



**NVIDIA**®

## **Performance Tools**

**Jeff Kiel, NVIDIA Corporation**

# Performance Tools Agenda



- **Problem statement**
- **Overview of GPU pipelined architecture**
- **NVPerfKit 2.0: Driver and GPU Performance Data**
  - **NVPerfHUD: The GPU Performance Accelerator**
  - **NVPerfSDK: Performance data integrated into your application**
- **NVPerfHUD ES Sneak Preview**
- **gDEDebugger: OpenGL performance analysis and debugging**
- **NVShaderPerf: Shader program performance**

# The Problem?



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

# The Solution? NVPerfKit!

**35% FPS improvement!\***  
**11 Rendering bugs fixed!\***

\*Average of 35% FPS improvement and 11 bugs fixed reported by  
over 100 users of NVPerfHUD in recent Developer Survey

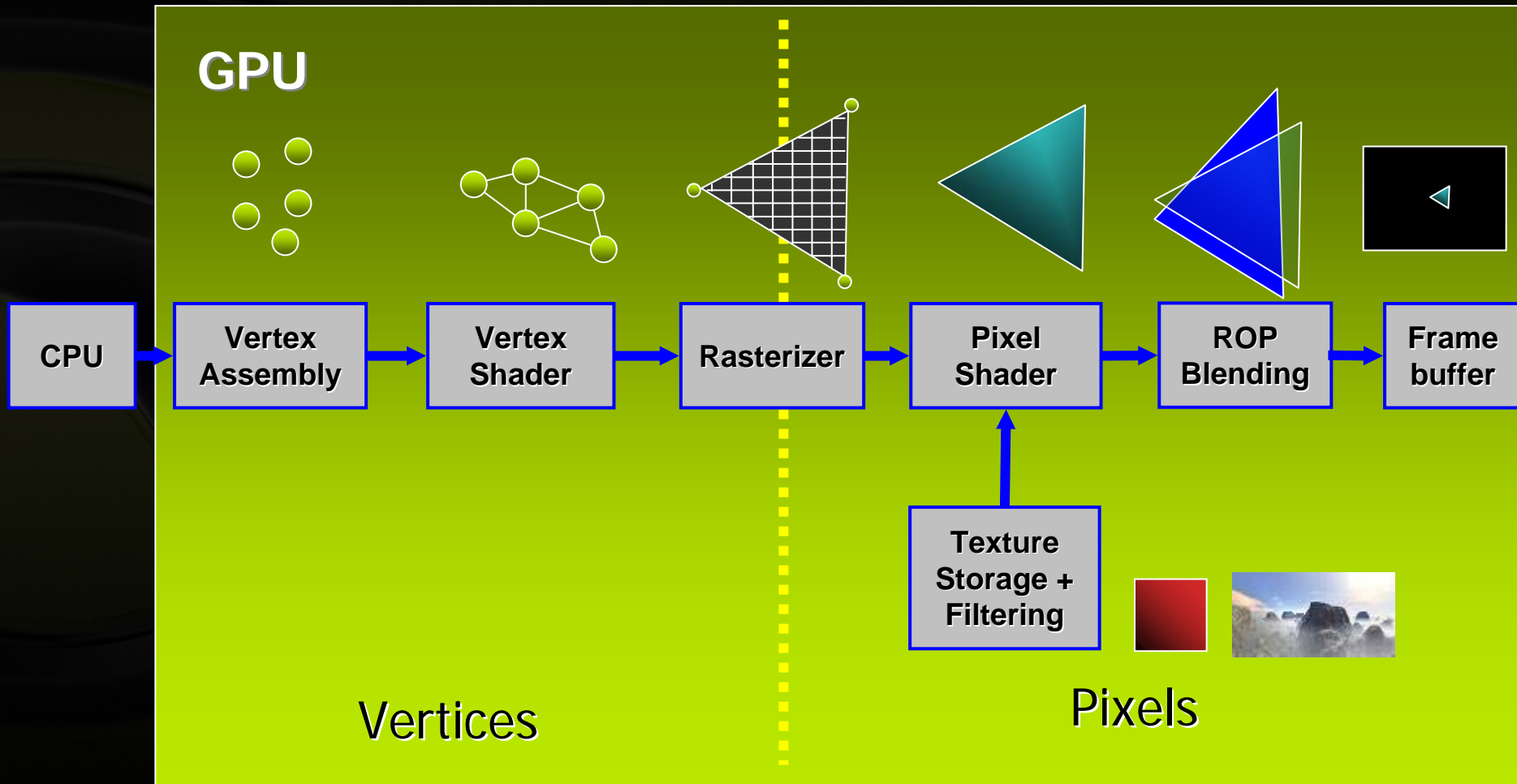
# GPU pipelined architecture



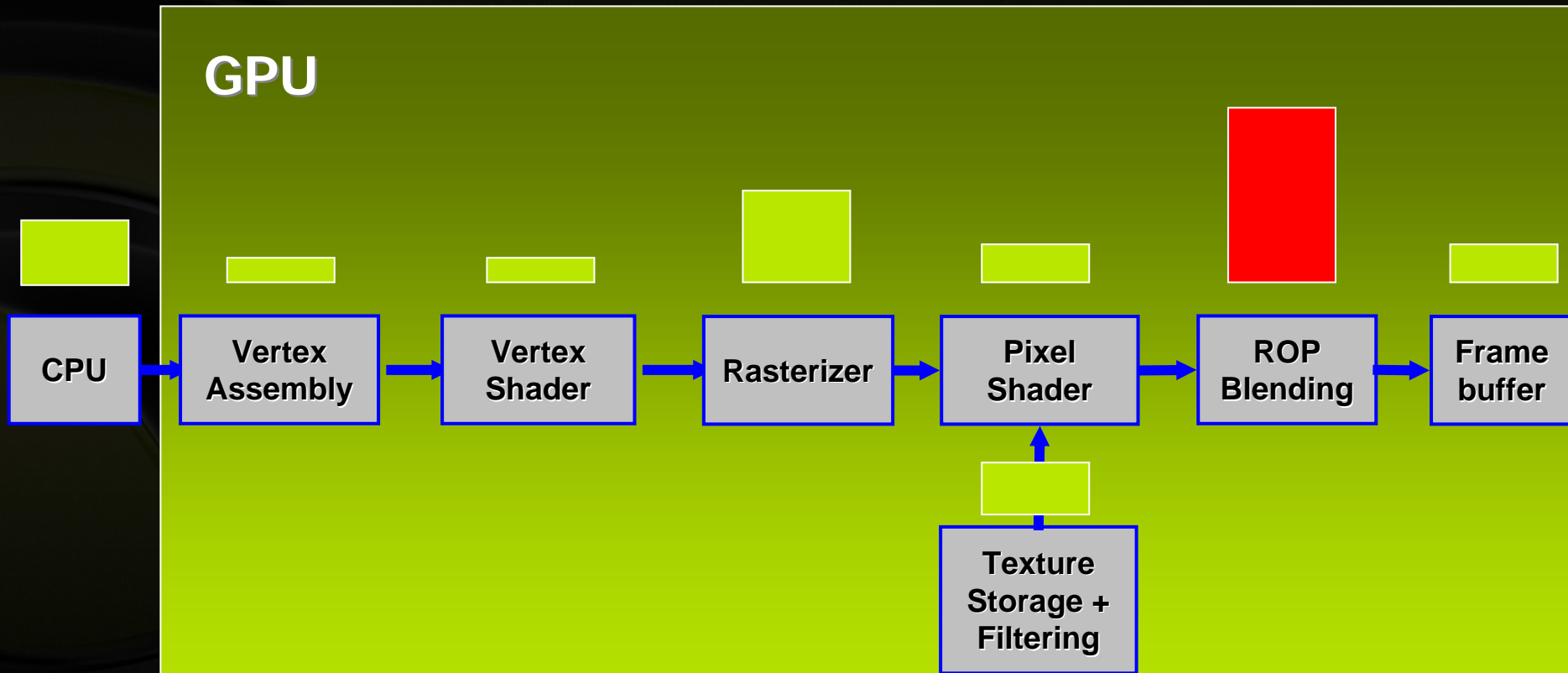
- **Pipelined architecture: each unit needs the data from the previous unit to do its job**
- **Method: Bottleneck identification and elimination**
- **Goal: Balance the pipeline**



# GPU Pipelined Architecture (simplified view)



# GPU Pipelined Architecture (simplified view)

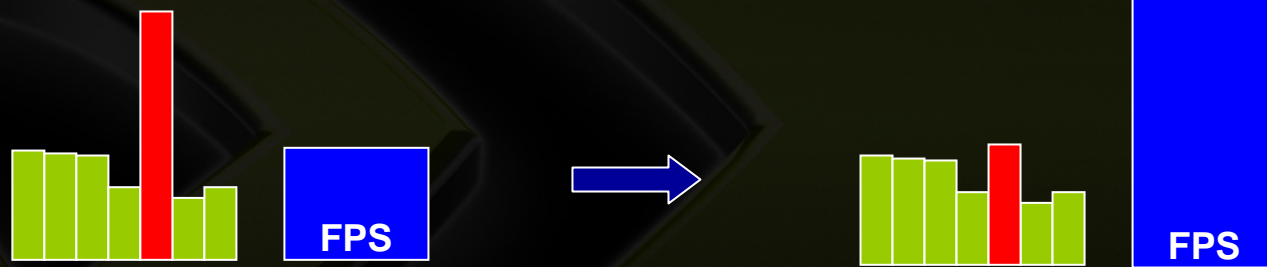
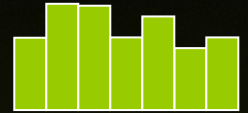


One unit can limit the speed of the pipeline...

# Classic Bottleneck Identification



Modify target stage to decrease workload

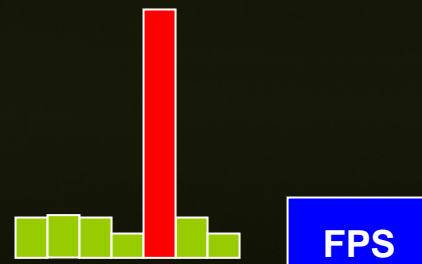
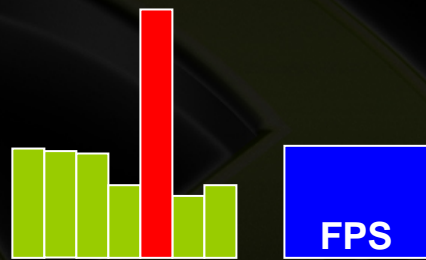
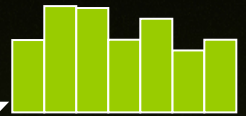


If performance/FPS improves greatly, this stage is the bottleneck  
Careful not to change the workload of other stages!

# Classic Bottleneck Identification



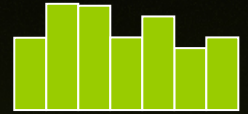
Rule out other stages, give them little or no work



If performance doesn't change significantly, this stage is the bottleneck  
Careful not to change the workload of target stage!



# Ideal Bottleneck Identification



- **Sample performance data in each subunit of the GPU pipeline while rendering**
  - Compare amount of work done to maximum work possible
  - Query the subunit for unit bottleneck information
- **NVPerfKit: The Ideal GPU Performance Tool!**
  - NVPerfHUD: The GPU Performance Accelerator
  - NVPerfAPI: Integrated in your application

**Analyze your application like an NVIDIA Engineer!**

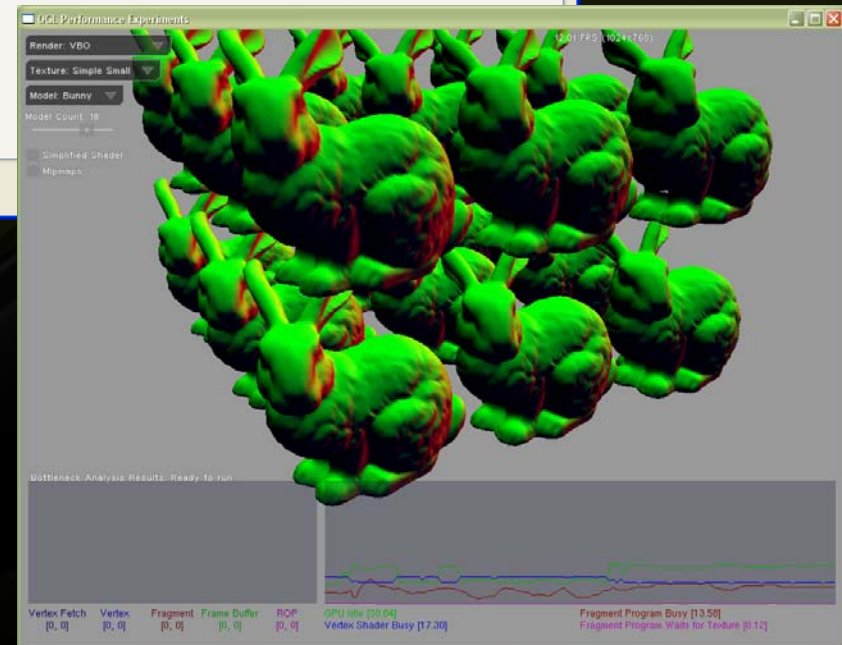
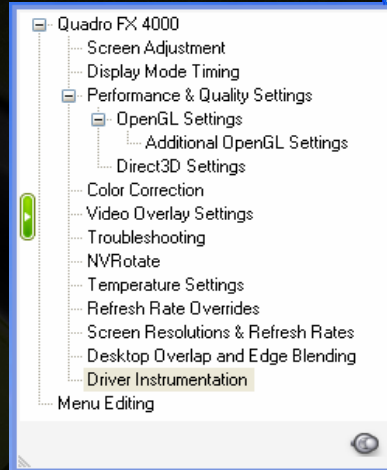
# What is in the NVPerfKit package?



