

ATI RenderMonkey IDE

RenderMonkey Introduction	3
Interface overview	4
Application Menu	5
The File menu	5
The Edit menu.....	5
The View menu	6
The Window menu.....	6
The Help menu	7
Application Toolbar	8
RenderMonkey File Format	9
Workspace View	9
Managing Effect Groups	10
Managing Effects	12
Managing Variables	13
Predefined Variables.....	15
Managing Render Passes.....	18
Managing Pixel and Vertex Shaders	20
Managing Render State Block	21
Application Preferences	22
Cycle time for pre-defined 'time' variable.....	23
Auto Refresh	23
Default Directories	24
Rendering Refresh Rate	24
Reset Camera on Effect Change	24
Modules	25

Preview Module	25
Output Module	30
Stream Mapping Module.....	30
Shader Editor Module	32
Editing Assembly	34
Editing the High Level Shading Language.....	36
Editing Notes	40
Editing Variables	40
Scalar Variables.....	41
Vector Variables.....	41
Matrix Variables	42
Color Variables	42
Model Variables	43
Texture, Cubemap and Volume Texture Variables	43
Renderable Texture Support.....	45
Editing Renderable Texture	48
Editing Render Target.....	49
Editing Camera Node Settings.....	49
Artist Editor.....	52
Editing variables in the artist editor module	54
Colors	54
Vectors.....	54
Scalars.....	55
<i>RenderMonkey Support and Feedback.....</i>	<i>56</i>

RenderMonkey Introduction

Many of the current challenges facing 3D graphics application developers are centered on creating and using programmable graphics shaders.

These programmable graphics shaders are at the heart of all future graphics chips. With the introduction of the Radeon 9000, they are now supported on the entry level PC and will soon trickle down to all other devices.

The developers with the ability to create and use these programmable shaders will be able to take advantage of all that the hardware offers and create applications that redefine the art of real-time graphics.

In order to help developers unlock the creative potential of these chips, ATI Technologies is developing a family of tools - the RenderMonkey™ toolset.

The RenderMonkey Integrated Development Environment (IDE) is the first of these components to be released.

Our motivation for developing the RenderMonkey IDE is to provide

- A powerful programmer's development environment for creating shaders.
- A standard delivery mechanism to facilitate the sharing of shaders amongst developers.
- A flexible, extensible framework that supports the integration of custom components and provides the basis for future tools development.
- An environment where not just programmers but artists and game designers can work to create mind-blowing special effects.
- A tool that can easily be customized and integrated into a developer's regular workflow.

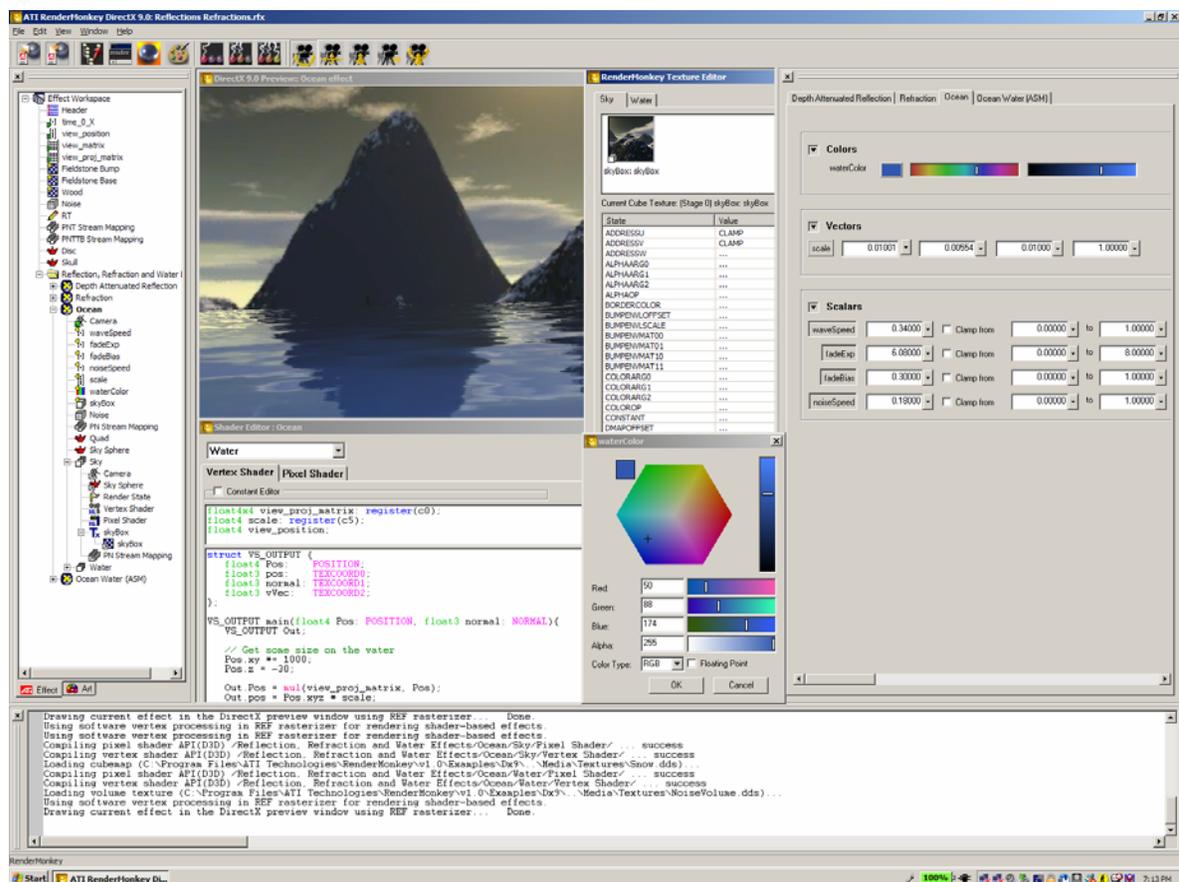
This release of the RenderMonkey IDE provides support for all shader models provided with DirectX 9.0 (including HLSL).

Interface overview

The RenderMonkey application interface has been designed to be intuitive for any developer that has used an IDE tool such as Microsoft® Visual Studio®.

The main interface consists of:

- a Workspace View which shows the *Effect Workspace* being edited
- an Output Window for compilation results and text messages from the application
- a Preview Window used to preview effects being edited
- Other editor modules such as editors for shaders themselves, or GUI editors for shader parameters. Shader parameters can be tagged as “*Artist Editable*” and then edited in a coherent way using either the artist editor module, or through the Workspace View Artist Tab.

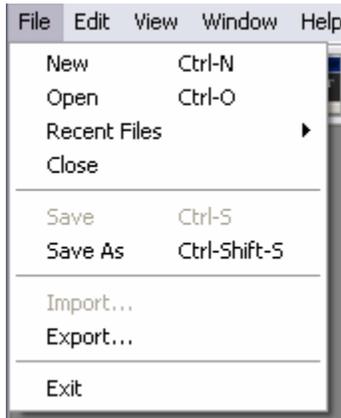


Application Menu

The application menu contains standard File, Edit, View, Window, and Help menu options.



The File menu



New (Ctrl-N) command creates a new Effect Workspace. By default, the new workspace starts out empty, with just the workspace node itself. If the user selected *New* while working on an unsaved workspace, the application will prompt the user to save currently opened workspace first.

Open (Ctrl-O) opens an existing Effect Workspace file.

Recent Files menu provides the user a list of 5 recently used RenderMonkey workspace files.

Close command closes the currently opened workspace. If the current workspace the user has been working on has been modified without being saved, the application will prompt the user to save the workspace prior to closing it.

Save (Ctrl-S) command saves the currently opened workspace.

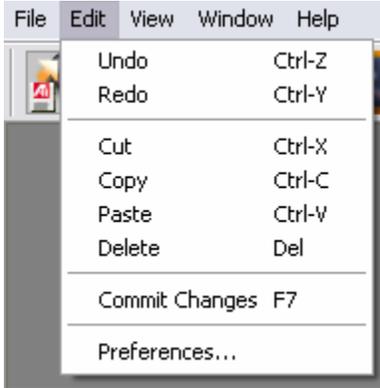
Save As (Ctrl-Shift-S) will prompt the user to change the current file name and/or location.

RenderMonkey IDE allows developers to create custom plug-ins supporting their own file format. To accomplish that, they can create importer and exporter plug-ins to convert the data from custom file formats to RenderMonkey run-time database format. The *Import* command allows the user to load a custom data file using one of the plug-ins, if any are found, and convert the data to RenderMonkey database format. Notice that version 1.0 of RenderMonkey does not include any importer plug-ins, however, examples will be provided with the SDK.

Similarly, the user can select *Export* command to convert from the RenderMonkey data format to a different file format. Version 1.0 includes ability to export from RenderMonkey native data format to Microsoft DirectX 9.0 .fx file format. There are certain restrictions on the syntax of data presented in the RenderMonkey workspace in order to output valid FX files.

Exit will close the application, prompting the user to save any currently opened Effect Workspace.

The Edit menu



The *Edit* menu contains the following commands: *Undo*, *Redo*, *Cut*, *Copy*, *Paste*, *Delete*, *Commit Changes*, and *Preferences* options.

The *Undo* command (Ctrl-Z) allows the user to undo the last undoable operation, and return RenderMonkey to its previous state. The application allows the users to undo all operations on the nodes done in the workspace view, for example, deleting, pasting, renaming of any nodes in the workspace. The shader editor also supports standard set of text operations undo functionality.

The *Redo* menu option (Ctrl-Y) will redo an undone operation.

Undo / Redo operations will cover node renaming, cut, copy, and paste operations, changing the Active Effect, and adding or deleting nodes.

The *Cut*, *Copy*, *Paste* and *Delete* operations will work on individually selected nodes, as well as with text in the text editors. Please note that these operations will not work on the Effect Workspace node itself since at any time you may not have more than one opened effect workspace. Note that you can cut / copy / paste nodes across multiple files in a single instance of the application.

Commit Changes (F7) will compile and commit the currently active shader in the shader editor.

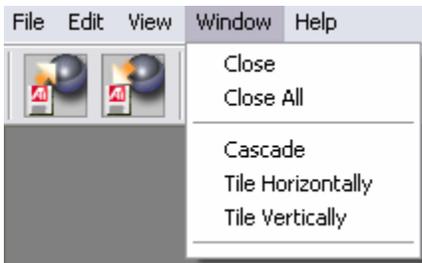
The *Preferences...* menu option will open up the application Preferences Dialog (see the section on *Application Preferences* below for more details).

The View menu



The *View* menu allows the user to open or close main RenderMonkey modules windows, such the workspace view window, the output window, the preview window and the artist editor window.

The Window menu



The *Window* menu contains standard window options such as *Close*, *Close All*, *Cascade*, *Tile Horizontally*, and *Tile Vertically* options. A list of opened windows will also be maintained at the bottom of the menu, allowing the user to quickly bring an opened window into focus by selecting the window from a list.

The Help menu



The help menu gives access to the *About* dialog. The about dialog contains version information, as well as contact information for application support or feedback:



Application Toolbar

The application has a toolbar for commonly used functions.



Open Workspace (File Open)



Save Workspace (File Save)



Toggle Workspace Window (View Workspace)



Toggle Output Window (View Output)



Toggle Output Window (View Preview)



Toggle Artist Editor Window (View Artist Editor)



Compile Active Shader (F6) (please refer to the Shader Editor Module section for details on this and the next two commands)



Compile All Shaders in Active Effect (F7)



Compile All Shaders in the Workspace (F8)



Rotate Camera (please refer to the Preview Module section for details)



Pan Camera



Zoom Camera



Camera Home



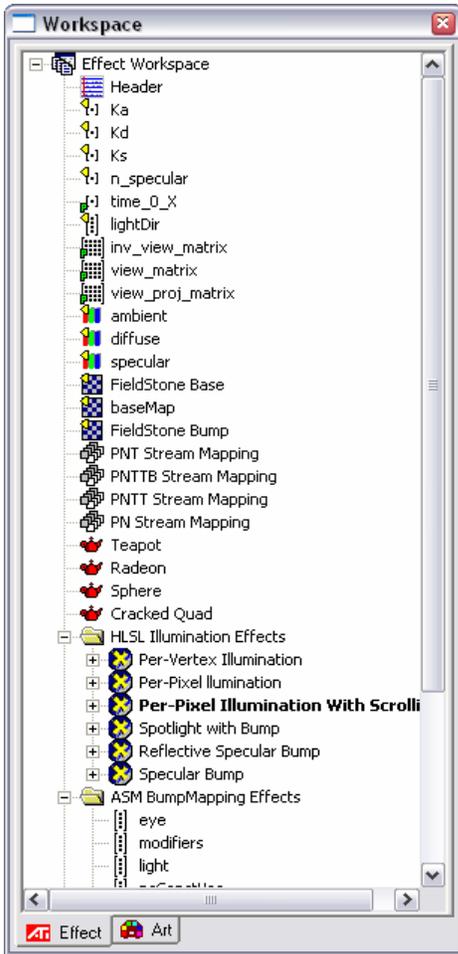
Overloaded Camera Mode

RenderMonkey File Format

Each set of visual effects in RenderMonkey is encapsulated in a single XML workspace, the “.rfx” file. All of the information necessary for recreation of each effect, excluding the actual textures and model data, is stored in this single file. It is user-readable and any game developer can create a converter from the RenderMonkey’s file format into their game engine script format. We chose XML to store effect workspaces for several reasons. Most importantly, XML is an industry standard with parsers readily available (RenderMonkey uses the Microsoft XML parser; there are other alternatives freely available). It allows easy data representation and it is user-extensible. Best of all, any user can just open an XML RenderMonkey file and read the file directly in Internet Explorer – it’s just another ASCII file format.

Workspace View

The Workspace View is a dockable window usually positioned on the left of the main interface containing a tabbed tree control which provides a high level view of the effect database.



The workspace view can be used to access all elements in the *Effect Workspace*. The idea is that individual effects are going to be grouped by their common attributes.

There are two tabs in the workspace tree view: *Effect* tab and *Art* tab. The *Effect* tab is used to view and modify the entire workspace – with all variables and passes visible. The *Art* tab is used to view only the artist-editable variables that are present in the workspace. The Art tab will only allow the user to edit artist-editable nodes, without the ability to add, delete, rename, etc. This functionality allows the programmers to develop the full effect and then allow the artists to modify the effect’s rendering output without worrying about accidentally modifying the effect’s contents.

