Performance Tools

Jeff Kiel, NVIDIA Corporation

SIGGRAPH2006

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
- gDEBugger: 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!

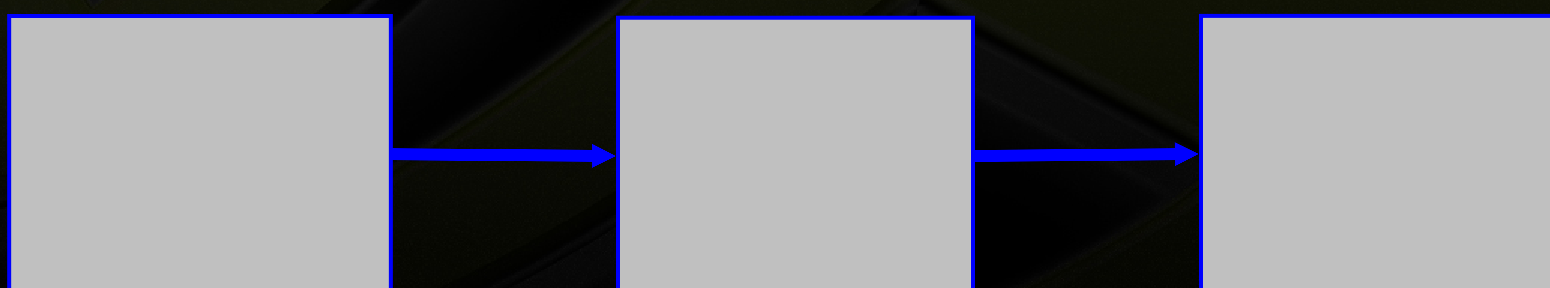
35% FPS improvement!*

11 Rendering bugs fixed!*

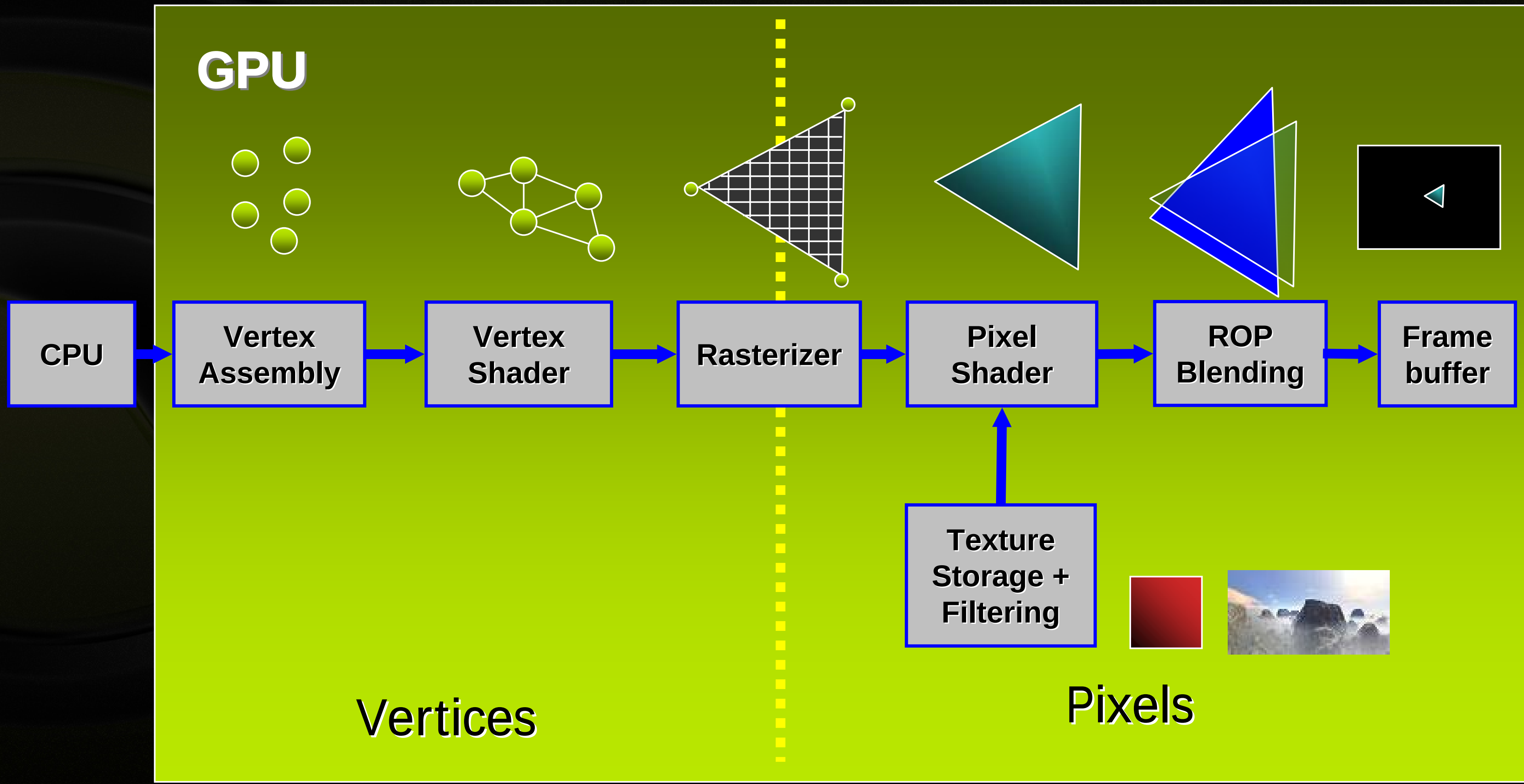
*Average of 35% FPS improvement and 11 bugs fixed reported by over 100 users of NVPerfHUD in recent Developer Survey

GPU pipelined architecture

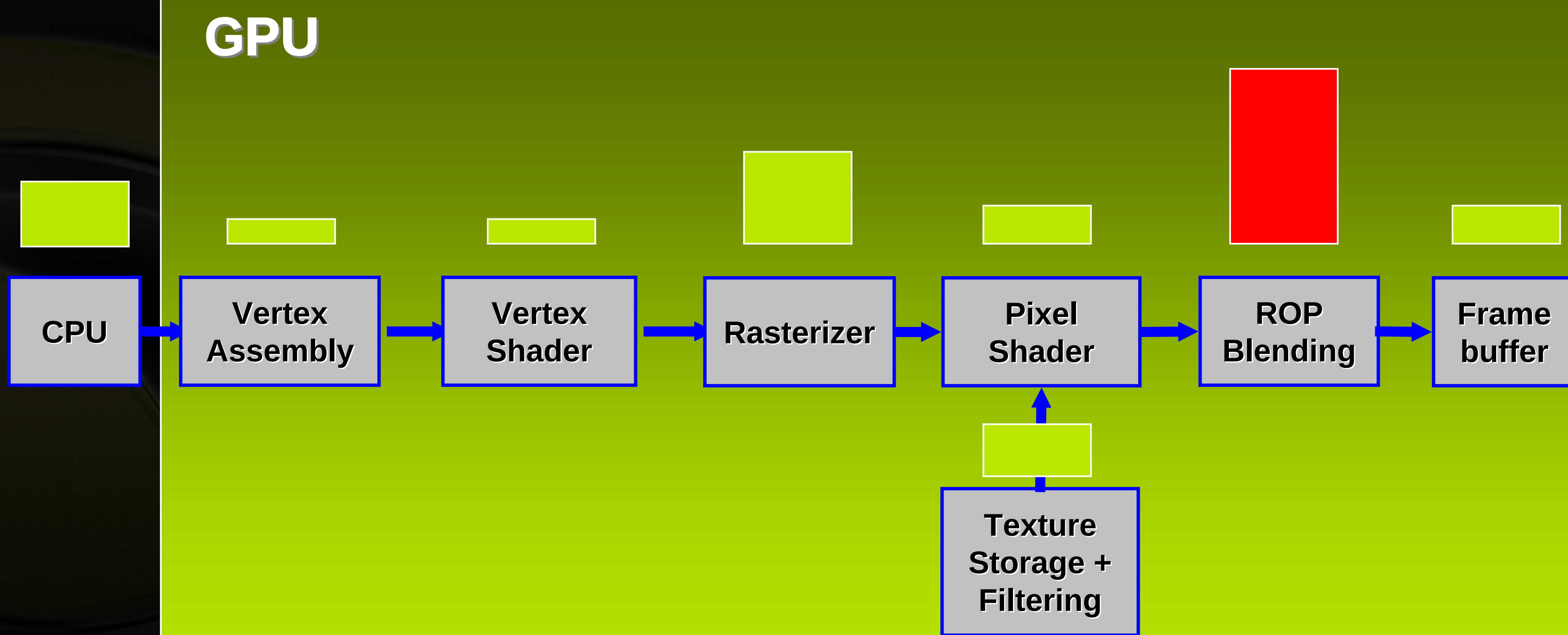
- Pipelined architecture: each unit needs the data from the previous unit to do its job
- Method: Bottleneck identification and elimination
- Goal: Balance the pipeline



GPU Pipelined Architecture (simplified view)

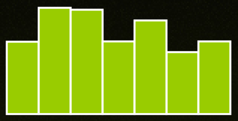


GPU Pipelined Architecture (simplified view)

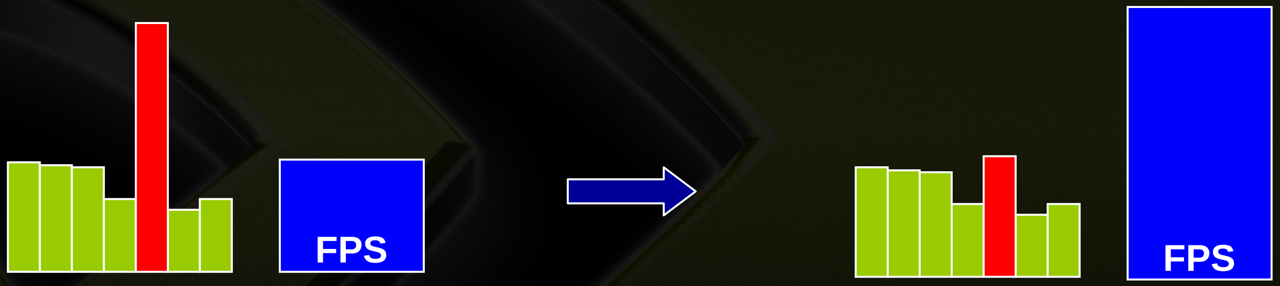


One unit can limit the speed of the pipeline…

Classic Bottleneck Identification

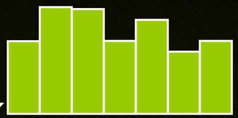


Modify target stage to decrease workload

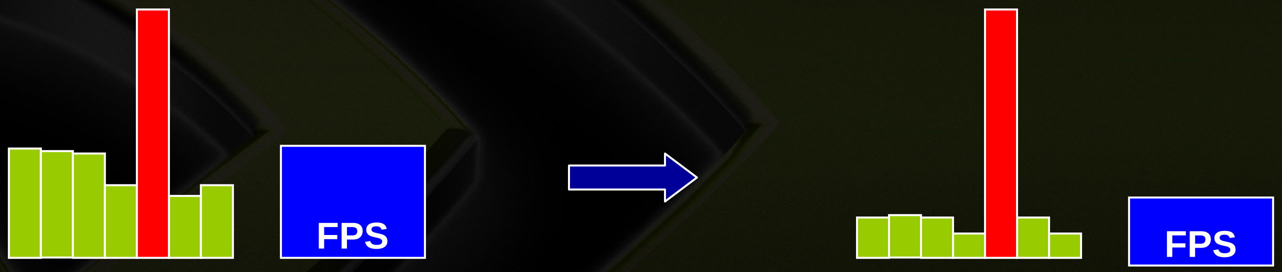


If performance/FPS improves greatly, this stage is the bottleneck
Careful not to change the workload of other stages!

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

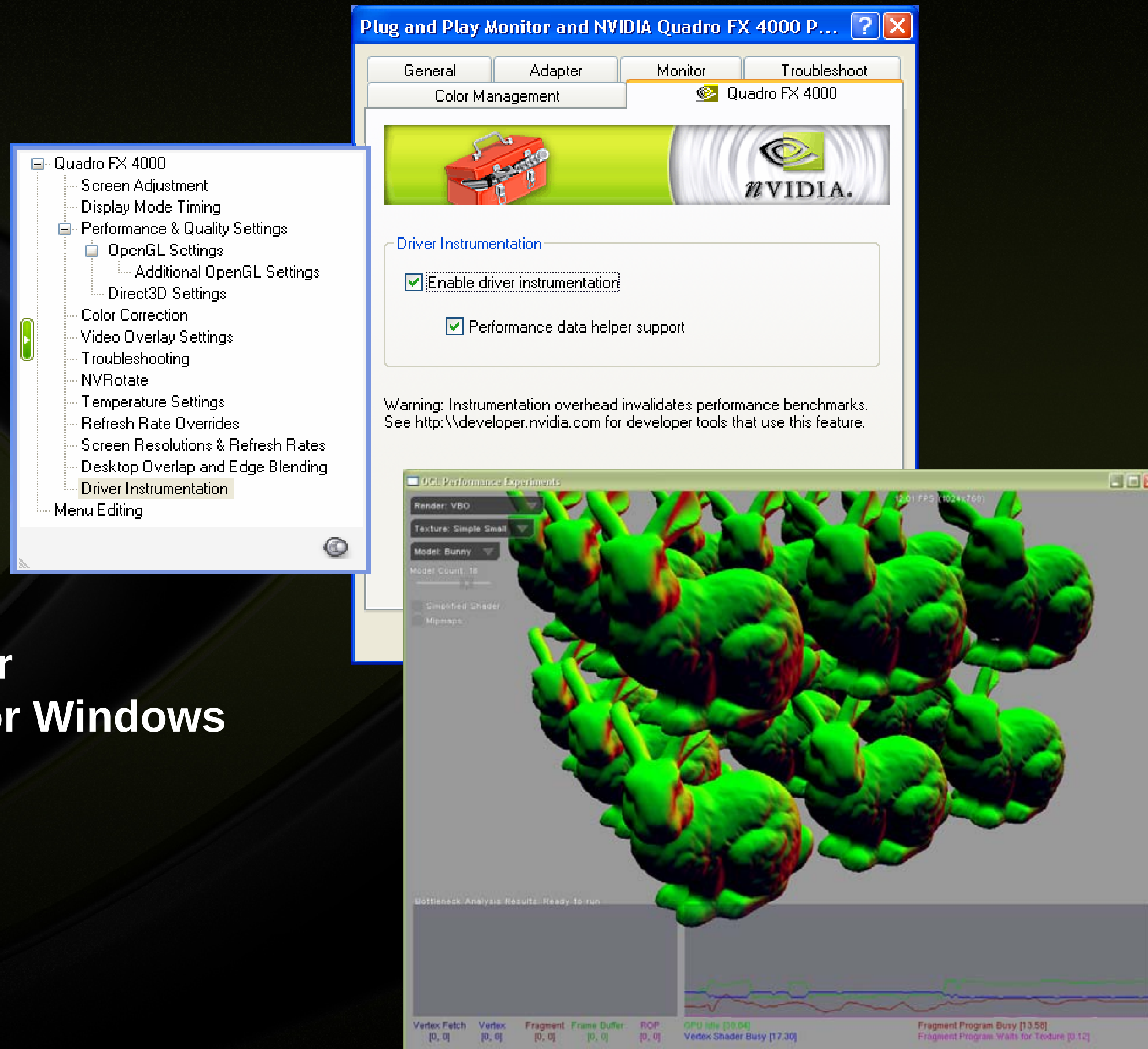
- **Sample performance data in each subunit of the GPU pipeline while rendering**
  - **Compare amount of work done to maximum work possible**
  - **Query the subunit for unit bottleneck information**