Instrumented Driver  
GLExpert  
NVPerfHUD  
NVPerfSDK  
NVPerfAPI  
Sample Code  
Helper Classes  
Documentation

## Tools

NVIDIA Plug-In for  
Microsoft PIX for Windows  
gDEDebugger 2.4  
NVDevCPL



# NVPerfKit Instrumented Driver



- **GLExpert functionality**
- **Exposes GPU and Driver Performance Counters**
- **Data exported via NVIDIA API and PDH**
- **Supports OpenGL and Direct3D**
- **Simplified Experiments (SimExp)**
- **Collect GPU and driver data, retain performance**
  - **Track per-frame statistics**
  - **Gather and collate at end of frame**
  - **Performance hit 1-2%**

# GLExpert: What is it?



- **Helps eliminate performance issues on the CPU**
- **OpenGL portion of the Instrumented Driver**
  - **Output to stdout or debugger**
  - **Different groups/levels of information detail**
  - **Controlled using environment variables in Linux, tab in NVDevCPL on Windows**
- **Information provided:**
  - **GL Errors: print when raised**
  - **Software Fallbacks: indicate when the driver is in fall back**
  - **GPU Programs: errors during compile or link**
  - **VBOs: show where they reside, mapping details**
  - **FBOs: print reasons for unsupported configuration**

# GLExpert: NVDevCPL tab



NVIDIA Developer Control Panel

NVIDIA Performance Monitor

Info

GL  
GLExpert

GLExpert Settings

- Report Errors
- Report Software Fallback Messages
- Report Vertex and Fragment Program Messages
- Report VBO Messages
- Report FBO Messages

Detail Level

Off Max

All errors with warnings on usage which may cause suboptimal runtime performance

Send Output To:

OK Cancel Apply

# NVPerfKit: Counter Types



- **SW/Driver Counters (Instrumented Driver)**
  - Insight into OpenGL and Direct3D driver performance
  - Exposed via NVPerfAPI, PIX, and PDH
- **Raw GPU Counters (Instrumented GPU)**
  - Real time performance monitoring
  - Exposed via NVPerfAPI, PIX, and PDH
- **Simplified Experiments (Instrumented GPU)**
  - In depth performance analysis and bottleneck determination
  - Exposed via NVPerfAPI
- **Instrumented GPUs**

Quadro FX 5500 & 4500  
GeForce 7900 GTX & GT  
GeForce 7800 GTX

GeForce 6800 Ultra & GT  
GeForce 6600

# OpenGL/Direct3D Counters



## ● General

- FPS
- ms per frame

## ● Driver

- Driver frame time (total time spent in driver)
- Driver sleep time (waiting for GPU)
- % of the frame time driver is waiting

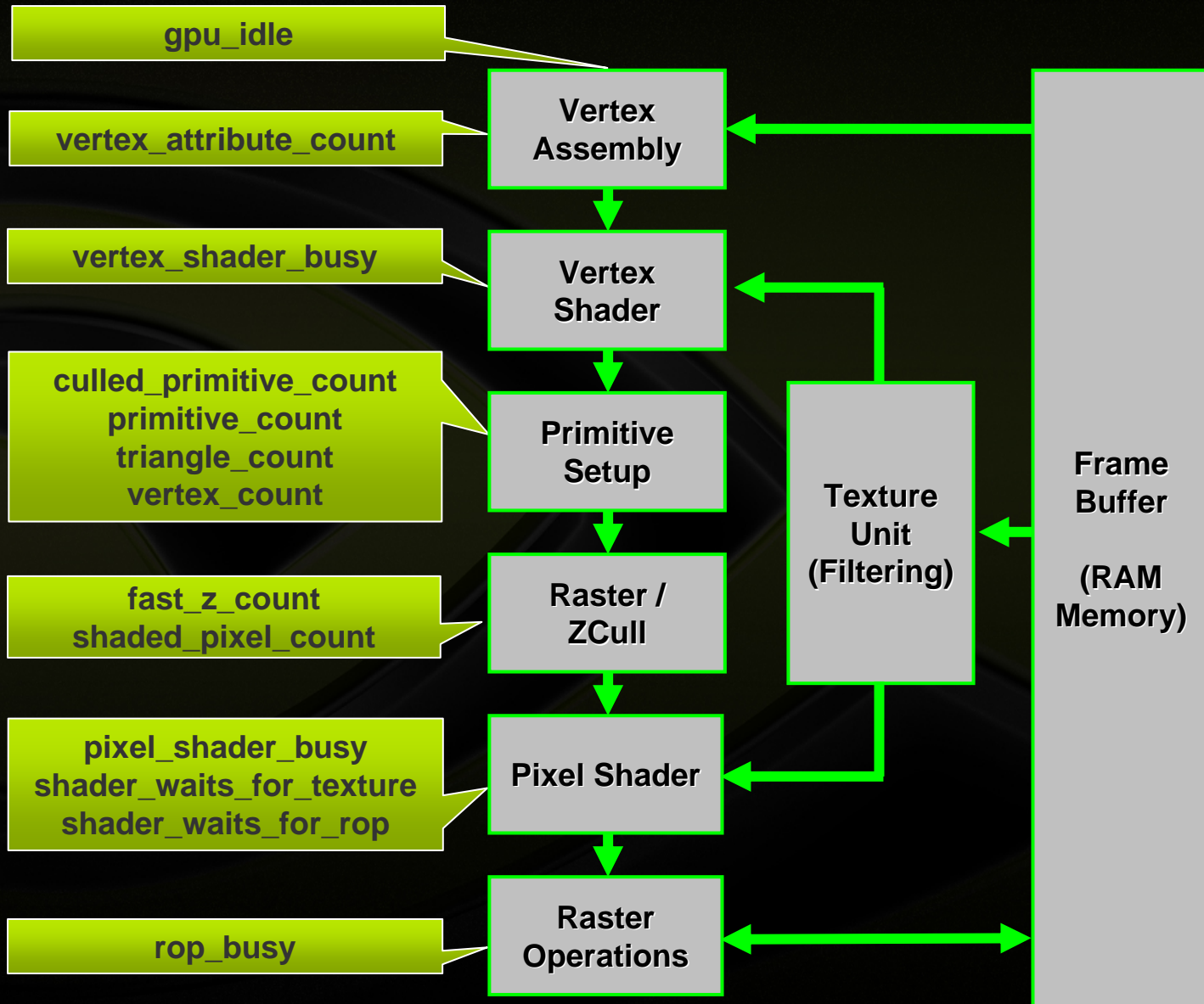
## ● Counts

- Batches, vertices, primitives
- (Direct3D) Triangles and instanced triangles
- (Direct3D) Locked render targets

## ● Memory

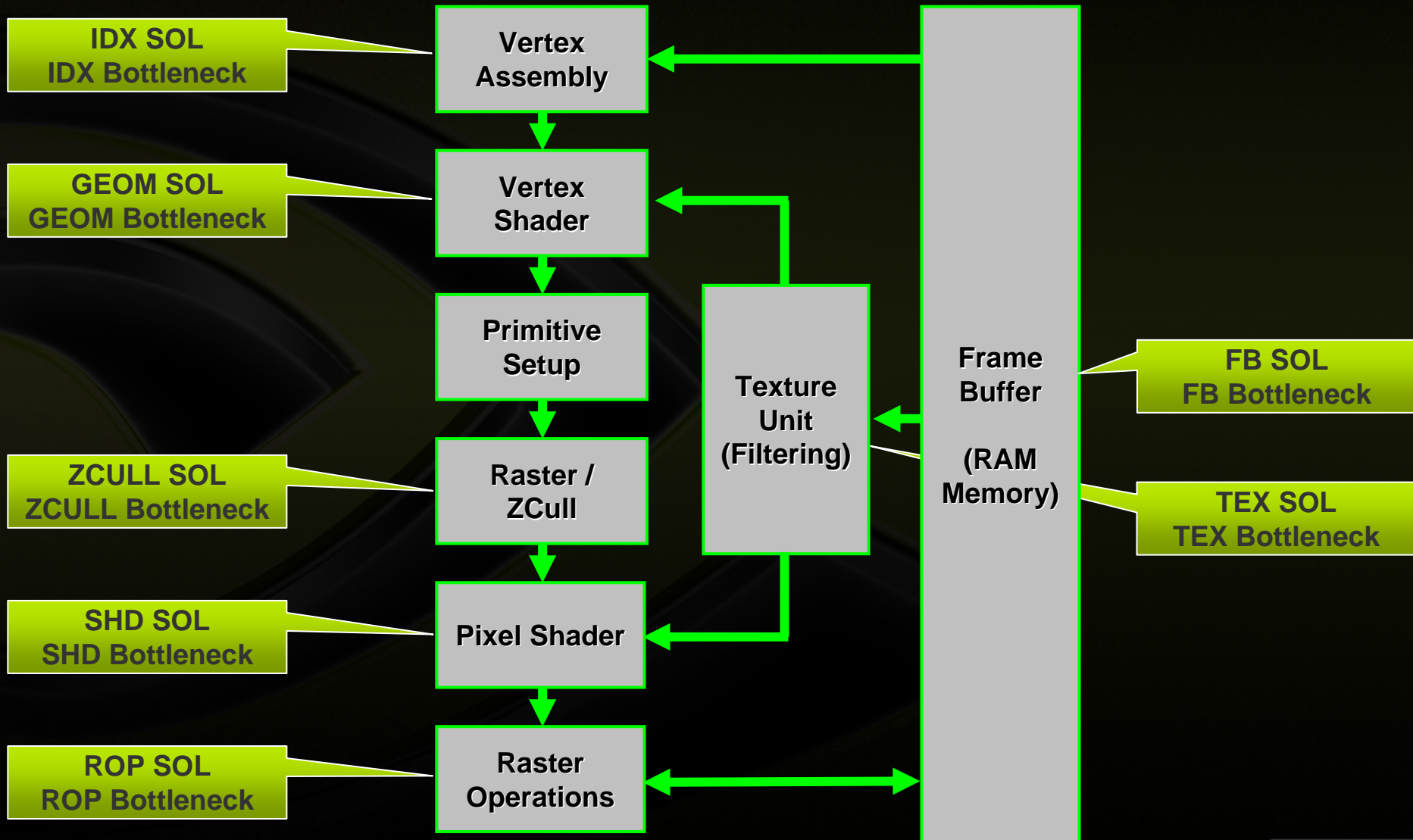
- AGP memory used
- Video memory used and total

# Realtime GPU Counters





# Simplified Experiments



# What is NVPerfHUD?



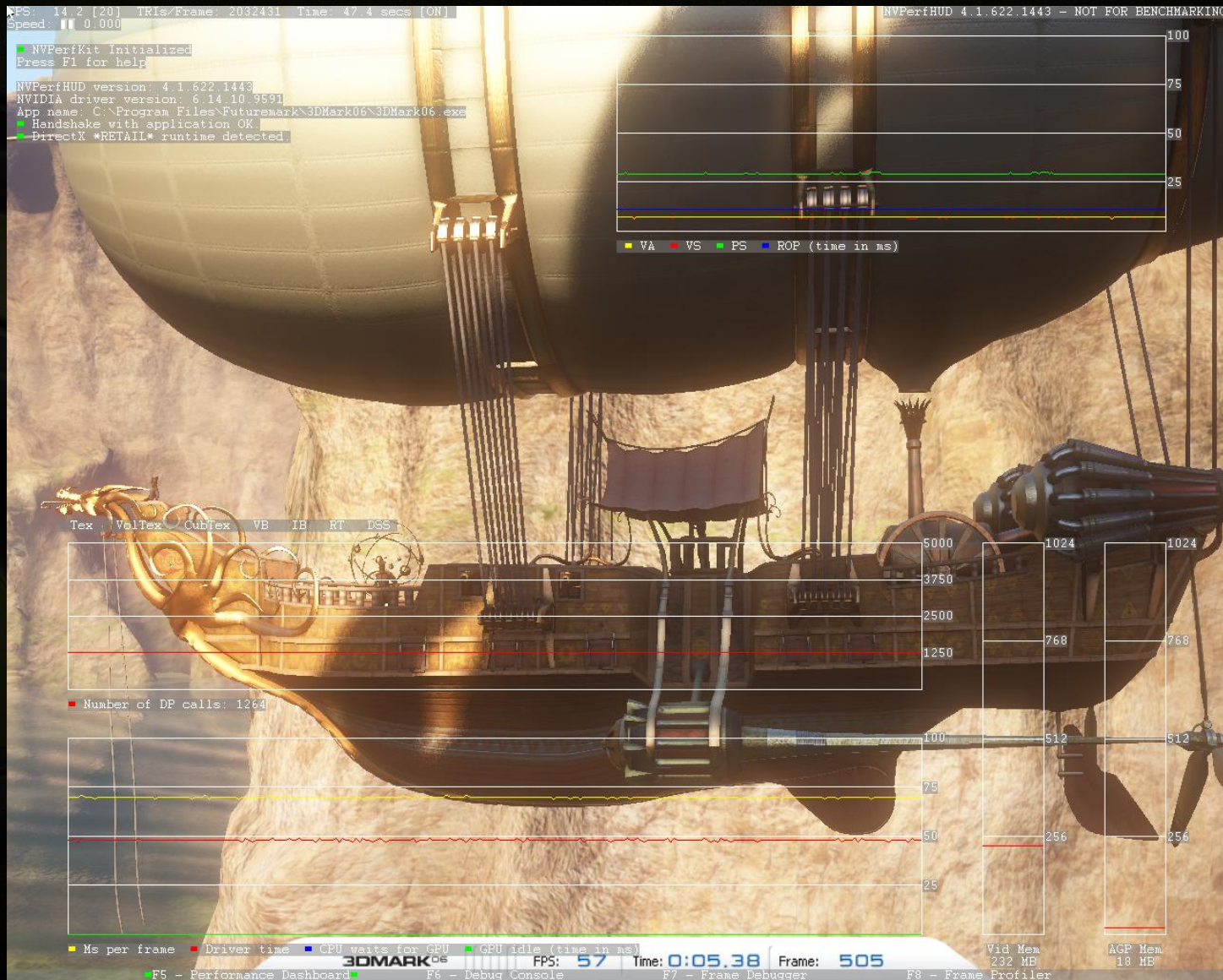
- **Direct3D Performance and Debugging Tool**
  - Overlay graphs and debugging tools on top of your application
  - Interactive tools for debugging and performance tuning
- **4 different screens**
  - Performance Dashboard
  - Debug Console
  - Frame Debugger
  - Frame Profiler (New in 4.0)

