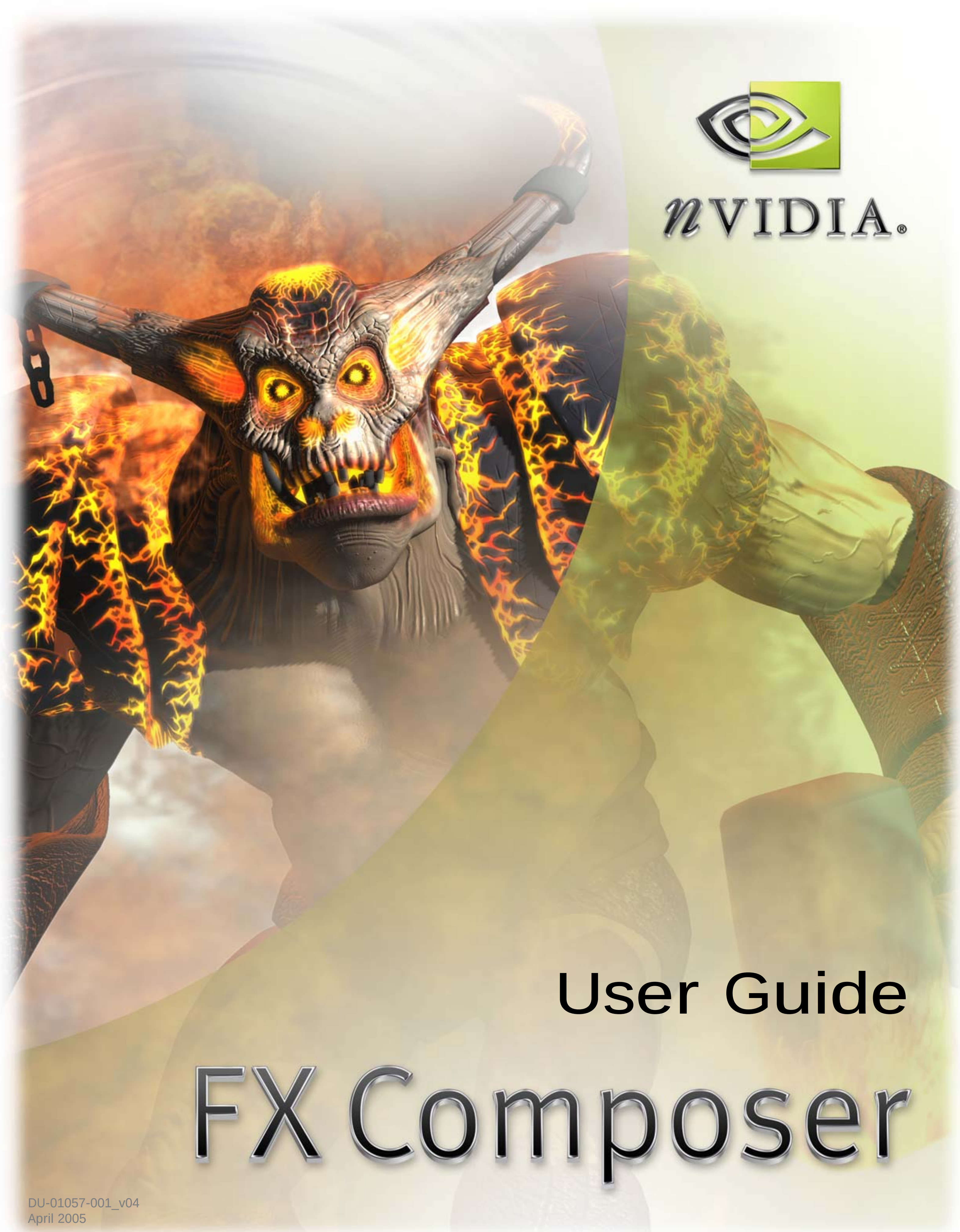


User Guide

FX Composer

DU-01057-001_v04
April 2005

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. About FX Composer

FX Composer 1.7 empowers developers to create high performance shaders in an integrated development environment with real-time preview and optimization features available only from NVIDIA. FX Composer was designed with the goal of making shader development and optimization easier for programmers while providing an intuitive user interface for artists customizing shaders in a particular scene

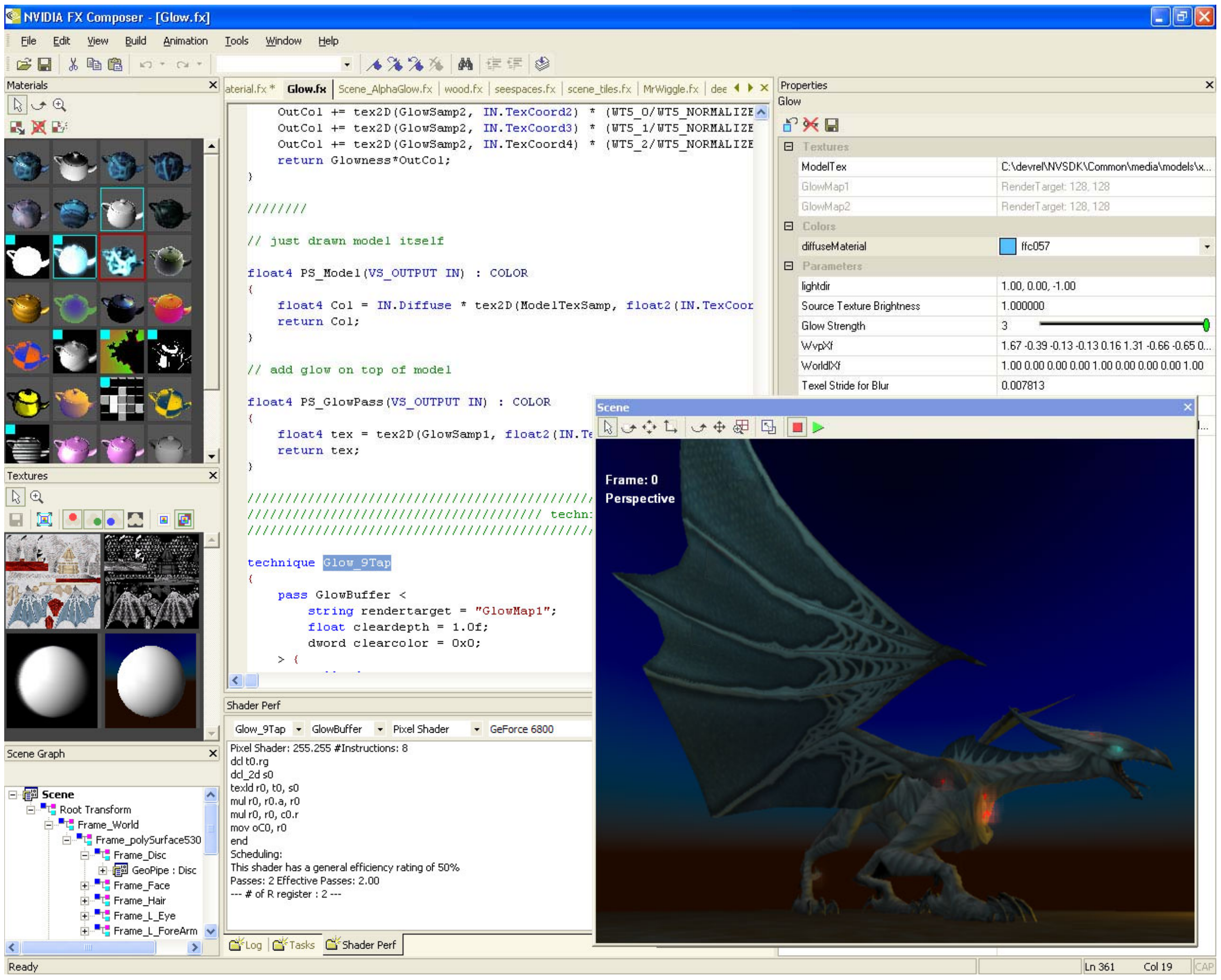
FX Composer supports all the standard features you would expect in an integrated development environment for high performance shaders:

- Sophisticated text editing with Intellisense and syntax highlighting
- Work directly with HLSL .FX files, creating multiple techniques & passes. Use the .FX files you create with FX Composer directly in your application
- Convenient, artist-friendly graphical editing of shader properties and some attributes
- Supports Microsoft DirectX 9.0 standard HLSL semantics & annotations
- Provides a plug-in architecture supporting import of custom scene data so you can view your shaders on your own models with lighting, animation, etc.

FX Composer also provides developers with visual debugging and advanced performance tuning features previously unavailable:

- Practical SDK enabling custom importers, exporters, and scene modifiers
- Powerful scripting support for automation and pipeline integration
- Visible preview of intermediate (generated) textures
- Capture of pre-calculated functions to texture look-up table
- Interactive compiler shows where the problems are – jump directly to problems in your HLSL source code
- Simulated performance results for the entire family of NVIDIA GPUs
- Empirical performance metrics such as GPU cycle count and pixel through-put

FX Composer consists of several panels that can be docked in the main window, as shown in Figure 1-1, or arranged outside of the main window to a more convenient place.


NVIDIA FX Composer - [Glow.fx]
File Edit View Build Animation Tools Window Help
Materials
Textures
Scene Graph
Scene
Root Transform
Frame_World
Frame_polySurface530
Frame_Disc
GeoPipe : Disc
Frame_Face
Frame_Hair
Frame_L_Eye
Frame_L_ForeArm
aterial.fx * | Glow.fx | Scene_AlphaGlow.fx | wood.fx | seespaces.fx | scene_tiles.fx | MrWiggle.fx | dee
Properties
Glow
Scene
Frame: 0
Perspective
Shader Perf
Glow_9Tap | GlowBuffer | Pixel Shader | GeForce 6800
Log | Tasks | Shader Perf
Ready
Ln 361 | Col 19 | CAP

```
    OutCol += tex2D(GlowSamp2, IN.TexCoord2) * (WT5_0/WT5_NORMALIZE
    OutCol += tex2D(GlowSamp2, IN.TexCoord3) * (WT5_1/WT5_NORMALIZE
    OutCol += tex2D(GlowSamp2, IN.TexCoord4) * (WT5_2/WT5_NORMALIZE
    return Glowness*OutCol;
}

////////

// just drawn model itself

float4 PS_Model(VS_OUTPUT IN) : COLOR
{
    float4 Col = IN.Diffuse * tex2D(ModelTexSamp, float2(IN.TexCoor
    return Col;
}

// add glow on top of model

float4 PS_GlowPass(VS_OUTPUT IN) : COLOR
{
    float4 tex = tex2D(GlowSamp1, float2(IN.Te
    return tex;
}

/////////////////////////////////////////////
////////////////////////////////////// techn
/////////////////////////////////////////////

technique Glow_9Tap
{
    pass GlowBuffer <
        string rendertarget = "GlowMap1";
        float cleardepth = 1.0f;
        dword clearcolor = 0x0;
    > {
```

