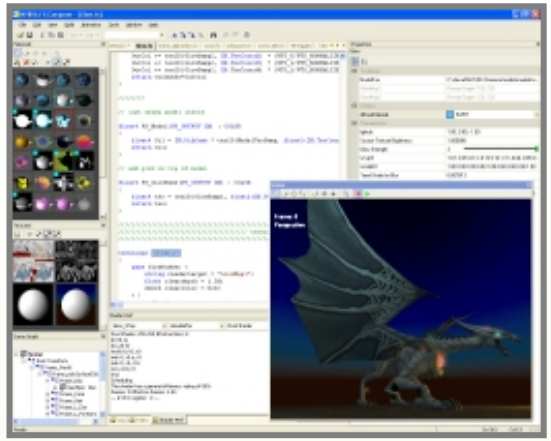
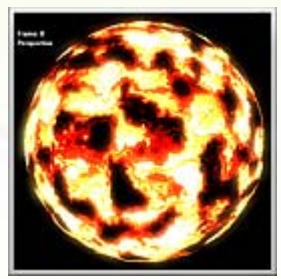
developer.nvidia.com

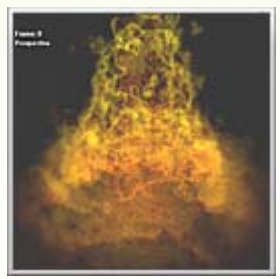
The Source for GPU Programming

- 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

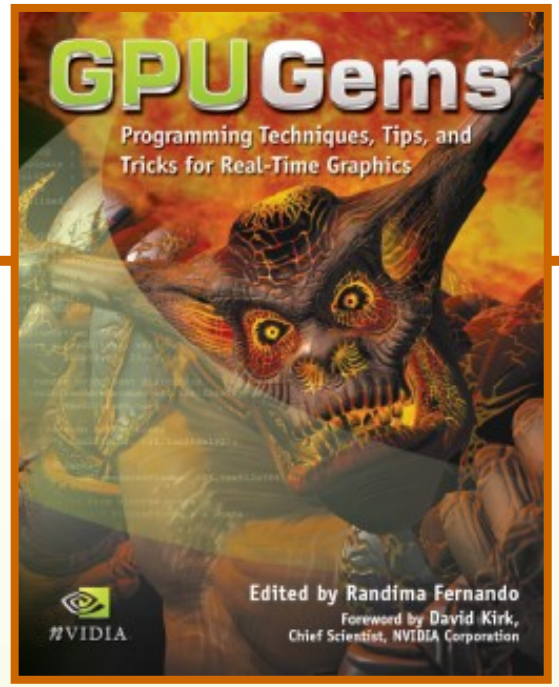


EverQuest® content courtesy Sony Online Entertainment Inc.





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:
  - Full color (300+ diagrams and screenshots)
  - Hard cover
  - 816 pages
  - CD-ROM with demos and sample code

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

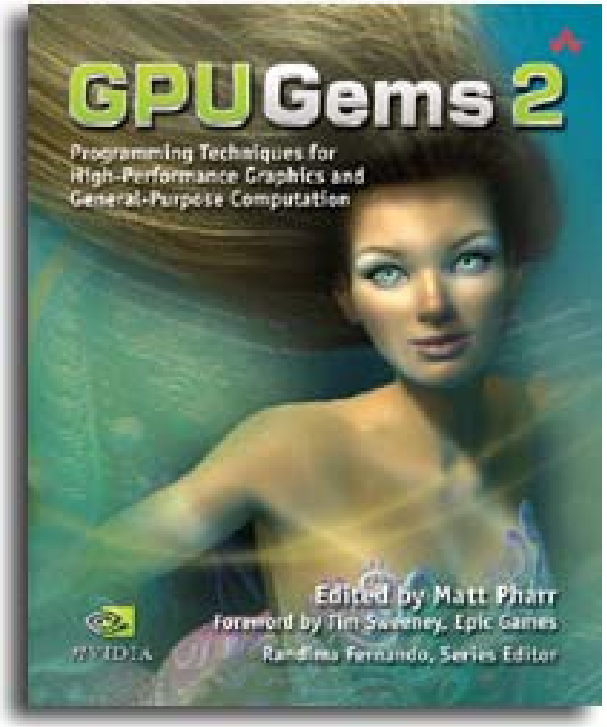
"This collection of articles is particularly impressive for its depth and breadth. The book includes product-oriented case studies, previously unpublished state-of-the-art research, comprehensive tutorials, and extensive code samples and demos throughout."

Eric Haines

Author of *Real-Time Rendering*

GPU Gems 2

Programming Techniques for High-Performance Graphics and General-Purpose Computation



- **880 full-color pages, 330 figures, hard cover**
- **$59.99**
- **Experts from universities and industry**

"The topics covered in *GPU Gems 2* are critical to the next generation of game engines."

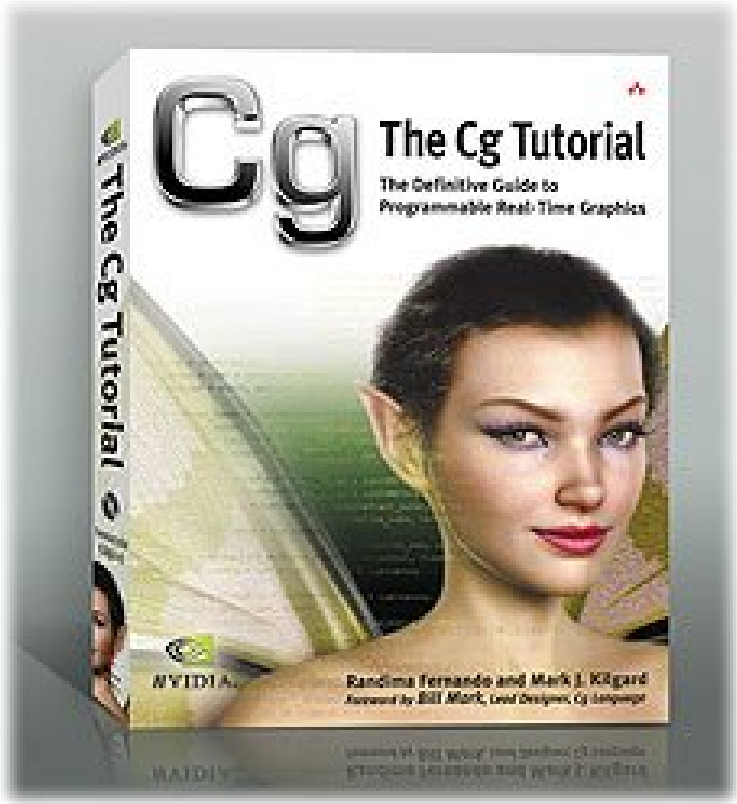
— Gary McTaggart, Software Engineer at Valve, Creators of Half-Life and Counter-Strike

