



**NVIDIA®**

## **Performance Tools**

**Raul Aguaviva and Jeff Kiel (NVIDIA Corporation)**

# Performance Tools Agenda



- GPU architecture at a glance
- Intel VTune: Code Profiling
- NVGLExpert: OpenGL API Assistance
- NVShaderPerf: Shader Performance
- NVPerfKit: Driver and GPU Performance Data
- NVPerfHUD: Interactive Performance Analysis

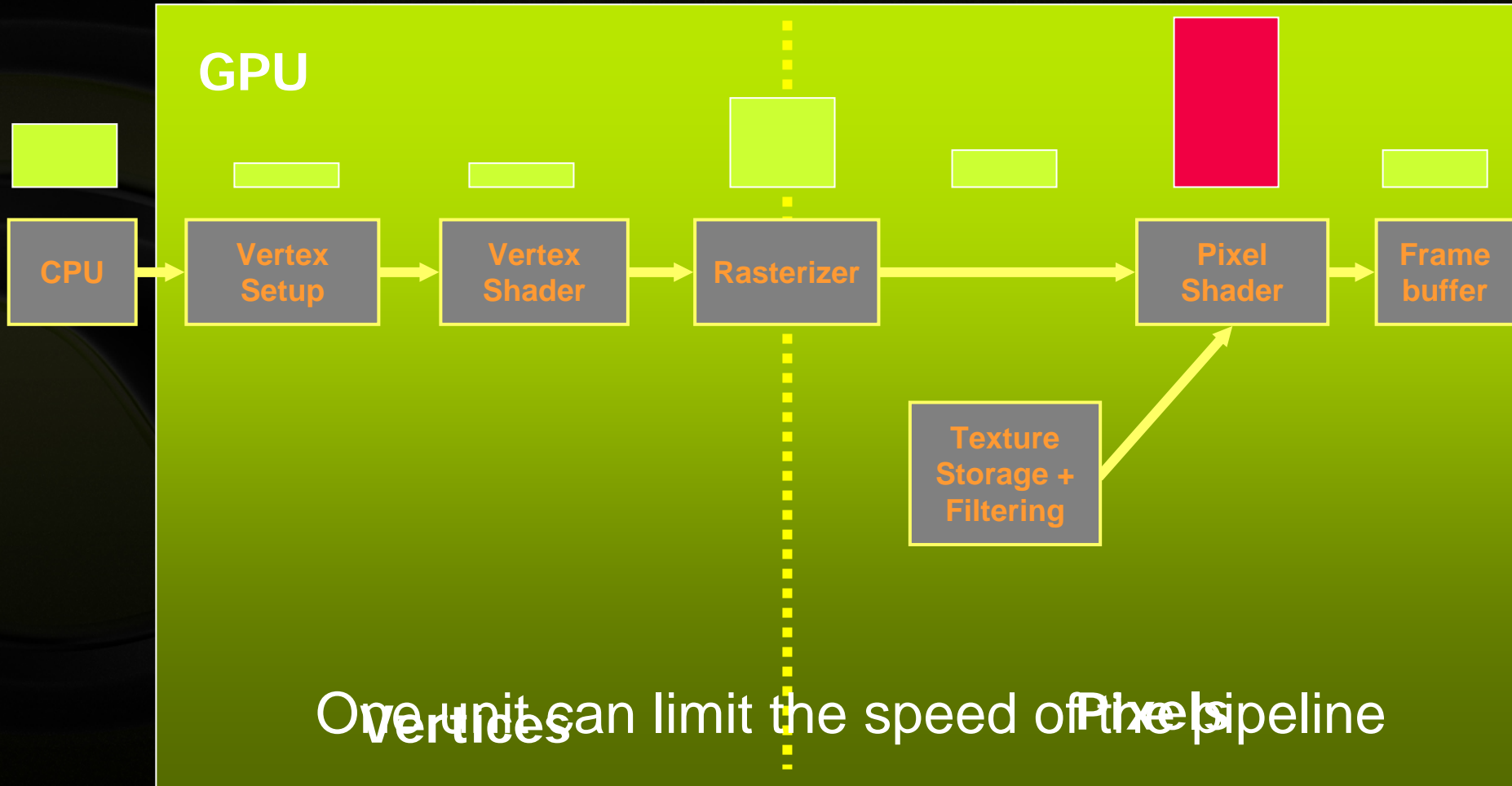
# GPU architecture at a glance



- **Pipelined architecture**
  - Each unit needs the data from the previous unit to do its job
- **Bottleneck identification and elimination**
- **Balancing the pipeline**

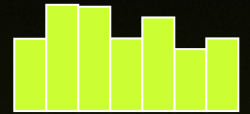


# GPU Pipelined Architecture (simplified view)

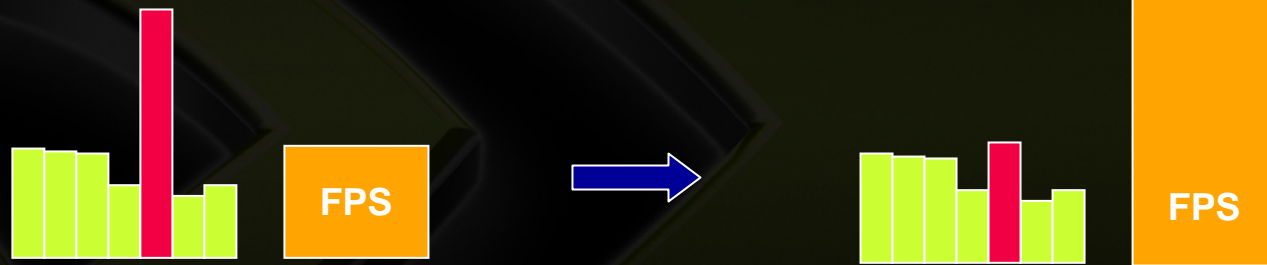




# Bottleneck Identification

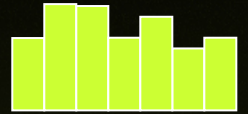


- **Modify the stage itself**
  - By decreasing its workload



- If performance/FPS 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

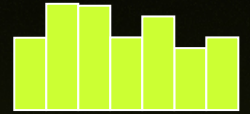


- If performance doesn't change significantly, then you know this is the bottleneck
- Careful not to change the workload of this stage!

# Bottleneck Identification



- Sample counters at different points along the pipeline
- Use NVPerfKit and NVPerfHUD
- How much work performed by each unit, compare to the maximum work possible







**Welcome Gary Carleton from Intel Corporation**



# NVGLExpert



- What is it and what does it do?
- Project status?

# What is it and what does it do?



- **Helps eliminate performance issues on the CPU**
- **Instrumented OpenGL driver**
  - Outputs information to file, console or debugger
  - Different groups and levels of information detail
- **Controlled by small GUI tool**
  - Windows tool sets appropriate registry entries
  - Linux tool sets environment variables
- **What it can do (today)**
  - Prints GL errors when the are raised
  - Indicates if the driver runs through a software fallback
  - Shows unexpected shader compile errors
  - Shows where your VBOs reside
  - Print reasons for `GL_FRAMEBUFFER_UNSUPPORTED_EXT`
- **Feature list will grow with future drivers**



# Project Status



- Will be delivered with next major driver release
- Windows and Linux
- Currently supports NV3x and NV4x architectures
- What types of things are interesting?

[NVGLExpert@nvidia.com](mailto:NVGLExpert@nvidia.com)

# NVShaderPerf



- What is NVShaderPerf?
- What's new with version 1.8?
- What's coming with version 2.0?



```

v2f BumpReflectVS(a2v IN,
    uniform float4x4 WorldViewProj,
    uniform float4x4 World,
    uniform float4x4 ViewIT)
{

```

# NVShaderPerf



```

    v2f OUT;
    // Position in object space
    OUT.Position = mul(IN.Position, WorldViewProj);
    // pass texture coordinates for fetching the normal map
    OUT.TexCoord.xyz = IN.TexCoord;
    OUT.TexCoord.w = 1.0;
    // compute the 4x4 transform from tangent space to object space
    float3x3 TangentToObjSpace;

    // first rows are the tangent and binormal scaled by the bump scale
    TangentToObjSpace[0] = float3(IN.Tangent.x, IN.Binormal.x, IN.Normal.x);
    TangentToObjSpace[1] = float3(IN.Tangent.y, IN.Binormal.y, IN.Normal.y);
    TangentToObjSpace[2] = float3(IN.Tangent.z, IN.Binormal.z, IN.Normal.z);
    OUT.TexCoord[0] = dot(World[0].xyz, TangentToObjSpace[0]);
    OUT.TexCoord[1] = dot(World[1].xyz, TangentToObjSpace[0]);
    OUT.TexCoord[2] = dot(World[2].xyz, TangentToObjSpace[0]);
    OUT.TexCoord[3] = dot(World[0].xyz, TangentToObjSpace[1]);
    OUT.TexCoord[4] = dot(World[1].xyz, TangentToObjSpace[1]);
    OUT.TexCoord[5] = dot(World[2].xyz, TangentToObjSpace[1]);
    OUT.TexCoord[6] = dot(World[0].xyz, TangentToObjSpace[2]);
    OUT.TexCoord[7] = dot(World[1].xyz, TangentToObjSpace[2]);
    OUT.TexCoord[8] = dot(World[2].xyz, TangentToObjSpace[2]);
    float4 worldPos = mul(IN.Position, World);
    // compute eye vector (going from shaded point to eye) in cube space
    float4 eyeVec = mul(worldPos - ViewIT[3]; // view inv. transpose contains eye position in world space
    OUT.TexCoord[9] = eyeVec.x;
    OUT.TexCoord[10] = eyeVec.y;
    OUT.TexCoord[11] = eyeVec.z;
    return OUT;
}

```

- Inputs
- HLSL (fragments)
- GLSL (fragments)
- !FP1.0
- ARBfp1.0
- PS1.x, PS2.x, PS3.x
- VS1.x, VS2.x, VS3.x
- Cg

## NVShaderPerf

GPU Arch:

- GeForce 7800 GTX
- GeForce 6X00, FX series
- Quadro FX series

C:\WINDOWS\system32\cmd.exe

```

dp3 r0.x, r1, r1
rsq r0.w, r0.x
nrm r0.xyz, t1
mad r1.xyz, r1, r0.w, r0
nrm r2.xyz, r1
nrm r1.xyz, t2
dp3 r2.x, r2, r1
max r1.w, r2.x, c9.x
pow r0.w, r1.w, c5.x
add r1.w, r0.w, -c7.x
mov r2.w, c6.x
add r2.w, r2.w, -c7.x
rcp r2.w, r2.w
mul_sat r2.w, r1.w, r2.w
mad r1.w, r2.w, c9.y, c9.z
mul r2.w, r2.w, r2.w
mul r1.w, r1.w, r2.w
mov r2.x, c9.w
add r2.w, r2.x, -c8.x
mad r1.w, r1.w, r2.w, c8.x
dp3 r0.x, r0, r1
mul r0.w, r0.w, r1.w

