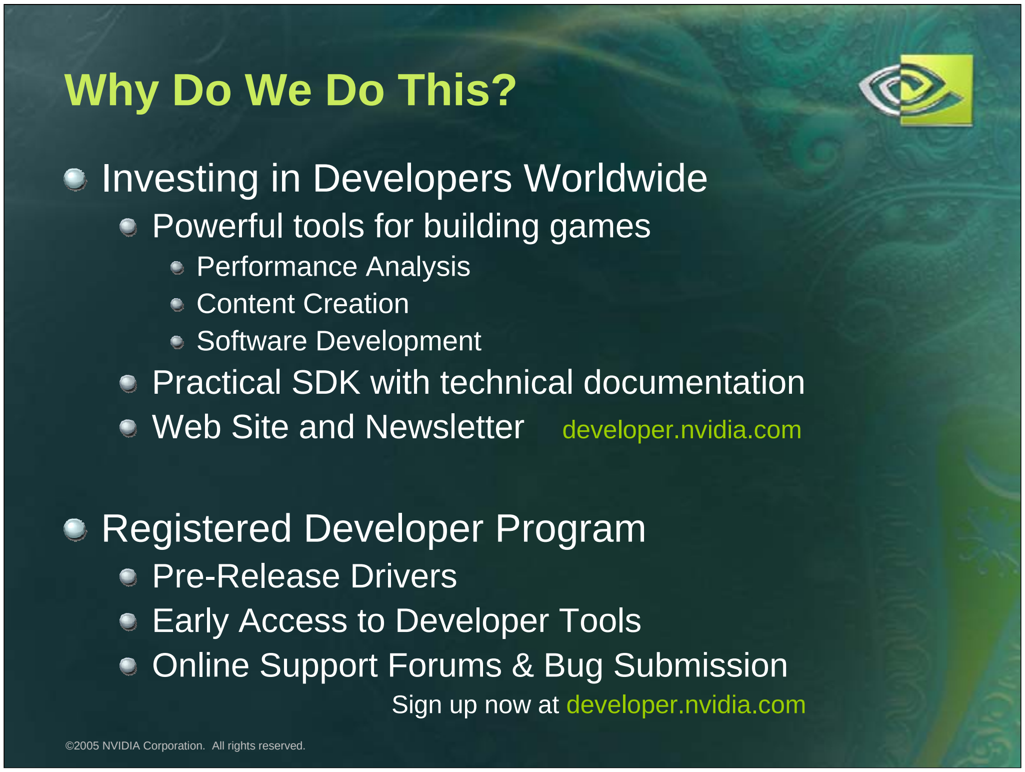
NVIDIA Developer Toolkit

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.

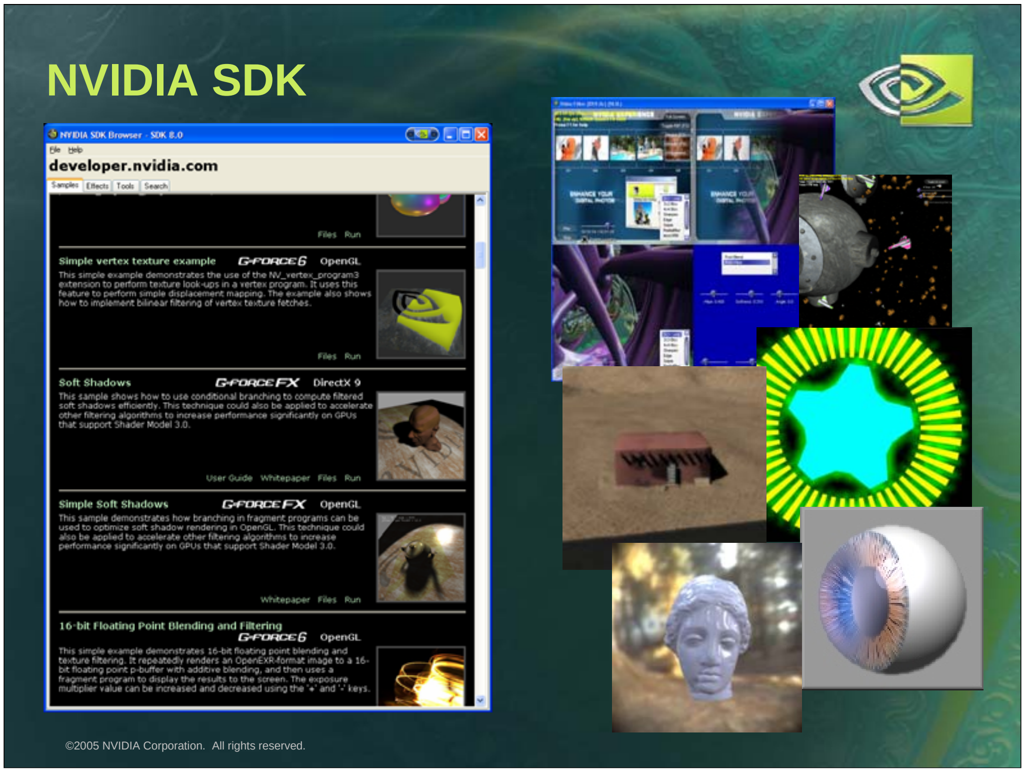
- **Tons of updated and all-new DirectX and OpenGL code samples with full source code and helpful whitepapers:**
  GPU Cloth, Geometry Instancing, Rainbow Fogbow, 2xFP16 HRD, Perspective Shadow Maps, Texture Atlas Utility, ...
- **Hundreds of effects, complete with custom geometry, animation and more:**
  Skin, Plastics, Flame/Fire, Glow, Gooch, Image Filters, HLSL Debugging Techniques, Texture BRDFs, Texture Displacements, HDR Tonemapping, and even a simple Ray Tracer!

SDK Browser

100+ code samples

250+ compelling effects

Search and filter to find exactly what you need



NVIDIA SDK

Simple vertex texture example GeFORCE 6 OpenGL

This simple example demonstrates the use of the NV_vertex_program3 extension to perform texture look-ups in a vertex program. It uses this feature to perform simple displacement mapping. The example also shows how to implement bilinear filtering of vertex texture fetches.

Soft Shadows GeFORCE FX DirectX 9

This sample shows how to use conditional branching to compute filtered soft shadows efficiently. This technique could also be applied to accelerate other filtering algorithms to increase performance significantly on GPUs that support Shader Model 3.0.

Simple Soft Shadows GeFORCE FX OpenGL

This sample demonstrates how branching in fragment programs can be used to optimize soft shadow rendering in OpenGL. This technique could also be applied to accelerate other filtering algorithms to increase performance significantly on GPUs that support Shader Model 3.0.