The actual *Effect Workspace* consists of these elements:

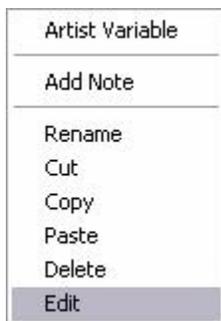
- Variables (shader parameters)
- Stream Mapping nodes
- Models
- Effect Groups
- Default Effects

-  Renderable Textures
-  Notes

All individual items in the workspace are referred to as nodes.

An *Effect Group* is a mechanism for organizing effects in a large workspace by the effect type or in any other way intuitive to the user of that workspace. For example, *Effect Groups* could be used to facilitate fallback versions of a particular effect to support a wide range of hardware and SDK versions.

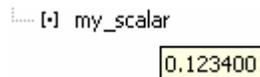
Effect Groups consist of one or more individual *Effects*. An *Effect* contains all information needed to implement a real-time visual effect, such as the shaders, their parameters and related states. Each effect consists of one or more passes (draw calls) and parameters global for each effect.



Standard node operations are supported by the workspace view, such as cut / copy / paste/ delete / rename for individual nodes. Each node represents a chunk of data in the effect workspace. If an editor / viewer module exists for a node, double clicking on the node will launch the module. The editor / viewer module may also be launched by right clicking on the desired node and selecting “*Edit*”.

All nodes in the workspace view have tool tips designed to instantly give detailed information as to the node contents. Such, for variables, the user will be able to see the value(s) stored in that variable node, for textures or model nodes, the user will be able to view the file that that resource uses.

For example, a tool tip for a scalar node will look like this:



Managing Effect Groups

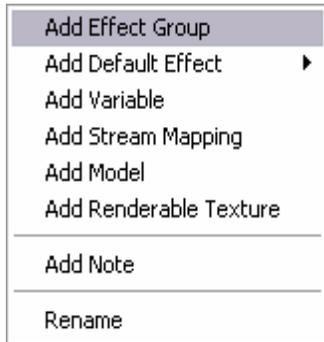
Each *Effect group* is used to encapsulate a series of related effects. For example, you may want to group all effects that use a noise function to render perturbation effects, such as clouds, fire or plasma, in one single effect group. Another good use for this node is to group various implementations of a single effect for fallback rendering in your engine.

The actual *Effect Group* consists of these elements:

- Variables (shader parameters)
-  Stream Mapping nodes
-  Models
-  Effects
-  Renderable Textures

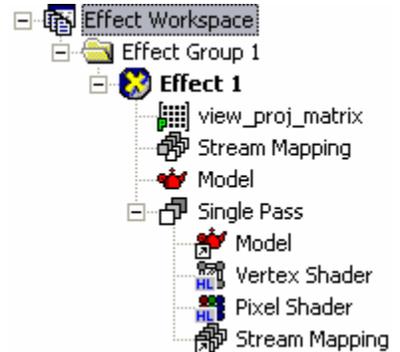
-  Notes

To create an *Effect Group*, the user should right click on the *Effect Workspace* node in the workspace view. The user will see the following context menu:



Selecting “Add Effect Group” adds a new *Effect Group* named “Effect Group #N” at the end of the current workspace. The *Effect Group* name is editable, and either double-clicking on it or right-clicking on it and then selecting “Rename” will allow you to change the name for that *Effect Group*.

When a new effect group is added, it is automatically created with a sample effect with one rendering pass. The pass inside that effect contains sample vertex and pixel shaders, and a sample geometry model.



To delete an effect group the user can either right-click on the effect group the user is trying to delete and select “Delete”, or simply by pressing the *delete* key when the desired effect group is selected.

The user can also cut / copy / paste / delete effect group from any point in the workspace.

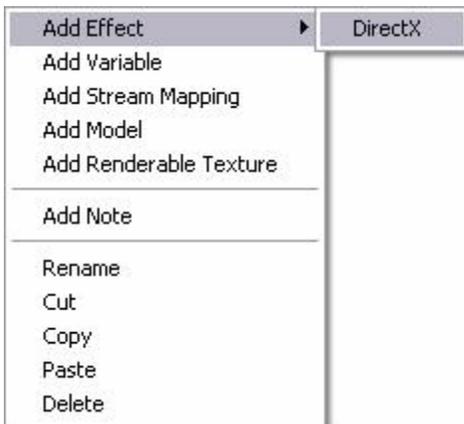
Managing Effects

Each *Effect* is used to draw a single, coherent visual effect in the viewer. You may have a single pass effect, or may want to use several draw calls to generate the look that you want.

The actual *Effect* consists of these elements:

- Variables (shader parameters)
-  Stream Mapping nodes
-  Models
-  Cameras
-  Passes
-  Renderable Textures
-  Notes

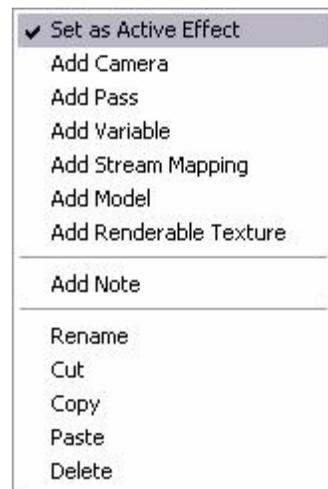
To create a new effect, the user should right click on the effect group that the effect should be added to. The “*Add Effect*” option will contain a sub-menu with current API options for the effect.



The selected effect will be created and added to the bottom the selected effect group. The effect name is editable and can be easily changed in the same manner as every other node name in the application.

Like a newly added effect group, when the new effect is added, it contains a sample rendering pass.

To preview the effect in the preview module the user should right click on the desired effect and select “*Set As Active Effect*” from the effect context menu:

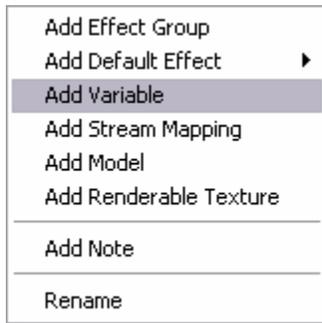


To delete an effect, the user can either right-click on the effect node and select “*Delete*” from the context menu, or selecting the effect node and pressing the *delete* key.

The user can cut / copy / paste effects from one effect group to another.

Managing Variables

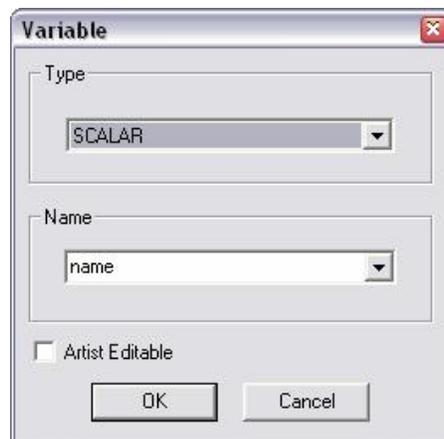
User-defined data is run-time data needed to render an *Effect*. Effects and individual passes will reference the user-defined data by variable name. The scope of variables follows the tree downwards.



The user can add data at any level of the workspace tree.

To add a new variable, the user must right click on the node to which the variable will be added (it can be the main effect workspace, an effect group, an individual effect or a pass) and select “*Add Variable*” from the following context menu:

The user then will see this dialog:



The user can then select from one of the supported data types. Variable types supported by RenderMonkey are:

- 0/1 *BOOLEAN* (true / false)
- [] *SCALAR* (a simple float variable)
- [] *VECTOR* (4D float variable)
- [] *MATRIX* (4×4 float matrix)
- [] *COLOR* (4D float variable, RGBA color representation)
- Texture variables:
 - [] *TEXTURE* (2D texture map)
 - [] *CUBEMAP* (Cube map texture)

-  `VOLUME_TEXTURE` (3D texture map)

By default new boolean, scalar, vector and matrix variables are created as not artist-editable. Color, texture, cubemap and volume texture variables are created as artist-editable by default. To make a variable visible in the *Artist Editor* or the *Art* tab in the workspace view, the user may right-click on that variable and select the “*Artist Variable*” menu option, or select the “*Artist Editable*” option while still in the *Add Variable* dialog. To remove the artist-editable property from a particular variable, right-click on the variable and select the “*Artist Variable*” menu option again. A check mark on that option indicates whether the variable is artist-editable or not. A small yellow flag  on the variable icon will indicate that the variable is artist editable. For example:

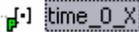
 my_scalar

The user can cut / copy / paste / delete variables from any place in the workspace. The names of the variables are editable and can be modified at any time by either double-clicking on the variable node or selecting “*Rename*” from the right-click menu for that variable.

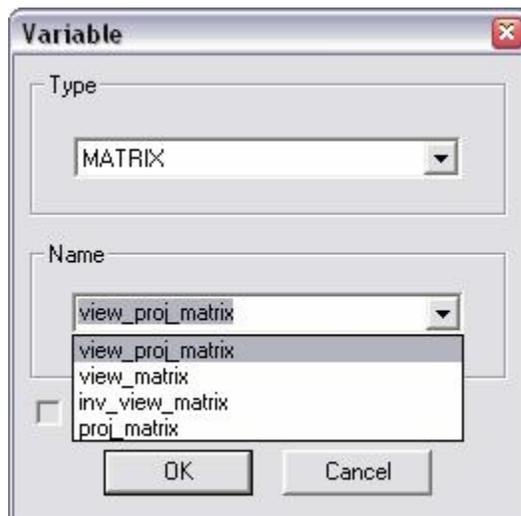


Please note that there are some naming restrictions places on variables that can be referred to within actual shader code. To be consistent with high level shading languages syntax, boolean, vector, matrix, color, and texture object nodes must conform to the language naming syntax. The name must not contain any spaces, must start with a letter and contain only alpha-numeric characters. If the user violates these restrictions for the above mentioned variable types, a warning will appear.

Predefined Variables

RenderMonkey provides a set of predefined variables for added shader development convenience. Such variables will display an appropriate tool tip (*Predefined Variable*) if the mouse hovers over them. Predefined variables are shader constants whose values get filled in at run-time by the viewer module directly at every frame. You cannot modify the values directly through the same user interface that you can use to edit other variables of similar types. A properly flagged predefined variable will be denoted in the workspace tree view with a  symbol over the nodes icon. For example: 

To add a predefined variable to the workspace, the user should select the appropriate type of predefined variable that they would like to use and then choose the name from the combo box control that will appear in the *Name* group of the *Add Variable* dialog. Note that the combo box will only appear if the selected type has some pre-defined variables. If the user then chooses another type for a given predefined variable name, it will not be appropriately initialized at run-time, as RenderMonkey identifies predefined variables by both name and type.



Another way to use any of these predefined variables in a shader, is to rename an existing variable (of the corresponding predefined variable's type) to match that of the desired predefined variable. The name must be a case-sensitive match for RenderMonkey to treat it as predefined.

RenderMonkey provides this set of predefined variables for your convenience:

- Scalars:
 - *time_0_X*: Provides a time value (in seconds) which repeats itself based on the "Cycle time" set in the *RenderMonkey Preferences* dialog. By default this "Cycle time" is set to 120 seconds. This means that the value of this variable cycles from 0 to 120 in 120 seconds and then goes back to 0 again.
 - *cos_time_0_X*: Provides the cosine of *time_0_X*.

- *sin_time_0_X*: Provides the sine of *time_0_X*.
- *tan_time_0_X*: Provides the tangent of *time_0_X*.
- *time_cycle_period*: Provides the “Cycle time” set in the *RenderMonkey Preferences* dialog. For the example above, this value would be 120.
- *time_0_1*: Provides a time value [0..1] which repeats itself based on the *Cycle time*. This means that if the time cycle was set to 120 seconds, then within the span of 120 seconds the value of this variable would grow from 0 to 1.
- *time_0_2PI*: Provides a time value [0..2PI] which repeats itself based on the *Cycle time* similarly to other time variables.
- *cos_time_0_2P*: Provides the cosine of *time_0_2PI*.
- *sin_time_0_2PI*: Provides the sine of *time_0_2PI*.
- *tan_time_0_2PI*: Provides the tangent of *time_0_2PI*.
- *viewport_width*: Provides the width (in pixels) of the preview window.
- *viewport_height*: Provides the height (in pixels) of the preview window.
- *viewport_inv_width*: Provides $1 / \text{viewport_width}$.
- *viewport_inv_height*: Provides $1 / \text{viewport_height}$.
- *random_fraction_1*: Provides a random number in the range [0..1].
- *random_fraction_2*: Provides a random number in the range [0..1].
- *random_fraction_3*: Provides a random number in the range [0..1].
- *random_fraction_4*: Provides a random number in the range [0..1].
- Vectors:
 - *view_direction*: Provides the view direction vector (world space)
 - *view_position*: Provides the view position (world space)
- Matrices:
 - *view_proj_matrix*: Provides the view projection matrix.
 - *view_matrix*: Provides the view matrix.
 - *inv_view_matrix*: Provides the inverse of the view matrix.
 - *proj_matrix*: Provides the projection matrix.

- *world_view_proj_matrix*: Provides the world view projection matrix. Note that since RenderMonkey version 1.0 does not support implementation of a scene graph, we have decided to keep the world matrix as identity, but provide this predefined variable for your development convenience. The user may apply this variable in their shader and when imported into their engine, they may provide appropriate value of the world view projection matrix through the engine's calculations.

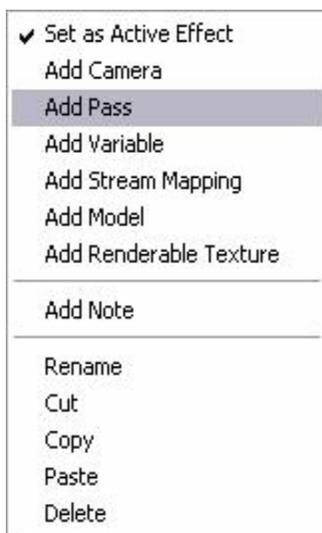
Managing Render Passes

An individual effect may have one or more rendering passes (draw calls).

Each rendering *Pass* may contain the following elements:

- Variables
-  A single Render State Block
-  A single Vertex Shader
-  A single Pixel Shader
-  Texture Objects
-  A single Camera Reference
-  A single Stream Mapping Reference
-  A single Geometry Model Reference
-  A single Render Target
-  Notes

To edit each individual element of the pass the user can double-click on the desired node and that will bring up an editor module appropriate for that node. Another way to start editing a node would be to right-click on a node and select “*Edit*” menu option for the node context menu.

