NVIDIA®

Performance Tools

Raul Aguaviva and Jeff Kiel (NVIDIA Corporation)

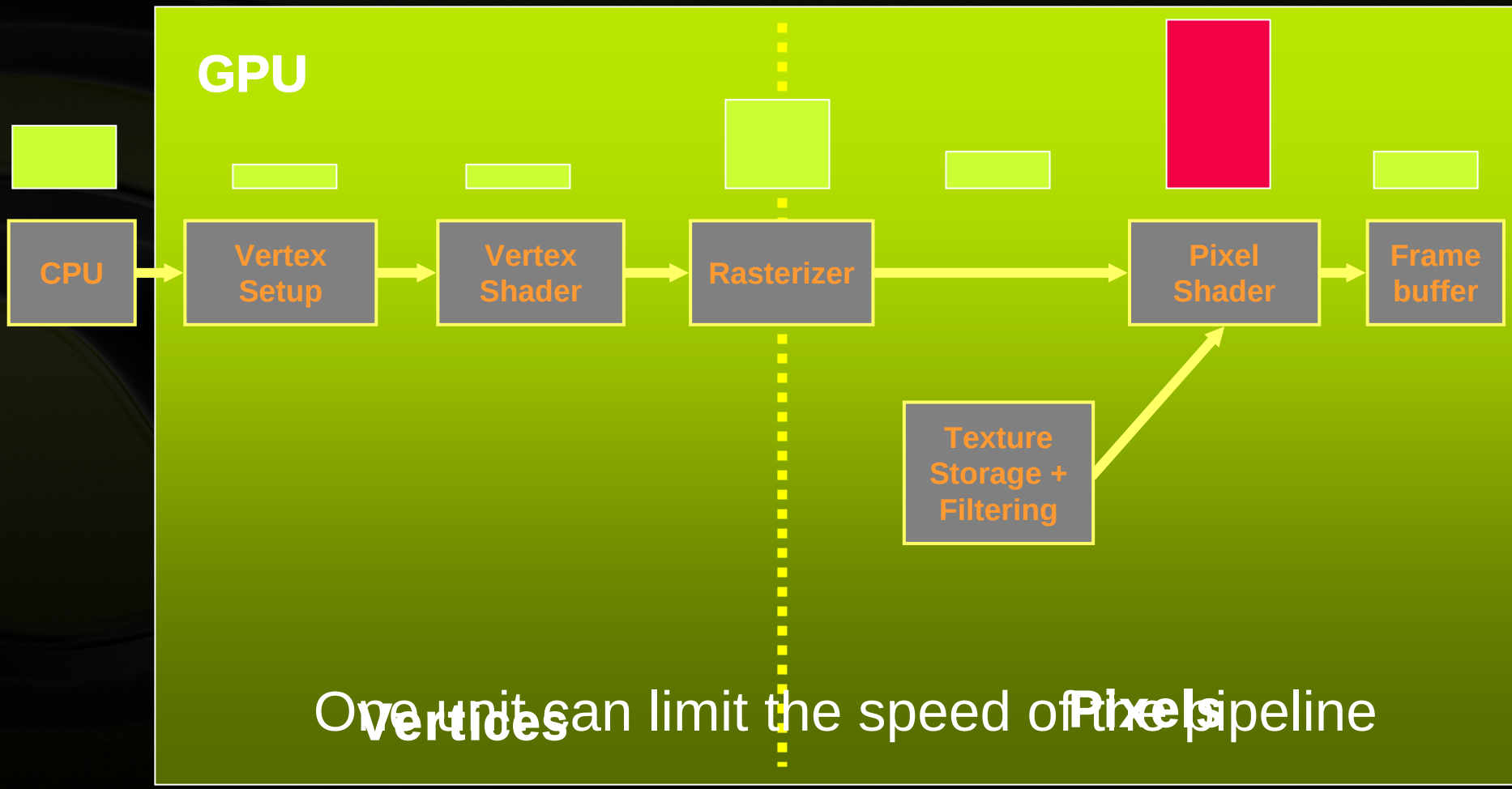
Performance Tools Agenda

- 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

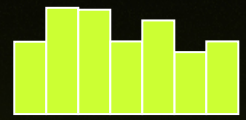
- Pipelined architecture
  - Each unit needs the data from the previous unit to do its job
- Bottleneck identification and elimination
- Balancing the pipeline

GPU Pipelined Architecture (simplified view)

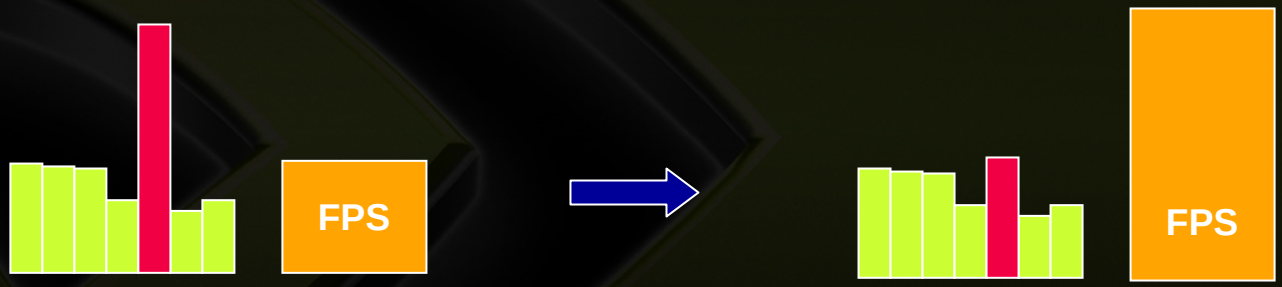


One unit can limit the speed of the pipeline

Bottleneck Identification

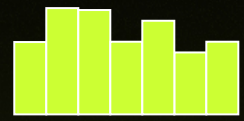


- Modify the stage itself
  - By decreasing its workload



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

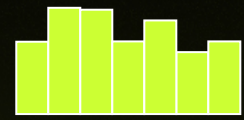
Bottleneck Identification



- 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

VTune

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

NVShaderPerf

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

NVShaderPerf

Inputs:

- HLSL
- 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

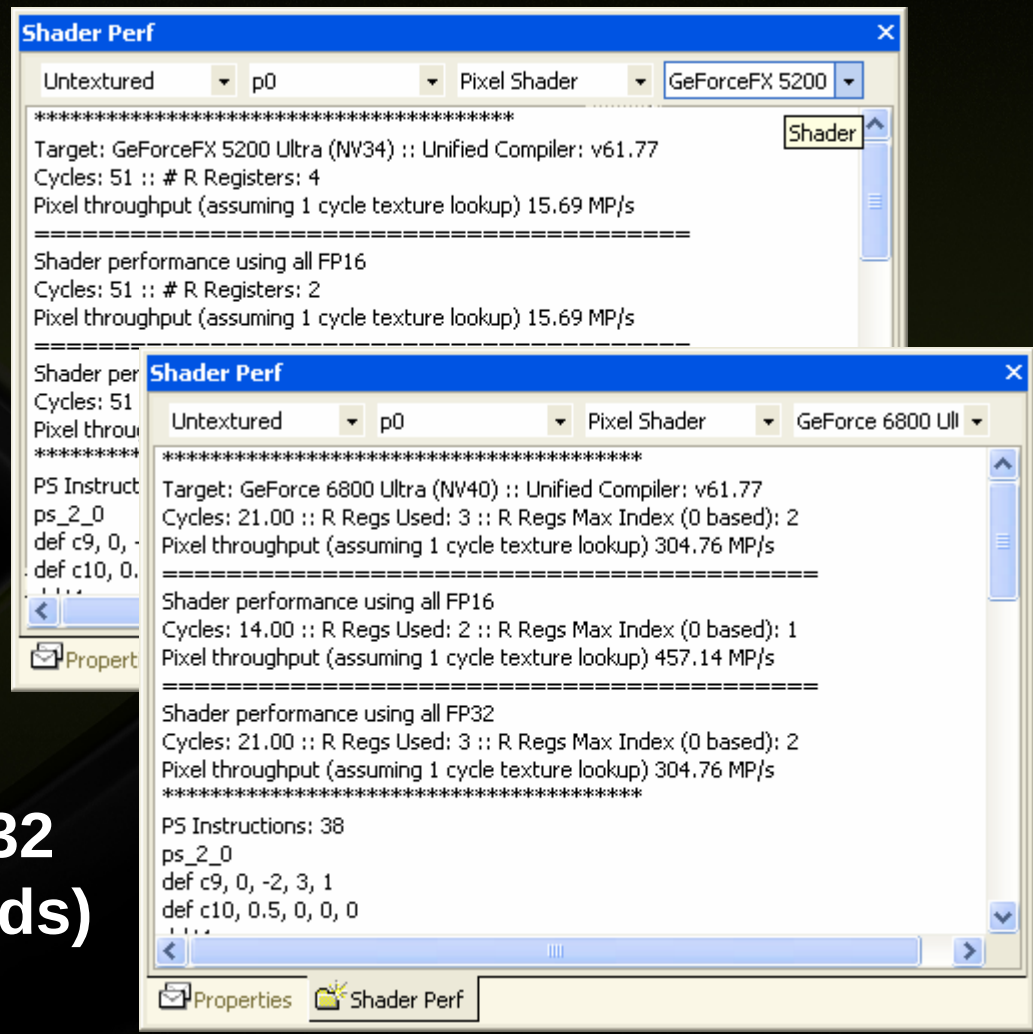
```
Shader performance using all FP32
Cycles: 21.00 :: R Regs Used: 3 :: R Regs Max Index (0 based)
Pixel throughput (assuming 1 cycle texture lookup) 304.76 MP

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_eyeNermel, 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 - 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 corn
ea2.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

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

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?

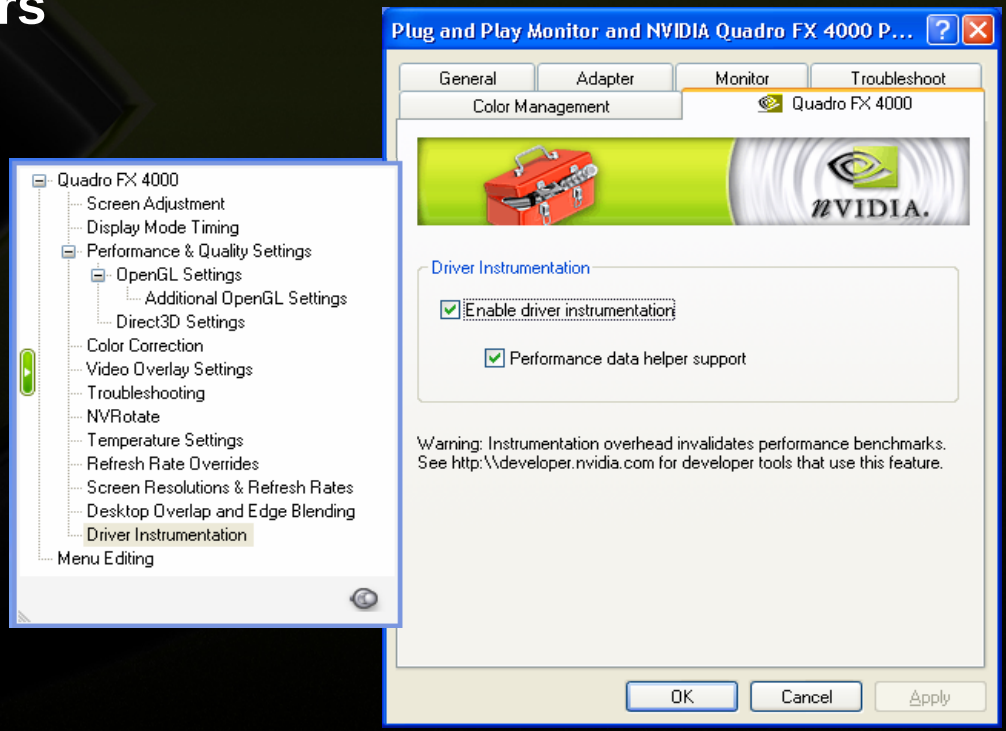
What is NVPerfKit?

- 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



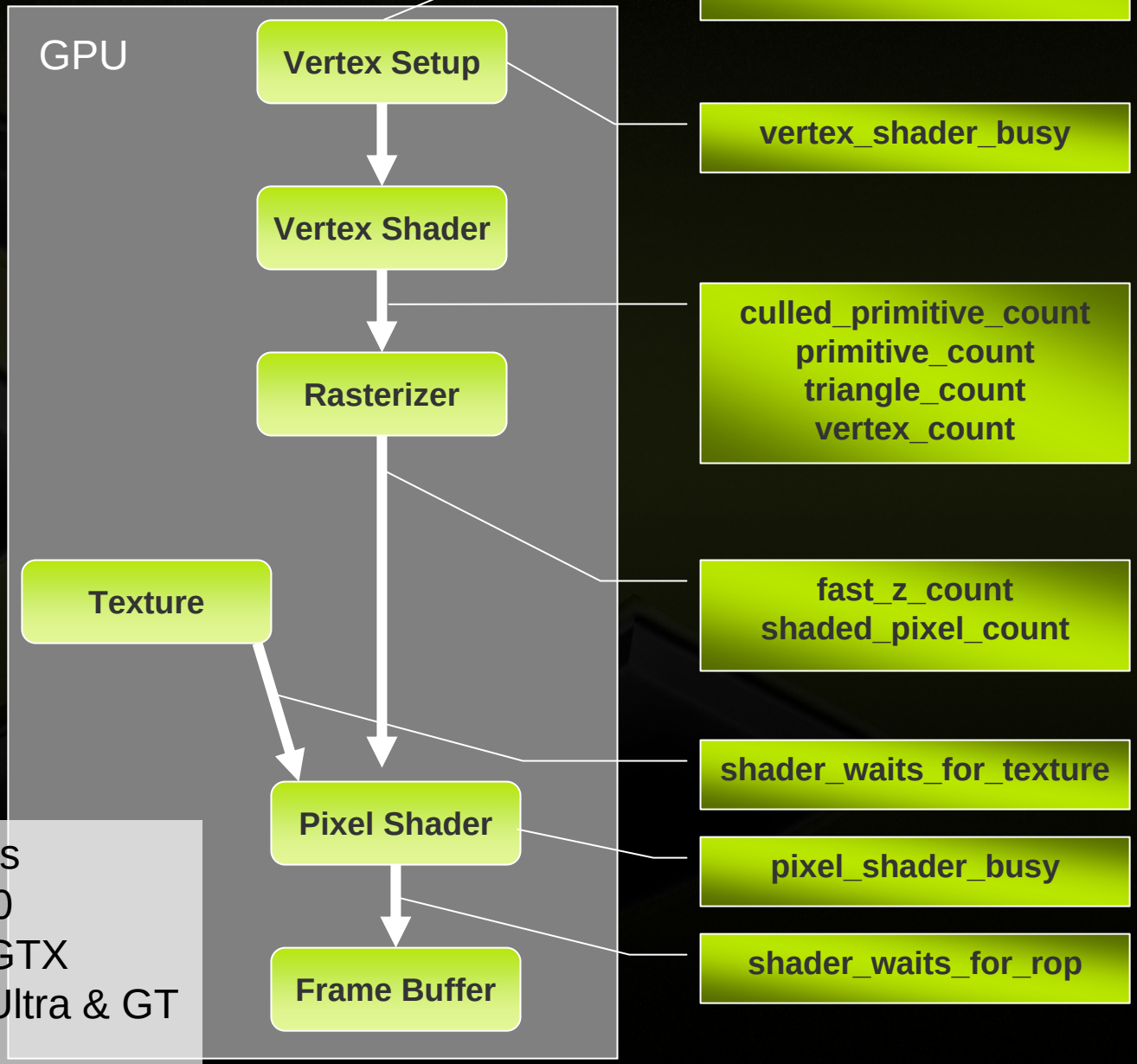
OpenGL Signals

