicates additional format notes (following the table).

Ext	Image Format	Input Channels	Output Channels	Compression	Bit Depth	Tmp Files
iff*, or no extension	Shake native	BW[A, Z], RGB[A, Z]	Same		8, 16, float	No
nri	Shake icon (only for GUI icons)	RGB[A]	Same		8	No
iff*	Alias/Wavefront Maya (licensed from Apple)	RGB[A, Z]	Same		8, 16, float	No
als, alias, (pix)	Alias/Wavefront Alias	RGB	Same		8	Yes
alsz	Alias/Wavefront Alias Z buffer	Z, BW[A, Z], RGB[A, Z]	Same		8,16, float	Yes
avi*	Microsoft video file format	RGBA	Same	Lossy, from 0 to 1, 1 = high quality		
bmp, dib	BMP	RGB	Same		8	
ct, ct16, mray	Mental Ray	RGBA	Same		8,16, float	No

Ext	Image Format	Input Channels	Output Channels	Compression	Bit Depth	Tmp Files
cin*	Kodak Cineon	RGB[A]	Same	None	16 (10 on disk)	Yes
dpx	DPX reader courtesy of Michael Jonas, Das Werk Gmbh, modified by Apple.	RGBA	Same	-	8, 16	Yes
.gif (read only)	GIF	RGB	-	-	8	No
jpeg, jpg, jfif*	JPEG	BW, RGB	Same	Lossy, from 0 to 100%. 100 = high quality	8	Yes
pbm, ppm, pnm, pgm	PBM	BW, RGB	Same		8	Yes
pic	Softimage	RGB[A]	Same		8	Yes
png	PNG	RGB[A], BW [A]	Same		8, 16	No
psd*	Adobe Photoshop	RGB[A]	RGBA		8, 16	
QuickTime*	Apple video file format, multiple codecs supported	RGB[A]	Same	Lossy, from 0 to 1, 1 = high quality	8, 16	
rgb, sgi, bw, raw, sgi raw*	SGI	BW[A], RGB[A]	RGB[A]	Lossless RLE	8, 16	No
rla*	Alias/Wavefront RLA (supports Z buffer)	BW[A,Z], RGB[A,Z]	Same		8, 16, float	No

Ext	Image Format	Input Channels	Output Channels	Compression	Bit Depth	Tmp Files
rpf*	RLA Rich Pixel Format. Use this type when saving RLA files with Z depth to be read into After Effects. Make sure the file extension is still .rla, but set the format to .rpf.	BW[A,Z], RGB[A,Z]	Same		8, 16, float	No
tdi	Alias/Wavefront Explore format (identical to .iff)	BW[A, Z], RGB[A,Z]	Same		8, 16, float	No
tdx	Alias/Wavefront Explore Tiled Texture Map	BW[A, Z], RGB[A,Z]	Same		8, 16, float	No
tga	Targa	RGB[A]	RGB[A]	On/Off	8	Yes
tif, tiff	TIFF	BW[A], RGB[A]	Same	4 options, see below.	8, 16, float	Yes
xpm	XPM	RGB[A]	Same		8	
yuv, qnt, qtl, pal*	YUV/Abekas/Quantel	RGB	Same	Uncompressed files with YUV encoding.	8	Yes
yuv 10-bit	Same	RGB	Same	Uncompressed files with YUV encoding	16 (10)	Yes

## Extensions

### iff

The Shake .iff format is not the same as the Amiga format with the same extension, although they share certain structural similarities. The .iff format is licensed to Alias/Wavefront for use with Maya, so Shake is ideally suited to work with Maya. Since Shake deals with this format internally, you get the best performance by maintaining your intermediate images in this format as well. It can be 8, 16, or 32 bits per channel, as well as maintain logarithmic information, alpha, and Z channels. Currently, not many packages explicitly support this format, but if the package supports the old TDI (.tdi) format, it works with .iff as well (for example, with Interactive Effect's Amazon 3D Paint).

### cin

Shake works with images bottom-up, meaning 0,0 is at the bottom-left corner. The Cineon and TIFF formats allow you to write the files either bottom-up or top-down. Because of Shake's bottom-up nature, the I/O time (actual render time remains the same) is four times greater when dealing with top-down Cineon or TIFF files. You can set how Shake writes the images—reading either way is no problem, except for the speed hit. This information is placed in a *startup* .h file.

Add the following lines to an .b file in your *startup* directory. (For more information on setting up your own .b files, see Chapter 17, "Customizing Shake.")

```
script.cineonTopDown = 1;
```

```
script.tiffTopDown = 1;
```

You can also set environment variables in your .cshrc (or .tcshrc or whatever):

```
setenv NR_CINEON_TOPDOWN
```

```
setenv NR_TIFF_TOPDOWN
```

Set these variables to force Shake into top-down mode.

By default, Cineon and TIFFs are set to the slower top-down, since many other software products do not recognize bottom-up images. If you write a bottom-up image and it appears upside down in another software package, you have four choices:

- Reset the TopDown switch/environment variable, and save the image again in Shake.
- Flip the image in Shake before saving it.
- Flip the image in the other software.
- Call the other vendor and request they properly support the file formats in question.

### jpeg, jpg, jfif

In the *FileOut* you can set the quality level and determine which channels are present in the file.

### **mov, avi**

The QuickTime format is available on Macintosh systems only. Avi files are written through QuickTime. If you select either as your output format, you have three additional options:

- **codec**—pop-up menu with a list of all available compressors. The default codec is Animation (RLE compression).
- **compressQuality**—0-1. A high value sets high quality and a larger file size.
- **framesPerSecond**—this is embedded in the file.

When QuickTime files are rendered from the interface, the Shake Viewer displays the thumbnail. You must close this window before the QuickTime file is actually completed. For this same reason, you cannot write over the file on disk while it is still being viewed.

### **psd (Photoshop) files**

There are two ways to import Photoshop files. First, you can use a *FileIn* node to import a .psd file and select either the merged layers or a single layer. These controls are located in the *FileIn* parameters.

The second way to import a Photoshop file is to use the File > Import Photoshop Script command. Each layer is imported as a separate file and fed into a *MultiLayer* node. For information on the Photoshop layering modes, see “Photoshop Layering Modes” on page 156.

### **rgb, sgi, bw, raw, sgiraw**

You have the option to set the channels saved into the output file.

### **rla, rpf**

Adobe After Effects and Autodesk 3ds max do not properly support the original Wavefront specifications for the RLA file concerning the Z channel. Therefore, you have to write the image in a specific format—rpf (Rich Pixel Format) in the *FileOut* with the .rla file extension in the filename, or else these packages do not recognize the extension.

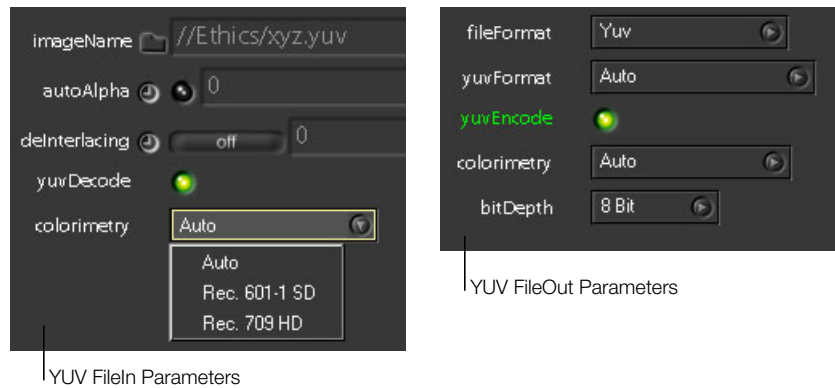
### **yuv, qnt, qtl, pal**

You have the choice to write out in NTSC, PAL, or 1920 x 1080 4:2:2 8 bit. You can also write out 10-bit YUV files.

**Note:** The .qnt and .qtl options do not appear in the fileFormat list in the *FileOut* parameters, and must be entered manually when setting your FileOut name.

When yuvFormat is set to “Auto,” the resolution is automatically determined by the resolution of the input node. The selected resolution is the smallest possible to fit the entire image. For example, if the image is smaller than NTSC, it is NTSC. If it is between NTSC and PAL, it is PAL; otherwise it is HD. You can also manually select the resolution. The *script.videoResolution* is no longer used for this purpose.

**Note:** The YUV file reader and writer supports Rec. 709 and Rec. 601-1 colorimetry coefficients, used primarily in HD YCbCr (HD-SDI).



### Table of File Sizes

In the following table, all sizes are for 3-channel images. Note that many images support optional alpha or Z channels, which add to the file size. A single channel image is typically one-third the size. The two sizes listed in each cell are for a *Ramp* (an example of extreme compression), and a completely random image, each in MB. Normal plates tend to be in between, usually closer to the higher value. This can give you a very wide variation in an image. For example, an .iff goes from 2.5 MB for a 2K 8-bit ramp to 9.1 MB for the same size/depth with a random image. If only one entry is listed, it is an uncompressed file.

Ext	NTSC, 8 Bits	NTSC, 16 Bits	NTSC, Float	2K, 8 Bits	2K, 16 Bits	2K, Float
cin (10 bits)		1.3		12.2		
iff	.74   1	1.7   2	2.9   3.9	2.5   9.1	11.5   18.2	22.5   35.8
jpg (100 %)	.02   .48		1.4   4.3			
mray	1.3	2.7	5.3	12.2	24.3	48.6
pic	.9   1		1.5   9.1			
rla	.8   .92	1.8   2	4	2.3   9.2	11.5   18.4	36.5
sgi	.74   1	1.8   2		2.3   9.3	16.5   18.5	
tif	.04   1.4	.07   1.6	3   3.7	.25   12.5	.2   17.8	27.4   33.3

Ext	NTSC, 8 Bits	NTSC, 16 Bits	NTSC, Float	2K, 8 Bits	2K, 16 Bits	2K, Float
xpm	.68   1.2		6.1   10.6			
yuv	.68		4 (HD)			

### About Resolution

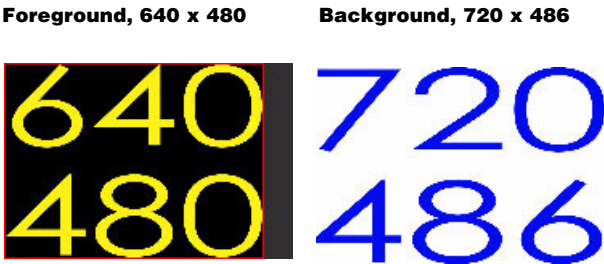
Shake has no “working resolution” to set. Resolution is determined by the input images. If a D1 resolution image is read in, you work at D1. When you composite two different resolution images, you are given the option to select either the foreground (first image) or the background (second image) resolution. For example, if you read in 50 2048 x 1556 elements or images, you are working at 2048 x 1556. If you composite all of the images over a 720 x 486 background, and you want the background resolution of that composite, then the foreground elements are cropped to the video resolution, or you can choose to remain at the 2048 x 1556 resolution.

Because of the Infinite Workspace, you don’t have to crop a lower-resolution element to match a higher-resolution element when compositing or applying transformations.

You can view the resolution of an element in the title bar of the Viewer, or position the cursor over the node and look at the help text in the lower-right window.

### Changing Resolution

You can change the resolution of an element in several ways. In the following examples, a 640 x 480 foreground element is composited over a 720 x 486 NTSC D1 element. The dark grey area designates space outside of the image frame.



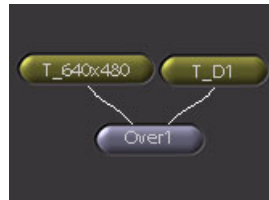
- Using Layering Nodes to Crop the Frame

To set resolution by compositing an image over a lower or higher resolution element (including a pure black plate generated with the *Color* function), select the background or foreground resolution in the clipMode parameter in the layering node.

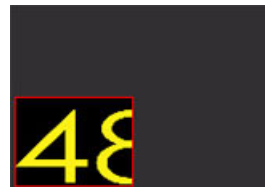
**Tree**

**Background, 720 x 486**

**Foreground, 640 x 480**



In the following example, a 320 x 240 black frame is created with an Image-*Color* node. The resolution of the foreground and background elements is set to 320 x 240 by assigning the background clipMode in the second *Over*.

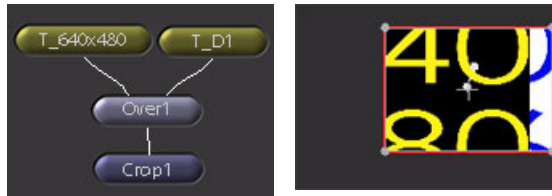


- Using the Transform nodes *Crop*, *Window*, or *Viewport* to crop the frame

The following three nodes cut into the frame without scaling and filtering the pixels.

- *Crop*: Arbitrarily supplies the lower-left and upper-right corners, and cuts into the frame at those coordinates. Also turns off the Infinite Workspace.
- *Window*: Supplies a lower-left coordinate, and then the image's resolution. Cuts off the Infinite Workspace.
- *Viewport*: Same as *Crop*, except it maintains the Infinite Workspace. This is useful for setting resolutions in preparation for later use of onscreen controls, as the controls are always fitted to the current resolution.

In the following example, a *Crop* node is attached, with the *Crop* parameters set manually to 244, 92, 700, and 430. These settings return a 456 x 338 resolution (yup, completely arbitrary). Notice the onscreen control to adjust the resolution manually.



- Using the Transform nodes *Resize*, *Fit*, or *Zoom* to scale the frame

The following three nodes change the resolution by scaling the pixels.

- *Resize*: You set the output resolution of the node, and the image is squeezed into that resolution. This usually causes a change in aspect ratio.



- *Fit*: Like *Resize*, except it pads the horizontal or vertical axis with black to maintain the same aspect ratio.



- *Zoom*: Same as *Resize*, except that you are supplied with scaling factors, so a zoom of 1, 1 is the same resolution; 2, 2, is twice the size; .5, .5 is half the size, and so on.

For more information on the individual transform nodes that can be used to change resolution, and tables listing the differences between the scaling functions, see “Setting Output Resolution and Scaling Images” on page 442.

## Functions Affecting Image Resolution

The functions in this section can be used to modify image resolution.

### Fit

The *Fit* function changes the image resolution, resizing the image to fit inside of a frame. *Fit* does not stretch the image in either axis; it zooms X and Y by the same amount until either value fits in the new resolution (that you specify). For example, if you have a 100 x 200 image, and fit it into a 250 x 250 resolution, it zooms the image by 25 percent ( $250/200 = 1.25$ ), and pads black pixels on the left and right edges.

Parameters	Type	Defaults	Function
<i>xSize/ySize</i>	int	width/height	The new output resolution.
<i>x/yFilter</i>	string	"default"	The filter to be used for resampling the image. "default" means sinc filter when resizing down, and mitchell when increasing resolution.
<i>preCrop</i>	int	1	Turns off the Infinite Workspace so that the letterbox area remains black.

### Synopsis

```
image Fit(  
    image,  
    int xSize,  
    int ySize,  
    const char * xFilter,  
    const char * yFilter,  
    int preCrop  
);
```

### Script

```
image = Fit(  
    image,  
    xSize,  
    ySize,  
    "xFilter",  
    "yFilter",  
    preCrop  
);
```

### Command Line

*shake -fit xSize ySize*

**See Also**

*Resize* (below), “*Crop*” on page 233, “*Viewport*” on page 235, “*Window*” on page 236, and “*Scale*” on page 464

**Resize**

The *Resize* function resizes the image to a given resolution.

Parameters	Type	Defaults	Function
<i>x,ySize</i>	float	width, height	The new X and Y resolution.
<i>filter</i>	string	"default"	See Chapter 16, "Filters."
<i>subPixel</i>	int	0	Turns on quality control. 0 = low quality, subpixel off. 1 = high quality, subpixel on.  If the new width or height is not an integer (either because it was set that way, or because of a proxy scale), you have a choice to snap to the closest integer (subpixel off) or generate transparent pixels for the last row and column (subpixel on).

**Synopsis**

```
image Resize( image,
    float xSize,
    float ySize,
    const char * filter,
    int subPixel
);
```

**Script**

```
image = Resize(
    image,
    xSize,
    ySize,
    "filter",
    subPixel
);
```

**Command Line**

```
shake -resize xSize ySize filter subPixel
```

**See Also**

“Crop” on page 233, “Window” on page 236, “Viewport” on page 235, *Zoom* (below), and “Scale” on page 464

**Zoom**

The Zoom function resizes the image to a given resolution.

Parameters	Type	Defaults	Function
<i>x,yScale</i>	float	1, xScale	The scaling factor, for example, .5 = half resolution, 2 = twice the resolution.
<i>filter</i>	string	"default"	See Chapter 16, "Filters."
<i>subPixel</i>	int	0	Turns on quality control. 0 = low quality 1 = high quality

**Synopsis**

```
image Zoom( image,  
    float xScale,  
    float yScale,  
    const char * filter,  
    int subPixel  
);
```

**Script**

```
image = Zoom(  
    image,  
    xScale,  
    yScale,  
    "filter",  
    subPixel  
);
```

**Command Line**

```
shake -zoom xScale yScale filter subPixel
```

**See Also**

“Crop” on page 233, “Window” on page 236, “Viewport” on page 235, “Resize” on page 212, and “Scale” on page 464

## Re-Mastering to a Different Resolution With Proxies

You can re-master your scene's resolution with proxy images. As an example, you can load NTSC and PAL images simultaneously, with one set as a proxy image. With the proper scaling factors, the scene can be reset to the other resolution by switching the proxy set.

For more information on working with proxies and high-resolution images, see Chapter 3, “Basics of Building a Script.”

## Working With Video

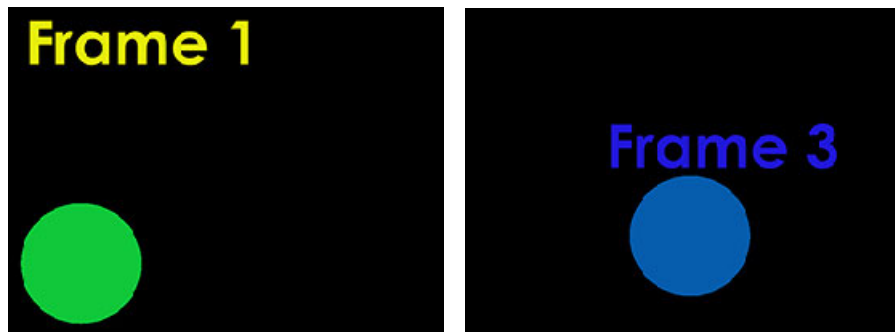
This section explains the use of the Global parameter fieldRendering to generate interlaced images for broadcast, and the deInterlacing parameter for importing interlaced video images. (This section is also a paean to the global implementation of digital television where all nations can sit down and watch field-less TV together in peace.)

The section, “About Aspect Ratio and Non-Square Pixels” on page 226, is also a recommended reference for working with video.

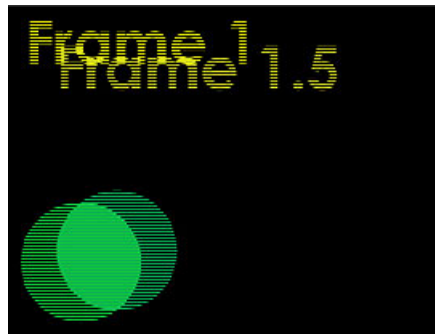
For a note on keying DV footage, see Chapter 7, “Keying.”

## What Are Video Fields?

Dividing frames into fields is a technique used in television broadcasting to reduce the perceived strobing of images by refreshing the display on subframes. By using fields, the viewer is exposed to twice as many frames. The following is an example of an emotionally touching animation showing the frames 1 and 3 (found in *doc/pix/video*). These are considered full frames (an interlaced video frame is divided into two fields)—the entire image is shown at the same moment in time.

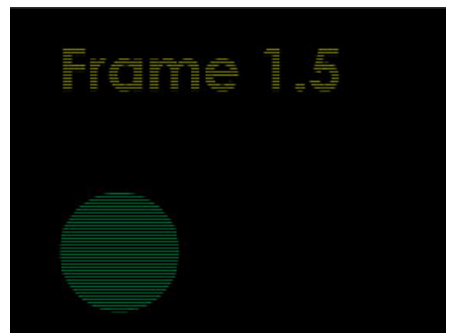


The following shows the clip up to (but not including) frame 3 converted to fields. The image on the left shows some information from frame 1, field 1 and from frame 1, field 2 (unconventionally labeled here as frame 1.5). The image on the right shows frame 2, field 2 (unconventionally labeled frame 2.5 for illustration purposes).



The visual information at a given time is limited to every other line, for example, all even lines. When you advance a half frame of time further, the visual information appears on the other lines, in this case, the odd lines. These are called fields.

The following images show field 1 (frame at time 1) and field 2 (frame at time 1.5) of frame 1. Notice the black lines across the images. When the two fields are put together, as in frame 1 above, the image is interlaced, because the fields are interlaced together.



The following is a close-up of the interlaced image.



Therefore, the interlace process produces two fields of half-height for every broadcast frame. When a television displays these images, it quickly shows the first field only, and then the second field only, and then proceeds to the next frame. This solution is interesting because each field sacrifices vertical resolution quality for the benefit of temporal quality.

### Common Problems With Interlaced Images

The interlace approach of a spatial solution to temporal issues creates two particular types of problems for digital image manipulation.

The first problem occurs when you have any animated parameter. The animation must be understood and applied at half-frame intervals. If you read in an interlaced clip and apply a static *Gamma*, no problems occur because both fields receive the same correction. If, however, you animate the gamma correction, you must turn on field rendering in order to evaluate the correct set of lines at the appropriate interpolated value. See the above examples to illustrate the need to break up color animation into half-frame fields. Field rendering is discussed below.

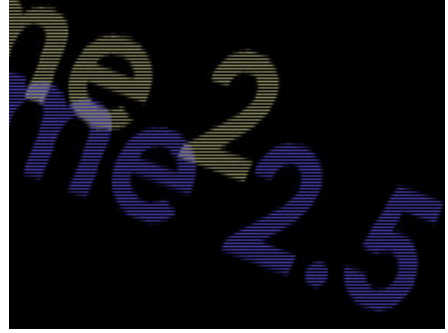
The second and trickier problem is with any node that has spatial effects, such as a *Blur* or a *Move2D*. If you pan an image up by 1 pixel in Y, you have effectively reversed time, because the even lines are moved to the odd field, and the odd lines are moved to the even field. The clip has extremely jerky movement, since every two fields are reversed. This would be the same as if you inverted filmic frames 1, 2, 3, 4, 5, 6 to 2, 1, 4, 3, 6, 5.

Another example of spatial problems arises in image rotation and scaling. The following images show a rotation without field rendering and with field rendering. Only the image on the right appears correct when broadcast.

**No Field Rendering**



**Field Rendering**

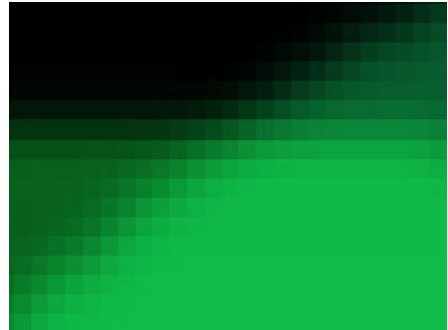


You need to be aware of field rendering with more than just transforms. Below is a close-up of frame 1 (from above), without and with a blur applied to it. Since the blur is applied uniformly to both fields, the result is what is called in the business, “Real Bad.”

**Original Image**

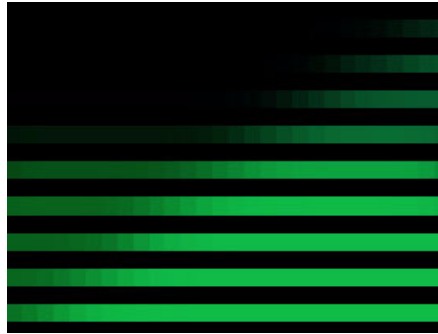


**Blurred Without Field Rendering**

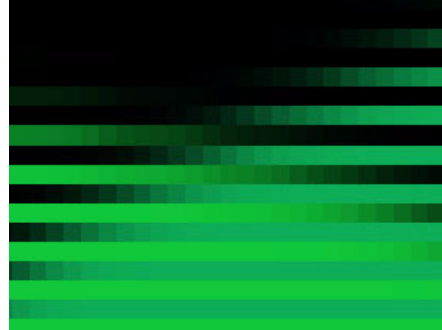


To illustrate this, one field is removed from the image below on the left. Notice that information from both fields intermingles due to the blur, as pixels from a different moment in time bleed into the current field. Turning on field rendering gives you the correct image, shown on the right. No image information bleeds between the two fields due to the blur.

**One Field of Improperly Blurred Image**



**Properly Blurred Image With Field Rendering**



### Field Rendering Settings

To correct the problems discussed above, turn on field rendering in the render controls of the Globals. There are three field rendering settings:

0 = Off

1 = Field rendering with odd field (counting from the top) first. This is generally the setting for PAL images.

2 = Field rendering with even field first. This is generally the setting for NTSC images.

With field rendering turned on, Shake separates the rendering into two separate fields. All animation and spatial effects are allocated to the proper field.

You do not have to use field rendering when you import interlaced images and apply static color corrections. For all other functions, or if you animate any value, turn on field rendering. The field rendering handles all transformations, filters, and warps by internally taking each field, removing the intermediate black lines, and then resizing the Y resolution back up to full frame. The software does this for each field, and then interlaces them back together again.

In the following example, the image is resized from 640 x 480 to 720 x 486. The left image, with inter-field bleeding, is without field rendering. The right image, which is clean, is with field rendering.

**Resize Without Field Rendering**



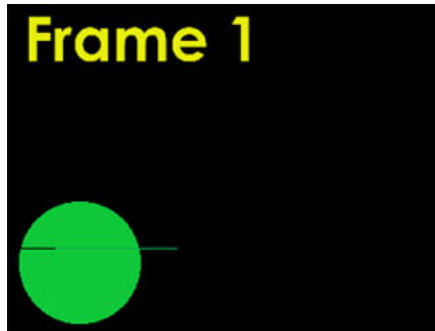
**Resize With Field Rendering**



Viewer zooming is another problem associated with field rendering. Unless you are at a 1:1 ratio for the Viewer (press **Home** to make sure), you likely have artifacts. For example, if you zoom the image down by half in the Viewer, you remove half of the information—an entire field. Therefore, if you see any artifacts in a field-rendered image, ensure you are at a 1:1 ratio for the Viewer.

**Note:** You can also click the Home button in the Viewer to reset the ratio to 1:1.

#### **Viewer Artifact**



#### **JPEGs and Fields**

Using the JPEG output for field rendering is not recommended. The compression bleeds information from one field into the other, which is just not good.

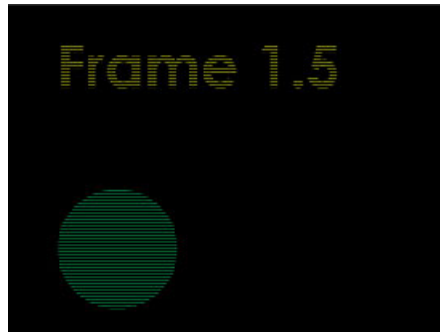
## Importing Interlaced Images

In addition to the `fieldRendering` parameter, you need to be aware of the *FileIn* `deInterlacing` parameter when importing interlaced images. Turn `deInterlacing` on when you import an interlaced image, either to odd (usually PAL) or even (usually NTSC). When enabled, it strips out the two fields from each other, placing field 1 at frame 1, and field 2 at frame 1.5. Each field is then copied and moved into the empty spatial place of the removed field. This ensures that all spatial effects are properly handled by the field rendering. This strategy is interesting because it doubles the amount of frames you have, but keeps the frames within the same duration. Go figure. You must turn on the Global field rendering to return the image to its interlaced status.

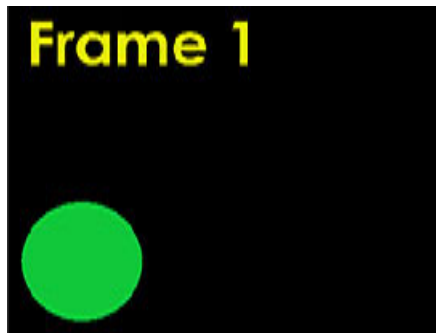
**Field 1**



**Field 2**



**De-interlaced Field 1**



**De-interlaced Field 2**



To see each field, turn off `fieldRendering` and use an increment of .5 in the Time Bar, then step through the animation with the **Left Arrow** and **Right Arrow** keys. Use this technique if you are not sure which field is dominant. If you step through the animation and the image seems to stutter every other field, switch your `deInterlacing` to even, and the motion should be continuous.

## Video Functions

Shake has several other video-oriented functions. Keep in mind that these operate with the assumption that field rendering is off, since they would be affected by the field rendering options like other functions. These functions include the following:

Tab	Function	Notes
Globals	<i>dropFrame</i>	Toggles 30 to 29.97 frames per second when enabled.
Time Bar	<b>T</b> on keyboard	Toggles timecode/frame display.
Image	<i>FileIn</i>	Has de-interlacing, as well as pulldown/pullup capabilities under the Timing subtab. See the " <i>The FileIn/SFileIn Function</i> " on page 79 for more information.
Color	<i>VideoSafe</i>	Limits your colors to video-legal ranges. See " <i>VideoSafe</i> " on page 225.
Layer	<i>Constraint</i>	Limits effects by certain criteria, either zone, change tolerance, channel, or field. Naturally, field is of interest here. You can affect a single field with this node. Generally done with field rendering off. See Chapter 5, "Compositing: The Layer Nodes," for more information.
Layer	<i>Interlace</i>	Interlaces two images, pulling one field from one image, and the second field from the other image. You can select field dominance. Generally done with field rendering off. See " <i>Interlace</i> " on page 222.
Other	<i>DeInterlace</i>	Retains one field from an image and creates the other field from it. You have three choices on how this is done. The height of the image remains the same. Generally done with field rendering off. See " <i>DeInterlace</i> " on page 223.
Other	<i>Field</i>	Strips out one field, turning the image into a half-height image. Generally done with field rendering off. See " <i>Field</i> " on page 224.
Other	<i>Swapfields</i>	Switches the even and odd fields of an image when fieldRendering is off. To do this when fieldRendering is on, just switch from odd to even or even to odd. Generally done with field rendering off. See " <i>SwapFields</i> " on page 224.

## Interlace

Located in the Layer tab, this function interlaces two images. You can control field dominance, whether the input images are themselves separate field images (for example, half Y resolution), or if the fields are extracted every other line.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground (0) and background resolution (1).
<i>field</i>	int	0	Tells what field the first image uses. 0 = Even (PAL) field 1 = Odd (NTSC) field
<i>mode</i>	int	0	Tells Shake if the input image is the same height as the result image. 0 = Replace. Takes every other field from input images; input images have the same height as the result. 1 = Merge. Takes the entire input image; input images are half the result image height.

### Synopsis

```
image Interlace(  
    image Foreground,  
    image Background,  
    int clipMode,  
    int field,  
    int mode  
);
```

### Script

```
image = Interlace(  
    Foreground,  
    Background,  
    clipMode,  
    field,  
    mode  
);
```

**Command Line**

```
shake -interlace image clipMode field mode
```

**DeInterlace**

Located in the Other tab, this function has three different modes, to replace one field of an interlaced image with one of three options:

- Replicate—replaces one field with the other.
- Interpolate—replaces a field with the average of the field above and below it.
- Blur—replaces a field with the average of the average of the field above and below it and the field itself. In other words, the field is replaced with 50 percent of itself, 25 percent of the field above, and 25 percent of the field below.

Parameters	Type	Defaults	Function
<i>field</i>	int	0	The field that is retained. 0 = even field 1 = odd field
<i>mode</i>	int	0	The mode in which the other field is replaced (see above). 0=Replicate 1=Interpolate 2=Blur

**Synopsis**

```
image DeInterlace( image, int field, int mode);
```

**Script**

```
image = DeInterlace( image, field, mode);
```

**Command Line**

```
shake -deinterlace field mode
```

## Field

Located in the Other tab, this function extracts the even or the odd field of the image, returning an image of half the Y resolution.

Parameter	Type	Default	Function
<i>field</i>	int	0	The field that is extracted. 0 = even field 1 = odd field

### Synopsis

```
image Field( image, int field);
```

### Script

```
image = Field( image, field );
```

### Command Line

```
shake -field field
```

## SwapFields

Located in the Other tab, this function switches the even and odd fields of an image.

### Synopsis

```
image SwapField( image );
```

### Script

```
image = SwapField( image );
```

### Command Line

```
shake -swapfield
```

VideoSafe

Located in the Color tab, this function clips “illegal” video values. As such, it is generally placed at the end of a composite. You can set the node for NTSC or PAL video, based on luminance or saturation. There is also an example (in the subtab) of a conditional statement for the videoGamma that toggles between 2.8 (NTSC) and 2.2 (PAL). Generally, these values are not touched.

Parameters	Type	Defaults	Function
<i>videoType</i>	int	0	0 = NTSC 1 = PAL
<i>processingType</i>	int	0	0 = luminance-based calculation 1 = saturation-based calculation
<i>chroma/ compositeRange</i>	float	100, 110	The pseudo-percentage of the clip, as specified by the actual video hardware.
<i>videoGamma</i>	float	videoType? 2.8 : 2.2	The gamma basis, based upon the videoType. NTSC uses 2.2 and PAL uses a gamma value of 2.8.

Synopsis

```
image VideoSafe( image,  
    int videoType,  
    int processingType,  
    float chromaRange,  
    float compositeRange,  
    float videoGamma  
);
```

Script

```
image VideoSafe( image,  
    videoType,  
    processingType,  
    chromaRange,  
    compositeRange,  
    videoGamma  
);
```

Command Line

```
shake -videosafe videoType processingType etc...
```

## About Aspect Ratio and Non-Square Pixels

Shake has several controls in the Global Parameters to help you work with non-square pixel images. These images are typically video images, or anamorphic film images. Different controls are used for the two types, due to the nature of the data that is manipulated.

For video, the primary concern is the distinction of the fields from each other, so the distortion is corrected by extending the image in the X direction and not the sensitive Y direction. For anamorphic film plates, the primary concern is the amount of data calculated, so the image is squeezed in Y, reducing the overall frame size for better interactivity. This not only corrects the distortion, but also speeds interactive renders by a factor of two.

You can find sample images in *doc/pix/aspect* to test the following discussion.

When you correct non-square pixel images, you need to know the aspect ratio of the image for transformations and corrections to the viewing process, so you can see the image without distortion. For the discussion of different aspect ratios, film anamorphic plates are used to illustrate how to work with such frames. Since the squeeze is the most drastic, the anamorphic images serve as the best example. Although the solution is specifically for film plates, and you should use a slightly different solution for video images, the principles and problems are similar.

The anamorphic process creates film frames that, when projected, offer extremely wide images. This is done by first filming the scene with a special lens that squeezes the incoming image by two only in the X direction so the scene fits on the physical piece of film.

Everything is very thin on the physical negative. When the film is projected in the theater, a reverse lens is applied that expands the image by two on the X axis, and returns the image to its very wide format. It is therefore important to realize that the widescreen data only exists in front of the lens when filming and on the projection screen. Usually at all points in between, you are ideally working with the squeezed image. This is also a fundamental principle when compositing squeezed elements—the ideal image is never scaled, but you still need to see the results as unsqueezed. Shake has a toolset to help you meet this criteria.

If you load the image *doc/pix/aspect/anamorphic\_2\_1.iff*, you see a low-resolution example of an anamorphic frame. The image resolution is 914 x 778, or half of a standard 1828 x 1556 anamorphic plate. You can see from the shape of the circle that the image is squeezed in X.



### Properly Viewing Squeezed Images

There are two general ways to view the image in its unsqueezed proportions. You can either change the resolution of the image with a *Zoom* or *Resize*, or adjust the Viewer aspect ratio or script proxyRatio. The first choice requires you to zoom the image down by 2 in Y (or up by 2 in X), do the compositing, and then zoom it back to its squeezed proportions. This requires you to scale the image down and then up, which has an inherent loss of quality. Therefore, the second choice to modify a Viewer aspect ratio or a script proxyRatio, is more appealing. The script proxyRatio is better for film elements, and the Viewer aspect ratio is better for video elements.

- **Video:** To change a viewer aspect ratio, expand the format subtree in the Globals and set the default viewerAspectRatio to 2 (since our ratio is 2:1 for this film plate). For video, use the appropriate ratio, listed in the Table of Common Aspect Ratios, below. This renders the image at full resolution, and then doubles the Viewer width. This does not affect your output resolution, and takes exactly the same amount of time to render files to disk. There is a speed hit in the interactivity to adjust the viewed frame. This is the parameter you should use with video elements, since you change the X dimension and leave the Y dimension, which contains the fields, at the same resolution.
- **Film:** With film elements, you should set the proxyRatio (underneath the useProxy subtree in the Globals tab) to .5 (one-half). Use of the proxy system reduces your render time for interactive tests by half. Unlike viewerAspectRatio, this halves the Y value, rather than doubles the X value. This, however, affects your output files, so be sure to set the proxyRatio back to 1 when you render your images to disk. Remember, you want to send squeezed files to the film recorder.

So, for this anamorphic film plate, open the useProxy subtree and enter a proxyRatio of .5 to correct the squeezed element.



### Function Aspect Ratio and the defaultAspect Parameter

Certain functions need to be aware of aspect ratio, specifically functions dealing with circles or rotation. For example, if you apply a *Rotate* node to an anamorphic plate, the image is distorted.



The *Rotate* node has an aspectRatio parameter. Set the parameter to .5, and the rotation is no longer distorted.



The *RGrad* node is backward from the other nodes (go figure). The `aspectRatio` for this should be `1/defaultAspect` (what it uses as its creation rule). Here, an *RGrad* with an `aspectRatio` of 1 is composited on the image.



Since it is distorted, change the `aspectRatio` of the *RGrad* to 2 and the world is beautiful.



#### **Automatic Setting of the `aspectRatio`**

A helpful parameter is the Global `defaultAspect` in the format subtree. The `aspectRatio` parameter in *Rotate* and other nodes sets this automatically when the node is created. Changing the `defaultAspect` does not change preexisting nodes—it only affects nodes that are created after you set the `defaultAspect`.

#### **Compositing Square Pixel Images on Squeezed Images**

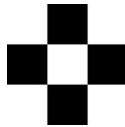
In this example, *square.iff* in the *doc/pix/aspect* directory represents a square pixel image, which is typical of 3D rendered images.



When composited over the image, there is distortion because of the proxyRatio.



There are two options to correct this. You can scale the X by half, or increase the Y by two. The first option ensures the highest quality, but means you have to render the original CG element at twice the resolution of the second option. In this example, the image is scaled in Y by 2 with either a *Transform–Scale* or *Zoom* node:

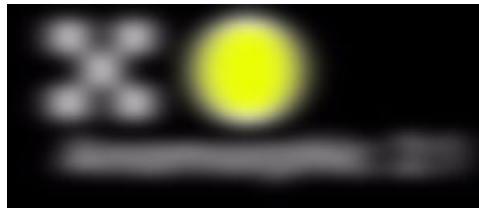


### 3D Software Renders

If your software allows, render your scene with the proper aspect ratio. This ensures the highest quality in your composite.

### Tuning Parameters in Squeezed Space

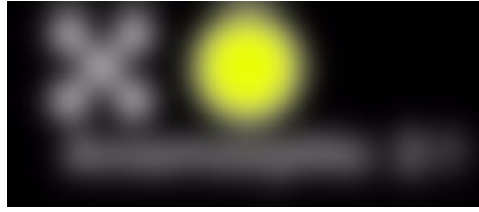
In addition to *Rotate* and *RGrad*, other nodes should be tuned with a squeezed aspect ratio. In the following example, a *Blur* node is applied to the image.



Because the default yPixel value is set to the xPixel value, you get twice the blur on the X parameter in the squeezed space. To correct this, double the Y value (or halve the X value) with an expression in the yPixels parameter:

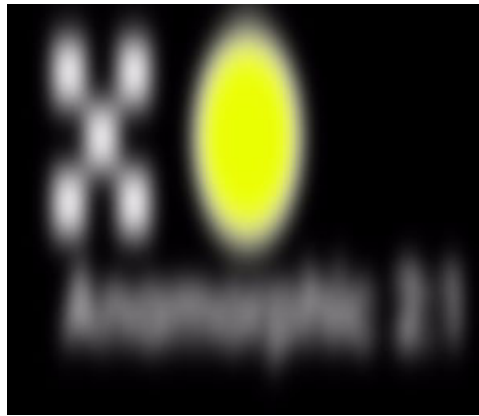
*xPixels\*2*

The blur now looks proportionately correct.



### Rendering Squeezed Images

Once your composite is complete, reset the `proxyRatio` to 1, and render. Do not change any other parameters. The resulting image appears squeezed on the X axis, but this distortion is corrected during the film projection process in the theater.



Although you worked on the image in a squeezed state, all elements are properly positioned when the `proxyRatio` is returned to 1. Use of the proxy system temporarily squeezes the image down and back up, thereby maintaining the pixel data.

### Handling Video Elements

Video elements also have a built-in squeeze. To make things extra tasty, each video format has its own aspect ratio. Using the `proxyRatio` parameter is not recommended for video elements because `proxyRatio` squeezes the image in Y. This almost certainly obscures your field separation. Instead, use the `viewerAspectRatio` parameter set to the aspect ratio of the video element. This stretches the Viewer in X, leaving the Y untouched. This also only affects the Viewer—you have exactly the same processing time (when going to disk), and your rendered images are not affected. However, in-Viewer playback suffers slightly.

You only need to change the defaultAspect for proper rendering. Unlike using proxyRatio, you set defaultAspect to 1/YourVideoAspectRatio. For example, PAL HD uses 1.422 as its aspect ratio. Therefore, set viewerAspectRatio to 1.422, and defaultAspect to 1/1.422. Shake resolves the expression of 1 divided by 1.422 to become 0.703235. All other principles of image manipulation apply.

**Preset Formats**

The format list in the Globals tab has a list of preset formats. You can also create your own formats.

**Table of Common Aspect Ratios**

This is a table of common aspect ratios (feel free to submit more), and the values you should place in specific parameters. For nodes, the aspectRatio parameter is taken from the defaultAspect value at the time the node is created. It is not changed if you later change defaultAspect. Additionally, *RGrad* is the inverse of the other aspectRatio parameters, for example, 1/defaultAspect, which accounts for its own column in the table. Initially setting the defaultAspect guarantees that all nodes automatically get the proper aspect ratio. Finally, follow the guidelines in Chapter 17, “Customizing Shake,” and save your Interface Settings to set default values for these parameters. For more information, see “General Settings” on page 626.

Format	aspect Ratio	proxy Ratio	Viewer Aspect Ratio	defaultAspect	aspect Ratio (common nodes)	aspect Ratio (RGrad)
2:1 Anamorphic Film	2	.5	NA	.5	.5	2
4:3 NTSC D1 720 x 486, 720 x 480	.9	NA	.9	1/.9 = 1.1111	1.11111	.9
16:9 NTSC 720 x 486	1.2	NA	1.2	1/1.2 = .83333	.8333	1.2
4:3 PAL D1 720 x 576	1.066	NA	1.06	1/1.066 = 0.938086	0.938086	1.066
16:9 PAL 720 x 576	1.422	NA	1.422	1/1.422	.703235	1.422

## Cropping Functions

This section describes several functions you can use to crop your images. *Window*, *Viewport*, and *Crop* are located in the Transform tab, and *AddBorders* is located in the Other tab.

### AddBorders

The *AddBorders* function is similar to a *Crop*, except it adds an equal amount of space to the left and right sides, or to the top and bottom sides. It is located in the Other tab because it is kind of a goof function.

Parameters	Type	Defaults	Function
<i>xBorder</i>	int	0	The amount of added pixels to the left and right border.
<i>yBorder</i>	int	0	The amount of added pixels to the bottom and top border.

### Synopsis

```
image AddBorders( image, int xBorder, int yBorder);
```

### Script

```
image = AddBorders( image, xBorder, yBorder);
```

### Command Line

```
shake -addborders xBorder yBorder
```

### Examples

```
shake lisa.iff -addborders 50 50
```

```
shake lisa.iff -addbor 50 50 -savescript example.sbk
```

### Crop

This function crops an image by defining the lower-left corner and the upper-right corner of the crop. Since the numbers can be greater or less than the boundaries of the image, you can make the image smaller, or expand the image with a black border. A *Crop* cuts elements beyond the frame area, so if you later use another transform (for example, *Pan*, *Move2D*, etc.) to move the image, black is brought in. This is how *Crop* differs from *Viewport*—*Viewport* does not cut off the image. See “*Viewport*” on page 235, for an example. *Window* is also the same command, but you set the lower-left corner, and then the X and Y resolution.

*Crop* is particularly helpful in that it sets a new frame area. Shake only processes nodes within the frame, so to speed up an operator you can limit its area with a *Crop*. This is essentially what is done with the *Constraint* node.

Parameters	Type	Defaults	Function
<i>cropLeft, Bottom, Right, Top</i>	float	0, 0, width, height	These represent the lower-left and upper-right corners of the crop region. Note that the area generated is up to, but not including, the upper right since 0,0 represents the lower-left corner of the lower-left pixel.

**Synopsis**

```
image Crop( image,
    int cropLeft,
    int cropBottom,
    int cropRight,
    int cropTop
);
```

**Script**

```
image = Crop( image,
    cropLeft, cropBottom
    cropRight, cropTop
);
```

**Command Line**

```
shake -crop cropLeft cropBottom etc...
```

**Example**

```
shake lisa.iff -crop 100 200 400 500
shake lisa.iff -crop -100 -100 width+100 height+100
```

Viewport

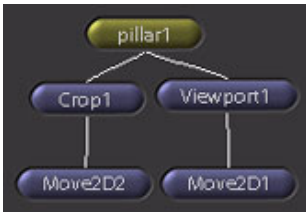
The *Viewport* function is exactly like a *Crop*, but it keeps the image information outside of the frame so you can perform later transformations.

Viewport Function Example

The following tree has a large input image (scaled down in the illustration) which is piped into a *Crop* node and a *Viewport* node, each with the same values. Both nodes are then piped into *Move2D* nodes with the same xPan values. The *Crop* result has black on the right edge after the pan; the *Viewport* result does not.

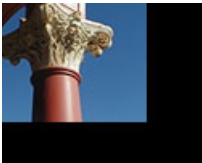
Node Tree

FileIn1



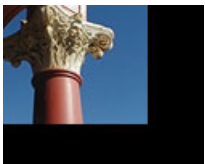
Crop1

Viewport1



Move2D1

Move2D2



Parameters	Type	Defaults	Function
<i>cropLeft</i> , <i>Bottom</i> , <i>Right</i> , <i>Top</i>	int	0, 0, width, height	The cropping window, with 0, 0 being the lower-left corner.

### Synopsis

```
image Viewport( image,
    int cropLeft,
    int cropBottom,
    int cropRight,
    int cropTop
);
```

### Script

```
image = Viewport( image,
    cropLeft,
    cropBottom,
    cropRight,
    cropTop
);
```

### Command Line

*shake -viewport cropLeft cropBottom cropRight cropTop*

### Window

The *Window* function is exactly like a *Crop*, but you enter the lower-left corner, and then the X and Y resolution.

Parameters	Type	Defaults	Function
<i>cropLeft, Bottom</i>	int	0, 0	The lower-left corner of the image.
<i>xRes, yRes</i>	int	width, height	The new output resolution of the image.

### Synopsis

```
image Window( image,
    int cropLeft,
    int cropBottom,
    int xRes,
    int yRes
);
```

### Script

```
image = Window( image, cropLeft, cropBottom, xRes, yRes);
```

### Command Line

*shake -window cropLeft cropBottom xRes yRes*

# Keying

## Chapter Summary

- About Keying and Spill Suppression
- Edge Treatment
- Keying DV Footage
- Keying Functions

## About Keying and Spill Suppression

This section discusses different strategies for pulling keys in Shake by combining nodes in different ways.

**Note:** If you are not familiar with *Primatte* and *Keylight* (Shake's primary bundled keyers), you are encouraged to read the keying tutorials prior to reading this section.

Keying can be loosely defined as the creation of a new alpha channel based upon the pixel color (a pure blue pixel, for example), luminance (a very dark pixel), or Z depth (a pixel 300 Z units away, for example) in an image. Keying is discussed as a separate process from masking, which can be loosely defined as the creation of an alpha channel by hand through the use of painting, roto splines, or imported alpha masks from 2D or 3D renders in other software packages. For more information on these masking techniques, see Chapter 10, "Using Masks."

## Pulling a Blue Screen or Green Screen

In the Key Tool Tab, the two primary nodes used to pull blue screen and green screen keys are *Primatte* and *Keylight*. In the Tutorials book, there are lessons devoted to *Primatte* and *Keylight*. Other functions in the Key tab include *ChromaKey*, *DepthKey*, *DepthSlice*, *LumaKey*, and *SpillSuppress*. These nodes are discussed later in this chapter. The *ColorReplace* node, although located in the Color tab, is also considered another key-pulling node.

Although there is a *ChromaKey* node in the Key tab, it is not particularly useful. No kidding. The same model and parameters appear in *ColorReplace*, but *ColorReplace* generally works much better.

**Note:** You should also investigate *Ultimate*, a third-party keyer with a long and Oscar-winning history of production use. Contact Ultimate for more information. It rocks in a completely law-abiding, non-violent, politically-correct fashion.

### Keying With Primatte

For a description of how to use *Primatte* to pull a key, see the *Primatte* tutorial in the *Shake 3 Tutorials* book.

### Keying With Keylight

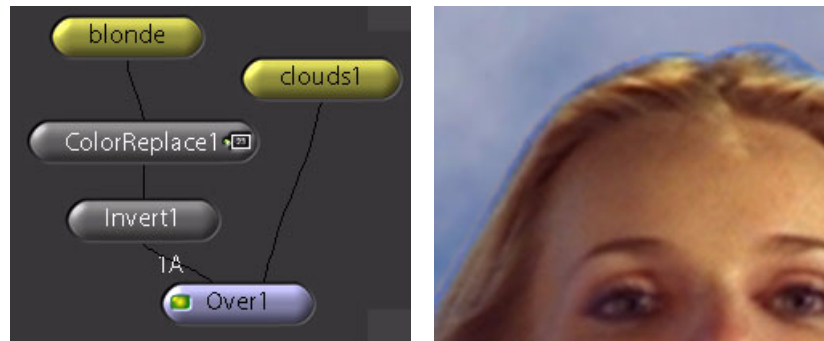
For a description of how to use *Keylight* to pull a key, see the *Keylight* tutorial in the *Shake 3 Tutorials* book.

### Keying With ColorReplace

*ColorReplace* is not the best keying tool, but it is a good node for making quick garbage masks. The following example uses *ColorReplace* to key the *blonde* image over the *clouds1* image.

#### To set up the key example:

- 1 Click the Image tab and select *FileIn*.
- 2 Go to *doc/pix/primatte* and load the *alex\_fg.jpg* and *clouds1.jpg* images. (The images are courtesy of Photron, which is fine, since it shows you how much better *Primatte* is than *ColorReplace*.)
- 3 Create the following tree:



- 4 Set the following parameters:
  - *ColorReplace* node—Click the Source Color box and then scrub on the blue screen in the blonde image. Set Replace Color to any other color (it just cannot be the same blue). Also, enable *affectAlpha* so that you pull a key.

- *Invert* node—Set the channels to “a” instead of rgba.
- *Over* node—Enable preMultiply.

Because *ColorReplace* puts white in the Source Color area of the alpha channel, use the *Invert* node to invert the image for the *Over* node.

- The initial settings yield blue fringes. In the *ColorReplace* parameters, set satFalloff to 1 to correct this. Also, if pure black or pure white pixels start to show transparent, set the valRange and valFalloff numbers to approximately .2 and .5.



You can see there is some crunchiness, particularly in the hair. This demonstrates why *ColorReplace* is usually used to pull a key to mask other operations such as color correctors, rather than the actual composite. There are examples of using *ColorReplace* in the “Blue and Green Spill Suppression” section below.

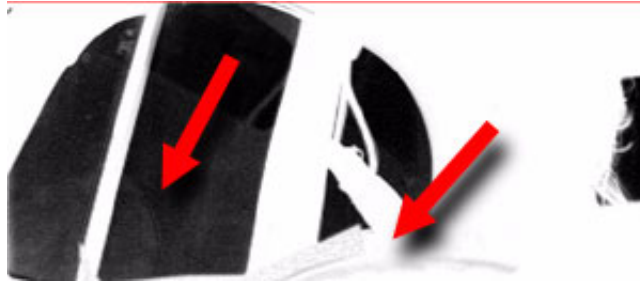
### Combining Keyers

You get the most flexibility when you combine keyers. Both *Primatte* and *Keylight* allow you to input a holdout matte. There are at least three typical ways of combining keys:

- Use the *Primatte* or *Keylight* holdout matte input.
- Use the *Primatte* arithmetic operator.
- Use *Max*, *IAdd*, *IMult*, *Over*, *Inside*, or *Outside*.

You can combine keys with the holdout matte input. Typically, you pull a basic key for soft edges and reflections. You also pull a second key which is very hard, and then soften it with filters.

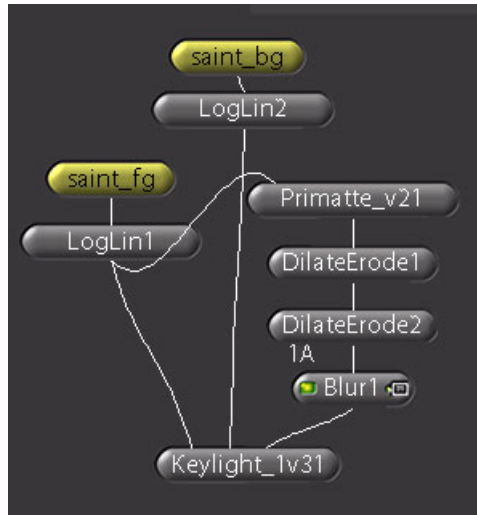
The following is an example using the *Keylight* sample images in *doc/pix/keylight*. The first image shows the initial key coming out of *Keylight*. The reflections are good, but there is some transparency near the seat belt and steering wheel.



Next, a key is pulled with *Primatte* on the same blue screen. With just foreground and background operators, the reflection is removed and the transparent areas filled in. This results in a very hard matte.



Then, filters are attached to the *Primatte* node. A *DilateErode* is added, and the xPixels set to 1 (this closes up any holes). You can also use a Median filter to do the same thing. A second *DilateErode* is applied, with the xPixels set to -5. This eats away at the matte. This filtering process cleans up the matte, making it more solid. A *Blur* softens it, and is fed into the third input of *Keylight*. The result is a solid matte with soft edges.



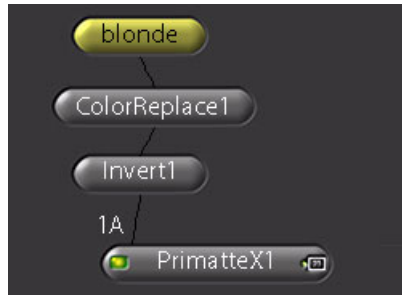
### **An Alternative for Making Hard Mattes**

Copy your *Keylight* node and boost the screenRange to .3 to harden the matte.

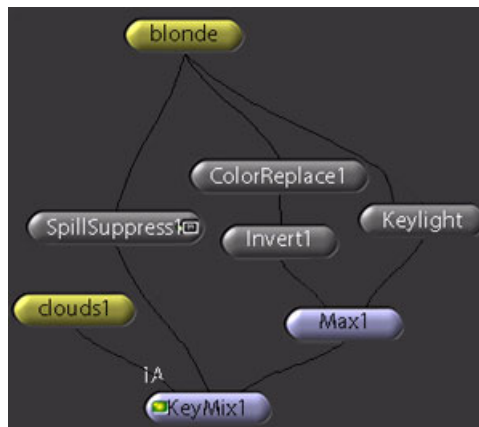
### **A Matte Touch-Up Tool**

In the Cookbook under “Key Macros” is the *KeyChew* macro. This is also a good tool for duplicating the *DilateErode* chain in the above example.

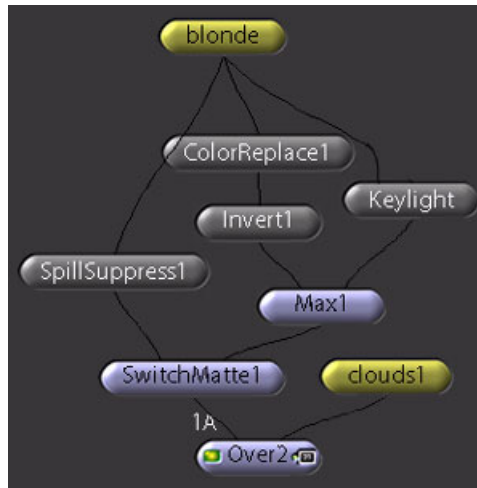
The next way to combine keys is specifically for *Primatte* using its arithmetic parameter. Normally, when you pull a key in *Primatte*, the alpha mask is replaced in the foreground image. When the arithmetic parameter is switched from replace to add, multiply or subtract, you can combine the mattes within *Primatte*. In the following tree, an initial key is pulled with *ColorReplace*, inverted, and then the arithmetic is switched to add in the *Primatte* node to combine them.



The last way to combine keys is to use layer nodes. In the following tree, a key is pulled with two different nodes and then combined with a *Max* node, which takes the greatest pixel value. The elements are combined with a *KeyMix*. Note that the *KeyMix* does not get an alpha channel, since neither *clouds1* nor *blonde* have an alpha channel. *KeyMix* only mixes the two images through the mask you have pulled.



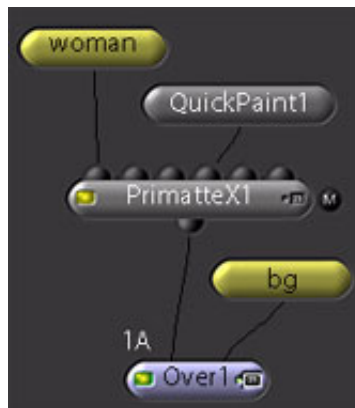
The following example uses an *Over* node instead of a *KeyMix* node. This generates an alpha channel out of the tree.



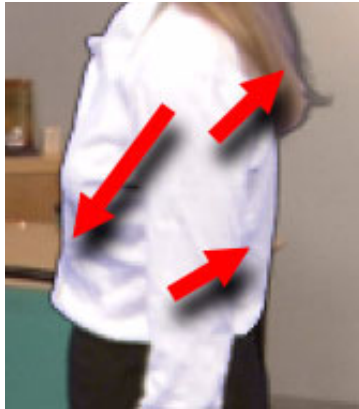
### Blue and Green Spill Suppression

Once the composite is pulled from the keyers and put back into the *KeyMix* or *Over* node, you can start to work with spill suppression. Blue spill occurs when light is reflected off of the blue screen and onto the foreground material.

The following examples use *doc/pix/primatte/woman.iff* and *bg.jpg*. Notice that there is quite a bit of blue spill on her shirt. In the following tree, the *Primatte* output is set to alpha only, a holdout matte is created for the line under her arm (with a *QuickPaint* node), and foreground and background scrubs are added. The composite is done with the *Over* node that has preMultiply enabled. Notice the blue spill that remains:



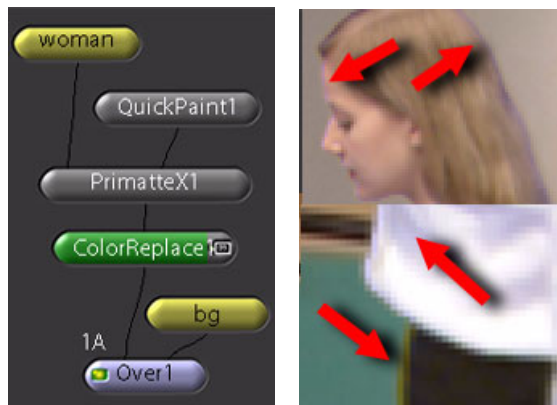
You may think that it would look better if you switch the *Primatte* output to comp and turn off the preMultiply in *Over*. (You'd be wrong.) This turns your blue edges into black edges that are more difficult to remove. (Techniques to help correct this are discussed below.) And, the blue spill is still apparent.



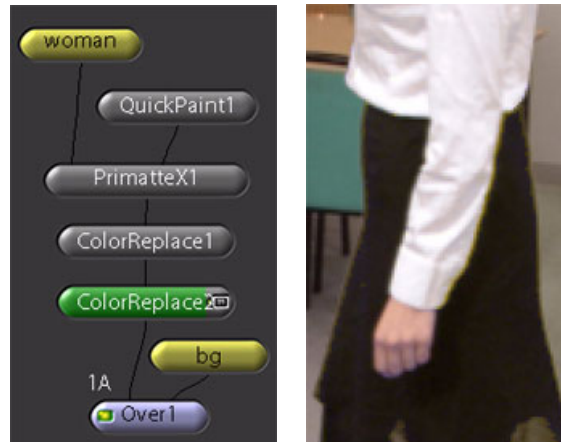
To avoid blue spill correction in your keyers, the following examples contain several sample trees to help you.

### Color Replace I

This technique is nice because it is fast, but often simply replaces blue spill with a different color. In the following tree, a *ColorReplace* is applied to the foreground, and replaces the blue with the color of the wall. As always, increase the satFalloff to around 1. The head looks great, but spill remains on the shirt, and there is now a yellow edge around the skirt.

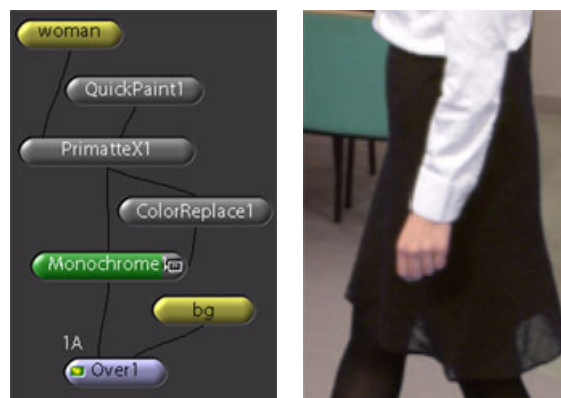


To correct this, drop the *valRange* to 0 and the *valFalloff* to approximately .4. Add a second *ColorReplace* to eliminate the blue spill on her shirt. Carefully select the blue on the shirt, and then replace it with a beige-white color. Also, move all Range parameters to 0 and all Falloff parameters to approximately .3. The shirt looks good, but there is still a yellow ring around her skirt.



### Color Replace II

A better technique is to use *ColorReplace* to mask a color correction. Replace *ColorReplace2* with a *Monochrome* node and pipe *Primatte* directly into it. Then, attach the output of *ColorReplace1* as the *Monochrome1* mask, and turn on the *affectAlpha* parameter in *ColorReplace*. This process turns off all saturation in the blue areas and turns it grey. This is good since the eye can detect luminance levels better than saturation. The skirt and the shirt look good.



As an alternative to *Monochrome*, you can use *AdjustHSV* or *Saturation*.

## SpillSuppress

The *SpillSuppress* node mathematically evaluates each pixel and compares the blue and green strength. If the blue is significantly stronger than the green, the blue is suppressed. Because of the luminance difference between blue and green, the *SpillSuppress* node tends to work better on blue spill than green spill. It also tends to push your images to a yellow color.



## HueCurves

The *HueCurves* node, located in the Color tab, enables you to boost colors or saturation based on the hue of the pixel you want to affect. *HueCurves* works by loading a parameter into the Curve Editor and tuning it—ignore the text fields in the node parameters. The X axis is the hue, and the Y axis depends on the parameter you are using. For the following example, load the saturation curve into the Curve Editor and grab the control point around .66, since blue has a hue of 66 percent. Drag the point down to decrease the saturation for the blue pixels.



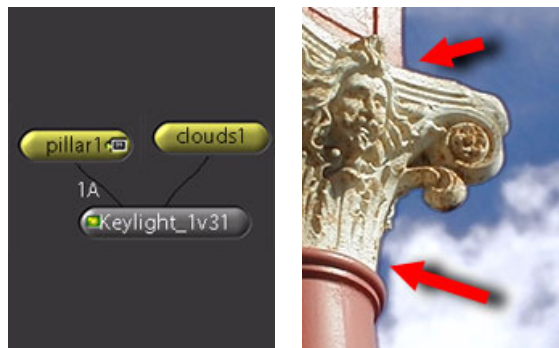
## Edge Treatment

Another typical problem in keying (OK, it is *the* problem with keying) is edge treatment.

The following example uses two images from the *doc/pix/primatte* directory:

- 1 In the Image tab, click *FileIn*.
- 2 Go to *doc/pix/primatte* and read in the *pillar1.jpg* and *clouds1.jpg* images.  
These images bring up an important tip for compositors: If your supervisors say it is fine to use the sky as a blue screen, ignore them.
- 3 In the Node View, select the *pillar1* node.
- 4 Click the Key tab and click the *Keylight* node.  
The *pillar1* node is connected to the *Keylight* node's foreground input.
- 5 Connect the *clouds1* node to the *Keylight* node's background input.
- 6 In the *Keylight* node parameters, click the screenColour box, and scrub the sky near the horizon.

A black line appears around the pillar.

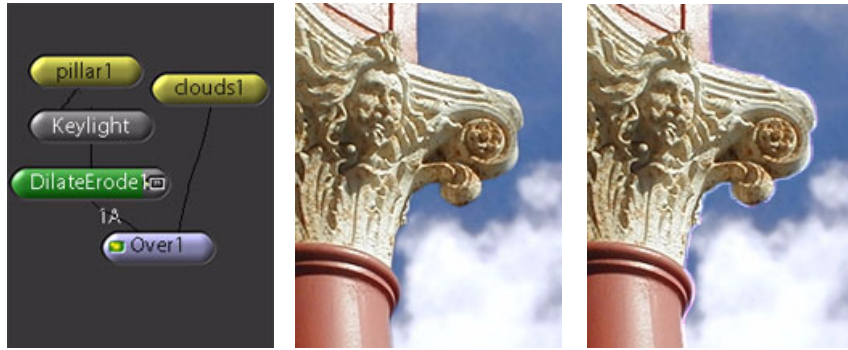


- 7 As mentioned earlier, it is better to composite after pulling your key because it gives you more flexibility. So, rewire the tree with an *Over* node. Make sure you turn on preMultiply in the *Over* node parameters and set the *Keylight* output to unPremult.



- 8 Next, attach a *DilateErode* node to the *Keylight* node. In the *DilateErode* channels parameter, enter “a” (replace rgba) to only affect the alpha. Set xPixels to -1, and it chews into the matte.

The next image illustrates a disabled preMultiply parameter in the *Over* node (because the mask–RGB premultiplied relationship is upset). White lines appear around the edges.



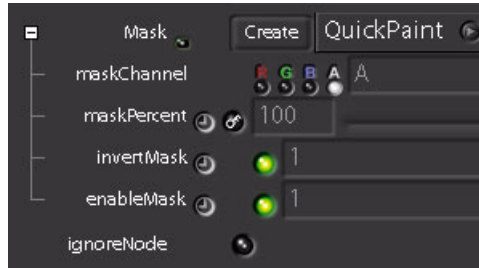
There is one little problem—in the lower-right corner of the image, the electrical power lines have disappeared. Because the details are so fine, the *DilateErode* chewed the lines away.



- 9 To correct this, go to the Mask input of the *DilateErode* node and select *QuickPaint* from the Mask pop-up menu.

A mask is attached to the *DilateErode*.

- 10** Paint across the electrical wires. This only dilates where the wires exist—the opposite of what you want—so open the Mask subparameters in the *DilateErode* and enable *invertMask*.



The edges are dilated except around the wires area.

For more information on the *QuickPaint* node, see “Using the QuickPaint Node” on page 527.

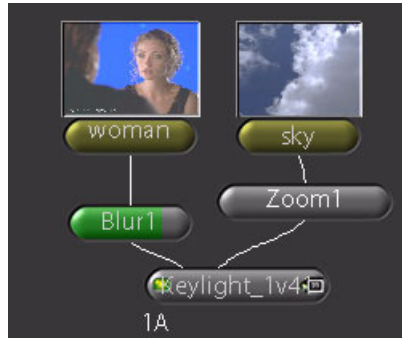
### Applying Effects to Blue Screen Footage

Problems can occur when you apply effects to blue screen footage. For example, suppose you want to blur your foreground but not the background. Your natural instinct is to apply a *Blur* node and then key and composite with *Keylight*. This is what is called “A Real Bad Idea.”

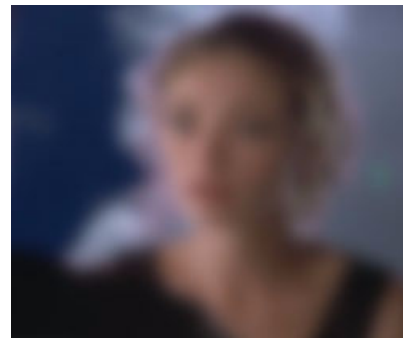
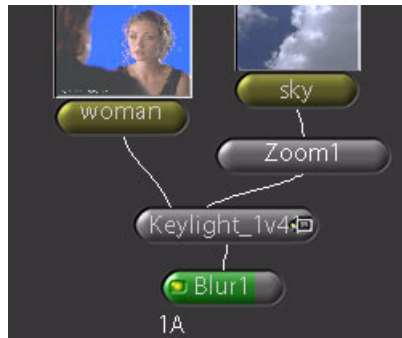
The following node tree uses the *doc/pix/woman.iff* and *doc/pix/sky.jpg* images. A hearty “thank you” to Skippingstone for the use of images from their short film *Doppelganger*.



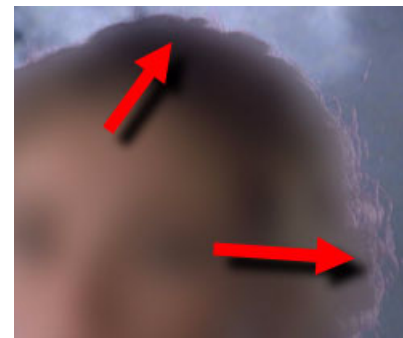
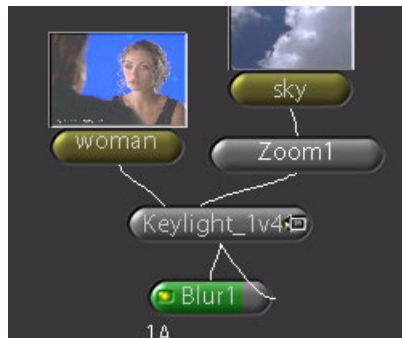
When the blur is applied, a blue edge is introduced along her neck line.



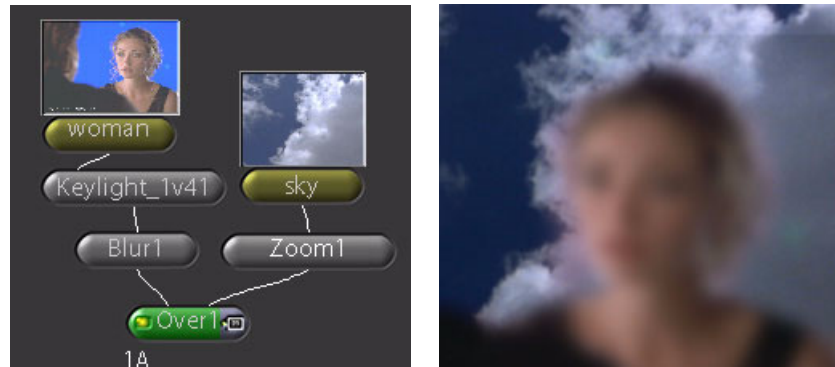
So, as bright kids, maybe you can place the *Blur* after the *Keylight* node in the tree. Also a bad idea—the background is blurred as well.



Perhaps you can mask the blur with the key from *Keylight*. (Hmm. Another bad idea.)



The problem with the above examples is that the keying node, in this case *Keylight*, is asked to do too much—it must simultaneously key, suppress spill, and composite. The solution is to use other nodes for the compositing. In the following example, the *Keylight* output parameter can be set to comp or on Black.



As a general principle, use the *Primatte* and *Keylight* background inputs as test composites, to later rewire into a composite with *Over* or *KeyMix*. This gives you several advantages:

- You can apply effects to the foreground material.
- You can transform the foreground material.
- You can color correct the foreground material.

## Keying DV Footage

First of all, what is wrong with you, trying to key off of DV footage? Now with the obligatory yelling out of the way, let's see what can be done to salvage this dismal situation.

Here, you have shot your no doubt perfect blue screen with your shiny brand new DV camera. There is a small portion of an image in *doc/pix/primatte* called *du.iff* if you want to sing along. Key the image and place it over a background. (A red field is used in this example.) You are no doubt horrified to find blocky chunks spewed all over your image. (That's not lunch, that's 4:1:1 signal!)

**DV Blue Screen**



**Key Pulled on DV Blue Screen**



What is up with this? In RGB color space, each pixel is represented with a value of Red, Green, and Blue. This is how video sees the world—in YUV (or YCrCb) color space. Each pixel is represented with a luminance value (Y) and two chrominance values: red/green difference (Cr) and blue/green difference (Cb). This is done because green has a higher luminance value, and is therefore more likely to display artifacts if too much information is taken away. In the video world, a signal described as 4:4:4 means for every four pixels in a row, you get four pixels each for Y, Cr, Cb. This is equivalent to 8-bit RGB. 4:2:2 means that you get four Y pixels and two each of Cr/Cb for every four pixels in a row. You get less information in a smaller package, giving you a little more speed and a usually acceptable loss of detail. 4:1:1, what DV uses, means you get four Y (luminance) pixels and a single each of Cr/Cb (red/green and blue/green difference). This means luminance can change every pixel, but color changes only every four pixels.

For example, if you look at the keyed DV footage, you see that each of those blocks is four pixels wide. A lot of data is saved. However, blue screen keyers pull keys from color, not luminance; thus, you get the chunkiness. Yuck!

Since the information in video is transferred from the YUV signal to the digital RGB, examine the original YUV channels. Attach a *Color–ColorSpace* node to the *FileIn* and declare the outSpace as yuv. Press R/G/B on the Viewer to look at the new YUV channels.

**Y (luminance)**

**U (Cr, or r/g difference)**

**V (Cb, or b/g difference)**

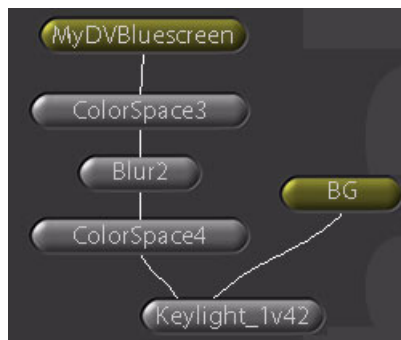


The luminance channel gets all of the necessary information, since the human eye is more susceptible to differences in luminance than hue. But the two color channels unfortunately look like “Space Invaders.”

**To correct the key:**

- 1 Attach a *Filter–Blur* node to the *ColorSpace* node (from the above paragraph).
- 2 In the *Blur* node’s channels parameter, blur only the U and V channels by switching the channels from rgba to gb.
- 3 Blur it slightly, approximately 8 pixels, and mostly on the x axis. The y axis should be minimal, approximately 1 pixel.
- 4 Attach a second *ColorSpace* node and declare the inSpace as yuv.

This converts it back to RGB space.



The key is greatly improved. Golly, it's mathematical!

**Without Blur on UV**



**With Blur on UV**



Another way to enhance your DV keying is to capture the blue/green screen DV footage through an analog capture board, or a board that can convert back up to 4:2:2. This significantly enhances the quality of the image and reduces the “blockiness” associated with 4:1:1 footage.

When you use straight YUV files, you can bypass the RGB to YUV conversion by turning off *yuvDecode* in the *FileIn*. Apply the *Blur*, and then use one *ColorSpace* to convert it to RGB space.

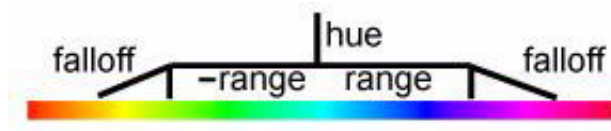
You may have better results with green screen than blue screen because of the higher luminance value.

## Keying Functions

The following section includes the keying functions located in the Key Tool Tab. The *ColorReplace* function, discussed above, is located in the Color Tool Tab. For more information, see Chapter 8, “Color Correction: Part I.”

### ChromaKey

The *ChromaKey* function examines the HSV values of an image and pulls a matte based upon the parameters. In the interface, you can scrub a color in the Viewer. However, disable the *matteMult* parameter before you scrub. The hue, saturation, and value of an image each has a set of parameters to describe the exact HSV values you are keying, as a range from that midpoint, a falloff value, and a sharpness value to describe the falloff curve. This is illustrated in the following image.



The *ChromaKey* function is not Shake’s strongest feature. It is recommended to use *Color–ColorReplace*; select your blue screen color; choose your destination color (any other color); and then enable *affectAlpha*.

A typical application is blue screen or green screen removal. You can stack multiple *ChromaKey* nodes to extract different ranges. With the *arithmetic* parameter, you can choose to add to, subtract from, or replace the current mask.

Parameters	Type	Defaults	Function
<i>hue, sat, val</i>	float	.666, 1, 1	Picks the center value to be pulled on hue, saturation, and value.
<i>Range</i>	float	.2, .4, .8	Plus and minus from the initial hue, sat, or value that is picked.
<i>Falloff</i>	float	.1, .8, .1	Describes the falloff range from hueRange that is picked, with the values ramping down.
<i>Sharpness</i>	float	.5, .5, .5	Describes the falloff curve from hueRange to hueFalloff. 0 = linear drop-off 1 = smooth drop-off
<i>matteMult</i>	int	1	Toggle to premultiply the RGB channels by the pulled mask. 0 = no premultiply 1 = premultiply
<i>arithmetic</i>	int	0	0 = replace existing mask 1 = add to existing mask 2 = subtract from existing mask

### Synopsis

```
image ChromaKey(
    image,
    float hue,
    float hueRange,
    float hueFalloff,
    float hueSharpness,
    float sat,
    float satRange,
    float satFalloff,
    float satSharpness,
```

```

float val,
float valRange,
float valFalloff,
float valSharpness,
int matteMult,
int arithmetic
);

```

### Script

```

image = ChromaKey(
    image,
    hue, hueRange, hueFalloff, hueSharpness,
    sat, satRange, satFalloff, satSharpness,
    val, valRange, valFalloff, valSharpness,
    matteMult,
    arithmetic
);

```

### Command Line

*-chromakey hue hueRange etc...*

### Example

*shake parrot.jpg -chromakey .66 .2 .1 .5 1 .4 .8 .5 1 .8 .1 .5*

### DepthKey

The *DepthKey* function creates a key in the alpha channel based on depth (Z) values. Values below *loVal* are set to 0, and values above *hiVal* are set to 1. Values in between are ramped. You also have roll-off control, plus a *matteMult* toggle. If there is no Z channel in your image, this function does not work.

If the Z channel is not directly imported with the *FileIn*, you can copy it over with the Copy command, using *z* as your channel.

**Note:** When you import from Maya, there is a strange *-1/distance Z* setting. This basically means that the numbers in *DepthKey* are impractical. To correct this, go to [www.highend2d.com](http://www.highend2d.com) and download the *MayaDepthKey* macro. The macro operates exactly like the normal *DepthKey*, but does the conversion math for you.

Parameters	Type	Defaults	Function
<i>loVal</i>	float	0	Any pixel below this value (as calculated per its depth) turns black.
<i>hiVal</i>	float	1	Any pixel above this value (as calculated by its depth) turns white.

Parameters	Type	Defaults	Function
<i>lo, hiSmooth</i>	float	0, 0	A roll-off factor to provide a smooth drop-off.
<i>matteMult</i>	int	0	Automatic premultiplication of the RGB channels. 0 = no premultiply 1 = premultiply

### Synopsis

```
image DepthKey(  
    image,  
    float loVal,  
    float hiVal,  
    float loSmooth,  
    float hiSmooth,  
    int mMult  
);
```

### Script

```
image = DepthKey(  
    image,  
    loVal,  
    hiVal,  
    loSmooth,  
    hiSmooth,  
    mMult  
);
```

### Command Line

*shake -depthkey loVal hiVal loSmooth etc...*

### DepthSlice

Similar to *DepthKey*, the *DepthSlice* function creates a slice in the alpha channel based on Z, as defined by a center point, and a drop-off range.

Parameters	Type	Defaults	Function
<i>center</i>	float	0	The center Z depth from which the slice is measured.
<i>lo, hi</i>	float	0	The distance included in the slice away from the center. lo adds distance towards the camera; hi adds thickness away from the camera.

Parameters	Type	Defaults	Function
<i>grad</i>	int	0	When enabled (1), there is a gradation from hi to lo. Beyond, the slice is still black.
<i>mirror</i>	int	0	When enabled, the effect is mirrored in Z.
<i>matteMult</i>	int	0	When enabled (1), the RGB channels are premultiplied by the new mask.

### Synopsis

```
image DepthSlice(
    image,
    float center,
    float lo,
    float hi,
    int grad,
    int mirror,
    int matteMult
);
```

### Script

```
image = DepthSlice(
    image,
    center,
    lo,
    hi,
    grad,
    mirror,
    matteMult
);
```

### Command Line

*shake -depthslice center lo hi etc...*

### Keylight (Plug-in)

*Keylight* is an Academy Award® winning keyer from Framestore CFC based in England. It accurately models the interaction of the blue or green screen light with the foreground elements, and replaces it with light from the new background. With this approach, blue spill and green spill removal becomes an intrinsic part of the process, and provides a much more natural look with less tedious trial and error. Soft edges such as hair and out-of-focus edges are pulled quite easily with the *Keylight* node.

To ensure the best results, try to always pull the key on raw plates. In the *Keylight* parameters, there is a colourspace toggle to indicate if the plate is in log, linear, or video color space—so you should not perform a color correction on film plates when feeding the plates into *Keylight*.

To understand the function in context, see “Lesson Five: Using Keylight,” in the *Shake 3 Tutorials* book.

Parameters	Type	Defaults	Function
<i>HoldOutMatte</i>	image		Indicates areas of the foreground that need to be corrected for blue spill, but should not become transparent. The replace color is used in these areas instead of the background.
<i>clipMode</i>	image	0	Sets the output resolution of the node, either the foreground image (1) or the background image (0) resolution.
<i>output</i>	string	"composite"	<p>Like <i>Primatte</i>, you have the option to do your composite with <i>Keylight</i>, but there are other output options as well:</p> <p><i>composite</i>: Renders the final composite.</p> <p><i>compOnBlack</i>: Renders the foreground objects over black.</p> <p><i>compOnReplace</i>: Renders the foreground objects over the Replace Colour. This is a good mode to test your composite.</p> <p><i>unpremultiplied</i>: Renders the foreground without premultiplying the matte. This is used when you want to continue to do transformations and color corrections after pulling the key. You then apply either a <i>MMult</i> node or enable preMultiply in an <i>Over</i> node to do the composite.</p> <p><i>status</i>: Black pixels represent areas that become pure background in the composite. Blue pixels represent areas that become spill-corrected foreground. Green pixels represent a blend of foreground and background pixels. Pure green is mostly foreground and dark green is mostly background.</p>

Parameters	Type	Defaults	Function
<i>r, g, b Screen</i>	float	0, 0, .9	The primary color of the screen color to be pulled, usually blue or green. Note that <i>Keylight</i> is tuned to the primary colors and is not effective on secondary colors (cyan, magenta, yellow). If trying to pull these colors, consider switching your image from RGB to CMY with the <i>ColorSpace</i> node, pull the key, and then switch back to RGB.
<i>screenRange</i>	float	0	Defines the ranges of colors that should be keyed out. The higher the number, the more screen is removed. A value of 0 gives the smoothest key; a value of .3 removes all the grey levels.
<i>r, g, b FgBias</i>	float	.5, .5, .5	Foreground bias is used to reduce the blue spill on foreground objects. <i>Keylight</i> uses this color to calculate which shades the screen color passes through as it interacts with the foreground elements. For example, blonde hair in front of a blue screen tends to go through a magenta stage. Setting the FG Bias to the blonde color ensures the magenta cast is properly neutralized. This value affects both opacity and spill suppression. Return it to .5, .5, .5 to effectively deactivate this effect. Avoid picking strong colors for the FG Bias. Muted shades work much better. Another way of looking at this color is as a way of preserving a foreground color that might otherwise be neutralized. For example, a pale green object in front of a green screen would normally become slightly transparent, with the background showing through instead of the pale green. By setting the FG Bias to the pale green, it is preserved in the composite.

Parameters	Type	Defaults	Function
<i>shadow, midtone, highlightBalance</i>	float	.5, shadowBalance, shadowBalance	This parameter can help you when the screen area is slightly off of a completely pure primary color, for example, cyan instead of pure blue. The transparency of the foreground is measured by calculating the difference between the dominant screen color (blue by default) and a weighted average of the other two colors (red and green). With the example of a cyan screen, there is a greater difference between the blue and the red than between the blue and the green, since cyan has more green than red. Setting the balance to 0 forces <i>Keylight</i> to ignore the second-most dominant color in the screen, which is green in the example. When set to 1, the weakest screen color (red) is ignored. There are three controls to tune the low, medium, and highlight ranges.
<i>shadow, midtone, highlightGain</i>	float	0, shadowGain, shadowGain	Increase the gain to make the main matte more transparent. This tends to tint the edges the opposite of the screen color—for blue screens edges become yellow. Decrease the gain to make the main matte more opaque.
<i>midTonesAt</i>	float	.5	Sets the level of the midtones used by the Balance and Gain controls. If you are working on a dark shot, for example, you may want to set the midtone level to a dark grey to make the controls differentiate between tones that would otherwise be considered shadows.
<i>r, g, bReplace</i>	float	.5, .5, .5	Spill can be replaced by the replaceColour. This occurs only in the opaque areas of the holdout matte. This is useful with blue areas in the foreground that you want to keep blue and opaque. The Replace Color is therefore blue.
<i>r, g, bExposure</i>	float	1, rExposure, rExposure	Allows color correction on the foreground element. This exactly mimics Shake's <i>Brightness</i> node.
<i>r, g, bGamma</i>	float	1, rGamma, rGamma	Applies a gamma correction to the foreground element. This exactly mimics Shake's <i>Gamma</i> node.

Parameters	Type	Defaults	Function
<i>saturation</i>	float	1	Applies a saturation correction to the foreground element. This exactly mimics Shake's <i>Saturation</i> node.
<i>colourspace</i>	string	"linear"	<p><i>Keylight</i> models the interaction of the blue/green light from the screen with the foreground elements. For these calculations to work correctly, you need to specify how pixel values relate to light levels. This is the function of the <i>colourspace</i> menu. Therefore, with Cineon plates (or other logarithmic files) you have the option to pull the key with or without a Delog operator before the key pull.</p> <p><i>log</i>—Color spaces are designed so that a constant difference in pixel values represents a fixed brightness difference. For example, in the Cineon 10-bit file format, a difference of 90 between two pixels corresponds to one pixel being twice as bright as the other.</p> <p><i>linear</i>—This color space has the brightness of a pixel proportional to its value. A pixel at 128 is twice as bright as a pixel at 64, for example.</p> <p><i>video</i>—Color space has a more complicated relationship, but approximately the brightness is proportional to the pixel value raised to the power of 2.2.</p>
<i>useHoldOutMatte</i>	int	1	If the third image input is used for a holdout matte, toggles the holdout matte on and off.
<i>bgColor</i>	int	1	Either pulls a key on the area outside of the frame (0), or asserts the background as the background color (for example, usually black).

### Synopsis

```
image = Keylight(
    image Background,
    image Foreground,
    image HoldOutMatte,
    int clipMode,
    string output,
    float rScreen,
```

```

float gScreen,
float bScreen,
float screenRange,
float rFgBias,
float gFgBias,
float bFgBias,
float shadowBalance,
float midtoneBalance,
float highlightBalance,
float shadowGain,
float midtoneGain,
float highlightGain,
float midTonesAt,
float rReplace,
float gReplace,
float bReplace,
float rExposure,
float gExposure,
float bExposure,
float rGamma,
float gGamma,
float bGamma,
float saturation,
string colourspace,
int useHoldOutMatte,
int bgColor,
);

```

### Script

```

image = Keylight(
    Background,
    Foreground,
    HoldOutMatte,
    clipMode
    "output",
    rScreen,
    gScreen,
    bScreen,
    screenRange,
    rFgBias,
    gFgBias,
    bFgBias,

```

```

shadowBalance,
midtoneBalance,
highlightBalance,
shadowGain,
midtoneGain,
highlightGain,
midTonesAt,
rReplace,
gReplace,
bReplace,
rExposure,
gExposure,
bExposure,
rGamma,
gGamma,
bGamma,
saturation,
"colourspace",
useHoldOutMatte,
bgColor
);

```

### LumaKey

The *LumaKey* function creates a key in the alpha channels based on overall luminance. Values below *loVal* are set to zero, and values above *hiVal* are set to 1. Values in between are ramped. You also have roll-off control, and an *mMult* toggle.

This is a fast way to place the luminance of an image into the alpha layer, but this can also be done with a *Reorder—rgbl*.

Parameters	Type	Defaults	Function
<i>loVal</i>	float	0	Any pixel below this value (as calculated per its luminance) turns black.
<i>hiVal</i>	float	1	Any pixel above this value (as calculated by its luminance) turns white.
<i>lo, hiSmooth</i>	float	0, 0	A roll-off factor to provide a smooth drop-off.
<i>mMult</i>	int	0	Automatic premultiplication of the RGB channels. 0 = no premultiply 1 = premultiply

## Synopsis

```
image LumaKey(  
    image,  
    float loVal,  
    float hiVal,  
    float loSmooth,  
    float hiSmooth,  
    int mMult  
);
```

## Script

```
image = LumaKey(  
    image,  
    loVal,  
    hiVal,  
    loSmooth,  
    hiSmooth,  
    mMult  
);
```

## Command Line

*shake -lumakey loVal hiVal loSmooth etc...*

## Primatte (Plug-in)

The *Primatte* plug-in is the latest update of Photron's Primatte keying software. *Primatte* allows you to scrub across an image to determine matte areas in order to pull a key (or alpha channel) for a composite. The plug-in also works on the RGB channels to suppress spill (the leaking of blue or green color onto the foreground objects).

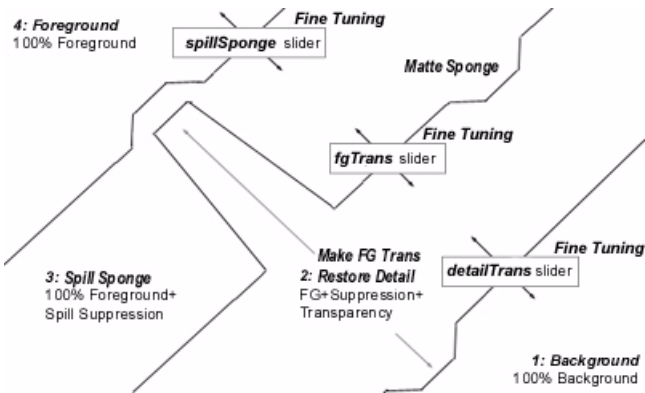
Note the script uses a special series of information that it passes to the *Primatte* plug-in. These numbers are encoded, which means that *Primatte* must be set using the interface.

It is recommended to read "Lesson Six: Using Primatte" in the *Shake 3 Tutorials*, as well as the "Blue and Green Spill Suppression" on page 243, to learn how to maximize the effectiveness of this node by combining it with other functions.

### Supplying the Background Image

Although you can output *Primatte* with a premultiplied foreground with no background image, you should supply a background input if possible, as *Primatte* still calculates in some of this information. If you do not supply this image, black ringing appears around the edges. If this is impractical, toggle the *replaceMode* parameter to use color and supply an appropriate Replace Color.

In *Primatte*, you assign color to one of four zones by clicking one of the eight large operator buttons, and then scrubbing for a color in the Viewer. These four zones are arranged around a center point in 3D color space, with each zone like a layer of an onion. The following is a chart of the operator/zone assignments. Note that decolor all scales the entire 3D space, and shifts all color either toward or away from the foreground. Therefore, it does not involve picking a color, only moving a slider.



Operator	Function
Center	This is the initial pixel scrub, and is always operation 0. This determines the center of the 3D polyhedron (described above), and is therefore extremely important. The multiplier slider modifies the size of the center area in 3D space. Therefore, a higher multiplier value expands the size of the background space, sucking in more of that color. The result is more transparent. A value lower than 1 reduces the amount of color sucked out by the initial background color pick.
Background	Assigns pixels to the background. They are 100 percent transparent with no spill suppression.

Operator	Function
<i>Decolor All</i>	Shrinks or expands the polyhedron between zone 4 (all foreground) and zone 3 (foreground plus spill suppress). By using a positive value on the slider, the shell expands, effectively shifting across the entire image more color from the foreground into the suppressed area. By using a negative value, the shell is contracted, thereby slipping more values away from the suppressed area into the foreground area. Because the shells cannot intersect, if you shrink it too much, you also crush the smaller interior shells, causing all values to shift toward the background.
<i>FineTuning</i>	<p><i>spillSponge</i>: When this mode is selected, the cursor motion in the Decolor slider performs a color adjustment of the sampled color against the background. After sampling a color region from the image, the more to the right the cursor moves, the less of the background color component (or spill) is included in that color region. The more to the left the cursor moves, the closer the color component of the selected region is to the spill suppress color.</p> <p><i>fgTrans</i>: Adjusts the transparency of the matte against the sampled color. After sampling a color region from the image, the more to the right the cursor moves, the more transparent the matte becomes in that color region. This is equivalent to Make FG Trans, but with more control.</p> <p><i>detailTrans</i>: Determines the transparency of the sampled color when it is close to the background color. The slider in this mode is useful for restoring the color of pixels that are faded due to similarity to the background color. If you slide to the left, picked areas are more opaque. This is equivalent to Restore Detail, but has more control.</p>
<i>Foreground</i>	When this mode is selected, the sampled pixels within the image window become 100 percent foreground. The color of the sampled pixels is the same color as in the original foreground image. The matte is completely white.
<i>Make FG Trans</i>	Makes foreground material have more transparency. This is for adjusting elements like clouds or smoke. It is the equivalent of fgTrans in FineTuning, except with no slider control.
<i>Matte Sponge</i>	Used to restore foreground areas lost during spill suppression. It only affects the alpha channel.

Operator	Function
<i>Restore Detail</i>	Removes transparency on background material. It is useful for restoring lost details such as hair. It is the equivalent of detailTrans in the Fine Tuning subtree, but with no slider.
<i>Spill Sponge</i>	Affects the color of the foreground, but not the matte. It suppresses the color you pick. This is usually used on spill areas that are known to be opaque, for example, blue spill on the face or body. If the change is too drastic, supplement or replace the operation with Fine Tuning—spillSponge.

The following table lists the other parameters in the *Primatte* plug-in.

Parameters	Type	Function
<i>foreground</i>	image	The blue screen or green screen image.
<i>background</i>	image	The background image. <i>Primatte</i> seems to work better with a background image when output is set to "on black" or "comp."
<i>garbageMatte</i>	image	Use to get rid of rigging or other elements you want to be transparent that are otherwise not pulled by the keying operation.
<i>boldoutMatte</i>	image	A matte input for things you want to be opaque.
<i>defocusedFg</i>	image	Can be used to take a blurred version of the blue screen to help deal with grain variation for film plates. Typically, you attach a <i>Blur</i> node to the blue screen footage and insert it here.
<i>replaceImage</i>	image	You can do spill suppression with a solid color, with the background image (when no replaceImage is supplied), or with an alternative replaceImage to provide the color that goes into spill-suppressed areas. Common inputs for this are: bg, bg with applied <i>Blur</i> , fg with applied <i>Blur</i> , fg with applied <i>Monochrome</i> , or fg with an <i>AdjustHSV</i> attached.
<i>clipMode</i>	int	The resolution to take, either the foreground (0) or the background (1).

Parameters	Type	Function
<i>output</i>	int	<p>You are not obliged to use <i>Primatte</i> for your composite. This is especially helpful for transformations on the foreground object after the matte is pulled. Pull the matte on the full-resolution image prior to any scaling. The output setting determines what is changed by <i>Primatte</i>:</p> <p><i>0 Alpha Only</i>—Only the matte is affected.</p> <p><i>1 On Black</i>—The foreground image and the matte are changed.</p> <p><i>2 Composite</i>—If there is an optional second input image, this composites that background in.</p> <p><i>3 Status</i> — This special output gives you a color code to determine where each of the pixels is positioned in the four <i>Primatte</i> zones.</p> <p><i>Black - Zone 1</i>—All background.</p> <p><i>Blue - Zone 2</i>—Transparent foreground.</p> <p><i>Green - Zone 3</i>—Suppressed foreground.</p> <p><i>Red - Zone 4</i>—All foreground.</p>
<i>arithmetic</i>	int	<p>Determines how <i>Primatte</i> affects the foreground matte channel.</p> <p><i>0 Replace</i>—Replaces the fg matte channel completely.</p> <p><i>1 Subtract</i>—Subtracts the <i>Primatte</i>-derived matte from the incoming matte.</p> <p><i>2 Multiply</i>—Multiplies the two mattes together.</p> <p><i>3 Add</i>—Adds the two mattes together.</p>
<i>processBG Color</i>	int	<p>This toggle tells Shake whether or not to consider the pixels outside of the frame. When disabled (default mode), outside of the image is assumed to be 100 percent transparent. When enabled, outside of the frame is treated the same way black pixels are treated by your <i>Primatte</i> scrubs.</p>
<i>g,bMatte Channel</i>	string	<p>The channels to be used for the garbage and holdout mattes. If there is no input for those image plugs, these parameters have no effect.</p>
<i>replaceMode</i>	int	<p>When you perform spill suppression through the use of the spillsponge operator or the fine tuning operator, these suppressed areas are shifted from blue (or whatever your center value is) toward a different color. By default, you use an image, either the background image or the replaceImage, if that input is used. You can toggle it to use color (1) to alternatively use the Replace Color.</p>

Parameters	Type	Function
<i>Replace Color</i>	float	The color used in the spill suppressed area. This is active only if <i>replaceMode</i> is set to 1—use color.
<i>Current Operation</i>	int	This slider picks the current scrub. Each scrub operation is maintained separately in memory. You can move the slider to return to any previous scrub and adjust or delete it. To see the type of operation you have done, look at the Scrub Color Button.
<i>Delete Op</i>	NA	Deletes the current operation (see above). It does not reset <i>Primatte</i> to 0, unless of course you delete all operations. For a better way to reset <i>Primatte</i> , select the <i>Primatte</i> node in the Node View, and <b>Ctrl</b> -click the <i>Primatte</i> function in the Key tab.
<i>Scrub Color</i>	NA	The currently picked color. To select the color again, click the button and scrub. You should scrub across the unmodified RGB values. The type of scrub appears in the Scrub Color Button top.
<i>Eval to End</i>	NA	As you adjust the Current Operation slider, you may want to see the composite only up to that point. Turn off Eval to End to evaluate only up to your current operation.
<i>Active</i>	NA	You can disable any operation you have done. This is used in conjunction with the Current Operation slider.
<i>Amount</i>	NA	The amount of adjustment. This slider disappears if the current operation does not have a modifier.

### Synopsis

```
image = PrimatteX(
    image foreground,
    image background,
    image garbageMatte,
    image holdoutMatte,
    image defocusedForeground,
    image replaceImage,
    int IDontKnow,
    int clipMode,
    int output,
    int arithmetic,
    int processBGColor,
    const char * gMatteChannel,
    const char * hMatteChannel,
    int replaceMode,
```

```

float rReplace,
float gReplace,
float bReplace,
const char * encryptedPrimatteSettings
);

```

### Script

```

image = PrimatteX(
    foreground,
    background,
    garbageMatte,
    holdoutMatte,
    defocusedForeground,
    replaceImage,
    something,
    clipMode,
    output,
    arithmetic,
    processBGColor,

    "gMatteChannel",
    "hMatteChannel",
    replaceMode,
    rReplace,
    gReplace,
    bReplace,
    "encryptedPrimatteSettings"
);

```

### SpillSuppress

The *SpillSuppress* function suppresses blue spill, with controls for red and green gain. This function can also be switched over for green spill suppression. *SpillSuppress* uses a color-correction algorithm, so the entire image is modified.

Parameters	Type	Defaults	Function
<i>rGain, gGain</i>	float	1, 1	Since the spill suppress darkens the overall image, you can use these to slightly compensate for the brightness in the other two channels.

Parameters	Type	Defaults	Function
<i>lumGain</i>	float	1	An overall luminance gain.
<i>screenColor</i>	int	0	Select color to suppress. 0 = Suppress blue 1 = Suppress green. gGain then converts to bGain.

**Synopsis**

```
image SpillSuppress( image,  
    float rGain,  
    float gGain,  
    float lumGain,  
    int screenColor  
);
```

**Script**

```
image = SpillSuppress( image,  
    rGain,  
    gGain,  
    lumGain,  
    screenColor  
);
```

**Command Line**

```
shake -spillsuppress rGain gGain lumGain screenColor
```

# Color Correction: Part I

## Chapter Summary

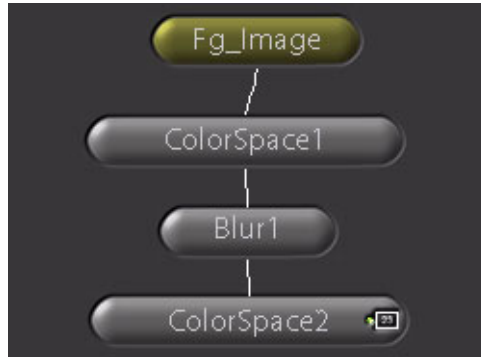
- Color Space
- Using the Color Picker
- Color Correction Tools
- Atomic-Level Functions
- Utility Correctors
- Consolidated Color Correctors
- Color Corrections With the Infinite Workspace

## Color Space

Shake works with a color range of 0 to 1 in RGB linear space. This has three implications:

- Shake allows you to work in different bit depths, thus 0 is considered black and 1 is considered white. Note that float allows you to go below 0 or above 1.
- When working with logarithmic (usually) Cineon plates, apply a *LogLin* function to avoid unpredictable results. The *LogLin* node allows you to jump from logarithmic to linear space, or linear to logarithmic space. For more information, see “The Logarithmic Cineon File” on page 349.
- To apply an effect such as *Blur* in a different color space (for example, to blur the color difference channels in a YUV image, but not the luminance), apply a *ColorSpace* node to the image to convert it to the different color space. Add the effect node, in this example the *Blur* node. To return to your original color space, add another *ColorSpace* node.

**Note:** To view the images in color, please refer to the PDF documentation.

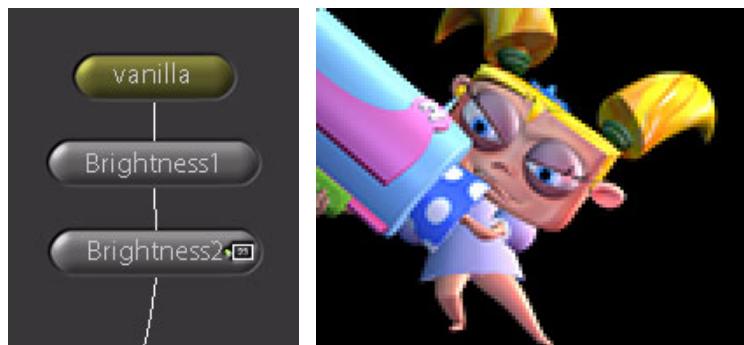


For a practical discussion on using this technique, see Chapter 7, “Keying.”

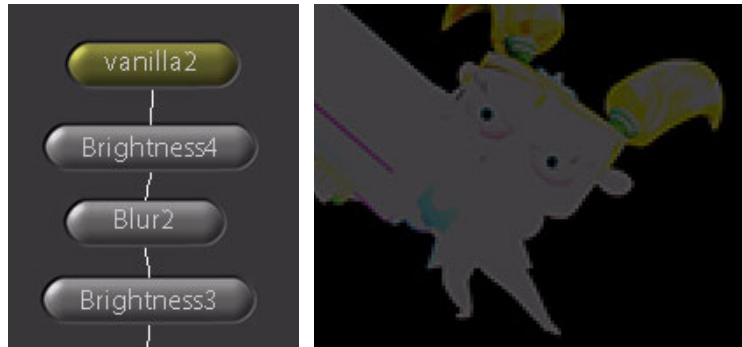
### About Color Correction Concatenation and Float Calculations

A powerful aspect of Shake’s color handling is that it concatenates many color corrections. This means if you have ten concatenating color functions in a row, it internally compiles the functions into one big lookup table, and executes that table. (Internally, one node is processed, rather than ten.) A second important benefit is that by not clipping data, much more mathematical information is preserved. Therefore, concatenation preserves quality and speeds processing time. This is good.

In the following example with concatenation, a *Brightness* node (set to 4) is applied to the *Vanilla* node. Next, a second *Brightness* (set to .25) is added. In the first node tree, the examples concatenate to multiply the image by 1, so no change occurs. Thanks go out to Wild Brain, Inc. for the use of the Vanilla image.



In the next example with no concatenation, all of the values are boosted to 1 when multiplied by 4. Because the *Blur* node breaks the concatenation, the values are dropped to a maximum value of .25 when a *Brightness* node (set to .25) is applied.



This is bad.

The following nodes concatenate:

- *Add*
- *Brightness*
- *Clamp*
- *Compress*
- *ContrastRGB* (but not *ContrastLum*)
- *DelogC*
- *Expand*
- *Fade*
- *Gamma*
- *Invert*
- *LogC*
- *Lookup*
- *Multi*
- *Set*
- *Solarize*

*AdjustHSV* and *LookupHSV* concatenate only with each other.

Also, a “C” (for concatenation) appears in the upper-left corner of the icons of the concatenating nodes.



### Masked Nodes Break Concatenation

If you mask a node, concatenation is broken. To avoid broken concatenation, use a node tree structure with *KeyMix*. For an example, see the “Use a Layer–*KeyMix* node” step (at the end of the tutorial) in Lesson Two: Intermediate Tutorial” in the *Shake 3 Tutorials* book.

To avoid broken concatenation, boost your bit depth to float with an Other–*Bytes* node. This means you can preserve your values above 1 or below 0. Use caution when using the *Bytes* node, as there is a processing penalty for the extra overhead. For more information, see “Bit Depth” on page 363.

