Performance Tools

Raul Aguaviva and Jeff Kiel (NVIDIA Corporation)

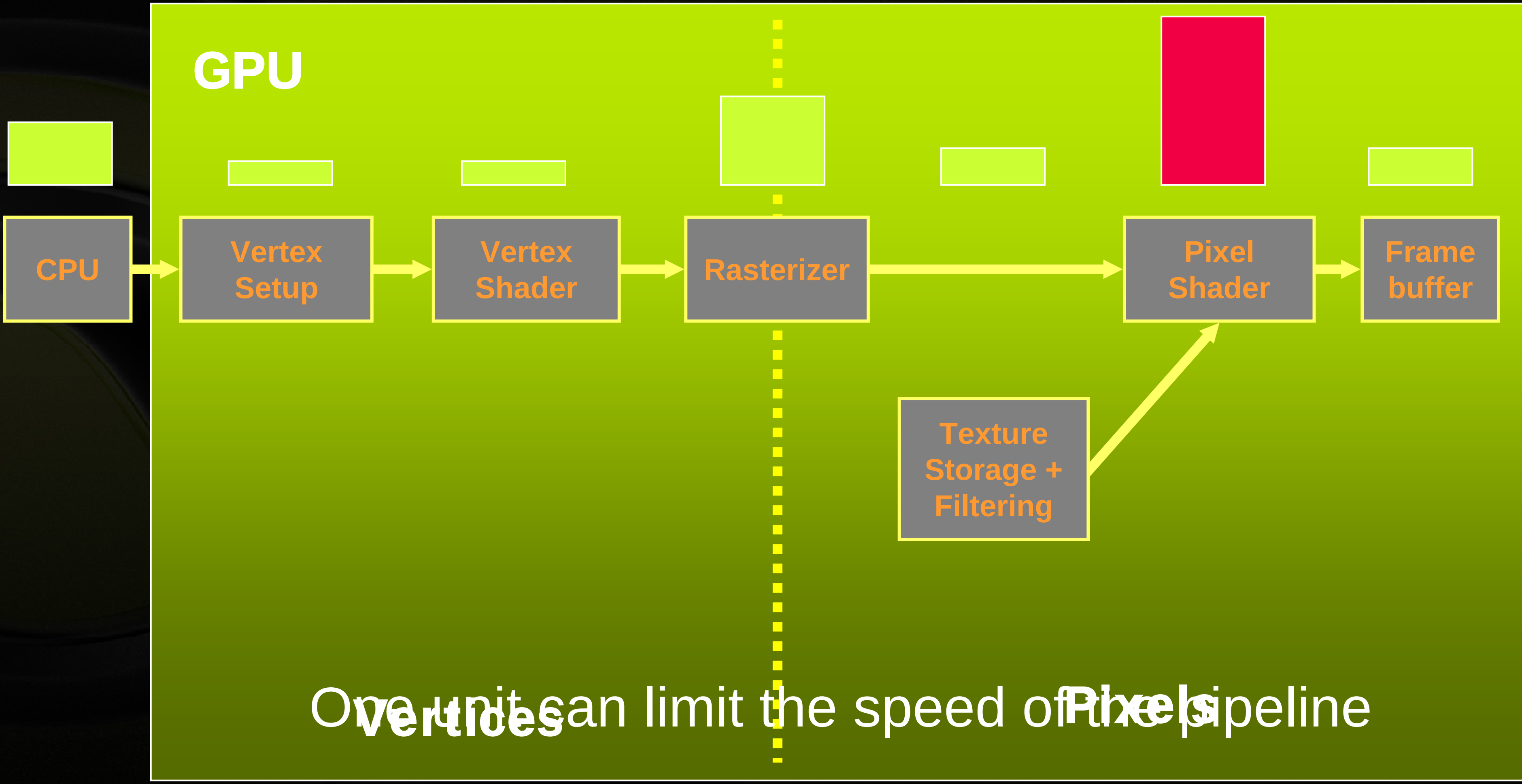
Performance Tools Agenda

- 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)



GPU

CPU

Vertex Setup

Vertex Shader

Rasterizer

Pixel Shader

Frame buffer

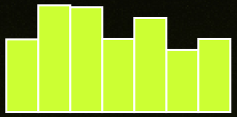
Texture Storage + Filtering

One unit can limit the speed of the pipeline

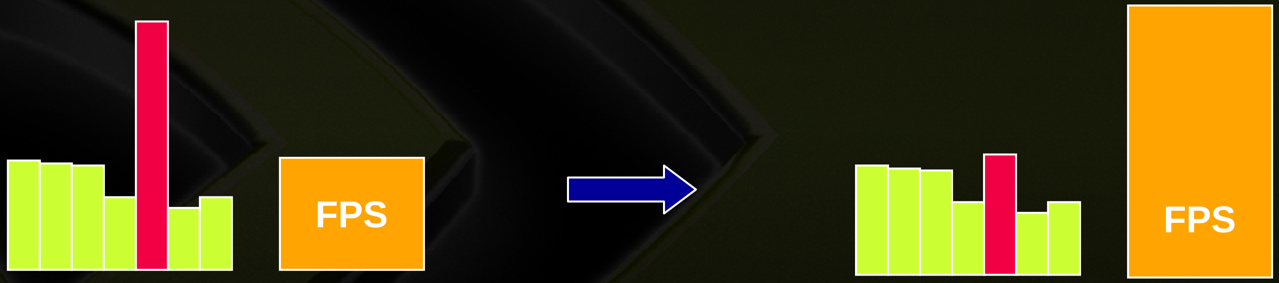
Vertices

Pixels

Bottleneck Identification

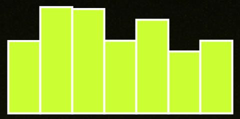


- Modify the stage itself
  - By decreasing its workload

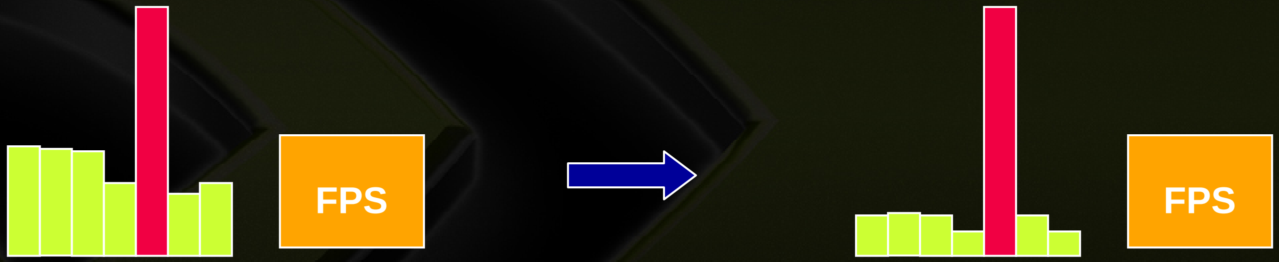


  - If performance/FPS improves greatly, then you know this is the bottleneck
  - Careful not to change the workload of other stages!

Bottleneck Identification

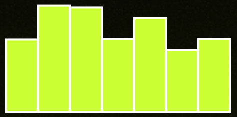


- Rule out the other stages
  - By giving all of them little or no work



- 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

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

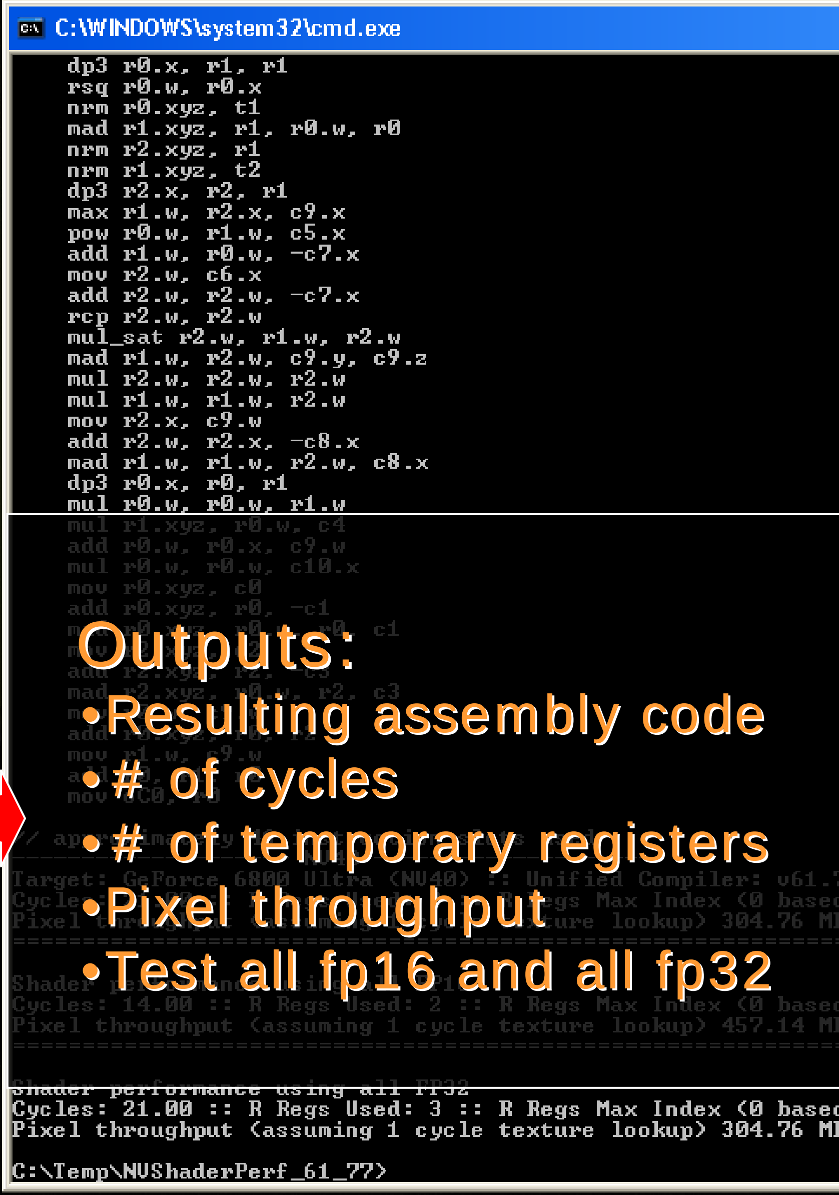
Inputs:

- •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



```
    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
    mul r0.w, r0.w, c10.x
    mov r0.xyz, c0
    add r0.xyz, r0, -c1
    m[illegible] r0.xyz, r0.w, r0, c1
    m[illegible]
    add r2.xyz, r2, -c3
    mad r2.xyz, r0.w, r2, c3
    m[illegible]
    add r0.xyz, r0, r2
    mov r1.w, c9.w
    a[illegible]
    mov oC0, r0
// ap[illegible]
Target: GeForce 6800 Ultra (NV40) :: Unified Compiler: v61.7
Cycles: [illegible] :: R Regs Max Index (0 based
Pixel throughput (assuming 1 cycle texture lookup) 304.76 MP
================================================================

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

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>
```

Outputs:

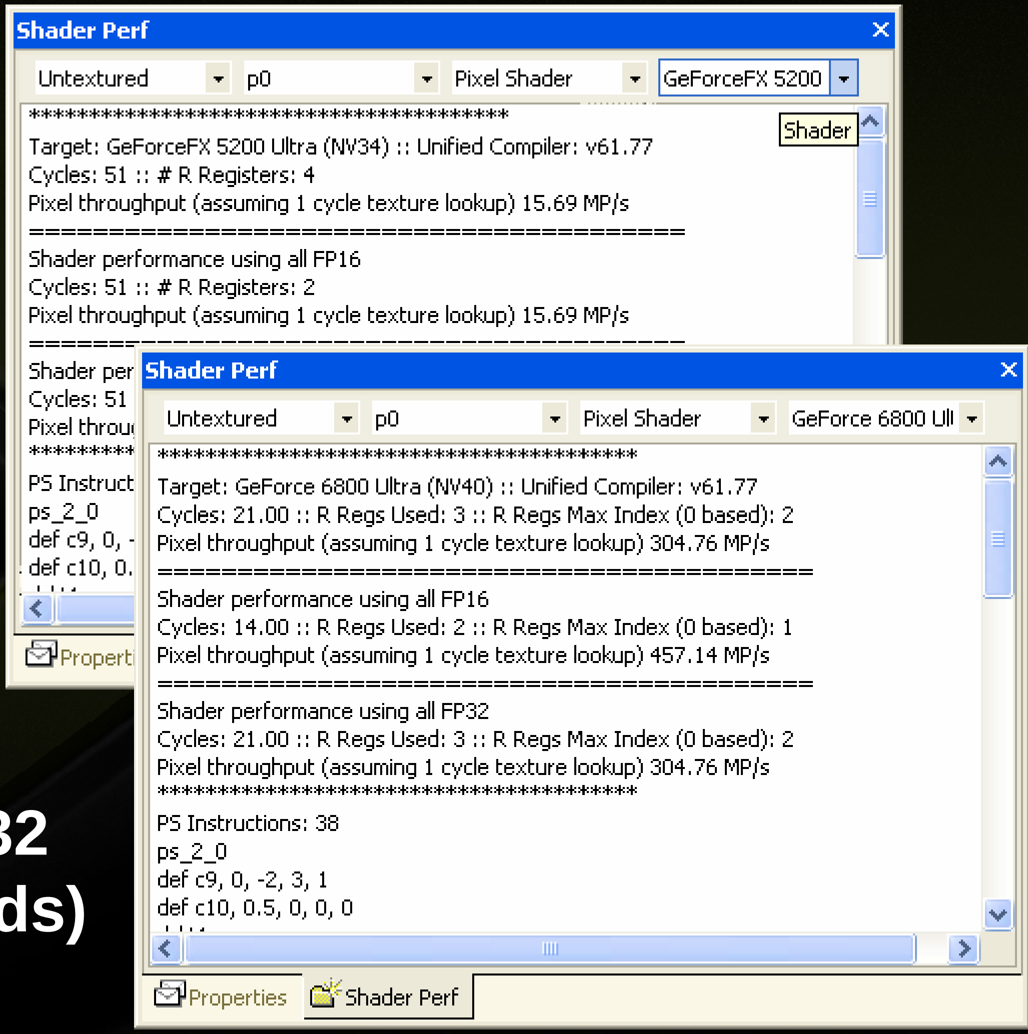
- •Resulting assembly code
- •# of cycles
- •# of temporary registers
- •Pixel throughput
- •Test all fp16 and all fp32

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)



Shader Perf

Untextured ▾ p0 ▾ Pixel Shader ▾ GeForceFX 5200 ▾

Shader

```
*******************************************
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.
```

Properti

Shader Perf

Untextured ▾ p0 ▾ Pixel Shader ▾ GeForce 6800 Ul ▾

```
*******************************************
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
```

