Practical Performance Analysis

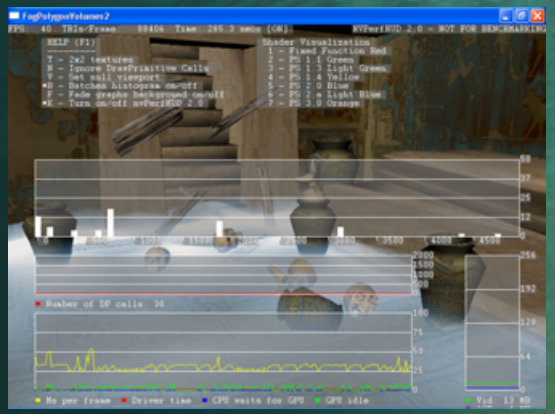
Koji Ashida
NVIDIA
Developer Technology Group

Overview

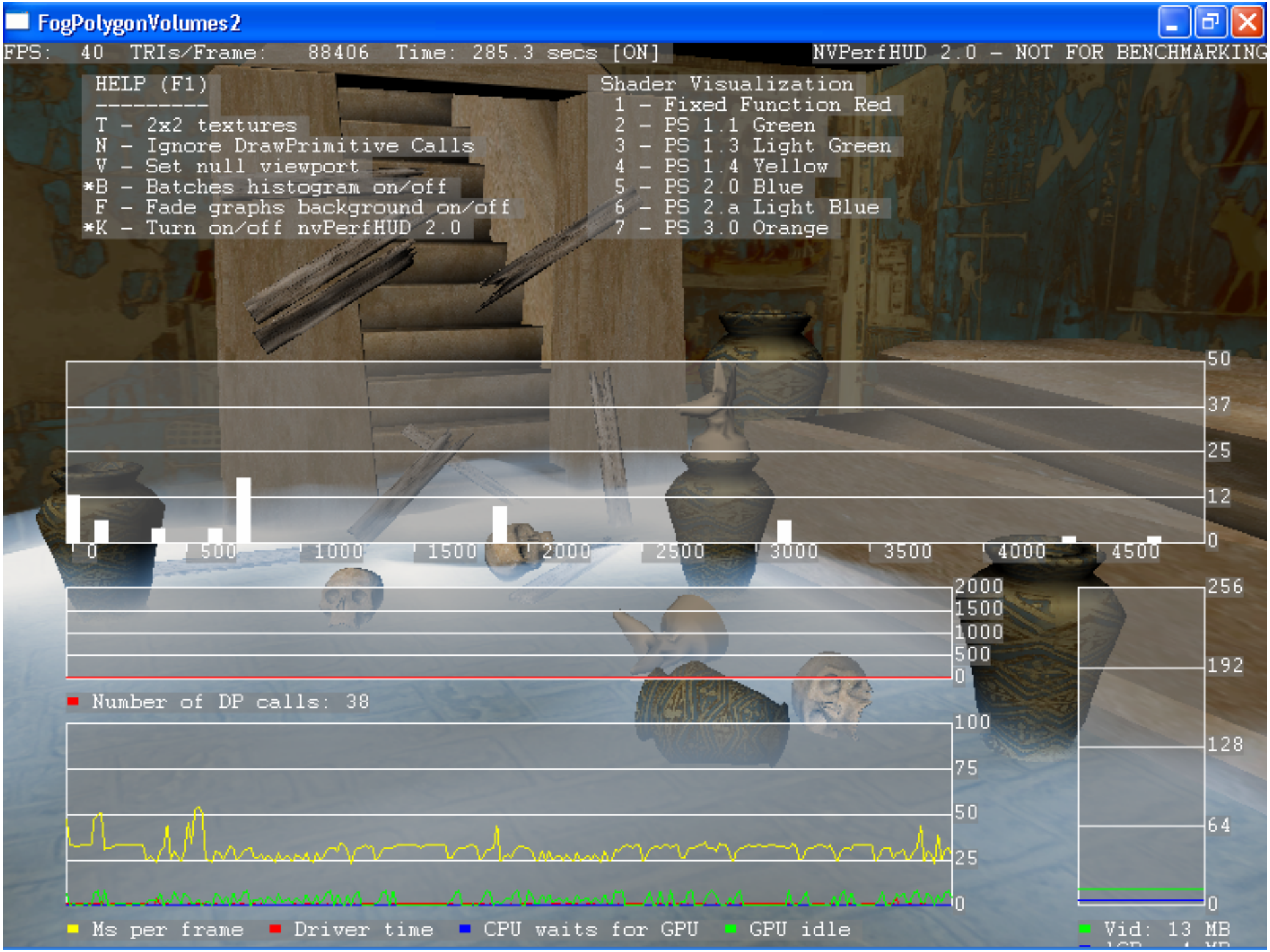
- Tools for the analysis
- Finding pipeline bottlenecks
- Practice identifying the problems

Analysis Tools

NVPerfHUD

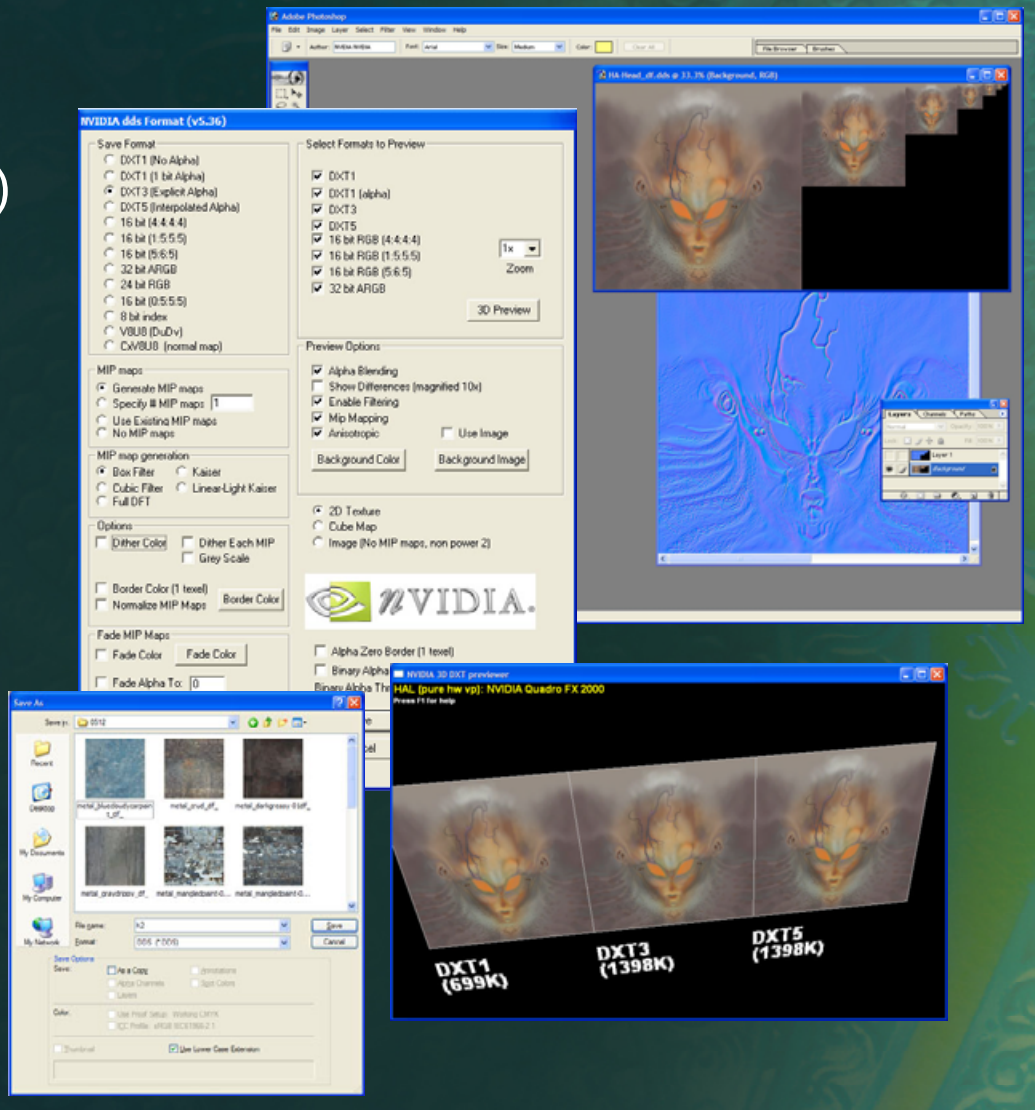


- Graph overlay of various vital statistics
- Metrics reported include:
  - GPU_Idle
  - Driver_Waiting
  - Time_in_Driver
  - Frame_Time
  - AGP / Video memory usage
  - # DIP/DP calls per frame and batch size histogram
- Now external to the driver, supports any Direct3D9 application
- Previously available only to registered developers


FogPolygonVolumes2
FPS: 40 TRIs/Frame: 88406 Time: 285.3 secs [ON]
NVPerfHUD 2.0 - NOT FOR BENCHMARKING
HELP (F1)
T - 2x2 textures
N - Ignore DrawPrimitive Calls
V - Set null viewport
*B - Batches histogram on/off
F - Fade graphs background on/off
*K - Turn on/off nvPerfHUD 2.0
Shader Visualization
1 - Fixed Function Red
2 - PS 1.1 Green
3 - PS 1.3 Light Green
4 - PS 1.4 Yellow
5 - PS 2.0 Blue
6 - PS 2.a Light Blue
7 - PS 3.0 Orange
50
37
25
12
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
2000
1500
1000
500
0
Number of DP calls: 38
100
75
50
25
0
256
192
128
64
0
Ms per frame
Driver time
CPU waits for GPU
GPU idle
Vid: 13 MB