```

## Outputs:

- Resulting assembly code
- # of cycles
- # of temporary registers
- Pixel throughput
- Test all fp16 and all fp32

```

Target: GeForce 6800 Ultra (NV40) :: Unified Compiler: v61.7
Cycles: 14.00 :: R Regs Used: 2 :: R Regs Max Index <0 based>
Pixel throughput (assuming 1 cycle texture lookup) 304.76 M
=====
Shader performance using all FP16
Cycles: 14.00 :: R Regs Used: 2 :: R Regs Max Index <0 based>
Pixel throughput (assuming 1 cycle texture lookup) 457.14 M
=====
Shader performance using all FP32
Cycles: 21.00 :: R Regs Used: 3 :: R Regs Max Index <0 based>
Pixel throughput (assuming 1 cycle texture lookup) 304.76 M
C:\Temp\NVShaderPerf_61_77>

```

# NVShaderPerf: In your pipeline



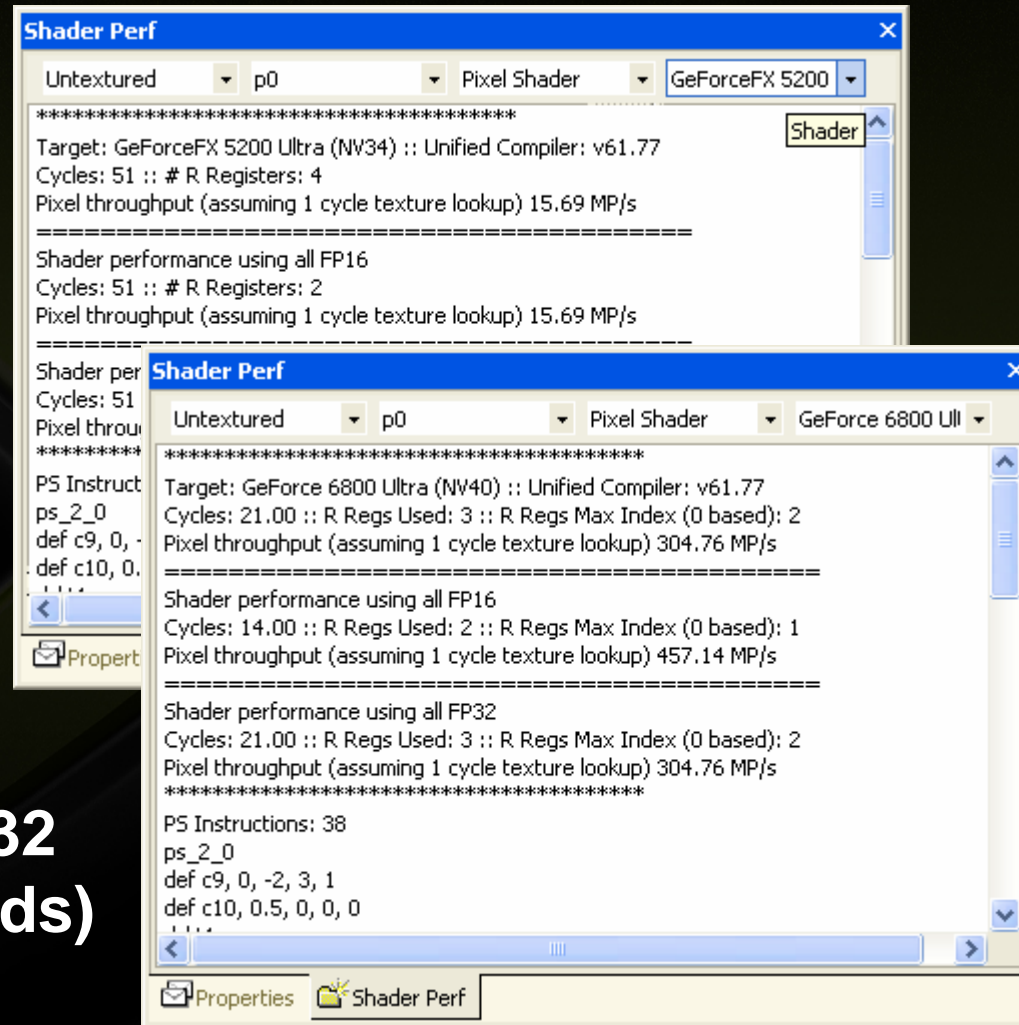
- **Test current performance**
  - against shader cycle budgets
  - test optimization opportunities
- **Automated regression analysis**
- **Integrated in FX Composer 1.7**



# FX Composer 1.7 – Shader Perf



- Disassembly
- Target GPU
- Driver version match
- Number of Cycles
- Number of Registers
- Pixel Throughput
- Forces all fp16 and all fp32 (gives performance bounds)



# NVShaderPerf 1.8



- **Support for GeForce 7800 GTX and Quadro FX 4500**
- **Unified Compiler from ForceWare 77.72 driver**
- **Better support for branching performance**
  - **Default computes maximum path through shader**
  - **Use `-minbranch` to compute minimum path**



# NVShaderPerf 1.8



```
////////////////////////////////////  
// determine where the iris is and update normals, and lighting parameters to simulate iris geometry  
////////////////////////////////////
```

```
float3 objCoord = objFlatCoord;  
float3 objBumpNormal = normalize( f3tex2D( g_eyeNormal, v2f.UVtex0 ) * 2.0 - float3( 1, 1, 1 ) );  
objBumpNormal = 0.350000 * objBumpNormal + ( 1 - 0.350000 ) * objFlatNormal;  
half3 diffuseCol = h3tex2D( g_irisWhiteMap, v2f.UVtex0 );  
float specExp = 20.0;  
half3 specularCol = h3tex2D( g_eyeSpecMap, v2f.UVtex0 ) * g_specAmount;
```

```
float tea;
```

```
float3 centerToSurfaceVec = objFlatNormal; // = normalize( v2f.objCoord )  
float firstDot = centerToSurfaceVec.y; // = dot( ce  
if( firstDot > 0.805000 )
```

```
{  
    // We hit the iris. Do the math.
```

```
    // we start with a ray from the eye to the surface  
    float3 ray_dir = normalize( v2f.objCoord - objEye  
    float3 ray_origin = v2f.objCoord;
```

```
    // refract the ray before intersecting with the iris  
    ray_dir = refract( ray_dir, objFlatNormal, g_refra
```

```
    // first, see if the refracted ray would leave the e  
    float t_eyeballSurface = SphereIntersect( 16.0, r  
    float3 objPosOnEyeBall = ray_origin + t_eyeball  
    float3 centerToSurface2 = normalize( objPosOn
```

```
if( centerToSurface2.y > 0.805000 )
```

```
{  
    // Display a blue color  
    diffuseCol = float3( 0, 0, 0.7 );  
    objBumpNormal = objFlatNormal;  
    specularCol = float3( 0, 0, 0 );  
    specExp = 10.0;
```

```
}  
else
```

```
{  
    // transform into irisSphere space  
    ray_origin.y -= 5.109000;
```

```
    // intersect with the Iris sphere  
    float t = SphereIntersect( 9.650000, ray_origin, ray_dir );  
    float3 SphereSpaceIntersectCoord = ray_origin + t * ray_dir;  
    float3 irisNormal = normalize( -SphereSpaceIntersectCoord );
```

Eye Shader from Luna

Maximum branch takes 674 cycles

Minimum branch takes 193 cycles.

```
C:\WINDOWS\System32\cmd.exe  
T:\tmp>t:\sw\devrel\sdk\tools\bin\release_pdb\nvshperf\nvshaderperf -a NU40 cornea2.txt  
-----  
Running performance on file Cornea2.txt  
-----  
Target: GeForce 6800 Ultra <NU40> :: Unified Compiler: v77.72  
Cycles: 674.25 :: R Regs Used: 12 :: R Regs Max Index <0 based>: 11  
Pixel throughput <assuming 1 cycle texture lookup> 9.50 MP/s  
T:\tmp>t:\sw\devrel\sdk\tools\bin\release_pdb\nvshperf\nvshaderperf -minbranch -a NU40 cornea2.txt  
-----  
Running performance on file Cornea2.txt  
-----  
Target: GeForce 6800 Ultra <NU40> :: Unified Compiler: v77.72  
Cycles: 192.82 :: R Regs Used: 12 :: R Regs Max Index <0 based>: 11  
Pixel throughput <assuming 1 cycle texture lookup> 33.33 MP/s  
T:\tmp>_
```

# NVShaderPerf – version 2.0



- Vertex throughput
- GLSL vertex program
- Multiple driver versions from one NVShaderPerf
- What else do you need?

[NVShaderPerf@nvidia.com](mailto:NVShaderPerf@nvidia.com)



# NVPerfKit



- What is NVPerfKit?
- Associated Tools
- NVPerfKit 2.0

# NVPerfKit: The Solution!



- Why is my app running at 13FPS after CPU tuning?
- How can I determine what is going in that GPU?
- How come IHV engineers are able to figure it out?



# What is NVPerfKit?



- **Driver and GPU performance counters**
  - Performance Data Helper (PDH)
  - Microsoft PIX for Windows
- **NVPerfHUD functionality inside any application**
- **Application triggered sampling**
- **OpenGL and Direct3D**

# NVPerfKit: What it looks like...





# What is in the NVPerfKit package?



## ● Instrumented Driver

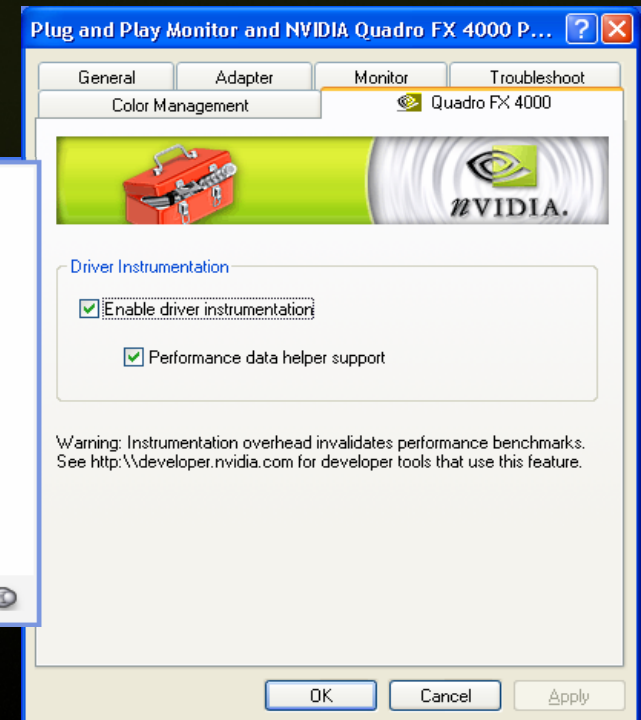
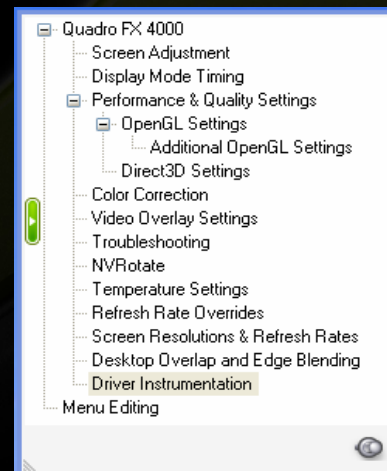
- Exposes GPU and Driver Performance Counters
- Supports OpenGL and Direct3D
- Supports SLI Counters