# How to use it

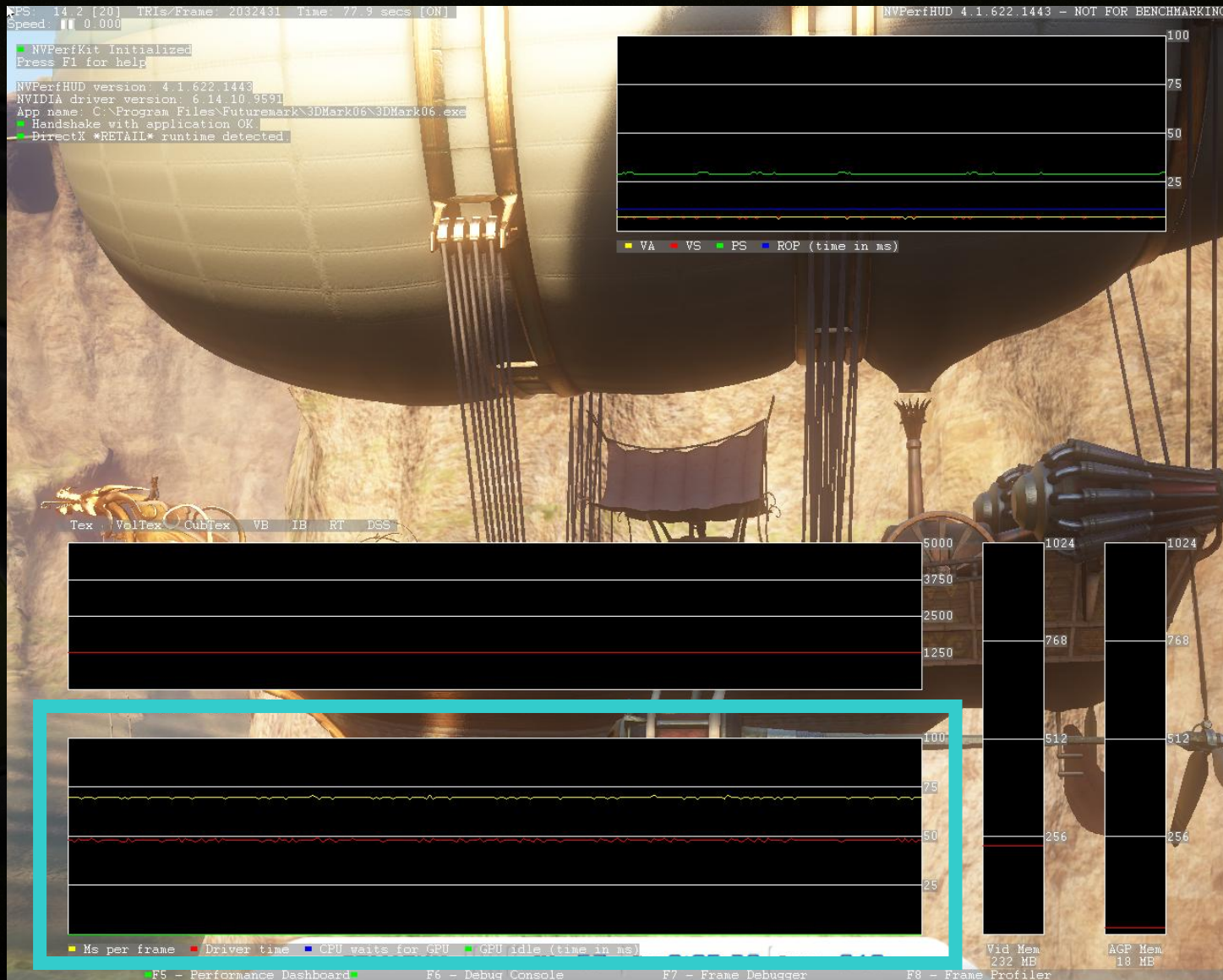


- **Drag and drop your application onto the NVPerfHUD icon**
- **Run through your application as you normally do until you find:**
  - **Functional problems: use the Frame Debugger**
  - **Performance problems: use the Dashboard graphs and Frame Profiler**