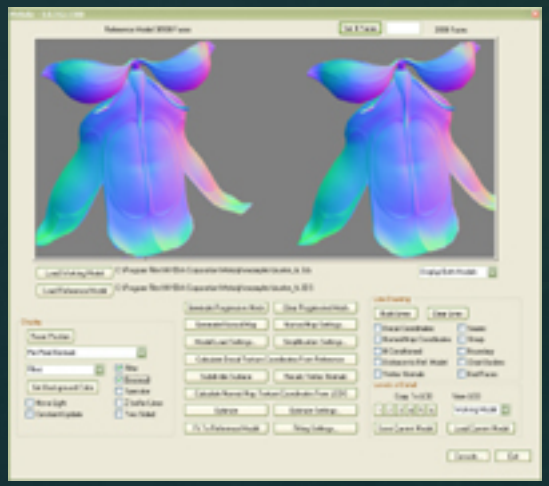
Texture Tools & Plug-ins

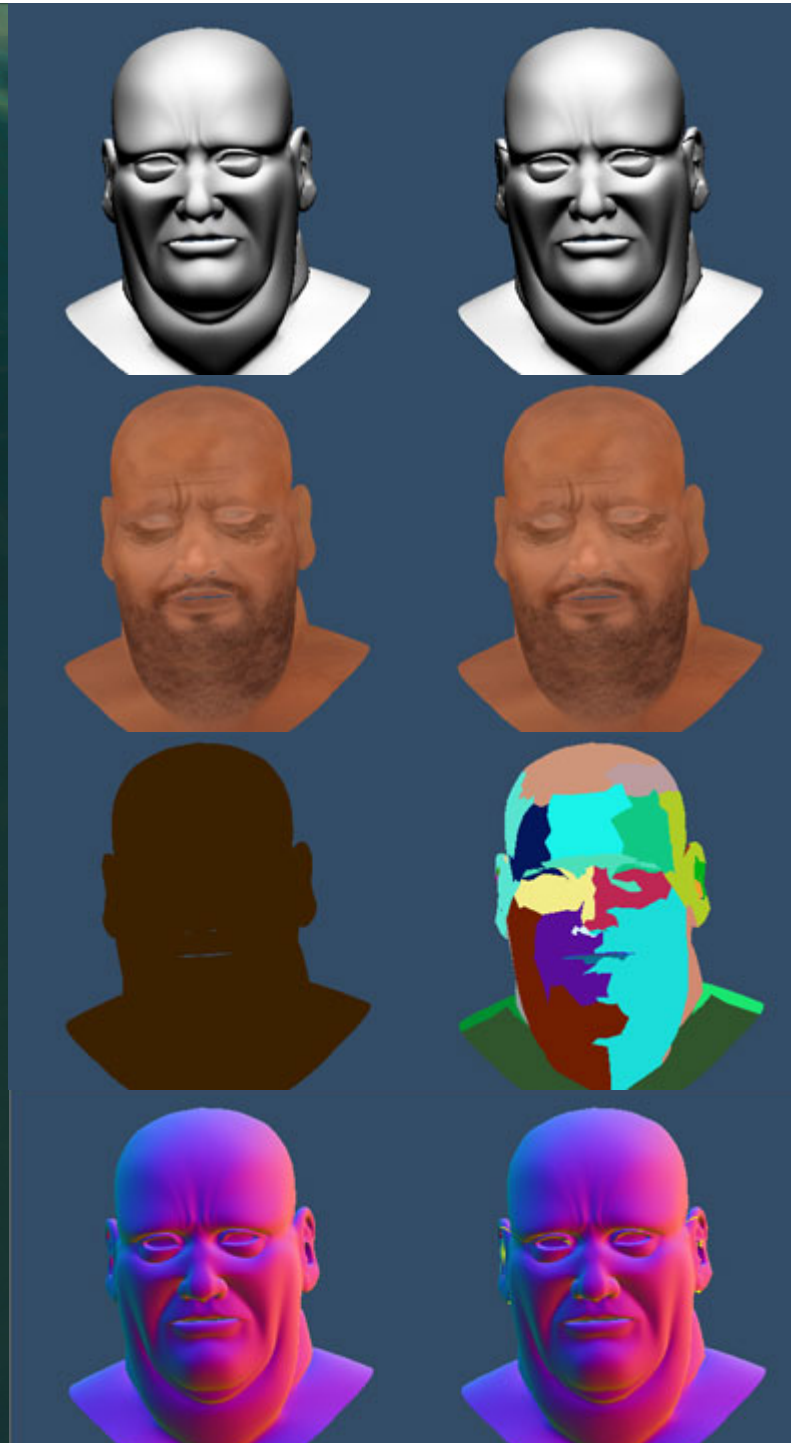
- Photoshop Plug-ins:
  - DXT compression (.dds)
  - Normal Map creation
  - 3D preview and diff
  - MIP map generation
- Command line and .lib
- DDS thumbnail viewer
- Texture Atlas Viewer and Creation Utility

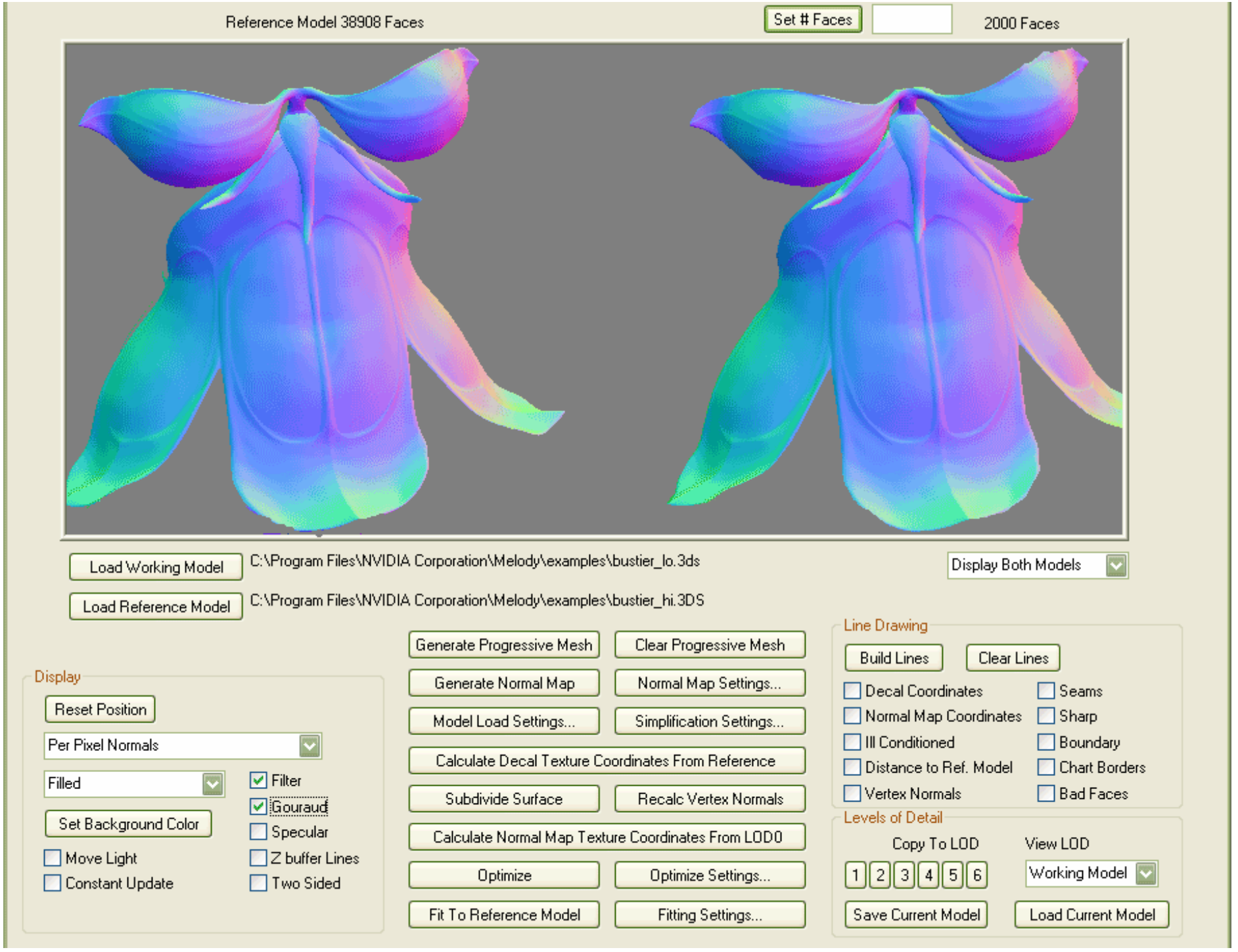


Melody

- Operates on high-resolution meshes (~1.6 million polys)
- Generates LODs using advanced progressive mesh decimation
- Mesh optimization & simplification
- Chart-based UV parameterization
- Raycast normal map generation






Reference Model 38908 Faces
Set # Faces
2000 Faces
Load Working Model
C:\Program Files\NVIDIA Corporation\Melody\examples\bustier_lo.3ds
Display Both Models
Load Reference Model
C:\Program Files\NVIDIA Corporation\Melody\examples\bustier_hi.3DS
Display
Reset Position
Per Pixel Normals
Filled
Set Background Color
Move Light
Constant Update
Filter
Gouraud
Specular
Z buffer Lines
Two Sided
Generate Progressive Mesh
Clear Progressive Mesh
Generate Normal Map
Normal Map Settings...
Model Load Settings...
Simplification Settings...
Calculate Decal Texture Coordinates From Reference
Subdivide Surface
Recalc Vertex Normals
Calculate Normal Map Texture Coordinates From LOD0
Optimize
Optimize Settings...
Fit To Reference Model
Fitting Settings...
Line Drawing
Build Lines
Clear Lines
Decal Coordinates
Normal Map Coordinates
Ill Conditioned
Distance to Ref. Model
Vertex Normals
Seams
Sharp
Boundary
Chart Borders
Bad Faces
Levels of Detail
Copy To LOD
1 2 3 4 5 6
View LOD
Working Model
Save Current Model
Load Current Model

Open EXR Library

- Support for .EXR image format
- HDR image format developed by ILM
  - www.openexr.org
- 16bit floating-point per component
- Library for .NET 2003

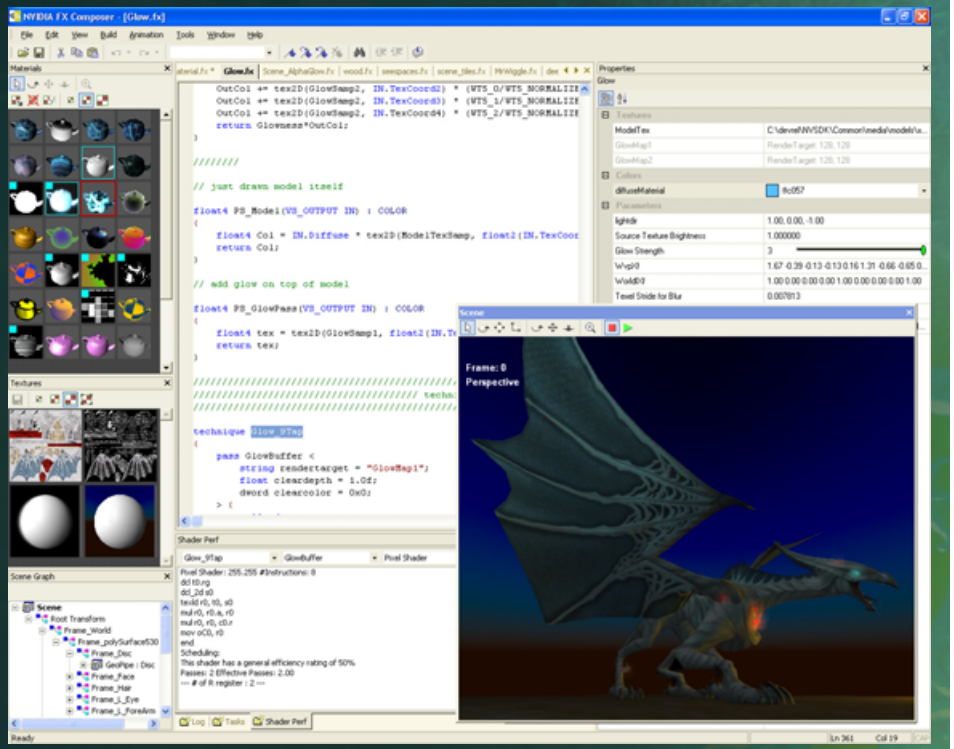
Utilities, libraries and more...