Textures	
ModelTex	C:\devrel\NVSDK\Common\media\models\x...
GlowMap1	RenderTarget: 128, 128
GlowMap2	RenderTarget: 128, 128
Colors	
diffuseMaterial	🟦 ffc057
Parameters	
lightdir	1.00, 0.00, -1.00
Source Texture Brightness	1.000000
Glow Strength	3
WvpXf	1.67 -0.39 -0.13 -0.13 0.16 1.31 -0.66 -0.65 0...
WorldIXf	1.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 1.00
Texel Stride for Blur	0.007813

```
Pixel Shader: 255.255 #Instructions: 8
dcl t0.rg
dcl_2d s0
texld r0, t0, s0
mul r0, r0.a, r0
mul r0, r0, c0.r
mov oC0, r0
end
Scheduling:
This shader has a general efficiency rating of 50%
Passes: 2 Effective Passes: 2.00
--- # of R register : 2 ---
```

Figure 1-1. FX Composer Main Window

System Requirements

- NVIDIA GeForce 3 or later
  An NVIDIA GeForce FX or more recent GPU is recommended
- Microsoft DirectX 9.0 SDK April 2005
  Requires d3dx9_25.dll specifically
- Windows 2000 or XP

1.1. References and Recommended Reading

- Microsoft DirectX web site
  http://www.microsoft.com/windows/directx/default.aspx
- NVIDIA Developer Relations web site
  http://developer.nvidia.com
- *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*
  http://developer.nvidia.com/object/gpu_gems_home.html

Chapter 2. Using FX Composer

FX Composer allows the user to configure their development environment in several ways. The editing area is always displayed in the main window and cannot be closed or undocked from the main window. Each optional element of the UI is displayed in a separate panel that can be moved, hidden, resized, docked to the main window, or free floating any where on the desktop. FX Composer supports systems with multiple displays, allowing free floating panels to be used on a second monitor and can be configured to display any combination of the following panels:

- **Log** panel
- **Error** panel
- **Properties** panel
- **Materials** panel
- **Textures** panel
- **Shader Perf** panel
- **Scene** panel
- **Scene Graph** panel

2.1. Panels

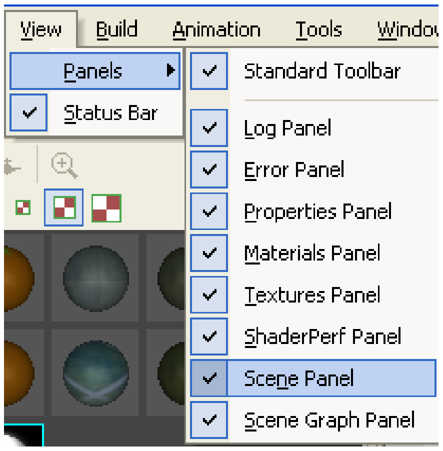
To display the various panels, use the **View ➔ Panels** and select the panels to be displayed in your work area.

Each panel is adjustable and can be dragged away from its docked position to float freely above the main window. Double-clicking the title bar of a dockable panel will also dock or undock it.

Several panels have buttons at the top associated with their major functions.

Table 1 lists these buttons and provides a description of each. These buttons are only displayed in panels where they can be used.

The main window of FX Composer is shown in Figure 1-1.



Where applicable, a toolbar is located at the top of each window panel. The toolbar contains only those icons that are relevant to the panel operation. Table 1 lists the icons and their meaning.

Table 2-1. Panel Tool Icons

Tool Icon	Description
↖	**Select Object**: Selects an object in the panel.
↻	**Rotate Scene:** Rotates the scene in the direction you drag.
✥	**Pan Scene:** Pans the scene in the Scene panel.
⊕	**Dolly Scene:** Zooms the camera in and out on the scene.
⤡	**Zoom Extents:** Resets the camera so that everything fits in the scene.
↘	**Apply Material:** Applies the selected material to the currently mesh selection.
✗	**Delete Material:** Removes this material and close the corresponding .fx file.
✳	**New Material:** Creates new materials and corresponding .fx file.
🔍	**Zoom:** Zooms in on a material or texture. Use Shift+ Click to zoom out.
▣	**Actual Size:** Displays texture at its actual size.
💾	**Save:** Saves the current selection
↻	**Rotate Scene:** Rotates all objects in the window in the direction you drag. (CTRL + L-mouse)
✥	**Pan Scene:** Moves the camera up/down or left/right. (SHIFT + L-mouse)
⤡	**Zoom Scene:** Moves the camera closer or further away. (CTRL + SHIFT + L-mouse)
■ ▶	**Stop/Start Animation:** Controls playback of scene animation. (CTRL+G=Start, CTRL+H=Stop)
🔴 🟢 🔵 ●	**Red, Green, Blue, Alpha** Toggles display of color channel for all textures in the panel.
▣	**From Scene:** Shows textures as used in **Scene** panel.
▣	**From Material:** Shows textures as used in **Materials** panel.

Table 2-2 lists the icons and their function and Figure 2-1 shows the toolbar.

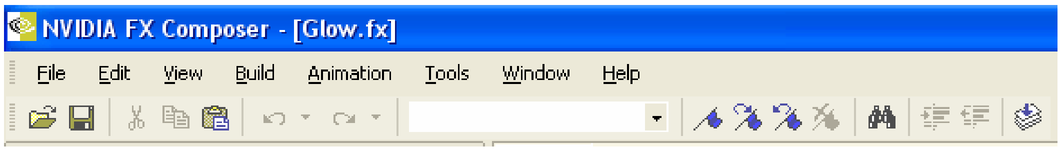


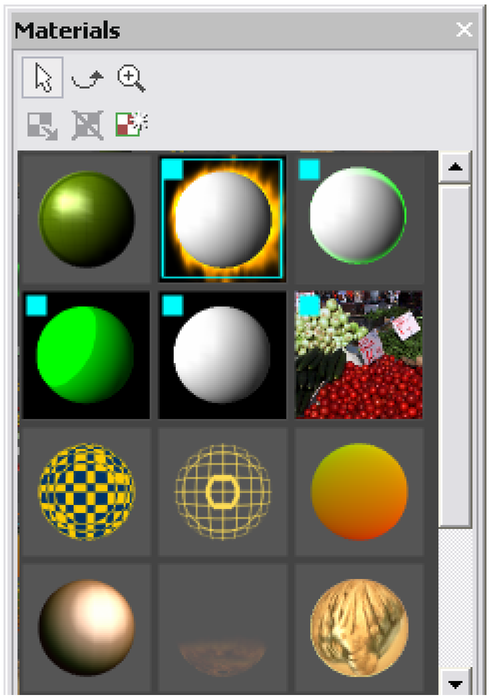
Figure 2-1. Application Toolbar

Table 2-2. Description of Application Toolbar Icons

Tool Icon	Description
	Open File: Opens the browser to find a file. (CTRL+O)
	Save: Saves a file. (CTRL+S)
	Cut: Deleted or cut code from the .fx file. (Shift+Delete)
	Copy: Copies highlighted text. (CTRL+C)
	Paste: Pastes the last copied text. (CTRL+V)
	Undo: Undoes the last action. (ALT+Backspace)
	Redo Redoes the last action. (CTRL+Y)
	Bookmarks: Flags lines of code and skips quickly between them. Toggle (apply/delete) Bookmark. (CTRL+F2) Go to Next Bookmark. (F2) Go to Previous Bookmark. (Shift+F2) Clear ALL Bookmarks
	Find: Finds specific words in a file. (CTRL+F)
	Indent/Un-indent: Indents and un-indents lines of code.
	Compile .fx: Compiles the code displayed in the Text Editor panel. Note that the compiler stops at each instance of an error in the code and does not continue until the error is corrected. Click the **Compile** key again to continue compiling. (CTRL+F7)

2.1.1. Materials Panel

The **Materials** panel is a collection of 3D viewports displaying a real-time preview of each material applied to a simple shape with default scene material. This allows you to visualize a material *and* see it applied in 3D.



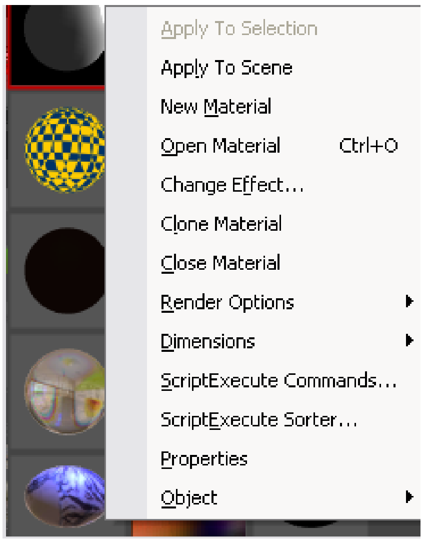
The blue square in the upper left corner of the material indicates that it is a scene material which is applied to the *entire* scene.

The materials without the blue square are materials that are applied to individual objects or sub-objects in the scene.

To open materials and scene materials, use the **File ➔ Open ➔ Material...** and select the material.

Use the **Apply** button to apply the selected material to the selected object in a scene or to the entire scene. You can also Right-click on a material to access a list of actions and select **Apply To Selection** or **Apply To Scene**.

Use the **Rotate** button to spin the materials around and see the different effects. The rotating action affects all materials in the panel but not the scene materials.

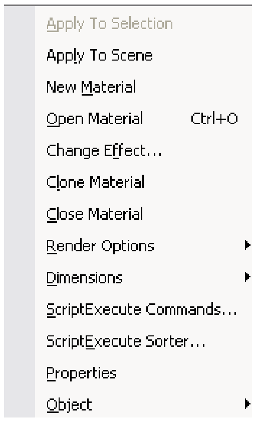


You can also Right-click within the panel to access **Materials** panel display options in a context menu. The menu allows you to apply a material to the current selection in the scene panel, create new materials, open or close existing materials, select the geometric primitive to which the materials should be applied, and set the display dimensions and rendermode.

Note: The **Materials** panel renders all the previews it displays in real-time, and therefore incurs a performance cost. To reduce the performance impact, close the **Materials** panel.

2.1.1.1. Material Panel Display Options

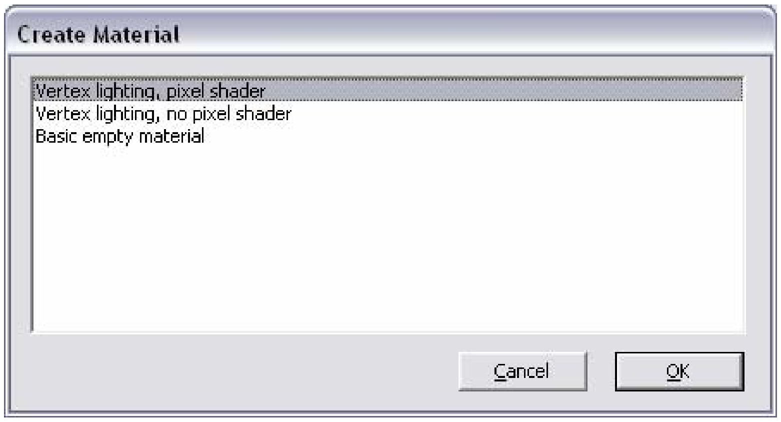
The display options are shown when you Right-Click anywhere in the **Materials** Panel.