Properties | Shader Perf

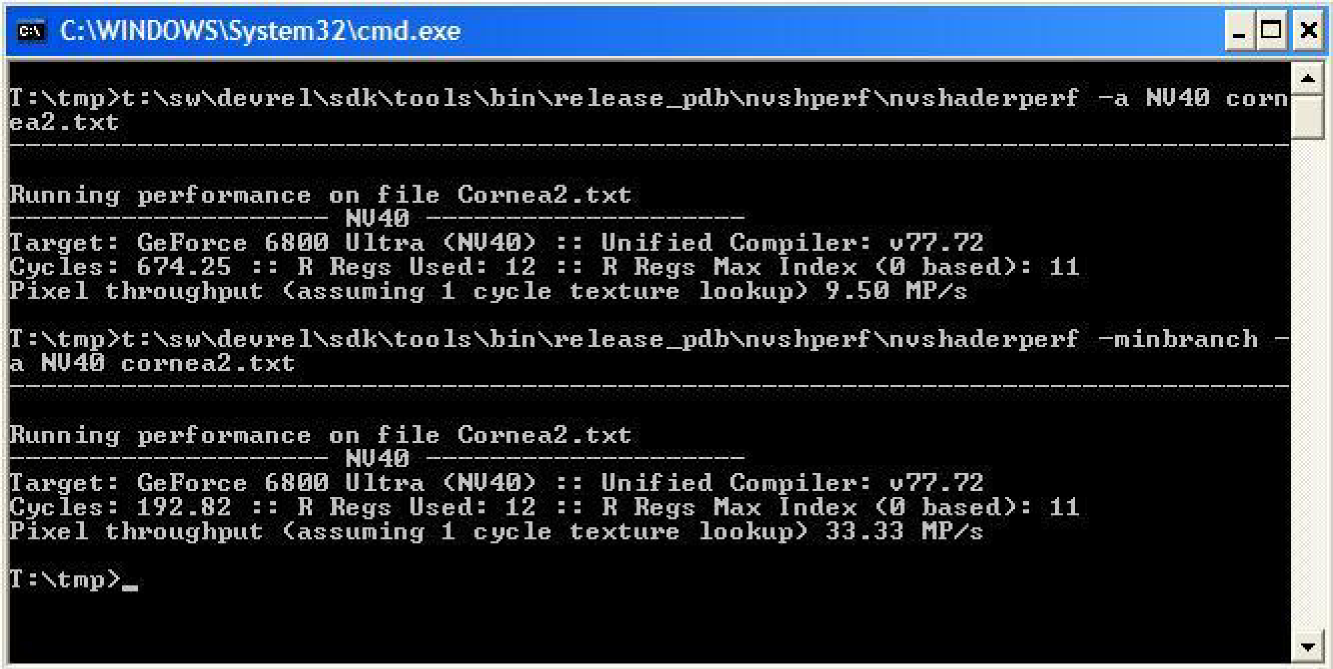
NVShaderPerf 1.8

- 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

Eye Shader from Luna
Maximum branch takes 674 cycles
Minimum branch takes 193 cycles.

```
////////////////////////////////////////////////////////////////////////////////
 // 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 );
```



```
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

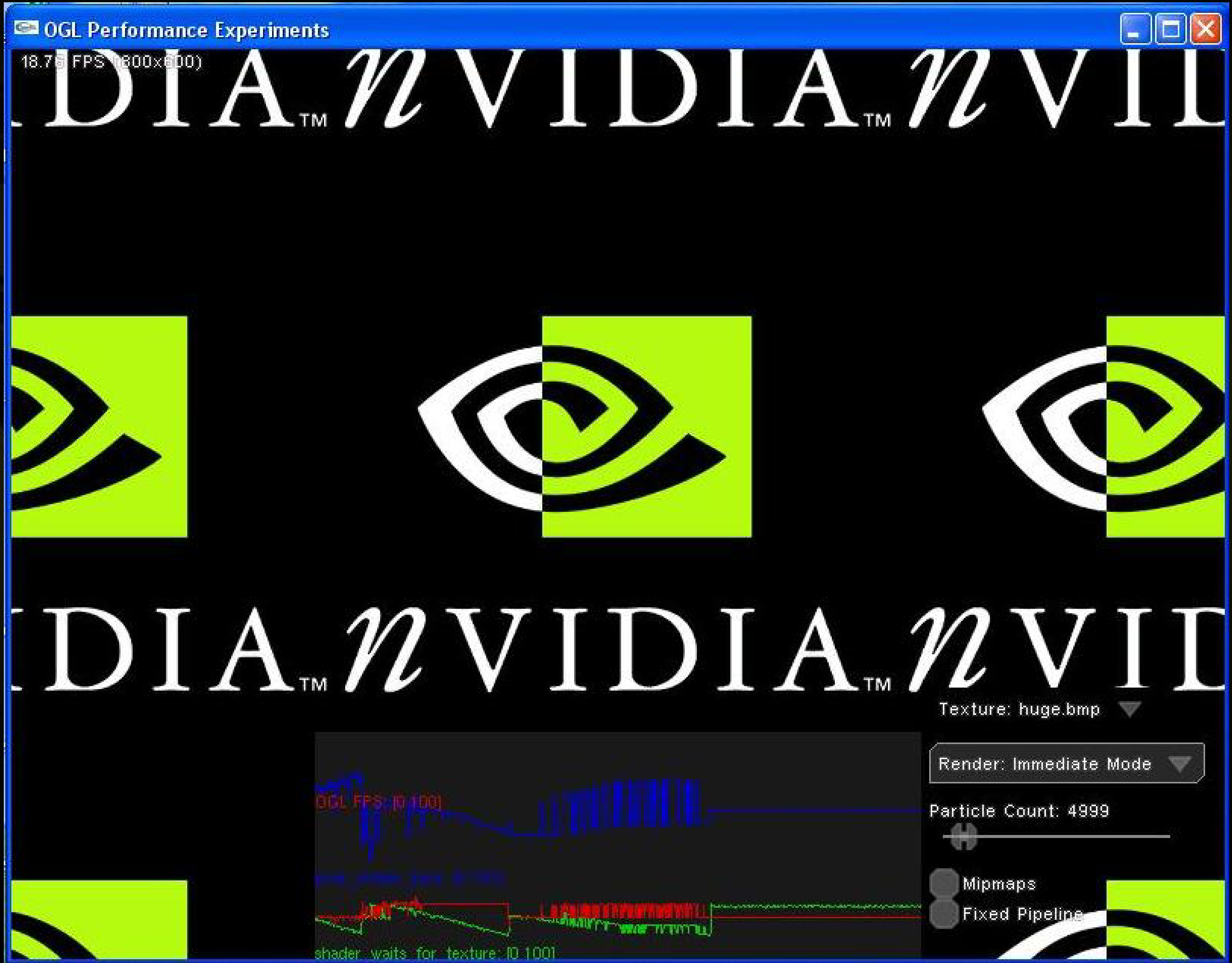
NVPerfKit: The Solution!

- 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...



OGL Performance Experiments

18.76 FPS (800x600)

Texture: huge.bmp

Render: Immediate Mode

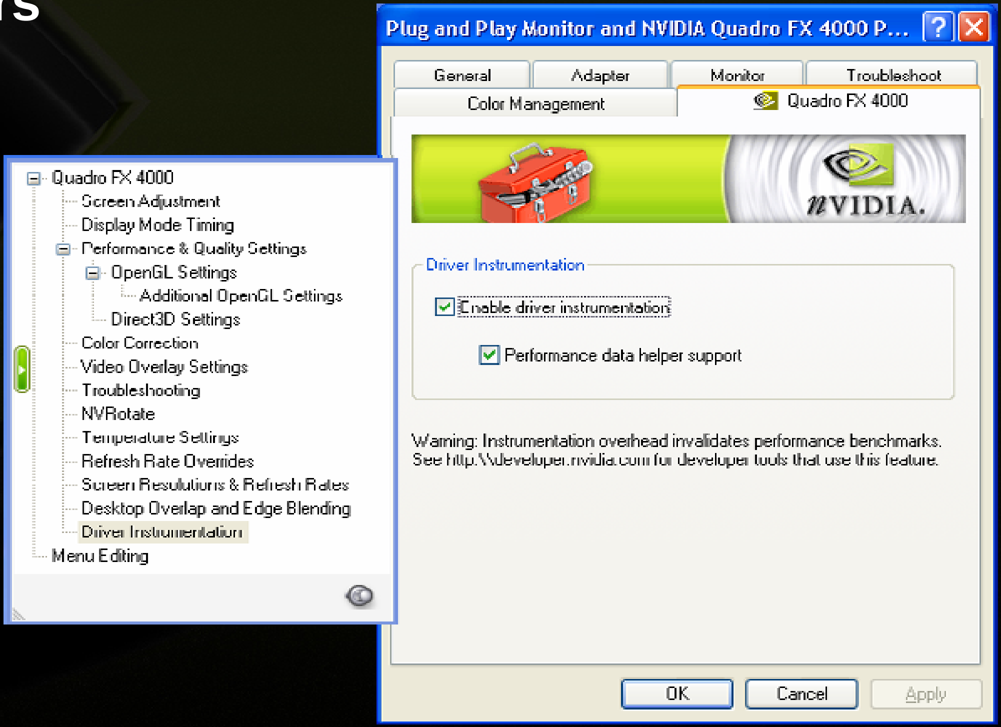
Particle Count: 4999

Mipmaps

Fixed Pipeline

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



Quadro FX 4000
- Screen Adjustment
- Display Mode Timing
- Performance & Quality Settings
  - OpenGL Settings
    - Additional OpenGL Settings
  - Direct3D Settings
- Color Correction
- Video Overlay Settings
- Troubleshooting
- NVRotate