- NVShaderPerf
  - Same technology as FX Composer Shader Perf panel
  - Support for DirectX and OpenGL shaders written in HLSL, !!FP1.0, !!ARBfp1.0, PS1.x and PS2.x
  - ShaderPerf reports for the entire family of NVIDIA GPUs
- NVMeshMender
  - fixes problem geometry
  - preps meshes for per-pixel lighting
- NVTriStrip
  - cache-aware triangle stripping
  - outputs stripts or lists

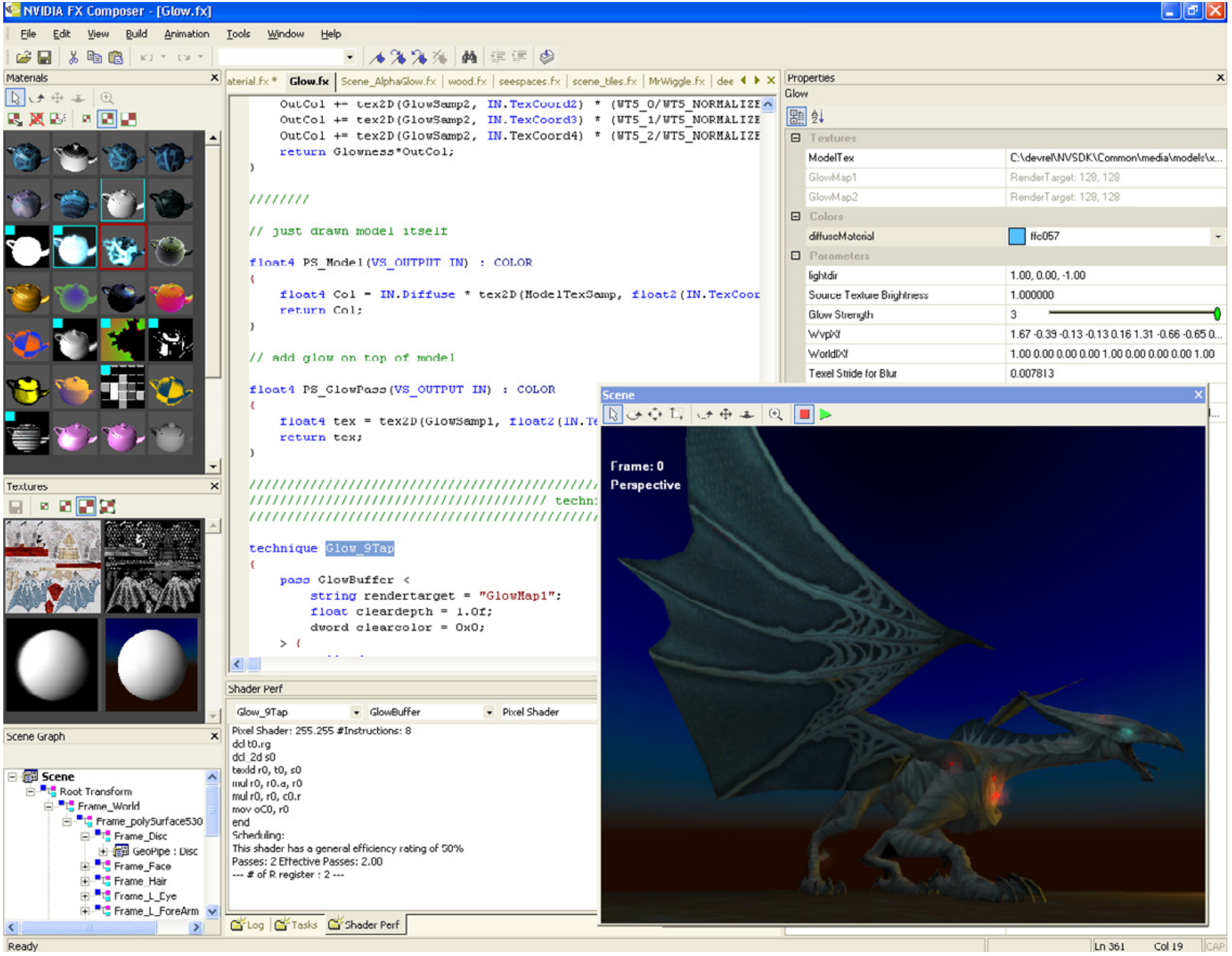
NVIDIA FX Composer

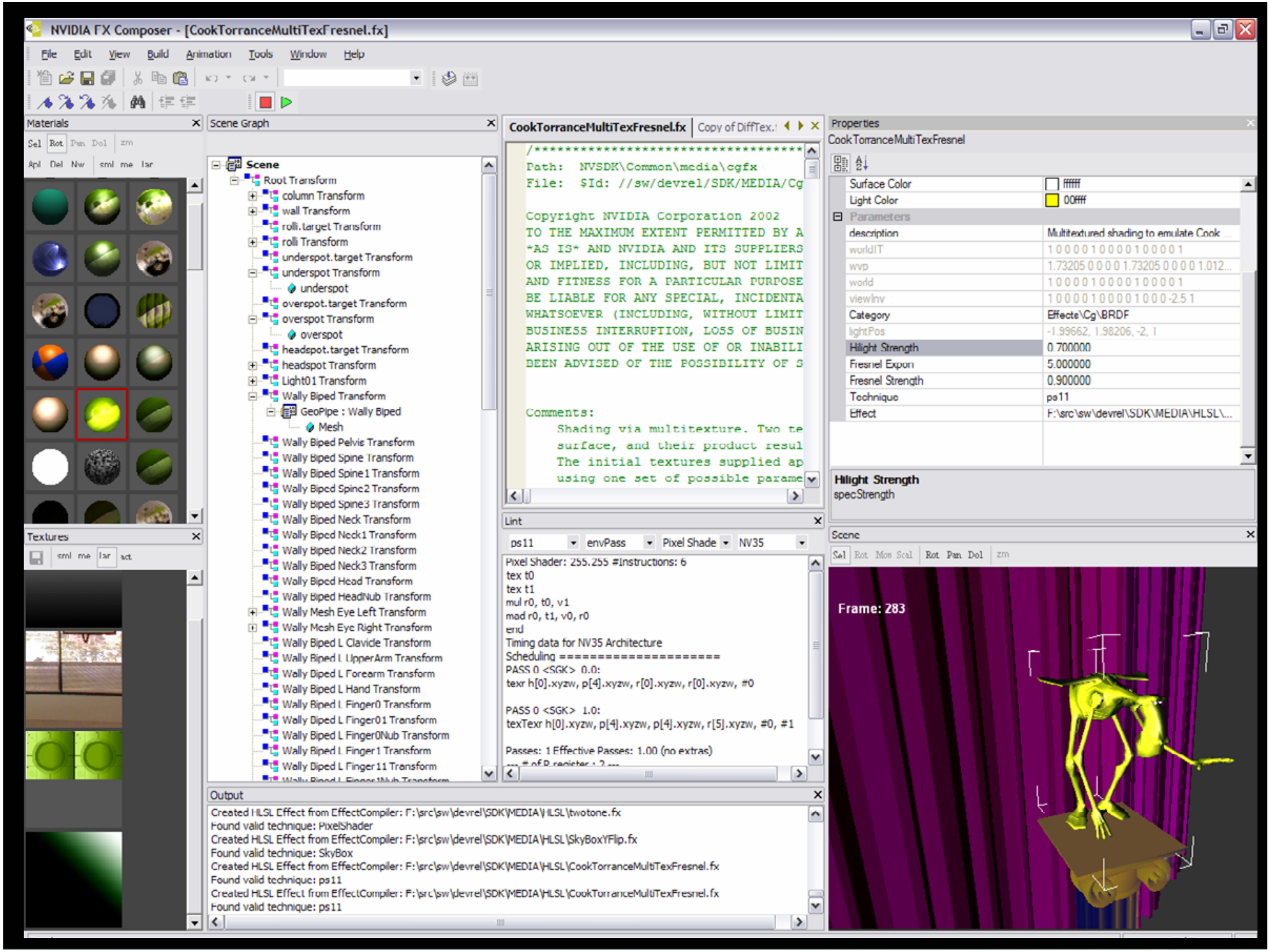
FX Composer empowers developers to create high performance shaders in an integrated development environment with real-time preview & optimization features available only from NVIDIA.

- CREATE your shaders in a high powered developer environment
- DEBUG your shaders with basic shader debugging features
- TUNE your shader performance with advanced analysis and optimization features



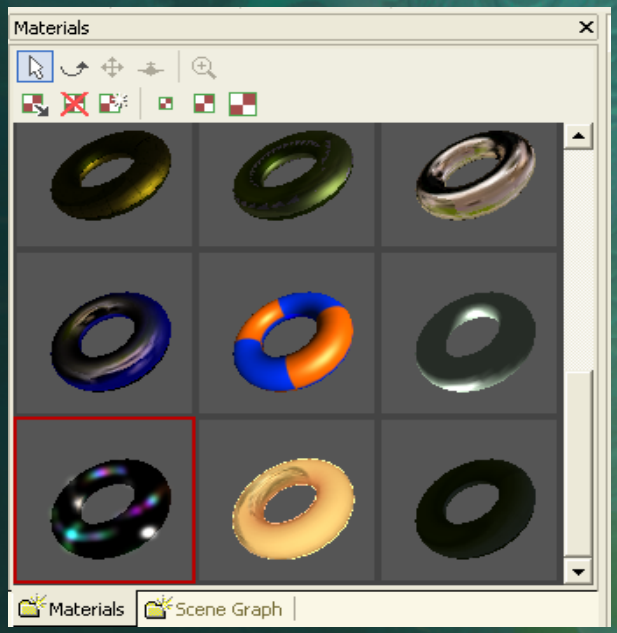
EverQuest® content courtesy Sony Online Entertainment Inc.