16-bit Floating Point Blending and Filtering GeFORCE 6 OpenGL

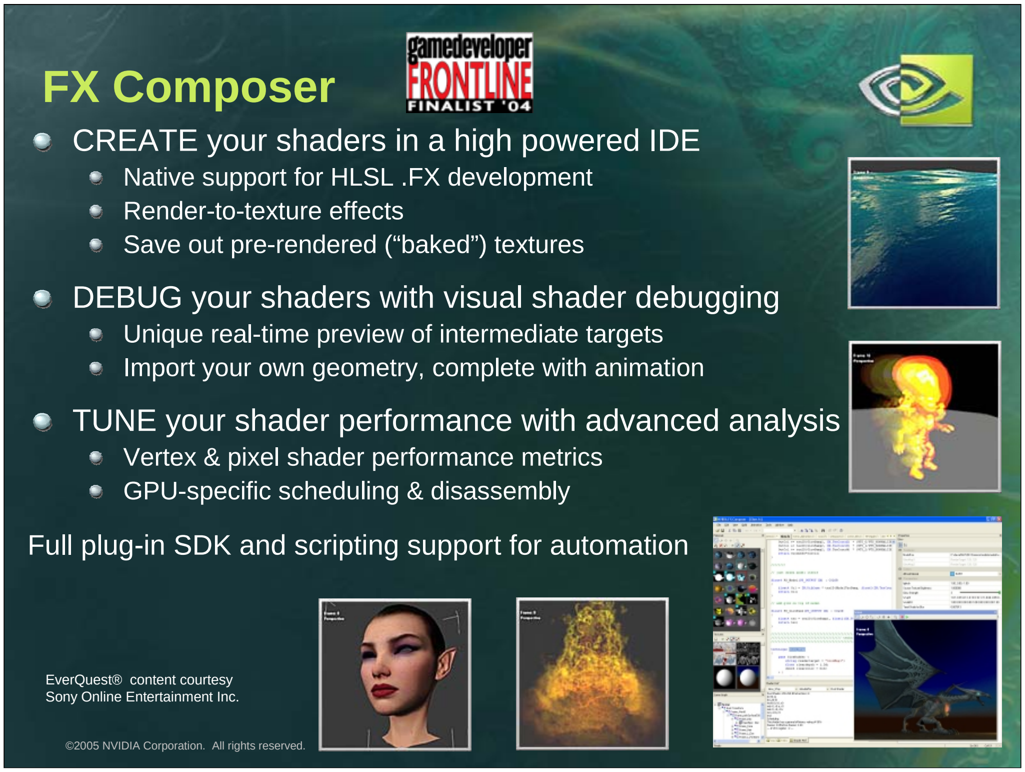
This simple example demonstrates 16-bit floating point blending and texture filtering. It repeatedly renders an OpenEXR-format image to a 16-bit floating point p-buffer with additive blending, and then uses a fragment program to display the results to the screen. The exposure multiplier value can be increased and decreased using the '+' and '-' keys.

Practical Code Samples

- Video Filtering
- Antialiasing with Post-processing
- Deferred Shading
- Geometry Instancing

Compelling Effects

- Image processing shaders
- Post-processing effects
- UV & MIP level debugging shaders
- Powerful examples of DXSAS



FX Composer

- 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

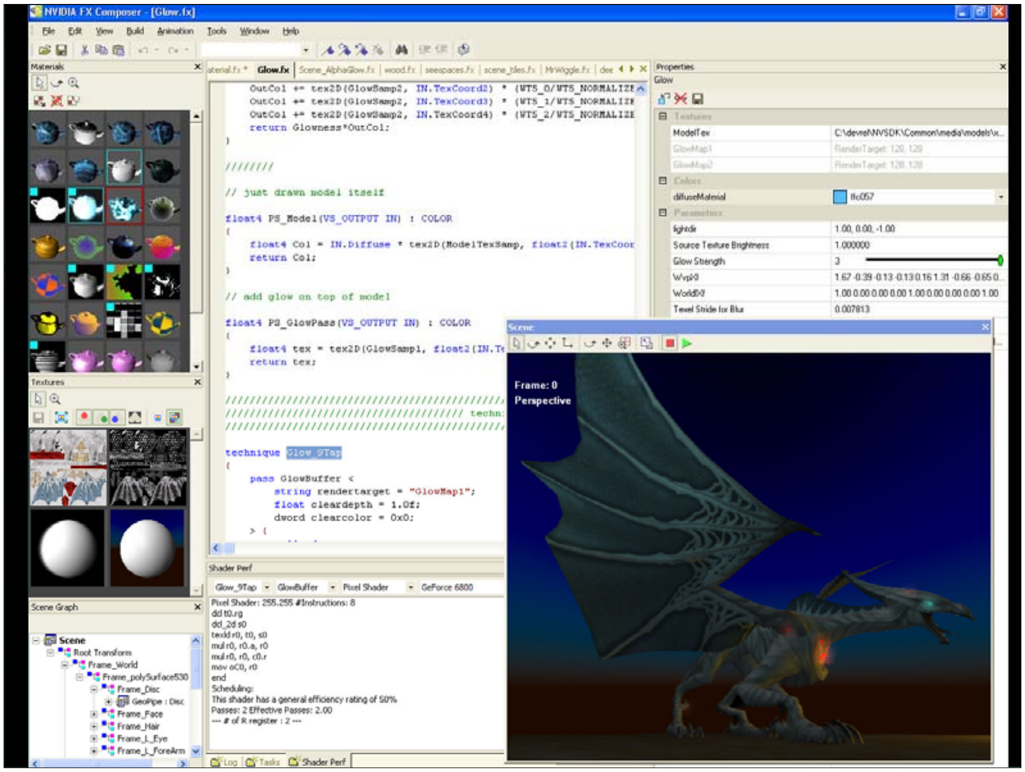
EverQuest® content courtesy
Sony Online Entertainment Inc.

Be sure to check out the complete User Guide and helpful tutorials!

Also see the full presentation dedicated to FX Composer.

EverQuest® content courtesy Sony Online Entertainment Inc.