Apply To Selection: Applies selected material to a selected object in the scene.

Apply To Scene: Applies the selected scene material to the scene.

New Material: Allows you to create a new material. The new material is displayed in the **Materials** Panel.



Vertex lighting, pixel shader
Vertex lighting, no pixel shader
Basic empty material

Open Material: Load a new .FX file from disk.

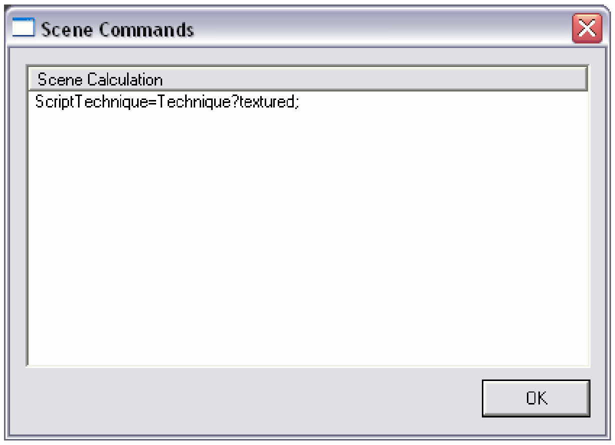
Change Effect...: Chang the .FX used to generate a material.

Close Material: Closes the selected material. Note that if that material will be removed from the scene even if it is in use.

Render Options: Changes the render options on *all* the displayed materials. Options are **Wireframe** and **No Cull**.

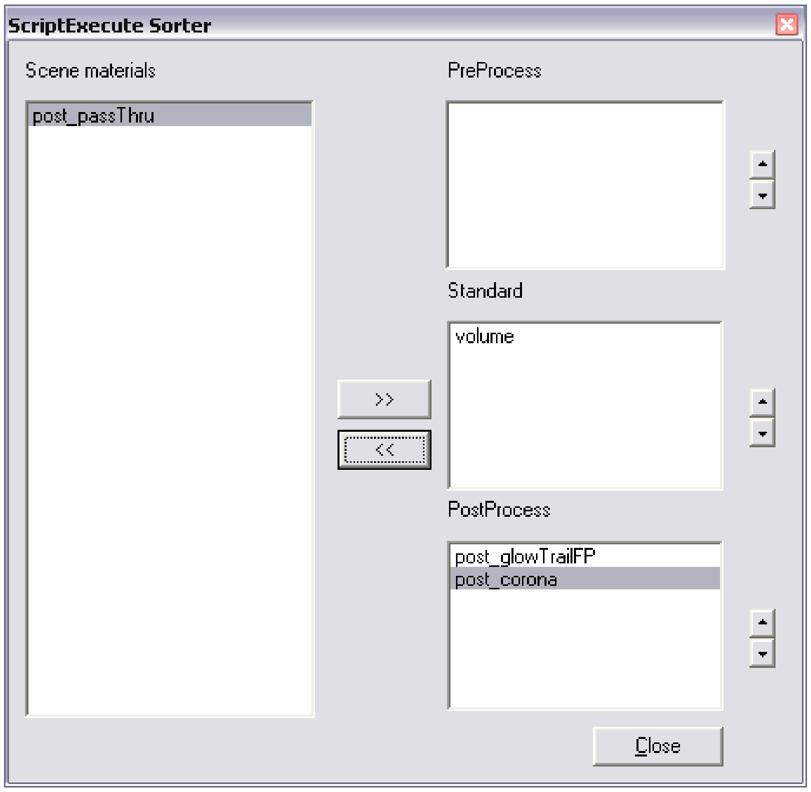
Dimensions: This functions the same as the **Zoom** icon 🔍. Selecting Dimensions changes the cursor to the **Zoom** icon. Successive clicks enlarge the materials. To reduce the materials, hold down the **Shift** key and click with **Zoom** cursor.

ScriptExecute Commands...: Displays the Scene Commands dialog with a list of ScriptExecute commands used by the material.



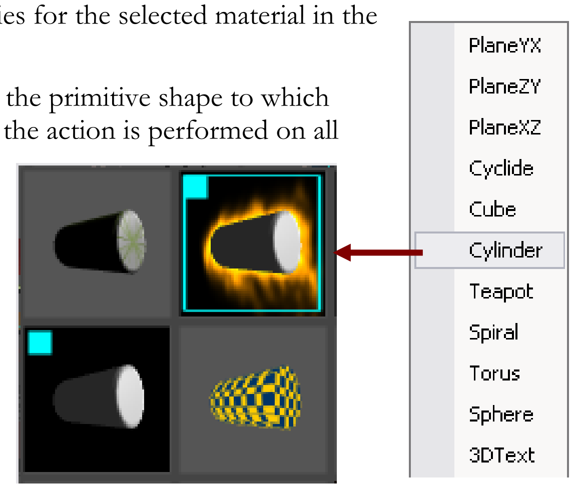
Scene Calculation
ScriptTechnique=Technique?textured;

ScriptExecute Sorter…: Displays a window that allows you to define and sort the PreProcess, Standard, and PostProcess effects used in the current scene.



Properties: Shows the properties for the selected material in the **Properties** panel.

Objects: Allows you to change the primitive shape to which materials are applied. Note that the action is performed on all materials, not just one.



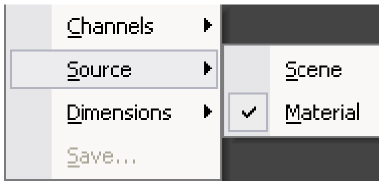
2.1.2. Textures Panel

The **Textures** panel displays the current textures for the selected material as well as any procedurally-generated textures and render targets. The **Textures** panel also enables visualization of cubemaps and normal maps.

The icons in the toolbar along the top are described in Table 1-1 on page 5. You can also Right-click anywhere in the **Textures** panel to access a context menu with the same options as the toolbar.

Color Channels: These options act the same as the icons on the tool bar. You can turn off or on the Red, Green, Blue and Alpha channels.

Source: These options act the same as the icons on the toolbar.



Scene: Functions the same as icon.

Material: Functions the same as icon.

Dimensions: These options act the same as the icons on the toolbar.

Zoom: Functions the same as the icon. Clicking on the texture with this cursor enlarges the texture in the panel. To Zoom out, hold down the **Shift** key and click on the texture.

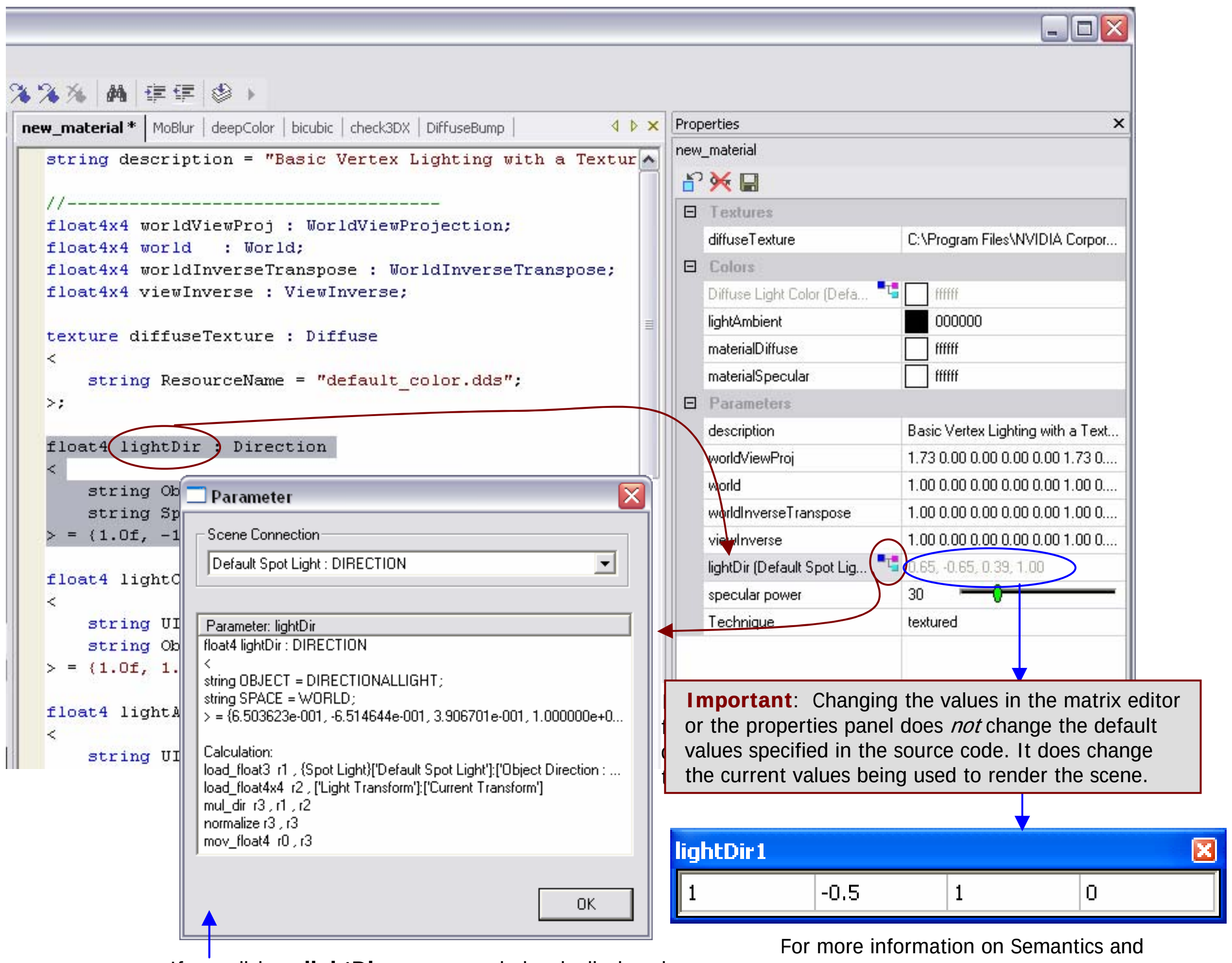
Actual Size: Functions the same as the icon. Displays textures at the actual size.

Save...: Saves textures.

2.1.3. Properties Panel

The **Properties** panel is used to view and change object properties. It is primarily used for material properties; however it can be used to view/change shapes, textures, and other items in the scene graph such as light. Table 1-1 on page 5 lists the functions of the toolbar icons.