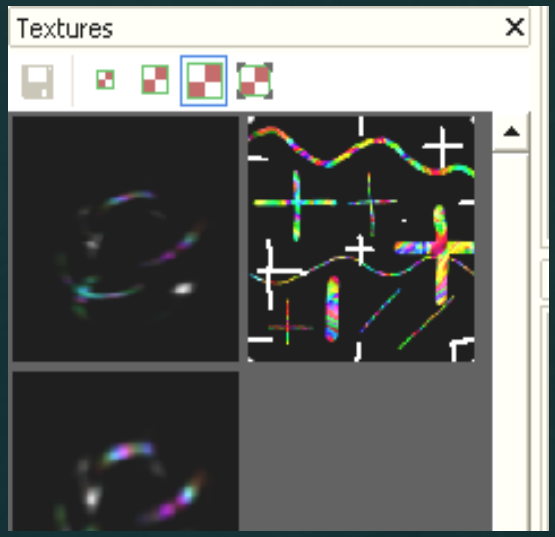

NVIDIA FX Composer - [Glow.fx]
File Edit View Build Animation Tools Window Help
Materials
Textures
Scene Graph
Scene
Root Transform
Frame_World
Frame_polySurface530
Frame_Disc
GeoPipe : Disc
Frame_Face
Frame_Hair
Frame_L_Eye
Frame_L_ForeArm
aterial.fx * Glow.fx Scene_AlphaGlow.fx wood.fx seespaces.fx scene_tiles.fx MrWiggle.fx dee
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;
> {
Shader Perf
Glow_9Tap
GlowBuffer
Pixel Shader
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 ---
Log Tasks Shader Perf
Properties
Glow
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...
WorldXf 1.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 1.00
Texel Stride for Blur 0.007813
Scene
Frame: 0
Perspective
Ready
Ln 361 Col 19
CAP


NVIDIA FX Composer - [CookTorranceMultiTexFresnel.fx]
File Edit View Build Animation Tools Window Help
Materials
Scene Graph
Scene
Root Transform
column Transform
wall Transform
rolli.target Transform
rolli Transform
underspot.target Transform
underspot Transform
underspot
overspot.target Transform
overspot Transform
overspot
headspot.target Transform
headspot Transform
Light01 Transform
Wally Biped Transform
GeoPipe : Wally Biped
Mesh
Wally Biped Pelvis Transform
Wally Biped Spine Transform
Wally Biped Spine1 Transform
Wally Biped Spine2 Transform
Wally Biped Spine3 Transform
Wally Biped Neck Transform
Wally Biped Neck1 Transform
Wally Biped Neck2 Transform
Wally Biped Neck3 Transform
Wally Biped Head Transform
Wally Biped HeadNub Transform
Wally Mesh Eye Left Transform
Wally Mesh Eye Right Transform
Wally Biped L Clavicle Transform
Wally Biped L UpperArm Transform
Wally Biped L Forearm Transform
Wally Biped L Hand Transform
Wally Biped L Finger0 Transform
Wally Biped L Finger01 Transform
Wally Biped L Finger0Nub Transform
Wally Biped L Finger1 Transform
Wally Biped L Finger11 Transform
CookTorranceMultiTexFresnel.fx
Copy of DiffTex.
/**********
Path: NVSDK\Common\media\cgfx
File: $Id: //sw/devrel/SDK/MEDIA/Cg
Copyright NVIDIA Corporation 2002
TO THE MAXIMUM EXTENT PERMITTED BY A
"AS IS" AND NVIDIA AND ITS SUPPLIERS
OR IMPLIED, INCLUDING, BUT NOT LIMIT
AND FITNESS FOR A PARTICULAR PURPOSE
BE LIABLE FOR ANY SPECIAL, INCIDENTA
WHATSOEVER (INCLUDING, WITHOUT LIMIT
BUSINESS INTERRUPTION, LOSS OF BUSIN
ARISING OUT OF THE USE OF OR INABILI
BEEN ADVISED OF THE POSSIBILITY OF S
Comments:
Shading via multitexture. Two te
surface, and their product resul
The initial textures supplied ap
using one set of possible parame
Properties
CookTorranceMultiTexFresnel
Surface Color ffffff
Light Color 00ffff
Parameters
description Multitextured shading to emulate Cook
worldIT 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
wvp 1.73205 0 0 0 0 1.73205 0 0 0 0 1.012...
world 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
viewInv 1 0 0 0 0 1 0 0 0 0 1 0 0 0 -2.5 1
Category Effects\Cg\BRDF
lightPos -1.99662, 1.98206, -2, 1
Hilight Strength 0.700000
Fresnel Expon 5.000000
Fresnel Strength 0.900000
Technique ps11
Effect F:\src\sw\devrel\SDK\MEDIA\HLSL\...
Hilight Strength
specStrength
Lint
ps11 envPass Pixel Shade NV35
Pixel Shader: 255.255 #instructions: 6
tex t0
tex t1
mul r0, t0, v1
mad r0, t1, v0, r0
end
Timing data for NV35 Architecture
Scheduling ==========
PASS 0 <SGK> 0.0:
texr h[0].xyzw, p[4].xyzw, r[0].xyzw, r[0].xyzw, #0
PASS 0 <SGK> 1.0:
texTexr h[0].xyzw, p[4].xyzw, p[4].xyzw, r[5].xyzw, #0, #1
Passes: 1 Effective Passes: 1.00 (no extras)
Textures
Scene
Rot Pan Dol
Frame: 283
Output
Created HLSL Effect from EffectCompiler: F:\src\sw\devrel\SDK\MEDIA\HLSL\twotone.fx
Found valid technique: PixelShader
Created HLSL Effect from EffectCompiler: F:\src\sw\devrel\SDK\MEDIA\HLSL\SkyBoxYFlip.fx
Found valid technique: SkyBox
Created HLSL Effect from EffectCompiler: F:\src\sw\devrel\SDK\MEDIA\HLSL\CookTorranceMultiTexFresnel.fx
Found valid technique: ps11
Created HLSL Effect from EffectCompiler: F:\src\sw\devrel\SDK\MEDIA\HLSL\CookTorranceMultiTexFresnel.fx
Found valid technique: ps11

Materials & Textures

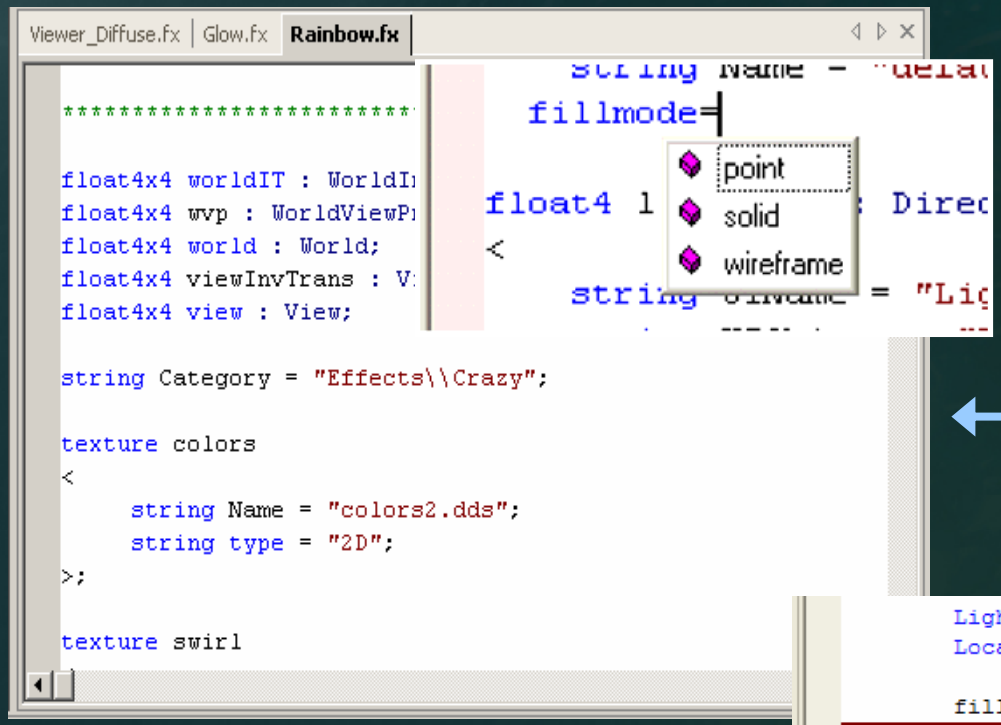
- Preview all the FX files in your scene at the same time
- Apply them to the appropriate parts of your scene in the Scene panel



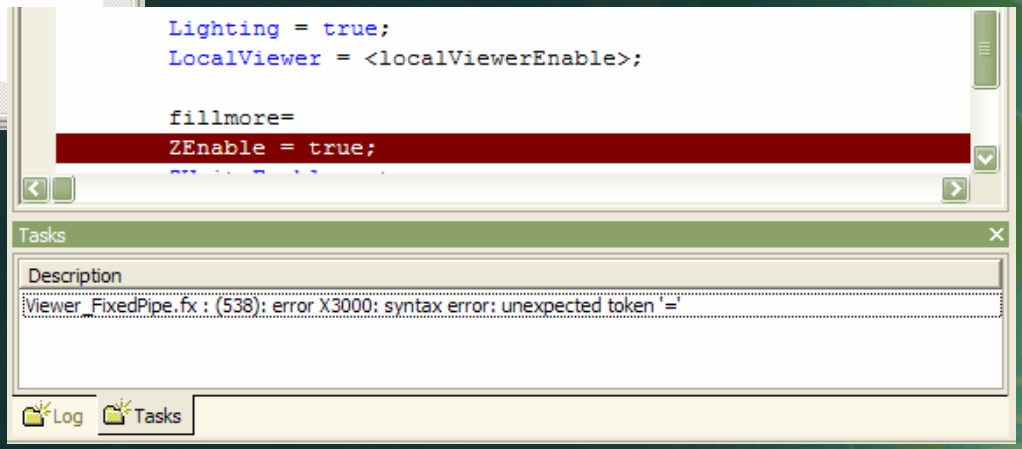


- View source textures
- Preview render targets
- Save any texture to disk!