## ● Tools

- NVDevCPL
- PIX Plugin
- NVAppAuth

## ● SDK

- Sample Code
- Helper Classes
- Docs





# OpenGL Signals



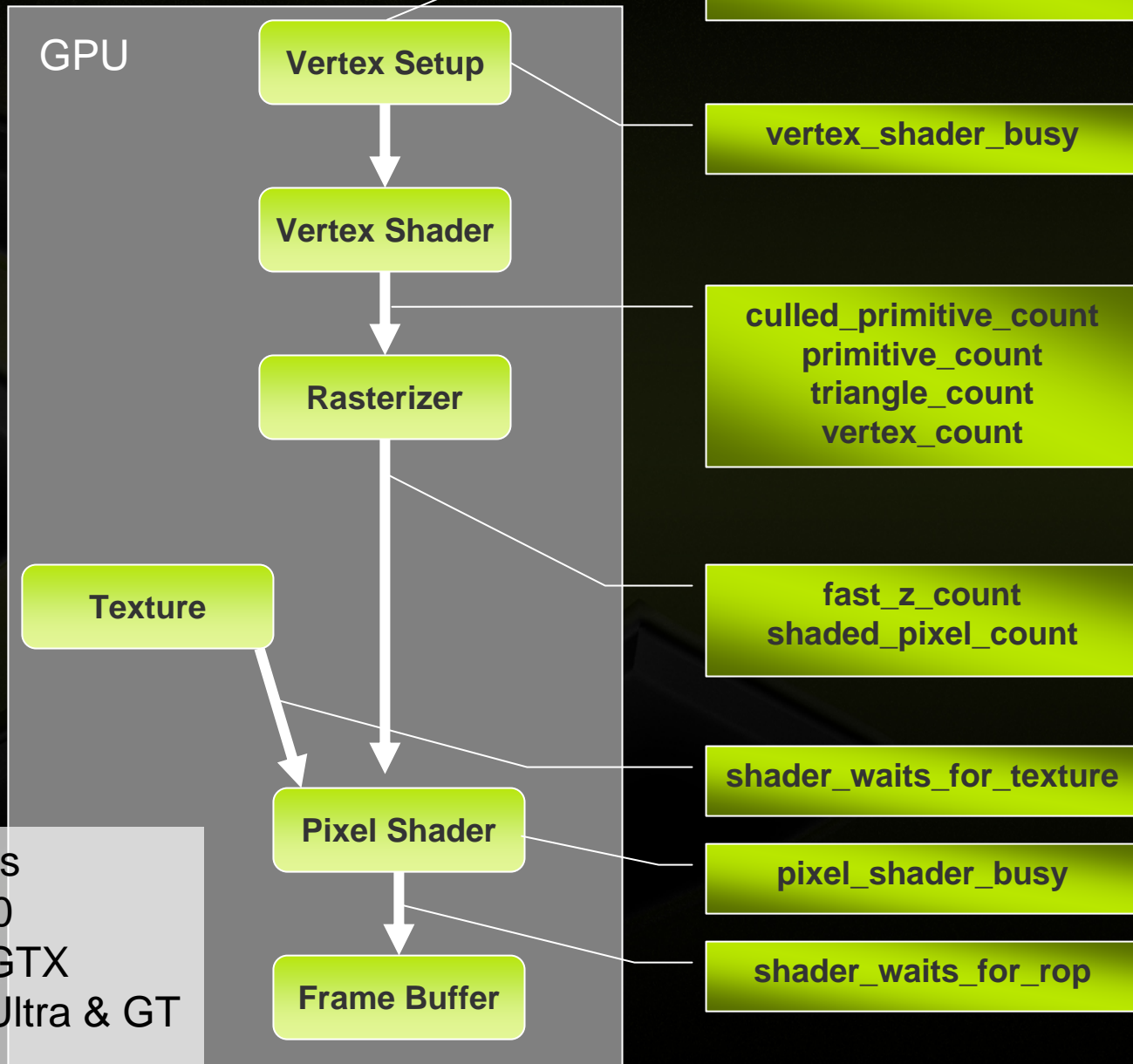
Counter Description	Official Name
FPS	OGL FPS
Frame Time (1/FPS)	OGL frame time mSec
Driver Sleep Time (driver waits for GPU)	OGL frame mSec Sleeping

# Direct3D Signals



Counter Description	Official Name
FPS	D3D frame FPS
Frame Time (1/FPS)	D3D frame time mSec
AGP Memory Used	D3D frame agpmem MB
Video Memory Used	D3D frame vidmem MB
Driver Time	D3D frame mSec in driver
Driver Sleep Time (driver waits for GPU)	D3D frame mSec Sleeping
Triangle Count	D3D frame tris
Batch Count	D3D frame num batches
Locked Render Targets Count	D3D Locked Render Targets

# GPU Signals



Supported GPUs  
Quadro FX 4500  
GeForce 7800 GTX  
GeForce 6800 Ultra & GT  
GeForce 6600



# NVPerfKit Demo: Pixel Shader Bound



# NVPerfKit Demo: Texture Bound





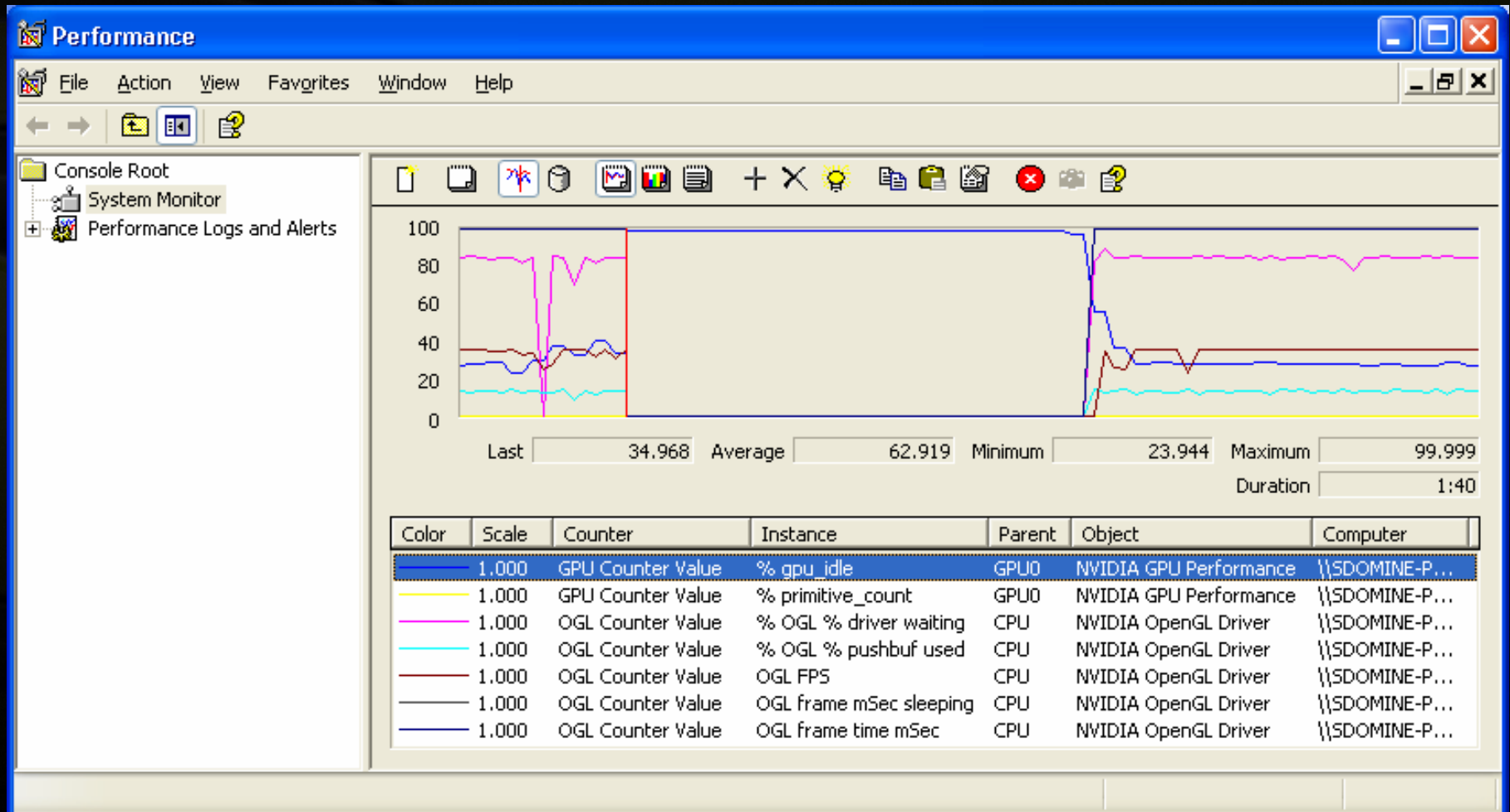
# What is PDH? What is Perfmon?