- **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
  - gDEBugger 2.4
  - NVDevCPL



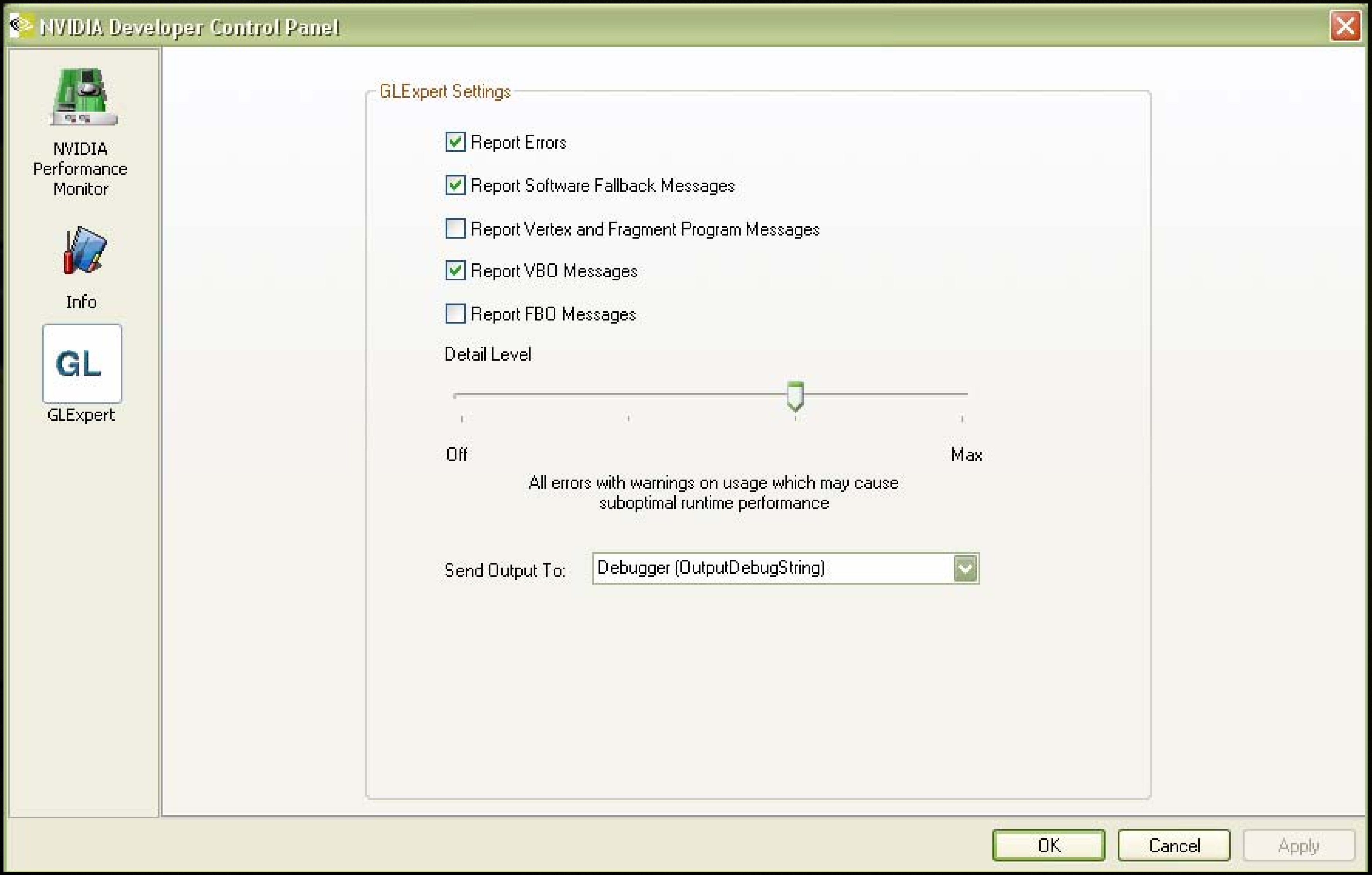
NVPerfKit Instrumented Driver

- 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



NVPerfKit: Counter Types

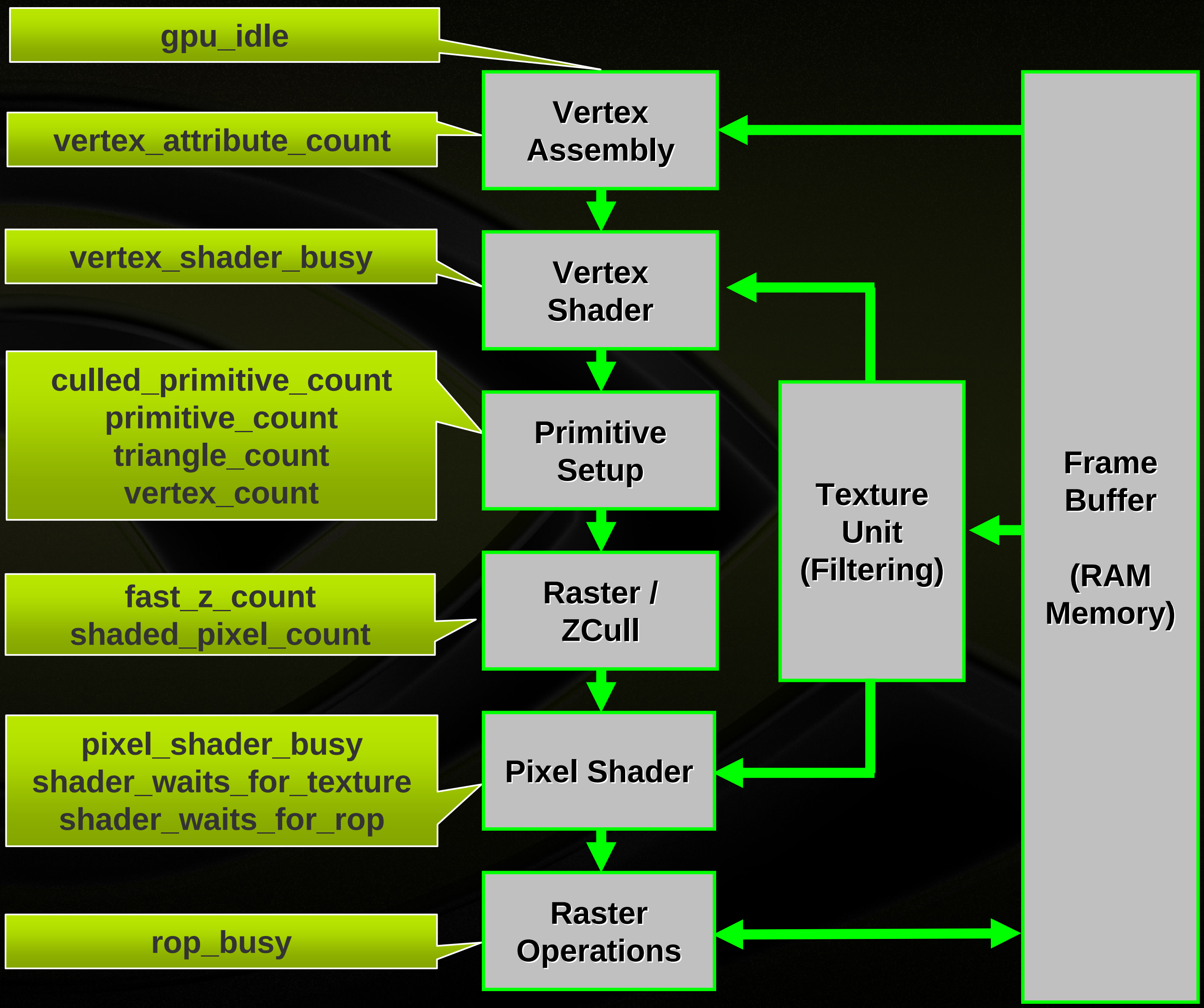
- SW/Driver Counters (Instrumented Driver)
  - Insight into OpenGL and Direct3D driver performance
  - Exposed via NVPerfAPI, PIX, and PDH
- Raw GPU Counters (Instrumented GPU)
  - Real time performance monitoring
  - Exposed via NVPerfAPI, PIX, and PDH
- Simplified Experiments (Instrumented GPU)
  -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

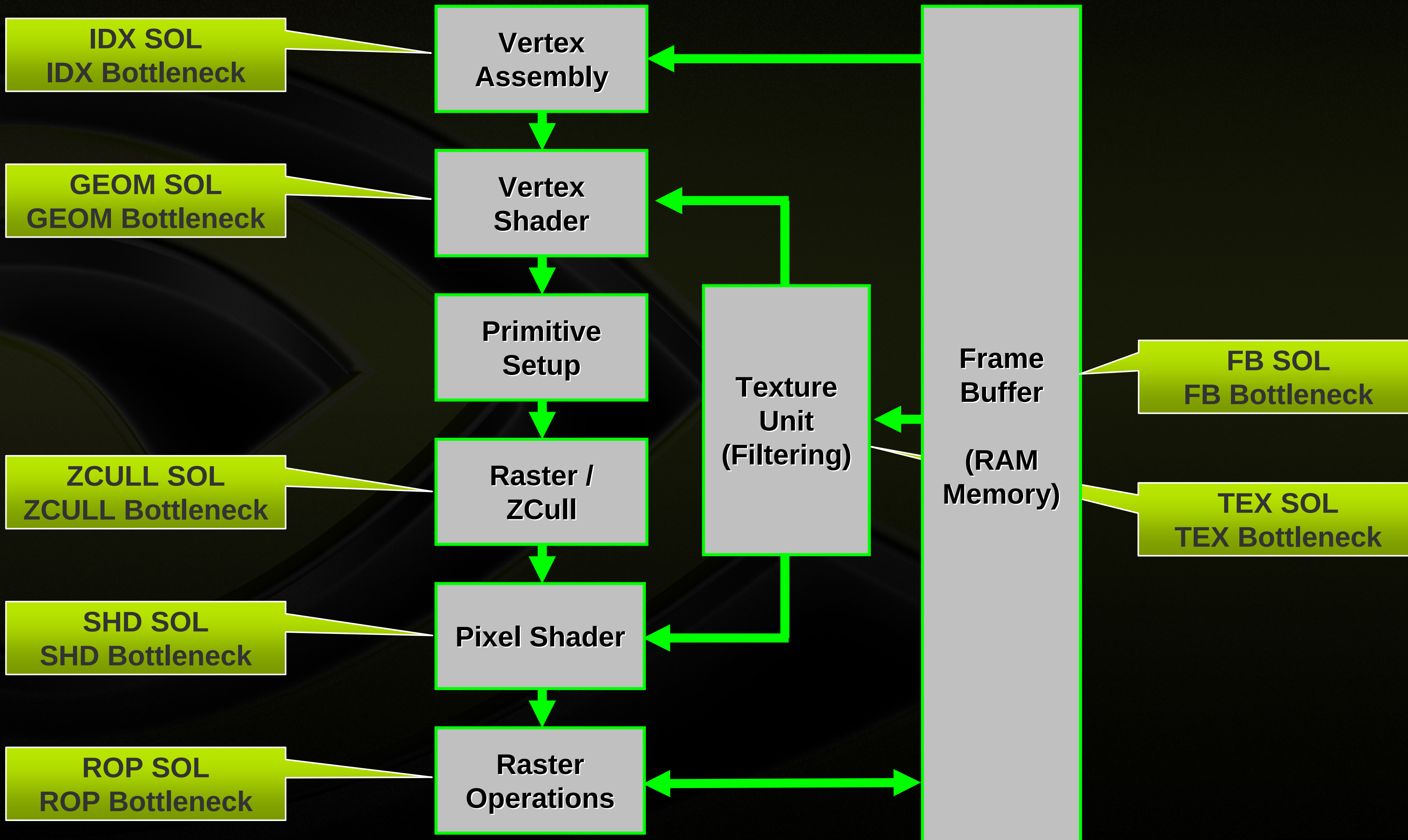
OpenGL/Direct3D Counters

- 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



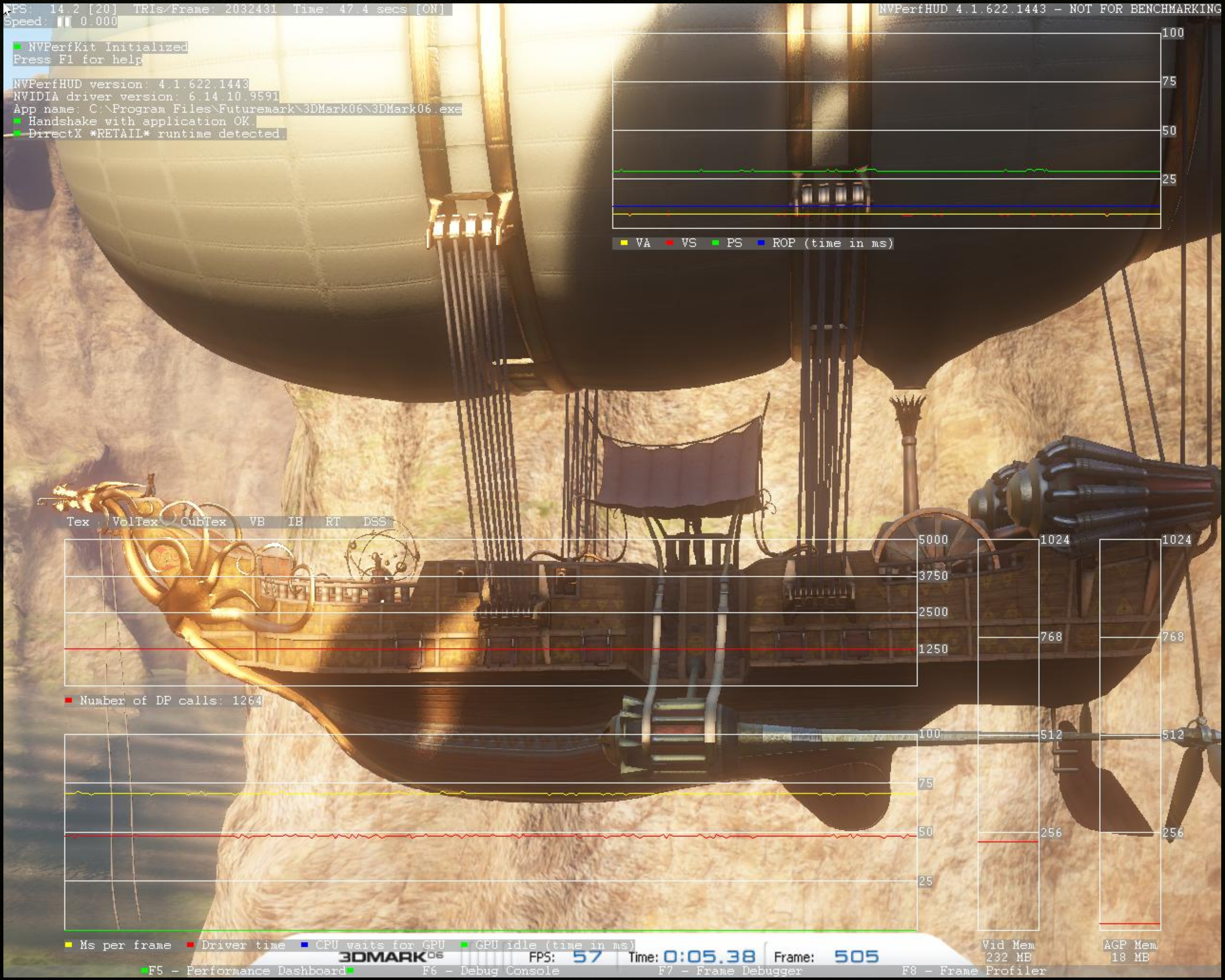
What is NVPerfHUD?