The options displayed in the **Properties** panel are created by parsing through the current FX file and evaluating the HLSL semantics and annotations used to describe each variable. For example, a specular exponent might be declared as shown in Figure 2-2, defining how the variable should be displayed in the **Properties** panel.



```
string description = "Basic Vertex Lighting with a Textur

//-------------------------------------
float4x4 worldViewProj : WorldViewProjection;
float4x4 world    : World;
float4x4 worldInverseTranspose : WorldInverseTranspose;
float4x4 viewInverse : ViewInverse;

texture diffuseTexture : Diffuse
<
    string ResourceName = "default_color.dds";
>;

float4 lightDir : Direction
<
    string Ob
    string Sp
> = {1.0f, -1

float4 lightC
<
    string UI
    string Ob
> = {1.0f, 1.

float4 lightA
<
    string UI
```

Textures	
diffuseTexture	C:\Program Files\NVIDIA Corpor...
Colors	
Diffuse Light Color (Defa...	☐ ffffff
lightAmbient	■ 000000
materialDiffuse	☐ ffffff
materialSpecular	☐ ffffff
Parameters	
description	Basic Vertex Lighting with a Text...
worldViewProj	1.73 0.00 0.00 0.00 0.00 1.73 0....
world	1.00 0.00 0.00 0.00 0.00 1.00 0....
worldInverseTranspose	1.00 0.00 0.00 0.00 0.00 1.00 0....
viewInverse	1.00 0.00 0.00 0.00 0.00 1.00 0....
lightDir (Default Spot Lig...	0.65, -0.65, 0.39, 1.00
specular power	30
Technique	textured

Parameter

Scene Connection: Default Spot Light : DIRECTION

```
Parameter: lightDir
float4 lightDir : DIRECTION
<
string OBJECT = DIRECTIONALLIGHT;
string SPACE = WORLD;
> = {6.503623e-001, -6.514644e-001, 3.906701e-001, 1.000000e+0...

Calculation:
load_float3  r1 , {Spot Light}['Default Spot Light']:['Object Direction : ...
load_float4x4  r2 , ['Light Transform']:['Current Transform']
mul_dir  r3 , r1 , r2
normalize r3 , r3
mov_float4  r0 , r3
```

Important: Changing the values in the matrix editor or the properties panel does *not* change the default values specified in the source code. It does change the current values being used to render the scene.

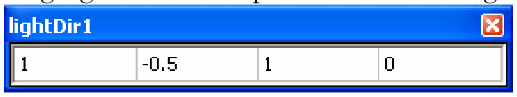
lightDir1			
1	-0.5	1	0

If you click on **lightDir**, a pop-up window is displayed showing how this object is connected to the scene and instructions that will be passed to the renderer.

For more information on Semantics and Annotations supported by FX Composer, click on **Help → Semantics & Annotations**

Figure 2-2. Using the Properties Panel

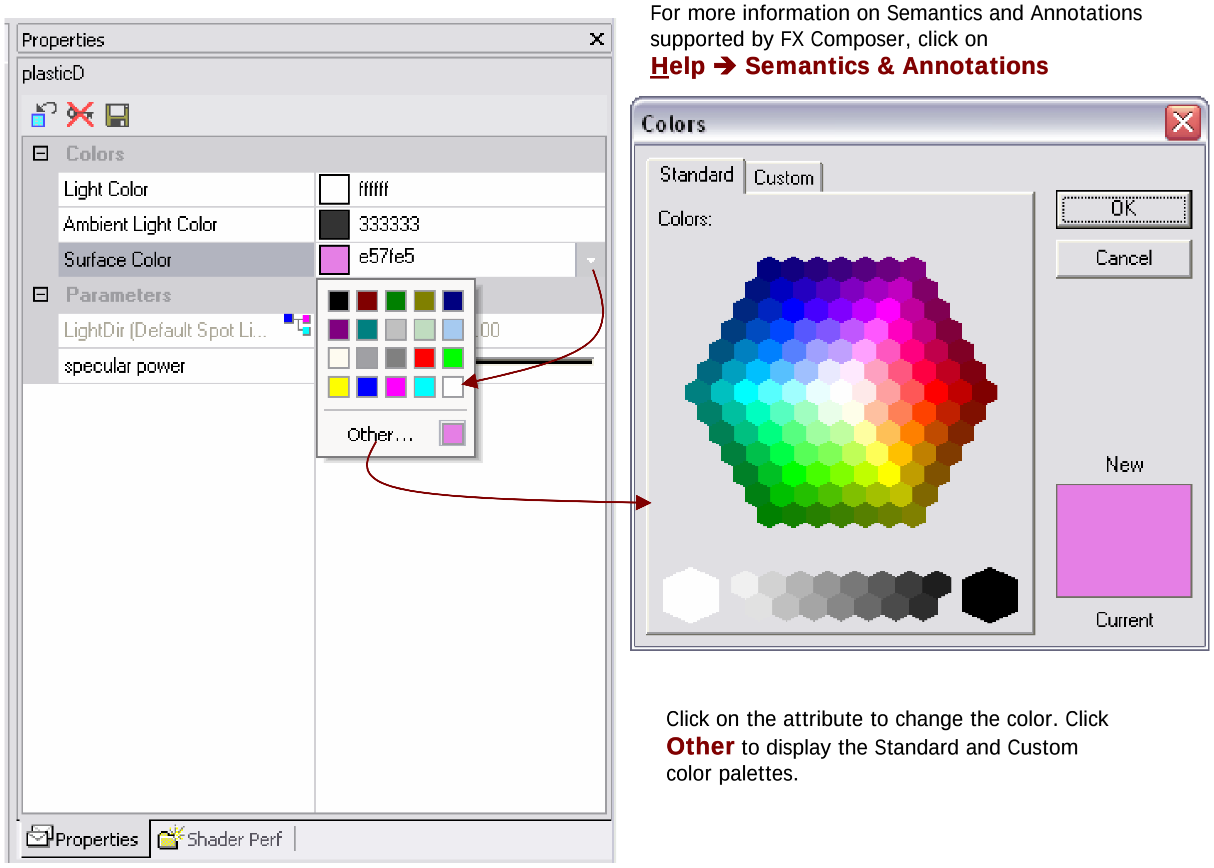
Vectors and matrices also get special treatment in the **Properties** panel. When defined with the proper semantics and annotations as shown in Figure 2-2, a matrix editor tear-off panel provides users with a convenient way to experiment with different values. Note that clicking on the scene graph icon next to **lightDir** causes FX Composer to display the assembly language calculations performed for this light in a pop-up window.



lightDir1

1	-0.5	1	0

FX Composer supports special color selection tear-off panels for editing color information (Figure 2-3). Semantics and annotations can be used to tell FX Composer that a variable is used to store color information and provide a descriptive name for the **Properties** panel.



Light Color	ffffff
Ambient Light Color	333333
Surface Color	e57fe5
LightDir (Default Spot Li...	
specular power	

For more information on Semantics and Annotations supported by FX Composer, click on **Help ➔ Semantics & Annotations**

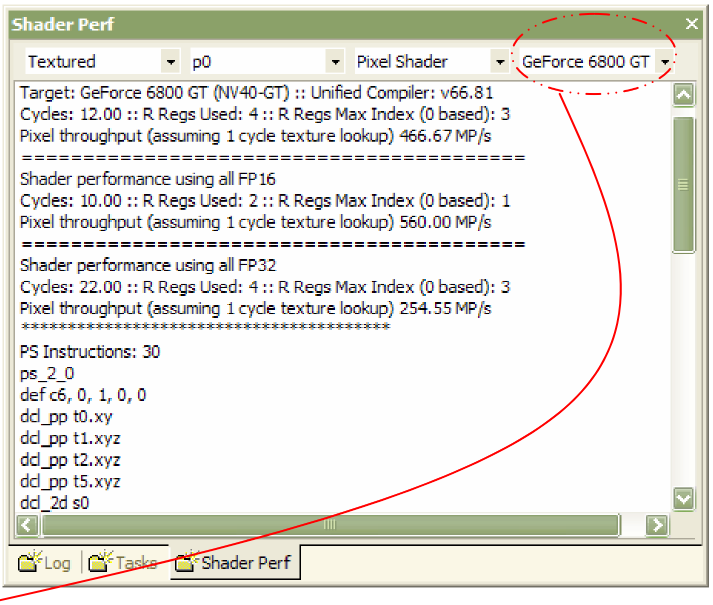
Click on the attribute to change the color. Click **Other** to display the Standard and Custom color palettes.

Figure 2-3. Color Selection Tear-off Panel

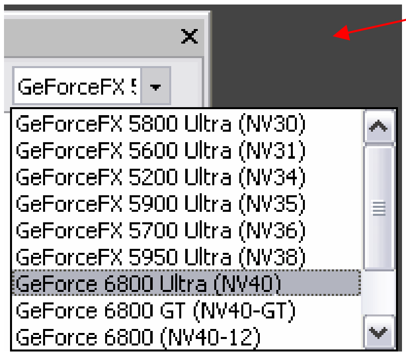
2.1.4. Shader Perf Panel

The **Shader Perf** panel (the tab is located next to the **Properties** panel tab) displays chip scheduling information for your .fx file. The **Shader Perf** panel allows you to choose the technique, the pass, either pixel or vertex shader, and the chip you want to profile.

Clicking on any material changes the data in the **Shader Perf** panel regardless if it is applied to a selection in the **Scene** panel.



```
Target: GeForce 6800 GT (NV40-GT) :: Unified Compiler: v66.81
Cycles: 12.00 :: R Regs Used: 4 :: R Regs Max Index (0 based): 3
Pixel throughput (assuming 1 cycle texture lookup) 466.67 MP/s
==========================================
Shader performance using all FP16
Cycles: 10.00 :: R Regs Used: 2 :: R Regs Max Index (0 based): 1
Pixel throughput (assuming 1 cycle texture lookup) 560.00 MP/s
==========================================
Shader performance using all FP32
Cycles: 22.00 :: R Regs Used: 4 :: R Regs Max Index (0 based): 3
Pixel throughput (assuming 1 cycle texture lookup) 254.55 MP/s
******************************************
PS Instructions: 30
ps_2_0
def c6, 0, 1, 0, 0
dcl_pp t0.xy
dcl_pp t1.xyz
dcl_pp t2.xyz
dcl_pp t5.xyz
dcl_2d s0
```


GeForceFX 5
GeForceFX 5800 Ultra (NV30)
GeForceFX 5600 Ultra (NV31)
GeForceFX 5200 Ultra (NV34)
GeForceFX 5900 Ultra (NV35)
GeForceFX 5700 Ultra (NV36)
GeForceFX 5950 Ultra (NV38)
GeForce 6800 Ultra (NV40)
GeForce 6800 GT (NV40-GT)
GeForce 6800 (NV40-12)

All supported GPUs are listed are listed in the drop-down menu.

2.1.5. Scene Panel

The **Scene** panel displays the current scene and has the usual controls for manipulating scenes. FX Composer includes GeoPipe plug-ins that support importing scenes stored in .nvb and .x files, both of which can contain skinning information. Use the **File ➔ Import Scene...** command to load a scene.