- **PDH: Performance Data Helper for Windows**
  - Win32 API for exposing performance data to user applications
  - Standardized interface with many providers and clients
- **Perfmon: (aka Microsoft Management Console)**
  - Win32 PDH client application
  - Low frequency sampling (1X/s)
  - Displays PDH based counter values:
    - OS: CPU usage, memory usage, swap file usage, network stats, etc.
    - NVIDIA: all of the signals exported by NVPerfKit



# Associated Tools: Perfmon



# Associated Tools: NVDevCPL



**NVIDIA Developer Control Panel**

**Available Counters**

- D3D frame mSec sleeping
- D3D frame num batches
- D3D frame time mSec
- D3D frame tris
- D3D frame vidmem MB
- D3D Locked Render Targets
- D3D SLI Linear Buffer Sync Bytes
- D3D SLI Linear Buffer Syncs
- D3D SLI P2P Bytes
- D3D SLI P2P transfers
- D3D SLI Render Target Sync Bytes
- D3D SLI Render Target Syncs
- D3D SLI Texture Sync Bytes
- D3D SLI Texture Syncs
- GPU
  - GPU\_Graphics
    - gpu\_idle
    - shaded pixel count

**Active Counters**

- All
  - D3D
    - CPU
      - D3D frame FPS
      - D3D frame mSec in driver
  - GPU
    - GPU\_Graphics
      - gpu\_idle
      - shaded\_pixel\_count
  - OpenGL
    - CPU
      - OpenGL FPS

**Counter Description**

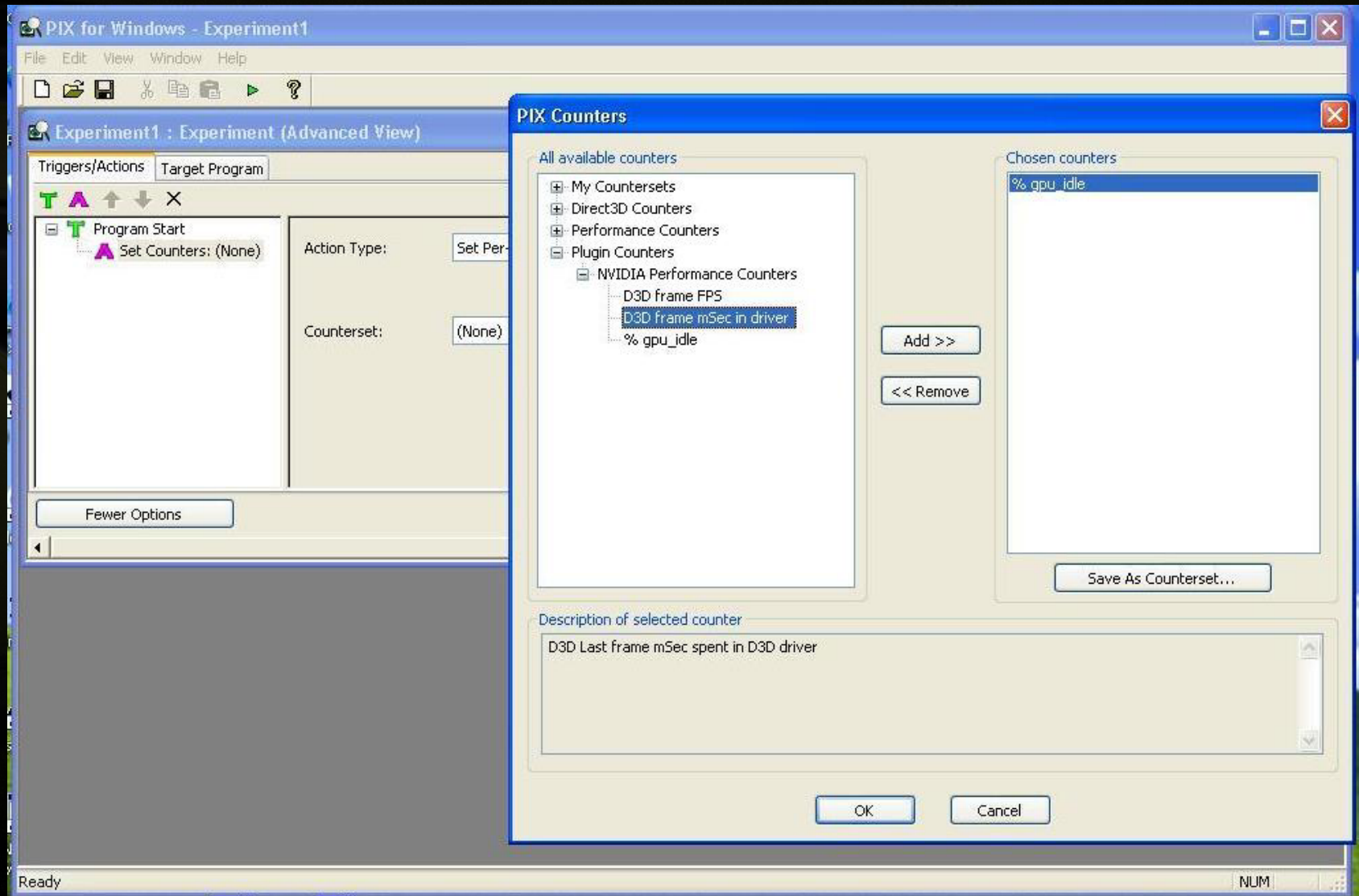
gpu\_idle : Time the graphics portion of the chip is idle.

**Settings**

Default location for counter configuration files (\*.ctr)  **Set Folder...**

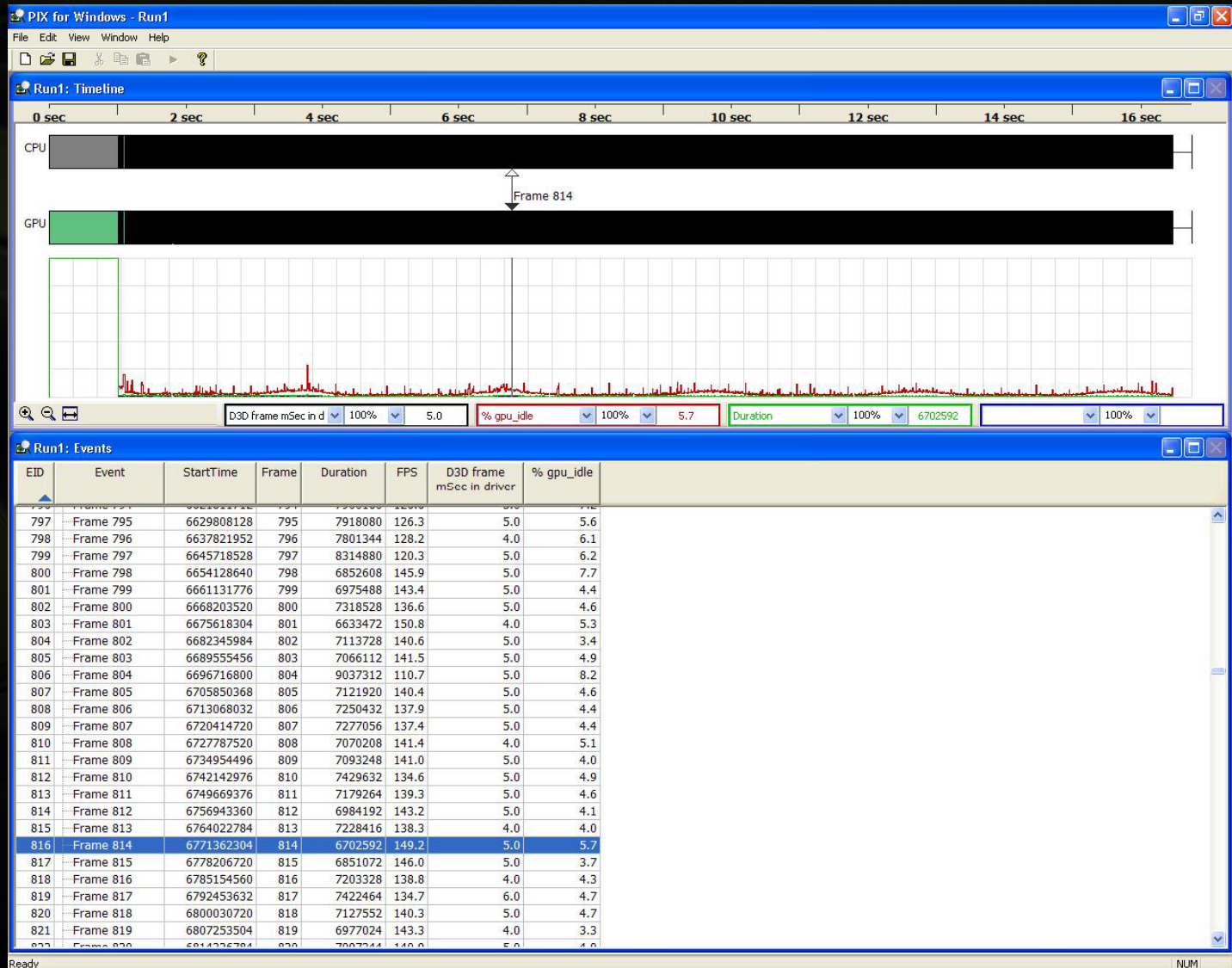
**Buttons:** Add >>, Remove <<, Save..., Load..., Clear, OK, Cancel, Apply

# Associated Tools: NVIDIA Plug-In for Microsoft PIX for Windows





# Associated Tools: NVIDIA Plug-In for Microsoft PIX for Windows



# Helper Classes and Code Samples



- **CPDHHelper: simplifies using PDH**

```
int nIndex = pdh.add("countername");  
pdh.sample();  
float fValue = pdh.value(nIndex);
```

- **CTrace: ring buffer for holding performance data**
- **CTraceDisplay: simple API agnostic graphing library**
- **OpenGL and Direct3D sample apps**
  - Integration of helper classes
  - Security mechanism usage



# Graphic Remedy's gDEBugger 2.0



The image displays the gDEBugger 2.0 interface, which is used for debugging 3D applications. The main window shows a 3D scene titled "Nature" with a landscape featuring a body of water, grass, and mountains under a cloudy sky. The interface includes several panels for monitoring and debugging:

- FPS Monitor:** Displays performance metrics for the scene.
  - Triangle count: 181328
  - Visible Cells: 32%
  - Current FPS: 35
- Alpha, Visibility, Terrain, Sky:** A panel for adjusting rendering settings.
  - Alpha Reference: 0.25
  - Alpha Booster: 1.50
  - Transparency AA: ☒
- gDEBugger - SceneGraph - Trial Version, 30 Days Left:** The main debugging window with a menu bar (File, Edit, View, Debug, Breakpoints, Tools, Help) and a toolbar. It includes a Performance Graph and a Counter List.

**Performance Graph:** A line graph showing various performance metrics over time. The Y-axis ranges from 0 to 100. The X-axis shows a timeline from 0 to 1000. The graph displays several lines representing different metrics, with a notable dip in performance around the 500 mark.