### Premultiplied Elements and CG Element Correction

You may see a problem with your image edges when color correcting an image from a 3D render. In the following example, *Vanilla*’s contrast is dropped to .4, which quickly reveals premultiplication problems.



To clean this up, insert an *MDiv* node before your color correction nodes (all color correction nodes). Make sure you premultiply your images before you composite with an *MMult* node, or enable preMultiply in the *Over* node.



In the next example, the *Fade* node and the *Add* node are within an *MDiv* node and an *MMult* node in the tree.

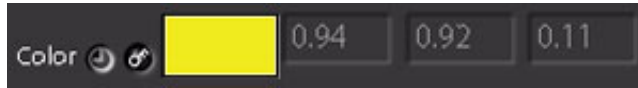


The *Fade* node is not affected by premultiplication because it operates on the RGBA channels evenly. In the above example, it is inside the *MDiv*/*MMult* node pair for concatenation with the *Add* node.

For a detailed description of premultiplication and its importance in compositing, see “About Premultiplication and Compositing” on page 368.

## Using the Color Picker

To access the Color Picker, click the Color Picker tab (next to the Node View and Curve Editor tabs). You can also access the Color Picker in the Parameters View of a node that contains color controls. For example, in the Color-*Add* node parameters, click a color box to show the Color Picker. Values selected in the Color Picker are fed into the three node parameters.



Once the color box is clicked, the Color Picker tab opens. Drag the cursor to select a point:

- To change the overall value of the wheel and the selected color, drag in the bottom of the greyscale bar.



- To select a color from an image, drag in the Viewer.

To animate the color values, use the associated Autokey button in the Parameter tab .

To read the different color channels for a node, open the color subtree. There are two levels:

- The first-level subtree allows you to modify a channel with the use of a single slider. The channels are Red, Green, Blue, Hue, Saturation, Value, Temperature, Magenta/Cyan, and Luminance.
- The second subtree lists each channel individually, and allows you to enter expressions.



The channel slider buttons can be considered reminders of the hot keys for the Virtual Color Picker. Offset (hot key **O**), however, is not included in the channel slider buttons.

To use the Virtual Color Picker in a color box, press a channel key **R** (Red), **G** (Green), **B** (Blue), **O** (Offset), **H** (Hue), **S** (Saturation), **V** (Value), **R** (Temperature), **C** (Cyan), **M** (Magenta), **Y** (Yellow), **L** (Luminance), and drag the cursor along the first line.



To sample color from the Viewer, enable Sample From Viewer in the Color Picker tab, and drag the cursor across a Viewer. If you are doing a color correction and the Color Picker is linked to a *Mult* node, the real-time update of the color correction interferes with onscreen selection of color, and causes a feedback loop. Switch on Use Source Buffer, and the values are taken from the parent node, not the current node. This is especially useful when doing key scrubs, as it picks the value off of the prekeyed image.



The buttons on the top of the Color Picker control what values are returned when you scrub, either the current pixel's value, the average value of the pixels you scrub, the minimum value, or the maximum value of the pixels scrubbed.

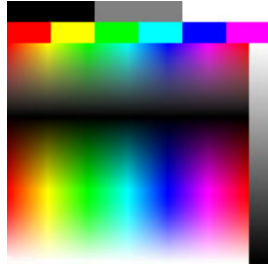


The Color Picker also has an associated Palette. To select a color, click a palette color button. You can also drag and drop between the palette buttons and other Color Picker switches.



To assign a color to a Palette switch, drag from a color switch to the Palette switch. To save the new color assignments, choose File > Save Interface Settings.

To quickly select a color from a color box, you can use the Pop-Up Color Palette. To access the palette, right-click on a color box and drag the cursor to select a color from the pop-up palette.



The Pop-Up Color Palette is available in the Parameters View of all nodes that contain color controls, or in the Color Picker.

### Customizing the Interface for Use With Palette and Color Picker

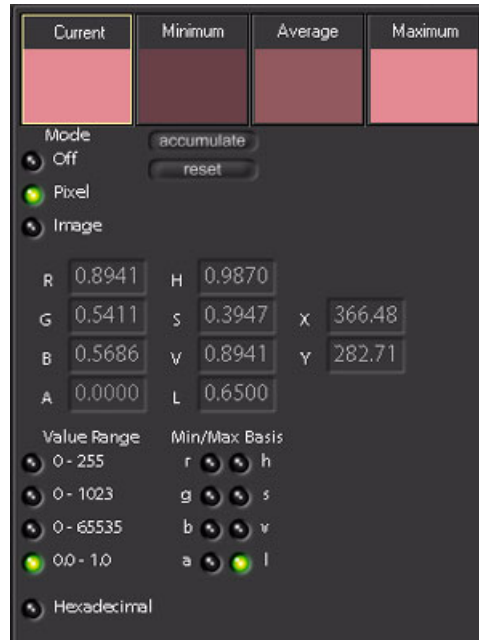
These commands are placed in your *ui.b* file. For more information on customizing Shake, see Chapter 17, “Customizing Shake.”

Code	Notes
<pre>nuiSetColor(1,1,0,0); nuiSetColor(2,1,0.5,0); nuiSetColor(3,1,1,0);</pre>	Assigns a color to a Palette swatch; the first number is the assigned box. Values are in a range of 0 to 1.
<pre>nuiPushControlGroup("Color"); nuiGroupControl("MyFunction.red"); nuiGroupControl("MyFunction.green"); nuiGroupControl("MyFunction.blue");     nuiPopControlGroup();     nuiPushControlWidget(         "Color",  nuiConnectColorTriplet(     kRGBToggle,     kCurrentColor,     1     )     );</pre>	Assigns a Color Picker to your custom macros. This code creates a subtree named "Color" that contains the three parameters—red, green, and blue—although these can be any three parameters. The last function ( <i>nuiConnectColorPCtrl</i> ) selects what color space the values are returned in, what type of value, and if you want to use the source buffer or not.

## About the Pixel Analyzer

The Pixel Analyzer is an analysis tool to find and compare different color values on an image. You can examine minimum, average, current, or maximum pixel values on a selection (that you make), or across an entire image.

**Note:** The Pixel Analyzer tool is not to be confused with the *PixelAnalyzer* node, found in the Other tab. For more information, see “PixelAnalyzer Node” on page 282.



When you drag across an image in the Viewer with the cursor, the values update in the Pixel Analyzer. You can usually use the default Pixel Analyzer settings: Click the color swatch that you want to examine, Curr (Current color value), Min (Minimum color value), Avg (Average color value), or Max (Maximum color value). You can toggle between the different color swatches repeatedly without dragging again. The values appear in the text fields below the color swatches.

Since the Pixel Analyzer keeps the examined pixels in memory, if you switch images in the same Viewer, the Pixel Analyzer updates its values based on the new image. Because of this, you can compare images or perform color corrections, as the color correction constantly updates the analyzer.

### Mode

- *Off*: Turns off the Pixel Analyzer.
- *Pixel*: Analyzes the pixels based upon the area scrubbed with the cursor.
- *Image*: Analyzes the entire image—no scrubbing is necessary.

### Accumulate

When you click the Accumulate button, all scrubbed pixels (not just the current pixel) are considered for Average, Min, and Max calculations—until the Reset button is clicked. So, Average calculates the average of every scrub since the last Reset, and Min and Max replace their values if a new minimum or maximum value is scrubbed. If the Analyzer is on Image Mode, this button has no effect.

### Reset

Resets the scrubbed buffer to black.

### Value Range

Shake describes its color in a range of 0 to 1 (0, 0, 0 is black; 1, 1, 1 is white). However, you can set this to a different numerical range, for example, 0, 0, 0 is black, and 255, 255, 255 is white. This is only a display setting for the numerical data—the images do not change.

### Hexadecimal

Toggles the numerical display to hexadecimal values.

### Min/Max Basis

The Min/Max Basis buttons set the channel for calculation of the Minimum and Maximum swatches. Normally, this is set to l (luminance). To determine the minimum values in only the red channel, toggle the Min/Max Basis to r (red). For example, one pure red pixel and one pure green pixel are equivalent pixels based on luminance. However, based on red, the green pixel has a minimum value of 0, and so Min returns a different number.

### Custom Entries

You can insert your own functions to return data using the following code. You provide a label and the function in a *ui.b* file. The two default plugs are called *exp10* and *expf*:

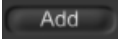

```
gui.pixelAnalyzer.customLabel1 = "exp10";  
gui.pixelAnalyzer.customFunc1 = "(int)(1024*log(l/0.18)/log(10))";  
gui.pixelAnalyzer.customLabel2 = "expf";  
gui.pixelAnalyzer.customFunc2 = "log(l/0.18)/log(10)";
```

### PixelAnalyzer Node

The *PixelAnalyzer* node, located in the Other tab, allows you to examine one or more areas of pixels over a frame range. The data is then stored as the average, minimum, and maximum values of each area on a frame-by-frame basis. This information can be used to assist in matching colors, or to reduce flicker in a plate, and are typically fed into expressions in a Color node.

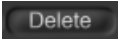
The working method for analyzing color is similar to the Tracker. However, the analyzer does not do any tracking itself. It merely grabs color values.

#### To analyze an area:

- 1 Attach the *PixelAnalyzer* node to an image.
- 2 Position the box in the area to examine, and adjust the box size as necessary.
- 3 To manually animate the box, use the Viewer Autokey button. Otherwise, the box remains stationary.
- 4 To create several analysis areas, use the Add button  in the lower portion of the parameters.
- 5 Verify the frame range in the analysisRange parameter.
- 6 Press the Analyze Forward (or Analyze Backward) button in the Viewer .

The analysis begins from the current frame. Each active box (controlled by the visibility toggle by the areaName) grabs color values for its area and then calculates the average, minimum, and maximum values for that area.

To delete an area box, do one of the following:

- In the Viewer, select the box (click the box), and then click Delete  in the *PixelAnalyzer* parameters.
- In the *PixelAnalyzer* parameters, click the areaName text field (highlighted green) and click Delete.

#### To save the data of an area:

- 1 Click the Save button in the lower portion of the *PixelAnalyzer* parameters.  
The “Save pixel analysis data to file” window appears.
- 2 Select or create the directory to store the saved the data, name the file, and click OK.

Parameters	Type	Defaults	Function
<i>analysisRange</i>	int	1	The frame range for the analysis. Analysis always starts on the current frame.
<i>limitProcessing</i>	int	1 (on)	When the analysis is launched, this toggle indicates if only the area inside the area boxes is updated.
<i>areaName</i>	string	area1, 2, etc.	The name of each area box. The user may change this name. The associated parameters for each area update their names based on the new name.

Parameters	Type	Defaults	Function
<i>areaAverageRed, Green, Blue, Alpha</i>	float	0	The average value of every pixel in the area.
<i>areaMinimumRed, Green, Blue, Alpha</i>	float	0	The minimum value of every pixel in the area.
<i>areaMaximumRed, Green, Blue, Alpha</i>	float	0	The maximum value of every pixel in the area.
<i>areaX, Y</i>	int	width/2, height/2	The center pixel of the area box.
<i>areaWidth, Height</i>	int	30	The area width and height.
<i>areaVisible</i>	int	1	A toggle to indicate if the area box is visible, and is therefore updated when the Analyze button is clicked. This is not typically controlled by the user, but is a result of the visibility toggle next to the area name parameter.
<i>areaEnabled</i>	int	10	

### Synopsis

```
image PixelAnalyzer(
    image In,
    const char * analysisRange,
    int limitProcessing,
    const char * areaName,
    float <areaName>AverageRed,
    float <areaName>AverageGreen,
    float <areaName>AverageBlue,
    float <areaName>AverageAlpha,
    float <areaName>MinRed,
    float <areaName>MinGreen,
    float <areaName>MinBlue,
    float <areaName>MinAlpha,
    float <areaName>MaxRed,
    float <areaName>MaxGreen,
    float <areaName>MaxBlue,
    float <areaName>MaxAlpha,
    int <areaName>X,
    int <areaName>Y,
    int <areaName>Width,
```

```

    int <arealName>Height,
    int <arealName>Visible,
    int <arealName>Enabled,
    ...area2 parameters...
);

```

### Script

```

image PixelAnalyzer(
    In,
    anaylsisRange,
    limitProcessing,
    "arealName",
    <arealName>AverageRed,
    <arealName>AverageGreen,
    <arealName>AverageBlue,
    <arealName>AverageAlpha,
    <arealName>MinRed,
    <arealName>MinGreen,
    <arealName>MinBlue,
    <arealName>MinAlpha,
    <arealName>MaxRed,
    <arealName>MaxGreen,
    <arealName>MaxBlue,
    <arealName>MaxAlpha,
    <arealName>X,
    <arealName>Y,
    <arealName>Width,
    <arealName>Height,
    <arealName>Visible,
    <arealName>Enabled,
    ...area2 parameters...
);

```

### Command Line

Not appropriate for command line.

## Color-Correction Tools

In general, Shake has three classes of color correctors. All color correction functions are located in the Color Tool Tab.

### Atomic-Level Correctors

Each atomic-level corrector (*Add*, *Brightness*, *Clamp*, *Lookup*, etc.) applies a single basic mathematical function to your color, such as add, multiply, gamma, and basic lookup curve functions. These functions usually concatenate together for speed and accuracy, and are fast ways to manipulate your data for a wide variety of purposes. In the Color tab, a graph that represents the behavior of the function appears on the icons. In the following illustration, notice the difference in the graph curves for the *Clamp*, *Compress*, and *Expand* nodes.



### Utility Correctors

The utility correctors are tools that prepare your data for other functions. Typically, these functions include *ColorSpace*, *ColorX*, *Lookup*, *MDiv*, *MMult*, *Reorder*, *Set*, *SetAlpha*, *SetBGColor*, and *VideoSafe*. There is also a certain degree of programmability in *ColorX* and the *Lookup* functions, so you can enter expressions on your image.

### Consolidated Correctors

The consolidated correctors (*ColorCorrect*, *ColorMatch*, *ColorReplace*, *HueCurves*) are your primary tools for tasks such as matching skin tones, shadow levels, spill suppression, and so on. The atomic-level correctors can also be found in these, but are wrapped in with many other tools to provide more control.

The following table includes general guidelines to help you understand why there are so many nodes in the Color tab. For example, you can perform an add operation with an *Add* node or with a *ColorCorrect* node, with no difference in the result. *Add*, however, is a bit easier to wade through, and concatenates with many other operators.

The following is a basic list of the color correction nodes and how they are useful.

Function	Explanation
<i>Add</i>	Raises or lowers colors evenly. This is the only legal color correction in log space.
<i>AdjustHSV</i>	Shifts the entire Hue, Saturation, or Value range.
<i>Brightness</i>	Multiplies the RGB channels by a single value.
<i>Clamp</i>	Good for clamping off values that go above or below a desired level.
<i>ColorCorrect</i>	A mondo-maximus titanic node of color-correctness. Does tons of things, and includes a premultiplied input flag.

Function	Explanation
<i>ColorMatch</i>	Matches your lows, midtones, and highlights to a second set of colors.
<i>ColorReplace</i>	Switches one color with another. It also makes a better chroma keyer than the <i>ChromaKey</i> node.
<i>ColorSpace</i>	Swaps your image into a different color space, such as, RGB, HSV, CMY, etc.
<i>ColorX</i>	For pixel-based mathematical expressions.
<i>Compress</i>	Squeezes your ranges up and down, and is particularly useful for fog effects with a <i>DepthKey</i> mask.
<i>ContrastLum/RGB</i>	Contrast. Use <i>Lum</i> to preserve your color levels.
<i>Expand</i>	Another levels adjuster that raises or lowers your low and high points.
<i>Fade</i>	Opacity control. Same as using <i>Mult</i> with identical values in all four channels.
<i>Gamma</i>	A gamma correction.
<i>HueCurves</i>	A node that isolates and adjusts an image based on its hue. Ideal for spill suppression.
<i>Invert</i>	Turns black to white and vice versa. Works best on normalized images between 0 and 1.
<i>LogLin</i>	Logarithmic color space conversion.
<i>Lookup/HLS/HSV</i>	Lookup expressions or curve manipulation on your image. It is faster for non-pixel based lookups than <i>ColorX</i> .
<i>LookupFile</i>	Pulls a lookup table from a file.
<i>MDiv/MMult</i>	Used to unpremultiply and premultiply an image by its alpha channel.
<i>Monochrome</i>	Gives you weighted control over turning your image black and white.
<i>Mult</i>	Multiplies your channels separately from each other.
<i>Reorder</i>	Swaps channels around within an image. See also <i>Layer-Copy</i> and <i>Layer-SwitchMatte</i> to grab channels from other images.
<i>Saturation</i>	Controls your saturation levels.
<i>Set</i>	Sets a channel(s) to a constant level.
<i>SetAlpha</i>	Sets the alpha level to a constant value, and crops the Infinite Workspace.

Function	Explanation
<i>SetBGColor</i>	Sets the color outside of the DOD.
<i>Solarize</i>	A misguided <i>Invert</i> function. Good for Beatles album covers.
<i>Threshold</i>	Clips your values.
<i>VideoSafe</i>	Clips values above video-legal ranges.

## Atomic-Level Functions

These nodes are described as atomic-level because they each apply a single mathematical operation. Due to their simplicity, they are very easy to work with. They are also ideal for use on the command line.

### Add

The *Add* function adds color to the R, G, B, A, or Z channels. This adds color to black areas, including those beyond the image frame, in case you move the image later. Any correction that occurs outside of the DOD can be corrected with the *SetBGColor* node. Shake's color is described in a range of 0 to 1, so adding -1, -1, -1 makes your image completely black. If you add the fifth value, depth, you effectively add a Z-channel with your add value to the image.

Parameters	Type	Defaults	Function
<i>red, green, blue, alpha, depth</i>	float	0, 0, 0, 0, 0	The amount to add to the RGBAZ channels of the image. Shake's color is described as 0 to 1, so add 1, 1, 1 to make your image completely white.

### Synopsis

```
image Add( image,
    float red,
    float green,
    float blue,
    float alpha,
    float depth
);
```

### Script

```
image = Add(image, red, green, blue, alpha, depth);
```

### Command Line

```
shake -add red green blue alpha depth
```

**Examples**

```
shake lisa.iff-add .2
shake lisa.iff-add .1 .5 .2
shake lisa.iff-add .1 .5 .2 .2
shake lisa.iff-add .1 .5 .2 .2 -savescript example.shk
```

**See Also**

“Mult” on page 298, “Gamma” on page 296

**Brightness**

The *Brightness* function is simply a multiplier on the RGB channels. It differs from *Fade* in that *Fade* also affects the alpha channel. For individual channel control, use *Mult*.  
You can also use this to convert your image to a single-channel alpha image by using a brightness of 0.

Parameter	Type	Default	Function
<i>value</i>	float	1	The brightness factor. A value greater than 1 increases brightness; less than 1 darkens it. A value of 0 is complete black and reduces your image to a single-channel alpha image.

**Synopsis**

```
image Brightness( image, float value);
```

**Script**

```
image = Brightness( image, value);
```

**Command Line**

```
shake -brightness value
```

**Examples**

```
shake lisa.iff-bri .2
shake lisa.iff-bri 2
shake lisa.iff-bri "Linear(0,0@1,3@20)" -t 1-20
```

**See Also**

“Mult” on page 298, “Fade” on page 295

### Clamp

The *Clamp* function clamps off values above and below a certain range. For example, if your redHi value is .7, any value above that is set to .7. You can isolate the red, green, blue, and alpha values.

Parameters	Type	Defaults	Function
<i>Lo</i>	float	0, 0, 0, 0	The low value that all values are set to if they are less than this number. 0 is no change.
<i>Hi</i>	float	1, 1, 1, 1	The high value that all values are set to if they are more than this number. 1 is no change.

### Synopsis

```
image Clamp( image,  
    float rLo,  
    float gLo,  
    float bLo,  
    float aLo,  
    float rHi,  
    float gHi,  
    float bHi,  
    float aHi  
);
```

### Script

```
image = Clamp( image,  
    rLo, gLo, bLo, aLo  
    rHi, gHi, bHi, aHi  
);
```

### Command Line

*sbake -clamp rLo gLo etc...*

### Examples

```
sbake lisa.iff -clamp .5  
sbake lisa.iff -clamp 0 0 0 0 .5  
sbake lisa.iff -clamp .5 rLo rLo 0 .6 rHi rHi 1
```

### See Also

“Compress” on page 291, “Expand” on page 294, “Lookup” on page 305, “LookupHLS” on page 309, “LookupHSV” on page 310

### Compress

The *Compress* function squeezes the image to fit within the Lo and Hi range you set. Unlike *Clamp*, the entire image is modified because it is fit between the two points.

Parameters	Type	Defaults	Function
<i>Lo</i>	float	0, rLo, rLo, 0	The new lowest value in the image. 0 is no change if Hi is set to 1.
<i>Hi</i>	float	1, rHi, rHi, 1	The new highest value in the image.

### Synopsis

```
image Compress( image,
    float rLo,
    float gLo,
    float bLo,
    float aLo,
    float rHi,
    float gHi,
    float bHi,
    float aHi
);
```

### Script

```
image = Compress( image,
    rLo, gLo, bLo, aLo
    rHi, gHi, bHi, aHi
);
```

### Command Line

```
shake -compress rLo gLo etc...
```

### Examples

```
shake lisa.iff -compress .5
shake lisa.iff -compress .4 .4 .4 0 .6 .6 .6
```

### See Also

“*Clamp*” on page 290, “*Expand*” on page 294, “*Lookup*” on page 305, “*LookupHLS*” on page 309, “*LookupHSV*” on page 310

### ContrastLum

The *ContrastLum* function applies a contrast change on the image, with a smooth falloff on both the low and high ends. You can also move the center of the contrast curve up and down (for example, move it down to make your overall image brighter). Note that this contrast is based on the luminance of the pixel, so it balances red, green, and blue, and you therefore do not shift your hue. So, an image with strong primary colors looks different than an image with the same values in a *ContrastRGB*, which evaluates each channel separately.

Also note that a roll-off is built into the contrast function, giving you a smooth falloff at the ends of the curve. This is controlled by the *softClip* parameter.

Parameters	Type	Defaults	Function
<i>value</i>	float	1	The contrast value. A higher value means it pushes your RGB values toward 0 and 1. A value of 0 is no change. A lower value is low contrast.
<i>center</i>	float	.5	This is the center of the contrast curve. A lower value is a brighter image.
<i>softClip</i>	float	0	This controls the drop-off of the curve. When increased to 1, you have maximum smoothness of the curve.

### Synopsis

```
image ContrastLum(  
    image In,  
    float value,  
    float center,  
    float softClip  
);
```

### Script

```
image = ContrastLum(  
    image,  
    value,  
    center,  
    softClip  
);
```

### Command Line

```
shake -contrastlum value center softClip
```

### Examples

```
shake lisa.iff -contrastl 2
```

```

shake lisa.iff -contrastl 2.3
shake lisa.iff -contrastl 2.8
shake lisa.iff -contrastl .8

```

### See Also

“*ContrastRGB*” on page 293, “*Gamma*” on page 296, “*Add*” on page 288, “*Mult*” on page 298

### ContrastRGB

The *ContrastRGB* function applies a contrast curve on each channel individually, so you can tune the channels separately. This differs from the *ContrastLum* function in that it only changes the pixel value according to its own channel. For a good example of how the functions differ, plug a *ColorWheel* into both a *ContrastLum* and *ContrastRGB*. Notice that the *ContrastLum* is weighed away from the green values. This also means the *ContrastRGB* runs the risk of shifting the hue as you adjust your channels.

Parameters	Type	Defaults	Function
<i>value</i>	float	1, 1, 1, 1	The contrast value. A higher value means it pushes the RGB values toward 0 and 1. A low value is low contrast. A value of 0 is no change.
<i>center</i>	float	.5, .5, .5, .5	The center of the contrast curve. A lower value makes that channel brighter. A higher value makes the image darker. Generally, these values are between 0 and 1.
<i>softClip</i>	float	0, 0, 0, 0	The roll-off value to give a smooth interpolation. A value of 0 is no roll-off.

### Synopsis

```

image ContrastRGB( image,
    float rValue,
    float gValue,
    float bValue,
    float aValue,
    float rCenter,
    float gCenter,
    float bCenter,
    float aCenter,
    float rSoftClip,
    float gSoftClip,
    float bSoftClip,
    float aSoftClip
);

```

**Script**

```
image = ContrastRGB(image,
    rValue,
    gValue,
    bValue,
    aValue,
    rCenter,
    gCenter,
    bCenter,
    aCenter,
    rSoftClip,
    gSoftClip,
    bSoftClip,
    aSoftClip
);
```

**Command Line**

```
sbake -contrastrgb rValue gValue etc...
```

**Example**

```
sbake lisa.iff -contrast 1.5 1.2 1 1
```

**See Also**

“ContrastLum” on page 292, “Gamma” on page 296, “Add” on page 288, “Mult” on page 298

**Expand**

The *Expand* function squeezes the data between two points on the X axis of a graph of an image, increasing the amount of pure black and white (on a per-channel basis) in the image. Compress squeezes them on the Y axis of the image graph.

Parameters	Type	Defaults	Function
<i>r, g, b, aLo</i>	float	0, rLo, rLo, rLo	Pixels less than or equal to Lo value go to 0. At 8 or 16 bits per channel, pixels less than this value are clamped at 0.
<i>r, g, b, aHi</i>	float	1, rHi, rHi, rHi	Pixels greater than or equal to Hi value go to 1. At 8 or 16 bits per channel, pixels greater than this value are clamped at 1.

**Synopsis**

```
image Expand( image,
    float rLo,
```

```

float gLo,
float bLo,
float aLo,
float rHi,
float gHi,
float bHi,
float aHi
);

```

### Script

```

image = Expand( image,
                rLo, gLo, bLo, aLo
                rHi, gHi, bHi, aHi
);

```

### Command Line

*sbake -expand rLo rHi gLo etc...*

### See Also

“Compress” on page 291, “ContrastLum” on page 292, “Lookup” on page 305, “LookupHLS” on page 309, “LookupHSV” on page 310

### Fade

The *Fade* function multiplies the RGBA channels. It differs from *Brightness* in that *Fade* also affects the alpha channel. For individual channel control, use *Multi*.

A neat trick is to fade to 0. This effectively deactivates all nodes above the *Fade* node in the tree.

**Note:** Premultiplication is not a concern with the *Fade* node, since *Fade* treats the RGBA channels evenly.

Parameter	Type	Default	Function
<i>value</i>	float	1	The brightness factor. Greater than 1 increases brightness; less than 1 darkens it. A value of 0 is complete black.

### Synopsis

```
image Fade( image, float value);
```

### Script

```
image = Fade( image, value);
```

### Command Line

*sbake -fade value*

**See Also**

“*Mult*” on page 298, “*Brightness*” on page 289

**Gamma**

The *Gamma* function applies a gamma to your image. A value of 1 is no change. Shake’s ability to use expressions can be particularly useful here, since to invert the gamma of 1.7, for example, you can just type 1/1.7 as your gamma value. Typing .588 isn’t nearly as slick.

Parameters	Type	Defaults	Function
<i>r, g, b, aGamma</i>	float	1, rGamma, rGamma, 1	The gamma value. Duh.

**Synopsis**

```
image Gamma( image,
    float rGamma,
    float gGamma,
    float bGamma,
    float aGamma
);
```

**Script**

```
image = Gamma( image, rGamma, gGamma, bGamma, aGamma );
```

**Command Line**

```
shake -gamma rGamma gGamma bGamma aGamma
```

**See Also**

“*ContrastRGB*” on page 293, “*ContrastLum*” on page 292, “*Add*” on page 288, “*Mult*” on page 298

**Invert**

The *Invert* function inverts the color curve, so white becomes black and black becomes white. A predominantly yellow image becomes predominantly blue if the red, green, and blue (rgb) channels are selected in the channels field.

*Invert* also works on the Z channel, but assumes the Z is normalized, for example, between 0 and 1. If this is not the case, you have an unpredictable result. If you need to invert a non-normalized Z channel, use *ColorX* with a formula similar to the following in the Z channel:

*MaxZRange-z*

Parameter	Type	Default	Function
<i>channels</i>	string	"rgba"	The channels you want to invert. You can use r, g, b, a, and/or z. To use multiple channels, list them out. For example, <i>rgz</i> inverts the red, green, and Z channels.

**Synopsis**

```
image Invert( image, const char * channels );
```

**Script**

```
image = Invert( image, "channels" );
```

**Command Line**

```
shake -invert channels
```

**Monochrome**

The *Monochrome* function turns any image black and white. Default values are weighed according to the human eye's different sensitivity to red, green, and blue, but you can override the weights. Notice that it reduces a 3-channel image (RGB) to a 1-channel image (BW), and a 4-channel image (RGBA) to a 2-channel image (BWA). If the three color channels are identical, it is more efficient to use a *Reorder* with a value of "rrr," since only the red channel is read in. *Monochrome* reads all three channels in, and therefore has more I/O activity.

Parameters	Type	Defaults	Function
<i>r, g, bWeight</i>	float	.3, .59, .11	The defaults are set according to the human eye's sensitivity to color, but you can balance the colors differently to push a certain channel.

**Synopsis**

```
image Monochrome( image,  
    float rWeight,  
    float gWeight,  
    float bWeight  
);
```

**Script**

```
image = Monochrome( image, rWeight, gWeight, bWeight);
```

**Command Line**

```
shake -monochrome rWeight gWeight bWeight
```

### Mult

The *Mult* function multiplies the R,G, B, A, or Z channels. To uniformly increase the Red, Green, Blue, and alpha channels, use the *Fade* node. To affect Red, Green, Blue, but leave alpha at the same amount, use *Brightness*.

Parameters	Type	Defaults	Function
red, green, blue, alpha, depth	float	1, 1, 1, 1, 1	The value by which the incoming pixels are multiplied. Note that multiplying the Z value does not necessarily add a Z channel like the <i>Add</i> function.

#### Synopsis

```
image Mult( image,  
    float red,  
    float green,  
    float blue,  
    float alpha,  
    float depth  
);
```

#### Script

```
image = Mult( image, red, green, blue, alpha, depth);
```

#### Command Line

*shake -mult red green blue alpha depth*

#### See Also

“*Brightness*” on page 289, “*Gamma*” on page 296, “*Fade*” on page 295

### Saturation

The *Saturation* function changes the saturation value of an image.

**Note:** You can also use *Monochrome* to select a different color balance.

Parameter	Type	Default	Function
<i>saturation</i>	float	1	The saturation multiplier.

#### Synopsis

```
image Saturation( image, float value );
```

#### Script

```
image = Saturation( image, value );
```

**Command Line**

*shake -saturation value*

**See Also**

“AdjustHSV” on page 316, “ColorSpace” on page 300

**Solarize**

The *Solarize* function is a partial inverse that reverses the high or low end, depending on the value of the hi/lo flag. Values above (“hi” or 1) or below (“lo” or 0) the threshold are reversed. The resulting images are similar to a photographic effect popular in the 1960s, where the print is exposed to light during development and results in an image with blended positive and negative value ranges. It gives a metallic effect to color images.

Parameters	Type	Defaults	Function
<i>value</i>	float	.5	The point when the image is inverted.
<i>mode</i>	int	1	0 means "lo," or below the threshold value that the color is inverted. 1 means "hi," or above the threshold.

**Synopsis**

```
image Solarize( image In, float value, int mode);
```

**Script**

```
image = Solarize( In, value, mode );
```

**Command Line**

*shake -solarize value mode*

**Threshold**

The *Threshold* function cuts channels off at a certain value, and turns everything below that cut-off value to 0. Each channel can have its own separate cut value. By using the crush parameter, you can boost everything above that value 1.

Parameters	Type	Defaults	Function
<i>red, green, blue, alpha</i>	float	0	Anything below this value goes to black.
<i>crush</i>	int	0	If this is set to 1, everything above the cut-off values goes to 1.
<i>softClip</i>	float	0	Provides a roll-off value.

## Synopsis

```
image Threshold(  
    image In,  
    float red,  
    float green,  
    float blue,  
    float alpha,  
    int crush,  
    float softClip  
);
```

## Script

```
image = Threshold(  
    In,  
    red,  
    green,  
    blue,  
    alpha,  
    crush,  
    softClip  
);
```

## Command Line

*shake -threshold red green blue alpha crush softClip*

## Utility Correctors

These tools are more applicable for preparing data for other operations.

### ColorSpace

The *ColorSpace* function converts an image from one color space to another color space. After the image is converted, you can use color correction functions that operate in the new color space logic for interesting effects. For example, if you place a *ColorSpace* node to convert from rgb to hls (hue, luminance, saturation), and then apply a *Color-Add* node, the *Add* node's red channel shifts your hue, instead of the red channel.

The optional r, g, bWeight arguments only affect rgb to hls, or hls to rgb conversions.

For command-line use, and compatibility with Shake version 2.1 scripts or earlier, use the dedicated space conversion functions *CMYToRGB*, *HLSToRGB*, *HSVToRGB*, *RGBToCMY*, *RGBToHLS*, *RGBToHSV*, *RGBToYIQ*, *RGBToYUV*, *YIQToRGB*, and *YUVToRGB*. These functions do not have arguments, with one exception: The HLS functions have optional r, g, and bWeight parameters.

Parameters	Type	Defaults	Function
<i>in/outSpace</i>	string	"rgb," "rgb"	Selects the incoming space and the output color space. For example, if you use one <i>ColorSpace</i> , you probably use rgb as your inSpace, and then something like hsv to convert it to hue/saturation/value space. After applying your operations, you usually apply a second <i>ColorSpace</i> function, with hsv as your inSpace and rgb as your outSpace.
<i>r/g/bWeight</i>	float	.3, .59, .11	The weighing of the three channels for the luminance calculation in conversions involving HSL. Luminance differs from value in that luminance calculates brightness based on the human eye's perception that green is brighter than an equal value in the blue channel.

**Synopsis**

```
image ColorSpace(  
    image,  
    string inSpace,  
    string outSpace,  
    float rWeight,  
    float gWeight,  
    float bWeight  
);
```

**Script**

```
image = ColorSpace(  
    image,  
    "inSpace",  
    "outSpace",  
    rWeight,  
    gWeight,  
    bWeight  
);
```

**Command Line**

```
shake -colorspace "inSpace" "outSpace"
```

## Examples

```
shake lisa.iff-colorspace "rgb" "b1s" -add .4 0 0 -colorspace "b1s" "rgb"
```

```
shake lisa.iff-rgbtob1s -add .4 0 0 -b1storgb
```

## ColorX

The *ColorX* function modifies each pixel in the image according to an expression that you supply. *ColorX* is normally slower than a dedicated, optimized function such as *Mult* or *Add*. Use dedicated operators whenever possible. You can also set up complex rules inside the function (see below).

## Synopsis

```
image ColorX( image,  
    float expression rExpr,  
    float expression gExpr,  
    float expression bExpr,  
    float expression aExpr,  
    float expression zExpr  
);
```

## Script

```
image = ColorX( image, rExpr, gExpr, bExpr, aExpr, zExpr);
```

For multi-line expressions,

```
image = ColorX( image,  
    {{expr1,expr2,expr3, rExpr}},  
    {{expr1,expr2,expr3, gExpr}},  
    bExpr,  
    aExpr,  
    zExpr  
);
```

## Command Line

```
shake -colorX "rExpr" "gExpr" "bExpr" etc...
```

Expressions can use the following variables:

- The variables r, g, b, a, and z refer to the value of the original channels (red, green, blue, alpha, and Z).
- The variables x and y are the coordinates of the pixel.
- The variables width and height are the width and height of the image.
- The variable time is the current frame number (time).

A large number of operators can be represented by an arithmetic expression, such as reordering, color correction, and gradient generation, or even circle drawing. Note that no spaces are allowed in the expressions, unless you can use quotes for more explicit grouping.

Task	Example: Each text field can be filled independently in the GUI—these are all command-line examples.
Reordering	<i>shake bg.iff -colorx b r a g</i>
Red correction	<i>shake bg.iff -colorx "r*1.2"</i>
Red and blue correction	<i>shake bg.iff -colorx "r*1.2" g "b*1.5"</i>
Same expression for red, green, and blue	<i>shake bg.iff -colorx "(r+g+b)/3" -reorder rrr</i>
X gradient in matte	<i>shake -colorx r g b "x/width"</i>
Y gradient in matte	<i>shake -colorx r g b "y/height"</i>
Blue spill removal	<i>shake primatte/woman.iff -colorx r g "b&gt;g?g:b"</i>
Random noise	<i>shake -colorx rnd(x*y) rnd(2*x*y) rnd(3*x*y)</i>
Turbulent noise (1 channel)	<i>shake -colorx turbulence2d(x,y,20,20)</i>
Clip alpha if Z is less than 20	<i>shake uboat.iff -colorx r g b "z&lt;20"</i>
Clip alpha if Z is more than 50	<i>shake uboat.iff -colorx r g b "z&gt;50"</i>
A smooth alpha gradient from Z units 1 to 70	<i>shake uboat.iff -colorx r g b "(z-1)/70"</i>

### See Also

“WarpX” on page 477, “LayerX” on page 182, “The TimeX Function” on page 82, “Expressions” on page 683

### LogLin

The *LogLin* function is typically used to handle logarithmic film plates, such as Cineon files from a film scanner, or when writing files out for film recording. It converts the images from the logarithmic color space to the linear color space for accurate compositing. You can then use the node at the end of your node tree to convert the images back into logarithmic space for film scanning. You can also use this around nodes that only work in 8 or 16 bits, functions such as *Primatte*, *Keylight*, or other plug-ins from third parties. *Ultimatte* is the exception, as it maintains float values. For a full description of this process, see “The Logarithmic Cineon File” on page 349.

The *LogLin* color parameters are linked together by default, so gBlack and bBlack reflect whatever the rBlack value. You can of course adjust these to further color correct your scanned plates.

**Note:** This node only does color correction—it does not change your bit depth or your file type. When Shake imports the Cineon files, a typically 10-bit file, it automatically promotes the files to 16 bits. This process has nothing to do with the color correction.

The default values are supplied by Kodak—if you apply a *LogLin* in Shake, you should get the same visual result as if you plugged in the same numbers into any other software package’s logarithmic converter. The range of the offset and black and white points are 0 to 1023, the range of a 10-bit file (8-bit is 0 to 255, 9-bit is 0 to 511). Every 90-point adjustment of these values is equivalent to a full f-stop of exposure.

Parameters	Type	Defaults	Function
<i>conversion</i>	int	0	This describes whether you convert from log to linear space, or linear to log space. 0 is log to lin, 1 is lin to log.
<i>offset</i>	float	0, 0, 0	These values offset the individual color channels.
<i>black, white</i>	float	90, 685	These set the black and white cutoff points. The default values are 95 and 685.
<i>nGamma</i>	float	.6	Generally, this number is not touched. The .6 is an average of the response curves, and may differ from stock to stock and even channel to channel. You can look it up on Kodak's Web site—see <i>products/Film/Motion Picture Film</i> , and then check the characteristic curves. Fun for everybody.
<i>dGamma</i>	float	1.7	The display gamma, according to Kodak, to compensate for the monitor lookup table. This was set to neutralize the Cineon system's standard monitor setting. Its inclusion here is more of a heritage thing. It is highly recommended that you leave it at 1.7.
<i>softClip</i>	float	0	The roll-off values on the white point. Default is 0, which gives a Linear break. By increasing this value, the curve is smoothed.

**Synopsis**

```
image LogLin(  
    image In,  
    int conversion,  
    float rOffset,
```

```

float gOffset,
float bOffset,
float rBlack,
float gBlack,
float bBlack,
float rWhite,
float gWhite,
float bWhite,
float rNGamma,
float gNGamma,
float bNGamma,
float rDGamma,
float gDGamma,
float bDGamma,
float rSoftClip,
float gSoftClip,
float bSoftClip
);

```

### Script

```

image = LogLin( image, conversion
    rOffset, gOffset, bOffset,
    rBlack, gBlack, bBlack,
    rWhite, gWhite, bWhite,
    rNGamma, gNGamma, bNGamma,
    rDGamma, gDGamma, bDGamma,
    rSoftClip, gSoftClip, bSoftClip
);

```

### Command Line

*shake -loglin conversion*

### See Also



“The Logarithmic Cineon File” on page 349



### Lookup

The *Lookup* function performs an arbitrary lookup on your image. It is extremely flexible, allowing you to mimic most other color correction nodes, and is generally much faster than the *ColorX* function.

For information regarding the curve editor in *Lookup*, *LookupHSV*, and *LookupHLS*, see “Loading and Viewing Curves in the Editor” on page 483.



The *Lookup* is defined as a function  $f(x)$ , where  $x$  represents the input color, ranging from 0 to 1. As you draw the graph of this function,  $x$  is on the X axis, and  $f(x)$  is on the Y axis. The following table lists the *Lookup* equivalents of other Shake color correction nodes.

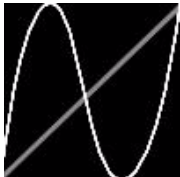

Shake Function	Brightness	Invert
Math Expression	$f(x) = x * \text{value}$	$f(x) = 1-x$
Lookup Expression	$x*1.5$	$1-x$
Graph (white is result, grey is input)		

Shake Function	Compress	Do Nothing
Math Expression	$f(x) = x * (\text{hi-lo}) + \text{lo}$	$f(x) = x$
Lookup Expression	"(x*.4)+0.3" (if lo = 0.3 and hi = 0.7)	"x"
Graph (white is result, grey is input)		

The following examples do custom lookups. The last two examples use Shake’s curve formats, but use the Value mode (the V at the end of the curve name), and input  $x$  as the value. All “keyframes” are between 0 and 1, and can take any value.

When using the interface, this is the default behavior—click the “load curve” icon in the Parameter View to load the curve into the Curve Editor.

Function	clipping	dampening
Lookup Expression	$x > .5 ? 0 : x$	$x * x$
Graph (white is result, grey is input)		

Function	Spline Lookup	Linear Lookup
Lookup Expression	$\text{CSplineV}(x, 0, 0 @ 0, 1 @ .25, 0 @ .75, 1 @ 1)$	$\text{LinearV}(x, 0, 0 @ 0, 1 @ .25, 0 @ .75, 1 @ 1)$
Graph (white is result, grey is input)		

Parameters	Type	Defaults	Function
$r, g, b, aExpr$	expression	"x", "x", "x", "x"	Use this function to change the input value, always represented by "x."

Synopsis

```
image Lookup( image,  
    float expression rExpr,  
    float expression gExpr,  
    float expression bExpr,  
    float expression aExpr  
);
```

## Script

```
image = Lookup(  
    image,  
    rExpr,  
    gExpr,  
    bExpr,  
    aExpr  
);
```

## Command Line

```
shake lookup "rExpr" "gExpr" "bExpr" "aExpr"
```

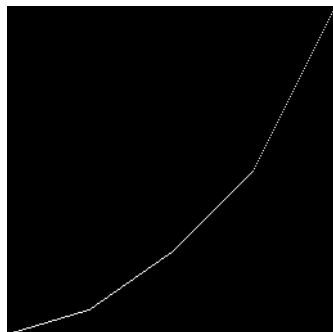
## See Also

“*LookupHLS*” on page 309, “*LookupHSV*” on page 310, “*ColorX*” on page 302

## LookupFile

Use the *LookupFile* function to apply a lookup table to any image by reading a text file. The file should consist of an arbitrary number of rows, and each row can have three or four entries, corresponding to red, green, blue, and possibly alpha. Shake determines the range of the lookup to apply based on the number of rows in the file—with “black” always mapping to 0 and “white” mapping to (n-1), where n is the number of lines in the file. Therefore, if your file contains 256 rows, Shake assumes that your entries are all normalized to be in the range of 0 (black) to 255 (white). If you have 1024 lines in your file, then “white” is considered to be a value of 1023. Interpolation between entries is linear, so lookups with only a few entries may show undesirable artifacts. For example, the following simple five-line lookup file produces the following lookup curve:

0	0	0	0
.3	.3	.3	.3
1	1	1	1
2	2	2	2
4	4	4	4



Because of this linear interpolation, you may want to instead use the standard *Lookup* node with lookups that do not have a large number of points.

Parameters	Type	Defaults	Function
<i>lookupFile</i>	string		The path to the lookup file.
<i>channels</i>	string	"rgba"	The channels that the lookup is applied.

**Synopsis**

```
image LookupFile( image,
    float expression lookupFile,
    float expression channels
);
```

**Script**

```
image = LookupFile(
    image,
    "lookupFile",
    "channels"
);
```

**Command Line**

```
shake -lookupfile lookupfile channels
```

**See Also**

“*Lookup*” on page 305, “*LookupHLS*” on page 309, “*LookupHSV*” on page 310, “*ColorX*” on page 302

**LookupHLS**

The *LookupHLS* function performs exactly like *Lookup*, except it works on the HLS channels instead of RGB channels.

Parameters	Type	Defaults	Function
<i>b, l, s, aExpr</i>	expression	"x"	Use this function to change the input value, always represented by "x."

**Synopsis**

```
image LookupHLS( image,
    float expression hExpr,
    float expression lExpr,
```

```
float expression sExpr,
float expression aExpr
);
```

### Script

```
image = Lookup( image, hExpr, lExpr, sExpr, aExpr);
```

### Command Line

```
shake -lookup "hExpr" "lExpr" "sExpr" "aExpr"
```

### See Also

“ColorX” on page 302, “Lookup” on page 305, “LookupHSV” on page 310

### LookupHSV

The *LookupHSV* function performs exactly like *Lookup*, except it works on the HSV channels instead of RGB channels.

Parameters	Type	Defaults	Function
<i>b, s, v, aExpr</i>	expression	"x"	Use this function to change the input value, always represented by "x."

### Synopsis

```
image LookupHSV( image,
float expression hExpr,
float expression sExpr,
float expression vExpr,
float expression aExpr
);
```

### Script

```
image = Lookup( image, hExpr, sExpr, vExpr, aExpr);
```

### Command Line

```
shake -lookup "hExpr" "sExpr" "vExpr" "aExpr"
```

### See Also

“LookupHLS” on page 309, “Lookup” on page 305, “ColorX” on page 302

### MDiv

The *MDiv* function divides the color channels by the alpha channel.

When you color correct a rendered (premultiplied) image, first apply an *MDiv* node to the image to make the image a non-premultiplied image, perform the color correction, and then add an *MMult* node to return the image to its premultiplied state. For more information on premultiplication, see “About Premultiplication and Compositing” on page 368.

Parameter	Type	Default	Function
<i>ignoreZero</i>	int	0	Tells Shake to ignore pixels with an alpha value of 0. 0 = Divide entire image. 1 = Ignore zero-value pixels.

#### Synopsis

```
image MDiv( image, int ignoreZero);
```

#### Script

```
image = MDiv( image, ignoreZero);
```

#### Command Line

*shake -mdiv ignorezero*

#### See Also

“*MMult*” on page 311, “*IDiv*” on page 174, “*IMult*” on page 175

### MMult

The *MMult* function multiplies the color channels by the matte.

This function is used to premultiply an image. When compositing with the *Over* node, Shake expects all images to be premultiplied. Premultiplication is usually automatic with 3D-rendered images, but not for scanned images. Also, use this to color correct a 3D-rendered image. Add an *MDiv* node to the image, perform your color corrections, and then add an *MMult* node into an *Over* operation.

**Note:** You can also choose to not insert a *MMult* node, and instead enable preMultiply in the *Over* node’s parameters.

For more information on premultiplication, see “About Premultiplication and Compositing” on page 368.

Parameters	Type	Defaults	Function
<i>ignoreZero</i>	int	0	Tells Shake to ignore pixels with an alpha value of 0. 0 = Multiplies entire image. 1 = Ignore zero-value pixels.

### Synopsis

```
image MMult( image, int ignoreZero);
```

### Script

```
image = MMult( image, ignoreZero);
```

### Command Line

*shake -mmult ignorezero*

### See Also

“MDiv” on page 311, “IDiv” on page 174, “IMult” on page 175

### Reorder

The *Reorder* function lets you shuffle channels. The argument to this command specifies the new order. A channel can be copied to several different channels. The letter “l” refers to the luminance pseudo-channel which can be substituted in place of the RGBA. If an expression is on a channel that does not exist, Shake creates the channel. You can use the Z channel as well. For example:

*shake -reorder zzzz*

places the Z channel into the RGBA channels for viewing.

To copy a channel from another image, use the *Copy* function.

Parameter	Type	Default	Function
<i>channels</i>	string	"rgba"	Indicates the new channel assignment. You can use any of the following:  r—Set the pixels of this channel to the values of the red channel.  g—Set the pixels of this channel to the values of the green channel.  b—Set the pixels of this channel to the values of the blue channel.  a—Set the pixels of this channel to the values of the alpha channel.  z—Set the pixels of this channel to the values of the Z channel.  l—Set the pixels of this channel to luminance of RGB.  0—Set the pixels of this channel to 0.  1—Set the pixels of this channel to 1.  n—Remove this channel from the active channels.

**Synopsis**

```
image Reorder( image, const char * channels );
```

**Script**

```
image = Reorder( image, "channels" );
```

**Command Line**

```
sbake -reorder channels
```

**Examples**

To copy the luminance into the matte channel:

```
-reorder rgbl
```

To copy the red channel to all three color channels and leave the alpha:

```
-reorder rrra
```

To remove the alpha channel:

```
-reorder rgbn
```

**See Also**

“Copy” on page 172, “ColorX” on page 302

**Set**

The *Set* function sets selected channels to a constant value. Alpha or depth channels can be added to images with *Set*. For example, *-set z .5* puts in a Z plane with a value of .5.

Parameters	Type	Defaults	Function
<i>channels</i>	string		The channels to be set.
<i>value</i>	float	1	The value of the channel.

**Synopsis**

```
image Set( image,  
    const char * channels,  
    float value  
);
```

**Script**

```
image = Set( image, "channels", value );
```

**Command Line**

```
sbake -set channels value
```

**See Also**

“*Reorder*” on page 312, “*ColorX*” on page 302

**SetAlpha**

The *SetAlpha* function is simply a macro for *Set* and *Crop* that sets the alpha to 1 by default. It exists because in the Infinite Workspace, color corrections extend beyond the frame of an image. By using the *Crop* in the macro, the *Set* function is cut off, making the image ready for transformations.

To remove the alpha channel from an image (turn an RGBA image into an RGB image), set the alpha value to 0.

Parameter	Type	Default	Function
<i>alpha</i>	float	1	From 0 to 1, this is the alpha value of the output image. By default, it is 1.

**Synopsis**

```
image SetAlpha( image, float alpha );
```

**Script**

```
image = SetAlpha( image, alpha );
```

**Command Line**

```
sbake -setalpha alpha
```

**See Also**

“*Set*” on page 313, “*Crop*” on page 453

**SetBGColor**

The *SetBGColor* function sets selected channels to the selected color outside of the Domain of Definition (DOD). For example, if you create an Image-*RGrad*, the green DOD box appears around the *RGrad* image in the Viewer. Everything outside of the DOD is understood to be black, and therefore does not have to be computed. To change the area outside of the DOD, attach a *SetBGColor* to the node and change the color.

This function is often used for adjusting keys, as the keyer may be pulling a blue screen, and therefore assigns the area outside of the DOD, which is black, as an opaque foreground. If the element is scaled down and composited, you do not see the background. To correct this, insert a *SetBGColor* before the keyed element is placed in the composite, for example, *ChromaKey > SetBGColor > Scale > Over*.

Parameters	Type	Defaults	Function
<i>mask</i>	string	"rgbaz"	The channels that are reset.
<i>red, green, blue, alpha, depth</i>	float	0, 0, 0, 0, 0	The values of the associated channel.

**Synopsis**

```
image Set(  
    image In,  
    const char * mask,  
    float red,  
    float green,  
    float blue,  
    float alpha,  
    float depth  
);
```

**Script**

```
image = Set(  
    In,  
    "mask",  
    red,  
    green,  
    blue,  
    alpha,  
    depth,  
);
```

**Command Line**

```
shake -set channels value
```

**VideoSafe**

For information on the *VideoSafe* function, see “*VideoSafe*” on page 225.

## Consolidated Color Correctors

The consolidated color corrector nodes are more complex than the other nodes. They are usually inappropriate for use on the command line, and have unique interfaces.

### AdjustHSV

The *AdjustHSV* function takes a specified color, described by its HSV values, and offsets the color. For example, you can take a red spaceship and turn it blue without affecting the green alien on it. It works similarly to the *ChromaKey* node.

To change a color, scrub with the Color Picker to isolate its hue, saturation, and value. Next, scrub with the target Color Picker. For example, if you have a hue of 0 (red), and enter a hueOffset of .66, you slide it to blue. The Range, Falloff, and Sharpness sliders help control how much of a range you capture.

Parameters	Type	Defaults	Function
<i>bue, sat, val</i>	float	0, 0, 0	These describe the HSV values of the target color to change.
<i>Offset</i>	float	0, 0, 0	This is what is added to the hue, sat, and val parameters, thereby changing the color.
<i>Range</i>	float	1, 1, 1	This is the range that is added to the HSV values to include a wider field of values.
<i>Falloff</i>	float	0, 0, 0	This describes the falloff range outside of Range.
<i>Sharpness</i>	float	1.5, 1.5, 1.5	This describes the drop-off curve of Falloff. 0 = linear drop-off 1.5 = smooth drop-off

### Synopsis

```
image AdjustHSV( image,  
    float hue,  
    float hueOffset,  
    float hueRange,  
    float hueFalloff,  
    float hueSharpness,  
    float sat,  
    float satOffset,  
    float satRange,  
    float satFalloff,  
    float satSharpness,  
    float val,
```

```

float valOffset,
float valRange,
float valFalloff,
float valSharpness
);

```

### Script

```

image = AdjustHSV( image,
    hue, hueOffset, hueRange,
    hueFalloff, hueSharpness,
    sat, satOffset, satRange,
    satFalloff, satSharpness,
    val, valOffset, valRange,
    valFalloff, valSharpness
);

```

### Command Line

*shake -adjusthsv hue hueOffset hueRange etc...*

### See Also

“ColorX” on page 302, “Lookup” on page 305, “LookupHLS” on page 309, “LookupHSV” on page 310

### ColorCorrect

The *ColorCorrect* function combines *Add*, *Mult*, *Gamma*, *ContrastRGB*, *ColorReplace*, *Invert*, *Reorder*, and *Lookup* in one node, and also gives you the ability to tune the image in only the shadow, midtone, or highlight areas.

**Note:** The *ColorCorrect* node breaks concatenation with connecting color correction nodes.

### The ColorCorrect Subtabs

The following table describes the *ColorCorrect* Parameter subtabs.

Subtab	Function
Master	Applies the same correction to the entire image.
Low Controls	Applies the correction primarily to the darkest portion of the image; the correction falls off as the image gets brighter.
Mid Controls	Applies the correction primarily to the middle range of the image.
High Controls	Applies the correction primarily to the highlights of the image; the correction falls off as the image gets darker.

Subtab	Function
Curves	Manual correction of the image using curves.
Misc	Secondary color correction, as well as invert, reorder, and premultiplication control.
Range Curves	Display of the different image ranges (shadows, midtones, highlights), their control curves, and the final concatenated curve of the color correction.

**Note:** You can only view one subtab at a time.

### The Master, Low, Mid, and High Color Controls Tabs

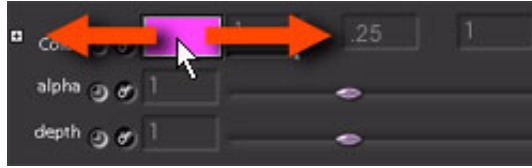
The first four color control tabs (Master, Low, Mid, and High Controls) are identical except for the portion of the image you are modifying. Each has the same matrix to Add, Multiply (Gain), and apply a Gamma to the RGB channels, as well as apply contrast on a per-channel basis (Contrast, Center, SoftClip).



When tuning the color with the matrix, you can choose from several methods:

- Numerically enter a value in the RGB text fields.
- Use the slider below each text field.
- Click the Color Picker box, and select a color from the Viewer or the ColorWheel (in the Color Picker).

- Use the Virtual Color Picker. Press the relative key, **R** (red), **G** (green), **B** (blue), **H** (hue), **S** (saturation), **V** (value), **L** (luminance), **M** (magenta), or **T** (temperature), and drag left or right on the parameter line. (You can do this anywhere on the line, not just over the Color Picker.)



- To group the R, G, and B sliders, press **V** (value) and drag left or right.

As an example, press **R** and drag right over the Add line. This adds to the red channel. When the color box is bright red, press **H** and drag left and right. The hue of the color shifts. This technique only modifies the color that is Added, Multiplied, etc. So, dragging a color box while pressing **S** (saturation) does not decrease the saturation of the image, but only the saturation of the color that you are adding (or multiplying, etc.) to the image.

The bottom portion of the tab contains buttons to toggle the channels from RGB display to a different color space model. You can display RGB, HSV, HLS, CMY, and TMV. For example, if the current image is displayed in the RGB color model, click HSV and the numbers are converted to HSV space. Notice the Add color does not change—only the numerical value.



**Note:** The “TMV” color space is Temperature/Magenta-Cyan/Value.

When the *ColorCorrect* function is saved into a script, the values are always stored in RGB space.

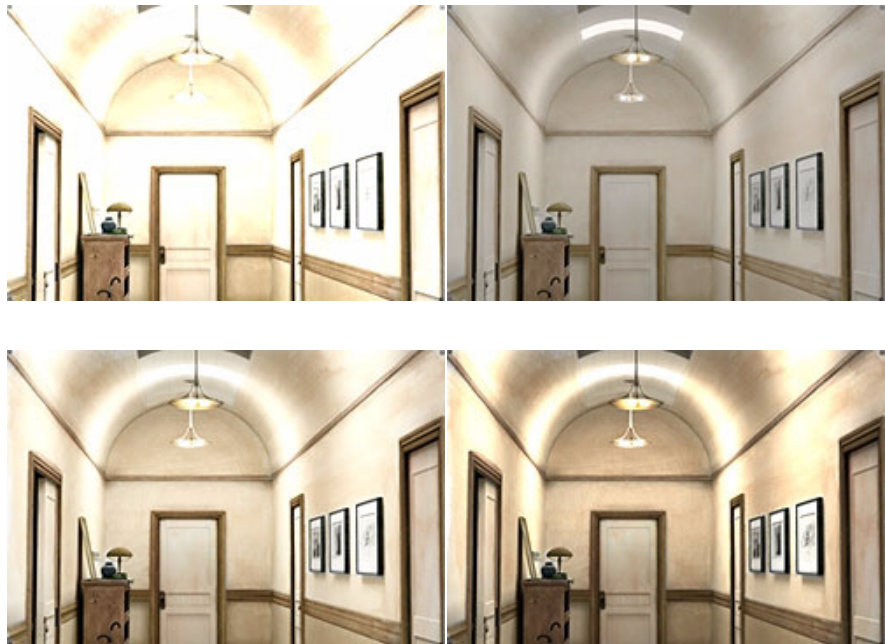
Each color picker has an Autokey and View Curves button associated with all three channels for that parameter. All three channels are treated equally.

### Working With Low, Mid, and High Ranges

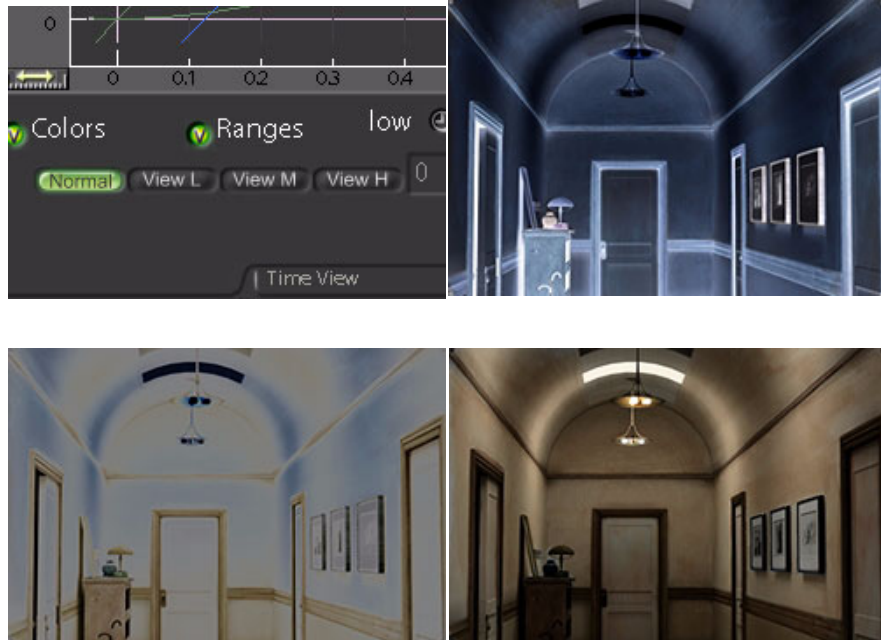
The following section discusses the differences in working with low, mid, and high color ranges. The first image is the original image (a hearty “thank you” to Skippingstone for the use of images from their short film *Doppelganger*).



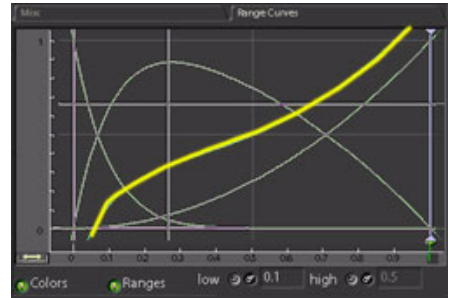
In the following examples, a Gain (multiply) of 2 is applied to each channel. The first example multiplies all pixels by 2. The pure blacks stay black, the whites flare out. However, when the gain is applied to the Low areas (the shadows), although the pure blacks stay black, the areas just above 0 are raised into the mid range, and this reduces the apparent contrast. A higher value solarizes the image. In the following images, a gain of 2 is applied in the Master tab, the Low, Mid, and the High tab.



You can control the range of the image that is considered to be in the shadows, midtones, and highlights in the Range Curves subtab. This tab displays your final color lookup operator as a curve, your mask ranges (to turn on the display, click the Ranges button at the bottom), and controls for the center of the low and high curves. Also, you can toggle the output from the Normal, corrected image to a display of the Low areas boosted to 1 and all else black; the Mid areas boosted to 1 and all else black; or the High areas boosted to 1 and all else black. A colored display is used, rather than a display based on luminance, since different channels have different values. In the following, the range viewer controls, with Low, Mid, and High, are selected.



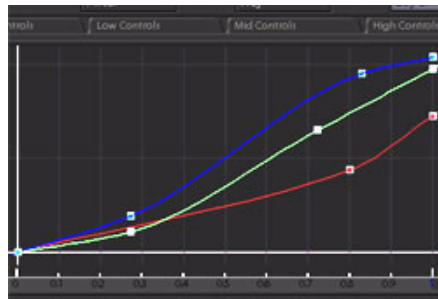
To control the mask areas, turn on the Ranges curve display at the bottom of the Range Curves tab. The left image below shows the default ranges. A curve of the final lookup is displayed in this illustration as a yellow line for clarity. Notice that the Low and High range curves' (grey curves sloping in from left and right) centers are set at .5. If you adjust the low or high values, you modify that range, as well as the mid-range curve. For example, the second image shows what happens when low is set down to .1. Notice the Low and Mid curves shift left, but the High curve remains unaffected.



### The Curves Tab

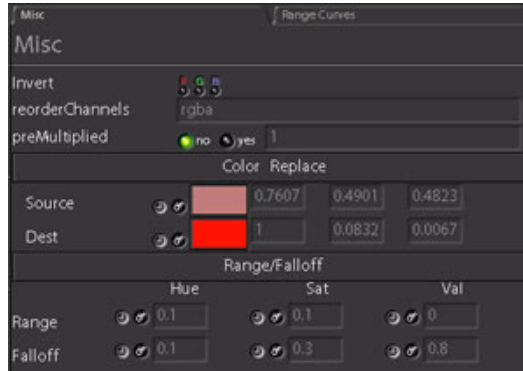
The Curves tab allows you to apply manual modifications to the lookup curve. Although this is generally used for manual adjustments, you can also apply functions using the standard Lookup expressions.

**Note:** To insert a new control point, **Shift**-click on a segment of the curve.



## The Misc Tab

The Misc tab contains several functions.



- **Invert**  
Invert uses the formula  $1-x$ , so float values may have odd results.
- **reorderChannels**  
Enter a string to swap or remove your channels as per the standard *Reorder* method.  
For more information, see “*Reorder*” on page 312.
- **preMultiplied**  
Enable preMultiplied if your image is premultiplied (typically, an image from a 3D render), and an *MDiv* is automatically inserted before the calculations, and an *MMult* is automatically added after the calculations.  
For more information on premultiplication, see “About Premultiplication and Compositing” on page 368.

■ Color Replace

Use the Color Replace tools for secondary color correction, as per the *ColorReplace* node. In this example, the skin tone of the character is selected and replaced with red. Sat Falloff is set to .3, Val Range to 0, and Val Falloff to .8.



**Order of Calculations**

Calculations are made in the following order:

- MDiv (optional)
- ColorReplace
- Invert
- Lookup Curves
- Gamma
- Mult
- Add
- Contrast
- Reorder
- MMult (optional)

**Parameters**

The following table lists the *ColorCorrect* parameters. Not all parameters are exposed in the interface.

Parameters	Type	Defaults	Function
<i>version</i>	string	"v1.0"	A placeholder to ensure compatibility in later versions of Shake. Does not affect the image.
<i>r, g, b, aExpr</i>	string	"x"	Lookup expressions as per the <i>Lookup</i> node.

Parameters	Type	Defaults	Function
<i>r, g, b, aContrast</i>	float	1	Global contrast values on a per-channel basis.
<i>r, g, b, aCenter</i>	float	.5	Global centers of the contrast curve. Shift it down to increase the brightness of the curve. Shift it above .5 to increase the darkness of the image. This only has effect if a contrast value other than 1 has been entered.
<i>r, g, b, aSoftClip</i>	float	0	Global roll-off value for the contrast curve.
<i>r, g, b, aGamma</i>	float	1	Global gamma values. 1 is no change.
<i>r, g, b, aGain</i>	float	1	A Global multiplier on the color.
<i>r, g, b, aAdd</i>	float	0	Global values added to the color channel.
<i>r, g, b, alContrast</i>	float	1	Low contrast values.
<i>r, g, b, alCenter</i>	float	.5	Low contrast center.
<i>r, g, b, alSoftClip</i>	float	0	Low softclip value.
<i>r, g, b, alGamma</i>	float	1	Low gamma values.
<i>r, g, b, alGain</i>	float	1	Low multiplier values.
<i>r, g, b, alAdd</i>	float	0	Low offset values.
<i>r, g, b, amContrast</i>	float	1	Medium contrast values.
<i>r, g, b, amCenter</i>	float	.5	Medium contrast center.
<i>r, g, b, amSoftClip</i>	float	0	Medium softclip value.
<i>r, g, b, amGamma</i>	float	1	Medium gamma values.
<i>r, g, b, amGain</i>	float	1	Medium multiplier values.
<i>r, g, b, amAdd</i>	float	0	Medium offset values.
<i>r, g, b, abContrast</i>	float	1	High contrast values.

Parameters	Type	Defaults	Function
<i>r, g, b, abCenter</i>	float	.5	High contrast center.
<i>r, g, b, abSoftClip</i>	float	0	High softclip value.
<i>r, g, b, abGamma</i>	float	1	High gamma values.
<i>r, g, b, abGain</i>	float	1	High multiplier values.
<i>r, g, b, abAdd</i>	float	0	High offset values.
<i>r, g, b, abInvert</i>	int	0	Inversion of the channels, for example, 1-x.
<i>low, highWeight</i>	float	.5	The center points of the two mask curves. Shifting lowWeight down from .5 means that less range is considered to be in the Low range. lowWeight adjustments do not affect the High area of the image.
<i>tOrder</i>	string	"GMAC"	The Gamma/Mult/Add/Contrast calculation order.
<i>affectAlpha</i>	int	0	ColorReplace's affectAlpha parameter. Not in the interface.
<i>r, g, bSource</i>	float	0	The source color for replacement.
<i>r, g, bReplace</i>	float	r, g, bSource	The color to replace r, g, bSource.
<i>low, rolllow</i>	float	.1	The hue range and falloff for replacement.
<i>high, rollhigh</i>	float	.1	The saturation range and falloff for replacement.
<i>lum, rolllum</i>	float	1	The luminance range and falloff for replacement.
<i>channels</i>	string	"rgba"	The reorder channels.
<i>preMult</i>	int	1	Whether the input image is premultiplied. 1 indicates that it is not premultiplied. 2 indicates that it is premultiplied.
<i>viewRange</i>	int	0	The output of the node: 0 = Normal mode. 1 = The image low-range is viewed, regardless of correction. 2 = The image mid-range is viewed. 3 = The image high range is viewed.

## Synopsis

```
image ColorCorrect(  
    image in,  
    const char* version = "v1.0",  
    float rExpr="x",  
    float gExpr="x",  
    float bExpr="x",  
    float aExpr="x",  
    float rContrast=1.0,  
    float gContrast=1.0,  
    float bContrast=1.0,  
    float aContrast=1.0,  
    float rCenter=0.5,  
    float gCenter=0.5,  
    float bCenter=0.5,  
    float aCenter=0.5,  
    float rSoftClip=0.0,  
    float gSoftClip=0.0,  
    float bSoftClip=0.0,  
    float aSoftClip=0.0,  
    float rGamma=1.0,  
    float gGamma=1.0,  
    float bGamma=1.0,  
    float aGamma=1.0,  
    float rGain=1.0,  
    float gGain=1.0,  
    float bGain=1.0,  
    float aGain=1.0,  
    float rAdd=0.0,  
    float gAdd=0.0,  
    float bAdd=0.0,  
    float aAdd=0.0,  
    float rlContrast=1.0,  
    float glContrast=1.0,  
    float blContrast=1.0,  
    float alContrast=1.0,  
    float rlCenter=0.5,  
    float glCenter=0.5,  
    float blCenter=0.5,  
    float alCenter=0.5,  
    float rlSoftClip=0.0,  
    float glSoftClip=0.0,
```

```
float blSoftClip=0.0,  
float alSoftClip=0.0,  
float rlGamma=1.0,  
float glGamma=1.0,  
float blGamma=1.0,  
float alGamma=1.0,  
float rlGain=1.0,  
float glGain=1.0,  
float blGain=1.0,  
float alGain=1.0,  
float rlAdd=0.0,  
float glAdd=0.0,  
float blAdd=0.0,  
float alAdd=0.0,  
float rmContrast=1.0,  
float gmContrast=1.0,  
float bmContrast=1.0,  
float amContrast=1.0,  
float rmCenter=0.5,  
float gmCenter=0.5,  
float bmCenter=0.5,  
float amCenter=0.5,  
float rmSoftClip=0.0,  
float gmSoftClip=0.0,  
float bmSoftClip=0.0,  
float amSoftClip=0.0,  
float rmGamma=1.0,  
float gmGamma=1.0,  
float bmGamma=1.0,  
float amGamma=1.0,  
float rmGain=1.0,  
float gmGain=1.0,  
float bmGain=1.0,  
float amGain=1.0,  
float rmAdd=0.0,  
float gmAdd=0.0,  
float bmAdd=0.0,  
float amAdd=0.0,  
float rhContrast=1.0,  
float ghContrast=1.0,  
float bhContrast=1.0,  
float ahContrast=1.0,
```

```

float rhCenter=0.5,
float ghCenter=0.5,
float bhCenter=0.5,
float ahCenter=0.5,
float rhSoftClip=0.0,
float ghSoftClip=0.0,
float bhSoftClip=0.0,
float ahSoftClip=0.0,
float rhGamma=1.0,
float ghGamma=1.0,
float bhGamma=1.0,
float ahGamma=1.0,
float rhGain=1.0,
float ghGain=1.0,
float bhGain=1.0,
float ahGain=1.0,
float rhAdd=0.0,
float ghAdd=0.0,
float bhAdd=0.0,
float ahAdd=0.0,
int rInvert=0,
int gInvert=0,
int bInvert=0,
int aInvert=0,
float lowWeight=0.5,
float highWeight=0.5,
const char *tOrder = "GMAC",
int affectAlpha = 0,
float rSource = 0,
float gSource = 0,
float bSource = 0,
float rReplace = "rSource",
float gReplace = "gSource",
float bReplace = "bSource",
float low = 0.1,
float rolllow = 0.1,
float high = 0.1,
float rollhigh = 0.1,
float lum = 1,
float rolllum = 1,
const char *channels = "rgba",
int preMult=1,

```

```
    int viewRange=0  
);
```

### **Script**

```
image ColorCorrect(  
    in,  
    "version",  
    "rExpr",  
    "gExpr",  
    "bExpr",  
    "aExpr",  
    rContrast,  
    gContrast,  
    bContrast,  
    aContrast,  
    rCenter,  
    gCenter,  
    bCenter,  
    aCenter,  
    rSoftClip,  
    gSoftClip,  
    bSoftClip,  
    aSoftClip,  
    rGamma,  
    gGamma,  
    bGamma,  
    aGamma,  
    rGain,  
    gGain,  
    bGain,  
    aGain,  
    rAdd,  
    gAdd,  
    bAdd,  
    aAdd,  
    rlContrast,  
    glContrast,  
    blContrast,  
    alContrast,  
    rlCenter,  
    glCenter,  
    blCenter,
```

alCenter,  
rlSoftClip,  
glSoftClip,  
blSoftClip,  
alSoftClip,  
rlGamma,  
glGamma,  
blGamma,  
alGamma,  
rlGain,  
glGain,  
blGain,  
alGain,  
rlAdd,  
glAdd,  
blAdd,  
alAdd,  
rmContrast,  
gmContrast,  
bmContrast,  
amContrast,  
rmCenter,  
gmCenter,  
bmCenter,  
amCenter,  
rmSoftClip,  
gmSoftClip,  
bmSoftClip,  
amSoftClip,  
rmGamma,  
gmGamma,  
bmGamma,  
amGamma,  
rmGain=,  
gmGain,  
bmGain,  
amGain,  
rmAdd,  
gmAdd,  
bmAdd,  
amAdd,  
rhContrast,

ghContrast,  
bhContrast,  
ahContrast,  
rhCenter,  
ghCenter,  
bhCenter,  
ahCenter,  
rhSoftClip,  
ghSoftClip,  
bhSoftClip,  
ahSoftClip,  
rhGamma,  
ghGamma,  
bhGamma,  
ahGamma,  
rhGain,  
ghGain,  
bhGain,  
ahGain,  
rhAdd,  
ghAdd,  
bhAdd,  
ahAdd,  
rInvert,  
gInvert,  
bInvert,  
aInvert,  
lowWeight,  
highWeight,  
"tOrder",  
affectAlpha,  
rSource,  
gSource,  
bSource,  
"rReplace",  
"gReplace",  
"bReplace",  
low,  
rolllow,  
high,  
rollhigh,  
lum,

```

    rolllum,
    "channels",
    preMult,
    viewRange
);

```

### Command Line

Yeah, right.

### ColorMatch

The *ColorMatch* function allows you to take an old set of colors (source color) in an image and match them to a new set (destination color). You can match the low, middle, and high end of the image. You can also perform Contrast, Gamma, and Add color corrections with Gamma as an inverse gamma to preserve highlights.

When you match color and use the Color Pickers, be aware of where you scrub from. If you color correct an image and then feed it into the comp, you may have to jump down to view the color-corrected image to get proper source color; otherwise you pull in modified color. This occurs after you have already fed in the destination color, since they are linked to the source color. Therefore, a good workflow is to select all three source colors, and then select the destination colors. Another scrubbing technique is to ignore the node when scrubbing (select the node and press **I** in the Node View), and then enable the node again when finished.

Parameters	Type	Defaults	Function
<i>r, g, bLowS</i>	float	0, 0, 0	The low end of the RGB of the source color.
<i>r, g, bLowD</i>	float	<i>r, g, bLowS</i>	The low end of the RGB destination color.
<i>r, g, bMidS</i>	float	.5, .5, .5	The middle of the RGB of the source color.
<i>r, g, bMidD</i>	float	<i>r, g, bMidS</i>	The middle of the RGB destination color.
<i>r, g, bHighS</i>	float	1, 1, 1	The high end of the RGB of the source color.
<i>r, g, bHighD</i>	float	<i>r, g, bHighS</i>	The high end of the RGB destination color.
<i>r, g, bContrast</i>	float	1, 1, 1	Contrast values for the three channels.
<i>r, g, bGamma</i>	float	1, 1, 1	The Gamma values. Note this is an inverse gamma function, so you retain your highlights as you raise the gamma.
<i>r, g, bAdd</i>	float	0, 0, 0	Adds color. Blacks are modified when this is raised.
<i>min, max</i>	float	0, 1	Sets the clipping for the function.

## Synopsis

```
image ColorMatch(  
    image In,  
    float rLowS,  
    float gLowS,  
    float bLowS,  
    float rLowD,  
    float gLowD,  
    float bLowD,  
    float rMidS,  
    float gMidS,  
    float bMidS,  
    float rMidD,  
    float gMidD,  
    float bMidD,  
    float rHighS,  
    float gHighS,  
    float bHighS,  
    float rHighD,  
    float gHighD,  
    float bHighD,  
    float rContrast,  
    float gContrast,  
    float bContrast,  
    float rGamma,  
    float gGamma,  
    float bGamma,  
    float rAdd,  
    float gAdd,  
    float bAdd,  
    float min,  
    float max  
);
```

## Script

```
image ColorMatch(  
    In,  
    rLowS,  
    gLowS,  
    bLowS,  
    rLowD,  
    gLowD,
```

```

    bLowD,
    rMidS,
    gMidS,
    bMidS,
    rMidD,
    gMidD,
    bMidD,
    rHighS,
    gHighS,
    bHighS,
    rHighD,
    gHighD,
    bHighD,
    rContrast,
    gContrast,
    bConstrast,
    rGamma,
    gGamma,
    bGamma,
    rAdd,
    gAdd,
    bAdd,
    min,
    max
);

```

### See Also

“*AdjustHSV*” on page 316, “*Lookup*” on page 305, “*ColorReplace*” on page 336

### ColorReplace

The *ColorReplace* function allows you to isolate a color according to its hue, saturation, and value, and then replace it with a different color. Other areas of the spectrum remain unchanged. This is especially useful for spill suppression.

To pull a mask of the affected source color, enable `affectAlpha` in the *ColorReplace* parameters. To better understand the parameters, you can attach the *ColorReplace* node to a *ColorWheel* node to observe the effects graphically.

Parameters	Type	Defaults	Function
<i>affectAlpha</i>	int	0	Toggles whether the alpha color is also adjusted by the color correction. If so, you can then easily use this as a mask for other operations.
<i>r, g, bSource</i>	float	0, 0, 0	The Source color to be replaced.
<i>r, g, bReplace</i>	float	r, g, bSource	The new color that is inserted instead of the Source color.
<i>bueRange</i>	float	.1	The range on the hue (from 0 to 1) that is affected. A range of .5 affects the entire hue range.
<i>bueFalloff</i>	float	.1	The falloff to 0 from the hueRange that is also pulled.
<i>satRange</i>	float	.1	The range of the saturation from the Source color; 1 is the entire range.
<i>satFalloff</i>	float	.1	The falloff from the satRange to 0.
<i>valRange</i>	float	1	The value range, with 1 as the entire range.
<i>valFalloff</i>	float	1	The falloff of the value to 0.

**Synopsis**

```
image ColorReplace(  
    image In,  
    int affectAlpha,  
    float rSource,  
    float gSource,  
    float bSource,  
    float rReplace,  
    float gReplace,  
    float bReplace,  
    float hueRange,  
    float hueFalloff,  
    float satRange,  
    float satFalloff,  
    float valRange,  
    float valFalloff  
);
```

## Script

```
image ColorReplace(  
    In,  
    affectAlpha,  
    rSource,  
    gSource,  
    bSource,  
    rReplace,  
    gReplace,  
    bReplace,  
    hueRange,  
    hueFalloff,  
    satRange,  
    satFalloff,  
    valRange,  
    valFalloff  
);
```

## See Also

*AdjustHSV* on page 316, *Lookup* on page 305, *ColorMatch* on page 334, *SpillSuppress* on page 271, *Keylight (Plug-in)* on page 258

## HueCurves

The *HueCurves* function allows you to perform various color corrections (Add, Saturation, Brightness) on isolated hues through the use of the Curve Editor. Typically, this tool is used for spill suppression, but you can also do color corrections once you understand how it is used.

### To color correct with HueCurves:

- 1 Find the hue of the area you want to color correct with the Color Picker. For example, if you have blue spill, the hue is approximately .66.
- 2 Load the parameter you want to use into the Curve Editor. For example, to load saturation into the Curve Editor, click the button to the left of “saturation” in the parameter’s name list.

When a parameter is loaded, the button is highlighted, and the curve appears in the Curve Editor.

- 3 Drag the control point near the hue (.66) down.

The saturation is decreased in that particular hue, turning the pure blues to grey.

By default, all curves have a value of 1 until you modify the value downward. Additionally, be careful with red-hued targets, as you may have to drag both the first and last control point on the curve.

Parameters	Type	Defaults	Function
<i>r, g, bSuppress</i>	curve float	N/A	Removes red, green, or blue from the hue area you identify when you drag the control point downward.
<i>r, g, bHue</i>	curve float	N/A	Adds red, green, or blue to the hue range you identify.
<i>saturation</i>	curve float	N/A	Removes saturation from the hue range you identify.
<i>luminance</i>	curve float	N/A	Removes luminance from your area.
<i>satLimit</i>		N/A	Sets the limit for saturation values.

**Synopsis**

```
image HueCurves(  
    image In,  
    curve float rSuppress,  
    curve float gSuppress,  
    curve float bSuppress,  
    curve float rHue,  
    curve float gHue,  
    curve float bHue,  
    curve float saturation,  
    curve float luminance,  
    curve float satLimit  
);
```

**Script**

```
image = HueCurves(  
    In,  
    rSuppress,  
    gSuppress,  
    bSuppress,  
    rHue,  
    gHue,  
    bHue,  
    saturation,
```

```

    luminance,
    satLimit
);

```

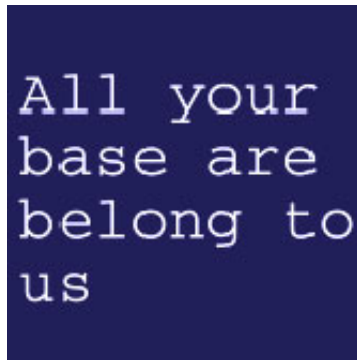
### Command Line

Not appropriate for command line.

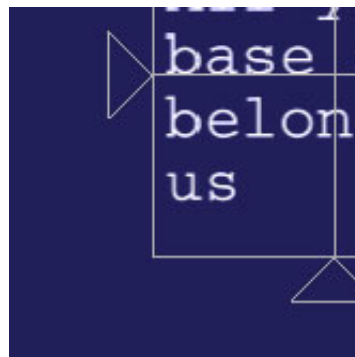
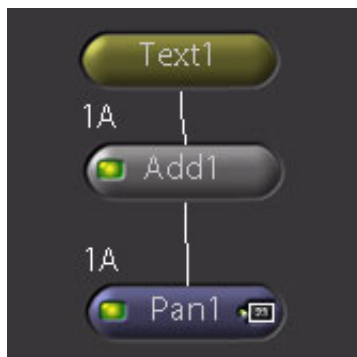
## Color Correction With the Infinite Workspace

Because the Shake engine applies effects to a potentially infinite canvas, unexpected results may occur when you color correct and pan a small element across a larger element. This occurs when an *Invert*, *Set*, or *Add* node is applied to an image, since the previously black pixels outside of the frame are changed to a different color.

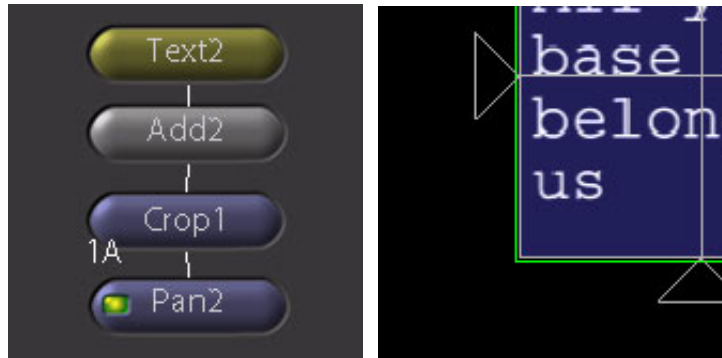
In the following example, a fine expression of fin de siecle angst is created with a *Text* node. The default black background color is raised to a blue with a Color-*Add* node.



When the image is panned, it is blue in the areas that were previously outside of the frame.



For a black outline, insert a *Crop* node before the *Pan* in the node tree:



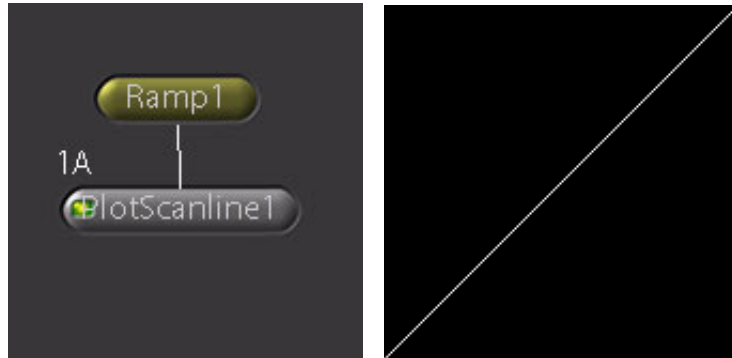
To color correct the area outside of the Domain of Definition (DOD, represented by the green box), use the *SetBGColor* node.



For more information on the Infinite Workspace, see “About the Infinite Workspace” on page 150. For more information on the DOD, see “The Domain of Definition (DOD)” on page 50.

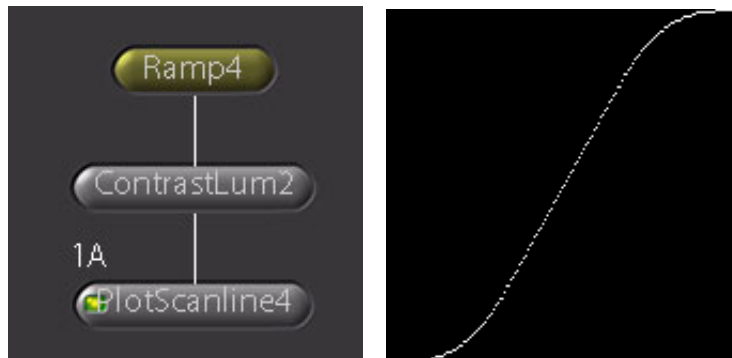
## Using the PlotScanline to Understand Color Correction Functions

To better understand some of the Shake color-correction nodes, use the Other—*PlotScanline* node or the Plotscanline Viewer Script. The *PlotScanline* node, located in the Other Tool Tab, looks at a single horizontal scanline of an image and plots the brightness value of a pixel for each X location. The most basic example of this is shown in the following illustration. The example begins with a simple horizontal gradient source image that varies linearly from 0 to 1. The *PlotScanline* resolution is set to 256 x 256 (for an 8-bit image).



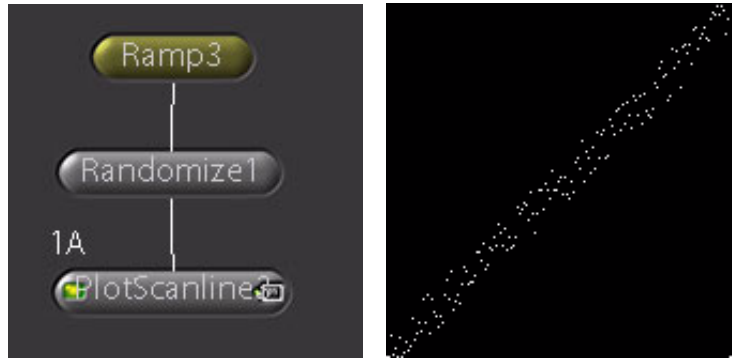
The ramp ranges from black (on the left) to white (on the right). This is reflected in the graph as a linear line.



When a node such as *ContrastLum* is inserted above the *PlotScanline* node, you can begin to understand the node. In the *ContrastLum* node, value is set to 1.5 and the center and softClip parameters are adjusted.



The effect on the ramp is reflected in the plot.

This also works for non-color correctors, and makes it an interesting analysis tool for the *Filter–Grain* or *Warp–Randomize* nodes.



You can also use the *PlotScanline*  and *Histogram ViewerScripts*  to observe image data, but these are applied directly on your image. To load the parameters, right-click the Viewer Script button and select a *Load Viewer Script Controls* option.

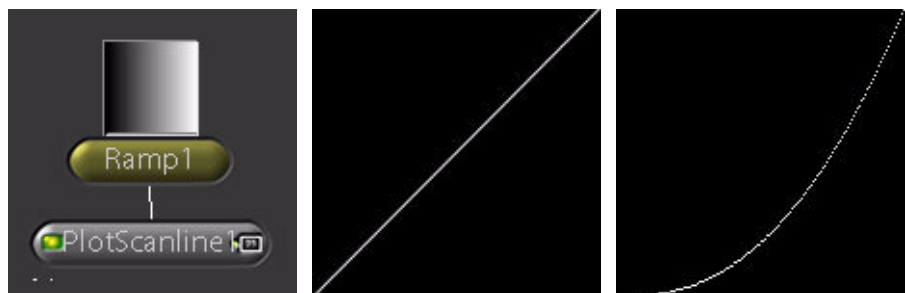
### PlotScanline

The *PlotScanline* function is an analysis tool that examines a line of an image and graphs the intensity of each channel per X position. It greatly helps to determine what a color correction function is doing. Although it can be attached to any image for analysis, it is often attached to a horizontal *Ramp* to observe the behavior of a color correction. Switch the Viewer to view the alpha channel (press **A** in the Viewer), and see the behavior of the alpha channel.

The following are some examples of using the *PlotScanline* function.

#### Example 1

A 256 x 256 8-bit black-and-white *Ramp*. Since there is a smooth gradation, with the center at .5, it is a straight line. Moving the center to .75 pushes the center to the right, making the entire image darker. This is reflected in the *PlotScanline*.



### Example 2

In this example, some color correction nodes are inserted and the values are adjusted. The *PlotScanline* indicates exactly what data gets clipped, crunched, compressed, or confused.



Parameters	Type	Defaults	Function
<i>width, height</i>	int	width, 512	The width and height of the PlotScanline. You likely want to set the width to 256 on an 8-bit image to get 1-to-1 correspondence.
<i>line</i>	int	0	The Y-line of the image to be analyzed. On a horizontal ramp, this does not matter, as they are all identical.

### Synopsis

```
image PlotScanline( image, int width, int height, int line );
```

### Script

```
image = PlotScanline(image, width, height, line);
```

### Command Line

*shake -plotscanline width height line*

### Histogram

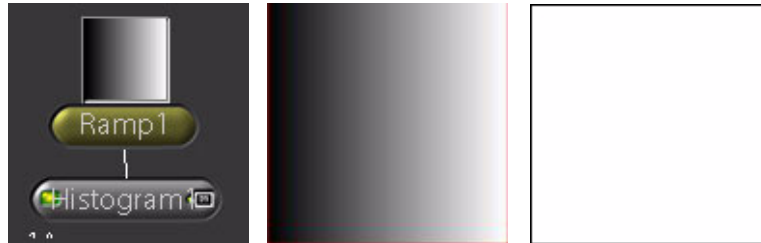
The *Histogram* function is an analysis tool that examines an image and graphs the occurrence per channel of each value. The X axis of the *Histogram* corresponds to the numerical value of a pixel. The Y axis is the percentage of pixels per channel with that value. The graph has nothing to do with the input pixel's original X or Y position in the image.

**Note:** You can also apply a *Histogram* or *PlotScanline* using the Viewer Scripts.

The following are some examples:

### Example 1

A 256 x 256 8-bit black-and-white *Ramp*. Since there are equal amounts of the entire range of pixels, the result is a solid white field. The orientation of the ramp has no bearing on the graph.



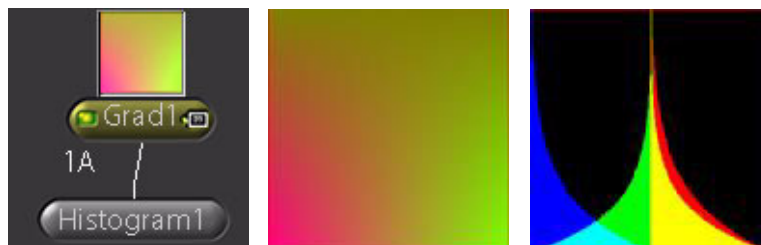
### Example 2

A 256 x 256 8-bit (ugly) *Color*. Since the color is set to (approximately) .75, .5, .25, each channel exists at only one position in the *Histogram*.



### Example 3

A 256 x 256 8-bit 4-corner *Grad*. The four corner values are of red (1, .5, .5, .5), of green (0, 1, .5, .5), and of blue (.5, 0, 0, 0). This reflects that most of red's values are around .5, ramping downward to 1, and no value is less than .5. In the Viewer, press **R**, **G**, **B**, or **C** to toggle through the different channels.



**Example 4**

Next, a *Brightness* of 2 is added between the *Grad* and the *Histogram*. The *maxPerChannel* parameter is enabled in the *Histogram* to better see the results. The result (the image is zoomed in here) is no odd values (all numbers are multiplied by 2), so there is a gap at every other value. It is a tell-tale sign of digital alteration when such regular patterns appear in a histogram.



Parameters	Type	Defaults	Function
<i>width, height</i>	int	512, 512	The width and height of the <i>Histogram</i> . You probably want to set the width to 256 on an 8-bit image for 1-to-1 correspondence.
<i>ignore</i>	int	0	<p>Tells Shake to ignore black or white pixels. For example, if you have a small element on a large black background, your histogram is skewed toward black. Disable black consideration to better analyze the image.</p> <p>0 = No ignore. 1 = Ignore values of 0. 2 = Ignore values of 1. 3 = Ignore values of 0 and 1.</p>
<i>maxPerChannel</i>	int	0	<p>This determines how the graph channels relate to each other. If you have a large red component and a small blue component, the blue is very short, making it difficult to see. Toggle this to view the blue channel in relation to itself, rather than the red.</p> <p>0 = Distribution of values relative to RGB total. 1 = Distribution of values relative to its own channel.</p>

**Synopsis**

```
image Histogram( image,  
                int width,
```

```
    int height,  
    int ignore,  
    int maxPerChannel  
);
```

### **Script**

```
image = Histogram(image, width, height, ignore, maxPerChannel);
```

### **Command Line**

*shake -histogram width height ignore maxPerChannel*



# Color Correction: Part II

## Chapter Summary

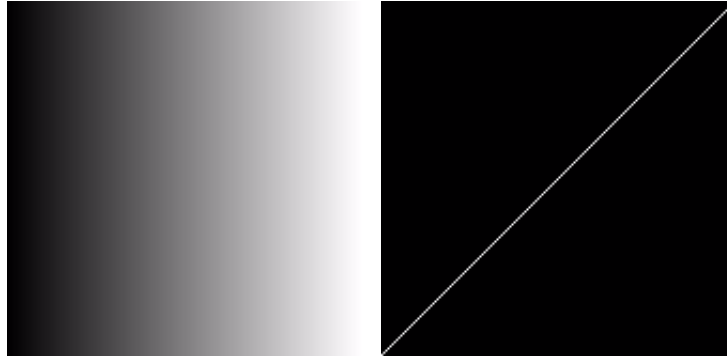
- The Logarithmic Cineon File
- Bit Depth
- About Premultiplication and Compositing

## The Logarithmic Cineon File

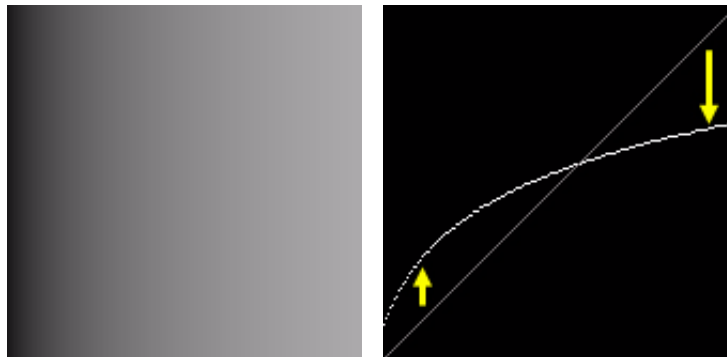
Kodak created the Cineon file format to support their line of scanners and recorders. Two things are typically associated with the Cineon file: The Cineon file itself (an uncompressed 10-bit image file) and the file's particular color representation. When graphed, this representation takes the form of a logarithmic curve, hence the term “logarithmic color” (or “log” for short). Although some refer to it as a “log space,” it is not actually a “space” such as YUV, RGB, or HSV.

**Note:** (A caveat, if you will.) This section is to be treated only as an overview of a subject that causes flame wars of such staggering proportions that everybody walks away dazed and filled with fear and loathing. You have been warned.

This image set represents linear space—every value has an equal mathematical distance in brightness to its neighbor.



The following image set represents a ramp converted to logarithmic color, where the brightness is weighted.



These images do not appear to have a pure black or white. The graph also shows that the highlights are flattened (compressed), and the blacks have more attention. This is why Cineon frames typically seem to have a very low contrast, mostly in the highlights. Since the Y axis represents the output data, it contains less data than the X axis. You can therefore store this in a smaller file than you might otherwise use, to which the Cineon 10-bit format is nicely adapted as good balance between color range and speed.

Log compression is not inherent or unique to the Cineon file format. You can store logarithmic color data in any file type, and you can store linear color data into a *.cin* file.

The easiest way to think about a logarithmic file is as a “digital negative,” a concept encouraged by Kodak literature. Unfortunately, Kodak decided to not actually invert the image. While the idea of a negative piece of film is easy to understand, this concept may cause some confusion. To see it properly, you must invert the film. In the same manner, you cannot work with a “digital negative” in your composite. You must first convert the image, in this case from the filmic log color to the digital linear color. This is called a log to lin color correction, and is performed with the Color-*LogLin* node.

It is important to remember that log to lin (or lin to log) conversion is simply a color correction. This workflow is explained below. However, it is first necessary to discuss how this color correction works.

A piece of film negative has up to approximately 13 stops of latitude in exposure. A stop is defined as a doubling of the brightness. The “digital negative,” the Cineon file, contains approximately 11 stops. A positive print cannot display as much range as a negative. The same rule applies to a computer monitor—it can only display about 7 stops of latitude. To be properly handled by the computer, information must be discarded. In a simplified example, an image contains a rounded-off range of 0 to 11 stops for black to white. Since the monitor can only display about 7 stops (for example, steps 1 through 8), the rest of the image (below 1 and above 8) is thrown away. These 7 steps, only a portion of the original negative, are then stretched back to 0 (black) and 11 (white). Because what used to be dark grey to medium-bright grey is now black to white, the image appears to have a higher contrast. However, information has been thrown away in the process. This loss is permanent when working in 8 or 16 bits, but can be retrieved when working in float. For more information, see “Logarithmic Color and Float Bit Depth” on page 356.

The extraction process is one way to control exposure. If you select the higher portions of the image (for example, 4 to 11 rather than 1 to 8), the image appears darker because you are remapping the brighter parts of the image down to black. This is the same as selecting to expose your camera for the brightest portion of your scene. The opposite occurs when selecting the lower portions of the image brightness.

These types of controls are paralleled in the *LogLin* node with the black and white point parameters. Every 90 points represents one stop of exposure. You can therefore control exposure with the *LogLin* node.

**Note:** Some Kodak documents state that 95 points represents one stop of exposure.

Once images are converted to linear color, they can be treated like other “normal” linear images. When a linear image is broken down to its steps of brightness, there is equal distance between two steps in the dark areas and two steps in the light areas. In a logarithmic file, however, the digital negative sees light areas differently than it sees dark areas, as described by the logarithmic curve that it is stored in. The distance between two steps in the dark areas is different than the distance between two steps in the bright areas. This is why color corrections and compositing should be done in linear color. This is explained in “The Hazards of Color Correcting in Logarithmic Color” on page 352.

Once you have finished your compositing, the images must be converted back to logarithmic representation (another *LogLin* node set to “lin to log”) and then rendered to disk. These images are then properly treated by the film recorder.

#### A Little Further Reading

Two Web sites are recommended for more information on this endlessly scintillating subject. The first is the specification for the logarithmic conversion and all of the parameters. It is terribly tedious, but contains good information:

[http://www.cineon.com/conv\\_10to8bit.php](http://www.cineon.com/conv_10to8bit.php)

The second recommended site contains a nice discussion of the film negative’s response to light:

<http://www.slonet.org/~mhd/2photo/film/how.htm>

### The Hazards of Color Correcting in Logarithmic Color

And you thought compositing was safe, apart from the bit where you work like a mole in a darkened room for days on end without being exposed to sunlight while being fed high fat-content food and drinks with a caffeine half-life level that can only be measured in decades.

If the logarithmic format is so great, why bother to convert back to the linear color? First, logarithmic color is unnatural to the eye—you have to convert back to linear color to see it properly. More importantly, compression also means that any color corrections applied in log color produce unpredictable results, since shadows, midtones, and highlights have an uneven application in many color correctors.

In this example, the first image is the original plate in log color. The second image is converted back to linear color, and therefore looks more natural to the eye.

**Plate in Log Color**



**Plate in Linear Color**



A Color-*Mult* node is applied to the log image and to the linear version with an *extremely slight* red color.

**Note:** You are invited to view the online PDF documentation to see the color images.

It is difficult to gauge the result in the first image, since it is still in log color. The second image has a nice tint to the pink clouds (assuming you want pink clouds).

**Mult in Log Color**



**Mult in Linear Color**



When the log image is converted to the more natural linear space, the “slight” red multiplier applied before the conversion has completely blown the image into the red range.

**Mult in Log Color Viewed in Linear Representation**



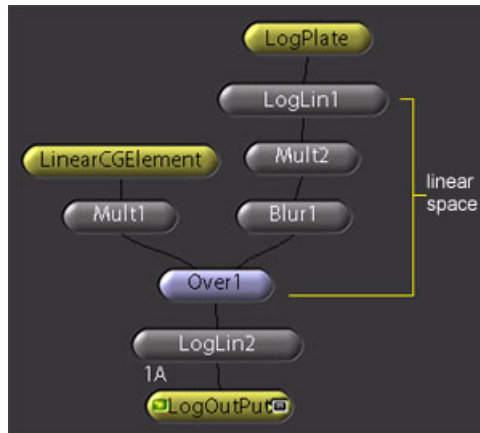
This is bad.

Therefore, you are mathematically urged to color correct in linear color, or view a conversion to linear using a VLUT while you adjust the color correction.

### Converting Between Logarithmic and Linear Color

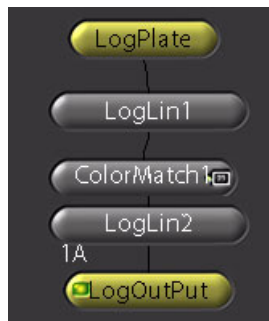
As previously mentioned, logarithmic images are simply a result of a color correction. This correction has been standardized by Kodak. Everyone’s conversion function is essentially the same. The Shake node that handles this correction is the *LogLin* node, and is located in the Color Tool Tab. You can convert from log to linear, or linear to log color.

Typically, a *LogLin* node is applied after a *FileIn* to convert the image from log to linear color. You do your composite in linear color, convert back to log at the very end, and attach the *FileOut*. In the following node tree, *LogLin1* has a conversion setting of log to lin, and *LogLin2* has lin to log.



This example also shows compositing in an imaginary 3D-rendered element, read in by the *FileIn* named *LinearCGElement*. These elements are almost always rendered in linear color, so no conversion is necessary.

If you are simply doing color timing, as in the following example, you have an added benefit: All the color corrections concatenate into one internal operation.



### Wedging and Color Timing

So, why have numbers in the *LogLin* function? These numbers are used to color correct your plate as a sort of calibration of your entire input/output system. There are (at least) two fundamental ways to do this: Wedging and Color Timing. The following two methods are only suggestions. Every facility probably has its own system—there is no standard. (Isn't film fun?)

When wedging a shot, you ensure that the files output from your entire digital process match the original film print. There are multiple places that color changes can intentionally or accidentally occur—exposing film in the camera, printing the workprint, scanning the data into digital format, digitally compositing, recording the digital plate onto film, and developing the print. You are saved on the one hand in that much of this process is out of your hands—the print lab guy was cranky because his delivered coffee was too cold, so what can you do? To limit discrepancies, use the *Wedge* macro in the “Cookbook” section of the *Shake 3 Tutorials*. This is a macro for *LogLin* that puts in preset variations of the offset values and contrast values, and steps up and down to try to find a match of the original scan. The process involves something like the following:

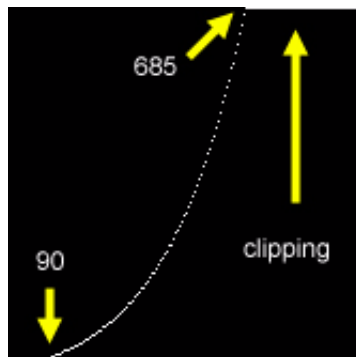
- Make a workprint of your original negative. This is the standard to which you compare your digital output.
- Scan the negative into a digital format (which usually creates log Cineon files).
- Create a *FileIn* for a single frame from the scanned image files, and attach the *Wedge* macro.
- Visually, take your best shot at the exposure level. (You may have to boost the blue up or down 22 points to see what may work.) Remember that 90 points is 1 f-stop of exposure and 45 points is half a stop.
- Render out 48 frames. The *Wedge* macro automatically brackets your initial pick up and down by whatever value you set as the colorStep. For a wide bracket, use a high number (such as 90). For a narrow bracket, use a lower number (such as 22). This process prints 48 different color, brightness, and contrast tests, then automatically returns the image to log color with the default settings. The internals of the *Wedge* macro are basically *LogLin* (log to lin) > *ContrastRGB* > *LogLin* (lin to log).
- Record and print your 48 frames.
- Compare the actual physical pieces of film to the frame of the original workprint on a light box using a loupe. This makes you look really cool because you appear to know what you are doing. The exposure numbers are printed on the frame by the *Wedge* macro. Select the frame that exactly duplicates the original print. If no frames match up, adjust your starting points for red, green, and blue, narrow your colorStep, and wedge again until you have a frame that looks correct.
- For all composites that use this plate, use the values you selected in the *Wedge* in your *LogLin* node converting from log to linear color. Keep your default values when returning to log color, with the exception of the conversion setting, which is linear to log.