EverQuest is a registered trademark of Sony Computer Entertainment America Inc.



```
    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;
    > {
```

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 ---
```

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

CREATE your shaders in a high powered development environment

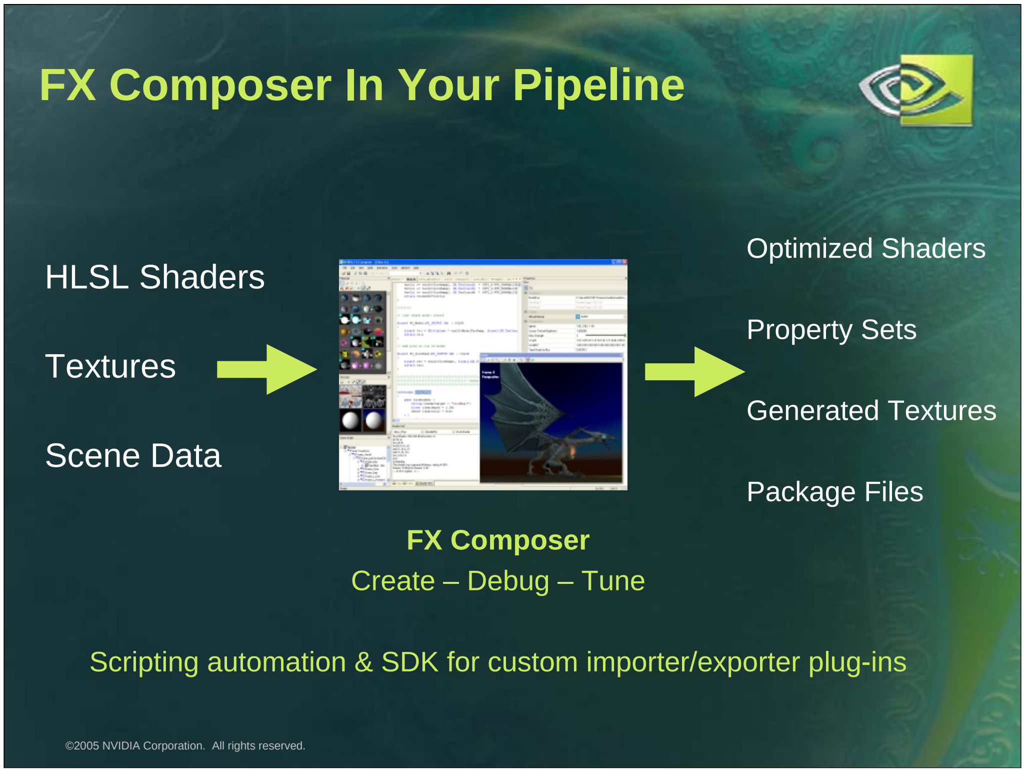
- Sophisticated text editing with intellisense (auto-complete) and syntax highlighting
- Work directly with HLSL .FX files, creating multiple techniques and passes
- Use the .FX files you create with FX Composer directly in your application
- Convenient, artist-friendly graphical editing of shader properties
- Supports Microsoft DirectX standard HLSL semantics and annotations
- Support for all DirectX 9.0 shader profiles
- Develop your shaders on your own models with lighting and animation

DEBUG your shaders with visual shader debugging features

- Interactive compiler helps you find and fix problems
- Visible preview of source and intermediate textures targets
- Interactive jump-to-error feature helps you fix problems quickly

TUNE your shader performance with advanced analysis and optimization

- Enables performance tuning workflow for vertex and pixel shaders
- Simulates performance for the entire family of GeForce FX GPUs
- Capture of pre-calculated functions to texture look-up table
- Provides empirical performance metrics such as GPU cycle count, register usage, pixels per second, and more
- Optimization hints notify you of performance bottlenecks



FX Composer In Your Pipeline

HLSL Shaders

Textures

Scene Data

FX Composer

Create – Debug – Tune

Optimized Shaders

Property Sets

Generated Textures

Package Files

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

Goes-in

•Supports all DirectX texture formats