**Counter List:** A table listing various performance counters and their values.

Counter Name	Real Value	Scaled Value	Width
GPU0: % shader_waits_for_texture	29	29 [1]	1px
GPU0: % vertex_shader_busy	11	11 [1]	1px
GPU0: % gpu_idle	0	0 [1]	1px
GPU0: % pixel_shader_busy	30	30 [1]	1px
GPU0: % texture_waits_for_shader	14	14 [1]	1px
CPU: OGL FPS	35	35 [1]	1px
CPU: % OGL % driver waiting	88	88 [1]	1px
CPU: OGL AGP/PCI-E usage (MB)	2	2 [1]	1px
CPU: OGL vidmem usage (MB)	52	52 [1]	1px

**Log:** A list of system events and DLL loading/unloading messages.

- Dll loaded: c:\windows\system32\msvcr71.dll
- Dll loaded: c:\windows\system32\rsaenh.dll
- Dll loaded: c:\windows\system32\crypt32.dll
- Dll loaded: c:\windows\system32\msasn1.dll
- Dll loaded: c:\windows\system32\uxtheme.dll
- Dll loaded: c:\windows\system32\msctf.dll
- Dll loaded: c:\windows\system32\nvoglt.dll
- Dll loaded: c:\windows\system32\mcd32.dll
- DLL unloaded: C:\WINDOWS\system32\MCD32.DLL
- Dll loaded: c:\windows\system32\mslbi.dll
- Dll loaded: c:\windows\system32\oleaut32.dll
- Dll loaded: c:\windows\system32\ole32.dll



# NVPerfKit 2.0



- **Simplified Experiments**
- **Targeted analysis to ease bottleneck determination**
  - Supplement PDH based single counters
  - Multi-pass experiments where multiple GPU counters are needed to compute results
  - Exposes all of the power of NVPerfHUD 4.0 to developers
- **More OpenGL and Direct3D counters**
- **NVPerfHUD 4.0**
- **Linux support**

# Simplified Experiments



## ● Usage:

```
NVPMAddCounter("ps_utilization");
NVPMAddCounter("vs_utilization");
NVPMAddCounter("gpu_idle");
NVPMAllocObjects(50);

int nNumPasses;
NVPMBeginExperiment(&nNumPasses);
for(int ii = 0; ii < nNumPasses; ++ii) {
    NVPMBeginPass(ii);

    // Draw the frame
    NVPMBeginObject(0);
    // DPs associated with object 0
    NVPMEndObject(0);

    NVPMBeginObject(1);
    // DPs associated with object 1
    NVPMEndObject(1);

    // ...
    NVPMEndPass(ii);
}
NVPMEndExperiment();
NVPMGetCounterValue("ps_utilization", 0, &fPSSol); // 0 == object id
NVPMGetCounterValue("vs_utilization", 0, &fVSSol);
```



**NVIDIA®**

**NVPerfHUD 4.0**

**Raul Aguaviva**



# Agenda



- **What is NVPerfHUD?**
- **How does it work?**
- **Demo**
- **Schedule**

# What is NVPerfHUD?



- **Stands for: PERFormance Heads Up Display**
  - **Overlays graphs and dialogs on top of your application**
  - **Interactive HUD**

# What is NVPerfHUD?



- **4 different types of HUD**
  - Performance Dashboard
  - Debug Console
  - Frame Debugger
  - Frame Profiler (New in 4.0)

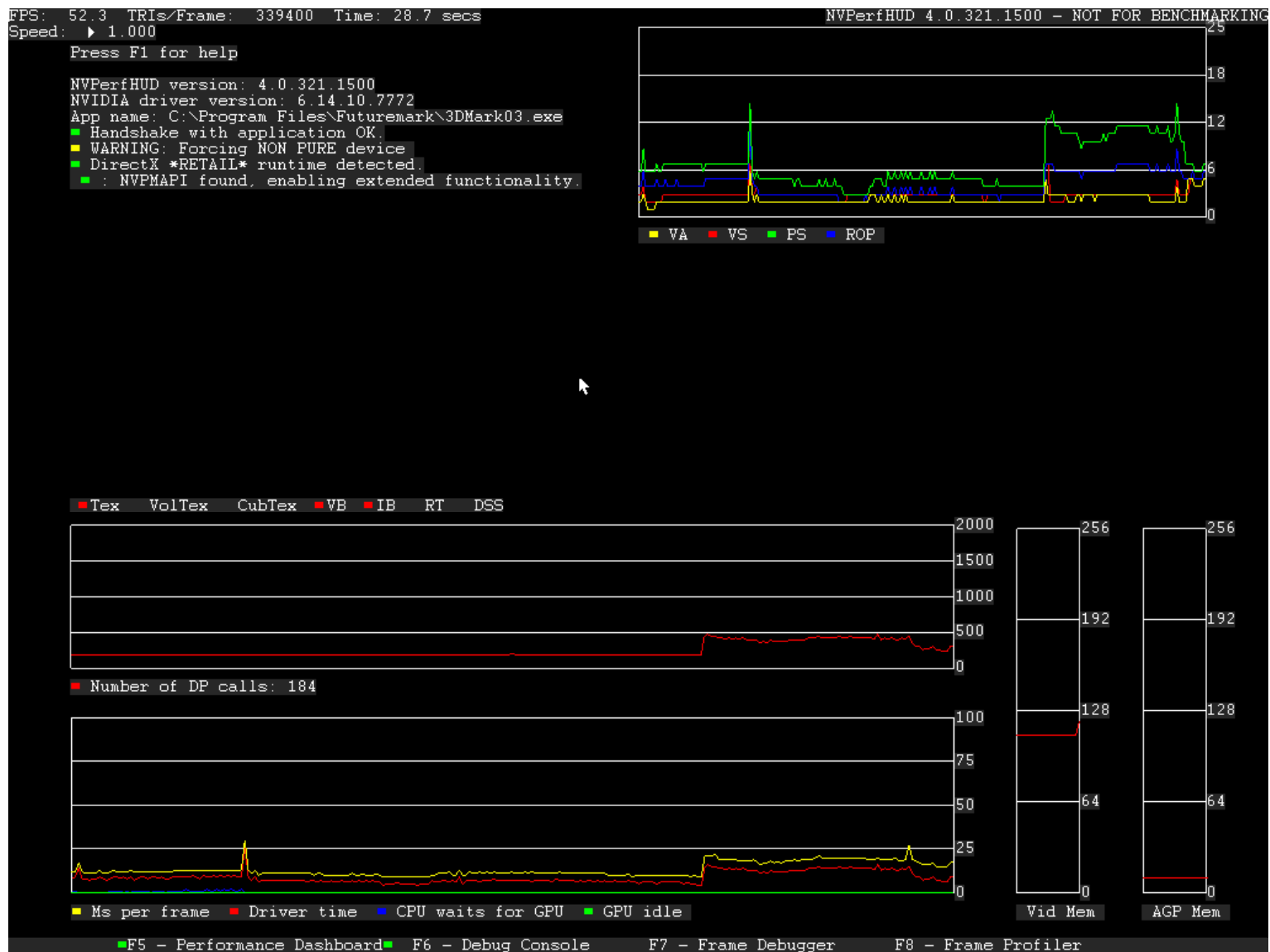


# How to use it



- **Run your application with NVPerfHUD**
- **Use it as you normally do until you find:**
  - **Functional problem: use the debugger**
  - **Low FPS: use the profiler**