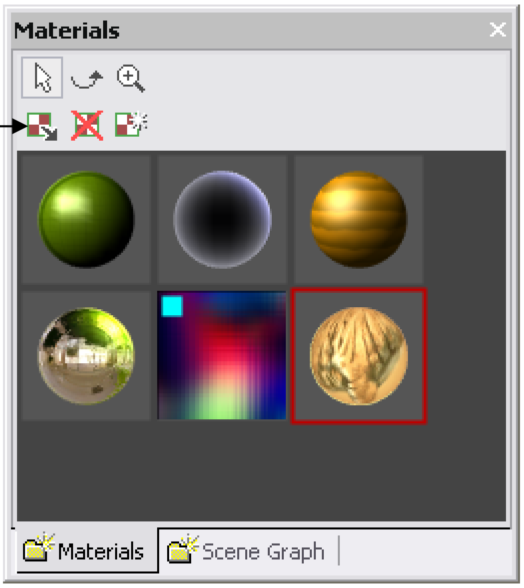
Select an object in the scene and apply a different material to it to change its appearance. Note that the light source remains the same; only the material of the surface changes.

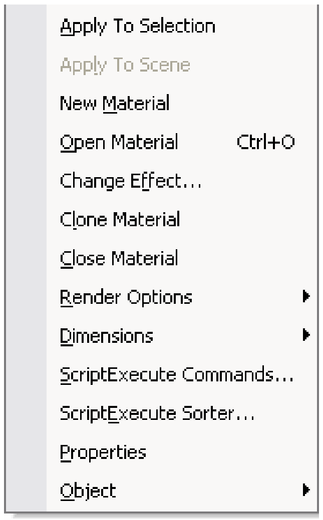
Frame: 0
Perspective

Frame: 0
Perspective

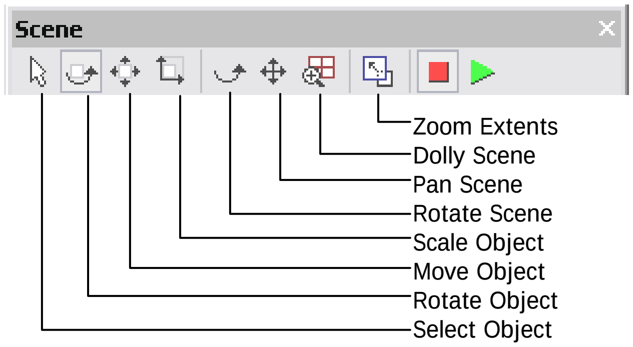
Select a material and click **Apply**. The material is applied to the selected object in the **Scene** panel.

You can also Right-Click on the material and select **Apply to Selection.**

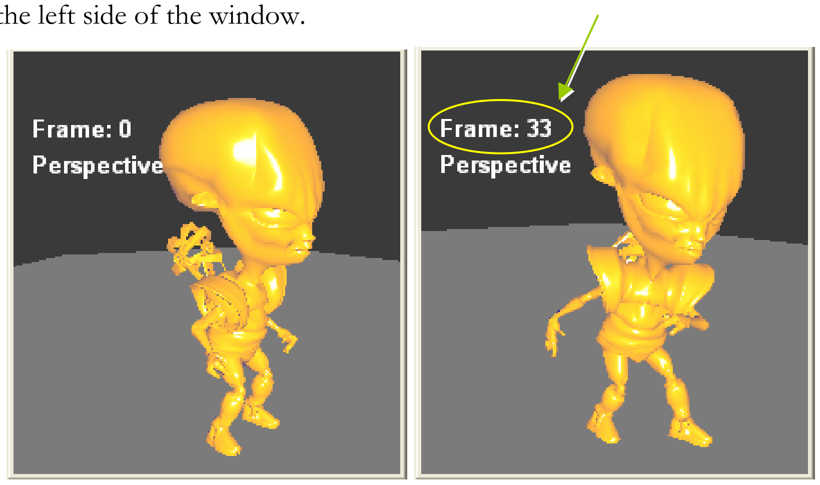




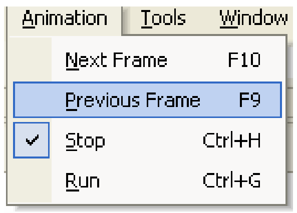
Use the tool icons in the **Scene** panel to manipulate objects and scenes.



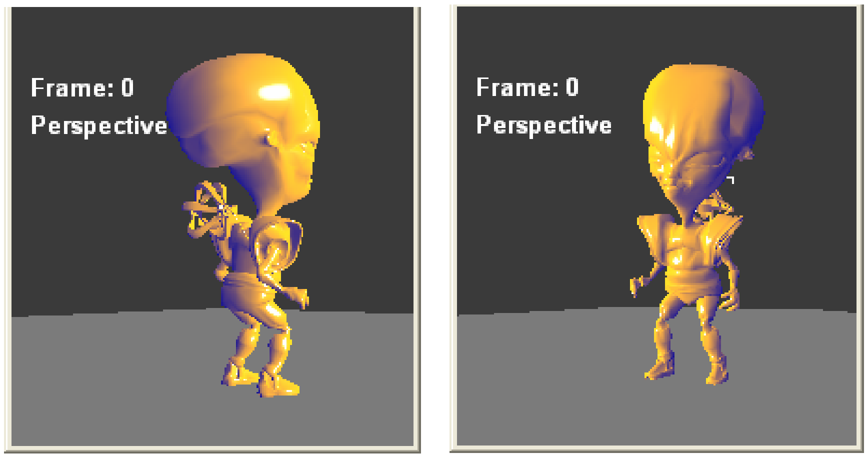
Use the animation controls in the toolbar to run the frames and see the animation associated with your scene. Note that the frame displayed is shown on the left side of the window.



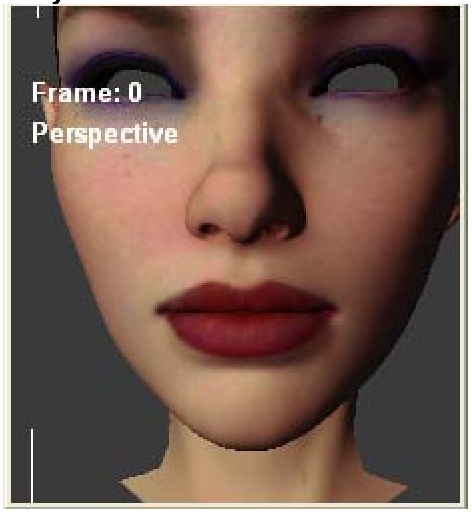
To move frame-by-frame, use Animation in the application toolbar or **F10** for Next frame and **F9** for Previous Frame.



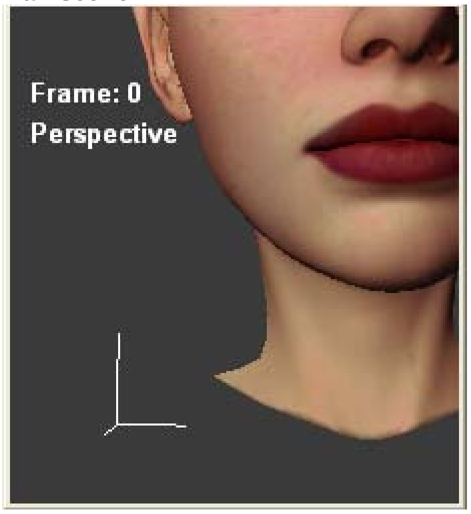
Rotated Scene



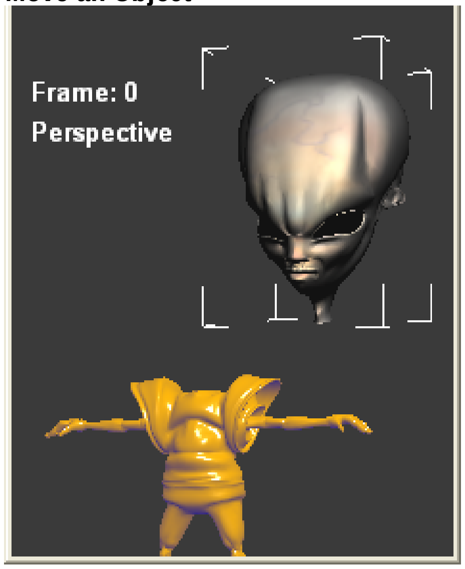
Dolly Scene



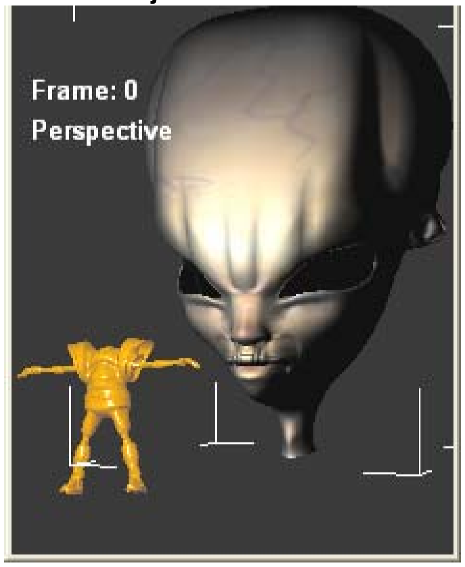
Pan Scene



Move an Object



Scale an Object

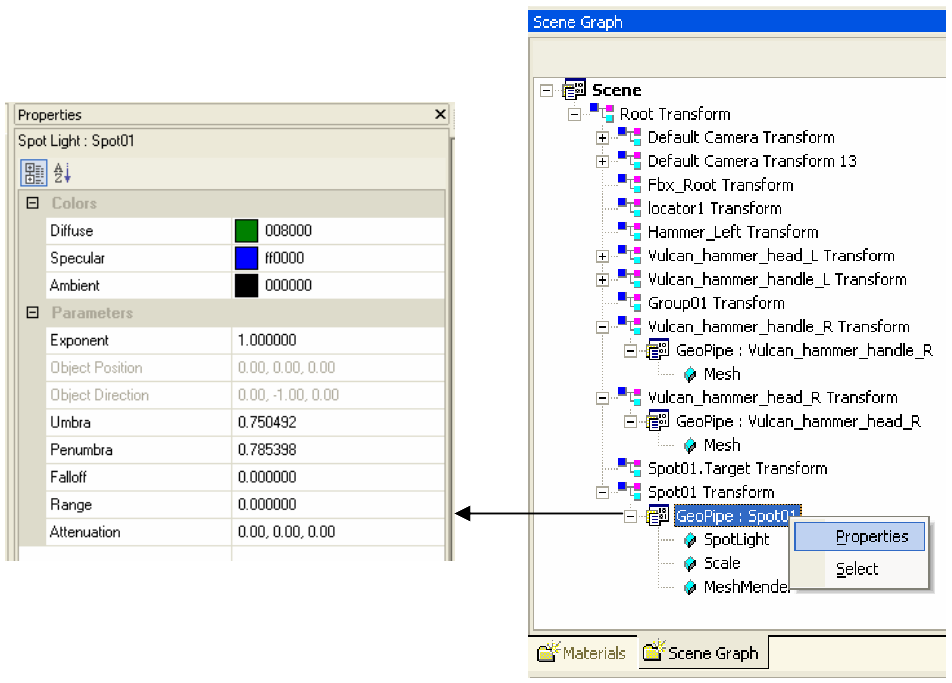


Note that turning on **Tools ➔ Options ➔ Draw** will always cause FPS to be displayed in the upper left.