- 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

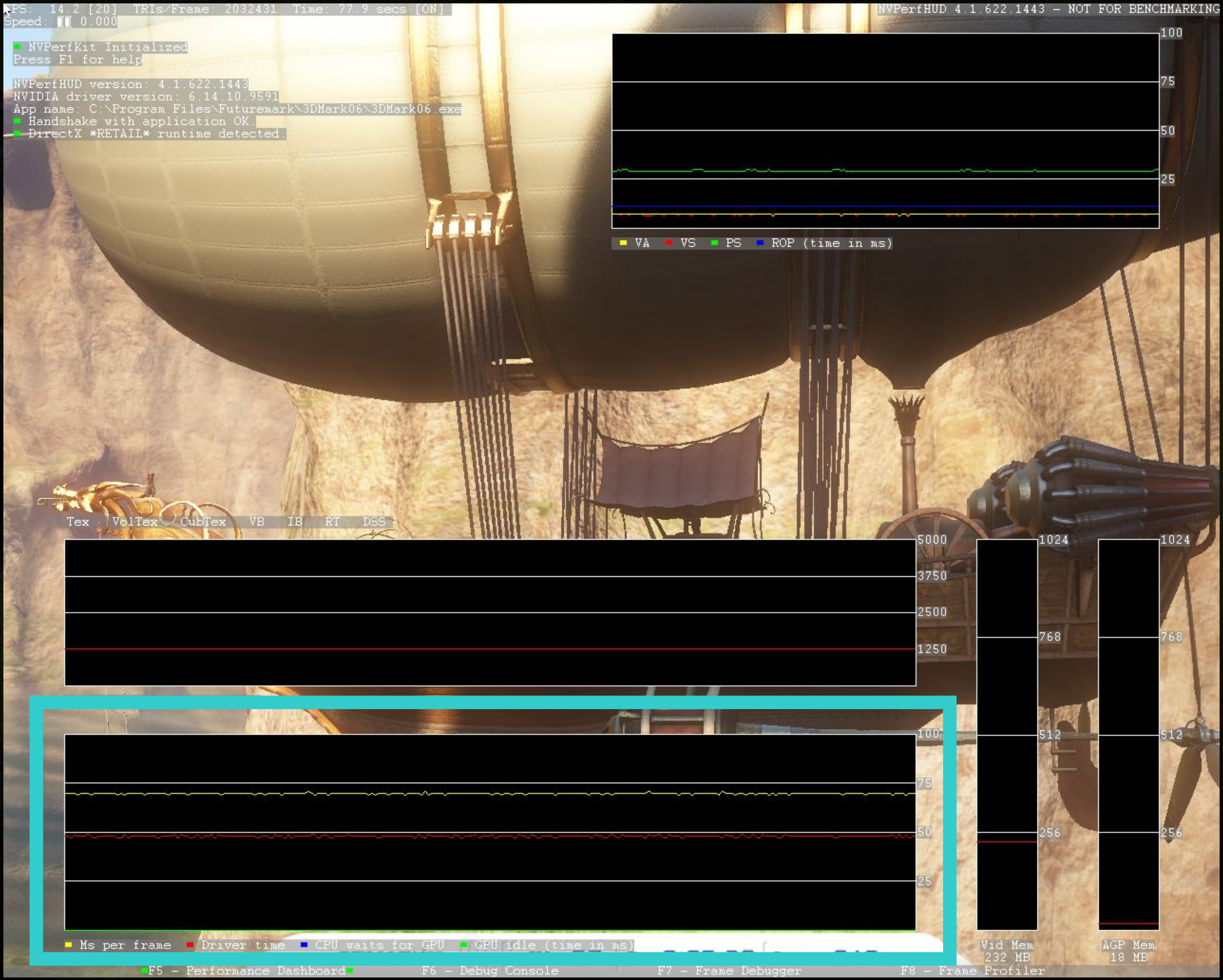
- Drag and drop your application onto the NVPerfHUD icon
- Run through your application as you normally do until you find:
  - Functional problems: use the Frame Debugger
  - Performance problems: use the Dashboard graphs and Frame Profiler

Demo: NVPerfHUD

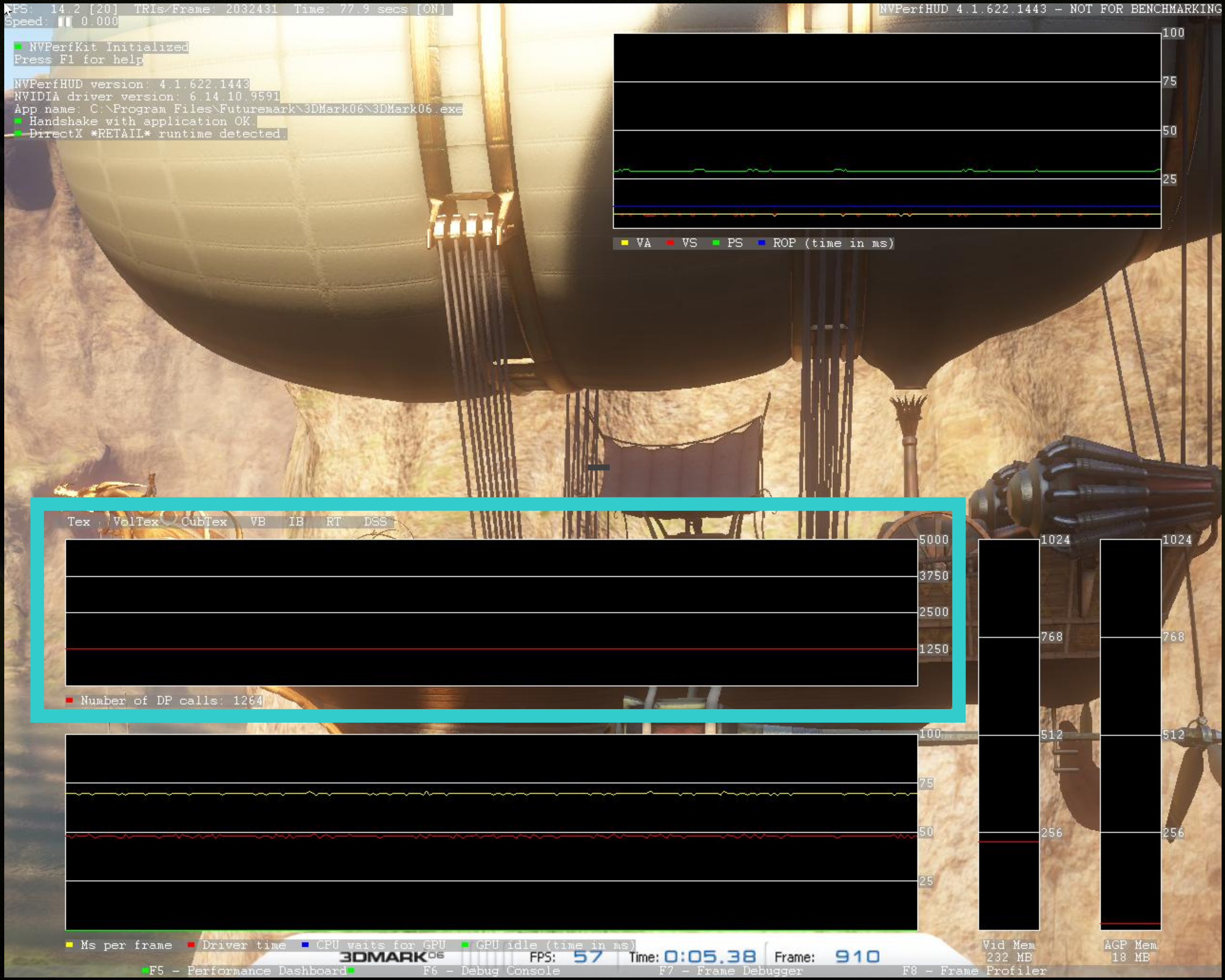


3DMark06 used with permission from Futuremark corporation

Demo: Performance Dashboard

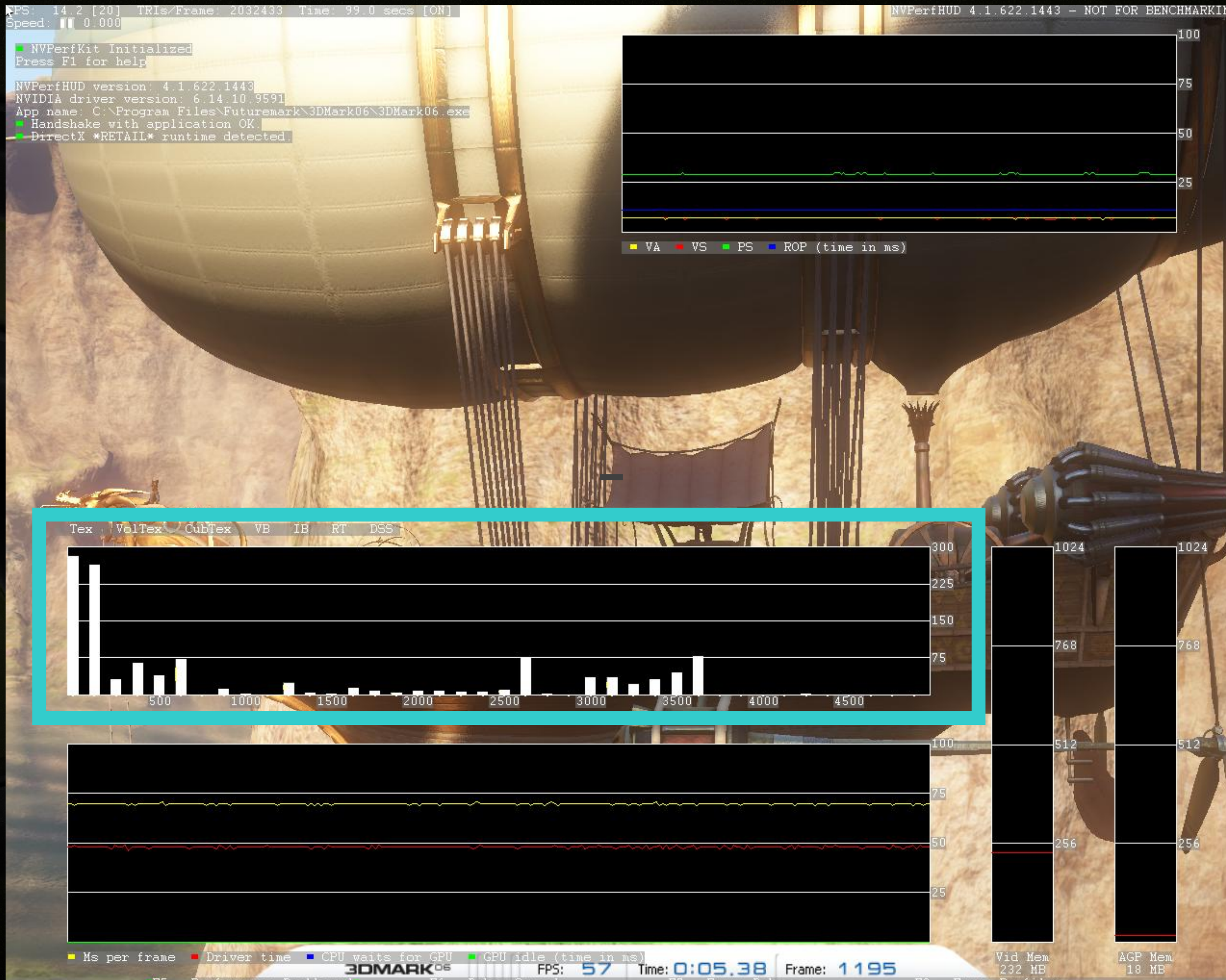


Demo: Performance Dashboard



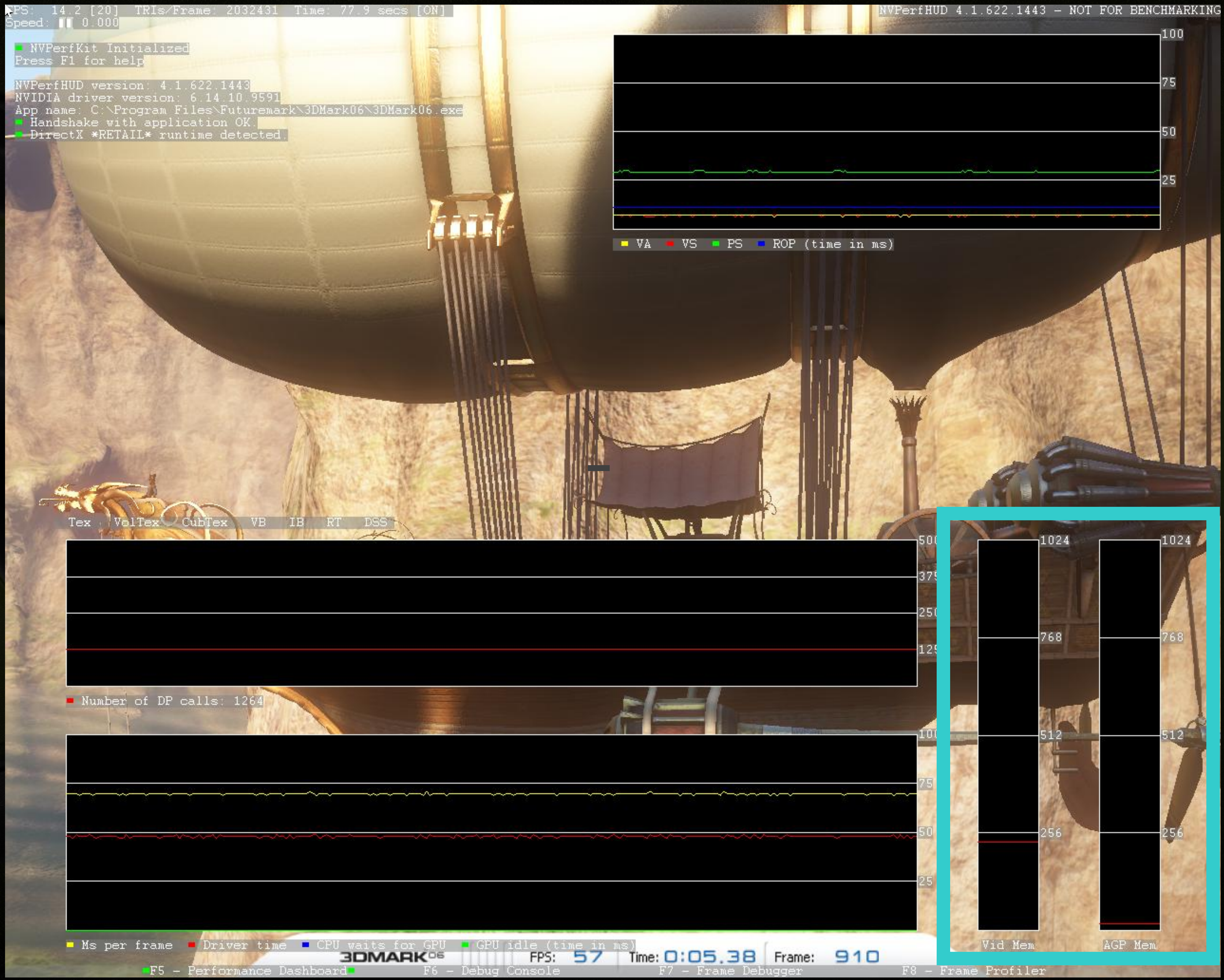
3DMark06 used with permission from Futuremark corporation