This process is generally done for every shot. At a larger studio, there is likely to be an entire department to do the color timing for you. Count yourself lucky to be in such a wise and far-sighted studio.

The second technique to handle the color process is to not handle it at all—let the Color Timer for the production or the developing lab handle it. These are folks who sit around in a cave and adjust exposures on film all day for a living. Nice. While this is easier, you do surrender some control over the process. However, this is a perfectly acceptable technique used in many large effects houses. When you employ this technique, you use the same values in your *LogLin* in and out of your color. You may adjust the numbers slightly, but make sure the same numbers are used in both operators. For example, because a physical piece of negative has a slight orange cast, your positive scan may have a slight green cast. You may want to adjust for this in the *LogLin* function. A good technique is to adjust your log to lin *LogLin*, copy the node (**Command+C** / **Ctrl+C**), and then paste a linked copy back in (**Shift+Command+V** / **Shift+Ctrl+V**). Next, adjust the conversion setting of the second *LogLin* to linear to log. If you adjust the original *LogLin*, the copy takes the same values since it is a linked copy.

### Logarithmic Color and Float Bit Depth

OK, here is where it gets tricky. If you examine the following logarithmic to linear conversion curve, you can see that clipping occurs above 685.



The *LogLin* function does have a roll-off operator, which helps alleviate the sharp corner at the 685 point, but this is inherently a compromise—you are throwing data away. You do not really see this data disposal in linear color. However, once you convert back to logarithmic, the clipping is evident.

In the following greyscale examples, the left image is a ramp with an applied log to lin conversion. It is now in linear color. The right image is the left image converted back into log color representation. Quite a bit of clipping has occurred.

**Log Greyscale Converted  
to Linear Color**

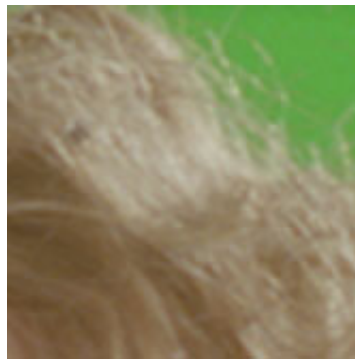


**Previous Example Converted  
Back to Log Color**



The following images are from a log plate. The left image is the original plate. The right image is the output plate, also in log color, that has been passed through the log > lin > log conversion process.

**Original Log Image**



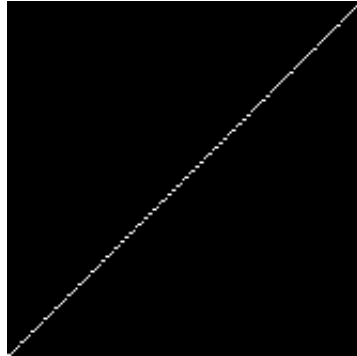
**Output Log Image**



Notice that the highlights in the hair detail are lost (and it looks more like taffy than hair, yum).

The following images are graphs that represent the log plates in the above illustrations. The left graph represents the entire range of the log image. Keep in mind this represents all of the potential values—few log plates have this entire range. The right image displays the result of the  $\log > \text{lin} > \log$  conversion process.

**Graph of Original Ramp**

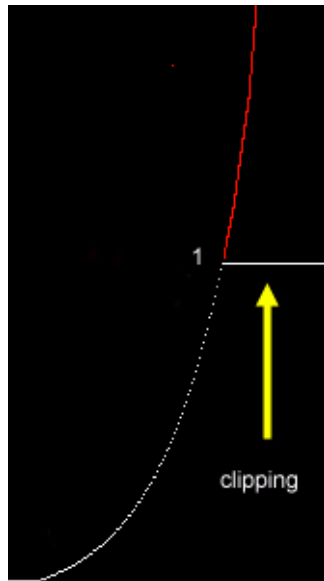


**Graph of Output Ramp**



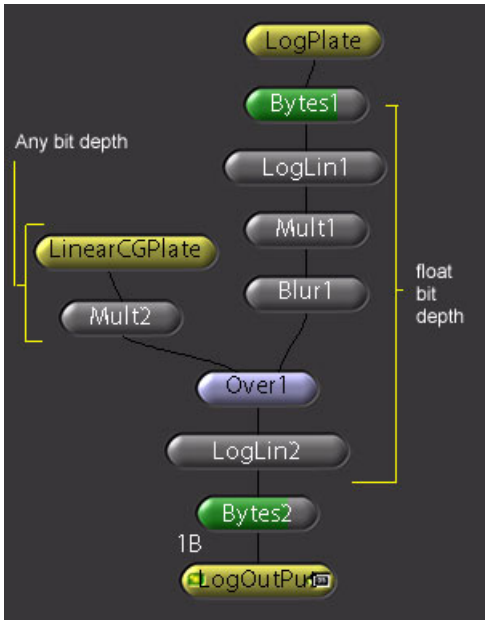
So, why is this happening? In the first part of this chapter (“The Logarithmic Cineon File” section), using an 11-stop example image, 7 steps were extracted and the rest thrown away in order to view and work on the image in the computer. These same 7 steps can be thought of as the steps between 95 and 685. As mentioned earlier, the rest is thrown away, or clipped.

If you look back at the original log > lin conversion graph, the curve suggests that it should continue past 1, but so far, they have been clipped at 1. In the following illustration, the red line represents the potential information derived from the color conversion. It is clipped at 1 because values can only be stored from 0 to 1 in 8 or 16 bits.



As the illustration suggests, if data could be preserved above 1, you could always access the data, and life would be happy. Fortunately (by design), you can convert your images to a higher bit depth called “float.” Whereas 8 and 16 bits are always in a 0 to 1 range, float can describe any number and is ideal for working with logarithmic images when converting to linear representation and back to log representation. If you keep your images in linear color because you are going out to a linear format, float is not necessary.

The following image shows a modification of the compositing tree from “Converting Between Logarithmic and Linear Color” on page 353. An *Other-Bytes* node is inserted, and bumps the image up to 4 bytes (32 bits, or float).




Notice that you are not obligated to promote your 3D-rendered element (here represented with *LinearCGPlate*) to float, since it is already in linear color. The second *Bytes* function at the end of the node tree is in case you render to an .iff format—you may want to convert it down to 16 bits for smaller storage costs. Cineon is always stored at 10 bits, and therefore does not need the second *Bytes* node.

Just to be handy, the following is a table of file sizes for 2K plates. Draw your own conclusions.

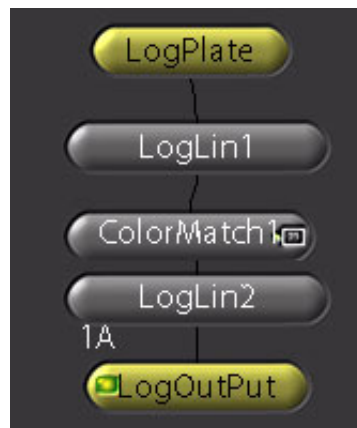
2K RGB Plate	Size
Cineon, 10 bits	12 MB
IFF, 8 bits	9 MB
IFF, 16 bits	18 MB
IFF, float	36 MB

In addition to storage requirements, working in float costs you render time, a *minimum* of 20 percent, but usually much much higher than that.

### Looking at Float Values

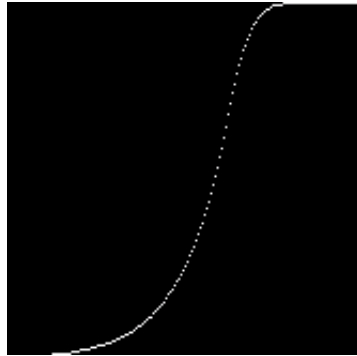
You can use the Float View ViewerScript  to view values that are above 1 or below 0. To look at these values, create a *Ramp*, and set depth to float. Then, apply a Color-*LogLin*. When you turn on the Float View, the top 35 percent turns white, indicating values above 1, and the lower 9 percent turns black, indicating values below 0. Everything else turns grey, indicating values between 0 and 1.

If you are just color timing (only doing color corrections that concatenate), there is no need to convert to float. All concatenating color correctors, including the two correctors to convert in and out of linear representation, are calculated simultaneously. Therefore, the following tree is still accurate.

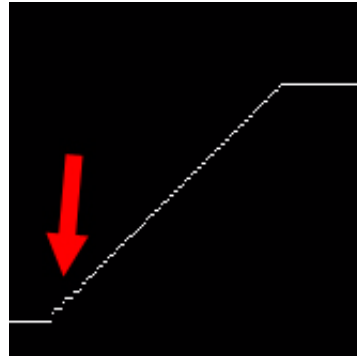


As an alternative to float, you can use the roll-off parameter in the *LogLin* node. However, this involves inherent compression and banding of your data. The roll-off parameter gives your curve a roll-off (compared to the standard log curves), which can help preserve some highlights, and allows you to stay in 16 bits. However, as shown in the next example, when the same image is converted back to log representation, there is still some cutoff in the upper and lower ranges, but the lower ranges also band. Use at your own risk.

**Graph of Linearization With Roll-Off**



**Graph of Relog With Roll-Off**



### Float Bit Depth and Third-Party Plug-Ins

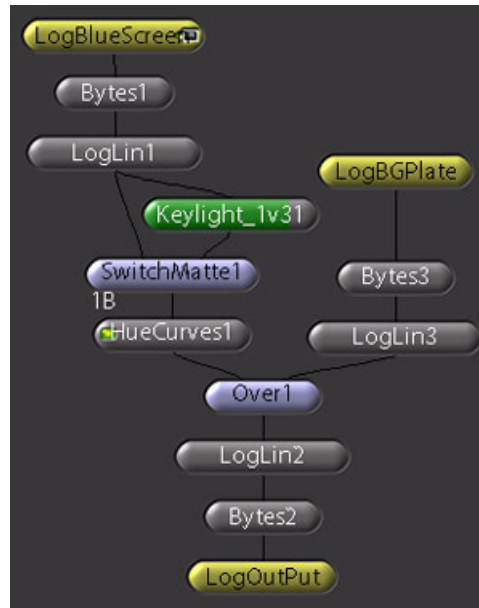
Most third-party plug-ins, including *Primatte* and *Keylight*, do not support float (only 8 and 16 bits). To ensure your highlights are maintained, do the following:

- Convert the images back to log color.
- Execute the third-party plug-in (assuming it is still accurate on non-linear images).
- Convert the images to linear representation.
- Continue with your composite.

If the plug-in works on a single channel (for example, both *Primatte* and *Keylight* can isolate their effect on the alpha channel), do the following:

- Create one branch for color modifications, and keep that branch in float.
- Create a second branch from the *FileIn* and pull the key.
- Copy the alpha back to the float RGB chain with a *Layer-SwitchMatte*.
- Since the keyers also do color correction, you have to compensate by using a *Color-HueCurves* or *Key-SpillSuppress* to do your spill suppression.

The following is a sample tree.



There are two exceptions:

- *Keylight* allows you to key, color correct, and composite in log color. Simply toggle colourSpace to log.
- *Ultimatte* preserves float data. No tricks necessary. Nice.

Applying gentle pressure to the plug-in manufacturer to support float images is also a nice alternative, but less productive in the short run.

## Bit Depth

Bit depth is a term to describe how precisely you want to describe a color range. The amount of steps in a color range is calculated by taking 2 to the  $n$ th power, with  $n$  representing the amount of bits. For example, a 1-bit image gives you two values—black or white. A 2-bit image gives you  $2^2$ , or 4 values per channel. How does this translate to image quality?

In a simplification, the following chart displays a black-and-white ramp in 1 bit, 2 bits, 3 bits, and 8 bits. Normally, you work with images 8 bits or above.

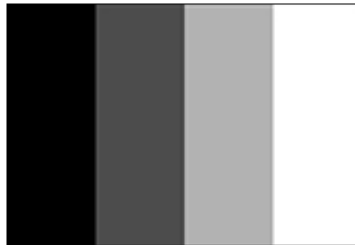
**1 Bit, 2 Values Total**



**Graph of 1-Bit Image**



**2 Bits, 4 Values Total**



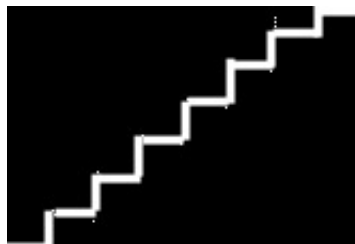
**Graph of 2-Bit Image**



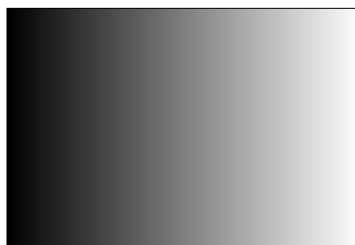
**3 Bits, 8 Values Total**



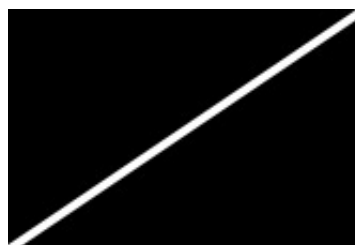
**Graph of 3-Bit Image**



**8 Bits, 256 Values Total**



**Graph of 8-Bit Image**



These graphs demonstrate that the more bits in an image, the finer the color transitions. Because of the responsiveness of film and the size of the screen, an image that looks fine at 8 bits on video may look terrible on film. Digital film compositing refers to bit depth on a per-channel basis, so 8 bits refers to 8 bits per channel, or 32 bits total for an RGBA image. It gets slightly confusing, since Shake can go up to 32 bits per channel, meaning 128 total on an RGBA image. To complicate things, since 8 bits equals 1 byte, images in Shake are set with a byte value of 1 (8 bits), 2 (16 bits), or 4 (32 bits, or float).

**Note:** The above examples of 1-, 2-, and 3-bit images are not supported by Shake, but are used for demonstration purposes.

### About Bit-Depth Independence

Most non-film composites work fine at 8 bits, the standard output of 3D renderers and most paint packages. However, there are times when you need more information in your images, forcing you up to 16 bits, or a whopping 65,000 (more or less) values per channel. A typical example is a ramp of similar values (for example, light blue to medium blue on an image of the sky) across a great amount of screen space. An 8-bit image, though sometimes indiscernible from 16 bits on a computer monitor, creates banding on the film out. This banding is similar to the above 3-bit example as compared to the 8-bit example. If your image is generated in 8 bits in a different package, you get no immediate improvement by merely bumping it up to 16 bits. In the above example, the ramp needs to be generated in 16 bits in order to take advantage of the extra precision of 16 bits.

An interesting note about 8-bit images is that there is no 50 percent point—you have a smidgen less than 50 percent grey and a smidgen more than 50 percent, but you cannot get an exact 50 percent value. This occasionally comes into play with macros.

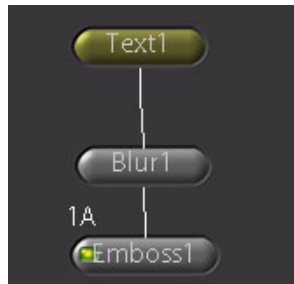
So, why not always work at 16 bit? Most film houses do, but the price is slower calculations, more memory required, and larger-sized image files.

Shake's 32-bit representation is special float description—values can go above 1 or below 0. In all of the above examples, the ramp ranges from 0 to 1. If you add two 8-bit ramps together, it adds the white values together ( $1 + 1$ ), but clips it at 1. This is fine visually, but you may later do other mathematical computations in which it is important to realize that  $1 + 1$  is 2, not 1. A good example is the Z channel, which is always in float. The Z channel is usually generated by a 3D render, and supplies the distance on a per-pixel basis of the object from the camera. Therefore, values could go from 0 to infinity. If you swap your Z channel into your red channel, you do not want it clipped off at 1, because you could not tell the difference between the pixels that are 2 units away and the pixels that are 1000 units away. A float representation, however, maintains these values.

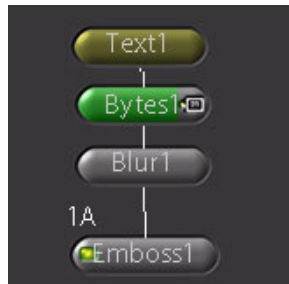
Shake recognizes and maintains the bit depth of incoming images—except for 10-bit Cineon files, which are boosted to 16. Because of the concatenation of color corrections, you are penalized less frequently when working at 8 bits than in other software. This is because the adjacent color corrections are collapsed into one mathematical lookup table, in effect doing the computation in float. The resulting image is returned to its source bit depth. However, with the use of the *Bytes* node, you have the option of promoting your image to a higher or lower bit depth. As the name implies, the *Bytes* node takes bytes as its argument, so a value of 1 equals 8 bits, 2 equals 16 bits, and 4 equals 32-bit (or float) values. (There is no “3 bytes” setting.) For information on the *Bytes* function, see “The Bytes Node” on page 367.

You may need to go to a higher bit depth when using functions, such as *Emboss* and *Blur*, since they naturally create smooth gradations. In the following example, the image on the left has a *Blur* node and an *Emboss* node applied. At 8 bits, terracing appears. By inserting a *Bytes* node (in the Other Tool Tab) set to 2 bytes (16 bits), the *Emboss* is smoothed.

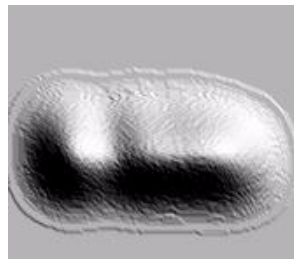
**8 Bits**



**16 Bits**



**2 Bits, 4 Values Total**



**16-Bit Image**



Be sure to promote the image before the *Blur*. This does not mean you need to do this before every *Blur*, but that blurs are prone to displaying banding when many operations are applied to the blurred image.

You can seamlessly layer images of different bit depths together. The lower bit-depth image is automatically promoted to the higher bit depth. (For example, an *Over* node with an 8-bit image and a 16-bit image results in a 16-bit image.) This is invisible to the user; it is automatic. To reverse this behavior, insert a *Bytes* node before the *Over* node on the 16-bit image to compress the image to 8 bits. Bit-depth level is calculated entirely locally. If you mix 8-bit and 16-bit images, only the sections of the node tree that come after the 8-bit image is composited with the 16-bit branch are calculated at 16 bits.

10-bit Cineon files are automatically promoted to 16 bits when read in or written by Shake, so you don't have to worry about any data loss. However, the linearization of the log files may result in loss unless you first promote your images to float. For more information, see “The Logarithmic Cineon File” on page 349.

**The Bytes Node**

The *Bytes* function pushes the input image into a different bit-depth resolution. The bit depth is counted in bytes-per-channel. To view the current bit depth of an image, you can look at the title bar of the Viewer, or look at the output of a shake *-info* on the command line. 10-bit Cineon files are automatically pushed to 16 bits when read by Shake.

**Note:** When compositing images of different bit depths, you do not need to force the images to conform; Shake automatically pushes the lower bit-depth image to the higher bit depth.

Parameter	Type	Default	Function
<i>outBytes</i>	int (1, 2, or 4)	bytes	Forces the incoming image into a new bit depth. 1 = 1 byte per channel, or 8 bits per channel. 2 = 2 bytes per channel, or 16 bits per channel. 4 = 4 bytes per channel, or a float per channel.

**Synopsis**

```
image Bytes( image, int outBytes );
```

**Script**

```
image = Bytes( image, outBytes );
```

**Command Line**

```
shake -bytes outBytes
```

**Examples**

```
shake lisa.iff -bytes 1
```

```
shake lisa.iff -bytes 2
```

```
shake lisa.iff -bytes 4
```

These commands convert the image to 8, 16, and 32 (float) bits, respectively.

## About Premultiplication and Compositing

This discussion of premultiplication and compositing is very important because it talks about compositing and color correction on a fundamental level. It details the process that makes standard compositing functions work, and defines the term “premultiplication” (and why you should know about it and order it now for the entire family before the holiday rush begins). The concept of premultiplication is important regardless of your compositing software.

Premultiplication should be considered when you have to modify a foreground element, and composite it over a background image. This “modification” classically means color correction, especially nodes that raise the black level such as *Add*, *Clamp*, *ColorMatch*, *ColorCorrect*, *Compress*, and *Contrast*, but there are consequences with filters as well.

Some software packages “take care of it” for you—they hide the state of premultiplication. This works great 9 times out of 10, but for that last 10 percent, you have no practical solution. Other compositing packages pretend it doesn’t exist and encourage you to chew on your matte, or modify foreground/background alpha multiplication curves. Taking a third approach, Shake gives you complete and explicit control over premultiplication. Although this does mean it can be painful, it helps you get around the typical problems with other software packages.

If you don’t feel like reading a long-winded dissertation on the mathematics of premultiplication, here is the abridged version:

- Rule Number 1: Color correct unpremultiplied images. To unpremultiply an image, use *Color-MDiv*.
- Rule Number 2: Filter and transform premultiplied images. To premultiply an image, use *Color-MMult*.

### The Definition of Premultiplied

The definition of a premultiplied image is, “An image that has its RGB channels multiplied by its alpha channel.”

Typically, images from a 3D render are premultiplied. This means the transparent areas have black in both the RGB areas and in the same areas of the alpha channel. A side effect of a premultiplied image is that the RGB channels never have a higher value than the alpha channel. The beauty of all this is that it has absolutely no meaning without understanding how compositing works, which is why this section is very long. (So, jam those toothpicks between the eyelids and read on.)

## Problems Caused When Premultiplication Is Ignored

There are two typical problems that occur when the premultiplied state of an image is ignored.

**To see an example of premultiplication problems:**

- 1 In the Image tab, click the *FileIn* node, go to *doc/pix/premult*, select the *bg.jpg* and *munch\_pre.sgi* images, and click OK.

The nodes appear in the Node View. These images appear courtesy of our fine friends at Oddworld Inhabitants for the game “Munch’s Oddysee™.” (Note for the purposes of illustration, these differ from the final composite, which looks much saucier than these examples.)

The image *munch\_pre.sgi* is rendered out of Maya, is premultiplied (the blacks in the alpha match the blacks in the RGB), and has an embedded alpha channel. Pictured in this image, by the way, is Munch. (We aren’t sure what Munch is—he can shoot lightning bolts out of his head, so that should count for something.)

**munch\_pre.sgi**



**munch\_pre.sgi,  
alpha channel**



**bg.jpg**



- 2 In the Node View, select the *munch\_pre* node, click the Layer tab, and then click the *Over* node.

An *Over* node is added to the *munch\_pre* node.

- 3 Connect the *bg* node to Background input (the second input) of the *Over* node:



- 4 Insert a *Color-ContrastLum* between the *munch\_pre* node and the *Over* node, and set the *ContrastLum* value to .5.

If you zoom into the edges, a white edge appears around Munch. This is problem number one: “The Nasty White Edges Problem.”



- 5 To replace the *ContrastLum* node, select it in the Node View and **Ctrl**-click the *Add* node (in the Color tab).

The *ContrastLum* node is replaced by an *Add* node.

- 6 In the *Add* parameters, boost the Color (red/green/blue) values to approximately .4.

**Note:** To adjust the color values, you can also press **O** (for Offset) and drag the mouse to the right over the *Add* Color Picker.

Although the *Add* node is only applied to the *munch\_pre* node, the entire image is brightened. This is problem number two: “The It’s-Too-Bright-Image Problem.”



If premultiplication status is ignored, you can have problems with edges, or with raised global levels. Many people see these types of errors, and assume it is a mask problem, so they pinch in the mask a bit. Even more extreme people have erroneously assumed you cannot color correct premultiplied images. These problems are easily solved through proper management of premultiplication (which still hasn’t been explained).

### The Math of Over and KeyMix

To understand why premultiplication problems occur, it is best to cut straight to the heart of compositing by understanding how a standard composite operator (foreground through a mask over a background) works. This has nothing to do with Shake, but in fact was worked out in the 1970s, mainly by two clever (but no doubt fun) fellows named Porter and Duff.

The following is the math equation they developed. In this equation, A is the foreground’s alpha channel:

$$Comp = (Fg * A) + ((1-A) * Bg)$$

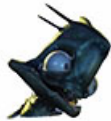
Rather than go into this too deeply, look briefly at the significant part,  $(Fg * A)$ . This is the definition of premultiplication—“RGB multiplied by its alpha.” To avoid the math, you can build the composite with other nodes, in effect constructing an *Over* node from scratch.

The following example (continued from the above section “Problems Caused When Premultiplication Is Ignored”) shows how to build the compositing equation.

**To build the compositing equation:**

- 1 Read in the *munch\_unpremult.jpg* and *munch\_mask.iff* images from *doc/pix/premult*. The *munch\_unpremult* image is a non-premultiplied image. It has no mask, so there is no correspondence between black pixels in the RGB channels and black pixels in the mask to describe transparency of the image.

**munch\_unpremult.jpg**



**munch\_mask.iff**

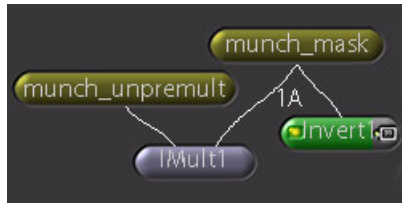


- 2 The first step in the formula is  $(Fg * A)$ . To duplicate this with nodes, attach a *Layer-IMult* node to *munch\_unpremult* and connect the *munch\_mask* node to the *IMult* background. The result is identical to *munch\_premult*—the RGB is multiplied by the mask.



- 3 Next, invert the foreground alpha channel  $(1 - A)$ . To do this, select the *munch\_mask* node, click the *Color* tab, and **Shift**-click the *Invert* node.

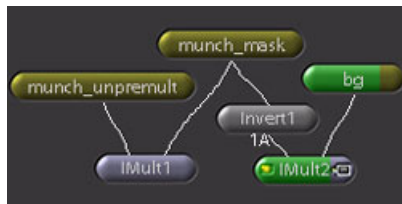
The *Invert* node is attached to the *munch\_mask* node as a separate branch.



- 4 The next step in the formula  $((1-A) * Bg)$  calls for you to multiply the background by the inverted alpha. In the Node View, select the *Invert* node, click the Layer tab, and click *IMult*. An *IMult* node is added to the *Invert* node.
- 5 Connect the *bg* node to the Background input of the *IMult2* node.

**Tree**

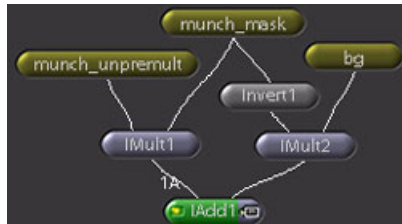
**IMult2**



- 6 Next, the crucial step—add the two results together. In the Node View, select *IMult1*, click the Layer tab, and click *IAdd*. Connect the *IMult2* node to the background input of the *IAdd* node.

$$Comp = (Fg * A) + ((1-A) * Bg)$$

The result is exactly the same as the *Over* node.



By punching a hole in the background, the alpha determines what is transparent when the two plates are added together. The key concept is that because you are adding, anything that is black (a value of 0) does not appear in the composite.

**IMult1**



**IMult2**



Since the *KeyMix* and *Over* nodes do this math for you, you do not need to create four nodes to do a composite. What is the difference between *Over* and *KeyMix*? *Over* assumes that the foreground is premultiplied (identical to *IMult1*). *KeyMix* is only for non-premultiplied images (identical to *munch\_unpremulti*). Strictly speaking, the math breaks down like this:

$$\text{KeyMix} = (Fg * A) + ((1-A)*Bg)$$

$$\text{Over} = Fg + ((1-FgA)*Bg)$$

In these formulas, the foreground of *Over* is already multiplied by the foreground alpha channel, thus the term premultiplied—it isn't multiplied in the composite because it was previously multiplied, usually by the 3D renderer.

### Unpremultiplying an Image

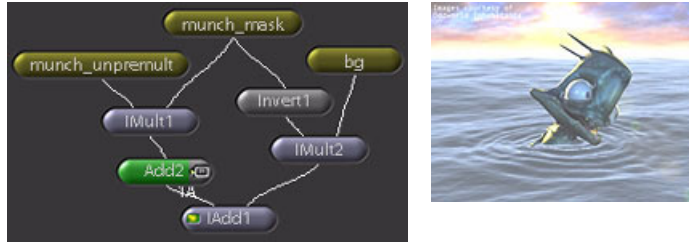
With this knowledge, you can go back and start to understand the errors that occur when the *ContrastLum* and *Add* functions are used with the *Over* function.

An *Add* node was originally attached to the premultiplied Munch.

#### To do the same thing here, do the following:

- 1 In the Node View, select the *IMult1* node, click the Color tab, and click *Add*.  
An *Add* node is attached to the *IMult1* node.
- 2 With the *Add* node selected, click the Layer tab, and click *IAdd*.  
An *IAdd* node is attached to the *Add* node.
- 3 Connect *IMult2* to the background input of the *IAdd* node.
- 4 In the *Add* node parameters (click the right side of the node to display its parameters), set Color to .5.

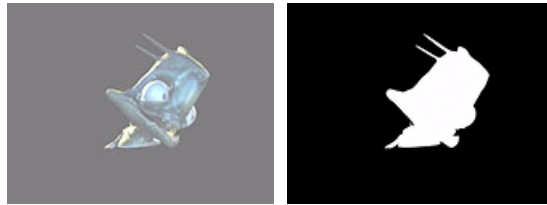
The same error occurs—the entire image brightens, not just Munch. This is bad.



For a clue, double-click the *Add* (*Add2*) node. There is no longer an exact correlation between the black areas of *munch\_mask* and the black areas of the RGB channels.

**Add2**

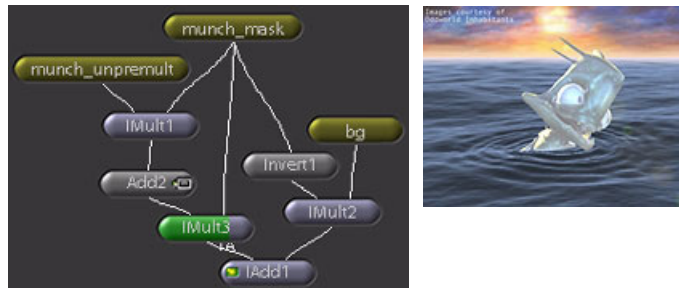
**munch\_mask**



Now things get a little odd. To reassert the mask, insert another Layer-*IMult* node after *Add* (*Add2*), and connect the *munch\_mask* node to the *IMult3* background input. The image is premultiplied again and appears to work.

**Tree With IMult3 Inserted**

**IAdd2**



Although this looks fine, mathematically speaking, there is an error. You need to have the correct compositing equation:

$$Comp = (Fg * A) + ((1-A)*Bg)$$

But, in fact, the above example multiplies the foreground twice (there are two *IMult* nodes), so you have this:

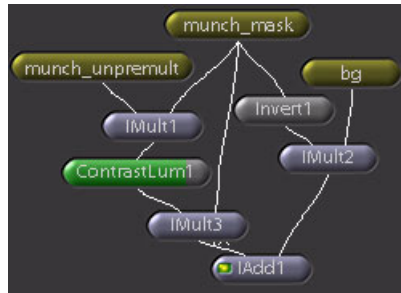
$$Comp = (Fg * A * A) + ((1-A)*Bg)$$

You may be thinking, “So what, big deal, it looks fine.” OK, to test to see if it looks fine in all cases, switch the *Add* node to a *ContrastLum* node.

**To test the equation:**

- 1 In the Node View, select the *Add* (*Add2*) node in the tree, click the Color tab, and then **Ctrl-click** *ContrastLum*.
- 2 In the *ContrastLum* parameters, set the contrast value to .5. Look very closely and notice a dark rim appears around the edge.

**Add Switched to ContrastLum**

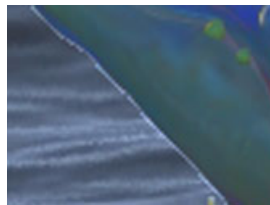


**IAdd1 Detail**



- 3 In the Node View, select the *IMult3* node and press **I**.  
The node is ignored, and the same nasty edges appear as before.

**IAdd1 Detail With  
IMult3 Ignored**



- 4 Press **I** again so the node is no longer ignored.  
So, what are you to do? Returning to your fourth-grade math class, dividing something by a number and then multiplying by the same number is the same as multiplying by 1—the equivalent of no change at all. Therefore, if you multiply the foreground by the mask one too many times, divide it by the alpha to balance it out.

Mathematically, this looks like this:

$$Comp = (Fg * A / A * A) + ((1-A)*Bg)$$

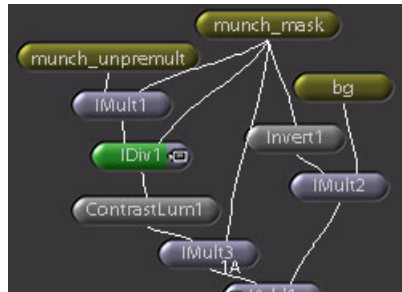
Or, abbreviating the equation, we return to the proper formula, since the two extra alphas (A) cancel out:

$$Comp = (Fg * A) + ((1-A)*Bg)$$

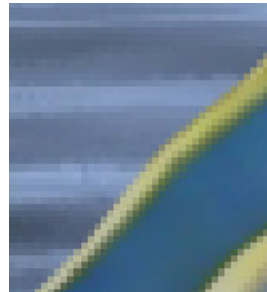
**To translate this into the node tree:**

- 1 In the Node View, select the *IMult1* node, click the Layer tab, and then click *IDiv*.  
An *IDiv1* node is attached to the *IMult1* node.
- 2 Connect the *munch\_mask* node to the *IDiv1* background input.  
The edge appears clean.

**IDiv Inserted**



**IAAdd1 Detail**



- 3 To test the result, select the *IDiv1* node and press **I** several times to ignore and show the node.  
By dividing by the mask, the image is unpremultiplied and ready for color correction.  
Therefore, you can conclude that color correction should be done on an unpremultiplied image.  
Now, here comes the kicker. The insertion of *IDiv1* and *IMult3* can be bypassed entirely.

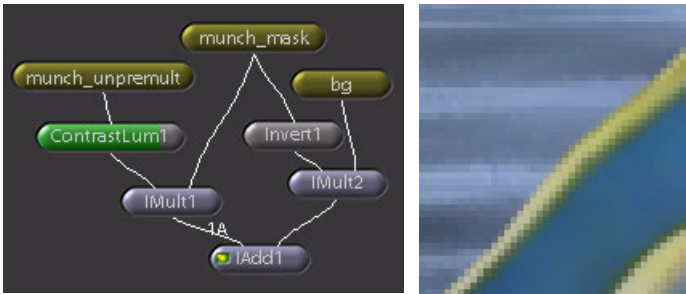
**To bypass the insertion of IDiv1 and IMult3:**

- 1 Delete the *IDiv1* and *IMult3* nodes.
- 2 Connect the *ContrastLum1* node between the *munch\_unpremult* node and the *IMult1* node.

Since *munch\_unpremult* is already an unpremultiplied image, you get the same clean result.

**IDiv and IMult3 Deleted,  
ContrastLum Moved Up**

**IAdd1 Detail**



**Remember this:**

- Rule Number 1: Color correct unpremultiplied images.

**Managing Premultiplication**

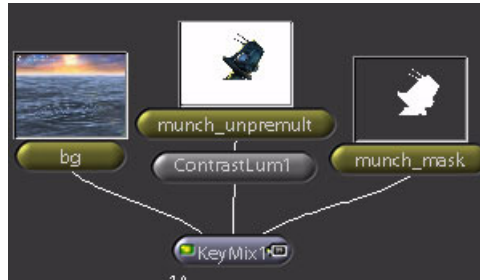
The above steps are an elaborate illustration to explain *Over*, *KeyMix*, and premultiplication. The basic difference between *KeyMix* and *Over* is:

KeyMix	Over
For non-premultiplied foreground images. The alpha can be anywhere, including in the foreground image.	For premultiplied foreground images, but can also enable premultiplication if necessary for unpremultiplied images. The alpha is in the foreground element.

Shake does not require you to constantly separate your alpha and to perform *IMult* and *IDiv* operations with the second image. Instead, there are nodes to do this for you. The following examples duplicate the current complex tree with simplified versions.

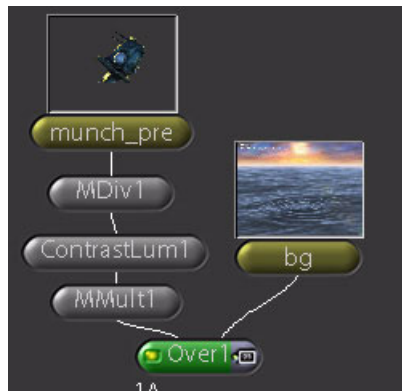
This example uses the Layer-*KeyMix* node, which handles unpremultiplied foreground elements, the background, and a mask to split between the two. Enable invert in the *KeyMix* parameters (you could also switch the foreground and background inputs).

#### Identical Tree Using KeyMix



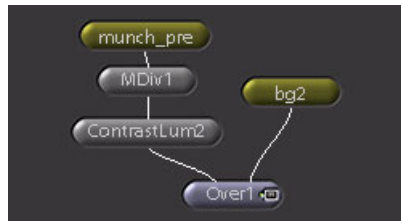
If you are using a premultiplied image straight out of a 3D render, there are special tools that unpremultiply and premultiply the RGB channels by the alpha. These tools are Color-*MDiv* and Color-*MMult*. “MDiv” is “Mask Divide,” and “MMult” is “Mask Multiply.” Used in a node tree, *MDiv* unpremultiplies the image, then you color correct, and then premultiply using *MMult*. With the premultiplied image as the foreground, the tree looks like the following.

#### Identical Tree With a Premultiplied Foreground

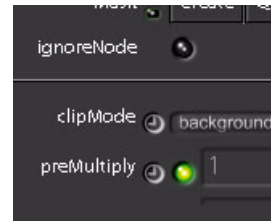


*Over* has a premultiplication shortcut. In the following tree, the *preMultiply* flag in the *Over* parameters is enabled to omit the *MMult* entirely.

### Identical Tree With Premultiplication Handled by Over1

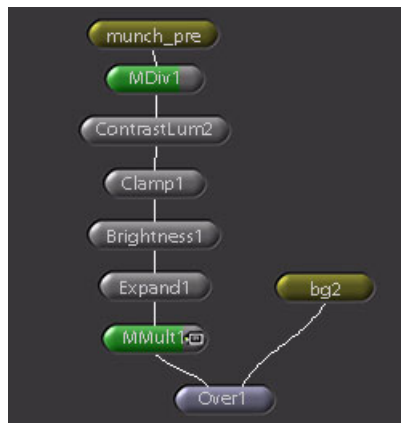


### Over1 Parameters



You are not required to apply an *MDiv* for each color correction. You can stack as many correctors up as you want.

### Multiple Corrections on One MDiv



## Filters and Premultiplication

Now to our next step—using filters (such as *Blur*):

### Remember this:

- Rule Number 2: Filter and Transform premultiplied images.

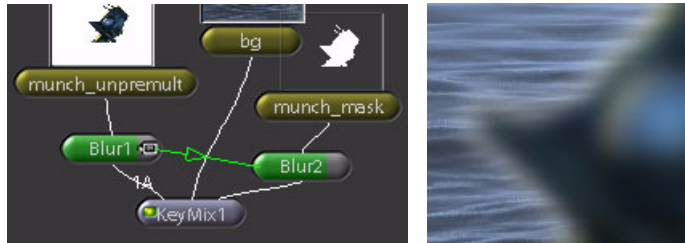
In the next example, an unpremultiplied image is filtered.

- In the node tree, a *Blur* node is added to the *munch\_unpremult* node.

The mask clips any soft gradation.



- The *Blur1* node is copied and cloned. (Copy the node and press **Ctrl+Shift+V**.)  
**Note:** Press **Command+E** / **Ctrl+E** to see that the *Blur2* node is slaved to the *Blur1* node.
- The *Blur2* node is applied to the *munch\_mask* node.  
 Notice a glowing highlight is introduced around the image.



To eliminate the glow, the blur is applied on an image that has nothing added to the rim (on a black background)—since black has a value of 0 and therefore does not add to the filtering.

- To fundamentally change the compositing order, the foreground is premultiplied by the mask with a *SwitchMatte* node, and then an *Over* node is used to composite—*KeyMix* is only for unpremultiplied images.



- The following image is the same tree with the premultiplied image. The *preMultiply* parameter is disabled in the *Over1* node (or else a black edge appears around the image).

#### Blur on a Premultiplied Image

#### Over1 Detail



Since transforms do a bit of filtering, technically speaking, you should perform transforms on a premultiplied image as well, although your tolerances are a little looser.

Typically, make sure your image is unpremultiplied by applying an *MDiv* to premultiplied images, do your color correction, apply an *MMult*, apply transforms and filters, and then composite.

#### Nodes Useful or Particularly Pesky When Dealing With Premultiplication

The following nodes can generally switch the premultiplication status of an image, although anything that operates on the mask differently from the RGB affects this status.

Node	Details
<i>ColorCorrect</i>	This color corrector has an option in its Misc tab to indicate if the incoming image is premultiplied. If it is, set it to "yes." <i>MDiv</i> and <i>MMult</i> are internally inserted into the computation.
<i>Reorder</i>	Allows you to switch a channel into the alpha channel, and could disrupt your premultiplication status. However, it isn't often used on images in the normal chain of color correct, position, composite—but is more of a utility node.
<i>DilateErode</i>	This node is typically used to add or chew the matte by a few pixels. Set your channels to just "a." Since you then modify the matte separately from the RGB, you make the image unpremultiplied.
<i>LumaKey</i> , <i>DepthKey</i> , <i>DepthSlice</i>	These keyers all have the option to immediately set your image to premultiplied.
<i>KeyLight</i>	This keyer can output either a premultiplied or unpremultiplied version, or just the alpha channel with the RGB untouched.

Node	Details
<i>Primatte</i>	This keyer can output either just the alpha or a premultiplied version. Setting it to unpremultiplied requires an external <i>MDiv</i> .
<i>AddMix</i>	This modified <i>Over</i> node allows you to manually break the premultiplied relationship by tweaking curves of how the matte multiplies the foreground and background images. It was inherited from another package, where it was used extensively because this particular system had no control over premultiplication.
<i>SwitchMatte</i>	Copies a channel from the second image to the alpha channel of the first image. You have the option to immediately premultiply the image (enabled by default).
<i>DropShadow</i>	This should be applied to premultiplied images only.

### Non-Black Premultiplied Images

Some packages output images that are considered premultiplied, but the background is not black. Extremely confusing. To work with these, try the *AEPremult* macro in the “Color Macros” section of the “Cookbook” in the *Shake 3 Tutorials*.

# Using Masks

## Chapter Summary

- About Masks
- Masking an Effect
- Masking a Layer
- Masking Filters
- Masking Using the Constraint Node

## About Masks

Masking is closely related to keying. Keying can be considered as the process of pulling a matte, typically from a green screen or a blue screen. Masking can be thought of as applying a matte to a layer or operation. You can also create a mask to add to the existing alpha channel of an image. For more information on keying, see Chapter 7, “Keying.”

In Shake, you can mask an effect, such as a color correction, or you can mask a layer. When a layer is masked, the area outside (or inside) of the mask turns black, becoming transparent. This section discusses these techniques, as well as special cases such as masking filters.

Import the masking example images:

- In the Image tab, click the *FileIn* node, go to *doc/pix/masks*, select the *car\_bg.jpg*, *woman\_pre.iff*, *sign\_mask2.iff*, and *car\_mask.iff* images, and click OK.

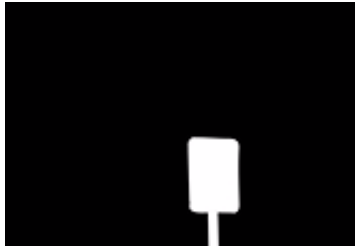
**car\_bg.jpg**



**woman\_pre.iff**



**sign\_mask2.iff**



**car\_mask.iff**



In this discussion, the above images demonstrate both concepts of masking and combine to create the following image.



## Masking an Effect

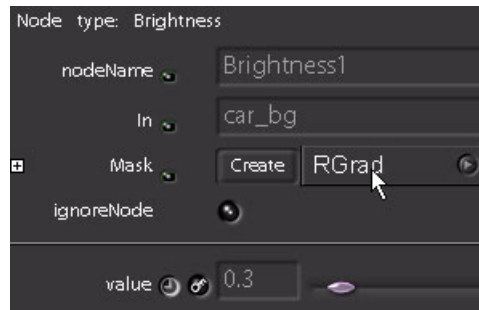
You can use the side input of a node to apply the effect of a node in only a masked area of an image.

### To mask an effect:

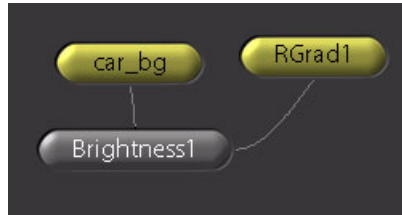
- 1 In the Node View, select the *car\_bg* node.

- 2 Click the Color tab, and then click *Brightness*.  
A *Brightness* node is added to the *car\_bg* image.
- 3 In the *Brightness* parameters, set the value to .3.  
The entire image darkens.
- 4 To create a mask that gives the appearance of a “spotlight,” do one of the following:
  - Create an *RGrad* (in its own branch), and connect the *RGrad* output to the M (mask) input (on the side) of the *Brightness* node.
  - In the *Brightness* parameters, click the Mask shape pop-up menu and select *RGrad*. Use the mask creation button.

**Note:** To create the node type already selected in the Mask shape menu, click Create. For information on the rotoscoping or paint tools and their onscreen controls to draw and edit masks, see Chapter 15, “Painting, Rotoscoping, and Other Image Functions,” on page 527.



An *RGrad* is connected as the mask input for the *Brightness* node, and the masked portion of the image is darkened.



### Adding Custom Nodes to the Mask Shape List

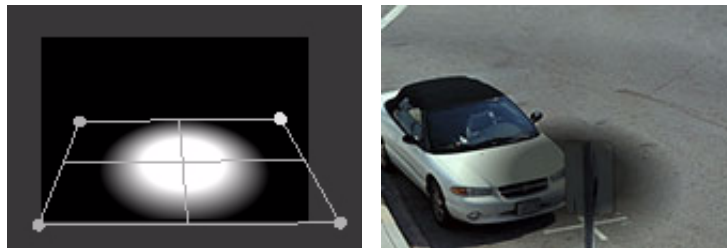
To add your own nodes to the Mask shape list, add a line similar to the following in a *ui.h* file:

```
nuiAddMaskCommand("QuickShape", "QuickShape();");  
nuiAddMaskCommand("QuickPaint", "QuickPaint(0);");
```

For more information on *ui.h* files, see Chapter 17, “Customizing Shake,” on page 623.

- 5 In the Node View, select the *RGrad* node, click the Transform tab, and click the *CornerPin* node.
- 6 Using the onscreen controls and the following image as a reference, adjust the *RGrad* to put the circle in perspective.

For more information on transforming with onscreen controls, see Chapter 12, “Transforms, Motion Blur, and Warping,” on page 435.



- 7 To invert the mask, expand the Mask controls (click the small plus sign [ + ] next to “Mask”) in the *Brightness* node, and enable invertMask.



The mask is inverted, and the masked portion of the image is lightened.



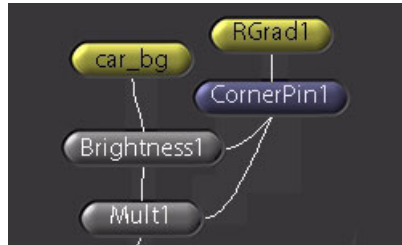
### When Not to Use a Mask

Mask inputs are useful for color corrections and transforms. However, masks should not be used for layer nodes. The logic is the complete opposite of what you think it should be. Honest. As the following example shows, even with color and transform nodes, masks should be used with caution.

#### Using the resulting node tree from the above example:

- 1 Select the *Brightness* node and apply a Color—*Mult* node.
- 2 In the Color controls of the *Mult* node parameters, set the Color to blue.  
A blue tint is created to color correct the dark areas of the background.
- 3 Connect the output of the *CornerPin* node to the M input of the *Mult* node.
- 4 To invert the mask on the *Mult* node, expand the Mask controls and enable invertMask.

The *Mult* (color correction) node is masked and the mask is inverted (like the *Brightness* node), so only the dark areas are tinted blue.



- 5 In the *Mult* node, adjust the Color controls to a deeper blue color.



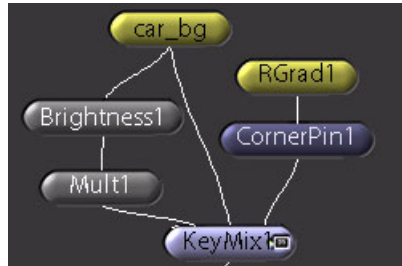
The result appears fine, but there are several problems with the above approach:

- Normally, the *Mult* and *Brightness* functions concatenate. By masking either function, you break concatenation. When concatenation is broken, processing time slows and accuracy decreases.
- Masking twice with the same node (the *RGrad* node in this example) slows processing.
- Your edges get multiple corrections, and tend to degrade. This is evident by the blue ring around the soft parts of the mask.

**Try a different approach:**

- 1 In the Node View, disconnect the masks.
- 2 Select the *Mult* node and add a *Layer-KeyMix* node.
- 3 Connect the *car\_bg* node output to the *KeyMix* node's Foreground input (the second input).

- 4 Connect the *CornerPin* node to the *KeyMix* node's Key input (the third input).



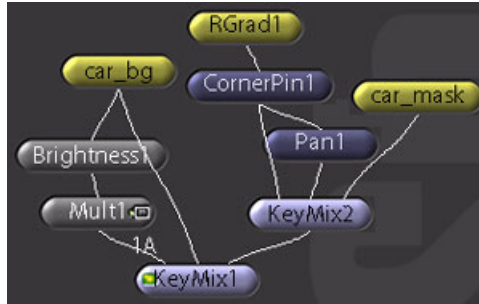
This ameliorates the above problems. The following images show the difference between the two result renders. In the right image that used the *KeyMix* node, there is no blue ring around the soft part of the mask area.



**Mask a pan (transform) to create depth between the street and the hood of the car:**

- 1 Select the *CornerPin* node, and apply a Transform–*Pan*.
- 2 Select the *CornerPin* node again, and **Shift**-click Layer–*KeyMix*.  
A *KeyMix2* node is added to the *CornerPin* node as a separate branch.
- 3 Connect the *Pan* node to the Foreground input of the *KeyMix2* node.
- 4 Connect the *car\_mask* node to the Key input of the *KeyMix2* node.

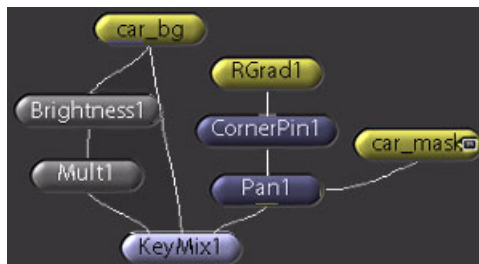
- 5 Connect the *KeyMix2* node to the Key input of the *KeyMix1* node.



- 6 Click the right side of the *KeyMix1* node to show the resulting image in the Viewer.
- 7 Click the left side of the *Pan* node to show the onscreen controls in the Viewer, and load the parameters.
- 8 Using the following illustration as a guide, pan the *RGrad* up slightly to the left.



**Note:** If you had simply connected the *car\_mask* image to the M input of the *Pan* node, rather than using the *KeyMix* method, you would have masked normally concatenating nodes and broken the concatenation between the *CornerPin* and *Pan* functions.



### Using Images Without an Alpha Channel

A masked image does not need an alpha channel. For example, JPEG files do not contain alpha channels. If you use an image with no alpha channel as the mask, you get no effect, since there is no mask.

Instead, switch the mask channel to R, G, or B in the Mask subtree to select a different channel to use as a mask. For the luminance of the image, apply a *LumaKey* node to your mask image and leave the channel at A, or apply a *Monochrome* node and select R,G, or B.

## Masking a Layer

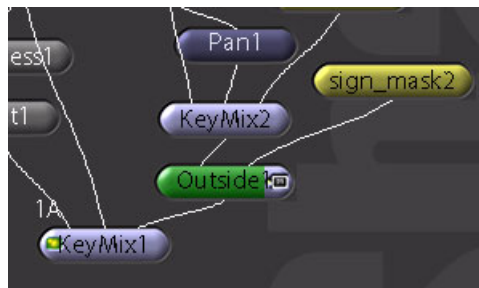
As another form of masking, you can use an image as a holdout matte to cut holes in an image. The typical functions used for this task are *Layer-Inside* and *Layer-Outside*. The *Inside* function puts the first image in only the white areas of the second image's alpha, and the *Outside* function puts the first image in only the black areas of the second image's alpha.

The following example continues with the result node tree from the above example.

Since the sign is further in the foreground of the scene, you do not want the sign to get the brightening effect. Use the *Outside* function to put the light mask outside of the sign mask, in effect punching a hole in the light mask with the sign mask.

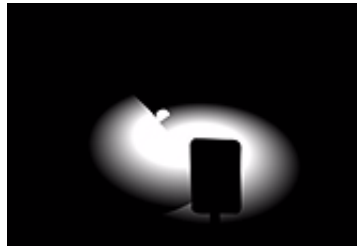
#### To use the Outside function:

- 1 In the Node View, select *KeyMix2* and apply a *Layer-Outside*.
- 2 Double-click the *Outside* node to load it in the Viewer.
- 3 Connect the *sign\_mask2* node to the Background input (the second input) of the *Outside* node.



The light mask is “outside” of the sign mask.

**Outside1**

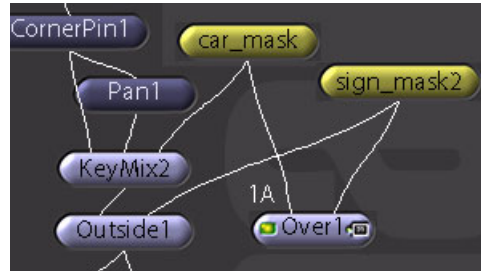


**KeyMix1**



**Do the same with the woman image:**

- 1 Using the following image as a guide, combine the *sign\_mask* and the *car\_mask* with an *Over* (or *Max*) node.

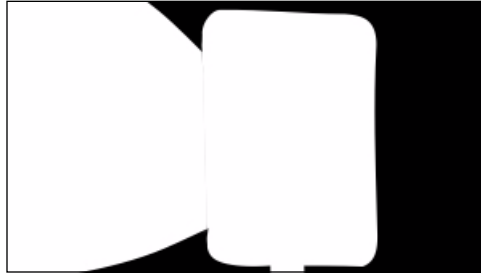


A slight problem occurs, however, as a matte line appears between the two masks.



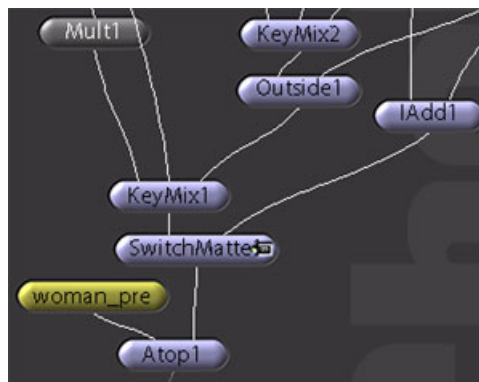
- 2 Select the *Over* node, and **Ctrl**-click *Layer-Add*.

The *Over* node is replaced with the *IAdd* node, and the line disappears.



Next, you can put the woman outside of the new mask *IAdd1* using the *Outside* function. Since this technique was used in the previous example, try a different approach. Use the *Atop* function—similar to *Over*, except the foreground only appears where there is an alpha channel on the background image.

- 3 Add a Layer–*SwitchMatte* function, and connect the *KeyMix1* node to the Foreground input of the *SwitchMatte* node.
- 4 Connect the *IAdd1* node to the Background input of the *SwitchMatte* node.  
The alpha channel is copied from *IAdd1* to *KeyMix1*.
- 5 In the *SwitchMatte* node parameters, enable *invertMatte* to invert the mask (since *Atop* only composites in the white areas in the background alpha mask).
- 6 In the *SwitchMatte* node parameters, disable *matteMult*.
- 7 Select the *woman\_pre* node and apply a Layer–*Atop* node.
- 8 Connect the *SwitchMatte* node to the Background input of the *Atop* node.



If the image looks incorrect, ensure that `matteMult` is disabled, and `invertMatte` is enabled in the *SwitchMatte* node parameters.



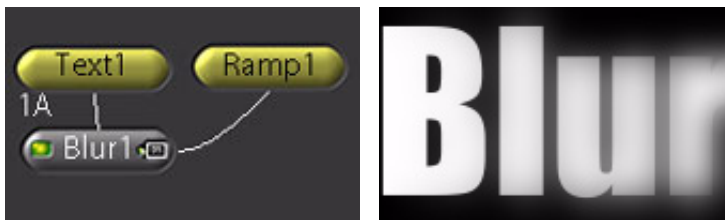
## Masking Filters

Filters have special masked versions of the node that not only mask an effect, but also change the amount of filtering based on the intensity of the second image. These take the same name as the normal filter node preceded by an *I*, for example, *Blur* and *IBlur*. This is much more convincing than using the mask input.

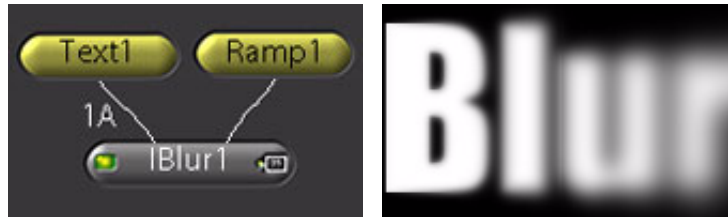
### To mask a filter:

- 1 Create an *Image–Text* node, and type some text in the text field. (Type in the second field labeled “text” since the first field is to change the name of the node.)
- 2 Adjust the `xFontScale` and `yFontScale` parameters so the text fills the frame.
- 3 Create an *Image–Ramp*, and set the `alpha1` parameter to 0.
- 4 Select the *Text* node, and add a *Filter–Blur* node.
- 5 Connect the *Ramp* node to the `M` input of the *Blur* node.
- 6 In the *Blur* parameters, set the `xPixels` and `yPixels` value to 200.

The result looks bad, rather like the following. Notice that the right side of the image merely mixes the completely blurred image with the non-blurred image.



- 7 Select the *Blur* node, and **Ctrl**-click Filter–*IBlur*.  
The *Blur* node is replaced with the *IBlur* node.
- 8 Disconnect the *Ramp* from the blur node's M input, and connect it to the *IBlur* node's second image input.
- 9 In the *IBlur* parameters, set the xPixels and yPixels value to 200.



The result is much nicer—the right side is blurred to 200 pixels, the middle is blurred to 100 pixels, and the left edge has no blur at all.

### The -mask/Mask Function

This function is only used in the script, but is created invisibly when you insert a side-input mask. The *Mask* function masks out the previous operation (in command-line mode) or a node that you specify when in scripting mode. This is how the GUI interaction of setting a mask is saved in script form. For more information see “About Masks” on page 383.

Parameters	Type	Defaults	Function
<i>mask</i>	image		The image to be used as a mask on the result of the first input image.
<i>maskChannel</i>	string	"a"	The channel of the mask image to be used as the mask.
<i>percent</i>	float	100	A gain control applied to the maskChannel. 100 percent is full brightness. 50 percent is half brightness. 200 percent is twice as bright, etc.
<i>invertKey</i>	int	0	A switch to invert the maskChannel. 0 = do not invert 1 = invert
<i>enableKey</i>	int	1	A switch to turn the key on and off. 0 = off 1 = on

## Synopsis

```
Mask(  
    image,  
    image mask,  
    const char * maskChannel,  
    float percent,  
    int invertKey,  
    int enableKey  
);
```

## Script

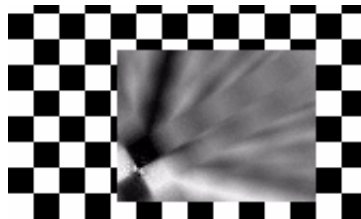
```
image = Mask(  
    image,  
    mask,  
    "maskChannel",  
    percent,  
    invertKey,  
    enableKey  
);
```

## Command Line

*shake -mask image maskChannel percent...*

## Masking Using the Constraint Node

The Layer-*Constraint* node also helps to limit a process. The *Constraint* node mixes two images according to a combination of modes. The modes are Area of Interest (AOI), tolerance, channel, or field. In the following example, the AOI is enabled and the area box is set. Only the area inside of the box of the second image is calculated.



## Constraint

The *Constraint* function is a multifunctional node that restricts the effect of nodes to limited areas, channels, tolerances, or fields. Toggle the type switch to select the constraint method. By selecting different constraint types, certain parameters become active, and others no longer have any effect. This is similar to *KeyMix* in that you mix two images according to a third constraint. *KeyMix* expects an image to be the constraint. Constraint allows you to set other types of constraints.

In many cases, the *Constraint* node also speeds calculation times considerably. The speed always increases with use of the ROI or field mode, and for many functions with use of the channel mode. Channel mode decreases calculation time when the output is a result of examining channels (for example, layer operations), but not when it must examine pixels, such as warps and several filters. The tolerance mode may increase calculation times, as it must resolve both input images to calculate their differences.

Parameters	Type	Defaults	Function
<i>clipMode</i>	int	1	Toggles between foreground and background resolution. 0 = foreground resolution 1 = background resolution
<i>type</i>	int	0	Selects the type of constraint. AOI - Area of Interest (1) – Draws a mixing box. Threshold (2) – Only changes within a tolerance are passed on. Channel (4) – Only specific channels are modified. Field (8) – Only a selected field is modified. Because of the labeling, you can do multiple types of constraining in the script by adding the numbers together; for example, 7 equals Area of Interest, Threshold and Channel are all active.
<i>left, right, bottom, top</i>	float	0, width, 0, height	These are active only if type is set to 1. (See type, above.) They describe a cropping box for the effect.
<i>rTol, gTol, bTol, aTol</i>	float	0, 0, 0, 0	Tolerance values to be used if type is set to 2. (See type, above.)

Parameters	Type	Defaults	Function
<i>tLevel</i>	int	0	Active only when type is equal to 2. This sets the Tolerance to "lo" or "hi." 0 = "lo." Changes are made only if the difference between image1 and image2 is less than the Tolerance values you set. 1 = "hi." Changes are made only if the difference between image1 and image2 is greater than the Tolerance values.
<i>tReplace</i>	int	0	Active when type is set to 2. Toggles whether the entire pixel is replaced, or just the channel that meets the Tolerance criteria.
<i>channels</i>	string	rgba	If type is set to 4 (see type, above), the operation only applies to these channels.
<i>field</i>	int	0	If type is set to 8 (see type, above), effect only applies to 1 field: 0 = even field 1 = odd field
<i>invert</i>	int	0	Inverts the selection, for example, everything beyond a color tolerance is included, rather than below, etc.

### Synopsis

```
image Constraint(
    image Foreground,
    image Background,
    int clipMode,
    int type,
    int left,
    int right,
    int bottom,
    int top,
    float rTol,
    float gTol,
    float bTol,
    float aTol,
    int tLevel,
    int tReplace,
    const char * channels,
```

```
    int field  
);
```

### Script

```
image Constraint(  
    Foreground,  
    Background,  
    clipMode,  
    type,  
    left,  
    right,  
    bottom,  
    top,  
    rTol,  
    gTol,  
    bTol,  
    aTol,  
    tLevel,  
    tReplace,  
    "channels",  
    field  
);
```

### Command Line

*shake -constraint image clipMode etc...*

### Examples

*shake lisa.iff-blur 30 -const lisa.iff 0 1 150 350 200 400*

*shake lisa.iff-solarize -const lisa.iff 0 2 0 0 0 0 "Linear(0,0@1,1@20)" -t 1-20*

### See Also

“KeyMix” on page 179, “Field” on page 224, “Interlace” on page 222, “SwapFields” on page 224, “SetDOD” on page 465, “Mix” on page 185



# Tracking

## Chapter Summary

- About Tracking, Stabilizing, and MatchMoving
- How a Tracker Works
- Tracking Strategies
- Modifying Tracks
- Saving Tracks
- Tracking Functions

## About Tracking, Stabilizing, and MatchMoving

This section discusses tracking in depth, including interface features, workflow issues, and tips on successful tracking and manipulation of tracking data. For specific formats of each node, see the functions listing at the end of this chapter. For a tutorial on how to use the Tracker, see “Lesson Seven: Tracking and Stabilization,” in the *Shake 3 Tutorials*.

There are three tracking nodes in Shake: *Tracker*, *Stabilize*, and *MatchMove*.

- *Tracker*: The *Tracker* node does not transform the input image. It is used only to generate tracks that can then be referenced by the *MatchMove* and *Stabilize* nodes, or nodes such as *Move2D* and *Pan* with standard linking techniques.
- *Stabilize*: The *Stabilize* node is used to correct unwanted movement in a shot. You can also use the *Stabilize* node for matchmoving with the *inverseTransform* parameter. Use of the *Stabilize* node's *inverseTransform* parameter is recommended rather than use of the *MatchMove* node, since you usually composite later with an *Over* node. This technique also gives you proper pass-through of onscreen controls for more intuitive control.


- *MatchMove*: The *MatchMove* node allows two input images—a Foreground (the first node input) and a Background (the second node input). By tracking one, two, or four points on the background, the foreground can be set to match the movement of the background. Placing a logo on the side of a moving truck is the classic example of a matchmove. The single advantage the *MatchMove* node has over the *Stabilize* node is that you can immediately test your track results.

**Note:** You can also attach a tracker to a *RotoShape* or paint stroke. For more information, see “Attaching a Tracker to a Paint Stroke” on page 532 and “Attaching a Tracker to a RotoShape” on page 548.

## Tracking Workflow

This is a general overview of the steps required to generate a track. The steps are further detailed in the following sections.

### General Tracking Workflow:

- 1 Play your background clip several times to determine a good tracking point.
- 2 In the Transform tool tab, select the tracking node (*Tracker*, *Stabilize*, or *MatchMove* node) and connect the node to the background clip.
- 3 Ensure the onscreen controls are visible in the Viewer.
- 4 Go to the frame that you want to start the track.
- 5 Position the tracker on the point you want to track, and adjust the reference pattern and the search region.
- 6 Ensure the trackRange parameter reflects the frame range you want to track. For *FileIn* nodes, it takes the range of the clip. For Shake-generated elements, you must provide a frame range, for example, 1-50.
- 7 Click Track Forward or Track Backward  to start the track processing. (Ideally, this needs to be done once, but never really happens that way.) The track generates animation curves.

**Note:** To stop a track, click the mouse.

### Stabilize Additions



If you are using the *Stabilize* node, include the following additional steps with the above General Tracking Workflow steps.

- 1 Determine if you need one-, two-, or four-point tracking.  
**Note:** For two- or four-point tracking, select 2 pt or 4 pt in the trackType parameters.
- 2 In the *Stabilize* parameters, set applyTransform to active in order to stabilize the plate.
- 3 If you are using *Stabilize* for matchmoving, toggle the inverseTransform parameter from stabilize to match.

### MatchMove Additions

If you are using the *MatchMove* node, include the following additional steps with the above General Tracking Workflow steps.

- 1 Attach the foreground element to the first input of the *MatchMove* node.
- 2 Determine if you need one-, two-, or four-point tracking.
- 3 In the *MatchMove* parameters, set the outputType to *Over* (or another compositing operation).
- 4 Set the applyTransform parameter to active (to match the motion).

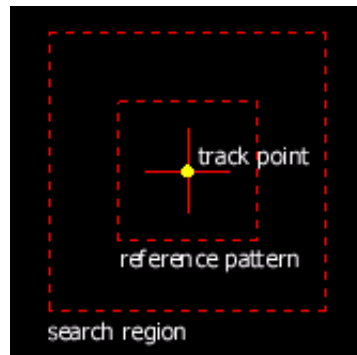
**Note:** If you are four-point tracking (make sure 4 pt is enabled in the trackType parameter), and you want to attach four arbitrary points on the foreground image to the background, click BG  to show the foreground . Next, position the four corners on the foreground element. These represent the corners that are plugged into the four tracking points. Click FG again to show the background.

- 5 You may have to set the outputType parameter to *Over* again, or the compositing mode you selected in step 3.

### Onscreen Tracking Controls

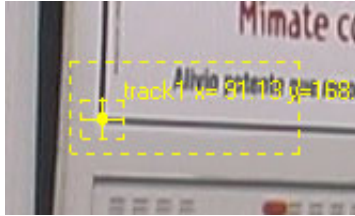
The *Tracker*, *Stabilize*, and *MatchMove* nodes share interface controls.

- A tracker consists of three onscreen controls:
  - Search region—the outer box
  - Reference pattern—the inner box
  - Track point—the center cross hair

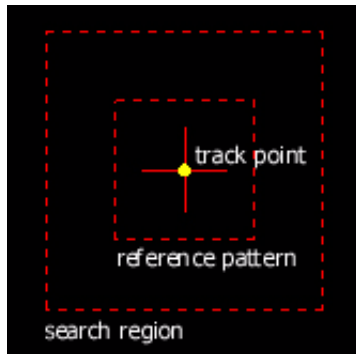


- To move the tracker, click a blank area inside of the search region or the track point, and drag. To resize the tracking region or the reference pattern, click and drag a corner and the boxes uniformly scale in X or Y. The larger the search region, the slower the track.

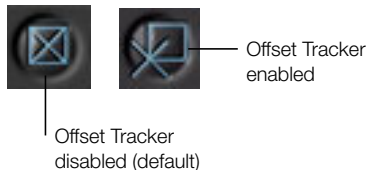
- To scale the search region non-uniformly, click and drag an edge of the search region. This is good for “leading” the track point. For example, a bus in a clip moves to the right. Scale the tracker search region to the right as well, since it doesn’t make sense to scan to the left of the current track pattern for a pattern match. (This is demonstrated in “Lesson Seven: Tracking and Stabilization,” in the *Shake 3 Tutorials*.)



- For four-point *MatchMove* and *Stabilize*, the trackers should be positioned in a counterclockwise order, starting in the lower-left corner. This ensures the proper alignment of your element when the transformation is applied.



- To offset the tracking area from where the keyframe is actually saved (for example, the tracking point moves offscreen or is obscured), click the Offset Tracker button.



When offset tracking is enabled, you can move the search pattern without moving the track key. When your tracking reference pattern goes offscreen, select a different tracking point. To reset the search area back to the key, click Reset Tracker.



Reset Tracker button

- To turn off a specific tracker, click the Visibility button located next to the track name inside the track node parameter list.







Visibility button



All visible trackers are processed when you click the track button, regardless of whether they have previously tracked keys.

A limitation of the *MatchMove* node is that it does not handle the pass-through of upstream onscreen controls, so those controls are in their nontransformed state.

### Tracking Viewer Buttons

The following table describes the Viewer toolbar buttons that become available with the tracking nodes.

Button	Function
	Track Backward/Track Forward. Click to start the tracking process. All visible trackers attempt to lay down new tracking keyframes. Tracking continues until one of the trackRange frame limits is reached—the upper limit if you are tracking forward or the lower limit if you are tracking backward, or until your correlation falls below your failureTolerance setting.
	Offset Track–Off. The track search region and the tracking point are linked. If you move one, the other follows.
	Offset Track–On. The track search region and the tracking point are offset from each other. If your original sample pattern becomes obscured, offset the search region to a different area. Your keyframes are still saved relative to the original spot.
	Reset Track. Returns an offset track search region back to the track point.

Button	Function
	<p>Track Display. Click to toggle between track display options:</p> <ul style="list-style-type: none"> <li>■ The left button displays all trackers, curves, and keyframes.</li> <li>■ The middle button displays just the trackers and keyframes.</li> <li>■ The right button displays just the trackers.</li> </ul> <p>You can also control visibility of individual trackers as described below.</p>
	<p>FG/BG. These buttons appear on the Viewer toolbar when the <i>MatchMove</i> node is active. Click FG/BG to toggle between adjusting the track points on the background (BG), or the corners of the foreground (FG) that are pinned to the track points (when 4 pt [four-point tracking] is enabled in the trackType parameter).</p>

Next to each track name is a Color Picker and a Visibility toggle.

To change the color of a tracker, click the Color Picker swatch.

To show or hide a tracker, click Visibility. When Visibility is enabled for a tracker, it tracks each time you click a track button (even if previously tracked). When Visibility is disabled for a tracker, it is not processed when you click a track button.



### Tracking Parameters

All trackers share the following parameters:

Parameter	Notes
<i>trackRange</i>	<p>The trackRange parameter is the potential frame range limit of your track. By default, the range is set to the clip range. For generated elements such as <i>RGrad</i>, it takes a range of 1. You can set new limits using Shake's standard range description, for example, 10-30x2. If you stop tracking and start again, it starts from the current frame until it reaches the lower or upper limit of your trackRange, depending on whether you are tracking forward or backward.</p>

Parameter	Notes										
<i>subPixelResolution</i>	<p>The subPixelResolution parameter determines the resolution of your track. The smaller the number, the more precise the track.</p> <p>Possible values:</p> <table> <tr> <td>1</td><td>Area is sampled at every pixel. Not very accurate or smooth, but very fast.</td></tr> <tr> <td>1/4</td><td>Area is sampled at every .25 pixels (16 times more than with a sampling of 1).</td></tr> <tr> <td>1/16</td><td>Area is sampled at every .0625 pixels (256 times more than with a sampling of 1).</td></tr> <tr> <td>1/32</td><td>Area is sampled at every .03125 pixels (1024 times more than with a sampling of 1).</td></tr> <tr> <td>1/64</td><td>Area is sampled at every .015625 pixels (4096 times more than with a sampling of 1).</td></tr> </table>	1	Area is sampled at every pixel. Not very accurate or smooth, but very fast.	1/4	Area is sampled at every .25 pixels (16 times more than with a sampling of 1).	1/16	Area is sampled at every .0625 pixels (256 times more than with a sampling of 1).	1/32	Area is sampled at every .03125 pixels (1024 times more than with a sampling of 1).	1/64	Area is sampled at every .015625 pixels (4096 times more than with a sampling of 1).
1	Area is sampled at every pixel. Not very accurate or smooth, but very fast.										
1/4	Area is sampled at every .25 pixels (16 times more than with a sampling of 1).										
1/16	Area is sampled at every .0625 pixels (256 times more than with a sampling of 1).										
1/32	Area is sampled at every .03125 pixels (1024 times more than with a sampling of 1).										
1/64	Area is sampled at every .015625 pixels (4096 times more than with a sampling of 1).										
<i>matchSpace</i>	<p>The pixels are matched according to the correlation between the selected color space—luminance, hue, or saturation. When an image has roughly the same luminance, but contrasting hues, you should switch to hue-based tracking.</p> <p>You can also adjust the weight of the color channels in the matchSpace subtree.</p>										
<i>referenceTolerance</i>	<p>A tracking correlation of 1 is a perfect score—there is an exact match between the original reference frame and the sampled area. When the referenceTolerance is lowered, you accept greater inaccuracy in your track. If tracked keyframes are between the referenceTolerance and the failureTolerance, they are highlighted in the Viewer. Also, in some cases, referenceBehavior is triggered if the tracking correlation is below the referenceTolerance.</p>										

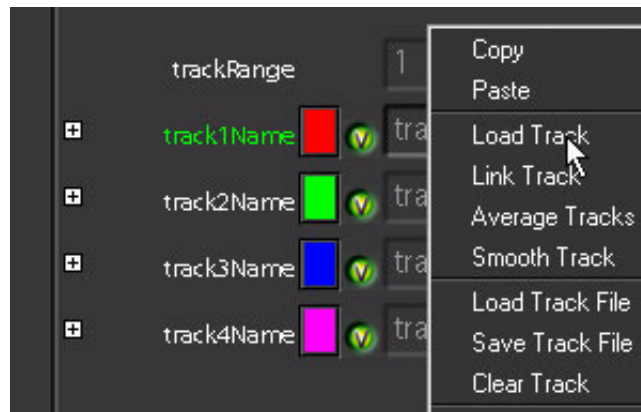
Parameter	Notes
<i>referenceBehavior</i>	This behavior dictates the tracking area reference sample. By default, the reference pattern is the first frame that the track is started, not necessarily the first frame of the trackRange. The last two behaviors in the referenceBehavior list measure the tracking correlation and match it to the referenceTolerance to decide an action.
use start frame	The new samples are compared to the reference pattern from the first frame of the track. If you stop tracking midway, and start again at a later frame, the later frame is used as the reference sample.
update every frame	The source sample is updated from the previous frame. This usually creates an inherent drift in the track, as tiny errors accumulate. This method is for movements that have drastic changes in perspective and scale.
update from keyframes	If you are using a failureBehavior of "predict location and don't create keys" or "don't predict location," a keyframe is not necessarily saved every frame. In this case, you may only want to update from the last frame with a valid keyframe.
update if above reference tolerance	This updates the reference sample from the previous frame if the correlation is above the referenceTolerance. The intent is to update every frame unless you know the point is obscured. If you use a predict mode and know there are obstructions, it keeps the reference area from updating if the point is completely obscured.

Parameter	Notes	
	update if below reference tolerance	This updates the reference sample from the previous frame if the correlation is below the referenceTolerance. This basically says, "If I can't get a good match, then resample." This is excellent for gradual perspective and scale shifts in the tracking area.
<i>failureTolerance</i>	If the correlation of a track falls below this value, it initiates the failureBehavior.	
<i>failureBehavior</i>	What occurs when the correlation drops below the failureTolerance:	
	stop	The tracker stops if the correlation is less than the failureTolerance. You can also press <b>Esc</b> to manually stop tracking.
	predict location and create key	If a failure is detected, then the tracker predicts the location of the keyframe based on a vector of the last two keyframes, and continues tracking in the new area.
	predict location and don't create key	Same as above, but it merely predicts the new search area and does not create new keyframes until a high correlation is obtained. This is excellent for tracked objects that pass behind foreground objects.
	don't predict location	In this case, the tracker merely sits in the same spot looking for new samples. New keyframes are not created.
	use existing key to predict location	This allows you to manually create keyframes along your track path. You then return to the start frame and start tracking. The search pattern starts looking where the preexisting motion path is.

Parameter	Notes
<i>limitProcessing</i>	This creates a Domain of Definition (DOD) around the bounding boxes of all active trackers. Only that portion of the image is loaded from disk when tracking, so the track is faster. This has no effect on the final output image.
<i>preProcess</i>	This toggles on the preprocessing for the tracking area. This applies a slight blur to reduce fluctuations due to grain. To control the blur amount, open the preProcess subtree.
<i>blurAmount</i>	The amount of blur applied when preprocessing.
<i>trackNName</i>	The name of the track. To change the name, click in the text field and enter the new name.
<i>trackNX/Y</i>	The actual track point in X and Y. Use this to link a parameter to a track point.
<i>trackNCorrelation</i>	The correlation value of that key to the original sample. A score of 1 is a perfect score. 0 is a very, very, very, very bad score.
<i>trackNWindow Parameters</i>	These multiple parameters control the windowing of the tracking box, and are not relevant to exported values.

### Tracking Right-Click Menu (trackName Text Field)

Right-click in the text field of a trackName to show the special menu to manipulate your tracks.

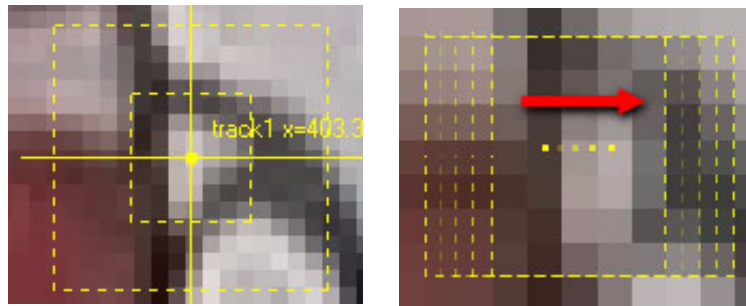


Function	Notes
Copy/Paste	The standard copy and paste commands.
Load Track	Displays a list of all currently existing tracks. Select one, and a copy of that track is loaded into both the X and Y parameters of the current tracker.
Link Track	Displays a list of all currently existing tracks. A link is made from the current tracker to the tracker you select in the Select Track pop-up window. Therefore, if you modify the tracker selected with the Link command, the current tracker is updated. If you delete the tracker you linked to, you lose your tracking data.
Average Tracks	Displays a list of all tracks. Select up to four tracks that you want to average together, and click OK. An expression is entered into the current trackX and Y fields that links back to your averaged tracks. You cannot choose to average your current track—it is deleted by this action.
Smooth Tracks	Displays a slider to execute a blur function on your tracking curves; the smooth value is the number of keyframes considered in the smoothing operation. To view the curves, open the trackName subtree and click the load parameter button to load the parameters into the Curve Editor. If you use Smoothing and Averaging operations as your standard workflow, it is recommended that you generate your tracks first with the <i>Tracker</i> node, and then link the tracks to a <i>MatchMove</i> or a <i>Stabilize</i> node.
Load Track File	Loads a Shake-formatted track file from disk. See the end of the tracking overview for the format of a track file.
Save Track File	Saves a Shake-formatted track file to disk.
Clear Track	Resets the current tracker. To reset the entire function, right-click an empty area of the Parameters tab and select Reset All Values.

## How a Tracker Works

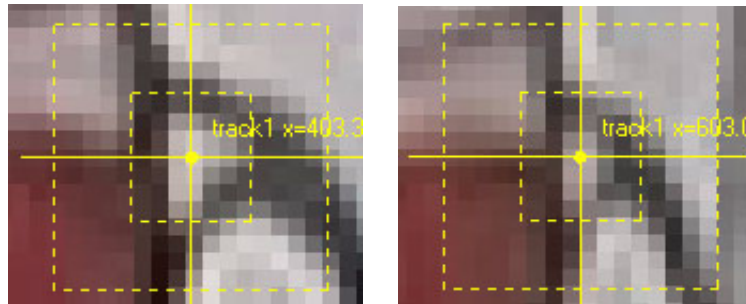
The tracker works by taking a small snapshot of pixels from one frame, called the reference pattern. This is represented by the inner box on the onscreen tracker. The tracker then advances to the next frame, and samples the area inside the search region (the larger tracker box). When sampling inside the search region, the tracker positions a box the same size as the reference pattern at the pixel in the first row, first column, and takes a sample. It then advances to the next pixel (or subpixel) column in the search region and takes a second sample. For every sample the tracker takes, it assigns a correlation value by comparing it to the reference pattern. When it has taken all of the samples, it assigns the new tracking point to the sample with the highest correlation value. It repeats this process on the following frames until the track range is complete.

The `referenceBehavior` controls if and when the reference pattern is updated. By default, the reference pattern is set to use the start frame (the first frame you start tracking), so even the last frame compares the samples to the first frame. Other modes can change this behavior.



The amount of samples taken in the search region is determined by the `subPixelResolution` parameter. A `subPixelResolution` of 1 positions the reference pattern box at every pixel to find a sample. This is not accurate, because most movement occurs at the subpixel level—the movement is more subtle than one pixel across, and therefore is factored in with other objects in the calculation of that pixel's color. The next resolution down in `subPixelResolution`,  $1/4$ , advances the reference pattern box in .25 pixel increments, and is more accurate.

The example here is a theoretical 1/2 resolution since the pattern is advanced in .5 pixel increments. Keep in mind that the lower the number, the more samples taken. At 1/4 resolution, it takes 16 times more samples per pixel than at a resolution of 1. At 1/64, it takes 4096 times more samples per pixel.



It is because of this that most trackers don't handle significant rotational movement very well—they (Shake's included) only test for panning changes, not rotational. If they did, they would have to multiply the amount of panning samples by the amount of degrees for the number of samples to take, which would be prohibitively costly at this stage. If you are tracking an object with rotational movement, try using a referenceBehavior set to update every frame. This means that the reference pattern is updated at every frame, so you are only comparing a frame with the frame before it, and not the first frame.

Also keep in mind that manual adjustments are standard for tracking problems.

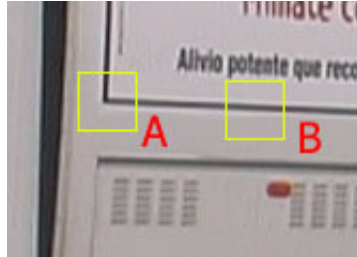
## Tracking Strategies

Despite all of the tracking demos we love to see (and show), tracking is rarely a magic bullet that works on the first attempt. This section discusses some strategies to help you get accurate tracks.

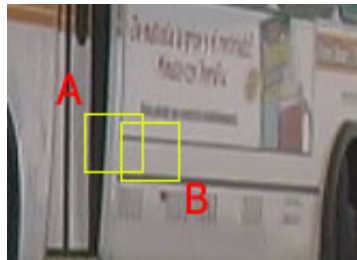
### Picking a Good Reference Pattern

The ideal reference pattern is one that doesn't change perspective, scale, or rotation, and does not move offscreen or become obscured by other objects. The ideal pattern also maintains overall brightness or color, is very high contrast, and is distinct from other patterns in the same neighborhood. Meanwhile, in the real world, we have to contend with all of these factors in our footage.

In this example, two possible reference pattern candidates include the corner at area A, or anywhere along the line near B. Area A is the better choice, since B can be matched anywhere along the horizontal black line, and probably on the dark line that is below B. One rule is to avoid similar patterns horizontally or vertically from your candidate pattern. If you can easily find one, so can the tracker.

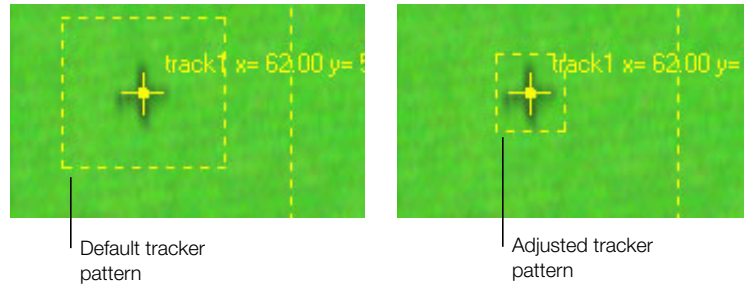


Another potential candidate is the word “Alivia” in the sign. However, as the clip advances, the text becomes an indecipherable blur. The A and B points also move closer together—the clip has significant scaling on the X axis and some scaling in Y due to the perspective shift. Although the overall brightness has dropped, contrast remains relatively high at the A point. The A point is a good candidate for tracking with referenceBehavior set to “update if below reference tolerance” or “update every frame.”



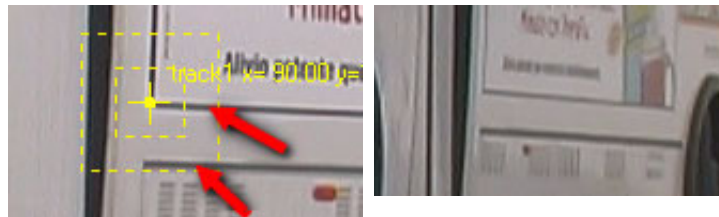
The reference sample should also be relatively constant and unique over time. Flickering lights, for example, are not good reference patterns. If the lights were regular enough, you could try to set the trackRange to match the flicker, that is, 1-5, 10-15, 20-25, and so on. Granted, this is awkward. A better solution is to set your failureBehavior to “predict location and don’t create key.”

The following example shows a track marker placed on a TV screen so the client could place an image on the TV. The default tracker reference pattern is unnecessarily large. The only interesting detail is the black cross in the middle. Otherwise, most of the pattern matches up with most of the rest of the image—green. To adjust for this, limit the reference pattern to more closely match the black crosshair.

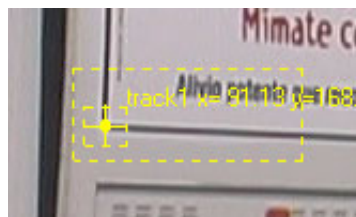


### Picking a Good Search Region

You should also suit your search region to match the clip's movement and the patterns near the reference pattern. With the default settings, the bus example clip does not track the lower-left corner of the sign very well. This is because the middle black horizontal line can easily be matched with the black line at the very base of the search region in later frames of the clip. The X axis is squeezed so much that vertical details disappear. Additionally, since the bus is moving to the right, there is no point in wasting cycles to the left of the point. Remember, the larger the search region, the more samples the tracker must take.



The corrected search region, illustrated below, is now high enough to not include the lower black line, and extends minimally to the left.



### Manually Coax Your Track

Another technique you can use is to manually insert tracking keyframes. For example, if you have 100 frames to track, you can put in a keyframe every 5 or 10 frames with the Autokey feature. A helpful trick is to set an increment of 5 or 10 in the Time Bar. Press the **Left Arrow** or **Right Arrow** to jump by the increment amount.

**Note:** To add a keyframe without moving an onscreen control, for example, to create a keyframe at frame 20 with the same value as a keyframe frame 19, turn Autokey off and then back on.

Once your keyframes are manually entered, return to frame 1, and set the `failureBehavior` to “use existing key to predict location.” The tracker searches along the tracker’s preexisting motion path to find matching patterns.

### Identify the Channel With the Highest Contrast

The tracker works best with a high-contrast reference pattern. The human eye sees contrast as represented by value. However, you may sometimes have higher contrast in saturation or hue, so switch to a different color space with the `matchSpace` parameter in the tolerances subtree. A shot may also have a higher contrast in a specific RGB channel than in the other channels. For example, the blue channel may have a larger range than the red or green channel. In such a case, apply a *Color-Reorder* node to the image, set the channels parameter to *bbb*, and then perform the track with luminance selected as the `matchSpace`.

### Delog Logarithmic Cineon Files

Because the logarithmic to linear conversion increases contrast, you may have better results on linear data than on logarithmic data. Of course, maybe not. Log files are fun that way.

### Do Not Reduce Image Quality

Ideally, you should track an image with the most amount of raw data. This means if you apply a *Brightness* node set to .5 to your image, you lose half of the color information. Therefore, track the image before the *Brightness* function is applied.

### Do Not Track Proxies

Do not ever track with proxy settings. Proxies are bad for two reasons: First, you filter the image so detail is lost. Second, you automatically throw data away because of data round-off. For example, if using 1/4 proxy, you automatically throw away four pixels of data in any direction, which means an 8 x 8 grid of potential inaccuracy.

### Do Not Track the Image at its Best Quality

Yup, that’s right, the exact opposite of what was just stated above. Isn’t compositing fun? It is often helpful to apply a *Color-Monochrome* node to your image and drop the blue channel out if you have particularly grainy footage. Another good strategy is to activate the `preProcess` flag in the tracker. This applies a small blur to the footage to reduce irregularities due to film grain.

In some cases, you may want to modify your images to improve contrast in the reference pattern, either with a *ContrastLum* node or *ContrastRGB* node. Since you use this image to generate tracks, you do not need to keep the contrast image for the rest of your composite.

### Tracking Images With Perspective, Scale, or Rotational Shifts

For images with significant change in size and angle, you can try two different referenceBehaviors, “update if below reference tolerance” or “update every frame.” The second choice is the more drastic, because you get an inherent accumulation of tiny errors when you update every frame. Therefore, try “update if below reference tolerance” first.

Another strategy is to jump to the midpoint frame of the clip and track forward to the end frame of the clip. Then return to the midpoint frame and track backward to the beginning of the clip.

A second strategy is to apply two *Stabilize* nodes. The first *Stabilize* node can be considered a rough stabilize. The second *Stabilize* then works off of the first, and therefore has a much better chance of finding acceptable patterns. Since the *Stabilize* nodes concatenate, no quality is lost.

### Tracking Obscured or Off-Frame Points

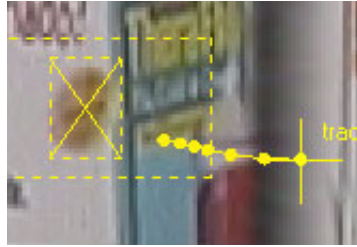
There are two basic techniques to correct track points that are obscured by moving off screen or an object passing in front of them.

The first strategy is to use a different failureBehavior, either “predict location and create key” or “predict location and don’t create key.” The first setting is good for nice linear behavior—it continues to lay down keyframes following the vector of the last two valid keyframes, so the two frames prior to the pattern become obscured. It is excellent for points that go off screen and never reappear. The second setting is a better choice for patterns that reappear because it continues to search on a vector, but only creates a keyframe if it finds another acceptable pattern. You have an even interpolation between the frame before the pattern was obscured, and the frame after it is revealed again.

The above strategies only work when the clip contains nice linear movement. The second strategy is to use Offset Tracker  (on the Viewer toolbar).

When these are offset, you can split the reference pattern and search region away from the track point. In the following example, the track is obscured by a lamp post, so the search region (not the point, just the region) is moved to a nearby pattern and tracking continues until the original pattern reappears.

Even though one region is examined, the points are saved in another region. The second tracking pattern should travel in the same direction as your original pattern.




## Modifying Tracks

A track can be modified in several ways to massage the tracking data. You can manually modify a track in the Viewer or in the Curve Editor, average tracks together, smooth tracks to remove noise, or remove jitter to smooth out a camera movement.

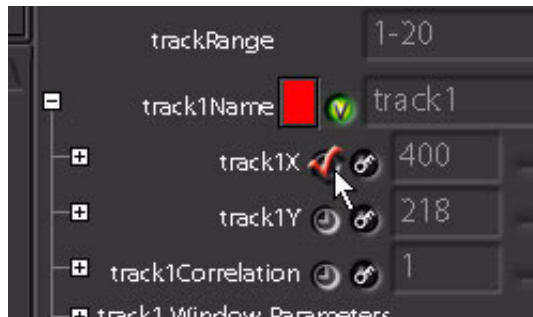
### Manually Modifying Tracks

To manually adjust a tracking point onscreen, enable Autokey  in the Viewer.

**Note:** To add a keyframe without moving an onscreen control, for example, to create a keyframe at frame 20 with the same value as a keyframe frame 19, turn Autokey off and then back on.

Use the **+** and **-** keys (next to the **Delete / Backspace** key) to zoom in and out of the clip. The zoom follows the cursor, so place the cursor on the key point in the Viewer and zoom in. Press **Home** or click Home  to return to normal view.

You can also adjust a tracking curve in the Curve Editor. In the tracking node parameters, open the trackName subtree and click the clock icons to load a parameter into the Curve Editor. In the following example, track1X (only the X parameter) is loaded into the Editor.



### Averaging Tracks

A common technique is to track forward from the first frame to the last, create a second track, and track backward from the last frame to the first. These two tracks are then averaged together to (hopefully) derive a more accurate track. If you plan to use this method, it is recommended that you use the *Tracker* node, and then load your tracks into *Stabilize* or *MatchMove* with the Load or Link Track functions (in the trackName text field right-click menu).

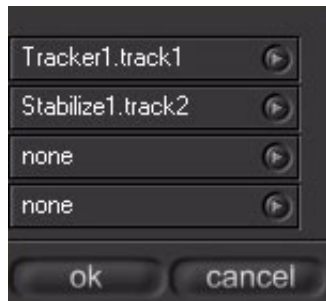
#### To average tracks:

- 1 Apply a *Tracker* node, and in the bottom of the *Tracker* parameters, click Add.
- 2 Click Add again.

There are a total of three trackers.

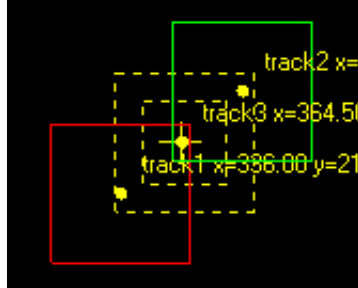
**Note:** You could also potentially use tracks from any other *Stabilize*, *MatchMove*, or *Tracker* node.

- 3 Create tracks on track1 and track2.
- 4 Right-click on track3 and select Average Tracks.
- 5 Select Tracker1.track1 and Tracker1.track2 as the first two inputs, respectively, and leave the last two inputs set to none. The following illustration shows *Stabilize1* to remind you that any tracking node can be a track source.



- 6 Click OK.

The third track, track3, is in the middle of the first two tracks.



This works by creating an expression in both the track3X and track3Y parameters. The expression for the X parameter looks like this:

$(Tracker1.track2X + Tracker1.track1X)/2$

Because these are linked to track1 and track2 on the *Tracker1* node, do not delete these. For more information on linking, see “Linking to Tracking Data” on page 421.

You can average up to four tracks at one time, but you can of course continue to manipulate your tracks with further functions, including Average Tracks.

### Smoothing Track Curves

You can smooth a track with the Smooth Tracks function in the *Tracker* parameters. Prior to smoothing the curve, you may want to copy the track (as a backup) to another tracker with the Load Track function on the second tracker.

#### To smooth a track curve:

- 1 Right-click the track you want to smooth and select Smooth Tracks.  
The Smooth Track window appears.

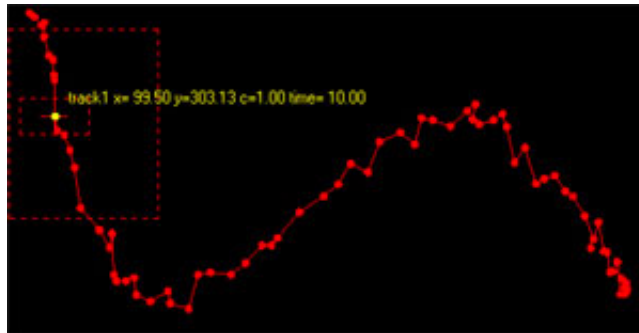


- 2 Enter a value (or use the slider) in the smoothValue field.

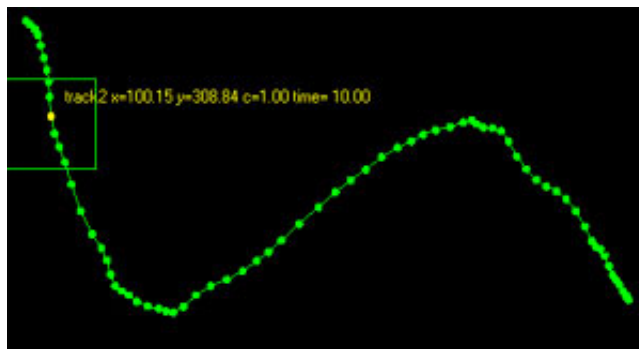
The default is 5, which means that 5 track points centered on the currently evaluated point are used to compute the current point's new, smoothed value. This is a standard Gaussian (bell-curve type) filter. In other words, if you leave it at 5, when the value of frame 12 is computed, frames 10, 11, 12, 13, and 14 are considered. If set to 3, it uses frames 11, 12, and 13.

The larger the smoothValue, the more points considered (and thus more calculations done) for every point in the curve. Even values for smoothValue use the next largest odd number of frames, but the end ones do not contribute as much.

As an example, the following is a noisy track curve (before smoothing):



After the track curve is smoothed:



### Linking to Tracking Data

Referencing track point data works similarly to referencing any other parameter within Shake. The twist here is that since you can rename the track point, you can change the name of the referred parameter. For example, if you have a *Tracker* node named *Tracker1*, and a track point set to its default name “track1,” do one the following:

- To reference the X track data, use: *Tracker1.track1X*
- If you change the name of the track point to “lowerleft,” then the reference is changed to *Tracker1.lowerleftX*.  
This applies to the Y data as well.
- You can also use Link Track (in the trackName text field right-click menu) to link one track to another track, simultaneously linking the X and Y curves.

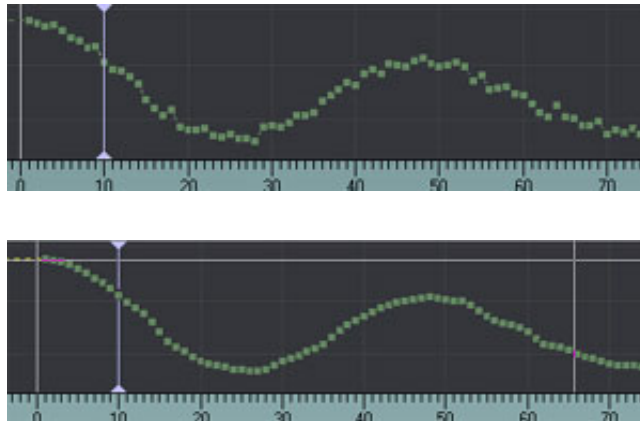
### Removing Jitter on a Camera Move

The following technique is useful when the clip contains a camera move that you want to preserve, but there is a lot of jitter in the shot. You need to stabilize the shot, but only by the small amount that is the actual jitter. To do this, you can combine the techniques mentioned above.

#### To remove jitter and preserve the camera move:

- 1 Track the plate with a *Tracker* node (for this example, called *Tracker1*).
- 2 In the *Tracker* parameters, click Add to create a second tracker in the same node.
- 3 Load track1 into track2 using Load Track (right-click in the track2 text field).
- 4 Right-click track2 and select Smooth Track.
- 5 Enter a smooth value, click Apply, and then click Done.

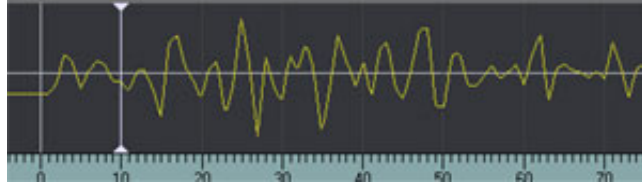
At this point, you have a track and a smoothed version of that track. The following example shows the Y curves of the two tracks.



- 6 Create a *Stabilize* node.
- 7 In the *Stabilize* node, expand track1, and enter the following expression in the track1X and track1Y parameters:
  - In track1X, enter: `Tracker1.track1X - Tracker1.track2X`
  - In track1Y, enter: `Tracker1.track1Y - Tracker1.track2Y`

You get only the difference between the two curves—the jitter.

**Note:** This illustration is scaled differently in Y than the above illustrations.



- 8 In the *Stabilize* node parameters, set `applyTransform` to active.

The plate is only panned by the amount of the jitter, and maintains the overall camera move.

### Working With Two-Point Tracking

There are several additional options when working with 2-point tracking. You can choose to pan, scale, and/or rotate the image. When setting the `applyScale` and `applyRotate` parameters, you have three choices: “none,” “live,” and “baked.”

In the `applyScale` and `applyRotate` parameters, enable “live” to use the mathematical calculation of the four curves (`track1X`, `track1Y`, `track2X`, and `track2Y`)—live mode takes the `track1` and `track2` expressions and creates scale or rotational curves you can view in the Curve Editor.

To convert curves into editable data (generate keyframes), click “baked” in the `applyScale` and `applyRotate` parameters.

**Note:** Two-point (or four-point) tracking is only available in the *MatchMove* and *Stabilize* nodes (not the *Tracker* node).

## Saving Tracks

Once a track is complete, you can save the track (in the *Tracker*, *MatchMove*, or *Stabilize* nodes) to apply to another tracking function, a paint stroke, or to a *RotoShape*.

**Note:** Although you cannot attach a saved track file to a paint stroke or *RotoShape*, you can add a tracking node to your script, load the saved track file in the tracking node, and then apply the track to your shape or paint stroke. For more information on attaching trackers to strokes and shapes, see “Attaching a Tracker to a Paint Stroke” on page 532 and “Attaching a Tracker to a RotoShape” on page 548.

### To save a track file:

- 1 Perform your track using the *Tracker*, *MatchMove*, or *Stabilize* node.
- 2 In the tracker Parameters tab, right-click the `trackName` field and select `Save Track File`.
- 3 In the “Save tracking data to file window,” navigate to the directory in which you want to save the track file, and enter the track filename.

- 4 Click OK.

The track file is saved.

**To load a track file:**

- 1 Add a *Tracker*, *MatchMove*, or *Stabilize* node.
- 2 In the tracker Parameters tab, right-click the trackName field and select Load Track File.
- 3 In the “Load tracking data from file” window, navigate to the track you want to load.
- 4 Click OK.

The track is applied to the tracker.

## Tracking File Format

The following is a sample saved track file for use with Save Track File or Load Track File (right-click in the track name text field).

**TrackName track1**

Frame	X	Y	Correlation
1.00	462.000	210.000	1.000
2.00	405.000	192.000	1.000
...etc...			

**Note:** The Save Track File or Load Track File command is different from the Load Expression command. The Load Expression command is available in the right-click menu of any node’s text field and looks for preformatted Shake expressions. For example, to load the above information into a *Move2D* node, you must load two files, one for the xPan parameter and one for the yPan parameter. Their formats are similar to the following:

*Linear(0,0462@1, 405@2, ....)*

and

*Linear(0,210@1, 192@2, ...)*

## Tracking Functions

The following section includes the tracking functions located in the Transform Tool Tab. For information on other Transform functions, see Chapter 12, “Transforms, Motion Blur, and Warping,” on page 435.

MatchMove

The *MatchMove* function is a dedicated tracking node to match a foreground element to a background element using 1-point (panning), 2-point (panning, scaling, or rotation), or 4-point (corner pinning) tracking. Unlike the *Tracker* or *Stabilize* nodes, *MatchMove* can perform the compositing operation, or you can pass the transformed foreground out for further modifications (blur, color corrections, etc.) before you do a composite.

The *MatchMove* node can generate up to four tracking points, or you can load in other tracks (created in Shake or on disk). To load another track, right-click in a trackName text field, and select Load Track.

The following table includes a description of the *MatchMove* parameters. Some of the *MatchMove* parameters that are shared with other tracking node parameters also appear in the “Tracking Parameters” section above.

Parameters	Type	Defaults	Notes
<i>applyTransform</i>	int	0	The foreground element is only transformed if applyTransform is active.
<i>trackType</i>	string	1 point	<p>You can do 1-point, 2-point, or 4-point matchmoves. Different options appear with the different types.</p> <ul style="list-style-type: none"><li>■ 1 point—Pans the foreground element to match the tracking point. You can optionally turn off the X or Y movement.</li><li>■ 2 point—Pops up two additional parameters, with the option to also match scaling and rotation of the background element (with the matchScale and matchRotate parameters).</li><li>■ 4 point—Performs cornerpin matchmoving on the foreground element. The pan, scale, and angle parameters disappear. Toggle to FG mode in the Viewer to adjust the 4 points that are pinned to the background. These points appear as <i>sourceNX/YPosition</i>.</li></ul>

Parameters	Type	Defaults	Notes
<i>applyX/Y</i>	int	1,1	These two toggles activate X and Y movement of the foreground element if <i>applyTransform</i> is active. These are saved in the script as <i>stabilizeX/Y</i> .
<i>applyScale</i>	int	1	In 2-point mode, toggles scaling of the foreground on and off. Under this parameter is a subparameter that returns the actual scaling and rotation value used in the transformation. You have the option to view the live calculated curve, or to bake the curve to create editable data (keyframes).
<i>scale</i>	float	NA	The calculated scale for 2-point matching. The scale at the <i>refFrame</i> is equal to 1, and all other frames are in reference to that frame.
<i>applyRotate</i>	int	0	In 2-point mode, this toggles rotation of the foreground on and off. You have the option to view the live calculated curve, or to bake the curve to create editable data (keyframes).
<i>rotate</i>	float	0	The calculated rotation for 2-point matching. The angle at the <i>refFrame</i> is equal to 0, and all other frames are in reference to that frame.
<i>sourceNX/YPosition</i>	float	0,0, Foreground.width, 0 Foreground.width, Foreground.height 0, Foreground.height	These 8 coordinates return the X/Y corner positions of the foreground element that are matched to the 4 tracking points when in 4-point mode. In the script, these coordinates are referred to as <i>x1</i> , <i>y1</i> , <i>x2</i> , <i>y2</i> , etc.
<i>x/yFilter</i>	string	default, xFilter	The transformation filter used. For more information, see Chapter 16, "Filters," on page 581.