Editing and Debugging

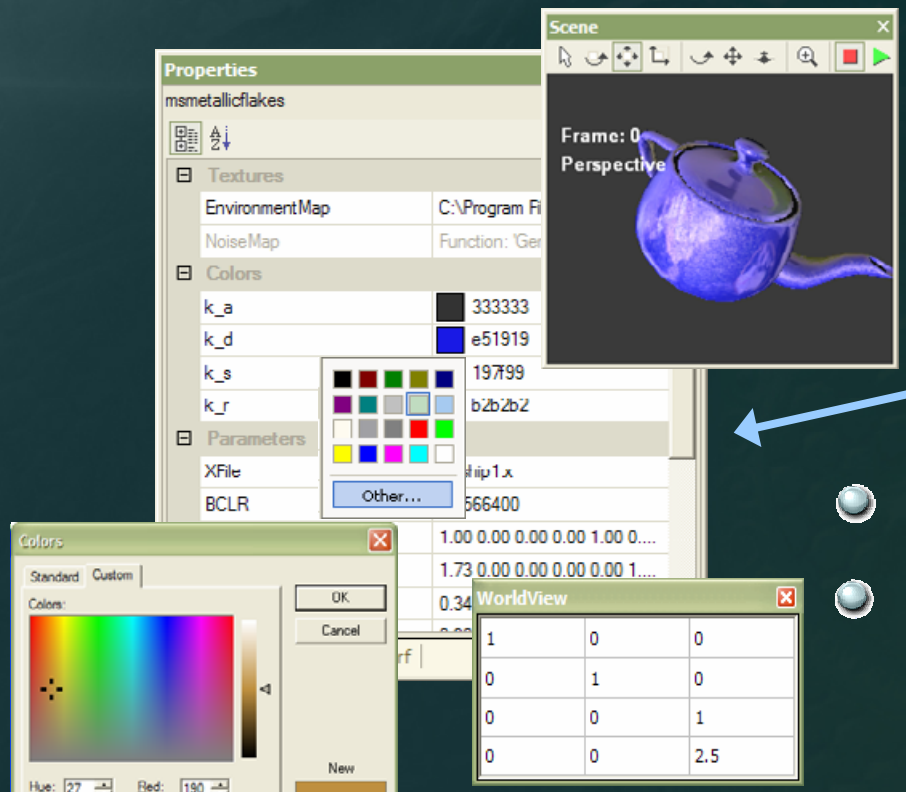


- Edit multiple .FX files
- Intellisense (auto-complete)
- Syntax highlighting

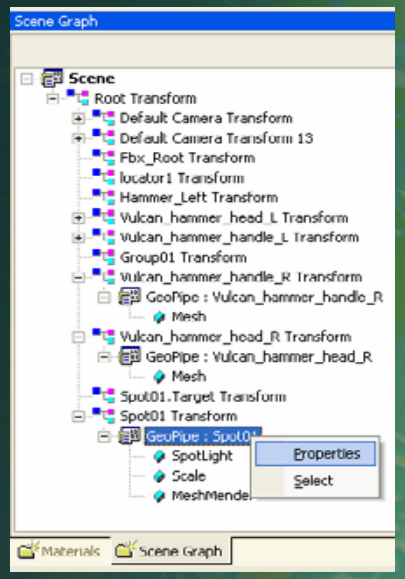


- Jump-to-error helps you find & fix problems quickly

Previewing & Customizing

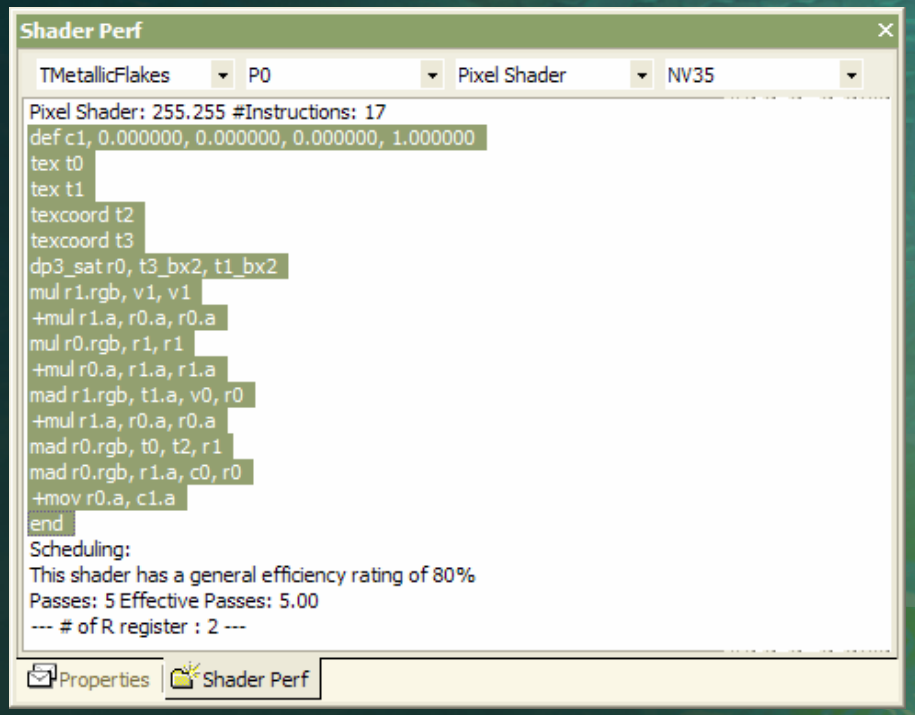


- Conveniently edit your shader parameters
- Automatic parsing of semantics & annotations
- Quickly select custom color values
- Update vector and matrix values using tear-off dialogs
- Navigate to select scene elements and edit properties in the Scene Graph Panel

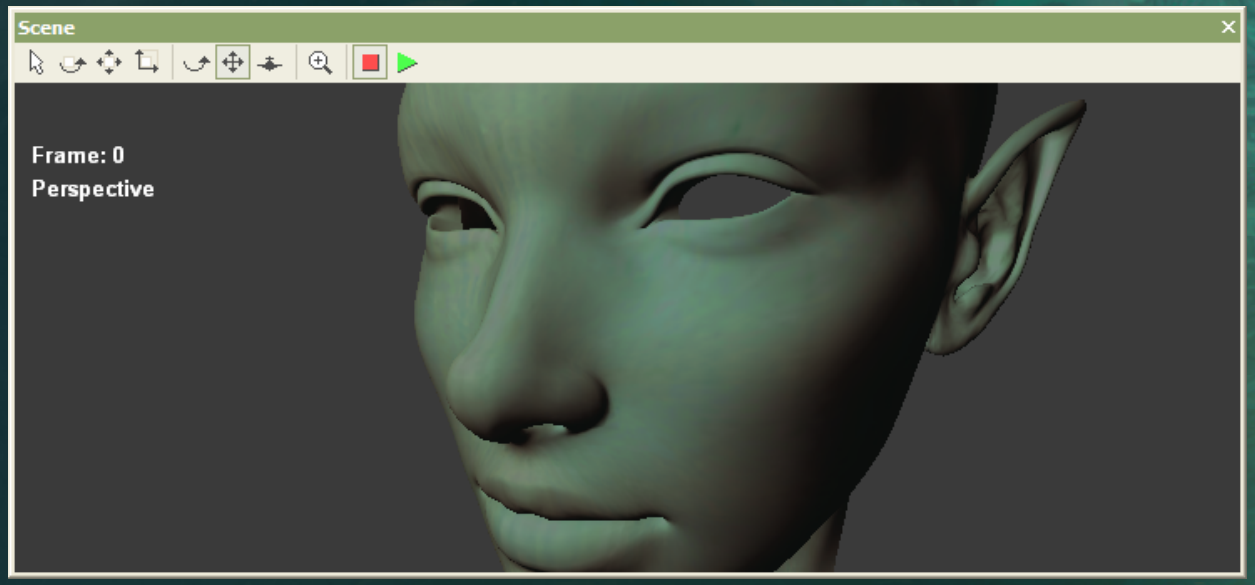


Tune Your Shader Performance

- User the Shader Perf panel
- Select the technique, pass and vertex or pixel shader to analyze
- Simulate pixel shader performance on any recent NVIDIA GPU
- Optimized DirectX Assembly
- NVIDIA performance analysis
  - GPU cycle count
  - Efficiency / utilization rating
  - Number of passes
  - Register usage



NVIDIA FX Composer



Scene Panel

- Preview your 3D scene in real-time
  - Apply materials to scene elements
  - Manipulate the scene elements or the entire scene
  - Use primitives or import .x models and .nvb scenes
  - Set your own key frames or play existing animations
  - Place lights and customize lighting properties
  - Select user-defined cameras or default scene camera

Frequently Asked Questions

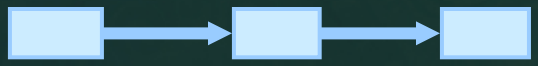
- Support Cg or GLSL?
- Integration with my engine?
- Integration with DCC applications?
- Compared to RenderMonkey, Effect Edit & others?
- Support for other platforms?

The latest version of FX Composer and the full FAQ at
http://developer.nvidia.com/fxcomposer