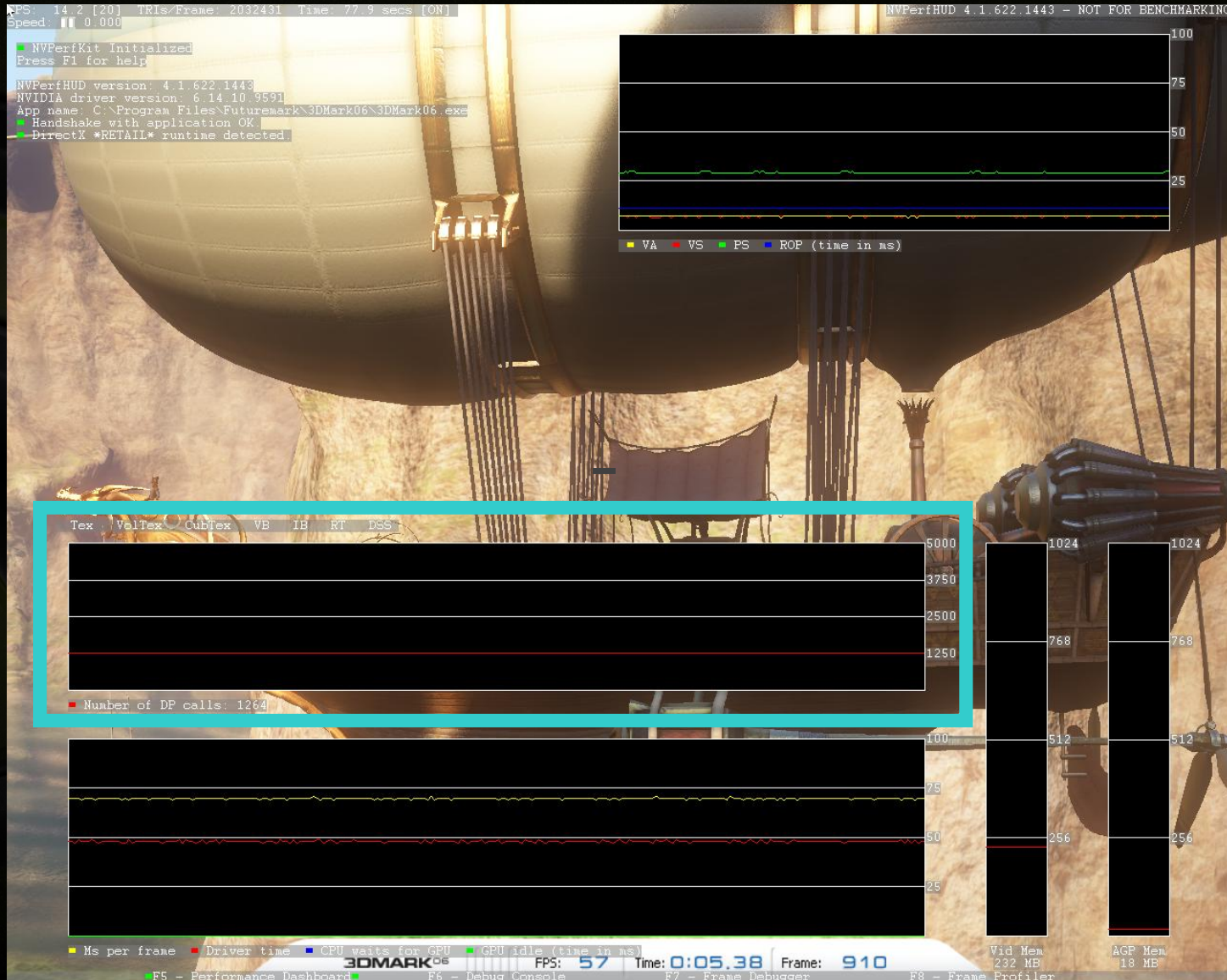
# Demo: NVPerfHUD



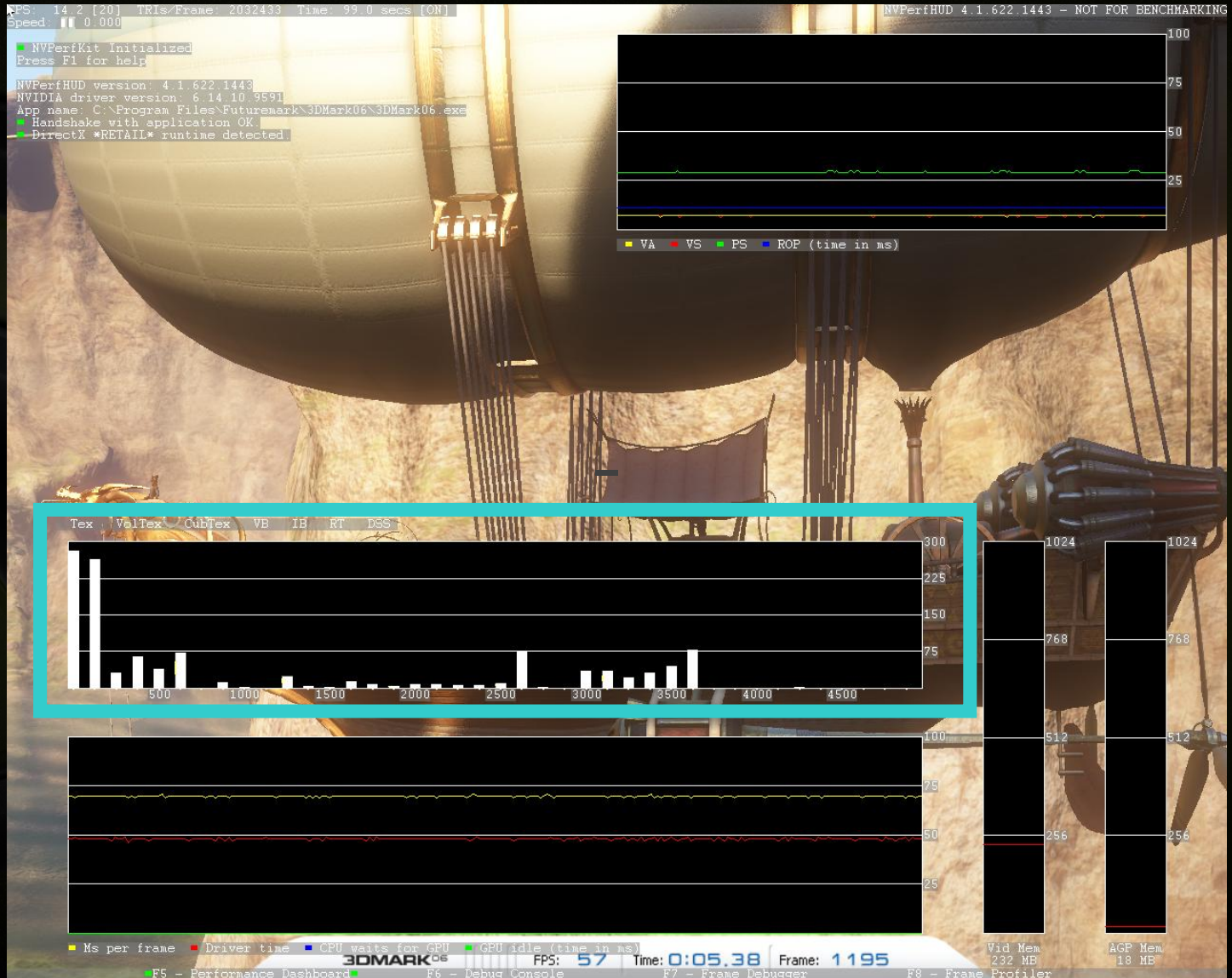
# Demo: Performance Dashboard



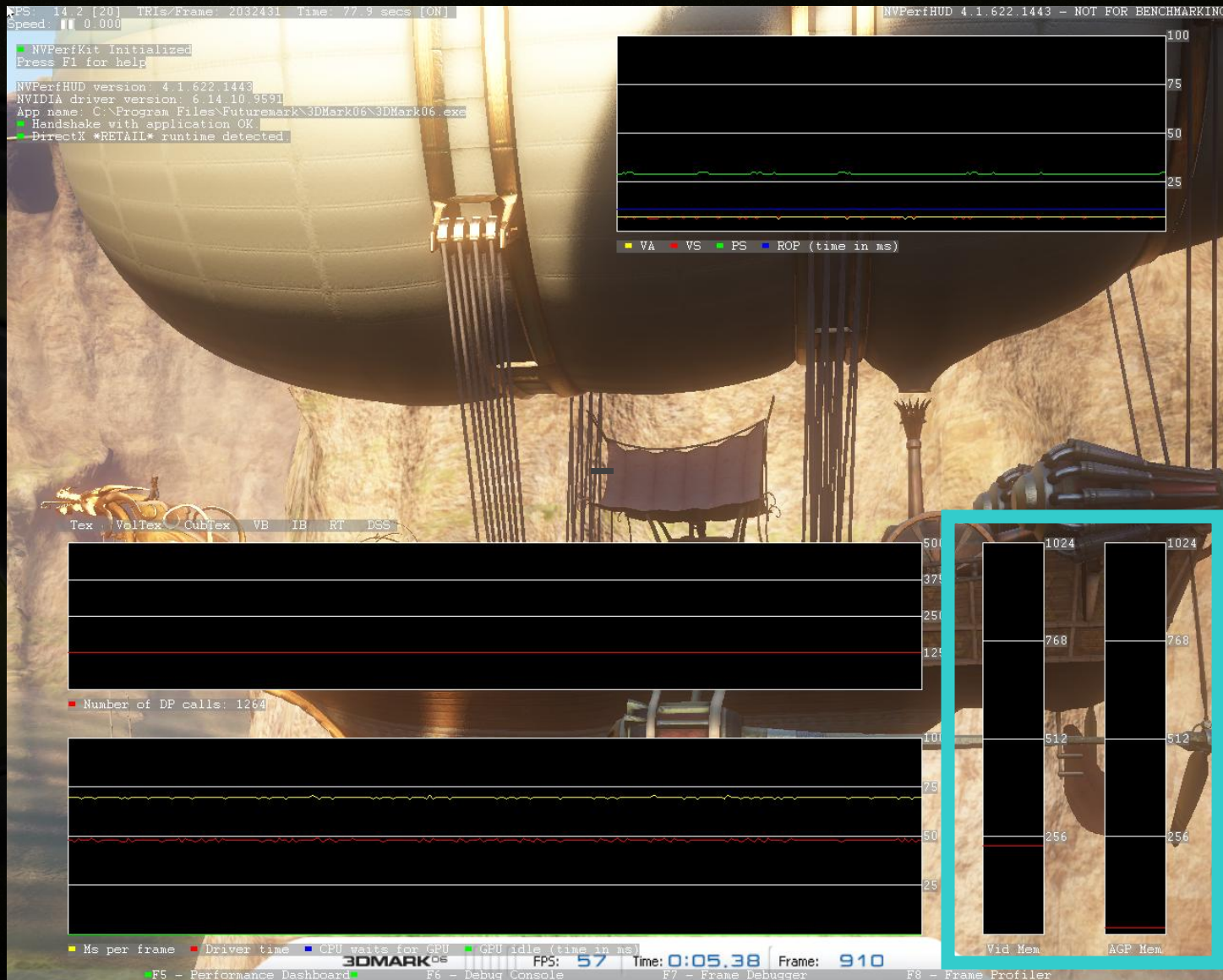
# Demo: Performance Dashboard



# Demo: Performance Dashboard

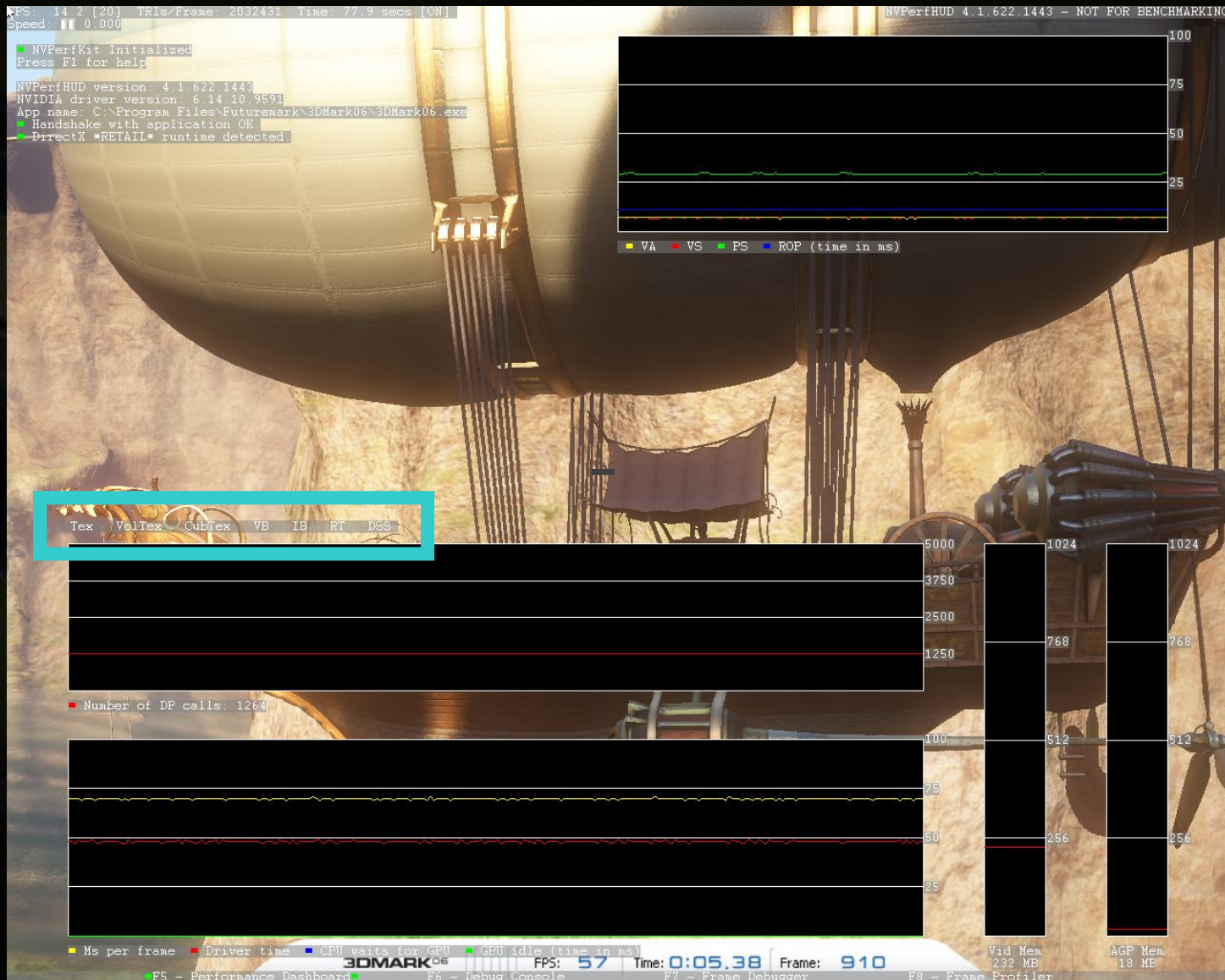


# Demo: Performance Dashboard





# Demo: Performance Dashboard



# Demo: Performance Dashboard

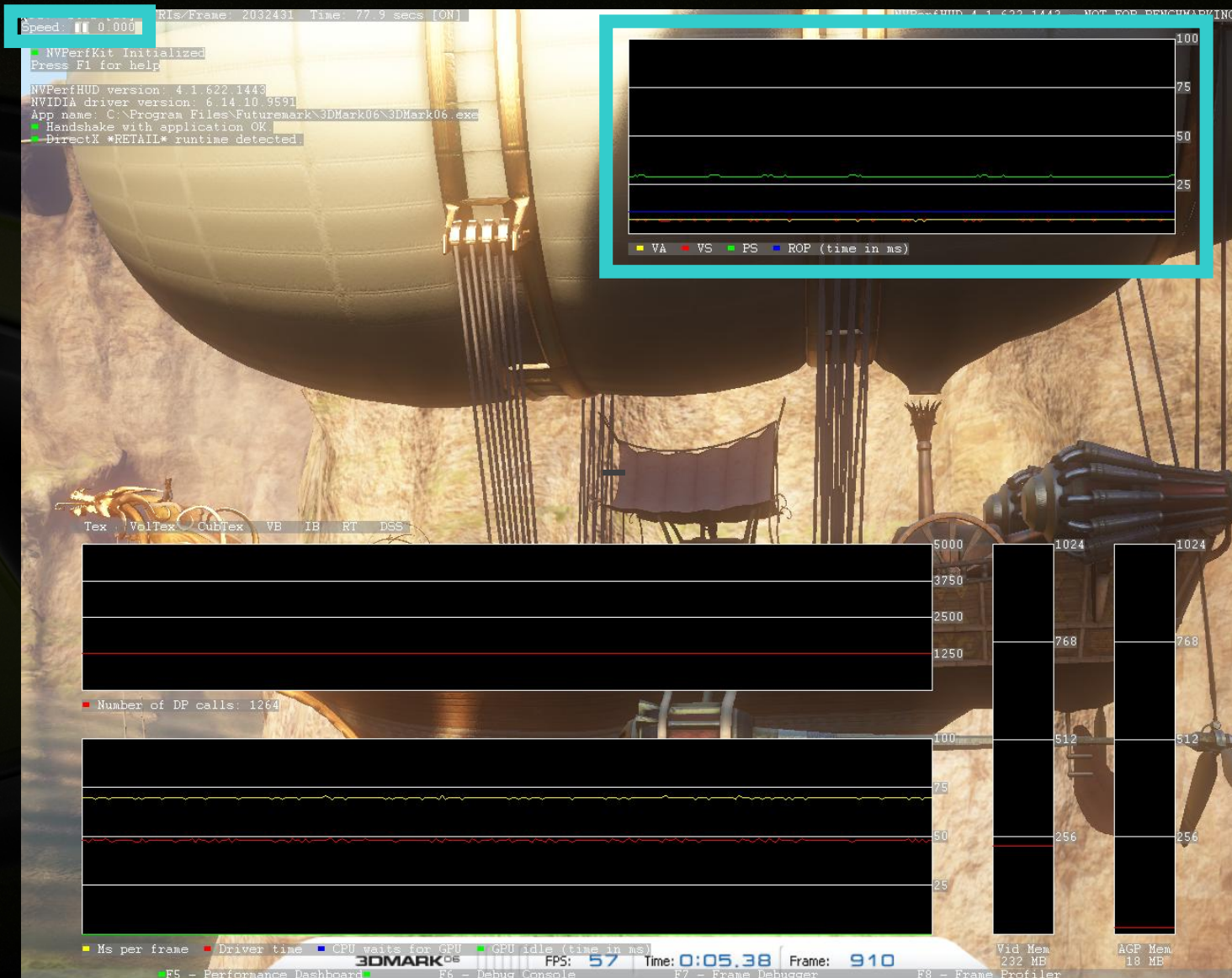


Resource creation monitor



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

# Demo: Performance Dashboard



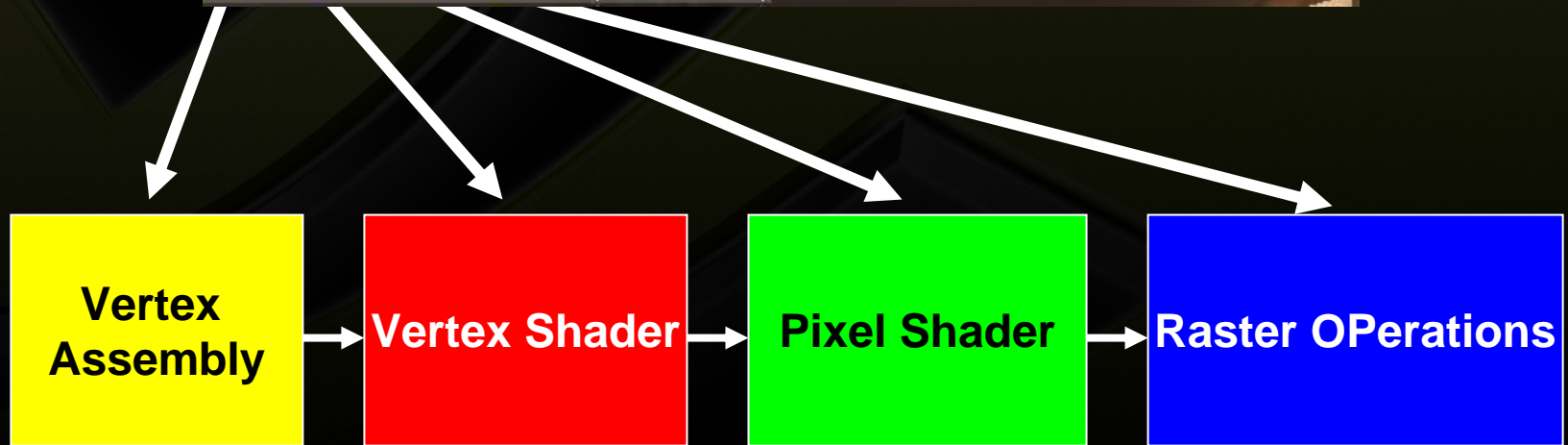
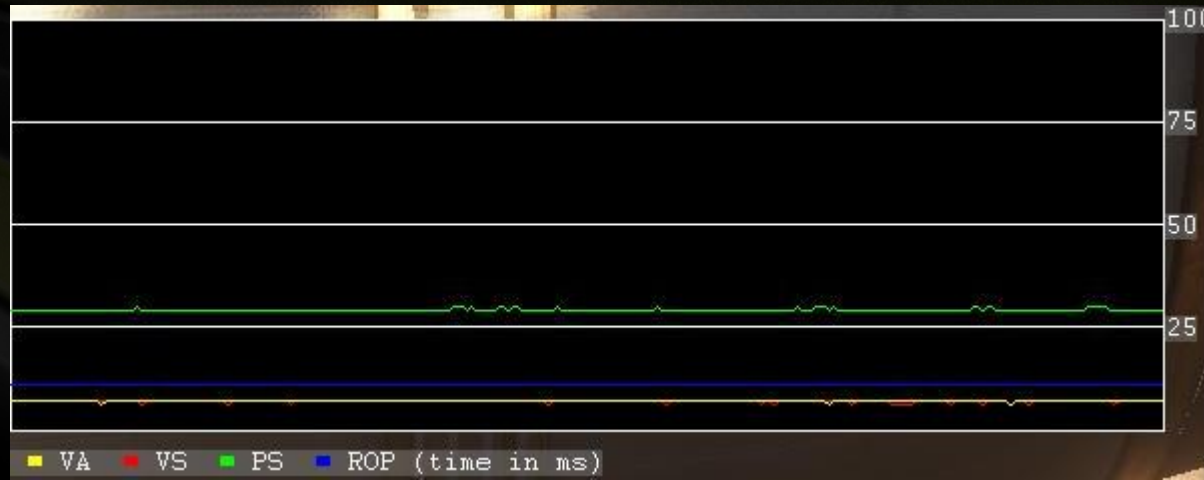
# Demo: Performance Dashboard