Parameters	Type	Defaults	Notes
<i>motionBlur</i>	float	0	Enables the motion blur for the foreground element. A value of 0 is no blur; 1 is the high setting. A mid-value is a trade-off between speed and quality. This value is multiplied by the Global motionBlur parameter.
<i>shutterTiming</i>	float	.5	Shutter length. 0 is no blur, whereas 1 represents a whole frame of blur. Note that standard camera blur is 180 degrees, or a value of .5. This value is multiplied by the Global parameter shutterTiming.
<i>shutterOffset</i>	float	0	The offset from the current frame that the blur is calculated. Default is 0; previous frames are less than 0. The Global parameter shutterOffset is added to this.
<i>refFrame</i>	float	1	The reference frame that is used to calculate the null state of the transformation. For example, scale has a value of 1 and rotate has a value of 0 at the reference frame.
<i>outputType</i>	string	Background	The compositing operation for the <i>MatchMove</i> node. Each one follows the standard Shake operator of the same name. To pass on a tracked foreground without compositing, select Foreground. You can also use this when modifying the foreground corner points, as the FG/BG button on the Viewer toolbar switches this setting.
<i>clipMode</i>	int	1 (Background)	Selects the output resolution of the node from the Background (1) or the Foreground (0).
<i>aspectRatio</i>	float	1	The aspect ratio for non-square pixel plates.

Parameters	Type	Defaults	Notes										
<i>subPixelResolution</i>	string	1/16	<p>The resolution of your track. The smaller the number, the more precise and slower your tracking.</p> <p>Possible values:</p> <table><tr><td>1</td><td>Area is sampled at every pixel. Not very accurate or smooth, but very fast.</td></tr><tr><td>1/4</td><td>Area is sampled at every .25 pixels (16 times more than with a sampling of 1).</td></tr><tr><td>1/16</td><td>Area is sampled at every .0625 pixels (256 times more than with a sampling of 1).</td></tr><tr><td>1/32</td><td>Area is sampled at every .03125 pixels (1024 times more than with a sampling of 1).</td></tr><tr><td>1/64</td><td>Area is sampled at every .015625 pixels (4096 times more than with a sampling of 1).</td></tr></table>	1	Area is sampled at every pixel. Not very accurate or smooth, but very fast.	1/4	Area is sampled at every .25 pixels (16 times more than with a sampling of 1).	1/16	Area is sampled at every .0625 pixels (256 times more than with a sampling of 1).	1/32	Area is sampled at every .03125 pixels (1024 times more than with a sampling of 1).	1/64	Area is sampled at every .015625 pixels (4096 times more than with a sampling of 1).
1	Area is sampled at every pixel. Not very accurate or smooth, but very fast.												
1/4	Area is sampled at every .25 pixels (16 times more than with a sampling of 1).												
1/16	Area is sampled at every .0625 pixels (256 times more than with a sampling of 1).												
1/32	Area is sampled at every .03125 pixels (1024 times more than with a sampling of 1).												
1/64	Area is sampled at every .015625 pixels (4096 times more than with a sampling of 1).												

**Synopsis**

```
image MatchMove(  
    image Background,  
    image Foreground,  
    int applyTransform,  
    const char * stabType,
```

```

float track1X,
float track1Y,
int matchX,
int matchY,
float track2X,
float track2Y,
int scale,
int rotation,
float track3X,
float track3Y,
float track4X,
float track4Y,
float x1, // These values are
float y1, // for the foreground
float x2, // cornerpin coordinates
float y2,
float x3,
float y3,
float x4,
float y4,
const char * xFilter,
const char * yFilter,
float motionBlur,
float shutterTiming,
float shutterOffset,
float referenceFrame,
const char * compositeType,
int clipMode,
const char * trackRange,
const char * subPixelRes,
const char * matchSpace,
float referenceTolerance,
const char * referenceBehavior,
float failureTolerance,
const char * failureBehavior,
int limitProcessing,
...
);

```

### Script

```

image = MatchMove(
    Background,

```

```

    Foreground,
    applyTransform,
    "trackType",
    track1X,
    track1Y,
    matchX,
    matchY,
    track2X,
    track2Y,
    scale,
    rotation,
    track3X,
    track3Y,
    track4X,
    track4Y,
    x1,
    y1,
    x2,
    y2,
    x3,
    y3,
    x4,
    y4,
    "xFilter",
    "yFilter",
    motionBlur,
    shutterTiming,
    shutterOffset,
    referenceFrame,
    "compositeType",
    clipMode,
    "trackRange",
    "subPixelRes",
    "matchSpace",
    float referenceTolerance,
    "referenceBehavior",
    float failureTolerance,
    "failureBehavior",
    int limitProcessing,
    ...
);

```

### Stabilize

The *Stabilize* function is a dedicated tracking node that locks down an image, removing problems such as camera shake or gate weave. You can do 1-point (panning), 2-point (panning, scaling, or rotation), or 4-point (cornerpinning) stabilization. Tracks can be generated in the *Stabilize* node, or can be read in. To read in a track, right-click in the trackName text field and select Load Track.

The following table includes a description of the *Stabilize* parameters. The *Stabilize* parameters that are shared with the *MatchMove* function are described in the *MatchMove* function section above. Parameters that are shared with all tracking nodes appear in the “Tracking Parameters” section, above.

Parameters	Type	Default	Notes
<i>applyTransform</i>	int	0	The foreground element is only transformed if applyTransform is active.
<i>inverseTransform</i>	int	0	Inverts the transformation. Use this to "unstabilize" the shot. For example, you stabilize a shot with a <i>Stabilize</i> , apply compositing operations, and then copy the first <i>Stabilize</i> to the end of the node tree. By inverting the transformation, it returns the shot to its former unstable condition.
<i>transformationOrder</i>	string	trsx	The order that transformations are executed, in the default example (reading backward), shear, scale, rotate, translate.

### Synopsis

```
image Stabilize(  
    image In,  
    int applyTransform,  
    int inverseTransform  
    const char * trackType,  
    float track1X,  
    float track1Y,  
    int stabilizeX,  
    int stabilizeY,  
    float track2X,  
    float track2Y,  
    int matchScale,  
    int matchRotation,  
    float track3X,
```

```

float track3Y,
float track4X,
float track4Y,
const char * xFilter,
const char * yFilter,
const char * transformationOrder,
float motionBlur,
float shutterTiming,
float shutterOffset,
float referenceFrame,
float aspectRatio,
...
);

```

### Script

```

image = Stabilize(
    In,
    applyTransform,
    inverseTransform,
    "trackType",
    track1X,
    track1Y,
    stabilizeX,
    stabilizeY,
    track2X,
    track2Y,
    matchScale,
    matchRotation,
    track3X,
    track3Y,
    track4X,
    track4Y,
    "xFilter",
    "yFilter",
    "transformationOrder",
    motionBlur,
    shutterTiming,
    shutterOffset,
    referenceFrame,
    aspectRatio,
    "subPixelRes",
    "matchSpace",

```

```

float referenceTolerance,
"referenceBehavior",
float failureTolerance,
"failureBehavior",
int limitProcessing,
"trackRange",
...
);

```

## Tracker

The *Tracker* function is used only to generate tracks. Unlike the *MatchMove* and *Stabilize* functions, it does not alter the input image. The *Tracker* function is used to create tracks that are referenced either in other tracking nodes or in non-tracking nodes such as *Move2D*, *Rotate*, etc. While *MatchMove* and *Stabilize* are limited to creating one, two, or four track points, the *Tracker* node can hold as many trackers as you want. To add a tracker, click Add at the bottom of the Parameter View. If your workflow continually uses Smoothing and Averaging of tracks for application in a *Stabilize* or *MatchMove*, you should probably generate the tracks with a *Tracker* node.

With the *Tracker* node, you can delete trackers, and save the tracks to disk. To save an individual track, right-click in the trackName field and select Save Track.

For a table of the *Tracker* function's parameters, see the "Tracking Parameters" on page 406.

## Synopsis

```

image Tracker(
    image In,
    const char * trackRange,
    const char * subPixelRes,
    const char * matchSpace,
    float referenceTolerance,
    const char * referenceBehavior,
    float failureTolerance,
    const char * failureBehavior,
    int limitProcessing,
    float referencFrame
    ...
);

```

## Script

```

image = Tracker(
    In,
    "trackRange",
    "subPixelRes",

```

```

"matchSpace",
float referenceTolerance,
"referenceBehavior",
float failureTolerance,
"failureBehavior",
int limitProcessing,
float referencFrame
...
);

```

### CornerPin

The *CornerPin* function can be used to push the four corners of an image into four different positions, or to extract four positions and place them into the corners. For more information on *CornerPin*, see “*CornerPin*” on page 451.

# Transforms, Motion Blur, and Warping

## Chapter Summary

- About Transformations
- Concatenation of Transformations
- Inverting Transformations
- Onscreen Controls
- Setting Output Resolution and Scaling Images
- About Motion Blur
- Applying a Smear Effect
- The Transform Functions
- About Warps
- The Warp Functions

## About Transformations

Shake has a wide variety of transformations that also include resolution changes. You can pan, rotate, scale, and shear, as well as cornerpin (4-corner warping). These nodes are linear operations, so you can apply motion blur if the parameters are animated.

The *Move2D* node is the most flexible operator, since it includes most of the parameters included in the other transformations. There is no processing time difference between using a *Move2D* node or a *Pan* node to pan an image, as *Pan* is a macro of *Move2D*. However, using a single *Move2D* node for multiple transforms is more efficient than applying a *Pan* node, a *Rotate* node, and then a *Scale* node, due to internal computations.

Shake has an Infinite Workspace—effects are not clipped when they move in and out of frame with a second operation. For more information, see “About the Infinite Workspace” on page 150.

## Concatenation of Transformations

Similar to the way color corrections concatenate, many of the transformations concatenate together. Like color corrections, adjacent transformations are concatenated. You can apply a *Move2D* node, a *Rotate* node, and a *CornerPin* node, and the three nodes are resolved by executing the three moves simultaneously. Not only is the render faster, but it is of a much higher quality (since the image is filtered one time instead of three times). As an example, if you apply a *Rotate* node to an image of the earth so that the lit side faces the sun, and then use a second *Rotate* node to orbit the earth around the sun, Shake determines the combined transformation and calculates it in one pass. This is important because it means you spend, in this case, half of the processing time, and only filter the image once, not twice. (Just for fun on a rainy day, try panning an image 20 times in a different compositor to see the destructive effects of repetitive filtering.) Like the color corrections, this only occurs with adjacent transformations. See “About Motion Blur” on page 445 for an example of using concatenation to your advantage.

This concatenation only takes place when the transform nodes have no non-concatenating nodes between them. For example, you cannot apply a *Rotate* node, an *Over* node, a *Blur* node, and then a *Move2D* node and have the *Rotate* and the *Move2D* concatenate.

The following nodes concatenate:

- *Move2D*
- *Move3D*
- *Pan*
- *Rotate*
- *Scale*
- *Shear*
- *Stabilize*
- *CornerPin*
- *CameraShake*








## Inverting Transformations






The *Move2D* and *CornerPin* nodes have an `inverseTransform` parameter. The `inverseTransform` parameter inverts a transformation. For example, a pan of 100 with `inverseTransform` activated becomes a pan of -100. The parameters do not change, just the effects of the parameters. In the case of *Move2D*, you can use `inverseTransform` to turn imported tracking data into stabilization data. In the case of *CornerPin*, you declare the four corners to go to four arbitrary points (pin an image into a second image of a television screen), or the inverse (extract what is on the television screen into its own image). The second technique is also helpful for generating texture maps from photos for 3D renders.

## Onscreen Controls

Some functions, mainly transformations, have onscreen controls to help you interactively control your images in the Viewer. These controls appear whenever the node's parameters are loaded.

When you use an active function with onscreen controls, a second shelf of buttons appears in that Viewer, and disappears when you load a different node's parameters. The following table shows the common onscreen control buttons.

Button	Shortcut	Button Name and Function
	Click Autokey.	Autokey—Auto keyframing is on. A keyframe is automatically created each time an onscreen control is moved. To enable, you can also right-click and select On-Screen Ctrl Auto Key On.  To manually add a keyframe without moving the onscreen controls, click Autokey off and on.
		On-Screen controls—Show On-Screen Ctrl. Displays the onscreen controls. Click to toggle between Show On-Screen Ctrl and Hide On-Screen Ctrl.
	Click and hold the On-Screen controls button, or right-click.	On-Screen controls—Show On-Screen Ctrl On Release. The onscreen controls are displayed when modifying the image, and they return when you release the mouse.
	Click and hold the On-Screen controls button, or right-click.	On-Screen controls—Hide On-Screen Ctrl. Turns off the onscreen controls.
		Delete Keyframe—Deletes the keyframe at the current frame. This is used because controls for functions such as <i>Move2D</i> , keyframes for xPan, yPan, xScale, yScale, and angle are created simultaneously. Delete Key deletes the keyframes from all the associated parameters at the current frame.  To delete all keyframes for a parameter, such as <i>Move2D</i> on all frames, right-click the Delete Keyframe button and select Delete All Keys.
		Lock Direction—A control can be moved in both the X and Y directions.
	Click and hold Lock Direction.	Lock Direction—A control can be moved in only the X direction.

Button	Shortcut	Button Name and Function
	Click and hold Lock Direction.	Lock Direction—A control can be moved in only the Y direction.
		On-Screen Control Color—Click the onscreen control color button to change the color of the onscreen controls.
		Toggle Path Display—Displays the motion path and the keyframe positions in the Viewer. You can select and move the keyframes onscreen.
	Click and hold Toggle Path Display.	Displays only the keyframe positions in the Viewer.
	Click and hold Toggle Path Display.	The motion path and keyframes are not displayed in the Viewer.

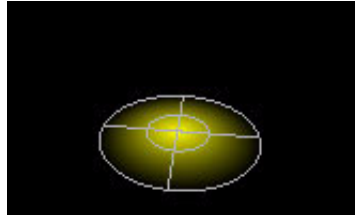
### Onscreen Controls and Additional Transformations

If you have two different transformations on a tree, all downstream controls are transformed along with the image. (This is mighty swank.)

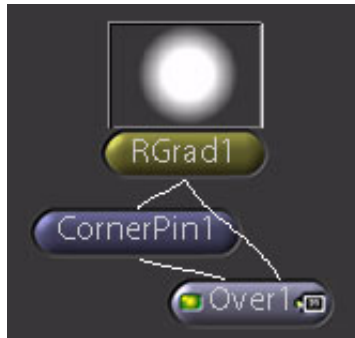
In the following example, an *RGrad* node is connected to a *CornerPin* node. The *CornerPin* node is used to place the *RGrad* in perspective.



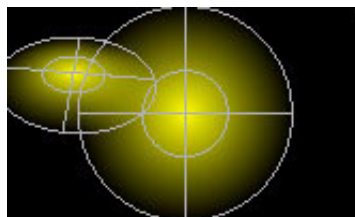
The *CornerPin* node is loaded in the Viewer, but the *RGrad* node parameters are loaded. The *RGrad* controls, instead of appearing perfectly round, also inherit the perspective shift of the *CornerPin* node. Therefore, you can always edit in context of your composite.



If you create a node tree where several versions of the same node are visualized, controls are displayed for each copy of the node. In this example, the *CornerPin* node is composited over the original *RGrad* node.



As shown in the following image, when tuning the *RGrad1* node while viewing the *Over1* node, multiple control sets are displayed. Moving one control modifies the other as well. To break this link, copy the original *RGrad* node and connect it to the new node.



**Note:** The *MatchMove* node is the only exception to this behavior. For nodes above the *MatchMove* node, the untransformed onscreen controls appear.

## The Controls

The most commonly used transform is the *Move2D* function. The *Move2D* function combines the controls of the *Rotate*, *Pan*, and *Scale* nodes. For special controls over the rotation center of *Move2D*, check out the *Rotate* controls.

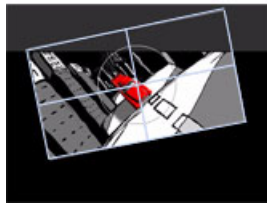
Since the onscreen controls are displayed outside of the frame, they provide a good way to find an image after it has been passed through an extreme transformation—zoom out of the Viewer until the transform box appears.

To quickly scrub through the animation, set Update mode (located in the upper right of the interface) to release **Update** **release** and move the time slider. The controls update, but the image does not until you release the mouse.

**Note:** To select release from the Update mode list, click and hold “manual” or “always” and select “release.”

### *Move2D*

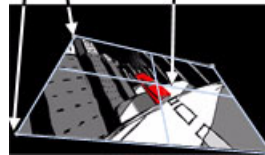
Move2D combines **Pan**, **Scale** and **Rotate**. You can still see the control after it passes beyond the frame.



### *CornerPin*

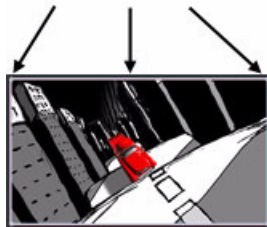
Grab a corner to move it

Grab the crossbar to pan the image.



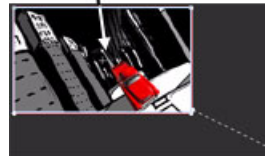
### *Crop*

Grab either the corners or the edges to adjust the crop.



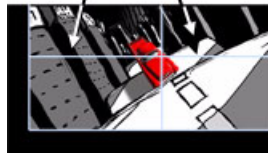
The image is resized after the crop.

You can also grab the crop center to move the entire window.

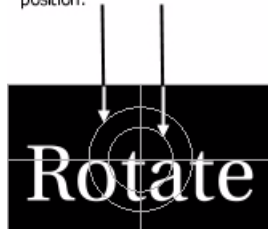


## *Pan*

Grab the crossbar to pan the image.



In this case, the outer ring is the angle, and the inner ring is used to change the center position.

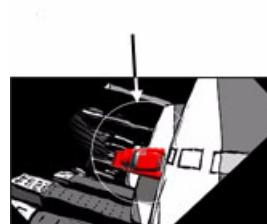


If you drag the center control, the image will not move; the pan values are changed to accommodate the rotation and the change of rotation center.



## *Rotate*

Grab the outer ring to rotate the shape around the rotation center.



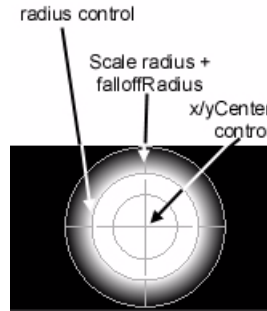
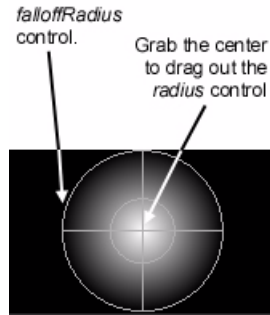
Here, we have rotated the image, but haven't moved the center.



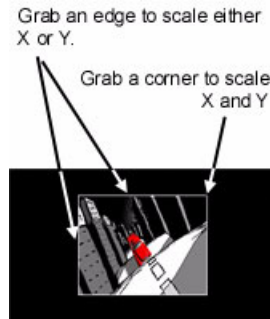
If you Ctrl+Drag the center, a rotated image will change position based on the new center location.



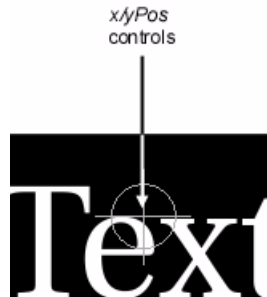
## RGrad



## Scale



## Text



## Setting Output Resolution and Scaling Images

There are two types of scaling functions:




- Functions that scale the size of the image in the frame, but do not change the actual resolution size (*Scale*, *Move2D*)
- Functions that scale the size of the image in the frame, and also change the resolution size (*Resize*, *Zoom*, *Fit*)








Additionally, the *Crop*, *Window*, and *ViewPort* nodes can be used to change the image resolution by cutting into an image or expanding its borders.



Finally, the *SetDOD* function crops in an area of interest, called the Domain of Definition (DOD).

The following tables discuss the differences between the scaling functions.

Shake Function	Changes Resolution	Changes Pixel Scale	Breaks Infinite Workspace	Changes Relative Aspect Ratio
<i>Scale, Move2D</i>		Yes		Yes
<i>Crop</i>	Yes		Yes	
<i>Window</i>	Yes		Yes	
<i>ViewPort</i>	Yes			
<i>SetDOD</i>			Yes	
<i>Resize</i>	Yes	Yes		Yes
<i>Fit</i>	Yes	Yes	Yes	
<i>Zoom</i>	Yes	Yes		Yes

Shake Function	Example	Example Parameters	Notes
Example Image		Resolution = 100 x 100 pixels	The unmodified image.
<i>Scale, Move2D</i>		xScale = .5 yScale = .5	<i>Scale</i> is a subset of the <i>Move2D</i> function. There is no processing speed increase when using <i>Scale</i> instead of <i>Move2D</i> .
<i>Scale, Move2D</i>		xScale = 1.6 yScale = 1.6	

Shake Function	Example	Example Parameters	Notes
<i>Crop</i>		-33 , -33, 133, 133	When using the default parameters, 0, 0, width, height, <i>Crop</i> breaks the Infinite Workspace, and resets the color beyond the frame to black.
<i>Crop</i>		33, 33, 67, 67	
<i>Window</i>		-33, -33, 166, 166	Identical to <i>Crop</i> , except you specify the output resolution in the third and fourth parameters.
<i>Window</i>		33, 33, 34, 34	
<i>ViewPort</i>		33, 33, 67, 67	Identical to <i>Crop</i> , except it does not cut off the Infinite Workspace, and is therefore primarily used to set a resolution.
<i>SetDOD</i>		33, 33, 67, 67	Used to limit the calculation area of the node tree to within the DOD; considerably speeds renders.
<i>Resize</i>		72, 48	

Shake Function	Example	Example Parameters	Notes
<i>Fit</i>		72, 48	<i>Fit</i> and <i>Resize</i> are identical, except that <i>Fit</i> preserves the pixel aspect ratio, padding the edges with black.
<i>Zoom</i>		.72, .48	<i>Zoom</i> and <i>Resize</i> are identical, except that <i>Zoom</i> gives a scaling factor and <i>Resize</i> gives a resolution for its parameters.

### About Motion Blur

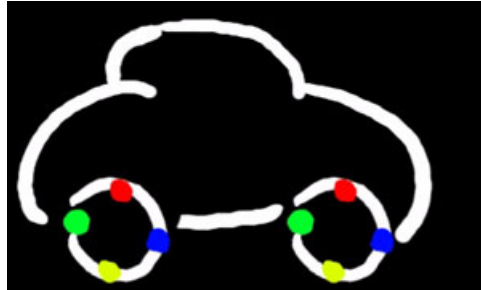
Motion blur can be applied to an animated transformation parameter. Each transform node has its own motion blur settings, so you can tune each one individually. There is also a global set of motion blur parameters that adjusts or replaces the existing values, depending on the parameter. You can set each node individually, and then set the Global motionBlur (in the Globals tab) value to 0 to turn them all off.

You can control the quality of the blur (0 is no blur, .5 is good, 1 is excellent, and higher is even better). To control the duration of the blur, use the shutterTiming parameter in the motionBlur subtree. By default, it goes from the current frame to halfway toward the next frame, a value of .5, or 180 degrees of camera shutter. Real-world film cameras are generally at 178 degrees. Use the shutterOffset parameter to control at what frame the shutter opening is considered to start. This value is 0 by default, so it starts at the current frame and calculates the motion that occurs up until the next frame. When set to -1, the motion from the previous frame up to the current frame is calculated.

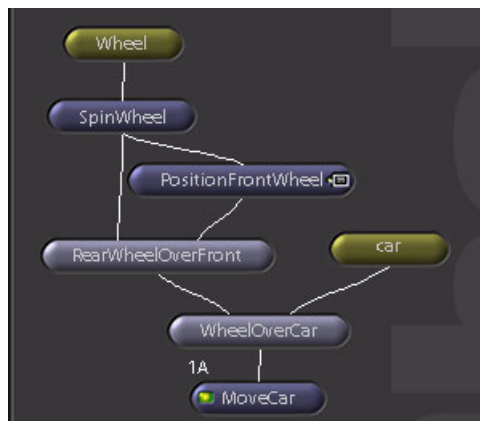
Concatenated transformations use the higher blur settings.

Sometimes, where the *Move2D* nodes are placed is a little counterintuitive due to the concatenation of transformations. Concatenations do not occur past any non-transformation node. For example, an *Over* node placed between two *Move2D* nodes breaks concatenation. To get around this, apply the *Move2D* nodes, and then composite.

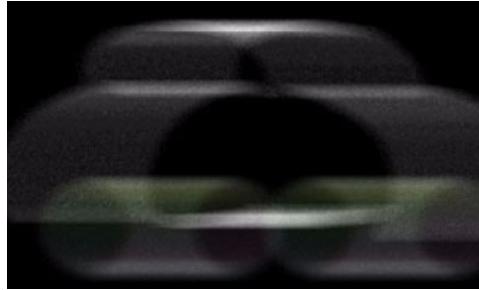
In this fine and photo-real example, two elements are composited together: A car body and a single wheel. The wheel is used twice, once for the back wheel and once for the front wheel.



In the node tree, a *Move2D* node (named *SpinWheel*) rotates the wheel, so the node is animated. The *PositionFrontWheel* node is a non-animated *Move2D* that pans a copy of *SpinWheel* to the front wheel position. The two wheel nodes are composited together (*RearWheelOverFront*), and the result is composited over the car body (*WheelsOverCar*). To animate the car forward, a *Move2D* node (*MoveCar1*) is applied, which simply pans the entire image right.



The result is inaccurate when the motion blur is applied because the *SpinWheel* node applies the blur for the turning wheel, and then three nodes later, the *MoveCar* node applies the blur to the already blurred wheels. Instead of the individual paths of the color dots on the wheels, the result is a horizontal smear.

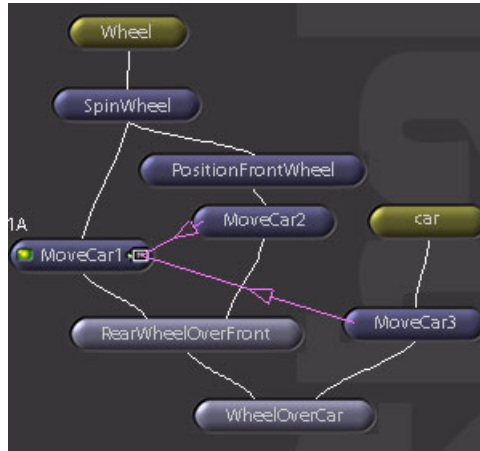


The following image shows the *WheelsOverCar* node immediately before the entire car is panned. The result is bad, since the wheels are blurred without consideration of the forward momentum added in a later node.

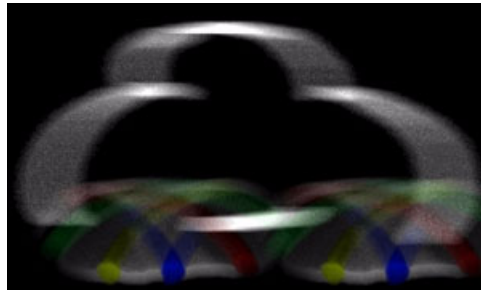


To correct this, it is more efficient to apply three *Move2D* nodes to pan the three elements separately, and then composite them together. In the following tree, the *MoveCar1*, *MoveCar2*, and *MoveCar3* nodes are identical, and *MoveCar2* and *MoveCar3* are linked to *MoveCar1*.

**Note:** To link a node, copy the node (**Command+C** / **Ctrl+C**) and clone the node (**Command+Shift+V** / **Ctrl+Shift+V**). To show the linked node, use **Ctrl+E**.



Because the *SpinWheel* and *MoveCar1* nodes are transformations, these nodes concatenate together. The *SpinWheel*, *PositionFrontWheel*, and *MoveCar2* nodes also concatenate together. The result is three transformations, the same amount as the previous tree, but with an accurate blur on the wheels.



## Applying a Smear Effect

To get motion blur on non-transformed elements, you can apply a smear effect using a *Move2D* node. In the *Move2D* parameters, use the *useReference* parameter (at the bottom of the parameters) and set the *referenceFrame* (in the *useReference* subtree) to *time*.

The following example uses a rendered clip of a swinging pendulum.

**To add blur to the pendulum with the *useReference* parameter:**

- 1 Locate the center of rotation for the pendulum, and enter the center values as the *Move2D* center.

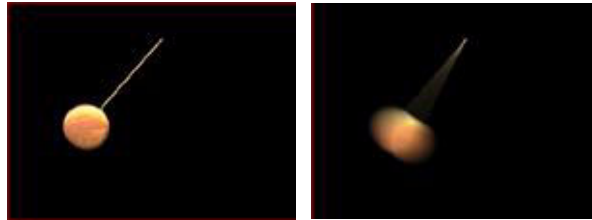
- 2 Approximate the rotation, and animate the angle to match the rotation. In this example, the angle is -40 at frame 5 and 40 at frame 24.
- 3 Set the referenceFrame (in the useReference subtree) to time.
- 4 Set motionBlur to 1.

When the animation is played back, the entire element swings out of frame due to the new animation.

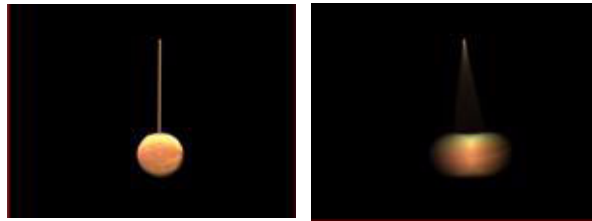
- 5 Enable useReference (set it to 1).

The element remains static, but the blur is still applied as if it were moving.

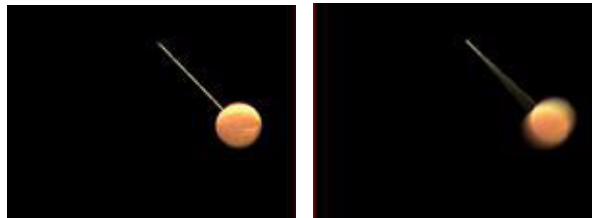
#### Frame 5



#### Frame 12



#### Frame 24



For a tutorial on this subject, see “Lesson Four: Using Local Variables With Expressions” in the *Shake 3 Tutorials*.

## The Transform Functions

The following section includes the transform functions located in the Transform Tool Tab.

For information on the transform functions used for tracking (*MatchMove*, *Stabilize*, and *Tracker*), see Chapter 11, “Tracking,” on page 401. For information on the transform functions that affect image resolution (*Fit*, *Resize*, and *Zoom*), see “Functions Affecting Image Resolution” on page 211. For information on cropping functions (*AddBorders*, *Crop*, *Viewport*, and *Window*), see “Cropping Functions” on page 233.

### CameraShake

The *CameraShake* function is a macro that applies noise functions to the pan values of *Move2D*. Since it is a good example of how to use noise, look at the actual macro parameters. Usually, a *Scale* is appended following a *CameraShake* node to make up for the black edges that appear. Because of the concatenation of transformations, this does not double-filter your image, so speed and quality are maintained.

Parameters	Type	Defaults	Function
<i>x, yFrequency</i>	float	1, 1	The x and y frequency of the shake. Higher numbers create more jitter.
<i>x, yAmplitude</i>	float	10, 10	The maximum amount of pixels moved in the camera shake.
<i>seed</i>	float	0	The random seed value. Change this number for different shaking rhythms.
<i>motionBlur</i>	float	0	Motion Blur quality level. 0 is no blur, whereas 1 represents standard filtering. For more speed, use less than 1. This value is multiplied by the Global Parameter <i>motionBlur</i> .
<i>shutterTiming</i>	float	.5	Shutter length. 0 is no blur, whereas 1 represents a whole frame of blur. Note that standard camera blur is 180 degrees, or a value of .5. This value is multiplied by the Global Parameter <i>shutterTiming</i> .
<i>shutterOffset</i>	float	0	This is the offset from the current frame at which the blur is calculated. Default is 0; previous frames are less than 0.

### Synopsis

```
image CameraShake( image,  
    float xFrequency,  
    float yFrequency,
```

```

float xAmplitude,
float yAmplitude,
float seed,
float motionBlur,
float shutterTiming,
float shutterOffset,
);

```

### Script

```

image = CameraShake(
    image,
    xFrequency, yFrequency,
    xAmplitude, yAmplitude,
    seed,
    motionBlur,
    shutterTiming,
    shutterOffset,
);

```

### Command Line

*shake -camerashake xFrequency yFrequency xAmplitude etc...*

### Examples

*shake lisa.iff -camera -t 1-30*

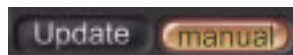
*shake lisa.iff -camera .5 2 30 30 -t 1-30*

*shake lisa.iff -camera -savescript example.shk*

### CornerPin

The *CornerPin* function can be used to push the four corners of an image into four different positions, or to extract four positions and place them into the corners. The first use is ideal for positioning an image in an onscreen television, for example. The second mode is handy for extracting texture maps, among other things. The four coordinate pairs start at the lower-left corner of the image (x0, y0), and work around in a counterclockwise direction to arrive at the upper-left corner of the image (x3, y3).

To perform an “unpin” (push four points on the image to the four corners of the image), switch the Update Mode to Manual, position the four points, enable inverseTransform, and click Update.



If the result is too blurred, lower the anti-aliasing value.

For information on four-point tracking, see Chapter 11, “Tracking,” on page 401.  
You can also use the *Move3D* function for perspective shifts.

Parameters	Type	Defaults	Function
<i>x0,y0, x1, y1, x2, y2, x3, y3</i>	float	0,0, width, 0, width,height, 0, height	Toggles between foreground (0) and background resolution (1).
<i>xyFilter</i>	string	"default"	The filtering type.
<i>inverseTransform</i>	int	0	Inverts the transform. In this case, it puts the four corners into the four coordinates (0, or pinning), or pulls the four coordinates to the corners (1, or unpinning).
<i>antialiasing</i>	float	1	Individual antialiasing. A value of 0 brings out more clarity.
<i>motionBlur</i>	float	0	Motion Blur quality level. 0 is no blur, whereas 1 represents standard filtering. For more speed, use less than 1. This value is multiplied by the Global Parameter motionBlur.
<i>shutterTiming</i>	float	.5	Shutter length. 0 is no blur, whereas 1 represents a whole frame of blur. Note that standard camera blur is 180 degrees, or a value of .5. This value is multiplied by the Global Parameter shutterTiming.
<i>shutterOffset</i>	float	0	This is the offset from the current frame that the blur is calculated. Default is 0; previous frames are less than 0.
<i>useReference</i>	int	0	This is to be used to apply motion blur to previously-animated elements. See "Lesson Four: Using Local Variables With Expressions" in the <i>Shake 3 Tutorials</i> .

**Synopsis**

```
image CornerPin(  
    image,  
    float x0, float y0,  
    float x1, float y1,  
    float x2, float y2,
```

```

float x3, float y3,
const char * xFilter,
const char * yFilter,
int inverseTransform,
float antialiasing,
float motionBlur,
float shutterTiming,
float shutterOffset,
int useReference
);

```

### Script

```

image = CornerPin(
    image,
    x0, y0, x1, y1,
    x2, y2, x3, y3,
    "xFilter", "yFilter",
    inverseTransform,
    antialiasing,
    motionBlur,
    shutterTiming,
    shutterOffset,
    useReference
);

```

### Command Line

*shake -cornerpin x0 y0 x1 y1 etc...*

### Examples

*shake lisa.iff -corner 228 283 275 284 341 426 154 422*

*shake lisa.iff -corner 228 283 275 284 341 426 154 422 default default 1*

### See Also

“*PinCushion*” on page 473, “*Move2D*” on page 454, “*Shear*” on page 467, “*Move3D*” on page 457

### Crop

For information on the *Crop* function, see “*Crop*” on page 233.

### Fit

For information on the *Fit* function, see “*Fit*” on page 211.

## Flip

The *Flip* function flips an image upside down. You can also use *Scale* or *Move2D* to invert the image by setting the *yScale* value to -1.

**Note:** *Flip* (or a *yScale* of -1 ) forces the buffering of the image into memory.

### Synopsis

```
image Flip( image );
```

### Script

```
image = Flip ( image );
```

### Command Line

*shake -flip*

## Flop

The *Flop* function flops the image left and right. Unlike the *Flip* function, this does not buffer the image into memory. You can also use *Scale* or *Move2D* to invert the image by setting the *yScale* value to -1.

### Synopsis

```
image Flop( image );
```

### Script

```
image = Flop ( image );
```

### Command Line

*shake -flop*

## MatchMove

For information on the *MatchMove* function, see “*MatchMove*” on page 425.

## Move2D

The *Move2D* function combines many of the other transform nodes, including *Pan*, *Scale*, *Shear*, and *Rotate*. The *xCenter* and *yCenter* parameters apply to both scaling and rotation centers. If you need different centers, you can append a second function (*Move2D*, *Rotate*, or *Scale*) and switch scaling or rotation to that node. This does not cost you any process time, as Shake concatenates neighboring transforms into one big transform. Therefore, you lose neither quality nor calculation time.

Also, Shake's Infinite Workspace comes into play when you have two transforms together. (For example, one transform rotates the image, and the second transform rotates the image back, without clipping the corners.) However, in terms of workflow, when you pan around small elements that are later composited on larger resolution backgrounds, you do not have to crop your small elements out to create a larger space.

Parameters	Type	Defaults	Function
<i>x, yPan</i>	float	0, 0	X and Y Pan values.
<i>angle</i>	float	0	The rotation angle.
<i>aspectRatio</i>	float	1	The aspectRatio of the image. Useful when you work with a Global aspectRatio and anamorphic frames and you do a rotation.
<i>x, yScale</i>	float	1, xScale	Scaling on the X and Y axis.
<i>x, yShear</i>	float	0, 0	Shearing on the X and Y axis.
<i>x, yFilter</i>	string	"default", "default"	For more information, see Chapter 16, "Filters."
<i>transformOrder</i>	string	"trsx"	The order the transform is executed, with: t = translate r = rotate s = scale x = shear By default, this is set to "trsx".
<i>inverseTransform</i>	int	0	Inverts the transform. This can quickly convert tracking data to stabilization data.
<i>motionBlur</i>	float	0	Motion Blur quality level. 0 is no blur, whereas 1 represents standard filtering. For more speed, use less than 1. This value is multiplied by the Global Parameter motionBlur.
<i>shutterTiming</i>	float	.5	Shutter length. 0 is no blur, whereas 1 represents a whole frame of blur. Note that standard camera blur is 180 degrees, or a value of .5. This value is multiplied by the Global Parameter shutterTiming.
<i>shutterOffset</i>	float	0	This is the offset from the current frame at which the blur is calculated. Default is 0; previous frames are less than 0.

Parameters	Type	Defaults	Function
<i>useReference</i>	int	0	<p>Applies the transform to the image or doesn't. If it doesn't, and you have animated values, it applies a motion blur to the image, but does not actually move it. This is good for adding blur to plates. See below for an example.</p> <p>0 = Move image. 1 = Smear-mode; image is not moved.</p>
<i>referenceFrame</i>	int	time	<p>Used for stabilization. Usually, this is set to "time," that is, in reference to itself. However, if <i>inverseTransform</i> is set to 1, you are stabilizing, with the assumption that any animation you have applied matches up to the source animation. Setting the reference frame locks the movement in to a specific frame.</p>

### Synopsis

```
image Move2D(
    image,
    float xPan,
    float yPan,
    float angle,
    float aspectRatio,
    float xScale,
    float yScale,
    float xShear,
    float yShear,
    float xCenter,
    float yCenter,
    const char * xFilter,
    const char * yFilter,
    const char * transformationOrder,
    int inverseTransform,
    float motionBlur,
    float shutterTiming,
    float shutterOffset,
    int useReference,
    float referenceFrame
);
```

Script

```
image = Move2D(
    image,
    xPan, yPan
    angle, aspectRatio,
    xScale, yScale,
    xShear, yShear,
    xCenter, yCenter,
    "xFilter", "yFilter",
    "transformOrder",
    inverseTransform,
    motionBlur,
    shutterTiming,
    shutterOffset,
    useReference,
    referenceFrame
);
```

Command Line

*shake -move2d xPan yPan angle aspectRatio etc...*

**Note:** For an example of using the useReference parameter, see “Applying a Smear Effect” on page 448.

Move3D

The *Move3D* function allows you to create perspective changes by rotating and visually moving the image in depth. It is similar in behavior to the *Move2D* function.

Some *Move3D* parameters, such as zPan, have no effect unless the fieldOfView parameter is set to a value greater than 0. The angle of the Z axis is controlled, as in *Move2D*, by the angle parameter. There is no shearing, but you can rotate the image in X or Y, and keep the fieldOfView set to 0 to get orthogonal shearing effects.

Parameters	Type	Defaults	Function
<i>x, y, zPan</i>	float	0, 0, 0	X, Y, and Z Pan values. zPan has no effect unless the fieldOfView value is set to greater than 0.
<i>x, y, zAngle</i>	float	0, 0, 0	The rotation angles around the X, Y, and Z axis.
<i>aspectRatio</i>	float	1	The aspectRatio of the image. Useful when you work with a Global aspectRatio and anamorphic frames and you do a rotation.
<i>x, y, zScale</i>	float	1, 1, 1	Scaling on the X, Y, and Z axis.

Parameters	Type	Defaults	Function
<i>x, yFilter</i>	string	"default", "default"	For more information, see Chapter 16, "Filters."
<i>transformOrder</i>	string	"trsx"	<p>The order the transform is executed, with</p> <p>t = translate</p> <p>r = rotate</p> <p>s = scale</p> <p>By default, this is set to "trs."</p> <p>If you are trying to transfer camera setups from 3D applications, note that this is the order transforms are pushed onto the matrix and that rotations are applied in the order X, Z, Y.</p>
<i>inverseTransform</i>	int	0	Inverts the transform. This can quickly convert tracking data to stabilization data.
<i>fieldOfView</i>	float	0	Designates the vertical field of view and affects the appearance of perspective shift. When this is 0, the view is considered to be orthogonal, with increasing perspective changes when you combine zPan, angleX, and angleY with a higher fieldOfView.
<i>motionBlur</i>	float	0	Motion Blur quality level. 0 is no blur, whereas 1 represents standard filtering. For more speed, use less than 1. This value is multiplied by the Global Parameter motionBlur.
<i>shutterTiming</i>	float	.5	Shutter length. 0 is no blur, whereas 1 represents a whole frame of blur. Note that standard camera blur is 180 degrees, or a value of .5. This value is multiplied by the Global Parameter shutterTiming.
<i>shutterOffset</i>	float	0	This is the offset from the current frame at which the blur is calculated. Default is 0; previous frames are less than 0.

Parameters	Type	Defaults	Function
<i>useReference</i>	int	0	<p>Applies the transform to the image or doesn't. If it doesn't, and you have animated values, it applies a motion blur to the image, but does not actually move it. This is good for adding blur to plates. For an example, see "Lesson Four: Using Local Variables With Expressions" in the <i>Shake 3 Tutorials</i>.</p> <p>0 = Move image. 1 = Smear-mode; image is not moved.</p>
<i>referenceFrame</i>	int	time	<p>Used for stabilization. Usually, this is set to "time," that is, in reference to itself. However, if <i>inverseTransform</i> is set to 1, you are stabilizing, with the assumption that any animation you have applied matches up to the source animation. Setting the reference frame locks the movement in to a specific frame.</p>

### Synopsis

```
image Move3D(
    image,
    float xPan,
    float yPan,
    float zPan,
    float angle,
    float angleX,
    float angleY,
    float aspectRatio,
    float xScale,
    float yScale,
    float zScale,
    float xCenter,
    float yCenter,
    float zCenter,
    const char * xFilter,
    const char * yFilter,
    const char * transformationOrder,
    int inverseTransform,
    float fieldOfView,
    float motionBlur,
    float shutterTiming,
```

```
float shutterOffset,  
int useReference,  
float referenceFrame  
);
```

### **Script**

```
image = Move3D(  
    image,  
    xPan, yPan, zPan  
    angle, angleX, angleY, aspectRatio,  
    xScale, yScale, zScale  
    xCenter, yCenter, zCenter,  
    "xFilter", "yFilter",  
    "transformOrder",  
    inverseTransform,  
    fieldOfView,  
    motionBlur,  
    shutterTiming,  
    shutterOffset,  
    useReference,  
    referenceFrame  
);
```

### **Command Line**

*shake -move3d xPan yPan zPan angle aspectRatio etc...*

### Orient

The *Orient* function rotates the image by 90-degree increments, resizing the image frame if necessary. (For example, for 1 turn on a 300 x 600 frame, it rotates it 90 degrees, and makes a 600 x 300 frame.) You can also apply a *Flip* node and *Flop* node to the image. To rotate in degrees, use a *Rotate* node or a *Move2D* node.

Parameters	Type	Defaults	Function
<i>nTurn</i>	int	0	Number of 90-degree increments the image is rotated, for example: 0 = no rotation. 1 = 90 degrees turn counterclockwise. 2 = 180 degrees turn counterclockwise. 3 = 270 degrees turn counterclockwise. 4 = no rotation.
<i>flop</i>	int	0	Flops the image left and right around the Y axis. 0 = no flop 1 = flop
<i>flip</i>	int	0	Flips the image up and down around the X axis. 0 = no flip 1 = flip

### Synopsis

```
image Orient( image,  
             int nTurn,  
             int flop,  
             int flip  
            );
```

### Script

```
image = Orient( image, nTurn, flop, flip );
```

### Command Line

```
shake -orient nTurn flop flip
```

### Pan

The *Pan* function pans the image with subpixel precision. To wrap an image around the frame (for example, anything that moves off the right edge of the frame reappears on the left), use the *Scroll* node.

Parameters	Type	Defaults	Function
<i>x, yPan</i>	float	0, 0	The X and Y pan values.
<i>motionBlur</i>	float	0	Motion Blur quality level. 0 is no blur, whereas 1 represents standard filtering. For more speed, use less than 1. This value is multiplied by the Global Parameter motionBlur.
<i>shutterTiming</i>	float	.5	Shutter length. 0 is no blur, whereas 1 represents a whole frame of blur. Note that standard camera blur is 180 degrees, or a value of .5. This value is multiplied by the Global Parameter shutterTiming.
<i>shutterOffset</i>	float	0	This is the offset from the current frame at which the blur is calculated. Default is 0; previous frames are less than 0.

### Synopsis

```
image Pan( image,  
    float xPan,  
    float yPan,  
    float motionBlur,  
    float shutterTiming,  
    float shutterOffset,  
);
```

### Script

```
image = Pan(  
    image,  
    xPan, yPan  
    motionBlur,  
    shutterTiming,  
    shutterOffset,  
);
```

### Command Line

*sbake -pan xPan yPan motionBlur etc...*

## Resize

For information on the *Resize* function, see “*Resize*” on page 212.

## Rotate

The *Rotate* function rotates the image with subpixel precision.

Parameters	Type	Defaults	Function
<i>angle</i>	float	0	The rotation angle. Positive values are counterclockwise rotation.
<i>aspectRatio</i>	float	1	The aspectRatio, for use with anamorphic frames.
<i>x, yCenter</i>	float	width/2, height/2	The X and Y rotate center values.
<i>motionBlur</i>	float	0	Motion Blur quality level. 0 is no blur, whereas 1 represents standard filtering. For more speed, use less than 1. This value is multiplied by the Global Parameter motionBlur.
<i>shutterTiming</i>	float	.5	Shutter length. 0 is no blur, whereas 1 represents a whole frame of blur. Note that standard camera blur is 180 degrees, or a value of .5. This value is multiplied by the Global Parameter shutterTiming.
<i>shutterOffset</i>	float	0	This is the offset from the current frame at which the blur is calculated. Default is 0; previous frames are less than 0.

### Synopsis

```
image Rotate( image,  
             float angle,  
             float aspectRatio,  
             float xCenter,  
             float yCenter,  
             float motionBlur,  
             float shutterTiming,  
             float shutterOffset,  
             );
```

### Script

```
image = Rotate(  
    image,  
    angle,
```

```

    aspectRatio,
    xCenter, yCenter
    motionBlur,
    shutterTiming,
    shutterOffset,
);

```

### Command Line

*sbake -rotate angle aspectRatio xCenter etc...*

### Scale

The *Scale* function scales the image with subpixel precision. To rotate and scale an image, you probably want to use the two independent nodes, *Rotate* and *Scale*, rather than a *Move2D* node, because you can control the centers independently. Unlike the *Resize* node, *Scale* does not change the image resolution.

If you scale by a negative number on Y, you buffer the image. You can also use the *Flip* and *Flop* nodes to invert your image.

Parameters	Type	Defaults	Function
<i>x, yScale</i>	float	1, 1	The scaling factor.
<i>x, yCenter</i>	float	width/2, height/2	The X and Y scale center values.
<i>motionBlur</i>	float	0	Motion Blur quality level. 0 is no blur, whereas 1 represents standard filtering. For more speed, use less than 1. This value is multiplied by the Global Parameter motionBlur.
<i>shutterTiming</i>	float	.5	Shutter length. 0 is no blur, whereas 1 represents a whole frame of blur. Note that standard camera blur is 180 degrees, or a value of .5. This value is multiplied by the Global Parameter shutterTiming.
<i>shutterOffset</i>	float	0	This is the offset from the current frame at which the blur is calculated. Default is 0; previous frames are less than 0.

### Synopsis

```

image Scale( image,
    float xScale,
    float yScale,
    float xCenter,

```

```

float yCenter,
float motionBlur,
float shutterTiming,
float shutterOffset,
);

```

### Script

```

image = Scale( image,
    xScale, yScale,
    xCenter, yCenter
    motionBlur,
    shutterTiming,
    shutterOffset,
);

```

### Command Line

*shake -scale xScale yScale xCenter etc...*

### Scroll

The *Scroll* function is similar to the *Pan* function, except the image wraps around the frame, and reappears on the opposite side. Because there is no way to track a pixel, there is no motion blur available in the *Scroll* node.

Parameters	Type	Defaults	Function
<i>x, yScroll</i>	float	0, 0	The X and Y panning values.

### Synopsis

```

image Scroll( image,
    float xScroll,
    float yScroll,
);

```

### Script

```

image = Scroll( image, xScroll, yScroll);

```

### Command Line

*shake -scroll xScroll yScroll*

### SetDOD

The *SetDOD* function limits the active area, or Domain of Definition (DOD), to a limited window. If you have a large element with only a small portion of non-black pixels, you can apply a *SetDOD* node to isolate the area and to speed all image processes.

Shake automatically assigns a DOD to an .iff image. For example, if you scale down an image, Shake limits the DOD to the scaled area. This remains in effect for later composites, but does not affect the current script.

You can procedurally access the DOD of an image with the following variables:

- dod[0]* Left edge of DOD
- dod[1]* Bottom edge of DOD
- dod[2]* Right edge of DOD
- dod[3]* Top edge of DOD

For example, calling *FileIn1.dod[0]* returns the minimum X value of the *FileIn1* node's DOD. For more information, see "The Domain of Definition (DOD)" on page 50.

Parameters	Type	Defaults	Function
<i>left</i>	int	0	The new left edge of the DOD.
<i>bottom</i>	int	0	The new bottom edge of the DOD.
<i>right</i>	int	width	The new right edge of the DOD.
<i>top</i>	int	height	The new top edge of the DOD.

**Synopsis**

```
image SetDOD( image,  
    int left,  
    int right,  
    int bottom,  
    int top  
);
```

**Script**

```
image = SetDOD( image,  
    left, right,  
    bottom, top  
);
```

**Command Line**

```
shake -setdod left right etc.
```

### Shear

The *Shear* function skews the image left and right, or up and down. Motion blur can also be applied.

Parameters	Type	Defaults	Function
<i>x, yShear</i>	float	0, 0	Shearing on the X and Y axis.
<i>x, yCenter</i>	float	width/2, height/2	The center of the shear.
<i>motionBlur</i>	float	0	Motion Blur quality level. 0 is no blur, whereas 1 represents standard filtering. For more speed, use less than 1. This value is multiplied by the Global Parameter <i>motionBlur</i> .
<i>shutterTiming</i>	float	.5	Shutter length. 0 is no blur, whereas 1 represents a whole frame of blur. Note that standard camera blur is 180 degrees, or a value of .5. This value is multiplied by the Global Parameter <i>shutterTiming</i> .
<i>shutterOffset</i>	float	0	This is the offset from the current frame at which the blur is calculated. Default is 0; previous frames are less than 0.

### Synopsis

```
image Shear(  
    image,  
    float xShear,  
    float yShear,  
    float xCenter,  
    float yCenter,  
    float motionBlur,  
    float shutterTiming,  
    float shutterOffset,  
);
```

### Script

```
image = Shear(  
    image,  
    xShear, yShear,  
    xCenter, yCenter,  
    motionBlur,  
    shutterTiming,
```

```
shutterOffset,  
);
```

### Command Line

*shake -shear xShear yShear xCenter yCenter etc...*

### Stabilize, Tracker

For information on the *Stabilize* and *Tracker* functions, see “Tracking Functions” on page 424.

### Viewport, Window

For information on the *Viewport* and *Window* functions, see “Cropping Functions” on page 233.

### Zoom

For information on the *Zoom* function, see “Zoom” on page 213.

## About Warps

Transformations work on entire images so that each pixel in the image is moved, rotated, or scaled the same amount. These are all called “Linear Transformations.” Even the *CornerPin* function applies the same transformation to every pixel—they are all affected by the manipulation of the four corners. Warps differ from transformations in that each pixel is considered an individual case, with the ability to be moved completely independently of its neighbor. For this reason, warps are sometimes called “Nonlinear Transformations.” Warps are good for making rippling patterns, randomizing images, and other effects. You can also input an arbitrary expression with the *WarpX* function, which is similar to *ColorX*, except it performs warping instead of color changes.

## The Warp Functions

The following section discusses the warping functions, located in the Warp Tool Tab.

### DisplaceX

The *DisplaceX* function is a general purpose warping tool that is similar to *WarpX*, except you can access a second warping image to control the distribution of a warp. Any formula can be entered in the x and y fields for custom warps. You can also create multiline expressions in this function.

Parameters	Type	Defaults	Function
<i>oversampling</i>	int	1	The actual number of samples per pixel equals this number squared. For better antialiasing, increase the number.
<i>x, yExpression</i>	float	x, y	<p>The expression for where the pixel information is pulled. Expressions of x and y return the same image. Expressions of x + 5, y + 5 pull the color from the pixel 5 units up and to the right of the current pixel. You can access the values of the warping image (the second one) with r, g, b, a, and z.</p> <p>Example expressions:</p> <pre>"x*r" "x + ((r-.5)*30*r)" "x + cos(y/140)*70*g" "x + r*r*cos(x*y/100)*100"</pre>
<i>x, yDelta</i>	float	0, xDelta	<p>Sets the maximum distance that any pixel is expected to move, but doesn't actually move it. From any given pixel, it may be affected by any pixel within the Delta distance. This means it must consider a much greater amount of pixels that may possibly affect the currently rendered pixel. This is bad. However, if you set a Delta value to too small an amount, you get errors if your expression tells the pixel to move beyond that limit. Therefore, it always takes some testing to balance between speed with errors, or accuracy with drastically slower renders. Our advice: Start small and increase the size until the errors disappear.</p>

### Synopsis

```
image DisplaceX(
    image,
    image,
    int oversampling,
    float expression xExpression,
    float expression yExpression,
    int xDelta,
    int yDelta
);
```

### Script

```
image = DisplaceX(  
    image,  
    image,  
    oversampling,  
    xExpression,  
    yExpression,  
    xDelta,  
    yDelta  
);
```

For multiline expressions:

```
image = DisplaceX(  
    image,  
    image,  
    oversampling,  
    {{ xExpr1, xExpr2, ... xExprN }},  
    yExpression,  
    xDelta,  
    yDelta  
);
```

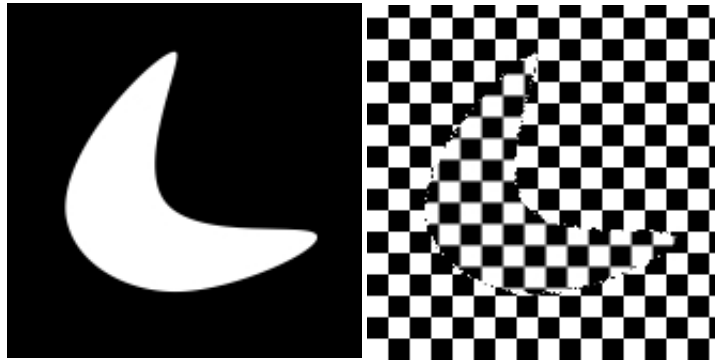
### Command Line

*shake -displacex oversampling xExpression etc...*

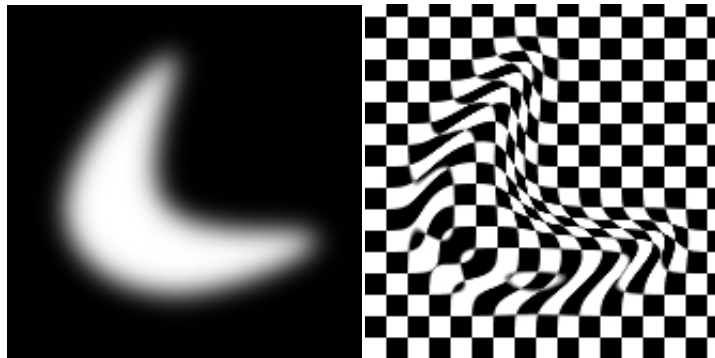
### IDisplace

The *IDisplace* function is a hard-wired version of *DisplaceX* to warp an image based on the intensity of a second image. The formula it uses is " $x-(a*xScale)$ " and " $y-(a*yScale)$ ".

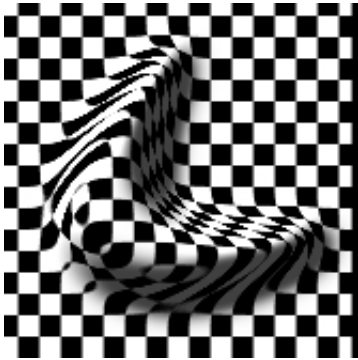
The following image is a checkerboard warped with a *QuickShape* node. Because the shape is black and white, with little grey, it is difficult to make out the distortion in the checkerboard.



As the following image demonstrates, it is often a good idea to insert a blur between a high contrast distortion image and the *IDisplace* node.



This looks really snazzy when combined with the Relief macro in the “Cookbook” section of the *Shake 3 Tutorials*.



Parameters	Type	Defaults	Function
<i>x/yScale</i>	float	0, xScale	The amount of pixels that the image is offset by the second image.
<i>x, yDOffset</i>	float	0, xDOffset	A panning factor applied to the image. Intensity is usually 0 to 1; 1 is 100 percent of the x/yScale factor.
<i>x, yChannel</i>	string	"a", "a"	The channel from the second image that is used to distort the first image.
<i>x, yDelta</i>	float	0, xDelta	The anticipated amount that the pixels will move. If this is too much, calculations slow down. If it is too little, there are black holes in the image.

**Synopsis**

```
image IDisplace(  
    image img,  
    image controlImg,  
    float xScale,  
    float yScale,  
    float xDOffset,  
    float yDOffset,  
    const char * xChannel,  
    const char * yChannel,  
    float xDelta,  
    float yDelta  
);
```

**Script**

```
image = IDisplace(  
    img,  
    controlImg,  
    xScale, yScale,  
    xMaskOffset, yMaskOffset,  
    "xChannel", "yChannel",  
    xDelta, yDelta  
);
```

**Command Line**

```
sbake -idisplace controlImg xScale ...
```

**PinCushion**

The *PinCushion* function is useful to do a lens correction for a proper match move. It distorts the corners of the image in and out to mimic lens distortion. You can push the values below 0 as well.

Parameters	Type	Defaults	Function
<i>oversampling</i>	int	1	The actual number of samples per pixel equals this number squared. For better antialiasing, increase the number.
<i>x, yFactor</i>	float	0, 0	The amount of distortion.

**Synopsis**

```
image PinCushion( image,  
    int oversampling,  
    float xFactor,  
    float yFactor  
);
```

**Script**

```
image = PinCushion(  
    image,  
    oversampling,  
    xFactor,  
    yFactor  
);
```

**Command Line**

```
sbake -pincushion oversampling xfactor yfactor
```

### Randomize

The *Randomize* function randomizes the position of each pixel within a certain distance. To randomize the color, create a *Rand* element, and *IMult* it by your image.

Parameters	Type	Defaults	Function
<i>oversampling</i>	int	1	The actual number of samples per pixel equals this number squared. For better antialiasing, increase the number.
<i>seed</i>	float	time	The random seed.
<i>x, yAmplitude</i>	float	0, 0	The amount of randomization in pixel distance.
<i>x, yOffset</i>	float	0, 0	An offset to the random pattern.

### Synopsis

```
image Randomize( image,  
    int oversampling,  
    float seed,  
    float xAmplitude,  
    float yAmplitude,  
    float xOffset,  
    float yOffset  
);
```

### Script

```
image = Randomize(  
    oversampling,  
    seed,  
    xAmplitude,  
    yAmplitude,  
    xOffset,  
    yOffset  
);
```

### Command Line

*shake -randomize oversampling seed xAmplitude etc...*

## Turbulate

The *Turbulate* function is similar to the *Randomize* function, except that it passes a continuous field of noise over the image, rather than just randomly stirring the pixels around. This is an expensive function.

Parameters	Type	Defaults	Function
<i>oversampling</i>	int	1	The actual number of samples per pixel equals this number squared. For better antialiasing, increase the number.
<i>detail</i>	int	1	The amount of fractal detail. The higher the number, the more iterations of fractal noise. This can be very expensive.
<i>x, yNoiseScale</i>	float	10, xNoiseScale	The scale of the waves.
<i>x, yAmplitude</i>	float	0, xAmplitude	The amount of randomization in pixel distance.
<i>x, yOffset</i>	float	0, xOffset	An offset to the random pattern.
<i>seed</i>	float	0	The random seed.

### Synopsis

```
image Turbulate( image,  
    int oversampling,  
    int detail,  
    float xNoiseScale,  
    float yNoiseScale,  
    float xAmplitude,  
    float yAmplitude,  
    float xOffset,  
    float yOffset,  
    float seed,  
);
```

### Script

```
image = Turbulate(  
    oversampling,  
    detail,  
    xNoiseScale,  
    yNoiseScale,  
    xAmplitude,  
    yAmplitude,
```

```

    xOffset,
    yOffset,
    seed
);

```

### Command Line

*shake -turbulate oversampling detail xNoiseScale etc...*

### Twirl

The *Twirl* function creates a whirlpool-like effect. It's fun.

Parameters	Type	Defaults	Function
<i>startAngle</i>	float	0	The amount of twirl near the Center of the rotation.
<i>endAngle</i>	float	0	The amount of twirl away from the Center.
<i>aspectRatio</i>	float	1	The aspectRatio of the image. Useful when you work with a Global aspectRatio and anamorphic frames, and you are doing a rotation.
<i>x, yCenter</i>	float	width/2, height/2	Center of the twirl.
<i>x, yRadius</i>	float	width/2, width/2	Radius of the twirl circle.
<i>bias</i>	float	1	Controls how much of the twist occurs between the center and the Radius. 0 means the outer Radius is not rotated; 1 means the center is not rotated.
<i>antialiasing</i>	float	1	Anti-aliasing on the effect.

### Synopsis

```

image Twirl(
    image,
    float startAngle,
    float endAngle,
    float aspectRatio,
    float xCenter,
    float yCenter,
    float xRadius,
    float yRadius,
    float bias,

```

```
float antialiasing  
);
```

### Script

```
image = Twirl(  
    image,  
    startAngle,  
    endAngle,  
    aspectRatio,  
    xCenter,  
    yCenter,  
    xRadius,  
    yRadius,  
    bias,  
    antialiasing  
);
```

### Command Line

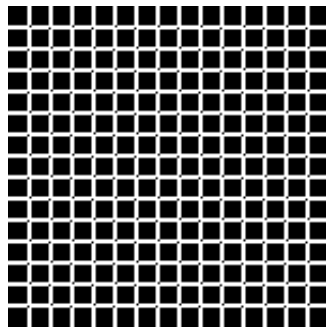
*shake -twirl startAngle endAngle aspectRatio etc...*

### WarpX

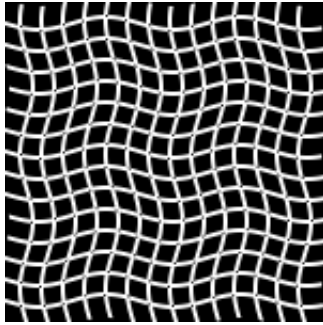
The *WarpX* function is a general purpose warping tool, similar to *ColorX*, except instead of changing a pixel's color, you change its position. Any formula can be entered in the x and y fields for custom warps. You can also create multiline expressions in this function.

One note of caution: *WarpX* does not properly set the DOD, so you may need to manually attach a Transform-*SetDOD* node following the *WarpX* node.

The following examples are on a grid. By modifying x and y, you specify from what pixel the information is pulled. For example,  $x+5$ ,  $y+5$  shifts the image left and down by 5 pixels.



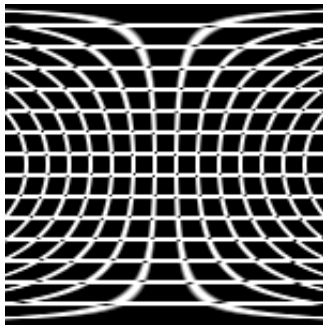
x  
y



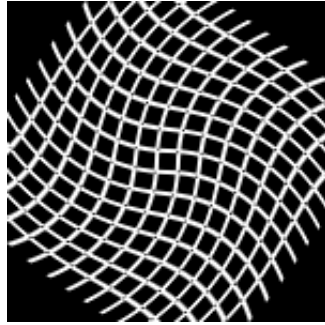
```
x+3*sin(y/10)
y+3*sin(x/10)
```



```
float xc=(x-width/2); float yc=(y-height/2); float r=sqrt(xc*xc+yc*yc); float
newr=r*sin(r/100); width/2+ newr*xc/r
float xc=(x-width/2); float yc=(y-height/2); float r=sqrt(xc*xc+yc*yc); float
newr=r*sin(r/100); height/2+ newr*yc/r
```



```
((x/width-0.5)*sin(3.141592654*y/height)+0.5)*width
y
```

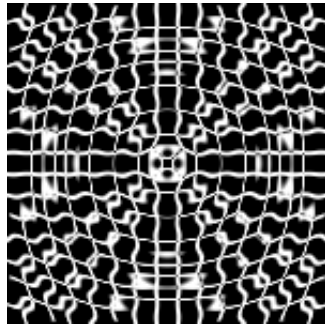


```
float xc=(x-width/2); float yc=(y-height/2); float r=sqrt(xc*xc+yc*yc); float
a=atan2(yc,xc);
```

```
float newA= a+3.141592654/2*r/200; width/2+r*cos(newA)
```

```
float xc=(x-width/2); float yc=(y-height/2); float r=sqrt(xc*xc+yc*yc); float
a=atan2(yc,xc);
```

```
float newA= a+3.141592654/2*r/200; height/2+r*sin(newA)
```

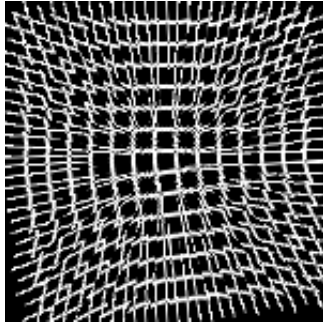


```
float xc=(x-width/2); float yc=(y-height/2); float r=sqrt(xc*xc+yc*yc); float
```

```
newr=r+3*sin(r/2); width/2+ newr*xc/r
```

```
float xc=(x-width/2); float yc=(y-height/2); float r=sqrt(xc*xc+yc*yc); float
```

```
newr=r+3*sin(r/2); height/2+ newr*yc/r
```

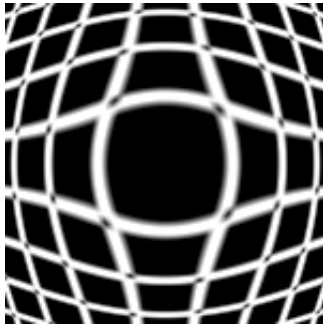


```
float xc=(x-width/2); float yc=(y-height/2); float r=sqrt(xc*xc+yc*yc); float
a=atan2d(yc,xc);
```

```
float newA= a+((int)a)%8-4; width/2+r*cosd(newA)
```

```
float xc=(x-width/2); float yc=(y-height/2); float r=sqrt(xc*xc+yc*yc); float
a=atan2d(yc,xc);
```

```
float newA= a+((int)a)%8-4; width/2+r*sind(newA)
```



```
float xc=(x-width/2); float yc=(y-height/2); float r=sqrt(xc*xc+yc*yc); float newr=r*r/
200;
```

```
width/2+ newr*xc/r
```

```
float xc=(x-width/2); float yc=(y-height/2); float r=sqrt(xc*xc+yc*yc); float newr=r*r/
200;;
```

```
height/2+ newr*yc/r.
```

Parameter	Type	Function
<i>oversampling</i>	int	The actual number of samples per pixel equals this number squared. For better antialiasing, increase the number.
<i>x, yExpression</i>	float	The expression to be placed. See above for examples.
<i>x, yDelta</i>	float	Sets the maximum distance that any pixel is expected to move, but doesn't actually move it. From any given pixel, it may be affected by any pixel with the Delta distance. This means it has to consider a much greater amount of pixels that may possibly affect the currently rendered pixel. This is bad. However, if you set a Delta value to too small an amount, you get errors if your expression tells the pixel to move beyond that limit. Therefore, it always takes some testing to balance between speed with errors, or accuracy with drastically slower renders. Our advice: start small and increase the size until the errors disappear.

### Synopsis

```
image WarpX( image,
  int oversampling,
  float expression xExpression,
  float expression yExpression,
  int xDelta,
  int yDelta
);
```

### Script

```
image = WarpX(
  image,
  oversampling,
  xExpression,
  yExpression,
  xDelta,
  yDelta
);
```

For multiline expressions:

```
image = WarpX(
  image,
  oversampling,
  {{ xExpr1, xExpr2, ... xExprN }},
  yExpression,
```

```
xDelta,  
yDelta  
);
```

**Command Line**

*shake -warpx oversampling xExpression etc..*

# Using the Curve Editor

## Chapter Summary

- About the Curve Editor
- Navigating the Curve Editor
- Working With Keyframes
- About Splines





## About the Curve Editor

The Curve Editor allows you to create, see, and modify animation curves or Lookup curves. You can change the curve type, as well as its cycling mode. For more information on specific curve types, see “About Splines” on page 502.

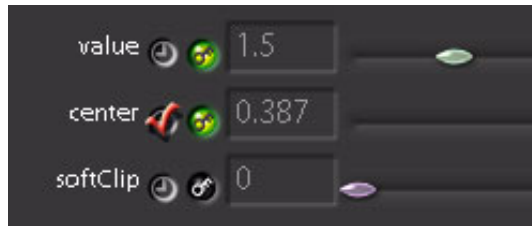
There are two curve editors—an editor for the animation of values and an editor for lookup-style curves that generally relate to color correction. This second type usually appears inside of the node’s parameter tab itself (for example, the Curves tab in the *ColorCorrect* node parameters), and is a subset of the animation curve editor.

## Loading and Viewing Curves in the Editor

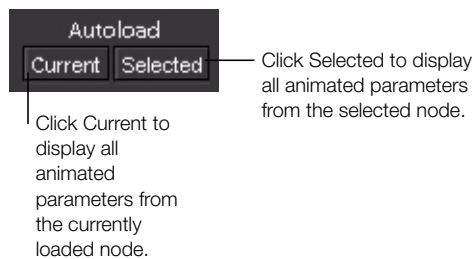
There are two basic ways to load curves into the Curve Editor:

- Inside of a Parameter tab, click Load Curve . A red check mark appears on the Load Curve button , and the parameter is loaded into the Curve Editor—it does not necessarily animate the value. Click Autokey  to create a keyframe for that frame (and enable load curve ).

In the following list, even though “value” is animated, only the “center” parameter is displayed in the Curve Editor.







- Certain functions have a built-in Curve Editor for color correction. The editor is embedded inside the Parameters tab. These functions are *Lookup*, *LookupHLS*, *LookupHSV*, and *HueCurves*. To activate the visibility of the curves, click the visibility button.
- You can also use the Viewing Filters in the Curve Editor to load a curve into the Curve Editor. Click Current to show all animated parameters from the currently loaded node. Click Selected to show all animated parameters from the selected nodes.



In the loaded parameters list, two columns toggle visibility and persistence.







- Click Visibility  to turn the display of a curve on and off in the Editor, but keep it in the list.
  - **Note:** You can also press **V** to toggle visibility.
  - Click Persistence  to keep a curve in the Editor regardless of the current/selected filter settings.
  - To remove a curve from the Editor, select the curve in the Curve List and press **Delete** / **Backspace**, or click Load Curve  in the node's parameters.
- The curve is removed from the editor, and the Load Curve button is disabled .

**Note:** If you rename a node that is already loaded into the Curve Editor, the new name is not automatically updated in the Curve List. To display the new node name, remove and reload the node into the Curve Editor.

## Navigating the Curve Editor

There are several tools to help you use the Curve Editor window.










- To frame a selected curve(s), press **F**.
- To frame all curves, press **Home** or click Home .
- To frame the selected curves, click Frame Selected Curves .
- To pan the Curve Editor:
  - Press the middle-mouse button, or **Option-click** / **Alt-click**, and drag.
  - Click and drag the navigation button .
- To zoom the editor:
  - Press **+** / **-** by the **Delete** (Macintosh) / **Backspace** (Linux/IRIX) key.
  - Press **Ctrl**-middle mouse or **Ctrl+Opt-click** / **Ctrl+Alt-click**.
  - Press **Ctrl** and click and drag the navigation button .
- To zoom the editor full screen, press the **Space bar**.

## Curve Editor Right-Click Menus

Function	Effect	Hot Key
<i>Add All Curves</i>	Adds all animated curves into the editor.	
<i>Remove Curves</i>	Removes selected curves from the Curve Editor. Does not delete the animation.	<b>Delete / Backspace</b>
<i>Visibility: Hide Curves</i>	Turns off visibility of selected curves. You can also toggle Visibility on the Curve List.	
<i>Visibility: Show Curves</i>	Shows selected curves. You select curves in the Curve List prior to this function.	
<i>Visibility: Toggle Visibility</i>	Inverts the visibility of all selected curves.	
<i>Select: All Curves</i>	Selects all curves in the Curve List.	<b>Command+A / Ctrl+A</b>
<i>Select: CVs</i>	Selects all keyframes on active curves. To select all keyframes on all loaded curves, press <b>Command+A / Ctrl+A</b> and then <b>Shift+A</b> .	<b>Shift+A</b>
<i>Display Timecode</i>	Toggles the time display from frames to timecode.	<b>T</b>
<i>Sticky Time</i>	When enabled, the current frame is set to the keyframe you are modifying.	<b>S</b>
<i>Time Snap</i>	When enabled, keyframes snap to frames, rather than float values.	
<i>Display Selected Info</i>	Displays data on selected curves and keyframes when active.	

## The Curve Editor Buttons

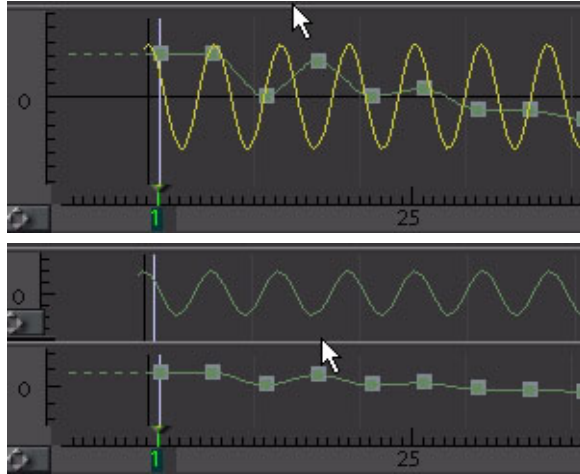
The following table describes the Curve Editor buttons.

Button	Name	Action
	Set Horizontal/Vertical Lock	Locks off movement on the X or Y axis. You can also press <b>X</b> to only allow movement on the X axis, press <b>Y</b> to allow movement on the <b>Y</b> axis. Press the key again to reenale movement.
	Keyframe Move Mode	This mode determines the behavior when keyframes are moved left and right past other non-selected keys. This behavior is discussed in "Keyframe Move Modes" on page 492.
	Reset Tangents	The Shake Hermite tangents are automatically set to the tangent of the curve. Modifying keys adjusts the tangents of neighbor keyframes. However, if you manually adjust a tangent, Shake recognizes this and disables this automatic adjustment. Click Reset Tangents to reset the tangents to their unset state.
	Flatten Tangents	Sets a Hermite-curve tangent horizontal to ensure smooth ease-in/outs. You can also press <b>H</b> .
	Apply Curve Function	Applies a function to the curve from a pop-up window. These functions are detailed in "Applying Functions to Curves" on page 494.
	Home	Frames all curves.
	Fit Selected Keys to Frame	Frames selected keyframes/curves.
	Display Waveform Toggle	Displays the waveform of any currently-loaded audio files from the Audio Panel.
	Color Pickers	These buttons appear in the Parameter tab for the <i>Lookup</i> , <i>LookupHSV</i> , and <i>LookupHLS</i> functions. They allow you to pick an input color (RGB, HSV, or HLS) and match it to a different color. Only visible curves receive keyframes on their curves. For example, if you only want to modify luminance, ensure the hue and saturation curves are disabled in a <i>LookupHLS</i> node.

## Splitting the Editor

You can separate the editor into two horizontal panes. This is useful when you have two or more curves with completely different Y scales, such as the pan and scale curves from a *Move2D* node. Each pane responds to its own visibility rules, so you can disable the curves in one pane, and enable the curves in another. The **V** key is particularly useful here, since the active pane is the last pane your cursor crossed.

To split the editor, grab the top of the editor and drag down.





## Working With Keyframes

This section discusses adding and modifying keyframes in the Curve Editor.

### Adding Keyframes

You can add and insert keyframes in several ways:

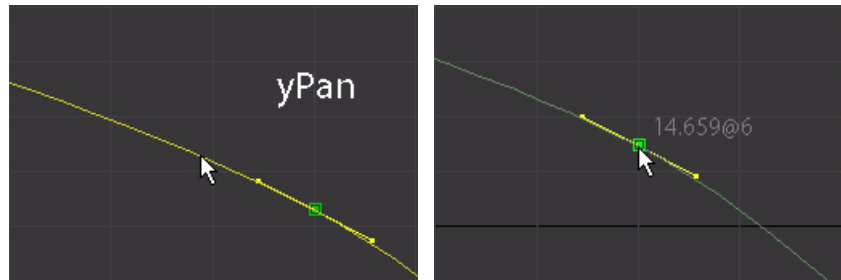
- In the node's parameter list, click Autokey . Modifying a value when Autokey is enabled creates a keyframe.
- In the Viewer toolbar, click Autokey  to create a keyframe for all onscreen parameters (for example, the *Move2D* node has x,yPan, x,yScale, angle, and x,yCenter associated with its onscreen controls).

**Note:** Creating keyframes in this manner overrides expressions.

- To insert a keyframe on a curve, **Shift**-click on a segment.
- Position the cursor over a curve and press **K**.

- In the Curve List, a keyframe is created when you enter or modify a value in the Val text field (next to the curve name). However, the parameter's Autokey button must be activated.
- The *Lookup*, *LookupHLS*, and *LookupHSV* functions have color picker pairs embedded in their dedicated Curve Editors that remain inside the Parameter tab. You can use these to enter keys. However, these keys are not related to time, rather to a particular color channel. Therefore, these keys become points on the color-correction curves. For more information, see “The Curve Editor Buttons” on page 487.

**Note:** In the Curve Editor, when the cursor passes over a curve, the curve name is highlighted; when the cursor passes over a keyframe, the keyframe values are displayed.



## Selecting Keyframes

In the Curve Editor, you can select keyframes in several ways:

- Click a keyframe, or **Shift**-click to select multiple keyframes.
- For a persistent Manipulator Box, press **B** while dragging. See “The Manipulator Box” on page 490.
- Drag-select a group of keyframes.

**Note:** When you drag-select a group of keyframes, a manipulator box appears around the selected keyframes. This box allows you to scale the keyframe group in X and Y, only X, or only Y; and allows you to move the group within the boundaries of the surrounding (not selected) keyframes. For more information, see “Modifying Keyframes” on page 490.

- **Shift**-drag to add keyframes to currently selected keyframes.
- **Command**-drag / **Ctrl**-drag to remove keyframes from currently selected keyframes.

**Note:** To remove keyframes from a group of selected keyframes within a manipulator box, press **Shift** and click the keyframes. For more information, see “Modifying Keyframes” on page 490.

- To select all keyframes on a curve, position the cursor over the curve, or select the curve in the Curve List, and press **Shift+A**.

- To select all keyframes in the Curve Editor, press **Command+A** / **Ctrl+A** (select all curves), and then press **Shift+A**.

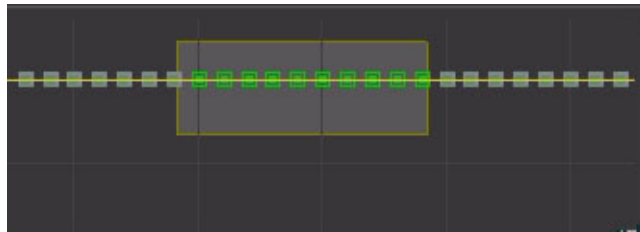
To deselect all frames, click an empty area of the Curve Editor.

## Modifying Keyframes

You can modify keyframes by selecting and moving the keyframes, creating a Manipulator Box, modifying them numerically, or by using the text fields in the Curve List.

### The Manipulator Box

You can use the Manipulator Box to move or scale a group of keyframes. The advantage the box has over simply selecting keyframes is that you can see the scale borders.



To use the Manipulator Box:

- Hold down the **B** key (for Box) and drag-select the keyframes.

The light grey manipulator box appears around the selected keyframes.



- To move the selection, position the cursor within the box and click and drag.



- To scale the selection in both X and Y, position the cursor at the corner of the box and click and drag. The box is scaled in X and Y around the opposite corner you select—if you grab the upper-right corner, the center of scaling is the lower-left corner.



- To scale the selection in X, position the cursor along either side edge of the box, and click and drag.



- To scale the selection in Y, position the cursor at the top or the bottom edge of the box, and click and drag.

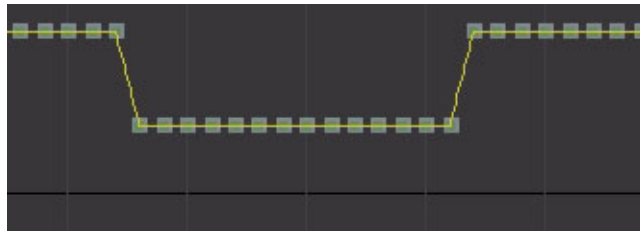
**Note:** Once a selection is made with the Manipulator Box, clicking on a keyframe or clicking outside of the box deselects the box.

- To add to the keyframes selected in the Manipulator Box, press **Shift** and click the additional keyframes.
- To remove selected keyframes, press **Shift** and click the keyframes you want to delete.

### Using Transform Hot Keys

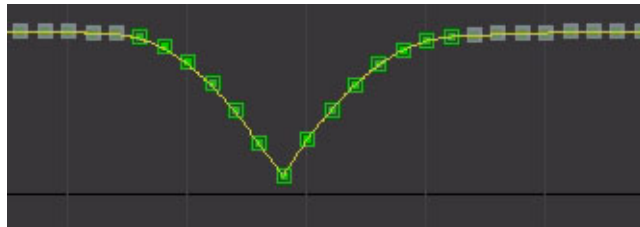
You can also move a group of keys using hot keys—you are not obliged to select the keys with the Manipulator Box.

*Move:* To move selected keys, press **Q** and drag.



*Scale:* Press **W** and drag; the first point clicked is the scaling center.

*Drop-off:* Press **E** and drag; a drop-off occurs on the pan.



### The Keyframe Text Fields

To modify a keyframe numerically, enter values in the text fields at the bottom of the Curve Editor window.

Key	Value	LTan	RTan
NA	NA	NA	NA

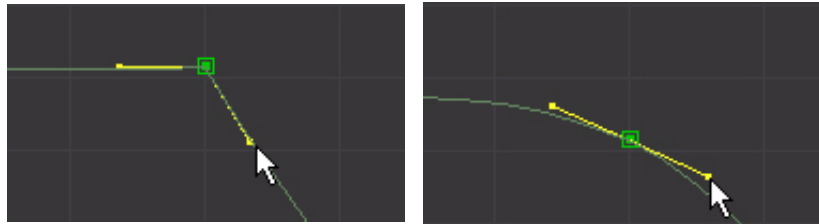
- The Key field is the time of the keyframe.
- The Value field is the value of the keyframe. Clever, eh?
- The LTan and RTan fields control the tangents. If the tangents are set to 0, the keyframe is flattened (on a Hermite curve). You can also press **H** to set horizontal tangents.


The text field displays “-----” when multiple keyframes are selected. To set all keyframes to that value, enter a number into the Value field.

- In the Curve List, the Val field displays the value of the curve at that point in time. You can also modify the value of a keyframe at that time in the Val text field next to the curve name. For the value to be saved, the Autokey button must be activated in the Parameter tab.

### Editing Tangents

- To break the tangents of a keyframe, press **Ctrl** and drag the tangent end.



- To rejoin a tangent, press **Shift** and click the tangent end.
- To reset a tangent, select the keyframe and click Reset .

### Keyframe Move Modes

In the Curve Editor, there are four keyframe move modes: Bound, Interleave, Push, and Replace. These modes control the behavior of the keyframes when the keyframes are moved left or right past non-active keyframes.

## Bound



When the Bound mode is set, the movement of a selected range of keyframes (whether contiguously or non-contiguously selected) is clamped by the adjacent keyframes. In the following example, keyframes A, B, and C are selected (highlighted in green) and moved left in the Curve Editor:

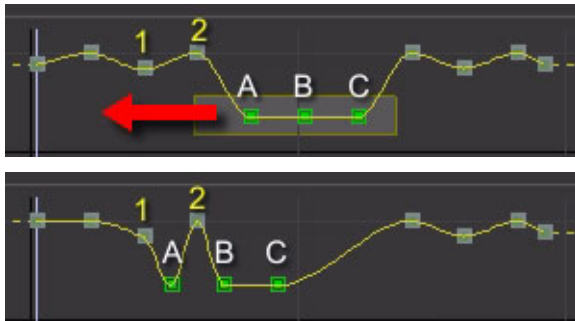


When moved, the selected keyframes cannot move beyond the adjacent keyframe in the curve, so keyframe A is at the same time as 2, and the distance between A and B shrinks. If you continue to drag left, keyframes A, B, and C are placed on the same frame.

## Interleave



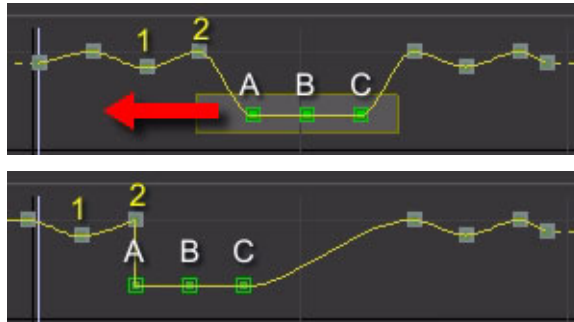
When the Interleave mode is set, the selected keyframes jump over the adjacent non-selected keyframes to the next segment of the curve.



## Push



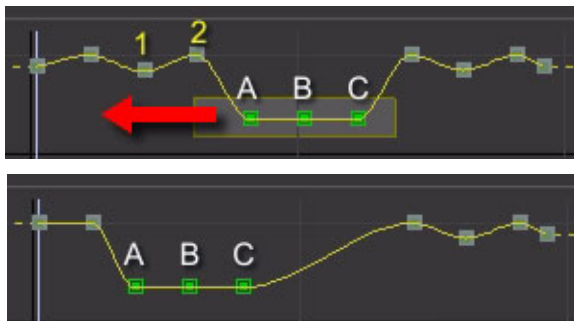
When the Push mode is set, the selected keyframes push the other keyframes along the curve. In the following example, the selected keyframes are pushed to the left of the Curve Editor. Therefore, keyframe A pushes keyframes 1 and 2, as well as all other keyframes to the left of keyframe 1.



## Replace



When the Replace mode is set, the selected keyframes replace the adjacent non-selected keyframe(s). In the following example, keyframes A, B, and C have slipped past the position of keyframes 1 and 2, removing them from the curve.




## Applying Functions to Curves

To apply functions such as Smooth and Jitter to curves or keyframes, use the function controls located in the lower right of the Curve Editor.

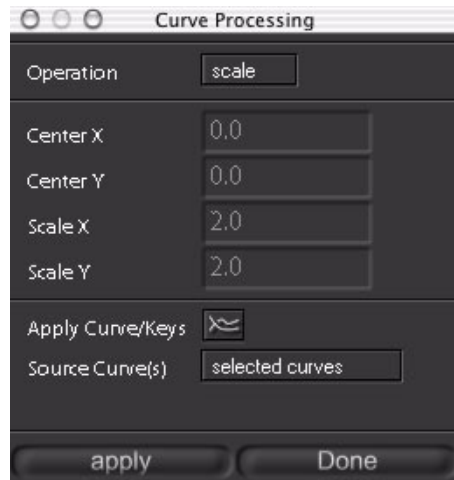
**To apply a function to a curve or to keyframes:**



- 1 Select the curve from the Curve List, or if applying the function to keyframes, select the keyframes in the Curve Editor.

**Note:** You can also select the curve from the Select Curves button, located to the right of the apply/curves buttons. Click and hold the default “selected curves” to choose from the curves that are loaded into the editor.

- 2 In the Curve Editor, click Apply Function .
- 3 In the Curve Processing window, select your Operation.

In the following example, the “scale” function type is selected.

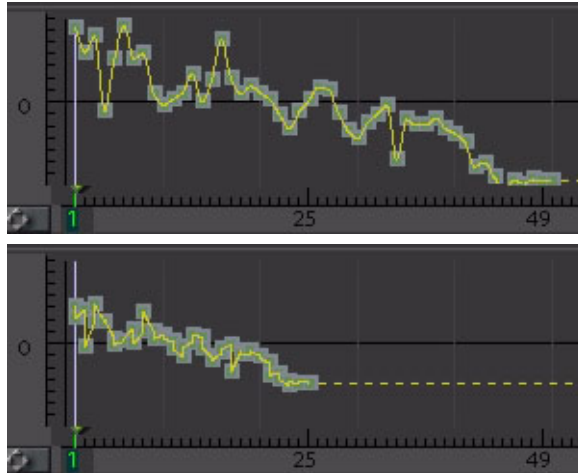


- 4 Where appropriate, enter the value(s) for the expression in the Amount field.
- 5 Do one of the following:
  - To apply the function to a selected curve, ensure “curves” is selected , and click apply.
  - To apply the function to selected keyframes, ensure “keys” is selected , and click apply.

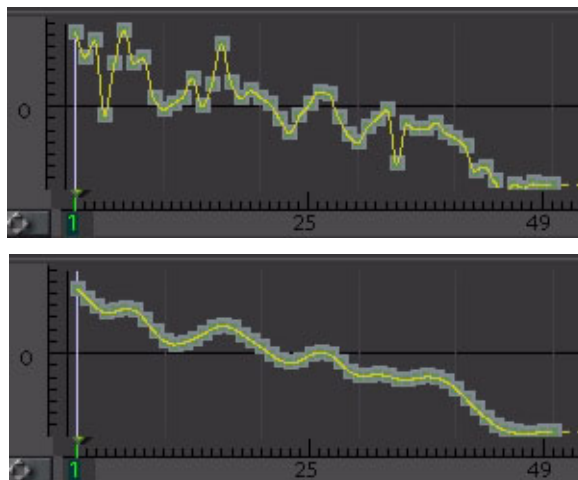
The selected function is applied to the selected curve.


The functions include:

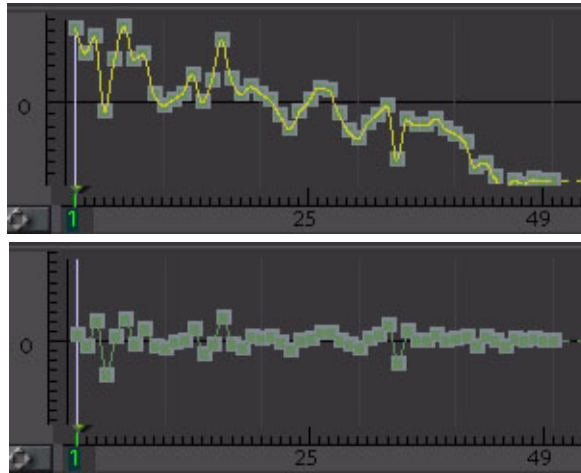
- *scale*: You can manually scale a curve using the Manipulator Box. This scale function in the Curve Processing window, however, allows you to enter specific scaling values. Enter the curve center and values. The following curve has a center of 0, 0 and .5, .5 for the scale values.



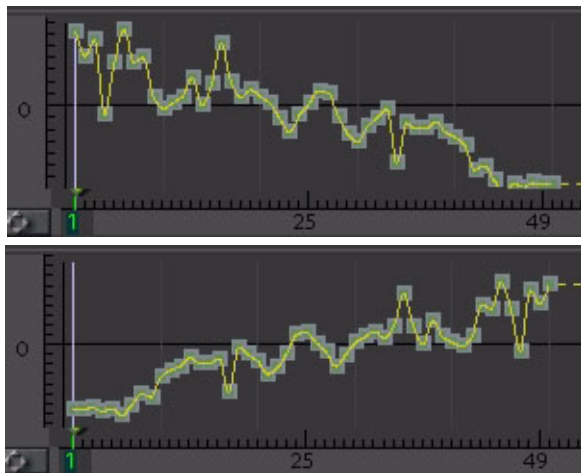
- *smooth*: “Blurs” the curve by the Amount value; the value that indicates how many neighbor keyframes are calculated in the smoothing. The higher the number, the smoother the result. In the following example, the second curve is the result of a smooth Amount of 10 applied to the first curve.



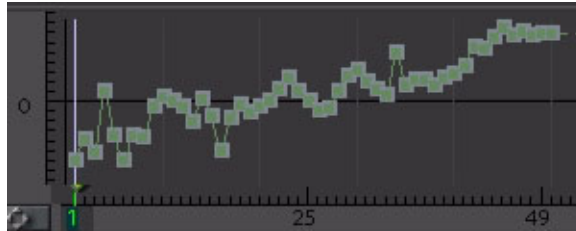
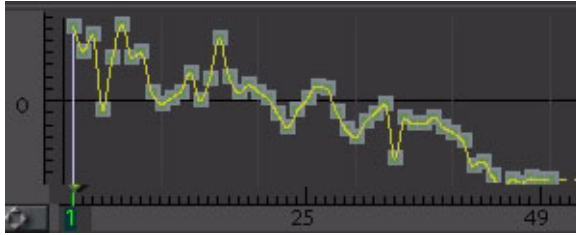
- *jitter*: The opposite of smooth, jitter removes all values except for the noise using the formula ( $Unmodified\ Curve - Smoothed\ Curve = Jitter$ ). Once the function is applied, the curve snaps down to approximately the 0 value, so the curve may disappear (press **F** or click Home  to reframe the editor). This is useful for stabilizing a jerky camera move. You can keep the overall camera move, but remove the jerkiness. The vertical scale of this image is much smaller than the first example snapshot.



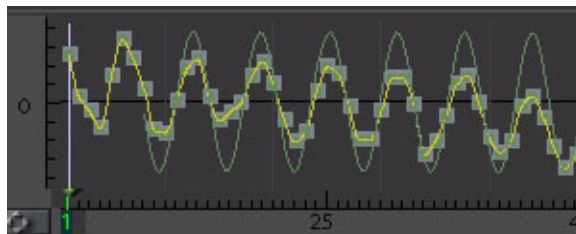
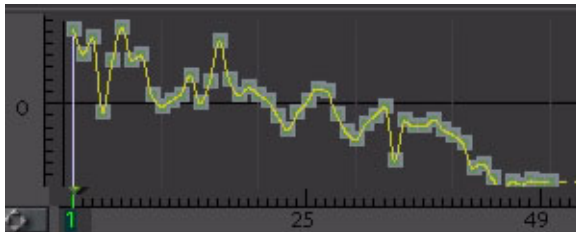
- *reverse*: Makes the curve go backward in time.



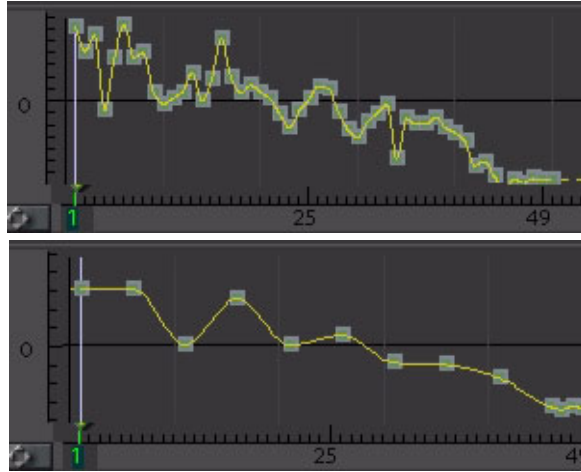
- *negate*: Flips the curve around the 0 point, so a value of 300 turns into a value of -300. Again, the curve may disappear, so press **F** or click Home to reframe the editor.



- *average*: Allows you to average two curves together. When selected, a button appears that allows you to select a second curve. Click this button to select the curve that is averaged with the current curve. In the following example, the random curve was averaged with a cos expression.



- *resample*: Replaces the curve or expression with a new sequence. This is useful for two purposes. First, you “bake” an expression, turning it into a keyframe curve. Second, you can adjust the number of keyframes that are on a curve. To set the resample, enter a frame range. For example, set *1-50* to enter 1 keyframe per frame from frames 1 to 50; *1-50x10* to enter only 5 keyframes every 10 frames, and so on.



## Copying and Pasting Keyframes

You can copy and paste keyframes from one curve to another curve.

**Note:** You cannot copy and paste keyframes from multiple curves.

### To copy and paste a keyframe:

- 1 In the Curve Editor, select the keyframe you want to copy and press **Command+C** / **Ctrl+C**.
- 2 Position the cursor over the curve (the same curve or a separate curve) at the time you want to paste the keyframe, and press **Command+V** / **Ctrl+V**.

The keyframe is pasted at the point in time the cursor is positioned.

### To copy and paste multiple keyframes on a single curve:

- 1 In the Curve List, select the curve that contains the keyframes you want to copy.
 


**Note:** To select a curve, you can also position the cursor over the curve in the Curve Editor. The curve name is displayed and the curve is highlighted in the Curve Editor.
- 2 Press **Shift+A** to select the keyframes on the selected curve.
- 3 Press **Command+C** / **Ctrl+C**.
- 4 In the Curve List, select the curve that you want to paste the keyframes.

- 5 In the Curve Editor, position the cursor at the point in time you want to paste the keyframes (the first keyframe pastes at the cursor position).
- 6 Press **Command+V** / **Ctrl+V** to paste the keyframes.

The keyframes are pasted at the point in time the cursor is positioned.

## Deleting Keyframes

To delete keyframes:

- Select the keyframes and press **Del** / **Delete** (Macintosh—the **del** key near the home and end keys).
- Move the time slider to the location of the keyframe, and click Delete Key . The Delete Key button only deletes keyframes related to the onscreen controls.
- If the keyframe is on an onscreen control, you can also press **Delete** / **Backspace** in the Viewer. This only deletes keyframes related to the onscreen controls.
- You can also use the right-click menu in the Parameter tab to delete keyframes. Go to the correct frame, right-click in the parameter's text field, and select Delete Current Key.

## Deleting Curves

To delete a curve:

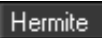
- Position the cursor over a curve, or select the curve in the Curve List, and press **Shift+A** (to select the points), and then press **Del** / **Delete**.
- In the Curve List, convert the curve to the Const (constant) curve type.
- In the node's parameters, right-click in the parameter text field and select Clear Expression.

**Note:** When a curve is deleted, it is replaced with a constant curve (set to the value of the curve at the point in time the curve was deleted).

## Modifying Curves

You can modify a curve type, its repetition mode, as well as apply filter effects (smooth, jitter extraction, etc.) on a curve.

### To change a curve type:

- 1 Select the curve (drag over the curve, or select the curve in the Curve List).
- 2 Click Curve Type  and select Hermite, Linear, CSpline, JSpline, NSpline, Step, or FCPBez (Final Cut Pro Bezier) curve. The most popular curves with the kids these days are Hermite, JSpline, and Linear.

For more information on curve types, see “About Splines” on page 502.

**Note:** You can only have one curve type per curve.

v	p	Node	Name	Val	Type	Cycle
		Move2D1	yPan	202.05	Hermite	KeepValue
		Move2D1	xPan	-248.0	Hermite	KeepValue
		Move2D1	angle	88.339	Hermite	KeepValue
Hermite(0,[0,68.81,68.81]@1,[90,0,0]@49,[88.3398,-14.2						
Hermite KeepValue						

You can also modify a value in the Val field in the Curve List.

Cycle Type **KeepValue** determines the behavior before the first keyframe and after the last keyframe:

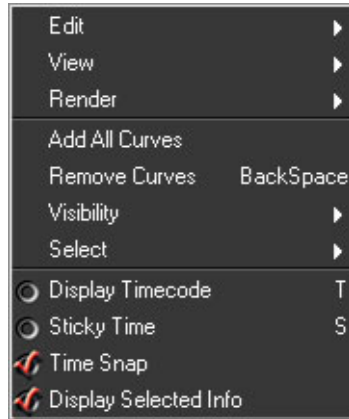
- KeepValue (the default setting): The value of the first and last keyframes is kept before the first keyframe and after the last keyframe.
- KeepSlope: Continues the tangent.
- RepeatValue: Repeats the curve between the first and last keyframes.
- MirrorValue: Reverses and repeats the curve.
- OffsetValue: Offsets and repeats the curve.

**Note:** To learn how to use local variables and expressions to control your curves, see “Lesson Four: Using Local Variables With Expressions” in the *Shake 3 Tutorials*.

For more information on the cycle types, see “About Splines” on page 502.

## The Right-Mouse Menu

The lower portion of the right-click menu in the Curve Editor includes additional options.



- Use Display Timecode to toggle between frame count and timecode count.
- Use Sticky Time to jump to the time of the keyframe you are modifying (so you view the proper frame).

**Note:** You can also press **S** to turn Sticky Time on and off.

- Use Time Snap to toggle the locking of the keyframes to frame increments.
- Use Display Selected Info to show the numerical information for selected keyframes.

## About Splines

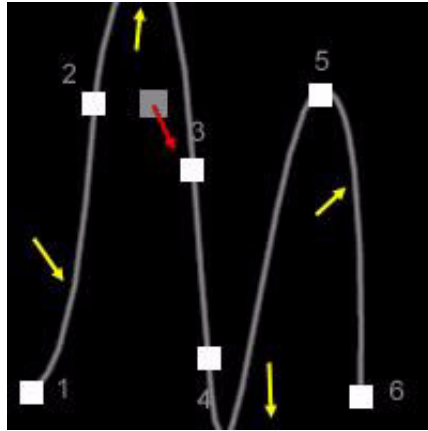
This section discusses the different spline types. Have your pillows ready, this has a high geek factor.

### Natural Splines

```
NSpline(cycle, value@key1,value@key2,...value@keyN)
```

```
NSplineV(time_value, cycle, value@key1,value@key2,...value@keyN)
```

The second-order continuity of natural splines ensures acceleration smoothly varies over time, so the motion is visually pleasing. The visual system is very sensitive to first- and second-order discontinuities, but not to higher-order discontinuities. But, in order to achieve the curvature continuity, the whole curve must be adjusted whenever a keyframe (CV) is moved. In the following example, when keyframe 3 is moved, the segments to keyframe 6 are changed. Bummer, even if the influence decreases very quickly as the number of intermediate keyframes increases. In addition, the keyframes completely define the curve, so there is no tangent control whatsoever. Beauty has its price.

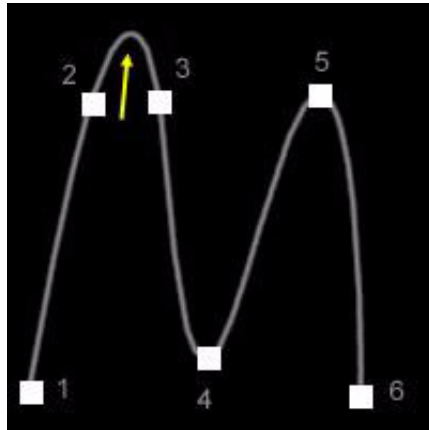


### Cardinal Splines

```
CSpline(cycle, value@key1,value@key2,...value@keyN)
```

```
CSplineV(time_value, cycle, value@key1,value@key2,...value@keyN)
```

Cardinal splines trade off curvature continuity for local control. When a keyframe moves, only 4 segments are affected (2 before, and 2 after the keyframe). In addition, for any keyframe, tangents are automatically computed to be parallel to the segment joining the previous keyframe and the next keyframe. They are the programmer's best friend because they are so simple to evaluate—only 4 points are needed, which simplifies data management (no tangent or other complicated stuff). Simplicity sometimes has the appearance of beauty.