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

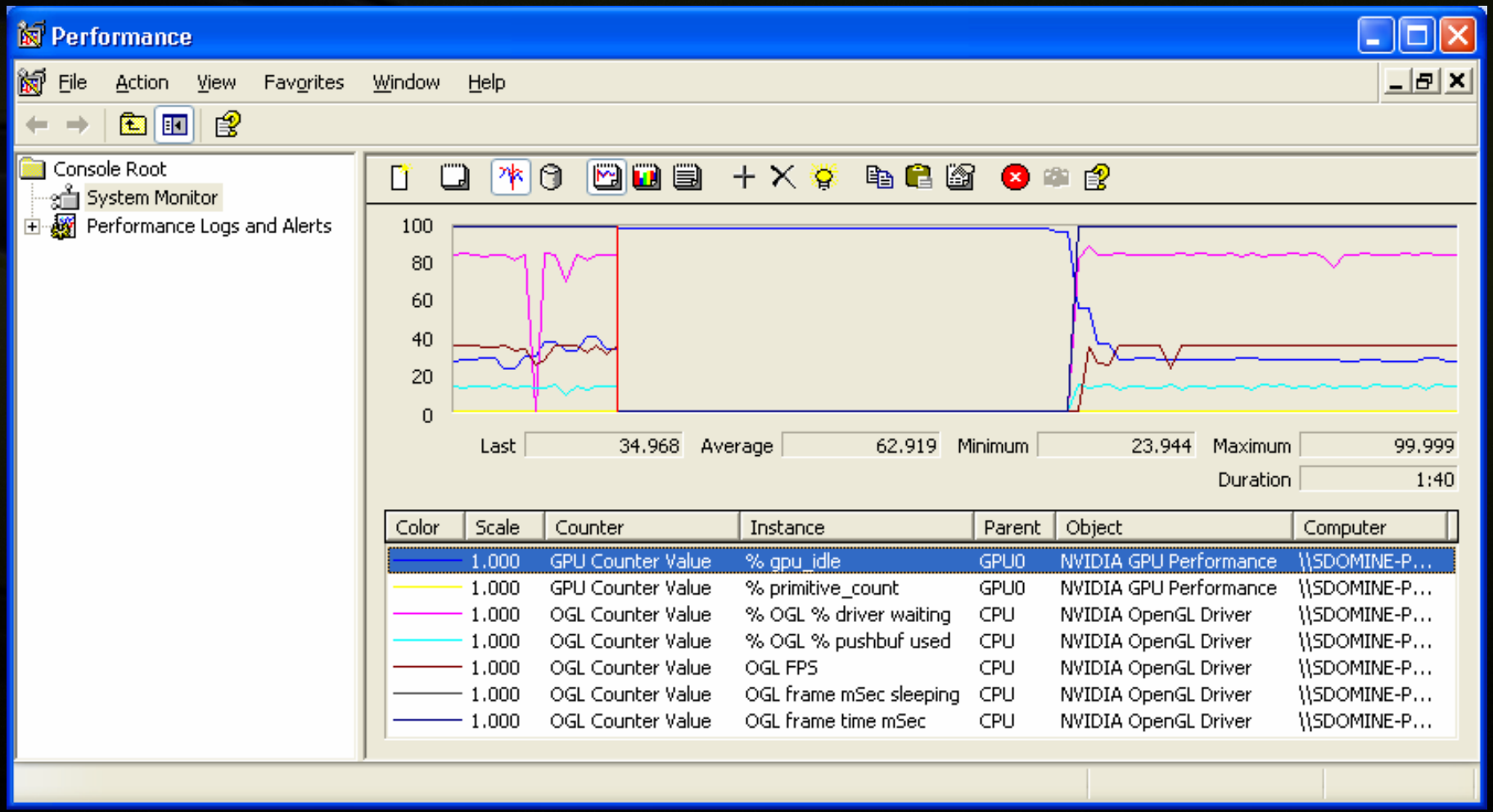
NVPerfKit Demo: Pixel Shader Bound

NVPerfKit Demo: Texture Bound

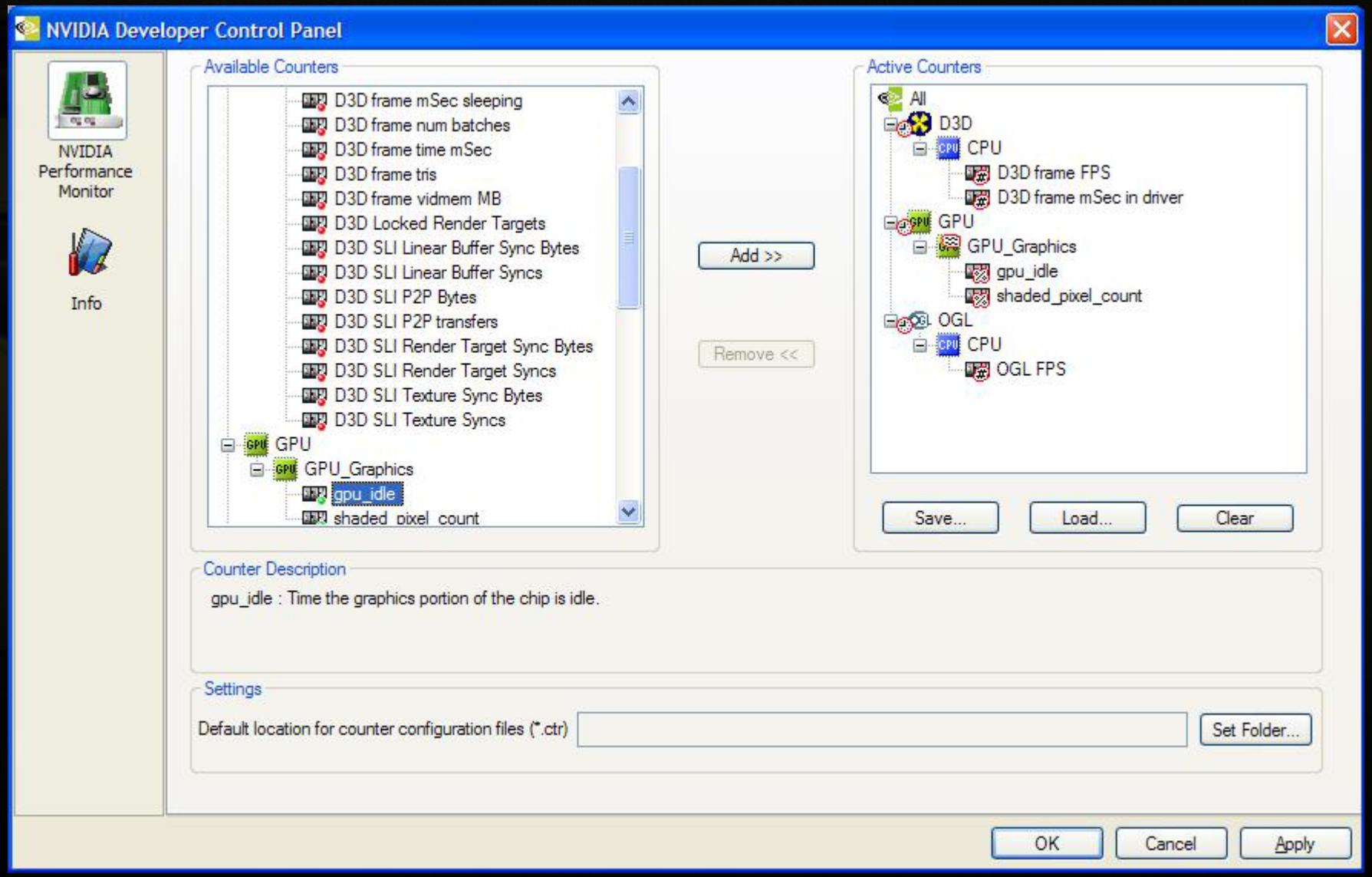
What is PDH? What is Perfmon?

- 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

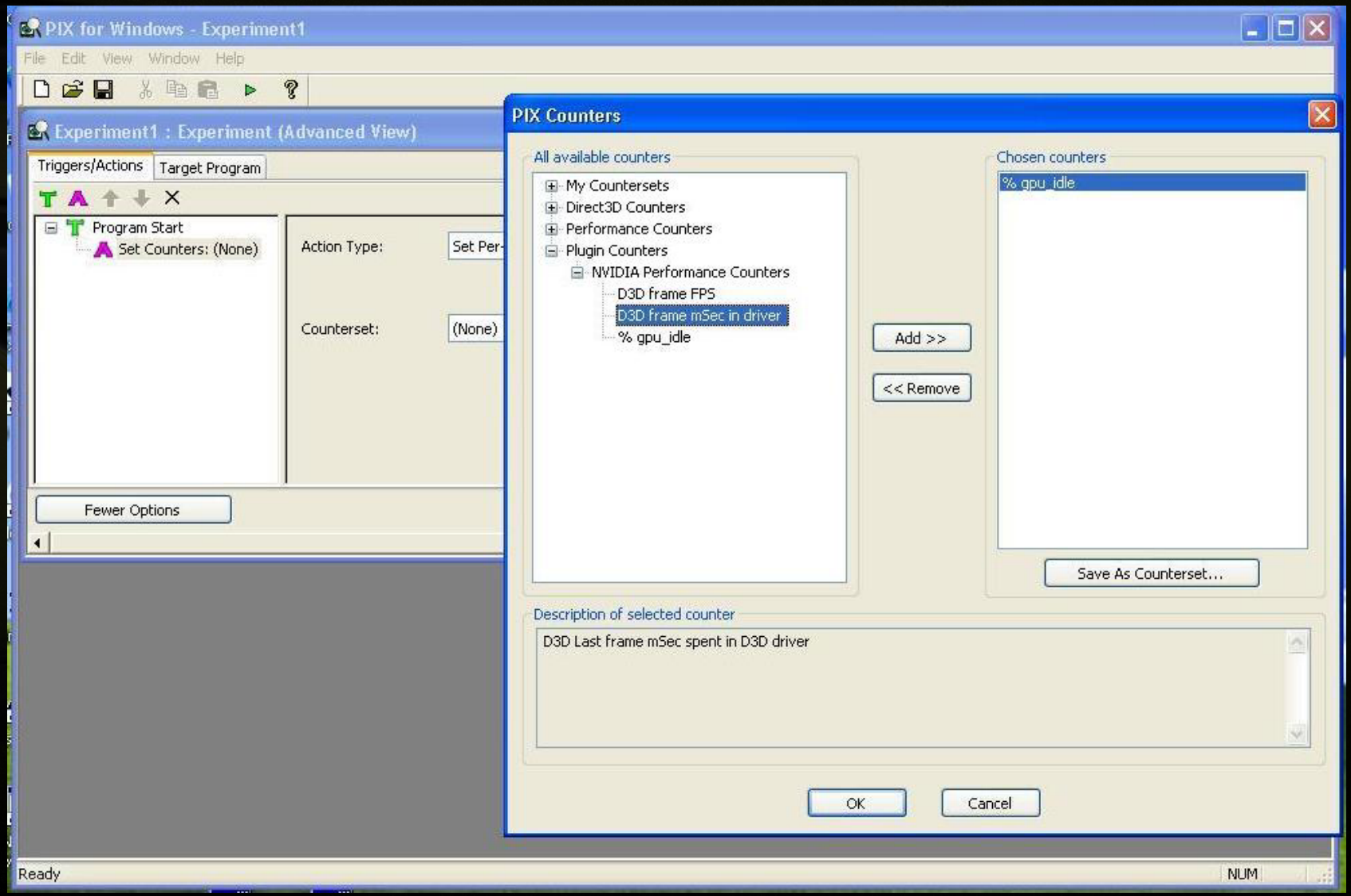
Associated Tools: Perfmon



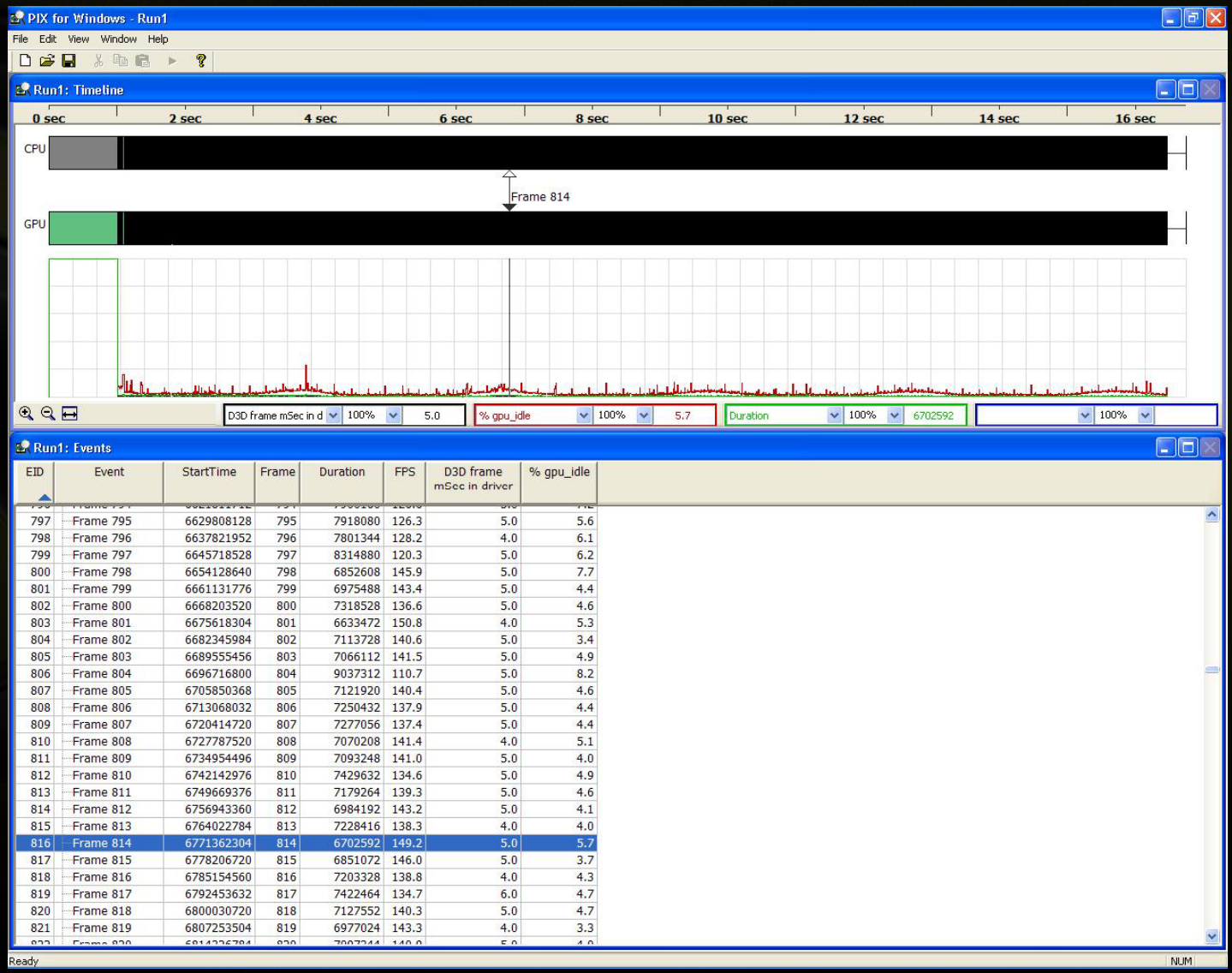
Associated Tools: NVDevCPL



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

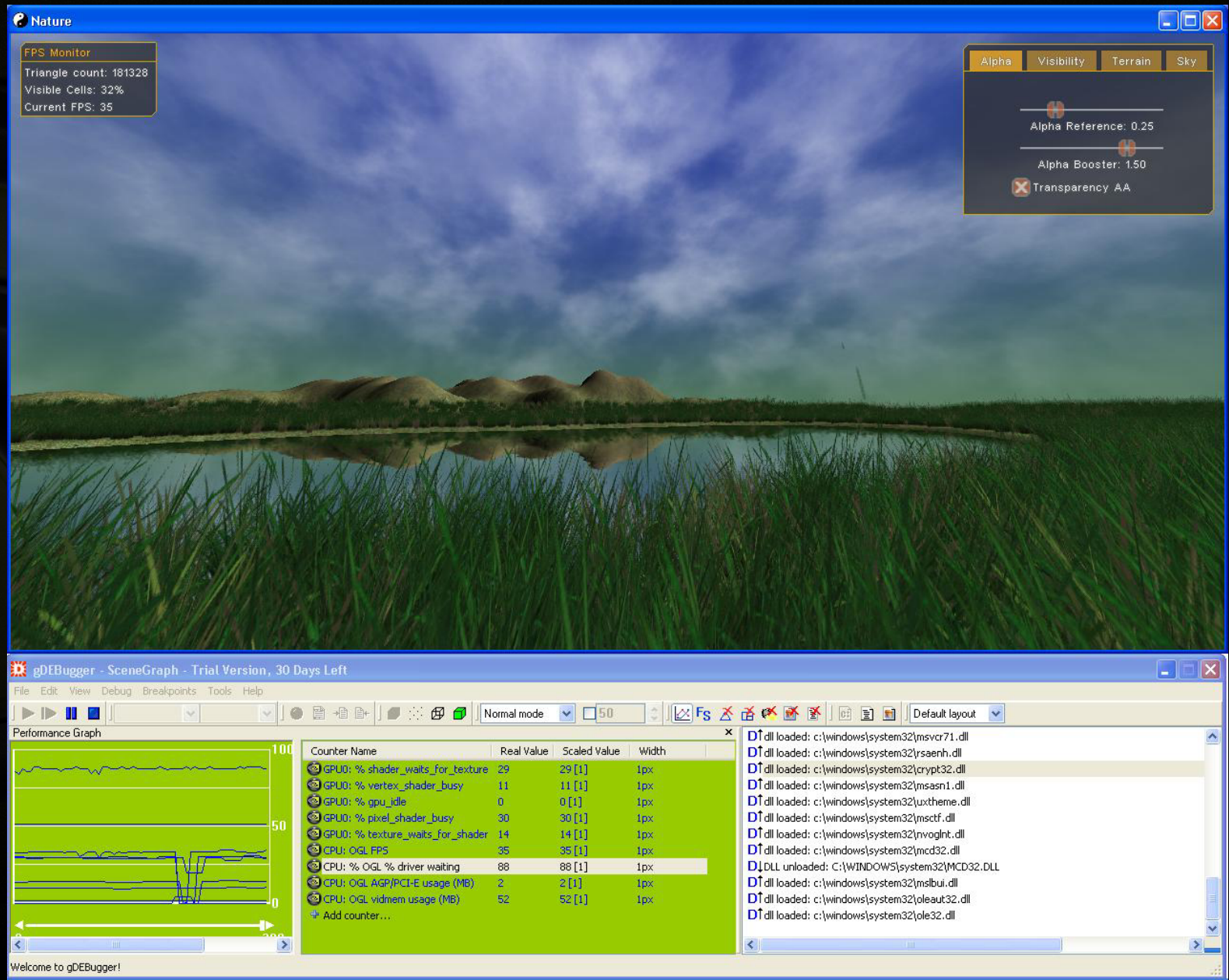
Helper Classes and Code Samples

- 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



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

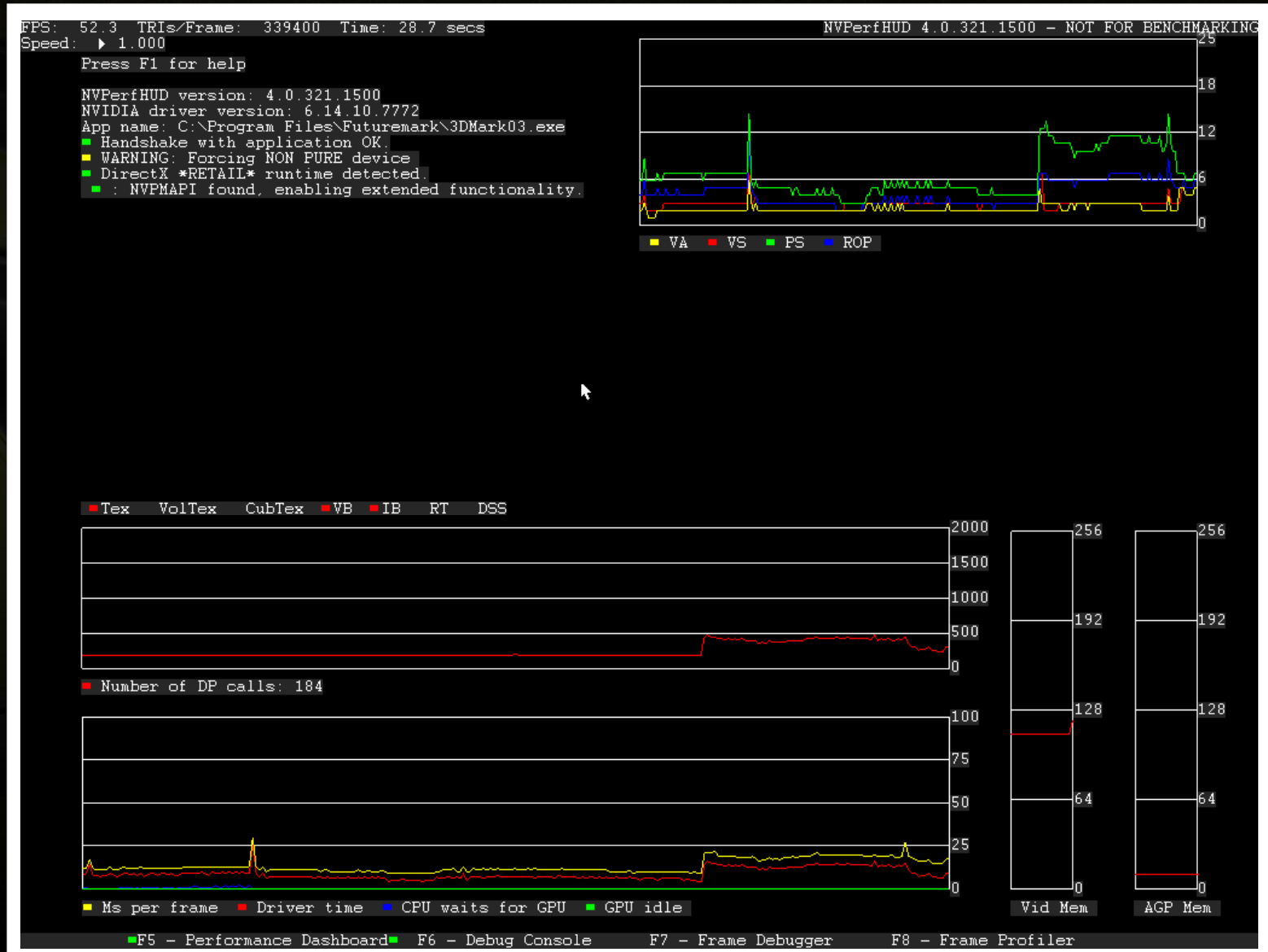
What is NVPerfHUD?

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

