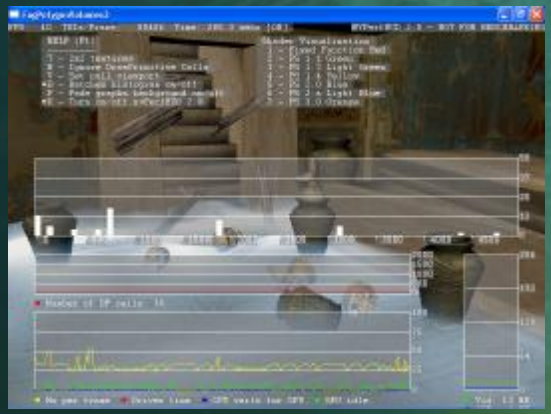
Overview

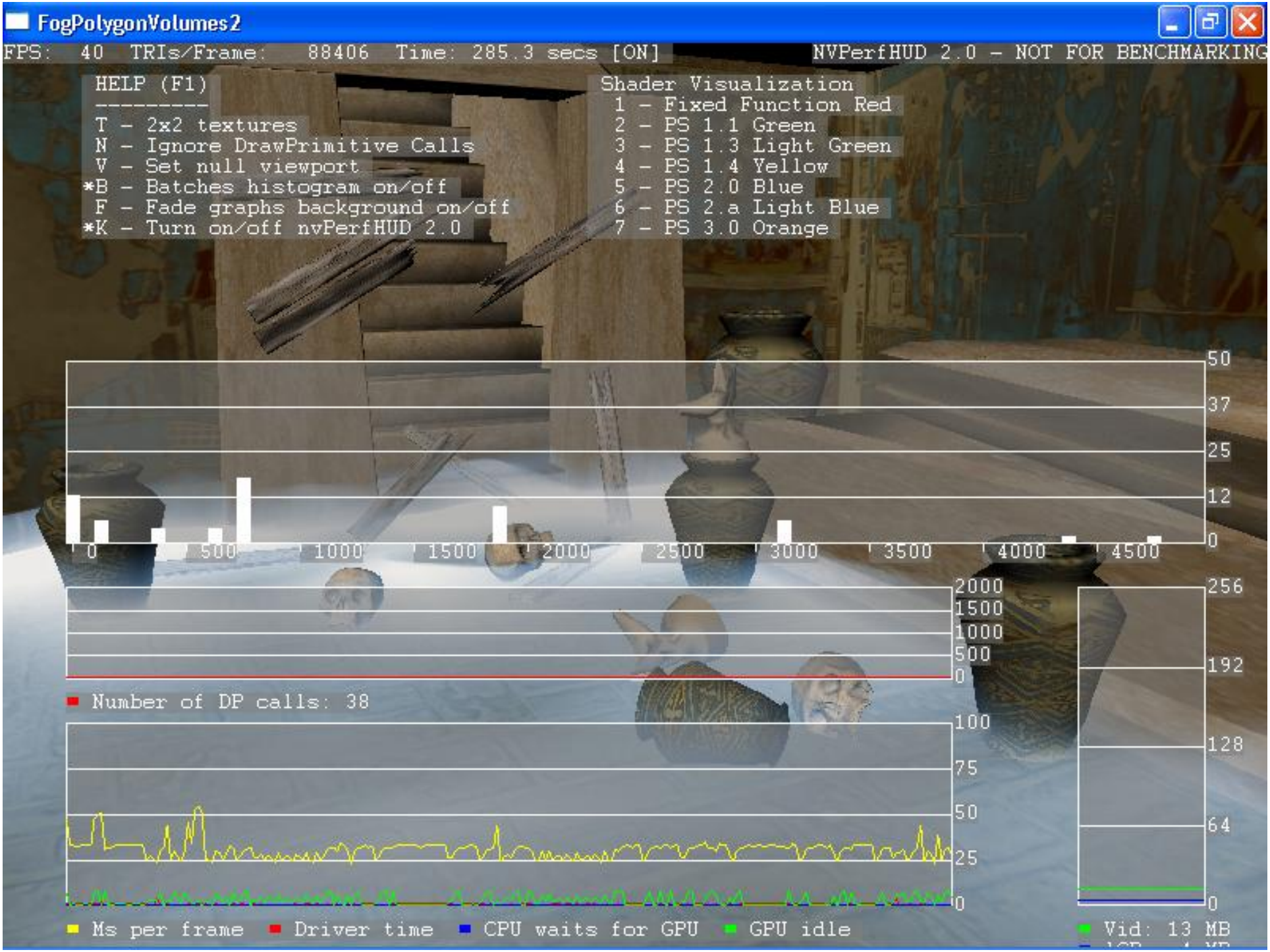
- Tools for the analysis
- Finding pipeline bottlenecks
- Practice identifying the problems

Analysis Tools

NVPerfHUD

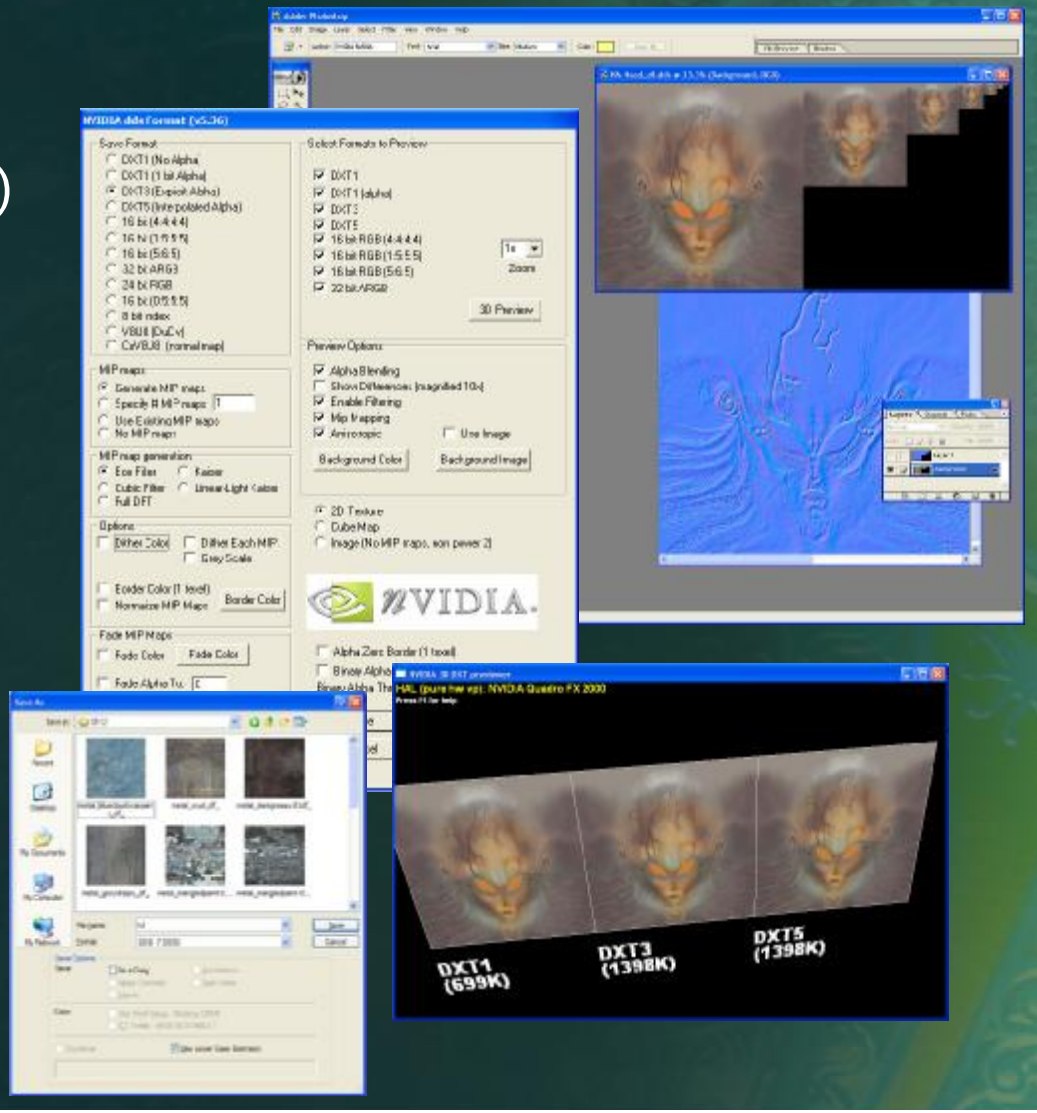


- Graph overlay of various vital statistics
- Metrics reported include:
  - GPU_Idle
  - Driver_Waiting
  - Time_in_Driver
  - Frame_Time
  - AGP / Video memory usage
  - # DIP/DP calls per frame and batch size histogram
- Now external to the driver, supports any Direct3D9 application
- Previously available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