Demo: Performance Dashboard

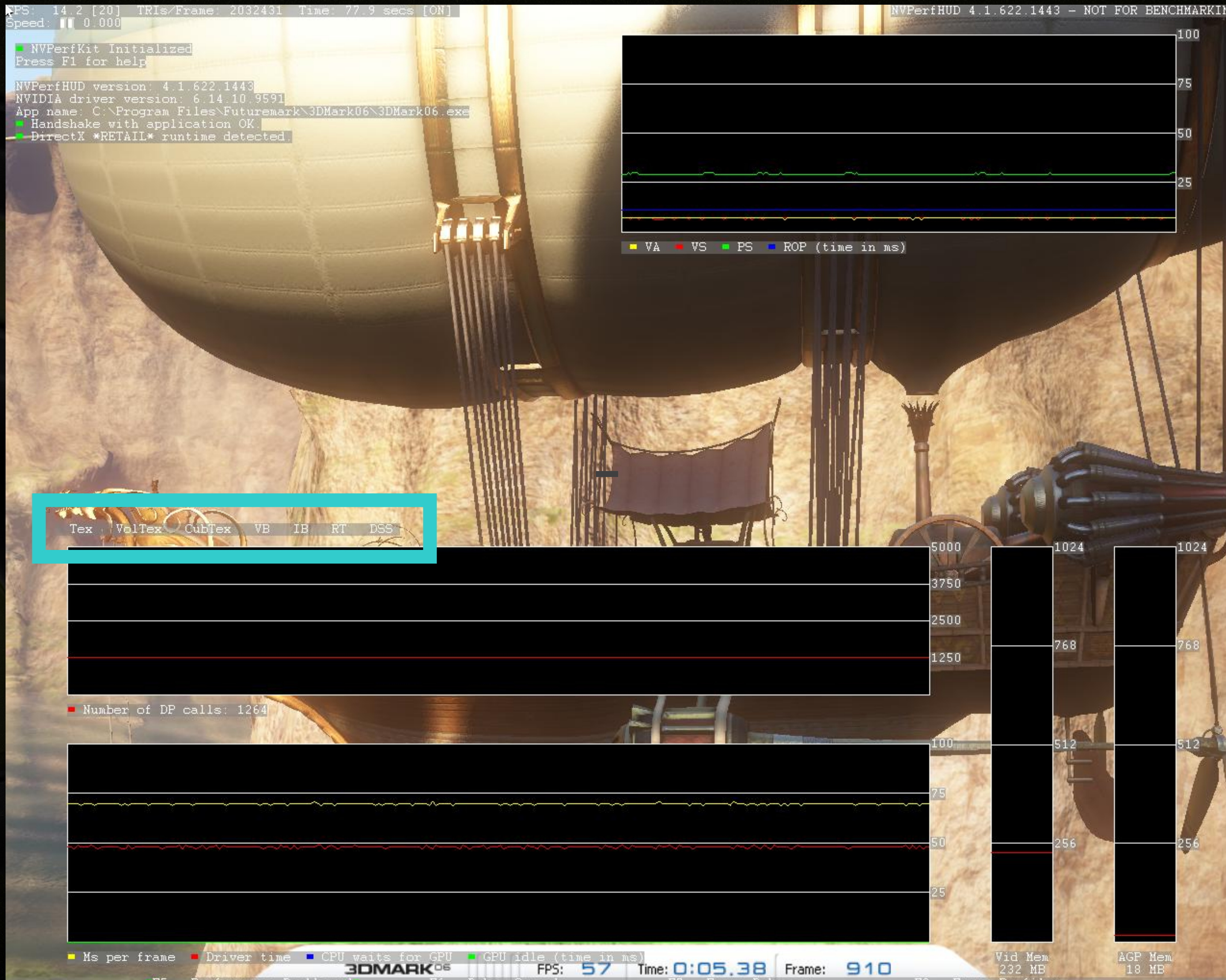


3DMark06 used with permission from Futuremark corporation

Demo: Performance Dashboard



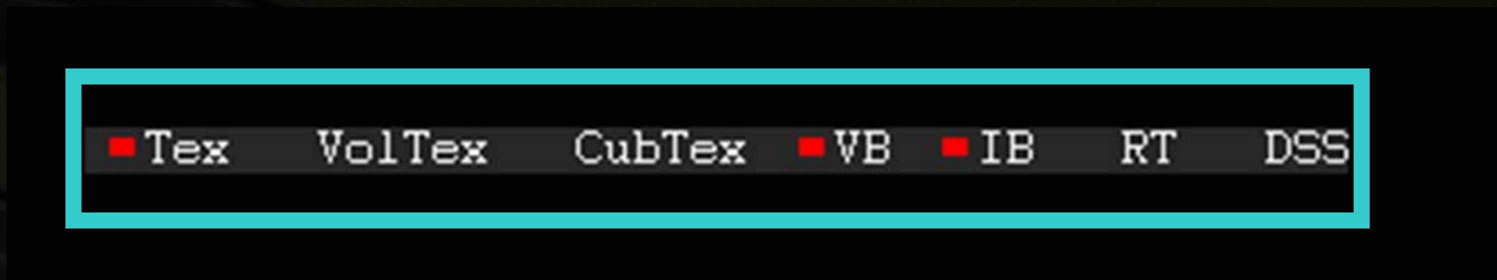
Demo: Performance Dashboard



3DMark06 used with permission from Futuremark corporation

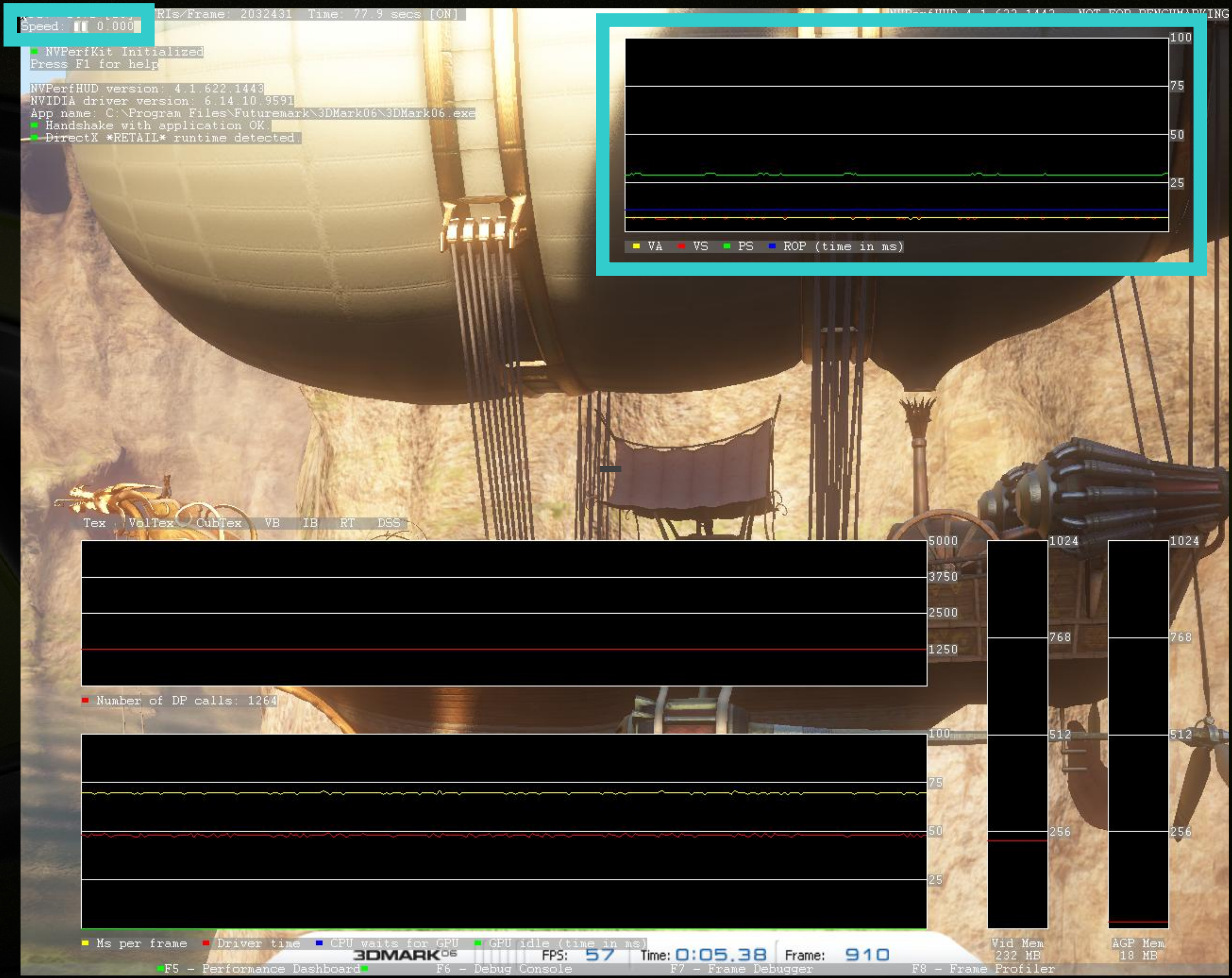
Demo: Performance Dashboard

Resource creation monitor



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

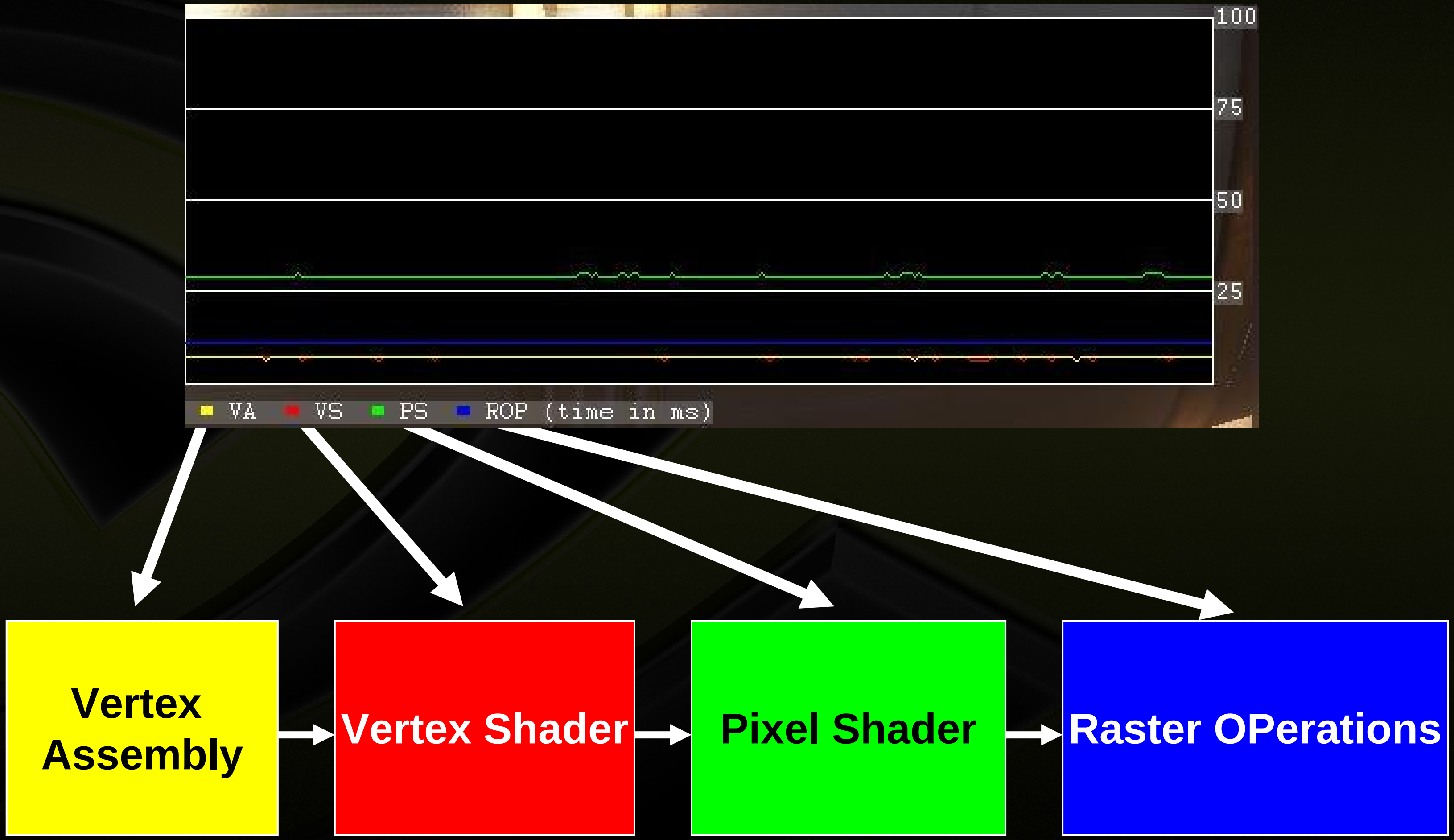