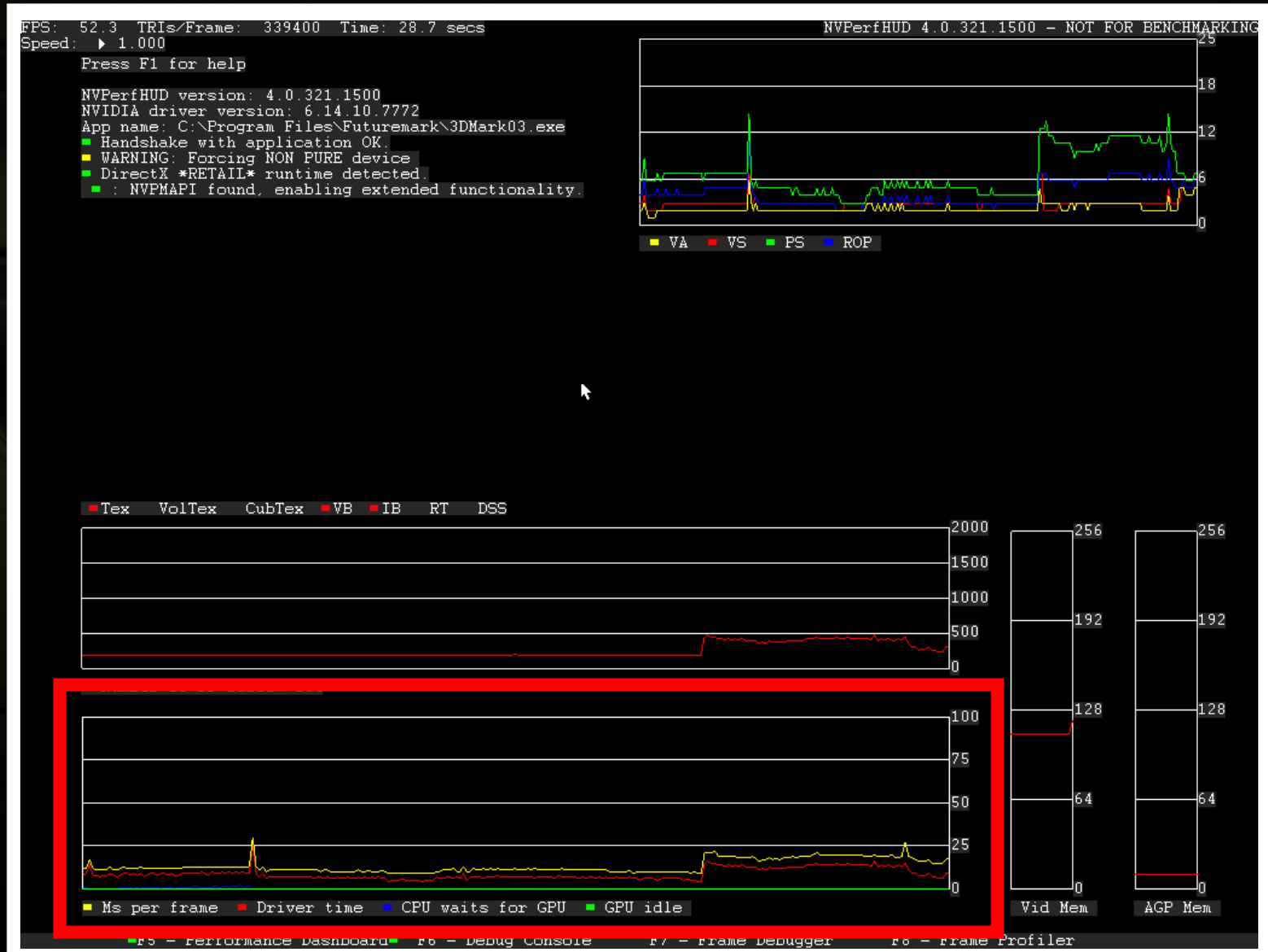
How to use it

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

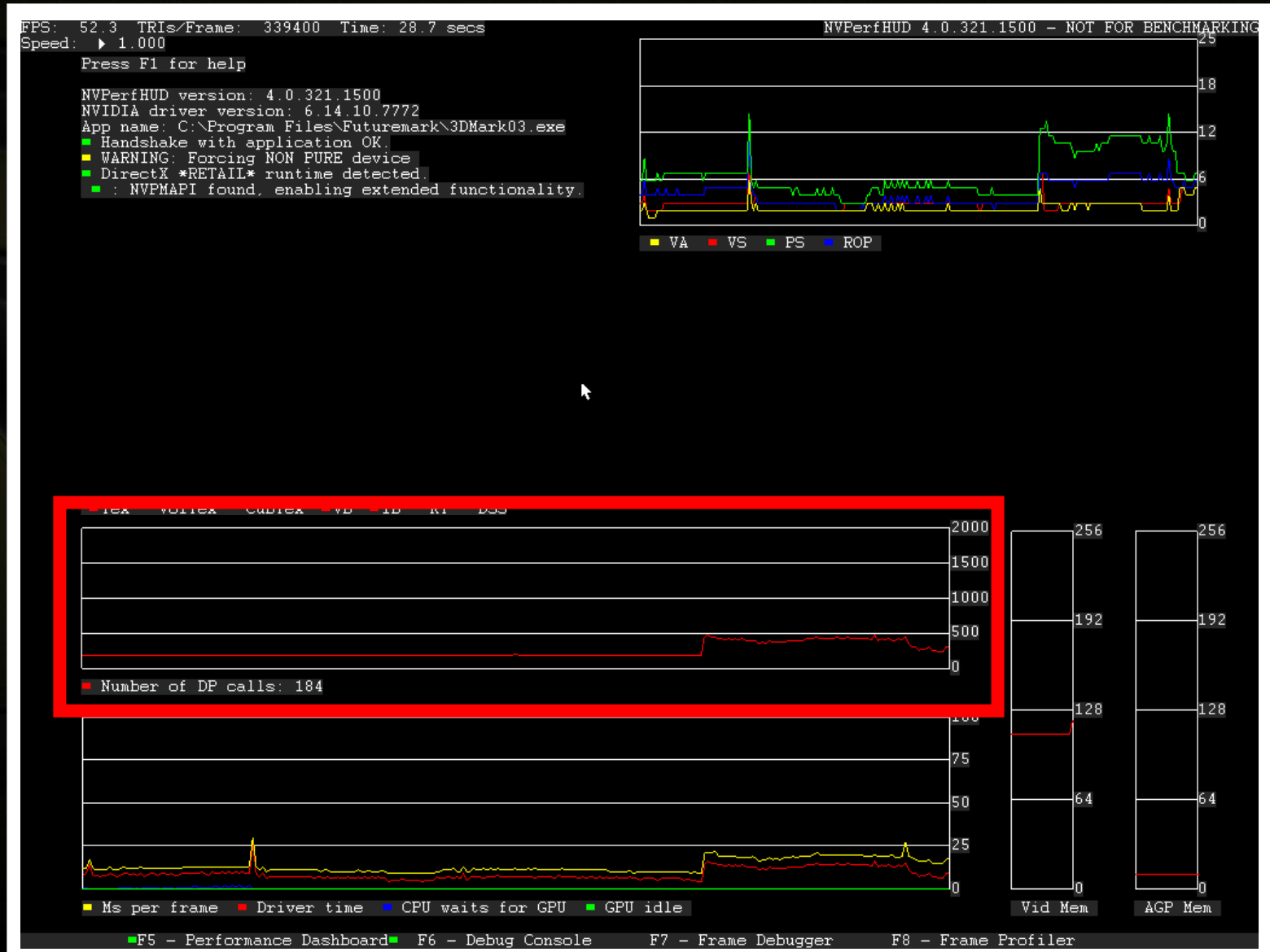
Performance Dashboard



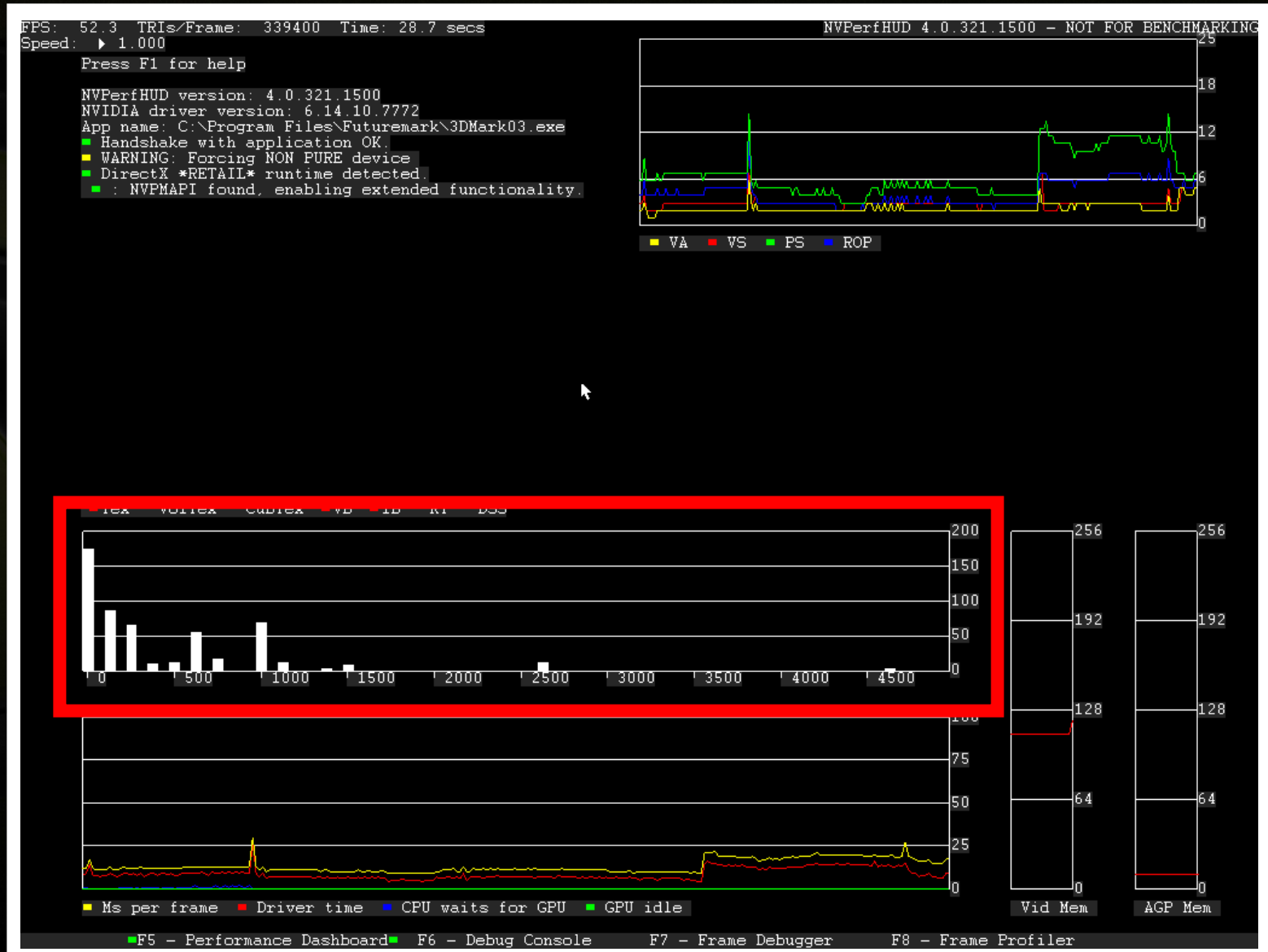
Performance Dashboard



Performance Dashboard



Performance Dashboard

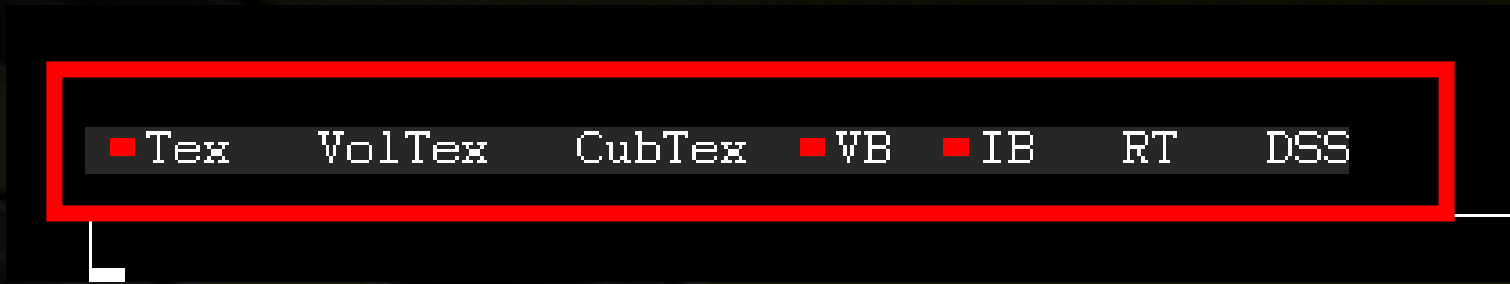


Performance Dashboard

Performance Dashboard

Performance Dashboard

- Resource monitor

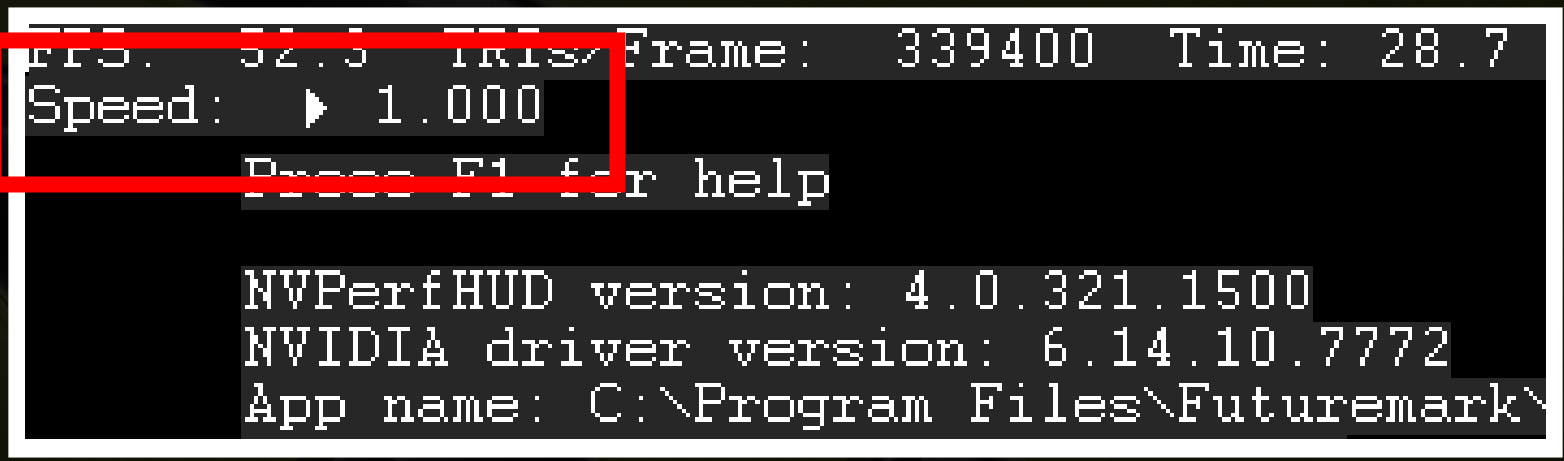


- Resources monitored
  - Textures
  - Volume Textures
  - Cube textures
  - Vertex Buffers
  - Index buffers
  - Stencil and depth surfaces

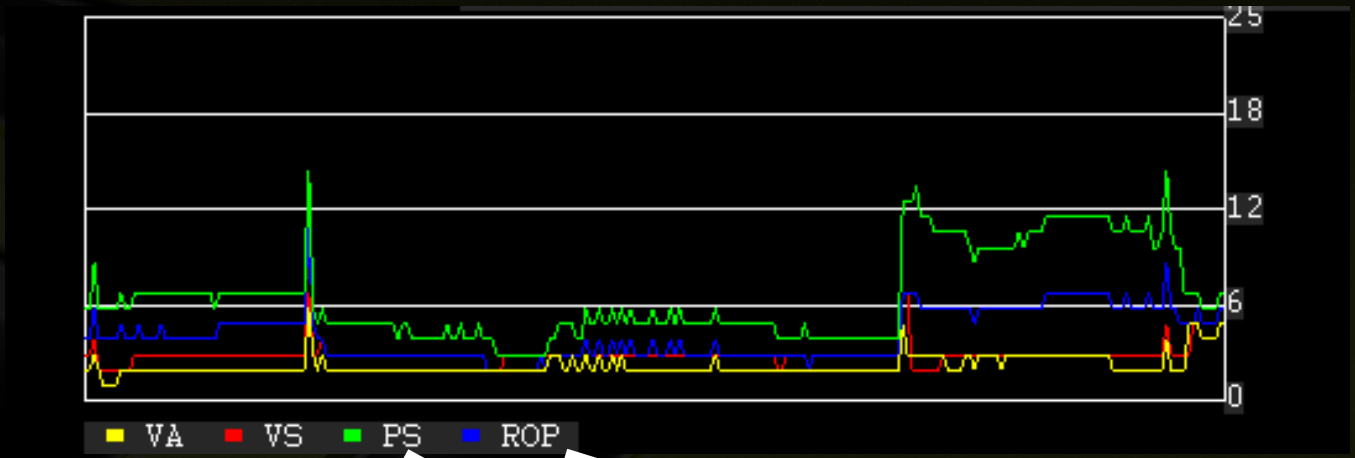
Performance Dashboard

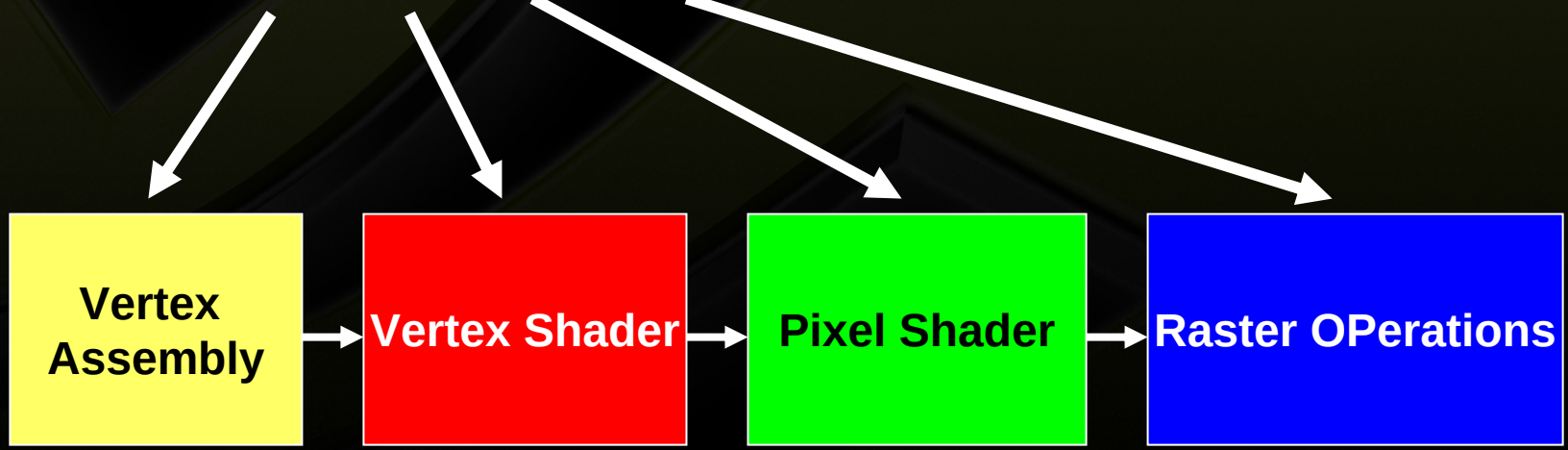
Performance Dashboard

- Speed control



The simplified graphics pipeline