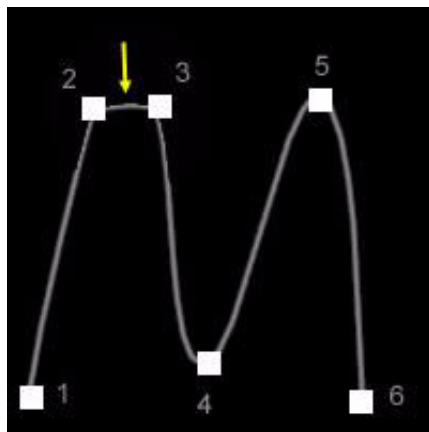


### Jeffress Splines

```
JSpline(cycle, value@key1,value@key2,...value@keyN)
```

```
JSplineV(time_value, cycle, value@key1,value@key2,...value@keyN)
```

Jeffress splines are similar to CSplines, except they are guaranteed to never overshoot. If two keyframes have the same Y value, a flat segment connects them. This is very nice for animation, since you have a good idea of your limits.

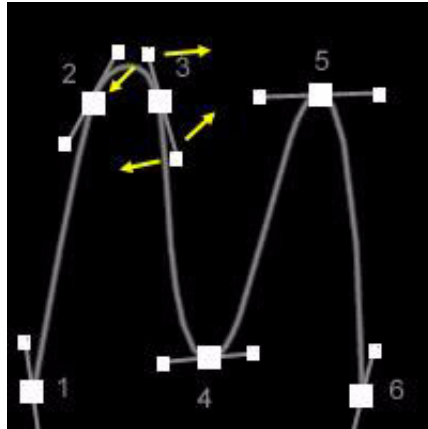


## Hermite Splines

```
Hermite(cycle, (value, tangent1, tangent2)@key1, ...  
(value, tangent1, tangent2)@keyN)
```

```
HermiteV(time_value, cycle, (value, tangent1, tangent2)@key1, ...  
(value, tangent1, tangent2)@keyN)
```

Hermite splines also give up on trying to produce curvature continuity, but they add tangent controls (so the animation is likely to look bad unless you eyeball the smoothness each time you move stuff around). You also have the ability to break the tangents (**Ctrl**-click on the handle end in the Curve Editor). It takes some effort to get right, but you can shape it the way you want. Some think that's the true beauty.

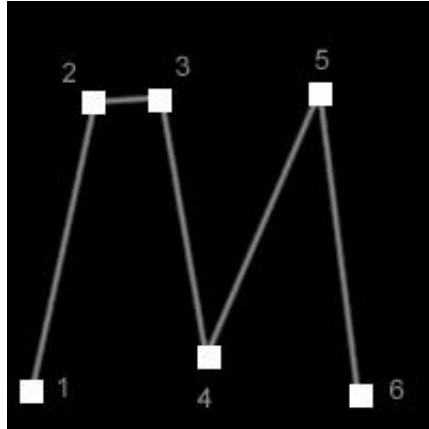


## Linear Splines

```
Linear(cycle, value@key1, value@key2, ... value@keyN)
```

```
LinearV(time_value, cycle, value@key1, value@key2, ... value@keyN)
```

With Linear splines, not much mystery. No smoothness, but you know exactly what you get. A kind of primitive beauty.

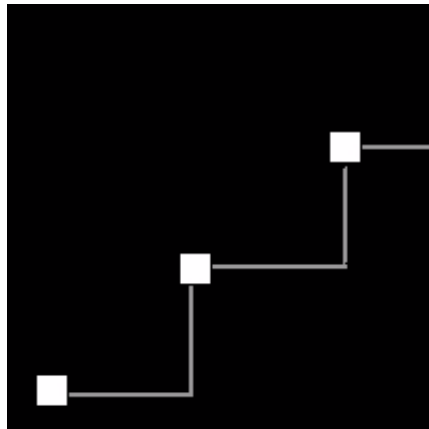


### Step Splines

```
Step(cycle, value@key1, value@key2, ... value@keyN)
```

```
StepV(time_value, cycle, value@key1, value@key2, ... value@keyN)
```

Step splines give a stair-stepping spline that maintains its value until the next keyframe. This is great for toggling functions.



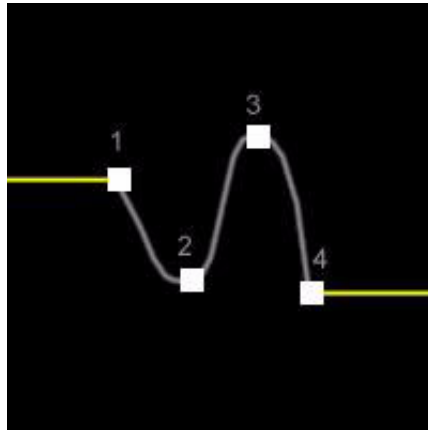
### Cycle Types

You can change how the curve cycles its animation before and after the curve ends. The cycle is represented by a dotted line in the Curve Editor. The value is declared with the first parameter of a curve, for example, Linear (CycleType,value@frame1,...). Each cycle type has a numeric code:

- 0 = Keep Value
- 1 = KeepSlope
- 2 = RepeatValue
- 3 = MirrorValue
- 4 = OffsetValue

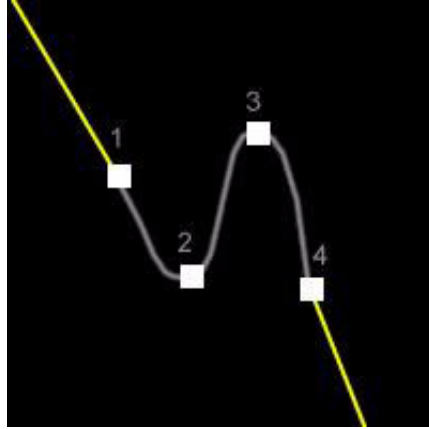
### KeepValue

Keeps the value of the first and last keyframe when a frame is evaluated outside of the curve's time range.



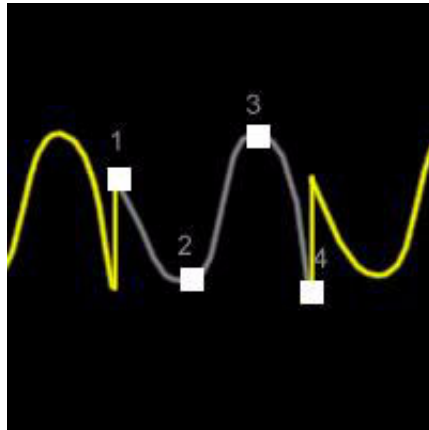
### KeepSlope

Takes the slope of the curve at the last keyframe and shoots a line into infinity.



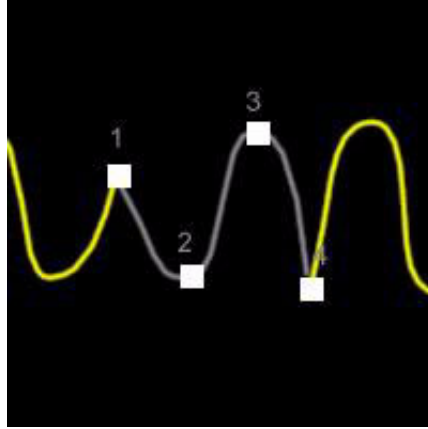
### RepeatValue

Loops the animation curve. It works best when you set the first and last points at the same Y value, and maintain a similar slope to ensure a nice animation cycle.



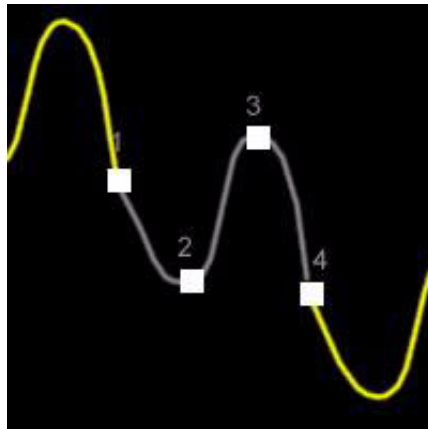
### **MirrorValue**

Also loops the animation, but inverts the animation each time the cycle repeats.



### **OffsetValue**

Also loops the animation, but offsets the repeated curve so that the end keyframes match up.





# Time View

## Chapter Summary

- About the Time View
- The Transition Function

## About the Time View

Use the Time View to view and arrange your nodes over time and to display a list of each node in the script. In the Time View, you can select nodes, evaluate nodes, and load parameters (as you can in the Node View). You can also set in and out points for your clips, as well as shift start and end frames to change the duration of your clip. Finally, the Time View allows you to change looping behavior on clips.

To display the Time View, click the Time View tab (on the right side of the Tool Tabs).

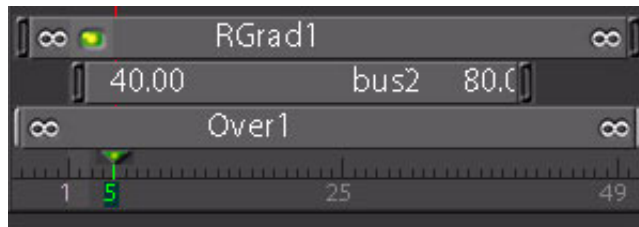
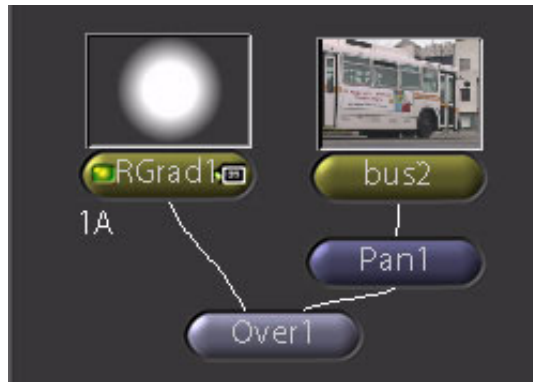


The Time View modifies parameters that are found inside each *FileIn* node. This means you can modify timing parameters in the Time View or with the Parameters tab.

As soon as timing changes are made, an internal node called *IReTime* is associated with a *FileIn* node. The *IReTime* node is saved into the script, but is invisible in the Node View. The *IReTime* parameters are controlled by the *FileIn* node parameters and in the Time View.

## Viewing Nodes in the Time View

Only image nodes or nodes with more than one input are listed in the Time View to reduce clutter. For example, in the following tree, the *Pan* node is not visible in the Time View.



The bus clip has 41 frames. Since the *RGrad* image is generated inside of Shake, it is considered to have an infinite time length. When nodes are combined, as with the *Over* node, the longer clip length is assumed.

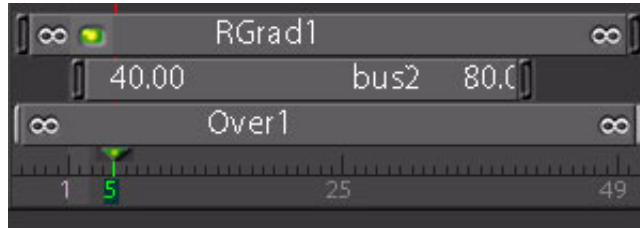
To display only currently active nodes, enable Select Group



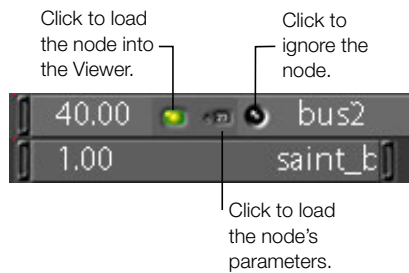
## Shifting Clips and In/Out Points

When a *FileIn* is created, timing characteristics are attached to the *FileIn* in the Parameter tab. Additional nodes on that clip inherit the source clip's characteristics, but you cannot adjust the In/Out points of following nodes.

In the following illustration, the *Over1* node has non-controllable clip ends. *RGrad1* and *bus2* have modifiable clip ends.



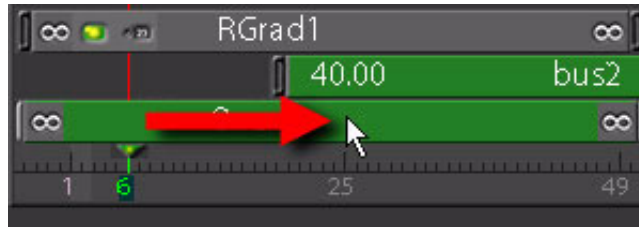
You can use any clip to evaluate a node, load its parameters, or toggle the Ignore switch (like in the Node View). When the cursor moves over a clip in the Time View, the buttons appear.

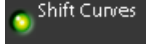


In the following example, the parameters for *bus2* are about to be loaded.



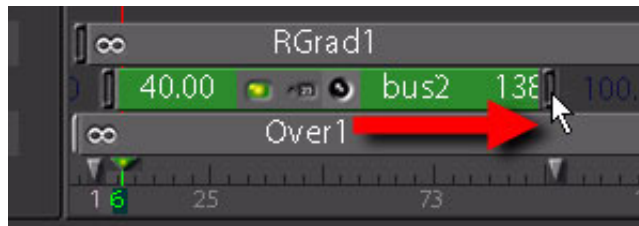
To adjust a clip location in time, drag left or right on any clip downstream from the node—the start and end frames are shifted uniformly. All attached nodes are also affected. In the following illustration, when *Over1* is dragged, all nodes above the *Over1* node in the tree (including the invisible *Pan1* node) are shifted. This means the frames of *bus2* are shifted in time, and any animation curves on *Pan1* and *RGrad1* are shifted as well. If only *bus2* is dragged, only *bus2* is modified in time unless Shift Curves is activated.



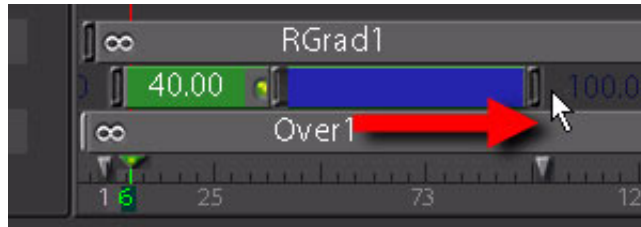
When Shift Curves is enabled , all curves attached to the shifted nodes are shifted as well, so the animation is carried with the shift. If Shift Curves is disabled, the curves remain locked in their positions.

To adjust individual start and end frames, press **Ctrl** and grab the end of a source node. In the following example, the *bus2* clip displays several bits of information. First, the clip endFrame has been extended to frame 60, so there is some information until frame 60. This endFrame corresponds to the endFrame parameter in the *FileIn* in the timing subtree.

If you grab the end of a clip, you set the expected active range. For example, if you drag the ends of *RGrad1*, it sets the boundaries for the *RGrad*. If you extend the end of the 41-frame *bus2* clip to 100, it assumes that there are more images on disk, which may not be the case. Note that the clip indicates that it expects frames up to 138 (on disk) to be placed up to frame 100 (in the Shake script).



To extend the last (or first) frames of a clip so the last frame (or first frame) is repeated, **Ctrl**-drag on a clip end. In the following example, the end of the *bus2* clip is **Ctrl**-dragged, and the last frame (80 on disk, 41 in the Shake script) is frozen to frame 100 in the Shake script.

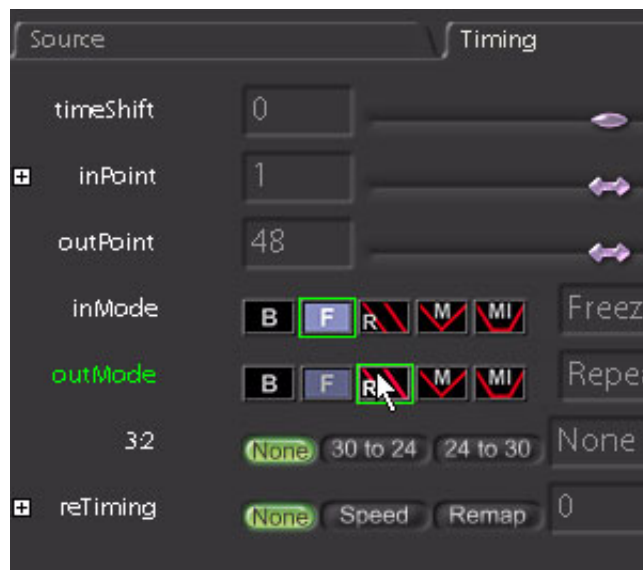


These controls are all stored in the *FileIn* parameters, with some of the controls in the timing subtree.

When you adjust the outer handles of a clip, you adjust the startFrames and endFrames. These determine at what frames the clip is visible.

When you adjust timeShift, you shift the startFrame and endFrame as well.

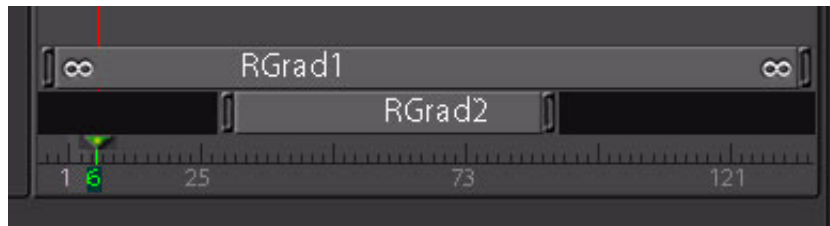
When you adjust the inner handles of a clip, you adjust the firstFrame and lastFrame parameters. These determine what part of the clip on disk is read. You can therefore choose to only read frames 10 to 20 of a 100-frame clip by adjusting the firstFrame and lastFrame parameters.



Notice that `firstFrame` and `lastFrame` reflect the frame range that is listed in the `imageName` parameter. If you change any of the parameters, the change is reflected in the other parameters.



Image nodes such as *RGrad* and *Ramp* have no preset range because they are generated by Shake. In the Time View, these types of nodes have infinity symbols on the clip edges to indicate that there is no end. To limit these nodes, grab the handles as you would with other source nodes.



## The Out Point

There are two different views on what the out point of a clip represents. For editors (usually), an out point is the frame at which there is no more image, and is therefore black. For a clip of 50 frames, the out point is then 51. To CG artists, the out point is the last frame to render, so the out point is 50. To add to the confusion, it must be kept in mind that even if you have 50 frames, you have meaningful information up to frame 50.99999... because of motion blur calculations and field rendering.

In the Time View, the right-click menu contains controls to handle the out point.

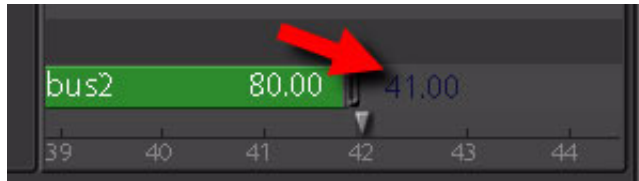
### InOut Point Display

This toggles the display of the in/out point in the Time View. When enabled, the in and out points are displayed when you click on the end of a clip.



### Const Point Display

When Const Point Display is enabled, the frame considered as the out point is toggled to the frame at which it becomes black, or the last frame on disk.



### FileIn Trim

The FileIn Trim toggle controls what happens when you drag the endFrame and lastFrame past each other (first image). In one mode, the buttons push each other together (second image). When enabled (third image), it continues past the trim point. This is useful because it keeps the original frame range around as reference.








### Changing Repeat Modes

You may have an endFrame that extends after your lastFrame parameter. For example, you may have a 20-frame clip that lasts from frames 1–100. The Repeat modes determine what happens in the extra 80 frames.

- **Ctrl**-drag the end of a clip to specify the repeat range.
- The repeat behavior is controlled in the Timing tab of the *FileIn* parameters.

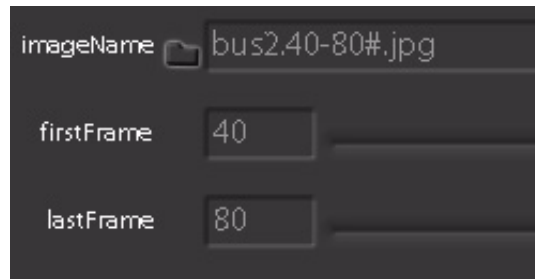
The behavior is controlled by the inMode and outMode settings—inMode controls frames between the startFrame and firstFrame, and outMode controls frames between lastFrame and endFrame. The following table lists the available Repeat modes.

Mode	Result	Example
<i>Black</i>	Black frames are inserted.	
<i>Freeze</i>	The first/last frames are repeated.	

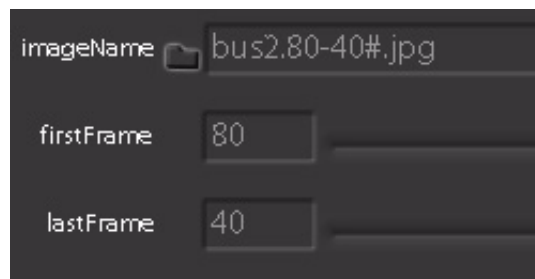
Mode	Result	Example
<i>Repeat</i>	The clip is continually repeated.	
<i>Mirror</i>	The clip is repeated, then reversed and repeated, with the cycle repeating until the clip's limits.	
<i>MirrorInc</i>	Like Mirror, but doesn't repeat start and end frame.	

## Reversing a Clip

The following is an example of one extremely hacky way to reverse a clip. Start with a 40-frame example clip.



Manually switch the firstFrame and lastFrame (the easiest way to flip).



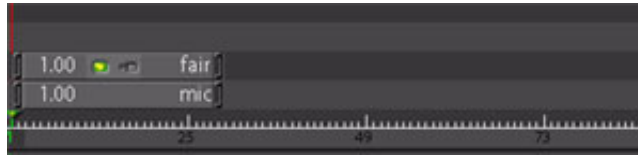
Yup, you'd think there would be some interactive way to do this.

## The Transition Function

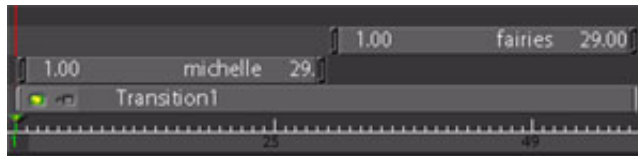
The *Transition* function is an editing node to mix or cut two clips together. It is unique in that it is really a shell to drive other functions that determine the mixing. Modifying it also modifies the timing of the second input.

The mixers can cut, which simply cuts to the second clip at the frame that it starts, or they can mix the second clip in over a specified frame range. The duration of the mix is determined by the overlap value, and starts at the frame that the second clip appears. If you modify the timing of the second clip, the overlap value changes as well.

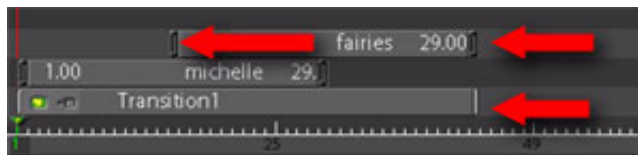
In the following example, there are two clips.



A *Transition* node appended to both clips offsets the second input clip.



You can now append one effect to both clips by attaching it to the *Transition* node. So, how is this different from just compositing the clips with an *Over* node and offsetting the second clip in time? It's different because you can easily dial in the overlap value to determine how many frames they overlap. Here, the overlap value is increased in the *Transition* node, and the second node shifts to the left as it increases. You can also shift the second clip, and read the overlap value in the *Transition* node.



On cut mode, the cut point occurs at the beginning of the second clip, not at the end of the first clip.

A common third parameter, *mixPercent*, is available in all mixers with the exception of “cut.” *mixPercent* determines the timing for the mixing. For example, for dissolve, if *mixPercent* is at 20, the second image is 20 percent mixed in and the first image is 80 percent. You can tune a curve interactively in the interface to adjust timing.

To create your own custom mixers in a *startup* .h file, you must do two things:

- Create a macro with two image inputs, *i1* and *i2*, and a float parameter named *mixPercent* that typically has the default value of “HermiteV(x,1,[0,50,50]@0,[100,50,50]@100)”. This gives you the animation curve that can then be tuned in the interface. You can also add other parameters.
- Declare the function as a mixer for *Transition* with the *nfxDefMixer* command in a *startup* .h file. The first parameter is the name of the mixer as it appears in the list. The second entry is the call to the macro:  
*nfxDefMixer("horizontalWipe","HWipe()");*

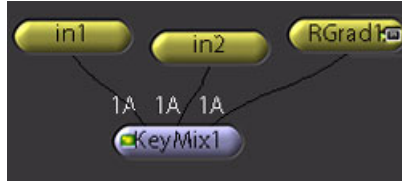
The following is an example from the *include/nreal.b* file for *horizontalWipe*:

```
image HWipe(  
    image i1=0,  
    image i2=0,  
    float blur=0,  
    int reverse=0,  
    float mixPercent="HermiteV(x,1,[0,50,50]@0,[100,50,50]@100)"  
)  
  
{  
    Color1 = Color(  
        max(i1.width,i2.width),  
        max(i1.height,i2.height),  
        1, 1, red, red, 1, 0);  
    Crop1 = Crop(Color1, 0, 0, width, height);  
    Pan1 = Pan(Crop1, mixPercent*width/100*(reverse?-1:1));  
    BlurMe = Blur(Pan1,blur,0,0);  
    IMult1 = IMult(i1, BlurMe , 1, 100, 0);  
    Invert1 = Invert(BlurMe , "rgba");  
    IMult2 = IMult(Invert1, i2, 1, 100, 0);  
    IAdd1 = IAdd(IMult1, IMult2, 1, 100);  
    return IAdd1;  
}  
  
nfxDefMixer("horizontalWipe","HWipe()");
```

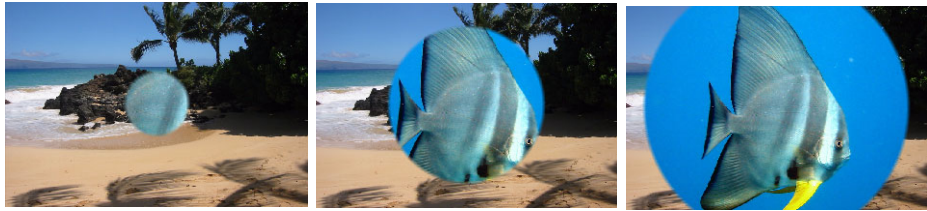
Notice that in *mixPercent* the default curve goes from 0,0 to 100,100 for *mixPercent*. Also, notice how the *Color* generator compares the two input resolutions to determine how large to make the *max* function.

## Roll Your Own

In this example, make your own transition mixer: Scale the radius of an *RGrad* node to create a radial wipe. Begin with a simple tree that feeds two *FileIn* nodes into a *KeyMix* node. The two clips are named *i1* and *i2* (to help later).



The following images show the effect that can be achieved by increasing and decreasing the the *RGrad* radius.



The tree is copied in the Node View (**Command+C** / **Ctrl+C**) and, pasted (**Command+V** / **Ctrl+V**) in a text file in your *\$HOME/nreal/include/startup* directory:

```
RGrad1 = RGrad(720, 486, 1, width/2, height/2, 1
    min(width,height)/4,
    min(width,height)/4, 0.5, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0);

in1 = FileIn("myclip.1-39#.iff",
    "Auto", 0, 0);

in2 = FileIn("myotherclip.1-39#.iff",
    "Auto", 0, 0);

KeyMix1 = KeyMix(in1, in2, RGrad1, 1, "A", 100, 0);

// User Interface settings

SetKey(
    "nodeView.KeyMix1.x", "156.75",
    "nodeView.KeyMix1.y", "127",
    "nodeView.RGrad1.x", "317.4916",
    "nodeView.RGrad1.y", "198.6512",
    "nodeView.in1.x", "67",
    "nodeView.in1.y", "201.125",
```

```

        "nodeView.in2.x", "202",
        "nodeView.in2.y", "198.3111"
    );

```

You can now prune a lot of the data, keep the bold sections of the above code, and format it as a macro. You also want to add the standard parameters of blur, mixPercent, and reverse. Copy the parameters from the *nreal.b* file's *HWipe* node (at the end of the file). Finally, calculate the resolution of the *RGrad* by comparing the two input sizes. The new bits appear in bold in the next example:

```

image RadialWipe(
    image in1=0,
    image in2=0,
    float blur=0,
    int reverse=0,
    float mixPercent="HermiteV(x,1,[0,50,50]@0,[100,50,50]@100)"
)
{

    RGrad1 = RGrad(
        max(in1.width,in2.width),
        max(in1.height,in2.height),
        1, width/2, height/2, 1,
        min(width,height)/4,          //This is the radius
        min(width,height)/4,          //This is the falloff
        0.5, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0);
    return KeyMix(in1, in2, RGrad1, 1, "A", 100, 0);
}

```

The maximum distance to expand the *RGrad* can be calculated by measuring the distance from the center to a corner, which can be done with the `distance()` function. (For more information, see “Expressions” on page 683.) Once this is calculated, multiply it by the mixPercent. Also, plug the blur value into the falloff parameter, with a check on the radius to see if falloff should equal 0 when radius equals 0. Also, add the command to load it as a mixer in the *Transition* node:

```

image RadialWipe(
    image in1=0,
    image in2=0,
    float blur=0,
    int reverse=0,
    float mixPercent="HermiteV(x,1,[0,50,50]@0,[100,50,50]@100)"
)

```

```

{
    RGrad1 = RGrad(
        max(in1.width,in2.width),
        max(in1.height,in2.height),
        1, width/2, height/2, 1,
mixPercent*distance(0,0,width/2,height/2)/100,
radius==0?0:blur,
        0.5, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0);
    return KeyMix(in1, in2, RGrad1, 1, "A", 100, 0);
}

```

```
nfxDefMixer("radialWipe", "RadialWipe()");
```

Now comes the tricky bit—reversing the mix. You may think multiplying by -1 inverts the transformation, but you'd be wrong. Instead, you often have to subtract the value from the maximum value that you expect, in this case the distance from the center to the corner. This is part of a conditional statement that tests to see if reverse is activated. Also, invert the mask in the *KeyMix* to help it out.

```

image RadialWipe(
    image in1=0,
    image in2=0,
    float blur=0,
    int reverse=0,
    float mixPercent="HermiteV(x,1,[0,50,50]@0,[100,50,50]@100)"
)

{
    RGrad1 = RGrad(
        max(in1.width,in2.width),
        max(in1.height,in2.height),
        1, width/2, height/2, 1,
reverse?distance(0,0,width/2,height/2)-
mixPercent*distance(0,0,width/2,height/2)/100:
mixPercent*distance(0,0,width/2,height/2)/100,
radius==0?0:blur,
        0.5, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0);
    return KeyMix(in1, in2, RGrad1, 1, "A", 100, reverse);
}

nfxDefMixer("radialWipe", "RadialWipe()");

```

Save all of this as a .h file in your *startup* directory.

As a final touch, open a ui .h file and add an on/off button for the reverse parameter:

```
nuxDefExprToggle( "RadialWipe.reverse" );
```

Now when you launch Shake, a new mixer in *Transition* is available.

Parameters	Type	Defaults	Function
<i>overlap</i>	int	0	The amount that the second clip is shifted earlier (to the left) to provide overlap of the two clips.
<i>mixer</i>	string	"cut"	Other default choices are "horizontalWipe," "verticalWipe," and "dissolve." However, you can add your own effects.
<i>blur</i>	float	0	This appears for horizontal and verticalWipe, and is used to soften the wiping edge.
<i>reverse</i>	int	0	This appears for horizontal and verticalWipe, and is used to flip the direction, for example, from left to right to right to left.

**Synopsis**

```
image Transition(  
    image i1,  
    image i2,  
    int overlap,  
    const char * mixer,  
    ....  
);
```

**Script**

```
image = Transition(  
    i1,  
    i2,  
    overlap,  
    "mixer",  
    ....  
);
```



# Painting, Rotoscoping, and Other Image Functions

## Chapter Summary

- Using the *QuickPaint* Node
- Using the *RotoShape* Node
- Other Image Functions
- Using the *QuickShape* Node

## Using the QuickPaint Node

The *QuickPaint* node is a touch-up tool to fix small element problems such as holes in mattes or scratches/dirt on your plates. It is a procedural paint that allows you to change strokes after they have been placed. This helps to emphasize its key feature: It is just another compositing tool that can easily be used in conjunction with the other Shake nodes. You can apply the effect and easily ignore it, remove it, or reorder it after you have applied your paint strokes. It is mighty handy, but it is not intended as a full-featured, paint-a-masterpiece paint package.

The first input of the node is for the background and also acts as the Clone source. The second input is for the Reveal source.

## Setting Resolution

You can apply a *QuickPaint* node to another node, or you can create a floating *QuickPaint* node that can be later applied to a different node with a Layer function (or as a mask operator). A floating node takes the resolution of the defaultWidth and defaultHeight.

To set the resolution, create an Image-*Color* node or a Transform-*Window* node and attach the *QuickPaint* node. Then, set the resolution in the *Color* or *Window* node parameters.

**Note:** In the *Color* node, the alpha channel is set to 1 by default.

Keep resolution in mind because *QuickPaint* does not paint beyond the boundaries of the frame.

## Edit Mode Versus Paint Mode


When the *QuickPaint* node is active, the associated tools appear on the Viewer toolbar.



Three subtabs, Paint Controls, Edit Controls, and Paint Globals, also appear in the Parameters tab.

The first button on the Viewer toolbar is the Paint/Edit mode toggle.

In Paint mode , you can apply new brush strokes, and the Paint subtab is selected in the Parameters tab.

In Edit mode , you can modify the current paint stroke or any previous paint stroke. You can control the paint characteristics (color, size, brush type, opacity, softness), the position or shape of the stroke, or apply a write-on/off effect. For more information on Edit mode, see “Modifying Paint Strokes” on page 531. In Edit mode, the Edit Controls subtab is selected.

Clicking the subtabs also toggles the Paint and Edit modes—if you click the Edit Controls subtab, the mode is set to Edit; if you click the Paint subtab, the mode is set to Paint. To quickly switch to Paint mode, select a brush type on the Viewer toolbar.

In Edit mode, select any stroke to modify by clicking on the invisible stroke—the stroke appears. You can also adjust the strokeIndex slider back and forth to expose previous strokes numerically.








## Using the Brushes

There are five basic brush types. One modifier changes the drop-off of the five brush types. To use a brush, make sure that you are in Paint mode (click the paint/edit toggle or select a brush), and paint.

To change the size of a selected brush, **Ctrl**-drag in the Viewer. You can also numerically set the brush size in the *QuickPaint* parameters.

To create a straight line, hold down **Shift** while drawing in the Viewer.

The following table contains the basic brush tools.

Button	Name	Action
  Soft falloff      Hard falloff	Hard/soft toggle	When soft is selected, paints any brush type with a soft falloff. When hard is selected, paints any brush type with a hard falloff. You can also press <b>F3</b> to toggle between the soft and hard falloff.  This is not a brush—it just modifies other brushes.
	Paint brush	Applies RGBA color to the first input.
	Smudge brush	Smears the pixels. The smear brush should always use the hard falloff setting.
	Eraser brush	Erases previously applied paint strokes only. Does not affect the background image.
	Reveal brush	Exposes the image connected to the second node input. If there is no input, it acts as an <i>Outside</i> node, and punches a hole through the paint and the first input source.
	Clone brush	Copies from whatever is created by the paint node or comes from the first image input. To move the brush target relative to the source, press <b>Shift</b> -drag.

Press **F1** to select your last-used brush type. With this, you can quickly toggle through paint > erase > paint.

### Picking a Paint Color








There are several ways to pick your paint color and opacity.



The Color Picker is located in the *QuickPaint* parameters. Use the standard procedures to select your color—from the Color Picker, with the Virtual Color Pickers, or by selecting a color from the image in the Viewer. With the cursor in the Viewer, you can also press **F2** or **P** to temporarily jump into Color Picker mode.

The Color box on the Viewer toolbar  indicates the current color.

When you paint, you apply unpremultiplied strokes, so if you adjust the alpha slider in the Parameters tab, it does not change what you apply to the RGB channels. However, modifying opacity changes all four channels.

## Other Viewer Controls

Button	Name	Action
	Active channels	These buttons indicate what channels are painted. For example, to only touch up an alpha channel, turn off the RGB channels.
	Frame mode toggle	When Frame mode is selected, you only paint on the current frame.
	Interp toggle	When interpolate mode is selected, brush strokes are interpolated. For example, paint a stroke on frame 1, and then go to frame 20, and paint a stroke on frame 20. When you scrub between 1 and 20, the stroke interpolates. Beyond frame 20 or before frame 1, the image is black. To insert a second interpolation stroke, click the Interp toggle until Interp is selected again, and use the strokeIndex slider to select the stroke to modify.
	Persist toggle	When persist mode is selected, the stroke persists from frame to frame. The stroke does not change unless you switch to Edit mode and animate the stroke.
	History Step	Use the History Step buttons to step backward or forward through your history. This slips you into Edit mode. As you step backward, the strokeIndex parameter in the Edit Controls subtab indicates the current stroke number. Although you can edit any brush at any time, this sets the point that you are evaluating your paint. You can then, for example, step back several strokes, insert a new stroke, and then step forward. The later strokes are placed on top of your new stroke because the new stroke is earlier in the step history.
	Magnet drag mode	When magnet drag mode is enabled, and you are in Edit mode, you can select a group of points on a stroke. If you click near the middle of the points and drag, the points near the selected point are dragged more than points farther away. You can also press and hold <b>Z</b> to temporarily activate this mode if you are in Linear drag mode.
	Linear drag mode	Click the Magnet drag mode button to toggle to Linear drag mode. When in Edit mode and you drag a group of points, they all move the same amount.

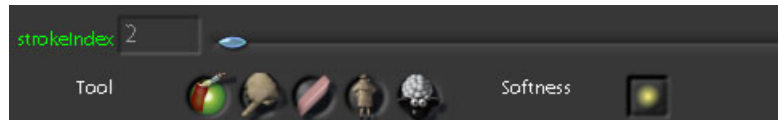
Button	Name	Action
	Delete Last Stroke	Removes your last stroke. Sadly, <i>QuickPaint</i> and Undo are not on good terms, thus we have this...
	New Canvas	Removes all strokes from your canvas.


## Modifying Paint Strokes

You can animate strokes with the Interpolation or Frame setting, or you can modify any stroke after it has been created with the Edit mode. To switch to Edit mode, click the Paint/Edit mode toggle, or click the Edit Controls subtab in the *QuickPaint* parameters. For more information on converting paint stroke modes, see “Converting Stroke Modes” on page 533.

In Edit mode, you can select a stroke in one of three ways:

- Click the stroke. An onscreen control appears on the stroke.
- Use the `strokeIndex` parameter in the Edit Controls subtab. Each stroke is assigned a number, and can be accessed by the `strokeIndex`.





- Use the History Step buttons . This not only selects the stroke, but only draws up to that stroke. Even though there may be later strokes in the history list, they are not drawn until you step back to them with the Forward History Step button.

To add to your selected points on a paint stroke, press **Shift** and drag the new points. **Ctrl**-drag to remove from the active points. You can also simply select a point and drag.

**Note:** To drag-select the control points of a stroke when multiple onscreen controls (from different nodes) are displayed in the Viewer, first move the cursor over the shape you want to edit. Next, drag-select the points. This behavior applies to *QuickPaint*, *RotoShape*, and *QuickShape* objects. For example, you can display the onscreen controls for the shapes of two different *QuickPaint* nodes by loading the parameters of one node into the Parameters1 tab, and **Shift**-clicking on the right side of the second node to display the parameters in the Parameters2 tab. Also, if the points you want to drag-select are within a DOD bounding box, move the cursor over the shape inside the DOD, and then drag-select the points.

To insert a new point, **Shift**-click on a segment. To remove a point(s), select the point and press **Delete**.

You can move selected points. Use the standard Autokey controls to set your keyframes. As discussed in the table above, you can drag the points in two different ways. If Linear drag mode  is selected, the knots move the same amount. If Magnet drag mode  is selected, the points nearest the cursor move the most. To temporarily activate this mode, press and hold the **Z** key and drag. Using these tools, you can animate your strokes.

### Attaching a Tracker to a Paint Stroke

In Edit mode, a preexisting track can be attached to a paint stroke.

#### To attach a track to a paint stroke:

- 1 Ensure the paint stroke is a Persistent stroke.

**Note:** For information on converting a paint stroke from Frame to Persistent mode, see “Converting Stroke Modes” on page 533. You can also convert the stroke after the track is attached.

- 2 In the Viewer, right-click the selected stroke and select an available track from the “Add tracker to stroke” list.

**Note:** Although only a single control point appears on the paint stroke when attached to the track, the track is applied to the entire paint stroke. You cannot apply a track to individual control points on a paint stroke.

The selected track information is then fed into the stroke’s panX and panY parameters as an offset.

Once a tracker is attached to a paint stroke, the track information is displayed in the Viewer on the paint stroke, as well as in the Edit Controls subtab next to the Convert Current Stroke button. The track information is only displayed if that stroke is selected in the strokeIndex or in the Viewer.

To remove the track, right-click the paint stroke in the Viewer and select “Remove tracker reference.”

Once the paint stroke is attached to the track and you have achieved the result you want, you can “bake” the tracker information into the stroke.

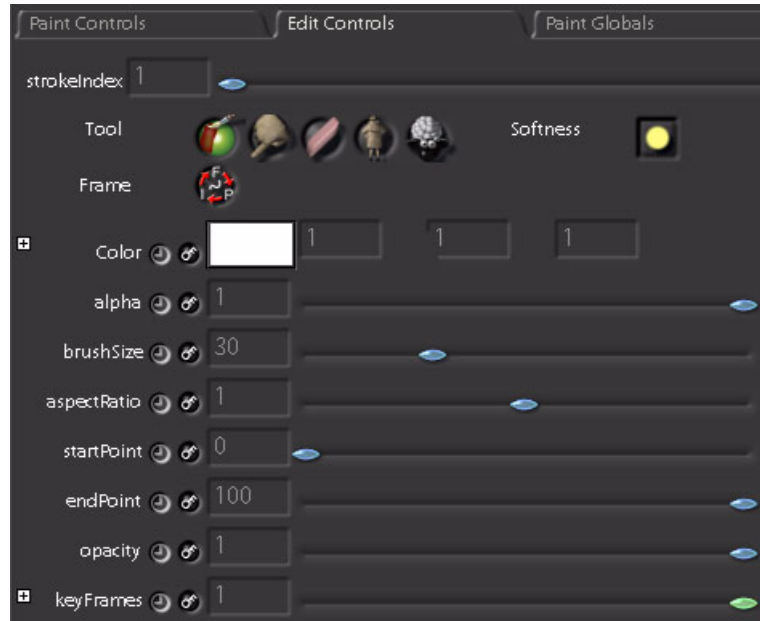
To bake the track:

- Ensure Edit mode is enabled.
- Select the stroke (click the stroke in the Viewer or select the stroke in the strokeIndex parameter in the Edit Controls subtab).
- Right-click the paint stroke and select “Bake tracker into panX/panY.”

The keyframes are applied to the paint stroke and the track information no longer appears in the Edit Controls subtab.

## Modifying Paint Stroke Parameters

You can also use the Edit Controls subtab in the *QuickPaint* parameters to modify your strokes.



In the Edit Controls subtab, click a brush type in the Tool row to switch brush types. You can also click the Softness falloff button to switch between a hard and soft falloff, or change the stroke type with the Convert Current Stroke button (see below). Additionally, you can alter or animate the color, alpha, opacity, brushSize, or aspectRatio of the current stroke.

The startPoint and endPoint parameters determine the percentage point that the stroke starts drawing and the point that it ends. You can therefore make a stroke animate its writing by setting keyframes for the endPoint from 0 to 100 over several frames. All stroke types can receive this animation, although the Frame mode only alters the current frame—no animation is applied (since the strokes only exist for one frame). Keep in mind they describe percentage of the spline. Therefore, if you change point positions relative to each other, you may also change at what point a certain percentage point is on the spline, giving you possible fluctuations in the line drawing animation.



The keyFrames parameter (in the Edit Controls subtab) is a placeholder so that keyframe markers appear in the Time Bar—it has no other interactive function.

### Converting Stroke Modes

You can convert paint strokes in the Edit Controls subtab by clicking the Convert Current Stroke button. For example, you can convert a stroke created in Frame mode to a Persistent stroke.

### Interpolating Paint Strokes

In this example, frame 1 contains 3 separate paint strokes, and frame 50 also contains 3 separate paint strokes. Interpolate the second paint stroke on frame 1 (the number “2”) with the second paint stroke on frame 50 (the number “5”).

- 1 On the *QuickPaint* toolbar, ensure Paint mode  is enabled.
- 2 Ensure Frame mode is enabled  .
- 3 At frame 1, draw three paint strokes.



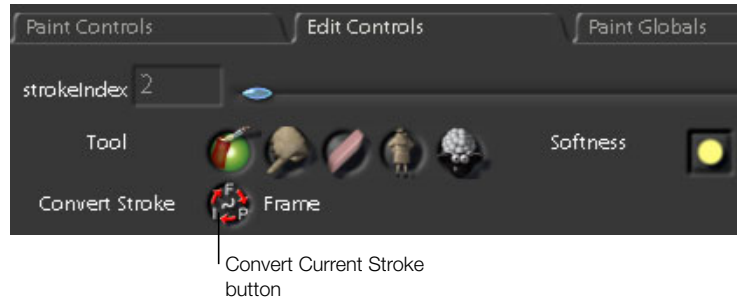
- 4 At frame 50, draw three more paint strokes.



**Note:** In the above illustrations, each number is a single paint stroke.

- 5 In the *QuickPaint* toolbar, click the Paint mode button to toggle to Edit mode.  
**Note:** You can also click the Edit Controls subtab in the *QuickPaint* parameters.
- 6 In the Edit Controls subtab, select stroke 2 from the *strokeIndex*.

- 7 Click the Convert Current Stroke button.



In the Convert Stroke window, the message “Converting Stroke 2” appears (because stroke 2 is selected in the strokeIndex).

- 8 In the Convert Stroke window, enable Interp.
- 9 Enter 2, 5 in the Stroke Range field.

This instructs Shake to combine paint strokes 2 and 5 into one interpolated stroke.

**Note:** Because there is more than one paint stroke on a frame, the comma syntax must be used for interpolation. If frame 1 contained only one paint stroke, and frame 50 contained only one paint stroke, and you wanted to interpolate the two strokes, you could enter 1-2 or 1, 2 in the Stroke Range of the Convert Stroke window to interpolate between paint stroke 1 and paint stroke 2 in the node.

As another example, if you wanted to interpolate between a stroke on frame 1 (stroke number 1), a stroke on frame 5 (stroke number 2), a stroke on frame 10 (stroke number 3), and a stroke on frame 15 (stroke number 4), enter 1, 2, 3, 4 to interpolate between all strokes.

- 10 Click OK.

Scrub between frames 1 and 50, and the 2 (the second paint stroke in the node) and the 5 (the fifth paint stroke in the node) interpolate.



Because Frame mode was enabled when you drew the paint strokes, the strokes that are not interpolated (the numbers 1, 3, 4, and 6) only exist at the frame in which they were drawn (frames 1 and 50).

### Converting Paint Strokes From Frame to Persistent

Normally, to paint a stroke that exists on all frames, you select the Persistent mode in the *QuickPaint* toolbar before you draw your strokes. In case you forget this step and draw your strokes in the default Frame mode, you can use the Convert Current Stroke feature to convert a paint stroke, or multiple paint strokes, to Persistent mode.

- 1 Once the paint stroke is drawn (in Frame mode), click the Edit mode button in the *QuickPaint* toolbar.

- 2 In the Edit Controls subtab, select the stroke in the strokeIndex.

**Note:** You can change the selected stroke later in the Convert Stroke window.

- 3 Click the Convert Current Stroke button.

- 4 In the Convert Stroke window, enable Persist.

To convert multiple strokes, do one of the following:

- To convert all paint strokes from Frame to Persistent, enter the whole stroke range in the “Convert stroke(s)” text field. For example, if you had a total of 12 strokes, enter *1-12* to convert all 12 strokes to Persistent mode.
- To convert selected strokes, enter the desired range in the “Convert stroke(s)” text field. For example, enter *3-8, 11* to convert paint strokes 3 through 8 and stroke 11 (of the 12 total strokes).

**Note:** Enter a frame range using standard Shake syntax (1-100, 20-50x3, etc.).

- To convert a single stroke, enter the stroke number in the Convert multiple strokes field.

- 5 Click OK.

The strokes are converted from Frame to Persistent.

### Converting Paint Strokes From Persistent to Frame

You can also convert a Persistent paint stroke to only appear in a specific frame range.

- 1 Once the paint strokes are drawn (in Persistent mode), click the Edit mode button in the *QuickPaint* toolbar.

- 2 In the Edit Controls subtab, select the stroke to convert in the strokeIndex.

- 3 Click the Convert Current Stroke button.

In the Convert Stroke window, the message “Converting Stroke from Persist to Frame” and the Frame Range field appear.

- 4 Enter the frame range for the paint stroke. For example, to draw the stroke on frames 1 and 3, and from frames 10 to 20, enter *1, 3, 10-20*.

- 5 Click OK.

The converted stroke appears on frames 1, 3 and 10 through 20.

### QuickPaint Hot Keys

The following table lists the *QuickPaint* hot keys.

Key	Function
<b>F1</b>	Last Brush type.
<b>F2</b>	Pick Color.
<b>P</b>	Pick Color.
<b>F3</b>	Hard/Soft toggle.
<b>Z</b>	Magnet drag in Edit mode.

### QuickPaint Function Parameters

The following tables list the *QuickPaint* parameters.

Parameters, Globals subtab	Type	Defaults	Function
<i>snapshotInterval</i>	int	20	The setting for how many strokes are applied before the image caches. For low resolution, you can probably set this lower, but if you set it too low when working with film plates, you spend all your time caching 2K plates, which is bad.
<i>maxPressure</i>	float	100	The maximum amount of pressure you can apply.
<i>pressureCurve</i>	curve	linear curve	You can control the pressure response of the stylus by loading this parameter into the Curve Editor. You can also, of course, change the Wacom settings outside of the software.
<i>compressSave</i>	int	on	When enabled, the node is saved in binary format, which is faster and smaller. When disabled, it saves it in an editable ASCII format (described below). See "Use Caution With compressSave" on page 538.
<i>moveExpression</i>	curve		This expression controls the drop-off curve for the Magnet drag mode when you move a group of points.

### Use Caution With compressSave

If you have a considerable amount of work, you should ensure that `compressSave` is enabled. There is a possibility of making a file larger than your machine can actually read back in.

Parameters, Paint subtab	Type	Defaults	Function
<i>Color</i>	float	1, 1, 1	The color of the paint stroke to be applied next.
<i>alpha</i>	float	1	The alpha channel of the paint stroke. This does not modify the color, as the strokes are not premultiplied.
<i>brushSize</i>	float	30	The size of the brush. You can also use <b>Ctrl</b> -drag in the Viewer to set the brush size.
<i>aspectRatio</i>	float	1	Aspect ratio of the circular strokes.
<i>constPressure</i>	int	off	When enabled, the Wacom stylus pressure is ignored.

Parameters, Edit subtab	Type	Defaults	Function
<i>Color</i>	float	1, 1, 1	The color of the current paint stroke.
<i>alpha</i>	float	1	The alpha channel of the current paint stroke. This does not modify the color, as the strokes are not premultiplied.
<i>opacity</i>	float	1	A fade value applied to the RGBA channels.
<i>brushSize</i>	float	30	The size of the brush. You can also use <b>Ctrl</b> -drag in the Viewer to set the brush size.
<i>aspectRatio</i>	float	1	Aspect ratio of the circular strokes.
<i>startPoint</i>	float	0	The percentage point on the stroke that the paint starts to draw. Can be used for write-on/off effects.
<i>endPoint</i>	float	100	The percentage point that the paint stroke stops drawing. Can be used for write-on/off effects.

Parameters, Edit subtab	Type	Defaults	Function
<i>keyframes</i>	curve	float	This has no purpose except as a placeholder to generate keyframe markers on the Time Bar. It is not modified by the user.
<i>panX, Y</i>	float	0, 0	Provides an offset to each stroke.

The following script parameters do not appear in the interface.

Script Parameters	Type	Defaults	Function
<i>version</i>	string	"v1.02"	The version of the paint node, not the software.
<i>frameState</i>	int	1	Persist = 0, Frame = 1, Interp = 2

### Synopsis

```

image QuickPaint(
image In1,
image In2,
const char * version,
float red,
float green,
float blue,
float alpha,
int frameState,
int snapshotInterval,
float brushSize,
float aspectRatio,
float maxPressure,
int constPressure,
curve float pressureCurve,
curve float moveExpression,
int compressSave,

const char * strokeData0,
int inPoint0,
int outPoint0,
float size0,
float aspect0,
float opacity0,
int spray0,
```

```

float red0,
float green0,
float blue0,
float alpha0,
float xOffset0,
float yOffset0,

const char * strokeDataN,
int inPointN,
int outPointN,
float sizeN,
float aspectN,
float opacityN,
int sprayN,
float redN,
float greenN,
float blueN,
float alphaN,
float xOffsetN,
float yOffsetN,
...
);

```

### **StrokeData Synopsis**

Each stroke has the following data in quotation marks when saved in ASCII format. When compressSave is enabled, this is written in a compressed format and is therefore illegible to us simple folk. However, it is faster and more compact when compressed.

"FORMAT TOOL MASK

NUMDATA;;TIME,TYPE;X;Y;P;X;Y;P;...X;Y;P;;TIME,TYPE;X;Y;P;....X;Y;P;" , followed by inPoint, outPoint, etc., up to yOffset.

StrokeData	Type	Function
TOOL	int	Paint = 1 Smudge = 2 Eraser = 3 Reveal = 4 Clone = 5
MASK	float	The active channels on which the paint is applied.
FORMAT		Reserved for future use.
NUMDATA	int	The number of data pieces per point, placed for future compatibility reasons. This is currently 3—the X,Y position and the pressure of each point.
TYPE	float	The time the data corresponds to.
TYPE	int	The mode the data corresponds to. Persist = 0 Frame = 3 Interp = 4 inPoint (Reserved for future use) = 1 outPoint (Reserved for future use) = 2
X,Y,P	float	X, Y position, pressure.

### Script

```
image QuickPaint(
    In1,
    In2,
    "version",
    red,
    green,
    blue,
    alpha,
    frameState,
    snapshotInterval,
    brushSize,
    aspectRatio,
    maxPressure,
    constPressure,
```

```

pressureCurve,
moveExpression,
compressSave,

"strokeData0",
inPoint0,
outPoint0,
size0,
aspect0,
opacity0,
spray0,
red0,
green0,
blue0,
alpha0,
xOffset,
yOffset,

"strokeDataN",
inPointN,
outPointN,
sizeN,
aspectN,
opacityN,
sprayN,
redN,
greenN,
blueN,
alphaN,
xOffsetN,
yOffsetN
...
);

```

### Command Line

This isn't really a command-line thing...

## Using the RotoShape Node

The *RotoShape* node can create multiple, spline-based shapes that can be used as an alpha channel for an element, or to mask a layer or an effect. For techniques on applying masks, see Chapter 10, “Using Masks.”

The *RotoShape* node is a newer, faster, more flexible, and more able rotoscoping tool that replaces the *QuickShape* node (that turned out to be not so quick).



*RotoShape* has the following advantages over *QuickShape*:

- You can create multiple shapes within the same node.
- You can have a soft-edge falloff on each shape that can be modified independently on each control point.
- You can make one shape cut a hole into another.
- It is much faster to enter keyframes.
- Once you break a tangent, the tangent remains at the angle you specify until you break the tangent again.


### Add Shapes Mode Versus Edit Shapes Mode

When the *RotoShape* node is active, the associated tools appear on the Viewer toolbar.

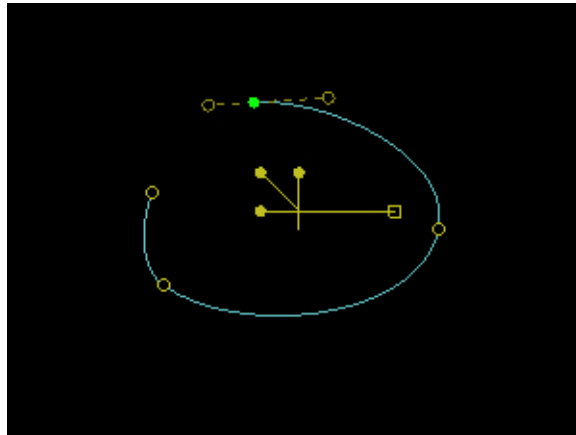


There are two modes in *RotoShape*: The Add Shapes mode  and the Edit Shapes mode . You draw your initial shape and add shapes in Add Shapes mode, and modify or animate the shape in Edit Shapes mode.

#### To add a shape:

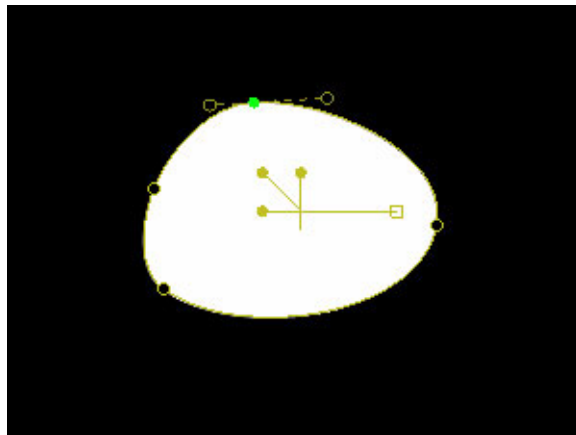
- 1 Click Add Shapes .
- 2 In the Viewer, draw the shape—click blank spots to append new points.
- 3 As you click, drag away from the point to create tangents.

In Add Shapes mode, the shape does not render until you complete and close the shape. In splines mode, the tangent always attempts to close itself with the first point, so there is some curvature around the end points. You can edit previously-placed points and tangents.



- To close the shape, click on the first point drawn.


The shape is filled, and the Edit Shapes mode is automatically activated.





Click a blank spot and drag to select a group of points. Drag to select a new group of points, **Shift**-drag to add to the group of active points, or **Ctrl**-drag to remove from the active group of points.

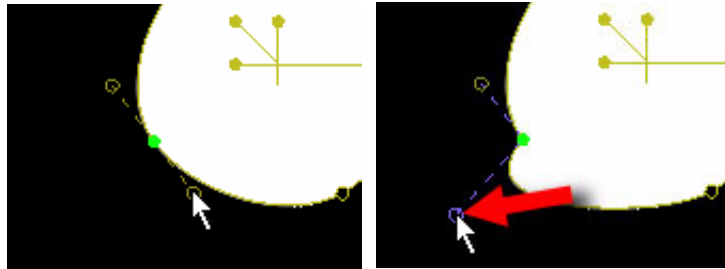
**Note:** To drag-select the control points of a shape when multiple onscreen controls (from different nodes) are displayed in the Viewer, first move the cursor over the shape you want to edit. Next, drag-select the points. This behavior applies to *RotoShape*, *QuickPaint*, and *QuickShape* objects. For example, you can display the onscreen controls for the shapes of two different *RotoShape* nodes by loading the parameters of one node into the Parameters1 tab, and Shift-clicking on the right side of the second node to display the parameters in the Parameters2 tab. Also, if the points you want to drag-select are within a DOD bounding box, move the cursor over the shape inside of the DOD, and then drag-select the points.

## Inserting and Modifying Points and Tangents

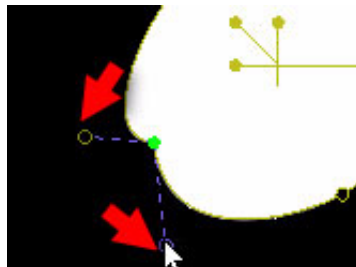
- To insert a new point, press **Shift** and click on a segment.
- To remove a point, select the point and press **Delete**, or click Delete Knot .

### To modify a tangent:



- Select the point and toggle Spline/Line mode   to change the selected points from splined to linear.
- **Ctrl**-drag the end of the tangent to break the tangent.



- Once the tangent is broken, you can release **Ctrl**. Now when you drag, the two tangents are locked relative to each other.

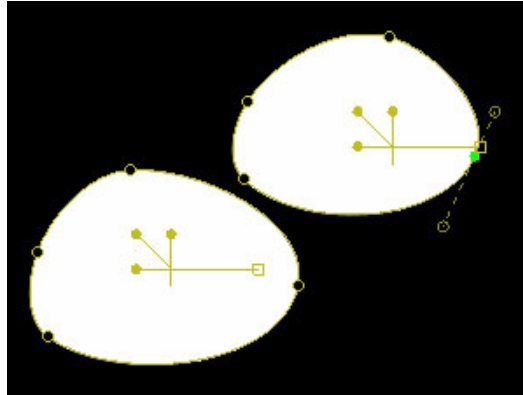


- To break the tangents again, press **Ctrl**.
- To realign the tangents, press **Shift** and click the tangent end. Tangents are independent of each other, but you may modify both tangents simultaneously by pressing **Shift**.

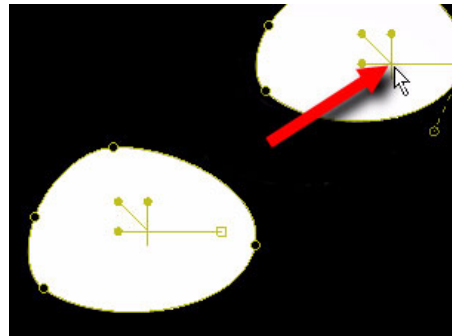
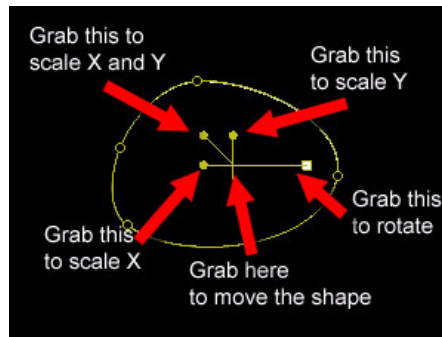
When Lock Tangents is selected  , the tangent angles are locked when the control points are moved, rotated, or scaled. When Unlock Tangents is selected  , the tangent angles are unlocked. Select Unlock Tangents when moving, scaling, or rotating to maintain the shape.

## Creating and Modifying Shapes

To create additional shapes, click Add Shapes  . Click in the Viewer to add another shape.

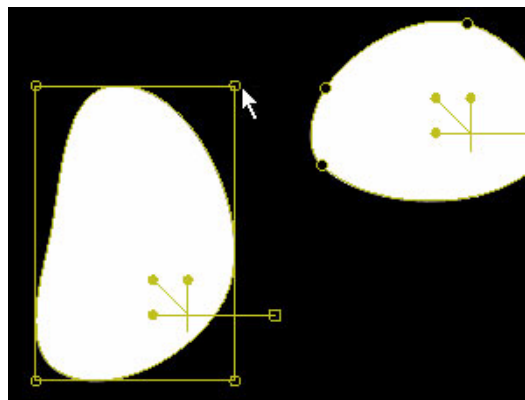
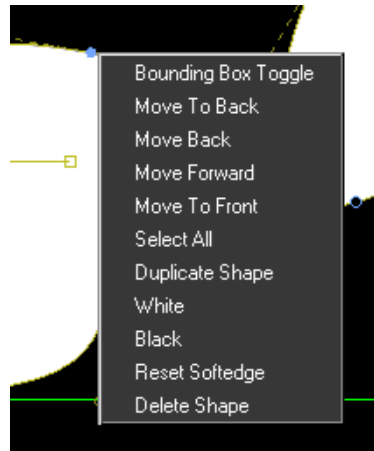


You can modify a shape in several ways. In the transform tool, the small knobs that go up and to the left are the Y and X scale parameters, respectively. The diagonal knob scales both X and Y. The longer knob to the right rotates the shape, and the center of the controls moves the shape.



**Note:** To move the transform tool without modifying the shape, press **Ctrl**.

If you right-click on a point, additional controls become available. The first option, the Bounding Box Toggle, yields a box that can be transformed to move and scale the shape.





**Note:** You can also right-click a point and select Delete Shape to delete the shape.

For information on making a shape black, or changing the order of multiple shapes, see “Right-Click Controls” on page 550.

## Animating Shapes

To animate a roto shape, ensure Autokey is enabled. When a shape is modified, a keyframe is created.

To set keyframes for only the current shape, set Key Current Shape/All Shapes to Key Current Shape . To set keyframes for all shapes, toggle to Key All Shapes .

At frames that contain shape keyframes, the control points appear outlined in blue in the Viewer. At frames that contain no keyframes for the shapes, the control points are outlined in yellow.

### Adding a Keyframe

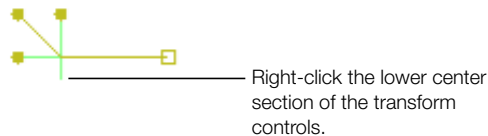
To manually add a keyframe without moving the shape, click Autokey off and on.

### Attaching a Tracker to a RotoShape

You can attach a preexisting track to a shape, or to multiple shapes. Once a tracker is attached to a shape and you are happy with the result, you can bake the track into the shape's panX and panY parameters.

To attach a track to a shape:

- In the Viewer, right-click the lower center portion of the shape's transform controls and select an available track from the "Attach tracker to shape" list.



**Note:** You may have to click more than once for the correct menu to appear.

To remove the tracker from the shape:

- Right-click the lower center portion of the shape's transform controls and select "Remove tracker reference."





To bake the track:

- Right-click the lower center portion of the shape's transform controls and select "Bake tracker into panX/panY."

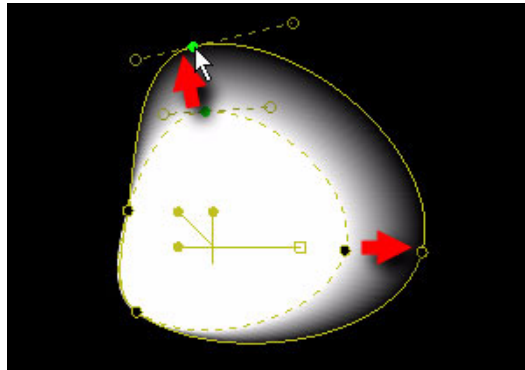
The selected track information is fed into the shape's panX and panY parameters.

## The Point Modes

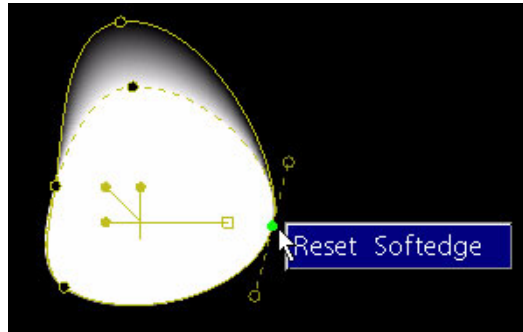
The following table describes the four point modes.

Icon	Name	Hot Key	Notes
	Group Mode	<b>F1</b>	Moves the main shape point and its associated edge point.
	Main Mode	<b>F2</b>	Only allows you to move main shape points. Edge points are not modified.
	Edge Mode	<b>F3</b>	Only moves edges. You can therefore move the edge away from the shape.
	Any Mode	<b>F4</b>	Allows you to pick either type of point.

To create a soft edge, toggle to Edge mode and drag a point.



To reset the soft edge, click Edge Mode, right-click on the point and select Reset Softedge.

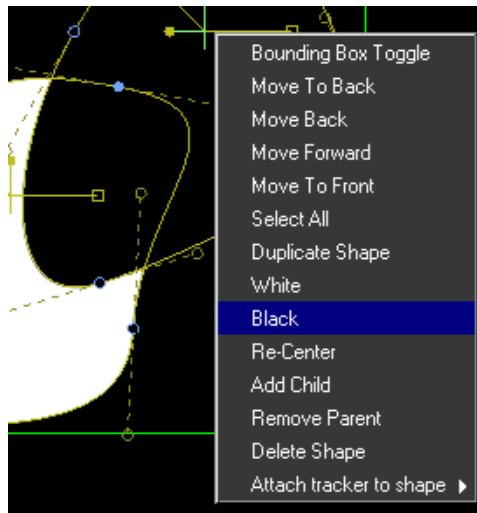


Be careful with the soft edges—if you create a shape with overlapping lines, rendering artifacts may appear. To clean up minor artifacts, apply a slight blur with the *Blur* node.

### Right-Click Controls

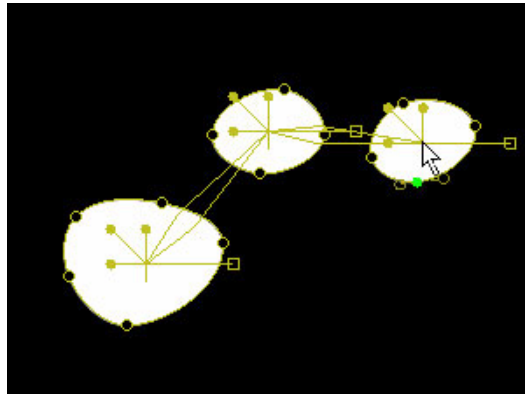
To access several additional functions, right-click a point or transform control.

To specify a shape as black, right-click on a point and select Black. The black shape can then be used to punch a hole in other shapes.

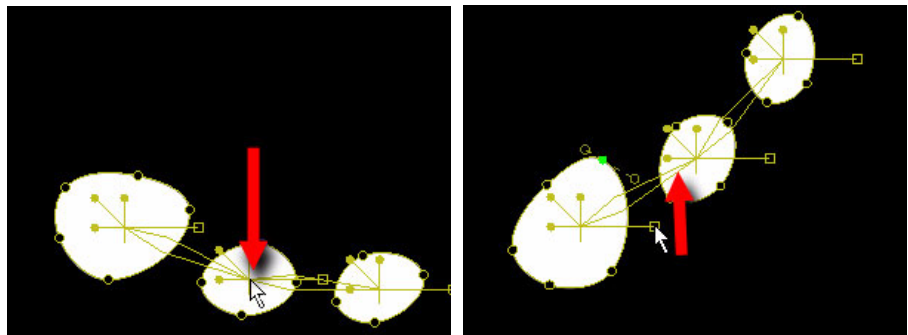


Use the Move functions (Move To Back, Move Back, Move Forward, and Move to Front) on the right-click menu to switch the order of the shapes.

When you right-click on the transform control, you can set up a skeleton relationship between your shapes. Right-click and select Add Child, and click on the transform control of the shape you want as a child of the current shape. To remove the link, right-click and select Remove Parent.



Once a link is established, modifying a shape affects its children.



#### Right-Click Menu on Transform Control





















Item	Notes
Bounding Box Toggle	Toggles the Bounding Box control for a shape on and off.
Move to Back	Moves the shape behind all other shapes.
Move Back	Moves the shape back one position in the shape order.
Move Forward	Moves the shape forward one position in the shape order.
Move to Front	Moves the shape in front of all other shapes.

Item	Notes
Select All	Selects all points on the shape.
Duplicate Shape	Duplicates the current shape.
White	Renders the shape with a white interior.
Black	Renders the shape with a black interior and can therefore be made to punch holes in other shapes.
Re-Center	Recenters the transform tool to be the center of the shape— <b>Ctrl</b> -drag to modify it without moving the shape.
Add Child	Click the transform tool of a second shape to make it a child of the current shape.
Remove Parent	Removes the current shape from the skeleton hierarchy.
Delete Shape	Deletes the current shape.
Attach Tracker To Shape	Calls up a list of precreated trackers that may be used as an offset to the shape.

#### Right-Click Menu on Point

Item	Notes
Bounding Box Toggle	Toggles the Bounding Box control for a shape on and off.
Move to Back	Moves the shape behind all other shapes.
Move Back	Moves the shape back one position in the shape order.
Move Forward	Moves the shape forward one position in the shape order.
Move to Front	Moves the shape in front of all other shapes.
Select All	Selects all points on the shape.
Duplicate Shape	Duplicates the current shape.
White	Renders the shape with a white interior.
Black	Renders the shape with a black interior and can therefore be made to punch holes in other shapes.
Reset Softedge	Repositions the edge knot on top of the main knot.
Delete Shape	Deletes the current shape.

## Viewer Buttons

Item	Notes
 	Add Shapes Mode/Edit Shapes Mode. Click Add Shapes Mode to draw a shape. Click Edit Shapes Mode to edit a shape. Closing a shape automatically activates the Edit Shapes Mode. <i>RotoShape</i> only renders when Edit Shapes Mode is active.
	Quickly toggles the rendering of the shape on and off.
  	Controls the tangent visibility. In Pick mode, only the active point displays a tangent. None hides all tangents, and All displays all tangents.
 	Lock Tangents toggle. When Lock Tangents is selected, the tangent angles are locked when the control points are moved, rotated, or scaled. When Unlock Tangents is selected, the tangent angles are unlocked. Set Unlock Tangents when moving, scaling, or rotating to maintain the shape.
 	Spline/Linear toggle. New points are created as splines or as linear points. Select a point and toggle this to specify the point as a certain type.
 	A really annoying onscreen control to pan the entire collection of shapes. It is off by default.
	Delete current knot/point(s).
   	Determines what points can be selected. Use Group mode to select the main shape and the edge points. Use Main mode to select only the main shape points. Use Edge mode to select only edge points, and use Any mode to pick either main or edge points.
 	Key Current Shape/Key All Shapes toggle. When Autokey is enabled (when animating), select Key Current Shape to keyframe only the current roto shape. Select Key All Shapes to keyframe all shapes.
	If the main onscreen transform tool is turned on, this toggles the visibility of the animation path. It serves no purpose if this tool is turned off.

### RotoShape Function Parameters

The following table lists the *RotoShape* parameters.

Parameters	Type	Defaults	Function
<i>width</i>	int	GetWidth()	Width of the image.
<i>height</i>	int	GetDefaultHeight()	Height of the image.
<i>bytes</i>	int	1	Bit depth, 1, 2, or 4 bytes/channel.
<i>x/yPan</i>	float	0, 0	A global pan applied to the entire shape.
<i>angle</i>	float	0	A global rotation applied to the entire shape—points are properly interpolated according to the rotation.
<i>x/yScale</i>	float	1, 1	A global scale applied to the entire shape.
<i>x/yCenter</i>	float	width/2, height/2	The center of transformation for the angle and x/yScale parameters.
<i>motionBlur</i>	float	0	The quality setting for the motionBlur. A quality of 0 turns off the blur. Unlike the normal blur, you can boost this past 1.
<i>shutterTiming</i>	float	1	The duration of the blur exposure. Default is .5, or one-half of a frame.
<i>shutterOffset</i>	float	0	The starting frame relative to the current frame. Default value is 0, which is the beginning of the current frame.
<i>shapeKeys</i>	float	N/A	Not to be modified by the user. It is just a placeholder.

#### Synopsis

```
image RotoShape(  
    int width,  
    int height,  
    int bytes,  
    float xPan,  
    float yPan,  
    float angle,  
    float aspectRatio,  
    float xScale,
```

```

float yScale,
float xCenter,
float yCenter,
float motionBlur,
float shutterTiming,
float shutterOffset,
[cv data]
);

```

### Script

```

image = RotoShape(
    width,
    height,
    bytes,
    xPan,
    yPan,
    angle,
    aspectRatio,
    xScale,
    yScale,
    xCenter,
    yCenter,
    motionBlur,
    shutterTiming,
    shutterOffset,
    [cv data]
);

```

### Command Line

This function is not really command-line compatible.

## Other Image Functions

The following sections describe other Image functions available in Shake.

### Checker

The *Checker* function generates a checkerboard within the width and height of the image. It is handy to test Warps, or to split a screen in half. To make a specific number of tiles per row or column (for example, 4 x 4), divide the width and the height by the number—height/4 yields 4 rows.

Parameters	Type	Defaults	Function
<i>width</i>	int	defaultHeight	Width of the image.
<i>height</i>	int	defaultWidth	Height of the image.
<i>bytes</i>	int	1	Bit depth, 1, 2, or 4 bytes/channel.
<i>xSize</i>	int	32	X-resolution of a checker.
<i>ySize</i>	int	xSize	Y-resolution of a checker.

### Synopsis

```
image Checker(  
    int width,  
    int height,  
    int bytes,  
    int xSize,  
    int ySize  
);
```

### Script

```
image = Checker( width, height, bytes,  
                xSize, ySize  
);
```

### Command Line

*sbake -checker width height bytes xSize ySize*

### Examples

*sbake -checker*

*sbake -check 300 200 1 150 100*

*sbake -check 300 200 1 150 "Linear(0,1@1,200@20)" -t 1-20*

### Color

The *Color* function generates a solid field of color within the width and height of the image.

Parameters	Type	Defaults	Function
<i>width</i>	int	defaultWidth	Width of the image.
<i>height</i>	int	defaultHeight	Height of the image.
<i>bytes</i>	int	1	Bit depth, 1, 2, or 4 bytes/channel.
<i>red, green, blue, alpha, depth</i>	float	0, 0, 0, 1, 0	Value of red, green, blue, alpha, and Z channels.

### Synopsis

```
image Color(  
    int width,  
    int height,  
    int bytes,  
    float red,  
    float green,  
    float blue,  
    float alpha,  
    float depth  
);
```

### Script

```
image = Color( width, height, bytes,  
    red, green, blue, alpha, depth  
);
```

### Command Line

*shake -color width height bytes etc...*

### Examples

*shake -color*

*shake -color 320 240*

*shake -color 320 240 1 0 0 1*

### Black/-black

The *Black/-black* command-line function is the *Color* command with preset values of 0, 0, 0, 0, 0. It is handy as a temporary background, or to set a resolution. Composite a node on top of the *Black* node, and set your compNode (in the *Over* node) to 1 to take the resolution of the *Black* node.

Parameters	Type	Defaults	Function
<i>width</i>	int	defaultHeight	Width of the image.
<i>height</i>	int	defaultWidth	Height of the image.
<i>bytes</i>	int	1	Bit depth, 1, 2, or 4 bytes/channel.

### Synopsis

```
image Black(  
    int width,  
    int height,  
    int bytes,  
);
```

### Script

```
image = Black( width, height, bytes);
```

### Command Line

*shake -black width height bytes*

### Examples

*shake -black 720 486*

*shake -black 300 200 2*

### ColorWheel

The *ColorWheel* function generates a primitive color wheel. It can also be used as a tool to determine what HSV/HLS commands, such as *AdjustHSV* and *ChromaKey*, are doing.

Parameters	Type	Defaults	Function
<i>width</i>	int	defaultWidth	Width of the image.
<i>height</i>	int	defaultHeight	Height of the image.
<i>bytes</i>	int	1	Bit depth, 1, 2, or 4 bytes/channel.
<i>satCenter</i>	float	0	Saturation of center area.

Parameters	Type	Defaults	Function
<i>satEdge</i>	1	float	Saturation of edge area.
<i>valCenter</i>	float	1	Value of center area.
<i>valEdge</i>	float	1	Value of edge area.

### Synopsis

```
image ColorWheel(
    int width,
    int height,
    int bytes,
    float satCenter,
    float satEdge,
    float valCenter,
    float valEdge
);
```

### Script

```
image = ColorWheel( width, height, bytes,
    satCenter, satEdge,
    valCenter, valEdge
);
```

### Command Line

*shake -colorwheel width height bytes etc...*

### Examples

*shake -colorwheel*

*shake -colorw 500 500*

*shake -colorw 500 500 1 1 0*

### FileIn/SFileIn

*FIBlend/FINearest/FIPullUpDown/FISpeed/IRetime*

The *FileIn* function reads in images from disk. The *SFileIn* is an advanced version of *FileIn* with more functionality, and can be additionally modified by the invisible functions *FIBlend*, *FINearest*, *FIPullUpDown*, *FISpeed*, and *IRetime*. These are “invisible” because they do not appear in the Node View, but are saved into the script to modify the *FileIn* and *SFileIn* functions.

For more information, see “The FileIn/SFileIn Function” on page 79.

### FileOut

The *FileOut* function allows you to write images to a disk or another external device. It recognizes local, absolute, or URL paths. For a URL address, place a // in front of the path. To write to another computer, write //Biggo/Drive/Directory/etc. For the command line, each *FileOut* is explicitly accompanied by a -fo (or -fileout). You can add multiple *FileOuts* along your command string to output different steps of the command.

For more on the syntax of the filename, see “The FileIn/SFileIn Function” on page 79.

For more information on file formats, see “File Formats” on page 201.

For more information on I/O, see “About Image Input and Output” on page 75.

For more information on time and clips, see “About Time” on page 84.

Parameters	Type	Defaults	Function
<i>imageName</i>	string		The path and filename of the output image.
<i>fileFormat</i>	string	"auto"	If no extension is given, the output format is .iff. To override this behavior, explicitly set the output format.
<i>codec (QuickTime only)</i>	string		A list of all available compressors. The default argument is "Default."
<i>compressionQuality (QuickTime only)</i>	float		The quality of the compression algorithm. A value of 1 is maximum size, maximum quality. 0 is minimum size, minimum quality.
<i>framesPerSecond (QuickTime only)</i>	int		The frames per second for the playback of the QuickTime compression.

### Synopsis

```
image FileOut ( image, const char * imageName);
```

### Script

```
image = FileOut( image, "imageName");
```

### Command Line

*shake input\_image imageName*

or

*shake input\_image -fileout imageName*

### Grad

The *Grad* function generates a gradation between four corners of different colors. The count order of the corners is: Corner 1 in the lower-left corner, corner 2 in the lower-right corner, and so on. For a simple ramp, use *Ramp*. For a radial gradation, use *RGrad*.

Parameters	Type	Defaults	Function
<i>width, height</i>	int	GetWidth(), GetHeight()	Width of the image.
<i>bytes</i>	int	1	Bit depth, 1, 2, or 4 bytes/channel.
<i>xMid, yMid</i>	float	.5, .5	The midpoints of the gradation.
<i>red, green, blue, alpha, depth</i>	float	1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0	Value of red, green, blue, alpha, and Z channels at each of the four corners. LL = Lower-left corner LR = Lower-right corner UR = Upper-right corner UL = Upper-left corner

### Synopsis

```
image Grad(  
    int width,  
    int height,  
    int bytes,  
    float xMid,  
    float yMid,  
    float rLL,  
    float gLL,  
    float bLL,  
    float aLL,  
    float dLL,  
    float rLR,  
    float gLR,  
    float bLR,  
    float aLR,  
    float dLR,  
    float rUR,  
    float gUR,  
    float bUR,  
    float aUR,  
    float dUR,
```

```
float rUL,  
float gUL,  
float bUL,  
float aUL,  
float dUL  
);
```

### Script

```
image = Grad( width, height, bytes, xMid, yMid, etc...);
```

### Command Line

*shake -grad width height bytes etc...*

## Using the QuickShape Node

The *QuickShape* function is an image generator to be used for animated garbage mattes. It is ideal for plugging into the Mask input of a node, or is used in conjunction with functions such as *Inside*, *Outside*, or *KeyMix*.


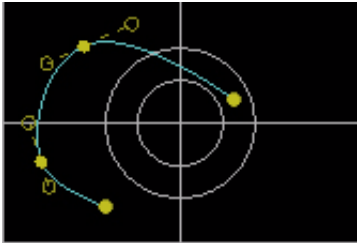

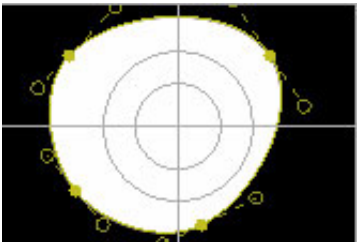
**Note:** The *QuickShape* node is an older node to create rotoshapes. The more flexible (and faster) *RotoShape* node is recommended. This node is maintained for compatibility purposes. The one advantage *QuickShape* has over *RotoShape* is its ability to propagate keyframe changes to other keyframes before or after the current frame.

Since these nodes create images like any node, you can modify the images with standard tools such as *Blur* or *DilateErode*.

You can enable motion blur for an animated *QuickShape*. Unfortunately, the shape does not use Shake's normal high-quality motion blur. It instead draws and renders several versions of the entire shape, so temporal aliasing occurs with extreme motion.

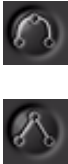
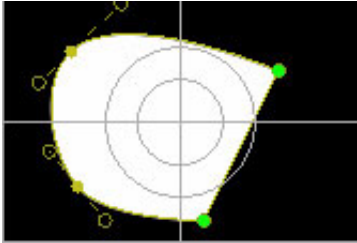

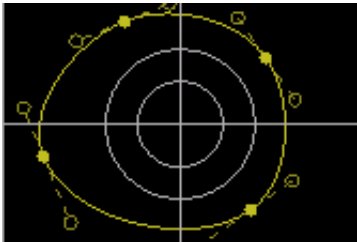
## Creating QuickShapes



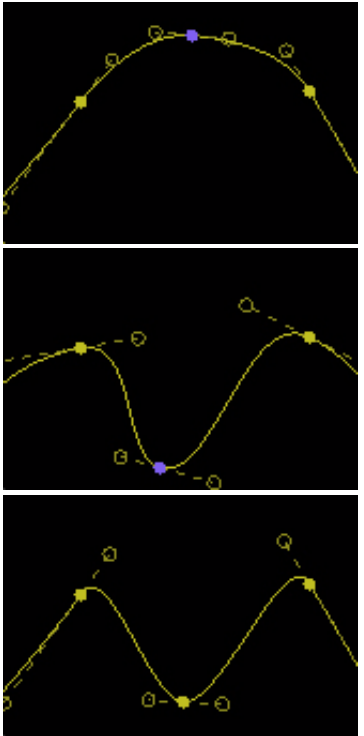



Naturally, you create a *QuickShape* node. When Display Onscreen Controls  is enabled for the Viewer, you can immediately add points to the shape.

Mode	Usage	Example
	Start in Build mode. In Build mode, each time you click on a blank spot, you append a new point between the last point and the first point. Click in the points, or, if you hold the mouse down, you can drag the new point around. You can also go back and change any key or tangent, or insert a point by clicking on a segment.	
	Once you are finished with the rough shape, switch to Edit mode (click the build/edit mode button). When you click on a blank spot, you do not append a new point; instead, you can drag-select several points to move them as a group. This also fills in the shape.	

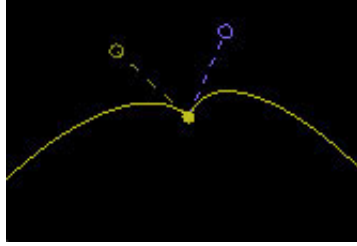
### Modifying QuickShapes

To select multiple points in Edit mode, drag-select around the points. The selected points can then be modified as a group.

Button	Usage	Example
	The Spline/Line mode buttons change the selected points from Linear to Smooth. Select the points and toggle the button to the setting you want. In this example, the two right points have been made Linear. When Linear is selected, no tangents are available.	
	Click the Fill button on the Viewer toolbar to turn the shape fill on and off. In Build mode, the shape is not filled. The filled shape is not just a display feature—it affects the composite.	

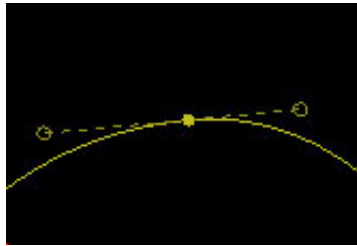
Button	Usage	Example
 	<p>The Lock Tangents button locks or unlocks the tangents of adjacent points when moving any point. In the first example, the tangents are unlocked. Therefore, the middle blue point is moved down. Shake tries to keep the tangents of the adjacent points smooth, and therefore moves the tangents.</p> <p>If Lock Tangents is enabled, the adjacent tangents stay locked in place. This provides accuracy for adjacent segments, but creates a more irregular shape.</p>	
	<p>The Show/Hide Tangents button displays or hides the tangents on the shape.</p>	
	<p>The Enable/Disable Transform Control button turns on and off the display of the transform tool for the <i>QuickShape</i>.</p>	
	<p>The Delete button deletes all selected points.</p>	

To break a tangent, **Ctrl**-click the tangent.

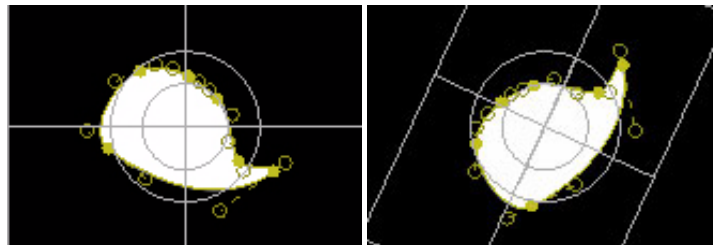


**Note:** No tangents are available when the points are set to Linear mode.

To reconnect the tangents, **Shift**-click the broken tangent.


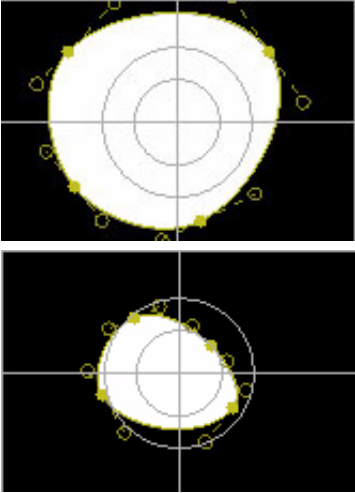




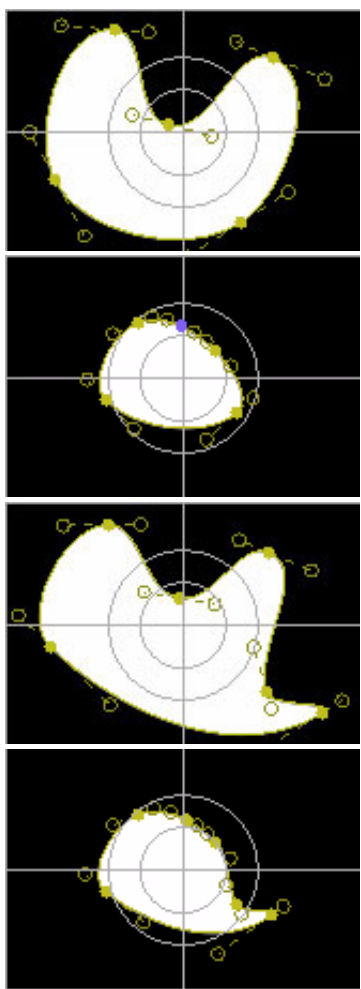

Use the transform tool to modify the entire shape. The transform tool includes pan, rotation, and scaling tools for the shape. Since this is a transformation, the points rotate properly in an angular fashion when interpolating in an animation, rather than just sliding linearly to the next position. The controls appear at the same resolution of the *QuickShape* node, so if you are dealing with 2K plates, you may want to enter a larger resolution for the *QuickShape*. Similarly, if you find the transform tool annoying, enter a resolution of 10 x 10. Neither of these techniques changes rendering speed due to the Infinite Workspace.



Animating QuickShapes

The following table discusses the *QuickShape* animation tools.

Button	Usage	Example
	To easily animate the <i>QuickShape</i> , enable Autokey and move the points. To enter a new keyframe, move to a new time, and change the shape's (or the control points of the shape) position. In this example, the shape is smaller on the second keyframe. As you drag the time slider, the shape interpolates between the two keyframes.	
	Delete a keyframe (if present) at the current frame.	

Button	Usage	Example
	<p>Here, a point is inserted and moved toward the center at the first keyframe. At the second keyframe's position, the shape is still round because Shake has maintained the smooth quality of the segment. If you instead turn on the Propagate buttons when you modify a point, the second keyframe's point position is also modified. For example, go back to keyframe 1, enable Propagate Forward, and insert a new point, dragging it outward. Jump to the second keyframe, and the new point is positioned in a relatively similar fashion in the second keyframe.</p> <p>If you have several keys, Propagate Forward or Backward can slow down your interactivity.</p>	
<p>To quickly scrub through an animation, toggle to Release or Manual Update mode (in the upper-right bar of the interface), and then move the time slider. The shapes draw in real time, but are not rasterized.</p>		

## QuickShape Function Parameters

The following table lists the *QuickShape* parameters.

Parameters	Type	Defaults	Function
<i>width</i>	int	GetDefaultWidth()	Width of the image.
<i>height</i>	int	GetDefaultHeight()	Height of the image.
<i>bytes</i>	int	1	Bit depth, 1, 2, or 4 bytes/channel.
<i>x/yPan</i>	float	0, 0	A global pan applied to the entire shape.
<i>angle</i>	float	0	A global rotation applied to the entire shape—points are properly interpolated according to the rotation.
<i>x/yScale</i>	float	1, 1	A global scale applied to the entire shape.
<i>x/yCenter</i>	float	width/2, height/2	The center of transformation for the angle and x/yScale parameters.
<i>motionBlur</i>	float	0	The quality setting for the motionBlur. A quality of 0 turns off the blur. Unlike the normal blur, you can boost this past 1.
<i>shutterTiming</i>	float	1	The duration of the blur exposure. Default is .5, or one-half of a frame.
<i>shutterOffset</i>	float	0	The starting frame relative to the current frame. Default value is 0, which is the beginning of the current frame.

### Synopsis

```
image QuickShape(  
    int width,  
    int height,  
    int bytes,  
    float xPan,  
    float yPan,  
    float angle,  
    float aspectRatio,  
    float xScale,  
    float yScale,  
    float xCenter,  
    float yCenter,
```

```

float motionBlur,
float shutterTiming,
float shutterOffset,
[ cv data]
);

```

### Script

```

image = QuickShape(
    width,
    height,
    bytes,
    xPan,
    yPan,
    angle,
    aspectRatio,
    xScale,
    yScale,
    xCenter,
    yCenter,
    motionBlur,
    shutterTiming,
    shutterOffset,
    [ cv data]
);

```

### Command Line

This isn't really a command-line thing.

### See Also

“*Rand*” on page 570, “*Grad*” on page 561, “*Ramp*” on page 571, “*Black/-black*” on page 558, “*Checker*” on page 556, “*ColorWheel*” on page 558, “*RGrad*” on page 573

### Rand

The *Rand* function generates a random field of noise. The field does not resample if you change the resolution or density—you can animate the density without pixels randomly changing. The seed is set to time by default so that it changes every frame, but you can of course lock this to one value.

Parameters	Type	Defaults	Function
<i>width</i>	int	GetWidth()	Width of the image.
<i>height</i>	int	GetHeight()	Height of the image.

Parameters	Type	Defaults	Function
<i>bytes</i>	int	1	Bit depth, 1, 2, or 4 bytes/channel.
<i>density</i>	float	1	The density from 0 to 1 of the pixels. A lower density results in fewer random pixels.
<i>seed</i>	float	time	Shake generates a random pattern. The trick for compositing is how to recreate a random pattern a second time. By locking in an initial value—the seed—you can reconstruct the same exact random pattern so that your image does not change every time you rerender. You can set the value to keep the same pattern, or just enter the keyword "time" so that every frame is different.

### Synopsis

```
image Rand(
    int width,
    int height,
    int bytes,
    float density,
    float seed
);
```

### Script

```
image = Rand( width, height, bytes, density, seed);
```

### Command Line

```
shake -rand width height bytes density seed
```

### Ramp

The *Ramp* function generates a linear gradation between two edges. You can set the direction of the ramp to horizontal or vertical. The *Ramp* is, among other things, a useful tool for analyzing color correction nodes. Attach a horizontal *Ramp* to the color correction node, and attach the node to a *PlotScanline* node. For a radial gradation, use *RGrad*. For a four-corner ramp, use *Grad*.

Parameters	Type	Defaults	Function
<i>width</i>	int	GetWidth()	Width of the image.
<i>height</i>	int	GetDefaultHeight()	Height of the image.

Parameters	Type	Defaults	Function
<i>bytes</i>	int	1	Bit depth, 1, 2, or 4 bytes/channel.
<i>orientation</i>	int	0	Horizontal or vertical orientation.
<i>center</i>	float	.5	The midpoint of the gradation. 0 = vertical orientation 1 = horizontal orientation
<i>red, green, blue, alpha, depth</i>	float	0, 0, 0, 1, 0, 1, 1, 1, 1, 0	Value of red, green, blue, alpha, and Z channels.

**Synopsis**

```
image Ramp(  
    int width,  
    int height,  
    int bytes,  
    int orientation,  
    float center,  
    float red1,  
    float green1,  
    float blue1,  
    float alpha1,  
    float depth1,  
    float red2,  
    float green2,  
    float blue2,  
    float alpha2,  
    float depth2  
);
```

**Script**

```
image = Ramp( width, height, bytes, orientation, etc...);
```

**Command Line**

```
shake -ramp width height bytes etc...
```

## RGrad

The *RGrad* function generates a radial gradation. You can also control the falloff to make circles. For a simple ramp, use *Ramp*. For a four-corner gradation, use *Grad*.

Parameters	Type	Defaults	Function
<i>width</i>	int	GetDefaultWidth()	Width of the image.
<i>height</i>	int	GetDefaultHeight()	Height of the image.
<i>bytes</i>	int	1	Bit depth, 1, 2, or 4 bytes/channel.
<i>x,yCenter</i>	float	width/2, height/2	The midpoints of the circle.
<i>aspectRatio</i>	float	defaultAspectRatio	The aspectRatio of the pixels.
<i>radius</i>	float	min(width,height)/4	The non-blurred radius of the center.
<i>falloffRadius</i>	float	min(width,height)/4	The blurred edge radius (the total width of the circle is radius + falloffRadius).
<i>falloff</i>	float	.5	The midpoint, in terms of percentage, of the falloff. 0 or 1 equals a hard-edge circle; .5 is a smooth ramp.
<i>red, green, blue, alpha, depthCenter</i>	float	1, 1, 1, 1, 0	The values of the center area.
<i>red, green, blue, alpha, depthEdge</i>	float	0, 0, 0, 0, 0	The values of the image edge.

### Synopsis

```
image RGrad(  
    int width,  
    int height,  
    int bytes,  
    float xCenter,  
    float yCenter,  
    float aspectRatio,  
    float radius,  
    float falloffRadius,  
    float falloff,  
    float redCenter,  
    float greenCenter,  
    float blueCenter,
```

```

float alphaCenter,
float depthCenter,
float redEdge,
float greenEdge,
float blueEdge,
float alphaEdge,
float depthEdge
);

```

### Script

```
image = RGrad( width, height, bytes, etc...);
```

### Command Line

*shake -rgrad width height bytes etc...*

### Text

The *Text* function calls on software that is basically identical to GL Render for character generation. You can type in your text, or use variables and links to insert characters, making it an ideal tool for generating slates.

You can use any TrueType (.tff) and Type 1 (PostScript, .pfa for ASCII and .pfb for binary) font. If an Adobe Font Metrics (.afm) file is present for the font (for example, you have MyFont.pfa and MyFont.afm), it is supported and provides kerning for the font. Shake first looks for fonts in its distribution directory, under fonts. You can also place them in a direct path by setting the environment variable *NR\_FONT\_PATH*. Finally, Shake also detects fonts placed in the standard directories for your OS:

*Macintosh OS X: All "Installed" directories.*

*Linux: /usr/lib/DPS/AFM*

*IRIX: /usr/lib/X11/fonts/Type1*

*Text* uses the Shake implementation of the GL Render. It allows you to not only manipulate the characters in 3D space (including X, Y, and Z position, rotation, and scaling), but also a camera field of view. Because of this, it is better to animate text within the *Text* (or *AddText*) function to ensure crisp, clean edges.

To select a font in the interface:

- In the *Text* node parameters, click (left mouse) the font pop-up menu.
- To preview a font in the Viewer, right-click the font name in the list.
- To select a font, click (left mouse) the font name in the list.

**Note:** For long font lists, click and drag the scroll tool to select a font from the list.

You can also use the following shortcuts at any time without any special formatting:

Text Shortcut	Writes
{parameter}	Prints either the local or global parameter value, for example: <i>Script = {scriptName}</i> writes <i>Script = My_Script.shk</i>
{nodeName.parameter}	Prints the parameter's value from a selected node, for example: <i>Red Val = {Multi1.red}</i> could write <i>Red Val = .6</i>
\n	New line. Example: <i>Hello\nWorld</i> returns Hello World
%f	Unpadded frame number.
%F	Four-digit padded frame number.
%t	Short non-dropframe timecode: no 00: (if not needed).
%T	Long non-dropframe timecode: hr:mn:sc:fr.
%tD	Short dropframe timecode: no 00: (if not needed).
%TD	Long dropframe timecode: hr:mn:sc:fr.
%H	Host name.
%U	Username.
%c,%C	Locale's center.
%d	Locale's day 01-31.
%D	Locale's abbreviated day name: Wed.
%E	Locale's full day name: Wednesday.
%m	Locale's month: 01-12.

Text Shortcut	Writes
%M	Locale's abbreviated month name: Nov.
%N	Locale's full month name: November.
%x,%X	Full date representation: mm/dd/yy.
%y	Year without century: 00-99.
%Y	Year as ccyy.

**Examples 1:**

Text String	Writes
My name is Dufus	My name is Dufus
My name is %U	If login = Dufus: My name is Dufus
My name is %U.\nToday is %M. %d	My name is Dufus. Today is Nov. 12
Mult red = {Mult1.red}	Assuming the node <i>Mult1</i> exists, and the red value is .46: Mult red = .46

To get special characters, such as umlauts, copyright symbols, etc., use octal and hexadecimal codes preceded by a \. These codes can be found in Unix with the man page for printf in its special characters section. For example, this was provided by a fine friend, Thomas Kumlhehn, at Double Negative. Copy and paste it into the *Text* node:

```
Auml=\xC4, Ouml=\xD6, Uuml=\xDC, \n auml=\xE4, ouml=\xF6, uuml=\xFC \n
Szett=\xDF \ntm=\x99, Dot=\x95, (R)=\xAE, (C)=\xA9
```

A good reference Web site for characters can be found at [www.asciitable.com](http://www.asciitable.com). (A case of Schlitz goes out to Christer Dahl for this tip.)

To use expressions, preface the text with a : All printed commands are in quotes. For example, if you want to print “Hell” from frames 1 to 10 and “o World” from frames 11 onwards, enter:

```
:time<11?"Hell":"o World"
```

Finally, you can also use full C formatting for your strings. This is initialized with a `:` at the start of the text string as well:

```
: sprintf(
    "Red = %4.2f at Frame %03f",
    Grad1.red1, time
)
```

To append strings from another parameter, use something like:

```
in Text1.text:    Hello
in Text2.text:    : Text1.text + " World"
```

Parameters	Type	Defaults	Function
<i>width, height</i>	int	GetWidth(), GetHeight()	The width and height of the image.
<i>bytes</i>	int	1	The bit-depth of the image. 1 = 8 bits 2 = 16 bits 4 = 32 bits, or float
<i>text</i>	string	text	The characters to be printed.
<i>font</i>	string		The font to be used.
<i>x, yFontScale</i>	float	100, xFontScale	The X and Y scaling of the font.
<i>leading</i>	float	1	The amount of spacing between each line.
<i>x, y, zPos</i>	float	width/2, height/2, 0	The 3D position of the characters.
<i>xAlign</i>	int	1	The horizontal alignment of the fonts. 1 = left 2 = center 3 = right alignment
<i>yAlign</i>	int	1	The vertical alignment of the fonts. 1 = up 2 = center 3 = down
<i>red, green, blue, alpha</i>	float	1, 1, 1, 1	The color of the text.
<i>x, y, zAngle</i>	float	0, 0, 0	The 3D rotation angles of the text.

Parameters	Type	Defaults	Function
<i>fieldOfView</i>	float	45	The aperture angle in degrees of the virtual camera.
<i>kerning</i>	float	0	The spacing between each letter. You can also have negative values.
fontQuality	float	1	The polygonalization factor of the font splines. This is conservatively set to a high value. For flat artwork, you can probably get away with a value of 0. When you have extreme perspective, you should keep it set to a high value.

### Synopsis

```
image Text(
    image,
    int width,
    int height,
    int bytes,
    const char * text,
    const char * font,
    float xFontScale,
    float yFontScale,
    float leading,
    float xPos,
    float yPos,
    float zPos,
    int xAlign,
    int yAlign,
    float red,
    float green,
    float blue,
    float alpha,
    float xAngle,
    float yAngle,
    float zAngle,
    float fieldOfView,
    float kerning,
    float fontQuality
);
```

## Script

```
image Text(  
    image,  
    width,  
    height,  
    bytes,  
    "text",  
    "font",  
    xFontScale,  
    yFontScale,  
    leading,  
    xPos,  
    yPos,  
    zPos,  
    xAlign,  
    yAlign,  
    red,  
    green,  
    blue,  
    alpha,  
    xAngle,  
    yAngle,  
    zAngle,  
    fieldOfView,  
    kerning,  
    fontQuality  
);
```

## Command Line

*-text text font xScale yScale etc...*

## Example 2

*slatemacro*

This macro is to set a *FileOut*, and inserts a slate at the frame before the first frame. For example, if your frame range is 1-100, frame 0 gets a printed slate. Load the script in the interface to see what it does, and move the time slider from frame 1 to 0. There are three text generators. One (*AText1*) provides the headers (for example, “Shot:” “Show:”, “Frame Range”, etc.). The second generator (*AddText2*) gets the information either from the script, or as inputs to the macro that are provided by the user. A last text generator (*AddText1*) prints the frame number in the corner if markFrame is set to 1. To open the macro in the interface, position the cursor over mySlate and press **Enter**. Then view the parameters for each subnode. Or, of course, open the script in a text editor.

For information on the *AddText* function, see “AddText” on page 164.

**Tile**

The *Tile* function is located in the Other Tool Tab. *Tile* does not generate an image, but makes small tiles of an image within that image. The more tiles created, the slower the processing (for example, more than 40 tiles).

Parameters	Type	Defaults	Function
<i>nX,YTile</i>	int	1, 1	Number of copies placed on each axis.

**Synopsis**

```
image Tile( image, int nXTile, int nYTile );
```

**Script**

```
image = Tile( image, nXTile, nYTile);
```

**Command Line**

```
sbake -tile nXTile nYTile
```

# Filters

## Chapter Summary

- About Filters
- Masking Filters
- The Filter Functions

## About Filters

While color corrections change the value of an individual pixel according to a mathematical equation (for example,  $\times 2$ ,  $-0.5$ , etc.), filters calculate the new value of a pixel by examining its neighbors, and passing it through what is called a Spatial Filter. Classic filter functions are blurs, image sharpen, emboss, and median filters. You can also create your own unique filters, of any resolution, with the *Convolve* function. Spatial filters are also applied when a geometrically transformed image is resampled, for example, following a *Scale* or *Rotate* node.

## Masking Filters

To have an image control the amount of filtering, it is recommended to use the special image-based filters such as *IBlur*, *ISharp*, *IRBlur* instead of simply masking the effect. Although the node is slower, the effect is higher quality.