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\
```

# Demo: The simplified graphics pipeline



# Demo: Debug Console



FPS: 14.1 [20] IRIs/Frame: 2040593 Time: 71.5 secs [ON] NVPerfHUD 4.1.622.1443 - NOT FOR BENCHMARKING  
Speed: 0.000

```
Time: 5137.97 ms, IDirect3DDevice9: CreateVertexBuffer(1740,8,0,1)
Time: 5138.04 ms, IDirect3DDevice9: CreateIndexBuffer(1344,8,INDEX16,1)
Time: 5284.50 ms, IDirect3DDevice9: CreateTexture(2048x2048,12,0,DXT3,1)
Time: 5419.50 ms, IDirect3DDevice9: CreateTexture(2048x2048,12,0,DXT5,1)
Time: 5568.65 ms, IDirect3DDevice9: CreateTexture(2048x2048,12,0,DXT3,1)
Time: 5578.64 ms, IDirect3DDevice9: CreateVertexBuffer(182496,8,0,1)
Time: 5578.92 ms, IDirect3DDevice9: CreateVertexBuffer(212912,8,0,1)
Time: 5579.12 ms, IDirect3DDevice9: CreateIndexBuffer(40704,8,INDEX16,1)
Time: 5631.39 ms, IDirect3DDevice9: CreateVertexBuffer(720,8,0,1)
Time: 5631.47 ms, IDirect3DDevice9: CreateVertexBuffer(1680,8,0,1)
Time: 5631.52 ms, IDirect3DDevice9: CreateIndexBuffer(180,8,INDEX16,1)
Time: 5632.02 ms, IDirect3DDevice9: CreateVertexBuffer(816,8,0,1)
Time: 5632.06 ms, IDirect3DDevice9: CreateVertexBuffer(1904,8,0,1)
Time: 5632.10 ms, IDirect3DDevice9: CreateIndexBuffer(204,8,INDEX16,1)
Time: 5635.67 ms, IDirect3DDevice9: CreateVertexBuffer(12408,8,0,1)
Time: 5635.74 ms, IDirect3DDevice9: CreateVertexBuffer(14476,8,0,1)
Time: 5635.79 ms, IDirect3DDevice9: CreateIndexBuffer(3396,8,INDEX16,1)
Time: 5692.70 ms, IDirect3DDevice9: CreateTexture(1024x128,1,0,L16,1)
Time: 5734.37 ms, IDirect3DDevice9: CreateTexture(1024x256,11,0,A8R8G8B8,1)
Time: 5741.84 ms, IDirect3DDevice9: CreateTexture(1x1,1,0,L8,1)
Time: 5761.01 ms, IDirect3DDevice9: CreateTexture(1024x256,11,0,DXT5,1)
Time: 5763.27 ms, IDirect3DDevice9: CreateVertexBuffer(320,8,0,1)
```

3DMARK06 FPS: 92 Time: 0:05.48 Frame: 767  
F5 - Performance Dashboard F6 - Debug Console F7 - Frame Debugger F8 - Frame Profiler

Clear Log Each Frame  
Stop Logging  
Fade Console  
Display only  
Application  
NVPerfHUD  
Save

# Demo: Frame Debugger



FPS: 16.3 [20] TRIs/Frame: 2040595 Time: 87.1 secs [ON] NVPerfHUD 4.1.622.1443 - NOT FOR BENCHMARKING

Nothing

Sampler: s0  
Type: RT\_TEXTURE  
2048x2048, R32F  
MIPs: 1 MIP: NONE  
Mag: PNT Min: PNT  
Aniso: 1

Sampler: s1  
Type: TEXTURE  
32x32, G8/8B/8/8  
MIPs: 1 MIP: NONE  
Mag: PNT Min: PNT  
Aniso: 1

Sampler: s2  
Type: TEXTURE  
1024x512, DXT1  
MIPs: 11 MIP: LINEAR  
Mag: LIN Min: LIN  
Aniso: 1

Sampler: s3  
Type: TEXTURE  
1024x1024, DXT1  
MIPs: 11 MIP: LINEAR  
Mag: LIN Min: LIN  
Aniso: 1

Sampler: s4  
Type: TEXTURE  
1024x1024, DXT1  
MIPs: 11 MIP: LINEAR  
Mag: LIN Min: LIN  
Aniso: 1

Sampler: s5  
Type: TEXTURE  
1024x1024, DXT5  
MIPs: 11 MIP: LINEAR  
Mag: LIN Min: LIN  
Aniso: 1

Type: RT\_TEXTURE  
1280x1024, A16B16G16R16F  
MIPs: 1

Prims Drawn: 1679953 Warnings: 72  
DrawIndexedPrimitive(D3DPT\_TRIANGLELIST, 0, 0, 2739, 3000, 2670)  
RT: 0x126a7ba0 BB: 0x00195b00

Step Back Step Forward Draw Call 1000/1259

Frame Buffer Export... Advanced... Hide All

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

# Demo: Advanced Frame Debug



FPS: 18.9 (20) TRS: Frame: 204955 - Time: 129.0 sec. [OK] WFFrameID: 4:1:622:1443 - NOT FOR DEMARKING

Vertex Shader | Vertex Shader | Pixel Shader | Pixel Operations

Vertex Shader

```
Call User DrawState::Render()
Type: D3D11_VERTEX_SHADER
Name: VertexShader
...

```

Pixel Shader

```
Call User DrawState::Render()
Type: D3D11_PIXEL_SHADER
Name: PixelShader
...

```

Pixel Operations

```
Call User DrawState::Render()
Type: D3D11_PIXEL_SHADER
Name: PixelShader
...

```

Draw Call 1000/1259

FS - Performance Dashboard | FS - Debug Console | FS - Frame Debugger | FS - Frame Pooler

FPS: 18.9 (20) TRS: Frame: 204955 - Time: 130.0 sec. [OK] WFFrameID: 4:1:622:1443 - NOT FOR DEMARKING

Vertex Shader | Vertex Shader | Pixel Shader | Pixel Operations

Vertex Shader

```
Call User DrawState::Render()
Type: D3D11_VERTEX_SHADER
Name: VertexShader
...

```

Pixel Shader

```
Call User DrawState::Render()
Type: D3D11_PIXEL_SHADER
Name: PixelShader
...

```

Pixel Operations

```
Call User DrawState::Render()
Type: D3D11_PIXEL_SHADER
Name: PixelShader
...

```

Draw Call 1000/1259

FS - Performance Dashboard | FS - Debug Console | FS - Frame Debugger | FS - Frame Pooler

FPS: 18.9 (20) TRS: Frame: 204955 - Time: 149.0 sec. [OK] WFFrameID: 4:1:622:1443 - NOT FOR DEMARKING

Vertex Shader | Vertex Shader | Pixel Shader | Pixel Operations

Vertex Shader

```
Call User DrawState::Render()
Type: D3D11_VERTEX_SHADER
Name: VertexShader
...

```

Pixel Shader

```
Call User DrawState::Render()
Type: D3D11_PIXEL_SHADER
Name: PixelShader
...

```

Pixel Operations

```
Call User DrawState::Render()
Type: D3D11_PIXEL_SHADER
Name: PixelShader
...

```

Draw Call 1000/1259

FS - Performance Dashboard | FS - Debug Console | FS - Frame Debugger | FS - Frame Pooler

FPS: 18.9 (20) TRS: Frame: 204955 - Time: 149.0 sec. [OK] WFFrameID: 4:1:622:1443 - NOT FOR DEMARKING

Vertex Shader | Vertex Shader | Pixel Shader | Pixel Operations

Vertex Shader

```
Call User DrawState::Render()
Type: D3D11_VERTEX_SHADER
Name: VertexShader
...

```

Pixel Shader

```
Call User DrawState::Render()
Type: D3D11_PIXEL_SHADER
Name: PixelShader
...

```

Pixel Operations

```
Call User DrawState::Render()
Type: D3D11_PIXEL_SHADER
Name: PixelShader
...

```

Draw Call 1000/1259

FS - Performance Dashboard | FS - Debug Console | FS - Frame Debugger | FS - Frame Pooler



# Frame Profiler



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

# Frame Profiler: Optimization Strategy



- **Group by render state (“state buckets”):** helps show most expensive states to render
- **Identify the bottleneck for the most expensive state bucket**
- **Curing the bottleneck with a common corrective action should result in increased performance**
- **Iterate...**

# Demo: Frame Profiler



FPS: 23.3 [20] TRIs/Frame: 2040595 Time: 163.9 secs [ON]

NVPerfHUD 4.1.622.1443 - NOT FOR BENCHMARKING

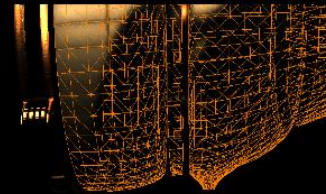
State buckets:

Vertex Shader  Pixel Shader  ROP  RT  Pixel Shader Constants

Calls	Time v	Pixels
56	11.337 ms	7846415
6	7.230 ms	5049927
76	5.890 ms	4100818
269	4.670 ms	3250771
37	4.605 ms	3221085
9	3.204 ms	2206133
51	2.655 ms	1853924
198	2.476 ms	1720619
9	2.430 ms	1691619
5	2.104 ms	1451777
76	1.776 ms	1238466

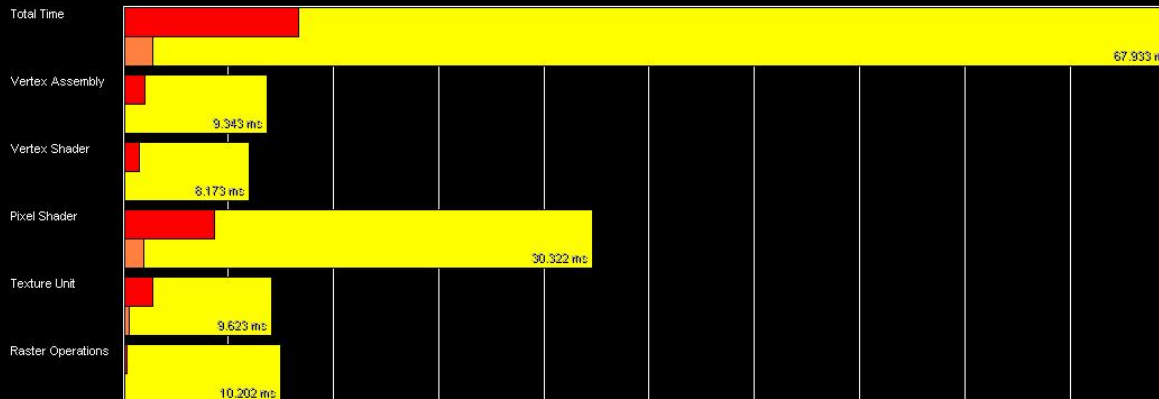
Draw calls in selected State Bucket:

Ord	Prims	Time v	Pixels
798	2552	1.870 ms	1301868
805	9086	1.480 ms	1041456
799	2082	1.270 ms	888649
806	8088	1.224 ms	851734
804	21752	0.759 ms	524753
807	11904	0.721 ms	495232
802	9917	0.605 ms	416983
808	3704	0.448 ms	306129
795	6214	0.429 ms	291468
803	10370	0.427 ms	274364
796	3568	0.355 ms	252653



Type: RT\_TEXTURE  
1280x1024, A16B16G16R16F  
MIPs:1

Unit Utilization Bars



Step Back

Step Forward

Draw Call 798/1259

Frame Buffer

Export...

Advanced...

Hide All

F5 - Performance Dashboard

F6 - Debug Console

F7 - Frame Debugger

F8 - Frame Profiler



# Demo: Advanced Frame Profiler



FPS: 23.4 (20) | T14: Frame: 204555 | Task: 242.2 sec: [CPU]

Draw Call: 98 | Time: 11.07 sec | Frame: 1259

Draw: 1.07 sec | Frame: 1259

47.82%

Vertex Assembly | Vertex Shader | Pixel Shader | Post-Operations

Vertex Shader

```
Vertex Shader Description **
Path: D:\FX\...
Type: D3DPS_1_0_SHADER
Format: X8X8X8
Precision: 100
Length: 10700 bytes
...

```

FPS: 23.4 (20) | T14: Frame: 204555 | Task: 242.2 sec: [CPU]

Draw Call: 98 | Time: 11.07 sec | Frame: 1259

Draw: 1.07 sec | Frame: 1259

47.82%

Vertex Shader

```
Vertex Shader Description **
Path: D:\FX\...
Type: D3DPS_1_0_SHADER
Format: X8X8X8
Precision: 100
Length: 10700 bytes
...

```

Vertex Shader Constants

```
Flighting Point Constants
C(0) = 0.000000 0.000000 0.000000 0.000000
C(1) = 0.000000 0.000000 0.000000 0.000000
...

```

Render Targets (4 of 20)

FPS: 24.6 (20) | T14: Frame: 204555 | Task: 245.0 sec: [CPU]

Draw Call: 98 | Time: 12.07 sec | Frame: 1259

Draw: 1.07 sec | Frame: 1259

47.82%

Vertex Assembly | Vertex Shader | Pixel Shader | Post-Operations

Pixel Shader

```
Pixel Shader Description **
Path: D:\FX\...
Type: D3DPS_1_0_SHADER
Format: X8X8X8
Precision: 100
Length: 10700 bytes
...

```

Render Targets (4 of 20)

Sampler 00

```
Type: TEXTURE
Texture: D:\FX\...
Mip: 11 MIP LINEAR
Mag: LINEAR MIP LINEAR
Aniso: 1

```

Sampler 01

```
Type: TEXTURE
Texture: D:\FX\...
Mip: 11 MIP LINEAR
Mag: POINT MIP POINT
Aniso: 1

```

Sampler 02