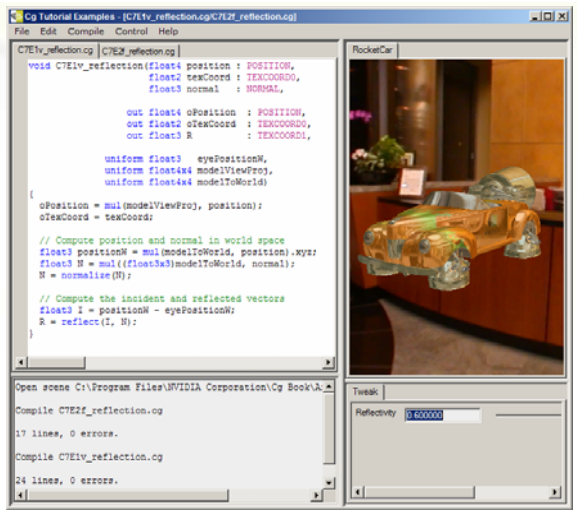
"*GPU Gems 2* isn't meant to simply adorn your bookshelf—it's required reading for anyone trying to keep pace with the rapid evolution of programmable graphics. If you're serious about graphics, this book will take you to the edge of what the GPU can do."

—Rémi Arnaud, Graphics Architect at Sony Computer Entertainment

The Cg Tutorial

- Discusses **graphics concepts** thoroughly
- Provides **complete examples**
- Provides a **complete hands-on framework** to try and modify the examples, out-of-the-box
- Includes **end-of-chapter exercises** and **further reading**






NVIDIA Developer Toolkit
March 2005

Why Do We Do This?

- Investing in Developers Worldwide
  - Powerful tools for building games
    - Performance Analysis
    - Content Creation
    - Software Development
  - Practical SDK with technical documentation
  - Web Site and Newsletter developer.nvidia.com
- Registered Developer Program
  - Pre-Release Drivers
  - Early Access to Developer Tools
  - Online Support Forums & Bug Submission

Sign up now at developer.nvidia.com

NVIDIA SDK

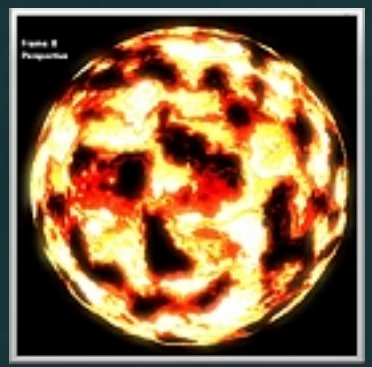
The Source for GPU Programming

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

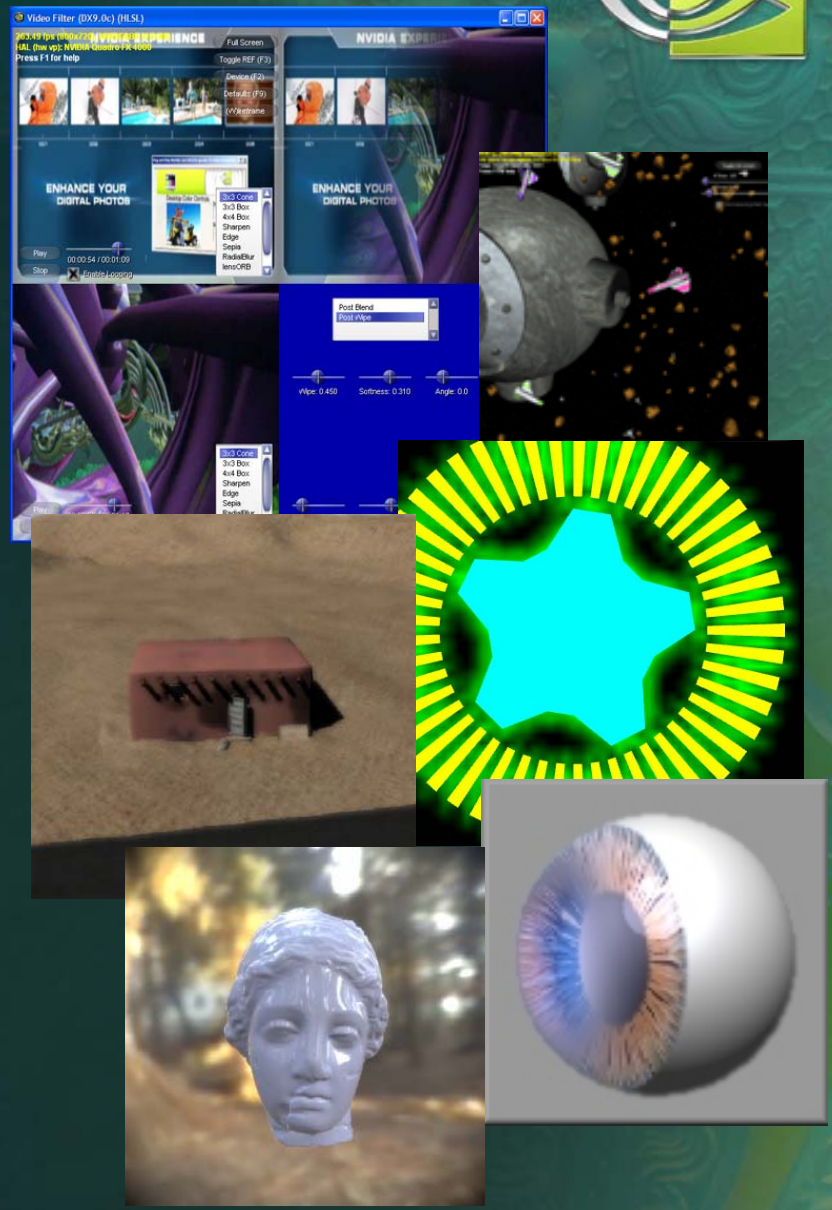
gamedeveloper FRONTLINE AWARD '04

- **Tons of updated and all-new DirectX and OpenGL code samples with full source code and helpful whitepapers:**
  - **GPU Cloth, Geometry Instancing, Rainbow Fogbow, 2xFP16 HRD, 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, HDR Tonemapping, and even a simple Ray Tracer!**