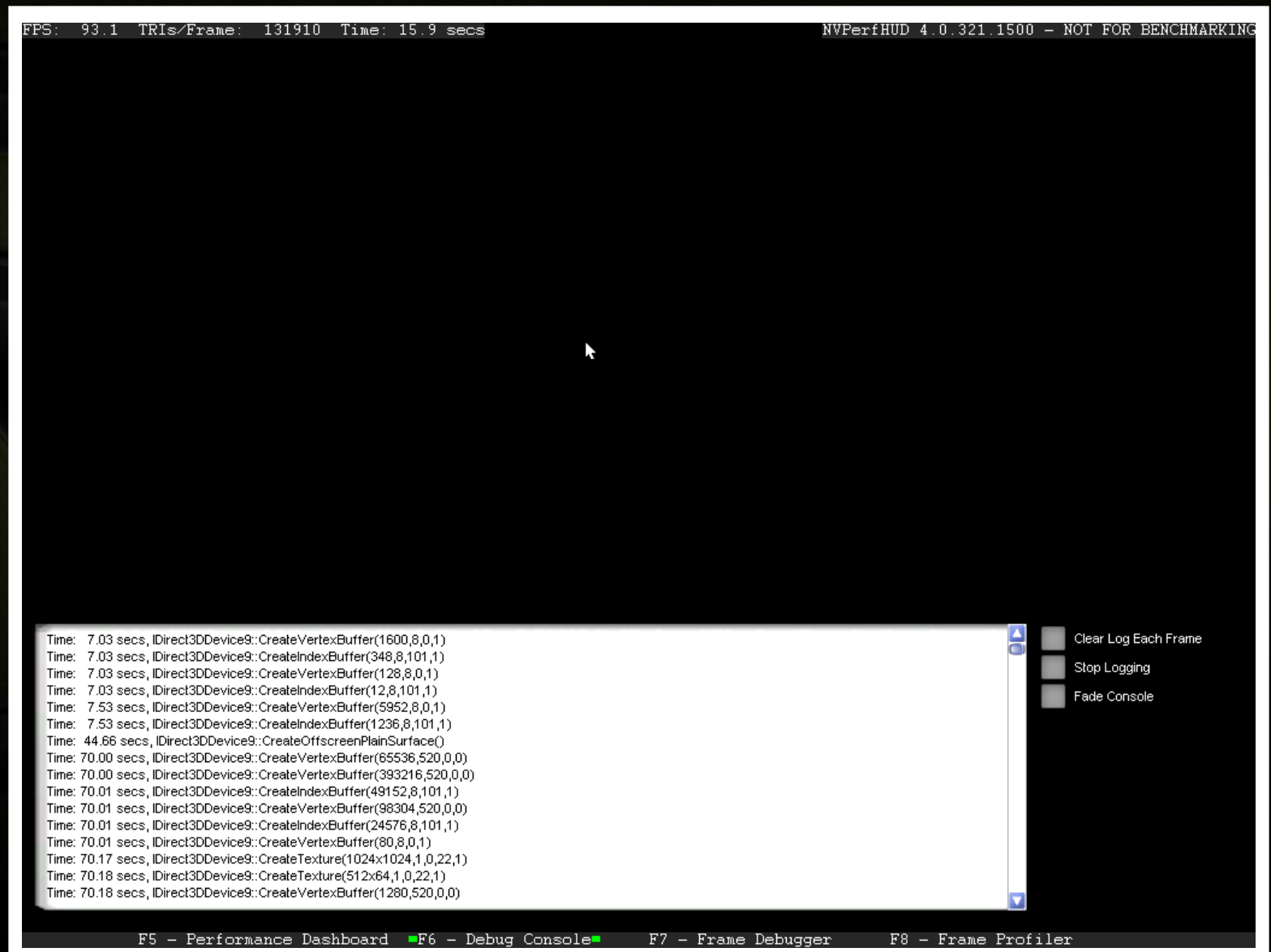




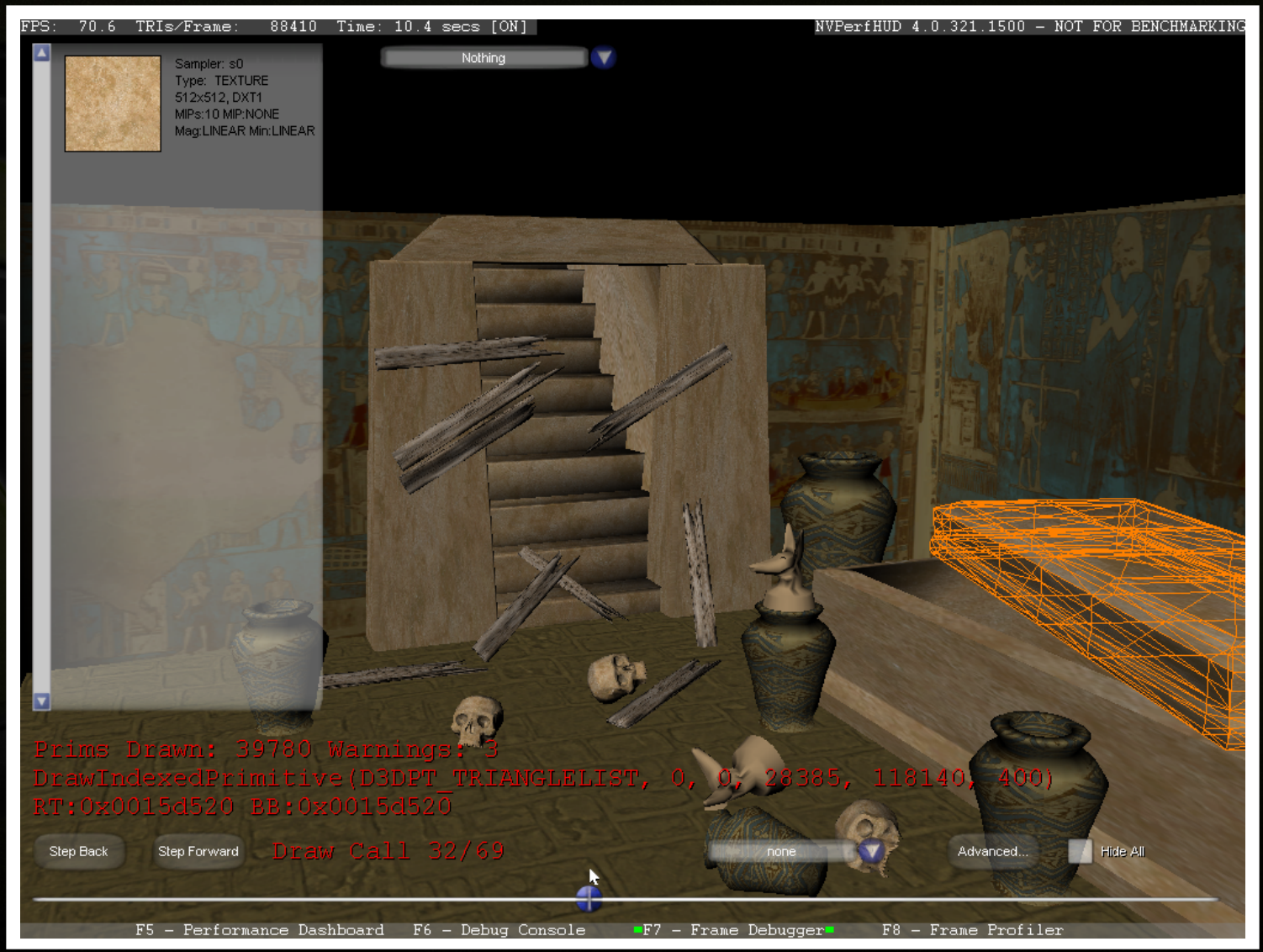
Performance Dashboard Demo

- Install
- Configure
- Drag & Drop

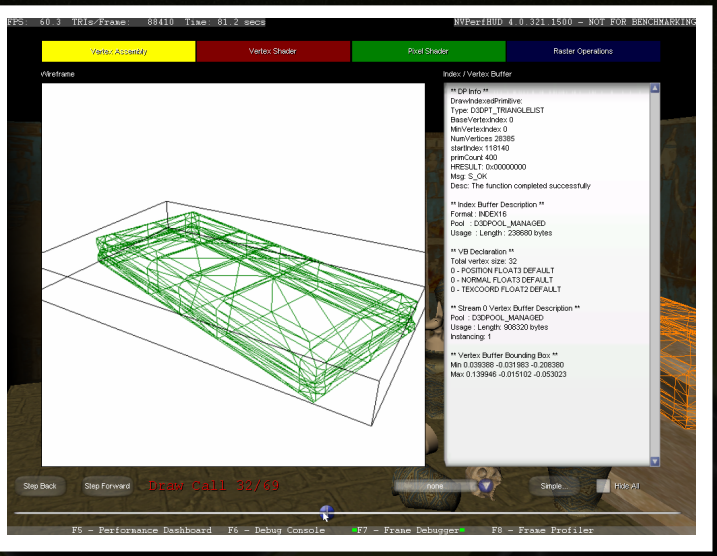
Debug Console

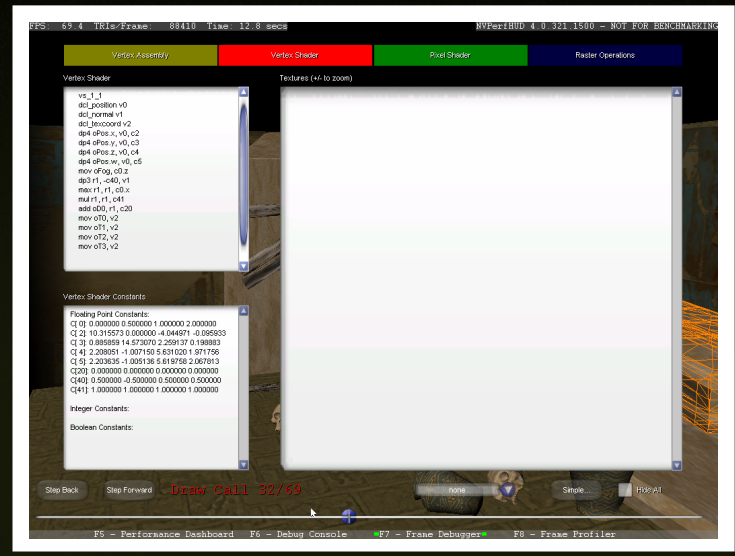


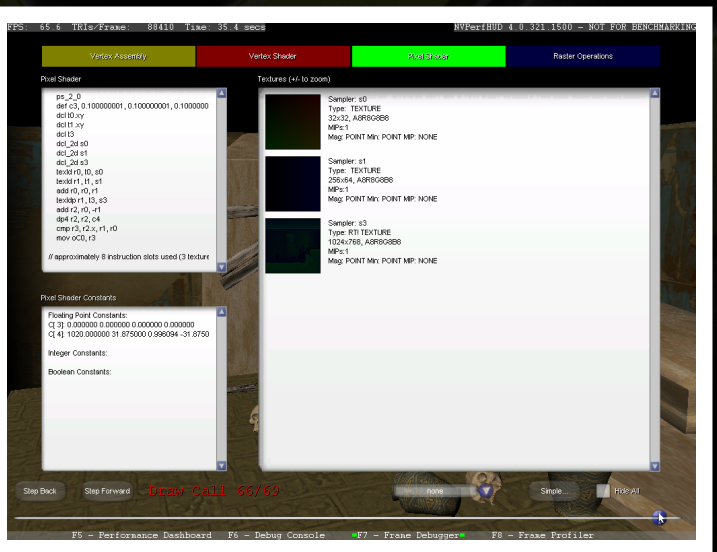
Frame Debugger

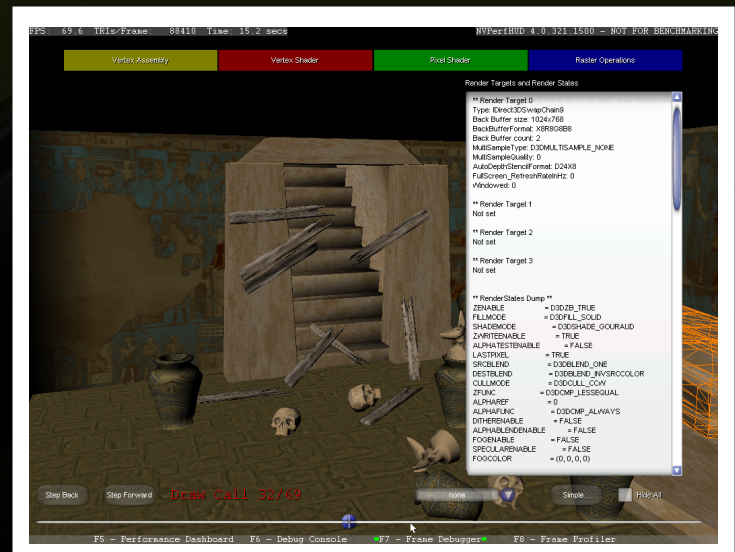


Frame Debugger, advanced view









Frame Profiler

- Measures performance counters
- strategy

Frame Profiler, measuring

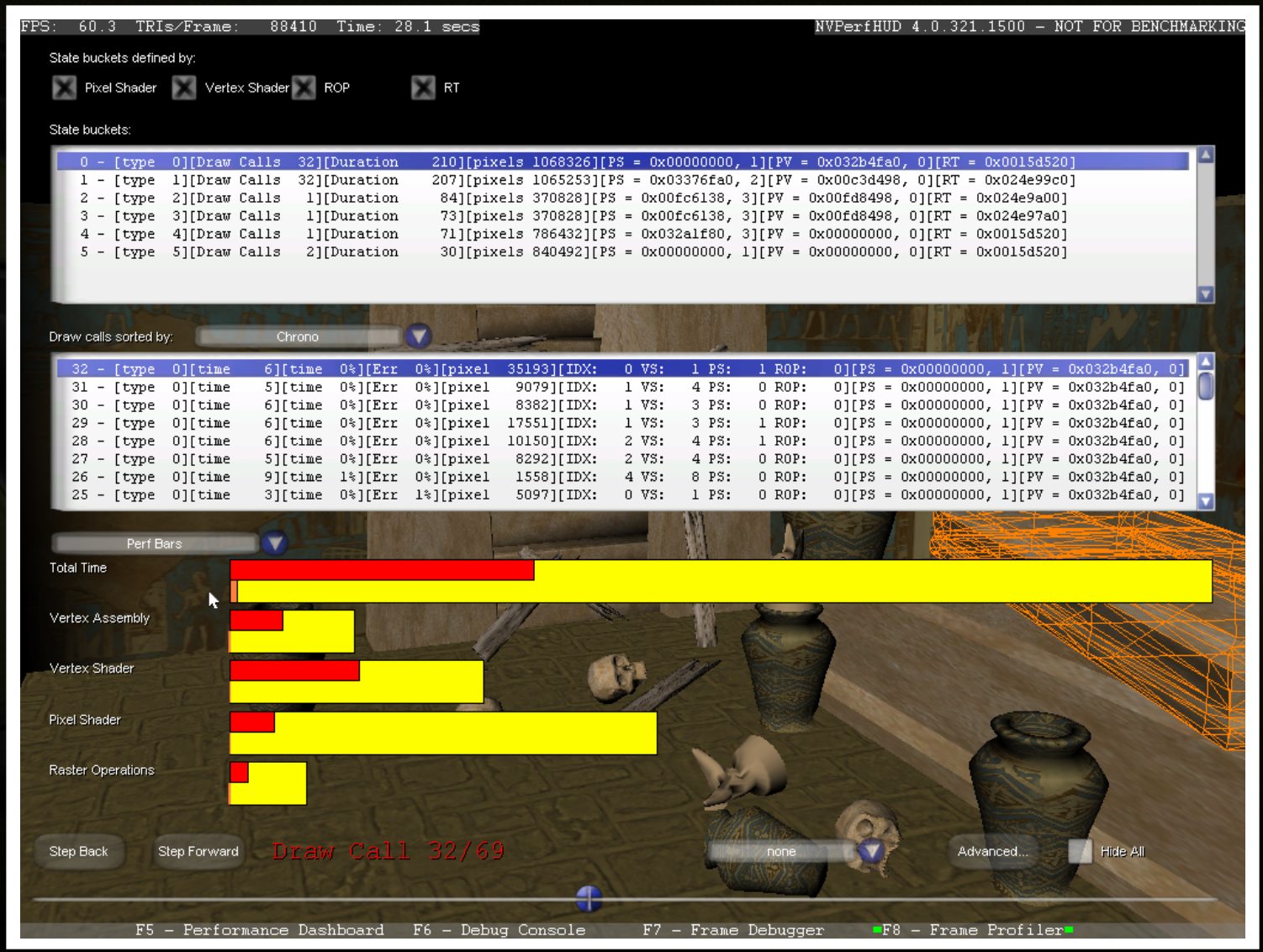
- NVPerfHUD uses NVPerfKit
  - uses ~40 Performance Counters (PC's)
- Can not read all of them at the same time
- Need to render THE SAME FRAME until all the PC's are read

Frame Profiler, strategy

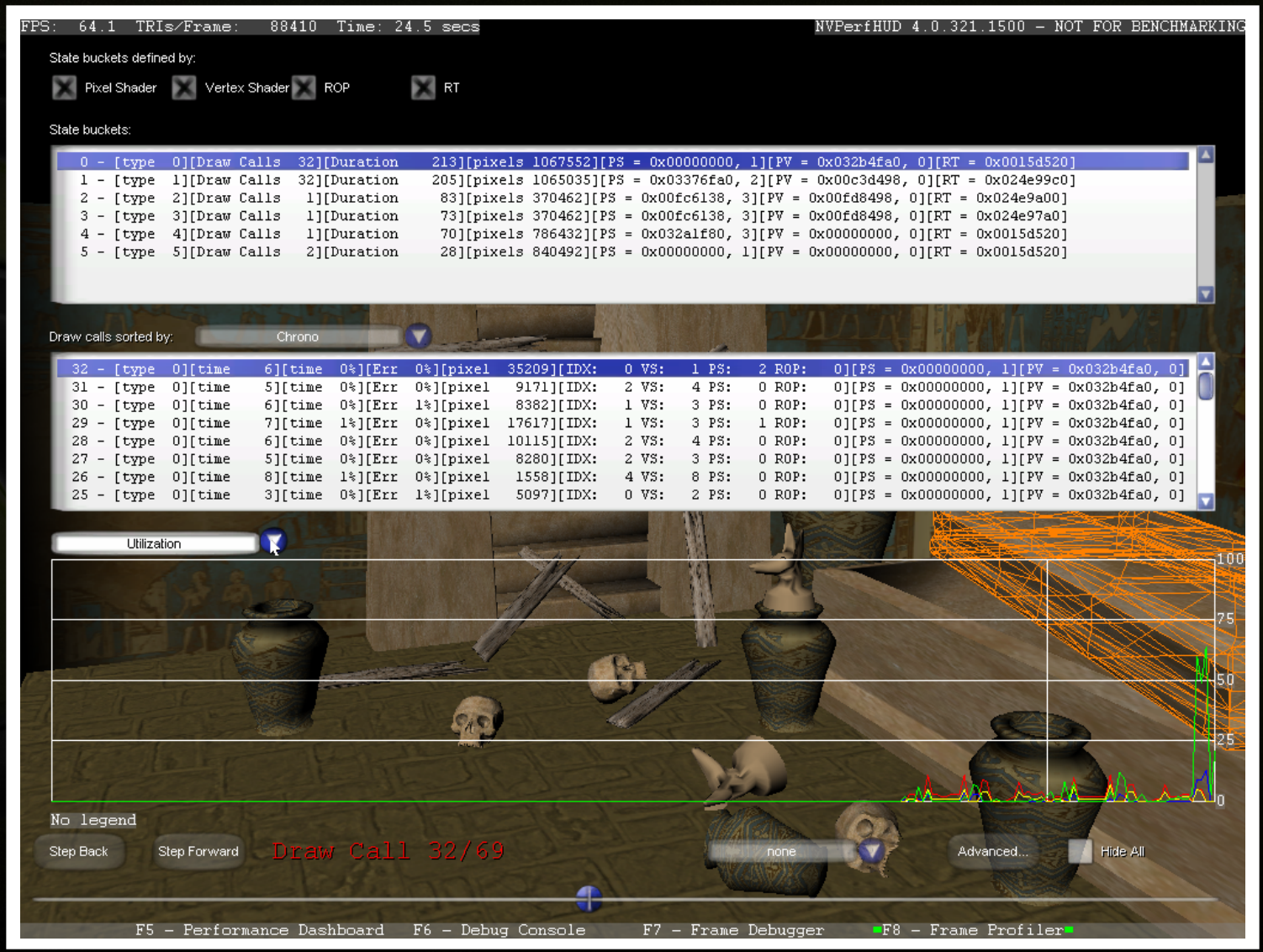
- Optimization Strategy:
  - Group by state is roughly grouping by bottleneck
  - These groups are called “state buckets”
- Procedure
  - Group draw calls by rendering state into state buckets
  - Identify the bottleneck of the most expensive state bucket
    - Solved by NVPerfHUD
  - Cure the bottleneck with a common corrective action

Frame Profiler Demo, measuring

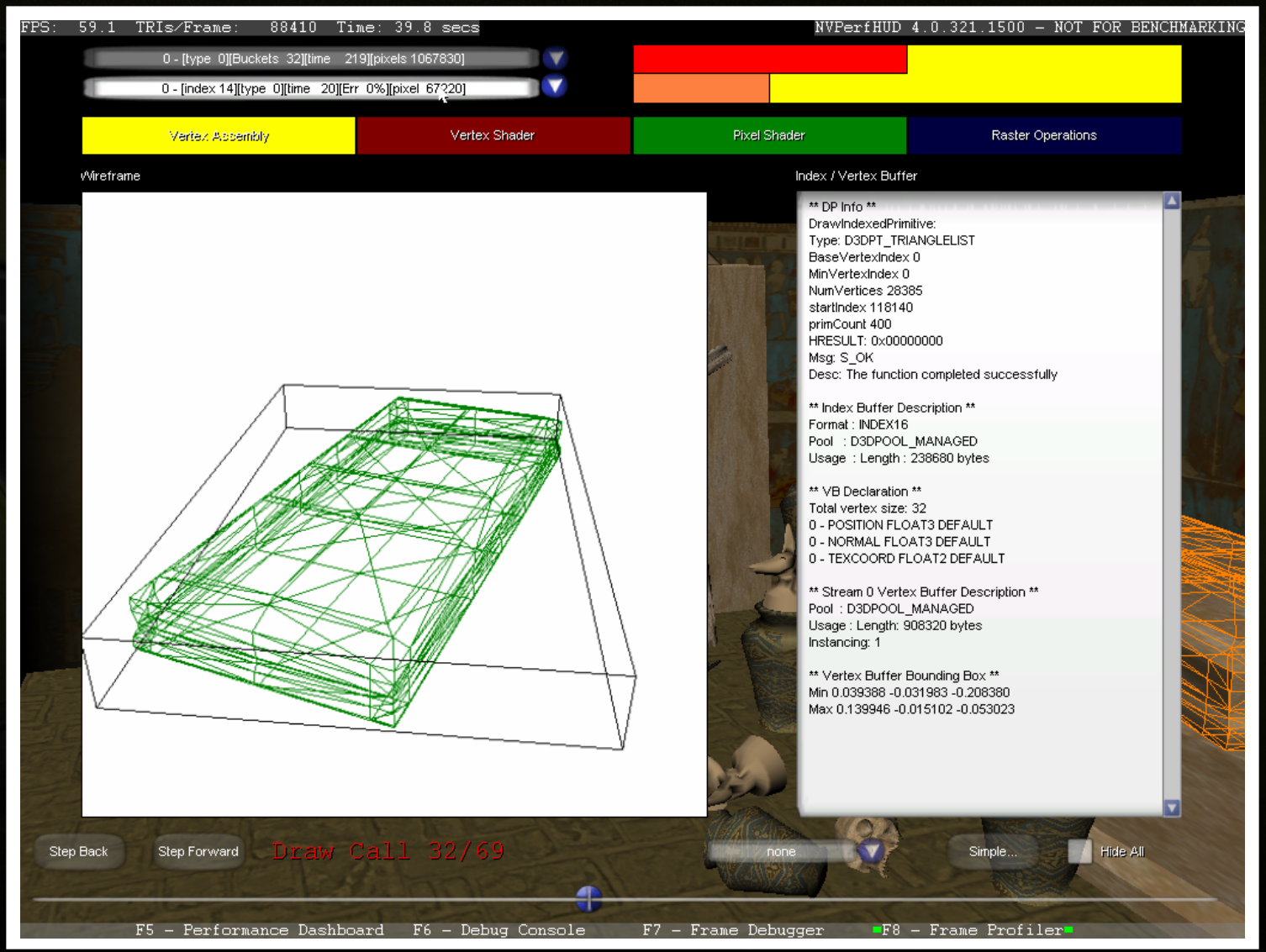
Frame Profiler Demo



Frame Profiler Demo



Frame Profiler Demo, advanced view



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

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

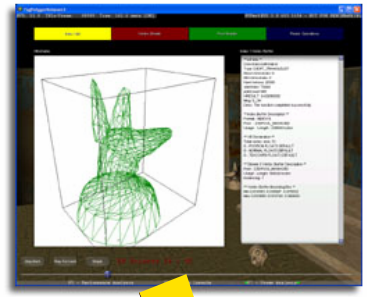
The way it's meant to be played

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 D3D application.

The latest release includes several new analysis modes:

- **Frame Analysis Mode**