- Temperature Settings
- Refresh Rate Overrides
- Screen Resolutions & Refresh Rates
- Desktop Overlap and Edge Blending
- Driver Instrumentation

Menu Editing

Plug and Play Monitor and NVIDIA Quadro FX 4000 P...

Driver Instrumentation

☑ Enable driver instrumentation

☑ Performance data helper support

Warning: Instrumentation overhead invalidates performance benchmarks.
See http:\\developer.nvidia.com for developer tools that use this feature.

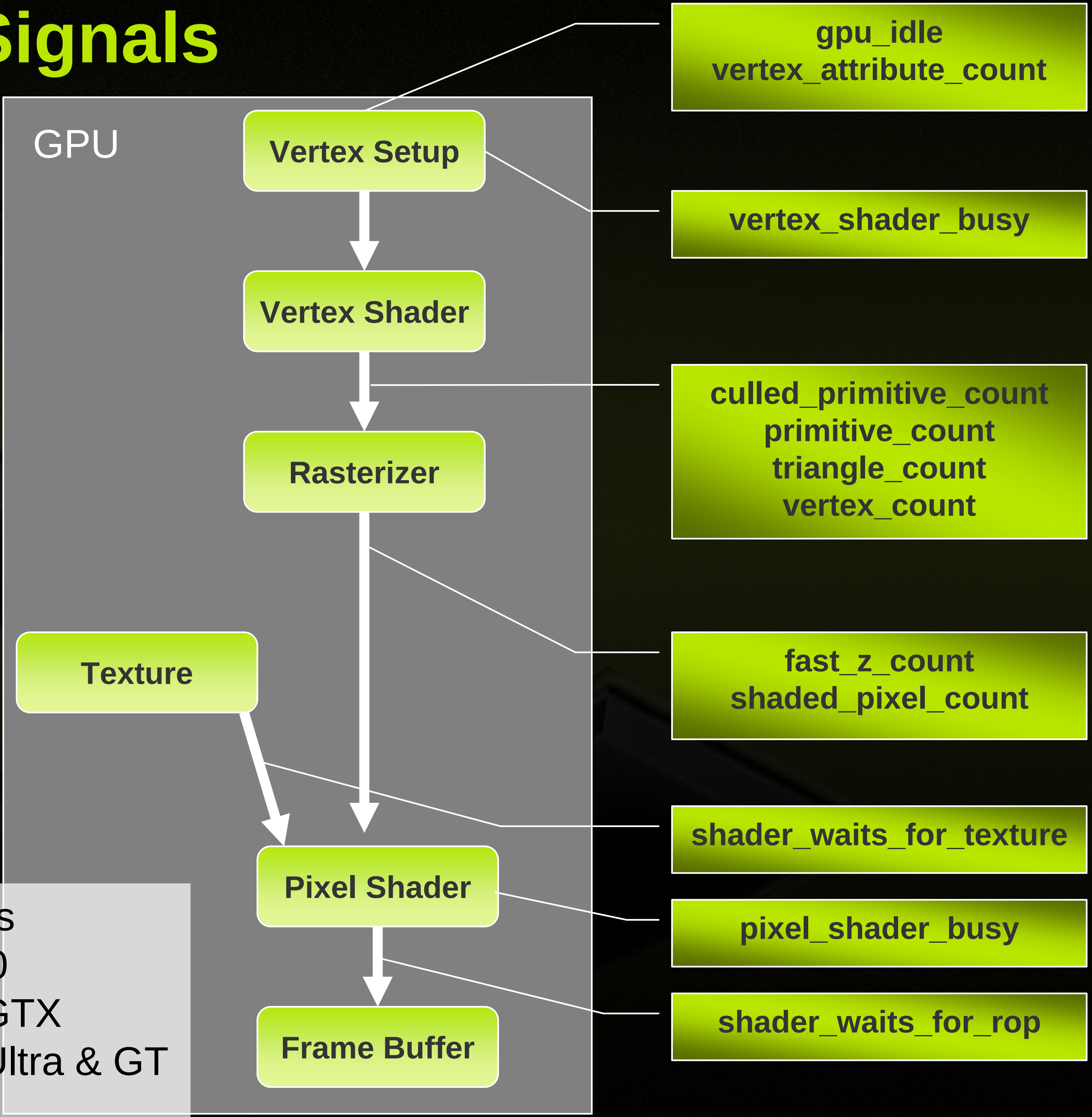
OpenGL Signals

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

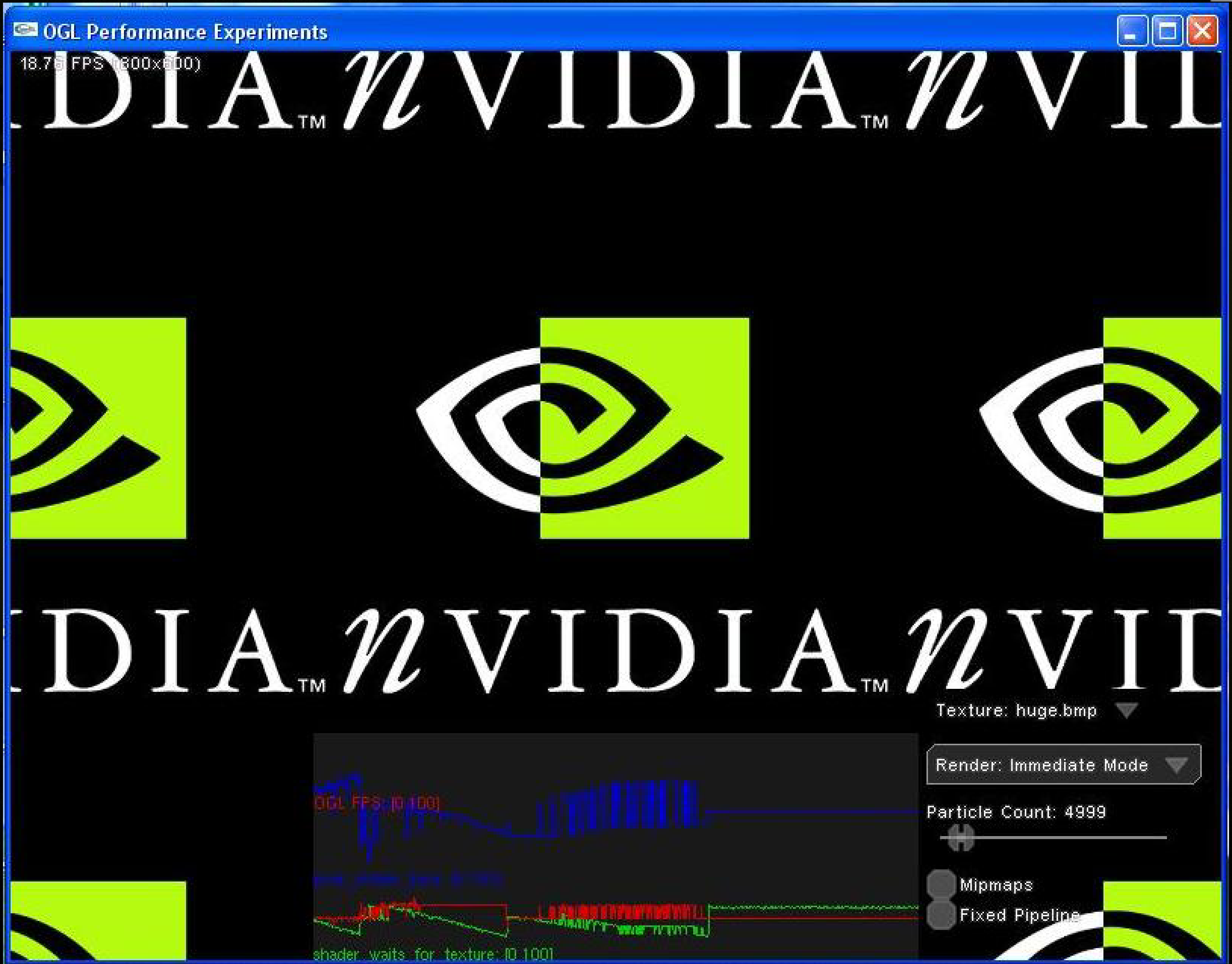
Counter Description	Official Name
FPS	D3D frame FPS
Frame Time (1/FPS)	D3D frame time mSec
AGP Memory Used	D3D frame agpmem MB
Video Memory Used	D3D frame vidmem MB
Driver Time	D3D frame mSec in driver
Driver Sleep Time (driver waits for GPU)	D3D frame mSec Sleeping
Triangle Count	D3D frame tris
Batch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



OGL Performance Experiments

18.76 FPS (800x600)

Texture: huge.bmp