NVIDIA SDK

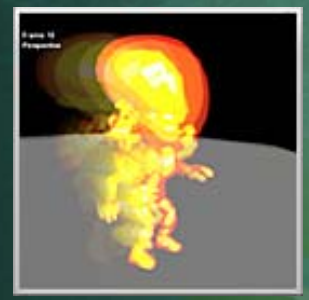


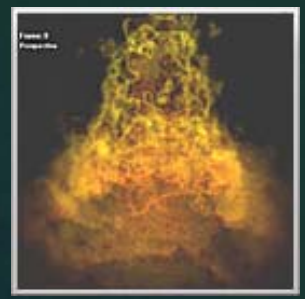
FX Composer

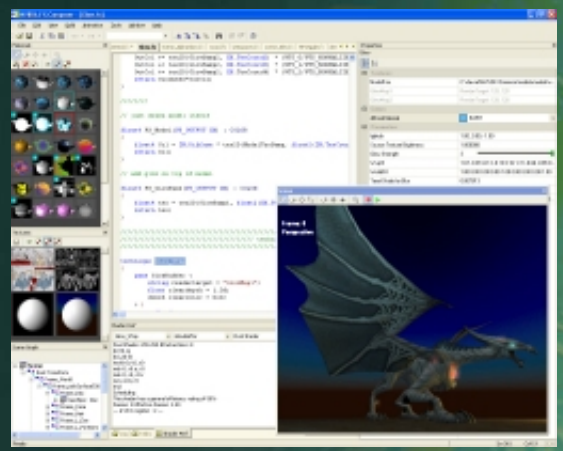
gamedeveloper FRONTLINE FINALIST '04

- CREATE your shaders in a high powered IDE
  - Native support for HLSL .FX development
  - Render-to-texture effects
  - Save out pre-rendered (“baked”) textures
- DEBUG your shaders with visual shader debugging
  - Unique real-time preview of intermediate targets
  - Import your own geometry, complete with animation
- TUNE your shader performance with advanced analysis
  - Vertex & pixel shader performance metrics
  - GPU-specific scheduling & disassembly

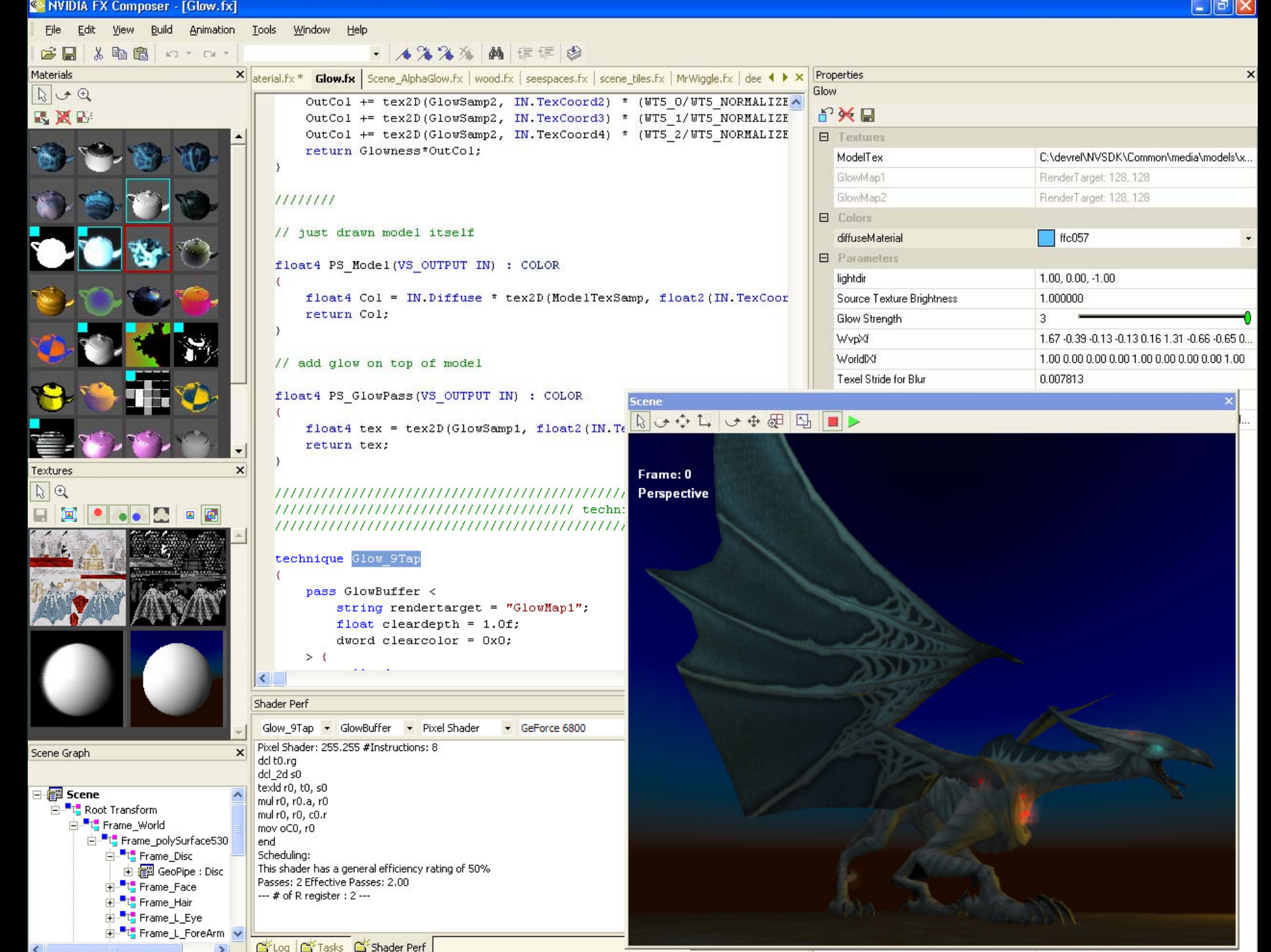
Full plug-in SDK and scripting support for automation







EverQuest® content courtesy
Sony Online Entertainment Inc.

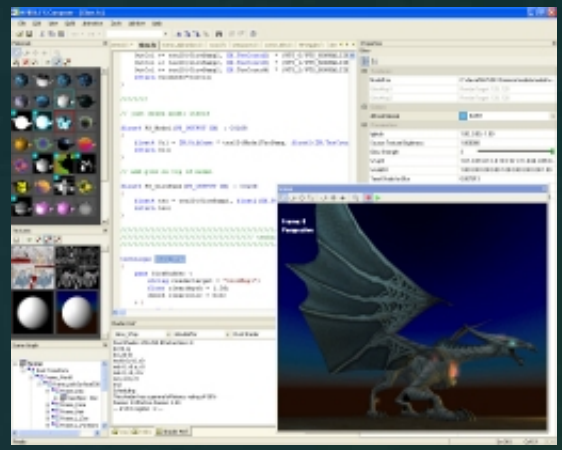

NVIDIA FX Composer - [Glow.fx]
File Edit View Build Animation Tools Window Help
Materials
Textures
Scene Graph
Scene
Root Transform
Frame_World
Frame_polySurface530
Frame_Disc
GeoPipe : Disc
Frame_Face
Frame_Hair
Frame_L_Eye
Frame_L_ForeArm
aterial.fx * Glow.fx Scene_AlphaGlow.fx wood.fx seespaces.fx 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 GeForce 6800
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