•Import scene data from .obj, .x, .ply, or .x files

Comes-out

•Use the Shader Perf panel to optimize your shaders

•Save properties sets as XML files (re-apply in engine)

•Save textures generated by HLSL functions (e.g. noise)

•Package shaders, textures & full scene in one file for review/collaboration

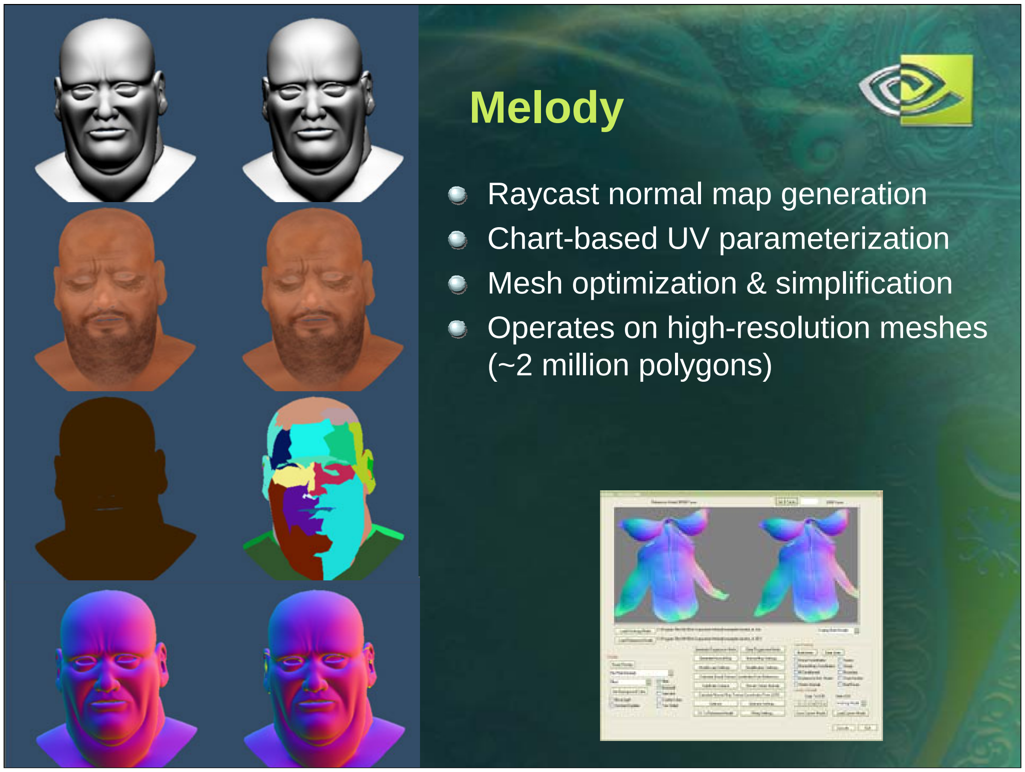
Scripting automation & SDK for custom importers/exporters

•Sample code for custom importers & exporters

•Save properties bundles in your own format

•Automate just about anything using .NET scripting

(e.g. screenshots for all shaders in a directory)



Melody

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

Head model comparison

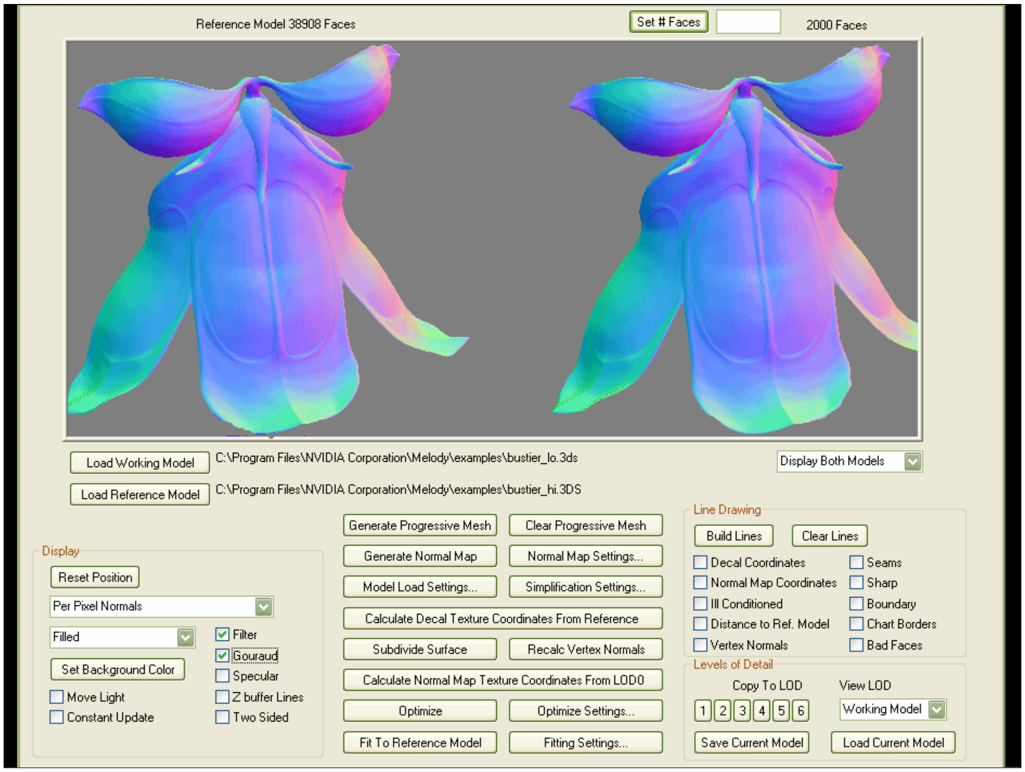
Left: 18k poly reference model

Right: 4k poly working model

(auto simplified using progressive mesh deformation algorithm)

Note various visualization modes (top to bottom):

1. Filled
2. Textured
3. Chart View
4. Object Space Normals



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

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

Per Pixel Normals

Filled

☑ Filter
☑ Gouraud
☐ Specular
☐ Z buffer Lines
☐ Two Sided
☐ Move Light
☐ Constant Update

☐ Decal Coordinates
☐ Normal Map Coordinates
☐ Ill Conditioned
☐ Distance to Ref. Model
☐ Vertex Normals
☐ Seams
☐ Sharp
☐ Boundary
☐ Chart Borders
☐ Bad Faces

View LOD: Working Model

Dawn's Bustier

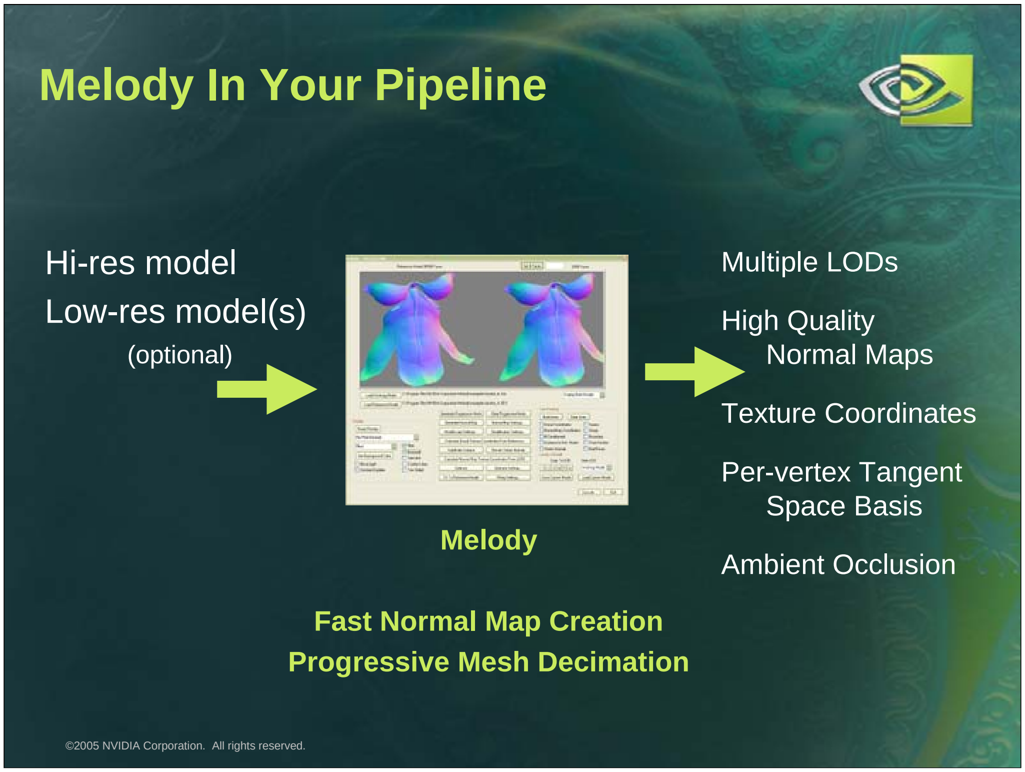
Left 38K polygons

Right 2K polygons with Melody normal map

Can you tell the difference?

Visualization Options Include:

- Vertex or per-pixel normals (textured or not)
- Charts & attribute groups
- Tangent to object space
- Tangent or object space light mapping
- Combined with Filled, Wireframe or Outlined mode



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