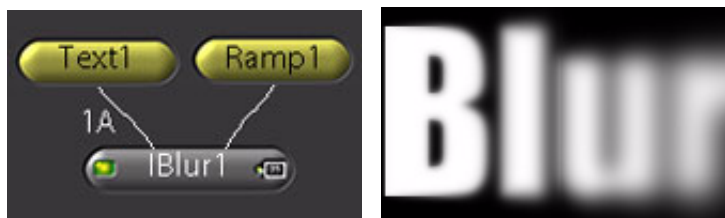
In the following example, some text in a *Text* node is blurred with a *Ramp* node that controls the amount of blur.



Normally, a *Blur* node is attached, and then a *Ramp* node is applied as the mask. The result is not very compelling, since you merely end up with a blend between sharp and blurred elements. Note the *Ramp* has an alpha value of 1 for both ends, so you should change the alpha1 value to 0.



Instead, use the dedicated *IBlur* node, with the *Ramp* node as the second input image, rather than a mask input.



### Filter Characteristics

Shake is engineered to use the highest-quality filtering process for transformations, blurs, and convolve effects. This is especially noticeable on animated effects, whether it is an animated transformation, or simply if the parameters of a blur are animated.

It is very difficult to decide on a single filter to use on both up and down resizing. For this reason, the “default” filter was built as it is: The default filter uses *mitchell* going up and *sinc* going down (the switch is automatic). The choice was made after projecting several versions of the same shots, processed with different filters, to a small panel of film professionals. Other filters, such as *box* (the closest thing to what users may know as “bilinear”), may give subjectively superior results in some cases (in particular on static imagery), but poor handling of high frequencies quickly becomes apparent.

Often, you can set different filters for X and Y, so you may increase the X resolution, but decrease the Y resolution. You usually have the option to set both directions, keeping in mind that the default automatically uses *mitchell* to zoom up and *sinc* to zoom down. If there are not two options, as in the *Resize* node, simply place two *Resize* nodes, one to zoom the X and the second to zoom the Y. As a result, your processing speed hit is minor.

The subpixel control affects the way fractional resize is performed. If your resize value is not an integer (for example, 512 zooming to 1024 is an integer by a factor of 2; zooming 512 to 1023 is not an integer as it is a factor of 1.998), you have subpixel sampling. For *Resize*, if the new width or height is not an integer (either because it was set that way, or because of a proxy scale), you have a choice to snap to the closest integer (subpixel off) or generate transparent pixels for the last row and column (subpixel on).

Filter	Description
<i>box</i>	Relatively inexpensive and gives a "boxy" look. Default size is 1 x 1.
<i>default</i>	By default, <i>mitchell</i> is used to resize up, and <i>sinc</i> to resize down.
<i>dirac</i>	<i>Dirac</i> and <i>impulse</i> are the same. Default size is 0 x 0.
<i>gauss</i>	<i>Gaussian</i> lacks in sharpness, but is good with ringing and aliasing. Default size is 5 x 5.
<i>impulse</i>	Fast but lower quality. Default size is 0 x 0.
<i>lanczos</i>	Similar to the <i>sinc</i> filter, but with less sharpness and ringing. Default size is 6 x 6.
<i>mitchell</i>	This is the default filter when scaling up. A good balance between sharpness and ringing, and so a good choice for scaling up. Default size is 4 x 4. This is also known as a high-quality Catmull-Rom filter.
<i>quad</i>	Like <i>triangle</i> , but more blur with fewer artifacts. Default size is 3 x 3.
<i>sinc</i>	This is the default filter when scaling down. Keeps small details when scaling down with good aliasing. Ringing problems make it a questionable choice for scaling up. Default size is 4 x 4. It can also deliver negative values, which can be interesting when working in float/channel bit depth.
<i>triangle</i>	Not highest quality, but fine for displaying a scaled image on your screen. Default size is 2 x 2.

## The Filter Functions

The following sections describe each filter function, and include parameters, defaults, and examples.

### ApplyFilter

Like *Blur*, the *ApplyFilter* function applies a filter—except you can choose the filter used in X and Y separately. You can then scale the default base range (in pixels) of the predefined filters. For instance, if the default number of pixels sampled on either side of the base pixel is 3 pixels, an xScale of 2 increases that range to 6 pixels.

You can change the filter type in the much faster *Blur* function. The *ApplyFilter* function exists only to allow absolute compatibility with images generated by other software packages. Note that *dirac* and *impulse* filters have no effect with *ApplyFilter*.

Parameters	Type	Defaults	Function
<i>x, yFilter</i>	string	"gauss", "gauss"	See "Filter Characteristics" on page 582.
<i>x, yPixels</i>	float	1, 1	The amount of filtering as described in Pixels.
<i>spread</i>	int	0	Tells Shake whether or not to consider outside of the frame.  0 = Compute with the frame. 1 = Compute outside of the frame.  Because of the Infinite Workspace, it is sometimes handy to compute outside of the frame as well, for example, if the <i>Blur</i> is placed after a <i>Scale</i> command. Note that if nothing is outside of the frame (black), you see a black edge.

**Synopsis**

```
image ApplyFilter( image,
    const char * xFilter,
    const char * yFilter
    float xPixels,
    float yPixels,
    int spread,
);
```

**Script**

```
image = ApplyFilter( image,
    "xFilter",
    "yFilter",
    xPixels,
    yPixels,
    spread
);
```

**Command Line**

```
shake -applyfilter xFilter yFilter etc...
```

**Example**

```
shake lisa.iff -apply gauss box 20 100
```

## Blur

The *Blur* function blurs the image. This is a Gaussian blur (by default), but you can change the filter for both X and Y. Use this function instead of the similar, but slower, *ApplyFilter* function.

Shake's *Blur* is about the only place you have to be concerned with the Infinite Workspace, since you can choose to blur only pixels inside or outside of the image. If your final image is clipped and you aren't sure why (for example, there are no *Crop* commands), go back and check the spread parameters of the *Blur*. Toggle the spread parameters to Outside Frame (1), and the clipping should disappear.

For more information on the Infinite Workspace, see "About the Infinite Workspace" on page 150.

Parameters	Type	Defaults	Function
<i>x, yPixels</i>	float	0, xPixels	The amount of blur as described in Pixels, for example, 200 blurs 200 pixels to either side of the current pixel.
<i>spread</i>	int	0	Tells Shake whether or not to consider outside of the frame. 0 = Compute within the frame (default). 1 = Compute outside of the frame. Because of the Infinite Workspace, it is sometimes handy to compute outside of the frame as well, for example, if the <i>Blur</i> is placed after a <i>Scale</i> command. Note that if nothing is outside of the frame (black), you see a black edge.
<i>x, yFilter</i>	string	"gauss", "gauss"	See "Filter Characteristics" on page 582.
<i>channels</i>	string	"rgba"	What channels Shake should blur. Any or all of RGBA. Default is "rgba".

### Synopsis

```
image Blur(  
    image In,  
    float xPixels,  
    float yPixels,  
    int spread,  
    const char * xFilter,  
    const char * yFilter,  
    const char * channels  
);
```

**Script**

```
image = Blur( In, xPixels, yPixels,  
             spread, "xFilter", "yFilter", "channels" );
```

**Command Line**

```
sbake -blur xPixels yPixels spread etc...
```

**Examples**

```
sbake lisa.iff -blur 400  
sbake lisa.iff -blur "Linear(0,0@1,637@20)" -t 1-20  
sbake lisa.iff -blur 0 500  
sbake lisa.iff -blur 100 500 0 "gauss" "gauss" "rg"
```

**See Also**

“PercentBlur” on page 614

**Convolve**

The *Convolve* function allows you to enter your own custom filter. Standard filters are in your *include/nreal.b* file, and consist of *sharpen*, *edge3x3*, *edge5x5*, *blur3x3*, *blur5x5*, and *laplace*. You can use these or use them as a model to create your own. Just place the filters in a standard plug-ins file. There is an example below.

Parameters	Type	Defaults	Function
<i>channels</i>	string	"rgba"	The channels to which the filter is applied.
<i>kernel</i>	string	"blur3x3"	<i>blur3x3</i> : A 3 x 3 pixel blur.
			<i>blur5x5</i> : A 5 x 5 pixel blur.
			<i>sharpen</i> : Uh, sharpening...
			<i>edge3x3</i> : A 3 x 3 pixel edge detection.
			<i>edge5x5</i> : A 5 x 5 pixel edge detection.
			<i>laplace</i> : Edge detection.
<i>percent</i>	float	100	The mixing percent between the modified image and the original image.
<i>absolute</i>	int	0	Some filters return negative values. When absolute is enabled, they invert to positive values.

## Synopsis

```
image Convolve( image,
    const char * channels,
    const char * kernel,
    float percent,
    int absolute
);
```

## Script

```
image = Convolve(
    image,
    "channels",
    "kernel",
    percent,
    absolute
);
```

## Command Line

*shake -convolve rgba kernel percent*

## Examples

*shake lisa.iff -convolve*

*shake lisa.iff -convolve rgb edge3x3*

## Example Kernel

These are placed by default in the *include/nreal.b* file, but may also be placed in any file as you place a macro, for example, in *include/startup/my\_file.b*.

- The first parameter is the kernel name, enclosed in quotes.
- The second parameter is the kernel size, in this case, 5 x 5.
- The third parameter is the gain factor. Each number is multiplied by this number, so 1 is no change.
- The fourth parameter is the offset, which is added to the result before it is clamped/quantized. 0 is no offset.

```
DefKernel(
    "edge5x5",
    5, 5,
    1,
    0,
    -1, -1, -1, -1, -1,
    -1, -1, -1, -1, -1,
    -1, -1, 24, -1, -1,
```

```

        -1, -1, -1, -1, -1,
        -1, -1, -1, -1, -1
    );

shake -convolve rgba edge5x5 100

```

### See Also

“Blur” on page 585, “Sharpen” on page 617, “Emboss” on page 593, “Median” on page 613

### Defocus

The *Defocus* blur function is a more accurate model of the blurring that occurs through an out-of-focus real-world camera lens. It flares out the high points, resulting in a circular, hexagonal, or octagonal shape around the highlights.

**Original Image**



**Real Camera Defocus**



**Image Blurred With Normal Gaussian Blur in Shake**



**Image Blurred With Defocus in Shake**



Parameters	Type	Defaults	Function
<i>x, yPixels</i>	float	11, 11	Kernel size for the defocus in pixels. Anything less than 3 does nothing. This is the general size of the flare.
<i>channels</i>	string	"rgba"	What channels are calculated for the blur.
<i>percent</i>	float	100	The mix between the blurred and non-blurred image.
<i>shape</i>	string	"gaussian"	Either "fast gaussian," "fast box," "circle," "square," "hexagon," or "octagon." This describes the shape of the flaring. The fast modes give you low quality but fast results.
<i>boostPoint</i>	float	.95	The image value where the superwhite boosting starts. If boostPoint is set to .75, RGB values above .75 are boosted to increase flaring effects. A high value generally decreases flare areas, since fewer candidate pixels are flared.
<i>superWhite</i>	float	1	Max value to boost image to. A value of 1 in the original will be boosted to this value. By boosting this, you increase the brightness of the flare area. Values around 50 yield a very strong boost.

### Synopsis

```
image Defocus( image,
    float xPixels,
    float yPixels,
    const char * channels,
    float percent,
    const char * shape,
    float boostPoint,
    float superWhite
);
```

### Script

```
image = Defocus( image,
    xPixels,
    yPixels,
    "channels",
    percent,
    "shape",
```

```

    boostPoint,
    superWhite
);

```

### Command Line

*shake -defocus xPixels yPixels channels etc...*

### See Also

“PercentBlur” on page 614, “Blur” on page 585, “ZDefocus” on page 620, “IDefocus” on page 605

### DilateErode

The *DilateErode* function isolates each channel and cuts or adds pixels to the edge of that channel. For example, to chew into your mask, set your channels to “a,” and then set the xPixels and yPixels value to -1. By default, you work on whole pixels. To switch to subpixel chewing, enable “soften.” Note that the soften parameter *really* slows the function. If you use the soften feature, use low values for xPixels and yPixels.

You often select “a” as your channel, and then apply a Color-*MMult* to multiply the RGB by the modified alpha.

Parameters	Type	Defaults	Function
<i>channels</i>	string	"rgba"	The affected channels. Any or all of RGBA.
<i>x, yPixels</i>	float	0, xPixels	The amount of pixels added or taken on an edge. Positive values add to the edge; negative values eat away at the edge.
<i>borders</i>	int	0	Tells Shake to consider or ignore the border pixels.
<i>soften</i>	int	0	Turns softening on or off, or affects the subpixel. If enabled, it considerably slows down the software at high x, yPixel values.
<i>sharpness</i>	float	0	The sharpness factor for the softening. A value of 0 gives a smooth gradation, whereas 2 gives you a sharp cutoff.

### Synopsis

```

image DilateErode( image,
    const char * channels,
    float xPixels,
    float yPixels,
    int borders,
    int soften

```

```
float sharpness
);
```

### Script

```
image = DilateErode(image,
    "channels",
    xPixels, yPixels,
    borders, soften, sharpness
);
```

### Command Line

*shake -dilateerode channels xPixels yPixels etc...*

### EdgeDetect

The *EdgeDetect* function is great for pulling out and massaging edges. You can control what is detected, the strength of the edge, and the ability to expand or soften the edge.

Parameters	Type	Defaults	Function
<i>strength</i>	float	1	A gain on the filter. The more strength, the more details appear. Numbers are practically from 1 to 10, but you can go much higher.
<i>threshold</i>	float	0	Pixels below the threshold are not revealed in the filter and turn black. The range is 0 to 1.
<i>binaryOutput</i>	int	0	When enabled, there is no grayscale, only black and white pixels.
<i>directionFilter</i>	int	0	Enables an effect similar to <i>Emboss</i> .
<i>directionFilterAngle</i>	float	0	The lighting angle. Only is in effect when <i>directionFilter</i> is on.
<i>despeckle</i>	int	0	Similar to a Median filter, this cleans up isolated pixels to the despeckle radius.
<i>x, yBlur</i>	float	0	Blurs the result.
<i>x, yDilateErode</i>	float	0	Applies the <i>DilateErode</i> effect to the first filter pass.
<i>rgbEdges</i>	int	0	Inherits color from the input image instead of just black and white.
<i>addEdgesToImage</i>	int	0	Mixes the filter result back into the input image.

Parameters	Type	Defaults	Function
<i>method</i>	string	"Sobel"	Your choices are "Sobel," "Babu," or "Fast." Babu is an algorithm that is extremely slow and cannot be used on large (2K or larger) plates. It is maintained for compatibility purposes.
<i>babuSteps</i>	int	8	The steps done for the Babu filter. The more steps you add, the slower the filter.
<i>bytes</i>	int	1	Output bit depth.

**Synopsis**

```
image EdgeDetect(  
    image In,  
    float strength,  
    float threshold,  
    int binaryOutput,  
    int directionFilter,  
    float directionFilterAngle,  
    int despeckle,  
    float xBlur,  
    float yBlur,  
    float xDilateErode,  
    float yDilateErode,  
    int rgbEdges,  
    int addEdgesToImage,  
    const char * method,  
    int babuSteps,  
    int bytes  
);
```

**Script**

```
image EdgeDetect(  
    In,  
    strength,  
    fthreshold,  
    binaryOutput,  
    directionFilter,  
    directionFilterAngle,  
    despeckle,  
    xBlur,  
    yBlur,
```

```

xDilateErode,
yDilateErode,
rgbEdges,
addEdgesToImage,
"method",
babuSteps,
bytes
);

```

## Command Line

*shake -edgedetect strength*

## Emboss

With the *Emboss* function, you control the gain and light direction.

**Note:** The *Emboss* function converts your image to a BWA image (since there is no color information).

If you do an extreme gain, you may start to see terracing on your image. To correct this, insert a *Bytes* node before the *Emboss* node, and boost your image to 2 bytes per channel. You can get interesting patterns with a *Bytes* node set to 2 bytes, followed by a *Blur* node, and then the *Emboss* node.

The elevation is set to 30 because it makes the median grey value .5.

Parameters	Type	Defaults	Function
<i>gain</i>	float	10	The amount of emboss.
<i>azimuth</i>	float	45	The direction of the light. 0 and 360 are from the right side of the image, 90 is from the top, etc.
<i>elevation</i>	float	30	This is the "height" of the light. 0 is parallel to the image; 90 is the same axis as a line from your nose to the image.

## Synopsis

```

image Emboss( image,
float gain,
float azimuth,
float elevation
);

```

## Script

```
image = Emboss( image, gain, azimuth, elevation );
```

## Command Line

*shake -emboss gain azimuth elevation*

## FilmGrain

Use the *FilmGrain* function to apply grain that corresponds to real film grain to an element. Grain is typically added to still or CG images so the images more closely match the inherent noisiness of film plates.




You can choose to apply a preset film stock, sample grain from an existing image, or create your own grain by adjusting the sliders.

### To sample grain from an image:

- 1 Attach a Filter—*FilmGrain* node to the element you want to add grain.
- 2 Ensure you are not in Proxy mode.
- 3 Load the image to be sampled in the Viewer.
- 4 Ensure the *FilmGrain* parameters are still active.
- 5 In the Viewer, drag boxes in areas you want to sample.

**Note:** The sampled areas should be very flat without detail that may disrupt the grain analysis. Small elements are perceived as grain detail, so the best sample areas are featureless walls, exposure cards, and so on.

You can sample as many areas as you want.

- 6 To undo a sample, do one of the following:
  - To undo a box drawing, click Undo Last Region .
  - To remove all boxes and start over, click Reset the Regions .
- 7 Once the boxes are set, click Analyze the Grain  on the Viewer. The parameters in the *FilmGrain* node are set to match the plate.

The following table contains the *FilmGrain* parameters.

Parameters	Type	Defaults	Function
<i>version</i>	string	"v3.0"	This parameter only appears in the script. It allows for versioning of the function and should therefore be left alone.
<i>intensity</i>	float	1	The intensity of the grain.
<i>grainSize</i>	float	1	Size of the grain, between 0 and 2.
<i>aspectRatio</i>	float	defaultAspectRatio()	Sets the aspect ratio of the grain to compensate for anamorphic or non-square pixel distortion.

Parameters	Type	Defaults	Function
<i>seed</i>	float	time	The random generation seed. Set this to a constant value to freeze the grain.
<i>filmStock</i>	string	"Custom"	<p>Allows the user to select from preset film stocks or to apply custom values. Accepted inputs are:</p> <p>"Custom"</p> <p>"Eastman 5245"</p> <p>"Eastman 5247"</p> <p>"Eastman 5248ac"</p> <p>"Eastman 5248nc"</p> <p>"Eastman 5287"</p> <p>"Eastman 5293ac"</p> <p>"Eastman 5293nc"</p> <p>"Eastman 5296"</p> <p>"Eastman 5298"</p> <p>"Kodak 5246ac"</p> <p>"Kodak 5246nc"</p> <p>"Kodak 5274ac"</p> <p>"Kodak 5274nc"</p> <p>"Kodak 5277"</p> <p>"Kodak 5279"</p> <p>"ac" indicates a stock that was scanned with aperture correction turned on.</p> <p>"nc" indicates no aperture correction.</p>
<i>r, g, bStdDev</i>	float	.05	This controls the relative intensity between the r, g, and b channels.
<i>r, g, bSoftness</i>	float	1.2	This value controls the softness of the grain. Note that this also affects the apparent size of the grain and thus, you may need to decrease grainSize to compensate.

Parameters	Type	Defaults	Function
<i>r, g, bFilmResponse</i>	float	-1	The value determines the distribution of the grain relative to the luminance of the image.
<i>colorCorr</i>	float	0	<p>This parameter specifies the apparent colorfulness of the grain.</p> <p>The value represents how closely the grain in each channel overlaps. This means that negative color-correlation values decrease the amount of overlap, which increases the apparent color of the grain, while positive values decrease its colorfulness.</p>

### Synopsis

```
image FilmGrain(
    image In,
    const char * version,
    float intensity,
    float grainSize,
    float aspectRatio,
    float seed,
    const char * filmStock,
    float rStdDev,
    float gStdDev,
    float bStdDev,
    float rSoftness,
    float gSoftness,
    float bSoftness,
    float rFilmReponse,
    float gFilmReponse,
    float bFilmReponse,
    float colorCorr
);
```

### Script

```
image FilmGrain(
    In,
    "version",
    intensity,
    grainSize,
    aspectRatio,
```

```
seed,
filmStock,
rStdDev,
gStdDev,
bStdDev,
rSoftness,
gSoftness,
bSoftness,
rFilmReponse,
gFilmReponse,
bFilmReponse,
colorCorr
);
```

**Command Line**

Not appropriate for the command line.

**Grain**

The *Grain* function adds grain to an image. It is used to simulate film grain for 3D-rendered elements. The *Grain* function is not as accurate as the newer *FilmGrain* node.

This node gives you complete per-channel control over grain size, density, softness, etc. The controls are explained below, with visual examples after the parameter list.

**Note:** If you have an RGB channel image, there is no grain if obeyAlpha is enabled, as there is an alpha value of 0.

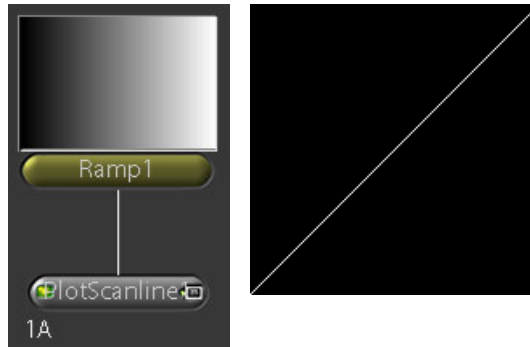
In general, when there is a parameter followed by the same parameter on a per-channel basis, the first one acts as a multiplier on the channel parameters. For example, a density of .5 multiplies r, g, bDensity by .5.

Parameters	Type	Defaults	Function
<i>size, r, g, bSize</i>	float	1	Overall size of the grain. "size" is a multiplier on r, g, bSize. You can have sizes less than 1.
<i>density, r, g, bDensity</i>	float	.8	The density of the grain. 1 is maximum density. "density" is a multiplier on r, g, bDensity.
<i>seed</i>	float	time	The random seed for the grain.

Parameters	Type	Defaults	Function
<i>filmResponse</i>	float	-1	The valid range is theoretically infinite, but practically is -1 and 1. -1 is typical of standard film, with grain applied to the entire range except the brightest whites, and black is the most affected. 1 is the inverse of that, withholding grain from the darks, with most grain on the whites. Default is -1.
<i>lGain, r, g, bGain</i>	float	.5, .5, rGain, rGain	The overall intensity of the grain. lGain (luminance) applies to all three channels equally, while r, g, bGain apply only to that channel.
<i>r, g, bBias</i>	float	0	Shifts the grain level higher or lower in intensity.
<i>r, g, bLowOffset</i>	float	0	From the midpoint of the Gain, squeezes the darker grain areas. This is a useful adjustment when the highlights are good and you want to modify only the level of the darker grain.
<i>r, g, bHighOffset</i>	float	0	From the midpoint of the Gain, squeezes the brighter grain areas.
<i>aspect</i>	float	1	The grain aspect ratio. An aspect of 2 stretches it twice as wide.
<i>softness, r, g, bSoftness</i>	float	0	A value above 0 softens the grain. A value below 0 sharpens the grain. The global softness is an additive factor, so it is added to the values of the per-channel parameters.
<i>obeyAlpha</i>	int	0	When enabled, grain is applied to the image through its alpha channel. When disabled, grain is applied to the entire image, and the alpha is ignored. Useful for applying grain to premultiplied CG images without contaminating the background black.

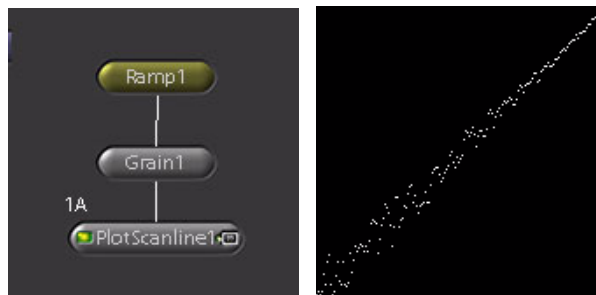
## Grain Example

In the following example, the first node tree consists of a *Ramp* node and a *PlotScanline* node. The *PlotScanline* node is added to analyze the image.

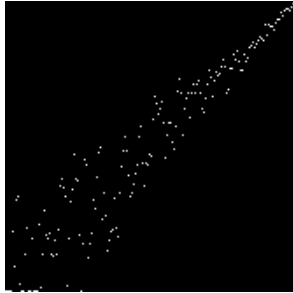


Because the ramp is black to white, a linear line appears in the *PlotScanline* from left to right when no grain is applied.

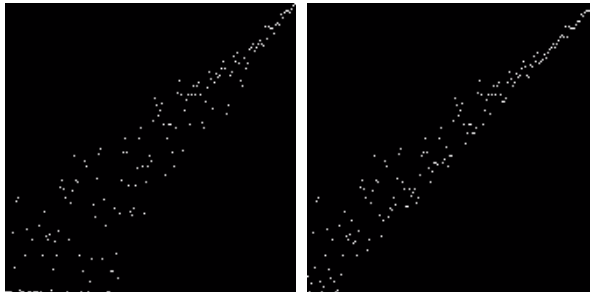
When a *Grain* node is inserted between the *Ramp* node and the *PlotScanline* node, noise is introduced that disturbs the line. There is more noise (grain) near the lower, dark area of the *PlotScanline*. This is due to the *filmResponse* of -1, which concentrates grain on the lower areas.



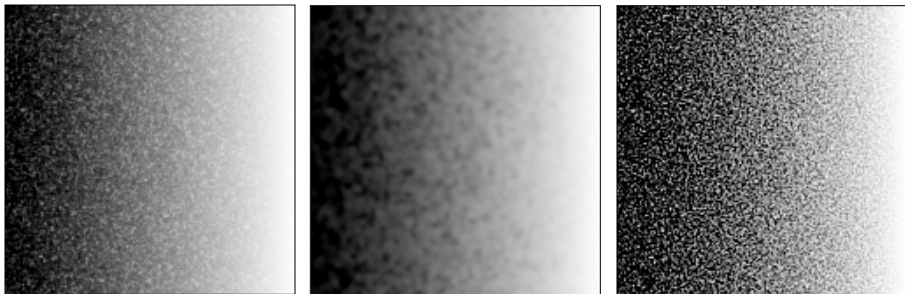
The next image is the result of increasing the lGain (or r, g, bGain on a per-channel basis), and increasing the range of the grain. This results in making it both lighter and darker simultaneously.



The following two images show the result of modifying the Bias and Gain. In the first image, the rBias is lowered (looking only at the red channel) to -.5. The grain shifts downward from the diagonal line, making it darker. This affects both the lighter grain (the dots above the original diagonal line) and the darker grain (the dots below the original diagonal line). To adjust only the lighter or darker grains, use the offset sliders. The second image shows a rLowOffset of .5, squeezing only the darker grains, leaving only lighter grains.



In the following images, the left image shows the standard softness. The middle image has a softness of 10, and the right image has a softness of -10.



## Synopsis

```
image Grain(  
    image In,  
    float size,  
    float density,  
    float seed,  
    float filmResponse,  
    float lGain,  
    float rGain,  
    float gGain,  
    float bGain,  
    float rSize,  
    float gSize,  
    float bSize,  
    float rDensity,  
    float gDensity,  
    float bDensity,  
    float aspect,  
    float rBias,  
    float rLowOffset,  
    float rHighOffset,  
    float gBias,  
    float gLowOffset,  
    float gHighOffset,  
    float bBias,  
    float bLowOffset,  
    float bHighOffset,  
    float softness,  
    float rSoftness,  
    float gSoftness,  
    float bSoftness,  
    int obeyAlpha  
);
```

## Script

```
image = Grain(  
    image,  
    size,  
    density,  
    seed,  
    filmResponse,  
    lGain, rGain, gGain, bGain,
```

```

rSize, gSize, bSize,
rDensity, gDensity, bDensity,
aspect,
rBias, rLowOffset, rHighOffset,
gBias, gLowOffset, gHighOffset,
bBias, bLowOffset, bHighOffset,
softness, rSoftness, gSoftness, bSoftness,
obeyAlpha
);

```

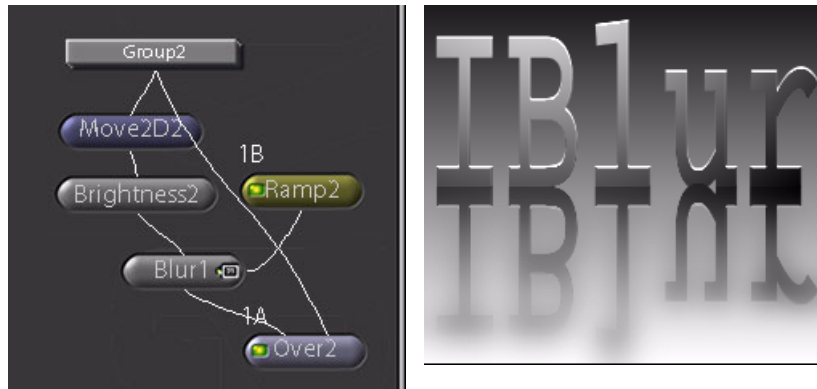
### Command Line

*shake -grain size density seed etc...*

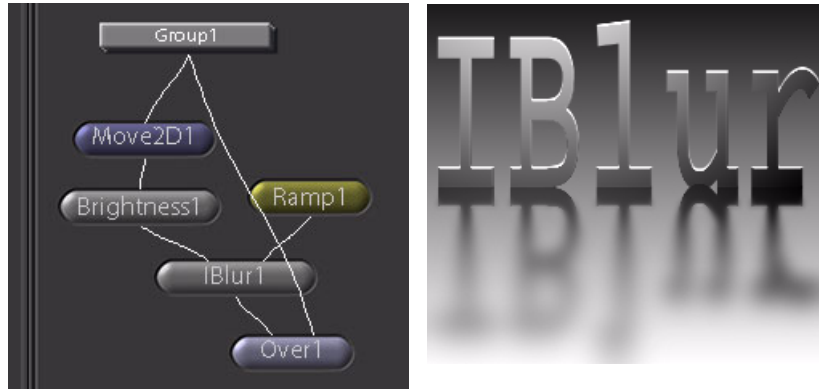
### IBlur

The *IBlur* function blurs the image, with the amount of blur set by a second control image. Maximum blur occurs in the white areas of the second image, and no blur occurs in the black areas.

In the following example, the first node tree uses a *Ramp* that is approximately one-half of the height of the image as a mask for a blur. Some bad blending occurs in the lower portion of the image.



In the second node tree, a *Ramp* is used as a second input into *IBlur*, and the result is a nice blend of the blurred and non-blurred areas.



The above script is saved as *doc/html/cook/iblur\_example.shk*.

**Note:** The *IBlur* function is much slower than the normal *Blur* function.

Parameters	Type	Defaults	Function
<i>controllmg</i>	image		The second image is the control image for the blur. Its brightness determines how much blur occurs in the first image.
<i>x, yPixels</i>	float	0, xPixels	The amount of blur as described in pixels. For example, 200 blurs 200 pixels to either side of the current pixel.
<i>spread</i>	int	0	Tells Shake whether or not to consider outside of the frame. 0 = Compute within the frame. 1 = Compute outside of the frame. Because of the Infinite Workspace, it is sometimes handy to compute outside of the frame as well, for example, if the <i>IBlur</i> is placed after a <i>Scale</i> node. Note that if nothing is outside of the frame (black), you see a black edge.
<i>x, yFilter</i>	string	"gauss"	See "Filter Characteristics" on page 582.
<i>steps</i>	int	5	The amount of steps. The intensity of the control image is divided up X amount of zones, with X equal to steps.

Parameters	Type	Defaults	Function
<i>stepBlend</i>	float	1	Controls the blending between the amount of regions (see below). If you put this at 0, each step has a constant blur value. If this is 1, there is a continuous blend between the different regions.
<i>controlChannel</i>	string	"a"	The channel of the second image to use to control the amount of blur.
<i>channels</i>	string	"rgba"	The channels of the first image to blur.
<i>invert</i>	int	0	Inverts the controlChannel.

### Synopsis

```
image IBlur(
    image img,
    image controlImg,
    float xPixels,
    float yPixels,
    int spread,
    const char * xFilter,
    const char * yFilter,
    int steps,
    float stepBlend,
    const char * controlChannel,
    const char * channels,
    int invert
);
```

### Script

```
image IBlur(
    img,
    controlImg,
    xPixels,
    yPixels,
    spread,
    "xFilter",
    "yFilter",
    steps,
    stepBlend,
    "controlChannel",
    "channels",
```

```
    invert
);
```

### Command Line

*shake -ibblur image xPixels yPixels spread etc...*

### See Also

“PercentBlur” on page 614, “About Masks” on page 383

### IDefocus

The *IDefocus* function is the *Defocus* function with a second image input to control the size of the defocused flaring. For more information, see “*Defocus*” on page 588.

Parameters	Type	Defaults	Function
<i>x, yPixels</i>	float	11, 11	Kernel size for the defocus in pixels. Anything less than 3 does nothing. This is the general size of the flare.
<i>channels</i>	string	"rgba"	What channels are calculated for the blur.
<i>percent</i>	float	100	The mix between the blurred and non-blurred image.
<i>shape</i>	string	"gaussian"	Either "fast gaussian," "fast box," "circle," "square," "hexagon," or "octagon." Describes the shape of the flaring. The fast modes give you low quality but fast results.
<i>boostPoint</i>	float	.95	The image value where the superwhite boosting starts. If boostPoint is .75, RGB values above .75 are boosted to increase flaring effects. A high value generally decreases flare areas, since fewer candidate pixels are flared.
<i>superWhite</i>	float	1	Max value to boost image. A value of 1 in the original is boosted to this value. By boosting this, you increase the brightness of the flare area. Values around 50 give you a very strong boost.
<i>steps</i>	int	5	The amount of regions. The intensity of the control image is divided up X amount of zones, with X equal to regions.
<i>stepBlend</i>	float	1	Controls the blending between the amount of regions (see below). If you put this at 0, each step has a constant blur value. If this is 1, there is a continuous blend between the different regions.

Parameters	Type	Defaults	Function
<i>controlChannel</i>	string	"a"	The channel of the second image used to control the amount of sharpen.
<i>invert</i>	int	0	Inverts the controlChannel.

### Synopsis

```
image IDefocus(
    image img,
    image controlImg,
    float xPixels,
    float yPixels,
    const char * channels,
    float percent,
    const char * shape,
    float boostPoint,
    float superWhite,
    int steps,
    float stepBlend,
    const char * controlChannel,
    const char * channels,
    int invert
);
```

### Script

```
image = IDefocus(
    img,
    controlImg,
    xPixels,
    yPixels,
    "channels",
    percent,
    "shape",
    boostPoint,
    superWhite
    steps,
    stepBlend,
    "controlChannel",
    "channels",
    invert
);
```

## Command Line

*shake -idefocus controllmg xPixels yPixels channels etc...*

## See Also

"PercentBlur" on page 614, "Blur" on page 585, "Defocus" on page 588

## IDilateErode

The *IDilateErode* function isolates each channel and cuts or adds pixels to the edge of that channel. For example, to chew into your mask, set your channels to "a," and then set the xPixels and yPixels values to -1. By default, you work on whole pixels. To switch to subpixel chewing, enable "soften." Note that the soften parameter *really* slows the function. If you use the soften feature, use low values for xPixels and yPixels.

Parameters	Type	Defaults	Function
<i>channels</i>	string	"rgba"	The affected channels.
<i>x, yPixels</i>	float	0, xPixels	The amount of pixels added or taken on an edge. Positive values add to the edge; negative values eat away at the edge.
<i>border</i>	int	0	Tells Shake to consider or ignore the border pixels.
<i>soften</i>	int	0	This toggle enables softening or affects the subpixel. If enabled, it considerably slows the software at high x, yPixel values.
<i>sharpness</i>	float	0	The sharpness factor for the softening. A value of 0 gives a smooth gradation, whereas 2 yields a sharp cutoff.
<i>steps</i>	int	5	The amount of regions that the image is divided into. The intensity of the control image is divided up X amount of zones, with X equal to regions.
<i>stepBlend</i>	float	1	Controls the blending between the amount of regions (see below). If you put this at 0, each step has a constant dilate value. If this is 1, there is a continuous blend between the different regions.
<i>controlChannel</i>	string	"a"	The channel of the second image to use to control the amount of dilate.
<i>invert</i>	int	0	Inverts the controlChannel.

## Synopsis

```
image IDilateErode(  
    image img,  
    image controlImg  
    const char * channels,  
    float xPixels,  
    float yPixels,  
    int borders,  
    int soften  
    float sharpness  
    float stepBlend,  
    int steps,  
    const char * controlChannel,  
    const char * channels,  
    int invert  
);
```

## Script

```
image = IDilateErode(  
    img,  
    controlImg,  
    "channels",  
    xPixels,  
    yPixels,  
    borders,  
    soften,  
    sharpness,  
    stepBlend,  
    steps,  
    "controlChannel",  
    "channels",  
    invert  
);
```

## Command Line

*shake -idilateerode controlImg channels xPixels yPixels etc...*

## IRBlur

The *IRBlur* function is an image-based version of the *RBlur* function, using the alpha mask of a second image (by default) to control the amount of radial blurring on an image. This is useful for faking motion blur effects.

In this example, the foreground objects (cubes) are rendered on a beach in a 3D software package with the depth information. The 3D image is composited over a background photo of the beach. The *DepthKey* node was used to extract a matte of the depth supplied by the 3D render. The matte is then fed into the second image of the *IRBlur* to give a zooming effect.



Parameters	Type	Defaults	Function
<i>x, yCenter</i>	float	width/2, height/2	The center point of the blur.
<i>iRadius</i>	float	0	The distance from the center that contains the blur sample area.
<i>oRadius</i>	float	width	The outer edge for the blur area.
<i>aspectRatio</i>	float	1	The aspect ratio of the pixels for anamorphic images.
<i>damp</i>	float	1	A gamma value on the blur.
<i>amplitude</i>	float	.5	The amount of blur. This number can also be negative.
<i>blurQuality</i>	float	.25	The amount of samples. A quality of 1, the maximum, is 64 samples.
<i>mirror</i>	int	0	Considers points beyond the center area if your amplitude is high enough when enabled.
<i>stepBlend</i>	float	5	Controls the blending between the amount of regions (see below). If set to 0, each step has a constant blur value. If set to 1, there is a continuous blend between the different regions.
<i>steps</i>	int	1	The amount of regions. The intensity of the control image is divided up X amount of zones, with X equal to regions. If you see stepping between the zones and stepBlend does not correct it, increase steps to a higher number. The more steps you use, the slower the process.

Parameters	Type	Defaults	Function
<i>controlChannel</i>	string	"a"	The channel of the second image to use to control the amount of sharpen.
<i>invert</i>	int	0	Inverts the controlChannel.

### Synopsis

```
image IRBlur(
    image img,
    image controlImg,
    float xCenter,
    float yCenter
    float iRadius,
    float oRadius,
    float aspectRatio,
    float damp,
    float amplitude,
    float blurQuality,
    int mirror,
    int steps,
    float stepBlend,
    const char * controlChannel,
    int invert,
);
```

### Script

```
image = IRBlur(
    img,
    controlImg,
    xCenter, yCenter,
    iRadius, oRadius,
    aspectRatio,
    damp,
    amplitude,
    blurQuality,
    mirror,
    steps,
    stepBlend
    "controlChannel",
    invert
);
```

## Command Line

*shake -irblur controllmg xCenter yCenter iRadius etc...*

## See Also

“RBlur” on page 615, “DepthKey” on page 256, and “About Motion Blur” on page 445.

## ISharpener

The *ISharpener* function is identical to the *Sharpen* function, except it uses a second control image to control the amount of sharpening across the first image. Un-sharp masking is used for the sharpening filter. This process blurs the image slightly, takes the difference between the blurred result and the input image, and adds that back over the input image. High values of x and yPixels (for example, greater than 3 percent of the image size) return undesirable results. You can also use the sharpen filter in the *Convolve* function, but ringing may result.

For an example of this process, see “IBlur” on page 602.

Parameters	Type	Defaults	Function
<i>controllmg</i>	image		The second image is the control image for the sharpen. The brightness of this image controls how much sharpening is applied.
<i>percent</i>	float	20	The amount of blend between the non-sharpened image (0) and the sharpened image (100).
<i>x, yPixels</i>	float	3, xPixels	The amount of blur as described in pixels. For example, 200 blurs 200 pixels to either side of the current pixel.
<i>steps</i>	int	5	This is the amount of regions. The intensity of the control image is divided up X amount of zones, with X equal to regions.
<i>stepBlend</i>	float	1	Controls the blending between the amount of regions (see below). If set to 0, each step has a constant blur value. If this is 1, there is a continuous blend between the different regions.
<i>controlChannel</i>	string	"a"	The channel of the second image to use to control the amount of sharpen.
<i>channels</i>	string	"rgba"	The channels of the first image to sharpen.
<i>invert</i>	int	0	Inverts the controlChannel.

## Synopsis

```
image ISharpen(  
    image img,  
    image controlImg,  
    float percent,  
    float xPixels,  
    float yPixels,  
    int steps,  
    float stepBlend,  
    const char * controlChannel,  
    const char * channels,  
    int invert  
);
```

## Script

```
image ISharpen(  
    img,  
    controlImg,  
    percent,  
    xPixels,  
    yPixels,  
    steps,  
    stepBlend,  
    "controlChannel",  
    "channels",  
    invert  
);
```

## Command Line

*shake -isharpen image percent xPixels yPixels*

### Median

The *Median* function applies a 3 x 3 median filter. It is good for the removal of individual dots and noise.

Parameters	Type	Defaults	Function
<i>channels</i>	string	"rgba"	The channels to which the filter is applied.
<i>threshold</i>	float	1	The change is allowed if the change is above or below this threshold value.
<i>thresholdType</i>	int	0	Defines whether the result is below (0) or above (1) the threshold to allow the change.

### Synopsis

```
image Median( image,
    const char * channels,
    float threshold,
    int thresholdType
);
```

### Script

```
image = Median( image, "channels", threshold, thresholdType);
```

### Command Line

```
shake -median channels threshold thresholdType etc...
```

### PercentBlur

The *PercentBlur* function blurs the image according to a percentage factor, rather than a pixel size. Therefore, 100 percent blurs with the entire image in the blur calculation.

Parameters	Type	Defaults	Function
<i>percent</i>	float	0	Percent of the image taken into consideration for the blur. 100 = the entire image, or a flat color. Default is 0.
<i>spread</i>	int	0	<p>Tells Shake whether or not to consider outside of the frame.</p> <p>0 = Compute within the frame (default).</p> <p>1 = Compute outside of the frame.</p> <p>Because of the Infinite Workspace, it is sometimes handy to compute outside of the frame as well, for example, if the <i>Blur</i> is placed after a <i>Scale</i> command. Note that if nothing is outside of the frame (black), you see a black edge.</p>
<i>channels</i>	string	"rgba"	What channels Shake should blur. Any or all of RGBA.

### Synopsis

```
image PercentBlur(  
    image,  
    float percent,  
    int spread,  
    const char * channels  
);
```

### Script

```
image = PercentBlur( image, percent, spread, "channels" );
```

### Command Line

*shake -percentblur percent spread*

### See Also

“*Blur*” on page 585

## Pixelize

The *Pixelize* function filters the images into square blocks, giving a mosaic look by averaging all of the pixels within the block.

Parameter	Type	Default	Function
<i>x, yPixels</i>	int	4, xPixels	The block size in pixels to be filtered.

### Synopsis

```
image Pixelize(  
    image,  
    int xPixels  
    int yPixels  
);
```

### Script

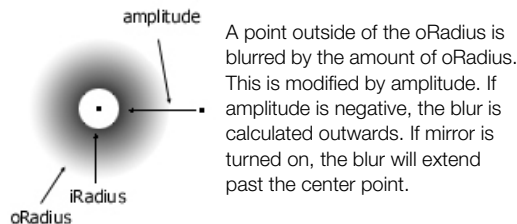
```
image = Pixelize( image, xPixels, yPixels);
```

### Command Line

```
sbake -pixelize xPixels yPixels
```

## RBlur

The *RBlur* function blurs the image radially from a center point, creating flare effects. Although an expensive process, it is extremely accurate. The entire image is blurred, based on the range from the inner radius to the outer radius. A pixel outside of the oRadius distance is blurred by the oRadius amount. A pixel inside of the iRadius amount is not blurred. A pixel between iRadius and oRadius is blurred by a percentage of its position between the two Radius parameters. You can also use the mirror parameter to blur beyond the center point, as well as use a negative amplitude to sample pixels away from the center rather than toward the center.



Parameters	Type	Defaults	Function
<i>x, yCenter</i>	float	width/2, height/2	The center point of the blur.
<i>iRadius</i>	float	10	The distance from the center that contains the blur sample area.
<i>oRadius</i>	float	20	The outer edge for the blur area.
<i>aspectRatio</i>	float	1	The aspect ratio of the pixels for anamorphic images.
<i>damp</i>	float	1	A gamma value on the blur.
<i>amplitude</i>	float	.5	The amount of blur. This number can also be negative.
<i>quality</i>	float	.25	The amount of samples. A quality of 1, the maximum, is 64 samples.
<i>mirror</i>	int	0	Considers points beyond the center area if your amplitude is high enough when enabled.

### Synopsis

```
image RBlur( image,
    float percent,
    float xCenter,
    float yCenter
    float iRadius,
    float oRadius,
    float aspectRatio,
    float damp,
    float amplitude,
    float quality,
    int mirror
);
```

### Script

```
image = RBlur( image, xCenter, yCenter, iRadius, etc....);
```

### Command Line

*shake -rblur percent xCenter yCenter iRadius etc...*

### Sharpen

Un-sharp masking is used for the sharpening filter. This process blurs the image slightly, takes the difference between the blurred result and the input image, and adds that back over the input image. High values of *x* and *yPixels* (for example, greater than 3 percent of the image size) return undesirable results. You can also use the sharpen filter in the *Convolve* function, but ringing may result.

Parameters	Type	Defaults	Function
<i>percent</i>	float	20	The amount of mixing between the sharpened and the original image. With a value of 100, none of the original image is present.
<i>x, yPixels</i>	float	3, xPixels	The amount of sharpening as described in pixels.
<i>channels</i>	string	"rgba"	What channels Shake should sharpen. Any or all of RGBA. Default is "rgba".

### Synopsis

```
image Sharpen( image,  
    float percent,  
    float xPixels,  
    float yPixels,  
    const char * channels  
);
```

### Script

```
image = Sharpen( image, percent, xPixels, yPixels, "channels" );
```

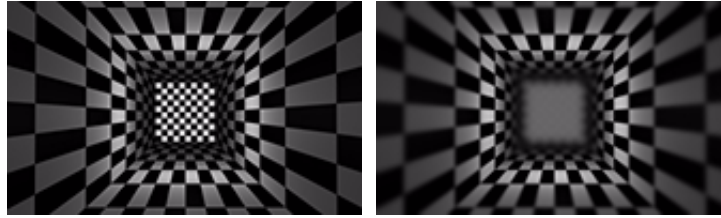
### Command Line

```
shake -sharpen percent xPixels yPixels
```

### ZBlur

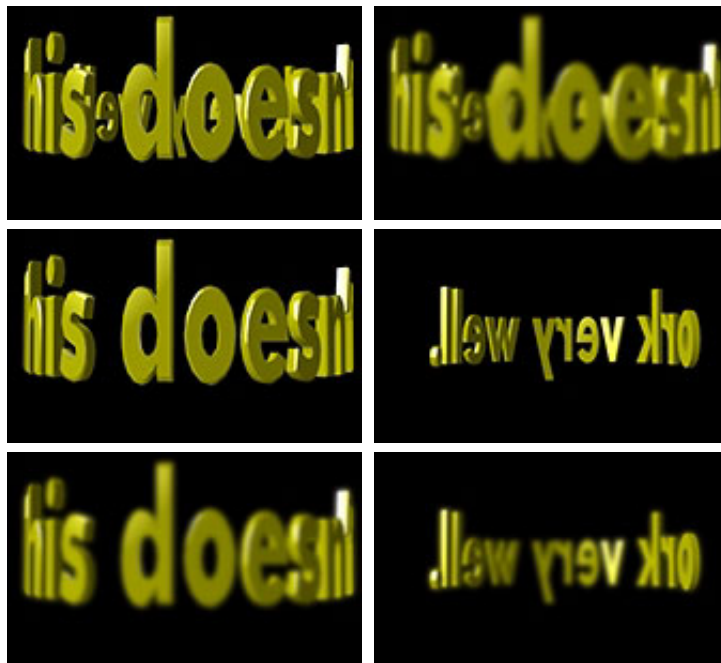
The *ZBlur* function is a dedicated version of *IBlur* used to simulate depth-of-field blurring based on the Z channel. You can set a center point of focus (*focusCenter*) and the range of drop-off to the maximum amount of blur. If you do not have a Z channel, you can copy a channel from another image. Therefore, if you are working in 8 bit or 16 bit, your ranges are between 0 and 1. For example, your *focusCenter* could not be set to 10, but is somewhere between 0 and 1.

The *ZBlur* function works best with gradual changes of Z, for example, looking down a hallway, represented in the following images.



In cases where foreground objects overlap blurred background objects, ringing results on the background. It is recommended that you separate the foreground from the background into two separate elements. For example, do two renders if you are generating elements in 3D.

The following example is from a 3D render. Ringing appears around the background letters of the text when normally passed into a *ZBlur*. Instead, separate the circle into foreground and background elements with the 3D render. These are then blurred and composited in Shake.



Parameters	Type	Defaults	Function
<i>amount</i>	float	20	The maximum amount of blur.
<i>near</i>	float	0	The value of distance toward the camera at which maximum blur occurs.
<i>far</i>	float	1	The value of distance away from the camera at which maximum blur occurs.
<i>focusCenter</i>	float	$(\text{far} - \text{near})/2 + \text{near}$	The distance from the camera at which there is no blur.
<i>focusRange</i>	float	0	The distance away from the focusCenter, both toward and away from the camera, that remains in unblurred.
<i>steps</i>	int	5	The amount of steps that the total range is divided between.
<i>stepBlend</i>	float	1	The mixing of the different steps. 0 is no mixing and good for getting a feel for your step ranges. 1 is complete, linear blending.

### Synopsis

```
image ZBlur( image,
    float amount,
    float near,
    float far,
    float focusCenter,
    float focusRange,
    int steps,
    float stepBlend
);
```

### Script

```
image ZBlur( image,
    amount,
    near,
    far,
    focusCenter,
    focusRange,
    steps,
```

```

    stepBlend
);

```

### Command Line

*shake -zblur amount near far etc...*

### See Also

“PercentBlur” on page 614, “Blur” on page 585, “IBlur” on page 602, “Defocus” on page 588

### ZDefocus

The *ZDefocus* function is identical to the *Defocus* blurring function, except you can do a realistic depth of field by using the image’s Z channel. The Z channel usually comes from a Z-depth 3D render, but of course can also be placed with Shake channel-swapping tools (*Reorder* and *Copy*).

Like *ZBlur*, this function works best when there is no abrupt overlapping of foreground objects over background. The foreground elements work fine, but ringing occurs on the background. If you are developing a shot of this nature, try to split your foreground into a separate element.



Parameters	Type	Defaults	Function
<i>x, yPixels</i>	float	11, 11	Kernel size for the defocus in pixels. Anything less than 3 does nothing. This is the general size of the flare.
<i>channels</i>	string	"rgba"	What channels are calculated for the blur.
<i>percent</i>	float	100	The mix between the blurred and non-blurred image.
<i>shape</i>	string	"gaussian"	Either "fast gaussian," "fast box," "circle," "square," "hexagon," or "octagon." Describes the shape of the flaring. The fast modes give you low quality but fast results.
<i>boostPoint</i>	float	.95	The image value where the superwhite boosting starts. If boostPoint is .75, RGB values above .75 are boosted to increase flaring effects. A high value generally decreases flare areas, since fewer candidate pixels are flared.

Parameters	Type	Defaults	Function
<i>superWhite</i>	float	1	Max value to boost image. A value of 1 in the original is boosted to this value. By boosting this, you increase the brightness of the flare area. Values around 50 yield a very strong boost.
<i>near</i>	float	0	The value of distance toward the camera at which maximum blur occurs.
<i>far</i>	float	1	The value of distance away from the camera at which maximum blur occurs.
<i>focusCenter</i>	float	$(\text{far} - \text{near}) / 2 + \text{near}$	The distance from the camera at which there is no blur.
<i>focusRange</i>	float	0	The distance away from the focusCenter, both toward and away from the camera, that remains unblurred.
<i>steps</i>	int	5	The amount of steps that the total range is divided between.
<i>stepBlend</i>	float	1	The mixing of the different steps. 0 is no mixing, and is good for getting a feel for your step ranges. 1 is complete, linear blending.

### Synopsis

```
image ZDefocus( image,
    float xPixels,
    float yPixels,
    const char * channels,
    float percent,
    const char * shape,
    float boostPoint,
    float superWhite
    float near,
    float far,
    float focusCenter,
    float focusRange,
    int steps,
    float stepBlend
);
```

### **Script**

```
image = ZDefocus( image,  
    xPixels,  
    yPixels,  
    "channels",  
    percent,  
    "shape",  
    boostPoint,  
    superWhite  
    near,  
    far,  
    focusCenter,  
    focusRange,  
    steps,  
    stepBlend  
);
```

### **Command Line**

*sbake -zdefocus xPixels yPixels channels etc...*

# Customizing Shake

## Chapter Summary

- Setting Preferences and Customizing Shake
- Locations for .h Files and Custom Icons
- General Settings
- The Curve Editor and Time Bar
- File Path and Browser Controls
- Function Tabs
- Node View
- Parameters Tab
- Viewers
- Template Preference Files
- Environment Variables for Shake
- Interface Devices and Styles

## Setting Preferences and Customizing Shake

This chapter explains how to customize the appearance of Shake, macro interactivity, and performance parameters. It also lists environment variables you can set to improve Shake's performance.

For information on creating Viewer scripts, see “Viewer Lookups, Viewer Scripts, and the Viewer DOD” on page 30.

For information on creating custom kernels for filters, see “*Convolve*” on page 586.

For a tutorial on creating a macro, see the “How to Make a Macro” lesson in the *Shake 3 Tutorials*.

For information on scripting, see “Script Manual” on page 692.

## Locations for .h Files and Custom Icons

This section discusses where to store files used to customize Shake.

### Preference Files Search Path

Shake uses two important files in the `<ShakeInstall>/Contents/Resources (<ShakeDir>/include for non-MacOS)` directory, named *nreal.b* and *nrui.b*. The first file, *nreal.b*, lists every function and all default performance settings. This file should not be touched by the user, but may be read as a formatting reference. The commands found in this file can be copied and placed in your own startup Directory (see below). The second file, *nrui.b*, contains the information to build the interface. It assigns menu names and members, tabs, buttons, slider settings, and all of the default settings when they are called up in the interface. The commands in this file can be copied and placed in your *ui Directory*. As with the *nreal.b* file, you should not modify the *nrui.b* file. Create your own preference files (.h files) to either add to or change Shake's performance.

**Note:** On Mac OS X, **Ctrl**+click / right-click the shake icon and select Show Package Contents to view the Shake package contents (includes the *nreal.b* and *nrui.b* files).

To add your own files, create the following directory subtree:

`<somewhere>/include/startup/ui`

“Somewhere” can be one or more of three possible locations (read by Shake in the order listed below):

- In the Shake Directory, *Contents/Resources/* and *Contents/Plugins/startup* (`<ShakeDir>/include/startup/ui` for non-Macintosh OS X installs). These directories are scanned every time Shake is launched by any user that is using Shake called from this directory.
- In any directory listed in the `$NR_INCLUDE_PATH` list. Set the `NR_INCLUDE_PATH` environment variable to point to a list of directories. This is usually done when sharing a large project among many users.

**Note:** For information on setting environment variables in Mac OS X, see “Environment Variables for Shake” on page 664.

Add the following line to a *.cshrc* or *.tcshrc* file in your `$HOME` directory:

```
setenv NR_INCLUDE_PATH " //Biggo/proj/include:/Documents/shake_settings/include"
```

Use the above for facility-wide or machine macros and settings that are read independently of who is using it. Because you can add multiple directories in the path list, you can have several locations of files.

- In the User Directory. This is for settings for your own personal use. Shake automatically scans the `$HOME/nreal/include` subtree.

Therefore, in `$HOME/nreal`, create the *include/startup/ui* subtree.

The User Directory can have the following subdirectories:

- *include/startup/ui*: For macros, machine settings, and interface settings.
- *settings*: For window layout settings.
- *icons*: For personal icons for the interface.
- *fonts*: For personal fonts.
- *autosave*: For scripts saved automatically every 60 seconds (by default) by Shake.
- You can also place macros inside of a script by copying and pasting with a text editor. This guarantees your macro is found by Shake when rendering. However, you do not have access to any interface-building functions. Additionally, if you load the script back into the interface and save it out again, the macros are lost.

Settings that change performance or add macros (see below) are located in *<somewhere>/include/startup*, and have a .h file extension, for example:

```
/Users/me/nreal/include/startup/memory_settings.b
```

This is referred to as the *startup* directory. Files in this location are referred to as *startup* .h files.

Settings that change the interface in some way (see below) are usually located in *<somewhere>/include/startup/ui*, and have a .h file extension, for example:

```
/Users/me/nreal/include/startup/ui/slider_settings.b
```

This is referred to as the *ui* directory or sometimes *startup/ui* directory. Files inside it are referred to as *ui* .h files.

Files that change additional default settings or add extra controls are in the templates directory, which is always under a *ui* directory:

```
/Users/me/nreal/include/startup/ui/templates/defaultfilter.b
```

All of these files can have any name, except “nreal.h” or “nrui.h,” which are reserved files used by Shake as the standard list of functions and settings. Also, they must have the .h extension, so a fast way to disable a file is to remove the extension.

Within a *startup* or *startup/ui* directory, files are loaded in no specific order. If it is important that a file is loaded before another file, this can be accomplished by one of the following:

- Add an *include* statement at the beginning of the file. For example, if *macros.b* relies on *common.b* being loaded before, start *macros.b* with  
*#include <common.b>*
- Put all the files you want to load in a directory (for example *include/mystuff*) and create a .h file in *startup* that contains only *include* statements:  
*#include <mystuff/loadmefirst.b>*  
*#include <mystuff/nowloadtbis.b>*  
*#include <mystuff/andtbistoo.b>*

Include files are never loaded twice, so it is okay if two .h files contain the same *#include <somefile.h>* statement.

If these files do not appear to be working, refer to the following checklist:

- Does the file have a .h extension?
- Is the file in a *startup* directory in one of the three possible locations (as described above)?
- If using a tcsh, and the file is in what you think is the NR\_INCLUDE\_PATH, is NR\_INCLUDE\_PATH actually set for that window? To test this, type the following in a tcsh window:  
*echo \$NR\_INCLUDE\_PATH*
- Have you checked the text window from which you launched Shake? This is where syntax problems are shown.

### Icons Search Path

Just as you can create preference files, you can create your own icons. The description of the actual icons can be found in “Using the Alternative Icons” on page 645. Icons can be found in one of three locations:

- *<ShakeDir>/Contents/Resources/icons* (*<ShakeDir>/icons* on non MacOS)—a directory, not to be confused with the important *icons.pak* file)
- *<UserDirectory>/icons* (see above)
- In any directory pointed to by \$NR\_ICON\_PATH, set the same way \$NR\_INCLUDE\_PATH is set

## General Settings

The // (forward slashes) indicate that line is commented out and inactive.

### Color Settings

#### Setting Tab Colors

In the *ui* directory:

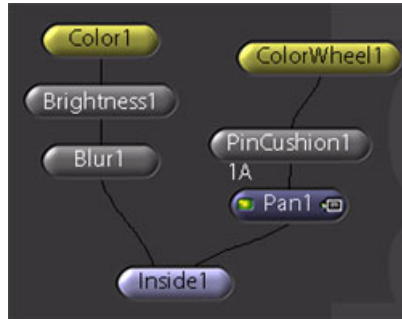


```
nuiPushToolBox( "Image" );  
  
nuiSetObjectColor( "ImageToolBox",  
    tabRed, tabGreen, tabBlue,  
    textRed, textGreen, textBlue);  
  
nuiPopToolBox( );
```

This is an excerpt from the *include/nrui.b* file. The Image tab is opened and assigned a color for both the tab and the text on the tab. Instead of numbers for the color values, variables are used here to indicate the parameter. Search for the variable names above or enter your own explicit values. Doing this does not automatically assign color to nodes within the tab.

### Setting Colors for the Nodes in the Node View

In the *ui* directory:



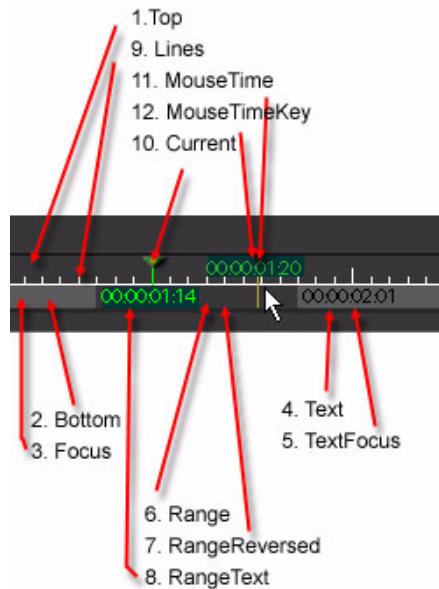
```
nuiSetMultipleObjectsColor(
    nodeRed, nodeGreen, nodeBlue,
    textRed, textGreen, textBlue,

    "DisplaceX",
    "IDisplace",
    "PinCushion",
    "Randomize",
    "Turbulate",
    "Twirl",
    "WarpX"
);
```

This command assigns colors to nodes in the Node View. The *nodeRed*, *green*, etc., and *textRed*, *green*, etc., are supposed to be float values. When coloring the nodes, keep in mind that the default artwork is a medium gray, so you can have numbers above 1 for the node color parameters to multiply it up.

## Setting Colors for the Time Bar

In the *ui* directory:

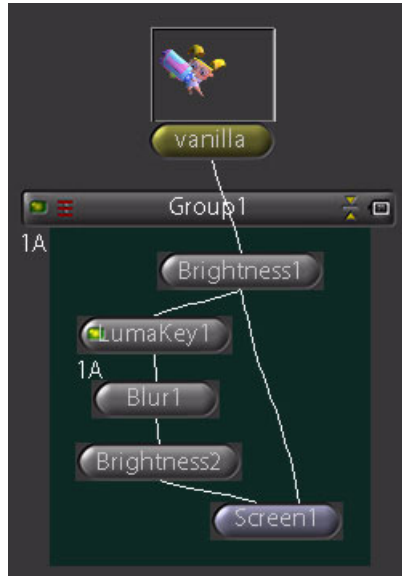


```
gui.color.timeSliderTop = 0x373737FF;  
gui.color.timeSliderBottom = 0x4B4B4BFF;  
gui.color.timeSliderFocus = 0x5B5B5BFF;  
gui.color.timeSliderText = 0x0A0A0AFF;  
gui.color.timeSliderTextFocus = 0x000000FF;  
gui.color.timeSliderRange = 0x373737FF;  
gui.color.timeSliderRangeReversed = 0x505037FF;  
gui.color.timeSliderRangeText = 0x0A0A0AFF;  
gui.color.timeSliderLines = 0xFFFFFFFF;  
gui.color.timeSliderCurrent = 0x00FF00FF;  
gui.color.timeSliderMouseTime=0xACAC33FF;  
gui.color.timeSliderMouseTimeKey=0xFCFC65FF;
```

These are just a few plugs to change the coloring of the text in all time-based windows, such as the Curve Editor, Time Bar, and so on. The numbers are, obviously, in hex just to make things more difficult. Ignore the 0x and the last FFs. Note you often have control over a basic color and its mouse-focused variation.

## Setting Colors for Groups in the Node View

In the *ui* directory:



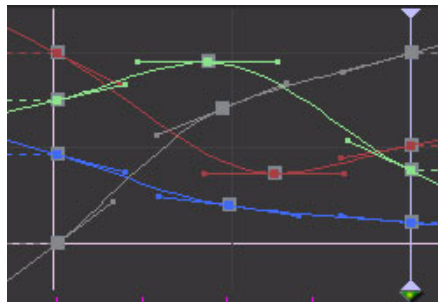
```
nuiSetObjectColor("Group", .75, .75, .75);
```

This sets the color of collapsed groups. If you set them to 1, the group takes on the color set in the Group's color setting:

```
nuiSetObjectColor("Group", 1., 1., 1.);
```

## Setting Colors for the Curves in the Editor

In the *ui* directory:



```
gui.color.curveDef = 0x658a61;  
gui.color.curveDefFoc = 0xcccc26;
```

```

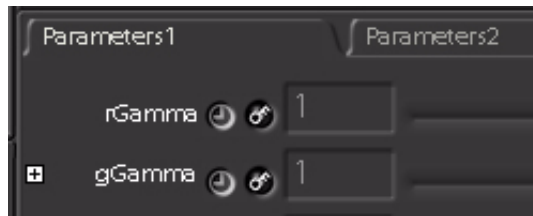
gui.color.curveDefSel = 0xcccc26;
gui.color.curveDefFocSel = 0xffff26;
//Curves starting with 'r' or 'R'
gui.color.curveR = 0xa74044;
gui.color.curveRFoc = 0xff0000;
gui.color.curveRSel = 0xff0000;
gui.color.curveRFocSel = 0xff8888;
//Curves starting with 'g' or 'G'
gui.color.curveG = 0x8de48d;
gui.color.curveGFoc = 0x00ff00;
gui.color.curveGSel = 0x00ff00;
gui.color.curveGFocSel = 0xaaffaa;
//Curves starting with 'b' or 'B'
gui.color.curveB = 0x406bf7;
gui.color.curveBFoc = 0x1818ff;
gui.color.curveBSel = 0x1818ff;
gui.color.curveBFocSel = 0x8888ff;
//Curves starting with 'a' or 'A'
gui.color.curveA = 0x888888;
gui.color.curveAFoc = 0xbbbbbbb;
gui.color.curveASel = 0xbbbbbbb;
gui.color.curveAFocSel = 0xeeeeee;

```

There are really only four basic curve types, the normal curve (*Def*), the focused curve (*DefFoc*), the selected curve (*DefSel*), and the focused, selected curve (*DefFocSel*). You then also have additional controls over curves that start with the letters r, g, b, and a.

### Setting Colors for Text

In the *ui* directory:



```

gui.fontColor = 0xFFFFFFFF;
gui.textField.fontColor = 0xFFFFFFFF;
gui.textField.tempKeyBackClr = 0xFFFFFFFF;

//the color of text on an active tab
gui.tabber.activeTint.red = .9;

```

```

gui.tabber.activeTint.green = .9;
gui.tabber.activeTint.blue = .87;

//the color of text on an inactive tab
gui.tabber.tint.red = .65;
gui.tabber.tint.green = .65;
gui.tabber.tint.blue = .63;

```

This colors the text in hexadecimal. There is a series of expressions near the very end of the *nrui.b* file that allows you to put in normalized RGB values that are then fed into the hex number, but you can also determine your color using the Color Picker.

*fontColor* = the color of the actual parameter name, messages, and also of macros without declared coloring.

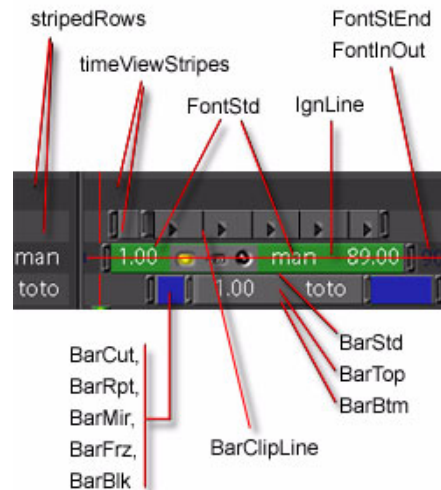
*textfield.fontColor* = the color of the values within the textfield.

*tempKeyBackClr* = a warning color for values entered when not in autosave mode for animated parameters. The value is not saved until autokey is enabled.

Some text colors can also be interactively modified in the *Global's* settings. These are saved into *<UserDir>/nreal/settings* when you choose File > Save Interface Settings.

### Setting Time View Colors

In the *startup* or *ui* directory:



```

gui.color.timeViewBarStd = 0x737373;
gui.color.timeViewBarTop = 0x909090;
gui.color.timeViewBarBtm = 0x303030;

```

```

gui.color.timeViewBarCut = 0x101010;
gui.color.timeViewBarRpt = 0x5a5a5a;
gui.color.timeViewBarMir = 0x5a5a5a;
gui.color.timeViewBarFrz = 0x424242;
gui.color.timeViewBarBlk = 0x0;
gui.color.timeViewBarClpLine = 0x0;
gui.color.timeViewFontInOut = 0x111144;
gui.color.timeViewFontStEnd = 0x441111;
gui.color.timeViewFontStd = 0xFFFFFFFF;
gui.color.timeViewIgnLine = 0xFF0000;
gui.color.strippedRowColLight = 0x373737;
gui.color.strippedRowColDark = 0x474747;
gui.color.timeViewDarkStripe = 0x373737;
gui.color.timeViewLightStripe = 0x474747;

```

The *BarCut*, *BarRpt*, *BarMir*, *BarFrz*, and *BarBlk* refer to the repeat modes, so each one has a different color.

### Creating a Custom Palette

In the *ui* directory:



```

nuiSetColor(1,1,0,0);
nuiSetColor(2,1,0.5,0);
nuiSetColor(3,1,1,0);
etc

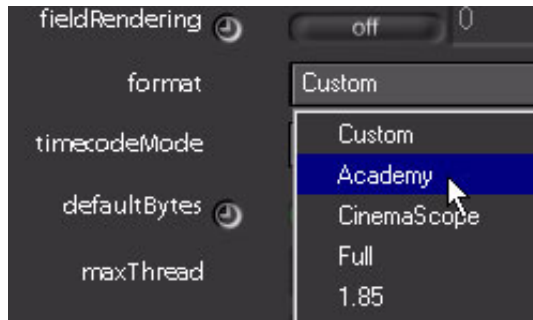
```

This assigns default colors to the palette icons, with the first number as the button number.

## Format

### Creating Custom Listings for the Format Pop-Up Menu

In the *startup* directory:



```
DefFormatType(  
    "string",  
    width,  
    height,  
    aspectRatio,  
    viewerAspectRatio,  
    framesPerSecond,  
    fieldRendering  
);  
  
DefFormatType(  
    "Academy", 1828, 1556, 1,1,24,"24 FPS"  
);
```

To set the default format choice whenever you launch Shake, you can use:

```
script.format = "FormatName";
```

### Automatically Assigning Default Width and Height to a Parameter in a Macro

In either *startup* or *ui* (typically inside of a macro's parameter setting):

```
image MyGenerator(  
    int width=GetDefaultWidth(),  
    int height=GetDefaultHeight(),  
    float aspectRatio=GetDefaultAspect(),  
    int bytes = GetDefaultBytes()  
)
```

These four commands check the default global settings and return the value at the time of node creation; they are not dynamically linked. Therefore, if you change the default parameters, the node's values do not change.

### Setting Format Defaults

In the *startup* directory:

```
script.defaultWidth = 720;  
script.defaultHeight = 486;  
script.defaultAspect = 1;  
script.defaultBytes = 1;  
script.format = "Full";
```

Using the *script.format* overrides the other settings—you either set the first four or the format settings, as shown above.

### Setting Maximum Viewer Resolution in the Interface

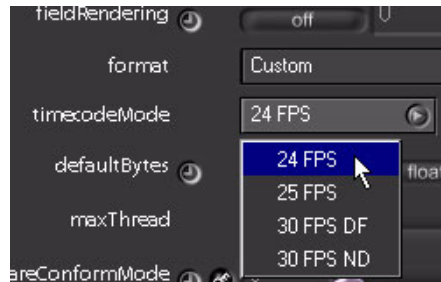
In the *ui* directory:

```
gui.viewer.maxWidth = 4096;  
gui.viewer.maxHeight = 4096;
```

By default, Shake protects the user from test rendering an enormous image by limiting the resolution of the Viewer to 4K. If the user accidentally puts a *Zoom* set to 200 on the composite, it does not try to render an enormous file, but instead only renders the lower-left corner of the image cropped at 4K. To change this behavior, set a higher or lower pixel resolution. These assignments have no effect on files written to disk.

### Creating Custom Listings for the Format Pop-Up Menu

In the *startup* directory:



```
DefTimecodeMode(  
    "Name",  
    fps,  
    numFramesToDrop,  
    numSecondsDropIntervals,
```

```
numSecondsDropException  
);
```

```
DefTimecodeMode("24 FPS", 24);  
DefTimecodeMode("30 FPS DF", 30, 2, 60, 600);  
These define the timecode modes for the Globals pop-up.  
To set the default timecodeMode, use:  
script.timecodeMode = "24 FPS";
```

### **Default Timecode Modes and Displays**

In the *startup* or *ui* directory:

```
script.framesPerSecond = 24;  
script.timecodeMode = "24 FPS";  
gui.timecodeDisplay = 0;
```

Set one or the other. Setting the *timecodeMode* allows you to use drop-frame settings. See above to set the Timecode Modes. The third line is to display the frames in the Curve Editor and Time Bar as frames or timecode. 1 = timecode; 0 = frames. The other timecode modes are: "25 FPS", "30 FPS DF", and "30 FPS ND".

## **Other Settings**

### **Autosave Frequency**

In the *ui* directory:

```
script.autoSaveDelay = 60;
```

This shows how often the autoSave script is done in seconds. The script is saved automatically in your User Directory as *autoSave1.shk*, *autoSave2.shk*, etc., up to *autoSave5.shk*. It then recycles back to 1. If you lose a script due to a crash, you can load in the autoSave script.

### **Undo Levels Amount**

In the *ui* directory:

```
gui.numUndoLevels= 100;
```

This shows how many steps of undo are available. Undo scripts are stored in the TEMP directory.

### **Amount of Processors to Assign to the Interface**

In the *ui* directory:

```
sys.maxThread = nrcGetAvailableProcessors();
```

This sets the number of processors when using the interface. The *nrcGetAvailableProcessors* automatically calculates and assigns all of them. If you only want to use a limited number of processors, assign that number here.

You can assign the number of processors to be used when batch processing with the *-cpus* flag. The default is 1. For example:

```
shake -exec my_script.shk -cpus 2
```

### Font Size for Menus and Pop-Up Menus

In the *startup* directory:

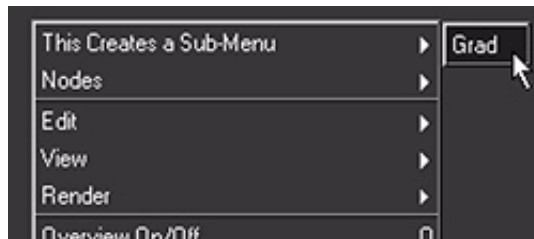


```
// It can take the following values:  
//tiny, small, medium, big, std  
gui.menu.fontSize= "std";
```

This should be in a *ui* .h file, but it must be set before the interface is built, so it goes in a *startup* file. The example is "tiny". "std" is the default.

### Adding Functions to the Right-Click Menu

In the *ui* directory:

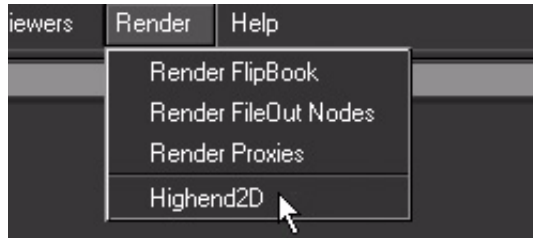


```
nuiPushMenu("NRiNodeViewPopup",1);  
nuiPushMenu(  
    "This Creates a Sub-Menu",0  
);  
nuiMenuItem(  
    "Grad",  
    nuiToolBoxMenuCall({{Grad()}})  
);  
nuiPopMenu();
```

This is an example that creates a subtab called “This Creates a Sub-Menu” in the Node View, and attaches the *Grad* function to its list. This is just one example. Take a look at the *nrui.b* file, where all right-click menus are built. The first line declares under what menu it is built, so typically these commands are added directly into the *nrui.b* file.

### Adding Functions Into a Menu

In the *ui* directory:



```
nuiOpenMenu( "Render" );
nuiMenuSeparator( );
nuiMenuItem(
    "Highend2D" ,
    LaunchBrowser(
        "http://www.highend2d.com" , 1
    )
);
nuiPopupMenu( );
```

This creates an entry in the Render menu, split from the other entries by a separator.

### Opening Scripts with Missing Macros

If you open a Shake script that contains a macro(s) that you do not have on your system, you have the option to load the script using a substitute node, or to not load the script at all using the *macroCheck* parameter in the Globals tab. To set the default *macroCheck* behavior to substitute a *MissingMacro* node, include the following in a .h file:

```
sys.useAltOnMissingFunc = 2
```

For information on the *macroCheck* parameter, see “The Global Parameters” on page 97.

### Memory and the Cache

Shake uses caching to speed up reprocessing. Caching saves a snapshot of certain nodes with all of the parameters that make up that node in a directory called the *cache directory*. When Shake processes any tree, it very quickly scans the directory to see if this exact node tree has been calculated before. If it has, it calls up the snapshot rather than recalculates the tree.

The cached images are labeled with input files, date of file creation, the node tree, and all of the parameters in the nodes, including resolution/proxy settings, to guarantee identical results. It is not accessible by the user.

The Shake cache is persistent. If you quit Shake, and then start the program the next day, it still calls up the files cached the previous day.

The following table lists the four settings for the cache.

Type	Setting	Function
None	0	No images are read from or written to the cache.
Read Only	1	Images are only read from the cache. No images are written to it.
Regular	2	Images are read from and written to the cache in a conservative algorithm. Only non-animated branches are cached.
Aggressive	3	Same as Regular, but animated branches are also cached. This results in a lot of cache activity, so make sure you have a large cache disk if you use this mode.

Shake also has transient caching, which is a temporary cache of viewed images. This means you can usually toggle between two images quickly. As you work on nodes that are not connected to the cached image, the cache is disposed.

The cache has a default maximum size of 512 MB. When files are written to the cache that push its size above the maximum limit, the oldest cached files are removed, allowing for new space. If you are using Aggressive caching, you should have a lot of disk space and RAM, otherwise you are constantly deleting old files and writing new ones, defeating the intended purpose of the cache.

By default, the cache is written to:

*/var/tmp/Shake/cache*

To clear disk space, you can remove this directory, but the caching information is lost. This is not vital, however, to the script. It simply forces Shake to completely rerender the compositing tree.

**Note:** Shake automatically creates the cache directories if they do not exist.

## Memory Settings

To manually declare your cache size, create a *startup* .h file. There are two main settings for the cache:

Function	What It Does
<code>cache.cacheMemory = 96;</code>	Controls how much memory is available to nodes when processing.
<code>diskCache.cacheMemory = 64;</code>	For caching images into memory.

The *cache.cacheMemory* setting works on a per-processor basis, and 96 MB is usually enough. For particularly complex scripts, you may want to bump this up. It is not necessary to increase this setting to a very large number to get better performance, since it controls the amount of memory Shake can use for processing—not for caching of frames (controlled by *diskCache.cacheMemory*). Processing memory is needed only by layer nodes and some operators that cannot work “in-place” (that is, require a temp buffer). However, most of these nodes grab only a small amount of memory. You can override this setting on the command line with the *-mem* setting.

**Note:** If you are working on a system with minimal RAM (for example, 256 MB), decrease the *cache.cacheMemory* setting to 64 MB.

On the other end, scaling *diskCache.cacheMemory* proportionally to the amount of memory available on your machine improves performance significantly, as more images are retained. Set disk cache to approximately 25 to 30 percent of your total memory, or enough to hold about four or five images in memory. A 2K, 4-channel float image takes about 51 MB of memory. There is a limit on what size a process can grow (2 GB on most operating systems). Past that point, the operating system either prevents the process from allocating memory, or just stops it.

The following table lists some additional functions for the *startup* .h files.

Function	What It Does
<code>diskCache.cacheLocation = "/var/tmp/Shake/cache";</code> <code>diskCache.cacheLocation = "C:/TEMP/Shake/cache";</code>	Set this to a location with a lot of unused space (and expect to use it all).
<code>diskCache.cacheMaxFile=2048;</code>	The maximum number of files in the cache before they are replaced. Least used, first out. It is recommended to leave this setting unchanged.

Function	What It Does
<code>diskCache.cacheMaxFileSize = 2048*2048*4*2;</code>	These values represent <i>width*height*channels*bytes</i> . This is probably the most difficult to calculate. You want to make sure that you can fit at the very least four to five images in memory cache. (The more images you can fit in memory, the better your interactivity.) If you want to cache 2048 x 1556 8-bit images, then you are looking at about 12.7 MB per image (derived from the formula above). Of course, some kind of balance needs to be found here with your working resolution, work habits, proxy settings, etc.
<code>diskCache.cacheSize</code>	The maximum size in MB of your cache directory.

### Caching Example

In this example, you have a dual-processor system with 1 GB of RAM and you want to leave at least 150 MB for operating system overhead.  $1024-150 = 874$  MB. A 2K image at float is  $2048 \text{ across} * 1556 \text{ high} * 4 \text{ channels} * 4 \text{ bytes-per-channel (float)} = 50,987,008$ . About 51 MB for float—16 bit is one-half of that. By default, the memory for the script is set to 96 MB per processor (`cache.cacheMemory = 96`). On a dual-processor machine this takes 192 MB.  $874-192 = 682$  MB. You want to have enough free RAM for 2 Viewer buffers at float 2K, so that's another 102 MB.  $682-102 = 580$  MB. You want to be able to keep at least 5 full 2K float images in the cache, that's  $5*51 = 255$  MB, so you set `diskcache.cacheMemory = 256`.  $580-256 = 324$  MB. This leaves 324 MB RAM left for Flipbooks and whatever else you may be doing. At 1/3 proxy (still at float, though), you are at about 6 MB per frame, so you can get about 80 frames into a Flipbook.

In short, give your favorite RAM manufacturer a call.

## The Curve Editor and Time Bar

### Setting the Time Bar Frame Range

In the *ui* directory:



```
gui.timeRangeMin=1;
gui.timeRangeMax=100;
```

That pretty much says it all, doesn't it?

## Default Timecode Modes and Displays

In the *startup* or *ui* directory:

```
script.framesPerSecond = 24;  
script.timecodeMode = "24 FPS";  
gui.timecodeDisplay = 0;
```

Set one or the other. Setting the *timecodeMode* allows you to use drop-frame settings. See above to set the Timecode Modes. The third line is to display the frames in the Curve Editor and Time Bar as frames or timecode. 1 = timecode; 0 = frames.

## File Path and Browser Controls

### Setting Default Browser Directories

In the *ui* directory:

```
gui.fileBrowser.lastImageDir= "C:/pix/" ;  
gui.fileBrowser.lastScriptDir= "$MYPROJ/shakeScripts/" ;  
gui.fileBrowser.lastExprDir= "//Server/shakeExpressions/" ;  
gui.fileBrowser.lastTrackerDir= "$MYPROJ/tracks/" ;  
gui.fileBrowser.lastAnyDir= "C:/Jojo/" ;
```

You can assign specific directories for the Browser to look in when you start the interface. You can assign different directories to different types of files, such as scripts, images, trackers, and expressions.

**Important:** There must be a slash at the end of the path.

### Using the UNC Filename Convention

In the *startup* directory:

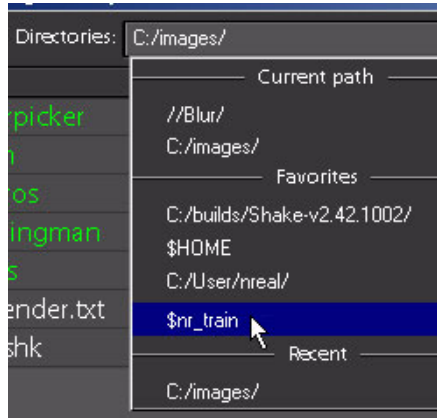


```
script.uncFileNames = 1;
```

Shake automatically assigns the UNC filename, that is, the entire file path name using the network address starting with *//MachineName//DriveName/path*. This ensures proper network rendering. However, you can turn this off by assigning the *uncFileNames* to 0, at which point local file paths are maintained. You can use local paths in either case, but they get converted when UNC is on.

### Adding Personal Favorites to the Browser


In the *ui* directory:



```
nuiFileBrowserAddFavorite(
    "D:/icons/scr/"
);

nuiFileBrowserAddFavorite(
    "$nr_train/"
);
```

All directories assigned here appear in your *Favorites* area of the *Directories* pop-up menu in the Browser for quick access.

To also bookmark a directory in the Browser, click Bookmark  and then choose File > Save Interface Settings. This saves a setting in your *\$HOME/nreal/settings* directory.

### Assigning a Browser Pop-Up Menu to a Parameter

In the *ui* directory:



```

nuxDefBrowseCtrl(
    "Macro.imageName",
    kImageIn
);

nuxDefBrowseCtrl(
    "Macro.imageName",
    kImageOut
);

nuxDefBrowseCtrl(
    "Macro.fileName",
    kAnyIn
);

nuxDefBrowseCtrl(
    "Macro.lookupFile",
    kExprIn
);

nuxDefBrowseCtrl(
    "Macro.scriptName",
    kScriptIn
);

nuxDefBrowseCtrl(
    "Macro.renderPath",
    kAnyOut
);

```

This assigns a folder button to a string so that you can relaunch the File Browser. The Browser remembers the last directories you used for any different type, so you can assign the type of file the Browser should look for as well with *kImageIn/Out*, etc. For example, if you have a macro that browses an image to be read in, use *kImageIn*, so when you click that button, it jumps to the last directory from which you read in an image.

- *kImageIn*: Directory of the last image input directory
- *kImageOut*: Directory of the last image output directory
- *kAnyIn*: Directory of the last input directory of any type
- *kAnyOut*: Directory of the last output directory of any type
- *kScriptIn*: Directory of the last script input directory
- *kScriptOut*: Directory of the last script output directory
- *kExprIn*: Directory of the last expression input directory
- *kExprOut*: Directory of the last expression output directory

## Automatic Launching of the Browser When Creating a Node

In the *ui* directory:

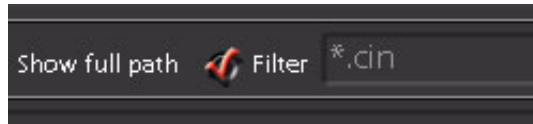
```
nuiToolBoxItem( "ProxyFileIn",
    {{
        const char *filename = getFileInName();
        filename ? ProxyFileIn(filename,0,2) :
        (image) 0
    }}
);
```

In this example, the Browser is called for the parameter filename in the *ProxyFileIn* macro. The macro has three parameters: Filename and two numbers (0 and 2). The *getFileInName* function automatically launches the Browser when the user creates this node in the interface. You can use:

- *getFileInName()*
- *getFileOutName()*
- *getScriptInName()*
- *getScriptOutName()*

## Automatic Browser File Filters

In the *ui* directory:



```
gui.fileBrowser.lastImageRegexp= "*.tif" ;
gui.fileBrowser.lastScriptRegexp= "*.shk" ;
gui.fileBrowser.lastExprRegexp= "*.txt" ;
gui.fileBrowser.lastTrackerRegexp= "*.txt";
gui.fileBrowser.lastAnyRegexp= "*";
```

You can assign specific filters for the Browser for different types of Browser activity. For example, if you only use Cineon files, you may want to use an assignment such as:

```
gui.fileBrowser.lastImageRegexp= "*.cin" ;
```

## Function Tabs

### Setting the Number of Node Columns in a Tab

In the `<ShakeDir>/include/nrui.h` or a *startup* file:



```
gui.doBoxColumns = 8;
```

This sets the number of columns for the nodes in the Tool Tab, which is sometimes called the “Do Box.” Unlike the other *ui.h* files, this must go in `<ShakeDir>/include/nrui.h`, placed right before the call to start building the Image tab. To activate it, uncomment the bold line in the *nrui.h* file:

```
//These control the color of text on an inactive tab

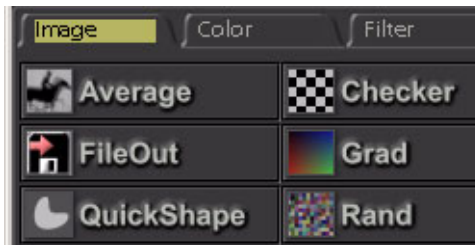
gui.tabber.tint.red = .65;
gui.tabber.tint.green = .65;
gui.tabber.tint.blue = .63;

//gui.doBoxAltFxIcons = 1;
//gui.doBoxColumns = 5;

nuiPushMenu("Tools");
nuiPushToolBox("Image");
    nuiToolBoxItem("Average", "const char *fileName = blah
    nuiToolBoxItem("Checker", Checker());
    nuiToolBoxItem("Color", Color());
    nuiToolBoxItem("ColorWheel", ColorWheel());
```

### Using the Alternative Icons

In *startup* or the `<ShakeDir>/include/nrui.h` file:



```
gui.doBoxAltFxIcons = 1;
gui.doBoxColumns = 8;
```

This calls the alternative icon set, which concentrates more on the name of the function. The alternative icons are stored in *icons/fxAlt* with the same name as the normal icons set, for example, *Image.Average.nri*, etc. The dimensions for these icons are 130 x 26. Because they are wider, you typically limit the columns to 5 in a normal Shake environment. For a macro on generating these icons, see the “Cookbook” section of the *Shake 3 Tutorials*. You can activate the icons in two places, either a *startup* file, or by uncommenting the following two bold lines in the *nrui.h* file:

```
//These control the color of text on an inactive tab

gui.tabber.tint.red = .65;
gui.tabber.tint.green = .65;
gui.tabber.tint.blue = .63;
//gui.doBoxAltFxIcons = 1;
//gui.doBoxColumns = 5;

nuiPushMenu("Tools");
nuiPushToolBox("Image");
    nuiToolBoxItem("Average", " const char *fileName = blah blah blah
    nuiToolBoxItem("Checker", Checker());
...

```

### Attaching a Function to a Button in the Tabs

In the *ui* directory:



```
nuiPushToolBox("Image");
nuiToolBoxItem("Flock", Flock(0,0,0));
nuiPopToolBox();
```

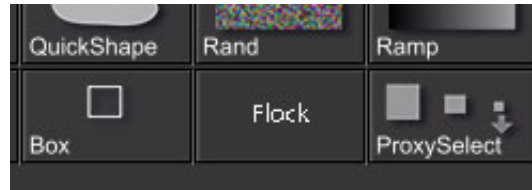
This places an icon that you have created into a tab that you assign. In this example, the icon is placed in the Image tab. If you use a custom name, such as *My\_Macros*, it creates that tab. The second line first attaches the icon, and then assigns the function with its default arguments to that button. They do not have to be the same name, but both are case sensitive. The icon is either found in *<ShakeDirectory>/icons*, your *<UserDirectory>/icons*, or in any directory pointed to with *\$NR\_ICON\_PATH*. The icons have the following characteristics:

- Although they can be any size, the standard resolution is 75 x 40 pixels.
- Do not use an alpha channel. Assign a *SetAlpha* (set to 0) or *Reorder* (set to rgbn) to remove the alpha channel.
- The filename is *TabName.Whatever.nri*. This example is therefore called *Image.Flock.nri*.
- The icon border is added automatically by Shake.

The section that says *Flock(0,0,0)* is the function of what that button actually does. You can assign any function to these—read in scripts, call multiple nodes, etc. If the function does not have default values for its parameters, they must be provided here.

### Attaching a Function to a Button Without an Icon

In the *ui* directory:

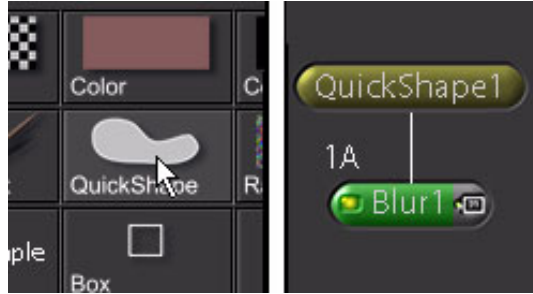


```
nuiPushToolBox( " Image" );
nuiToolBoxItem( "@Flock", Flock(0,0,0) );
nuiPopToolBox();
```

Note the *@* sign before the icon name. This creates a button with whatever text you supply.

## Creating Multiple Nodes With One Function

In the *ui* directory:



```
nuiToolBoxItem(  
    "QuickShape",  
    Blur(QuickShape())  
);
```

You can create multiple nodes with one button click when you call up a function. For example, if you always attach a *Blur* node to a *QuickShape*, you can do this by embedding one function within another. The first argument for a function is usually the image input. By substituting the value (usually 0) with a different function, that function feeds into your first function. In the above example, *QuickShape* is fed into *Blur*.

## Light Hardware Mode

In the *ui* directory:

```
sys.hardwareType = 1;
```

This command opens Shake without any borders on buttons, making the interactivity a little faster for graphically slower machines. A value of 0 is the normal mode; a value of 1 is the lightweight mode. Note the artwork isn't updated for the darker interface, so it looks a bit odd.

## Node View

### Setting Default Node View Zoom Level

In the *ui* directory:



```
gui.nodeView.defaultZoom=1;
```

These are plugs specifically for the Time View. They use the same hex syntax as the other color plugs.

## Parameters Tab

These are commands typically assigned to help lay out your macros by setting slider ranges, assigning buttons, etc. These behaviors are typically assigned to specific parameters. They can be applied either globally (all occurrences of those parameters) or to a specific function. For example, if there is a triplet of parameters named red, green, blue, Shake automatically assigns a Color Picker to it. However, for a parameter such as depth, you want to specify actions based on whether it is a bit-depth related function (and therefore assign a button choice of 8, 16, or float bit depth) or a Z-depth related function (in which case you probably want some sort of slider). To assign it to a specific function, preface the parameter name with the function name, such as *MyFunction.depth*.

All parameters, unless overridden by Shake's factory-installed rules, are assigned a slider with a range of 0 to 1.

### Assigning a Color Picker

In the *ui* directory:



```

nuiPushControlGroup( "Color" );
    nuiGroupControl( "Func.red" );
    nuiGroupControl( "Func.green" );
    nuiGroupControl( "Func.blue" );
nuiPopControlGroup();
nuiPushControlWidget(
    "Color",
    nuiConnectColorTriplet(
        kRGBToggle,
        kCurrentColor,
        1
    )
);

```

This assigns a button to three sliders so that you can scrub across an image and retrieve color information. You can select the current color, the average color, the minimum color, or the maximum color values. You can also assign a toggle switch to select the input node's color or the current node's color. For example, for pulling keys, you probably want to use the input node color since you are scrubbing (usually) blue pixels, rather than the keyed pixels. You can also choose to return different color spaces other than RGB. Assigning a Color Picker creates a subtree of those parameters.

Notice that you must first group the parameters into a subtree (the first five lines of the above example).

Color Pickers automatically appear if you name your triplet *red*, *green*, *blue* or *red1*, *green1*, *blue1* or *red2*, *green2*, *blue2*.

There are three parameters for the *nuiConnectColorPCtrl* function. The first one is the color space, which can be declared with either a string (for clarity) or an integer:

```

kRGBToggle    0
kHSVToggle    1
kHLSSToggle    2
kCMYToggle    3

```

The second parameter describes the type of value to be scrubbed—the current, average, minimum, or maximum. Again, you can use either the word or the integer.

```

kCurrentColor  0
kAverageColor  1
kMinColor      2
kMaxColor      3

```

The last parameter is a toggle to declare whether you use the current node's pixel values or the input node's pixel values. You use either 0 or 1:

0 = current node

1 = input node

Use of the current node may possibly cause a feedback loop. Typically, for color corrections you use current node, for keyers, the input node.

Therefore, the above example creates a subtree called *Color* for the function called *MyFunction*. The scrubber returns RGB values, of which only the current value is returned. When the Color Picker is called, the Use Source Buffer is turned on.

### Assigning the Old Color Picker

In the *ui* directory:



```
nuiPushControlGroup("Func.Color");
    nuiGroupControl("Func.red");
    nuiGroupControl("Func.green");
    nuiGroupControl("Func.blue");
nuiPopControlGroup();
nuiPushControlWidget(
    "MyFunction.Color",
    nuiConnectColorPCtrl(
        kRGBToggle,
        kCurrentColor,
        1
    )
);
```

This is an older version of the Color Picker without the cool extra controls.

### Changing Default Values

In the `<ShakeDir>/include/nrui.b` file:

```
nuiPushToolBox("Color");
    nuiToolBoxItem("Add", Add(0,0,0,0,0,0));
```

```

nuiToolBoxItem("AdjustHSV", AdjustHSV());
nuiToolBoxItem("Brightness", Brightness(0,1));
nuiToolBoxItem("Clamp", Clamp());
nuiToolBoxItem("ColorCorrect", ColorCorrect());
...

```

In the *include/nreal.b* file, most functions have their default values declared, but not all of them. To override the default values when you call the function, modify the line that loads the function in the interface. If every parameter in a function has a default value in a function, you can call the function with something like:

```
nuiToolBoxItem("Clamp", Clamp());
```

Normally, *Clamp* has about 8 values. Here, the 0 represents in the first argument, the input image. 0 is used to indicate that no images are expected to be inserted, so you can attach it to the active node. However, you can add additional parameters. For example, the *Brightness* line above it has (0,1), 0 for the image input (no input) and 1 for the brightness value. Change the 1 to a different value to override it. You only need to supply the parameters up to the one you want. For example, the following is the call for the *Text* function:

```
nuiToolBoxItem("Text", Text());
```

To override the default font for the *Text* function, you have to supply the width, height, bytes, text, and finally the font. The rest you can ignore afterward:

```

nuiToolBoxItem("Text", Text(
    GetDefaultWidth(),
    GetDefaultHeight(),
    GetDefaultBytes(),
    "Yadda Yadda",
    "Courier"
))
;

```

## Grouping Parameters in a Subtree

In the *ui* directory:



```
nuiPushControlGroup("Func.timeRange");
    nuiGroupControl("Func.inPoint");
    nuiGroupControl("Func.outPoint");
    nuiGroupControl("Func.timeShift");
    nuiGroupControl("Func.inMode");
    nuiGroupControl("Func.outMode");
```

```
nuiPopControlGroup();
```

This groups parameters into a subtree that can be opened and closed by the user. This example, although it says "Func", is for the *FileIn* node.

## Setting Slider Ranges

In the *ui* directory:



```
nuiDefSlider(
    "Func.yPan", 0, height
);

nuiDefSlider(
    "Func.angle", -360, 360
);
```

```
nuiDefSlider(  
    "Funct.aspect", 0, 2, .2, .01  
);
```

Even though the sliders are in relatively the same position, there are different numbers in the text fields. You can set slider ranges and precision with this function. The first line assigns a slider range just for the `yPan` parameter of the *Move2D* function. Note the use of the `height` variable so the range adjusts according to the input image. The second line assigns a range for the `angle` parameter in any node. The third line also has optional precision parameters, which are *granularity* and *notch spacing*.

- *granularity* represents the truncation point. Shake cuts off all values smaller than your truncation value, that is, if your granularity is `.01`, a value of `.2344` becomes `.23`. Granularity is a “hard” snap—if granularity is set to `0.001`, you cannot get anything but multiples of `0.001` when you slide. Also, granularity cannot be anything but a multiple of `10` (+ or -).
- *notch spacing* represents at what value interval the magnets appear. A value of `.1` means the magnets are at `.1`, `.2`, `.3`, etc. The default value is `.1`. Notch spacing is a “soft” snap—the slider tends to stick longer to multiples of notch spacing, but does not prevent the selection of other values. Think of it as a tiny notch in a flat line where a ball rolls: The ball tends to get stuck in the notch, but if you keep pushing, it eventually gets out.

### Pop-Up Menus

In the *ui* directory:



```
nuxDefMultiChoice("Defocus.shape",  
    "fast gaussian|fast box|circle"  
);
```

This pop-up menu, from the *Defocus* function, allows you to use a pop-up menu for strings. Note this only supplies strings and not numbers, so you have to do some tricky math inside the macro itself. For more information, see “Script Manual” on page 692.

## Creating Radio Buttons

In the *ui* directory:



```
nuxDefRadioBtnCtrl(  
    "Text.xAlign",  
    1, 1, 0,  
    "1|ux/radio/radio_left",  
    "2|ux/radio/radio_center",  
    "3|ux/radio/radio_right"  
);
```

This example is for the *Text* node. This code creates a series of radio buttons that are mutually exclusive. The naming convention assumes that you have four icons for each name, with the icon names *name.on.nri*, *name.on.focus.nri*, *name.off.nri*, and *name.off.focus.nri*. If no icons exist, you can choose to not use icons, which then gives a label with an on/off radio button instead. The code has these parameters:

```
nuxDefRadioBtnCtrl(  
    const char *name,  
    int useIcon,  
    int useLabel,  
    int animatable,  
    curve string state0, ....  
);
```

You can place as many icons as you want. The height of Shake's standard parameters icons is 19 pixels, though this can change. The output parameter for *Primatte* and *Keylight* is a good example.

You can make your own radio buttons with the *RadioButton* function. This function is discussed in the “Cookbook” section of the *Shake 3 Tutorials*.

## Creating Push-Button Toggles

In the *ui* directory:

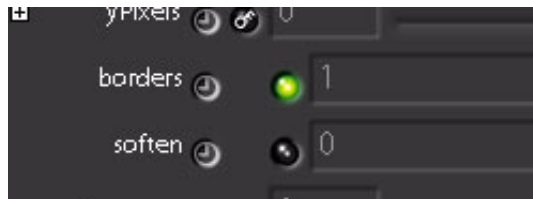


```
nuxDefExprToggle( "Func.parameter",  
    "repl.nri|repl.focus.nri",  
    "interp.nri|interp.focus.nri",  
    "blur.nri|blur.focus.nri"  
);
```

This assigns a series of buttons to toggle through integers starting at 0. The first line is assigned a value of 0, the second line assigned a value of 1, the third assigned a value of 2, etc. You can place as many toggles as you want. There are two buttons for each assignment, the normal button, and a second button for when the cursor passes over the button to signify that you can press it. Note the standard buttons are all in the subdirectory *ux*, but this is not a requirement. Shake includes a series of precreated icons that are packed into the *icons.pak* file and are inaccessible to the user, but are understood by this code. Your custom icons can be any size, but the default height is 19 pixels. You cannot have an alpha channel attached to an icon. Use *SetAlpha* (set to 0) or *Reorder* (set to *rgbn*) to remove the alpha channel. They can be placed in *<ShakeDirectory>/icons*, the *<UserDirectory>/icons*, or *\$NR\_ICON\_PATH*.

## Creating On/Off Buttons

In the *ui* directory:



```
nuxDefExprToggle( "Func.param" );
```

This is similar to the push-button toggles, but you only have two values, on and off—off a value of 0, and on a value of 1. The icon assignment is automatic.

## Making a Parameter Non-Animateable

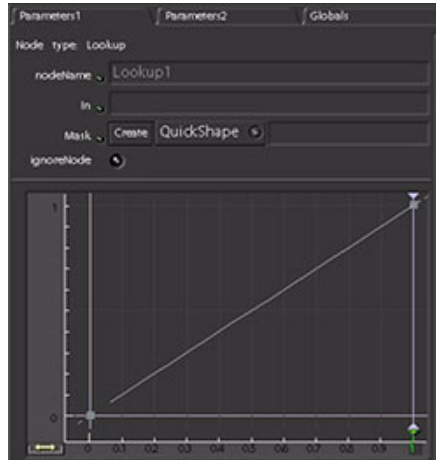
In the *ui* directory:

```
nriDefNoKeyPCtrl( "DilateErode.soften" );
```

This designates that no autokey buttons appear.

## Placing a Curve Editor Into a Parameter Tab

In the *ui* directory:



```
nuiPushControlGroup( "colorExpr" );
    nuiGroupControl( "Lookup.rExpr" );
    nuiGroupControl( "Lookup.gExpr" );
    nuiGroupControl( "Lookup.bExpr" );
    nuiGroupControl( "Lookup.aExpr" );
nuiPopControlGroup();

//Makes all curves invisible by default
registerCurveFunc( "colorExpr" );
//This makes all curves visible by default
registerCurveFuncVisible( "colorExpr" );

gui.colorCtrl.curveEditorDirection = 0;
//When it is 1, the layout is vertical
//When this equals 0, the layout is
//horizontal
```

This code loads a Curve Editor embedded inside the Parameter tab. The first six lines of code simply group parameters together. The last line then attaches the parameters to the Curve Editor embedded in the Parameter tab.

## Viewers

This section discusses Viewer settings and onscreen controls.

### Settings

#### Setting Maximum Viewer Resolution in the Interface

In the *ui* directory:

```
gui.viewer.maxWidth = 4096;  
gui.viewer.maxHeight = 4096;
```

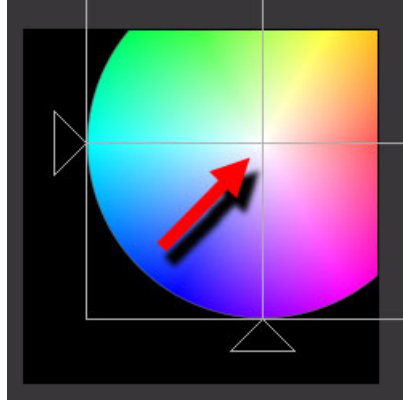
By default, Shake protects the user from test rendering an enormous image by limiting the resolution of the Viewer to 4K. If the user accidentally puts a *Zoom* set to 200 on the composite, it does not try to render an enormous file, but instead only renders the lower-left corner of the image cropped at 4K. To change this behavior, set a higher or lower pixel resolution. These assignments have no effect on files written to disk.

### Onscreen Controls

Onscreen controls are automatically built according to how you name your parameters in your macro, with one exception—to make a cross-hair control. The following is the list of parameters it takes to make certain controls. For the illustrations, the controls are attached to their appropriate functions. For example, the pan controls are attached to a *Pan* function and scaling to a *Scale* function. Simply naming the parameters does not necessarily give you the functionality you want.

### Panning Controls

In the *startup* macro file:

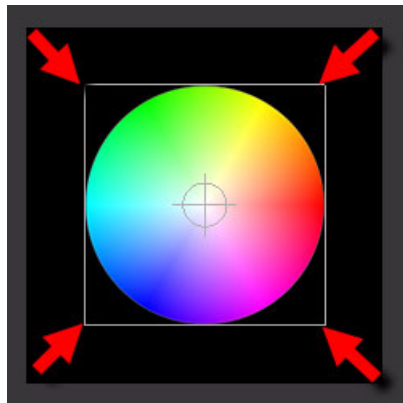


```
float xPan=0,  
float yPan=0
```

This gives you the lattice to pan around. You can grab anywhere on the cross bars.