Demo: Performance Dashboard



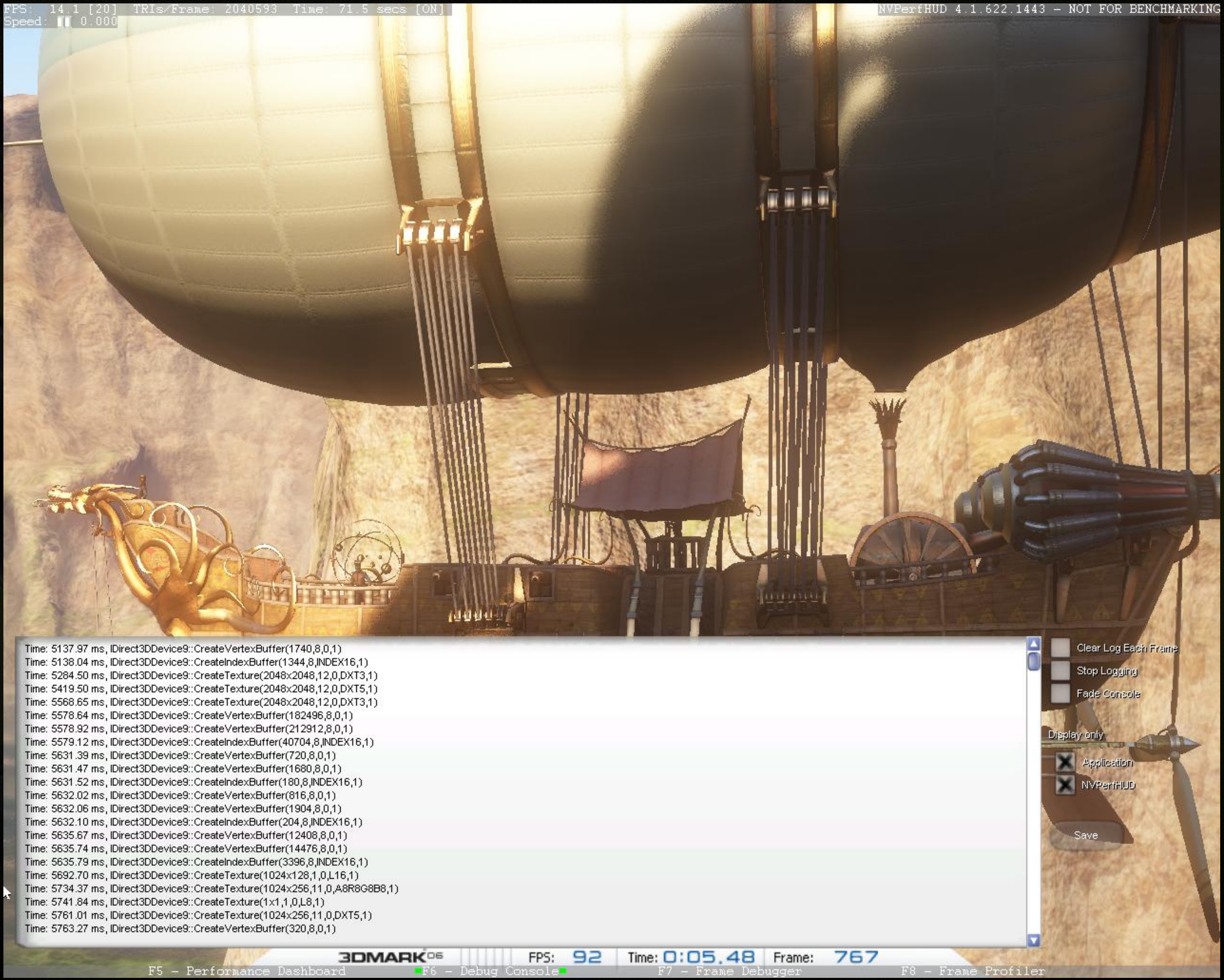
Demo: Performance Dashboard

Speed control

Demo: The simplified graphics pipeline

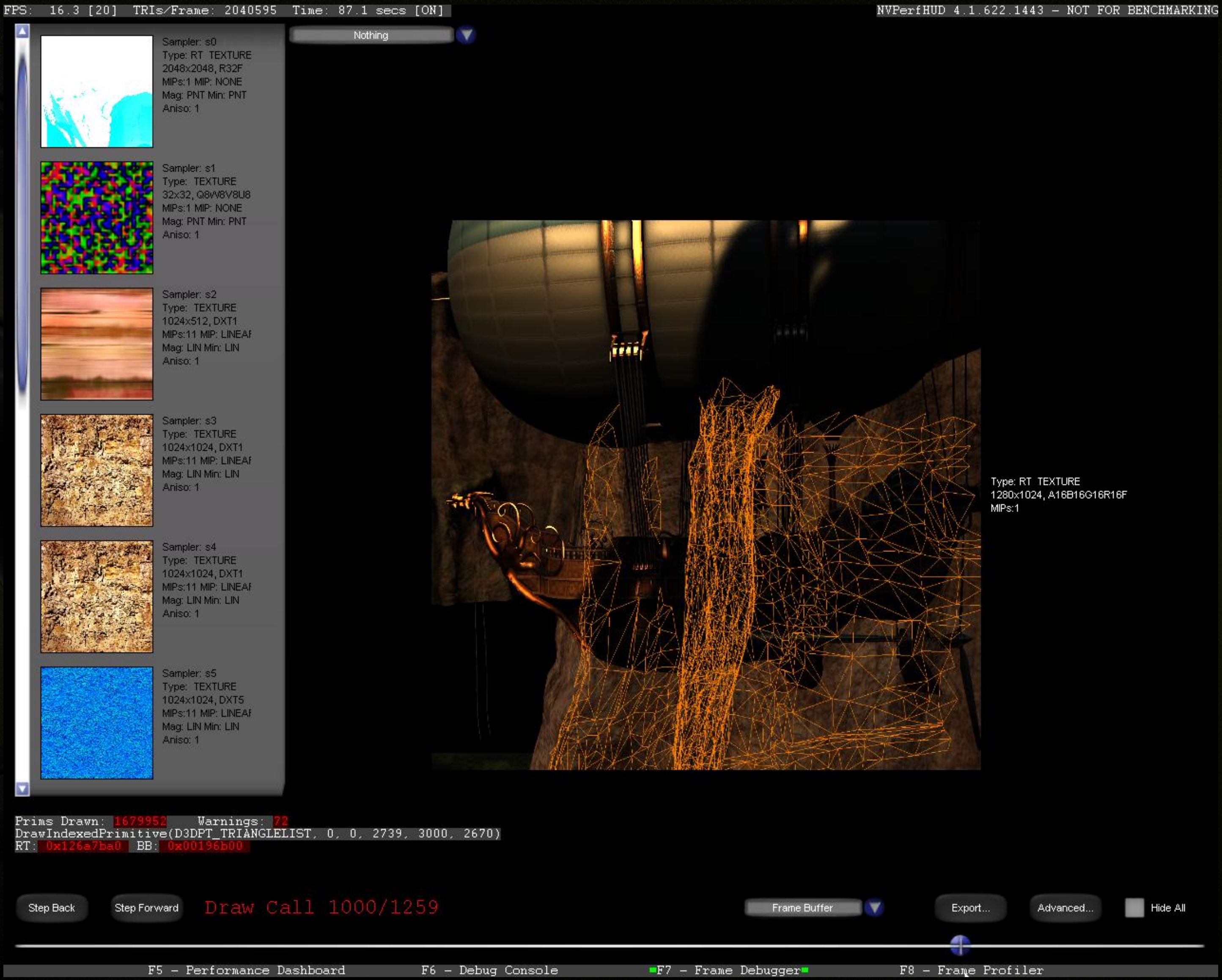


Demo: Debug Console



3DMark06 used with permission from Futuremark corporation

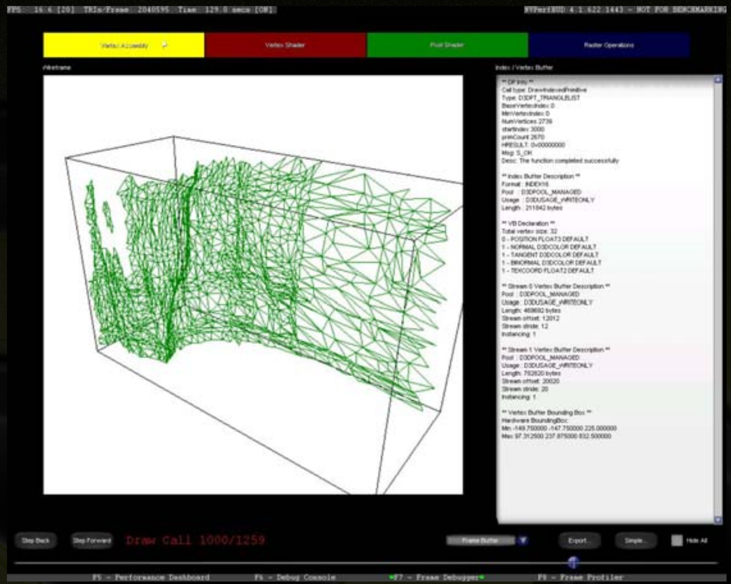
Demo: Frame Debugger

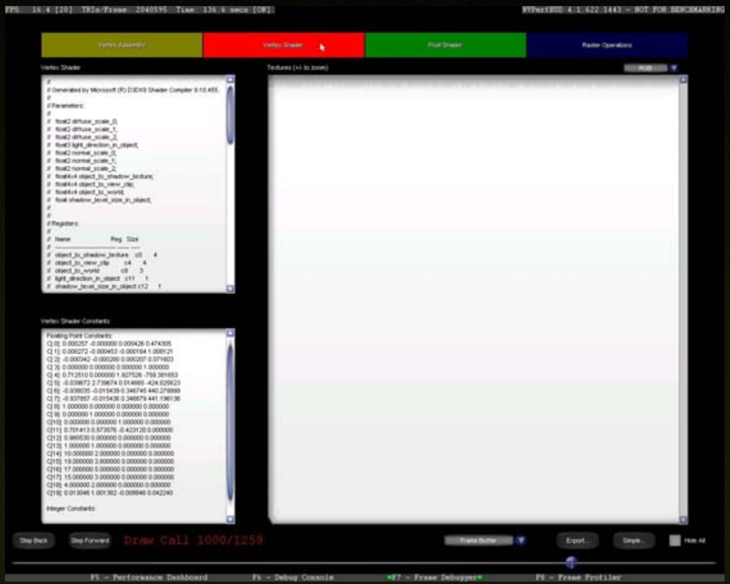


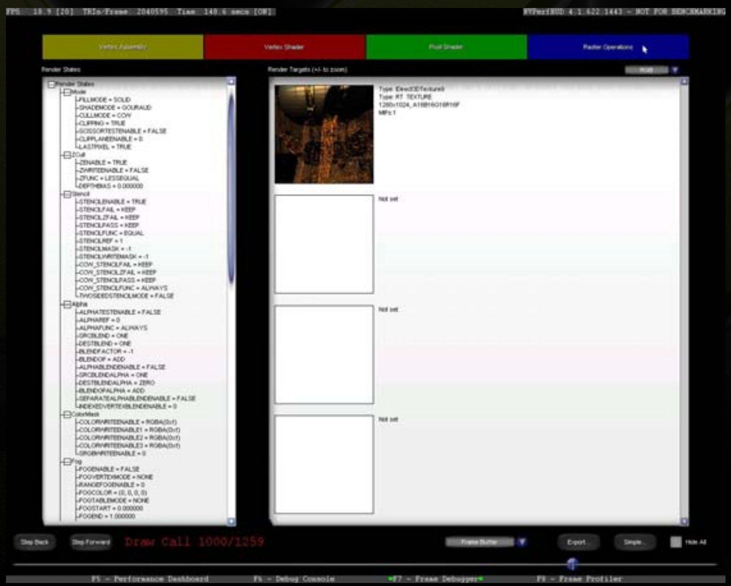
3DMark06 used with permission from Futuremark corporation