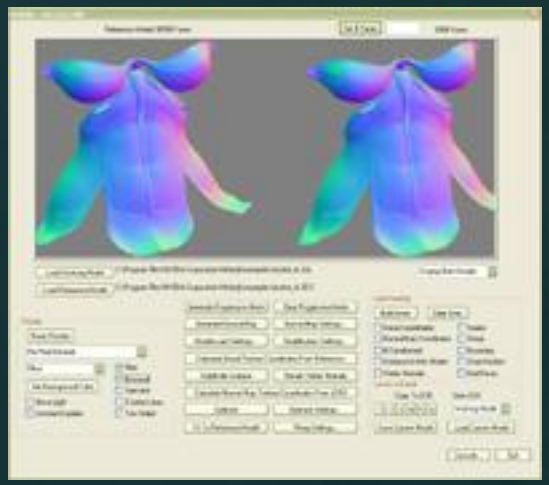
Texture Tools & Plug-ins

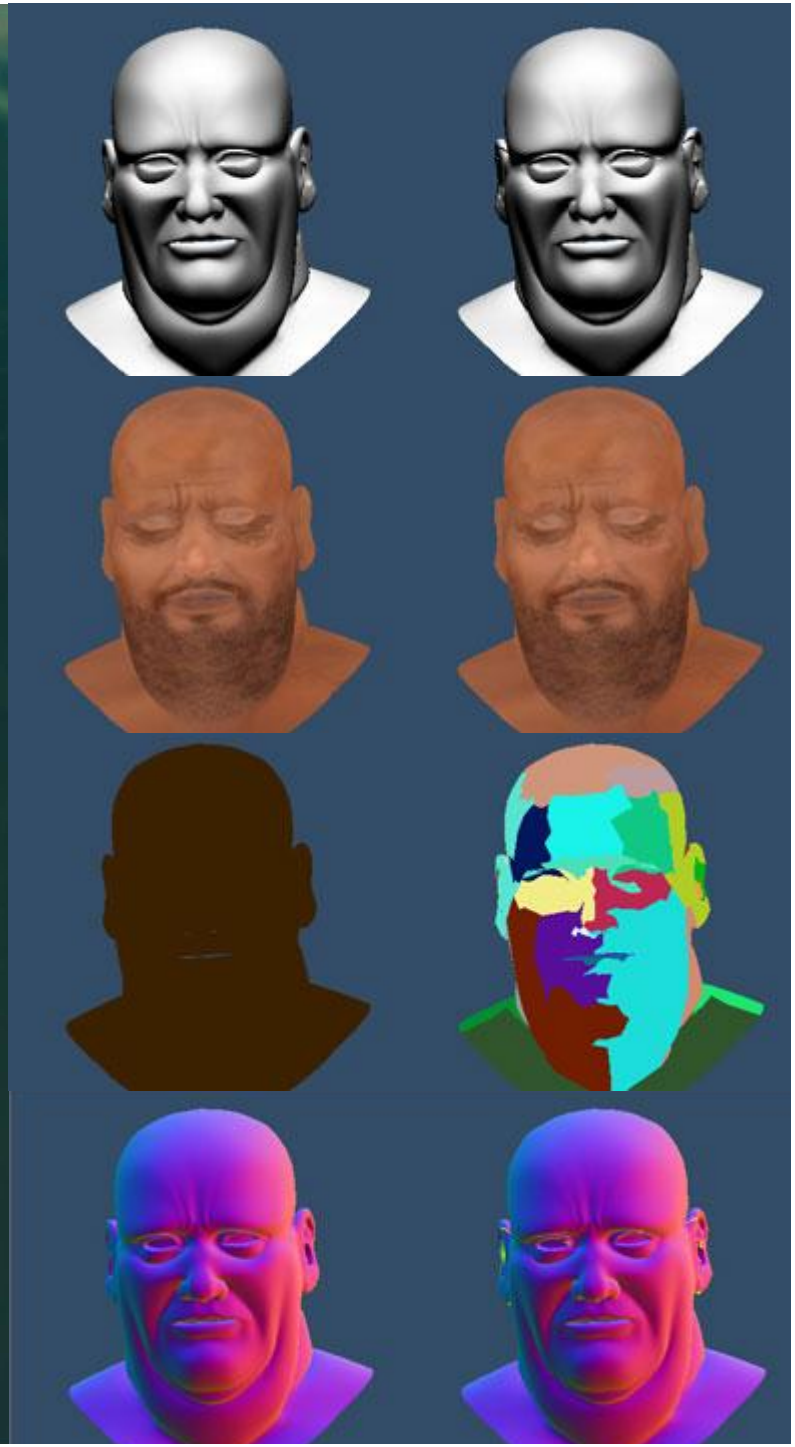
- Photoshop Plug-ins:
  - DXT compression (.dds)
  - Normal Map creation
  - 3D preview and diff
  - MIP map generation
- Command line and .lib
- DDS thumbnail viewer
- Texture Atlas Viewer and Creation Utility

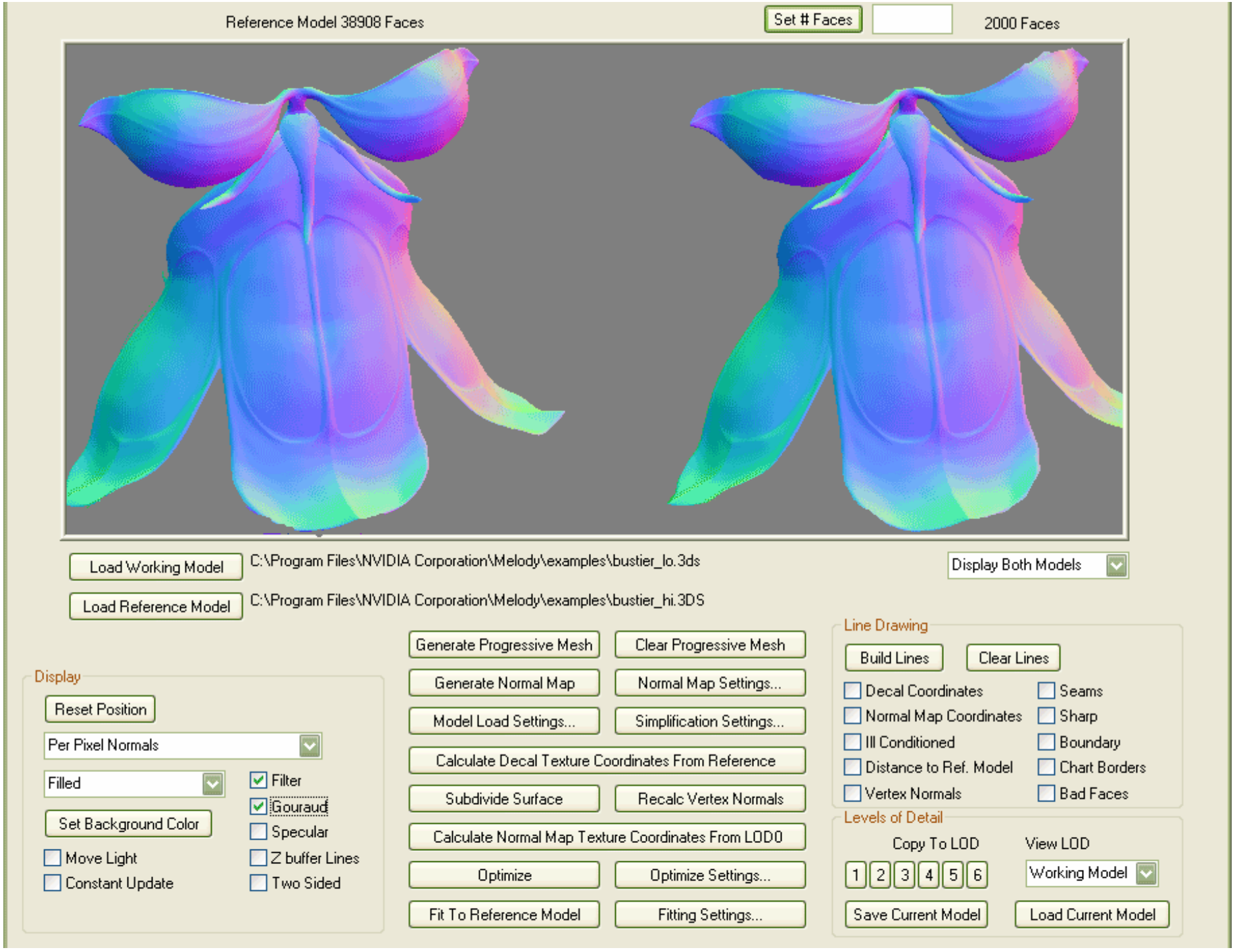


Melody

- 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







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

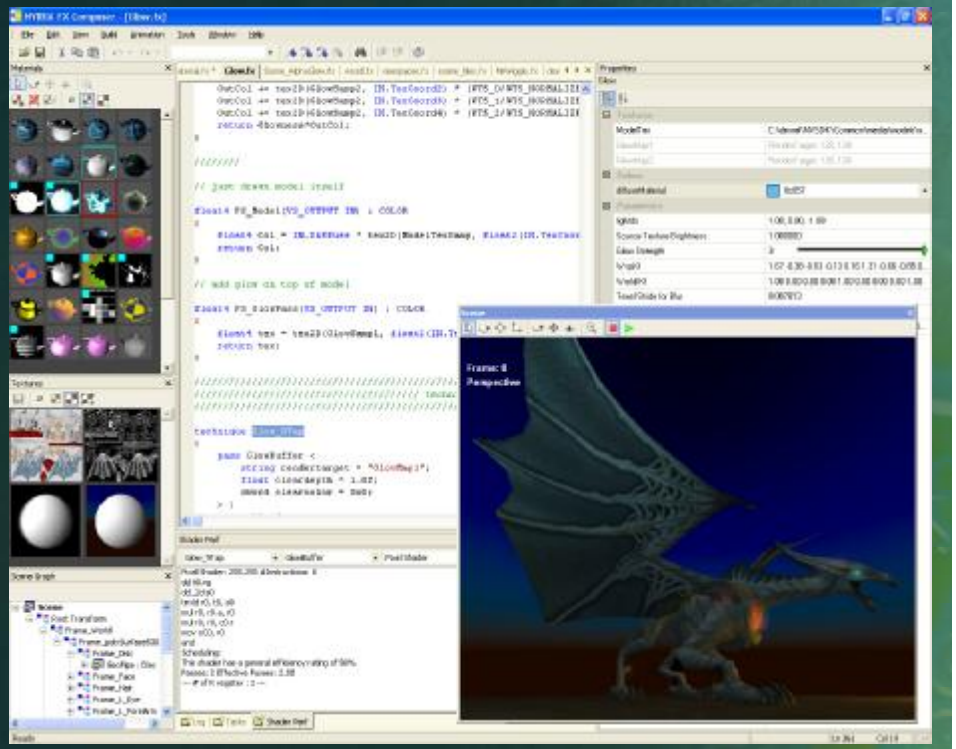
Utilities, libraries and more...