FX Composer In Your Pipeline

HLSL Shaders

Textures

Scene Data



Optimized Shaders

Property Sets

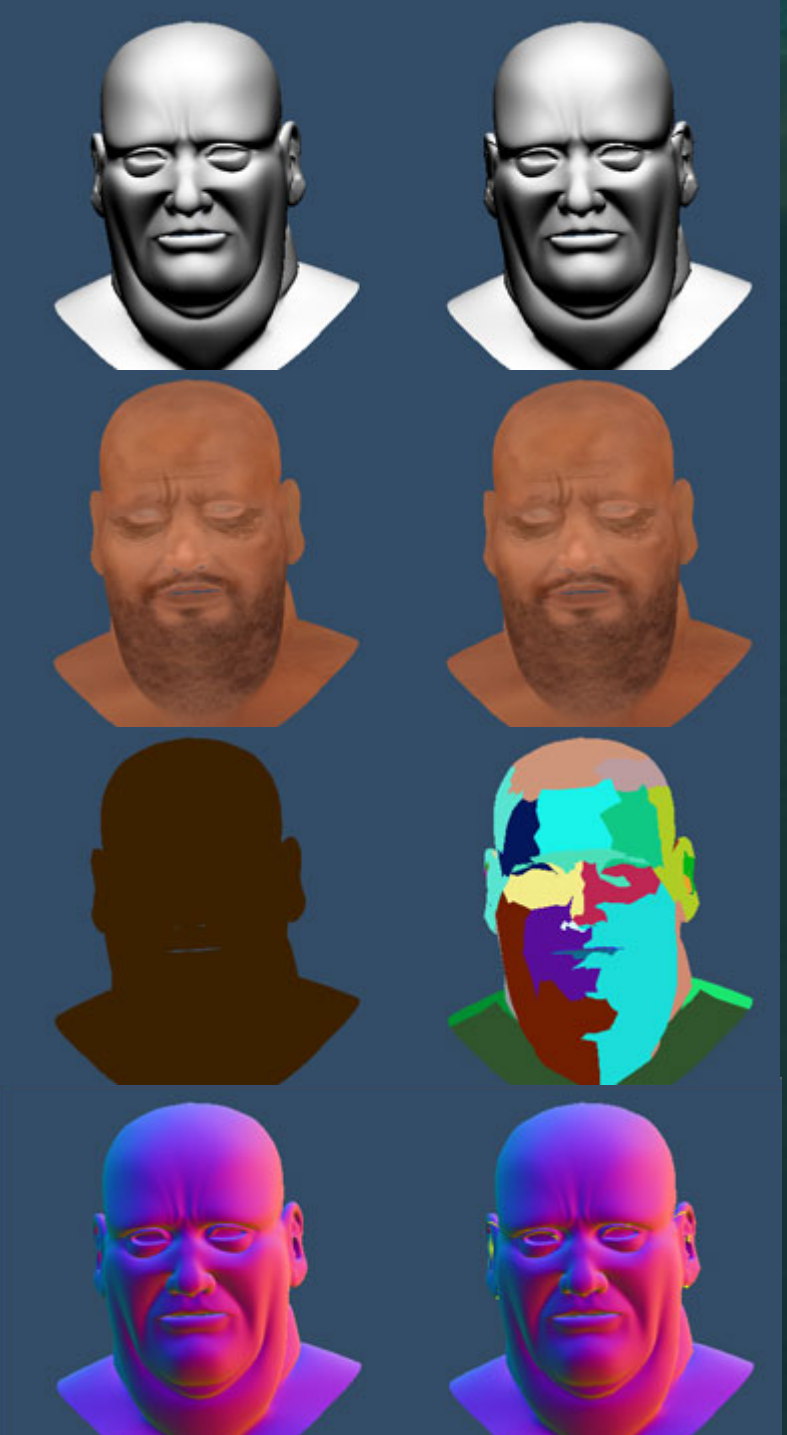
Generated Textures

Package Files

FX Composer

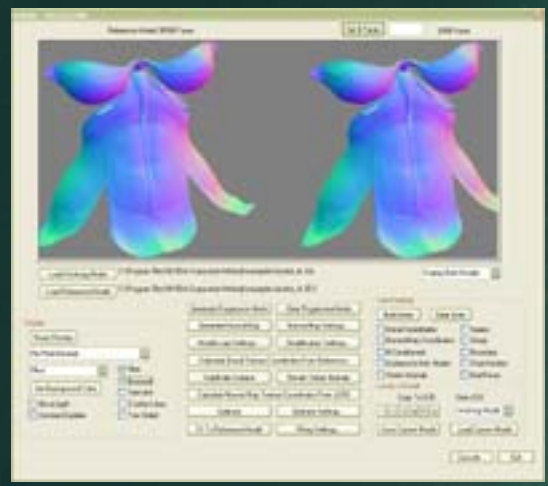
Create – Debug – Tune

Scripting automation & SDK for custom importer/exporter plug-ins



Melody

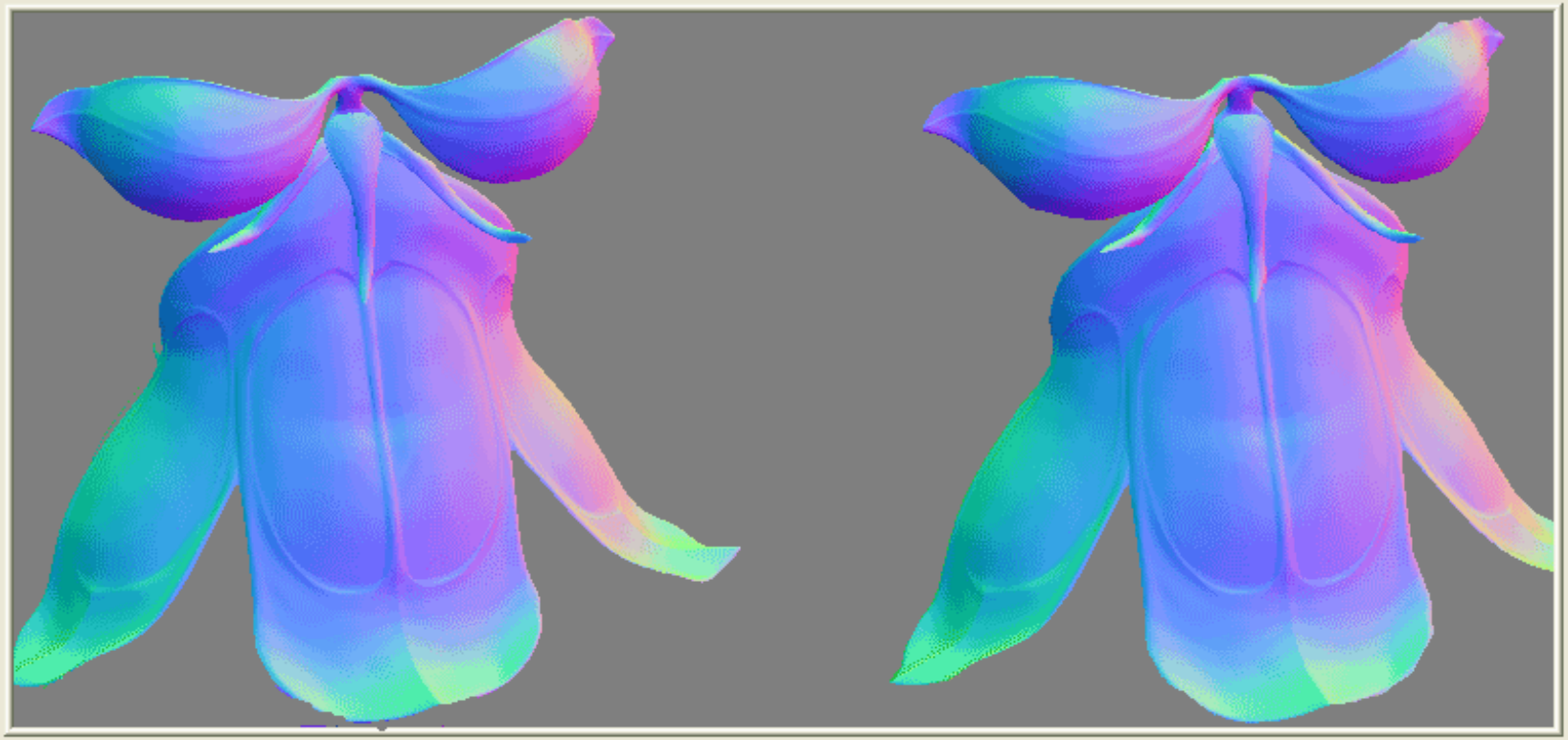
- Raycast normal map generation
- Chart-based UV parameterization
- Mesh optimization & simplification
- Operates on high-resolution meshes (~2 million polygons)