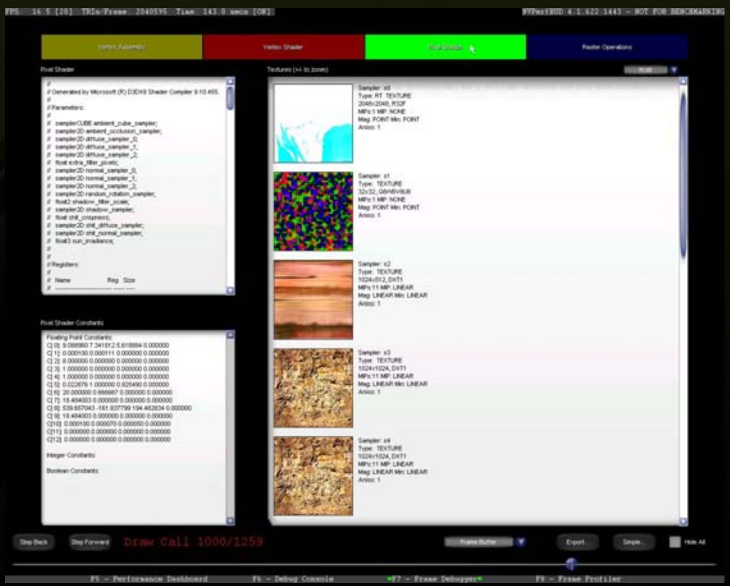
SIGGRAPH2006

Demo: Advanced Frame Debug









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

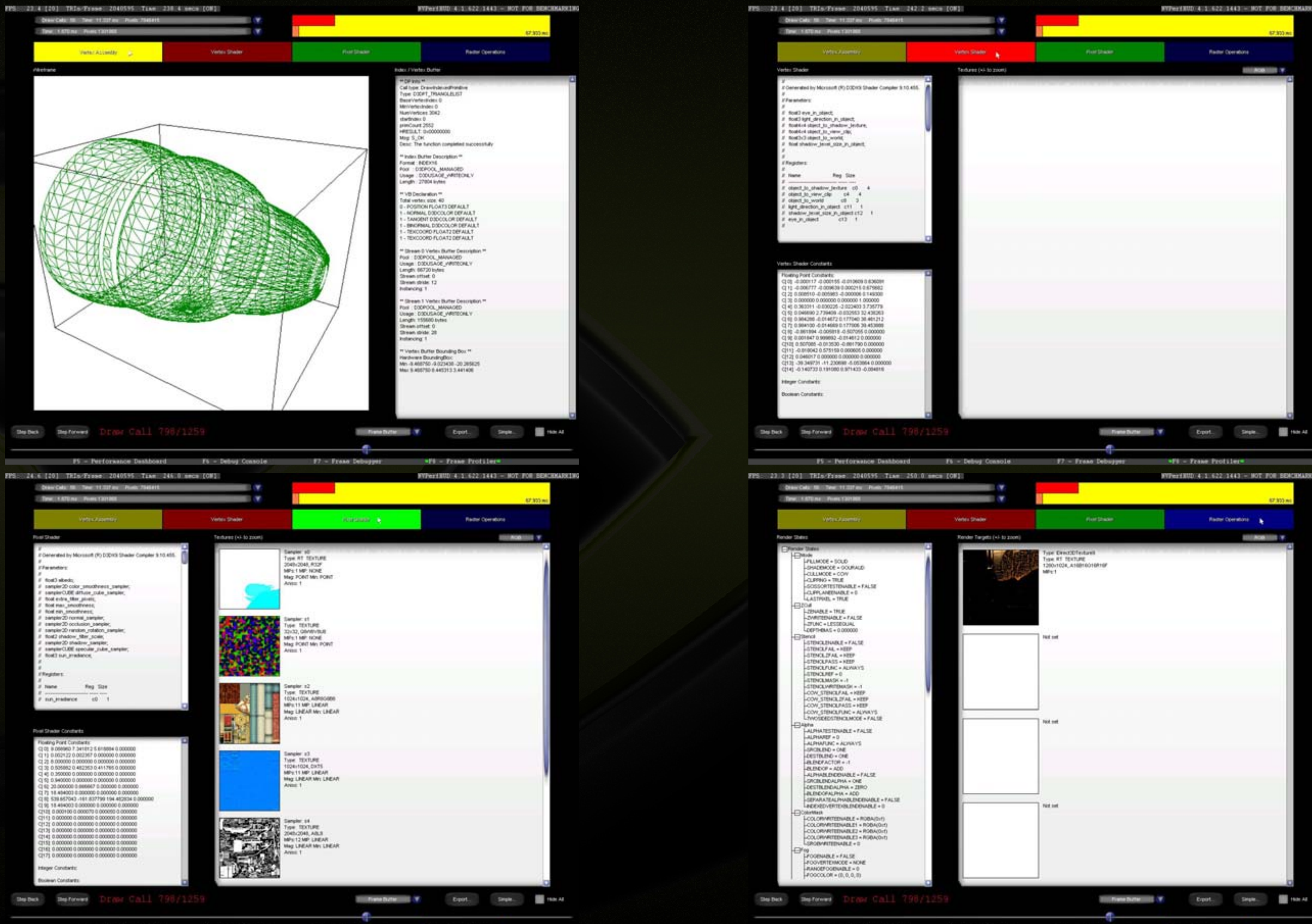
Frame Profiler: Optimization Strategy

- 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

3DMark06 used with permission from Futuremark corporation

Demo: Advanced Frame Profiler



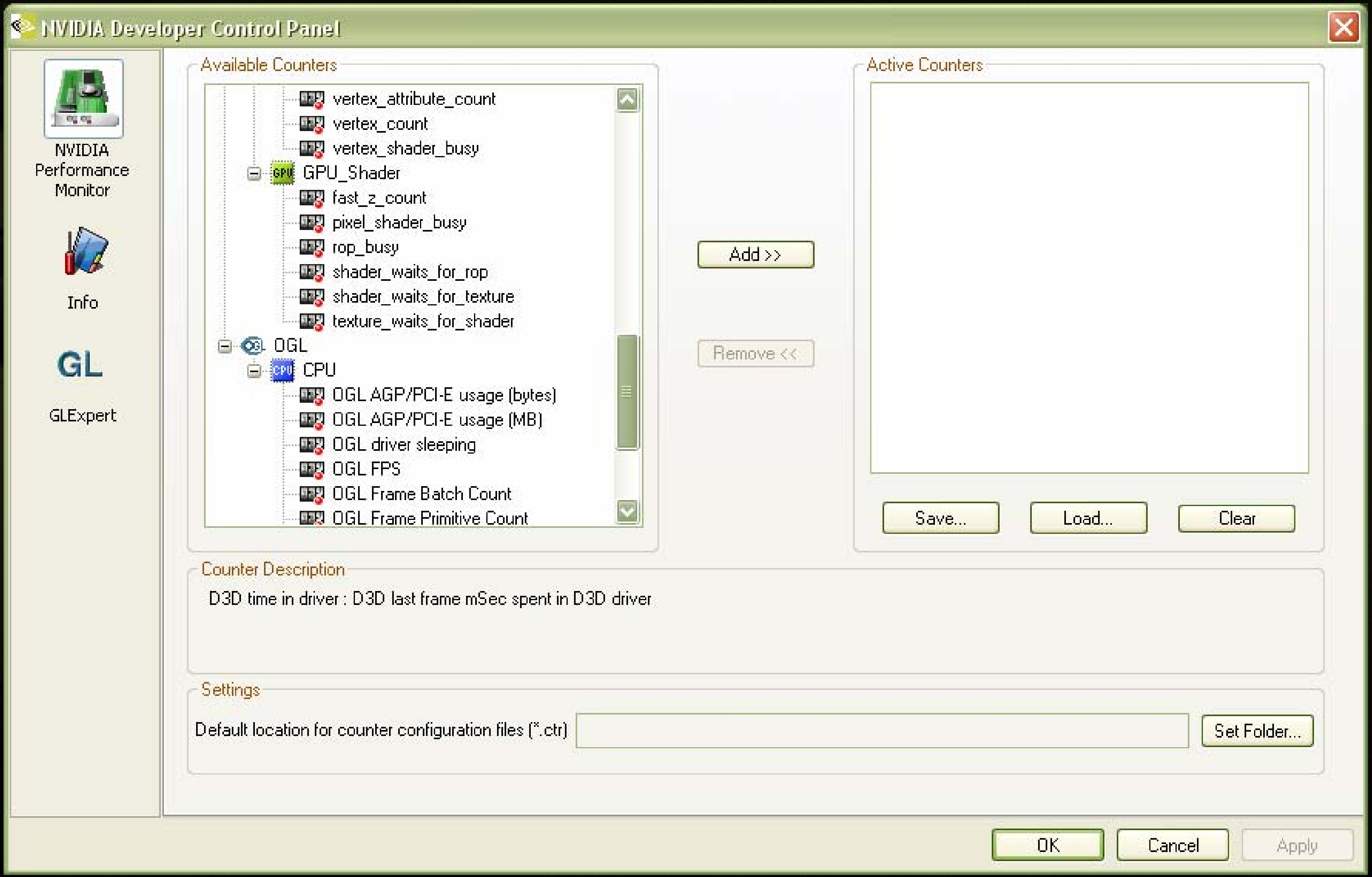
Freezing the application

- 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?

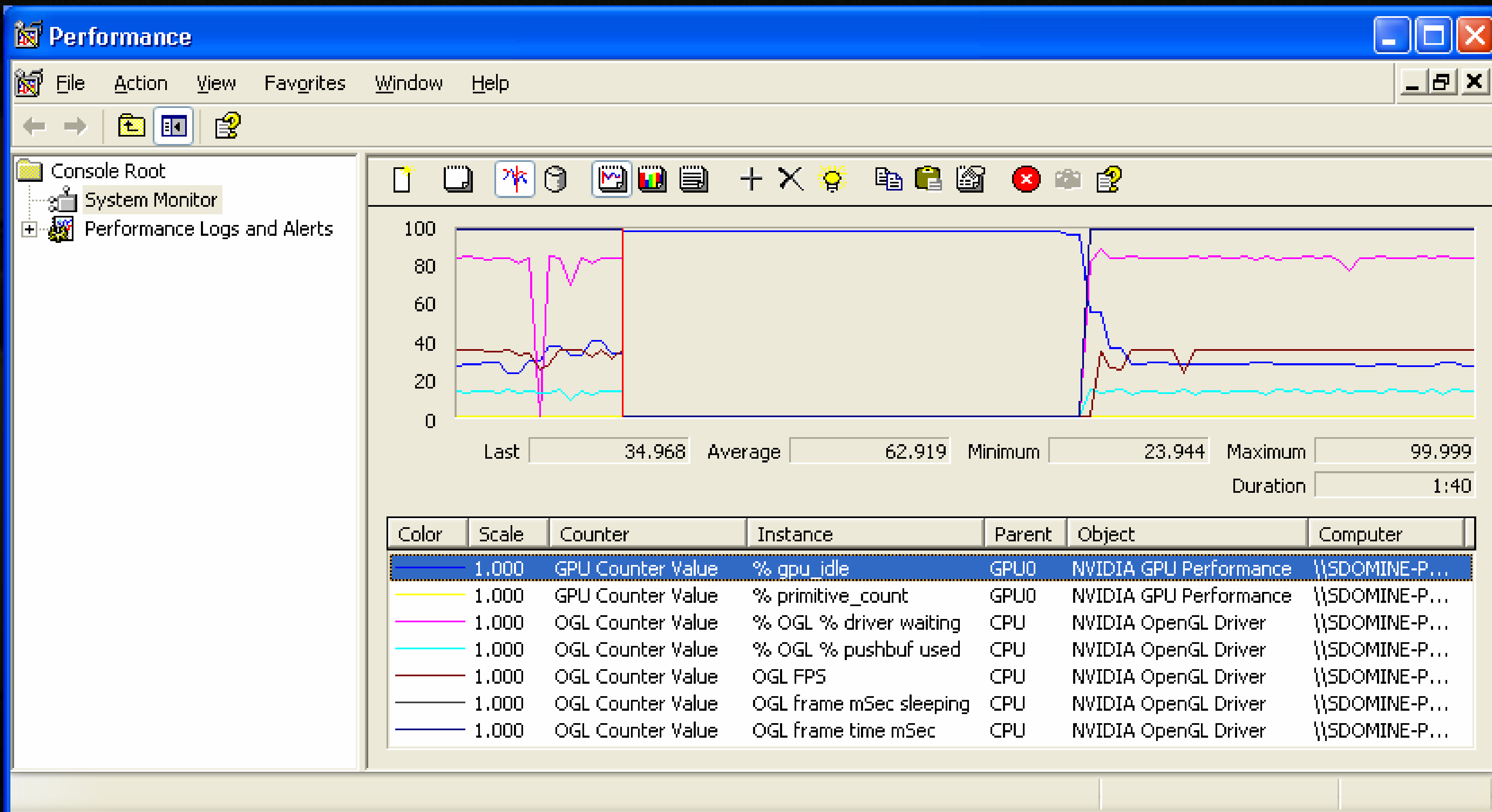
- PDH: Performance Data Helper for Windows
  - Win32 API for exposing performance data to user applications
  - Standard interface, many providers and clients
  - Sample code and helper classes provided in NVPerfSDK
- Perfmon: (aka Microsoft Management Console)
  - Win32 PDH client application
  - Perfmon’s 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



SIGGRAPH2006

Graphing results: Perfmon



NVPerfAPI

- 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

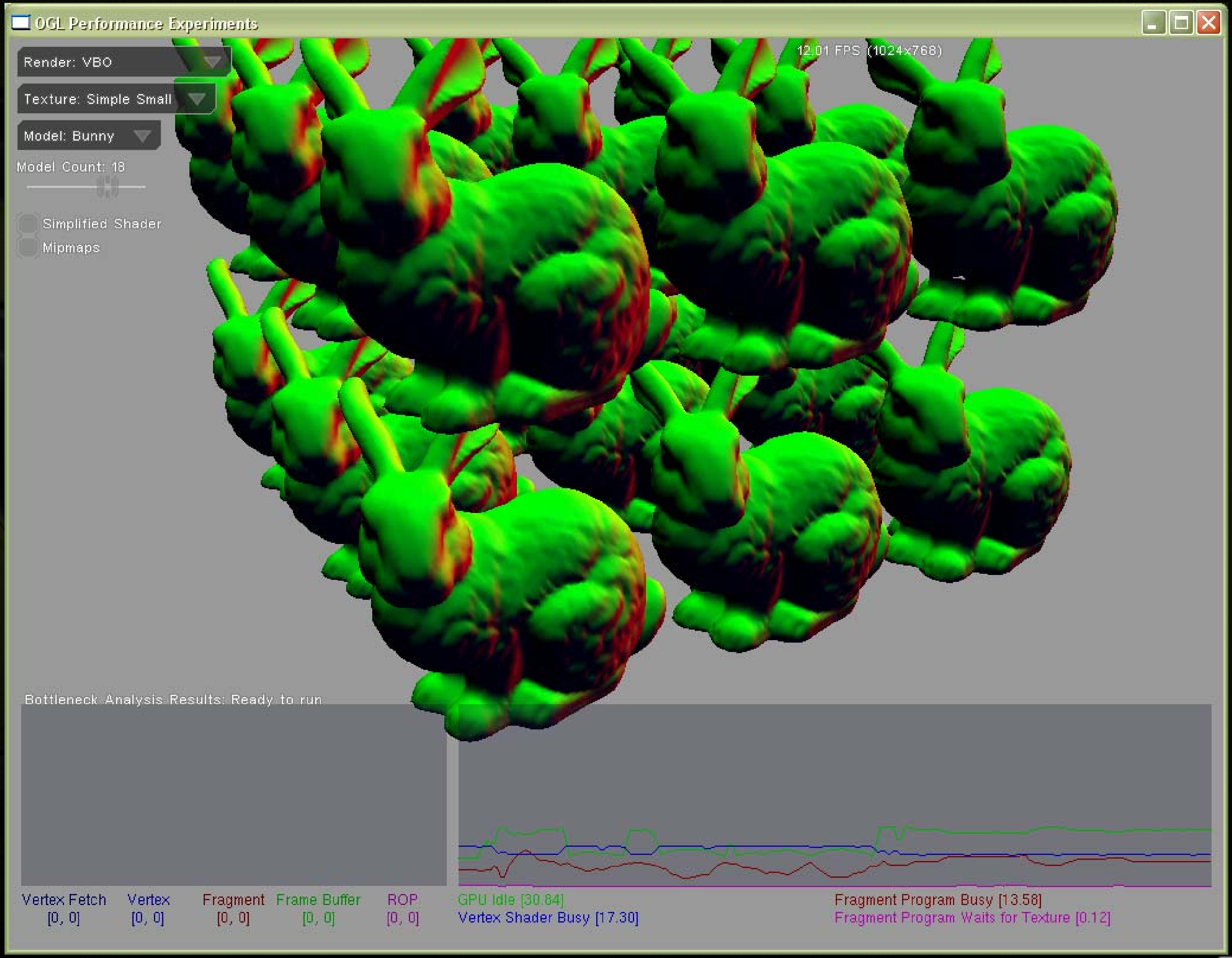
NVPerfAPI: Real Time

```
// 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

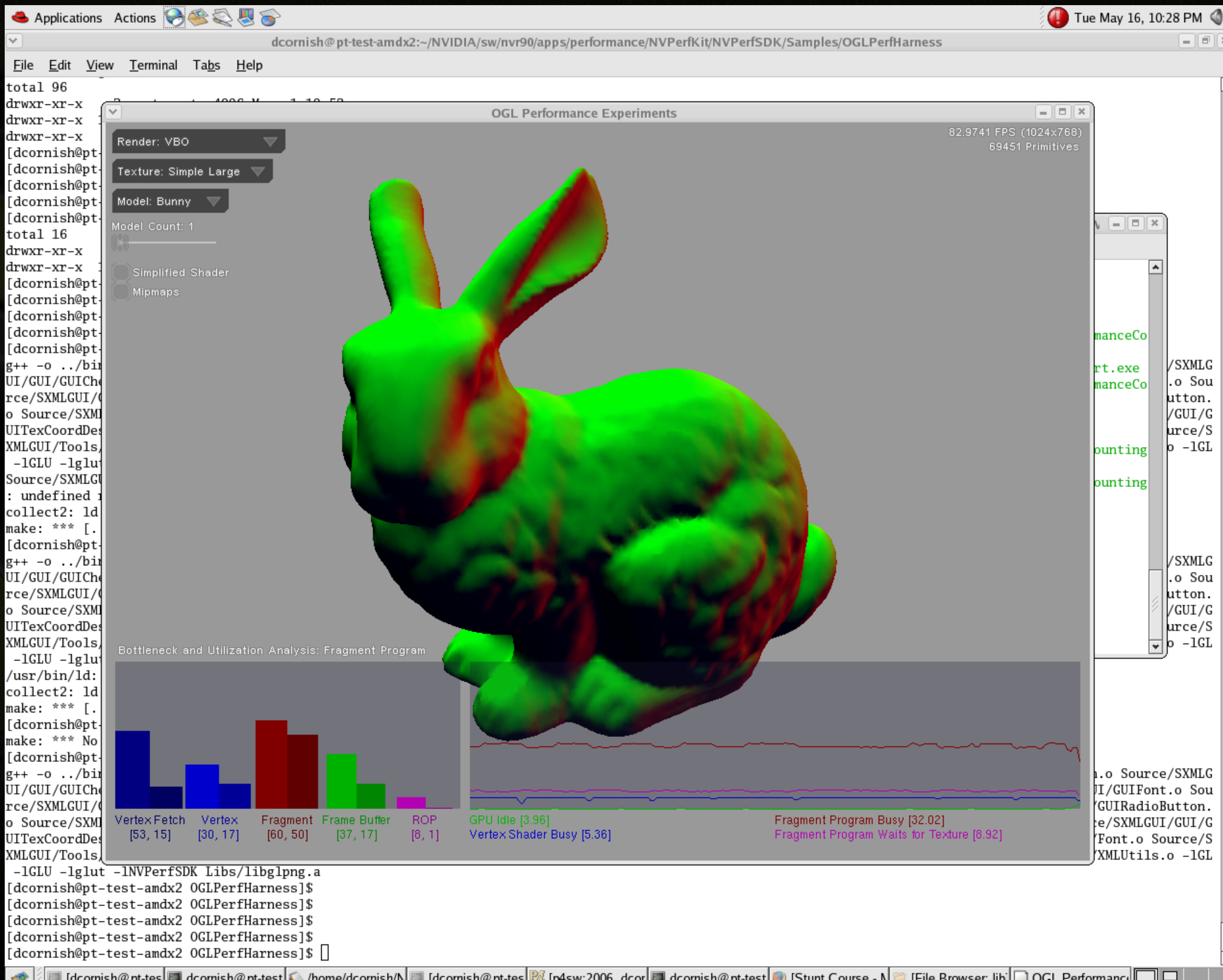
```
NVPMAddCounter("GPU Bottleneck");
NVPMAllocObjects(50);