2.1.6. Scene Graph Panel

The **Scene Graph** panel is used to browse through the current scene transform hierarchy and select objects. You can use it to select items that wouldn't easily be available in the UI, such as individual bone transforms, etc. It is also useful to see how a scene is built.

To use this window, you can, for example, select GeoPipe : Spot01 and then Right-Click to display the GeoPipe properties in the **Properties** panel (if not already displayed).

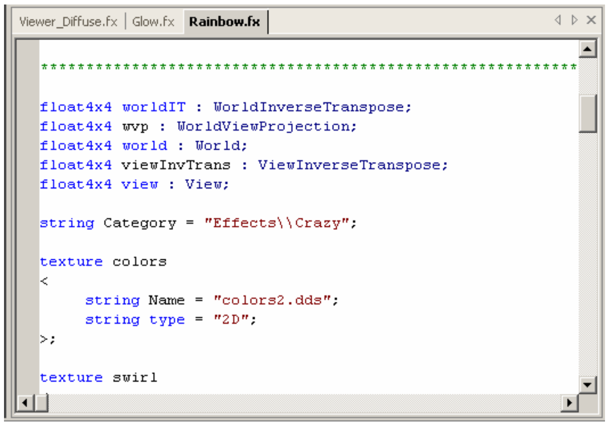


Spot Light : Spot01

Colors	
Diffuse	🟩 008000
Specular	🟦 ff0000
Ambient	⬛ 000000
Parameters	
Exponent	1.000000
Object Position	0.00, 0.00, 0.00
Object Direction	0.00, -1.00, 0.00
Umbra	0.750492
Penumbra	0.785398
Falloff	0.000000
Range	0.000000
Attenuation	0.00, 0.00, 0.00

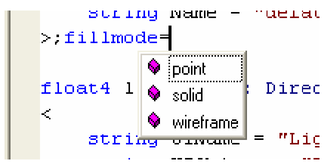
2.2. Editing Area

The editing area contains the FX file code and is enabled with syntax highlighting (keywords in colors). It acts like the editor in Microsoft Visual Studio. All Material files opened are listed as tabs across the top of the panel to allow you to switch between files easily. Use the scrollbars and bookmarks to move through the file.



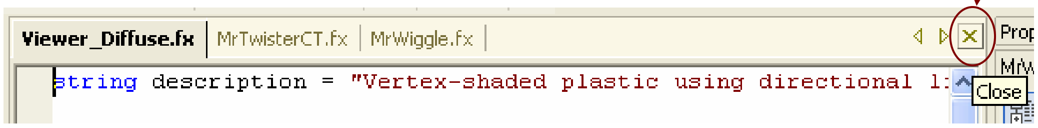
```
***************************************************************

float4x4 worldIT : WorldInverseTranspose;
float4x4 wvp : WorldViewProjection;
float4x4 world : World;
float4x4 viewInvTrans : ViewInverseTranspose;
float4x4 view : View;

string Category = "Effects\\Crazy";

texture colors
<
     string Name = "colors2.dds";
     string type = "2D";
>;

texture swirl
```

The editing area uses syntax highlighting to automatically color HLSL keywords and comments. It also provides convenient intellisense completion, allowing you to select from a list of supported keywords appropriate in the current context. For example, you can type fillmode= and select from a list of options that are displayed.



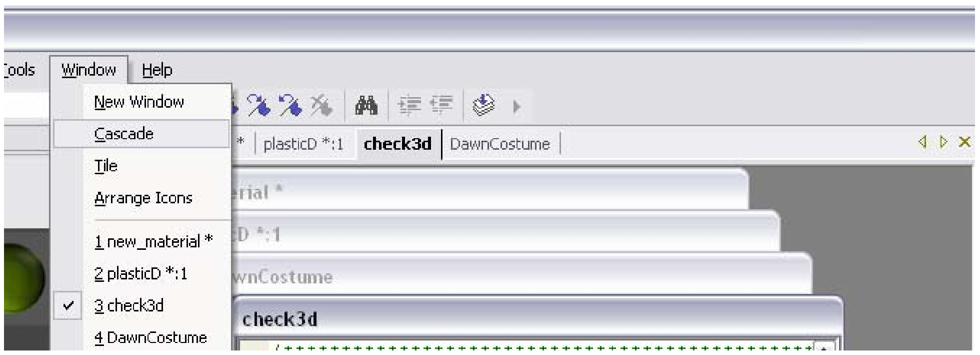
```
>;fillmode=

float4 l                : Direc
<
     string             = "Lig
```

Every time you open a project or a material, a tab is placed along the top of the editing area. Clicking on the tab displays the source code in the editing area and the objects properties in the **Properties** panel. To close a project or material, click on the tab and then click the X box (Close) located at the far right.



```
string description = "Vertex-shaded plastic using directional l:
```

The editing area acts like a window in that you can cascade the files, tile the files, or open a new file window. Use the **Window** option on the application tool bar to manipulate the FX file displays.



Chapter 3. Full Scene Effects

FX Composer supports the *Microsoft DirectX Semantics and Annotations Specification* (DXSAS).

The *custom* scene commands supported in earlier versions of FX Composer have been deprecated.

Waiting for Chris M. Text….

Chapter 4. Miscellaneous Hints and Tidbits…

4.1. FX Composer Projects and Packages

This section detail the difference between the two storage formats; .fxcomposer (referred to as a *package*) and .fxproj (referred to as a *project*). The .fxproj file format is new to FX Composer.

Both file formats describe a complete scene, with shader, texture & geometry information. The essential difference between the two formats is how the various components are stored:. Other differences are:

- Packages keep textures as part of the package, Projects reference textures.
- Packages reference a binary geometry chunk for optimal storage; Projects embed geometry as base64 encoded.
- Packages store .fx files separately; Projects store .fx files as part of the XML.

4.1.1. FX Composer Package Files

The .fxcomposer package files are actually .zip archives that contain the following components:

- An XML scene/material description (`SceneGraph.xml`)
- A binary component for mesh/animation data (`Binary.bin`)
- .fx files and textures.

The project file tries to be the best of both worlds: an easy to parse, human-readable XML file format, with a binary chunk for data that doesn't store well as text (the XML refers to this data via an offset).

Package files are ideal for shipping a complete scene with effects and textures in a single binary. They are also compressed and store geometry optimally, so they save space. The package files are not as good if you have a lot of packages all using the same texture maps—the result is duplicated media being stored in each package.

4.1.2. Projects (.fxproj)

Projects are standalone XML files describing an entire FX Composer scene and its geometry. Geometry data is stored as base64 encoded binary at various nodes in the XML.

Project files are a great way to store your scene if you want to hand modify the file or read it from an external tool. They also save file size, since they store references to texture files, not the actual media.

The disadvantage of Projects is that you need to ship the textures with the project.

4.1.3. XML Data

Both projects and packages have similar XML tree elements. The main difference is that geometry data is stored in a separate binary chunk for packages, but it is embedded in the XML for projects (because they are intended to be standalone). As far as the XML content is concerned:

- The first line or so contain the file version and the build version of FX Composer used to save the file.
- Next there is the project-specific search paths.
- Then the scene graph with materials, hierarchy, parameter data, etc.
- Next, a list of FX Composer objects/plug-ins required to successfully display this package.
- Last in the file are the original paths of the media objects in the .fxcomposer compressed zip file so that FX Composer knows where they were originally stored.

Full paths to all media and .fx files are always stored in the FX Composer package or project. When a project or package is opened, FX Composer looks for the original drive copies. If the original drive copies are not found, FX Composer loads the archived version into a *MediaCache* directory underneath the main binary. If found, you are prompted to use the alternative. This behavior is configurable under **File→Settings**….

4.2. Configuration Files

The FX Composer/data directory has a few useful configuration files that can be changed. All files are XML, and should be backed up before you tweak!

- **`fxcomposer_config.xml`**
  Contains the list of standard materials offered when you click the 'new material' button. You can add your own standard materials along with text descriptions here, useful to give your artists a game-specific list of effects to use.
  `fxedit.xml`
  Contains rules for syntax highlighting. You can change the keywords that get highlighted and the colors used to highlight them.
- **`fxmapping.xml`**
  Lists the standard set of semantics and annotations supported by FX Composer. The `<semantic name="foo">` field represents FX Composer's standard internal names for these values. Optional `<mapping name="foo">` fields enable you to remap your semantic/annotation names to those used by FX Composer.
- **`defaultscene.fx`**
  This is FX Composer's default scene command list. It is a very simple instruction list which simply clears the background to grey and draws all the elements in the scene. You can change the contents of this file and recompile it, just like other scene command materials; the advantage being that you will permanently change the default rendering method of FX Composer. Use with care!
- **`plugins.inf`**
  Lists the plugins FX Composer should load, including ShaderPerf and geometry import plugins for .NVB and .x files. A future FX Composer SDK will allow developers to create their own plugins.

4.3. Mesh Sections

Both .nvb and .x have mesh sections. But .x has the concept of polygon attributes which mean that within a mesh section you can have different material properties. .Eeach mesh section in a .nvb file has its own material.

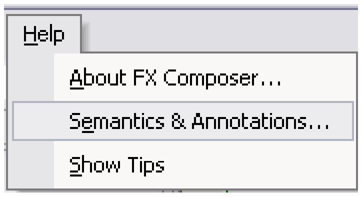
It's up to the FX Composer importer as to how it gets this data in. Internally FX Composer has mesh sections, and per-polygon materials. But the UI currently only exposes the mesh sections in the scene window (you can also modify all materials in the Materials panel).

An FX Composer material refers to a .fx file, and a list of parameters for that .fx file. More than one material can refer to the same .fx file with different parameters for each material.

In the future, FX Composer will get better at manipulating these things.

4.4. Semantics & Annotations

The Help files contain a link to a list of known semantics and annotations from the Microsoft specification. The Supported column indicates values that FX Composer currently supports. The FX Composer Only column indicates additional semantics/annotations that are not part of the Microsoft specification, but are supported by FX Composer. This list is continually updated.