Reference Model 38908 Faces

Set # Faces

2000 Faces



Load Working Model C:\Program Files\NVIDIA Corporation\Melody\examples\bustier_lo.3ds

Load Reference Model C:\Program Files\NVIDIA Corporation\Melody\examples\bustier_hi.3DS

Display Both Models

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...

Display

Reset Position

Per Pixel Normals

Filled

Set Background Color

- [ ] Move Light
- [ ] Constant Update
- [x] Filter
- [x] Gouraud
- [ ] Specular
- [ ] Z buffer Lines
- [ ] Two Sided

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

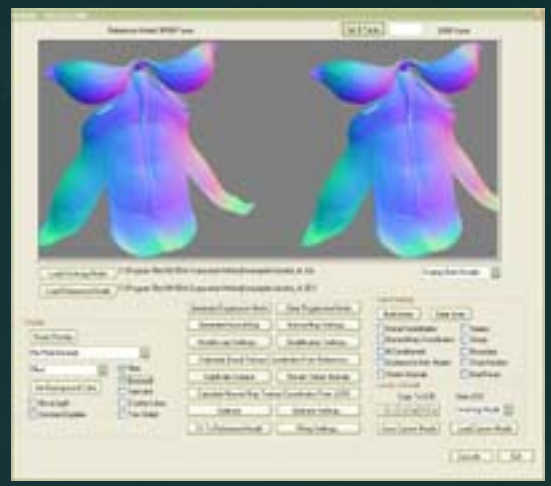
Save Current Model | Load Current Model

Melody In Your Pipeline

Hi-res model

Low-res model(s)

(optional)