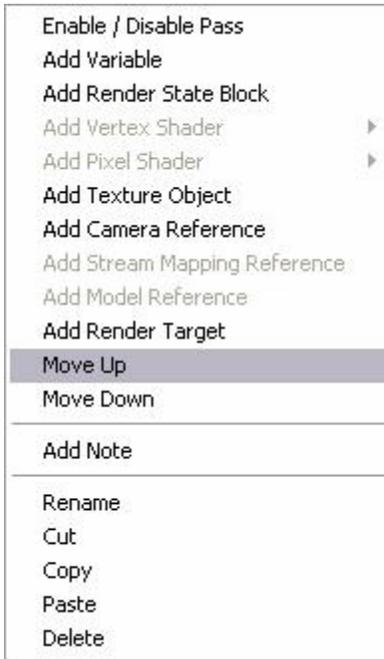


To create a new rendering pass, the user should right-click on the effect to which the pass will be added and click “*Add Pass*” on the effect context menu.

By default each pass will be created with a sample vertex shader and pixel shader and a sample geometry model. The user can modify those at any time. The pass name is editable and can be renamed at any point.

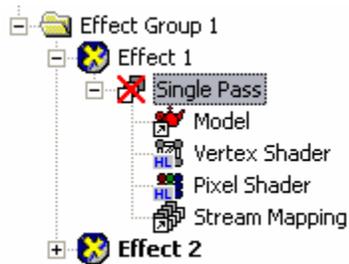
To delete a pass the user can either select the pass and press the *delete* key or right-click on the pass and select “*Delete*” on the pass context menu (see below).

The passes are drawn in the order in which they are arranged within their parent effect. To move a pass up or down the user can right-click on the desired pass and select “*Move Up*” or “*Move Down*” from the pass context menu, or hold down the control key and press the up / down arrows while the pass is selected in the workspace view:



The user may also wish to disable a particular pass to aid him/her in shader debugging. To do that, they can select “*Enable / Disable Pass*” from the pass context menu (accessible by right-clicking on the desired pass). A disabled pass will have this icon on the left of its name  to denote that it is disabled. To enable a pass just click on the same menu option again.

Example of workspace view with a disabled pass:

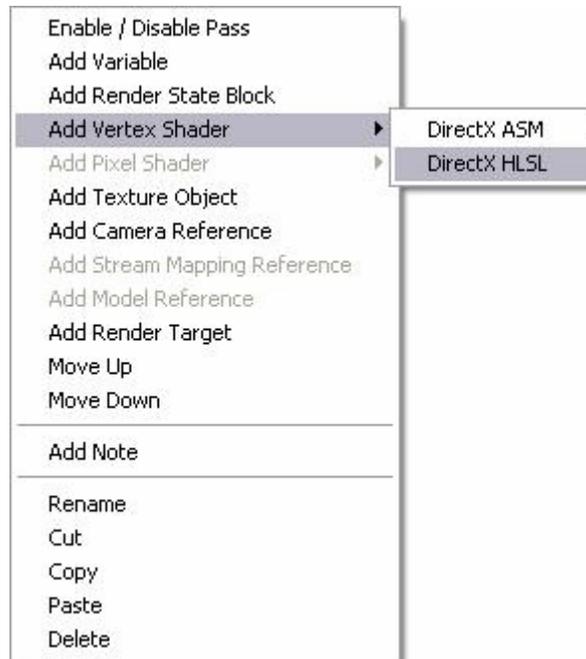


The user can cut / copy / paste passes from one effect to another.

Managing Pixel and Vertex Shaders

This version of RenderMonkey supports *Vertex Shaders* vs_1_0 - vs_2_0 and *Pixel Shaders* versions ps_1_1 – ps_2_0.

To create new *pixel* or *vertex* shaders, the user must select an effect to which they want to add the shader, then right-click on the effect and select either “Add Pixel Shader” or “Add Vertex Shader” menu options, as well as the desired vertex or pixel shader type in order to add the desired shader.



DirectX ASM (Assembly) shader nodes will have  icon for the vertex shaders and  icon for the pixel shaders. DirectX High Level Shading Language shader nodes will have  icon for the vertex shaders and  icon for the pixel shaders. Depending on the type of shader selected the editor will be chosen appropriately – the editor for HLSL shaders is different from an ASM Shader editor. Editing, and compilation of *ASM* and *HLSL* shaders is discussed later in the section on the Shader Editor module.

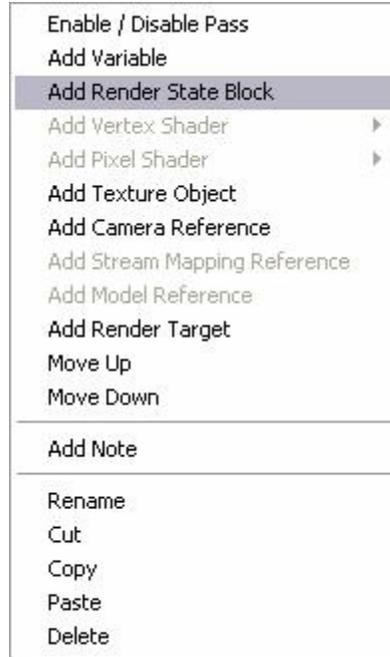
The names of the created shaders are editable at any point. The user can cut / copy / paste individual shaders from one pass to another, provided there is never more than one pixel and one vertex shader in the pass.

To delete a pixel or a vertex shader the user should select the shader node and then either press the *delete* key or right-click on the node or select “Delete” from the shader context menu.

Managing Render State Block

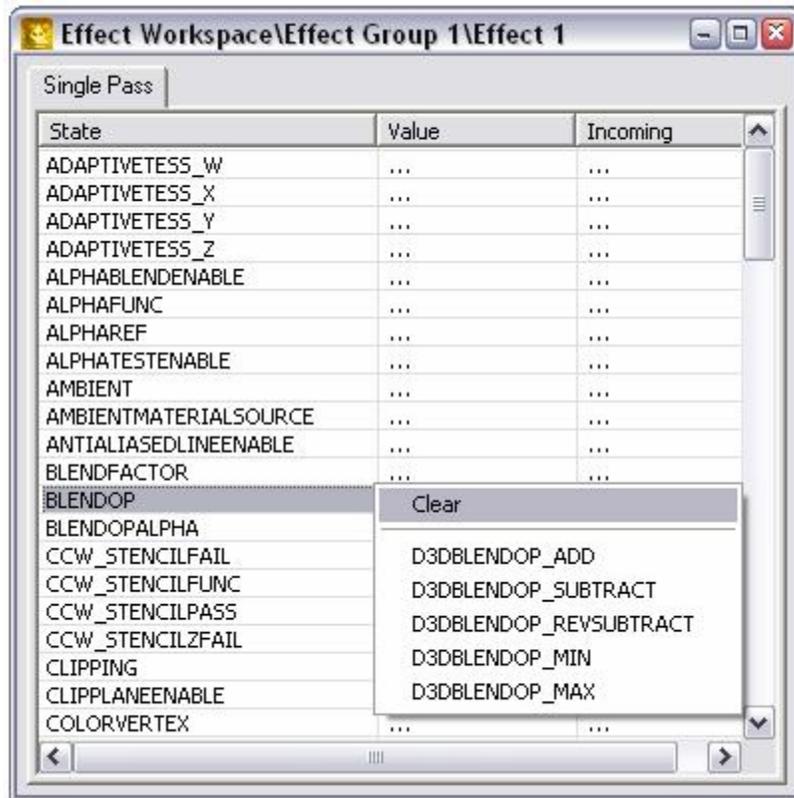
Each pass may have a number of *Render States* that it may want to either inherit from a higher-level *Pass* or set directly. To do that, the user creates a *Render State Block* at least in one place within an *Effect Workspace*.

To create a *Render State Block* node the user must right-click on a pass that they want to add the block to, and select “*Add Render State Block*” from the pass context menu:



If no render state block is defined within a pass, the application will traverse the workspace tree upwards from the current pass to find a render state block node and will inherit the render states from the first render state block found. When you create render state block node in a pass, it inherits the values from the first higher-level render state block found in the active effect. If there are no render state blocks created within the active effect, the application will look through the passes in the default DirectX effect to see if any of them define a render state block. If there are no other render state block found prior to the one created, it will not inherit any values. By default the incoming values for the rendering states within a render state block are set to DirectX default values for those states (please refer to DirectX documentation for actual default state values).

Changing the render state values in the created render state block node will override inherited values. Note that for upward traversal the application only looks in the pass within the current effect and the default effect. The render state block in other effects don't propagate their values. To edit any of the render states in a render state block, you can double-click on the render state block node or right-click on the node and select “*Edit*” from the node context menu.



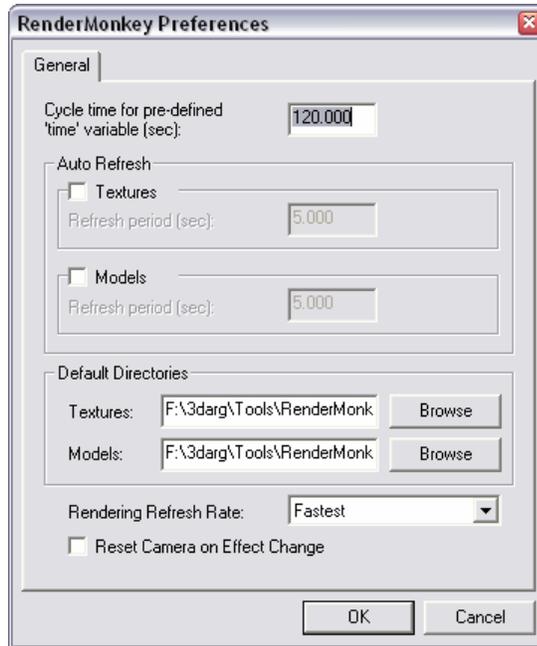
To edit a particular render state, left click in the *Value* column for that render state and either select from a set of predefined values or type a value directly if none were supplied (see example in above for the blending op). If the selected render state block inherits from a higher-level render state block, the incoming values will be shown in the *Incoming* column of the render state editor.

To delete a render state block, the user can simply select that node and either press the *delete* key or right-click on the node or select "*Delete*" from the node context menu.

The render state block nodes can be cut / copied / pasted much like other nodes in the effect workspace.

Application Preferences

The preferences dialog can be invoked through the application menu option "Preferences..." under the "Edit" menu. This dialog allows the modification of application settings, and affects all subsequent RenderMonkey sessions. The following settings may be modified on the "*General*" preferences page:



Cycle time for pre-defined 'time' variable

This option allows the user to modify the cycle period for all predefined time variables (such as `time_0_1` for example. Please refer to the section on Predefined Variables earlier for more details). The default value is set to 120 seconds; however the user may enter any positive integer number to control the cycling of the time.

Auto Refresh

RenderMonkey has the ability to continuously scan the disk for modified textures and models used for rendering the currently active effect in the workspace. If the file has been modified, the application will reload the resources from that file and update the rendering of the current effect. This functionality can be very useful for artists as they can be modifying the resources in another application (for example, editing the textures in Adobe Photoshop), while maintaining the most up-to-date rendering of the effect they are working on.

The application will check if a file has been modified every n seconds, where n is equal to the application preference for that particular resource. To enable automatic update of texture resources, select the *Textures* checkbox in the application preferences dialog. To set the refresh period for texture files disk scan, either keep the default value of 5 seconds refresh period or enter another positive integer number value. Similarly, to enable automatic refresh of models rendering resources, select the *Models* checkbox in the preference dialog and define a custom refresh rate if desired or keep the default value.

Note that only the resources used in rendering of the selected active effect will be scanned for file modifications and updated.

Default Directories

The next two application preferences allow the user to specify default directories used by RenderMonkey as automatic starting point for locating texture and model resource files. Every time when a new model or texture is created, RenderMonkey will search starting in the specified default directory. If the user loads a workspace file, in the event that RenderMonkey fails to find the sources in the saved directory links, it will attempt to locate the resources by looking in the default directories for the resource type. If that will be the case, RenderMonkey will notify the user about a different location for their resource files via a dialog box, if it successfully matched the missing items filenames with files in the default directories.

To specify the default directory for texture resources, the user to type or select a folder in the *Textures* field. Similarly, the user can specify the models default loading directory by providing a default *Models* directory value.

Rendering Refresh Rate

The user can control the frequency with which the *Preview* window will refresh its contents while rendering the active effect. If the user selects *VSync* option, the preview window will wait until vertical retrace is completed to refresh itself – note that selecting that options limits the frame rate for the preview window to the monitor's vertical refresh rate. If the user selects *Fastest*, the preview window will refresh itself immediately thus resulting in significantly higher frame rate – but tearing artifacts may be visible in certain effects.

Reset Camera on Effect Change

When the user switches active effects being rendered in the preview window, the application preference value for *Reset Camera on Effect Change* controls whether the preview window camera settings will be reset upon switching the active effect to the default values (thus bringing it into the origin) if the check box for that preference is selected. Or if the check box is not selected, the trackball orientation will not be modified upon switching to a new active effect.

Modules

Preview Module

The *Preview* module is used to interactively preview effects in the opened workspace. The viewer module automatically recognizes the API of the effect and opens an appropriate renderer for the effect. Version 1.0 of RenderMonkey includes DirectX 9.0 effect rendering, however, using the plug-in architecture, it is easily possible to write other types of preview modules.

To view a particular effect, you must select it to be the currently active effect in the workspace. To do that, right-click on the effect node and select “*Set As Active Effect*” from the context menu for that effect.



RenderMonkey provides an interface for controlling the camera settings for displaying the active effect. That is done by using the *Camera* nodes and the *Camera Editor*. Camera nodes and camera reference nodes are used to specify view orientations for each rendering pass. Camera nodes are placed under an effect, and the camera node marked as “*Active*” will be manipulated by the *Preview* window trackball. A *Camera Reference* is added to a Pass to indicate that the referenced camera settings should be used when rendering that pass. See *Editing Camera Settings* section to get more details about setting up and editing the camera nodes.