```
Type: TEXTURE
Texture: D:\FX\...
Mip: 11 MIP LINEAR
Mag: LINEAR MIP LINEAR
Aniso: 1

```

Sampler 03

```
Type: TEXTURE
Texture: D:\FX\...
Mip: 11 MIP LINEAR
Mag: LINEAR MIP LINEAR
Aniso: 1

```

Sampler 04

```
Type: TEXTURE
Texture: D:\FX\...
Mip: 11 MIP LINEAR
Mag: LINEAR MIP LINEAR
Aniso: 1

```

Pixel Shader Constants

FPS: 23.3 (20) | T14: Frame: 204555 | Task: 250.0 sec: [CPU]

Draw Call: 98 | Time: 12.07 sec | Frame: 1259

Draw: 1.07 sec | Frame: 1259

47.82%

Vertex Assembly | Vertex Shader | Pixel Shader | Post-Operations

Render Targets (4 of 20)

Render State

```
Blend
BLEND = BLEND
BLENDMODE = SRCALPHA
BLENDOP = ADD
...

```

Render Targets (4 of 20)

Type: D3DRT\_10\_TEXTURE

1300x1000, ARGB8888, 16BIT

# Freezing the application



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

# How do I use NVPerfKit counters?



- **PDH: Performance Data Helper for Windows**
  - Win32 API for exposing performance data to user applications
  - Standard interface, many providers and clients
  - Sample code and helper classes provided in NVPerfSDK
- **Perfmon: (aka Microsoft Management Console)**
  - Win32 PDH client application
  - Perfmon's sampling frequency is low (1X/s)
  - Displays PDH based counter values:
    - OS: CPU usage, memory usage, swap file usage, network stats, etc.
    - NVIDIA: all of the counters exported by NVPerfKit
- **Good for rapid prototyping**

# Enable counters: NVDevCPL



**NVIDIA Developer Control Panel**

**Available Counters**

- vertex\_attribute\_count
- vertex\_count
- vertex\_shader\_busy
- GPU\_Shader**
- fast\_z\_count
- pixel\_shader\_busy
- rop\_busy
- shader\_waits\_for\_rop
- shader\_waits\_for\_texture
- texture\_waits\_for\_shader
- OGL
- CPU
- OGL AGP/PCI-E usage (bytes)
- OGL AGP/PCI-E usage (MB)
- OGL driver sleeping
- OGL FPS
- OGL Frame Batch Count
- OGL Frame Primitive Count

**Active Counters**

Buttons: Add >>, Remove <<, Save..., Load..., Clear

**Counter Description**

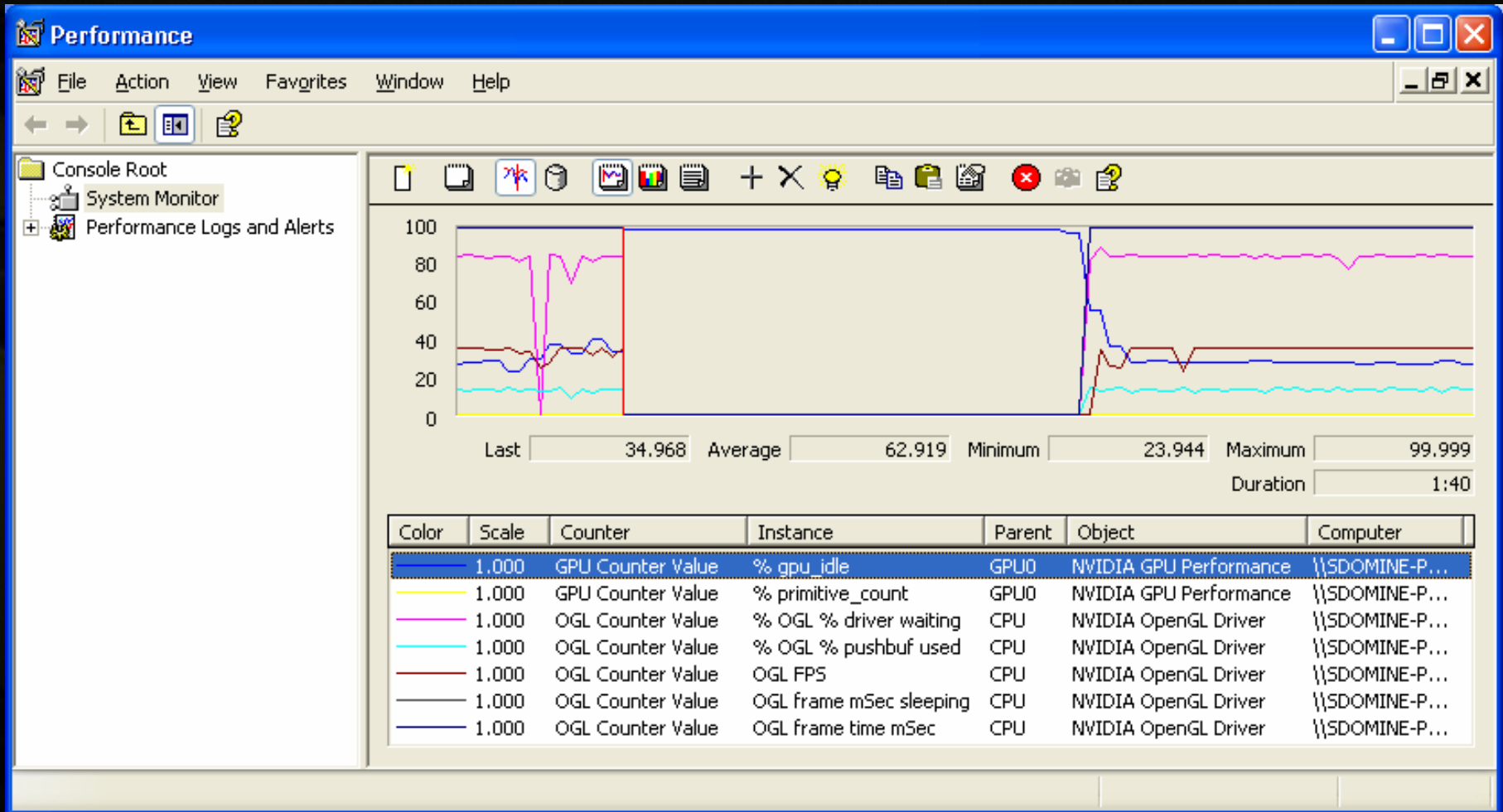
D3D time in driver : D3D last frame mSec spent in D3D driver

**Settings**

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

Buttons: OK, Cancel, Apply

# Graphing results: Perfmon







- **NVIDIA API for easy integration of NVPerfKit**
  - No more enable counters in NVDevCPL, run app separately
  - No more lag from PDH
- **Simplified Experiments**
  - Targeted, multipass experiments to determine GPU bottleneck
  - Automated analysis of results to show bottlenecked unit
- **Use cases**
  - Real time performance monitoring using GPU and driver counters, round robin sampling
  - Simplified Experiments for single frame analysis

# NVPerfAPI: Real Time



```
// Somewhere in setup
NVPMAddCounterByName("vertex_shader_busy");
NVPMAddCounterByName("pixel_shader_busy");
NVPMAddCounterByName("shader_waits_for_texture");
NVPMAddCounterByName("gpu_idle");

// In your rendering loop, sample using names
NVPMSample(NULL, &nNumSamples);
NVPMGetCounterValueByName("vertex_shader_busy", 0, &nVSEvents,
    &nVSCycles);
NVPMGetCounterValueByName("pixel_shader_busy", 0, &nPSEvents,
    &nPSCycles);
NVPMGetCounterValueByName("shader_waits_for_texture", 0,
    &nTexEvents, &nTexCycles);
NVPMGetCounterValueByName("gpu_idle", 0, &nIdleEvents,
    &nIdleCycles);
```

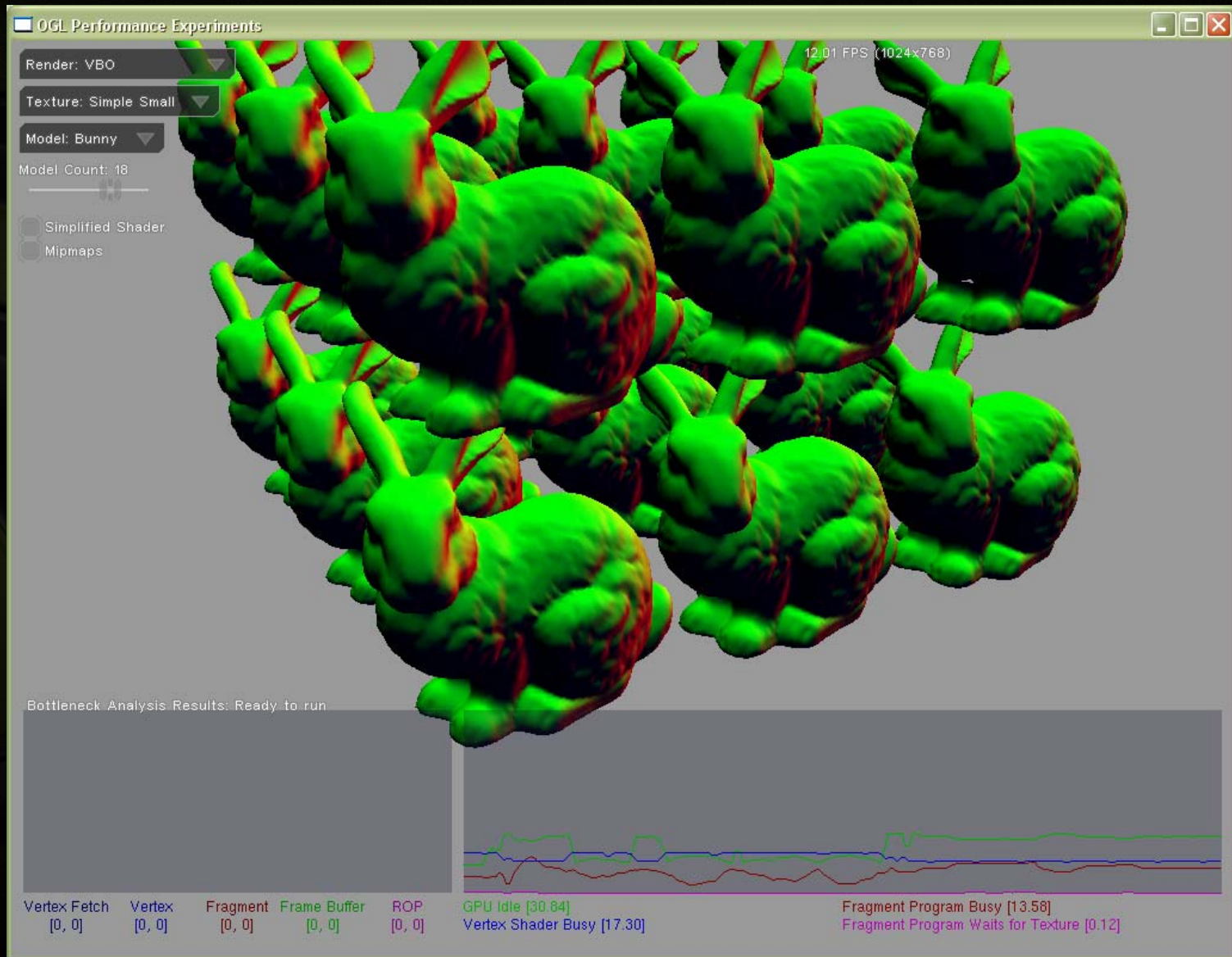
# NVPerfAPI: Real Time



```
// Somewhere in setup
nVSBusy = NVPMGetCounterByName("vertex_shader_busy");
NVPMAddCounter(nVSBusy);
nPSBusy = NVPMGetCounterByName("pixel_shader_busy");
NVPMAddCounter(nPSBusy);
nWaitTexture = NVPMGetCounterByName("shader_waits_for_texture");
NVPMAddCounter(nWaitTexture);
nGPUIdle = NVPMGetCounterByName("gpu_idle");
NVPMAddCounter(nGPUIdle);

// In your rendering loop, sample using IDs
NVPMSample(aSamples, &nNumSamples);
for(ii = 0; ii < nNumSamples; ++ii) {
    if(aSamples[ii].index == nVSBusy) {
    }
    if(aSamples[ii].index == nPSBusy) {
    }
    if(aSamples[ii].index == nWaitTexture) {
    }
    if(aSamples[ii].index == nGPUIdle) {
    }
}
```

# NVPerfAPI: Real time sampling



# NVPerfAPI: Simplified Experiments



```
NVPMAddCounter("GPU Bottleneck");
NVPMAAllocObjects(50);

NVPMBeginExperiment(&nNumPasses);
for(int ii = 0; ii < nNumPasses; ++ii) {
    // Setup the scene, clear Zbuffer/render target
    NVPMBeginPass(ii);

    NVPMBeginObject(0);
    // Draw calls associated with object 0 and flush
    NVPMEndObject(0);

    NVPMBeginObject(1);
    // Draw calls associated with object 1 and flush
    NVPMEndObject(1);

    // ...
    NVPMEndPass(ii);
}

NVPMEndExperiment();
NVPMGetCounterValueByName("GPU Bottleneck", 0, &nGPUBneck, &nGPUCycles);
NVPMGetGPUBottleneckName(nGPUBneck, pcString); // Convert to name

// End scene/present/swap buffers
```

# NVPerfAPI: Simplified Experiments



- **GPU Bottleneck experiment**
  - Run bottleneck and utilization experiments on all units
  - Process results to find bottlenecked unit
- Individual unit information can be queried
- Can run individual unit experiments
- Events: % utilization or % bottleneck...best way to visualize data
- Cycles: microseconds that the experiment ran, helps recompute the numerator for sorting

```
NVPMGetCounterValueByName("IDX BNeck", 0, &nIDXBneckEvents,  
    &nIDXBNeckCycles);
```