Melody

Multiple LODs

High Quality Normal Maps

Texture Coordinates

Per-vertex Tangent Space Basis

Ambient Occlusion

Fast Normal Map Creation

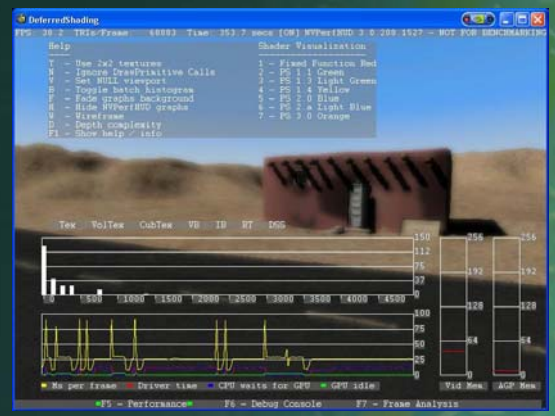
Progressive Mesh Decimation

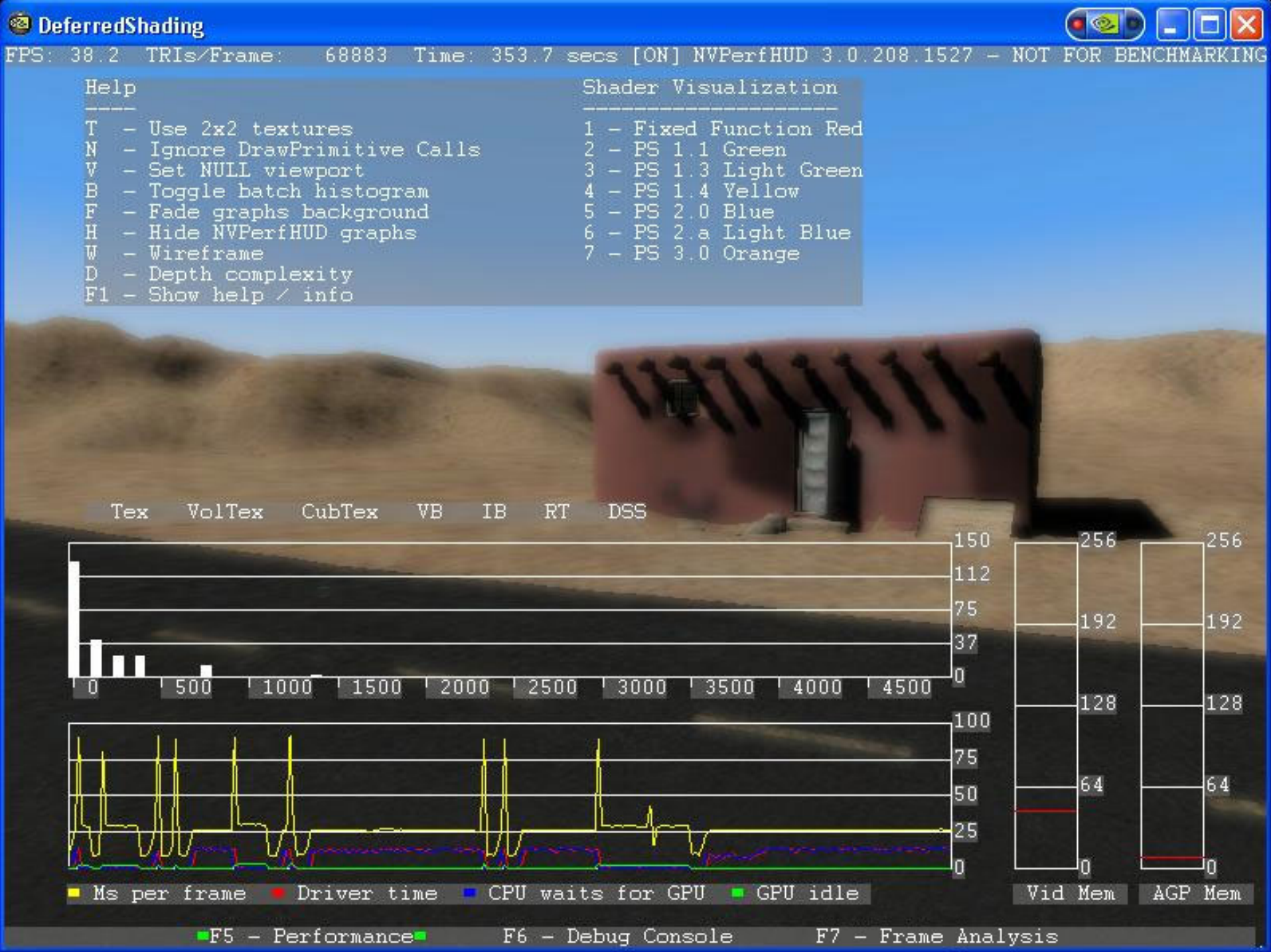
Texture Tools & Plug-ins

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

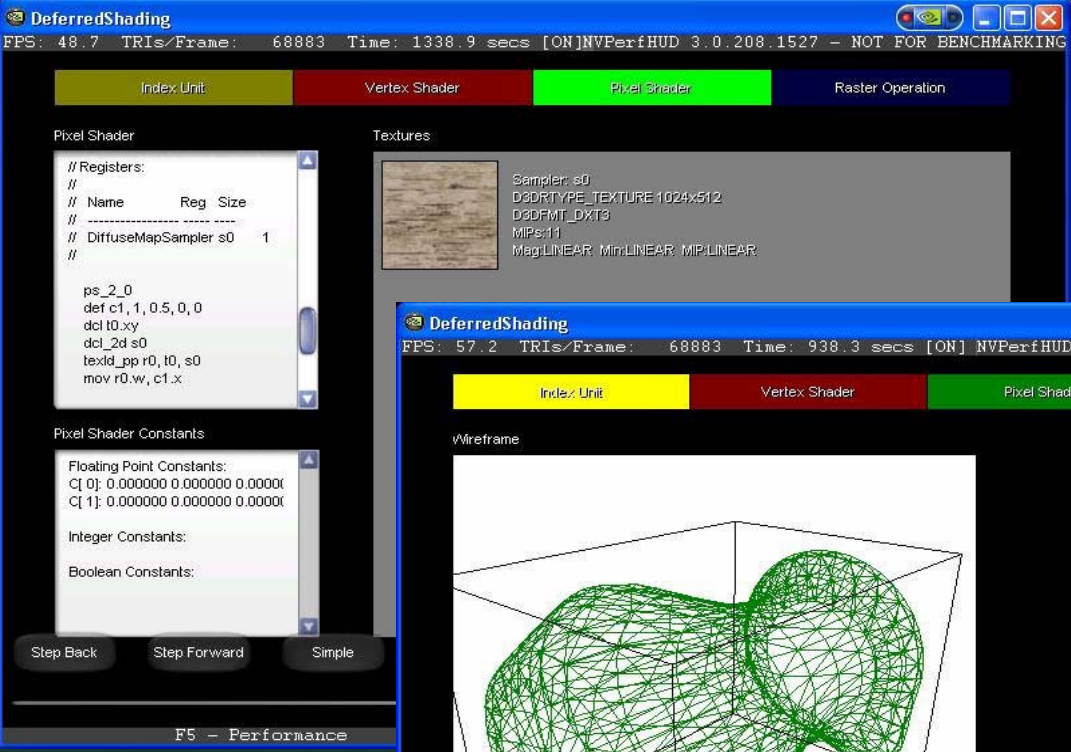
NVPerfHUD 3

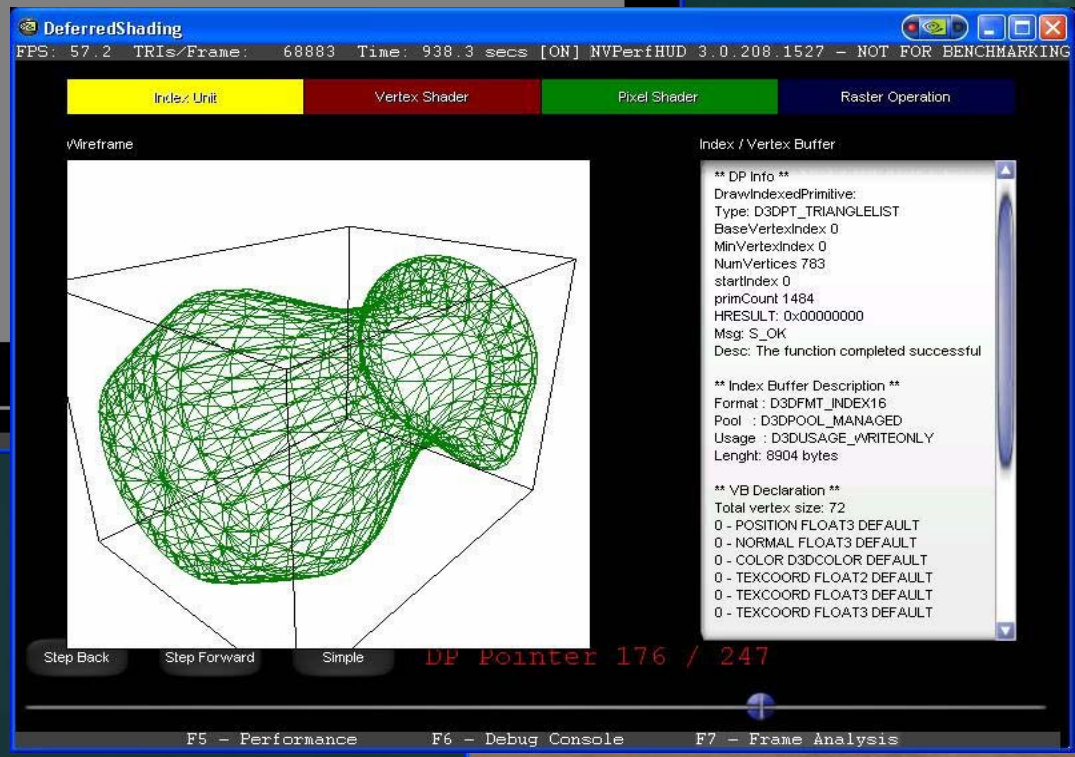
- Graph overlay of various vital statistics
  - Shown on top of your running application
  - Perform pipeline experiments to identify bottlenecks
- Debug Console shows runtime warnings, errors and custom messages from your application
- Frame Analysis Mode
  - Freeze the current frame and step through it one draw call at a time
  - Use advanced State Inspectors for each stage in the graphics pipeline