- 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

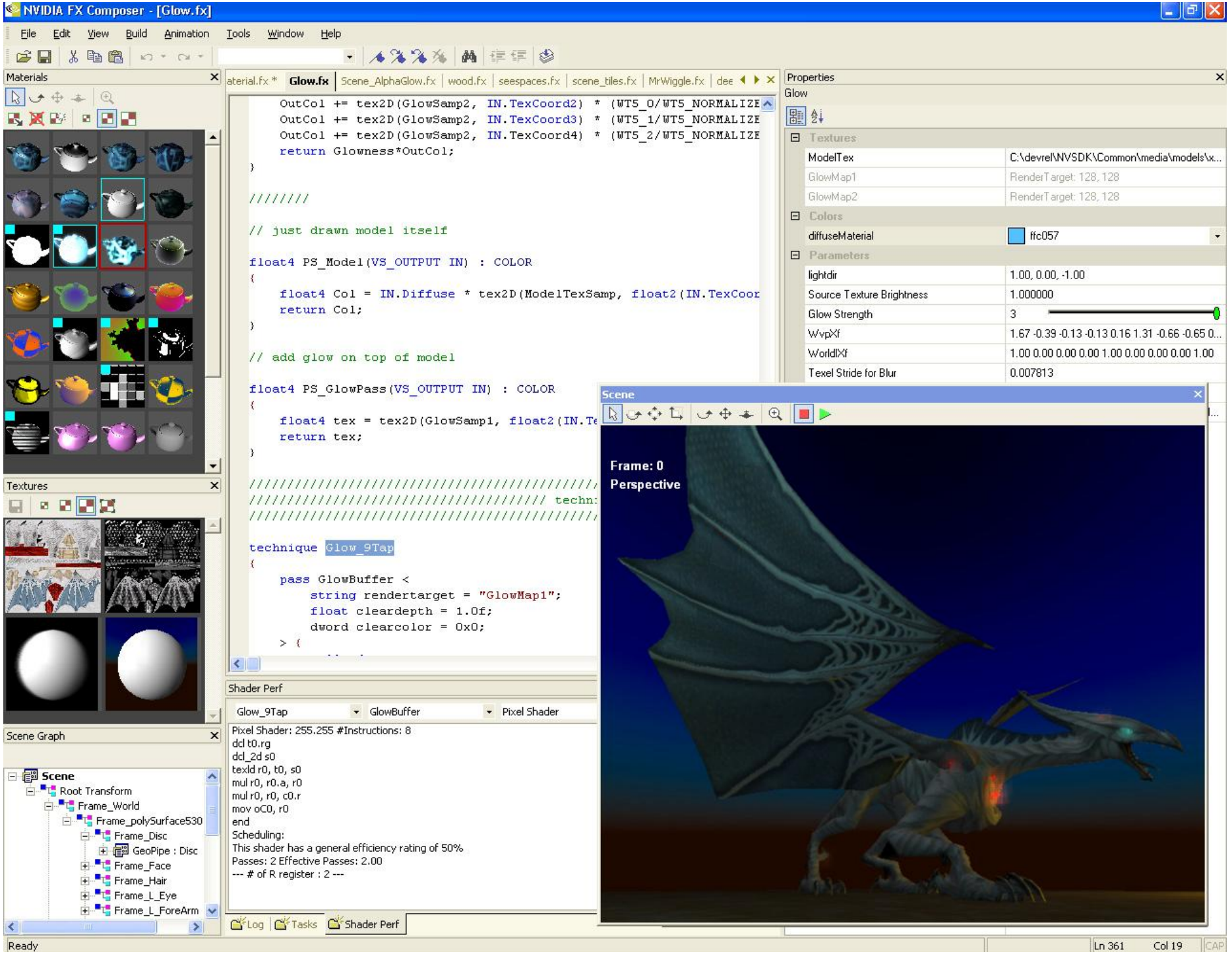
NVIDIA FX Composer

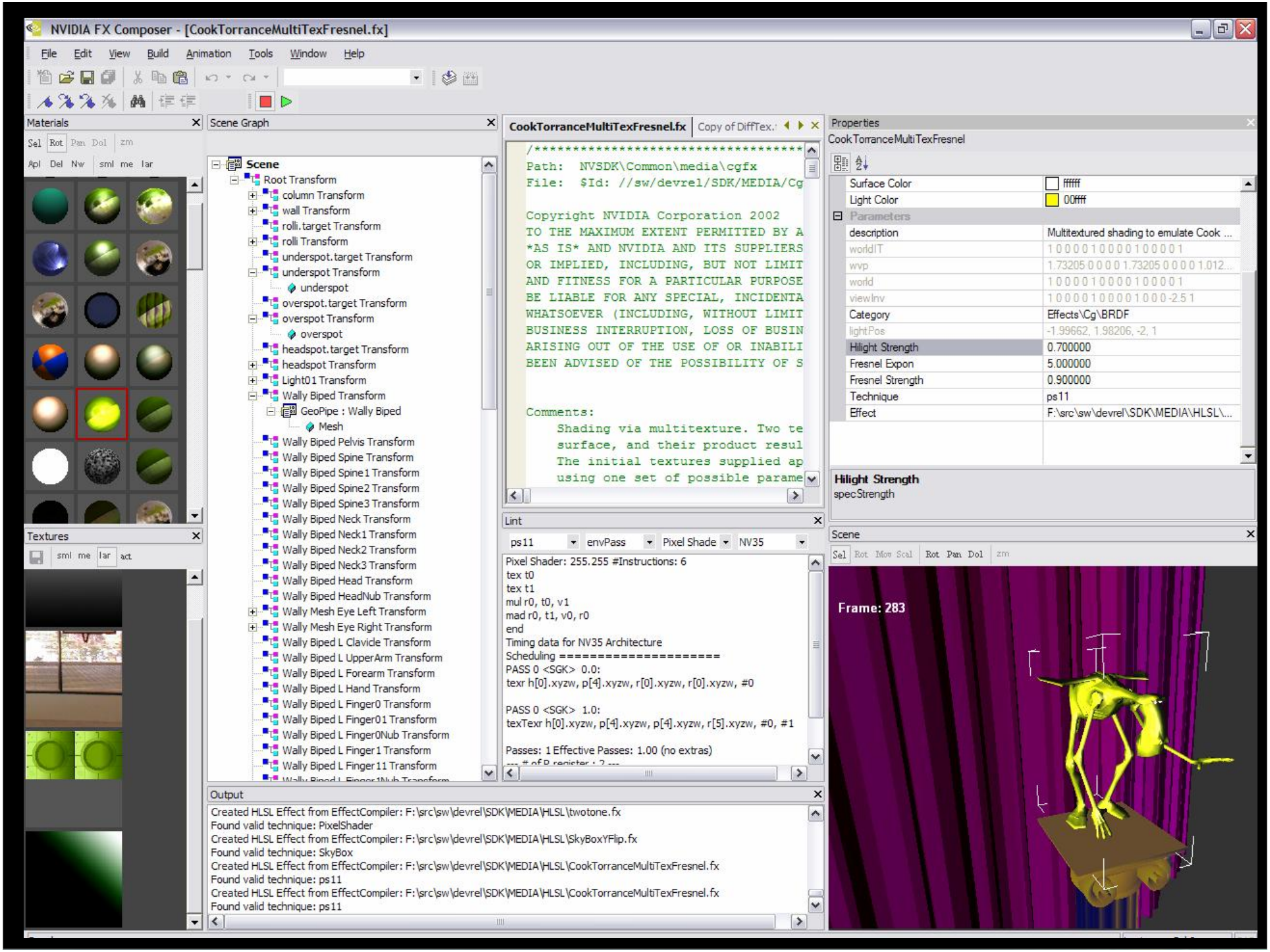
FX Composer empowers developers to create high performance shaders in an integrated development environment with real-time preview & optimization features available only from NVIDIA.

- 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



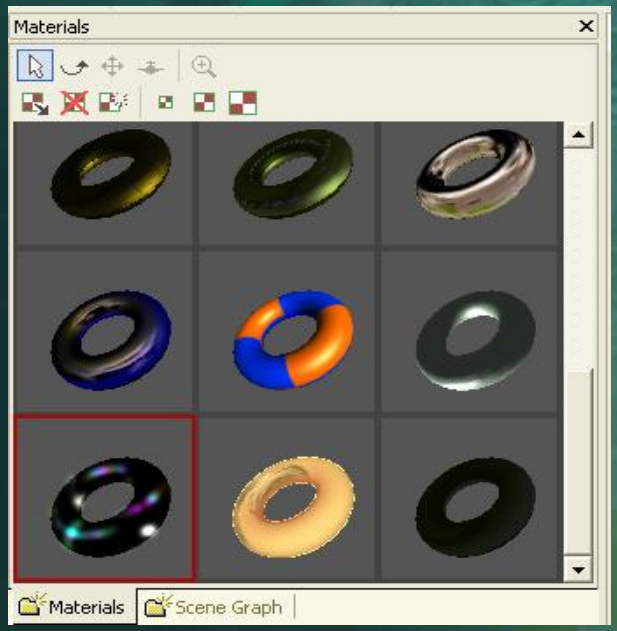
EverQuest® content courtesy Sony Online Entertainment Inc.