### Scaling Controls

In the *startup* macro file:

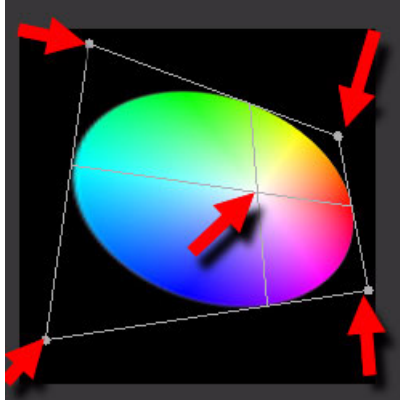


```
float xScale=1,  
float yScale=1,  
float xCenter=width/2,  
float yCenter=height/2
```

This gives you the border and center controls to change the scale center. You can grab on a corner to scale X and Y, or on an edge to scale X or Y.

## CornerPin Controls

In the *startup* macro file:



```
float x0=0,  
float y0=0,  
float x1=width,  
float y1=0,  
float x2=width,  
float y2=height,  
float x3=0,  
float y3=height
```

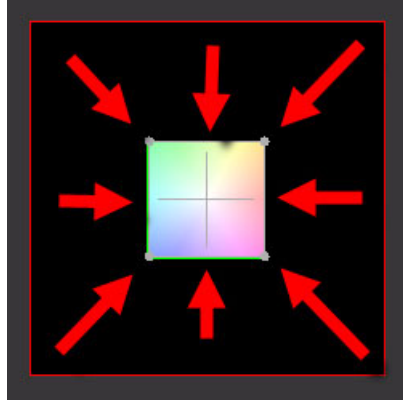
In the UI file:

```
nuiPushControlGroup("Func.Corner Controls");  
    nuiGroupControl("Func.x0");  
    nuiGroupControl("Func.y0");  
    nuiGroupControl("Func.x1");  
    nuiGroupControl("Func.y1");  
    nuiGroupControl("Func.x2");  
    nuiGroupControl("Func.y2");  
    nuiGroupControl("Func.x3");  
    nuiGroupControl("Func.y3");  
nuiPopControlGroup();
```

Grab any corner or the cross hair in the middle to adjust the position of your image. The grouping code for the UI file is included, so you do not have to look at all eight parameters in your list.

## Box Controls

In the *startup* macro file:



```
int left = width/3,  
int right = width*.66,  
int bottom = height/3,  
int top = height*.66
```

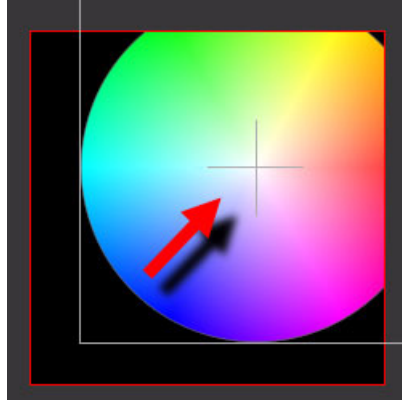
In the UI file:

```
nuiPushControlGroup("MyFunction.Box Controls");  
    nuiGroupControl("MyFunction.left");  
    nuiGroupControl("MyFunction.right");  
    nuiGroupControl("MyFunction.bottom");  
    nuiGroupControl("MyFunction.top");  
nuiPopControlGroup();
```

This creates a movable box. You can grab corners or edges, or the inside cross hair. This example is applied to a *SetDOD* function. *Layer-Constraint* and *Transform-Crop* also use these controls. In this example, integers are used for values that assume you are cutting off pixels, but you can also use float values.

## Offset Controls

In the *startup* macro file:

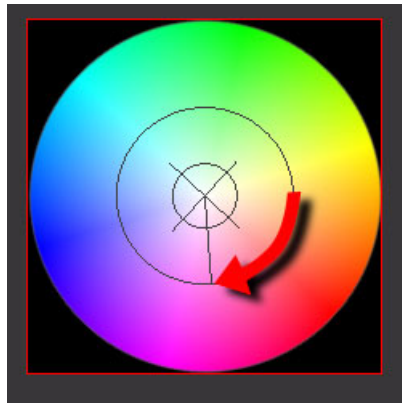


```
float xOffset=0,  
float yOffset=0
```

This is similar to the *Pan* controls, except you get a cross hair. This is used by the *Other-DropShadow* function.

## Rotate Controls

In the *startup* macro file:

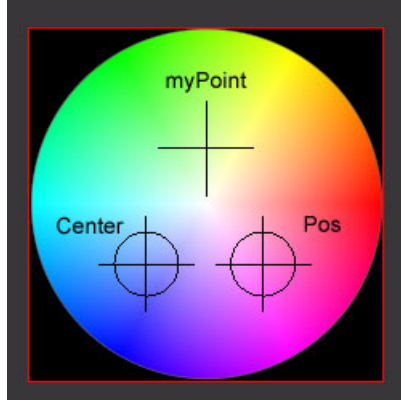


```
float angle = 0,  
float xCenter=width/2,  
float yCenter =height/2,
```

This gives you a rotation dial and a center control. This example is plugged into a *Rotate* function.

## Point Controls

In the *startup* macro file:



```
float xCenter = width*.33,  
float yCenter = height*.33,  
float xPos = width*.66,  
float yPos = height*.33,  
float myPointX = width/2,  
float myPointY = height*.66
```

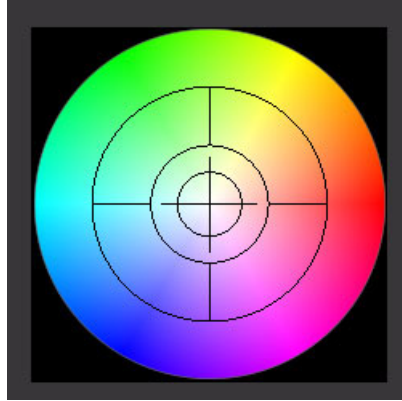
In the UI file:

```
nuiAddPointOsc( "Func.myPoint" );
```

These three sets of parameters create a cross hair control. *Center* and *Pos* are default names—the *Center* pair is also associated with the angle and the scale parameters. However, the last point is completely arbitrary, as long as it ends in an uppercase X and Y. In the *ui* file, you must also declare that these are an XY pair.

## Radius Controls

In the *startup* macro file:



```
float radius = width/6,  
float falloffRadius = width/6,  
float xCenter=width/2,  
float yCenter = height/2
```

This is basically for *RGrad*, but maybe you can do some more with it...

## Template Preference Files

You can add additional parameters and default settings by adding files into the *startup/ui/templates* directory. Each time Shake is launched, it adds these extra parameters. For example, if you always want the Proxy Filter to be “box” instead of “default,” and you always want a slider in the Globals tab called *switch1*, create a .h file under the templates directory with:

```
SetProxyFilter("box");  
curve int switch1=0;
```

Basically, take a saved script and strip out the lines you want to set as defaults, and save it as a .h file into *templates*.

## Environment Variables for Shake

This section discusses two ways to set environment variables, and the variables recognized by Shake. At the end of the section, some examples of aliases are provided.

**Warning** Incorrectly setting environment variables can lead to problems running Shake and with your operating system. If you are not comfortable with changing these types of settings, consult your system administrator for guidance.

Environment variables are strings of information, such as a specific hard drive, file name, or file path, set through a shell (for example, in Terminal on an OS X system) that is associated with a symbolic name (that you determine). This information is stored in a hidden file. Each time you launch Shake, the operating system and the Shake application look at the hidden file to set the environment variables. In other words, defining environment variables is the equivalent of setting user-defined system preferences.

As a simple example, you can set an environment variable that specifies a folder that Shake scans (on launch) for additional fonts used by the *Text* or *AddText* nodes.

To set environment variables on a Macintosh OS X system, create and edit a “.plist,” or property list, file. Using the *.plist* sets variables for Shake whether it is launched from the Terminal or from the Shake icon.

Using the above example of a font folder, to instruct Shake to read the */System/Library/Fonts* folder, set the following environment variable in your *.plist* file:

```
<key>NR_FONT_PATH</key>
<string>/System/Library/Fonts</string>
```

Another way to define environment variables is to use the *setenv* command in a *.tcshrc* (enhanced C shell resource) file. Each time the Terminal is launched, the *.tcshrc* file is read. The environment variables defined by the *.tcshrc* file are only read by Shake when launched from the Terminal.

Using the above example of a font folder, to instruct Shake to read the */System/Library/Fonts* folder, set the following environment variable in your *.tcshrc* file:

```
setenv NR_FONT_PATH /System/Library/Fonts
```

**Note:** The *.tcshrc* file can be used on all Shake platforms (OS X, Linux, and IRIX).

Common uses for a user’s personal *.plist* or *.tcshrc* file include:

- Define commonly-used aliases for commands. As a simple example, you can set an environment variable to launch Shake from the Terminal.

An alias in the command line is not the same as an alias on the Macintosh operating system. In the OS, an alias merely points to another file. In the command line, you create an alias to assign your own name to a command.

**Note:** If you do not have environment variables set on your OS X system, you can still launch Shake from the Terminal by typing complete path to Shake:

```
/Applications/Shake3/shake.app/Contents/MacOS/shake
```

To easily set the Shake path in the Terminal, do the following:

- Launch Terminal.
- In the Finder, navigate to the Shake application (usually located in the *Shake3* folder in the *Applications* folder).

- Drag the Shake icon to the Terminal.

The Shake path is automatically entered in the Terminal.

- Set environment variables for Shake. For example, you can specify the location of important files that your Shake script needs when opened.
- Specify the Shake directory.

### Creating the .plist Environment File

Each time you log in, the system searches for an environment file, named *environment.plist*. This file sets an environment for all processes (launched by the logged-in user). In the Terminal, you create a directory called *.MacOSX* that contains the environment file. You also create the environment file (using a text editor), and move the file into the *.MacOSX* directory.

#### To set environment variables in Macintosh OS X using the .plist file:

- 1 Log in using your personal login.

- 2 Launch Terminal.

By default, you should be in your Home (`$HOME`) directory. Your Home directory is your own directory in the Users folder. For example, if John Smith logs in and launches the Terminal, the following message is displayed in the Terminal:

```
john-smiths-Computer:~] john%
```

- 3 In the Terminal, type the following command to create a directory in your Home directory called *.MacOSX*:

```
mkdir $HOME/.MacOSX
```

and press **Return**.

An invisible directory (indicated by the “.” in front of the directory name) is created in your Home directory.

To ensure the *.MacOSX* directory was created, type:

```
ls -als
```

and press **Return**.

All files, including the new invisible *.MacOSX* directory, are listed.

- 4 Next, launch TextEdit (or other text editor) to create a file to set your variables.

**Note:** If you have installed and are familiar with the Apple Developer tools, you can use the Property List Editor application to create or edit variables. The Property List Editor application is located in *Developer/Applications*.

- 5 In the text document, create the following file (if reading this in the PDF version of the reference guide, you can copy the following and paste it into the text document) and edit the information.

**Note:** The following is an example file for instructional purposes only.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist SYSTEM "file://localhost/System/Library/DTDs/
    PropertyList.dtd">
<plist version="0.9">
<dict>
<key>MyProject</key>
<string>/Documents/MyBigFilm</string>
<key>NR_INCLUDE_PATH</key>
<string>/Documents/MyBigFilm/macros</string>
<key>NR_ICON_PATH</key>
<string>/Documents/MyBigFilm/icons</string>
</dict>
</plist>
```

This sets the variable *MyProject* to */Documents/MyBigFilm*. This tells Shake that all files associated with *MyProject* (that could be a script, directory, etc.) are located in */Documents/MyBigFilm*. As a result, if you type *MyProject* in the browser, it returns */Documents/MyBigFilm*, and can then be set as a favorite. This file also sets the *NR\_INCLUDE\_PATH* (points to the directory or directories that you want Shake to scan for macros and personal machine or user interface settings), and *NR\_ICON\_PATH* (points to a directory that you can save your own icons for Shake functions).

- 6 In TextEdit, choose Format > Make Plain Text.

The document is converted to a *.txt* file.

- 7 Choose File > Save.

- 8 In the “Save as” field of the Untitled.txt window, enter:

*environment.plist*

Be sure to remove the *.txt* file extension.

- 9 Save the file to your Home directory (in TextEdit, select Home in the “Where” pop-up list), and click Save.

- 10 In the Save Plain Text window, click “Don’t append.”

The file is saved in your Home directory with the extension *.plist*.

- 11 Quit TextEdit.

In order for Shake and your system to access the environment variables, the *environment.plist* file must be saved to the *.MacOSX* directory (created in step 3).

- 12** To save the *environment.plist* file to your *.MacOSX* directory, move the file (using the Terminal) from your Home directory to the *.MacOSX* directory. In the Terminal, do the following:

- To ensure you are still in your Home directory, type the “present working directory” command:

```
pwd
```

Using the example from step 2, this should return:

```
/Users/john
```

- Enter the following:

```
mv environment.plist .MacOSX
```

The *environment.plist* file is moved into the *.MacOSX* directory.

- To confirm the *environment.plist* file is located in the *.MacOSX* directory, enter:

```
cd .MacOSX
```

This command moves you into the *.MacOSX* directory.

- Enter:

```
ls
```

The content of the *.MacOSX* directory, the *environment.plist*, is listed.

- 13** Log out and then log in again.

**To edit the .plist file:**

- In the Finder, choose Go > Go to Folder (**Command+Shift+G**).
- In the “Go to the folder:” text field, enter the path to the invisible *.MacOSX* folder:

```
/Users/john/.MacOSX
```

- Click Go.

The *environment.plist* file appears in the folder.

- Open the *.plist* file in TextEdit (or other text editor).
- Once your changes are made, choose File > Save.
- Quit TextEdit.

### Using the *.tcshrc* Environment File

You can also set environment variables (or aliases) using a *.tcshrc* file. Like the above *.plist* file example, you can create the *.tcshrc* file in a text editor, or directly in a shell using vi, pico, or other shell editor. Unlike the *.plist* file, however, you do not save the *.tcshrc* file to the *.MacOSX* directory. Instead, the *.tcshrc* file is saved into your Home (*\$HOME*) directory.

Usually, you define environment variables in *tsch* with the *setenv* command, for example:

```
setenv audio /Volumes/shared/footage/audio_files/
```

This variable instructs Shake to automatically look in */Volumes/shared/footage/audio\_files/* when you import an audio file into Shake.

At login, the default */etc/csh.cshrc*, followed by any *.tcsbrc* files in your login directory, is run. This sequence is repeated whenever a new *tsch* is spawned, for example, when you launch Terminal.

**Note:** As mentioned above, Shake only reads the *.tcsbrc* environment file when Shake is run from the Terminal (the file is not applied when Shake is launched from the application icon).

To add a variable for Terminal commands, enter the following formatting (edit to suit your own project) into *\$HOME/.cshrc* or *\$HOME/.tcsbrc*:

```
setenv NR_INCLUDE_PATH " //Biggo/proj/include:/Documents/
shake_settings/include"
```

The following is an example of *.tcsbrc* file for illustration purposes only:

```
setenv shake_dir /Applications/Shake3/shake.app/
Contents/MacOS/shake
setenv shk_demo /Documents/project_03
set path = (. $shake_dir $path)
setenv NR_INCLUDE_PATH /Documents/project_03
setenv NR_FONT_PATH /System/Library/Fonts
alias ese vi $HOME/.tcshrc
alias s. source $HOME/.tcshrc
alias lt ls -latr
alias ui cd $HOME/nreal/startup/ui
alias st cd $HOME/nreal/include/startup
alias shake $shake_dir
```

This file sets the Shake directory, as well as points to the directories that you want Shake to scan for macros, user interface settings, etc. (*/Documents/project\_03*), and fonts (*/System/Library/Fonts*).

**Note:** Alias definitions or environment variables saved in a *.tcsbrc* file, are read the next time you log in. To make the alias or environment variable effective immediately, update your alias definition by sourcing out *.tcsbrc*. Type the following:

```
source .tcshrc
```

To edit the *.tcshrc* file, use *pico* or *vi* (or other shell editor). For example. Once your changes are made, save the *.tcsbrc* file. Like the *.plist* example above, you can edit the *.tcsbrc* file in a text editor.

## Shake Variables

Shake recognizes the following variables:

- *shell variables*: The Shake Browser recognizes an environment variable, for example, *\$pix* in the Browser if Shake is run with that environment setting.
- *NR\_CINEON\_TOPDOWN*: When set, that is,  
*setenv NR\_CINEON\_TOPDOWN*  
Cineon frames are written in the slower top-down method for compatibility with other less protocol-observant software.
- *NR\_FONT\_PATH*: Points to a directory where you want Shake to scan for additional fonts used by the *Text/AddText* functions. Fonts can also be found in the *<UserDirectory>/* fonts directory.
- *NR\_ICON\_PATH*: Points to a directory where you can save your own icons for Shake functions. Icons can also be found in the *<ShakeDirectory>/icons* or the *<UserDirectory>/icons* directories.
- *NR\_INCLUDE\_PATH*: Points to the directory or directories that you want Shake to scan for macros and personal machine or UI settings. These directories should have *startup/ui* as subdirectories. For example:  
*setenv NR\_INCLUDE\_PATH /shots/show1/shake\_settings*  
should have */shots/show1/shake\_settings/include/startup/ui*
- *NR\_SHAKE\_LOCATION*: Points Shake to a nonstandard installation area for IRIX. Default installation is */usr/nreal/<ShakeDir>* (for IRIX).
- *NR\_TIFF\_TOPDOWN*: This is identical for *NR\_CINEON\_TOPDOWN*, except it applies to TIFF files.
- *TMPDIR*: Points to the directory you want to use as your temporary disk space directory.
- *NR\_GLLINFO*: Information is printed for Flipbooks.

### To Test Your Environment Variable

There is a simple way to test if your environment variable exists. In a Terminal, type *echo*, followed by the environment variable, for example:

```
echo $myproj
```

and the proper value should be returned.

## Alias

An alias is a pseudonym or shorthand for a command or series of commands, for example, a convenient macro for frequently used command or a series of commands. You can define as many aliases as you want (or, as many as you can remember) in a *.tcsbrc* file.

To see a current list of aliases, type the following in a shell:

```
alias
```

To start Shake from the Terminal window:

```
Alias shake /Applications/Shake3/shake.app/Contents/MacOS/shake
```

To determine how many users are currently working on the system:

```
Alias census 'who | wc -l'
```

To display the day of the week:

```
alias day date +"%A"
```

To display all Shake processes that are running:

```
alias howmany 'ps -aux | grep shake'
```

## Interface Devices and Styles

This section discusses considerations when using a Stylus, setting mouse behavior, using a two-monitor system, and setting the monitor resolution on an IRIX system.

### Using a Stylus

- In the Global Parameters, enable the parameter.
- Set the parameter virtualSliderSpeed to 0.

When virtualSliderMode is enabled, the left button always uses the virtual sliders when a text port is clicked. Normally, you have to press **Ctrl** and drag. Just drag the mouse left and right in the text field, and it increases and decreases beyond the normal slider limits.

**Note:** The stylus does not allow you to use your desk space the same way as with a mouse, so you have to enable virtualSliderMode.



### Left-Handed Users

If you are left-handed, set the mouse behavior within the operating system itself. On IRIX, the settings are located in the Desktop > Customize > Mouse page tab.

## Dual-Head Monitors

Choose View > Spawn Viewer Desktop to create a new Viewer window that floats above the normal Shake interface. You can then move the Viewer to a second monitor, clearing up space on the first for node editing operations. This only works when both monitors are driven by the same graphics card.

The following is a handy *ui Directory* command to automatically create the second Viewer Desktop:

```
gui.dualHead= 1;  
// This is an example of what you can do to open a second  
// viewer desktop on the other monitor.  
if(gui.dualHead) spawnViewerDesktop(1290,10,1260,960);
```

For information on using a broadcast video monitor, see “Using a Broadcast Monitor” on page 63.

## Setting the Monitor Resolution for an IRIX System

To set your resolution on an SGI system, use either *xsetmon* (a graphical interface), or use the command-line *setmon* (these are SGI specifications).

To set your resolution, use the command “setmon” in the command line, AS ROOT:

```
/usr/gfx/setmon -x 1600x1024_72
```

The allowed values depend on your graphics card. If you are using an Octane with MXI graphics, go to the */usr/gfx/ucode/MGRAS/vof/2RSS/* directory to see the allowed choices. In the case of the MXI Octane, you have:

1024x768_60.sdb	1280x1024_72.sdb
1024x768_60p.sdb	1280x1024_76.sdb
1024x768_72.sdb	1280x492_120s.sdb
1024x768_76.sdb	1280x960_30i.sdb
1024x768_96s.sdb	1600x1024_72.sdb
1024x768_96s_vs.sdb	1600x1200_60.sdb
1280x1024_49.sdb	1600x1200_60_32db.sdb
1280x1024_50.sdb	1920x1035_60.sdb
1280x1024_50_2f.sdb	1920x1080_60_32db_gdm24w.sdb
1280x1024_59.sdb	616x492_140os.sdb
1280x1024_60.sdb	640x480_60.sdb
1280x1024_60_2f.sdb	640x486_30i.sdb
1280x1024_60p.sdb	768x576_25i.sdb

So: */usr/gfx/setmon -x [settingFileWithoutExtension]*

In both cases, you must log out and log in again; no reboot is needed.

# Macros, Expressions, and Scripting

## Chapter Summary

- Making Macros
- Variables, Linking, and Expressions
- Functions, Variables, and Expressions
- Script Manual

## Making Macros

The MacroMaker is an interactive tool to help you create new nodes by combining previously existing nodes. Shake's scripting language is essentially C-like, so you have access to an entire programming language to manipulate your scripts. Because the MacroMaker cannot account for all possibilities, consider the MacroMaker a tool to help you get started in using Shake's language. Use the MacroMaker to build the initial files, and then modify these files to customize your macros.

**Note:** For online portions of this section, choose Help > Customizing Shake.

For more information, such as about macro structure, common errors when making macros, and several examples, see “Macros” on page 703. For a tutorial on making macros, see “Lesson Eight: Making a Macro” in the *Shake 3 Tutorials*.

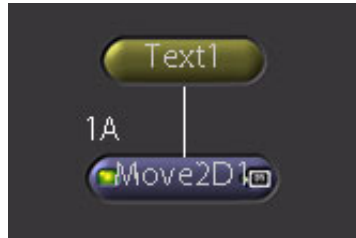
**Note:** If you open a Shake script that contains macros that you do not have on your system, you have the option to load the script using a substitute node, or to not load the script at all. For more information, see “The Global Parameters” on page 97.

## Creating the Node Structure

First, create the node structure for the function (what you want to occur in the node). This can be very simple or very complex. To help illustrate macro building, the following example creates a function that randomly scales, pans, and rotates your image, similar to *CameraShake*, but with more moving parameters.

### To create the node structure:

- 1 Click the Image tool tab, and click *Text*.  
The *Text* node is added to the Node View.
- 2 Click the Transform tool tab, and add a *Move2D* node.



- 3 Enter the following parameters:

Parameter	Value
<i>xPan</i>	$(\text{turbulence}(\text{time}, 2) - .5) * 20$
<i>yPan</i>	$(\text{turbulence}(\text{time} + 100, 2) - .5) * 20$
<i>angle</i>	$(\text{turbulence}(\text{time} + 200, 2) - .5) * 10$
<i>xScale</i>	$\text{turbulence}(\text{time} + 300, 2) / 2 + .75$

In these examples, the turbulence function generates continual noise between 0 and 1 (see “Expressions” on page 683). Since you do not want all fluctuations to have the same pattern, offset the seed each time you call the function (*time*, *time + 100*, *time + 200*, *time + 300*). Then, simple math is used to get the values in an appropriate range. For example, in *angle*, .5 is subtracted to adjust the value to a range of -.5 to .5. The result is then multiplied by 10, and yields a range between -5 and 5 degrees of rotation.

- 4 Click and drag on the Time Bar to test the animation.

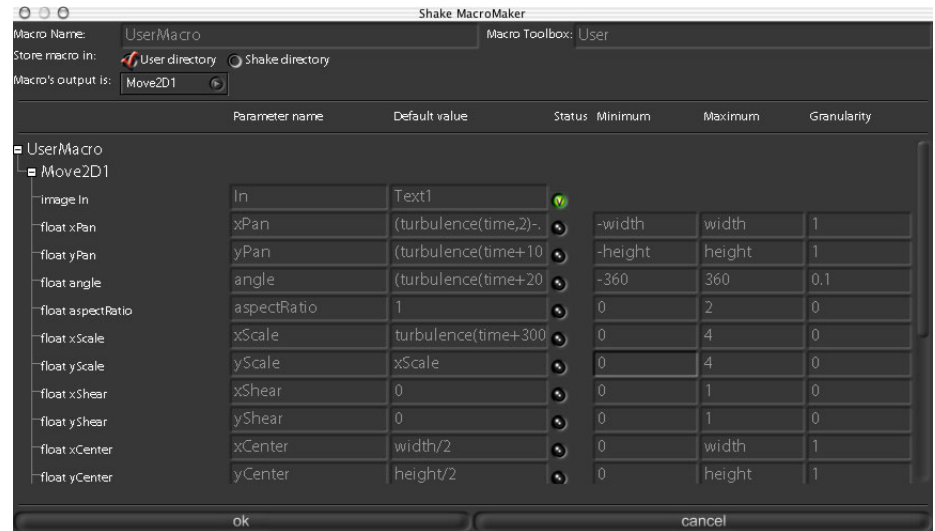
### Making the Macro

Since the above steps are tedious to manually recreate, create a macro.

### To create the macro:

- 1 In the node tree created above, select the *Move2D* node, and press **Shift+M** (or right-click and select Macro > Make Macro).

The MacroMaker is launched.



In the top portion of the Shake MacroMaker window, you specify the filename, the save location, and the tab where the node appears.

- 2 In the Macro Name field, enter *Random.Move*.
- 3 In the Macro Toolbox field, enter *Transform*.  
**Note:** Case sensitivity is always important.
- 4 Leave the *Store macro in* selection at the default *User directory* setting.
- 5 Leave the *Macro's output is* set to *Move2D1* (there are no other choices in this example).

Parameter	Value
<i>Macro Name</i>	The name of the macro you create. It is also the name of the file you save (see below).
<i>Macro Toolbox</i>	The tool tab that stores the macro. If a tab does not exist, it creates a new one.

Parameter	Value
<i>Store Macro in:</i>	<p>User directory: Saves the macro in your <code>&lt;UserDirectory&gt;/nreal/include/startup</code> as <code>MacroName.b</code> and a second ui file in <code>&lt;UserDirectory&gt;/nreal/include/startup/ui</code> as <code>MacroNameUI.b</code>.</p> <p>Shake Directory: Saves the macro in the Shake distribution directory, as <code>&lt;ShakeDir&gt;/include/startup/MacroName.b</code> and <code>&lt;ShakeDir&gt;/include/startup/ui/MacroNameUI.b</code>.</p> <p>For more information on these directories and their functions, see “Locations for .h Files and Custom Icons” on page 624.</p>
<i>Macro's output is:</i>	Presents a list of all nodes that are included in the macro (just one for the <i>Move2D</i> example). Select the node to pass out of the macro. Shake usually does a correct guess for this node if you have only one terminating branch.

The lower portion of the Shake MacroMaker window lists all of the nodes and the parameters that can be exposed in the created node. For example, since the image is fed into a *Move2D* node, the image In parameter is enabled.

Parameter	Value
<i>Parameter name</i>	The name of the slider (in the case of float and int values) or the knot input (in the case of image inputs). Arbitrarily renaming your parameters is not recommended (such as <i>eggHead</i> ), as you have the benefit of Shake's default GUI behaviors (pop-up menus, subtrees, color pickers, on/off switches, etc.) if they have the same name.
<i>Default value</i>	All current values of the nodes are fed into text fields. To set a default value for exposed parameters (that is, parameters with the visible Status light on), set the value here.
<i>Status</i>	Enable Status to expose this parameter. If you expose an image, it adds an input to the top of the node. If you expose anything else, it gives you a slider or text field in the Parameter View.
<i>Minimum</i>	For slider-based parameters, sets the lower slider limit.
<i>Maximum</i>	For slider-based parameters, sets the upper slider limit.
<i>Granularity</i>	Sets how much the slider jumps between values.

Since you already have most of the parameters needed in this example, you only want to expose *motionBlur*, *shutterTiming*, and *shutterOffset* values.

- 6 Click the V buttons for *motionBlur*, *shutterTiming*, and *shutterOffset*.



- 7 Click OK.

The new push-button node appears in the Transform tab.



- 8 Add the new node to the Node View.

In the *RandomMove* parameters, only the *motionBlur* settings are available, and have automatically collapsed into a subtree (due to the default Shake behavior).

#### To modify the macro:

- 1 If you placed your macro in your User Directory, go into your \$HOME directory, and then into the *nreal/include/startup* subdirectory.

In the startup directory, a new file called *RandomMove.b* appears.

- 2 Open the *RandomMove.b* file in a text editor:

```
image RandomMove(
    image In=0,
    float motionBlur=0,
    float shutterTiming=0.5,
    float shutterOffset=0
)
{
    Move2D1 = Move2D(
        In,
        (turbulence(time,2)-.5)*20,
        (turbulence(time+100,2)-.5)*20,
        (turbulence(time+200,2)-.5)*10,
        1,
        turbulence(time+300,2)/2+.75,
    )
}
```

```

        xScale,
        0, 0,
        width/2, height/2,
        "default", xFilter,
        "trsx", 0,
        motionBlur,
        shutterTiming,
        shutterOffset,
        0,

    );

    return Move2D1;
}

```

Edit the macro in this file. The parameters *motionBlur*, *shutterTiming*, and *shutterOffset* are declared in the first few lines, and then the assigned default values. The image *input In* is also assigned a value of 0, so there is no expected image input when the macro is created.

```

image RandomMove(
    image In=0,
    float motionBlur=0,
    float shutterTiming=0.5,
    float shutterOffset=0
)
...

```

- 3 Modify the behavior of the macro. Each turbulence function has a frequency of 2, for example, `turbulence(time,2)`. To create a new parameter to modify the frequency, add the following line (in bold):

```

image RandomMove(
    image In=0,
    float frequency = 2,
    float motionBlur=0,
    float shutterTiming=0.5,
    float shutterOffset=0
)
...

```

- 4 Now that you have a new parameter, you must substitute it into the macro body wherever you want that value to apply. Insert the variable into the macro body:

```

image RandomMove(
    image In=0,

```

```

float frequency = 2,
float motionBlur=0,
float shutterTiming=0.5,
float shutterOffset=0
)
{
    Move2D1 = Move2D(
        In,
        (turbulence(time,frequency)-.5)*20,
        (turbulence(time+100,frequency)-.5)*20,
        (turbulence(time+200,frequency)-.5)*10,
        1,
        turbulence(time+300,frequency)/2+.75,
        xScale,
        0, 0,
        ...

```

- 5 Save the file.
- 6 To see the results of the modification, restart Shake.

## Modifying the Macro Interface

The macro file in the startup directory merely creates the function. The interface is built by Shake each time it is launched. Therefore, the MacroMaker also creates a second file in the *startup/ui* subdirectory that creates a button and sets slider ranges for the node. For example, if you created the new frequency slider in the above example, you may have noticed that the slider only goes from 0 to 1. You can modify this in the ui file.

### To modify the macro interface:

- 1 In the text editor, open the *RandomMoveUI.b* file (created in the above example) in the *nreal/include/startup/ui* subdirectory.

```

nuiPushMenu( "Tools" );
    nuiPushToolBox( "Transform" );
        nuiToolBoxItem( "@RandomMove", RandomMove() );
    nuiPopToolBox();
nuiPopMenu();
nuiDefSlider( "RandomMove.motionBlur", 0, 1, 0, 0, 0 );
nuiDefSlider( "RandomMove.shutterTiming", 0, 2, 0, 0, 0.01 );
nuiDefSlider( "RandomMove.shutterOffset", -1, 1, 0, 0, 0.01 );

```

The first line opens the Tool Tabs. The second line opens the Transform tab. To place the macro in a different tab, change the word “Transform” to a different name. The third line creates the button. The first occurrence of the word “RandomMove” is preceded by an @ sign, indicating that there is no icon (not related to Shake’s unpadding frame wildcard) for the button, so therefore the text *RandomMove* should be printed on the button. The second listing of the word “RandomMove” is the function that is called when you click the button. Because you have default arguments already supplied in the macro, you do not need to supply any arguments. If you did supply arguments, they are placed between the parentheses: (). The last lines basically say “For the *RandomMove* function, set the slider ranges for the shutterOffset parameter between -1 and 1, with some extra granularity settings.” Since this sets the slider range, you can copy it and adapt it for the new frequency parameter.

- 2 Copy the last line of code and paste in a copy. Change the word *shutterOffset* to *frequency*, and adjust the values (based on the line in bold below):

```
...
nuiDefSlider( "RandomMove.shutterTiming" , 0 , 2 , 0 , 0 , 0.01 );
nuiDefSlider( "RandomMove.shutterOffset" , -1 , 1 , 0 , 0 , 0.01 );
nuiDefSlider( "RandomMove.frequency" , 0 , 10 , 0 , 0 , 0.01 );
```

#### To add an icon to the button:

- 1 Create a 75 x 40 pixel image.
- 2 Save the image as *TabName.Name.nri* to your <UserDirectory>/nreal//icons directory. In the above example, it would be called *Transform.RandomMove.nri*.

**Note:** You must strip out the alpha channel of the image. You can do this in Shake with a *SetAlpha* node set to a value of 0. Set the *FileOut* to your <UserDirectory>/nreal//icons directory, with the name *TabName.Name.nri*, and render the file.

- 3 In the *RandomMoveUI.h* file, remove the @ sign on line 3, and save the text file.
- 4 Restart Shake.

The *RandomMove* node appears with the icon.

**Note:** In all cases of the above code, case sensitivity is important. If the macro name is in all capital letters, then all occurrences of the name must also be in all capital letters. The same restriction applies to parameter and icon names.

For more information on customizing Shake, see Chapter 17, “Customizing Shake,” or “Attaching Parameter Widgets” on page 709.

## Variables, Linking, and Expressions

The following section discusses linking variables and parameters and using expressions.

### Linking Parameters

By linking variables and parameters, you can access information in any node and use it in a different node.

To use a parameter inside of the same node, type the parameter name. For example, the *Move2D* node links the *yScale* as equal to the *xScale* by default. To show the expression editor, enter any letter into the text field and press **Enter**. A plus sign appears next to the parameter. Click the plus sign to expand the parameter and enter your expression.



To use a parameter in a different node, enter the node name, followed by the parameter name, separated by a period:

*node.parameter*

The following example links the value parameter from the *Fade1* node and multiplies it by 2.



You can declare a variable anywhere in the script. However, to make the variable available in the interface, you must prefix it with the curve declaration:

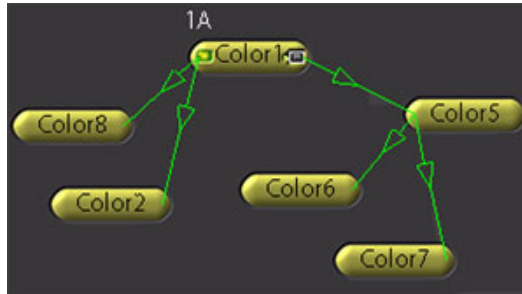
*curve parameter\_name = expression;*

For example, use the following to declare the *my\_val* variable for the above example:

*curve my\_val = 1;*

To clone entire nodes in the interface, copy the node and then press **Command+Shift+V** / **Ctrl+Shift+V**. This links all parameters in the copy to the original. If you modify anything in the copy, the link on that particular parameter is gone, so be careful of onscreen controls.

To view linked nodes in the Node View, press **Ctrl+E**.



### Linking to a Parameter at a Different Frame

You can use the syntax *parameterName@@time* to view a value at a different frame. For example, the following looks at the *yPan* parameter of the *Move2D1* node at the previous frame:

*Move2D1.yPan@@(time-1)*

### Variables

You can declare your own variables, as shown above. However, each image node also carries the information about that node, using the variables *width*, *height*, and *bytes*. When you refer to these, they work exactly the same as above. For example, by default, the center of rotation on *Move2D* is set to:

*width/2*

This places the center at the midpoint of the current image.

When referring to these from a different node, place the node name before the variable:

*node\_name.width*

In some cases, problems may occur. For example, in a *Resize* node, if you set the *Resize* equal to *width/2*, you potentially cause a loop because it is changing width based upon a value that is looking for the width. To solve this, Shake always refers to the input width, height, and bit depth when you refer to these from inside of that node. Therefore, *width/2* takes the incoming width and divides it by 2. When you refer to the width, height, and bit depth of a different node, you use that node's output values.

At any time, you can also use the variable *time*, which refers to the current frame number. For example, enter *cos(time/5)\*50* in the angle parameter text field in a *Rotate* node for a nice “rocking” motion.

### Expressions

The following section provides examples of using expressions (which can help out the lazy compositor by doing your work for you). In any parameter, you can combine any value with a math expression, trigonometry function, an animated curve, a variable, or even a conditional expression. For example, as mentioned above, the center of an image can be found by using:

```
xCenter = width/2
yCenter = height/2
```

These take the per-image variables *width* and *height* and divides them by 2.

You can type an expression in any field. Some functions, such as *ColorX*, *WarpX*, and *TimeX*, even support locally-declared variables. For more information and a list of examples, see “*ColorX*” on page 302.

If you are using the command-line method, you may have to enclose your expressions in quotes to avoid problems with the operating system reading the command. For example, instead of:

```
sbake my_image.iff-rot 45*6
use
sbake my_image.iff-rot "45*6"
```

Examples	Explanation
<i>1/2.2</i>	1 divided by 2.2. Gives you the inverse of 2.2 gamma.
<i>2*Linear(0,0@1,200@20)</i>	Multiplies the value of an animated curve by 2.
<i>2*my_curve</i>	Multiplies a variable by 2.
<i>sqrt(my_curve-my_center)/2</i>	Subtracts <i>my_center</i> from <i>my_curve</i> , takes the result square root, and then divides by 2.
<i>time&gt;20?1:0</i>	If time is greater than 20, then the parameter is 1, otherwise it equals 0.
<i>cos(time/5)*50</i>	Gives a smooth ping-pong between -50 and 50.

### Precedence

The above operators are listed in order of precedence—the order Shake evaluates each operator, left to right. If this is difficult to keep up with (and it is), make liberal use of parentheses to force the order of evaluation. For instance:

```
a = 1 + 2 * 4 - 2
```

This expression does "2\*4" first, since the "\*" has precedence over "+" and "-" which gives you "a=1+8-2". Then from left to right, Shake does "1+8", giving "a=9-2", finally resulting in "a=7". To add and subtract before multiplying, use parentheses to control the evaluation.

a = ( 1 + 2 ) \* ( 4 - 2 )

This results in "a=3\*2" or "a=6".

**Note:** Parentheses have the highest precedence in an expression.

## Functions, Variables, and Expressions

All of the math functions can be found in the *include/nreal.b* file. You can declare your own functions in your own .h file.

To set an expression on a string (text) parameter, you need to add a : (colon) at the start of the expression; otherwise, it is treated as text rather than compiled and evaluated.

Arithmetic Operators	Definition
*	Multiply
/	Divide
+	Add
-	Subtract
Relational Operators	Definition
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
Logical Operators	Definition
&&	And
	Or
!	Not

Conditional Expression	Definition
expr1?expr2:expr3	If expr1 is true (non-zero), then to expr2, else do expr3.

Global Variables	Definition
time	Current frame number.

The following table shows variables that are carried by each node.

Image Variables	Definition
parameterName	Value of <i>parameterName</i> from inside of that node.
nodeName.parameterName	Value of <i>parameterName</i> in nodeName from outside of that node.
parameterName@@time	Allows you to access a value at a different frame. For example: <i>Blur1.xPixel@@(time-3)</i> looks at the value from 3 frames earlier.
bytes	The number of bytes in that image. This takes the input bit depth when called from inside of the node, and the output bit depth when called from outside of the node.
width	Width of the image. Takes the input width when called from inside of the node, and the output width when called from outside of the node.
height	Height of the image. Takes the input height when called from inside of the node, and the output height when called from outside of the node.
_curlImageName	Returns the name of the actual file being used for the current frame. Useful when plugged into a <i>Text</i> node: <i>{FileIn1._curlImageName}</i>
dod[0], dod[1], dod[2], dod[3]	The variable for the Domain of Definition (DOD): xMin, yMin, xMax, yMax, respectively.

The following table shows channel variables used in nodes such as *ColorX*, *LayerX*, *Reorder*, etc. Check the documentation for specific support of any variable

In-Node Variables	Definition
nr, ng, nb, na, nz	New red, green, blue, alpha, Z channel.
r, g, b, a, z	Original red, green, blue, alpha, Z channel.

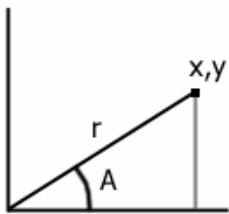
In-Node Variables	Definition
l	Luminance channel for <i>Reorder</i> .
n	Null channel. Strips out the alpha in <i>Reorder</i> when used like this: <i>rgbn</i>
r2, g2, b2, a2, z2	Second image's channel for <i>LayerX</i> .
Math Functions	Definition
abs(x)	Integer absolute value. $\text{abs}(-4) = 4$ . Be careful, as this returns an integer, not a float. Use <i>fabs</i> for float.
biasedGain(value, gain, bias)	Gives a <i>ContrastLum</i> -like curve that gives roll-off between two values.
cbrt(x)	Cubic root. $\text{cbrt}(8) = 2$
ceil(x)	Truncates to next integer. $\text{ceil}(5.3) = 6$
clamp(x, lo, hi)	Clamps x to between lo and hi. $\text{clamp}(1.5, 0, 1) = 1$
exp(x)	Natural exponent. $\text{exp}(0) = 1$
fabs(x)	Float absolute value. $\text{fabs}(-4.1) = 4.1$
floor(x)	Truncates to next lowest integer. $\text{floor}(5.8) = 5$
fmod(x,y)	Float modulus. Returns the remainder in float. $\text{fmod}(11.45, 3) = 2$ , for example, $(3 \times 3 + 2.45 = 11.45)$
log(x)	Natural log. $\log(1) = 0$
log10(x)	Returns base 10 log. $\log_{10}(10) = 1$
M_PI	A variable set to pi at 20 decimal places.
max(a,b)	Returns maximum between a and b. $\text{max}(5, 10) = 10$
max3(a,b,c)	Returns maximum between a, b, and c. $\text{max3}(5, 2, 4) = 5$
min(a,b)	Returns minimum between a and b. $\text{min}(5, 10) = 5$

Math Functions	Definition
<code>min3(a,b,c)</code>	Returns minimum between a, b, and c. $\text{min3}(5,2,4) = 2$
<code>a%b</code>	Modulus. $27\%20 = 7$
<code>pow(x,y)</code>	Returns x to the y power. $\text{pow}(2,4) = 16$
<code>round(x)</code>	Rounds number off. Values below x.5 are rounded to x, values equal to or above x.5 are rounded to x + 1. $\text{round}(4.3) = 4$
<code>sqrt(x)</code>	Square root. $\text{sqrt}(9) = 3$

Noise Functions	These are ideal for WarpX and ColorX.
<code>noise(seed)</code>	1-dimensional cubic spline interpolation of noise.
<code>noise2d(seed,seed)</code>	2d noise.
<code>noise3d(seed,seed,seed)</code>	3d noise.
<code>noise4d(seed,seed,seed,seed)</code>	4d noise.
<code>lnoise(seed)</code>	1d linear interpolation of noise.
<code>lnoise2d(seed,seed)</code>	2d noise.
<code>lnoise3d(seed,seed,seed)</code>	3d noise.
<code>lnoise4d(seed,seed,seed,seed)</code>	4d noise.
<code>fnoise(x,xScale)</code>	1d fractal noise based on noise().
<code>fnoise2d(x,y,xScale,yScale)</code>	
<code>fnoise3d(x, y, z, xScale, yScale, zScale)</code>	
<code>turbulence(x, xScale)</code>	A cheaper, rougher version of fnoise().
<code>turbulence2d(x, y, xScale, yScale )</code>	Continuous 2d noise.
<code>turbulence3d(x, y, z, xScale, yScale, zScale)</code>	Continuous 3d noise.

Noise Functions	These are ideal for WarpX and ColorX.
rnd(seed)	Hash-based pseudo-random numbers. Non-hash based RNG (like rand() or drand48()) should not be used in Shake because they cannot be reproduced from one machine to another. Also even on the same machine, repeated evaluations of the same node at the same time would produce different results.
rnd1d(seed, seed)	1d random value.
rnd2d(seed,seed,seed)	2d random value.
rnd3d(seed,seed,seed,seed)	3d random value.
rnd4d(seed,seed,seed,seed,seed)	4d random value.

Trig Functions (in radians)	Definition
M_PI	A variable set to pi at 20 decimal places.
acos(A)	Arc cosine in radians.
asin(A)	Arc sine.
atan(A)	Arc tangent.
atan2(y,x)	Returns the radian verifying $\sin(a) = y$ and $\cos(a) = x$ .
cos(A)	Cosine.
sin(A)	Sin.

Trig Functions (in degrees)	Definition
 $\begin{aligned}\text{sin}d(A) &= y/r \\ \text{cos}d(A) &= x/r \\ \text{tan}d(A) &= y/x \\ \text{atan2}d(y,x) &= A\end{aligned}$	<p>Yummy trigonometry! Welcome back. For those of you who may have forgotten, here is a helpful chart for some commonly used equations.</p>
<code>acosd(A)</code>	Arc cosine in degrees.
<code>asind(A)</code>	Arc sine in degrees.
<code>atand(A)</code>	Arc tangent in degrees.
<code>atan2d(y,x)</code>	Returns the angle verifying $\sin(a) = y$ and $\cos(a) = x$ .
<code>cosd(A)</code>	Cosine in degrees.
<code>distance(x1,y1,x2,y2)</code>	Calculates the distance between two points, (x1,y1) and (x2, y2).
<code>sind(A)</code>	Sin in degrees.
<code>tand(A)</code>	Tangent in degrees.
String Functions	Definition
<code>stringf("xyz", ...)</code>	<p>Since you basically can write books on this, here is an example. Otherwise, it is recommended to purchase a book on C. There are also several examples in “Macro Examples” on page 720. This example takes the <i>scriptName</i> parameter and uses the system function <code>echo</code> to print it:</p> <pre>extern "C" int system(const char*); const char *z= stringf("echo %s",scriptName); system(z);</pre>
<code>printf("xyz", ...)</code>	

String Functions	Definition
<code>strlen("mystring")</code>	Returns the length of the string.
<code>strsub( const char *string, int offset, int length</code>	Extracts a string from another string.

### Curve Functions

The curve functions with implicit time (Linear, CSpline, etc.) all assume that time is the first argument, so the following statements are identical:

```
LinearV(time, 0, 1@1, 20@20)
```

```
Linear(0, 1@1, 20@20)
```

You can, however, adjust the time value explicitly with the V version of each curve type. For more information on spline types, see “About Splines” on page 502.

These are the cycle type codes:

0 = KeepValue

1 = KeepSlope

2 = RepeatValue

3 = MirrorValue

4 = OffsetValue

Curve Functions	Definition
<code>biasedGain(x,gain,bias)</code>	Gives a smoothly-ramped interpolation between 0 and 1, similar to Shake's contrast curve. Gain increases the contrast, and bias offsets the center.
<code>Linear(cycle, value@key1, value@key2, ...)</code>	Linear interpolation from value at keyframe 1 to value at keyframe 2, etc.

Curve Functions	Definition
LinearV(time_value, cycle, value@key1, value@key2, ...)	Linear interpolation from value at keyframe 1 to value at keyframe 2, etc.
CSpline(cycle, value@key1, value@key2, ...)	Cardinal-spline interpolation, also known as Catmull-Rom splines.
CSplineV(time_value, cycle, value@key1, value@key2, ...)	Cardinal-spline interpolation, also known as Catmull-Rom splines.
JSpline(cycle, value@key1, value@key2, ...)	Jeffress-spline interpolation.
JSplineV(time_value, cycle, value@key1, value@key2, ...)	Jeffress-spline interpolation.
NSpline(cycle, value@key1, value@key2, ...)	Natural-spline interpolation.
NSplineV(time_value, cycle, value@key1, value@key2,...)	Natural-spline interpolation.

Curve Functions	Definition
Hermite(cycle, [value,tangent1,tangent2]@key1, [value,tangent1,tangent2]@key2, ...)	Hermite-spline interpolation.
HermiteV(time_value, cycle, [value,tangent1,tangent2]@key1, [value,tangent1,tangent2]@key2, ...)	Hermite-spline interpolation.

## Script Manual

When Shake saves a script, it creates a file that is basically in the C programming language. This makes the product open and flexible, as you can add your own functions, or use programming structures to do procedurally what is extremely tedious by hand. You can also quickly make minor modifications that are cumbersome to do in the interface (for example, changing a *FileIn* to read *BG2.iff* instead of *BG1.rla*). This section of the guide explains the basic principles of this scripting language, and how they can be manipulated to make your own macros.

The examples in “Attaching Parameter Widgets” on page 709, provide step-by-step user interface scripting examples.

See “Lesson Eight: Making a Macro,” in the *Shake 3 Tutorials*, for information on making macros interactively or in a script.

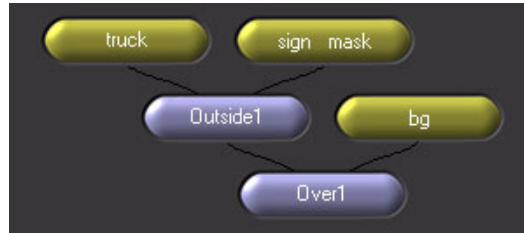
## Scripting Principles

### Scripting Controls

- To generate a test script, go to the *doc/pix/truck* directory. Enter the following command to create a script and save it as *start.shk*:

```
shake truck.iff -outside sign_mask.iff -over bg.iff -savescript start.shk
```

The truck is composited over the background, with the sign mask as a holdout mask, and a script named *start.shk* is saved. The composite itself is unimportant, but the tree structure is important. The following image shows how the script appears in the interface.



- To test the script result, type:

```
shake -script start.shk
```

The image of the truck is composited over the street image.

The following is essentially what the script itself looks like:

```
SetTimeRange("1");
SetFieldRendering(0);
SetFps(24);
SetMotionBlur(1, 1, 0);
SetQuality(1);
SetUseProxy("Base");
SetProxyFilter("default");
SetPixelScale(1, 1);
SetUseProxyOnMissing(1);
SetDefaultWidth(720);
SetDefaultHeight(486);
SetDefaultBytes(1);
SetDefaultAspect(1);
SetDefaultViewerAspect(1);
SetTimecodeMode("24 FPS");
SetDisplayThumbnails(1);
SetThumbSize(15);
SetThumbSizeRelative(0);
SetThumbAlphaBlend(1);

// Input nodes

bg = SFileIn("bg.iff", "Auto", 0, 0);
sign_mask = SFileIn("sign_mask.iff", "Auto", 0, 0);
truck = SFileIn("truck.iff", "Auto", 0, 0);
```

```
// Processing nodes

Outsidel = Outside(truck, sign_mask, 1);
Over1 = Over(Outsidel, bg, 1, 0, 0);
```

The first section contains controls for the script. The controls are all optional or can be overridden on the command line. See “Standard Script Commands and Variables” on page 718 for more information. The following sections discuss the body of the script, the sections listed under the “Input nodes,” and “Processing nodes” comments.

### Why “SFileIn” and Not “FileIn” Node?

The *SFileIn* node is an improvement on the older *FileIn* node. The interface button *FileIn* is linked to *SFileIn*, but the older name is retained. The *SFileIn* node allows extra subfunctions for timing to be attached.

### Variables and Data Types

Shake assigns types of data to a variable name. A variable works as a sort of shopping cart (pick any metaphor) to carry around your information to be used again somewhere else. The variables in the following code (excerpted from above) are bold. The comment lines (lines starting with //) are omitted:

```
bg = SFileIn("bg.iff", "Auto", 0, 0);
sign_mask = SFileIn("sign_mask.iff", "Auto", 0, 0);
truck = SFileIn("truck.iff", "Auto", 0, 0);
Outsidel = Outside(truck, sign_mask, 1);
Over1 = Over(Outsidel, bg, 1, 0, 0);
```

The above code assigns three variables (*bg*, *sign\_mask*, *truck*) to three *SFileIn* nodes. These are connected to an *Outside* node and then to an *Over* node, which are also assigned variable names, *Outsidel* and *Over1*.

**There are two ways to load a script into the interface:**

- Save the script and load it into the interface with the Terminal command *shake start.shk*, or click the Load button in the interface.
- Copy the script as text from the HTML browser/text editor and paste it into the Node View (**Command+V** / **Ctrl+V**). When you paste it into the interface, it points out that the filepaths are local, and probably not correct in terms of the location of the images and where the interface expects to find the images. Therefore, browse to the directory that contains the images, and click OK.

A good reason to use a script is that you can quickly set the path for the images by using copy and paste functions in a text editor.

Because the above script was created with a local filepath (in the */doc/pix/truck* directory), the images can only be found if the script is run from the truck directory. For example, a local directory for the truck image:

*/doc/pix/truck/truck.iff*

In order to run the script from any location—so the images can be found regardless of the location of the saved script—the absolute path of the images is required. The following is an example of an absolute directory for the same image (truck) is:

*/Server02/VolumeX/Scene12/doc/pix/truck/truck.iff*

**To reset the filepaths to an absolute path:**

- 1 To determine the absolute path of the images, go into the */doc/pix/truck* directory (using the command line) and type the Present Working Directory command:

*pwd*

- 2 Enter the filepath before the image names to make the path absolute. Shake does not prompt you to find the absolute path each time you paste it into the interface. Changes are in bold.

```
bg = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck/  
bg.iff", "Auto", 0, 0);  
sign_mask = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck/  
sign_mask.iff", "Auto", 0, 0);  
truck = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck/  
truck.iff", "Auto", 0, 0);  
Outsidel = Outside(truck, sign_mask, 1);  
Overl = Over(Outsidel, bg, 1, 0, 0);
```

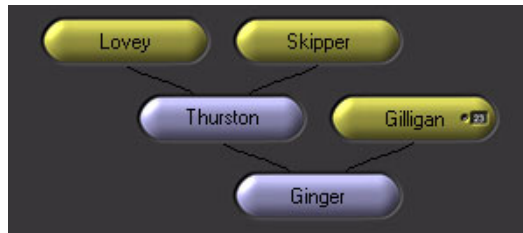
The left side is the variable name. You can change the variable name, but you must change the names wherever they appear.

**3** Change the variable names:

```
Gilligan = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck/  
bg.iff", "Auto", 0, 0);  
Skipper = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck/  
sign_mask.iff", "Auto", 0, 0);  
Lovey = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck/  
truck.iff", "Auto", 0, 0);  
Thurston = Outside(Lovey, Skipper, 1);  
Ginger = Over(Thurston, Gilligan, 1, 0, 0);
```

Therefore, the left side of the above lines is the variable that you assign a value. The right side is the value, usually coming from a function such as *SFileIn*, *Outside*, or *Over*. The function is called by its name, followed by its arguments between parentheses. The arguments, called parameters, are very specifically organized and always specify the same thing. For example, the third parameter of *Outside* is 1—a numeric code to determine whether you take the foreground or background resolution (see “*Outside*” on page 190). The line ends with a semicolon.

**4** Feed the modified script back into the Shake interface.



The following steps illustrate how variables can help you. Insert the *Mult* color correction node to change the truck color. The script format for *Mult* (in “*Mult*” on page 298) looks like the following:

```
image Mult(  
    image In,  
    float red,  
    float green,  
    float blue,  
    float alpha,  
    float depth  
);
```

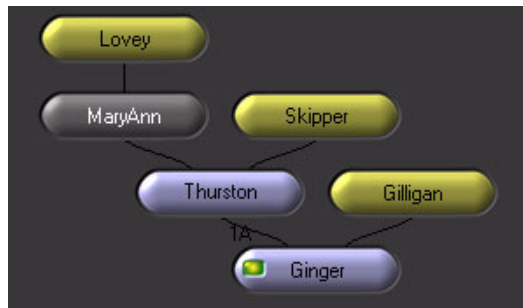
To add the *Mult* node, assign a variable name (here, *MaryAnn*) and then the parameters. Since *Lovey* is the variable name of the truck, it is fed in as the image. The numbers turn the truck yellow. Since *Thurston* (the *Outside* node) previously loaded the *Lovey* node, switch it to *MaryAnn*, the new *Mult* node.

**Note:** The premultiplied state of the truck image is ignored for this example.

**5** Insert the *Mult* node into the script:

```
Gilligan = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck/
bg.iff", "Auto", 0, 0);
Skipper = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck/
sign_mask.iff", "Auto", 0, 0);
Lovey = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck
truck.iff", "Auto", 0, 0);
MaryAnn = Mult(Lovey, 1, 1, .2);
Thurston = Outside(MaryAnn, Skipper, 1);
Ginger = Over(Thurston, Gilligan, 1, 0, 0);
```

The result appears as follows in the interface:



Only four parameters are entered for the *Mult* node—the alpha and depth parameters are omitted. Therefore, the alpha and depth parameters default to a value of 1. You can also see that *Mult* looks for two types of data: An *image* (labeled *In*) and *floats* (labeled *red*, *green*, *blue*, etc.). A *float* is any number that contains a decimal place, for example, 1.2, 10.00, .00001, or 100,000.00. The following table lists the five types of data.

Data Type	Notes
image	An image.
float	A number with a decimal place, such as .001, 1.01, or 10,000.00.
int	Integer. A number with no decimal place, such as 0, 1, or 10,000.

Data Type	Notes
string	"this is a string"
curve float or curve int	A special case of float or int that designates that the number has the possibility of being changed during script execution or when tuning it in the interface.

Shake is usually smart enough to convert an integer into a float, so when the following is entered:

```
MaryAnn = Mult(Lovey, 1, 1, .2);
```

Shake does not freak out that you enter two integers (1, 1) for the red and green multipliers. However, this is an infrequent case concerning the conversion of floats and integers, as you cannot put in a string or an image for those arguments. For example:

```
MaryAnn = Mult(Lovey, "1", Gilligan, .2);
```

does not work because you are drastically mixing your data types (plugging an image into a float). There is one exception to this: When you enter 0 as an image input, indicating you do not want to designate any image input.

#### **To designate that a function has no image input:**

Place a 0 in the image position. This example has no image input for the background image, the second argument:

```
MaryAnn = Over(Thurston, 0);
```

So far, you have only assigned variables to image types. In this next example, create a float variable so you can multiply the truck image by the same amount on the red, green, and blue channels. Call this variable *mulVal*.

**Note:** For these examples, you must save the script and use Load or Reload Script—the Copy/Paste does not work properly.

- 1 Create a script variable and plug it into the *Mult* node:

```
float mulVal = .6;
Gilligan = SFileIn("/Server02/VolumeX/Scene12/doc/pix
truck/bg.iff", "Auto", 0, 0);
Skipper = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck
sign_mask.iff", "Auto", 0, 0);
Lovey = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck
truck.iff", "Auto", 0, 0);
```

```
MaryAnn = Mult(Lovey, mulVal, mulVal, mulVal);
Thurston = Outside(MaryAnn, Skipper, 1);
Ginger = Over(Thurston, Gilligan, 1, 0, 0);
```

*Mult* now takes .6 for its red, green, and blue parameters. To change all three, modify *mulVal* on the first line of the script and execute the script again. This is swell, and you are probably patting yourself on the back for how blazingly clever you are. However, here is a problem. The variable *mulVal* is only applied when you load the script—Shake does not retain it for later use. When the script is loaded (with Load Script, not Copy/Paste) into the interface, the *Mult* parameters all read .6, not *mulVal*. There is no place in the interface that you can find the *mulVal* parameter and modify it and have the *Mult* take the argument. If you are immediately executing the script, this is not a problem. If you are loading the script and are going to interactively edit it, this is a problem. You therefore must declare *mulVal* as a curve float, a special type of float that tells Shake to wait until frame calculation to resolve the variable.

- 2 Convert the *mulVal* variable to a changeable curve type:

```
curve float mulVal = .6;
Gilligan = SFileIn("/Server02/VolumeX/Scene12/doc/pix
truck/bg.iff", "Auto", 0, 0);
Skipper = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck/
sign_mask.iff", "Auto", 0, 0);
Lovey = SFileIn("/Server02/VolumeX/Scene12/doc/pix/truck/
truck.iff", "Auto", 0, 0);
MaryAnn = Mult(Lovey, mulVal, mulVal, mulVal);
Thurston = Outside(MaryAnn, Skipper, 1);
Ginger = Over(Thurston, Gilligan, 1, 0, 0);
```

- 3 Load the script into the interface with Load Script.
- 4 Open the local Parameters tab in the Globals tab to reveal the *mulVal* slider.

In short, if you load a script to be modified in the interface, you probably want to declare it as a curve type of data. If you are going to execute the script on the command line and it is not going to change (that is, it is not animated), you can omit the curve data type. There are some examples in the following sections.

## Functions

Shake is a collection of functions. Some functions modify images, such as *Blur*. Some return a number, like *distance()* (calculates the distance between two points) or *cosd()* (a cosine function in degrees, not radians). Other functions build the interface, such as *nuiToolBoxItem*. The *nuiToolBoxItem* function loads a button into the interface and attaches a function to the button. When you call a function, you assign a variable to it.

The following example reads the angle parameter for the *Rotate1* node, and returns the cosine in degrees, assigning it to the variable named *myCos*:

```
curve float myCos = cosd(Rotate1.angle);
```

You can also use functions inside of other functions. In this example, the *cosd* function is inside of a *Rotate* function:

```
Rotate1 = Rotate(FileIn1, cosd(45));
```

When you place a function inside of another, it is called a “nested function.” In both cases, you call the function, and then enclose its parameters in parentheses. When you assign a variable to a function, terminate the line with a semicolon, as shown in both of the examples above. Everything in between the parentheses can be formatted however you want, so the following two examples are identical:

```
Mult1 = Mult(FileIn1, 1,1,1);
```

and

```
Mult1 = Mult(  
    FileIn1,  
    1,  
    1,  
    1  
);
```

### Function Formats

So, where are all of these functions and how do you find their formats? Typically, they are organized by the type of data they manipulate.

Here is a handy way to get a function format: Create the node(s) in the interface, select and copy the node(s) (**Command+C** / **Ctrl+C**), and paste the nodes into a text editor.

For more information:

- For image functions, see their relative chapters. For example, for information on the *Rand* function, see “*Rand*” on page 570.
- For mathematical functions, see “Expressions” on page 683.
- For Shake settings, see “Setting Preferences and Customizing Shake” on page 623.
- For interface building functions, see “Setting Preferences and Customizing Shake” on page 623.
- Examples abound in `<ShakeDir>/include/nreal.h` and `<ShakeDir>/include/nrui.h`, as well as in the “Cookbook” section of the *Shake 3 Tutorials*.

## Script Comments

To temporally comment out lines in a script, use the following symbols:

```
# This line is commented out

// This line is also commented out

This is not commented out //But this is

/*
All of
these lines
are
commented out
*/
```

## Conditional Statements

Like in C, you can use conditional statements in your scripts. These are particularly useful in macros, where a conditional parameter is input by the user. Keep in mind that the conditional statements require you to work in the script and cannot be built with interface tools.

### Conditional Expression

To simply switch a parameter, use the in-parameter conditional expression *expr1?expr2:expr3*, which reads as, “if *expr1* is true, use *expr2*; otherwise, use *expr3*.” For example, *time>10?0:1* reads as, “if time is greater than 10, then set the value to 0; otherwise, set it to 1.”

In the interface, you can also use the *Select* node to switch between any number of input nodes. (For more information on the *Select* node, see “*Select*” on page 198.) This strategy allows you to stay within the interface without resorting to the script. However, the difference is that Shake only builds the part of the script that it needs for conditional statements listed below. *Select* has the overhead of all of its input nodes.

Finally, *++i* or *--i* is supported for increments. *i++* is as well, but returns a warning message.

### If/Else

This is used to express decisions. If the test expression is true, then the first statement is executed. If false (and an “else” part exists), then the second statement is executed. If “else” is left off, Shake does nothing and moves on.

```
if (expression evaluates to non-zero) {
    do_this
```

```

} else if {
    do_this
} else if {
    do_this
} else {
    do_this
}

or just

if (expression evaluates to non-zero) {
    do_this
} else if {
    do_this
}

```

### **For**

Normally, the three expressions of the "for" loop are an initialization ( $a=1$ ), a relational test ( $a<10$ ), and an increment ( $a++$ ): `for (a=1; a<10; a++)`. So "a" is set to one, checked if it is less than 10, then if that is true, a statement performed, and "a" is incremented by one. The test is checked again and this cycle continues until "a" is found to be not less than 10 (untrue).

```

for (initialize; test; increment)
{
    do_this
}

```

### **While**

If the given expression is true or non-zero, the following statements are performed, and the expression is reevaluated. The cycle continues until the expression becomes false.

```

while (this_expression_is_true)
{
    do_this
}

```

If there is no initialization or reinitialization, "while" often makes more sense than "for".

### **Do/While**

This variation of "while" is different in that it tests at the bottom of the loop, so the statement body is done at least once. In the "while" above, the test may be false the first time and so nothing is done. Note on all of the other statements, a semicolon was not needed. This expression does need it at the end.

```
do {
    do_this
} while (this_expression_is_true);
```

## Macros

As discussed in the “Making Macros” on page 673, you can use preexisting functions to build your own function. The functions are called macros. This section discusses additional information, such as basic macro structure, common errors when creating macros, and sample macros.

The macro files can be saved into the script that uses them, or can be saved in a file with a .h file extension. These files are saved in your *startup* directory, located either in *<ShakeDir>/include/startup*, *<UserDir>/nreal/include/startup*, or a startup directory under a directory specified by the environment variable `$NR_INCLUDE_PATH`. For more information, see “Setting Preferences and Customizing Shake” on page 623.

### Basic Macro Structure

The following is what a macro looks like:

```
dataType MacroName(
    dataType parameterName=defaultParameterValue,
    dataType parameterName=defaultParameterValue,
    ...
)
{
    Macro Body
    return variable;
}
```

For example, the following function, called *angle*, calculates the angle between two points, returns a float, using the *atan2d* function. (For more information, see the “Trig Functions (in degrees)” section of the table in “Functions, Variables, and Expressions”.) Notice that the default parameter value is optional, so if you use this particular function, all four values must be supplied:

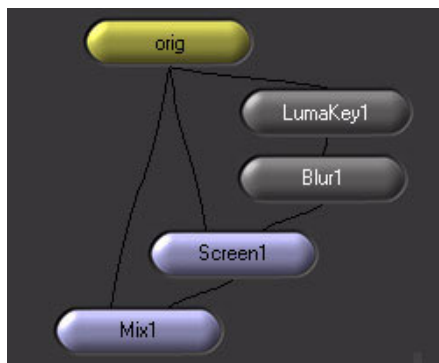
```
float angle(
    float x1,
    float y1,
    float x2,
    float y2
)
{
    return atan2d(y2-y1,x2-x1);
}
```

Because the macro is so simple (one function), there is no Macro Body per se; the function is attached to the return statement, which indicates what is spit out of the macro. To use this function, use something like the following:

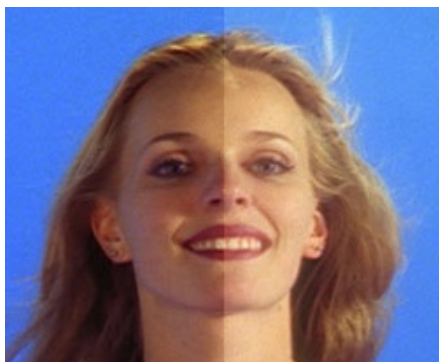
```
myAngle = angle(0,0,100,100);
```

that returns 45.0.

The following is an example of an image function. It adds that “vaseline-on-the-lens” effect (usually reserved for actresses who didn’t have time for makeup). The following represents the node tree, with a split-screen example of the effect.



The *LumaKey* node is used to extract only the highlights. The highlights are blurred, and then applied back on the original image with the *Screen* node, which is nice for glows and reflections. The *Mix* node is used to control how much of the original image shows through. The example image shows the original image on the left, and the macro results on the right. (This photo is courtesy of Photron.)



The following are the nodes reformatted as a macro. The macro parameters are bold.

```
image SoftGlow(  
image In=0,
```

```

float blur=0,
float lowClip=.3,
float hiClip=.9,
float percent=100
)
{
LumaKey1 = LumaKey(In, lowClip, hiClip, 0, 0, 1);
Blur1 = Blur(LumaKey1, blur, xPixels, 0, "gauss", xFilter,
"rgba");
Screen1 = Screen(In, Blur1, 1);
Mix1 = Mix(In, Screen1, 1, percent, "rgba");
return Mix1;
}

```

The macro name is *SoftGlow*, and it outputs an image (line 1). The parameters are expecting one image input named *In* (line 2). This gives you one input at the top of the node. If you want two image inputs, there would be two image parameters, and so on. The other values are all float, controlling the macro settings (lines 3-6). These are assembled in the macro body, and the return statement indicates that *Mix1* is output. The *low* and *hiClip* parameters determine the levels of the highlights.

If you save this into a startup .h file, for example, *\$HOME/nreal/include/startup/SoftGlow.h*, it is immediately available on the command line. You can also find a copy of this called *SoftGlow.h* in *docs/html/cook/macros*.

Type:

*shake -help softglow*

to return:

*-softglow [blur] [lowClip] [hiClip] [percent]*

#### File Name Versus Macro Name

Names of files have *nothing* to do with names of macros. Only the function name is important when called in the script. You can also have multiple macros per file.

## Loading Image Macros Into the Interface

When you start the Shake interface, the macros do not appear in the interface. A separate file and set of functions are required to load the macros in the interface. These ui functions are saved in a subdirectory of *startup* called *ui*. The following ui code creates a button in the Filter tab with no icon that is labeled “SoftGlow,” and calls up the *SoftGlow* function when clicked. Save it in `<UserDir>/nreal/include/startup/ui` or `$NR_INCLUDE_PATH/startup/ui`. You can also find a copy of this in *docs/html/cook/macros* called *SoftGlowUI.h*.

```
nuiPushMenu( "Tools" );
    nuiPushToolBox( "Filter" );
        nuiToolBoxItem( "@SoftGlow", SoftGlow() );
    nuiPopToolBox();
nuiPopMenu();
```

The @ sign indicates that no icon is associated with the button. If you have an icon, omit the @ sign (do not add the tab prefix or image type extension) and save an icon image, 75 x 40 pixels in size, in your icons directory (`<ShakeDir>/icons` or `<UserDir>/nreal/icons` or `$NR_ICON_PATH`), naming it *Filter.SoftGlow.nri*. To see this process listed step-by-step, see “Lesson Eight: How to Make a Macro” in the *Shake 3 Tutorials*.

The images for the icons have the following characteristics:

- 75 x 40 pixels
- no alpha channel
- named as *TabName.FunctionName.nri*

When calling the icon in the ui.h file, omit the *TabName.* and *.nri*.

- font: 11-point Arial
- saved in `<ShakeDir>/icons` or `<UserDir>/nreal/icons` or `$NR_ICON_PATH`

## Typical Errors When Creating Macros

The following table contains a list of typical errors in macro creation. When diagnosing a macro, first run the macro in the command line: *shake -help myFunction*. If nothing appears, there is a problem with your *startup* .h file. If it works fine, move on to the interface, and read any error messages in the console window. The console window is your number-one diagnostic tool.

Error Behavior	Probable Cause
In the command line, the function doesn't appear when you type the following: <i>shake -help myFunctionName</i>	<ul style="list-style-type: none"><li>■ The file is not saved in a <i>startup</i> directory. See above.</li><li>■ The file does not have a .h file extension, or it has an improper extension.</li><li>■ The name of the macro (the second word in the file) is not the same as what you have called in the help command.</li></ul>
Function does not appear in the interface.	<ul style="list-style-type: none"><li>■ The ui file is not saved in a <i>ui</i> directory. See above.</li><li>■ The ui file does not have a .h file extension, or it has a .txt extension.</li><li>■ The name of the macro (the second word in the file) is not the same as what you have called in the ui file, possibly because of capitalization errors.</li></ul>
In the interface, the button appears without an icon.	You have not removed the @ sign in your ui.h file. Otherwise, follow "Lesson Eight: How to Make a Macro" in the <i>Shake 3 Tutorials</i> .
The icon appears as a dimple.	<p>The icon cannot be found.</p> <ul style="list-style-type: none"><li>■ Make sure the icon is saved in an icons directory. See above.</li><li>■ The icon should be named <i>TabName.FunctionName.nri</i>.</li><li>■ The ui code should say: <i>nuiToolBoxItem("FunctionName",Function());</i> NOT <i>nuiToolBoxItem("TabName.FunctionName.nri",Function());</i></li><li>■ Check capitalization.</li></ul>
The icon is fine, but nothing happens when you click it.	<ul style="list-style-type: none"><li>■ Check capitalization errors.</li><li>■ Check that the correct function is called in the ui file and that the function exists. (For example, type <i>shake -help functionname</i> in the command line.)</li><li>■ Check that default arguments have been specified.</li></ul>

## Setting Default Values for Macros

There are two places to set default values for your macros. The first is in the *startup* .h file when you declare the parameters. The following is from the example above:

```
image SoftGlow(  
    image In=0,  
    float blur=0,  
    float lowClip=.3,  
    float hiClip=.9,  
    float percent=100  
)  
...
```

Each of these has a default value assigned. Note that the image has 0, which indicates “no input.”

These values are applied to both the command line or the gui defaults. If you do not supply a default argument, you must enter a value when you call the function. It is therefore recommended that you enter defaults.

The second location is in the *ui.b* file when the function is called. To override the startup defaults, enter your own in the *ui.b* file. Normally, you have something like the following in your *ui.b* file:

```
nuiPushMenu("Tools");  
    nuiPushToolBox("Filter");  
        nuiToolBoxItem("@SoftGlow",SoftGlow());  
    nuiPopToolBox();  
nuiPopMenu();
```

This sets new default values just for the interface:

```
...  
        nuiToolBoxItem("@SoftGlow",SoftGlow(0,100,.5,1, 95));  
...
```

## Changing Default Settings

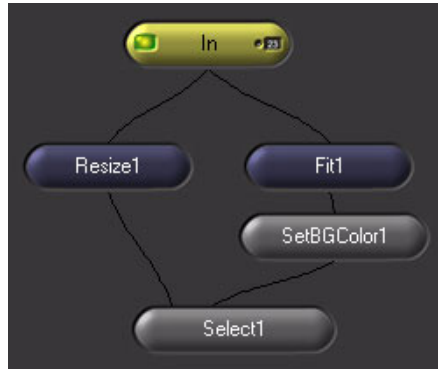
Most of Shake’s functions (third-party development excepted) are stored in two files in *<ShakeDir>/include* in *nreal.b* and *nrui.b*. *nreal.b* is the complete list of all functions and settings. The *nrui.b* file builds the interface. You can modify these files to suit your needs, but it is strongly recommended that you make a backup of these files before you begin. Otherwise, if you mess up the file, you mess up Shake. Way to go.

These files are also an excellent source for examples.

## Attaching Parameter Widgets

If you take a look at the “Parameters Tab” on page 649, and experiment with different nodes in the interface, you see that you can attach many behaviors to parameters. This section takes a raw macro and shows different examples of behaviors to change the parameters sliders to more efficient widgets.

The example macro that is created in the following example is called *VidResize*. It takes an image of any size and resizes it to video resolution. There are slider controls to specify NTSC or PAL (*vidformat*), to maintain the aspect ratio (*keepAspect*), and if you do keep the aspect ratio, the color of the exposed background area. The following image represents the original node tree.



### The Sample Macro VidResize

- 1 Quit Shake.
- 2 To load the macro, copy the *VidResize.h* file from *doc/html/cook/macros* to your *\$HOME/nreal/include/startup* directory.
- 3 Copy the *VidResizeUI.h* file from *doc/html/cook/macros* into your *\$HOME/nreal/include/startup/ui* directory.
- 4 Start the Shake interface.
- 5 Create a *Grad* node with the following parameter settings:
  - Set the width to 400.
  - Set the height to 200.
- 6 Attach the Transform–*VidResize* node and test each slider. To have any effect, the *keepAspect* parameter needs to jump to 2.

```
image VidResize(  
    image In=0,  
    int keepAspect = 1, //keeps aspect ratio or not  
    int vidFormat = 0, //This select NTSC (0) or PAL (1) res
```

```

float bgRed = 0, //if keeping aspect ratio, determines
float bgGreen =0, //the bg color
float bgBlue =0
)
{
    curve int yRes = vidFormat ==0?486:576;
    Fit1 = Fit(In, 720, yRes, "default", xFilter, 1);
    Resize1 = Resize(In, 720, yRes, "default", 0);
    SetBGColor1 = SetBGColor(Fit1, "rgbaz",
        bgRed, bgGreen, bgBlue, 0, 0
    );
    Select1 = Select(keepAspect, Resize1, SetBGColor1, 0, 0);
    return Select1;
}

```

The ui file looks like this:

```

nuiPushMenu("Tools");
    nuiPushToolBox("Transform");
        nuiToolBoxItem("@VidResize",VidResize());
    nuiPopToolBox();
nuiPopMenu();

```

The only tricky portions in the macro so far are the *Select* function, and the logic to choose the height. When the branch of *Select* equals 1, the first image input is passed through. When it equals 2, the second branch is fed through. The *keepAspect* parameter is connected to *Select* to choose the branch you want. The other tricky portion is the height logic, the first line of the macro body. It declares an internal variable (it cannot be seen outside the macro) named *yRes*. It tests *vidFormat* to see if it equals 0. If so, it sets *yRes* to equal 486, the height of an NTSC image. Otherwise, it sets *yRes* to 576, the standard PAL image height.

### Setting Slider Ranges

The first inconvenience about the macro, when used in the interface, is that the *keepAspect* slider goes from 0 to 1. The values you want are 1 or 2. Since you do not want to have to manually enter the format, it is to set the slider range from 1 to 2. This is done in the ui file *VidResizeUI.b*. To find the format, see “Parameters Tab” on page 649. If you are reading this document electronically, you can copy and paste the command from the document into the text file.

- 1 Load the *ui/VidResizeUI.b* file into the text editor.
- 2 Add the *nuiDefSlider* function to set a slider range:

```

nuiPushMenu("Tools");
    nuiPushToolBox("Transform");
        nuiToolBoxItem("@VidResize",VidResize());

```

```

nuiPopToolBox();
nuiPopupMenu();
nuiDefSlider("VidResize.keepAspect", 1, 2);

```

The word *VidResize* of *VidResize.keepAspect* indicates that you only want to modify the *keepAspect* slider in the *VidResize* function. If you do not specify this, for example, you just have *nuiDefSlider("keepAspect", 1, 2)*; and it tries to apply that slider range to all parameters named *keepAspect*, no matter what function they are in.

- 3 Save *VidResizeUI.h* and start Shake again.
- 4 Create the *VidResize* node (there is no need to test it on an image).

The *keepAspect* slider now goes from 1 to 2.

### Inappropriate Behavior in All the Wrong Places

If you start to see controls on parameters that you have not created, or if you see other functions that have odd behaviors, make sure you have specified what function receives the control. If you set:

```
nuiDefSlider("depth", 1, 2);
```

anytime Shake sees a parameter named “depth” (for example, if somebody makes a macro to set bit depth to a certain value), it takes a range of 1 to 2. Therefore, ensure that you preface the depth with a function name:

```
nuiDefSlider("MyFunction.depth", 1, 2);
```

### Creating an On/Off button

Rather than using a value (1 or 2) to indicate what the slider does, you can create an on/off button. When the button is on, you want to maintain the aspect ratio. When off, you do not want to maintain the aspect ratio. Again, the format information can be found in “Parameters Tab” on page 649. If you are reading this document electronically, you can copy and paste the command.

- 1 Replace the slider range function in the *VidResizeUI.h* file:

```

nuiPushMenu("Tools");
nuiPushToolBox("Transform");
nuiToolBoxItem("@VidResize", VidResize());
nuiPopToolBox();
nuiPopupMenu();
nuxDefExprToggle("VidResize.keepAspect");

```

The problem here is that it always returns a value of 0 or 1—0 is off and 1 is on. You want values of 1 or 2 because that is what the macro is counting on. Therefore, you must edit the macro.

- 2 In the startup file *VidResize.h*, add 1 to the test of *keepAspect* in the *Select* function:

```
...
    SetBGColor1 = SetBGColor(Fit1, "rgbaz",
        bgRed, bgGreen, bgBlue, 0, 0
    );
    Select1 = Select(keepAspect+1, Resize1, SetBGColor1, 0, 0);
    return Select1;
}
```

- 3 Save the file and start Shake.

The *keepAspect* parameter has an on/off button.



### Attaching Color Pickers and Subtrees

Using a slider to select a color is not nearly as impressive to your arch-foes as using the Color Picker, so attach *bgRed*, *bgGreen*, and *bgBlue* to a Color Picker to interactively pick your color. For the format information, see “Parameters Tab” on page 649.

Since this is an interface change, edit the ui *VidResizeUI.h* file:

```
nuiPushMenu("Tools");
    nuiPushToolBox("Transform");
        nuiToolBoxItem("@VidResize",VidResize());
    nuiPopToolBox();
nuiPopMenu();
nuxDefExprToggle("VidResize.keepAspect");
nuiPushControlGroup("VidResize.Background Color");
    nuiGroupControl("VidResize.bgRed");
    nuiGroupControl("VidResize.bgGreen");
    nuiGroupControl("VidResize.bgBlue");
nuiPopControlGroup();
nuiPushControlWidget("VidResize.Background Color",
    nuiConnectColorTriplet(kRGBToggle,kCurrentColor,1)
);
```

This file is interesting because the first five new lines are the code to make a subtree. Even if you do not add the last lines to attach the Color Picker, the three color sliders are still organized in a subtree named *Background Color*.

The last three lines attach the Color Picker to the subtree called *Background Color*, and sets the picker to select an RGB (as opposed to HSV, HLS, or CMY) Color Picker that uses the Current (as opposed to the Average, Maximum, or Minimum scrub) color.

The Color Picker is added to the interface.



### Attaching Button Toggles

Next, attach a button to toggle the *vidFormat*. Since the 0 and 1 settings are not very intuitive for the video format selection, create buttons labeled “NTSC” and “PAL.” The following examples show two ways to attach a button toggle.

- 1 Create a directory called *icons/ux* in your `<UserDir>/nreal` directory:  
`mkdir -p $HOME/nreal/icons/ux`
- 2 Copy all of the icons that begin with *vr\_* from *doc/html/cook/macro\_icons* to your `$HOME/nreal/icons/ux` directory. There are a total of eight files.

When clicked, the first button toggles between PAL (*vr\_pal.off.nri*) and NTSC (*vr\_ntsc.off.nri*). Two additional buttons, *vr\_pal.off.focus.nri* and *vr\_ntsc.off.focus.nri*, indicate when the cursor is over the button. These are called focus buttons. To view the code, see “Parameters Tab” on page 649.

- 3 To add the toggle button function, edit the *VidResizeUI.b* file:

```
nuiPushMenu( "Tools" );
    nuiPushToolBox( "Transform" );
        nuiToolBoxItem( "@VidResize", VidResize() );
    nuiPopToolBox();
nuiPopMenu();

nuxDefExprToggle( "VidResize.keepAspect" );

nuiPushControlGroup( "VidResize.Background Color" );
    nuiGroupControl( "VidResize.bgRed" );
    nuiGroupControl( "VidResize.bgGreen" );
```

```

        nuiGroupControl( "VidResize.bgBlue" );
nuiPopControlGroup( );
nuiPushControlWidget( "VidResize.Background Color",
        nuiConnectColorTriplet( kRGBToggle, kCurrentColor, 1)
    );

nuxDefExprToggle( "VidResize.vidFormat",
        "ux/vr_ntsc.off.nri|ux/vr_ntsc.off.focus.nri",
        "ux/vr_pal.off.nri|ux/vr_pal.off.focus.nri"
    );

```

The new lines list the normal button, followed by the focus button. The *icons* directory is automatically scanned, but notice you have specified the *ux* subdirectory. The value returned is always 0 for the first entry, 1 for the next entry, 2 for the third entry, and so on. You can place as many entries as you want. Each button click toggles you to the next choice.

- 4 Save the file and start Shake again.
- 5 Create the *VidFormat* node again.

The *vidFormat* parameter has a PAL/NTSC toggle.



The button created in the above steps is a single button that toggles through multiple choices. This is fine for binary (on/off) functions, but less elegant for multiple choice toggles. In the next example, create radio buttons. Radio buttons are similar to toggle buttons, except that you simultaneously see all available buttons. This code lists only one button name. Shake automatically assumes there is an *on*, *on.focus*, *off*, and *off.focus* version of each button in the directory you specify. If you copied the *vr\_* buttons earlier, you indeed have all of these buttons. How special. The code looks like the following in “Parameters Tab” on page 649.

- 1 Again, edit the *VidResizeUI.b* file:

```

nuiPushMenu( "Tools" );
    nuiPushToolBox( "Transform" );
        nuiToolBoxItem( "@VidResize", VidResize() );
    nuiPopToolBox();
nuiPopMenu();

nuxDefExprToggle( "VidResize.keepAspect" );

```

```

nuiPushControlGroup("VidResize.Background Color");
    nuiGroupControl("VidResize.bgRed");
    nuiGroupControl("VidResize.bgGreen");
    nuiGroupControl("VidResize.bgBlue");
nuiPopControlGroup();
nuiPushControlWidget("VidResize.Background Color",
    nuiConnectColorTriplet(kRGBToggle,kCurrentColor,1)
);

nuxDefRadioBtnCtrl(
    "VidResize.vidFormat", 1, 1, 0,
    "0|ux/vr_ntsc",
    "1|ux/vr_pal"
);

```

The numbers immediately to the left of the icon listing, for example, the 0 in "0|ux/vr\_ntsc", show the value returned when that button is clicked. The nice thing about radio buttons is that they can return strings, float, or int. For example, the *channel* parameter in the *KeyMix* node selects the channel to do the masking, with R,G,B, or A returned, all strings. Because your macro *VidResize* only understands 0 or 1 to be meaningful, use 0 and 1 as your return values.

- 2 Save the text file and start Shake again.

The radio buttons appear in the *VidFormat* parameters.



### Making Radio or Toggle Buttons

There is an unofficial function to create these buttons. That's right, you can use a macro to help make your macros.

In *doc/html/cook/macros*, copy *radiobutton.b* and *relief.b* to your startup directory.

In the command line, type something like the following:

```
shake -radio "NTSC size" 79 vr_ntsc $HOME/nreal/icons/ux -t 1-4 -v
```

This creates four buttons named *vr\_ntsc.on.nri*, *vr\_ntsc.on.focus.nri*, *vr\_ntsc.off.nri*, and *vr\_ntsc.off.focus.nri* in *\$HOME/nreal/icons/ux*. Each button is 77 pixels wide and each one says *NTSC size*. You must do this in the command line because at the moment the *FileOut* does not work in a macro unless it's the command line, and frames 1-4 create the four different files.

### Attaching Pop-Up Menus

Although the radio buttons are pretty cool, another frequent option is a pop-up menu. The pop-up menu is good when there are more choices than space in a standard Parameters tab. The only catch is that they return strings (words), not numbers, so you have to add some logic in your macro to interpret the strings. The following is the ui code, which can also be found in "Parameters Tab" on page 649.

- 1 Add the following code to the ui file to create a pop-up:

```
nuiPushMenu("Tools");
    nuiPushToolBox("Transform");
        nuiToolBoxItem("@VidResize",VidResize());
    nuiPopToolBox();
nuiPopMenu();

nuxDefExprToggle("VidResize.keepAspect");

nuiPushControlGroup("VidResize.Background Color");
    nuiGroupControl("VidResize.bgRed");
    nuiGroupControl("VidResize.bgGreen");
    nuiGroupControl("VidResize.bgBlue");
nuiPopControlGroup();
nuiPushControlWidget("VidResize.Background Color",
    nuiConnectColorPCtrl(kRGBToggle,kCurrentColor,1)
);
nuxDefMultiChoice("VidResize.vidFormat", "NTSC|PAL");
```

You can add as many entries as you want, and each entry is separated by the “|” symbol. There is a problem with pop-up menus, however. As mentioned above, pop-up menus only return the word you have entered. Therefore, you must change the startup macro file. In *VidResize.b* in the startup directory, change *vidFormat* to be a string instead of an integer, placing its default parameter as either NTSC or PAL, both in quotes. You also use an *if/else* statement below it to evaluate *vidFormat* and assign the proper *yRes* value based on the *vidFormat* value.

- 2 Edit the *VidResize.b* file in *startup* to change the *vidFormat* logic to deal with strings instead of integers:

```
image VidResize(
image In=0,
int keepAspect=1, //keeps aspect ratio or not
string vidFormat="NTSC", //This select NTSC or PAL res
float bgRed=0, //if keeping aspect ratio, determines
float bgGreen=0, //the bg color
float bgBlue=0
)
{
    if (vidFormat=="NTSC"){
        yRes=486;
    } else {
        yRes=576;
    }
    // curve int yRes = vidFormat=="0"?486:576;
    Fit1 = Fit(In, 720, yRes, "default", xFilter, 1);
    Resize1 = Resize(In, 720, yRes, "default", 0);
    SetBGColor1 = SetBGColor(Fit1, "rgbaz",
        bgRed, bgGreen, bgBlue, 0,0
    );
    Select1 = Select(keepAspect+1, Resize1, SetBGColor1, 0, 0);

    return Select1;
}
```

- 3 Save the file and start Shake again.

The pop-up menu appears in the parameters.



You can alternatively avoid the use of the *if/else* statement and use something similar to what you had before:

```
curve int yRes = vidFormat == "NTSC"?486:576;
```

The reason the *if/else* is used is that you usually have multiple entries on your pop-up menu, and the conditional expression gets a bit unwieldy, so it's better to create a long *if/else* statement. It's unwieldy and long, like that run-on sentence.

Standard Script Commands and Variables

The following tables include the standard script commands and variables.

Standard Script Commands

Script Controls	Command-Line Equivalent	Notes
SetTimeRange("1-100");	-t 1-100	The time range, for example, frames 1-200. Note this is always in quotes.
SetFieldRendering(1);	-fldr 1	Field Rendering. 0 = off 1 = odd - PAL 2 = even - NTSC
SetFps(24);	-fps 24	Frames per second.
SetMotionBlur(1, 1, 0);	-motion 1 1 , -shutter 1 1	Your three global motion blur parameters.
SetQuality(1);	-fast 0 or 1	Low (0) or high (1) quality.
SetProxyFilter("default");	-proxyfilter	The default filter, taken from the "Filter Characteristics" on page 582.
SetProxyScale(1,1);	-proxyscale 1 1	Proxy scale and ratio for the script. See "About Proxies" on page 103.

Script Controls	Command-Line Equivalent	Notes
SetPixelScale(1,1):	-pixelscale 1 1	Pixel scale and ratio for the script. See “About Proxies” on page 103.
SetDefaultWidth(720); SetDefaultHeight(480); SetDefaultAspect(1); SetDefaultBytes(1); SetDefaultViewerAspect(1);		If a node goes to black or if you create an image node such as <i>RGrad</i> , it takes this resolution by default.
SetFormat("Custom")		Sets your defaults automatically for resolution and aspect from the precreated list of formats. These formats are stored in your <i>startup</i> .h file in the following format:  <i>DefFormatType("Name", width, height, aspectRatio, framesPerSecond, fieldRendering);</i>

## Variables

Common Variables	Notes
width	Returns the width of the current node.
height	Returns the height of the current node.
bytes	Returns the bit depth in bytes, either 1, 2, or 4.
in, outPoint	Returns the in and out frames of the clip in time.
time	The current frame number.
width/2	The center of the image in X.
height/2	The center of the image in Y.
dod[0], dod[1], dod[2], dod[3]	The left, bottom, right, and top edges of the Domain of Definition (DOD) of the current image.
parameterName	The value of a parameter within the same node.
nodeName.parameterName	The value of a parameter in a separate node named <i>nodeName</i> .

## Macro Examples

The following are several examples of macros. Many of these macros can be found in the Cookbook section of the *Shake 3 Tutorials*, but they are not installed by default.

### Parameter Testing: AutoFit

This macro resizes an image to fit a specified size along one dimension. For example, suppose the images in a document are 300, 250, or 166 pixels wide. No matter what size the screen snapshot, you can quickly resize it to one of the standard sizes and easily keep the aspect ratio.

*shake uboat.iff -autofit 166 w*

This calculates an image that is 166 x 125 pixels in size. It is not necessary to calculate the height on your own.

Here is the code:

```
image AutoFit(image img=0, int size=166, int sizeIs=1)
{
    curve w=0;
    curve h=1;
    return Resize(img,
        sizeIs ? size*width/height :size ,
        sizeIs ? size : size*height/width
    );
}
```

The first line creates the parameters. Note that calculate is expecting an integer. The next two lines assign a value to both *w* and *b*. This way, the user can type in 0 or 1 to calculate width or height, or enter *w* or *b*, which is easier to remember. Since Shake assigns values to *w* and *b*, it is able to determine what axis to calculate. The *Resize* function uses embedded *if/then* statements to test the *sizeIs* parameter. Also, you do not change the value of *w* or *b* during processing, so they are not declared as the curve int type of data.

### Text Manipulation I: RandomLetter

This function generates a random letter up to a certain frame, at which point a static letter appears. You can select the color, switch frame, position, and size of the letter, as well as the font. This function uses the standard C *stringf* function to pick a random letter, and assigns it to the variable *rdLetter*. This is pumped into the *Text* function, which uses a conditional expression to evaluate if the current time is before the *staticFrame* time (set to frame 10 by default).

```
image RandomLetter(
    int width=720,
    int height=486,
    int bytes=1,
```

```

const char * letter="A",
int staticFrame=10,
float seed=time,
const char * font="Courier",
float xFontScale=100,
float yFontScale=100,
float xPos=width/2,
float yPos=height/2,
float red=1,
float green=1,
float blue=1,
float alpha=1
)
{
  curve string rdLetter = stringf("%c",
    'A'+(int)floor(rnd1d(seed,time)*26));
  return Text(
    width, height, bytes,
    time < staticFrame? "{rdLetter}": "{letter}",
    font,
    xFontScale, yFontScale, 1, xPos, yPos,
    0, 2, 2, red, green, blue, alpha, 0, 0, 0, 45
  );
}

```

### Text Manipulation II: RadioButton

This is an excerpt from the *RadioButton* function that is used to generate radio buttons for the interface. The radio button code requires four icons to support it: *name.on.nri*, *name.on.focus.nri*, *name.off.nri*, and *name.off.focus.nri*. Since it is tedious to write out four separate files, automatically change the file extensions over time with this excerpt from the macro:

```

image RadioButton(
const char *text="linear",
...
string fileName=text,
string filePath="/Documents/Shake/icons/ux/radio/",
int branch=time,
..
)
{
  curve string fileState =
    branch == 1 ? ".on.nri" :

```

```

        branch == 2 ? ".on.focus.nri" :
        branch == 3 ? ".off.nri" : ".off.focus.nri";
    curve string fileOutName = filePath + "/" +
        fileName + fileState;
    ...
    return FileOut(Saturation1, fileOutName);

```

Since you typically calculate this with a command such as:

```
sbake -radio "HelloWorld" -t 1-4
```

it generates *HelloWorld.on.nri* at frame 1, *HelloWorld.on.focus.nri* at frame 2, *HelloWorld.off.nri* at frame 3, and *HelloWorld.off.focus.nri* at frame 4.

### Text Manipulation III: A Banner

This little trick takes a string of letters and prints it, one letter at a time. It declares a variable within the string section of a *Text* node:

```

Text1 = Text(720, 486, 1,
    {{ string logo = "My Logo Here";
        stringf("%c", logo[(int) clamp(time-1, 0, strlen(logo))])
    }},
    "Courier", 100, xFontScale, 1, width/2, height/2,
    0, 2, 2, 1, 1, 1, 1, 0, 0, 0, 45);

```

This uses *strlen* to determine the length of the string and extract the letter that corresponds to the current frame.

### Text Manipulation IV: Text With a Loop to Make a Clock Face

This eager little example lays out a clock face. A *for* loop is used to count from 1 to 12, and prints the number into a *Text* function with a *stringf* function. (You should just be able to print the value of count with {count}, but it doesn't work. Go figure.) The *cosd* and *sind* functions are also used to calculate the position of the number on the clock face. Keep in mind that zero degrees in Shake points east, as it does in all Cartesian math. The *falloffRadius* serves no purpose in the function except to complete the onscreen control set to give a center and a radius widget:

```

image Clock(
    image In=0,
    float xCenter=width/2,
    float yCenter=height/2,
    float radius=150,
    float falloffRadius=0
)
{

```

```

NumComp=Black(In.width,In.height,1);
for (int count=1;count<=12;++count)
{
    NumComp = Over(
        Text(In.width, In.height, 1, {{ sprintf("%d",count) }},
        "Courier", radius*.25, xFontScale, 1,
        cosd(60+(count-1)*-30)*radius+xCenter,
        sind(60+(count-1)*-30)*radius+yCenter,
        0, 2, 2, 1, 1, 1, 1, 0, 0, 0, 45),
        NumComp
    );
}

return Over(NumComp,In);
}

```

### Text Manipulation V: Extracting Part of a String

This function can be used to extract the filename from a *FileOut* or *FileIn* node so you can print it on a slate. Use it in a *Text* function.

```

const char *getBaseFilename(const char *fileName)
{
    extern const char * strchr(const char *, char);
    const char *baseName = strchr(fileName, '/');
    return baseName ? baseName+1 : fileName;
}

```

To use it, place a line in the text parameter of a *Text* function, such as:

```
{getBaseFilename(FileIn1.imageName)}
```

### Tiling Example: TileExample

This allows you to take an image and tile it into rows and columns, similar to the *Tile* node in the Other tab. However, this one also randomly moves each tile, as well as scales the tiles down. The random movement is generated with the turbulence function (see “Variables, Linking, and Expressions” on page 681). Because of this, it is less efficient than the *Tile* function, which can be viewed in the `<ShakeDir>/include/nreal.h` file.