# Performance Dashboard



# Performance Dashboard





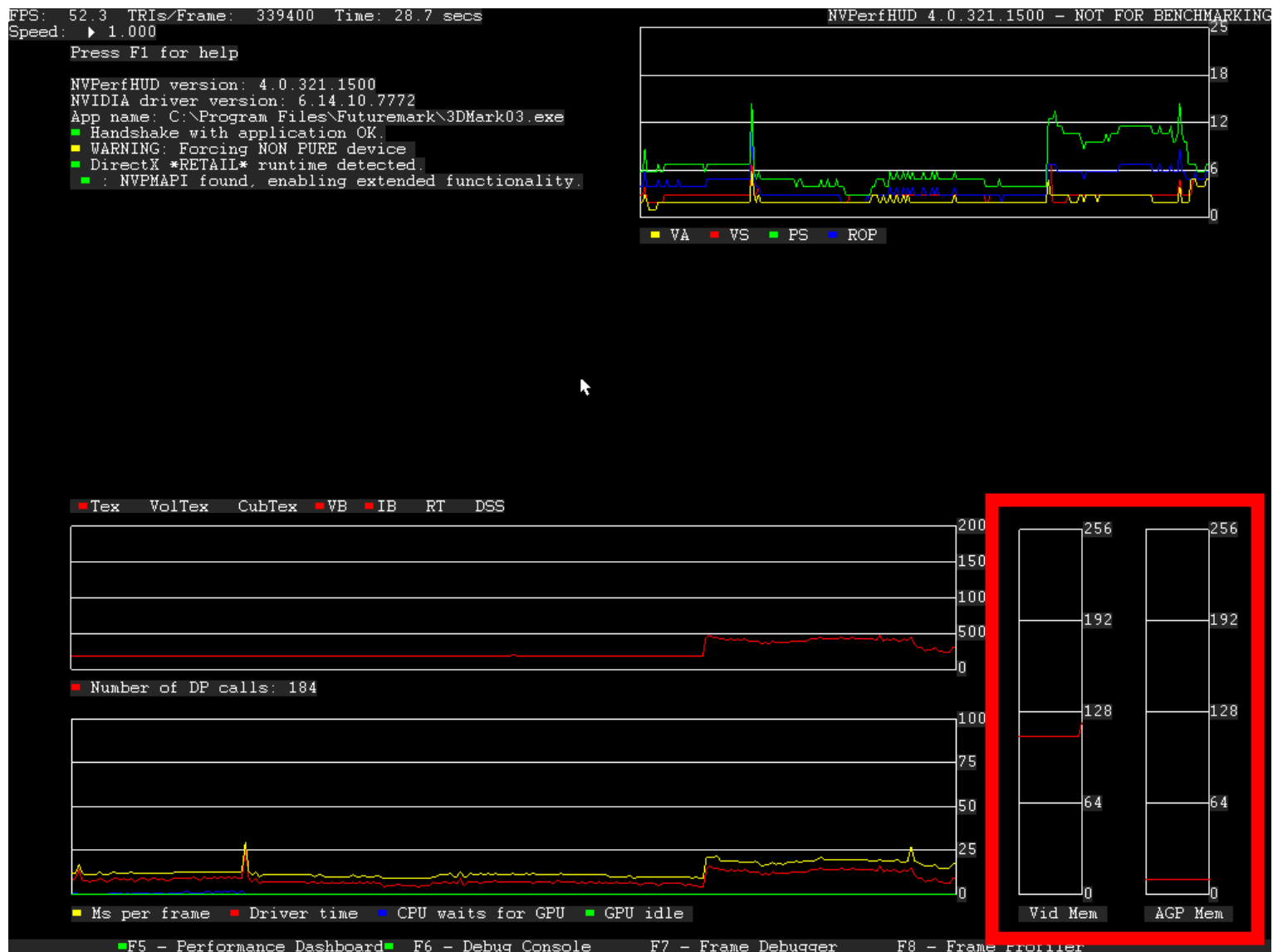
# Performance Dashboard



# Performance Dashboard

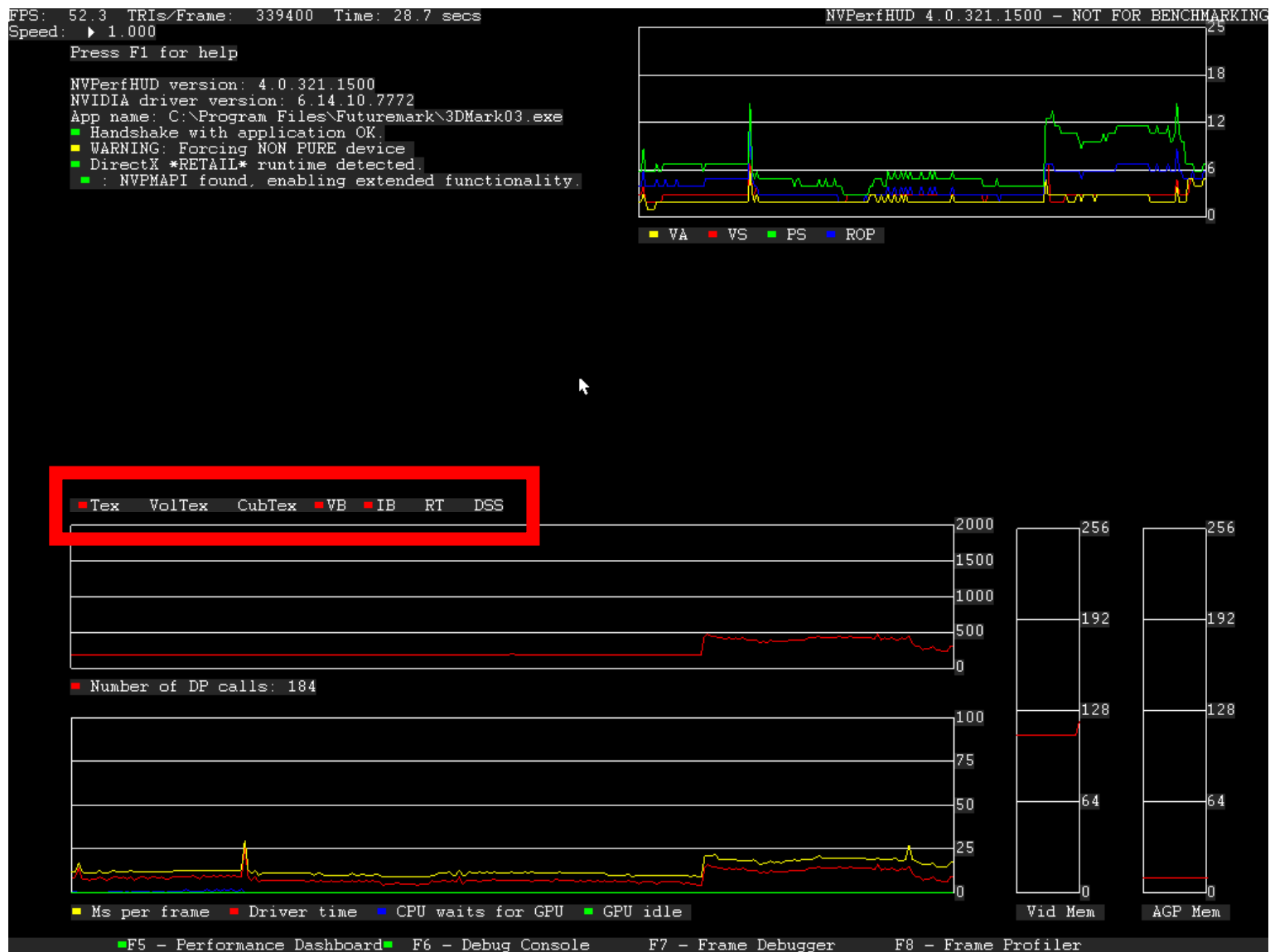


# Performance Dashboard





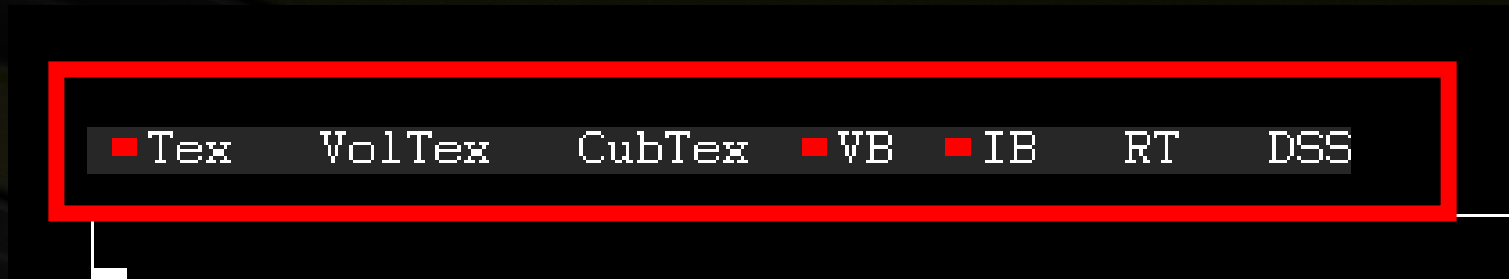
# Performance Dashboard



# Performance Dashboard



## ● Resource monitor



## ● Resources monitored

- Textures
- Volume Textures
- Cube textures
- Vertex Buffers
- Index buffers
- Stencil and depth surfaces

# Performance Dashboard





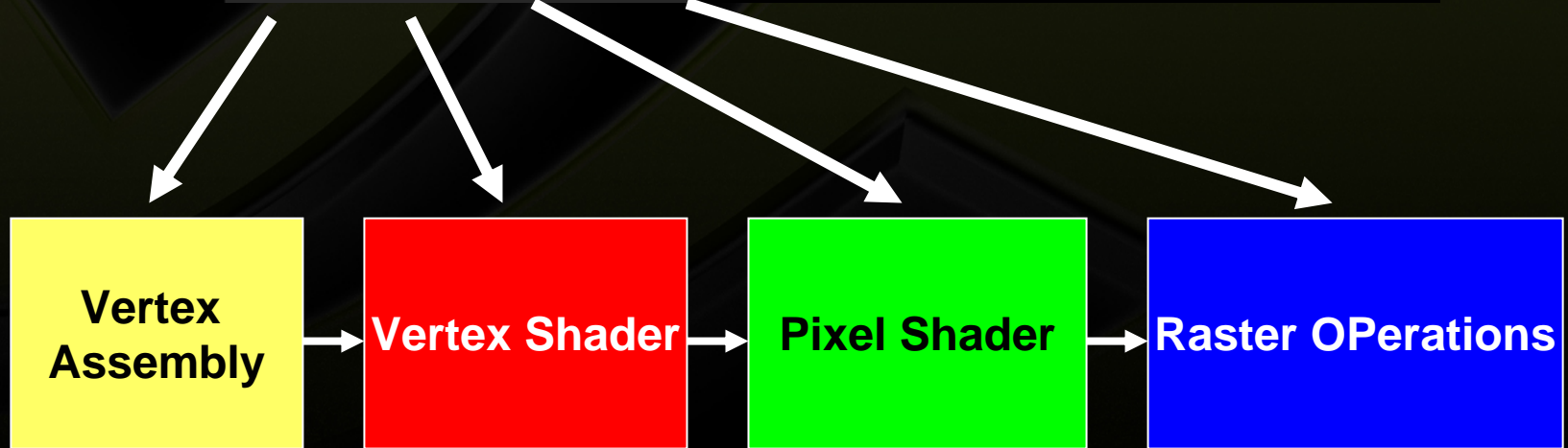
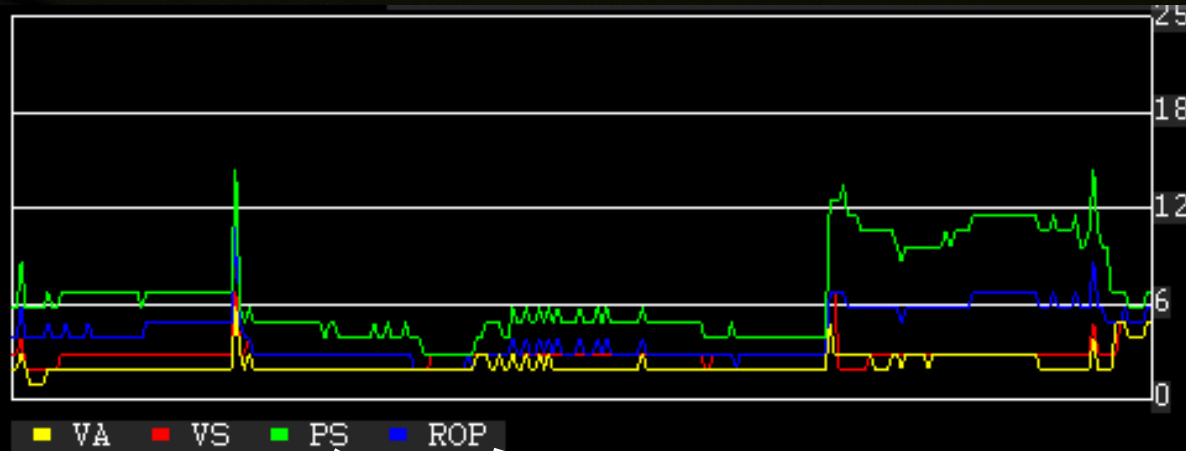
# Performance Dashboard



## ● Speed control

```
FPS: 52.5 Tris/Frame: 339400 Time: 28.7  
Speed: ▶ 1.000  
Press F1 for help  
  
NVPerfHUD version: 4.0.321.1500  
NVIDIA driver version: 6.14.10.7772  
App name: C:\Program Files\Futuremark\
```

# The simplified graphics pipeline



# Performance Dashboard Demo



- **Install**
- **Configure**
- **Drag & Drop**



# Debug Console



FPS: 93.1 TRIs/Frame: 131910 Time: 15.9 secs

NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING

Time: 7.03 secs, IDirect3DDevice9::CreateVertexBuffer(1600,8,0,1)

Time: 7.03 secs, IDirect3DDevice9::CreateIndexBuffer(348,8,101,1)

Time: 7.03 secs, IDirect3DDevice9::CreateVertexBuffer(128,8,0,1)

Time: 7.03 secs, IDirect3DDevice9::CreateIndexBuffer(12,8,101,1)

Time: 7.53 secs, IDirect3DDevice9::CreateVertexBuffer(5952,8,0,1)

Time: 7.53 secs, IDirect3DDevice9::CreateIndexBuffer(1236,8,101,1)

Time: 44.66 secs, IDirect3DDevice9::CreateOffscreenPlainSurface()

Time: 70.00 secs, IDirect3DDevice9::CreateVertexBuffer(65536,520,0,0)

Time: 70.00 secs, IDirect3DDevice9::CreateVertexBuffer(393216,520,0,0)

Time: 70.01 secs, IDirect3DDevice9::CreateIndexBuffer(49152,8,101,1)

Time: 70.01 secs, IDirect3DDevice9::CreateVertexBuffer(98304,520,0,0)

Time: 70.01 secs, IDirect3DDevice9::CreateIndexBuffer(24576,8,101,1)

Time: 70.01 secs, IDirect3DDevice9::CreateVertexBuffer(80,8,0,1)

Time: 70.17 secs, IDirect3DDevice9::CreateTexture(1024x1024,1,0,22,1)

Time: 70.18 secs, IDirect3DDevice9::CreateTexture(512x64,1,0,22,1)