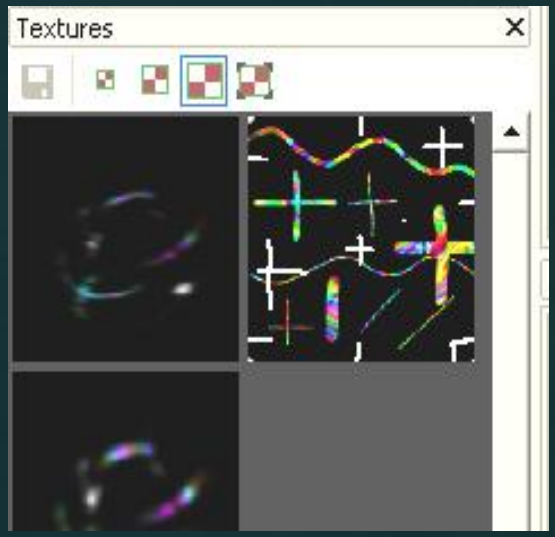




Materials & Textures

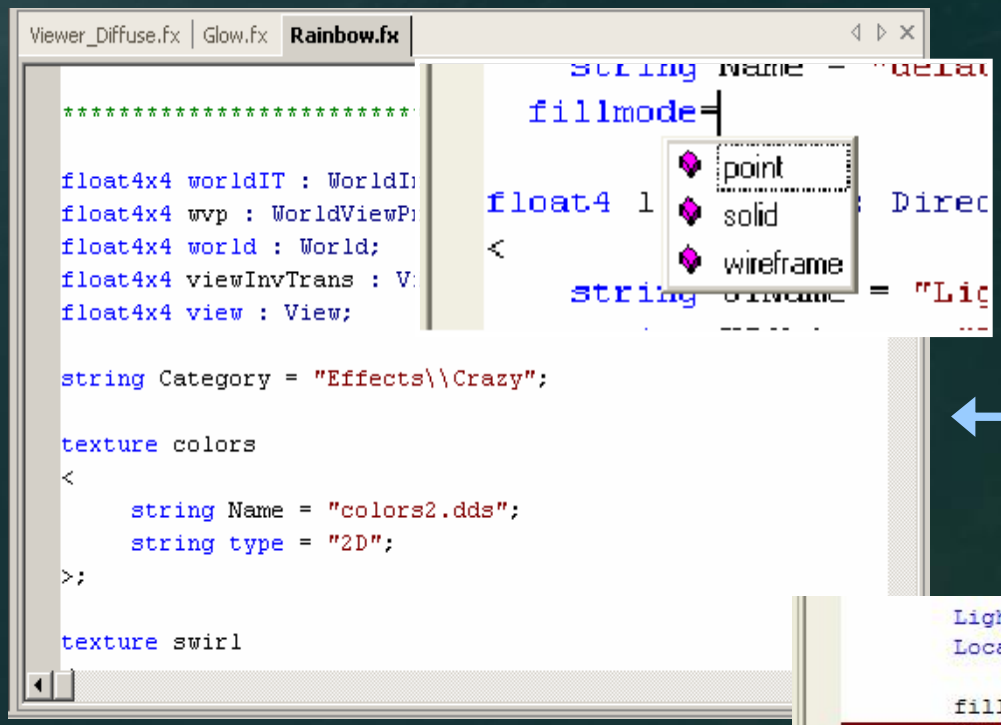
- Preview all the FX files in your scene at the same time
- Apply them to the appropriate parts of your scene in the Scene panel



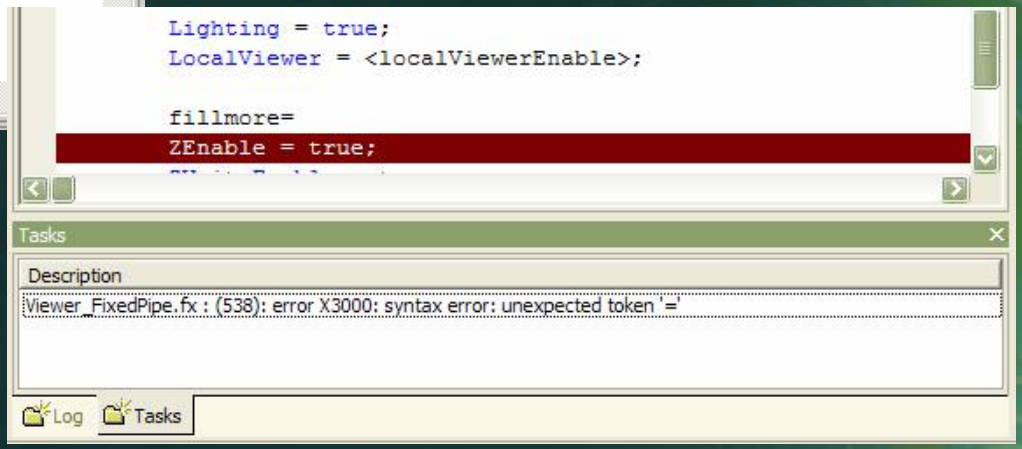


- View source textures
- Preview render targets
- Save any texture to disk!