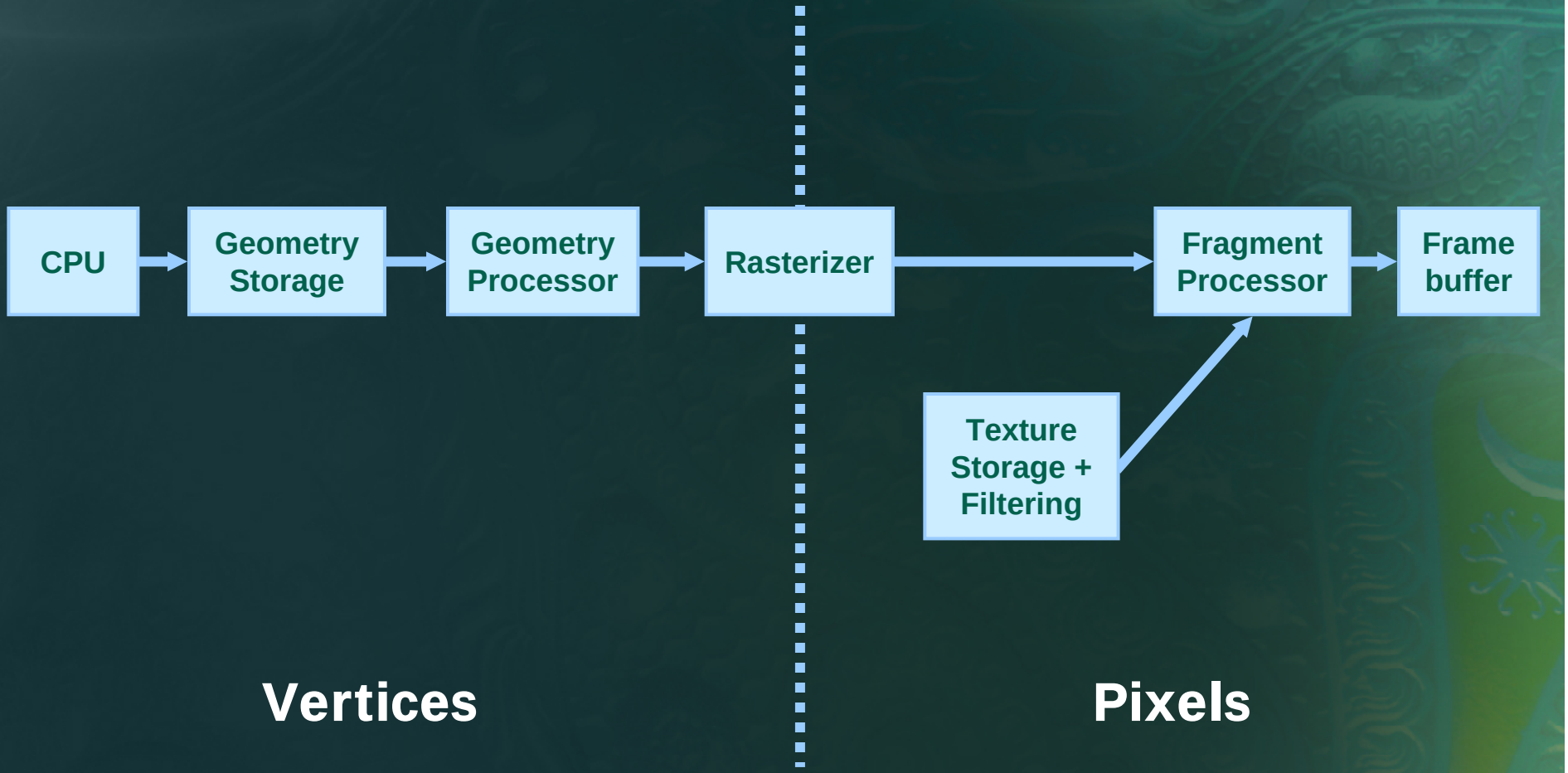
Performance Tuning

Basic Principles

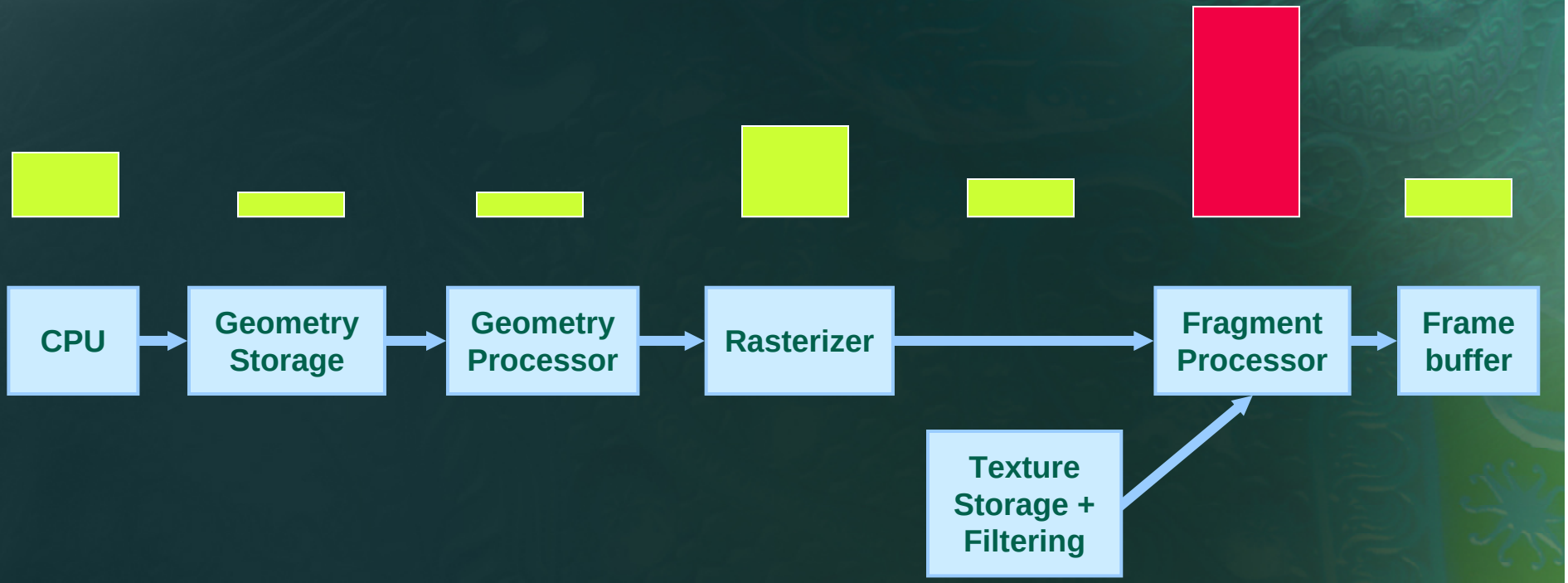
- Pipelined architecture
- Bottleneck identification and elimination
- Balancing the pipeline



Pipelined Architecture

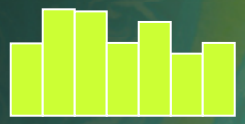


The Terrible Bottleneck



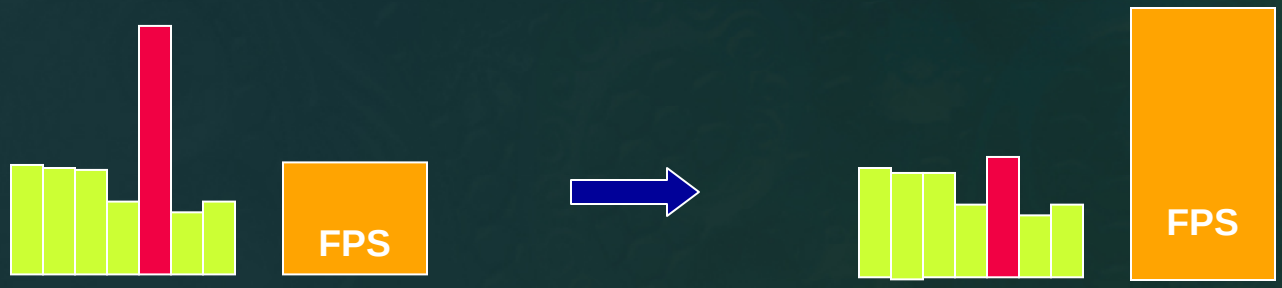
Bottleneck Identification

- Two ways to identify the bottleneck
- Modify the stage itself
- Rule out the other stages

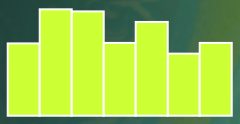


Bottleneck Identification

- Modify the stage itself
  - By decreasing its workload

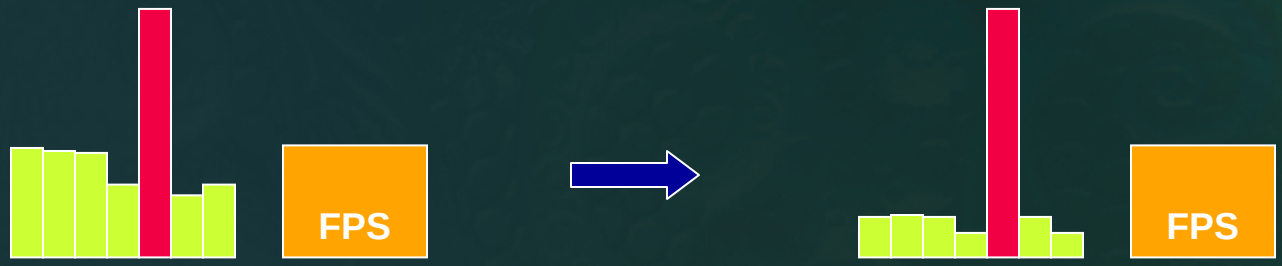


  - If performance improves greatly, then you know this is the bottleneck
  - Careful not to change the workload of other stages!



Bottleneck Identification

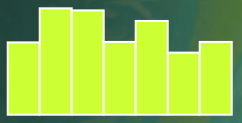
- Rule out the other stages
  - By giving all of them little or no work



  - If performance doesn't change significantly, then you know this is the bottleneck
  - Careful not to change the workload of this stage!

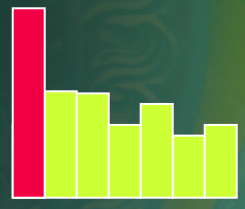
Bottleneck Identification

- Most changes to a stage affect others as well
- Can be hard to pick what test to do
- Let’s go over some tests



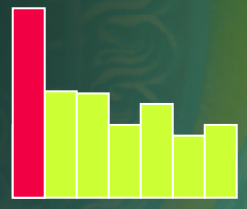
Bottleneck Identification: CPU

- Problem?
- Could be the game
  - Complex physics, AI, game logic
  - Memory management
  - Data structures
- Could be incorrect usage of API
  - Check debug runtime output for errors and warnings
- Could be the display driver
  - Too many batches



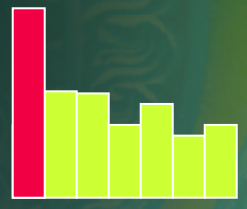
Bottleneck Identification: CPU

- Reduce the CPU workload
- Temporarily turn off
  - Game logic
  - AI
  - Physics
  - Any other thing you know to be expensive on the CPU as long as it doesn't change the rendering workload



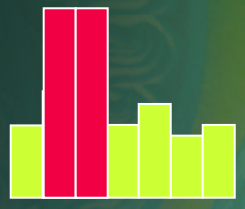
Bottleneck Identification: CPU

- Rule out other stages
- Kill the DrawPrimitive calls
  - Set up everything as you normally would but when the time comes to render something, just do not make the DrawPrimitive* call
  - Problem: you don’t know what the runtime or driver does when a draw primitive call is made
- Use VTUNE or NVPerfHUD



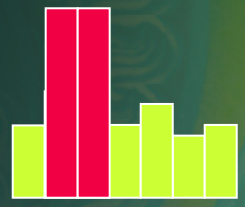
Bottleneck Identification: Vertex

- Problem?
- Transferring the vertices and indices to the card
- Turning the vertices and indices into triangles
- Vertex cache misses
- Using an expensive vertex shader



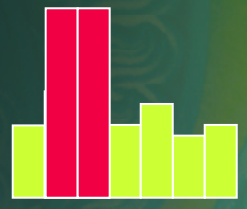
Bottleneck Identification: Vertex

- Reduce vertex overhead
- Use simpler vertex shader
- Send fewer Triangles??
  - Not good
- Decrease AGP Aperture??
  - Maybe not good
  - Use NVPerfHUD to check video memory
  - If it’s full then you might have textures in AGP



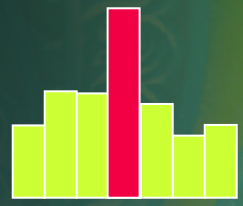
Bottleneck Identification: Vertex

- Rule out other stages
- Render to a smaller backbuffer; this can rule out
  - Texture, Frame buffer, Pixel shader
- Test for a CPU bottleneck
- Can also render to smaller view port instead of smaller backbuffer. Still rules out
  - Texture, Frame buffer, Pixel shader



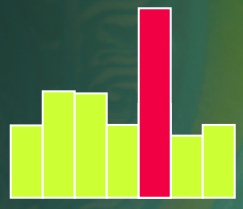
Bottleneck Identification: Raster

- Rarely the bottleneck, spend your time testing other stages first



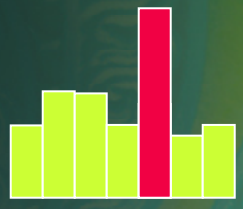
Bottleneck Identification: Texture

- Problem?
- Texture cache misses
- Huge Textures
- Bandwidth
- Texturing out of AGP



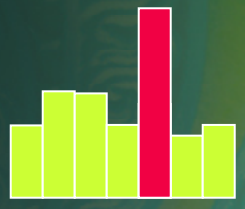
Bottleneck Identification: Texture

- Reduce Texture bandwidth
- Use tiny (2x2) textures
  - Good, but if you are using alpha test with texture alpha, then this could actually make things run slower due to increased fill. It is still a good easy test though
- Use mipmaps if you aren’t already
- Turn off anisotropic filtering if you have it on



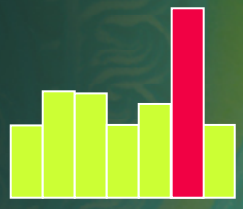
Bottleneck Identification: Texture

- Rule out other stages
- Since texture is so easy to test directly, we recommend relying on that



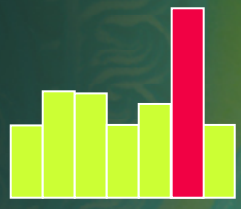
Bottleneck Identification: Fragment

- Problem?
- Expensive pixel shader
- Rendering more fragments than necessary
  - High depth complexity
  - Poor z-cull