Goes-in

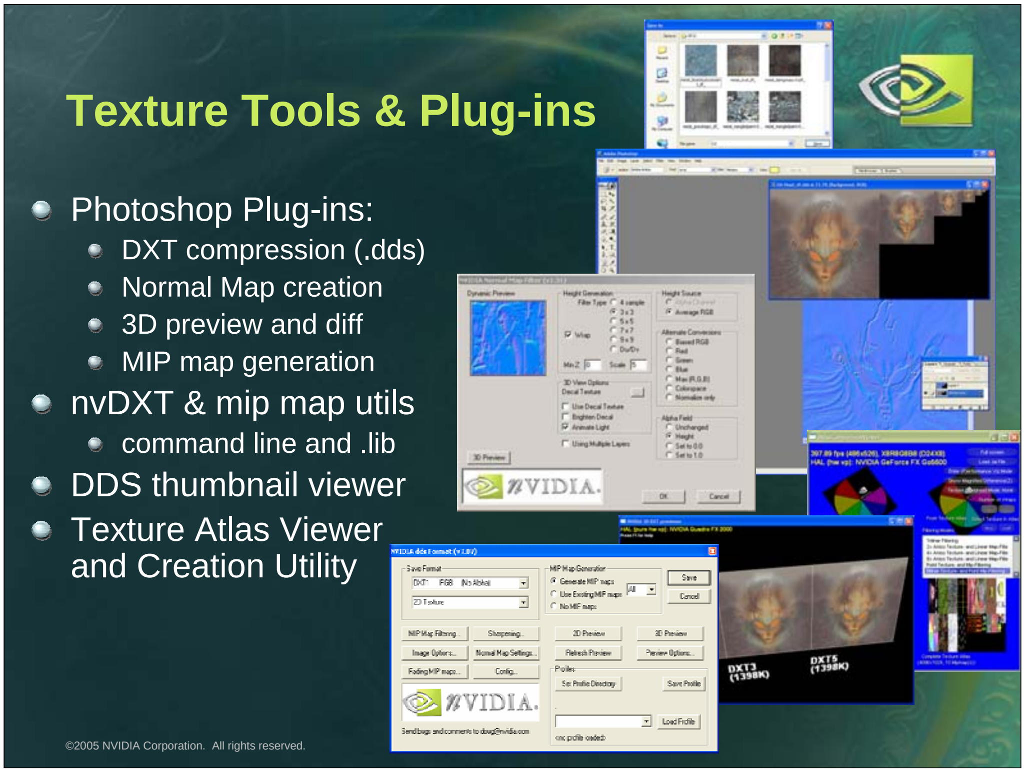
- Import models from .obj, .3ds or .ply files.
- Import a low-res model or generate LODs using Melody

Comes-out

- Multiple LODs ()
- Normal map file for each low-res model (tons of options)
- Object space and tangent space normal maps
- FP support (8 or 32 bits / channel)
- Displacement maps
- Re-sampled color map (normal & color)
- Greyscale Height map
- Generates ambient occlusion term

Visual Preview

- Simple object manipulation
- Powerful visualization modes



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

Photoshop Plug-ins

•Allows saving in various formats, incl. DXTC (.dds files)

•Generate mip-maps in variety of ways

(box filter to Kaiser gamma-space filter)

•Preview w/ 3D rendering options

(e.g. anisotropy, format comparisons etc.)

•Create normal maps from height maps

nvDXT & mip map utilities

•Command line interface for pipeline automation

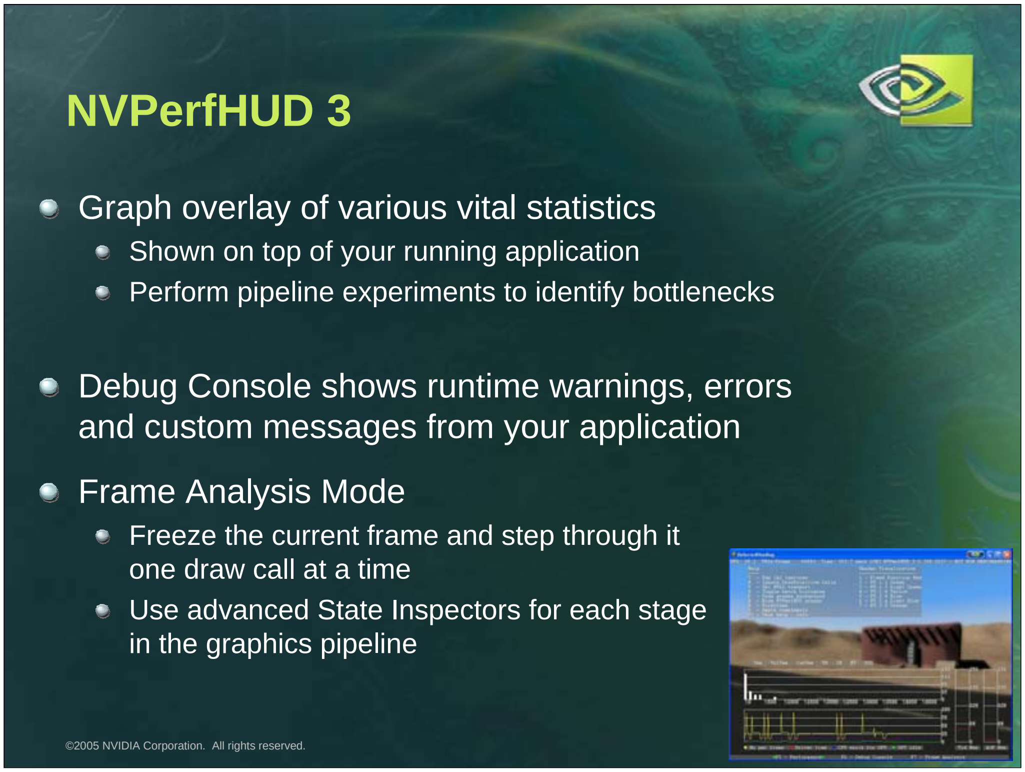
•Available .lib for tools integration

•Detach & Stitch utilities to manipulate MIP maps

DDS Thumbnail Viewer

•Provides convenient preview of DXT compressed files.

Also check out our Texture Atlas whitepaper...



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

Be sure to check out the User Guide for full details!

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

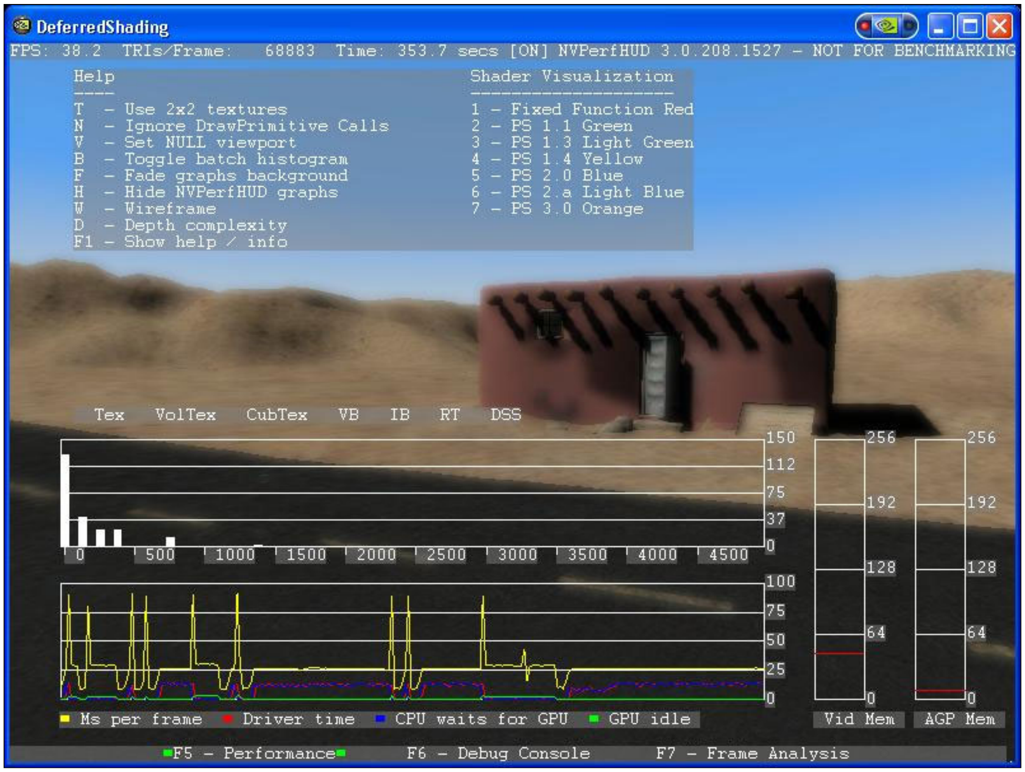
Use timing graphs and directed experiments to identify bottlenecks.

F6 Debug Console Mode

Review messages from the DirectX Debug runtime, NVPerfHUD warnings and custom messages from your application.

F7 Frame Analysis Mode

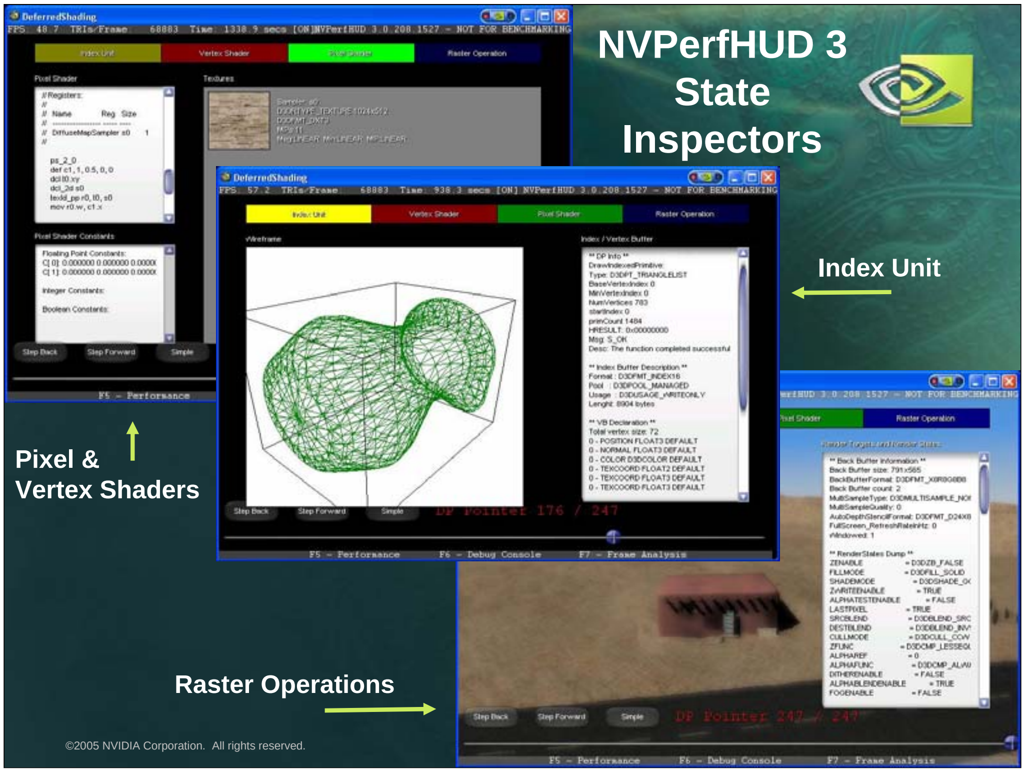
Freeze the current frame and step through your scene one draw call at a time, using advanced State Inspectors for state of the graphics pipeline.