Editing and Debugging

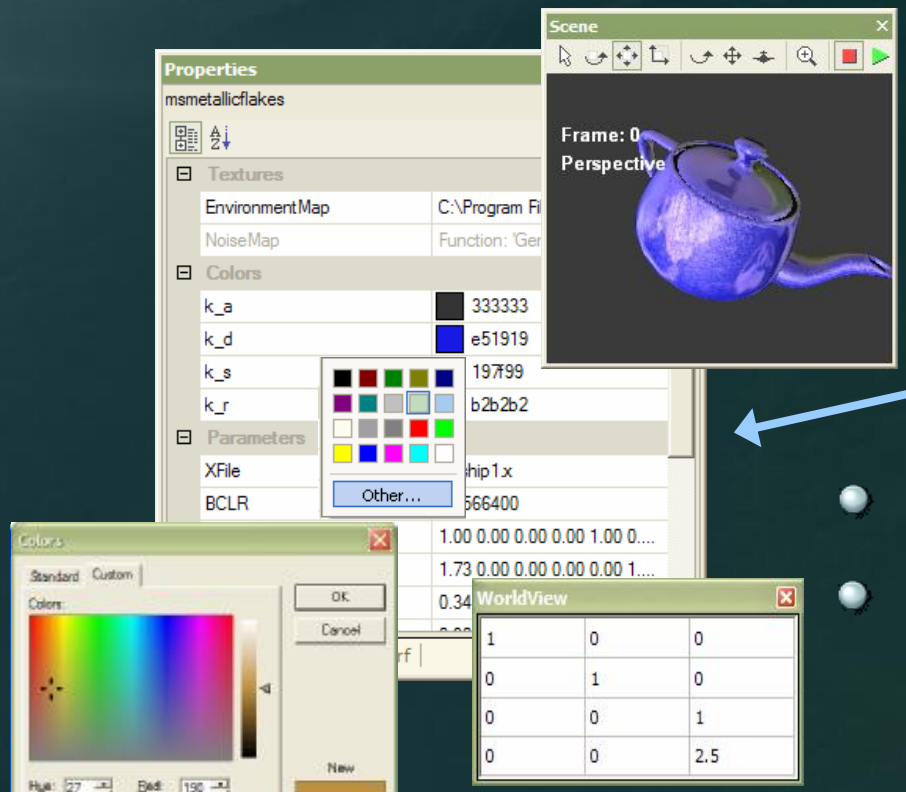


- Edit multiple .FX files
- Intellisense (auto-complete)
- Syntax highlighting

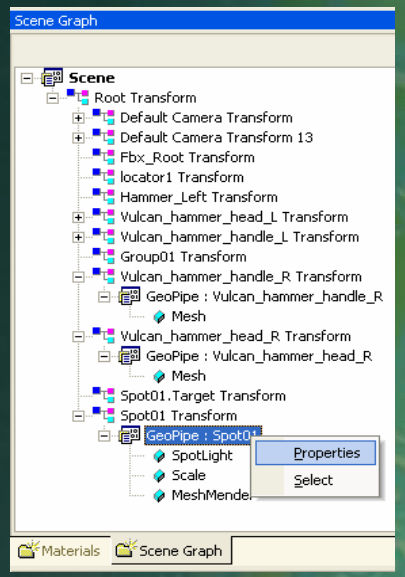


- Jump-to-error helps you find & fix problems quickly

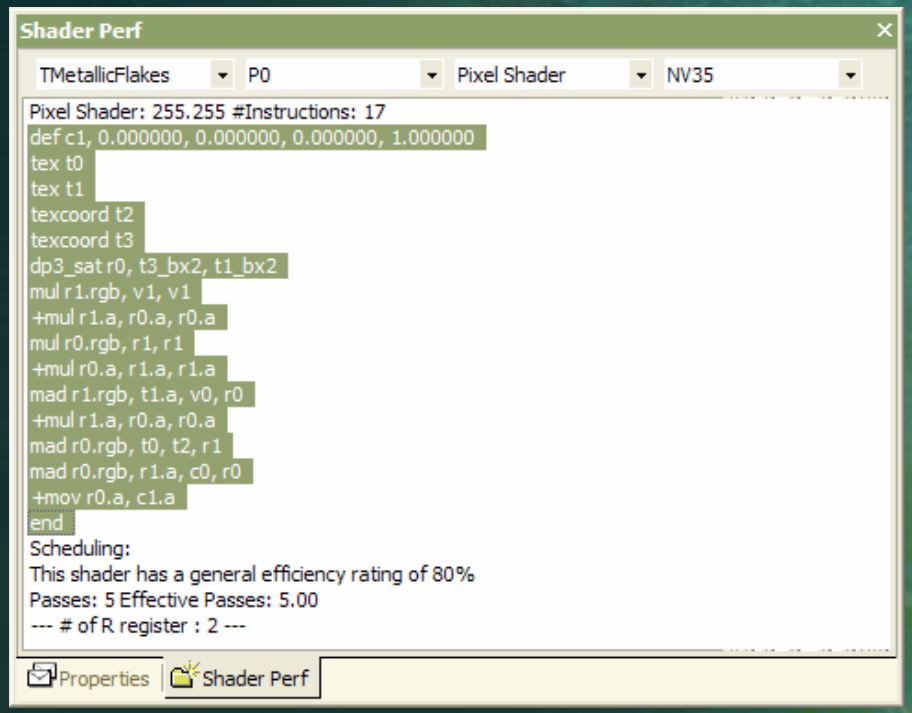
Previewing & Customizing



- 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

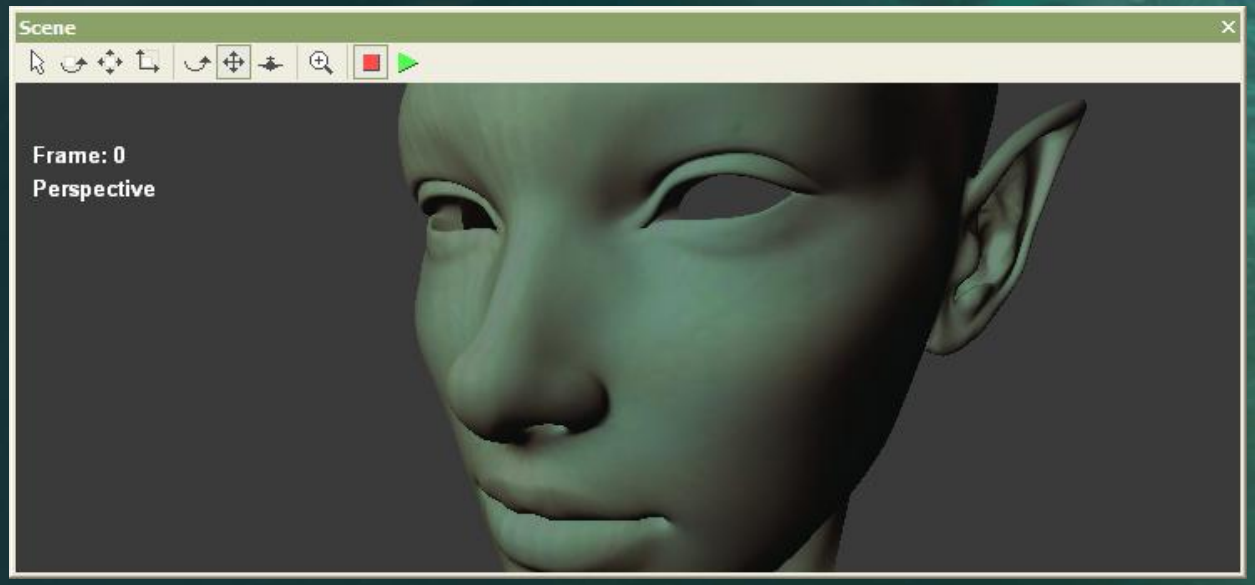


Tune Your Shader Performance



- 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

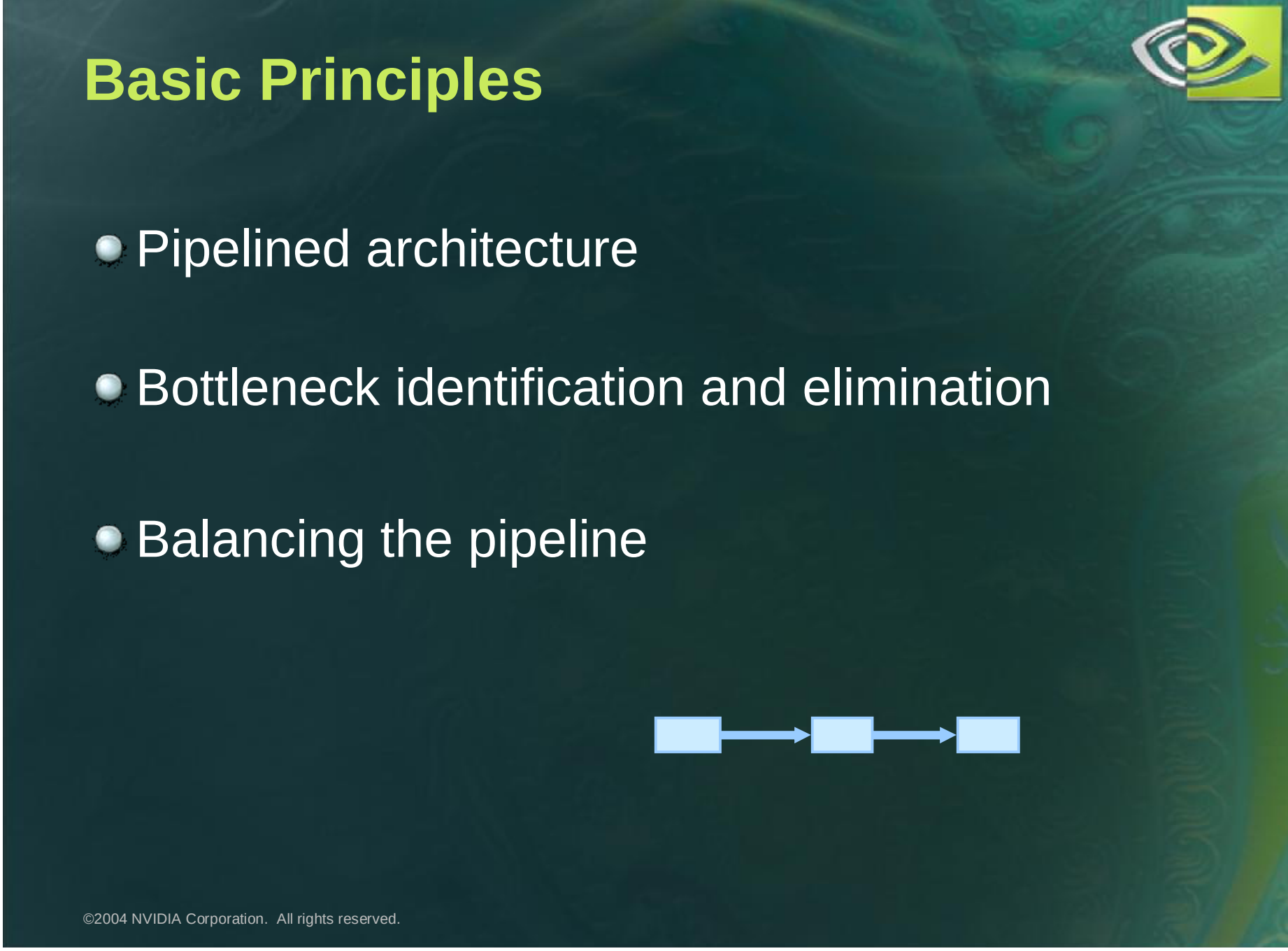
Frequently Asked Questions

- 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

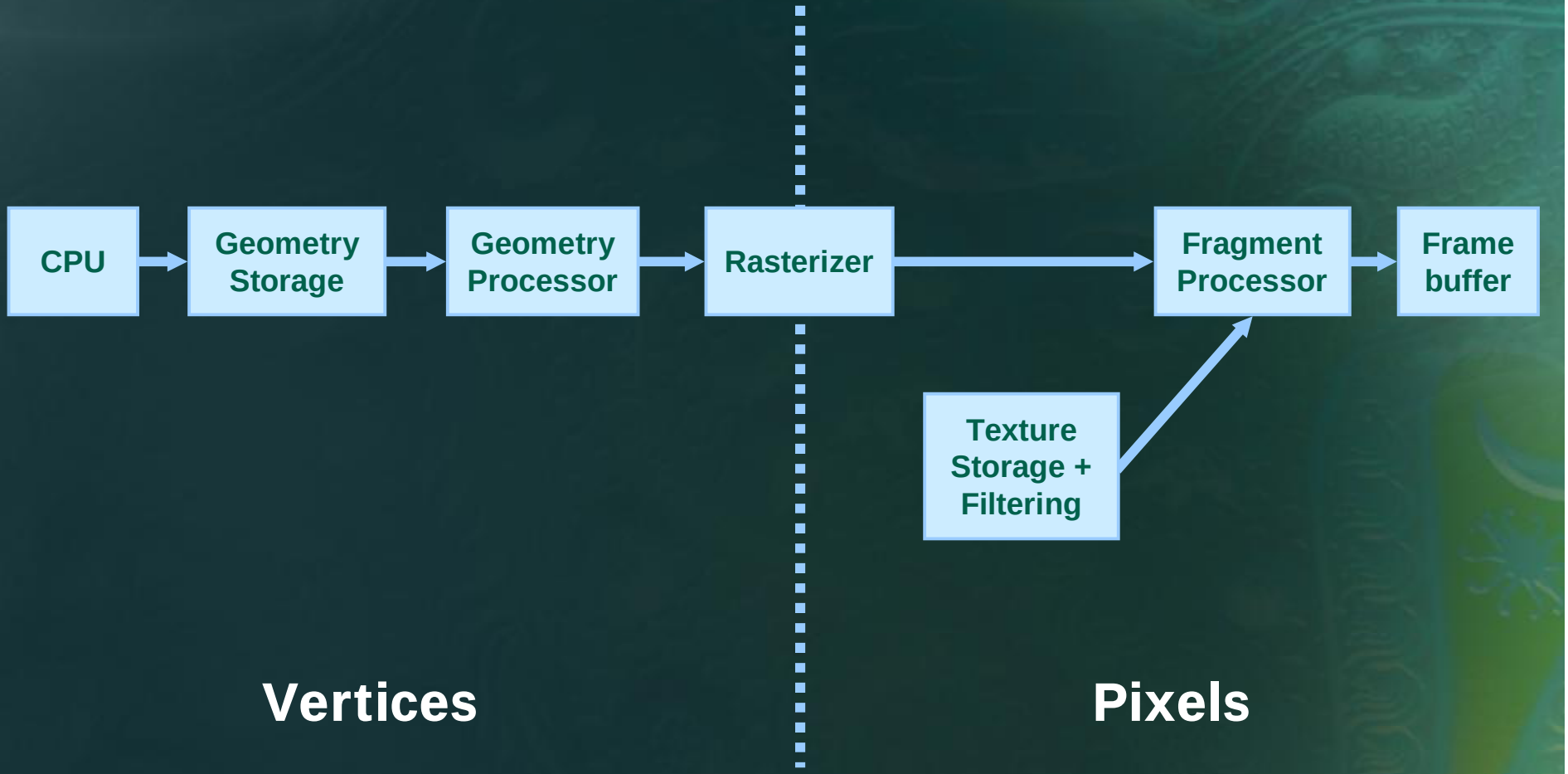
Performance Tuning

Basic Principles

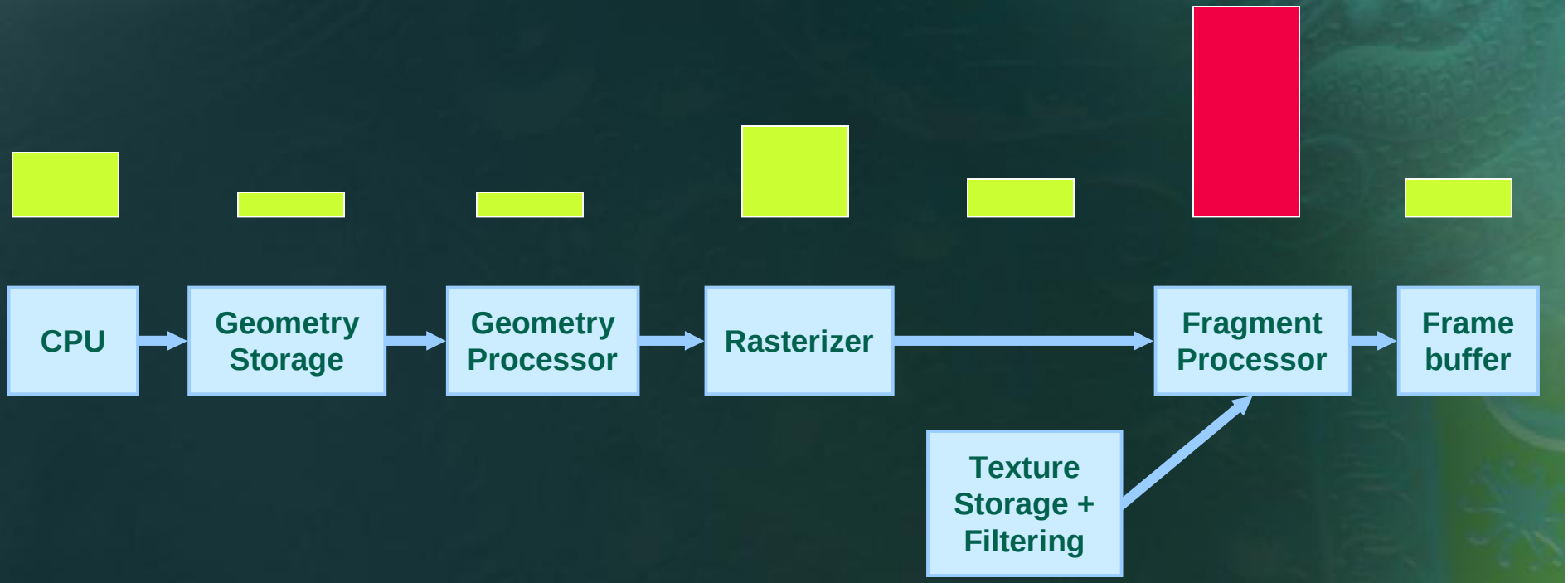


- Pipelined architecture
- Bottleneck identification and elimination
- Balancing the pipeline

Pipelined Architecture



The Terrible Bottleneck

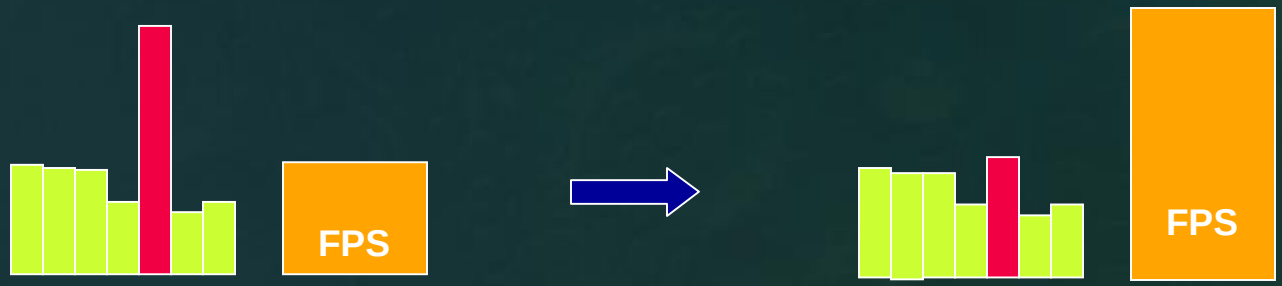


Bottleneck Identification

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

Bottleneck Identification

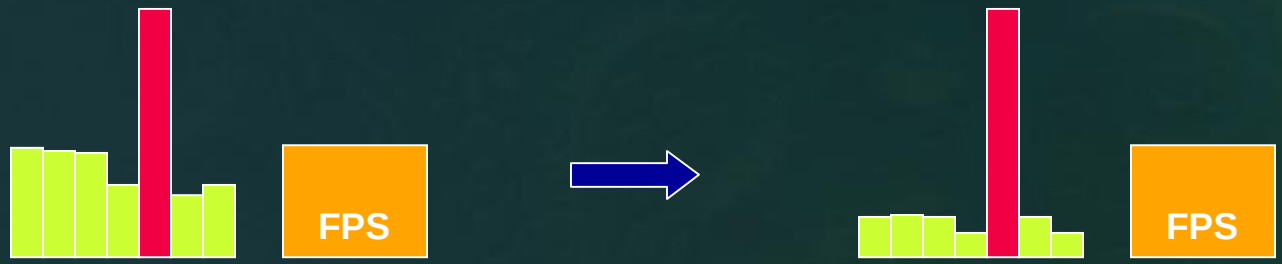
- Modify the stage itself
  - By decreasing its workload



  - 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



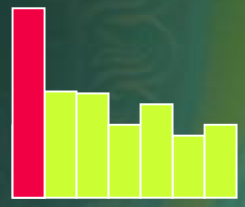
  - If performance doesn't change significantly, then you know this is the bottleneck
  - Careful not to change the workload of this stage!

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

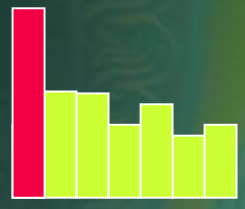
Bottleneck Identification: CPU

- 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