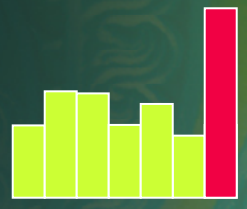
Bottleneck Identification: Fragment

- Modify the stage itself
- Just output a solid color
  - Good: does no work per fragment
  - But also affects texture, so you must then rule out texture
- Use simpler math
  - Good: does less work per fragment
  - But make sure that the math still indexes into the textures the same way or you will change the texture stage as well



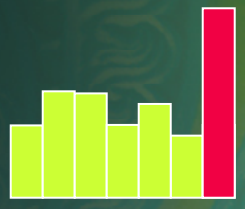
Bottleneck Identification: FB

- Problem?
- Touching the buffer more times than necessary
  - Multiple passes
- Tons of alpha blending
- Using too big a buffer
  - Stencil when you don’t need it
  - A lot of time dynamic reflection cube-maps can get away with r5g6b5 color instead of x8r8g8b8



Bottleneck Identification: FB

- Modify the stage itself
- Use a 16 bit depth buffer instead of a 24 bit one
- Use a 16 bit color buffer instead of a 32 bit one



Practice

Practice: Clean the Machine

- Use the right drivers
- Release build of the game (optimizations on)
- Check debug output for warnings or errors but.....
- Use the release D3D runtime!!!
- No maximum validation
- No driver overridden anisotropic filtering or anti-aliasing
- V-sync off

Practice: Example 1

Practice: Example 1

- Dynamic vertex buffer
  - BAD creation flags

```
HRESULT hr = pd3dDevice->CreateVertexBuffer(
                        6* sizeof( PARTICLE_VERT ),
                        0,      //declares this as static
                        PARTICLE_VERT::FVF,
                        D3DPOOL_DEFAULT,
                        &m_pVB,
                        NULL );
```

Practice: Example 1

- Dynamic vertex buffer
  - GOOD creation flags

```
HRESULT hr = pd3dDevice->CreateVertexBuffer(
                    6* sizeof( PARTICLE_VERT ),
                    D3DUSAGE_DYNAMIC |
                    D3DUSAGE_WRITEONLY,
                    PARTICLE_VERT::FVF,
                    D3DPOOL_DEFAULT,
                    &m_pVB,
                    NULL );
```

Practice: Example 1

- Dynamic Vertex Buffer
  - BAD Lock flags

```
m_pVB->Lock(0, 0,(void**)&quadTris, 0);
```

- No flags at all!?
  - That can’t be good....

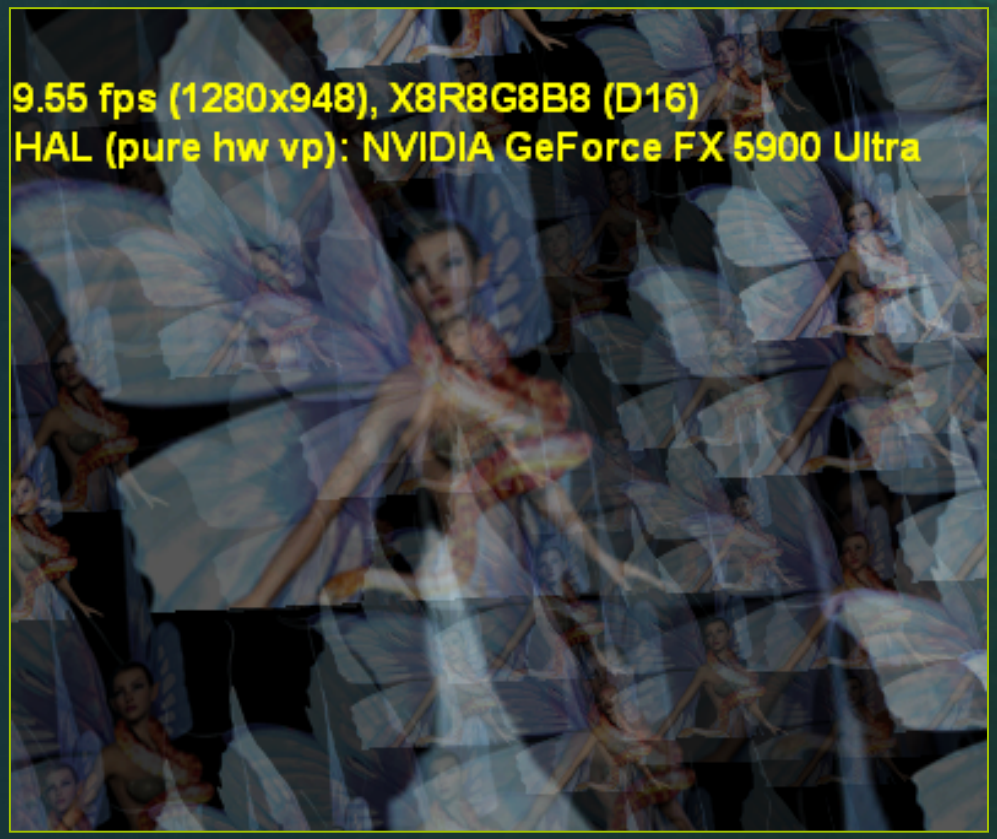
Practice: Example 1

- Dynamic Vertex Buffer
  - GOOD Lock flags

```
m_pVB->Lock(0, 0,(void**)&quadTris,
D3DLOCK_NOSYSLOCK | D3DLOCK_DISCARD);
```

- Use D3DLOCK_DISCARD the first time you lock a vertex buffer each frame
  - And again when that buffer is full
  - Otherwise just use NOSYSLOCK

Practice: Example 2



Practice: Example 2

- Texture bandwidth overkill
- Use mipmaps
- Use dxt1 if possible
  - Some cards can store compressed data in cache
- Use smaller textures when they are fine
  - Does the grass blade really need a 1024x1024 texture? --- Maybe

Practice: Example 3

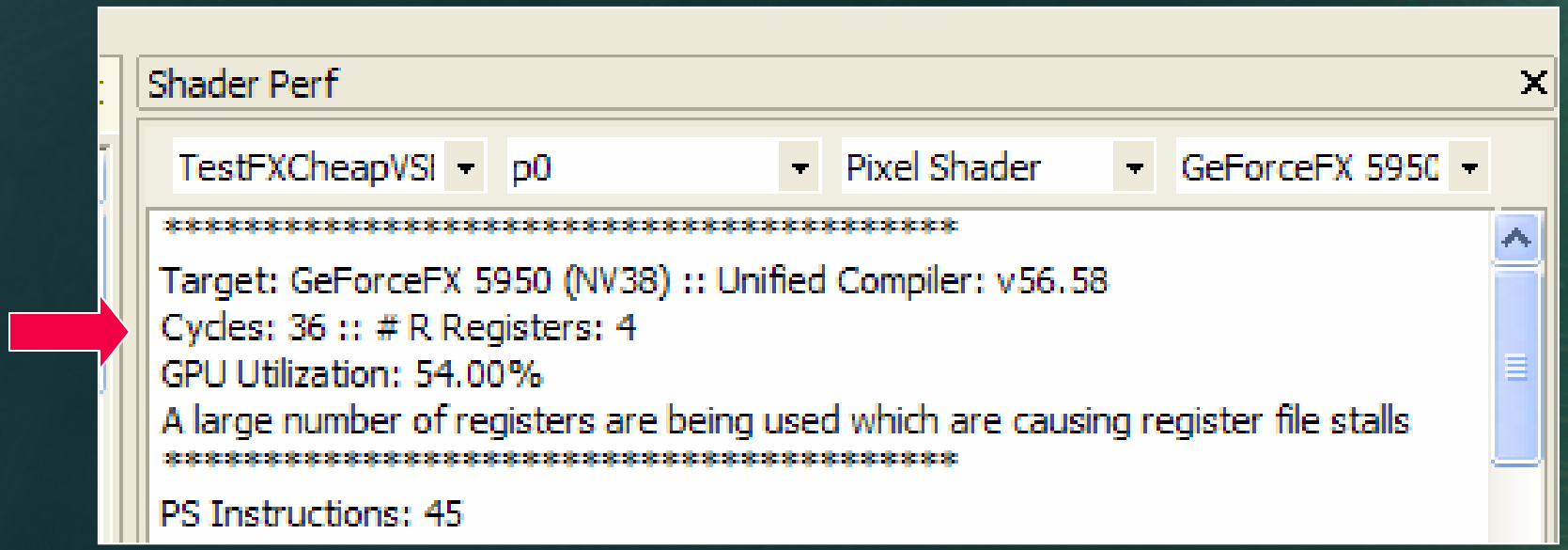
Practice: Example 3

- Expensive pixel shader
- Can have huge performance effect
- Only 3 verts, but maybe a million pixels
  - That’s only 1024x1024

Look at all the pixels!!

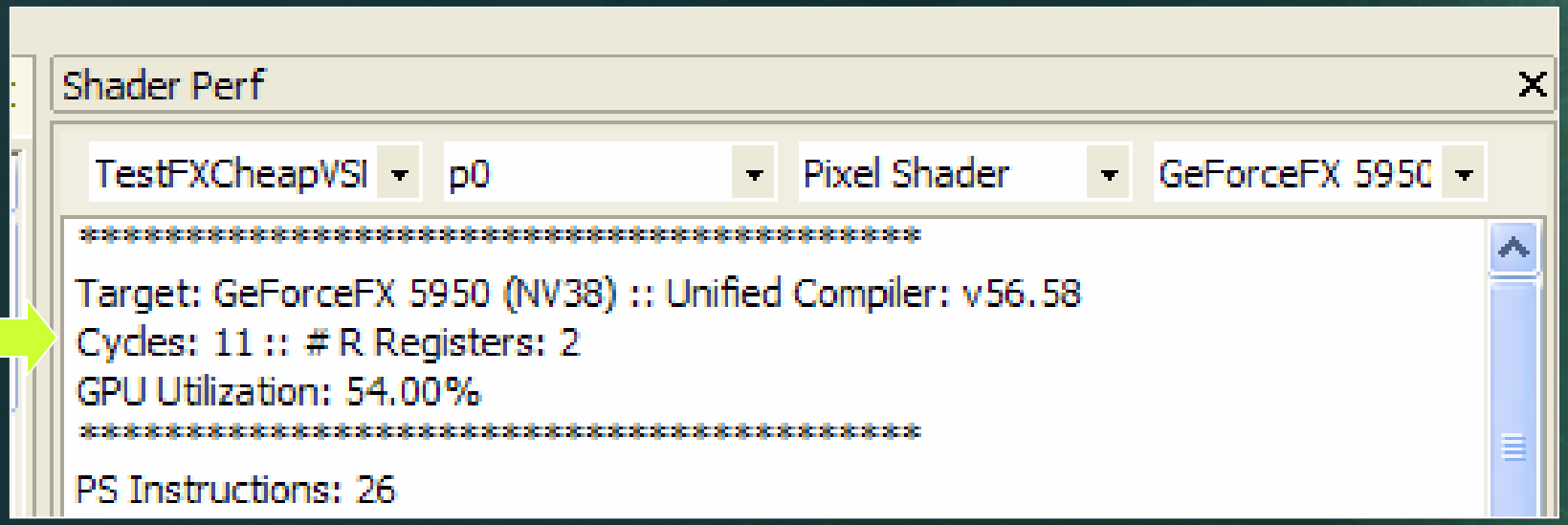
Practice: Example 3

- 36 cycles BAD



Practice: Example 3

- 11 cycles GOOD



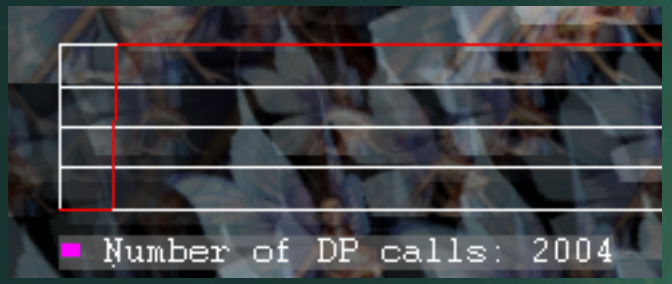
Practice: Example 3

- What changed?
- Moved math that was constant across the triangle into the vertex shader
- Used ‘half’ instead of ‘float’
- Got rid of normalize where it wasn’t necessary
  - See Normalization Heuristics
  - http://developer.nvidia.com