```
NVPMGetCounterValueByName("IDX SOL", 0, &nIDXSOLEvents,  
    &nIDXSOLECycles);
```

# NVPerfAPI: SimExp



Applications Actions Tue May 16, 10:28 PM

dcornish@pt-test-amdx2:~/NVIDIA/sw/nvr90/apps/performance/NVPerfKit/NVPerfSDK/Samples/OGLPerfHarness

File Edit View Terminal Tabs Help

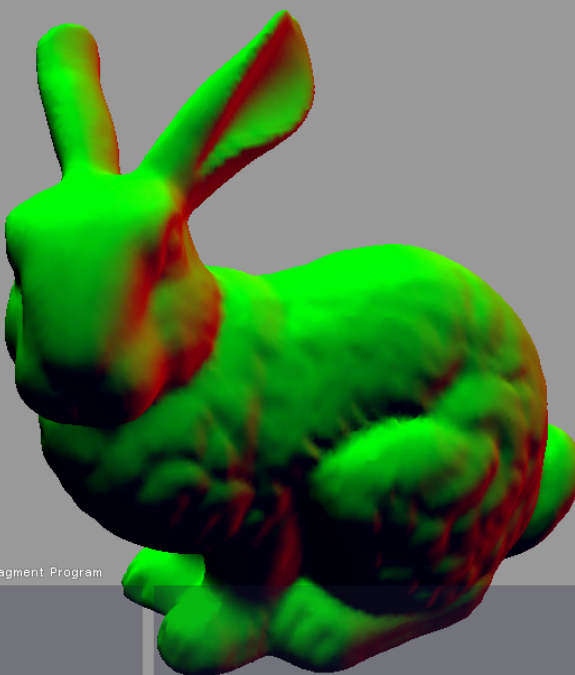
```
total 96
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
[dcornish@pt-
[dcornish@pt-
[dcornish@pt-
[dcornish@pt-
[dcornish@pt-
[dcornish@pt-
total 16
drwxr-xr-x
drwxr-xr-x
[dcornish@pt-
[dcornish@pt-
[dcornish@pt-
[dcornish@pt-
[dcornish@pt-
[dcornish@pt-
g++ -o ../bin
UI/GUI/GUICh
rce/SXMLGUI/
o Source/SXM
UITexCoordDes
XMLGUI/Tools,
-IGLU -lglu
Source/SXMLGI
: undefined
collect2: ld
make: *** [.
[dcornish@pt-
g++ -o ../bin
UI/GUI/GUICh
rce/SXMLGUI/
o Source/SXM
UITexCoordDes
XMLGUI/Tools,
-IGLU -lglu
/usr/bin/ld:
collect2: ld
make: *** [.
[dcornish@pt-
make: *** No
[dcornish@pt-
g++ -o ../bin
UI/GUI/GUICh
rce/SXMLGUI/
o Source/SXM
UITexCoordDes
XMLGUI/Tools,
-IGLU -lglu -INVPerfSDK Libs/libglpng.a
[dcornish@pt-test-amdx2 OGLPerfHarness]$
[dcornish@pt-test-amdx2 OGLPerfHarness]$
[dcornish@pt-test-amdx2 OGLPerfHarness]$
[dcornish@pt-test-amdx2 OGLPerfHarness]$
[dcornish@pt-test-amdx2 OGLPerfHarness]$
[dcornish@pt-test-amdx2 OGLPerfHarness]$
```

OGL Performance Experiments

Render: VBO Texture: Simple Large Model: Bunny Model Count: 1

82.9741 FPS (1024x768) 69451 Primitives

Simplified Shader Mipmaps



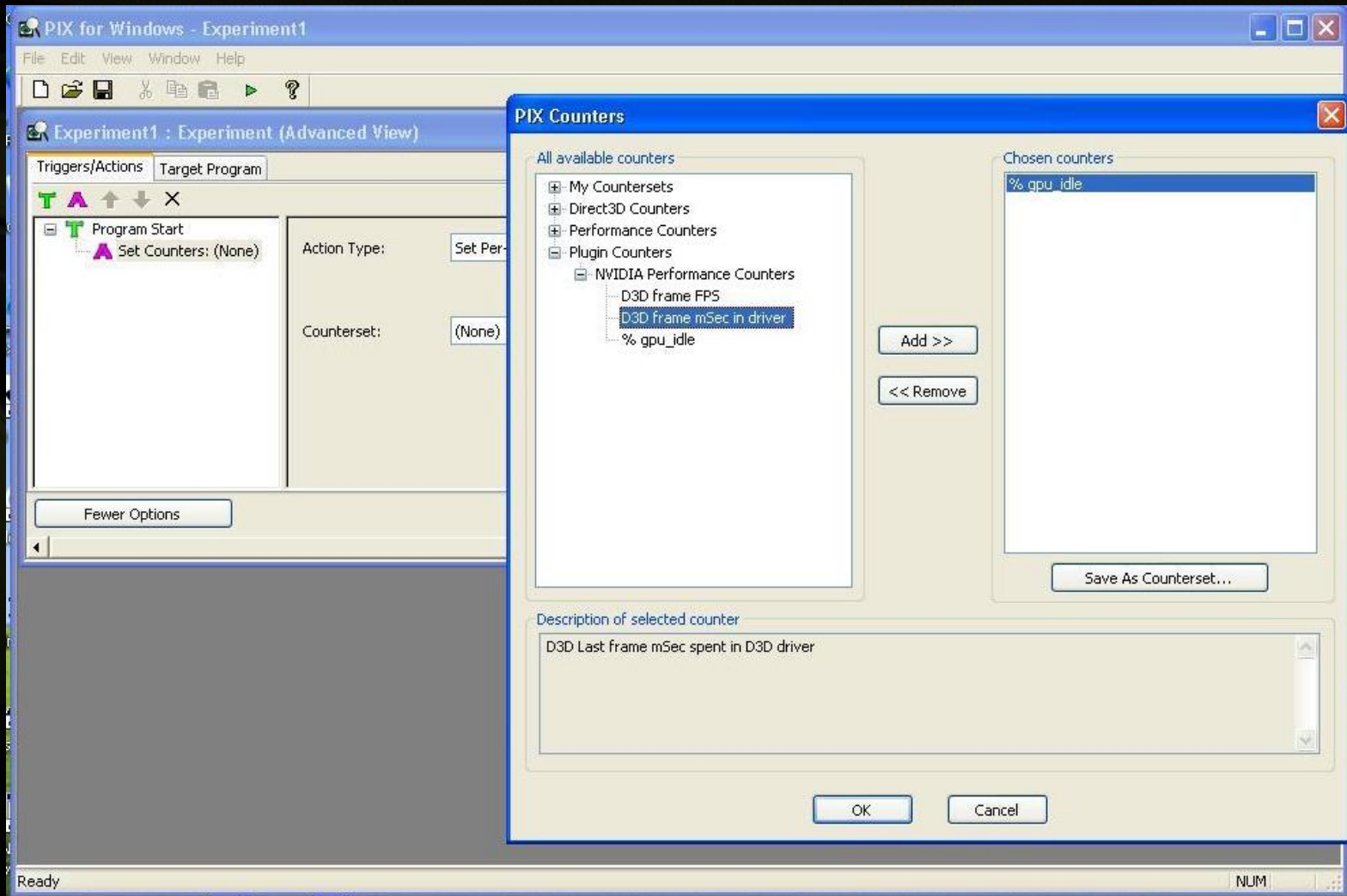
Bottleneck and Utilization Analysis: Fragment Program

Stage	Count
Vertex Fetch	[53, 15]
Vertex	[30, 17]
Fragment	[60, 50]
Frame Buffer	[37, 17]
ROP	[8, 1]

GPU Idle [3.96] Vertex Shader Busy [5.36] Fragment Program Busy [32.02] Fragment Program Waits for Texture [8.92]

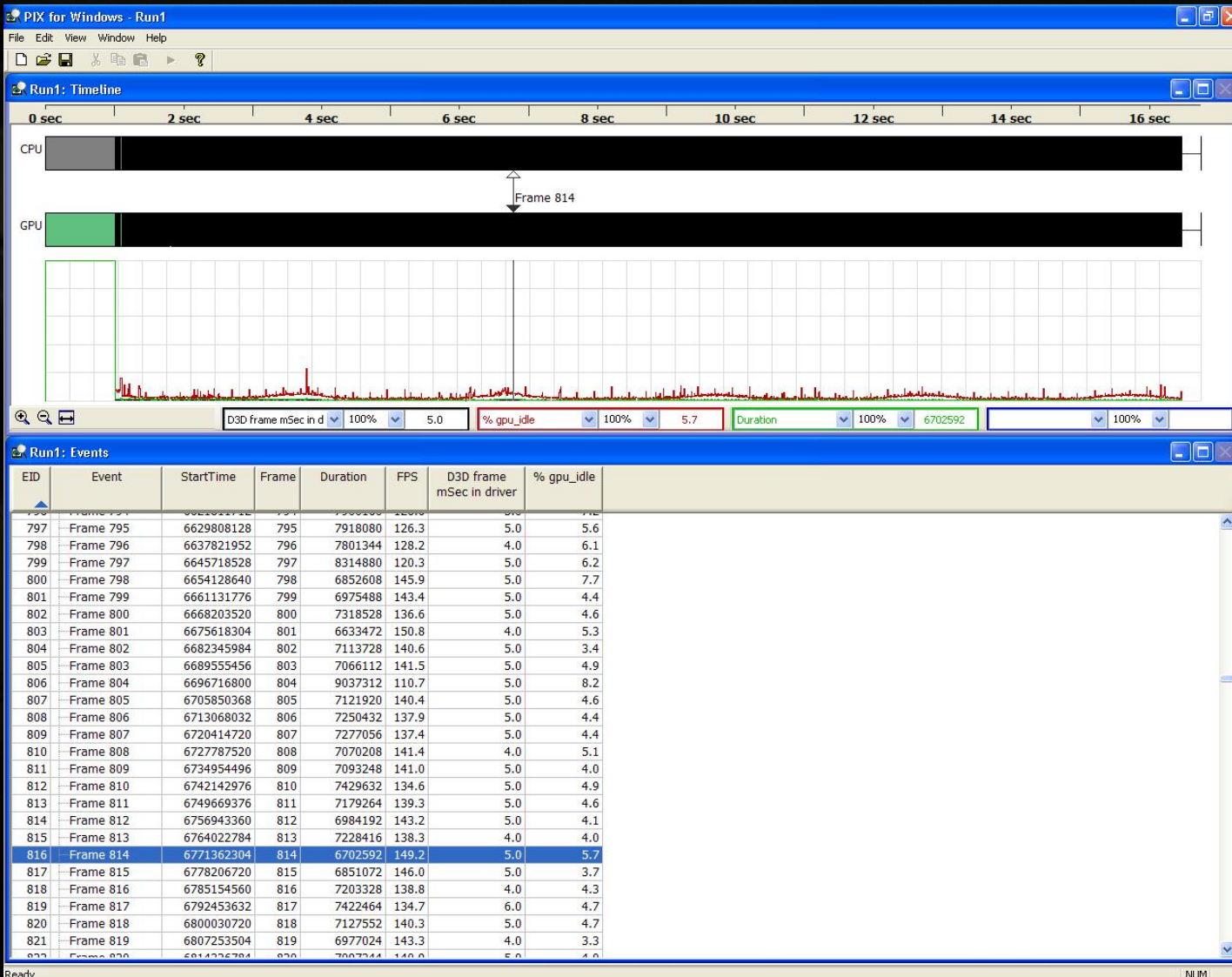
o Source/SXMLG  
I/GUIFont.o Sou  
e/SXMLGUI/GUI/G  
Font.o Source/S  
XMLUtils.o -lGL

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





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



# Project Status



- **NVPerfKit 2.0 for Windows 32bit available now at [developer.nvidia.com](http://developer.nvidia.com)**
- **NVPerfKit 2.1 (August 2006)**
  - **NVPerfHUD 4.1**
  - **ForceWare Release 90 Driver**
  - **Windows 64 bit**
  - **Linux 32 bit and 64 bit**
- **Instrumented GPUs**

**Quadro FX 5500 & 4500  
GeForce 7900 GTX & GT  
GeForce 7800 GTX**

**GeForce 6800 Ultra & GT  
GeForce 6600**

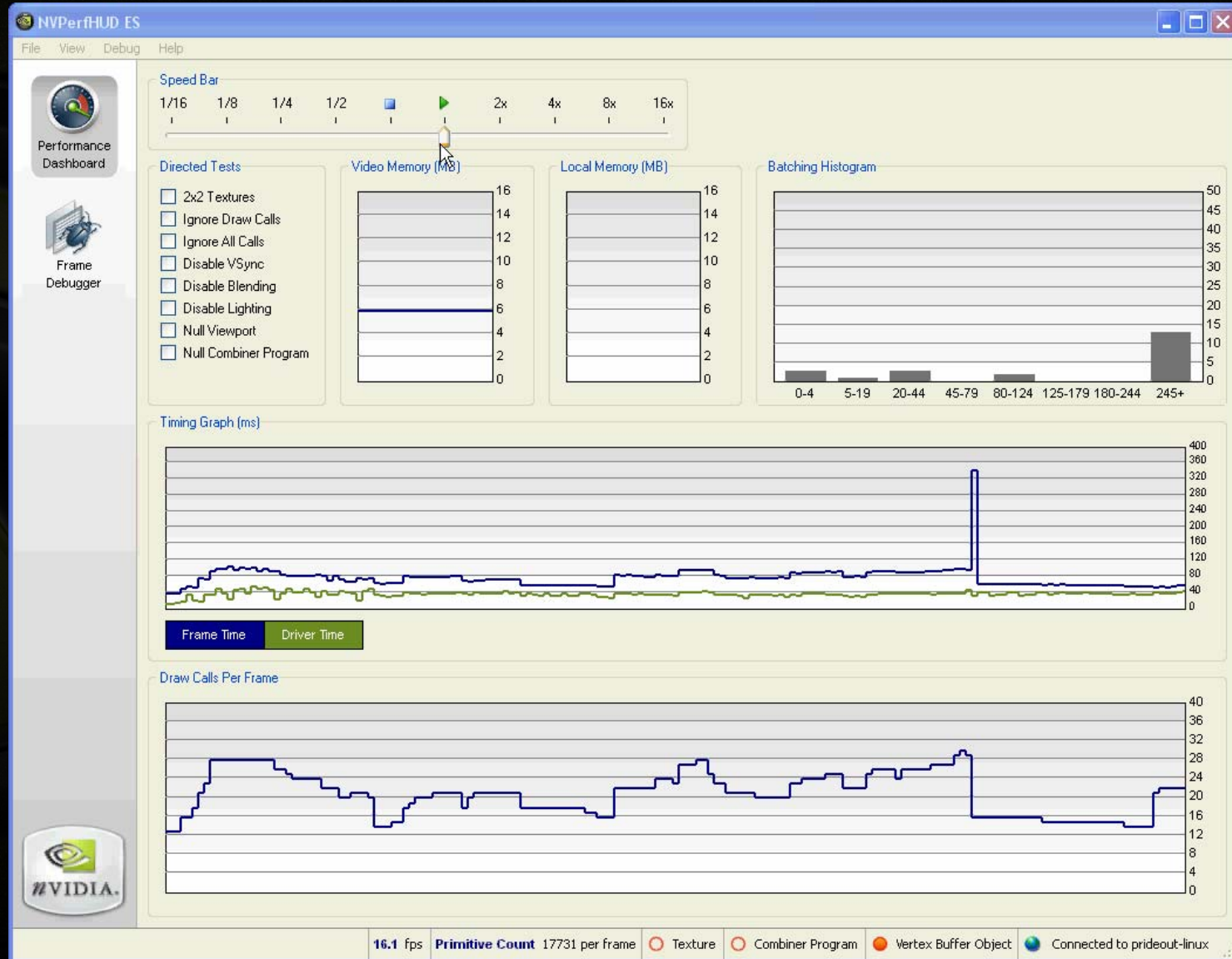
**Feedback and Support: [NVPerfKit@nvidia.com](mailto:NVPerfKit@nvidia.com)**

# NVPerfHUD ES



- **Developing similar performance tools for handheld developers using OpenGL ES**
- **Application runs on real hardware with Instrumented Driver**
- **IDE runs on host computer (Linux or PC)**
- **Same debugging and performance analysis tools that are available on NVPerfHUD 4.0!**

# Performance Dashboard



# Debugging Features



NVPerfHUD ES

File View Debug Help

Performance Dashboard

Frame Debugger

Frame Scrubber Call Trace Geometry Viewer State Viewer Texture Viewer

Color

Depth

Navigation Bar

Draw Call 1 / 16

glDrawElements(GL\_TRIANGLES, 3597, GL\_UNSIGNED\_SHORT, 0x00000000)

#VIDIA.

Connected to prideout-linux


# Texture Viewer

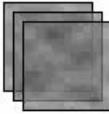



NVPerfHUD ES

File View Debug Help





Frame Scrubber Call Trace Geometry Viewer State Viewer **Texture Viewer**

  
Texture Object 4  
128 x 128

  
Texture Object 333  
32 x 32



The main area displays a cross-shaped preview of texture data. The vertical bar shows a waterfall scene, and the horizontal bar shows a rocky stream bed. A white crosshair is positioned on the right side of the horizontal bar.

  **1:1** Miplevel 0  Cross 

Navigation Bar

Draw Call 11 / 11

Connected to localhost

SIGGRAPH2006

# Graphic Remedy's gDEDebugger



The screenshot displays the gDEDebugger interface for the application 'Nature'. The main window shows a 3D rendered scene of a landscape with a lake, grass, and mountains under a cloudy sky. In the top-left corner, an 'FPS Monitor' panel shows: Triangle count: 181328, Visible Cells: 32%, Current FPS: 35. In the top-right corner, a settings panel shows: Alpha Reference: 0.25, Alpha Booster: 1.50, and Transparency AA (checked).

The bottom section of the interface contains several debugging panels:

- OpenGL Function Calls History:** Lists recent OpenGL calls such as `glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)` and `glUseProgramObjectARB(3)`.
- OpenGL State Variables:** Shows current state variables like `GL_VIEWPORT (0, 0, 400, 400)` and `GL_PROJECTION_MATRIX (2.00, 0.00, 0.00, 0.00)...`.
- Calls Stack:** Shows the current call stack, with `tpDrawScene - grteapotapplication.cpp, line 1206` selected.
- Properties:** Displays details for the selected function: `tpDrawScene`, File path: `c:\program files\graphic remedy\gdebugger\examples\teapot\src\grteapotapplication`, Line number: 1206, Module Name: `c:\program files\graphic remedy\gdebugger\examples\teapot\GRTeaPot.exe`.
- Performance Graph:** A line graph showing performance metrics over time.
- Counter Name Table:**

Counter Name	Value
Frames/sec: Context 1	64
CPU 0 Utilization	5
GPU0: % vertex_shader_busy	0
GPU0: % gpu_idle	92
GPU0: vertex_count	n
- Performance Dashboard:** A bar chart showing performance metrics: Fra... (64), CPU... (5), GPU... (0), GPU... (92), GPU... (100).
- Function Calls Statistics:**

OpenGL Function Name	%	# of Calls in Previ
glMaterialfv	9.30	4
glMatrixMode	9.30	4
glPopMatrix	6.98	3
glPushMatrix	6.98	3
glRotatef	6.98	3
glIsProgramObjectARB	6.98	3

# Free gDEDebugger Licenses!



- **OpenGL ARB and Graphic Remedy Academic License Program**
  - Annual program for academic OpenGL developers
  - One year license for full featured version, including all software updates
  - Limited number of free licenses available for non-commercial, non-academic developers
- Details: <http://academic.gremedy.com>



# NVShaderPerf



- **What is NVShaderPerf?**
- **What's coming with version 2.0?**