Time: 70.18 secs, IDirect3DDevice9::CreateVertexBuffer(1280,520,0,0)

☐ Clear Log Each Frame

☐ Stop Logging

☐ Fade Console

F5 - Performance Dashboard

F6 - Debug Console

F7 - Frame Debugger

F8 - Frame Profiler

# Frame Debugger



FPS: 70.6 TRIs/Frame: 88410 Time: 10.4 secs [ON] NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING

Sampler: s0  
Type: TEXTURE  
512x512, DXT1  
MIPs:10 MIP:NONE  
Mag:LINEAR Min:LINEAR

Nothing

Prims Drawn: 39780 Warnings: 3  
DrawIndexedPrimitive(D3DPT\_TRIANGLELIST, 0, 0, 28385, 118140, 400)  
RT:0x0015d520 EB:0x0015d520

Step Back Step Forward Draw Call 32/69 none Advanced... Hide All

F5 - Performance Dashboard F6 - Debug Console F7 - Frame Debugger F8 - Frame Profiler



# Frame Debugger, advanced view



FPS: 60.3 Tris/Frame: 88410 Time: 81.2 secs NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING

Vertex Assembly | Vertex Shader | Pixel Shader | Raster Operations

Vertex Shader

Index / Vertex Buffer

GP sub 1  
DrawIndexPrimitive:  
Type: D3DPT\_TRIANGLELIST  
BaseVertexIndex: 0  
MinVertexIndex: 0  
NumVertices: 23885  
startIndex: 151840  
primCount: 400  
HRESULT: 0x00000000  
Msg: S\_OK  
Desc: The function completed successfully

Index Buffer Description \*\*  
Format: INDEX16  
Pool: D3DPPOOL\_MANAGED  
Usage: Length: 238880 bytes  
Instances: 1

VB Declaration \*\*  
Total vertex size: 32  
0 - POSITION FLOAT3 DEFAULT  
1 - NORMAL FLOAT3 DEFAULT  
6 - TEXCOORD FLOAT2 DEFAULT

Stream 0 Vertex Buffer Description \*\*  
Pool: D3DPPOOL\_MANAGED  
Usage: Length: 900320 bytes  
Instances: 1

Vertex Buffer Bounding Box \*\*  
Min: 0.036388 -0.031983 -0.208380  
Max: 0.139948 -0.015102 -0.853023

Step Back Step Forward Draw Call 32/69

F5 - Performance Dashboard F6 - Debug Console F7 - Frame Debugger F8 - Frame Profiler

FPS: 65.4 Tris/Frame: 88410 Time: 12.8 secs NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING

Vertex Assembly | Vertex Shader | Pixel Shader | Raster Operations

Vertex Shader

Textures (4x to zoom)

Vertex Shader Constants

Fixed Point Constants:  
Q1 0 0.000000 0.000000 1.000000 2.000000  
Q2 10.315573 0.000000 -0.044971 -0.056503  
Q3 0.005559 14.573072 2.259137 0.198903  
Q4 2.208951 -1.007150 5.631020 1.971756  
Q5 2.203635 -1.005136 5.619758 2.087613  
Q20 0.000000 0.000000 0.000000 0.000000  
Q40 0.500000 -0.500000 0.500000 0.500000  
Q41 1.000000 1.000000 1.000000 1.000000

Integer Constants:  
Boolean Constants:

Step Back Step Forward Draw Call 32/69

F5 - Performance Dashboard F6 - Debug Console F7 - Frame Debugger F8 - Frame Profiler

FPS: 65.6 Tris/Frame: 88410 Time: 35.4 secs NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING

Vertex Assembly | Vertex Shader | Pixel Shader | Raster Operations

Pixel Shader

Textures (4x to zoom)

Pixel Shader Constants

Fixed Point Constants:  
Q1 0 0.000000 0.000000 0.000000 0.000000  
Q4 1020.000000 31.875000 0.996094 -31.8750

Integer Constants:  
Boolean Constants:

Step Back Step Forward Draw Call 66/69

F5 - Performance Dashboard F6 - Debug Console F7 - Frame Debugger F8 - Frame Profiler

FPS: 69.6 Tris/Frame: 88410 Time: 15.2 secs NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING

Vertex Assembly | Vertex Shader | Pixel Shader | Raster Operations

Render Targets and Render States

Render Target 0  
Type: ETEXTURE2D  
Back Buffer size: 1024x768  
Back Buffer format: DXGI\_FORMAT\_R8G8B8A8\_UNORM  
Back Buffer count: 2  
MultiSampleType: DXGI\_SAMPLE\_TYPE\_1  
MultiSampleQuality: 0  
AutoGenerateMipMaps: 0  
FullScreen\_RefreshRateHz: 0  
VSynced: 0

Render Target 1  
Not set

Render Target 2  
Not set

Render Target 3  
Not set

Render States Dump \*\*  
ZENABLE = D3DZ\_TRUE  
FILLMODE = D3DFILL\_SOLID  
SHADERMODE = D3DSHADER\_QUIRKY  
ZWRITABLE = TRUE  
ALPHATESTENABLE = FALSE  
LASTPIXEL = TRUE  
SRCBLEND = D3DBLEND\_ONE  
DESTBLEND = D3DBLEND\_INVSRCOLOR  
CULLMODE = D3DCULL\_CW  
ZFUNC = D3DCMP\_LESSEQUAL  
ALPHAREF = 0  
ALPHABLEND = D3DBLEND\_ALPHABLEND  
ALPHABLENDENABLE = FALSE  
OTHERBLENDABLE = FALSE  
FOGBLEND = FALSE  
SPECULARENABLE = FALSE  
FOGCOLOR = (0, 0, 0, 0)

Step Back Step Forward Draw Call 32/69

F5 - Performance Dashboard F6 - Debug Console F7 - Frame Debugger F8 - Frame Profiler



# Frame Profiler



- Measures performance counters
- strategy

# Frame Profiler, measuring



- **NVPerfHUD uses NVPerfKit**
  - **uses ~40 Performance Counters (PC's)**
- **Can not read all of them at the same time**
- **Need to render THE SAME FRAME until all the PC's are read**



# Frame Profiler, strategy



- **Optimization Strategy:**

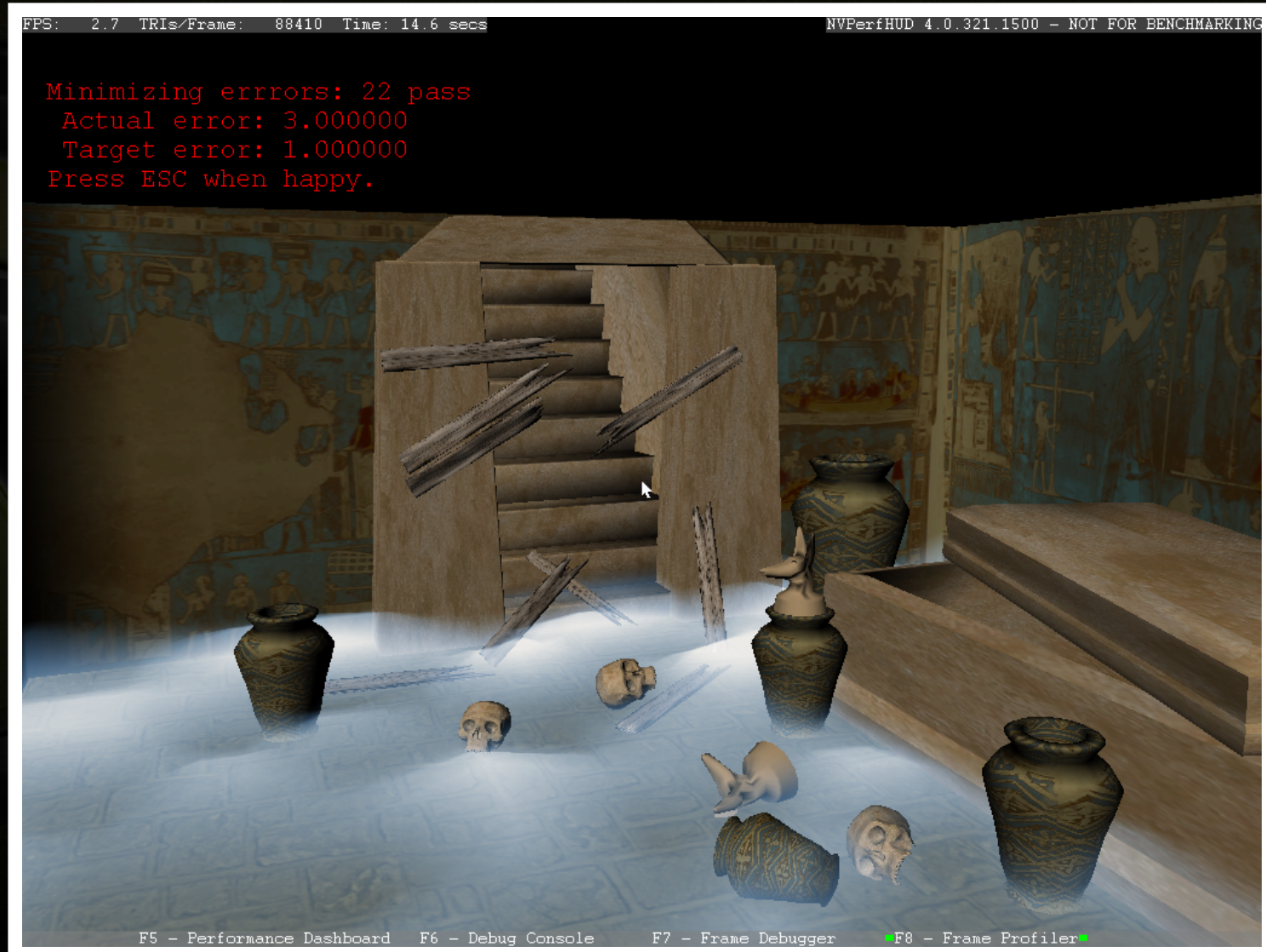
- Group by state is roughly grouping by bottleneck
- These groups are called “state buckets”

- **Procedure**

- Group draw calls by rendering state into state buckets
- Identify the bottleneck of the most expensive state bucket
  - Solved by NVPerfHUD
- Cure the bottleneck with a common corrective action