By default, any effect rendered in RenderMonkey uses an implicit, default camera defined in the preview window. That camera is not visible to the user but it is active from the start. If an effect does not have a camera node defined by the user, the effect is rendered using the settings for the default camera. If the user added one or more camera nodes to the effect and camera

references to passes, they can select one of the camera nodes to be the active camera for rendering of the effect. One can think of RenderMonkey's camera nodes in the following fashion: the camera node controls the settings of the rendering camera for rendering of each draw call. The user can set up the entire effect by using a single camera (either an implicit default one or a specifically defined camera node). On the other hand, the user may wish to render separate draw calls with different camera settings (perhaps a different camera position is desired, as to render from light's point of view). In that case, the user can define separate camera nodes for the purpose.

Only one camera can be active at one time, however, and that is the camera which gets modified as the user applies the trackball interface in the preview window to modify the camera. As the user rotates or pans the camera around in the preview window, the active camera's values get updated to reflect the new settings. The buttons on the main application toolbar can be used to control the mode for the trackball:



Rotate Camera:

Selecting this mode locks the active camera in the rotation mode for the active camera. The user will be able to modify the orientation of the camera by using the left mouse button in the preview window. This is default starting mode for trackball.



Pan Camera:

Selecting this trackball mode locks the active camera in panning mode. Using the left mouse, the user will be able to pan the camera in the preview window.



Zoom Camera:

Selecting this mode locks the active camera in the zoom mode – by using the left mouse button the user can bring the camera closer or further from the viewer.



Camera Home:

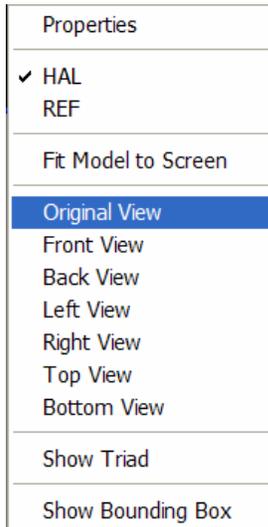
Pressing on this toolbar button will move the active camera to the origin.



Overloaded Camera Mode:

Selecting this mode, the user can use the overload mode for the trackball: left mouse button will rotate the camera, Ctrl-left mouse button will pan the camera, and middle mouse button or the mouse wheel will zoom the camera in and out.

The user can also select one of the predefined view settings from the Preview window's menu.



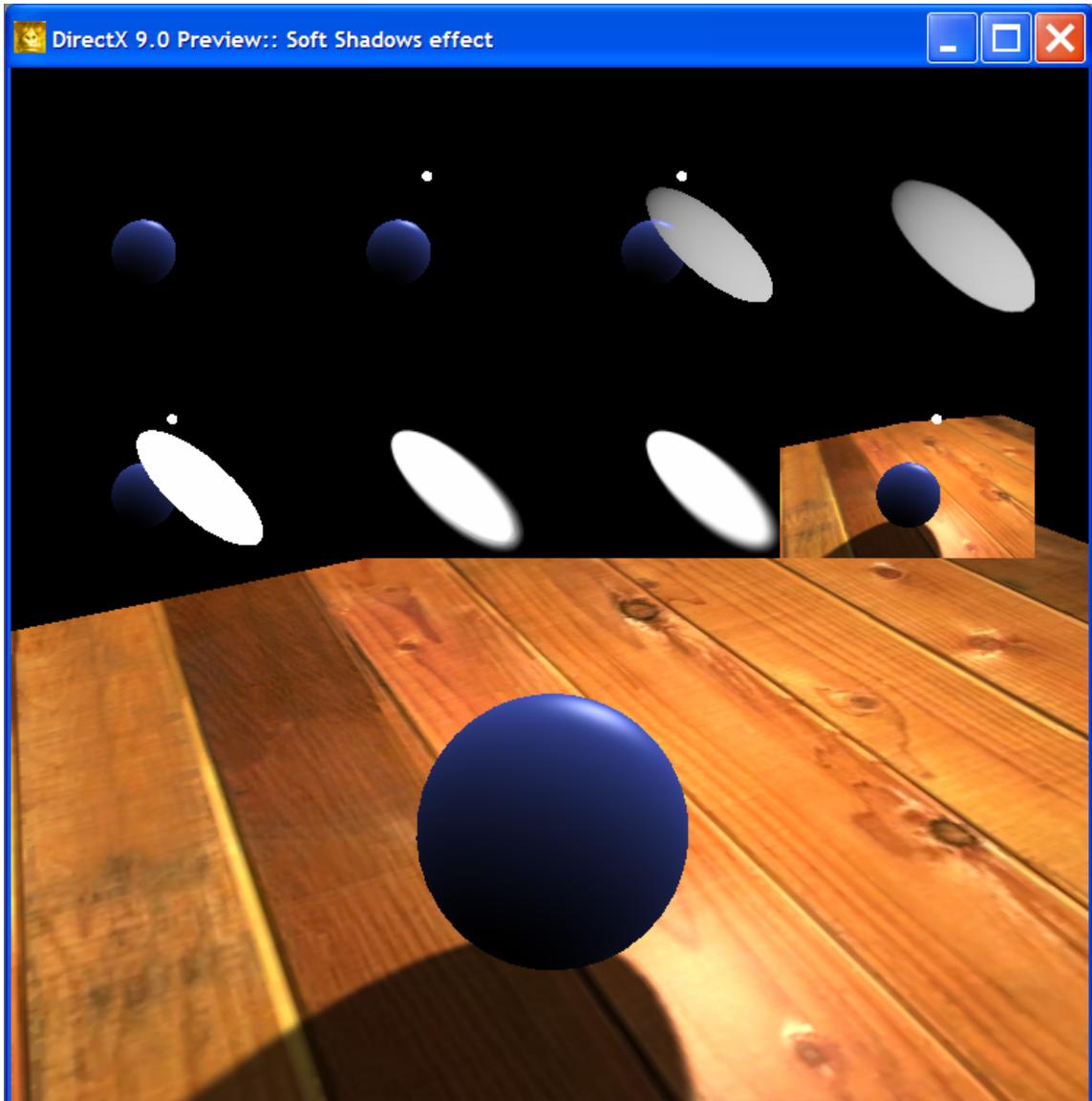
Right-clicking in the preview window will bring up the menu displayed on the left. The user can select from one of the following views defined for the preview window: *Original*, *Front*, *Back*, *Left*, *Right*, *Top* and *Bottom* views.

The user can also use the DirectX preview window's menu to select the type of device used for rendering current effect. By default, RenderMonkey will try to create HAL rendering device if hardware settings permit. However, if user's hardware fails to support hardware acceleration of shaders, or if they simply wish to view their effect using the DirectX reference rasterizer, they can select that setting by selecting *REF* option from the preview window. Once the user selected the reference rasterizer, the contents of the preview window will be updated only on demand, as rendering shader-based effects using software rasterizer can be quite slow. To update the rendering, the user can press the space bar in the preview window. When rotating or panning or zooming in the preview window, the user will see the bounding box of rendered objects and the coordinate axes triad in the preview window when they are modifying the camera.

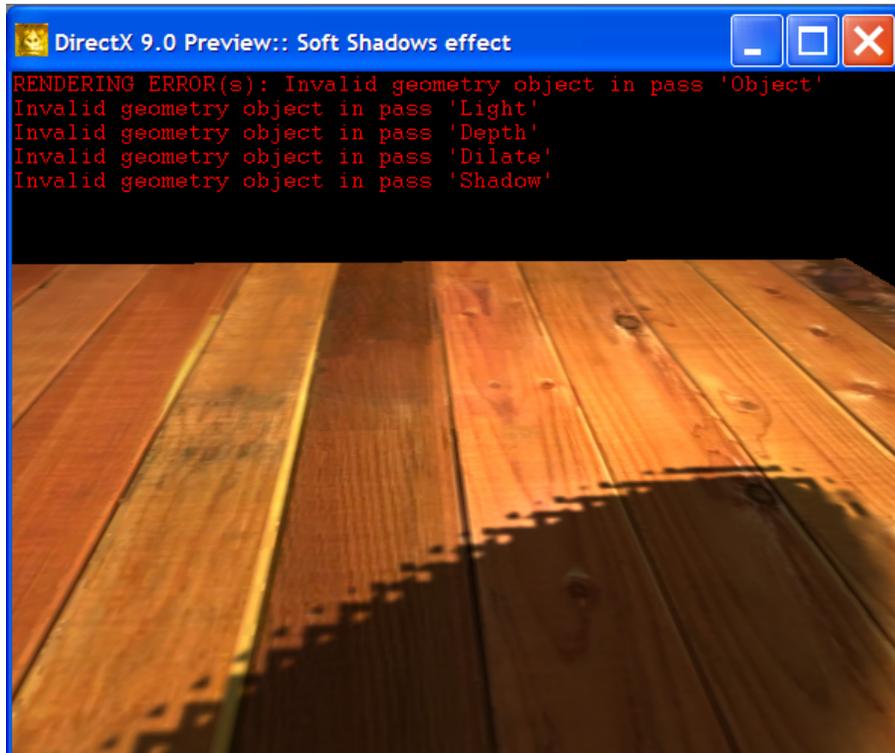
The *Fit Model To Screen* menu option will modify the active camera to fit best to the bounding box of the models in the passes that use this active camera. Note that if any of the passes in the effect do not use the active camera, *Fit Model To Screen* mode will not include these passes for computing the fitting bounding box. Another precaution about this setting and geometry bounding boxes: if any passes use vertex shaders that modify vertex positions, the original bounding box will not correspond to actual output vertex positions and thus will not be correct.

Pressing 'h' in the preview window displays all keyboard shortcuts available for the window. The user can press 's' to toggle display the rendering frame rate of the active effect. The user can press 't' to reload all textures used for rendering of the active effect, and press 'm' to reload all models used for rendering of the active effect. Pressing 'b' will toggle display of the geometry bounding box and pressing 'a' will toggle display of the coordinate axes triad. Pressing 'u' will force an update of all rendering resources used to render currently active effect, such as reloading textures, recreating the buffers, etc.

Pressing 'p' in the preview allows the user to view the output of each pass in an array of mini viewports. Each individual viewport contains the output of each pass and all the passes prior to the pass. If a pass outputs to a render target, the viewport for that pass will contain the contents of the renderable texture generated by the pass. In the example below, you see the output of a soft shadows effect where the shadow is generated in multiple passes blurring the shadow from the light in successive blurs and then composited onto scene.



The preview window provides automatic error reporting about invalid or missing resources that are used for rendering currently active effect for the developer of the effect. If you render an effect which has a constant that isn't linked correctly to a RenderMonkey variable node or if the effect attempts to render using an invalid stream map, the renderer module will display an error in the output window as well as in the preview window itself. The image below shows errors for the shadows effect if all of the model-related resources were deleted:

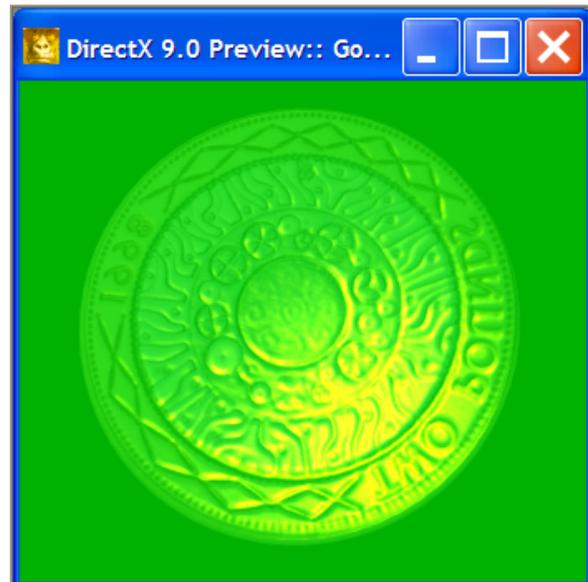


As you can see, the preview window reports that certain passes will not be rendered correctly because they are missing a geometry object reference.

In the case of an effect that uses render targets and passes rendering into renderable textures, if the user creates a pass that uses a renderable texture before it was properly initialized by another pass that would render into that texture, the renderable texture starts out initialized to bright green, to provide instant visual feedback to the user about that setup. For example, the image on the left below is the correct rendering of a glow on a golden coin. The effect is created by rendering the image of a coin into a renderable texture and then applying a glow on that texture and compositing the two images. If we delete the pass that renders the golden coin into a renderable texture, you will see that the glow pass uses a green texture and the overall result in this particular case is clearly incorrect:



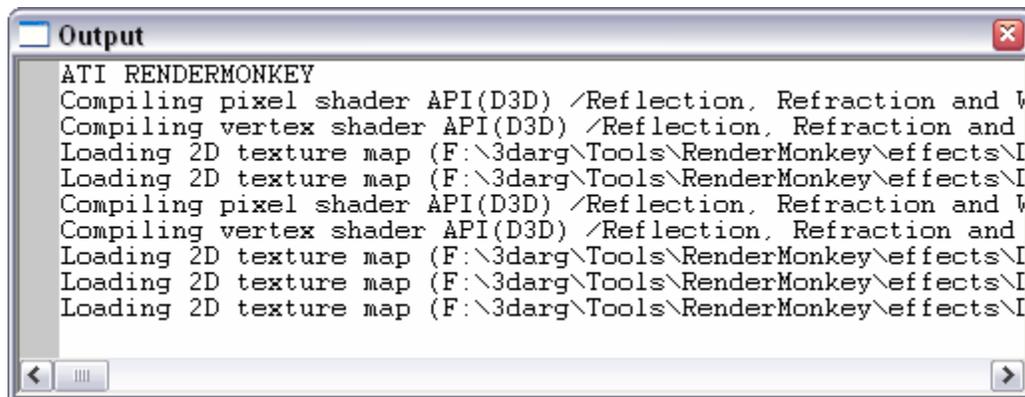
Correct rendering of the effect.



Incorrect renderable texture initialization output.

Output Module

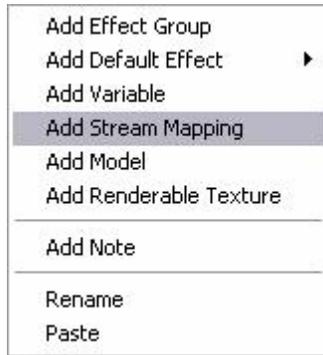
The output module is a docked window typically located on the bottom of the main application interface. That window is used to output the results of shader compilation and other application text messages. The output window is linked with the shader editor for compilation error highlighting.



Stream Mapping Module

The *Stream Mapping* module is used for stream setup for the geometry model within a pass. A *Stream Mapping* node can be created at any point in the workspace (directly under the effect workspace, directly under an effect group, within an effect group or in an individual pass).