  Freeze your application and single-step through the current frame's draw calls to see what is happening inside the GPU at each stage of your graphics pipeline, from the Index Unit, Vertex Shader, Pixel Shader & Render Operations units.
- **Debug Console Mode**
  This mode shows you DirectX Debug Runtime messages, and custom messages from your application.
- **Performance**
  The powerful frame analysis experiments from the previous release are still available in PerfHUD 3.

Now Available!

The opt-in mechanism has changed, so code alterations are necessary 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

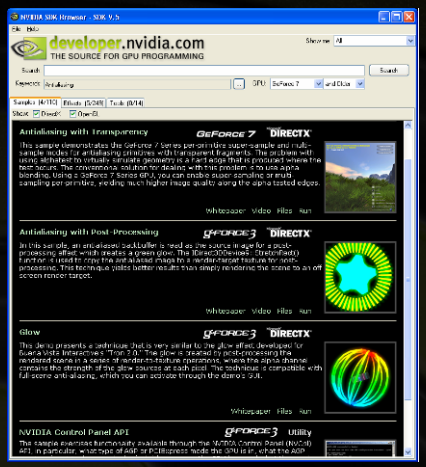
Questions?

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

NVGLExpert@nvidia.com
NVShaderPerf@nvidia.com
NVPerfKIT@nvidia.com
NVPerfHUD@nvidia.com
FXComposer@nvidia.com

NVIDIA SDK

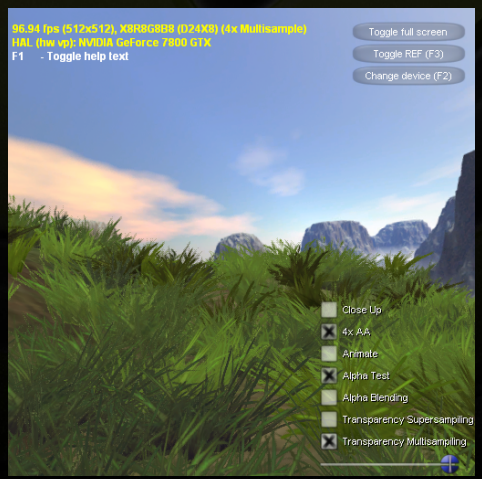
The Source for GPU Programming

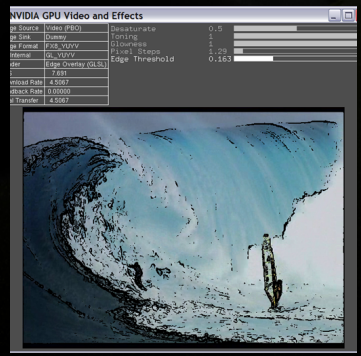


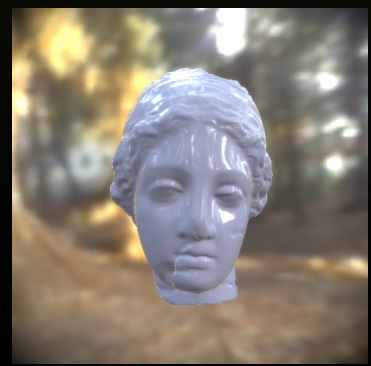
Hundreds of code samples and effects that help you take advantage of the latest in graphics technology.

gamedeveloper FRONTLINE AWARD '04

- 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!**

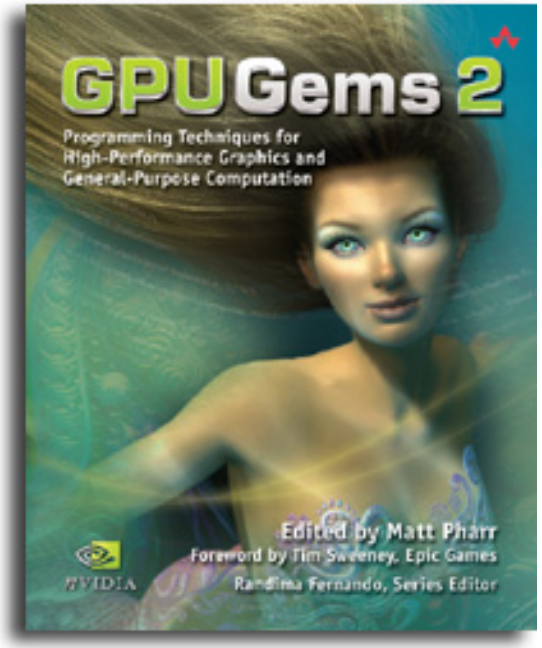






GPU Gems 2

Programming Techniques for High-Performance Graphics and General-Purpose Computation

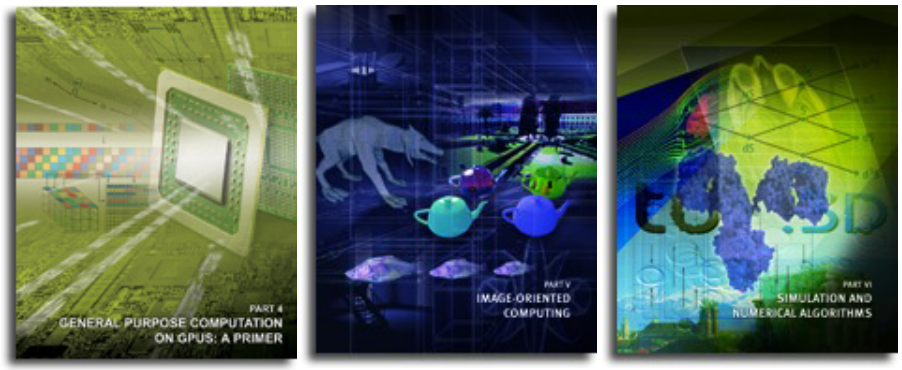


- **880 full-color pages**
- **330 figures**
- **Hard cover**
- **$59.99**
- **Experts from universities and industry**

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**