# Frame Profiler Demo, measuring



# Frame Profiler Demo





# Frame Profiler Demo





# Frame Profiler Demo, advanced view



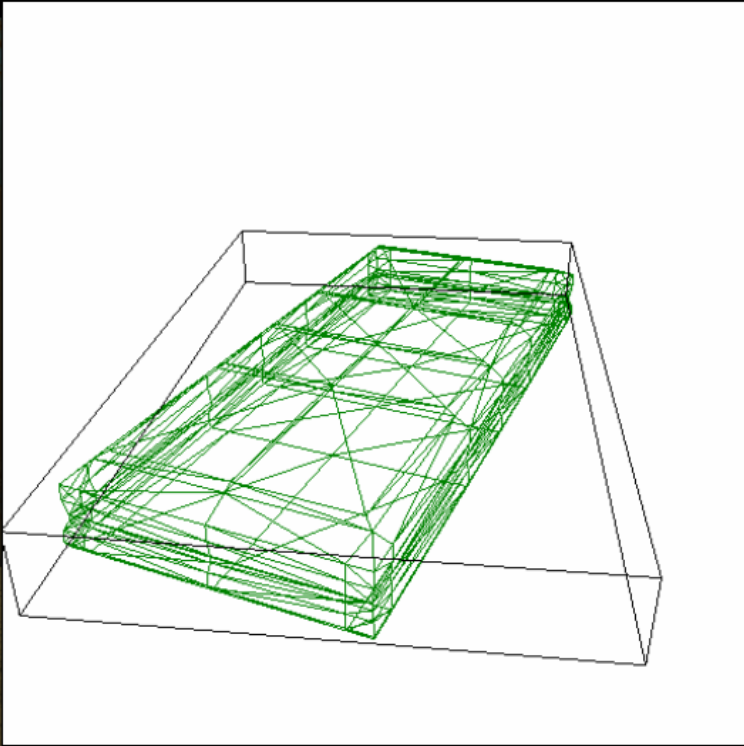
FPS: 59.1 TRIs/Frame: 88410 Time: 39.8 secs NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING

0 - [type 0][Buckets 32][time 219][pixels 1067830]

0 - [index 14][type 0][time 20][Err 0%][pixel 67220]

Vertex Assembly Vertex Shader Pixel Shader Raster Operations

Wireframe



Index / Vertex Buffer

```
** DP Info **  
DrawIndexedPrimitive:  
Type: D3DPT_TRIANGLELIST  
BaseVertexIndex: 0  
MinVertexIndex: 0  
NumVertices: 28385  
startIndex: 118140  
primCount: 400  
HRESULT: 0x00000000  
Msg: S_OK  
Desc: The function completed successfully  
  
** Index Buffer Description **  
Format: INDEX16  
Pool: D3DPOOL_MANAGED  
Usage: Length: 238680 bytes  
  
** VB Declaration **  
Total vertex size: 32  
0 - POSITION FLOAT3 DEFAULT  
0 - NORMAL FLOAT3 DEFAULT  
0 - TEXCOORD FLOAT2 DEFAULT  
  
** Stream 0 Vertex Buffer Description **  
Pool: D3DPOOL_MANAGED  
Usage: Length: 908320 bytes  
Instancing: 1  
  
** Vertex Buffer Bounding Box **  
Min 0.039388 -0.031983 -0.208380  
Max 0.139946 -0.015102 -0.053023
```

Step Back Step Forward Draw Call 32/69 none Simple... Hide All

F5 - Performance Dashboard F6 - Debug Console F7 - Frame Debugger F8 - Frame Profiler

# About freezing the application



- Only possible if the application uses time-based animation
- Stop the clock
  - Intercept: QueryPerformanceCounter(), timeGetTime()
  - NO RDTSC!!
- $Pos += V * DeltaTime$



# Schedule



- **Beta: August**
- **Release : September**



NVPerfHUD 3 - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://developer.nvidia.com/object/nvperfhud\_home.html

Go nvperfhud 3.0

Firefox Help Firefox Support Plug-in FAQ Sign In - Yahoo! Uber Gizmo, the Ga... Microsoft Visual C++ DirectX Amazon.com: All Pr... sapere.it

**developer.nvidia.com**  
THE SOURCE FOR GPU PROGRAMMING

Search


WWW developer.nvidia.com

---

REGISTERED DEVELOPERS  
JOIN

HOME  
NEWS  
DOCUMENTATION  
TOOLS & SDKs  
PARTNERS

NEWSLETTER SIGN-UP  
EVENTS CALENDAR  
DRIVERS  
CONTACT



Legal Info

## NVPerfHUD 3

### Quick Links

- ♦ [Introductory Video](#)
- ♦ [Downloads](#)
- ♦ [GameDev.net Review](#)

### Overview

Modern GPUs generate images through a pipelined sequence of operations. A pipeline runs only as fast as its slowest stage, so tuning graphical applications for optimal performance requires a pipeline-based approach to performance analysis. NVPerfHUD analyzes your graphics pipeline performance and provides real-time statistics you can use to diagnose performance bottlenecks in your 3D application.

The latest release includes several new features and update modes:

- ♦ **Frame Analysis Mode**  
Freeze your application and single-step through the current frame to see what is happening inside the GPU at each stage of your graphics pipeline. You can view the Vertex Shader, Pixel Shader & Render operations.
- ♦ **Debug Console Mode**  
This mode shows you DirectX Debug Runtime messages, and custom messages from your application.
- ♦ **Performance Mode**  
The powerful performance analysis experiments from the previous release are still available in Performance Mode.

The opt-in mechanism for enabling NVPerfHUD 3.0 requires no code alterations if you have already enabled NVPerfHUD 2.0 in your application.

Be sure to check out the Getting Started instructions in the [NVPerfHUD User Guide](#), and read through the methodology for effectively identifying and crushing performance bottlenecks in your application. We've also created a [Quick Reference Card](#) with tips and shortcuts that you can keep at your fingertips. Both these documents are available in English, Japanese, Chinese, and Korean.

See our "NVIDIA Performance Analysis Tools" talk from [GDC 2005](#) for more information on how to analyze your applications using NVPerfHUD. In addition, our "Practical Performance Analysis and Tuning" talk from [GDC 2004](#) explains the theory of pipeline analysis and bottleneck removal.

### Downloads

### Developer Reviews


NVPerfHUD has way more features than I expected! Blew me away!!  
- *Remy Saville*  
*Graphics Programmer*  
*Relic Entertainment, Inc.*

[Frame Analysis Mode] is by far the most impressive mode of NVPerfHUD. ... A few moments in this mode does more to show how advanced graphic pipelines work than anything else I have seen. ... NVPerfHUD should be of interest to anybody who develops software that employs DirectX graphics. Even if you don't have an NVIDIA card it's worth the price of a GeForce 6600 or 6800 to get this tool.  
- *Bryan Mau*  
[GameDev.net Review](#)

NVPerfHUD 3 - it's simply amazing. I use it virtually every day.  
- *Chris King, President*  
*IDV, Inc.*

NVPerfHUD is a great tool for debugging and performance analysis.  
- *Richard Schubert*  
*Graphics/Effects Programmer*  
*Yager Development GmbH*

I just wanted to drop a note to thank you for the hard work on the NVPerfHUD. This new version is incredibly useful. Not a day goes by that I don't appreciate the support NVIDIA gives developers with tools such as this.  
- *Matt Shaw*  
*Director of Technology*  
*Mythic Entertainment*



Done

Start

2 Win... Oxford ... 4 Fire... 4 Micr... Total C... nvPerf... unai^M... Microso...

5:59 PM

# Questions?



- **Developer tools DVDs available at our booth**
- **Online: <http://developer.nvidia.com>**

[NVGLExpert@nvidia.com](mailto:NVGLExpert@nvidia.com)

[NVShaderPerf@nvidia.com](mailto:NVShaderPerf@nvidia.com)

[NVPerfKIT@nvidia.com](mailto:NVPerfKIT@nvidia.com)

[NVPerfHUD@nvidia.com](mailto:NVPerfHUD@nvidia.com)

[FXComposer@nvidia.com](mailto:FXComposer@nvidia.com)



# The Source for GPU Programming

[developer.nvidia.com](http://developer.nvidia.com)

- Latest News
- Developer Events Calendar
- Technical Documentation
- Conference Presentations
- GPU Programming Guide
- Powerful Tools, SDKs and more ...



**nVIDIA**®

Join our FREE registered developer program for early access to NVIDIA drivers, cutting edge tools, online support forums, and more.

[developer.nvidia.com](http://developer.nvidia.com)

©2004 NVIDIA Corporation. NVIDIA, and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation. Nalu is ©2004 NVIDIA Corporation. All rights reserved.



# NVIDIA SDK

## The Source for GPU Programming



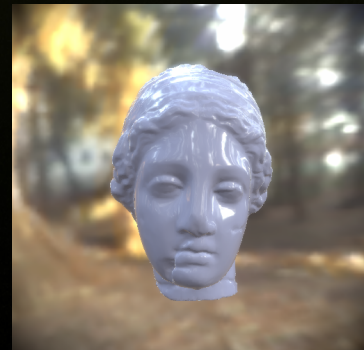
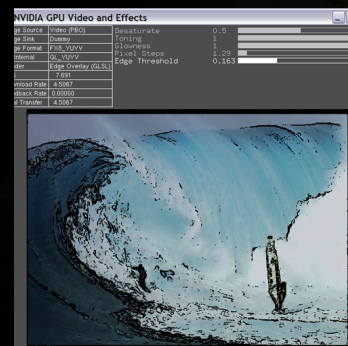
Hundreds of code samples and effects that help you take advantage of the latest in graphics technology.

- Tons of updated and all-new DirectX and OpenGL code samples with full source code and helpful whitepapers:

**Transparency AA, GPU Cloth, Geometry Instancing, Rainbow Fogbow, 2xFP16 HRD, Perspective Shadow Maps, Texture Atlas Utility, ...**

- Hundreds of effects, complete with custom geometry, animation and more:

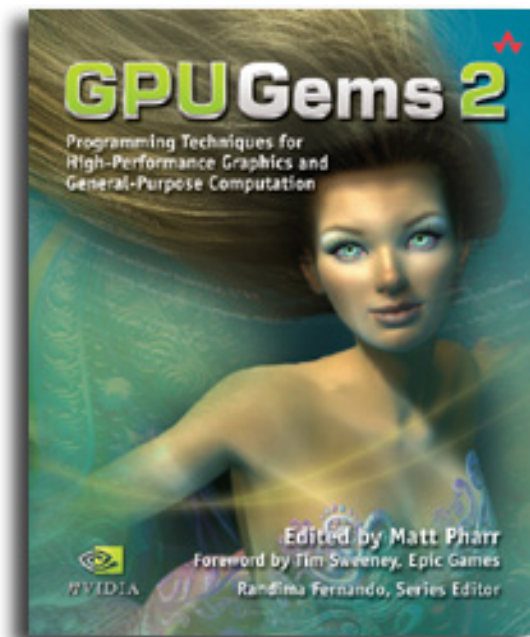
**Shadows, PCSS, Skin, Plastics, Flame/Fire, Glow, Image Filters, HLSL Debugging Techniques, Texture BRDFs, Texture Displacements, HDR Tonemapping, and even a simple Ray Tracer!**



# 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



### Graphics Programming



- Geometric Complexity
- Shading, Lighting, and Shadows
- High-Quality Rendering

### GPGPU Programming



- General Purpose Computation on GPUs: A Primer
- Image-Oriented Computing
- Simulation and Numerical Algorithms