Note that all semantics and annotations are case insensitive

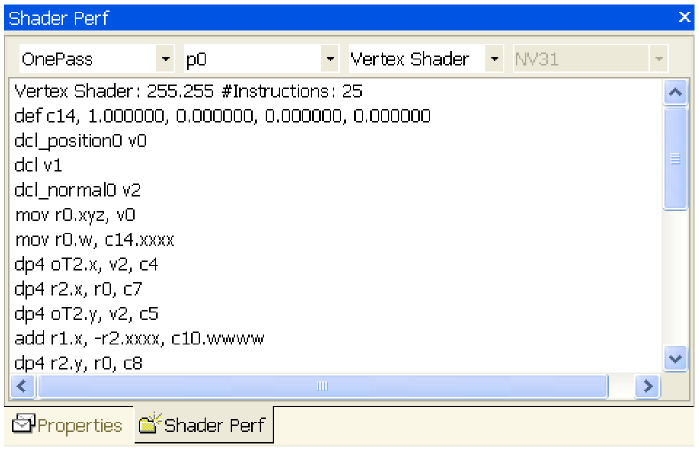
Tutorial #1
Measuring and Managing Shader Performance with FX Composer

FX Composer does more than provide you with a way to view, edit, and test shaders on models and scenes. By using FX Composer's performance tools and **Shader Perf** Panel display, you can assess and modify the efficiency and performance of your shaders—not just in general terms, but even for running on various different specific GPUs.

In this brief tutorial, we'll look at a sample shader file and make some changes to see how they affect the performance of that shader. You will find the accompanying sample code in \MEDIA\fxcomposer\tutorials\perf_tutor.fxcomposer project.

Sources of Performance Information

FX Composer provides performance information through two major sources: the **Shader Perf** panel and the **Scene** panel. These panels are visible by default – the **Scene** panel displays your model with the currently-assigned shader(s), and the **Shader Perf** panel will be hidden under the Properties panel (tap the Shader Perf tab to reveal it). Both panels are dockable anywhere on screen or can be their own free-floating windows. If you don't see these panels, you can enable them from **View➔Panels➔ShaderPerf** panel to see them.



```
Vertex Shader: 255.255 #Instructions: 25
def c14, 1.000000, 0.000000, 0.000000, 0.000000
dcl_position0 v0
dcl v1
dcl_normal0 v2
mov r0.xyz, v0
mov r0.w, c14.xxxx
dp4 oT2.x, v2, c4
dp4 r2.x, r0, c7
dp4 oT2.y, v2, c5
add r1.x, -r2.xxxx, c10.wwww
dp4 r2.y, r0, c8
```

The **Shader Perf** panel contains a large text area showing you the compiled DirectX 9.0 assembly-language instructions that are executed for your HLSL shader. The pulldown menus at the top of the **Shader Perf** panel allow you to select each part of the selected HLSL FX shader – the technique, pass within that technique, vertex or pixel shaders, and (for pixel shaders) specific profiling based upon the GPU used (DirectX 9 is smart enough when compiling to consider this aspect at runtime). This last pulldown allows you to evaluate shader performance against a wide range of GPUs, from low-end consumer to the high-end workstation, to assess their specific problems and find their "sweet spots."

Performance analysis is presented in the main body of the **Shader Perf** panel, first with an instruction count before the shader begins, and then (for pixel shaders) with a report of its scheduling efficiency at the bottom of the listing.

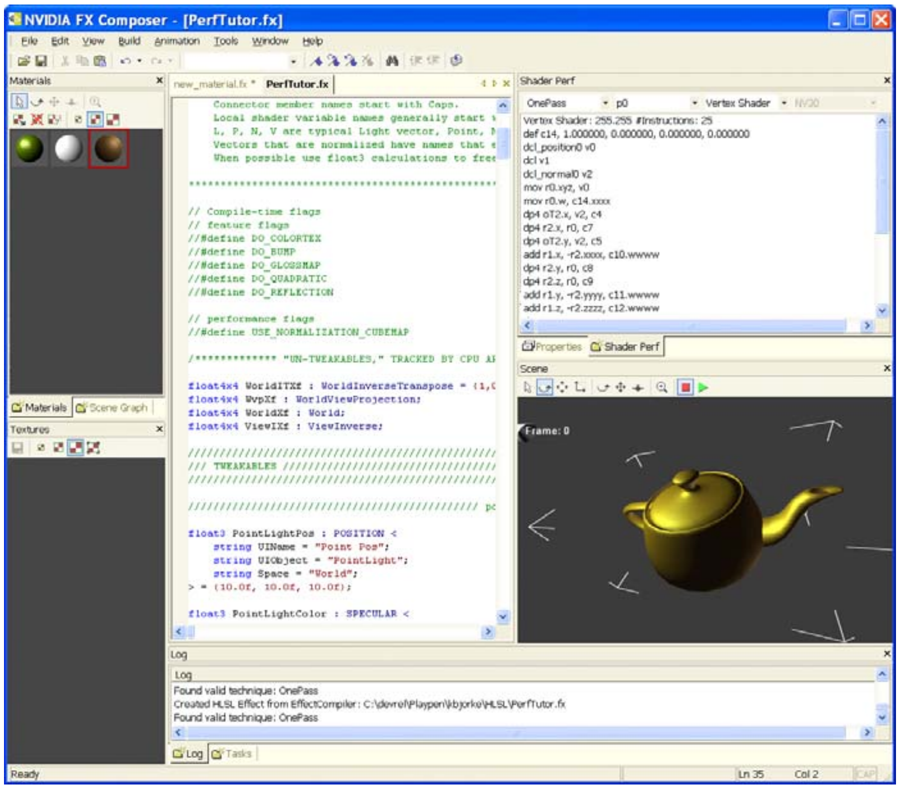
The **Scene** panel can give you further assessment of the efficiency of your shader, by displaying a frames-per-second number for the render. This number will be visible for supported GPUs (that is, those with scheduling info available) when **Tools➔Options➔Draw Always** is turned on.

Try It Yourself:
The PerfTutor.fx Shader

To try the Performance Tools yourself, use a shader called PerfTutor.fx. The PerfTutor.fx is a very generic shader using per-pixel lighting, two different kinds of lamps, and a variety of options (set by #define flags) that permit you to try different methods to get the same or nearly-the-same results and see the effects on final image and scene performance.

Use the following steps to create the scene in the example shown in Example 1:

1. Start FX Composer.
2. Make sure that the **Scene** panel is visible and the **Shader Perf** panel is available.
3. From the **File** menu, select **File➔Load Project.**
4. Open **MEDIA/fxcomoser/tutorials/perftutor.fxcomposer.**
5. Right-click on the **Scene** panel and select **New Scene From Shape➔Teapot.**
6. Select the **teapot** geometry in the **Scene** panel.
7. Select the **Perf Tutor** material in the **Materials** panel.
8. Right-click on the material and select **Apply to Selection** to assign the **PerfTutor** material to the teapot. Your display should look something like the shown in Example 1.



```
    Connector member names start with Caps.
    Local shader variable names generally start
    L, P, N, V are typical Light vector, Point,
    Vectors that are normalized have names that
    When possible use float3 calculations to free

****************************************************

// Compile-time flags
// feature flags
//#define DO_COLORTEX
//#define DO_BUMP
//#define DO_GLOSSMAP
//#define DO_QUADRATIC
//#define DO_REFLECTION

// performance flags
//#define USE_NORMALIZATION_CUBEMAP

/************* "UN-TWEAKABLES," TRACKED BY CPU AP

float4x4 WorldITXf : WorldInverseTranspose = (1,0
float4x4 WvpXf : WorldViewProjection;
float4x4 WorldXf : World;
float4x4 ViewIXf : ViewInverse;

/////////////////////////////////////////////////
/// TWEAKABLES /////////////////////////////////
/////////////////////////////////////////////////

//////////////////////////////////////////////// po

float3 PointLightPos : POSITION <
    string UIName = "Point Pos";
    string UIObject = "PointLight";
    string Space = "World";
> = (10.0f, 10.0f, 10.0f);

float3 PointLightColor : SPECULAR <
```

```
Vertex Shader: 255.255 #Instructions: 25
def c14, 1.000000, 0.000000, 0.000000, 0.000000
dcl_position0 v0
dcl v1
dcl_normal0 v2
mov r0.xyz, v0
mov r0.w, c14.xxxx
dp4 oT2.x, v2, c4
dp4 r2.x, r0, c7
dp4 oT2.y, v2, c5
add r1.x, -r2.xxxx, c10.wwww
dp4 r2.y, r0, c8
dp4 r2.z, r0, c9
add r1.y, -r2.yyyy, c11.wwww
add r1.z, -r2.zzzz, c12.wwww
```

```
Found valid technique: OnePass
Created HLSL Effect from EffectCompiler: C:\devrel\Playpen\kbjorke\HLSL\PerfTutor.fx
Found valid technique: OnePass
```

Example 1. perftutor.fx

The PerfTutor.fx file contains a number of compile-time #define flags that let you change the way the code functions with a minimum of typing. These flags can turn on and off specific features of the shader, or control the way shading is calculated. The flags are right at the head of the shader code:

```
// Compile-time flags
// feature flags
//#define DO_COLORTEX
//#define DO_BUMP
//#define DO_GLOSSMAP
//#define DO_QUADRATIC
//#define DO_REFLECTION

// performance flags
//#define USE_NORMALIZATION_CUBEMAP
```

By default, these flags are all commented-out. We'll change them later to see their effects on final shader performance.

Looking at the Shader Perf Panel

Playing with the pulldowns **Shader Perf** panel, you can select any of the available techniques within PerfTutor.fx (OnePass, MultiPass, AmbiOnly, DirOnly, PtOnly).

Select **AmbiOnly**, the pass **p0** (the only pass in this particular technique), **Pixel Shader**, and **NV30**.

The display of the shader panel fills with NV30-specific information about this particular (a deliberately very short) pixel shader:

```
Pixel Shader: 255.255 #Instructions: 7
def c2, 1.000000, 0.000000, 0.000000, 0.000000
mov r0.rgb, c1
mul r0.rgb, r0, c0
mov r0.a, c2.r
mov oC0, r0
end
Scheduling:
This shader has a general efficiency rating of 100%
Passes: 3 Effective Passes: 0.00
--- # of R register: 2 ---
```

GPU Efficiency

Switching the hardware profile to NV36 changes the lines at the bottom slightly:

```
Scheduling:
This shader has a general efficiency rating of 33%
There are opportunities to change a MOV instruction (executed on
the SHD) to a MUL by 1 instruction for execution on CMB/FPB
Passes: 3 Effective Passes: 0.00
--- # of R register: 2 ---
```

What does this mean? Will the shader run slower on the (usually) faster NV36 GPU?

Fortunately not – what it does mean is that this shader could be tweaked to run even faster on NV36 by adjusting the code to use the NV36's greater floating point resources. For such a small pixel shader, the code would need careful tweaking. For a larger shader, the compiler is clever enough to adjust things on the fly for maximum performance. Efficiency numbers rate the shader against what the maximum possible throughput of the specific GPU might be (a shader perfectly-fitted to the NV36 could potentially run three times faster than this one).