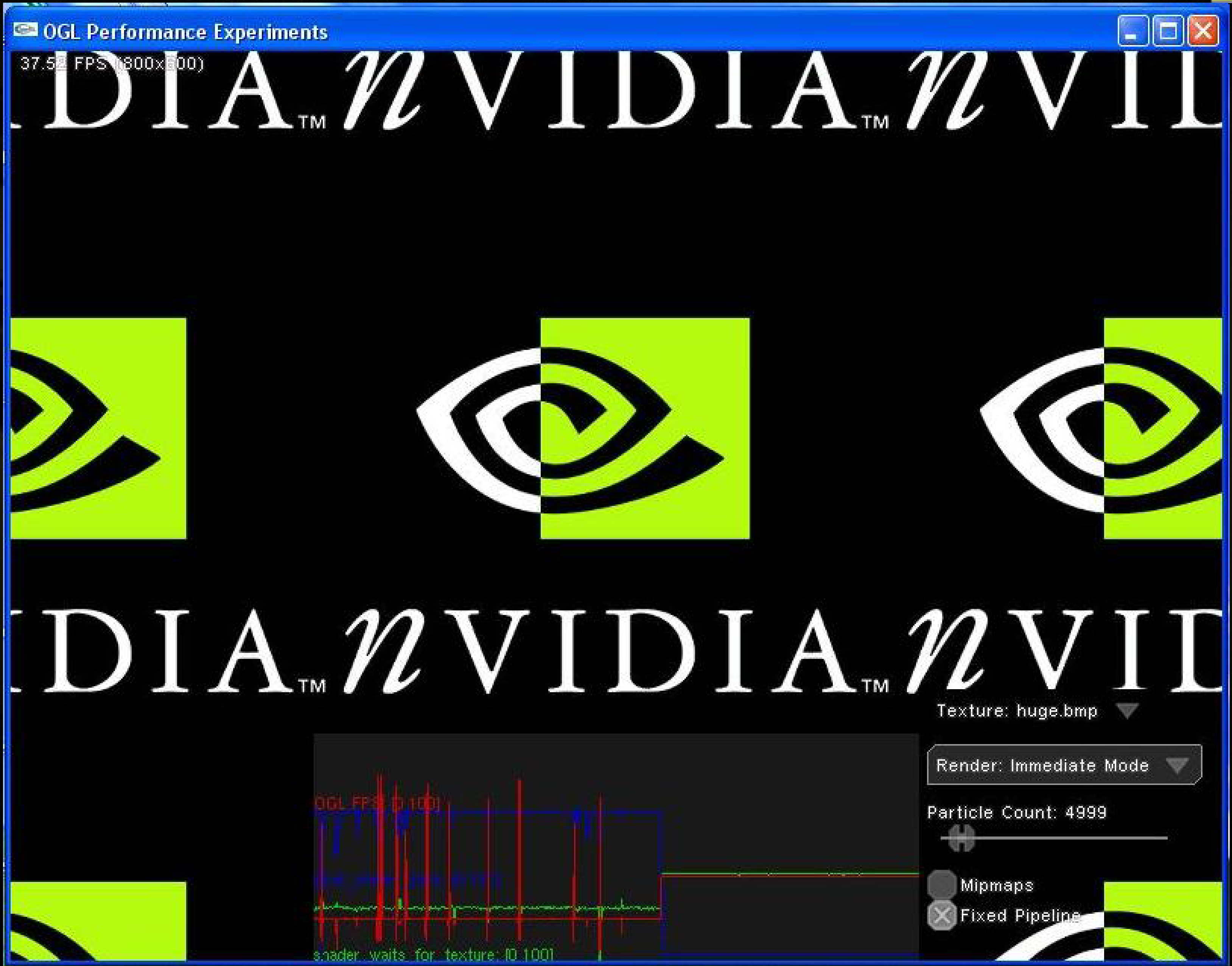
Render: Immediate Mode

Particle Count: 4999

○ Mipmaps

○ Fixed Pipeline

NVPerfKit Demo: Texture Bound



OGL Performance Experiments

37.52 FPS (800x600)

Texture: huge.bmp

Render: Immediate Mode

Particle Count: 4999

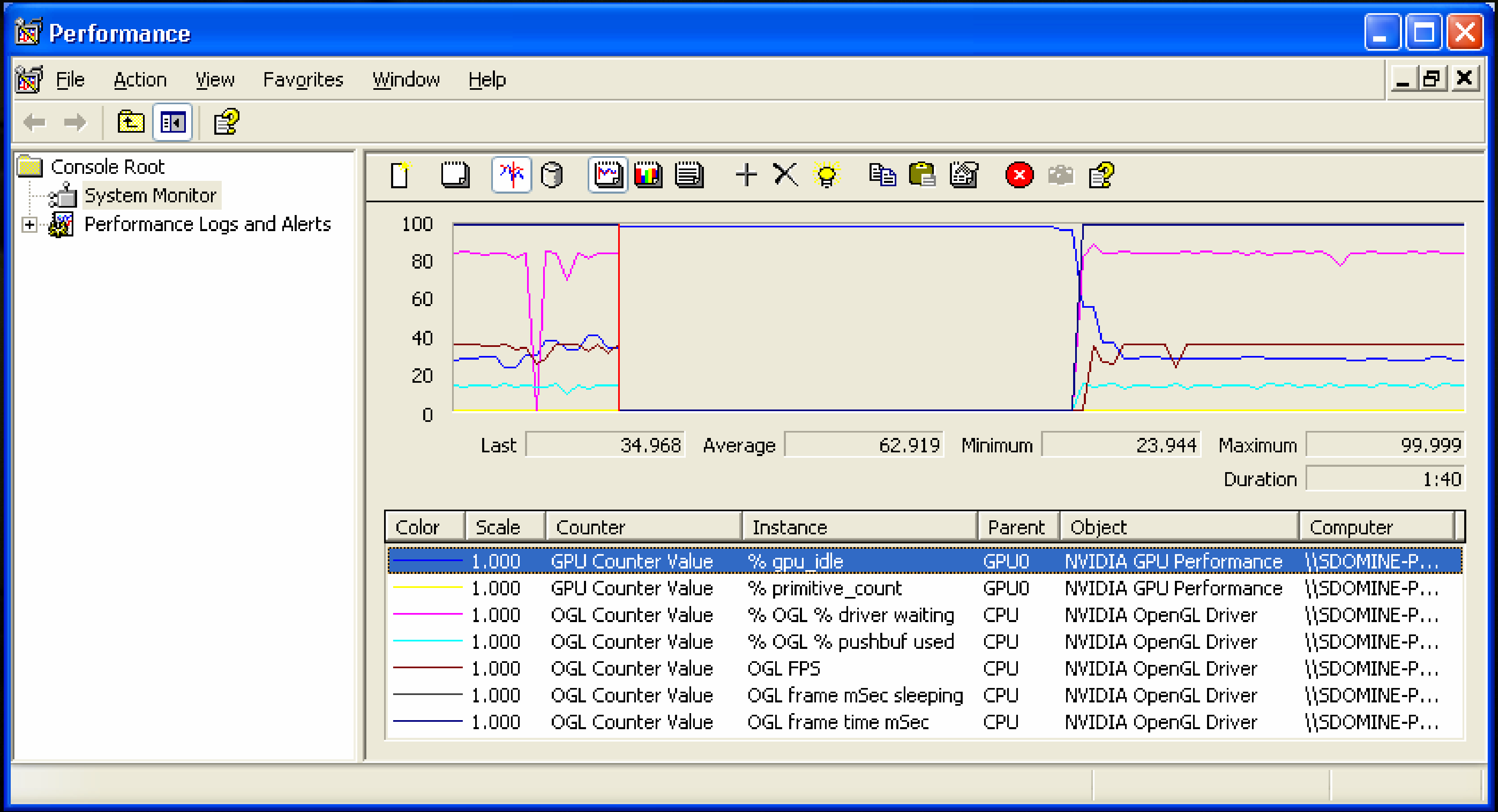
○ Mipmaps

☒ Fixed Pipeline

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

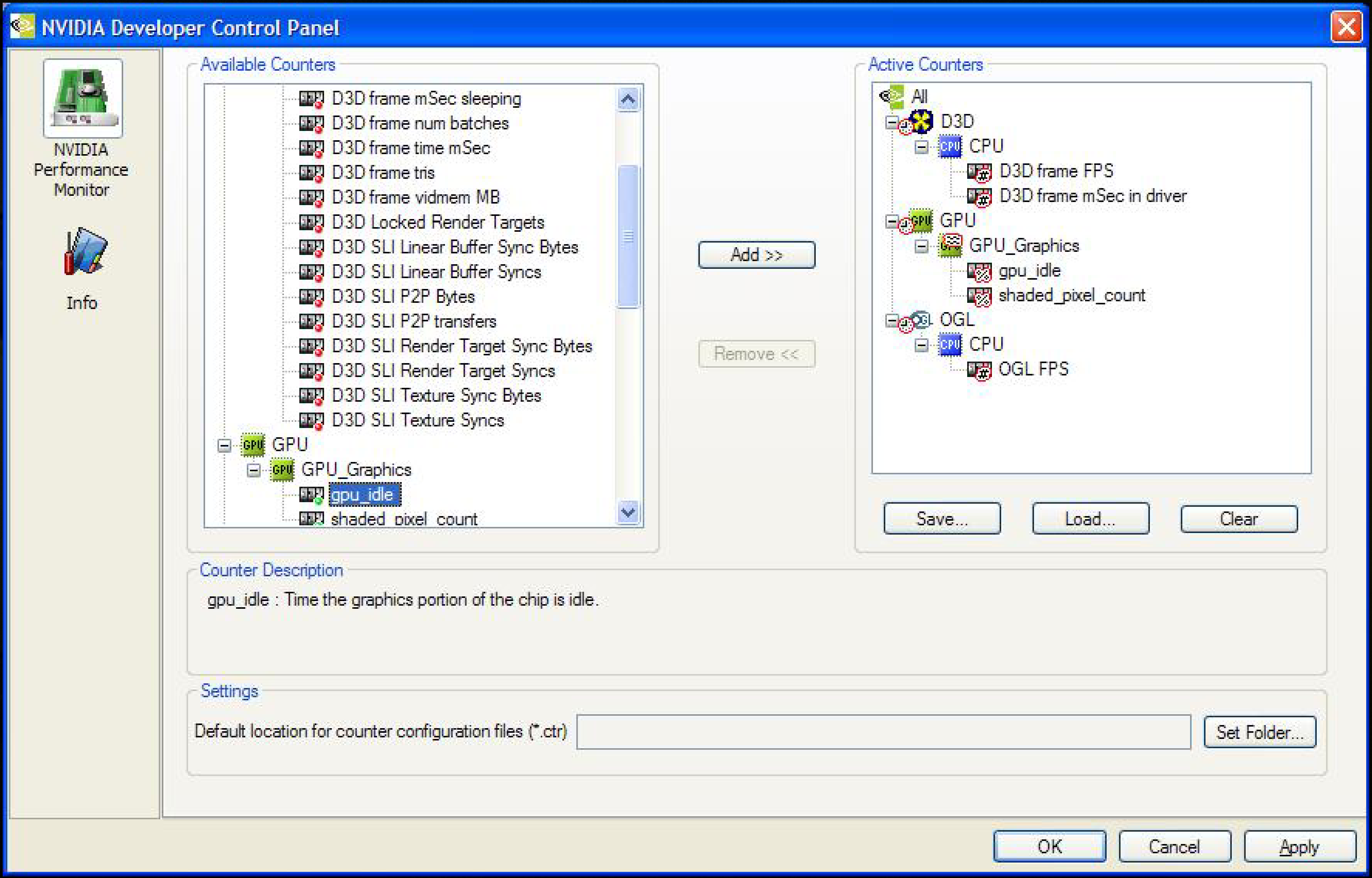


Last 34.968 Average 62.919 Minimum 23.944 Maximum 99.999

Duration 1:40

Color	Scale	Counter	Instance	Parent	Object	Computer
▬	1.000	GPU Counter Value	% gpu_idle	GPU0	NVIDIA GPU Performance	\\SDOMINE-P...
▬	1.000	GPU Counter Value	% primitive_count	GPU0	NVIDIA GPU Performance	\\SDOMINE-P...
▬	1.000	OGL Counter Value	% OGL % driver waiting	CPU	NVIDIA OpenGL Driver	\\SDOMINE-P...
▬	1.000	OGL Counter Value	% OGL % pushbuf used	CPU	NVIDIA OpenGL Driver	\\SDOMINE-P...
▬	1.000	OGL Counter Value	OGL FPS	CPU	NVIDIA OpenGL Driver	\\SDOMINE-P...
▬	1.000	OGL Counter Value	OGL frame mSec sleeping	CPU	NVIDIA OpenGL Driver	\\SDOMINE-P...