```
DeferredShading
FPS: 38.2  TRIs/Frame:   68883  Time: 353.7 secs [ON] NVPerfHUD 3.0.208.1527 - NOT FOR BENCHMARKING
    Help                                      Shader Visualization
    ----                                      --------------------
    T  - Use 2x2 textures                     1 - Fixed Function Red
    N  - Ignore DrawPrimitive Calls           2 - PS 1.1 Green
    V  - Set NULL viewport                    3 - PS 1.3 Light Green
    B  - Toggle batch histogram               4 - PS 1.4 Yellow
    F  - Fade graphs background               5 - PS 2.0 Blue
    H  - Hide NVPerfHUD graphs                6 - PS 2.a Light Blue
    W  - Wireframe                            7 - PS 3.0 Orange
    D  - Depth complexity
    F1 - Show help / info
```

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



NVPerfHUD 3 State Inspectors

Index Unit

Pixel &
Vertex Shaders

Raster Operations

```
FPS  48.7  TRIs/Frame    68883  Time  1338.9 secs [ON]NVPerfHUD 3.0.208.1527 - NOT FOR BENCHMARKING
```

```
// Registers:
//
//  Name             Reg   Size
//  ---------------- ----- ----
//  DiffuseMapSampler s0      1
//

    ps_2_0
    def c1, 1, 0.5, 0, 0
    dcl t0.xy
    dcl_2d s0
    texld_pp r0, t0, s0
    mov r0.w, c1.x
```

```
Floating Point Constants:
C[ 0]: 0.000000 0.000000 0.0000(
C[ 1]: 0.000000 0.000000 0.0000(

Integer Constants:

Boolean Constants:
```

```
FPS  57.2  TRIs/Frame    68883  Time  938.3 secs [ON] NVPerfHUD 3.0.208.1527 - NOT FOR BENCHMARKING
```

```
** DP Info **
DrawIndexedPrimitive:
Type: D3DPT_TRIANGLELIST
BaseVertexIndex: 0
MinVertexIndex: 0
NumVertices: 783
startIndex: 0
primCount: 1484
HRESULT: 0x00000000
Msg: S_OK
Desc: The function completed successful

** Index Buffer Description **
Format : D3DFMT_INDEX16
Pool   : D3DPOOL_MANAGED
Usage  : D3DUSAGE_WRITEONLY
Lenght: 8904 bytes

** VB Declaration **
Total vertex size: 72
0 - POSITION FLOAT3 DEFAULT
0 - NORMAL FLOAT3 DEFAULT
0 - COLOR D3DCOLOR DEFAULT
0 - TEXCOORD FLOAT2 DEFAULT
0 - TEXCOORD FLOAT3 DEFAULT
0 - TEXCOORD FLOAT3 DEFAULT
```

```
DP Pointer 176 / 247
```

```
** Back Buffer information **
Back Buffer size: 791x565
BackBufferFormat: D3DFMT_X8R8G8B8
Back Buffer count: 2
MultiSampleType: D3DMULTISAMPLE_NO
MultiSampleQuality: 0
AutoDepthStencilFormat: D3DFMT_D24X8
FullScreen_RefreshRateInHz: 0
Windowed: 1

** RenderStates Dump **
ZENABLE              = D3DZB_FALSE
FILLMODE             = D3DFILL_SOLID
SHADEMODE            = D3DSHADE_G
ZWRITEENABLE         = TRUE
ALPHATESTENABLE      = FALSE
LASTPIXEL            = TRUE
SRCBLEND             = D3DBLEND_SRC
DESTBLEND            = D3DBLEND_INV
CULLMODE             = D3DCULL_CCW
ZFUNC                = D3DCMP_LESSEQ
ALPHAREF             = 0
ALPHAFUNC            = D3DCMP_ALW
DITHERENABLE         = FALSE
ALPHABLENDENABLE     = TRUE
FOGENABLE            = FALSE
```

```
DP Pointer 247 / 247
```

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.

Index Unit State Inspector

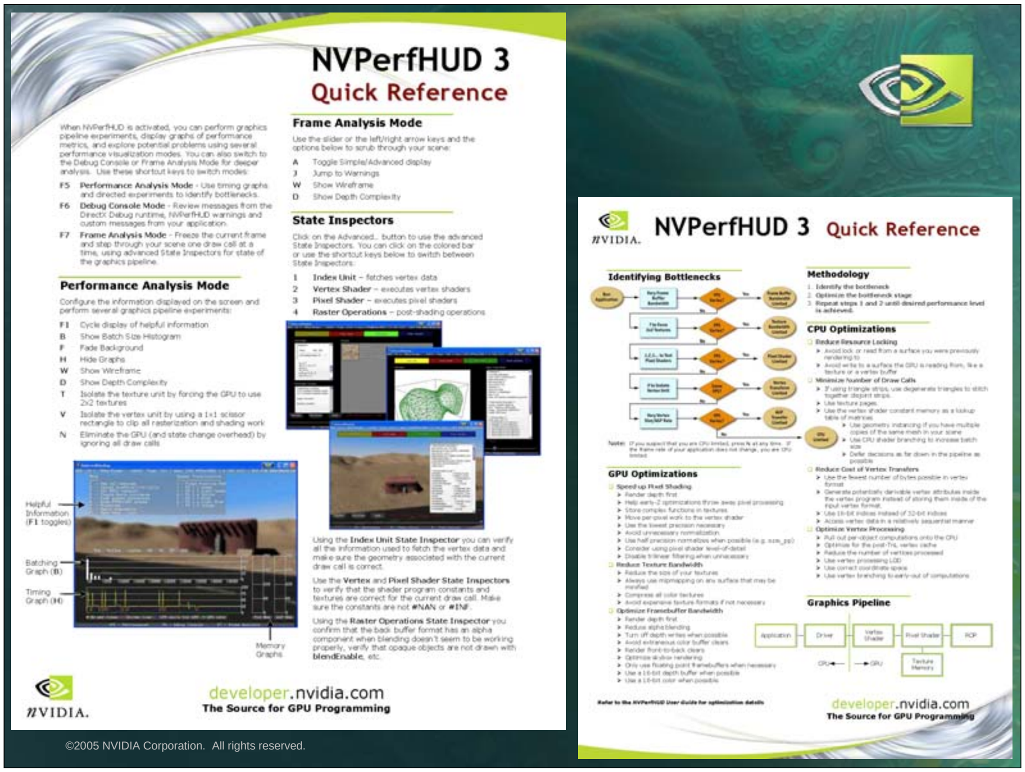
Verify all the information used to fetch the vertex data and make sure the geometry associated with the current draw call is correct.

Vertex and **Pixel Shader State Inspectors**

Verify that the shader program constants and textures are correct for the current draw call. Make sure the constants are not **#NAN** or **#INF**.

Raster Operations State Inspector

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.



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.

developer.nvidia.com
The Source for GPU Programming

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

Refer to the NVPerfHUD User Guide for optimization details

Methodology

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

CPU Optimizations

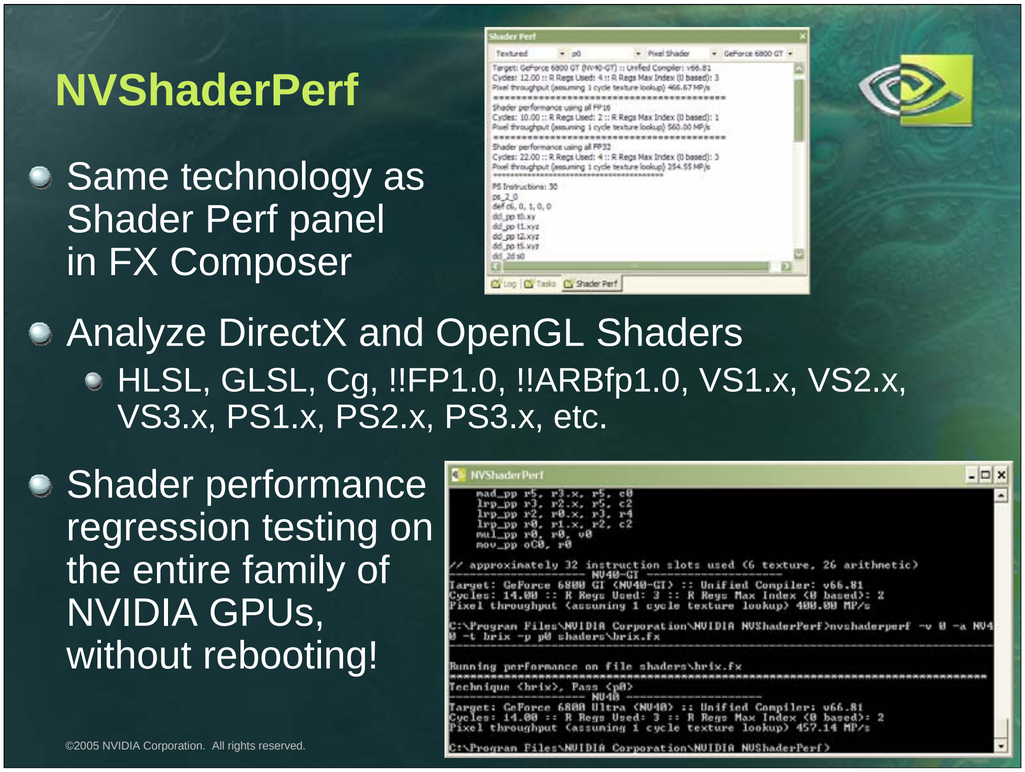
- Reduce Resource Locking
  - Avoid lock or read from a surface you were previously rendering to
  - Avoid write to a surface the GPU is reading from, like a texture or a vertex buffer
- Minimize Number of Draw Calls
  - If using triangle strips, use degenerate triangles to stitch together disjoint strips
  - 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

developer.nvidia.com
The Source for GPU Programming

This handy Quick Reference sheet is included in the NVPerfHUD installer.

Be sure to read through the complete User Guide with chapters on:
- Enabling NVPerfHUD
- Bottleneck Identification
- Crushing Bottlenecks
- Advanced State Inspectors
- Troubleshooting



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!