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

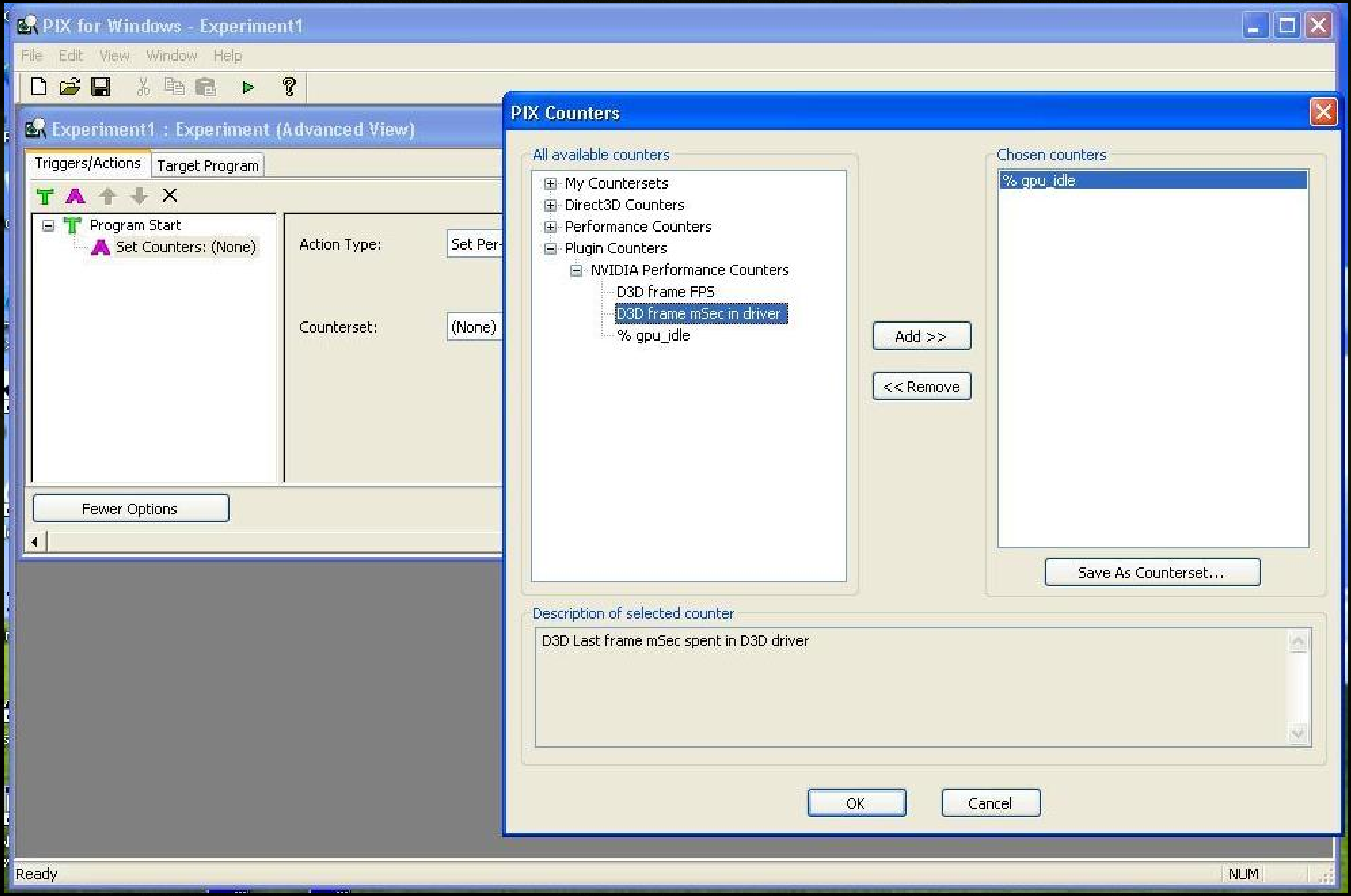
- GPU Bottleneck experiment
  - Run bottleneck and utilization experiments on all units
  - Process results to find bottlenecked unit
- Individual unit information can be queried
- Can run individual unit experiments
- Events: % utilization or % bottleneck…best way to visualize data
- Cycles: microseconds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

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

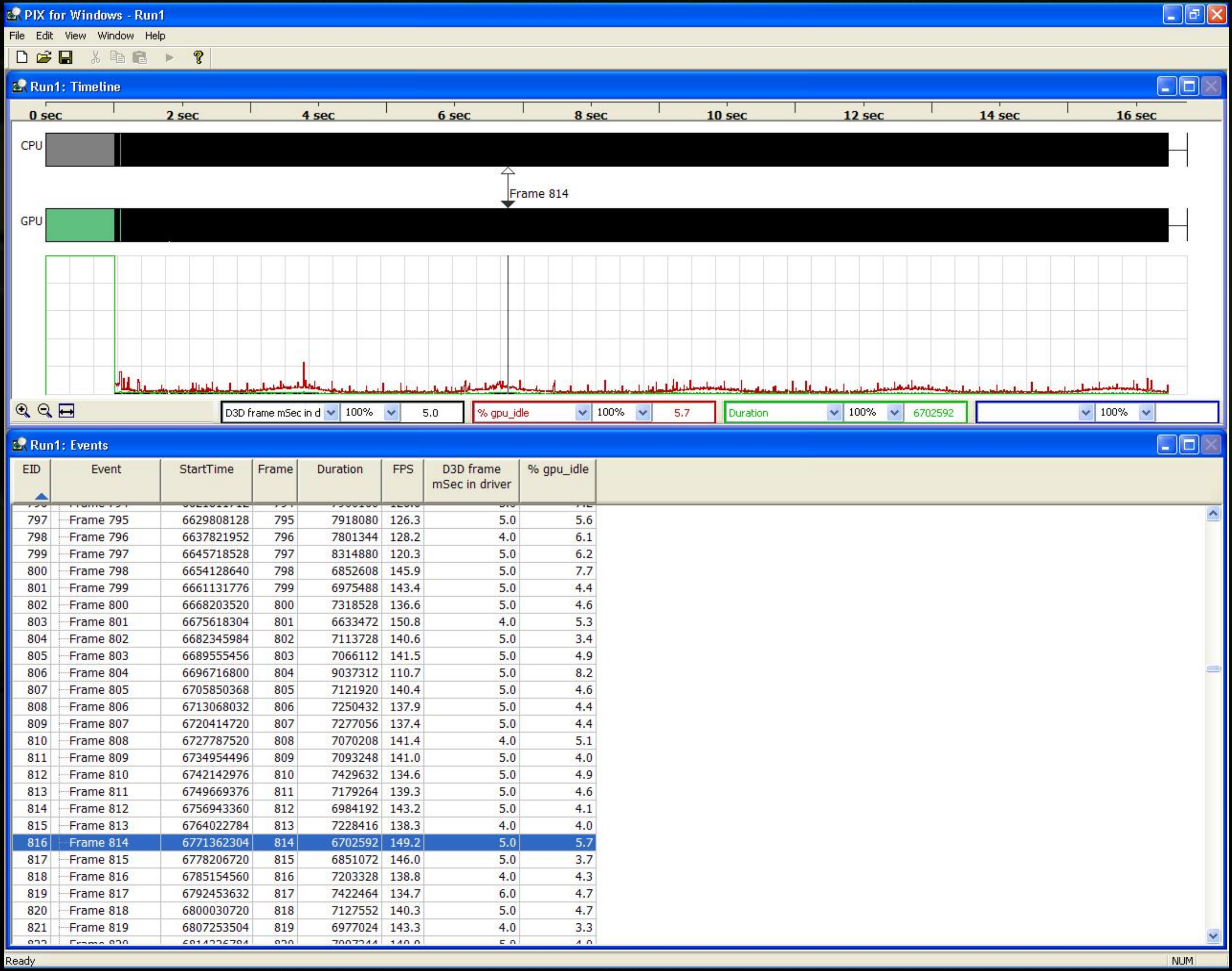
NVPerfAPI: SimExp



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



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



EID	Event	StartTime	Frame	Duration	FPS	D3D frame mSec in driver	% gpu_idle
797	Frame 795	6629808128	795	7918080	126.3	5.0	5.6
798	Frame 796	6637821952	796	7801344	128.2	4.0	6.1
799	Frame 797	6645718528	797	8314880	120.3	5.0	6.2
800	Frame 798	6654128640	798	6852608	145.9	5.0	7.7
801	Frame 799	6661131776	799	6975488	143.4	5.0	4.4
802	Frame 800	6668203520	800	7318528	136.6	5.0	4.6
803	Frame 801	6675618304	801	6633472	150.8	4.0	5.3
804	Frame 802	6682345984	802	7113728	140.6	5.0	3.4
805	Frame 803	6689555456	803	7066112	141.5	5.0	4.9
806	Frame 804	6696716800	804	9037312	110.7	5.0	8.2
807	Frame 805	6705850368	805	7121920	140.4	5.0	4.6
808	Frame 806	6713068032	806	7250432	137.9	5.0	4.4
809	Frame 807	6720414720	807	7277056	137.4	5.0	4.4
810	Frame 808	6727787520	808	7070208	141.4	4.0	5.1
811	Frame 809	6734954496	809	7093248	141.0	5.0	4.0
812	Frame 810	6742142976	810	7429632	134.6	5.0	4.9
813	Frame 811	6749669376	811	7179264	139.3	5.0	4.6
814	Frame 812	6756943360	812	6984192	143.2	5.0	4.1
815	Frame 813	6764022784	813	7228416	138.3	4.0	4.0
816	Frame 814	6771362304	814	6702592	149.2	5.0	5.7
817	Frame 815	6778206720	815	6851072	146.0	5.0	3.7
818	Frame 816	6785154560	816	7203328	138.8	4.0	4.3
819	Frame 817	6792453632	817	7422464	134.7	6.0	4.7
820	Frame 818	6800030720	818	7127552	140.3	5.0	4.7
821	Frame 819	6807253504	819	6977024	143.3	4.0	3.3

SIGGRAPH2006

Project Status

- NVPerfKit 2.0 for Windows 32bit available now at 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