DeferredShading
FPS: 38.2 TRIs/Frame: 68883 Time: 353.7 secs [ON] NVPerfHUD 3.0.208.1527 - NOT FOR BENCHMARKING
Help
T - Use 2x2 textures
N - Ignore DrawPrimitive Calls
V - Set NULL viewport
B - Toggle batch histogram
F - Fade graphs background
H - Hide NVPerfHUD graphs
W - Wireframe
D - Depth complexity
F1 - Show help / info
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
Tex VolTex CubTex VB IB RT DSS
150
112
75
37
0
0 500 1000 1500 2000 2500 3000 3500 4000 4500
100
75
50
25
0
Ms per frame
Driver time
CPU waits for GPU
GPU idle
256
192
128
64
0
Vid Mem
AGP Mem
F5 - Performance
F6 - Debug Console
F7 - Frame Analysis

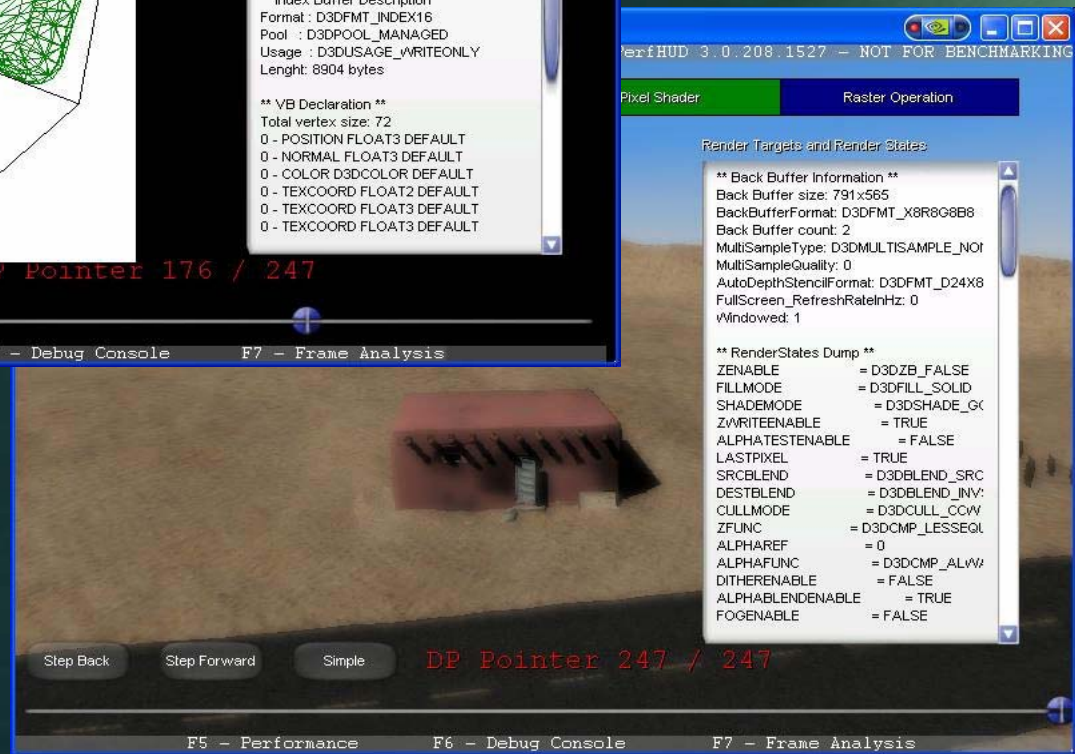
NVPerfHUD 3 State Inspectors





Index Unit

Pixel & Vertex Shaders



Raster Operations

NVPerfHUD 3
Quick Reference

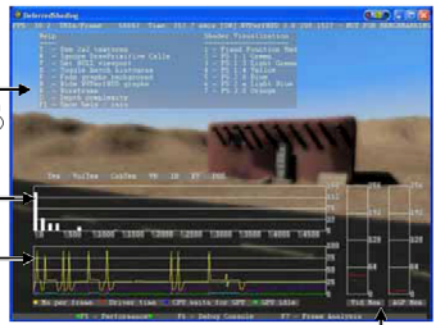
When NVPerfHUD is activated, you can perform graphics pipeline experiments, display graphs of performance metrics, and explore potential problems using several performance visualization modes. You can also switch to the Debug Console or Frame Analysis Mode for deeper analysis. Use these shortcut keys to switch modes:

- F5 **Performance Analysis Mode** - Use timing graphs and directed experiments to identify bottlenecks.
- F6 **Debug Console Mode** - Review messages from the DirectX Debug runtime, NVPerfHUD warnings and custom messages from your application.
- F7 **Frame Analysis Mode** - Freeze the current frame and step through your scene one draw call at a time, using advanced State Inspectors for state of the graphics pipeline.

Performance Analysis Mode

Configure the information displayed on the screen and perform several graphics pipeline experiments:

- F1 Cycle display of helpful information
- B Show Batch Size Histogram
- F Fade Background
- H Hide Graphs
- W Show Wireframe
- D Show Depth Complexity
- T Isolate the texture unit by forcing the GPU to use 2x2 textures
- V Isolate the vertex unit by using a 1x1 scissor rectangle to clip all rasterization and shading work
- N Eliminate the GPU (and state change overhead) by ignoring all draw calls



Frame Analysis Mode

Use the slider or the left/right arrow keys and the options below to scrub through your scene:

- A Toggle Simple/Advanced display
- J Jump to Warnings
- W Show Wireframe
- D Show Depth Complexity

State Inspectors

Click on the Advanced... button to use the advanced State Inspectors. You can click on the colored bar or use the shortcut keys below to switch between State Inspectors:

1. **Index Unit** – fetches vertex data
2. **Vertex Shader** – executes vertex shaders
3. **Pixel Shader** – executes pixel shaders
4. **Raster Operations** – post-shading operations

Using the **Index Unit State Inspector** you can verify all the information used to fetch the vertex data and make sure the geometry associated with the current draw call is correct.

Use the **Vertex** and **Pixel Shader State Inspectors** to verify that the shader program constants and textures are correct for the current draw call. Make sure the constants are not #NAN or #INF.

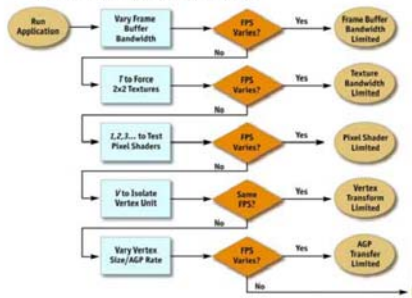
Using the **Raster Operations State Inspector** you confirm that the back buffer format has an alpha component when blending doesn't seem to be working properly, verify that opaque objects are not drawn with **blendEnable**, etc.