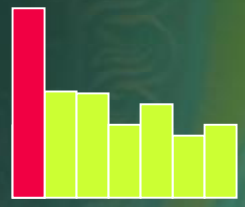
Bottleneck Identification: CPU

- 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



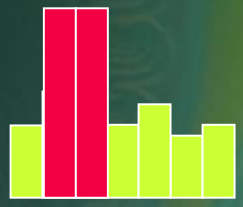
Bottleneck Identification: CPU

- 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



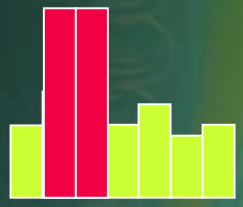
Bottleneck Identification: Vertex

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



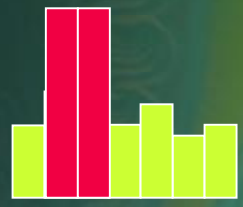
Bottleneck Identification: Vertex

- 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



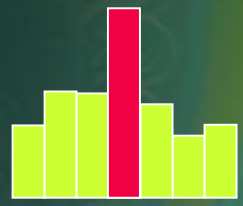
Bottleneck Identification: Vertex

- 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



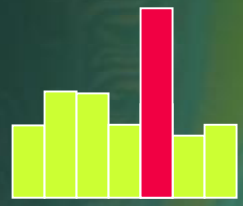
Bottleneck Identification: Raster

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



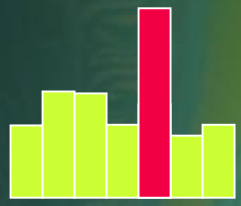
Bottleneck Identification: Texture

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



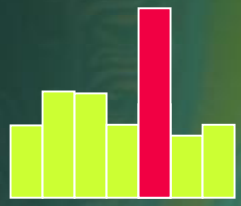
Bottleneck Identification: Texture

- 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



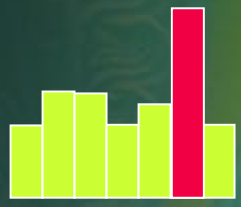
Bottleneck Identification: Texture

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



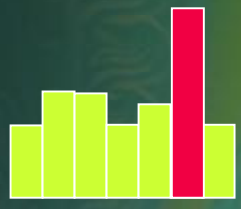
Bottleneck Identification: Fragment

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



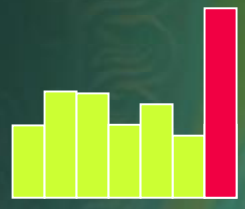
Bottleneck Identification: Fragment

- 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



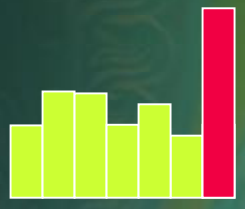
Bottleneck Identification: FB

- 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....