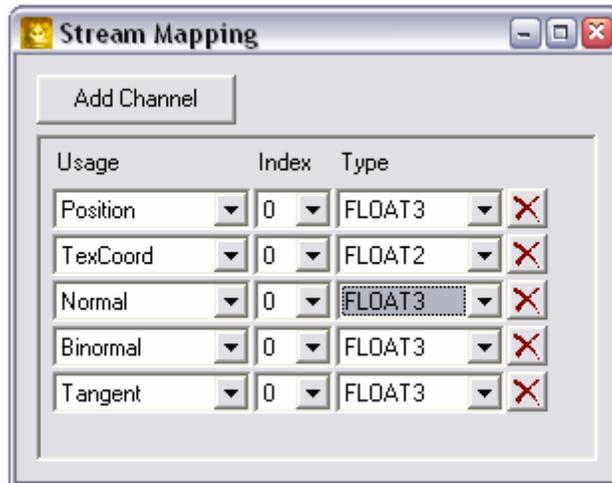
To create a stream mapping node, the user may right-click on a parent node (an effect, a pass, an effect workspace, or an effect group) and select “*Add Stream Mapping*” menu option from the context menu (example here is from the effect workspace context menu):



This creates an empty stream mapping node.

To delete a stream mapping node, the user can either delete the node by selecting it first and then pressing the *delete* key or by right-clicking on the node and selecting “*Delete*” from the node context menu.

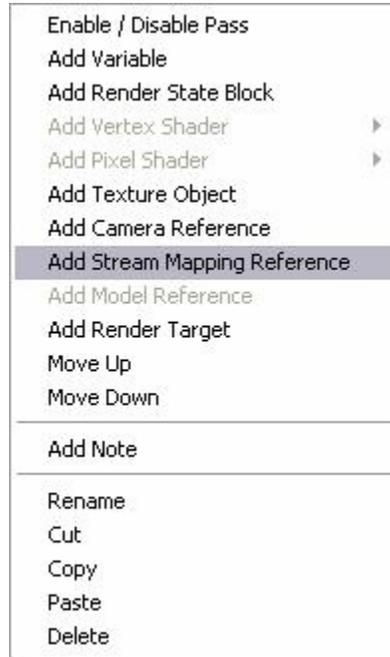
Once a stream mapping node is created, the user can edit it by double-clicking on the node or by right-clicking on the stream mapping node and selecting “*Edit*”, which will bring up the stream mapping editor module:



To add new stream channels to the stream, the user can click on the “*Add Channel*” button in the stream mapping editor. Then the user can select the desired usage for that stream, and select the usage index and type.

To delete a *stream channel*, the user can click on the “X” button on the right of the channel.

The user can also add a *Stream Mapping Reference* to an already existing stream mapping node in the workspace tree. To create a *Stream Mapping Reference* the user should select a pass for which that reference is being created and right-click on it to view the pass context menu. From that menu the user should select “Add Stream Mapping Reference”:



An empty stream mapping reference is then created. That reference is initially not linked to any stream mapping nodes. The red line on the stream mapping reference icon denotes that the reference isn't correctly resolved. For Example:  Stream Map Ref To link a reference to a stream mapping node the user has to right-click on the stream mapping reference node and select a stream mapping node from the “References” list, or select “Rename” from its context menu, then type the name of the stream mapping node that they want to link to.

If the user-typed stream mapping node name is found and resolved correctly, the stream mapping reference node will have this icon:  Stream Mapping Please note the arrow in the icon which denotes that it is a reference rather than the actual stream mapping node.

Shader Editor Module

To edit a particular shader, the user can either double-click on the shader node or select “Edit” from the shader context menu. This will open the *shader source editor*, which is a tabbed window used to edit shaders within an *Effect*. Each tab denotes a vertex (or pixel) shader per *Pass* in the *Effect*. There is one shader source editor for all shaders within an *Effect*.

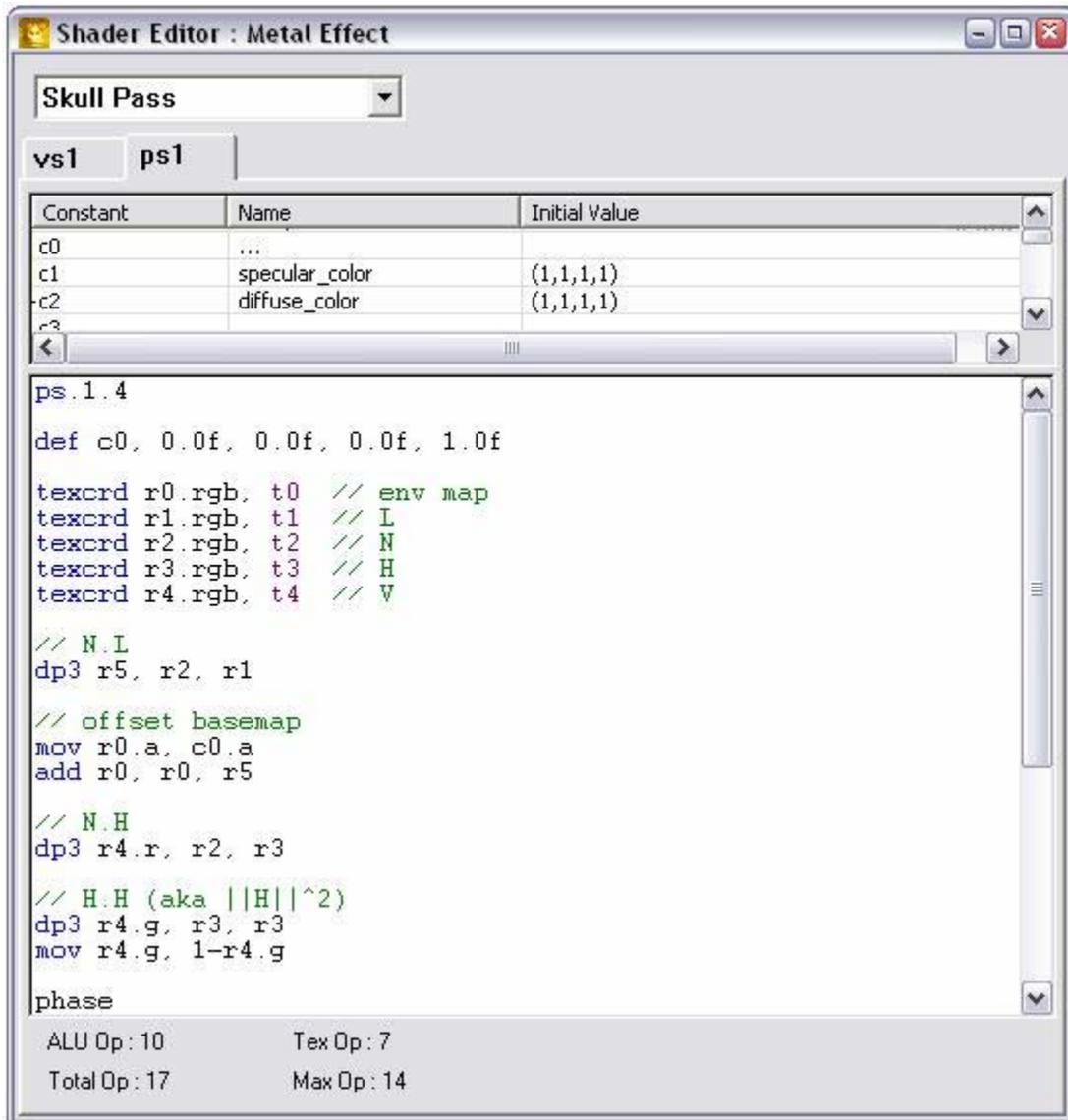
Depending on the language used for the shader, the High Level Shading Language editor or the assembly shader editor will be automatically selected to edit that shader. The features of each particular interface will be described in the sections below.



To compile the shader being edited, the user should click one of the (“*Compile Shader*”) buttons on the main toolbar or use the accelerator (F7 by default). Pressing that button not only compiles the current shader, but it also internally saves the changes of the code of from the shader editor into the shader node. If the user modified the shader text and then tries to close the editor without committing (compiling) the changes, the user will be prompted to commit changes for that particular shader to save the updated shader code.

Editing Assembly

Assembly shader editor window consists of two panes – the top pane is used to bind RenderMonkey variable nodes to shader constant registers and the bottom pane is used to directly edit the shader text.

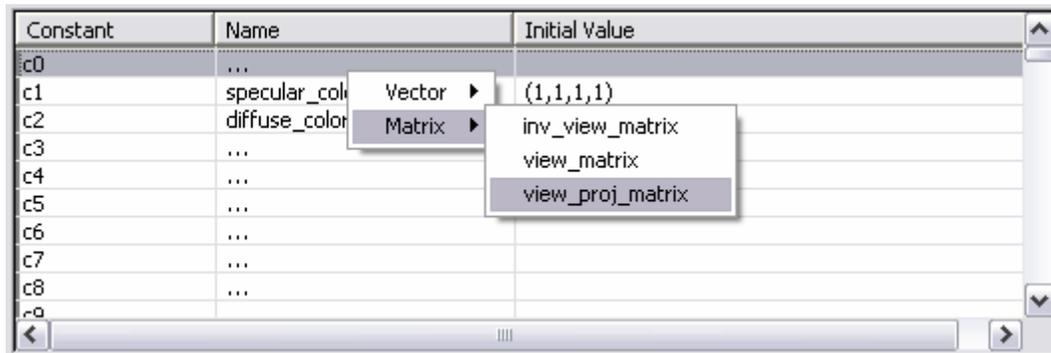


The constant store editor is a list view with three columns. Each row represents values for one particular register. The first column, "Constant", denotes the index of that register. The second column, "Name", shows the node that is linked to that register, or "..." if there isn't a variable linked to that register. The third column shows the initial value of variable node linked to the register.

Binding a RenderMonkey variable node to a constant store register means that the software will actually bind the internal values of the nodes directly to the register values. Within

RenderMonkey IDE, vector and colors nodes are represented by 4 different floats, scalars are mapped to 4 floats having the same value, and matrices are represented by 16 floats.

To bind a RenderMonkey node to a register, the user should right-click on the field in the “Name” column for the constant and select a variable node from the popup menu that will appear. The popup menu will contain all variables that are within scope of the shader being edited. Once a node is selected, the user will see its name appear in the “Name” column for the selected register and the current values of the node will be displayed in the “Initial Value” column.



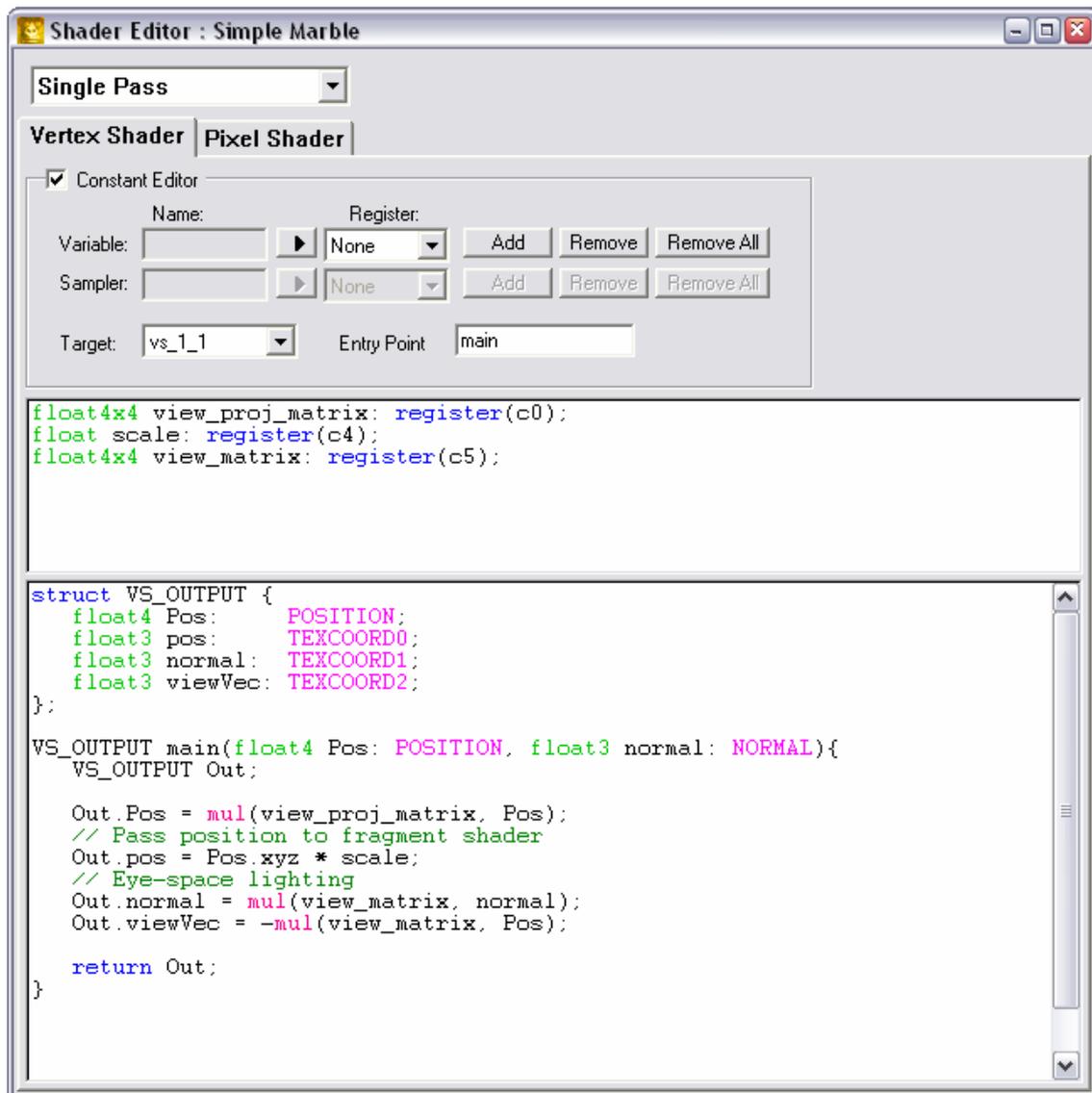
To clear a constant store register, the user should select *Clear* menu option from the popup menu for the register. The name of the variable previously linked to that node will be replaced by “...” and the “Initial Value” column will be cleared.