▬	1.000	OGL Counter Value	OGL frame time mSec	CPU	NVIDIA OpenGL Driver	\\SDOMINE-P...

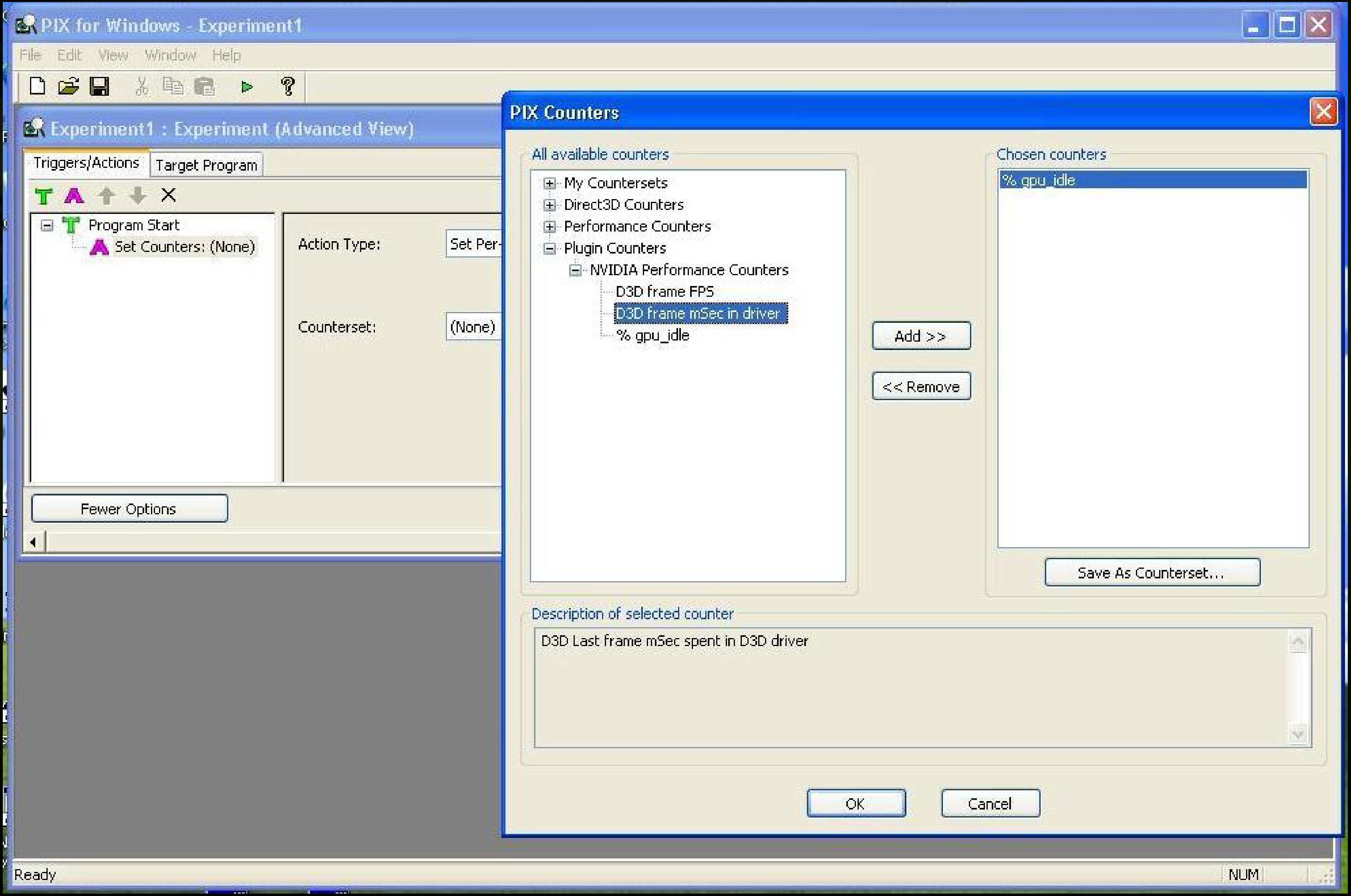
Associated Tools: NVDevCPL



gpu_idle : Time the graphics portion of the chip is idle.

Default location for counter configuration files (*.ctr)

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



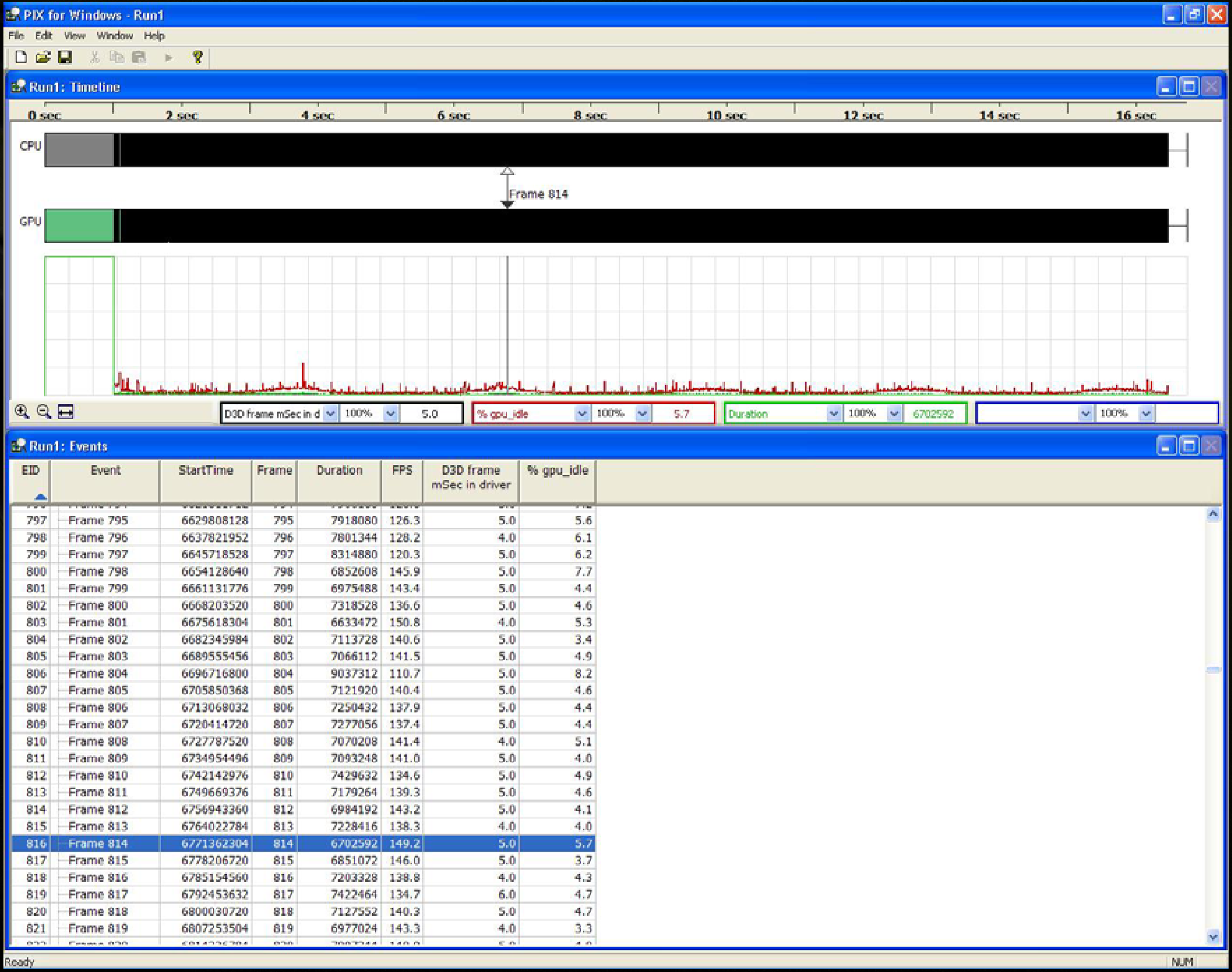
Action Type: Set Per-

Counterset: (None)

Chosen counters: % gpu_idle

Description of selected counter: D3D Last frame mSec spent in D3D driver

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



D3D frame mSec in d | 100% | 5.0

% gpu_idle | 100% | 5.7

Duration | 100% | 6702592

100%

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

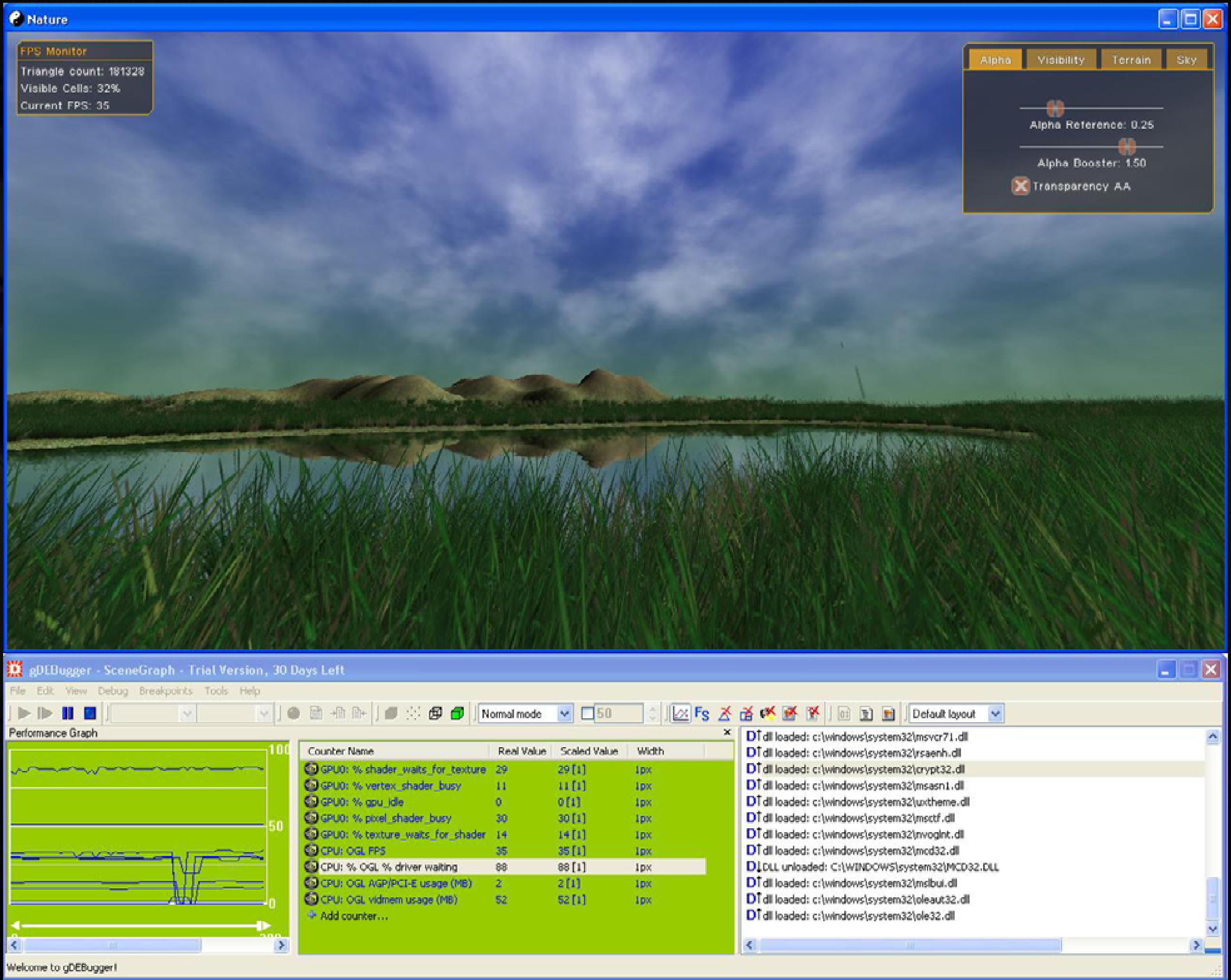
Helper Classes and Code Samples

- 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



FPS Monitor

Triangle count: 181328

Visible Cells: 32%

Current FPS: 35

Alpha Reference: 0.25

Alpha Booster: 1.50

☒ Transparency AA

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