```

v2f BumpReflectPS(a2v IN,
    uniform float4x4 WorldViewProj,
    uniform float4x4 World,
    uniform float4x4 ViewIT)

```



# NVShaderPerf

```

// Position in scene 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.TexCoord1.x = dot(World[0].xyz, TangentToObjSpace[0]);
OUT.TexCoord1.y = dot(World[1].xyz, TangentToObjSpace[0]);
OUT.TexCoord2.x = dot(World[0].xyz, TangentToObjSpace[1]);
OUT.TexCoord2.y = dot(World[1].xyz, TangentToObjSpace[1]);
OUT.TexCoord2.z = dot(World[2].xyz, TangentToObjSpace[1]);
OUT.TexCoord3.x = dot(World[0].xyz, TangentToObjSpace[2]);
OUT.TexCoord3.y = dot(World[1].xyz, TangentToObjSpace[2]);
OUT.TexCoord3.z = dot(World[2].xyz, TangentToObjSpace[2]);
float4 worldPos = float4(IN.Position, 1.0);
// compute the eye vector (going from shaded point to eye) in cube space
float4 eyeVec = worldPos - ViewIT.view;
OUT.TexCoord1.w = eyeVec.x;
OUT.TexCoord2.w = eyeVec.y;
OUT.TexCoord3.w = eyeVec.z;
return OUT;
}

```

```

float4 BumpReflectPS(v2f IN,
    uniform sampler2D NormalMap,
    uniform samplerCUBE EnvironmentMap,
    uniform float BumpScale) : COLOR
{
    // fetch the bump normal from the normal map
    float3 normal = tex2D(NormalMap, IN.TexCoord1.xy);
    normal = normalize(float3(normal.x * BumpScale, normal.y * BumpScale, normal.z * BumpScale));
    // transform the bump normal into object space
    // then use the transformed normal to fetch the cube map
    // used to fetch the cube map
    // (we multiply by 2 only to increase precision)
    float3 eyevec = float3(IN.TexCoord1.w, IN.TexCoord2.w, IN.TexCoord3.w);
    float3 worldNorm;
    worldNorm.x = dot(IN.TexCoord1.xyz, normal);
    worldNorm.y = dot(IN.TexCoord2.xyz, normal);
    worldNorm.z = dot(IN.TexCoord3.xyz, normal);
    float3 cubeMapCoord = float3(worldNorm.x * 2, worldNorm.y * 2, worldNorm.z * 2);
    return texCUBE(EnvironmentMap, cubeMapCoord);
}

```

- Inputs:
- GSI, CG, HLSL
  - PS1.x, PS2.x, PS3.x
  - VS1.x, VS2.x, VS3.x
  - !!FP1.0
  - !!ARBfp1.0

## NVShaderPerf

- GPU Arch.
- GeForce 7X00
  - GeForce 6X00
  - Geforce 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
mul r1.xyz, r0.w, c4
add r0.w, r0.x, c9.w
add r0.xyz, r0, c1
mad r0.xyz, r0, c6
add r2.xyz, r2, c3
mad r1.xyz, r0.w, c3
add r0.xyz, r0, c3
mov r1.w, r0.w
mov r0.w, r0.w
mov r0.w, r0.w

// apply the eye vector
Target: GeForce 6800 Ultra (440) :: Unified Compiler: 061
Cycles: 14.00 :: R Regs Used: 2 :: R Regs Max Index <0 has
Pixel throughput (assuming 1 cycle texture lookup) 457.14
=====
Shader performance using all FP16
Cycles: 21.00 :: R Regs Used: 3 :: R Regs Max Index <0 has
Pixel throughput (assuming 1 cycle texture lookup) 304.76
=====
C:\Temp\NVShaderPerf_61_77>

```

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

# NVShaderPerf: In your pipeline



- **Test current performance**
  - Compare with shader cycle budgets
  - Test optimization opportunities
- **Automated regression analysis**
- **Integrated in FX Composer 1.8**

# FX Composer 1.8 – 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)

The screenshot displays two instances of the Shader Perf tool. The top window is for a GeForceFX 5200 GPU, showing a target of GeForceFX 5200 Ultra (NV34) with a Unified Compiler v61.77. It reports 51 cycles, 4 registers, and a pixel throughput of 15.69 MP/s. The bottom window is for a GeForce 6800 Ultra GPU, showing a target of GeForce 6800 Ultra (NV40) with a Unified Compiler v61.77. It reports 21.00 cycles, 3 registers, and a pixel throughput of 304.76 MP/s. The bottom window also shows a disassembly view with instructions like 'ps\_2\_0', 'def c9, 0, -', and 'def c10, 0, 0, 0'.

```
*****
Target: GeForceFX 5200 Ultra (NV34) :: Unified Compiler: v61.77
Cycles: 51 :: # R Registers: 4
Pixel throughput (assuming 1 cycle texture lookup) 15.69 MP/s
=====
Shader performance using all FP16
Cycles: 51 :: # R Registers: 2
Pixel throughput (assuming 1 cycle texture lookup) 15.69 MP/s
=====

Shader per
Cycles: 51
Pixel throu
*****
PS Instruct
ps_2_0
def c9, 0, -
def c10, 0,
.....
*****
Target: GeForce 6800 Ultra (NV40) :: Unified Compiler: v61.77
Cycles: 21.00 :: R Regs Used: 3 :: R Regs Max Index (0 based): 2
Pixel throughput (assuming 1 cycle texture lookup) 304.76 MP/s
=====
Shader performance using all FP16
Cycles: 14.00 :: R Regs Used: 2 :: R Regs Max Index (0 based): 1
Pixel throughput (assuming 1 cycle texture lookup) 457.14 MP/s
=====
Shader performance using all FP32
Cycles: 21.00 :: R Regs Used: 3 :: R Regs Max Index (0 based): 2
Pixel throughput (assuming 1 cycle texture lookup) 304.76 MP/s
*****
PS Instructions: 38
ps_2_0
def c9, 0, -2, 3, 1
def c10, 0.5, 0, 0, 0
.....
*****
```

# NVShaderPerf 1.8



- **Support for GeForce 7800 GTX and Quadro FX 4500**
- **Unified Compiler from ForceWare Release 80 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_eyeNermel, v2f.UVtex0 ) * 2.0 + f3tex2D( 1 - 1.0, v2f.UVtex0 ) * 2.0 );  
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( centerToSurfaceVec, objFlatNormal )  
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 - objEyeCoord );  
    float3 ray_origin = v2f.objCoord;
```

```
    // refract the ray before intersecting with the iris  
    ray_dir = refract( ray_dir, objFlatNormal, g_refract );
```

```
    // first, see if the refracted ray would leave the eye  
    float t_eyeballSurface = SphereIntersect( 16.0, ray_origin, ray_dir );  
    float3 objPosOnEyeBall = ray_origin + t_eyeballSurface * ray_dir;  
    float3 centerToSurface2 = normalize( objPosOnEyeBall - objFlatCoord );
```

```
    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 -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



- Improved vertex performance simulation and calculation of vertex throughput
- GLSL vertex program
- Multiple driver versions from one NVShaderPerf
- Much smaller footprint
- New programmatic interface
- Integration into FX Composer 2.0

Support and Feedback: [NVShaderPerf@nvidia.com](mailto:NVShaderPerf@nvidia.com)

# Questions?



- **Developer tools DVDs available at our booth**
  - NVPerfKit 2.0
  - NVPerfHUD 4.0 Overview Video
  - NVPerfHUD 4.0 Quick Reference Card
  - User Guides
- **Online:**
  - <http://developer.nvidia.com/NVPerfKit>
  - <http://developer.nvidia.com/NVPerfHUD>
  - <http://developer.nvidia.com/NVShaderPerf>
- **Feedback and Support:**
  - [NVPerfKit@nvidia.com](mailto:NVPerfKit@nvidia.com)
  - [NVPerfHUD@nvidia.com](mailto:NVPerfHUD@nvidia.com)
  - [NVShaderPerf@nvidia.com](mailto:NVShaderPerf@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 ...



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

**nVIDIA**

[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.