Please note that if the user binds a matrix to a particular constant, then the three constants below that constant will be overwritten with the rows of that matrix.

The source editor has support for customizable syntax coloring for pixel and vertex shader assembly code. There is also full clipboard support for standard editing operations.

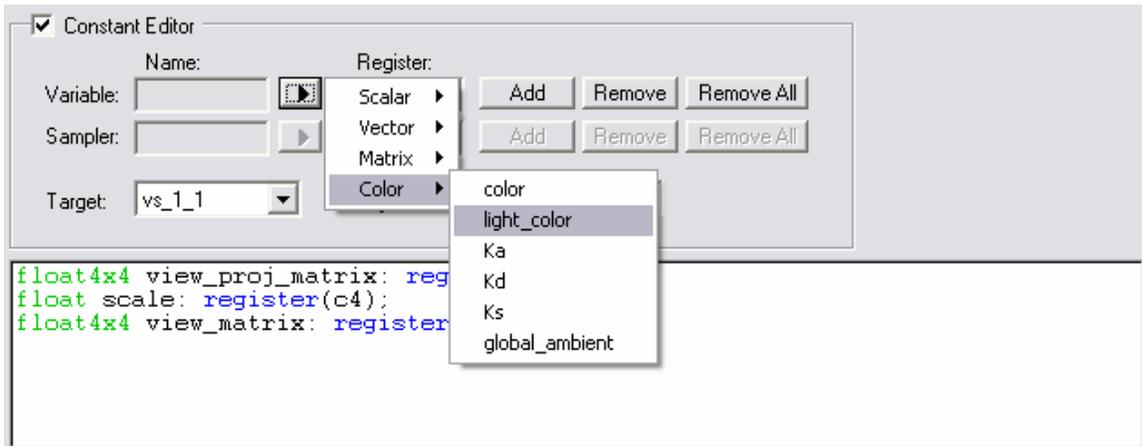
Editing the High Level Shading Language

The High Level Shading Language editor consists of three sections. The UI widgets at the top of the editor are used to manage shader parameters for HLSL shaders. The text editor control in the middle portion of the editor is used to view the declaration block of an HLSL shader which contains parameter declaration. This editor pane is not editable by the user – the declaration block is solely controlled through the UI widgets in the top portion of the editor. This is necessary to ensure the proper mapping from RenderMonkey variable and texture objects to High Level Shading Language parameters. The bottom pane is the editor widget to edit the actual shader text.



To map a RenderMonkey variable node (a vector, a color, a matrix or a scalar node) the user should left-click on the arrow button next to the variable's Name label:

Variable: . This action opens up a popup menu containing a list of all variable nodes within the scope of the shader being edited. The user should then select a variable node from that popup menu:



At that point the label under the “Name” column will change to the name of the node that was selected by the user. Next the user should click on the “Add” button to add that variable node to the declaration block and map it internally as a shader constant. Or if the parameter is already mapped to the shader and the declaration block, the user can click “Remove” to remove it from the shader.

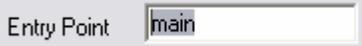
The user should be aware that the mapping process of RenderMonkey nodes to the declaration block depends on the node being named exactly the same as the parameter in the declaration block. IF the node that was mapped to the declaration block parameter was either deleted or renamed, the user won't be able to remove it from shader constant list or the declaration block. In that case, the user can choose to remove all parameter declarations by clicking on “Remove All” button. That action removes both the constant mapping and clears the shader declaration block.

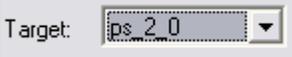
Similar process is used to bind RenderMonkey texture objects to the sampler declarations in HLSL. The user should first create valid texture objects for each texture stage they want to use with texture references. Then the user can follow the process for managing variable mapping to map samplers by using the widgets in the “Sampler” row. Left-clicking on the arrow button next to the sampler label (Sampler: ) will open a list of available texture objects that can be mapped as HLSL sampler objects. The name of the texture reference is used as the name for the sampler. Then the user can either add or remove that sampler object in the same manner as above. Same restrictions apply as far as managing nodes that are mapped to sampler objects.

If the user wishes to bind a parameter to a particular register, they should select the register by clicking on the register combo box and selecting from the list of registers available: Separate register sets exist for variables and sampler mapping. 

The user should note that for High Level Shading Language parameter definition, the RenderMonkey nodes they desire to map must be named within the constraints of the High Level

Shading Language, otherwise improper naming will result in compilation errors. Please refer to the language manual to learn of valid naming conventions.

By default, an HLSL shader entry point is set to main. The user can change it by typing a different name in the entry point edit field: 

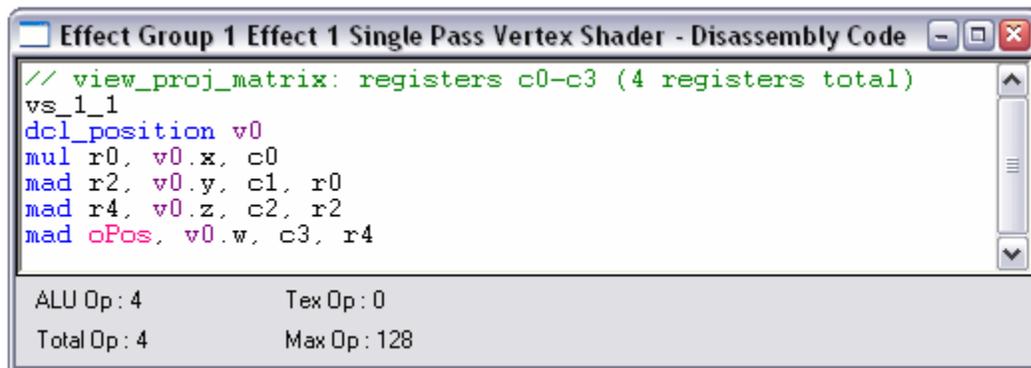
Every HLSL shader must provide a compilation target. To do that, the user should select from a list of available targets from the *Target* combo box: . The target sets are separate for pixel and vertex shader – please refer to High Level Shading Language documentation for explanation of each target value.

The bottom pane of the editor is used to edit the actual text of the shader. The shader text must contain at least one function with the same name as the specified entry point for the shader to compile. The shader text editor has High Level Shading Language customizable syntax coloring.

HLSL Disassembly Window

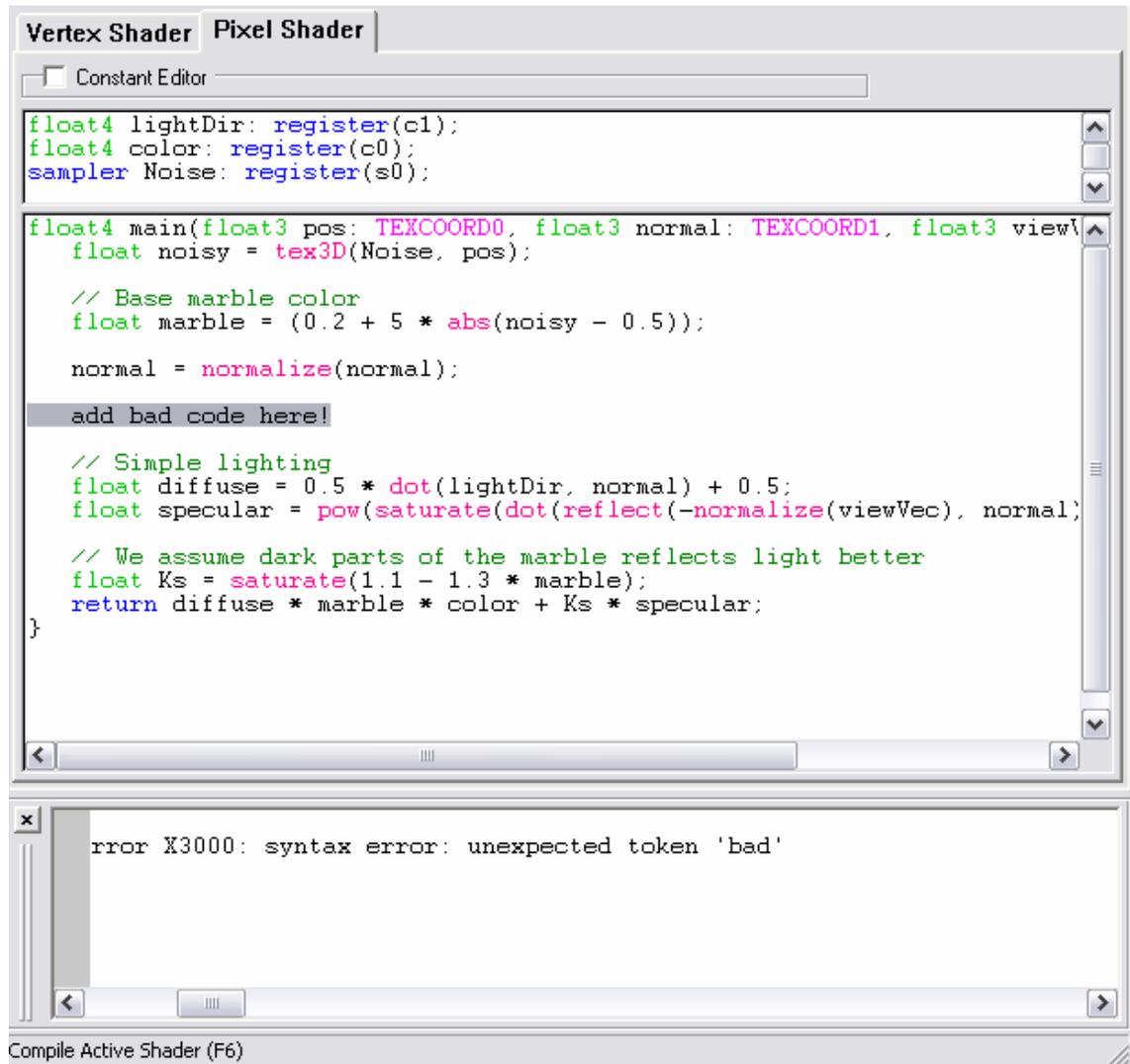


Upon successful compilation of an HLSL Shader, the disassembly code from the compiled shader is available to view. Selecting “Show Disassembly Code...” from the shader editor’s context menu will bring up the Disassembly Window to view. The disassembler window shows code information such as the number of ALU instructions (*ALU Op #*), the number of texture instructions used by the shader (*Tex Op #*), the total number of instructions generated by this shader (*Total Op #*) and the maximum number of instructions allowed for a particular compile target (*Max Op #*). This information is very useful when targeting specific hardware with HLSL shaders as well getting the best performance out of your shaders.



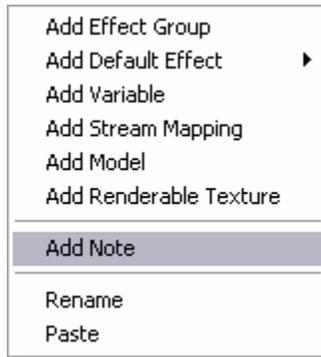
Assembly or Compilation Errors

The source editor supports line highlighting for assembly or compilation errors. If a particular shader has an error, it will be reported in the output module window. The user then can double-click the error in the output window and the line containing the error will be highlighted in the shader source editor for that shader:

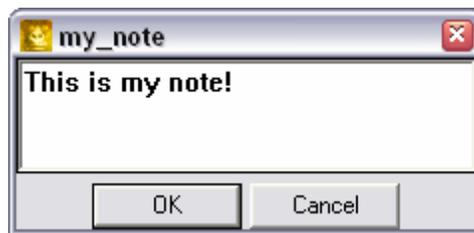


Editing Notes

To add a node note to an existing node, select the “Add Note” note option from the nodes context menu:



To edit a note the user should either double-click on the note node or select “*Edit*” from the right-click menu for that node. The note editor is a simple text editor, and can be used to add documentation to specific workspace nodes.



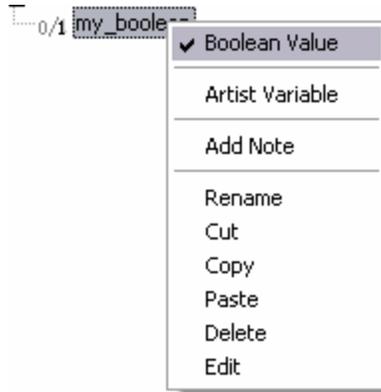
The contents of any note node added to the workspace will be displayed in the tooltip for that node.

Editing Variables

To edit a variable the user should either double-click on the variable node or select “*Edit*” from the right-click menu for that node.

Boolean Variables

Boolean variables do not actually have an associated editor, as the values are simply modified through the “Boolean Value” option in the nodes context menu:



Scalar Variables

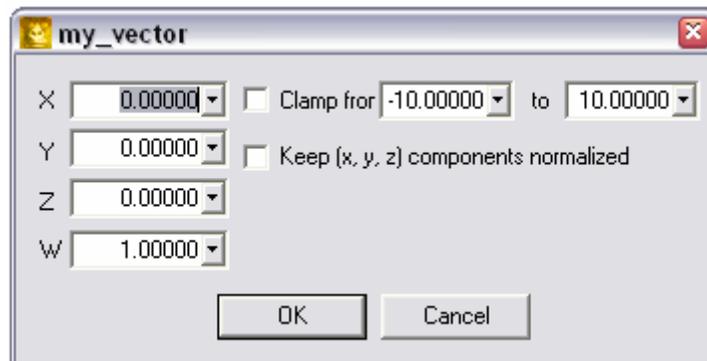
Each scalar can be edited via the scalar editor module:



The scalar can be edited by either directly typing the value in the main edit box, or by interactively using a popup slider which will be in the same range as the clamping bounds (regardless whether the user chooses to clamp the vector or not). The user can preview the changes to the rendered effect by modifying the value of the scalar interactively in the preview window, but at any point, the user can select “Cancel” to undo the changes to the variable. If “OK” is pressed, the new value for the scalar variable is propagated to the database.