Practice: Example 4

Practice: Example 4

- Too many batches
- Was sending every quad as it’s own batch
- Instead, group quads into one big VB then send that with one call

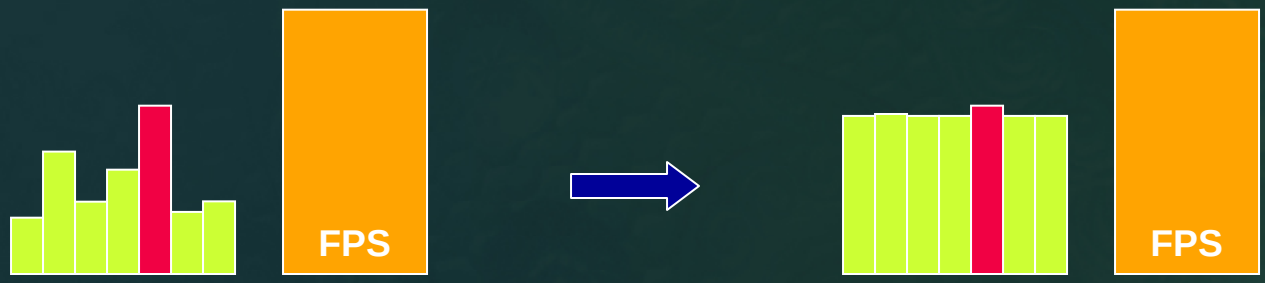


Practice: Example 4

- What if they use different textures?
- Use texture atlases
- Put the two textures into a single texture and use a vertex and pixel shader to offset the texture coordinates

Balancing the Pipeline

- Balance the pipeline by making more use of un-bottlenecked stages
- Careful not to make too much use of them



Summary

- NVIDIA offers many tools for performance analysis
- Pipeline architecture is ruled by bottlenecks
- Identify bottlenecks with quick tests
- Use NVPerfHUD to analyze your pipeline
- Use FX Composer to help tune your shaders

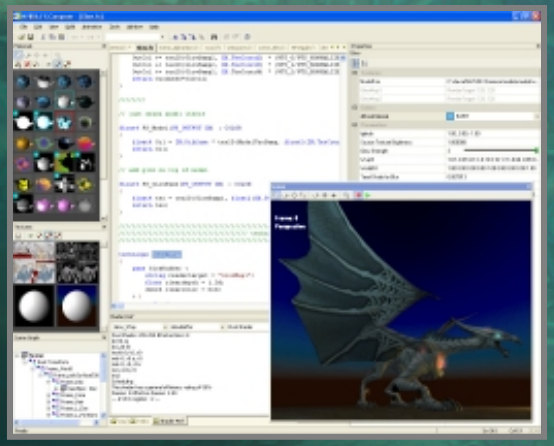
For More Info...

- Download presentations, SDK, Tools, etc: http://developer.nvidia.com
- Questions, requests, and comments for Tools and SDK: sdkfeedback@nvidia.com
- About this presentation: kashida@nvidia.com

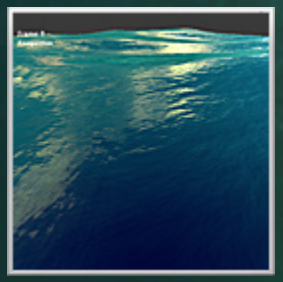
developer.nvidia.com

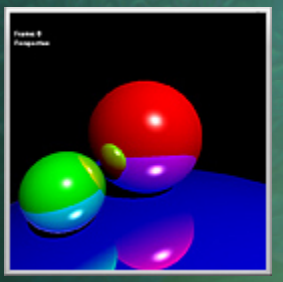
The Source for GPU Programming

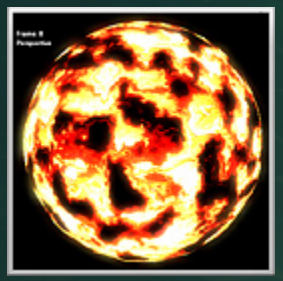
- Latest documentation
- SDKs
- Cutting-edge tools
  - Performance analysis tools
  - Content creation tools
- Hundreds of effects
- Video presentations and tutorials
- Libraries and utilities
- News and newsletter archives

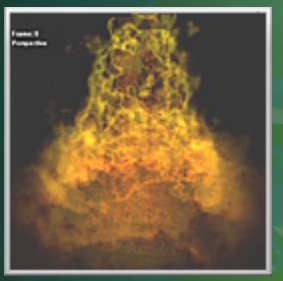


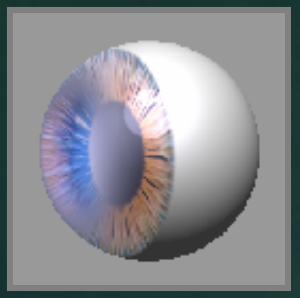
EverQuest® content courtesy Sony Online Entertainment Inc.









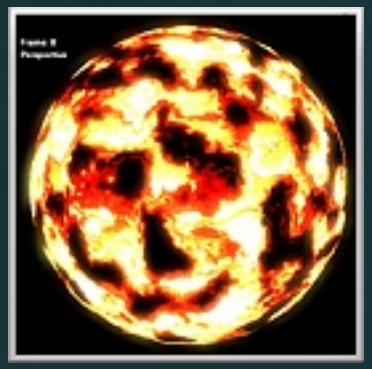


NVIDIA SDK

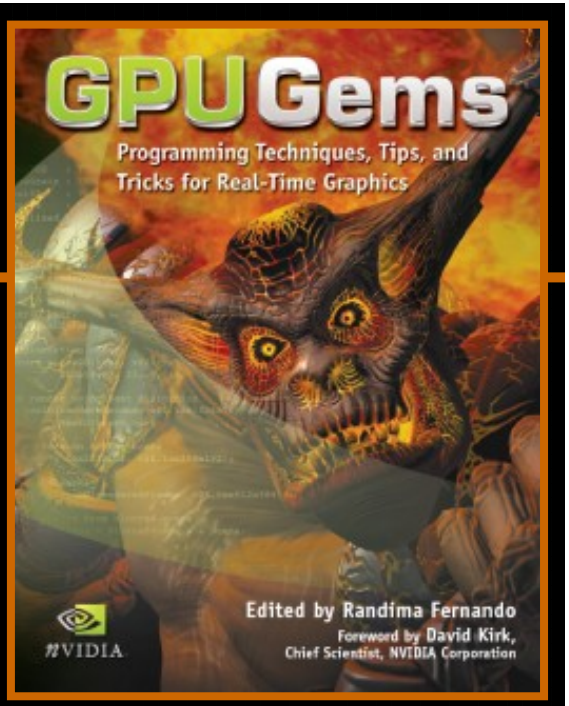
The source for real-time developers

Hundreds of code samples and effects that help you take advantage of the latest in graphics technology.

- **Tons of updated and all-new DirectX and OpenGL code samples with full source code and helpful whitepapers:**
  - **Geometry Instancing, Rainbow Fogbow, Blood Shader, Perspective Shadow Maps, Texture Atlas Utility, ...**
- **Hundreds of effects, complete with custom geometry, animation and more:**
  - **Skin, Plastics, Flame/Fire, Glow, Gooch, Image Filters, HLSL Debugging Techniques, Texture BRDFs, Texture Displacements, Tonemapping, and even a simple Ray Tracer!**



GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics



- Practical real-time graphics techniques from experts at leading corporations and universities
- Great value:
  - Contributions from industry experts
  - Full color (300+ diagrams and screenshots)
  - Hard cover
  - 816 pages

For more, visit:
http://developer.nvidia.com/GPUGems

"*GPU Gems* is a cool toolbox of advanced graphics techniques. Novice programmers and graphics gurus alike will find the gems practical, intriguing, and useful."

Tim Sweeney

Lead programmer of *Unreal* at Epic Games

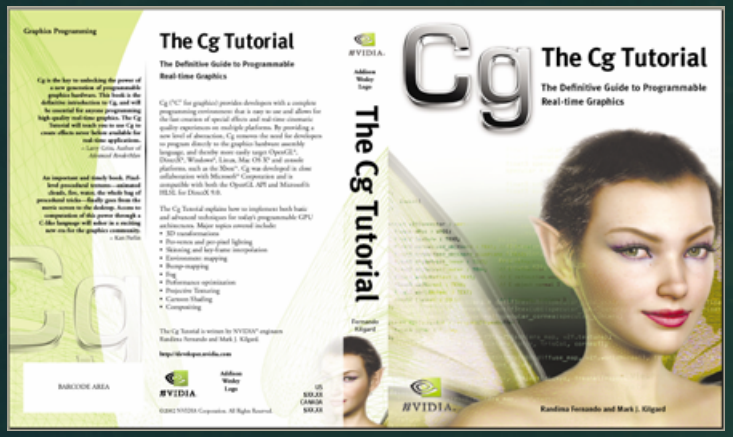
"This collection of articles is particularly impressive for its depth and breadth. The book includes product-oriented case studies, previously unpublished state-of-the-art research, comprehensive tutorials, and extensive code samples and demos throughout."

Eric Haines

Author of *Real-Time Rendering*

The Cg Toolkit

- NVIDIA Cg Compiler
  - Vertex (DirectX 9, OpenGL 1.4)
  - Pixel (DirectX 9)
- Cg Standard Library
- Cg Runtime Libraries for DirectX and OpenGL
- NVIDIA Cg Browser
- Cg Language Specification
- Cg User's Manual
- Cg Shaders
  (assorted pre-written programs)



- The Cg Tutorial
  (developer.nvidia.com/CgTutorial)