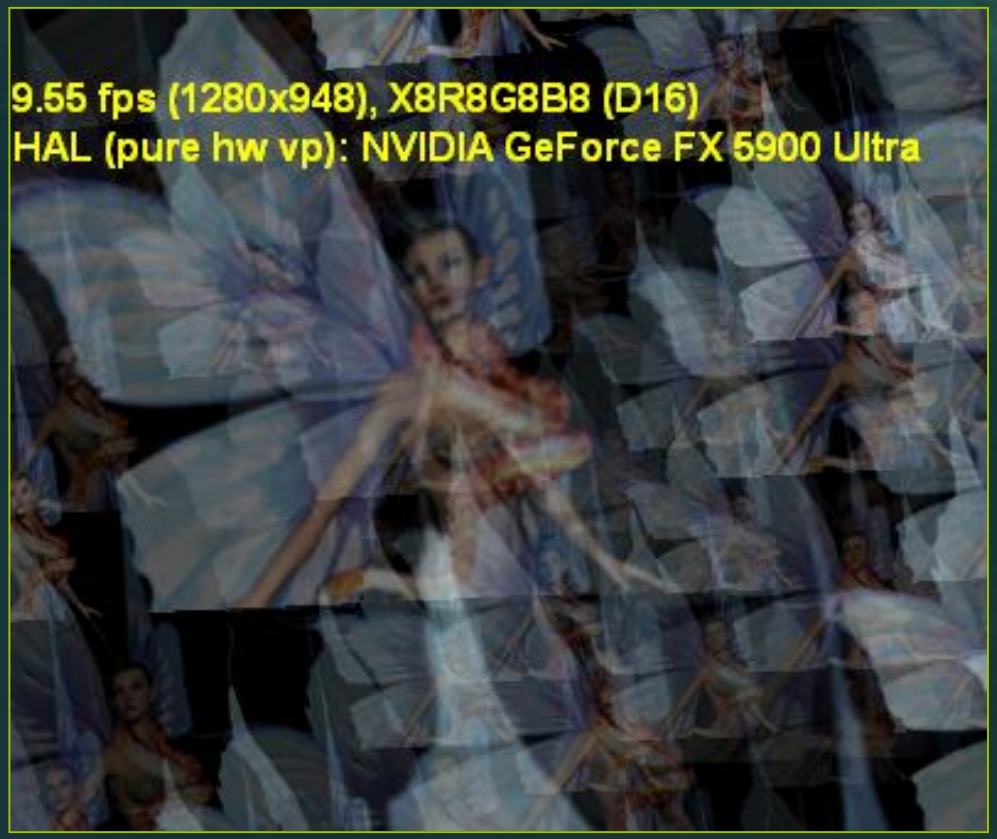
Practice: Example 1

- 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

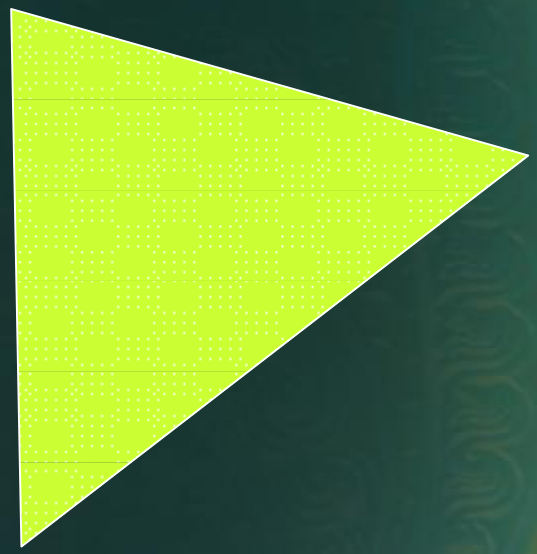
- Texture bandwidth overkill
- Use mipmaps
- Use dxt1 if possible
  - Some cards can store compressed data in cache
- Use smaller textures when they are fine
  - Does the grass blade really need a 1024x1024 texture? --- Maybe

Practice: Example 3

Practice: Example 3

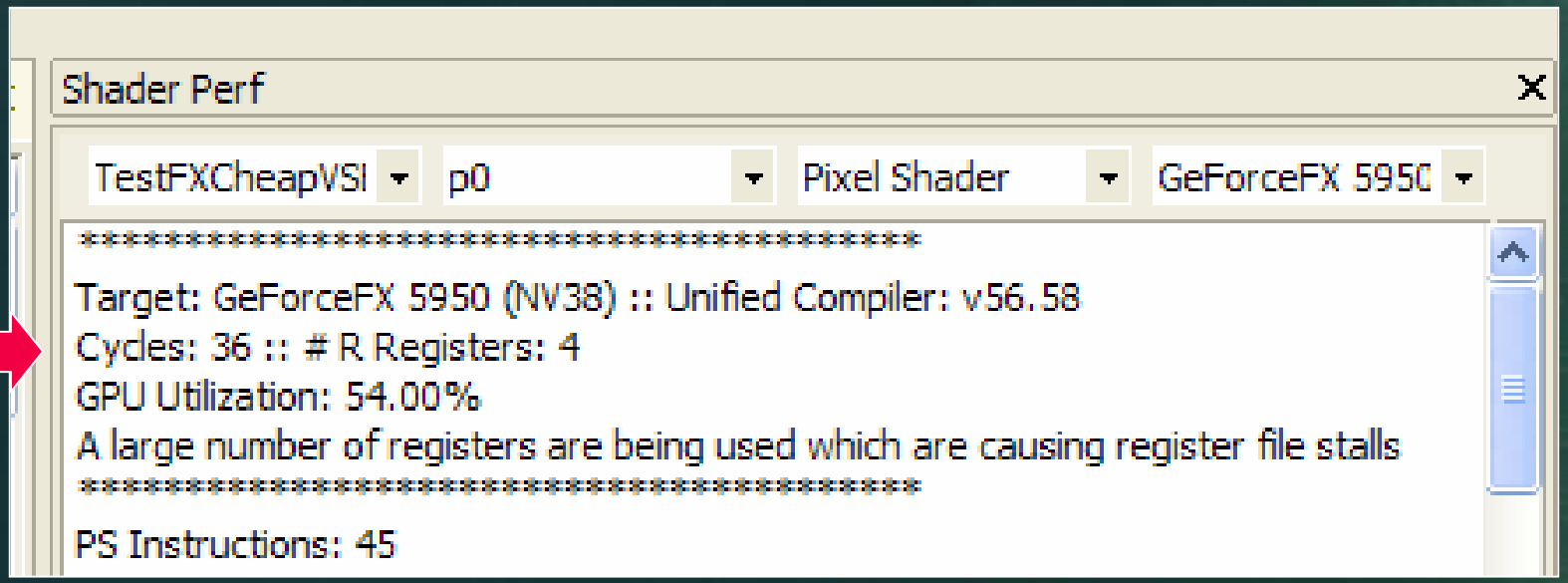
- Expensive pixel shader
- Can have huge performance effect
- Only 3 verts, but maybe a million pixels
  - That's only 1024x1024

Look at all the pixels!!