Vector Variables

Each vector can be edited via the vector editor module:

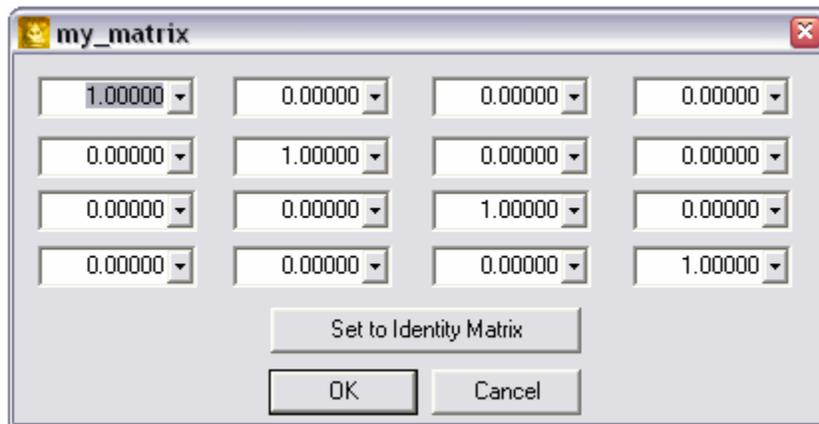


Each vector component can be edited by either directly typing the value in the component edit box or by interactively using a popup slider for each component. The sliders' ranges will be

the same as the clamping bounds for the vector (regardless whether the user chooses to clamp the vector or not). The user may also select to keep the vector normalized by selecting “Keep vector normalized” check box. The user can preview the changes to the rendered effect by modifying the value of the vector interactively in the preview window, but at any point, the user can select “Cancel” to undo the changes to the variable. If “OK” is pressed, the new value for the vector variable is propagated to the database.

Matrix Variables

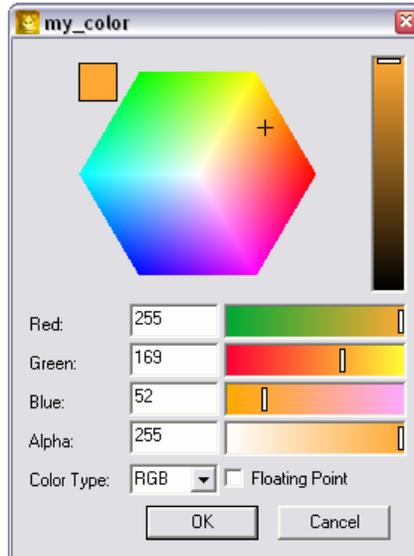
Each matrix variable can be edited via the matrix editor module:



Each matrix component can be edited by either directly typing the value in the component edit box or by interactively using a popup slider for each component. The slider range is preset to be between [-100.0; 100.0], however, typing a value outside of that range expands the range to that value. The user can also set the matrix to an identity matrix by clicking the appropriately named button. Similarly to the other variables, the user can select to keep the changes to the variable values or to dismiss it by selecting either “OK” or “Cancel” variables.

Color Variables

Each color variable can be edited via the color picker module:



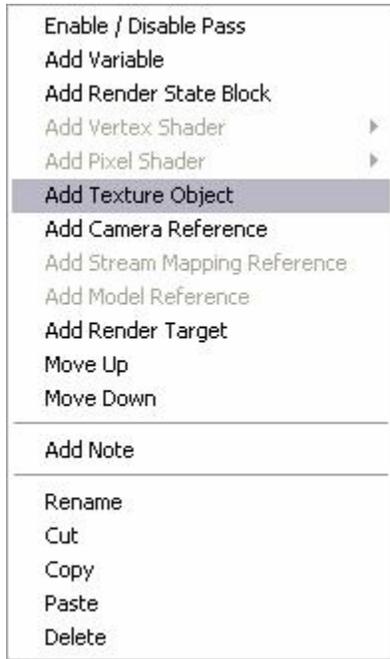
The user can edit color using either RGB or HSV mode by either directly typing the values in the appropriate edit boxes for each component (R, G, B, A or H, S, V, A) or interactively selecting color from the color wheel or color sliders for each component or modify the intensity of the color being edited by using the vertical intensity slider. The value of the color is shown in the color swatch at the top left corner of the color picker. Similarly to the other variables, the user can select to keep the changes to the color variable or to dismiss them by selecting either “OK” or “Cancel” variables.

Model Variables

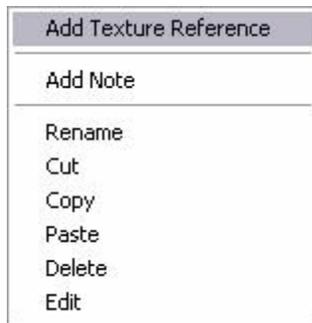
To edit a geometry model variable, the user should double-click on the variable node and when a file dialog opens up, the user can select the file for the geometric model that they want to load. Currently RenderMonkey supports .3DS and .X model file format.

Texture, Cubemap and Volume Texture Variables

To use texture-based variables, the user has to first create a texture variable using “*Add Variable*” dialog in the desired location of the workspace. That texture variable is used to select a file from which to load the texture. To actually use a texture within a *Pass*, the user should select the desired *Pass* and select “*Add Texture Object*” menu option:

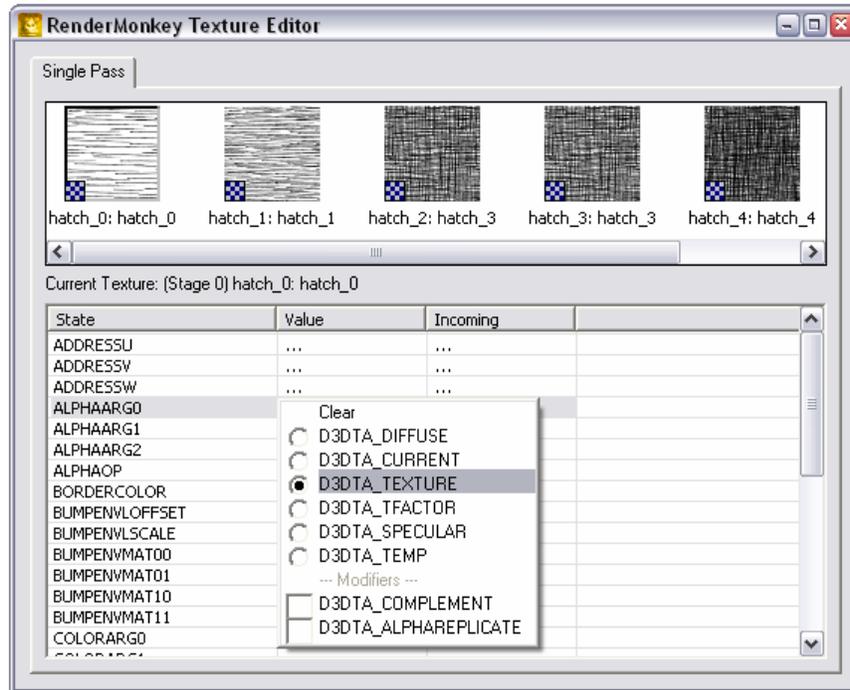


This creates an empty texture object. Next the user should add a texture reference to a texture variable in the workspace. To do that, the user should select “*Add Texture Reference*” from the right-click menu for the texture object:



This creates an empty texture reference  `baseMap`. To actually create the reference to a texture variable the user should type the name of the variable they want to reference. If a valid texture variable will be successfully found, then the red line across the texture reference will be removed. Red line across the texture reference icon denotes that the texture variable wasn't successfully referenced.

The user should also specify texture state values (filtering, clamping, etc) for a particular texture reference node. To do that, the user can launch texture editor by double-clicking on a valid texture reference node.



The texture editor has tabs for each individual *Pass* within an *Effect*. The top of the texture editor contains a list of texture references within the selected *Pass*. By clicking on a texture icon the user can select to view and set texture states for that texture. To set a particular state, the user should left click on the *Value* field next to the state they are trying to edit and either select a value from the predefined set of values for that state or type of a value if none was provided.

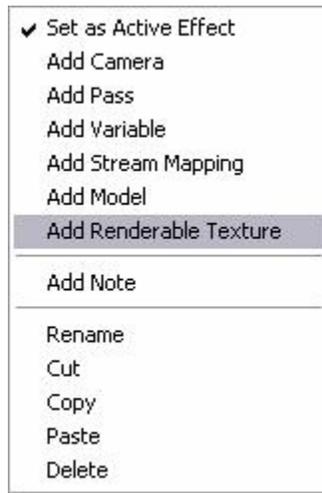
Also note that only the texture objects with valid texture references will have  icon or a thumbnail image. If the texture object's texture reference isn't correctly linked, then that object will be displayed with  icon.

Renderable Texture Support

The user can render output of any given *Pass* to a texture and then sample the contents of that texture in any other *Pass*. To add that functionality to your workspace, here is the sequence of steps you must follow:

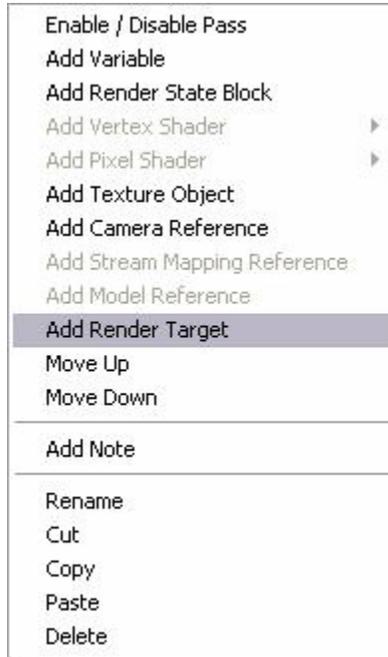
1. Create a renderable texture at any point in the workspace. Only one *Pass* can render output to that texture at a time.

To add a renderable texture, click on any node that you would like to add it to and select "Add Renderable Texture" from the context menu that appears at that point:

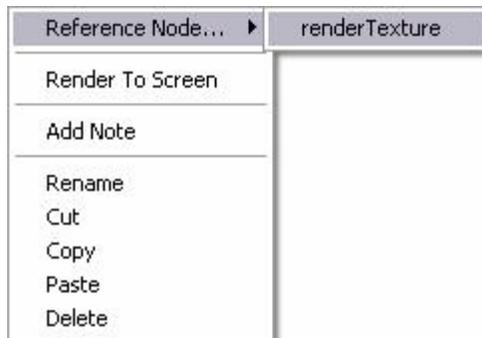


2. You will see a new node appear in the tree with this icon: . This node is the renderable texture node that you will link later to a render target and to a texture object to sample from this renderable texture.

3. Next you need to add a render target to the *Pass* that is going to output to the renderable texture. Select the *Pass* node and right-click on it to select the context menu for that *Pass* – choose “Add Render Target” to add a new render target (the node will have this icon  next to it once it's created):

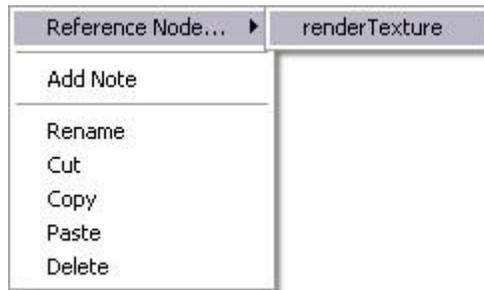


4. Next you must link the render target node to the renderable texture that you've created. To do that, you can either rename the render target node to *exactly the same* name as the renderable texture node that you want to link it to, or you can right-click on the render target node and select a node to reference from a context menu that will appear:



5. At this point the output of the *Pass* that owns the render target node is drawn to the renderable texture.
6. Next, let's link the renderable texture to a *Pass* that is going to sample from it. To do that, you must first create a texture object and a texture reference within that *Pass* (see the section on managing textures above). Once a texture reference exists, you must link it to the renderable texture by either renaming the texture reference node to the *exactly the*

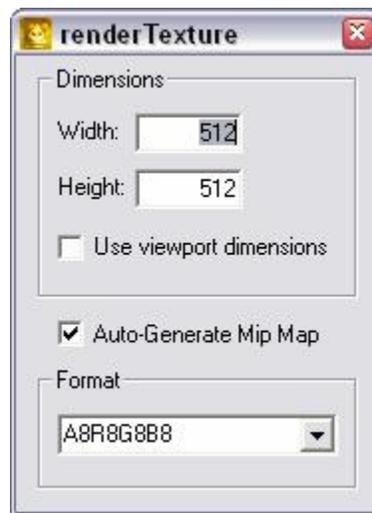
same name as the renderable texture or by right-clicking on the texture reference node and selecting the renderable texture you want to link it to from the *Reference Node* menu:



7. At this point you can use the texture object as you would normally use it in your shader (assembly or HLSL).

Editing Renderable Texture

To edit a renderable texture node, double-click on the node itself  to open the renderable texture editor module:



In that editor you can change the dimensions of the renderable texture: to change either width or height of the texture, type the integer dimension that you wish into the appropriate edit box and press “Enter” to propagate the changes and create new renderable texture. You may also bind the texture to use the dimensions of current viewport by checking “*Use viewport dimensions*” button. Selecting “Auto-Generate Mip Map” will enable the mip-map chain to be auto generated during the rendering process.

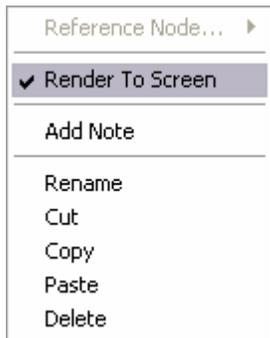
To change the format of the renderable texture, the user can select from a list of predefined formats by selecting them from the *Format* combo box control.

Editing Render Target

To edit a render target node, the user should double-click on the node itself (🎯) to open the render target editor window, or select “Edit” from the node right-click context menu:



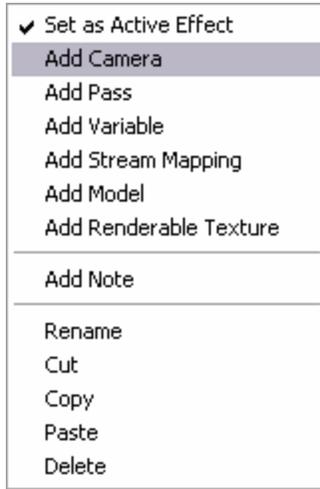
From that editor, the user can select whether to clear the renderable texture by checking or unchecking “*Enable color clear*” button. If the user chose to clear the texture, they can select the color they wish to clear it to by clicking on the *Clear Color* button and selecting the color from the dialog that will appear. The user can also select whether to enable depth clearing by checking or unchecking “*Enable depth clear*” button. If depth clearing is enabled, the user can select the value used.