Shader Passes

Let's switch to a more complex example. Select "OnePass" and the Pixel Shader. The compiled code for NV30 is now much longer – 48 instructions. At the bottom, we'll see:

```
Scheduling:
This shader has a general efficiency rating of 100%
Passes: 68 Effective Passes: 0.00
--- # of R register: 4 ---
Switching profiles to the NV36:
Scheduling:
This shader has a general efficiency rating of 49%
There are opportunities to change a MOV instruction (executed on
the SHD) to a MUL by 1 instruction for execution on CMB/FPB
Passes: 40 Effective Passes: 0.00
--- # of R register: 4 ---
```

While the efficiency rating of resource usage on the NV36 has dropped, look at the number of passes. On NV30, this shader would require 68 cycles through the shader core to execute – but on NV36, with its improved floating-point performance, only 40 passes are needed. This equates to an automatic improvement of around 30% (even at the same clock speed), over the older NV30 architecture.

Tweaking the Code

In this example, we are going to set a compile-time flag and see the results on shader performance.

In the **Editing** Panel, un-comment the definition of
`//#define USE_NORMALIZATION_CUBEMAP`
by removing the leading `//` marks so the line appears as:

```
#define USE_NORMALIZATION_CUBEMAP
```

You are able to see the color shift immediately from green to blue and black, showing that FX Composer recognizes these instructions.

Now press **Ctrl-S** or select **File→Save** to save these changes and press the **Build** button to recompile the shader (or select **Build→Compile** or press **Ctrl-F7**). The panel information is updated. You can see a new texture in the **Textures** panel (if displayed), showing a normalization cube map. When this option is enabled, the cube map is generated by PerfTutor.fx, using the DirectX 9.0 virtual machine (the specific functions are defined in the header file normalization.fxh). PerfTutor.fx is written to replace all calls to `normalize()` in the pixel shaders with `texCUBE()` calls when this texture is present.

What effect does this have on our shader performance numbers? We can check in the **Shader Perf** Panel.

The newly-compiled shader actually has *more* assembly-language instructions—59 instructions, versus the previous 48—but checking the efficiency and pass counts tell us that instruction counts can be deceptive.

For NV30, still at 100% efficient use of the GPU, the number of shader passes, the count of clock cycles needed to execute this shader, has declined from 68 to 60, a 13% improvement. So, even though the shader itself is *longer*, the execution of the shader on NV30 is *faster*.

The story for NV36 is even stronger. The GPU efficiency rises slightly, and the number of passes drops from 40 to 30—a 33% speed increase for this high-end GPU.

Frames per Second

For GPUs that the performance analyzer recognizes an FPS counter can be displayed in the Render panel. To see the counter, turn on the **Tools➔Options➔Draw Always** option, and then either wiggle the model in the render window using the transform tools (e.g., **Rotate Scene**) or press the green **Run Animation** button (you can use **Ctrl-G**/**Ctrl-H** to toggle animation on/off).

A few things to remember when using this frame counter:

- The frame counter is showing you the speed of FX Composer in this window on this model. It is not showing you the frame rate of your model in your game engine. The frame counter is there as a means for you to evaluate relative performance between shader methods within FX Composer on your computer with your GPU—not as an absolute counter of efficiency and speed. Use the **Shader Perf** panel numbers to create those estimates and calibrate them against your game engine's own characteristics.
- The technique used in the **Scene** panel is the one specified in the **Properties** panel for this shader—not the technique specified in the **Shader Perf** panel.
- While the **Shader Perf** panel can evaluate many different GPUs, the **Scene** panel shows you the value for just one—the GPU installed in your system.
- When **Draw Always** is engaged, FX Composer is using the CPU to the maximum. When **Draw Always** is turned off, FX Composer only draws the scene when something changes. If you're multitasking (running Microsoft's Visual Studio or a game engine simultaneously with FX Composer) you may see resource fighting. Turn off **Draw Always** to get best performance from external applications.

Moving Forward

Now you can experiment with setting other PerfTutor options, checking the characteristics of other GPUs, editing the code, and more importantly, applying these same lessons to your own .fx shaders. Have fun!

Tutorial #2. Optimizing a Bump Mapping Shader

Building on what you learned in Tutorial #1, this tutorial provides some additional techniques you can use in performance tuning your shaders, using a simple tangent-space bump mapping example. You can find the accompanying sample code in the following location:

`\MEDIA\fxcomposer\perf_bumpplastic.fxcomposer project`.

Shader Optimization

Optimizing pixel shaders for modern graphics processors is a non-trivial task. In previous generations of hardware there was usually a one-to-one correspondence between pixel shader assembly code and the instructions that got executed by the hardware. One could often assume that each instruction would take a single clock cycle to execute. As GPUs have become more complex, and more like CPUs in many ways, the gap between the assembly language abstraction and the hardware has become larger. Modern GPUs include multiple pixel pipelines, each of which may include multiple math units and texture units. This means that the GPU can potentially execute multiple math and texture instructions in a single cycle. The ordering of instructions can affect the extent to which these execution units are used optionally.

Performance is also affected by register usage. The GPU processes groups of pixels in parallel, but only has a finite amount of register storage. Therefore using more registers means that fewer pixels can be processed at a time, and performance is reduced. NVIDIA GPUs support 16-bit "half" precision floating point numbers, which can help to improve performance by reducing register storage requirements. Data dependencies between instructions can also affect performance. In addition, these performance characteristics can vary widely between different GPUs. Using high level shading languages such as HLSL adds another layer of abstraction which can further obscure performance issues.

Fortunately, technology has come to the rescue. NVIDIA's Unified Compiler, which is built into the graphics driver, takes high-level pixel and vertex shader assembly code and generates optimized micro-code for the target GPU. The **Shader Perf** panel in FX Composer simulates the Unified Compiler to allow you to analyze the performance of your shaders and report how they will execute on different GPUs.

Despite the automated optimization in the Unified Compiler, there are still a few simple rules that you can follow to ensure that your shaders run as fast as possible on all hardware.

- **Optimize your algorithms**
  Premature optimization is the root of all evil. Before attempting any of the lower-level optimizations described below, take a high-level look at the shader and see if there is some way you could rearrange the code so that it that requires less work.
- **Perform calculations only as often as necessary**
  Some calculations can be moved from the pixel shader to the vertex shader, and the results passed to the pixel shader as texture coordinates. A common example is calculating a view vector. This varies linearly, so can be calculated per-vertex and interpolated. Texture coordinate interpolation is not always free, but this is almost always a performance win overall. In a similar way, sometimes calculations can be moved off the hardware completely. For example, multiplying a light color by a material color can be done on the CPU.
- **Replace complex math functions with texture look-ups**
  Complex expressions such as lighting functions can be encoded into textures, and then accessed using a single texture lookup. This sacrifices some interactivity (the parameters can not be changed without rebuilding the texture), but is almost always faster on current hardware. The FX runtime includes a virtual machine that allows you to write functions in HLSL that can be used to generate textures very easily.
- **Use half precision where possible**
  Half precision is almost always sufficient for representing colors and unit-length vectors, and requires half the register storage. Vectors representing distances in world coordinates will usually need to use full float precision. Remember that there are an infinite number of real numbers, but half precision floats can only represent 65536 of them!
- **Use lower pixel shader versions if possible**
  GeForce FX series GPUs include support for the fixed-point type and instructions used in Direct3D Pixel Shader v1.4 and below. If your shader can be expressed in a lower version pixel shader, it may run faster than the equivalent operations in pixel shader v2.0 and FX Composer supports all of these shader profiles

FX Composer Shader Performance Tools

The **Shader Perf** panel in FX Composer makes it easy to see how much modifying your shader effects performance. It is based on an architectural simulator of the GeForce FX shader hardware, and can simulate hardware from the GeForce FX 5200 to the FX 5950. It displays four important statistics:

- **PS Instructions**
  The number of Direct3D pixel shader instructions used
- **Cycles**
  The number of clocks cycles the optimized shader will take to execute on the selected hardware. Note that this is often lower than the number of instructions, since multiple instructions can be executed per clock cycle.
- **# R registers**
  This is how many full-precision "R" registers the optimized shader uses. This is affected by how many temporary variables your shader uses, and can have a big effect on performance. Note that two half-precision registers can fit in a single R register, so if you are using half-precision, double this number to get the actual number of registers used.
- **GPU utilization**
  This is a measure of how well the shader utilizes the GPU's math and texture functional units. Sometimes rearranging your code or switching to half precision will allow the optimizer to make better use of these units.

Optimizing Your Shader One Step at a Time

Once you have loaded the perf_bumpplastic.fxcomposer project, click on the BumpPlasticPerf.fx material to make it the active material and look in the Properties panel. Each step in optimizing the shader is implemented as a separate "technique". You can switch between the techniques by selecting them from the **Techniques** pull-down menu in the properties panel. By clicking on the **Shader Perf** panel and selecting **Pixel shader** from the pull down menu, you can compare the performance of the different optimization steps.

The shader implements simple tangent space bump mapping with a color map, using the Blinn/Phong lighting model.

If you read through the BumpPlasticPerf.fx file you will see the section of HLSL code that corresponds to each technique. The vertex shader is the same for each technique; we only modify the pixel shader.

Step 1. The initial version of the shader **`BumpPlasticPS_0`** uses math for the lighting calculation. It calls a function **Phong** to calculate the Blinn/Phong lighting model given the dot products between the normal vector, light vector and half angle vector. It uses float precision for everything.

Step 2. In this version of the shader, we remove the code to normalize the normal vector "N". Since the normal is being read from a normal map, we can assume that it will be almost unit length, apart from de-normalization caused by linear texture filtering. If you switch between step 1 and step 2 you might be able to notice a slight change in the appearance of the specular highlights. Note that although this only reduces the number of instructions by one (the **`NRM`** instruction), it reduces the number of cycles by 2 because the **`NRM`** macro is expanded to several instructions (**`DP3`**, **`RSQ`**, **`MUL`**).

Step 3. In this step we replace the lighting math with a texture lookup. We use the same **Phong** to build a texture that encodes the lighting function. In the shader we can then replace the function call with a 2D texture lookup based on **`N.L`** and **`N.H`**.

Step 4. This is identical to step 2, but we have changed all the variables to half precision. This reduces the number of R registers required to 2.

Step 5. In the final step we convert the shader to pixel shader version 1.1. Obviously this will not be possible with all shaders, but for this example it illustrates the performance benefits..

The table below shows the relative performance of each optimization step. Frames per second numbers were measured on a NV35 with a full screen window.

Step	FPS	Instrs.	Cycles	#R regs	Comment
0	75	23	14	4	Original shader
1	85	22	12	4	Remove normalize (N)
2	103	21	9	4	Replace math with texture lookup
3	143	21	5	2	Change floats to half precision
4	148	10	4	2	Convert to PS1.1

Working through all four optimization steps in this sample shader yields a 2x performance improvement (measured by FPS). The largest benefits come from replacing math with texture lookups and using half precision. By following these same simple rules and making use of the performance tools in FX Composer, you can ensure that your shaders will run as fast possible and provide the best end user experience on all hardware.

Index

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com