NVIDIA.

developer.nvidia.com
The Source for GPU Programming

NVIDIA. NVPerfHUD 3 Quick Reference

Identifying Bottlenecks



Note: If you suspect that you are CPU limited, press N at any time. If the frame rate of your application does not change, you are CPU limited.

GPU Optimizations

- **Speed up Pixel Shading**
  - Render depth first
  - Help early-Z optimizations throw away pixel processing
  - Store complex functions in textures
  - Move per-pixel work to the vertex shader
  - Use the lowest precision necessary
  - Avoid unnecessary normalization
  - Use half precision normalizes when possible (e.g. nrm_pp)
  - Consider using pixel shader level-of-detail
  - Disable trilinear filtering when unnecessary
- **Reduce Texture Bandwidth**
  - Reduce the size of your textures
  - Always use mipmapping on any surface that may be minified
  - Compress all color textures
  - Avoid expensive texture formats if not necessary
- **Optimize Framebuffer Bandwidth**
  - Render depth first
  - Reduce alpha blending
  - Turn off depth writes when possible
  - Avoid extraneous color buffer clears
  - Render front-to-back clears
  - Optimize skybox rendering
  - Only use floating point framebuffers when necessary
  - Use a 16-bit depth buffer when possible
  - Use a 16-bit color when possible

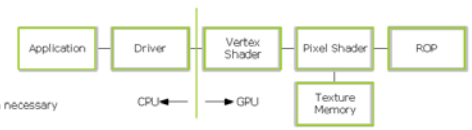
Methodology

1. Identify the bottleneck
2. Optimize the bottleneck stage
3. Repeat steps 1 and 2 until desired performance level is achieved.

CPU Optimizations

- **Reduce Resource Locking**
  - Avoid lock or read from a surface you were previously rendering to
  - Avoid write to a surface the GPU is reading from, like a texture or a vertex buffer
- **Minimize Number of Draw Calls**
  - If using triangle strips, use degenerate triangles to stitch together disjoint strips.
  - Use texture pages.
  - Use the vertex shader constant memory as a lookup table of matrices
  - Use geometry instancing if you have multiple copies of the same mesh in your scene
  - Use CPU shader branching to increase batch size
  - Defer decisions as far down in the pipeline as possible
- **Reduce Cost of Vertex Transfers**
  - Use the fewest number of bytes possible in vertex format
  - Generate potentially derivable vertex attributes inside the vertex program instead of storing them inside of the input vertex format.
  - Use 16-bit indices instead of 32-bit indices
  - Access vertex data in a relatively sequential manner
- **Optimize Vertex Processing**
  - Pull out per-object computations onto the CPU
  - Optimize for the post-TnL vertex cache
  - Reduce the number of vertices processed
  - Use vertex processing LOD
  - Use correct coordinate space
  - Use vertex branching to *early-out* of computations

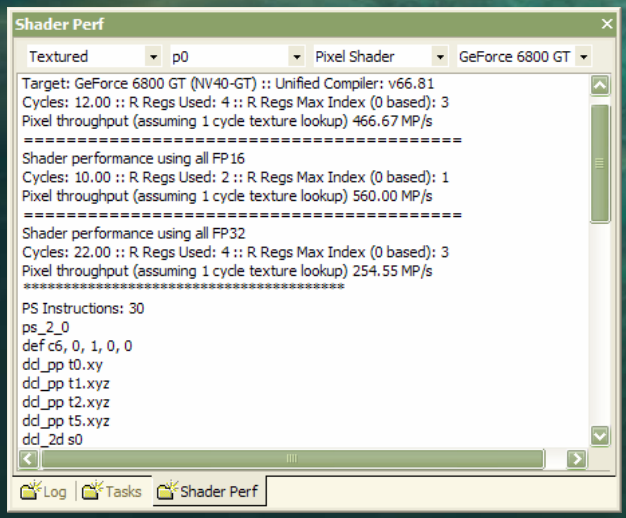
Graphics Pipeline



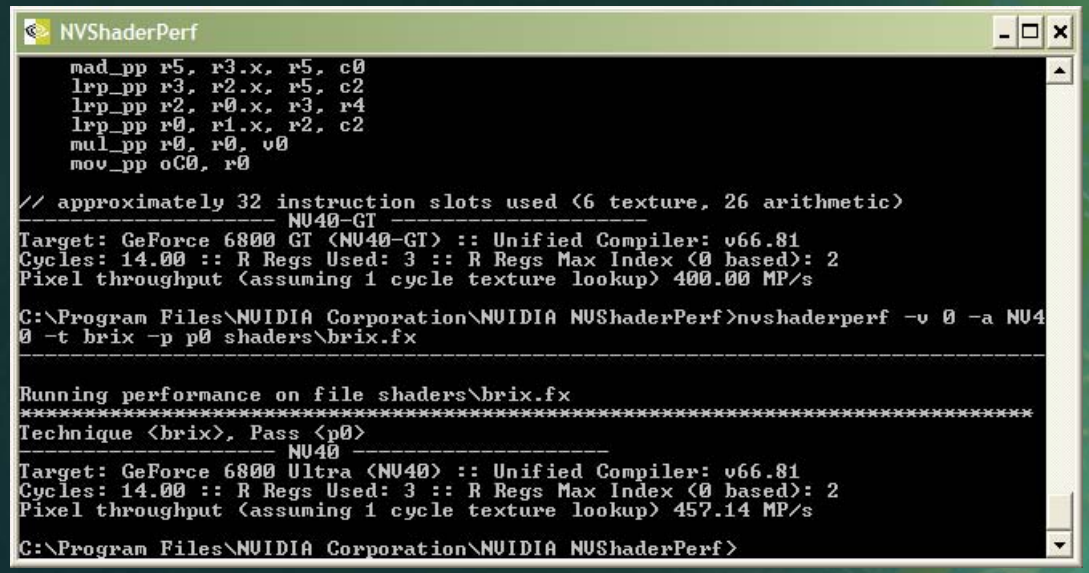
Refer to the *NVPerfHUD User Guide* for optimization details

developer.nvidia.com
The Source for GPU Programming

NVShaderPerf



- Same technology as Shader Perf panel in FX Composer
- Analyze DirectX and OpenGL Shaders
  - HLSL, GLSL, Cg, !!FP1.0, !!ARBfp1.0, VS1.x, VS2.x, VS3.x, PS1.x, PS2.x, PS3.x, etc.
- Shader performance regression testing on the entire family of NVIDIA GPUs, without rebooting!



Utilities, libraries and more...

- NVMeshMender (C++ src code)
  - Fixes problem geometry
  - Prepares meshes for per-pixel lighting
- NVTriStrip (.lib & src code)
  - cache-aware creation of optimized tri lists or strips

Questions / Feature Requests?

All of this and more, available now at

developer.nvidia.com

The Source for GPU Programming

Please send questions, feature requests & comments about our SDK and developer tools to:

sdkfeedback@nvidia.com