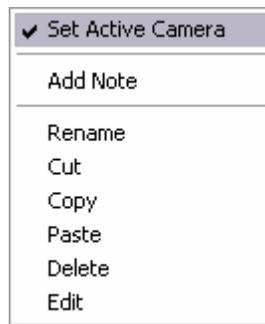
To have a pass render to the screen instead of into a renderable texture, the user can select the “Render To Screen” option from the render target context menu. When rendering to the screen instead of into the renderable texture, the render target icon will change from the standard icon (🎯), to a special icon (🎯) to indicate the change.

Editing Camera Node Settings

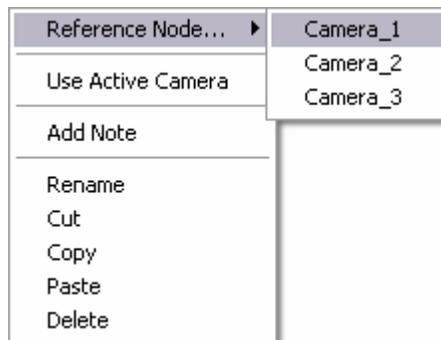
🎥 Camera nodes and 🎥 Camera Reference nodes are used to specify view orientations for each rendering pass. Camera nodes are placed under an effect, and the camera node marked as “Active” will be manipulated by the *Preview* window trackball. A *Camera Reference* is added to a Pass to indicate that the referenced camera settings should be used when rendering that pass. To add a *Camera* node to an effect, right click the effect node and select “Add Camera” from the context menu:



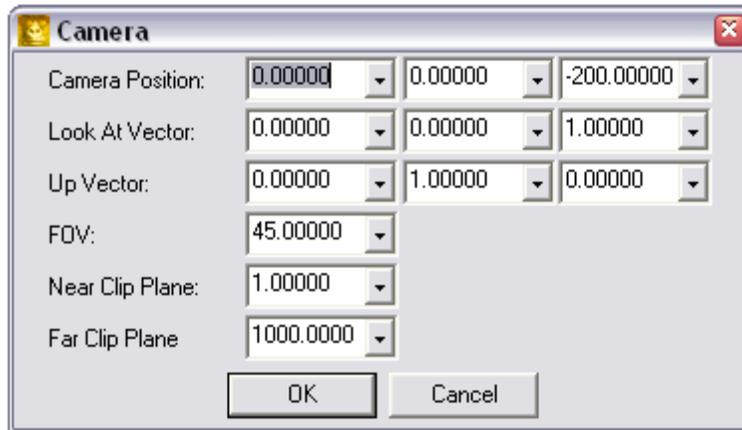
The “Active” camera node is marked with a small check . For Example: Camera. To make a specific Camera node “Active”, right click the camera node, and select “Set Active” from the context menu:



To enable a camera node to affect a specific rendering pass, a *Camera Reference* must be added to that pass. To add a camera reference to a pass, select “Add Camera Reference” from the pass context menu. To ensure the camera reference is referencing the desired Camera, select the appropriate camera node from the camera reference’s context menu:



To edit a camera node, double click on the camera node itself (), or select “Edit” from the right-click context menu:



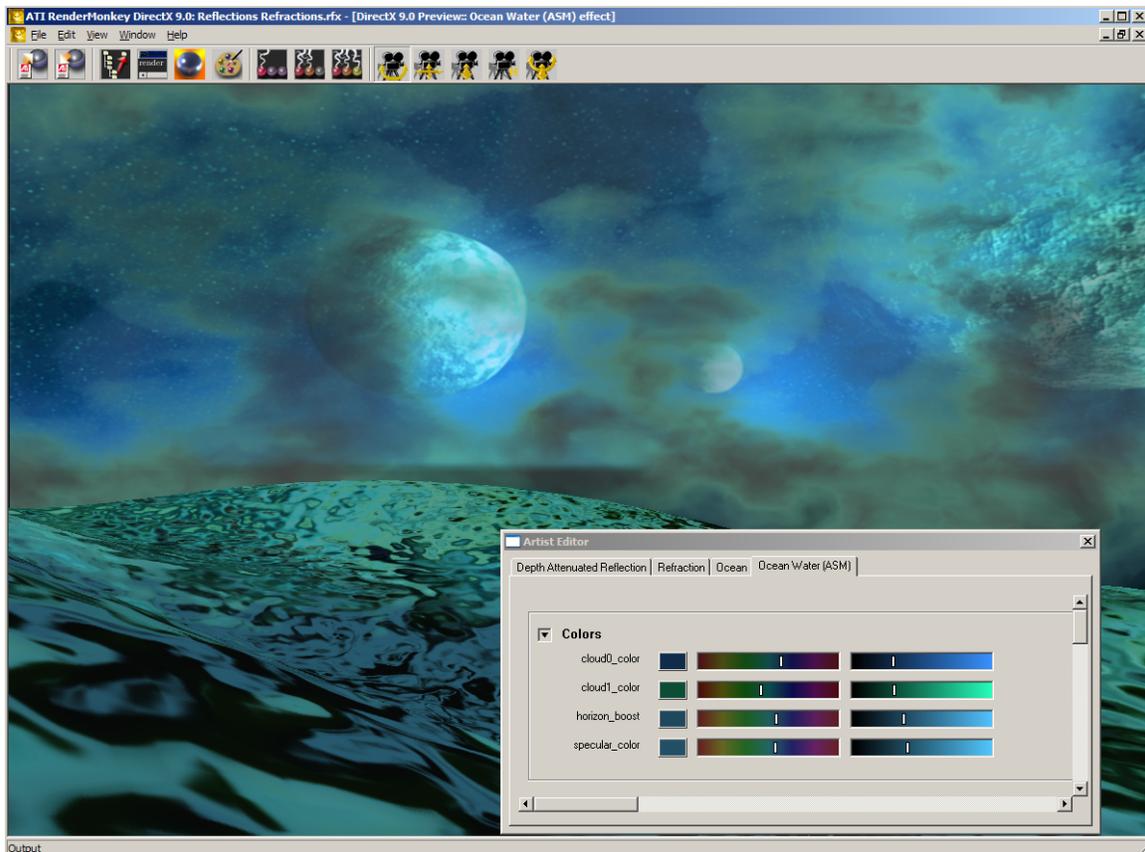
The Camera Editor allows the user to manipulate the *Camera Position*, the *Look At* vector, the *Up* vector, the *Field Of View (FOV)*, and the *Near / Far Clip Planes* values. If the camera editor for the active camera is opened and the user is manipulating the trackball in the preview window, the values will be updated after the trackball has changes its values.

Artist Editor

One of the problems that shader developers face in production is how to present the shaders to the 3D artists to allow the artists to experiment with the shader parameters to achieve desired *Effects*. RenderMonkey's solution for this problem is the *Artist Editor* module combined with the *Art* tab in the workspace view.

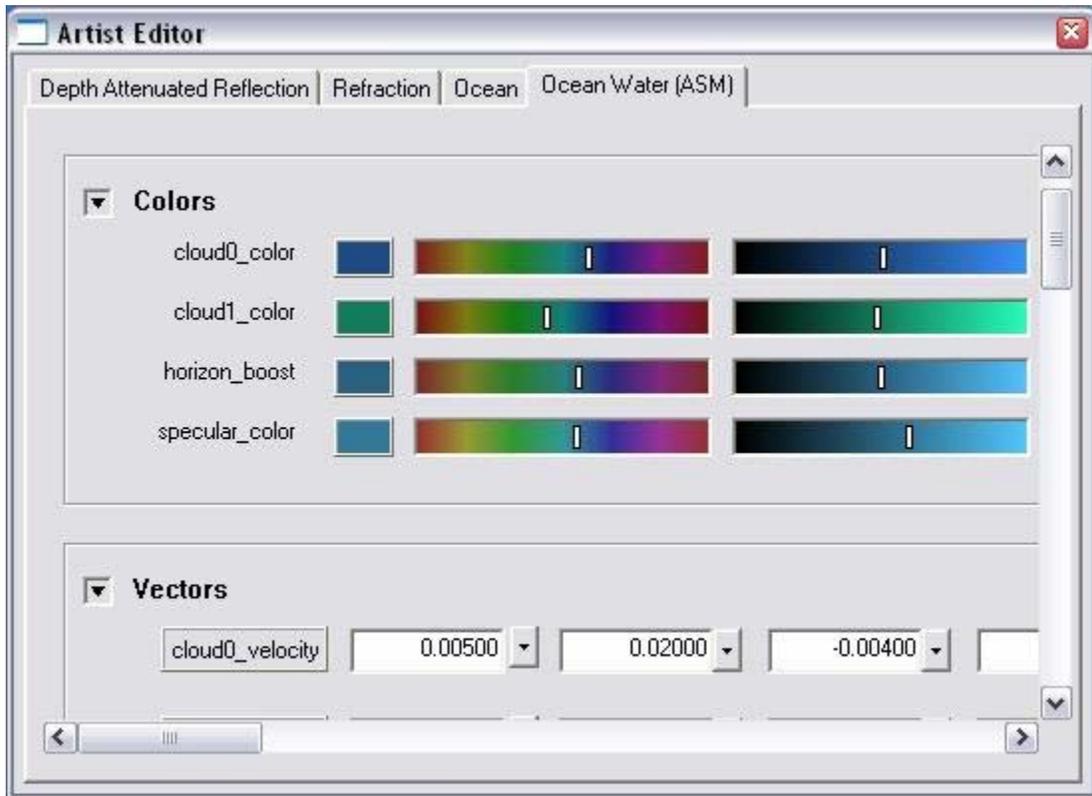
A shader developer can select certain variables in the shader *Effect Workspace* to be flagged as “*artist-editable*” variables. To do that, the user selects “*Artist Variable*” from the right-click menu for the desired variable node and a small yellow flag icon will be overlaid over the icon for that variable. Then the shader developer can give the *Effect Workspace* with their shaders to the artists. The artist can select the *Art* tab from the workspace view to only view artist variables present in the workspace. For added convenience the artist can edit artist variables of supported types in the artist editor module. Currently the supported types for the artist editor are vectors, scalars and colors, however, any variable can be flagged as artist variable and accessed from the *Art* workspace tab.

To open the artist editor, the user can either click the  button on the application toolbar, or select “*Artist Editor*” from *View* menu in the main application menu.



The artist editor is a tabbed window with tabs for each *Effect Workspace*, *Effect Group*, *Effect* or *Pass* that contains artist editable variables. If the node contains no artist editable variables of supported types, it won't appear as the tab in the artist editor.

Artist editable variables are arranged by their types in groups: color, vector and scalar groups. Each group can be expanded or collapsed by clicking on  button within the group.



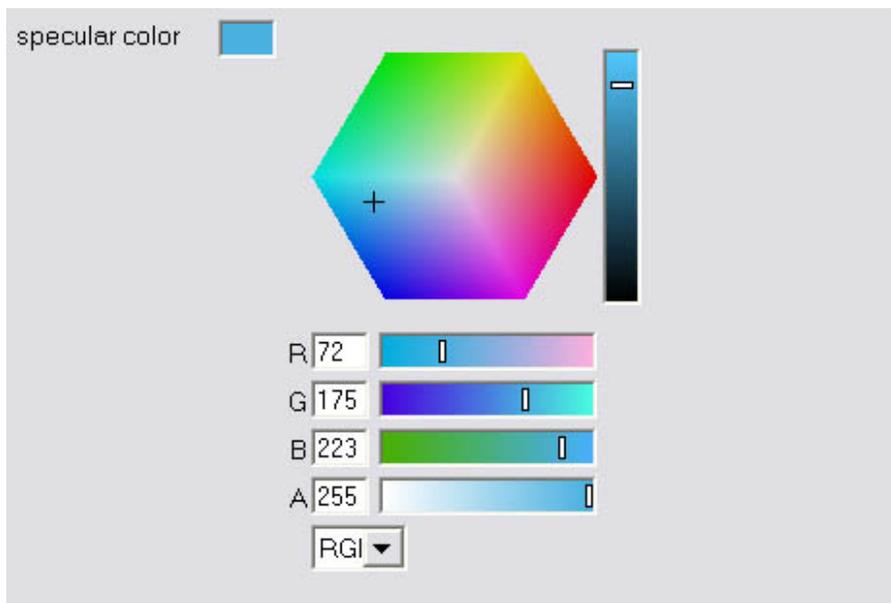
Editing variables in the artist editor module

Colors

Each color variable has three related controls – a color swatch button for opening the full color picker module, a hue slider and an intensity slider:



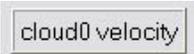
If you click on the  button, you will get an expanded set of controls for editing color with more precision, as shown in Figure 42.

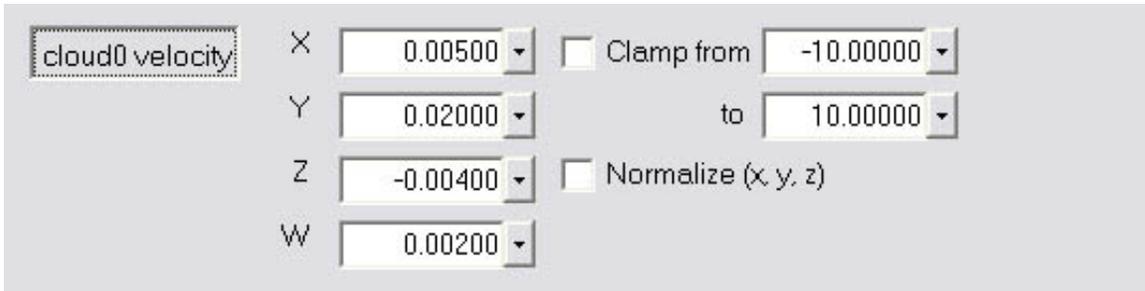


Vectors

Each vector variable has five related controls – a label button that opens up the full vector editor, and four components edit boxes with popup slider buttons for editing each vector component interactively:



If the user clicks on the  button for a particular vector, they will see an expanded set of controls for editing vectors with more precision and control:



Scalars

Each scalar variable has two related controls – a label button which opens up the full scalar editor and an edit box with a popup slider button for editing the slider value directly.



If the user clicks on the `shininessFactor` button, they will see an expanded set of controls for editing scalar variables in the artist editor interface:



RenderMonkey Support and Feedback

As with all the tools and samples provided by ATI, we welcome feedback from the developers who spend every day “in the trenches” solving real problems.

ATI is committed to providing you with the tools you need to make your job easier. In order to do this, we need you to tell us what works and what doesn't. What additions or enhancements would you like to see? What additional problem area exists that we're not currently helping with?

Please help us to help you by providing as much feedback as possible to devrel@ati.com.