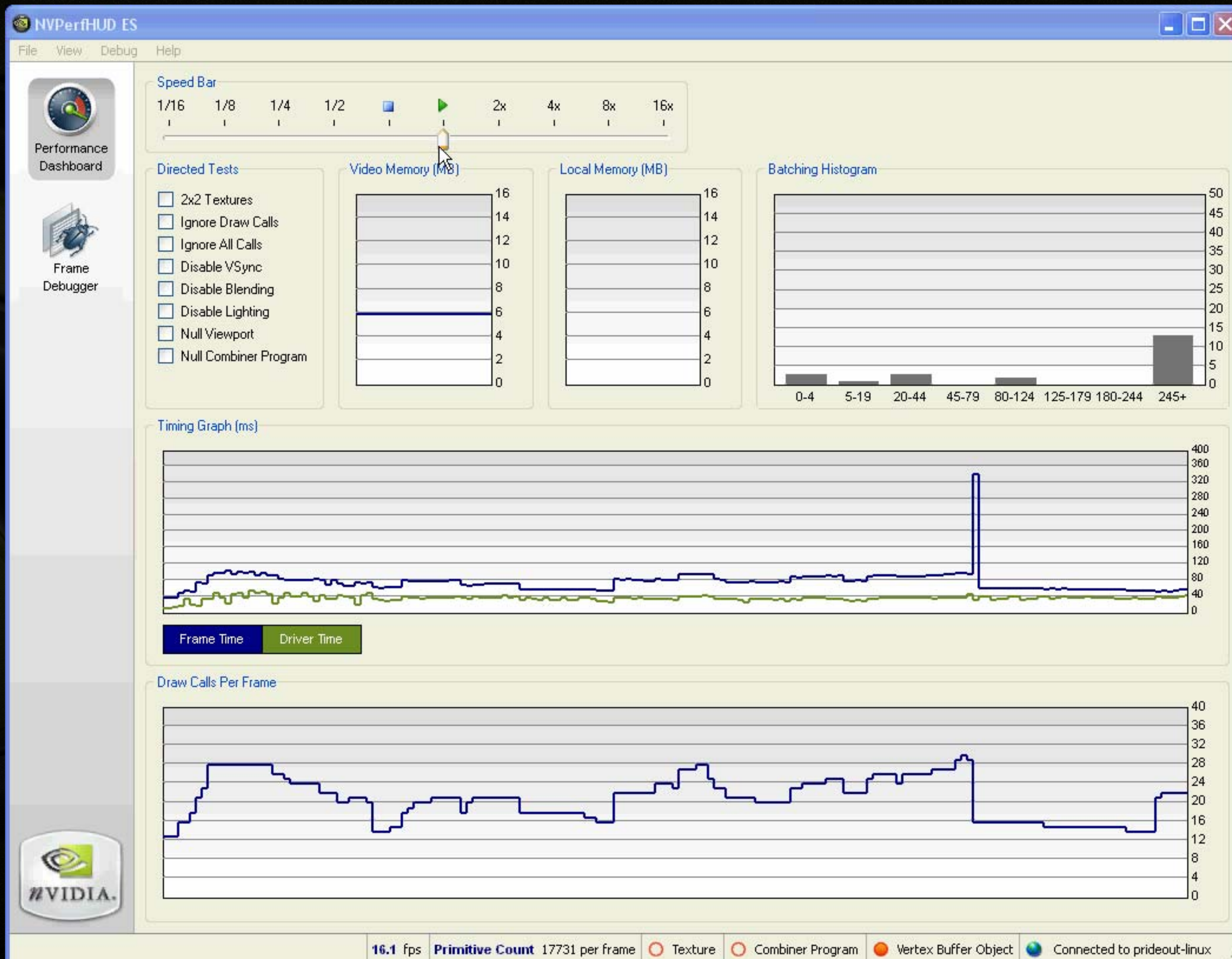
GeForce 6800 Ultra & GT
GeForce 6600

Feedback and Support: NVPerfKit@nvidia.com

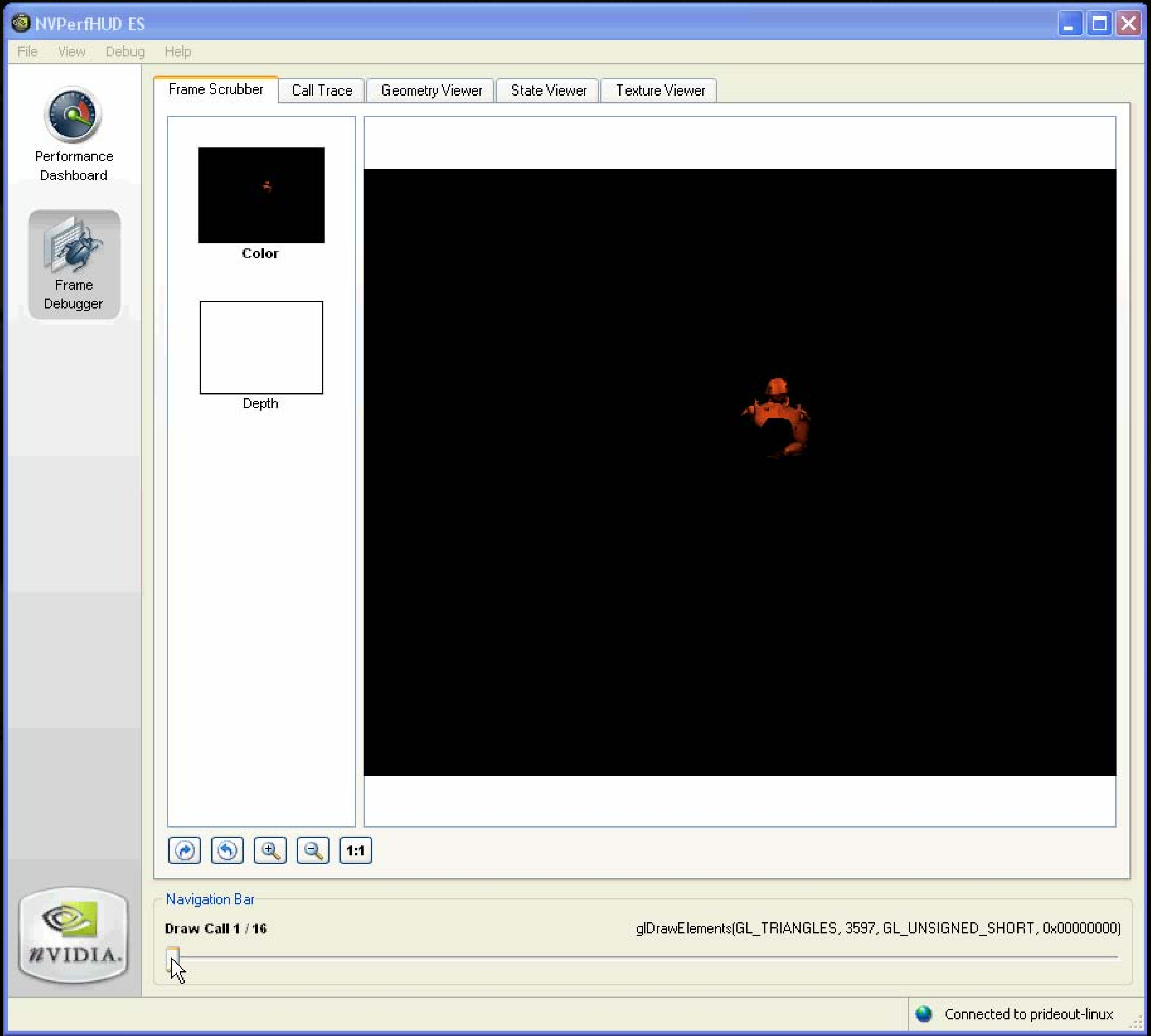
NVPerfHUD ES

- 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!

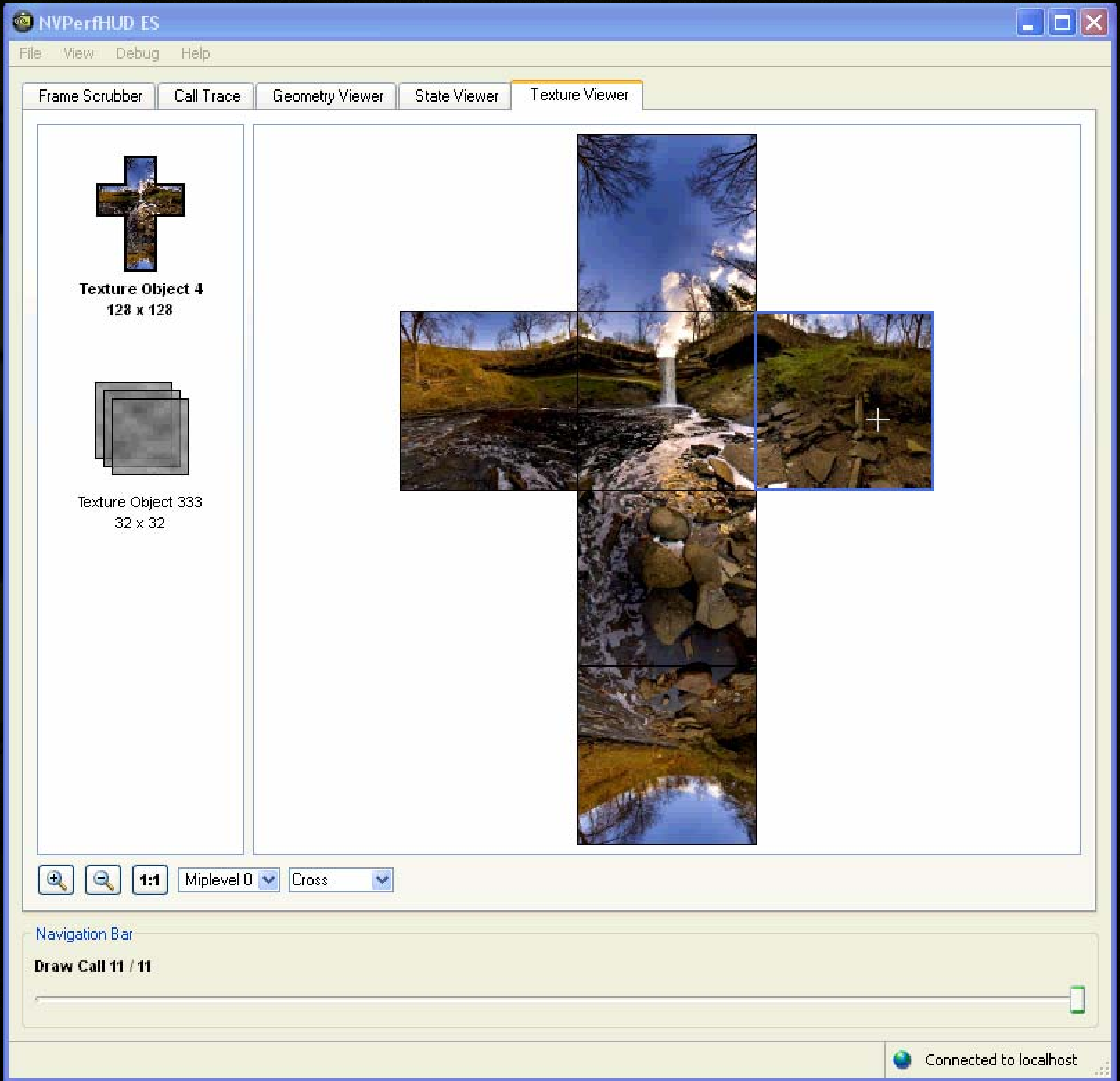
Performance Dashboard



Debugging Features



Texture Viewer



Graphic Remedy's gDEBugger

Free gDEBugger 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?

NVShaderPerf

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

GPU Arch:
- GeForce 7X00
- GeForce 6X00
- Geforce FX series
- Quadro FX series

NVShaderPerf

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
```

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

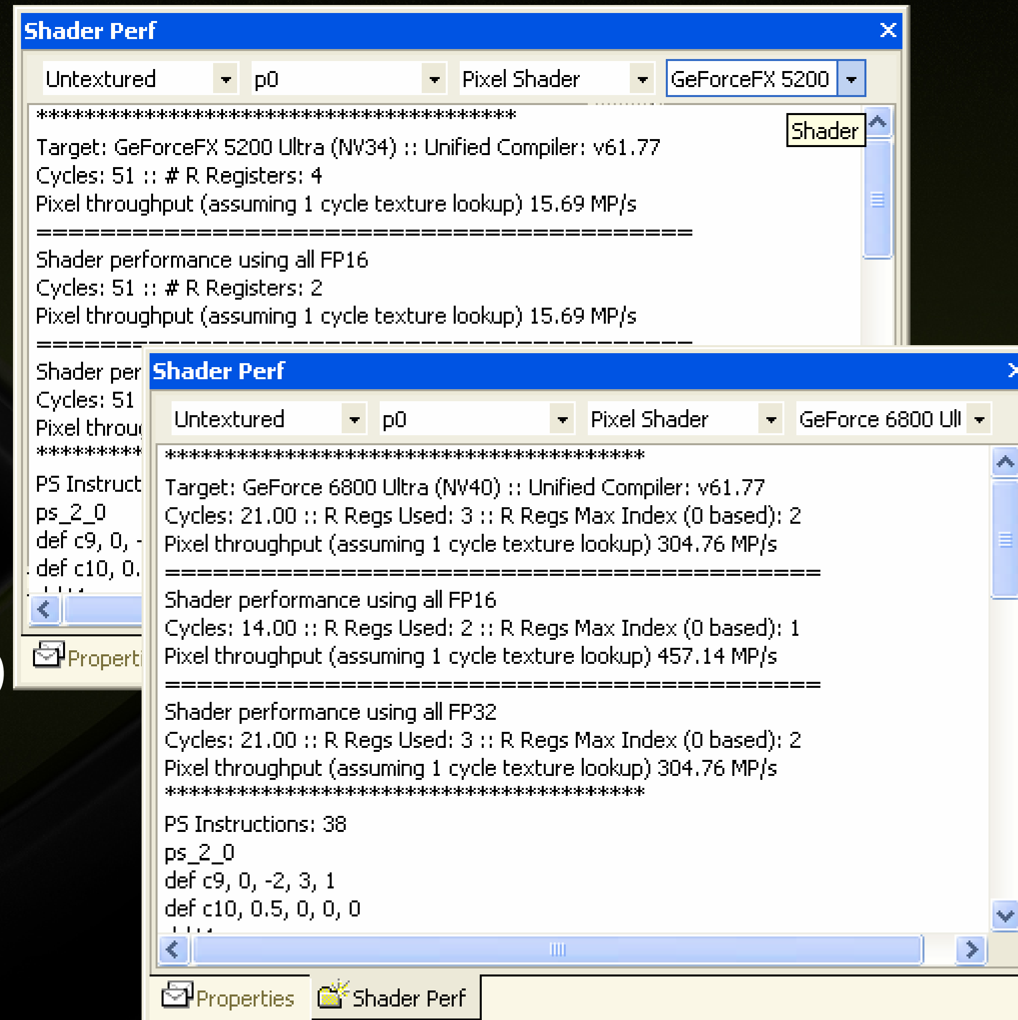
```
Shader performance using all FP16
Cycles: 14.00 :: R Regs Used: 2 :: R Regs Max Index (0 bas
Pixel throughput (assuming 1 cycle texture lookup) 457.14

Shader performance using all FP32
Cycles: 21.00 :: R Regs Used: 3 :: R Regs Max Index (0 bas
Pixel throughput (assuming 1 cycle texture lookup) 304.76

C:\Temp\NVShaderPerf_61_77>
```

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)



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
objBumpNormal = 0.350000 * objBumpNormal + ( 1 - 0.350000 ) * objFlatNorm
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 - objEy
  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 NV40 cornea2.txt
-------------------------------------------------------------------------------

Running performance on file Cornea2.txt
-------------------- NV40 --------------------
Target: GeForce 6800 Ultra (NV40) :: 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 NV40 cornea2.txt
-------------------------------------------------------------------------------

Running performance on file Cornea2.txt
-------------------- NV40 --------------------
Target: GeForce 6800 Ultra (NV40) :: 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

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
  NVPerfHUD@nvidia.com
  NVShaderPerf@nvidia.com
  FXComposer@nvidia.com


The Source for
GPU Programming
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
©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.