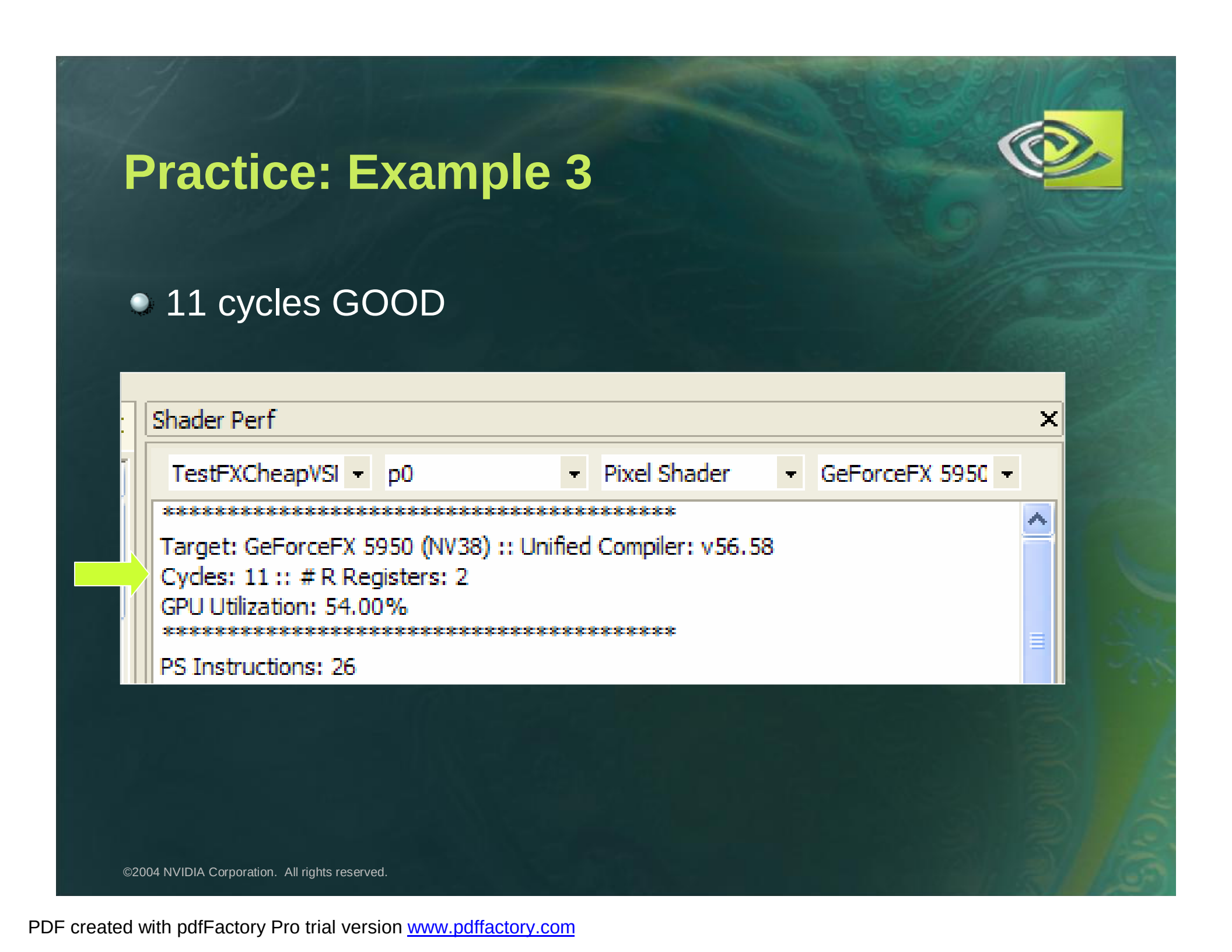


Practice: Example 3

- 36 cycles BAD



Practice: Example 3



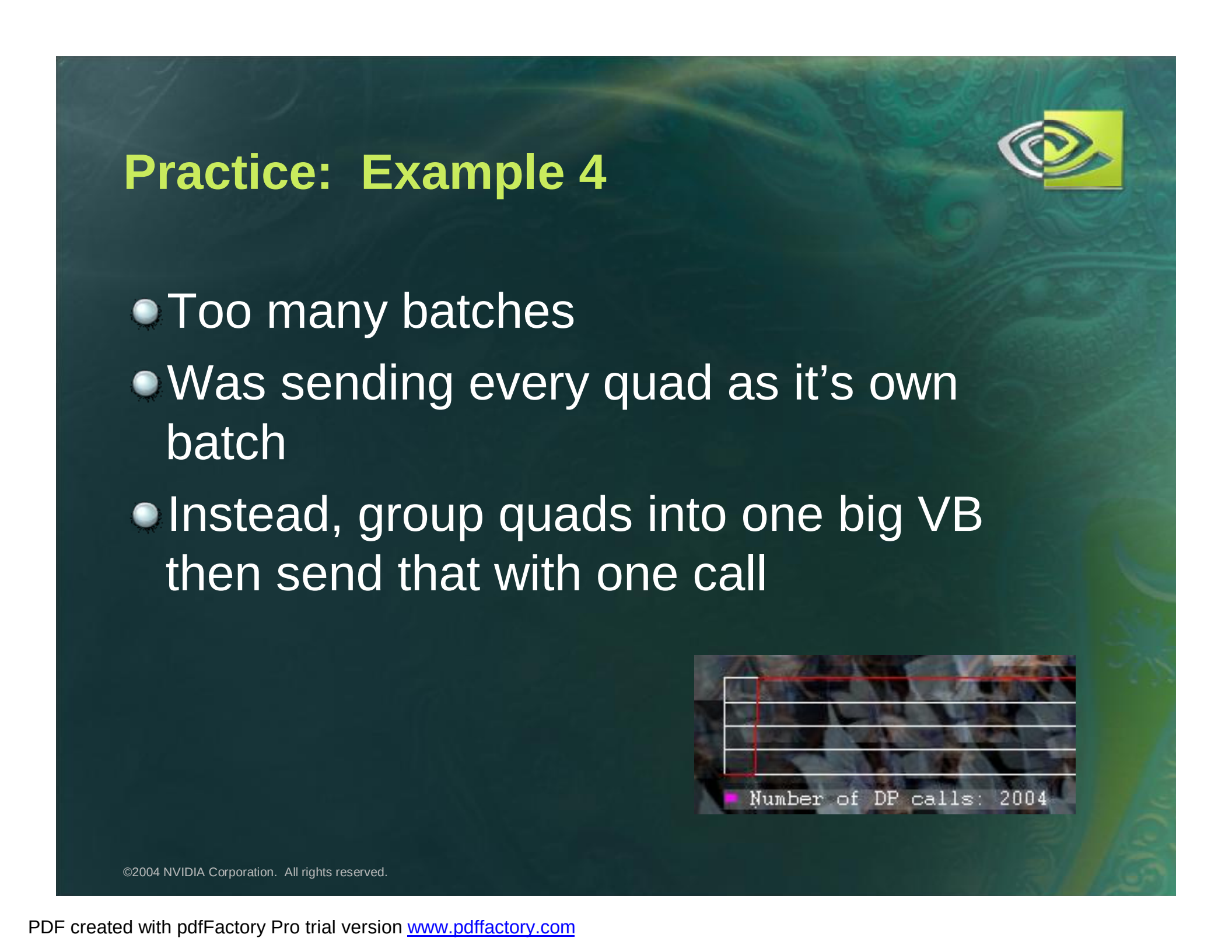
- 11 cycles GOOD

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



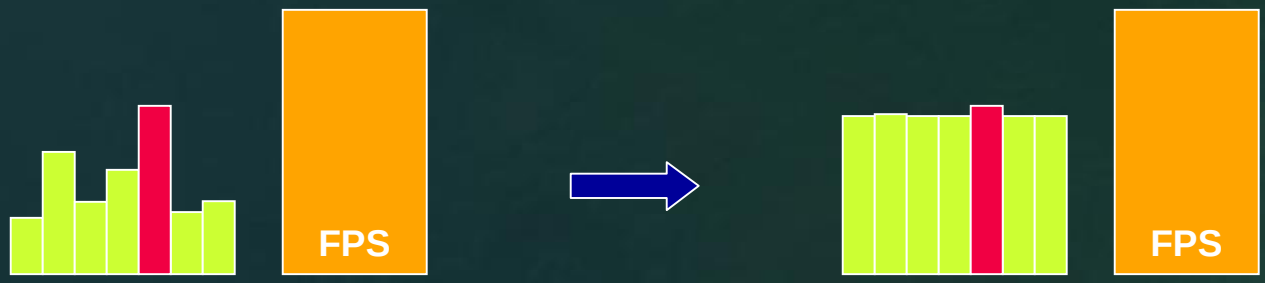
- Too many batches
- Was sending every quad as it's own batch
- Instead, group quads into one big VB then send that with one call

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

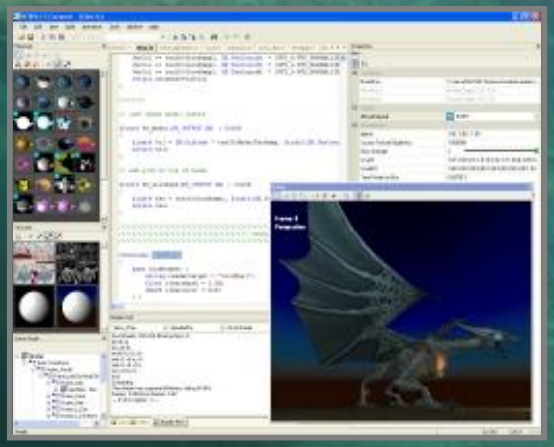
For More Info...

- 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.

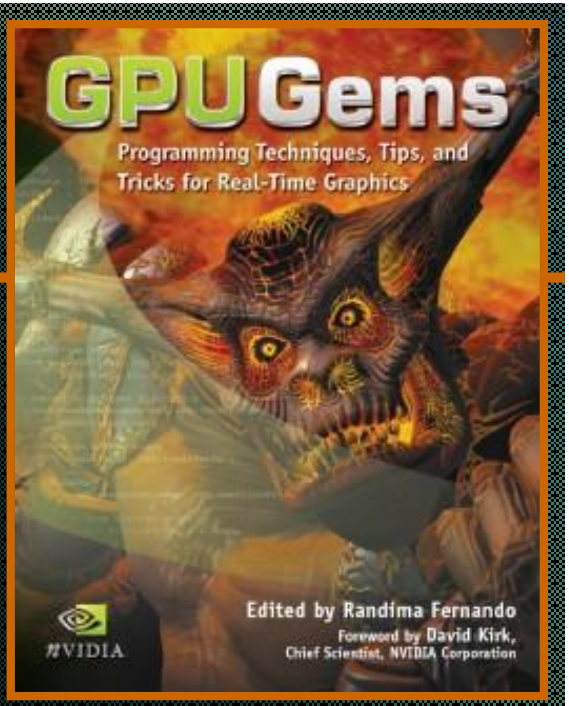
NVIDIA SDK

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

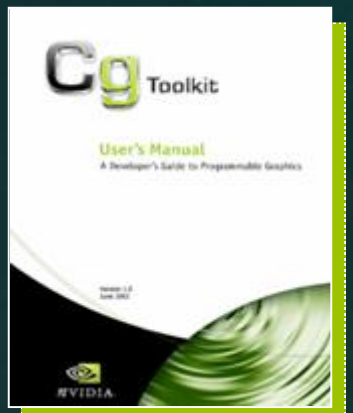
"This collection of articles is particularly impressive for its depth and breadth. The book includes product-oriented case studies, previously unpublished state-of-the-art research, comprehensive tutorials, and extensive code samples and demos throughout."

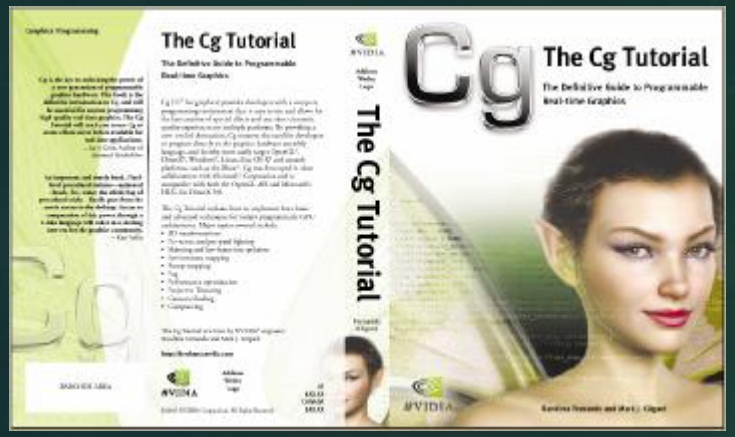
Eric Haines

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)