```

image TileExample(
    image In=0,
    int xTile=3,
    int yTile=3,
    float angle=0,
    float xScale=1,
    float yScale=1,

```

```

        float random=30,
        float frequency=10
    )
    {
        result = Black(In.width, In.height, In.bytes);
        curve float xFactor=(In.width/xTile)*.5;
        curve float yFactor=(In.height/yTile)*.5;

        for (int rows=1; rows<=yTile; ++rows){
            for (int cols=1; cols<=xTile; ++cols){

                Move2D1 = Move2D(In,
                -width/2+xFactor+((cols-1)*xFactor*2)+
                (turbulence(time+(cols+1)*(rows+1),frequency)-.5)*random,
                -height/2+yFactor+((rows-1)*yFactor*2)+
                (turbulence(time+(cols+1)/(rows+1),frequency)-.5)*random,
                angle, 1,
                1.0/xTile*xScale, 1.0/yTile*yScale
                );
                result=Over(Move2D1,result);
            }
        }
        return result;
    }

```

# Rendering

## Chapter Summary

- The Render Parameters Window
- The Render File Menu
- The *FileOut* Function

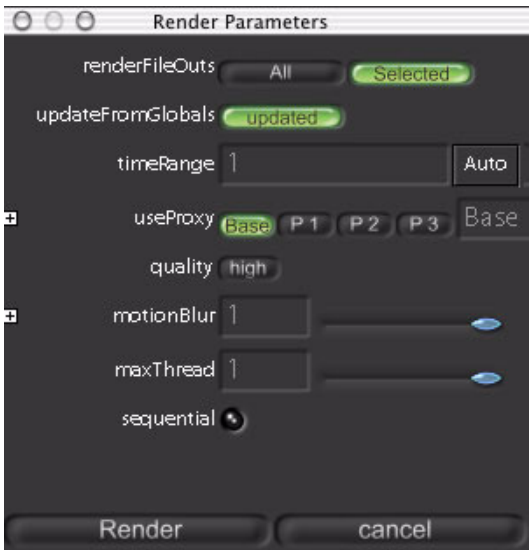
## The Render Parameters Window

When a render is called with the Render menu, the Render Parameters window opens. This window overrides the Global settings for your render. Note that these settings are not saved into the script; they only control how the GUI renders. To render to disk, you must attach an Image-*FileOut* node.

### To render:

- 1 Attach Image-*FileOut* nodes to the nodes you want to render.  
**Note:** To render only specific *FileOuts*, select the *FileOut* nodes in the Node View.
- 2 Choose Render > Render *FileOut* Nodes.

The Render Parameters window opens.



- 3 In the Render Parameters window, ensure the *timeRange* (for example, 1-100) and other parameters are correct, and click Render.

#### Render Parameters Window

Parameters	Notes
<i>renderFileOuts</i>	Indicates whether all <i>FileOut</i> nodes, or just the active nodes, are rendered.
<i>updateFromGlobals</i>	Indicates if your settings match the Globals settings (updated), or if you have modified the settings (update now), in which case the button allows you to update the settings from the Globals.
<i>timeRange</i>	Set a new time range using Shake's standard frame syntax, for example, 1-100 renders 1 to 100, 10-20x2 renders frames 10, 12, 14, up to 20, and so on.
<i>useProxy</i>	Sets your proxy settings.
<i>quality</i>	When this is set to lo (0), anti-aliasing is disabled. This results in poorer image quality, but improved render speed.
<i>motionBlur</i> , <i>shutterTiming</i> , <i>shutterOffset</i>	Can be used to set new motion blur settings for quality, shutter exposure length, and the offset.

Parameters	Notes
<i>maxThread</i>	Specifies how many processors are devoted to the render on a multiprocessor machine.
<i>sequential</i>	If you have multiple <i>FileOut</i> nodes, it may be more efficient to render the nodes sequentially, that is, each file one by one, rather than all at the same time. You may trade output inefficiencies with caching or input redundancy.

## The Render File Menu

Function	Notes
<i>Render Flipbook</i>	Renders a Flipbook of the contents of the current Viewer. It first launches the Flipbook Parameters Window that allows you to override the Global parameters. To cancel the render, press <b>Esc</b> in the Flipbook window. See “The Flipbook” on page 57 for instructions on how to use the Flipbook.
<i>Render FileOut Nodes</i>	Renders <i>FileOut</i> nodes in the Node View. Press <b>F</b> in the Node View to frame all active nodes. You have the option to render only the active <i>FileOut</i> nodes or all <i>FileOut</i> nodes.
<i>Render Proxies</i>	Renders the proxy files for your <i>FileIn</i> nodes, leaving your <i>FileOut</i> nodes untouched. For more information on proxies, see “About Proxies” on page 103.

## The FileOut Function

The *FileOut* function allows you to write images to disk or to another external device. It recognizes local, absolute, or URL paths. For a URL address, place a // in front of the path. To write to another computer, write *//Biggo/Drive/Directory/etc*. For the command line, each *FileOut* is explicitly accompanied by a *-fo* (or *-fileout*). You can add multiple *FileOut* nodes along your command string to output different steps of the command.

For more information on executing scripts, see Appendix B, “The Command-Line Manual,” on page 737.

For more information on the syntax of the filename, see “*The FileIn/SFileIn Function*” on page 79.

For more information on file formats and I/O, see “About Image Input and Output” on page 75.

For more information on time and clips, see “Time Notation for a FileIn” on page 84.

**Supported File Formats**

Parameters	Type	Defaults	Function
<i>imageName</i>		string	The path and filename of the output image.
<i>fileFormat</i>	string	"auto"	If no extension is given, the output format is .iff. To override this behavior, explicitly set the output format.
<i>codec (QuickTime only)</i>	string		A list of all available compressors. The default argument is "Default".
<i>compressionQuality (QuickTime only)</i>	float		The quality of the compression algorithm. A value of 1 is maximum size, maximum quality. 0 is minimum size, minimum quality.
<i>framesPerSecond (QuickTime only)</i>	int		The frames per second for the playback of the QuickTime compression.

**Synopsis**

```
image FileOut ( image, const char * imageName);
```

**Script**

```
image = FileOut( image, "imageName");
```

**Command Line**

*shake input\_image imageName*

or

*shake input\_image -fileout imageName*

**See Also**

“The FileIn/SFileIn Function” on page 79

# Keyboard Shortcuts and Hot Keys

## Hot Key List

In some instances, the hot keys vary on different platforms. In the following hot key lists, the Macintosh commands appear first, followed by a forward slash (/) and the IRIX/Linux commands. In several instances, both platform options work on the Macintosh.

### General Windowing

Command	Notes
<b>Alt</b> -click and drag, or middle-mouse button and drag	Pan the window.
<b>Space bar</b>	Expand window up and down.
<b>Command+Opt</b> -click / <b>Ctrl+Alt</b> -click and drag, or <b>Ctrl</b> +middle-mouse button and drag	Zoom some windows (Curve Editor, Node View, Time Bar).
<b>Esc</b>	Stop processing.
<b>U</b>	Update the Viewer.
<b>Shift</b> -middle-mouse button and drag a tab, or <b>Shift+Opt</b> -click / <b>Shift+Alt</b> -click and drag a tab	Tear off a tab as a floating window.
<b>T</b>	Toggle Time Code/ Frame View in Time-based displays.

### The GUI Viewers

Command	Notes
<b>N</b>	Create/Copy New Viewer.
<b>F</b>	Fit frame to image.

Command	Notes
<b>Shift+F</b>	Fit frame to window.
<b>Ctrl+F</b>	Fit Viewer to image.
<b>Opt-drag / Alt-drag</b>	Pan image.
<b>+ / -</b>	Zoom image in Viewer.
<b>Home</b>	Reset view.
<b>R, G, B, A, C</b>	Toggle Red, Green, Blue, Alpha, and Color views.
<b>1</b>	Toggle A/B image buffers.
<b>2</b>	Toggle color channels.
<b>3</b>	Toggle update mode.
<b>4</b>	Toggle Viewer Scripts.
<b>5</b>	Toggle Compare mode.

### Flipbooks

Command	Notes
<b>.</b> (period; consider it as >)	Play forward.
<b>,</b> (comma; consider it as <)	Play backward.
<b>Shift+&gt;</b>	Ping-pong playback.
<b>Shift-drag</b>	Shuttle playback.
<b>Opt-drag / Alt-drag</b>	Reveal compare buffer, if set.
<b>Ctrl+&gt;</b>	Play through once.
<b>Space bar</b>	Stop playing and rendering.
<b>?</b>	Continue rendering.
<b>R, G, B, A, C</b>	View a channel.
<b>i</b>	Toggle numeric representation of channel values.
<b>Home</b>	Reset zoom to 1.

Command	Notes
- / = (by delete key / by Backspace key)	Zoom in and out.
+ / - (on numeric keypad)	Increase playback rate.
t	Toggle real-time playback.
d	Double/single buffer toggle, SGI only.
Left Arrow / Right Arrow keys	Advance forward or backward through frames.
h	If in compare mode, set to horizontal split.
v	If in compare mode, set to vertical split.
s	If in compare mode, switch split.
f	If in compare mode, fade split, Irix only.
Esc	Close window.

### Function Tabs Workspace

Command	Notes
Click node in Tool Tab	Insert node into tree.
Shift-click node in Tool Tab	New branch.
Ctrl-click node in Tool Tab	Replace current node.
Shift+Ctrl-click in Tool Tab	Create a new node tree.
Opt-drag / Alt-drag	Pan window.

### Curve Editor

Command	Notes
Opt-drag / Alt-drag	Pan window.
Command+Opt-drag / Ctrl+Alt-drag	Zoom window.
+ / - (by delete key / by Backspace key)	Zoom in and out.

Command	Notes
<b>Opt</b> -drag / <b>Alt</b> -drag on numbered axis	Pan only in that direction.
Drag on numbered axis	Scale that direction.
<b>S</b>	Sync time to current keyframe.
<b>T</b>	Toggle time code/frame display.
<b>Shift</b> -drag	Select keys.
<b>Command</b> -drag / <b>Ctrl</b> -drag	Deselect keys.
<b>Command+A</b> / <b>Ctrl+A</b>	Select all curves.
<b>Shift+A</b>	Select all control points on active curves.
<b>B</b>	When drag-selecting keys, this keeps the Manipulator Box active to enable transform controls.
<b>Q</b>	Allows you to pan the selected points.
<b>W</b>	Scales the selected points, with the initial click point the center of scaling.
<b>E</b>	A non-linear scaling of the points.
<b>K</b>	Insert a key at the frame the cursor is at over the selected spline.
<b>V</b>	Toggle visibility of selected curves.
<b>X, Y</b>	Allow movement only on X or Y axis.
<b>H</b>	Flatten tangents horizontally.
<b>Command</b> -click / <b>Ctrl</b> -click on tangent	Break tangent.
<b>Shift</b> -click on tangent	Rejoin broken tangents.
<b>del</b> (by home/end keys) / <b>Delete</b>	Delete active keyframes.
<b>delete</b> / <b>Backspace</b>	Remove curves from Editor (does not delete the curves).
<b>F, Ctrl+F</b>	Frame selected curves.
<b>Shift+F</b>	Frame selected control points.
<b>Home</b>	Frame all curves.

## Node Workspace

Command	Notes
<b>Shift</b>	When you move a node and the grid is enabled, holding <b>Shift</b> allows the node to move freely. When the grid is disabled, holding <b>Shift</b> locks it to the grid. Grid controls are in the Globals tab.
<b>Command+X / Ctrl+X</b>	Removes selected nodes and places them into the paste buffer.
<b>Command+C / Ctrl+C</b>	Copies the selected nodes into the paste buffer.
<b>Command+V / Ctrl+V</b>	Pastes the buffer into the Node View. You can also copy nodes from the Node View and paste them into a text document, or copy the text and paste it into the Node View.
<b>del</b> (by home/end keys) / <b>Delete</b>	Deletes the selected nodes. If the branching is not complicated, the noodles between the parent(s) and children automatically reattach to each other.
<b>Command+Z / Ctrl+Z</b>	Undo up to 100 steps. Rearranging nodes counts as a step.
<b>Command+Y / Ctrl+Y</b>	Redo your steps unless you have changed values after several undos.
<b>+</b>	Zooms into the Node View (also use <b>Command+Opt-click / Ctrl+Alt-click</b> and drag, or <b>Ctrl</b> + middle-mouse button and drag).
<b>-</b>	Zooms out of the Node View (also use <b>Command+Opt-click / Ctrl+Alt-click</b> and drag, or <b>Ctrl</b> + middle-mouse button and drag).
<b>Home</b>	Centers all nodes.
<b>O</b>	Turns on the Overview window to help navigate in the Node View.
<b>F</b>	Frames all selected nodes into the Node View.
<b>Shift+F</b>	Frames all selected nodes into the Node View, but maintains zoom level.

Command	Notes
<b>Command+F / Ctrl+F</b>	<p>Activates nodes according to what is entered in the Search string field in the Select Nodes by Name window.</p> <ul style="list-style-type: none"> <li>■ <i>Select by name.</i> Enter the search string, and it immediately activates nodes that match. For example, if you enter only <i>f</i>, <i>FileIn1</i> and <i>Fade</i> are selected. If you enter <i>fi</i>, just the <i>FileIn1</i> is selected.</li> <li>■ <i>Select by type.</i> Select by node type. For example, enter <i>Transform</i>, and all <i>Move2D</i> and <i>Move3D</i> nodes are selected.</li> <li>■ <i>Select by expression.</i> Allows you to enter an expression. For example, to find all nodes with an angle parameter greater than 180: <i>angle &gt; 180</i></li> <li>■ <i>Match case.</i> Sets case sensitivity.</li> </ul>
<b>Command+A / Ctrl+A</b>	Selects all nodes.
<b>Shift+A</b>	Selects all nodes attached to the current group.
<b>!</b>	All selected nodes are deactivated, all deactivated nodes are activated.
<b>Shift+U</b>	Adds all nodes upstream from the currently active nodes to the active group.
<b>Shift+D</b>	Adds all nodes downstream from the currently active nodes to the active group.
<b>Shift+Up Arrow</b>	Adds one upstream node to the current selection.
<b>Shift+Down Arrow</b>	Adds one downstream node to the current selection.
<b>L</b>	Performs an automated layout on the selected nodes.
<b>Shift+L</b>	Stacks nodes closely to each other vertically.
<b>X</b>	Snaps all selected nodes into the same column.
<b>Y</b>	Snaps all selected nodes into the same row.
<b>G</b>	Visually collapses selected nodes into one node. When saved out again, they are remembered as several nodes. To ungroup the nodes, press <b>G</b> again.

Command	Notes
<b>Shift+G/Consolidates Groups</b>	Temporarily activates Grid when moving nodes, or consolidates two or more groups into a larger group.
<b>Ctrl+G</b>	Ungroups nodes.
<b>Opt+G / Alt+G</b>	Groups nodes together into an open Cluster.
<b>M</b>	Opens a group into a subwindow.
<b>I</b>	Turns off selected nodes when activated. Select the nodes again and press <b>I</b> to reactivate them. You can also load the parameters into the Parameter View and enable <i>ignoreNode</i> .
<b>E</b>	Pulls the active nodes from the tree, and reconnects the remaining nodes to each other.
<b>Shift+M</b>	Launches the MacroMaker with the selected nodes as the macro body.
<b>B</b>	Opens a macro into a subwindow so you can review wiring and parameters. You cannot change the nodes inside of the subwindow.
<b>Opt+B / Alt+B</b>	Closes the macro subwindow when the cursor is placed outside of the open macro.

### Parameter Workspace

Command	Notes
<b>Ctrl-drag</b>	Virtual sliders.
<b>Tab</b>	Advance to next text field.
<b>Shift+Tab</b>	Go to previous text field.
<b>Shift+Left/Right Arrow</b>	Increase text field value by 10 x normal increment.
<b>Ctrl+Left/Right Arrow</b>	Increase text field value by normal increment.
<b>Opt / Alt+Left/Right Arrow</b>	Increase text field value by 1/10 x normal increment.
Right-click	Access pop-up menus.
Right-click on Color Picker	Pops up color palette.

Command	Notes
Drag on parameter name	Copy parameter to target parameter.
<b>Shift</b> -drag on parameter name	Link parameter from target parameter.

## The Command-Line Manual

Shake started in its infancy as a command-line compositor—you can conceivably execute a 500-node script that is typed out in a Terminal. “Conceivably,” but not practically, since nodes such as *Primatte*, *Stabilize*, *QuickPaint*, and *RotoShape* have unwieldy formats. And, of course, you must type out 500 nodes on the command line (which is nobody’s idea of a fun way to spend the day). However, using a Terminal remains an ideal method to execute many daily image-processing functions, such as:

- image resizing
- bit depth or channel reordering
- standardized color corrections (log to lin conversion, gamma corrections, etc.)
- file format conversion
- Flipbooking an image sequence
- execution of scripts
- accessing image information
- quick composites of 3D-rendered elements over backgrounds

These functions can be rendered in the command line more quickly and efficiently than in the interface (if you are comfortable with typing in the Terminal), since they involve relatively straightforward commands. The other major use of the command line is to execute scripts that you have created in the interface and then saved to disk.

This section discusses some general principles about the command-line shell, and then lists several examples. The last section is a list of frequently used functions. Since every node can potentially be used, this not a complete list.

**Note:** All example images are located in the *doc/pix* directory. All examples assume you are in this directory or below, as noted.

## Viewing, Converting, and Writing Images

Shake does three basic things on the command line: It executes image operations, executes scripts, or views images. The following are some simple examples to start the discussion.

- To display images, type the name of the images (in the *doc/pix/truck* directory):

```
shake truck.iff bg.iff sign_mask.iff
```

For instructions on how to use the Flipbook Viewer, see “The Flipbook” on page 57.

- To convert or save a file, append the *-fileout*, or *-fo*:

```
shake truck.iff -fo test.rla
```

The above command saves an image called *test.rla* in the .rla format. For a list of supported formats and their extensions, see “About Image Input and Output” on page 75.

- To compare two images, load the first image and then the second after the *-compare* flag:

```
shake bg.iff -compare sign_mask.iff
```

On most operating systems, use **Opt**-click / **Alt**-click to drag between the two images. Press **H**, **V**, and **F** for horizontal, vertical, and fading wipes. For Linux systems, you must press **Shift+Ctrl**-click and drag.

## Time and Viewing Image Sequences

Shake assumes time is set to frame 1 unless you specify a time range. This is done with the *-t* option, and a frame numbering symbol, such as:

```
shake alien/alien.#.iff -t 1-50
```

The frame numbering is substituted with a symbol (*#* in the above example) which indicates that the file is padded to four places. The following table lists other symbols that can be used in frame numbering.

Shake Format	Reads/Writes
image.#.iff	image.0001.iff, image.0002.iff, etc.
image.%04d.iff	image.0001.iff, image.0002.iff, etc.
image.@.iff	image.1.iff, image.2.iff, etc.
image.%d.iff	image.1.iff, image.2.iff, etc.
image.@@@.iff, image.###.iff	image.001.iff, image.002.iff, etc.
image.%03d.iff	image.001.iff, image.002.iff, etc.
image.10-50#.iff	image.0010.iff at frame 1, 11 at frame 2, etc.

The *-t* option is extremely flexible. You can choose to render frame ranges, stepped ranges, individual frames, or any combination.

Time Range	Number of Frames	Frames Rendered
-t 1-100	100	1, 2, 3... 100
-t 1-100x2	50	1, 3, 5... 99
-t 1-100x20	5	1, 21, 41... 81
-t 1-20,30-40	31	1, 2, 3... 20, and 30, 31, 32... 40
-t 1-10x2,15,18,20-25	13	1, 3, 5... 9, 15, 18, 20, 21, 22... 25
-t 100-1	100	100, 99, 98... 2

Therefore, to convert a sequence of images, use a line similar to the following:

```
shake alien/alien.#.iff -fo test.@.jpg -t 1-30 -v
```

The following table includes some clever renumbering tricks.

Renumbering Clips	
In <i>doc/pix/bus</i> : shake bus2.40-79#.jpg -t 1-40 -fo toto.#.iff -v	Shifts frame numbering to start at 1.
shake bus2.#.jpg -t 40-79 -fo toto.#-39.iff -v	Also shifts frame numbering to start at 1.
shake bus2.#.jpg -t 40-79 -fo toto.80-#.iff -v	Reverses timing starting at frame 1.
shake bus2.#.jpg -t 40-79 -fo toto.118-#.iff -v	Reverses timing within the same frame range.
shake bus2.#.jpg -t 40-79 -fo toto.@.iff -v	Unpads the clip.
shake bus2.40-79x2#.jpg -t 1-20 -fo toto.#.iff -v	Halves the timing of the clip.
shake bus2.40-79x.5#.jpg -t 1-80 -fo toto.#.iff -v	Doubles the timing of the clip.

Appending Functions

Append optional functions with the - (dash) sign to preface each function call, followed by a space for any arguments it might take. Not all functions take arguments, but most do. In the following, *Blur* has an argument of 50 (the amount of blur you want to apply):

```
shake truck.iff -blur 50
```

You can append as many functions as you want:

```
shake truck.iff-blur 50 -invert r -z 2
```

The following is a good example of a common command-line test of 3D-rendered imagery:

```
shake truck.iff -outside sign_mask.iff -over bg.iff
```

As long as there is no ambiguity, it is not necessary to type the entire function name. For example:

```
shake bg.iff-bri 2
```

calls the *Brightness* function since there are no functions starting with *bri*. However, calling:

```
shake bg.iff -con 2
```

informs you that it cannot choose between *Conform*, *Constraint*, *ContrastLum*, *ContrastRGB*, or *Convolve*. To solve the problem, replace your command with enough letters to end the ambiguity:

```
shake bg.iff-contrastl 2
```

This calls the *ContrastLum* function.

In regard to capitalization and function names, function names are always capitalized in the interface. In the command line, function names are always lowercase so you do not have to press the **Shift** key. The exception is when you call a function in quotes, such as the *Linear* function used to animate parameters:

```
shake truck.iff-blur "Linear(0,0@1,20@20)" -t 1-20
```

For more information on animation curves, see “About Splines” on page 502.

Most functions are image manipulators—they modify the image you load. Others are controls to modify how Shake executes the command. For example, *-brightness 2* brightens an image, but *-fps 24* loads a Flipbook preset to 24 frames per second. Although image functions always occur in a linear fashion—the order of commands matters—controls can be placed anywhere. For example, the following lines are identical:

```
shake alien/alien.#.iff-rotate 45 -t 1-50 -cpus 2
```

```
shake -cpus 2 -t 1-50 alien/alien.#.iff-rotate 45
```

However, the following two lines are different:

```
shake truck/truck.iff -pan 200 0 -rotate 135
```

```
shake truck/truck.iff -rotate 135 -pan 200 0
```

If you place a second control on a line, it replaces the previous setting. For example:

```
shake alien/alien.#.iff -t 1-50 -fps 24 -bri 2 -fps 30 -t 10-20
```

plays at 30 frames per second and only render frames 10-20.

## Getting Help

How do you know what *Blur* is expecting? Aside from using the product non-stop for five years, you can also type:

*shake -help blur*

to return what arguments that function takes. *Blur* has six arguments:

*-blur [xPixels] [yPixels] [spread] [xFilter] [yFilter] [channels]*

Often, you don't need to specify all arguments. Shake indicates if it expects more arguments than you have supplied with a nice cheery error message.

For additional information, refer to the relative function pages. For example, the *Blur* function can be found in Chapter 16, "Filters."

You also do not have to know the entire function name. For example, your shrill call for help on blur probably called up about seven different functions, so try:

*shake -help mul*

and you see that Shake simply looks for the string "mul".

How can you quickly launch these docs?

*shake -doc*

## Argument Flow

Looking at the arguments in the *Blur* function (the *Blur* function can be found in Chapter 16, "Filters,"), it expects an image as its first input, which is labeled *In*:

```
image Blur(  
    image In,  
    float xPixels,  
    float yPixels,  
    int spread,  
    const char * xFilter,  
    const char * yFilter,  
    const char * channels  
);
```

However, the command line differs from the interface in that it always assumes the first image argument is coming from the previous argument in a linear flow, so omit the first image argument. Therefore, in *Blur*, the first argument on the command line is the *xPixels* parameter.

Shake assumes the previous image is fed into the function that follows. However, if you have multiple input images, only the last image has the effect applied.

In this example, only *sign\_mask.iff* is blurred:

```
shake truck.iff bg.iff sign_mask.iff -blur 50
```

Occasionally, you want to perform different operations on two different images within the same command. This can be done with the branching commands *-fi* (*FileIn*) and *-label*. Label a branch with the *-label* function, and start a new branch with the *-fi* function. The following example blurs the background, and then assigns it a temporary label of BG. It then reads the truck, brightens it (the truck, but not the background), and composites the result over the blurred background:

```
shake bg.iff -blur 50 -label BG -fi truck.iff -bri 2 -over BG
```

## Scripts

Execute presaved scripts with the *-exec* command. These scripts are usually generated from the interface. The *-exec* function scans the script and executes all *FileOut* nodes. If there are no *FileOut* nodes in the script, the function does nothing.

```
shake -exec my_script.shk -v
```

The *-v* stands for “verbose,” and provides feedback on the progress of the render.

You can override many settings in the script. For example:

```
shake -exec my_script.shk -v -proxys .5 1 -t 20-30 -motion 1 1
```

renders at half-proxy scale, full-proxy ratio, only frames 20 to 30, and activates motion blur without having to edit the script itself.

You can save a script from the command line with the *-savescript* function:

```
shake truck/truck.iff -outside truck/sign_mask.iff -over truck/bg.iff -savescript toto.shk
```

If there is no *FileOut* in the script, you can test the script with the *-script* command. It is similar to the *-exec* function in that it executes *FileOut* nodes. However, it also views every branch in the script, and that can be awkward when rendering. It is really just for testing scripts:

```
shake -script toto.shk
```

## Command-Line Controls

< > indicate a mandatory argument. [] indicate an optional argument.

**Note:** All examples are executed inside *doc/pix/truck*.

The following functions are unique to the command line.

Time Range Control	Notes
-t <frameRange>	See “Time and Viewing Image Sequences” on page 738.

I/O Functions	Notes
<code>-exec &lt;script.shk&gt;</code>	Renders only <i>FileOut</i> nodes in a script. See “Time and Viewing Image Sequences” on page 738.
<code>-fi &lt;image&gt;</code>	<i>FileIn</i> . Allows branching operations. See “Argument Flow” on page 741.
<code>-fo &lt;image&gt;</code>	<i>FileOut</i> . Writes the image to disk in the format of the file extension. If no extension is given, it is saved in .iff format.
<code>-savescript &lt;script.shk&gt;</code>	Saves all of the previous commands into a script for later execution. Also helpful to get scripting formats.
<code>-script &lt;script.shk&gt;</code>	Reads a script and tests it. All branches are viewed; all <i>FileOut</i> nodes are rendered to disk.
Viewing Controls	Notes
<code>-compare &lt;image&gt;</code> Example: <code>shake bg.iff-blur 20 -compare bg.iff</code>	Allows you to compare two images. <b>Opt</b> -click / <b>Alt</b> -click ( <b>Shift+Ctrl</b> -click on Linux) to drag between the images. Press <b>H</b> or <b>V</b> to split the screen between horizontal and vertical splits.
<code>-fullscreen</code>	IRIX only. Blackens the entire screen around the Flipbook.
<code>-fps &lt;framesPerSecond&gt;</code>	Frames per second for the playback. You can also press <b>+</b> / <b>-</b> on the number keypad to set this. The actual and target fps are displayed at the top of the Flipbook.
<code>-gui</code>	Launches your functions in the interface.
<code>-monitor [keepFrames] [w] [h]</code>	With no options, this only displays the current frame, discarding the frame when you go to the next frame. When <i>keepFrames</i> is set to 1, it keep all frames in memory. When set to 0, it discards them. You can also specify the width and height of the monitor window.
<code>-view [zoom]</code>	Can be used to view intermediate stages of a string of nodes. You can also list an optional zoom level for the Viewer.

Information Controls	Notes
<i>-doc</i>	Launches this fine and flawless documentation.
<i>-help</i> , <i>-b</i> [functionName]	Without any arguments, lists all functions available in Shake. When you supply a string, it matches any functions with that string in it. See “Getting Help” on page 741.
<i>-info</i>	Lists size, type, bit depth, and channel information regarding your image.
<i>-version</i>	Prints the version of Shake.
<i>-v</i> , <i>-vv</i>	Verbose. When <i>-v</i> is on, the rendering time is printed for each frame. <i>-vv</i> displays a percentage of render completion as it renders.

Render Controls	Notes
<i>-cpus</i> <procs>	The number of CPUs to use. Default is 1.
<i>-createdirs</i>	Used with <i>-renderproxies</i> to create the necessary subdirectories for the proxies. It does not affect <i>FileOut</i> renders.
<i>-fldr</i> <field>	Turns on field rendering. 1 = odd/PAL, 0 = even/NTSC.
<i>-mem</i> <Mb>	Sets the maximum amount of memory to use.
<i>-motion</i> <quality> <override>	Sets the motion blur quality. In the command line, you must enable <i>override</i> (set it to 1). When executing a script, you either multiply what is saved as the global motion blur value (override = 0), or override it (override = 1).
<i>-node</i> <nodeName>	Only executes that node. For use with <i>-exec</i> .
<i>-nocache</i>	Disables the call to the cache, forcing complete recomputation of each element.
<i>-noexec</i>	Does nothing except compile the script to see if there are errors.
<i>-pixelscale</i> <scale> [ratio]	Sets the <i>pixelscale</i> and <i>ratio</i> . Not frequently used in the command line. See “About Proxies” on page 103.
<i>-proxyscale</i> <scale> [ratio]	Sets the <i>proxyscale</i> and <i>ratio</i> . Does a low-resolution test render of your commands. You can specify two numbers—the scale and ratio, or p1/p2/p3 which uses the preset proxies. See “About Proxies” on page 103.

Render Controls	Notes
<code>-renderproxies</code> [p1] [p2] [p3]	Renders <i>FileIn</i> nodes already saved into a script as the preset proxies. If used without arguments, renders what is saved in the script, otherwise you can specify which subproxies you want, p1, p2 , and/or p3. Usually used with <i>-createdirs</i> .
<code>-sequential</code>	Sequentially executes functions. For use with <i>-exec</i> . When you have multiple <i>FileOut</i> nodes, it may be more efficient to render them one at a time with this flag, rather than simultaneously.
<code>-shutter</code> <shutterTiming> [offset]	Sets motion blur shutter controls.
Quality Controls	
<code>-fast</code> [quality]	When no argument is given (just <i>-fast</i> ), it disables anti-aliasing to speed the render. When using <i>-exec</i> to render a script that has the global quality set to low, you can turn it back on with <i>-fast 1</i> .
Masking Controls	
<code>-evenonly</code> <image> Example: <code>shake bg.iff -rotate 45 -evenonly bg.iff</code>	Only executes the previous commands on the even fields of the image, and mixes it back in with the image you supply.
<code>-isolate</code> <image> <channel> Example: <code>shake bg.iff -blur 50 -isolate bg.iff a</code>	Only executes the previous commands on the channels you specify, and mixes it back in with the image you supply.
<code>-mask</code> <image> [channel] [percent] [invert] Example: <code>shake bg.iff -blur 50 -mask truck.iff</code>	Masks off the previous commands you specify, with the mask coming from image. You can specify the channel and percentage, or invert the mask.
<code>-oddonly</code> <image>	Only executes the previous commands on the odd fields of the image, and mixes it back in with the image you supply.
<code>-roi</code> <left> <bottom> <w> <h>	Only executes a square portion of the image that you specify.

## Labeling Controls

<code>-curve</code> [type] <name> <defaultValue>	Create a global variable available to the script that is loaded or other functions executed. See below for an example.
<code>-label</code> <name>	Allows you to temporarily label an image for later use in the command line.

## Frequently Used Functions

Since you can use *any* of the functions in Shake, the following tables of frequently used functions are only partial lists. However, the tables represent the functions that are both practical to use (not too many parameters), and are useful in the command line, such as file resizing, basic channel manipulation, and quick generation of test elements. Typically, the functions have more options than are listed here. For more information on the functions, see the relative function sections in the appropriate chapter. For example, the *Ramp* image function is located in Chapter 15, “Painting, Rotoscoping, and Other Image Functions.”

## Image Functions

<code>-addtext</code> <text>	A quick way to generate text, as you only have to supply the text itself, without having to enter the width, height, and bit depth.
<code>-black</code> [w] [h] [bitDepth]	A quick way to generate black.
<code>-color</code> <w> <h> <bitDepth> <r> <g> <b> [a] [z]	A color field.
<code>-ramp</code> [w] [h] [bitDepth] [orientation]	A ramp. Use an orientation of 1 for vertical, 0 for horizontal.
<code>-text</code> <w> <h> <bitDepth> <text>	Text.

## Color Functions

<code>-brightness</code> <value>	Multiplies the RGB channels by value.
<code>-contrastlum</code> <contrast> [center]	Performs a contrast, with the pivot point at center.
<code>-delogc</code>	A log to linear color conversion to quickly see Cineon plates.
<code>-gamma</code> <value>	Gamma.

## Color Functions

<code>-logc</code>	A linear to log color conversion to convert files into log color space.
<code>-luminance</code>	A quick function to generate a black-and-white image based on the luminance. Generates a 1-channel image.
<code>-mult &lt;r&gt; &lt;g&gt; &lt;b&gt; [a] [z]</code>	Multiplies color on a per-channel basis.
<code>-saturation &lt;value&gt;</code>	Saturation change. No, really.

## Channel Functions

<code>-colorspace</code> Example: <code>shake bg.iff-colorsp rgb bls</code>	Converts images to a different colorspace. Indicate the source space and the destination space.
<code>-copy &lt;image&gt; &lt;clipmode&gt; &lt;channels&gt;</code>	Copies a channel from your listed image to the incoming image. <i>Clipmode</i> indicates the resolution you want, with 1 as the second image, 0 as the input image.
<code>-forcergb</code>	Forces a BW or BWA image into RGB or RGBA format. This is unnecessary in the GUI and can be handled by <i>FileOut</i> for output, but is awkward in the command line, thus this function's <i>raison-d'etre</i> .
<code>-reorder &lt;channels&gt;</code> Examples: <code>shake truck.iff-reorder aaaa</code> <code>shake truck.iff-reorder rgbl</code> <code>shake bg.iff-reorder grb</code>	Swaps channels. Using letter codes (r, g, b, a, z, l for luminance, and n for null), indicate what channel should go in the r, g, b, a, and z channels.
<code>-setalpha &lt;value&gt;</code>	Sets the alpha to your value. To remove an alpha channel from an image, enter 0.
<code>-switchmatte &lt;image&gt;</code>	Similar to <i>Copy</i> , except only the alpha is swapped. The returned image is premultiplied.

## Compositing Functions

<code>-inside &lt;maskImage&gt;</code>	An inclusion matte. Multiplies the incoming image by the alpha channel of the image you specify.
<code>-isuba &lt;Image&gt;</code>	Extracts the absolute difference between the two images.

### Compositing Functions

<code>-mix &lt;Image&gt; [percent]</code>	Mixes two images together. 50 percent is half of each image.
<code>-outside &lt;maskImage&gt;</code>	An exclusion matte. Multiplies the incoming image by the inverse of the alpha channel of the image you specify.
<code>-over &lt;backgroundImage&gt;</code>	Puts the incoming image, assumed to be the premultiplied foreground, over the background.
<code>-under &lt;foregroundImage&gt;</code>	Puts the incoming image beneath the image you list, which is assumed to be premultiplied.
<code>-screen &lt;image&gt;</code>	Performs the <i>Screen</i> operation, which is good for reflections and glows.

### Resizing Functions

<code>-addborders &lt;xBorder&gt; &lt;yBorder&gt;</code>	Pads the image out with black around the edges.
<code>-crop &lt;left&gt; &lt;bot&gt; &lt;right&gt; &lt;top&gt;</code>	Crops the image to the two corners you specify. The origin is in the lower-left corner.
<code>-fit &lt;xRes&gt; &lt;yRes&gt;</code>	Resizes the image to the resolution you specify, maintaining the aspect ratio.
<code>-resize &lt;xRes&gt; &lt;yRes&gt;</code>	Resizes the image to the resolution you specify.
<code>-window &lt;left&gt; &lt;bot&gt; &lt;xRes&gt; &lt;yRes&gt;</code>	Crops the window at the specified lower-left corner, and to the resolution you specify.
<code>-z &lt;zoom&gt;</code>	A shortcut for <i>Zoom</i> ; both X and Y are zoomed by the same amount.
<code>-zoom &lt;xZoom&gt; &lt;yZoom&gt;</code>	Zooms the image in X and Y independently.

### Filter Functions

<code>-blur &lt;xPixels&gt; [yPixels]</code>	<i>Blur</i>
--	-------------

### Transform Functions

<code>-flip</code>	Turns the image upside down.
<code>-flop</code>	Turns the image backward.

## Transform Functions

<i>-pan</i> <xPan> <yPan>	Pans the image.
<i>-rotate</i> <angle>	Rotates the image.
<i>-scale</i> <xScale> <yScale>	Scales the image without changing the resolution.

## Video Field Functions

<i>-deinterlace</i> <field> [mode]	Deinterlaces the image. 0 equals the even field, 1 equals the odd field. For mode: 0 = replication of the line below. 1 = interpolation of the missing line between two lines above and below. 2 = blur. 50 percent of the line, 25 percent each of the two lines above and below the missing line.
<i>-evenonly</i> <image> Example: <i>shake bg.iff -rotate 45 -evenonly bg.iff</i>	Executes the previous commands on the even fields of the image, and mixes it back in with the image you list.
<i>-field</i> <field>	Extracts just one field, creating a half-height image. 0 = even field, 1 = odd field.
<i>-fldr</i> <field>	Activates field rendering. 1 = odd/PAL, 0 = even/NTSC.
<i>-interlace</i> <image> <clipMode> <field>	Interlaces the two images. When <i>clipMode</i> is set to 1, the BG resolution is taken; when set to 0, the incoming image resolution is taken. 0 = even field, 1 = odd field.
<i>-oddonly</i> <image>	Only executes the previous commands on the odd fields of the image, and mixes it back in with the image you supply.
<i>-pulldown</i> <image> <field> [offset] Example: <i>shake bus/bus2.40-79#.jpg 0 -t 1-50</i>	Acts as a <i>FileIn</i> , converts from 24 fps to 30 fps, interlacing the frames to add the extra frames. 0 = even field, 1 = odd field. The offset amount is the frame that the interlacing starts after the beginning.

## Video Field Functions

<code>-pullup &lt;image&gt; &lt;field&gt; [offset]</code>	Acts as a <i>FileIn</i> . Removes the 3:2 pulldown interlacing, converts from 30 fps to 24 fps. 0 = even field, 1 = odd field. The offset amount is the frame that the interlacing starts after the beginning.
---	--

<code>-swapfields</code>	Switches the odd and even fields.
--------------------------	-----------------------------------

## Other Functions

<code>-average &lt;image&gt; &lt;sStart&gt; &lt;sEnd&gt; &lt;dStart&gt; &lt;dEnd&gt; &lt;bytes&gt; &lt;gain&gt; &lt;scale&gt;</code>	Acts as a <i>-bytes &lt;bytes&gt;</i> , averaging frames down from <i>sStart</i> and <i>sEnd</i> to <i>dStart</i> and <i>dEnd</i> . Usually use 1,1,1 for the last three arguments.
--	---

<code>-bytes &lt;bytes&gt;</code>	1 = 8 bits, 2 = 16 bits, 4 = float.
-----------------------------------	-------------------------------------

## Examples

### Looking at Images

In *doc/pix/truck*:

```
shake bg.iff sign_mask.iff
```

<code>shake *</code>	Uses the UNIX-style wildcard, *. This means "anything."
----------------------	---

<code>shake *.iff</code>	Indicates "anything with an .iff extension."
--------------------------	--

<code>shake bg.iff -compare sign_mask.iff</code>	Compares the two images, using <b>Opt</b> -click / <b>Alt</b> -click to drag between the two. For Linux, <b>Shift+Ctrl</b> -click and drag.
--	---

### Launching Flipbooks

In *doc/pix*:

```
shake alien/alien.#.iff -t 1-50
```

```
shake bus/bus2.#.jpg -t 40-79
```

```
shake bus/bus2.40-79#.jpg -t 1-40
```

```
shake fan/fan.@.iff -t 1-5
```

### Cool Text Tricks

<code>shake -addtext %t -t 1-20</code>	Prints time code.
--	-------------------

<code>shake -addtext %T -t 1-20</code>	Prints full time code.
--	------------------------

### Cool Text Tricks

<i>shake -addtext %f -t 1-20</i>	Prints the current frame.
<i>shake -addtext %F -t 1-20</i>	Prints the padded current frame.
<i>shake -addtext "%D, %d %M"</i>	Prints the current date.

### Converting Image File Formats

In <i>doc/pix/alien</i> : <i>shake alien.#.iff -t 1-50 -fo temp.@.sgi</i>	Writes 50 unpadded images in the .sgi format.
In <i>doc/pix/fan</i> : <i>shake fan.@.iff -t 1-5 -fo fan.#.tif</i>	Writes 5 padded images in the .tif format.

### Adding and Removing Channels

In <i>doc/pix/alien</i> : <i>shake alien.0001.iff -lum -fo toto.iff</i>	Creates a black-and-white image with an alpha channel.
<i>shake alien.0001.iff -reorder rrra -fo tutu.iff</i>	Creates a black-and-white image with an alpha channel.
<i>shake tutu.iff -forcergb</i>	Makes a 1-channel image into a 3-channel image without modifying the image.
<i>shake alien.0001.iff -reorder rgbn</i>	Removes the alpha channel.
<i>shake alien.0001.iff -setalpha 0</i>	Removes the alpha channel.
<i>shake alien.0001.iff -reorder rgbl</i>	Puts the luminance into the alpha channel.

### Changing Bit Depth

In <i>doc/pix/truck</i> : <i>shake bg.iff -bytes 1 -fo bit8.iff</i>	Converts the image to 8 bits (what this image was anyway).
--	--

## Changing Bit Depth

*shake bg.iff -bytes 2 -fo bit16.iff* Converts the image to 16 bits.

*shake bg.iff -bytes 4 -fo bit32.iff* Converts the image to float.

## Manipulating Fields

In *doc/pix/bus*:  
*shake -pulldown bus2.40-79#.jpg 0 -t 1-50 -fo fps30. #.iff -v* 3:2 pulldown, converting from 24 fps to 30 fps. The 1-50 is derived by subtracting 40 from 79, adding 1, and then multiplying that by 1.25.

*shake -pullup fps30. #.iff 0 -t 1-40 -fo fps24. #.iff -v* 3:2 pulldown converting from 30 fps to 24 fps. The 1-40 is derived by dividing 50 (the amount of images) by 1.25.

## Averaging Frames

In *doc/pix/truck*:  
*shake -average bus2.40-79#.jpg 1 40 1 20 1 1 1 -t 1-20* Averages the 40 frames down to 20 frames.

*shake bus2.40-79x2#.jpg -mix bus2.41-79x2#.jpg -t 1-20* Uses the frame numbering steps in the *FileIn* to get the averaging, and mixes them together with the *Mix* function.

## Resizing Images

In *doc/pix/truck*:  
*shake bg.iff -z 3* Zooms the image up by 3.

*shake bg.iff -z .333*  
*shake bg.iff -z 1/3* Both zoom the image down by 3.

*shake bg.iff -zoom 2 1* Zooms the image to twice as wide.

*shake bg.iff -resize 720 486* Zooms the image to NTSC resolution.

*shake bg.iff -fit 720 486* Zooms the image to NTSC resolution, maintaining the original aspect ratio.

## Renumbering Clips

In <i>doc/pix/bus</i> : <i>shake bus2.40-79#.jpg -t 1-40 -fo toto.#.iff -v</i>	Shifts frame numbering to start at 1.
<i>shake bus2.#.jpg -t 40-79 -fo toto.#-39.iff -v</i>	Also shifts frame numbering to start at 1.
<i>shake bus2.#.jpg -t 40-79 -fo toto.118-#.iff -v</i>	Reverses timing.
<i>shake bus2.#.jpg -t 40-79 -fo toto.@.iff -v</i>	Unpads the clip.
<i>shake bus2.40-79x2#.jpg -t 1-20 -fo toto.#.iff -v</i>	Halves the timing of the clip.
<i>shake bus2.40-79x.5#.jpg -t 1-80 -fo toto.#.iff -v</i>	Doubles the timing of the clip.

## Compositing

In *doc/pix/truck*:  
*shake truck.iff -outside*  
*sign\_mask.iff -over bg.iff*  
*shake bg.iff -under truck.iff*

## Animating Parameters

In <i>doc/pix/truck</i> : <i>shake bg.iff -blur</i> <i>"Linear(0,0@1,100@20)" -t 1-20</i>	A linear animation from a value of 0 at frame 1 to a value of 100 at frame 20.
In <i>doc/pix/truck</i> : <i>shake bg.iff -rotate</i> <i>"Linear(0,0@1,360@11)" -t 1-10</i>	Animates a rotation from 0 degrees at frame 1 to 360 degrees at frame 11. Notice only up to frame 10 is rendered, hopefully giving a smooth loop.
In <i>doc/pix/truck</i> : <i>shake bg.iff -rotate</i> <i>"Linear(0,0@1,360@11)" -t 1-10 -motion .5 1</i>	Sets motion blur to half quality.

## Animating Parameters

In <i>doc/pix/truck</i> : <i>shake truck.iff -curve A</i> <i>"JSpline(1,0@1,10@5,300@15)" -pan A A -t 1-15</i>	Creates a temporary curve named A. It has values of 0 at frame 1, 10 at frame 5, and 300 at frame 15. It also continues its slope after frame 15. This curve is then fed into the x and y pan parameters of the <i>Pan</i> .
In <i>doc/pix/truck</i> : <i>shake bg.iff -rotate</i> <i>"time*time" -t 1-75 -motion .5</i> <i>1 -cpus 2</i>	Uses the <i>time</i> variable, placing it in quotes to multiply it by itself. As the frame count increases, the angle increases. Also, both CPUs are enabled for a 2-processor computer.
In <i>doc/pix/truck</i> : <i>shake truck.iff -pan</i> <i>"cos(time*.5)*100"</i> <i>"sin(time)*50" -motion .5 1 -t</i> <i>1-13</i>	Uses the <i>cos</i> and <i>sin</i> functions to animate the truck in a figure eight.
In <i>doc/pix/truck</i> : <i>shake bg.iff -rotate</i> <i>"cos(time*.5)*50+25" -t 1-13 -</i> <i>motion .5 1</i>	

## Substituting Values with -curve

If you have the following in a script called *myscript.shk*:

```
Text1 = Text(720, 486, 1, {{ stringf("myVal = %d %s", myVal, slatestring); }}, "Utopia Regular",  
100, xFontScale, 1, width/2, height/2, 0, 2, 2, 1, 1, 1, 1, 0, 0, 45);
```

you substitute values in with the *-curve* option:

```
shake -curve int myInt 5 -curve string slatestring "SuperSlate" -script myscript.shk
```

and you get a nice image that says "myVal = 5 SuperSlate".

Or you can use:

```
shake -curve int myVal 5 -curve string slatestring "SuperSlate" -text 720 486 1 ' :stringf(\\ "myVal =  
%d\\n%s\\", myVal, slatestring ) ; '
```

Note the extra backslashes—this quotes it once for the shell and then again for Shake. The shell sees `\\` and makes it `\` then `\` and makes it `"` so you get a result of `"` going into the Shake wrapper script, that is then passed properly into the Shake executable. Whew.

## Tips

### File Completion

The following is a shortcut to typing filenames using the file completion feature in the shell. When you press **Tab**, all potential files that match what you type are listed. For example, in *doc/pix*:

```
sbake trTab
```

```
lists
```

```
sbake truck/
```

Press **Tab** again to list all three image files within that directory. If you type another “t,” that is:

```
sbake truck/tTab
```

```
lists
```

```
sbake truck/truck.iff
```

Therefore, you can type the entire line with very few keystrokes. If you type the following:

```
sbake tTabtTabTabTabTabTabTab
```

the following line is listed:

```
sbake truck/truck.iff truck/bg.iff truck/sign_mask.iff
```

So press **Enter**.

### Repeating Previous Commands

There are two ways to repeat previous commands.

- Press the Up arrow on the command line. Each time you press it, the previous command is listed, stepping back through your history. Change portions of the command with the **Left Arrow** and **Right Arrow** keys. Press the **Down Arrow** to take you to the next command in the history list.
- Press the ! key. Press !! to repeat the last command (although pressing the **Up Arrow** is easier). Type !s to repeat the last command that started with “s.”

**Wildcards**

Use wildcards in the command line so you don't need to type much. The following table lists some of the wildcards.

Wildcards	
<p>*</p> <p>Matches everything of any length.</p>	<p>In <i>doc/pix/truck</i>:</p> <p><i>shake *</i></p> <p>Shows all files within that directory simultaneously.</p> <p><i>shake *.iff</i></p> <p>Shows all files within that directory with a .iff extension.</p> <p><i>shake *g*</i></p> <p>Displays <i>bg.iff</i> and <i>sign_mask.iff</i>.</p>
<p>?</p> <p>This is used to match anything for only that position.</p>	<p>In <i>doc/pix/alien</i>:</p> <p><i>shake alien.000?.iff</i></p> <p>Displays the first 9 images simultaneously.</p> <p><i>shake alien.002?.iff</i></p> <p>Displays all frames in the twenties.</p>
<p>[range-range]</p> <p>This can describe a range, between letters or numbers.</p>	<p>In <i>doc/pix/alien</i>:</p> <p><i>shake alien.000[1-5].iff</i></p> <p>This will display the first 5 images simultaneously.</p> <p>In <i>doc/pix/alien</i>:</p> <p><i>shake [l-z]*</i></p> <p>Displays all images that start with the letters l to z, lowercase.</p>

**Math and Expressions**

When performing math on the command line, enclose the math in "quotation marks":

In *doc/pix/truck*:

*shake truck.iff -pan "cos(time\*.5)\*100" "sin(time)\*50" -motion .5 1 -t 1-13*

**Multiple Words**

To use more than one word in a parameter that expects a string (letters), again, use "quotation marks":

*shake -addtext "kilroy was here"*



# Index

- .h files
  - locations of 624
- .plist file 666
- .tcslerc file 665, 668
- 10-bit image files 349–363
  - converting using LogLin 304
- 2K images
  - and caching 102, 639–640
  - and QuickShapes 566
  - and real-time playback 59
  - file sizes of 207, 360
- 3:2 Pulldown 85
- 3D polyhedron (in Primatte) 266
- 3D renders
  - and bit-depth 365
  - and premultiplication 276, 368
  - and the Z channel 365
- 4K/6K images 101

## A

- About Shake menu 16
- Absolute path 695
- Academy format 101
- Accumulate
  - in Pixel Analyzer tool 282
- Activate Viewer Lookup Table button 26
- Add 288–289
  - command-line examples 289
  - command-line usage 288
  - function description 288
  - modifying image channels with 131
  - parameter list 288
  - script 288

- synopsis 288
  - usage described 286
- AddBorders 233
  - command-line usage 233
  - function description 233
  - parameter list 233
  - script 233
  - synopsis 233
- AddMix 161–164
  - example 161
  - function description 161
  - parameter list 163
  - script 164
  - synopsis 163
- Add Notes command 49
- Add Script command 17
- AddShadow 197–198
  - command-line usage 197
  - function description 197
  - parameter list 197
  - script 197
  - synopsis 197
- Add Shapes mode 543
- AddText 164–166
  - command-line usage 166
  - function description 164
  - parameter list 164
  - script 166
  - synopsis 165
- AdjustHSV 316–317
  - command-line usage 317
  - function description 316
  - parameter list 316

- script 317
- synopsis 316
- usage described 286
- AIFF audio files 89
- Alias 670
- All (nodes) command 148
- alpha channel 237
  - and compositing 125
  - removing from an image with SetAlpha 314
  - viewing 128, 149
- als / alias / alsz files 202
- Anamorphic film
  - aspect ratio 232
  - images 226
- Animating parameters 46
  - See also* Curve Editor and Keyframes
- Animation curves 502–509
  - See also* Curve Editor
- AOI 396
- Aperture markings button 26
- Apply Curve function 487
- ApplyFilter 583–584
  - command-line example 584
  - command-line usage 584
  - function description 583
  - script 584
  - synopsis 584
- Area of Interest 396
  - constraint type option 397
- Argument flow 741
- Arithmetic operators 684
  - in Primatte 239
- Aspect ratio
  - and film elements 227
  - and non-square pixels 226
  - and video elements 227
  - common ratios 232
  - for the broadcast monitor 63
  - functions to be cautious with 228
- Aspect ratios
  - anamorphic film 232
- Assign color
  - in Primatte 266
- Associated Nodes command 148
- Atomic-level correctors 153, 286, 288–300

- Atop 167
  - command-line usage 167
  - function description 167
  - math and LayerX syntax 154
  - parameter list 167
  - script 167
  - synopsis 167
- Audio 89–95
  - extracting curves from 94
  - file parameters 93
  - loading and refreshing files 90
  - mixing and exporting 93
  - new functionality 12
  - previewing 91
  - scrubbing 92
  - viewing and editing 92
- Audio panel 90
- Autokey button 42, 46, 437
  - for animating color values 278
- Autosave
  - setting frequency value 635
- Average curves operation 498
- avi files 202, 206

**B**

- Background color (with DOD) 54–55
- Background image (for Primatte) 265
- Backup interval times 40
- Bit depth 363–367
  - changing in command-line mode 751
- Black 558
  - command-line usage 558
  - function description 558
  - parameter list 558
  - script 558
  - synopsis 558
- Black repeat mode 518
- Black shape 550
- Blue channel hot key 279
- Blue screen
  - applying effects to 249–251
  - creating keys from 237
- Blue spill 243
  - removal using ColorX expressions 303

- Blur 585–586
    - and Infinite Workspace 585
    - applying in different color spaces 273
    - command line examples 586
    - command line usage 586
    - function description 585
    - script 586
    - synopsis 585
  - bmp files 202
  - Bookmark button 71
  - Bound mode (for keyframes) 493
  - Box controls
    - customizing 661
  - Box filter 582, 583
  - Brightness 289
    - command-line examples 289
    - command-line usage 289
    - function description 289
    - modified by Lookup function 306
    - modifying image channels with 131
    - parameter list 289
    - script 289
    - synopsis 289
    - usage described 286
  - Broadcast monitor 63
    - aspect ratio 63
    - button 28, 63
    - enabling 63
    - navigating in 64
    - using 61
  - Browser 69–74
    - adding as a pop-up for parameters 642
    - adding personal favorites 642
    - auto launching when creating a node 644
    - automatic file filters for 644
    - navigating in 70
    - opening 69
    - selecting files 72
    - setting default directories 641
    - viewing controls 73
  - Brushes 528
  - Buffer tabs button 25
  - Buttons
    - attaching functions to 646
    - creating on/off buttons 656
    - in the Viewer 24
  - bw files 203, 206
  - Bytes 367
    - avoid breaking concatenation with 276
    - command-line usage 367
    - function description 367
    - parameter list 367
    - script 367
    - synopsis 367
- C**
- Cache 637
    - example of 640
    - settings for high-resolution images 102
  - Calculations
    - order of, in ColorCorrect 325
    - scrubbing 282
  - CameraShake 450–451
    - command-line examples 451
    - command-line usage 451
    - function description 450
    - parameter list 450
    - script 451
    - synopsis 450
  - Cardinal splines 503
  - Change update mode button 25
  - Change Viewer script button 26
  - Channel
    - constraint type option 397
    - functions in command-line mode 747
    - variables 685
  - Channels
    - about 129
    - changing the number of 131
    - editing in command-line mode 751
    - in YUV color space 253
    - modifying 278
    - shuffling with Reorder 312
    - viewing 130
  - Checker 556
    - command-line usage 556
    - function description 556
    - parameter list 556

- script 556
- synopsis 556
- ChromaKey 254–256
  - command-line example 256
  - command-line usage 256
  - function description 255
  - parameters list 255
  - script 256
  - synopsis 255
- CinemaScope format 101
- Cineon files 202–203, 205, 349–363
  - and tracking 416
  - using LogLin for 303
  - working with 273
- Clamp 290
  - command-line examples 290
  - command-line usage 290
  - function description 290
  - parameter list 290
  - script 290
  - synopsis 290
  - usage described 286
- Clip alpha
  - using ColorX expressions 303
- Clipped images 152
- Clipping
  - modified by Lookup function 307
- Clips
  - handles 515
  - in and out points 512
  - looping 91
  - renumbering in command-line mode 753
  - repeat modes 518
  - reversing 519
- Clock icon 45
- Clone brush 529
- Close window button 25
- Clusters (in Node View)
  - defined 142
  - new behavior 14
- CMYToRGB 301
- Codec
  - Flipbook settings 61
  - for QuickTime files 206
- Color 557
  - command-line usage 557
  - function description 557
  - functions in command-line mode 746
  - gradients 561, 573
  - parameter list 557
  - script 557
  - synopsis 557
- Color channels
  - hot keys 279
  - modifying 278
- ColorCorrect 317–334
  - Color Replace 325
  - Curves tab 323
  - function description 317
  - Invert function 324
  - lookup curve 323
  - Misc tab 324
  - order of calculations 325
  - parameter list 325–327
  - preMultiplied 324
  - reorderChannels 324
  - script 331
  - subtabs 317
  - synopsis 328
  - usage described 286
- Color correction
  - and preMult on 3D renders 276
  - concatenation and float 274
  - examining with PlotScanline 342
  - on premultiplied images 311
  - tools 285–347
  - with Infinite Workspace 340
- Color correctors
  - atomic-level 286
  - consolidated 286, 316–347
  - utility 286, 300–315
- ColorMatch 334–336
  - function description 334
  - parameter list 334
  - script 335
  - synopsis 335
  - usage described 287

- Color Palette
  - assigning colors 279
  - pop-up palette 280
- Color Picker 278–279
  - accessing in Parameters View of a node 278
  - assigning to the Parameters Tab 649
  - associated with parameters 45
  - customizing the Shake interface for 280
  - expressions subtree 278
  - modifying channels 278
  - virtual 279, 319
- Color Picker Palette 279
- Color Replace
  - in ColorCorrect 325
- ColorReplace 336–338
  - creating a key with 238
  - for spill suppression 244
  - function description 336
  - masking a color correction 245
  - parameter list 337
  - script 338
  - synopsis 337
  - usage described 287
- Colors
  - assigning in Primatte 266
  - assigning to the Color Palette 279
  - Colorwheel 558
  - creating custom palettes 632
  - for node clusters 142
  - in QuickPaint 529
  - interface settings 626
  - logarithmic and linear 352
  - sampling from the Viewer 279
  - selecting from an image 278
- Color sliders
  - grouping 319
- ColorSpace 300–302
  - command-line examples 302
  - command-line usage 301
  - function description 300
  - parameter list 301
  - script 301
  - synopsis 301
  - usage described 287
- Color space
  - DV footage 252
  - models 320
  - RGB 252
  - Shake's color range 273
- Color swatches (Pixel Analyzer tool) 281
- Color timing 354
- Color values
  - animating 278
  - displaying in the Terminal 24
- ColorWheel 558–559
  - command-line usage 559
  - function description 558
  - parameter list 558
  - script 559
  - synopsis 559
- ColorX 302–303
  - command-line usage 302
  - function description 302
  - script 302
  - synopsis 302
  - usage described 287
  - using expressions with 302
- Command-Line Manual 737–756
- Command-line mode
  - appending functions 739
  - argument flow 741
  - channel functions 747
  - color functions 746
  - compositing functions 747
  - controls 742
  - file formats 751
  - filter functions 748
  - help 741
  - I/O functions 743
  - image functions 746
  - information controls 744
  - labeling controls 746
  - launching the Flipbook 58
  - masking controls 745
  - quality controls 745
  - rendering controls 744
  - resizing functions 748
  - scripts 742
  - timing and image sequences 738

- tips 755
- transform functions 748
- video field functions 749
- viewing, converting, and writing images 738
- viewing controls 743
- Common 168–169
  - command-line examples 169
  - command-line usage 169
  - function description 168
  - parameter list 168
  - script 169
  - synopsis 168
- Compare buffers
  - using 28–30
- Compare Mode button 27
- Compositing
  - about 153
  - and the alpha channel 125
  - elements of any resolution 129
  - in command-line mode 747
  - math description 154
- Compress 291
  - command-line examples 291
  - command-line usage 291
  - function description 291
  - modified by Lookup function 306
  - parameter list 291
  - script 291
  - synopsis 291
  - usage described 287
- Compression controls 202
- Concatenation
  - and masked nodes 276
  - functions that concatenate 275
  - how to avoid breaking 276
  - in color correction 274
  - indicator on nodes 276
  - of transformations 436
- Conditional expression 685
- Conditional statements 701
- Connecting nodes 136
- Consolidated color correctors 286, 316–347
- Const Point Display (Time View) 517
- Constraint 169–172, 221, 397–399
  - and masking 396
  - command-line examples 399
  - command-line usage 172, 399
  - function description 169, 397
  - parameter list 170, 397
  - script 171, 399
  - synopsis 171, 398
- Contextual menu 636
  - See also* Right-click menu
- Continue Rendering command 59
- ContrastLum 292–293
  - command-line examples 292
  - command-line usage 292
  - function description 292
  - parameter list 292
  - script 292
  - synopsis 292
  - usage described 287
- ContrastRGB 293–294
  - command-line example 294
  - command-line usage 294
  - function description 293
  - parameter list 293
  - script 294
  - synopsis 293
  - usage described 287
- Control points
  - inserting in a curve 323
- Convolve 586–588
  - command-line examples 587
  - command-line usage 587
  - example kernel 587
  - function description 586
  - parameter list 586
  - script 587
  - synopsis 587
- Copy 172–173
  - command-line usage 173
  - function description 172
  - modifying image channels with 131
  - parameter list 172
  - script 173
  - synopsis 172
- Copying nodes 136, 147
- Copy parameter command 50

- CornerPin 434, 440, 451–453
    - command-line usage 453
    - function description 451
    - parameter list 452
    - script 453
    - setting up controls 660
    - synopsis 452
  - Create local variable 49
  - Create Viewer 20
  - Crop 209, 233–234, 440
    - command-line example 234
    - command-line usage 234
    - parameter list 234
    - scaling properties of 443
    - script 234
    - synopsis 234
  - Cropping images
    - with Transform nodes 209
  - ct / ct16 files 202
  - Cursor 25
  - Curve Editor 483–509
    - about 483
    - buttons 487
    - curve processing 495
    - editing HueCurves 338
    - hot keys 13, 731
    - loading and viewing curves 483
    - navigating in 485
    - new enhancements 13
    - placing into a Parameter Tab 657
    - right-click menu 486, 502
    - setting colors for 629
    - splitting panes 488
  - Curves
    - adding jitter 497
    - applying functions to 494
    - averaging 498
    - cycle types 501, 507
    - deleting 500
    - in expressions 690–692
    - inserting a control point 323
    - loading and viewing 483
    - modifying 500
    - negating 498
    - resampling 499
    - reversing 497
    - scaling 496
    - shifting 514
    - smoothing 496
  - Curves Tab 323
  - Custom Entries
    - in Pixel Analyzer tool 282
  - Custom icons
    - adding to a button 680
    - locations for 624
  - Customization 623–672
    - alternate icons 645
    - creating on/off buttons 656
    - Function tabs 645
    - push-button toggles 656
    - radio buttons 655
  - Customizing Shake 623–672
  - Custom nodes
    - adding to Mask shape list 386
  - Cyan channel hot key 279
  - Cycle types (for curves) 507
- D**
- D1 NTSC 101, 232
  - D1 PAL 101, 232
  - Dampening
    - modified by Lookup function 307
  - Default filter 583
  - Defaults
    - Color Picker default values 651
    - for formats 634
    - Node View zoom level 649
    - Parameters Tab 649
    - slider ranges 653
    - timecode modes 635
  - Defocus 588–590
    - command-line usage 590
    - function description 588
    - parameter list 589
    - script 589
    - synopsis 589
  - DeInterlace 221, 223
    - command-line usage 223
    - parameter list 223

- script 223
  - synopsis 223
  - Delete Keyframe button 42, 437
  - DepthKey 256–257
    - command-line usage 257
    - function description 256–257
    - MayaDepthKey macro 256
    - parameter list 256–257
    - script 257
    - synopsis 257
  - DepthSlice 257–258
    - command-line usage 258
    - function description 257–258
    - parameter list 257–258
    - script 258
    - synopsis 258
  - dib files 202
  - Digital negatives 351
  - DilateErode 241, 590–591
    - command-line usage 591
    - function description 590
    - in sample matte 241
    - parameter list 590
    - script 591
    - synopsis 590
  - Dirac filter 583
  - Disk-Based Flipbook 60
  - Disk space 77
  - DisplaceX 468–470
    - command-line usage 470
    - function description 468
    - parameter list 469
    - script 470
    - synopsis 469
  - Display DOD and image border button 27
  - Do/While (scripting usage) 702
  - Documentation (Help menu) 21
  - Domain of Definition 50–56
    - and rotoshapes 53
    - assigning 51
    - background color 54
    - color correcting outside of 341
    - DOD button 26
    - in the Viewer 36
    - SetDOD node 53
    - testing rendering times with 51
    - Viewer and display differences 36
    - ways to alter 51
  - Double buffer command 59
  - Double buffering 57, 59
  - dpx files 203
  - dropFrame 221
  - DropShadow 198
    - command-line usage 198
    - function description 198
    - parameter list 198
    - script 198
    - synopsis 198
  - DV footage
    - color space 252
    - keying 251–254
- ## E
- EdgeDetect 591–593
    - command-line usage 593
    - function description 591
    - parameter list 591–592
    - script 592
    - synopsis 592
  - Edge treatment 247
  - Edit Menu 19
  - Edit mode (painting) 528
  - Edit Shapes mode 543
  - Effects
    - applying to blue screen footage 249–251
    - masking 384
  - Emboss 593–594
    - command-line usage 594
    - function description 593
    - modifying image channels with 131
    - parameter list 593
    - script 593
    - synopsis 593
  - Environment variables 664
    - Mac OSX 666
    - testing 670
  - Exit command 18
  - Expand 294–295
    - command-line usage 295
    - function description 294

- parameter list 294
- script 295
- synopsis 294
- usage described 287
- Expressions 681, 683
  - and Lookup function 306
  - arithmetic operators 684
  - channel variables 685
  - command-line usage 683
  - conditional 685
  - curve functions 690–692
  - examples of 683
  - for selecting nodes 136
  - global variables 685
  - image variables 685
  - in command-line mode 756
  - in the Color Picker 278
  - logical operators 684
  - math functions 686
  - noise functions 687
  - precedence of operators 683
  - relational operators 684
  - string functions 689
  - trig functions 688
  - using with parameters 47
  - with ColorX 302

## F

- Fade 295–296
  - and premultiplication 277
  - command-line usage 295
  - function description 295
  - parameter list 295
  - script 295
  - synopsis 295
  - usage described 287
- Favorites List (in Browser) 71
- FG/BG (Tracker) buttons 406
- Field 221, 224
  - Constraint type option 397
  - parameter list 224
- Field chart 32
- Field rendering 217
  - and Viewer zooming 219
  - settings 218

- Fields and JPEG files 219
  - changing in command-line mode 752
  - described 214
- File Browser 69–74
  - Favorites List 71
  - opening 69
- File formats 201–208
  - and temp files 78
  - command-line mode 751
  - file sizes of 207
  - padding image filenames 201
  - QuickTime 201
  - supported 202
  - tracking 424
- FileIn 79, 221, 559
  - and time notation 84
  - and Time View 511
  - command-line usage 89
  - deinterlacing parameter 220
  - parameter list 86
  - proxy parameters 122
  - script 88
  - synopsis 87
  - time shifting 80
- FileIn Trim 518
- File Menu 17
- Filenames
  - conventions in this book 10
- FileOut 560, 727–728
  - command-line usage 560
  - function description 560
  - parameter list 560
  - script 560
  - synopsis 560
- File paths 77
  - conventions in this book 10
- Files
  - selecting 72
  - sizes of 207
- Film
  - anamorphic plates 226
  - and aspect ratio 227
  - and high-resolution images 101
  - using proxies 103

- FilmGrain 594–597
  - function description 594
  - parameter list 594–596
  - script 596
  - synopsis 596
- FilmStock
  - for FilmGrain filter 595
- Filters 581–622
  - and premultiplication 379
  - ApplyFilter 583–584
  - Blur 585–586
  - box 582, 583
  - characteristics 582
  - Convolve 586–588
  - default 583
  - defined 581
  - Defocus 588–590
  - DilateErode 590–591
  - dirac 583
  - EdgeDetect 591–593
  - Emboss 593–594
  - FilmGrain 594–597
  - gaussian 583
  - Grain 597–602
  - IBlur 602–605
  - IDefocus 605–607
  - IDilateErode 607–608
  - impulse 583
  - in command-line mode 748
  - IRBlur 608–611
  - ISharpEN 611–612
  - lanczos 583
  - masking 394, 581
  - masking versions 394
  - Median 613
  - Mitchell 582
  - Mitchell method 583
  - PercentBlur 614
  - Pixelize 615
  - quad 583
  - RBlur 615–616
  - Sharpen 617
  - sinc 583
  - sinc method 582
  - triangle 583
  - ZBlur 617–620
  - ZDefocus 620–622
- Fit 210, 211–212
  - command-line usage 211
  - parameter list 211
  - scaling properties of 443
  - script 211
  - synopsis 211
- Flip 454
  - command-line usage 454
  - script 454
  - synopsis 454
- Flipbook 57–59
  - and QuickTime 60
  - animation controls 59
  - hot keys 730
  - Launch Flipbook button 28
  - launching 57
  - launching from the command line 58, 750
  - memory requirements 59
  - rendering 727
  - Viewer controls 58
- Float bit depth
  - and Logarithmic color 356
  - and third-party plug-ins 362
  - explained 359
- Float calculations 274
- Flop 454
  - command-line usage 454
  - function description 454
  - script 454
  - synopsis 454
- Flush cache command 18
- Fonts
  - defaults for menus 636
  - setting paths 625
- Foreground transparency
  - Keylight plug-in 261
- Format pop-up menu
  - creating custom listings 633
- Four-point tracking 404
- Frame range
  - setting in the Globals tab (timeRange) 100
  - setting in the Time Bar 64

- Frame rate
  - increase/decrease hot keys 59
  - increasing or decreasing 57
- Frames
  - averaging, in command-line mode 752
  - displaying in the Viewer 35
- Frames/timecode button 27
- Freeze repeat mode (playback) 518
- Functions
  - Add 288–289
  - AddBorders 233
  - AddMix 161–164
  - AddShadow 197–198
  - AddText 164–166
  - AdjustHSV 316–317
  - and scripting 699
  - appending in command-line mode 739
  - ApplyFilter 583–584
  - Atop 167
  - Black 558
  - Blur 585–586
  - Brightness 289
  - Bytes 367
  - CameraShake 450–451
  - Checker 556
  - ChromaKey 254–256
  - Clamp 290
  - Color 557
  - ColorCorrect 317–334
  - ColorMatch 334–336
  - ColorReplace 336–338
  - ColorSpace 300–302
  - ColorWheel 558–559
  - ColorX 302–303
  - Common 168–169
  - Compress 291
  - Constraint 169–172, 397–399
  - ContrastLum 292–293
  - ContrastRGB 293–294
  - Convolve 586–588
  - Copy 172–173
  - CornerPin 434, 451–453
  - Crop 233–234
  - declaring in expressions 684
  - Defocus 588–590
  - DeInterlace 223
  - DepthKey 256–257
  - DepthSlice 257–258
  - DilateErode 590–591
  - DisplaceX 468–470
  - DropShadow 198
  - EdgeDetect 591–593
  - Emboss 593–594
  - Expand 294–295
  - Fade 295–296
  - Field 224
  - FileIn 559
  - FileOut 560, 727–728
  - FilmGrain 594–597
  - Fit 211–212
  - Flip 454
  - Flop 454
  - formats of 700
  - Gamma 296
  - Grad 561–562
  - Grain 597–602
  - Histogram 344–347
  - HueCurves 246, 338–340
  - IAdd 173–174
  - IBlur 602–605
  - IDefocus 605–607
  - IDilateErode 607–608
  - IDisplace 470–473
  - IDiv 174–175
  - IMult 175–176
  - Inside 176
  - Interlace 222–223
  - Invert 296–297
  - IRBlur 608–611
  - ISharpener 611–612
  - ISub 177
  - ISubA 178
  - KeyMix 179–180
  - Layer 180–182
  - LayerX 182–183
  - LogLin 303–305
  - Lookup 305–308
  - LookupFile 308–309
  - LookupHLS 309–310
  - LookupHSV 310

- LumaKey 264–265
- Mask 395
- MatchMove 425–430
- Max 183–184
- MDiv 311
- Median 613
- Min 184
- Mix 185
- MMult 311–312
- Monochrome 297
- Move2D 454–457
- Move3D 457–460
- Mult 298
- MultiLayer 186–190
- nested 700
- Orient 461
- Outside 190–191
- Over 191–192
- Pan 462
- PercentBlur 614
- PinCushion 473
- Pixel Analyzer 282–285
- Pixelize 615
- PlotScanline 343–344
- QuickPaint 527–542
- QuickShape 562–580
- Ramp 571–572
- Rand 570–571
- Randomize 474
- RBlur 615–616
- Reorder 312–313
- Resize 212–213
- RGrad 573–574
- Rotate 463–464
- RotoShape 542–555
- Saturation 298–299
- Scale 464–465
- Screen 192
- Scroll 465
- Select 198–199
- Set 313–314
- SetAlpha 314
- SetBGColor 314–315
- SetDOD 465–466
- SFileIn 559

- Sharpen 617
- Shear 467–468
- Solarize 299
- SpillSuppress 246, 271–272
- Stabilize 431–433
- SwapFields 224
- SwitchMatte 193–194
- Text 574–580
- Threshold 299–300
- Tile 580
- TimeX 82–83
- Tracker 433–434
- Transition 520–525
- Turbulate 475–476
- Twirl 476–477
- Under 194–195
- VideoSafe 225
- Viewport 235–236
- Warper 468–470
- WarpX 477–482
- Window 236
- Xor 195
- ZBlur 617–620
- ZCompose 196
- ZDefocus 620–622
- Zoom 213

- Function tabs
  - customizing 645
  - hot keys 731



- Gamma 296
  - command-line usage 296
  - function description 296
  - parameter list 296
  - script 296
  - synopsis 296
  - usage described 287
- Gamma/Offset/LogLin button 32
- Ganging sliders 46
- Gaussian filter 583
- gif files 203
- Global parameters 44, 96
  - default values for 96
- Global variables 685

- Grad 561–562
    - command-line usage 562
    - function description 561
    - parameter list 561
    - script 562
    - synopsis 561
  - Grain 597–602
    - command-line usage 602
    - function description 597
    - graphic example 599
    - parameter list 597–598
    - script 601
    - synopsis 601
  - Graphs of Lookup function 306
  - Green channel hot key 279
  - Green screen keys 237
  - Green spill 243
  - Grid Snap 139
  - Grip to desktop button 25
  - Grouping color sliders 319
  - Groups (nodes) 140
    - exposing parameters for 141
    - new behavior 14
    - setting colors for 629
- H**
- Hard mattes 241
  - Help 65
    - in command-line mode 741
    - menu 21
  - Hermite splines 505
  - Hexadecimal
    - in Pixel Analyzer tool 282
  - Hide Others menu 17
  - Hide Shake 16
  - High-resolution images 101
  - Histogram 344–347
    - button 27, 33
    - command-line usage 347
    - examples of 345–346
    - function description 344
    - parameter list 346
    - script 347
    - synopsis 346
  - History Step in QuickPaint 530
  - HLSToRGB 301
  - Holdout mattes 239, 391
  - Home directory 666
  - Hot keys 729–736
    - conventions for different platforms 10
    - for color channels 279
    - in the Viewer 37
    - QuickPaint 537
  - HSVToRGB 301
  - HSV values, illustrated 254
  - Hue channel hot key 279
  - HueCurves 246, 338–340
    - function description 338
    - parameter list 339
    - script 339
    - synopsis 339
    - usage described 287
- I**
- I/O functions
    - command-line mode 743
  - IAdd 173–174
    - combining with keys 239
    - command-line usage 174
    - function description 173
    - math and LayerX syntax 154
    - parameter list 173
    - script 174
    - synopsis 173
  - IBlur 602–605
    - command-line usage 605
    - function description 602
    - parameter list 603–604
    - script 604
    - synopsis 604
  - Iconify Viewer button 22, 25
  - Icons
    - custom (locations for) 624
    - customizing 645
    - search path 626
    - standard size 647
  - IDefocus 605–607
    - command-line usage 607
    - function description 605
    - parameter list 605–606

- script 606
- synopsis 606
- IDilateErode 607–608
  - command-line usage 608
  - function description 607
  - parameter list 607
  - script 608
  - synopsis 608
- IDisplace 470–473
  - command-line usage 473
  - function description 470
  - parameter list 472
  - script 473
  - synopsis 472
- IDiv 174–175
  - command-line usage 175
  - function description 174
  - math and LayerX syntax 154
  - parameter list 174
  - script 175
  - synopsis 174
- If/Else statements 701
- iff files 202, 205
- Ignoring nodes 138
- Image functions, in command-line mode 746
- Images
  - absolute paths of 695
  - anamorphic 226
  - changing the number of channels 131
  - command-line functions 738
  - high-resolution 101
  - input and output 75
  - interlaced 216
  - reading and writing 75
  - resizing in command-line mode 752
  - saving 75
  - selecting colors from 278
  - unpremultiplying 373
  - viewing channels 130
- Image sequences 76
- Image variables 685
- Importing
  - interlaced images 220
  - Photoshop files 17, 156
- Impulse filter 583
- IMult 175–176
  - combining with keyers 239
  - command-line usage 176
  - function description 175
  - math and LayerX syntax 154
  - parameter list 175
  - script 176
  - synopsis 175
- In/Out Points (clips)
  - time shifting 512
- Include files (for customization) 626
- Infinite Workspace 150–152
  - and color correction 340
  - and the Blur node 585
  - and transformations 455
  - disabling 152
- Information controls
  - command-line mode 744
- InOut Point Display (Time View) 517
- Inputs (nodes)
  - switching 137
- Inside 176
  - combining with keyers 239
  - command-line usage 176
  - function description 176
  - math and LayerX syntax 154
  - parameter list 176
  - script 176
  - synopsis 176
- Interface
  - assigning processors to 635
  - Curve Editor settings 629
  - customization directory location 625
  - customizing for Color Picker 280
  - custom palette 632
  - devices and styles 671
  - Group settings 629
  - loading macros 706
  - node group colors 627
  - saving settings 18, 279
  - tab colors 626
  - Text color settings 630
  - Time Bar color settings 628
  - Time View color settings 631

- Interlace 221, 222–223
    - command-line usage 223
    - parameter list 222
    - script 222
    - synopsis 222
  - Interlaced images
    - common problems with 216
    - importing 220
  - Interleave (for keyframes) 493
  - Interpolating paint strokes 534
  - Invert 296–297
    - function description 296
    - in ColorCorrect 324
    - modified by Lookup function 306
    - parameter list 297
    - script 297
    - synopsis 297
    - usage described 287
  - invertMask 387
  - Invert Selection command 149
  - IRBlur 608–611
    - command-line usage 611
    - function description 608
    - parameter list 609–610
    - script 610
    - synopsis 610
  - IRIX
    - and audio palyback 13
    - and the .tcsorc file 665
    - double buffering 57
    - exiting Shake 18
    - keyboard info 10
    - viewing online documentation 9
  - ISharpener 611–612
    - command-line usage 612
    - function description 611
    - parameter list 611
    - script 612
    - synopsis 612
  - ISub 177
    - command-line usage 177
    - function description 177
    - math and LayerX syntax 155
    - parameter list 177
    - script 177
    - synopsis 177
  - ISubA 178
    - command-line usage 178
    - function description 178
    - math and LayerX syntax 155
    - parameter list 178
    - script 178
    - synopsis 178
- J**
- Jeffress splines 504
  - jif files 203, 205
  - Jitter (curve operation) 497
  - JPEG files 202–203, 205
    - and fields 219
- K**
- Kernel
    - Convolve example 587
  - Keyboard commands
    - conventions in this guide 10
  - Keyboard shortcuts 729–736
    - for thumbnails 144
  - KeyChew macro 241
  - Keyers
    - ChromaKey 254–256
    - combining 239
    - LumaKey 264–265
    - Primatte 265–271
    - SpillSupress 271–272
  - Keyframes 488–500
    - adding 488
    - copying and pasting 499
    - delete button 42
    - deleting 500
    - inserting for tracking 416
    - Manipulator Box 490
    - modifying 490
    - move modes 492
    - selecting 489
    - text fields 492
    - toggling on and off 43
    - transform hot keys 491

- Keying 237
  - defined 237
  - DV footage 251–254
  - edge treatment 247
  - from a green or blue screen 237
  - reflections 240
  - with ColorReplace node 238
- Keylight plug-in 237, 258–264
  - parameter list 259–262
  - script 263
  - synopsis 262
- KeyMix 179–180
  - command-line usage 180
  - example 179
  - function description 179
  - math and LayerX syntax 155
  - parameter list 179
  - script 180
  - synopsis 180
  - understanding its math 370
- Keys 237
- Key tab 237, 254

## L

- Labeling controls
  - command-line mode 746
- Lanczos filter 583
- Launch Flipbook button 28
- Layer 180–182
  - function description 180
  - parameter list 181
  - script 182
  - synopsis 181
- Layer nodes
  - for combining keys 242
- Layers
  - in Photoshop 156
  - masking 391
- LayerX 182–183
  - command-line usage 183
  - function description 182
  - parameter list 182
  - script 183
  - synopsis 182

- Layout controls 140
- Light Hardware mode 648
- Linear color space
  - converting using LogLin 303
- Linear drag mode 530
- Linear Lookup
  - modified by Lookup function 307
- Linear splines 505
- Linking
  - nodes 143
  - parameters 47, 681
  - tracks 411
- Linux
  - and audio playback 13
  - and the .tcschrc file 665
  - exiting Shake 18
  - keyboard info 10
  - overlay info hot key 58
  - viewing online documentation 9
- Load/Save button 39
- Loading
  - expressions 50
  - interface settings 18
  - Tracks 411
- Lock Direction button 42, 437
- Lock Tangents button 553
- Logarithmic color space 349–363
  - and float bit depth 356
  - converting using LogLin 303
  - correcting in 352
- Logical operators 684
- LogLin 303–305, 351
  - command-line usage 305
  - function description 303
  - parameter list 304
  - rolloff parameter 362
  - script 305
  - synopsis 304
  - usage described 287
- Lookup 305–308
  - command-line usage 308
  - function description 305
  - graphs and expressions 306
  - script 308
  - synopsis 307

- Lookup curve
  - example 308
  - in ColorCorrect 323
- LookupFile 308–309
  - command-line usage 309
  - function description 308
  - parameter list 309
  - script 309
  - synopsis 309
  - usage described 287
- LookupHLS 309–310
  - command-line usage 310
  - function description 309
  - parameter list 309
  - script 310
  - synopsis 309
  - usage described 287
- LookupHSV 310
  - command-line usage 310
  - function description 310
  - parameter list 310
  - script 310
  - synopsis 310
  - usage described 287
- Lookup Table button 26
- LumaKey 264–265
  - command-line usage 265
  - function description 264–265
  - parameter list 264
  - script 265
  - synopsis 265
- Luminance 278
  - hot key 279
  - In YUV color space 252

## M

- Machine settings
  - directory location 625
- Macintosh
  - keyboard info 10
  - setting environment variables 665
- macroCheck 99, 637
- MacroMaker 673
  - image of 675
  - parameter list 675–676

- Macros 673, 703
  - adding custom icons to 680
  - attaching button toggles 713
  - attaching color pickers and subtrees 712
  - attaching parameter widgets 709
  - attaching pop-up menus 716
  - basic structure 703
  - creating on/off buttons for 711
  - creating the node structure 673
  - default width and height 633
  - directory location 625
  - examples 720
  - KeyChew 241
  - loading into the interface 706
  - making 674
  - MayaDepthKey 256
  - missing from a script 99, 637
  - modifying 677
  - modifying the macro interface 679
  - opening 143
  - setting default values for 708
  - setting slider ranges 710
  - text manipulation 720–723
  - typical errors 707
- Magenta hot key 279
- Magnet drag mode 530
- Make Macro command 150
- Manipulator Box (for keyframes) 490
- Mask
  - command-line mode 395
  - command-line usage 396
  - function defined 395
  - parameter list 395
  - script 396
  - synopsis 396
- Masking
  - a layer 391
  - an effect 384
  - controls in command-line mode 745
  - defined 237, 383
  - filters 394, 581
  - for images with no alpha channel 391
  - with the Constraint node 396
- Masks
  - inverting 387

- using different channels for 391
  - when not to use 387
- Mask shape list
  - adding custom nodes to 386
- Match case 136
- MatchMove 425–430
  - function description 425
  - general description 402
  - parameter list 425
  - script 429
  - synopsis 428
  - workflow 403
- Math
  - compositing 154
  - functions and definitions 686
- Matrix (in ColorCorrect) 318
- Mattes
  - garbage matte 268
  - hard 241
  - holdout matte 239, 268
  - touch-up tools 241
- Max 183–184
  - combining with keyers 239
  - command-line usage 184
  - function description 183
  - math and LayerX syntax 155
  - parameter list 183
  - script 183
  - synopsis 183
- Maya
  - file compatibility 205
  - importing Z channel info 256
- MayaDepthKey macro 256
- MDiv 311
  - command-line usage 311
  - function description 311
  - parameter list 311
  - script 311
  - synopsis 311
  - usage described 287
- Median 613
  - command-line usage 613
  - function description 613
  - parameter list 613
  - script 613
  - synopsis 613
- Memory
  - and the cache 637
  - settings 639
- Menus
  - (Mac OS X only) 16
  - adding functions to 637
  - default font sizes for 636
  - Edit 19
  - File 17
  - Help 21
  - Render 20
  - Tools 19
  - Viewers 20
- Min 184
  - command-line usage 184
  - function description 184
  - math and LayerX syntax 155
  - parameter list 184
  - script 184
  - synopsis 184
- Min/Max Basis
  - in Pixel Analyzer tool 282
- MirrorInc repeat mode 519
- Mirror repeat mode 519
- Misc tab 324
- Mitchell filter 582, 583
- Mix 185
  - command-line usage 185
  - function description 185
  - math and LayerX syntax 155
  - parameter list 185
  - script 185
  - synopsis 185
- mixPercent 521
- MMult 311–312
  - command-line usage 312
  - function description 311
  - parameter list 311
  - script 312
  - synopsis 312
  - usage described 287

- Monitors
    - aspect ratio 63
    - Broadcast monitor 63
    - extra Viewers for 22
    - setting resolution for IRIX 672
    - setting up dual monitors 672
  - Monochrome 297
    - command-line usage 297
    - function description 297
    - modifying image channels with 131
    - parameter list 297
    - script 297
    - synopsis 297
    - usage described 287
  - Motion blur 445–448
  - Mouse
    - functions described 10
    - setup for left-handed users 671
  - Move2D 435, 440, 443, 454–457
    - command-line usage 457
    - function description 454
    - parameter list 455
    - script 457
    - synopsis 456
  - Move3D 457–460
    - command-line usage 460
    - function description 457
    - parameter list 457–459
    - script 460
    - synopsis 459
  - mov files 206
  - Moving nodes 136
  - mray files 202
  - Mult 298
    - command-line usage 298
    - function description 298
    - modifying image channels with 131
    - parameter list 298
    - script 298
    - synopsis 298
    - usage described 287
  - MultiLayer 186–190
    - button control 188
    - function description 186
    - parameter list 189
    - script 190
    - synopsis 189
    - with Photoshop files 156
- N**
- Natural splines 502
  - Negate (curve operation) 498
  - Nested functions 700
  - New Canvas button 531
  - Nodes
    - Add 288–289
    - AddBorders 233
    - AddMix 161–164
    - AddShadow 197–198
    - AddText 164–166
    - AdjustHSV 316–317
    - aligning 140
    - ApplyFilter 583–584
    - Atop 167
    - Black 558
    - Blur 585–586
    - Brightness 289
    - Bytes 367
    - CameraShake 450–451
    - Checker 556
    - ChromaKey 254–256
    - Clamp 290
    - cloning 681
    - Color 557
    - ColorCorrect 317–334
    - ColorMatch 334–336
    - ColorReplace 336–338
    - ColorSpace 300–302
    - ColorWheel 558–559
    - ColorX 302–303
    - Common 168–169
    - Compress 291
    - connecting 136
    - Constraint 169–172, 397–399
    - ContrastLum 292–293
    - ContrastRGB 293–294
    - Convolve 586–588
    - Copy 172–173
    - copying 136, 147
    - copying and pasting 136

- CornerPin 434, 451–453
- creating 133
- creating multiples with one button 648
- Crop 233–234
- cutting 147
- Defocus 588–590
- DeInterlace 223
- deleting 135, 147
- DepthKey 256–257
- DepthSlice 257–258
- DilateErode 590–591
- disconnecting 135
- DisplaceX 468–470
- DropShadow 198
- EdgeDetect 591–593
- Emboss 593–594
- Expand 294–295
- extracting 134
- Fade 295–296
- Field 224
- FileIn 559
- FileOut 560, 727–728
- FilmGrain 594–597
- finding 19, 148
- Fit 211–212
- Flip 454
- floating 134
- Flop 454
- Gamma 296
- Grad 561–562
- Grain 597–602
- grouping 140
- Histogram 344–347
- HueCurves 246, 338–340
- IAdd 173–174
- IBlur 602–605
- IDefocus 605–607
- IDilateErode 607–608
- IDisplace 470–473
- IDiv 174–175
- ignoring 138
- IMult 175–176
- inserting 133
- Inside 176
- Interlace 222–223
- Invert 296–297
- IRBlur 608–611
- ISharpEN 611–612
- ISub 177
- ISubA 178
- KeyMix 179–180
- Layer 180–182
- LayerX 182–183
- linking 143
- loading into a Viewer 138
- loading parameters 138
- LogLin 303–305
- Lookup 305–308
- LookupFile 308–309
- LookupHLS 309–310
- LookupHSV 310
- LumaKey 264–265
- match case 136
- MatchMove 425–430
- Max 183–184
- MDiv 311
- Median 613
- Min 184
- Mix 185
- MMult 311–312
- Monochrome 297
- Move2D 454–457
- Move3D 457–460
- moving 136
- Mult 298
- MultiLayer 186–190
- organizing 139
- Orient 461
- Outside 190–191
- Over 191–192
- Pan 462
- Pasting 136
- PercentBlur 614
- PinCushion 473
- Pixel Analyzer 282–285
- Pixelize 615
- PlotScanline 343–344
- QuickPaint 527–542
- QuickShape 562–580
- Ramp 571–572

- Rand 570–571
- Randomize 474
- RBlur 615–616
- Renaming 139
- Reorder 312–313
- replacing 134
- Resize 212–213
- RGrad 573–574
- Rotate 463–464
- RotoShape 542–555
- Saturation 298–299
- Scale 464–465
- Screen 192
- Scroll 465
- Select 198–199
- select by expression 136
- select by name 136
- select by type 136
- selecting and deselecting 135
- selecting downstream 149
- selecting upstream 149
- Set 313–314
- SetAlpha 314
- SetBGColor 314–315
- SetDOD 465–466
- setting interface colors for 627
- SFileIn 559
- Sharpen 617
- Shear 467–468
- Solarize 299
- SpillSuppress 246, 271–272
- Stabilize 431–433
- SwapFields 224
- switching inputs 137
- SwitchMatte 193–194
- Text 574–580
- Threshold 299–300
- thumbnails 144
- Tile 580
- TimeX 82–83
- Tracker 433–434
- Transition 520–525
- Turbulate 475–476
- Twirl 476–477
- Under 194–195
- ungrouping 141
- VideoSafe 225
- Viewport 235–236
- Warper 468–470
- WarpX 477–482
- Window 236
- Xor 195
- ZBlur 617–620
- ZCompose 196
- ZDefocus 620–622
- Zoom 213
- Node View
  - and Tool Tabs 132
  - contextual menu 147
  - hot keys 733
  - new improvements 14
  - Overview 132
  - setting default zoom level 649
- Noise functions 687
- Noodles 125
  - deleting 137
  - disconnecting 135
- nreal.h file 36, 587, 624
- nri files 202
- nrui.h file 624
- NTSC 101, 232

**O**

- Offset
  - hot key 279
  - setting up controls 662
  - tracking 404–405
  - usingAdjustHSV 316
- Offset Track button 405
- On/Off buttons
  - creating 656
  - creating in macros 711
- Open Script command 17
- Orient 461
  - command-line usage 461
  - function description 461
  - parameter list 461
  - script 461
  - synopsis 461

- Out Points (clips) 516
- Output resolution 442
- Outside 190–191, 391
  - combining with keys 239
  - command-line usage 191
  - function description 190
  - math and LayerX syntax 155
  - parameter list 190
  - script 191
  - synopsis 191
- Over 191–192
  - combining with keys 239
  - command-line usage 192
  - function description 191
  - math and LayerX syntax 155
  - parameter list 191
  - script 192
  - synopsis 191
  - understanding its math 370

## P

- Padding (when naming image files) 201
- Paint brush 529
- Painting 527–542
  - See also* QuickPaint
- Paint mode 528
- Paint strokes
  - attaching to a tracker 532
  - converting from Frame to Persistent 536
  - converting modes 533
  - Interpolating 534
  - modifying 531
  - modifying parameters 533
- PAL 101, 232
- Palettes
  - custom 632
  - in Color Picker 279
- pal files 204, 206
- Pan 441, 462
  - command-line usage 462
  - function description 462
  - parameter list 462
  - script 462
  - synopsis 462

- Panning controls
  - setting up 659
- Parameters
  - animating 46
  - animating in command-line mode 753
  - editing 44
  - grouping in a subtree 653
  - linking 47, 681
  - linking at different frames 682
  - viewing for a node 43
  - viewing in grouped nodes 141
- Parameters Tab
  - adding a Curve Editor to 657
  - right-click menu 49
  - setting defaults 649
- Parameters View 43
- Parameter widgets
  - attaching to macros 709
- Parent/Child relationships for shapes 551
- Pasting nodes 136, 147
- Paths 77
  - and FileIn/SFileIn 79
  - preference files for 624
- pbm / ppm / pnm / pgm files 203
- PDF browser path 9
- Peak Meter 91
- PercentBlur 614
  - command-line usage 614
  - function description 614
  - parameter list 614
  - script 614
  - synopsis 614
- Per-channel view 34
- Persist toggle 530
- Photoshop
  - files 206
  - importing images from 17, 160
  - layering modes 156
  - new image support 11
- Photoshop transfer modes 160
- pic files 203
- PinCushion 473
  - command-line usage 473
  - function description 473
  - parameter list 473

- script 473
- synopsis 473
- Ping-Pong playback command 59
- Pixel Analyzer 282–285
  - parameter list 283–284
  - saving data 283
  - script 285
  - synopsis 284
- Pixel Analyzer tool
  - Accumulate 282
  - Custom Entries 282
  - Hexadecimal 282
  - Min/Max Basis 282
  - Mode 281
  - Reset 282
  - Value Range 282
- Pixelize 615
  - command-line usage 615
  - function description 615
  - parameter list 615
  - script 615
  - synopsis 615
- pix files 202
- Platforms 7
- Play 59
  - backward 59
  - Flipbook controls 57
  - once 59
- PlotScanline 343–344
  - button 26, 32
  - command-line usage 344
  - examples of 342–344
  - function description 343
  - parameter list 344
  - script 344
  - synopsis 344
  - using to understand color correction 342
- Plug-ins
  - Keylight 258–264
  - Primatte 265–271
- png files 203
- Point controls
  - setting up 663
- Point modes for RotoShapes 549
- Points
  - contextual menu 552
  - modifying on a paint stroke 531
- Pop-up Color Palette 280
- Pop-up menus 49
  - attaching to macros 716
  - default font sizes for 636
- PostScript fonts 574
- Precedence
  - in expressions 683
- Preferences 623
  - color 626
  - environment variables 664
  - general 626
  - search path 624
  - templates directory 664
  - Viewers 658
- Premultiplication
  - and 3D renders 277
  - and filters 379
  - and the Fade function 277
  - explained 368
  - managing 377
  - typical problems 369
  - with Over 379
- Previewing audio 91
- Primatte plug-in 237, 265–271
  - 3D polyhedron 266
  - arithmetic operator 239
  - assigning colors 266
  - parameter list 266–270
  - script 271
  - supplying the background image 265
  - synopsis 270
  - using the arithmetic parameter 242
- Processors
  - assigning to interface 635
- Proxies 103–123
  - and offline images 112
  - and YUV files 113
  - changing preset defaults 109
  - compatibility with other functions 119
  - customizing the presets 108
  - defined 103
  - interactiveScale 105

- network rendering 117
- parameters list 120
- pregenerating 112
- pregenerating with a script 118
- remastering resolutions with 214
- rendering on the command line 115
- setting 106
- Proxy button 41
- proxyRatio 227–228
- psd files 203, 206
  - See also* Photoshop
- Pulldown / Pullup 85
- Purge Memory Cache command 18
- Push (for keyframes) 494
- Push-button toggles
  - creating 656



- qnt files 204
- qtl files 204, 206
- Quad filter 583
- Quadrants
  - expanding the window 16
- Quality controls
  - command-line mode 745
- QuickPaint 527–542
  - See also* Painting
  - attaching a tracker 532
  - brushes 528
  - converting strokes 533
  - Edit mode 528
  - Frame mode 530
  - function parameters 537
  - general description 527
  - History Step button 530
  - hot keys 537
  - interpolating strokes 534
  - Interpolation mode 530
  - Linear drag mode 530
  - Magnet drag mode 530
  - new behavior 12
  - Paint mode 528
  - parameters list 537
  - Persist mode 530
  - picking a color 529

- resolution 527
- script 541
- StrokeData synopsis 540
- synopsis 539
- QuickShape 562–580
  - command-line usage 570
  - function description 562
  - parameter list 569
  - script 570
  - synopsis 569
- QuickShapes
  - animating 567
  - Build mode 563
  - creating 563
  - modifying 564
- QuickTime
  - playback controls 62
  - files 201, 203, 206
  - and Disk-Based Flipbooks 60
- Quit 17

## R

- Radio buttons
  - creating 655
- Radius controls
  - setting up 664
- RAM
  - for Flipbook playback 59
- Ramp 571–572
  - command-line usage 572
  - function description 571
  - parameter list 571
  - script 572
  - synopsis 572
- Rand 570–571
  - command-line usage 571
  - function description 570
  - parameter list 570
  - script 571
  - synopsis 571
- Randomize 474
  - command-line usage 474
  - function description 474
  - parameter list 474

- script 474
- synopsis 474
- Random noise
  - using ColorX expressions 303
- Range 322
- raw files 203, 206
- RBlur 615–616
  - command-line usage 616
  - function description 615
  - parameter list 616
  - script 616
  - synopsis 616
- Real-time playback 57, 59
- Real-time toggle 59
- Re-center image 58
- Recover Script 18
- Red channel
  - correction with ColorX expressions 303
  - hot key 279
- Redo 19
- Reference pattern 403
- Reflections
  - keying 240
- Relational operators 684
- Reload Script 17
- Renaming nodes 139
- Render
  - All FileOuts 148
  - command-line mode 744
  - Disk Flipbooks 20, 60, 148
  - FileOut Nodes 20
  - Flipbook 20, 147
  - proxy parameters 122
- Render Disk Flipbook 20, 148
- Rendered images
  - color correcting with MDiv 311
- Render File menu 727
- Rendering 725–728
  - field rendering 217
  - field rendering settings 218
  - parameters 726
- Render Menu 20
- Render Parameters window 725
- Render Proxies 20
- Render Selected FileOuts 148
- Renumbering clips
  - in command-line mode 739
- Reorder 312–313
  - command-line examples 313
  - command-line usage 313
  - function description 312
  - modifying image channels with 131
  - parameter list 312
  - script 313
  - synopsis 313
  - usage described 287
- Reordering
  - channels 312
  - in ColorCorrect 324
  - using ColorX expressions 303
- Repeat modes (for clips) 518
- Replace (for keyframes) 494
- Resample (curve operation) 499
- Reset Track button 405
- Reset Viewer button 28
- Resize 210, 212–213
  - command-line usage 212
  - filters used 582
  - parameter list 212
  - scaling properties of 443
  - script 212
  - synopsis 212
- Resizing
  - images 208
  - in command-line mode 748
  - thumbnails 146
  - windows 66
- Resolution 208–214, 527
  - and QuickPaint 527
  - and transformations 442
  - changing 208
  - functions for modifying 211
  - of Viewers 22
  - remastering with proxies 214
  - setting for the Viewer 634
- Retiming 80
  - parameters for 82
- Reveal brush 529
- Reverse (curve operation) 497
- Reversing a clip 519

- rgb files 203, 206
- RGBToCMY 301
- RGBToHLS 301
- RGBToHSV 301
- RGBToYIQ 301
- RGBToYUV 301
- RGrad 442, 573–574
  - command-line usage 574
  - function description 573
  - parameter list 573
  - script 574
  - synopsis 573
- Right-click
  - in the Viewer 22
- Right-click menu
  - See also* Contextual menu
  - adding functions to 636
  - Clear Expression 50
  - Clear Tab 49–50
  - control points 547
  - Curve Editor 486, 502
  - Node View 147
  - Parameters tab 49
  - RotoShapes 547
  - Time View 516
  - Tracking 410
  - transform controls 551
  - Viewer Lookup Table 31
- rla files 203, 206
- Rotate 441, 463–464
  - command-line usage 464
  - function description 463
  - parameter list 463
  - script 463
  - synopsis 463
- Rotate controls
  - onscreen controls 440
  - setting up 662
- RotoShape 542–555
  - Add Shapes mode 543
  - parameter list 554
  - script 555
  - synopsis 554
- RotoShapes
  - Add Shapes mode 543
  - animating 547
  - creating and modifying 546
  - deleting 547
  - Edit Shapes mode 543
  - inserting points 545
  - modifying points and tangents 545
  - new behavior 12
  - parameter list 554
  - Point modes 549
  - skeleton relationships 551
  - transform controls 550–551
  - Viewer buttons 553
- rpf files 204, 206

## S

- Sample (color) From Viewer 279
- Saturation 298–299
  - command line usage 299
  - function description 298
  - parameter list 298
  - script 298
  - synopsis 298
  - usage described 287
- Saturation hot key 279
- Saving
  - expressions 50
  - interface settings 18
  - Pixel Analyzer data 283
  - scripts 17
  - track files 411
- Saving tracks 423
- Scale 442, 464–465
  - command-line usage 465
  - function description 464
  - parameter list 464
  - scaling properties of 443
  - script 465
  - synopsis 464, 465
- Scale (curve operation) 496
- Scaling
  - and transformations 442
  - functions compared 443
  - setting up controls for 659

- Screen 192
  - command-line usage 192
  - function description 192
  - math and LayerX syntax 155
  - parameter list 192
  - script 192
  - synopsis 192
- Scripting
  - commands 718
  - conditional statements 701
  - controls 692
  - if/else statements 701
  - principles of 692
- Script manual 692–724
- Scripts
  - and functions 699
  - commenting 701
  - data types 697
  - described 692
  - do/while 702
  - in command-line mode 742
  - loading into the interface 695
  - missing macros 99, 637
  - nested functions 700
  - recovering 18
  - variables and data types 694
  - while 702
- Scroll 465
  - command-line usage 465
  - function description 465
  - parameter list 465
  - script 465
- Scrubbing
  - Color Picker 279
  - with ChromaKey 254
- Scrub command 59
- Search path 624
  - for icons 626
- Search region
  - scaling 404
- Search region (Tracker) 403
- Select 198–199
  - command-line usage 199
  - function description 198
  - script 199
  - synopsis 199
- Selecting files 72
- Selecting nodes 135
- Sequences
  - images 76
  - reading and writing 74
- Services menu 16
- Set 313–314
  - command-line usage 313
  - function description 313
  - modifying image channels with 131
  - parameter list 313
  - script 313
  - synopsis 313
  - usage described 287
- SetAlpha 314
  - command-line usage 314
  - function description 314
  - parameter list 314
  - script 314
  - synopsis 314
  - usage described 287
- SetBGColor 314–315
  - command-line usage 315
  - function description 314
  - parameter list 315
  - script 315
  - synopsis 315
  - usage described 288
- SetDOD 465–466
  - command-line usage 466
  - function description 465
  - parameter list 466
  - scaling properties of 443
  - script 466
  - synopsis 466
- Settings
  - color 626
  - Curve Editor colors 629
  - environment variables 664
  - general 626
  - Group colors 629
  - memory 639

- preference files 624
  - Text colors 630
  - Time Bar colors 628
  - Time View colors 631
- SFileIn 79, 559
  - and retiming 80
- sfx files 204
- sgi files 202, 203, 206
- sgiraw files 203, 206
- Shake
  - customizing 623–672
  - getting Help 21
  - Home page 21
  - new features 11
  - Reference Guide conventions 10
  - supported platforms 7
  - training and resources info 9
  - user interface 15–67
- Shapes 542–555
  - See also* RotoShapes)
- Sharpen 617
  - command-line usage 617
  - parameter list 617
  - script 617
  - synopsis 617
- Shear 467–468
  - command-line usage 468
  - function description 467
  - parameter list 467
  - script 467
  - synopsis 467
- Shift Curves 514
  - and timing changes 514
- sidefx files 204
- Sinc filter 582, 583
- Skeleton relationships for shapes 551
- Slider ranges
  - beyond limits 44
  - setting in macros 710
  - setting up 653
- Sliders
  - ganging 46
- Smear 448
- Smooth (curve operation) 496
- Smoothing
  - curves 496
  - tracks 411
- Smudge brush 529
- Solarize 299
  - command-line usage 299
  - function description 299
  - parameter list 299
  - script 299
  - synopsis 299
  - usage described 288
- Sound files 12
  - See also* Audio
  - extracting curves from 94
- Spatial filter 581
- Spawn Viewer Desktop 20, 22
- SpillSuppress 246, 271–272
  - command-line usage 272
  - function description 271–272
  - parameter list 271–272
  - script 272
  - synopsis 272
- Spill suppression 237, 243
- Spline Lookup
  - modified by Lookup function 307
- Splines 502–506
  - Cardinal 503
  - Hermite 505
  - Jeffress 504
  - linear 505
  - natural 502
  - step 506
- Squeezed (anamorphic) images 227
  - compositing with square pixel images 229
  - rendering 231
- Stabilize 431–433
  - function description 431
  - general description 401
  - parameter list 431
  - script 432
  - synopsis 431
  - workflow 402
- Startup directory 625
- Step splines 506

- Stop Playing/Rendering 59
- String functions 689
- Stylus 671
- subPixelResolution (for tracking) 412
- SwapFields 221, 224
- SwitchMatte 193–194
  - command-line usage 194
  - function description 193
  - modifying image channels with 131
  - parameter list 193
  - script 193
  - synopsis 193

## T

- Tablet usage 45
- Tabs
  - arranging 66
  - attaching functions to buttons 646
  - color settings 626
  - setting up node columns 645
- Tangents
  - editing 492
- Targa files 202
- tdi files 204
- tdx files 204
- Temperature channel hot key 279
- Template preference files 664
- Temporary files 77
- Text 442, 574–580
  - command-line usage 579
  - cool command-line tricks 750
  - function description 574
  - manipulating in macros 720–723
  - parameter list 577
  - script 579
  - setting colors for 630
  - synopsis 578
- tga files 204
- Threshold 299–300
  - command-line usage 300
  - Constraint type option 397
  - function description 299
  - parameter list 299
  - script 300
  - synopsis 300
  - usage described 288
- Thumbnails 144
  - keyboard shortcuts 144
  - Resizing 146
  - transparency of 146
- tiff files 204
- Tile 580
  - command-line usage 580
  - function description 580
  - parameter list 580
  - script 580
  - synopsis 580
- Tiling with a macro 723
- Time Bar 46, 64
  - frame range settings 640
  - setting colors for 628
- Timecode
  - default mode 641
  - displaying in the Viewer 35
  - setting default modes 635
- Time range
  - command-line mode 739
- Time shifting 80
- Time View 511–525
  - setting colors for 631
  - Viewing nodes in 512
- TimeX 82–83
  - command-line usage 83
  - expressions 82
  - function description 82
  - parameter list 83
  - script 83
  - synopsis 83
- Title Bar 16
- TMV color space 320
- Toggles
  - creating 656
- Tools Menu 19
- Tool Tabs
  - and Node view 132
  - described 15
  - Key 254
- Track Backward/Track Forward buttons 405
- Track Display button 406

- Tracker 433–434
    - attaching to a paint stroke 532
    - function description 433
    - general description 401
    - how it works 412
    - reference pattern 403
    - script 433
    - search region 403
    - synopsis 433
  - Tracking 401–434
    - and Cineon files 416
    - file format 424
    - linking to track data 421
    - manually inserting keyframes 416
    - new enhancements 12
    - off-frame points 417
    - onscreen controls 403
    - paint strokes 532
    - parameters 406–410
    - reference pattern 413
    - removing jitter 422
    - right-click menu 410
    - strategies 413
    - two-point 423
    - Viewer buttons 405
    - workflow 402
  - Tracks
    - averaging 411, 419
    - clearing 411
    - linking 411
    - loading 411
    - modifying 418
    - saving 411, 423
    - smoothing 411
    - smoothing curves of 420
  - Transform
    - controls for RotoShapes 551
    - functions in command-line mode 748
    - nodes for cropping 209
  - Transformations 435–445
    - concatenation of 436
    - inverting 436
    - multiple transforms in a tree 438
    - onscreen controls 437
  - Transition 520–525
    - creating your own 522
    - function description 520
    - parameter list 525
    - script 525
    - synopsis 525
  - Triangle filter 583
  - Trig functions 688
  - Trim controls 518
  - TrueType fonts 574
  - Turbulate 475–476
    - command-line usage 476
    - function description 475
    - parameter list 475
    - script 475
    - synopsis 475
  - Turbulent noise 303
  - Tweaker windows 44
  - Twirl 476–477
    - command-line usage 477
    - function description 476
    - parameter list 476
    - script 477
    - synopsis 476
  - Two-point tracking 423
- ## U
- ui.h file 280
  - ui directory 625
  - Ultimatte plug-in 238
  - UNC filename convention 641
  - Under 194–195
    - command-line usage 194
    - function description 194
    - math and LayerX syntax 155
    - parameter list 194
    - script 194
    - synopsis 194
  - Undo 19
    - changing levels of 40
    - setting levels 635
  - Undo/Redo button 40
  - Ungroup 141
  - Unpremultiplying 373

- Update button 41
- User Directory 624
- User interface 15–67
- Utility correctors 286, 300–315

## V

- Value channel hot key 279
- Value Range
  - in Pixel Analyzer tool 282
- Variables 682, 718, 719
  - adding to the interface 681
  - environment 664
  - for channels 685
  - image variables in each node 685
  - recognized by Shake 670
- Video 214–225
  - and aspect ratio 227
  - aspect ratio 231
  - common problems 216
  - field functions in command-line mode 749
  - field rendering settings 218
  - fields described 214
  - functions 221–225
  - importing interlaced images 220
  - timecode display 35
- Video functions
  - Constraint 221
  - DeInterlace 221
  - dropFrame 221
  - Field 221
  - FileIn 221
  - Interlace 221
  - SwapFields 221
  - VideoSafe 221
- VideoSafe 221, 225, 315
  - command-line usage 225
  - parameter list 225
  - script 225
  - synopsis 225
  - usage described 288
- Viewer
  - contextual menus 38
  - tracking buttons 405
- Viewer buttons 24, 25–28
- Viewer Channel button 24–25
- Viewer controls (for the Flipbook) 58
- Viewer DOD 30
  - button 26
- Viewer lookups 30
- Viewers 21–39
  - creating 22
  - deleting 22
  - expanding 22
  - hot keys 37
  - loading images into 22
  - minimizing 22
  - on second monitors 22
  - preferences 658
  - resolution of 22
  - selecting 22
  - setting max resolution 658
- Viewer script controls
  - activating 31
- Viewer scripts 26, 30
  - creating your own 36
- Viewers Menu 20
- Viewing an alpha channel 128
- Viewing controls
  - command-line mode 743
- Viewport 235–236
  - command-line usage 236
  - for cropping 209
  - function description 235
  - parameter list 235
  - scaling properties of 443
  - script 236
  - synopsis 236
- Virtual Color Picker 279, 319
- VLUTs 30
  - activating 31
  - creating your own 36
  - using to simulate logarithmic space 35

## W

- Wacom tablet 45
- Warper 468–470
- Warps
  - general description 468

WarpX 477–482

- command-line usage 482
- function description 477
- parameter list 481
- script 481
- synopsis 481

Wav audio files 89Wedging 354While (scripting usage) 702Wildcards 74, 756Window 236

- command-line usage 236
- for cropping 209
- function description 236
- parameter list 236
- scaling properties of 443
- script 236
- synopsis 236

Windows

- expanding 66
- OS functions 67
- panning 66
- resizing 66
- zooming 66

**X**Xor 195

- command-line usage 195
- function description 195
- math and LayerX syntax 156
- parameter list 195
- script 195
- synopsis 195

xpm files 204

## Y

YCrCb defined 252Yellow channel hot key 279YIQToRGB 301yuv files 204

- defined 252

YUVToRGB 301

## Z

ZBlur 617–620

- command-line usage 620
- function description 617
- parameter list 619
- script 619
- synopsis 619

Z channel

- blurring with ZBlur 617
- button 27, 33
- defocusing 620
- display properties 130
- for DepthKey function 256
- for DepthSlice function 257
- inverting 296
- multiplying 298
- placing in RGB channels with Reorder 312

ZCompose 196

- command-line usage 196
- function description 196
- math and LayerX syntax 156
- parameter list 196
- script 196
- synopsis 196

ZDefocus 620–622

- command-line usage 622
- function description 620
- parameter list 620–621
- script 622
- synopsis 621

Zoom 210, 213

- command-line usage 213
- parameter list 213
- scaling properties of 443
- script 213
- synopsis 213

Zoom in 58, 147Zoom out 58, 147