```
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
```

```
    mad_pp r5, r3.x, r5, c0
    lrp_pp r3, r2.x, r5, c2
    lrp_pp r2, r0.x, r3, r4
    lrp_pp r0, r1.x, r2, c2
    mul_pp r0, r0, v0
    mov_pp oC0, r0

// approximately 32 instruction slots used (6 texture, 26 arithmetic)
-------------------- NV40-GT --------------------
Target: GeForce 6800 GT (NV40-GT) :: Unified Compiler: v66.81
Cycles: 14.00 :: R Regs Used: 3 :: R Regs Max Index (0 based): 2
Pixel throughput (assuming 1 cycle texture lookup) 400.00 MP/s

C:\Program Files\NVIDIA Corporation\NVIDIA NVShaderPerf>nvshaderperf -v 0 -a NV4
0 -t brix -p p0 shaders\brix.fx
--------------------------------------------------------------------------------

Running performance on file shaders\brix.fx
********************************************************************************
Technique (brix), Pass (p0)
-------------------- NV40 --------------------
Target: GeForce 6800 Ultra (NV40) :: Unified Compiler: v66.81
Cycles: 14.00 :: R Regs Used: 3 :: R Regs Max Index (0 based): 2
Pixel throughput (assuming 1 cycle texture lookup) 457.14 MP/s

C:\Program Files\NVIDIA Corporation\NVIDIA NVShaderPerf>
```

Identify shader performance issues:

•Run once across all your shaders to create a baseline report

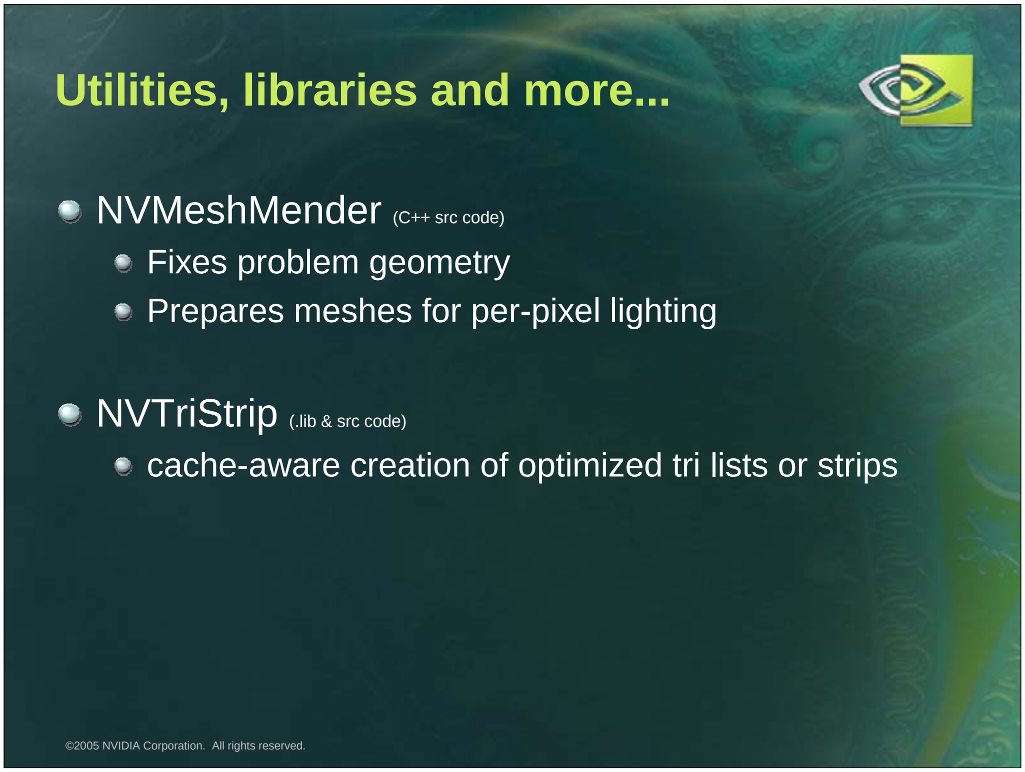
•Run the same version again later to identify which shaders have improved / degraded performance

Identify driver performance issues:

•Run once across all your shaders to create a baseline report

•Then run a different version on the same shaders to identify improvements and/or regressions

Please let us know if you find any driver performance issues!!



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

NVMeshMender:

•Creates tangent basis for per-pixel lighting

•Creates smoothed normals

•Creates u, v coordinates

•Handles mirrored (u,v) and cylindrical wrapping

NVTriStrip library:

•Library to stripify arbitrary geometry meshes

•Flexibly optimizes for postTnL vertex caches

•Option to stitch strips using degenerate triangles

•Option to remap indices to improve vertex buffer locality

•Outputs 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