```
dll loaded: c:\windows\system32\msvcr71.dll
dll loaded: c:\windows\system32\rsaenh.dll
dll loaded: c:\windows\system32\crypt32.dll
dll loaded: c:\windows\system32\msasn1.dll
dll loaded: c:\windows\system32\uxtheme.dll
dll loaded: c:\windows\system32\msctf.dll
dll loaded: c:\windows\system32\nvoglnt.dll
dll loaded: c:\windows\system32\mcd32.dll
DLL unloaded: C:\WINDOWS\system32\MCD32.DLL
dll loaded: c:\windows\system32\msctfime.ime
dll loaded: c:\windows\system32\oleaut32.dll
dll loaded: c:\windows\system32\ole32.dll
```

Welcome to gDEBugger!

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);
```

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

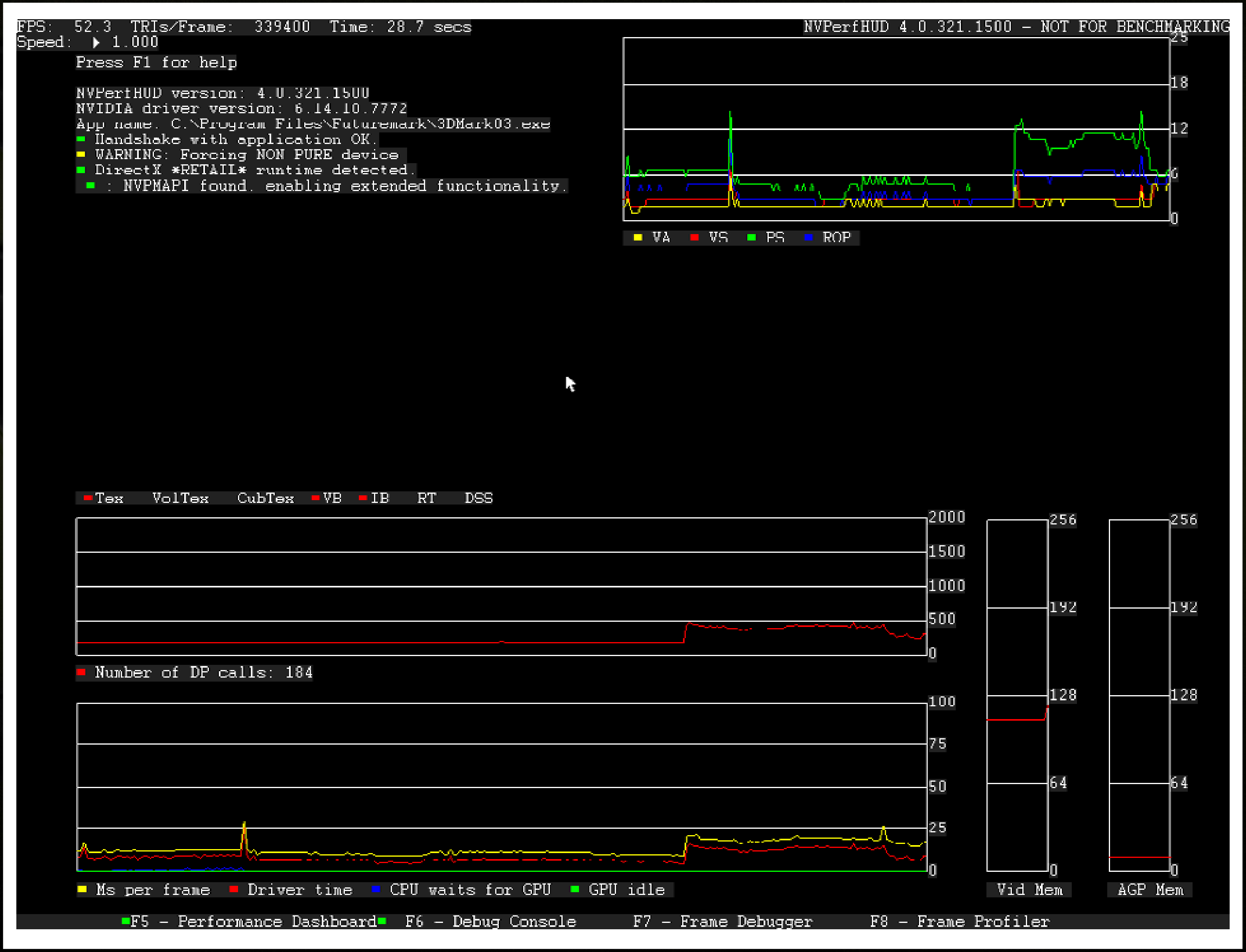
What is NVPerfHUD?

- 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

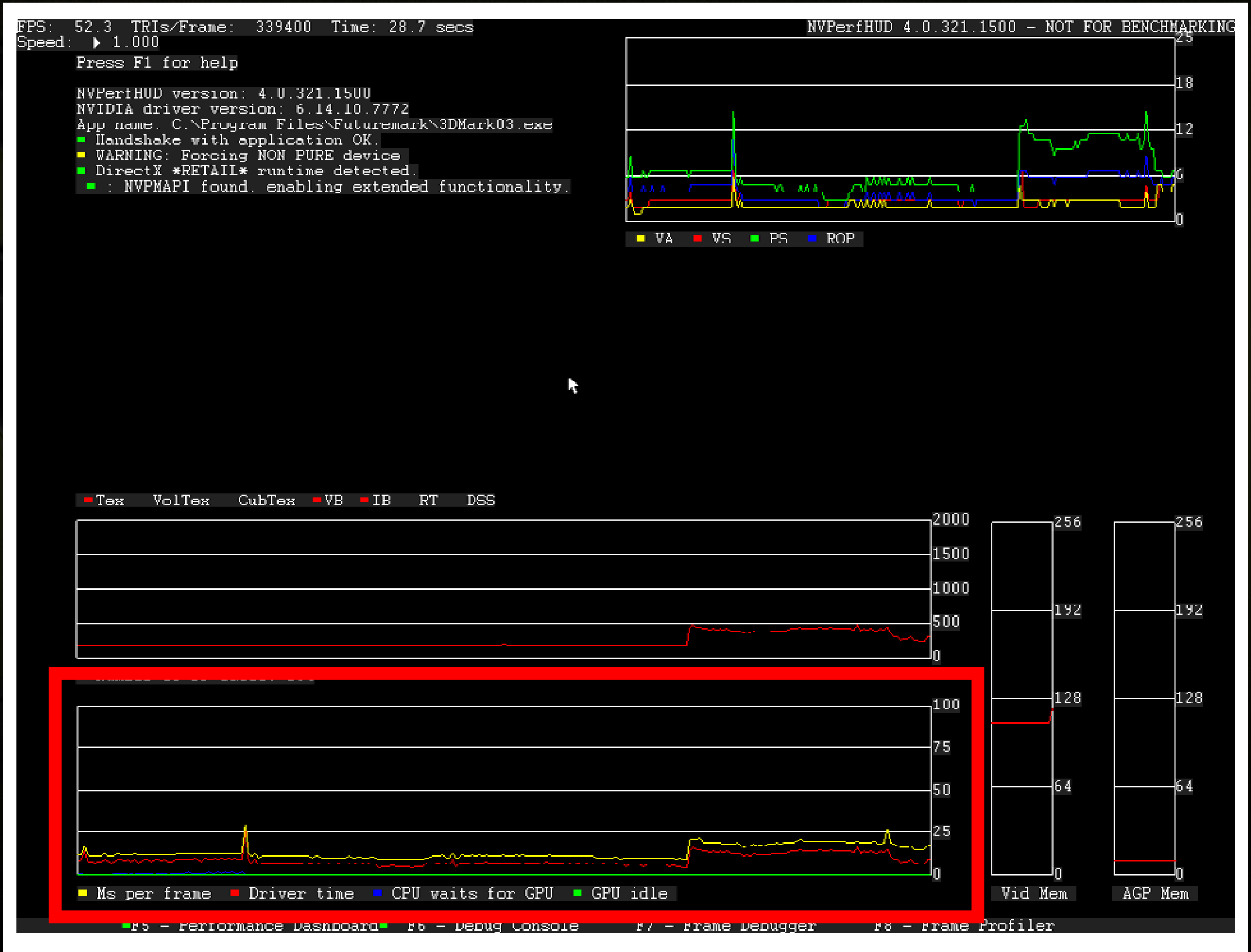
Performance Dashboard



```
FPS:  52.3  TRIs/Frame:  339400  Time: 28.7 secs
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\3DMark03.exe
        ■ Handshake with application OK.
        ■ WARNING: Forcing NON PURE device
        ■ DirectX *RETAIL* runtime detected.
         ■ : NVPMAPI found. enabling extended functionality.
```

NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING
25
18
12
6
0
■ VA ■ VS ■ PS ■ ROP
■ Tex VolTex CubTex ■ VB ■ IB RT DSS
2000
1500
1000
500
0
■ Number of DP calls: 184
100
75
50
25
0
■ Ms per frame ■ Driver time ■ CPU waits for GPU ■ GPU idle
256
192
128
64
0
Vid Mem
256
192
128
64
0
AGP Mem
■F5 - Performance Dashboard■ F6 - Debug Console F7 - Frame Debugger F8 - Frame Profiler

Performance Dashboard

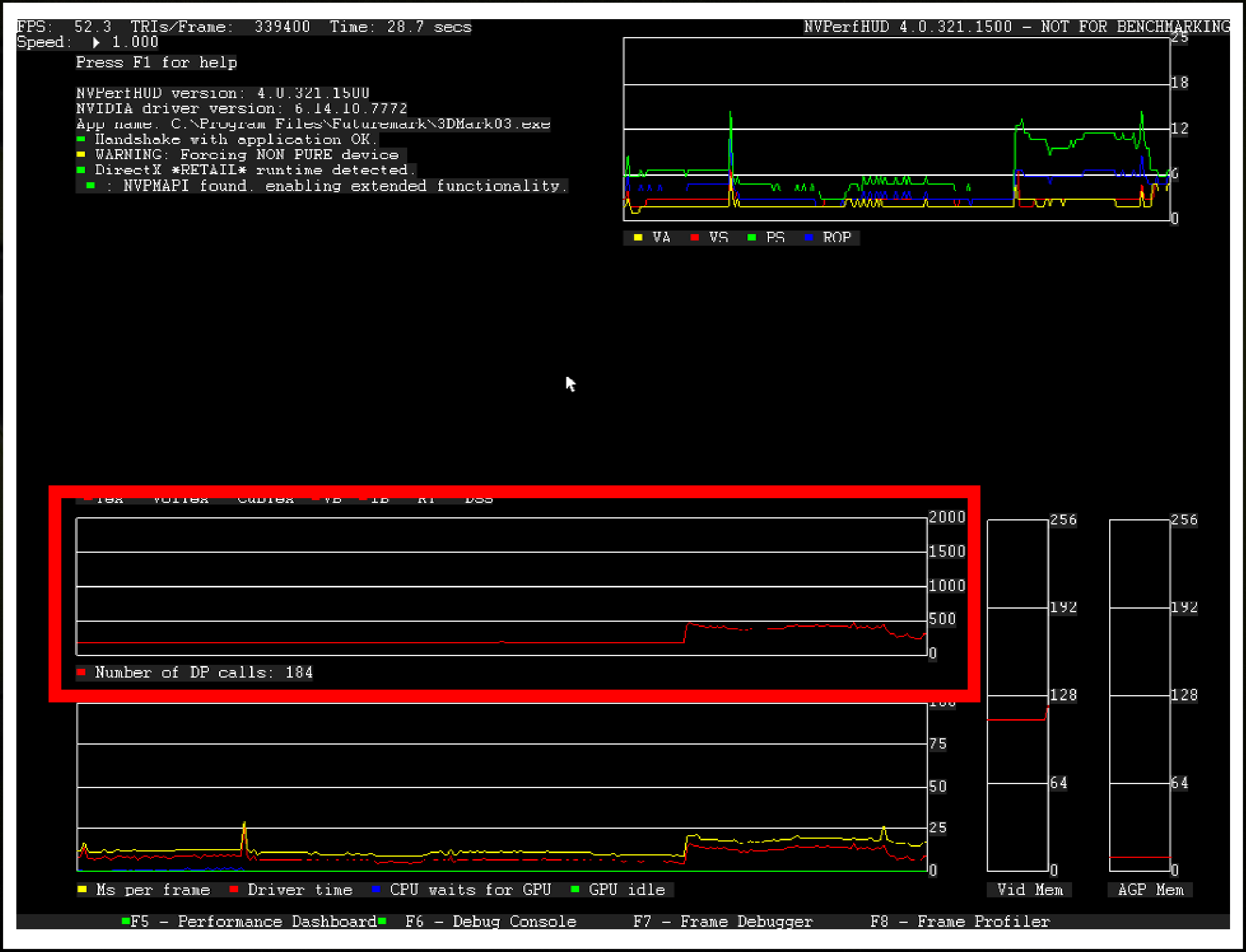


```
FPS:  52.3  TRIs/Frame:  339400  Time: 28.7 secs
Speed:  ▶ 1.000
        Press F1 for help

        NVPerfHUD version: 4.0.321.1500
        NVIDIA driver version: 6.14.10.7772
        App name. C.\Program Files\Futuremark\3DMark03.exe
        ■ Handshake with application OK.
        ■ WARNING: Forcing NON PURE device
        ■ DirectX *RETAIL* runtime detected.
         ■ : NVPMAPI found. enabling extended functionality.
```

```
NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING
```

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

Performance Dashboard



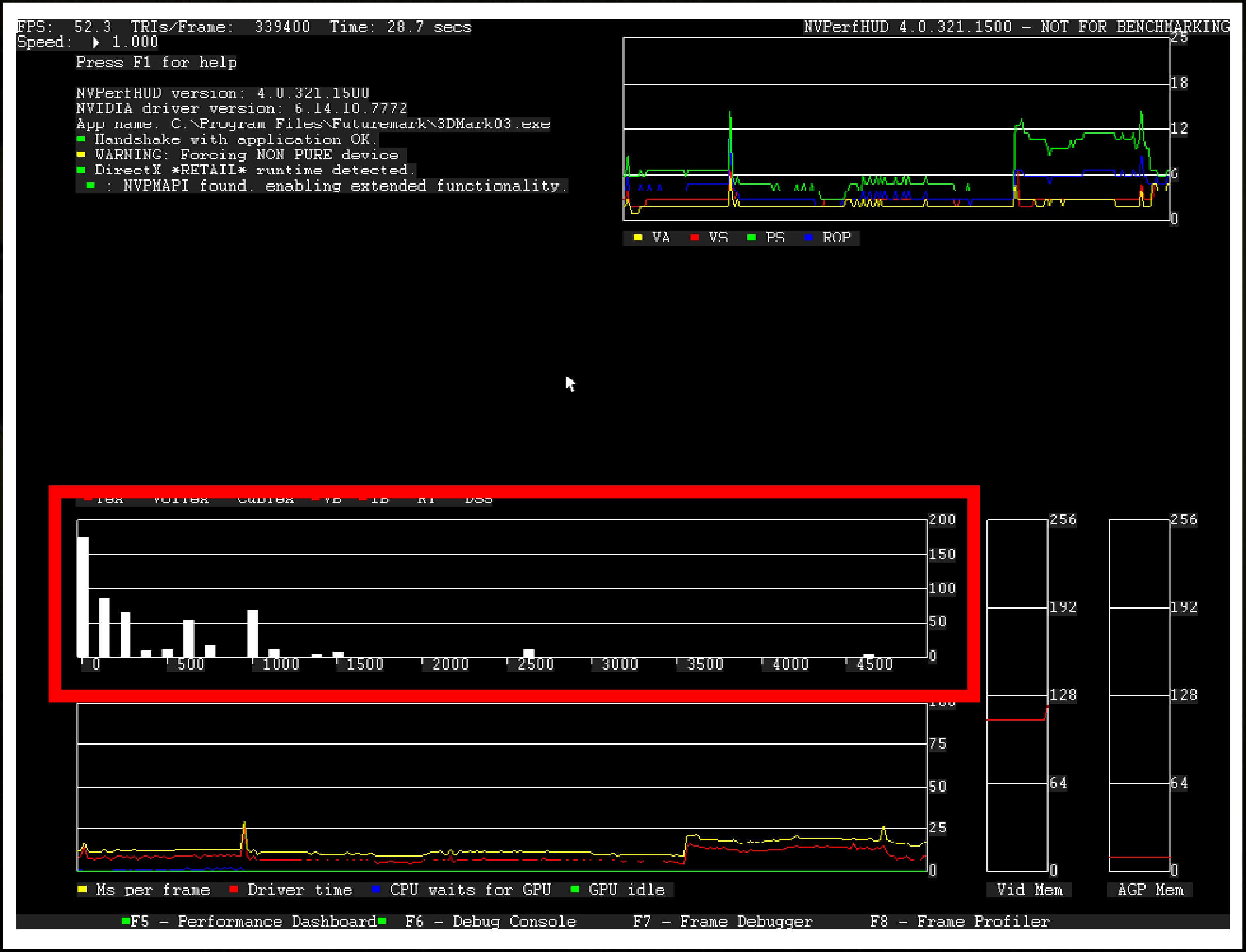
```
FPS:   52.3   TRIs/Frame:   339400   Time: 28.7 secs
Speed:   ▶ 1.000
        Press F1 for help

        NVPerfHUD version: 4.0.321.1500
        NVIDIA driver version: 6.14.10.7772
        App name: C:\Program Files\Futuremark\3DMark03.exe
        ■ Handshake with application OK.
        ■ WARNING: Forcing NON PURE device
        ■ DirectX *RETAIL* runtime detected.
         ■ : NVPMAPI found, enabling extended functionality.
```

```
NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING
```

```
■ Number of DP calls: 184
```

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

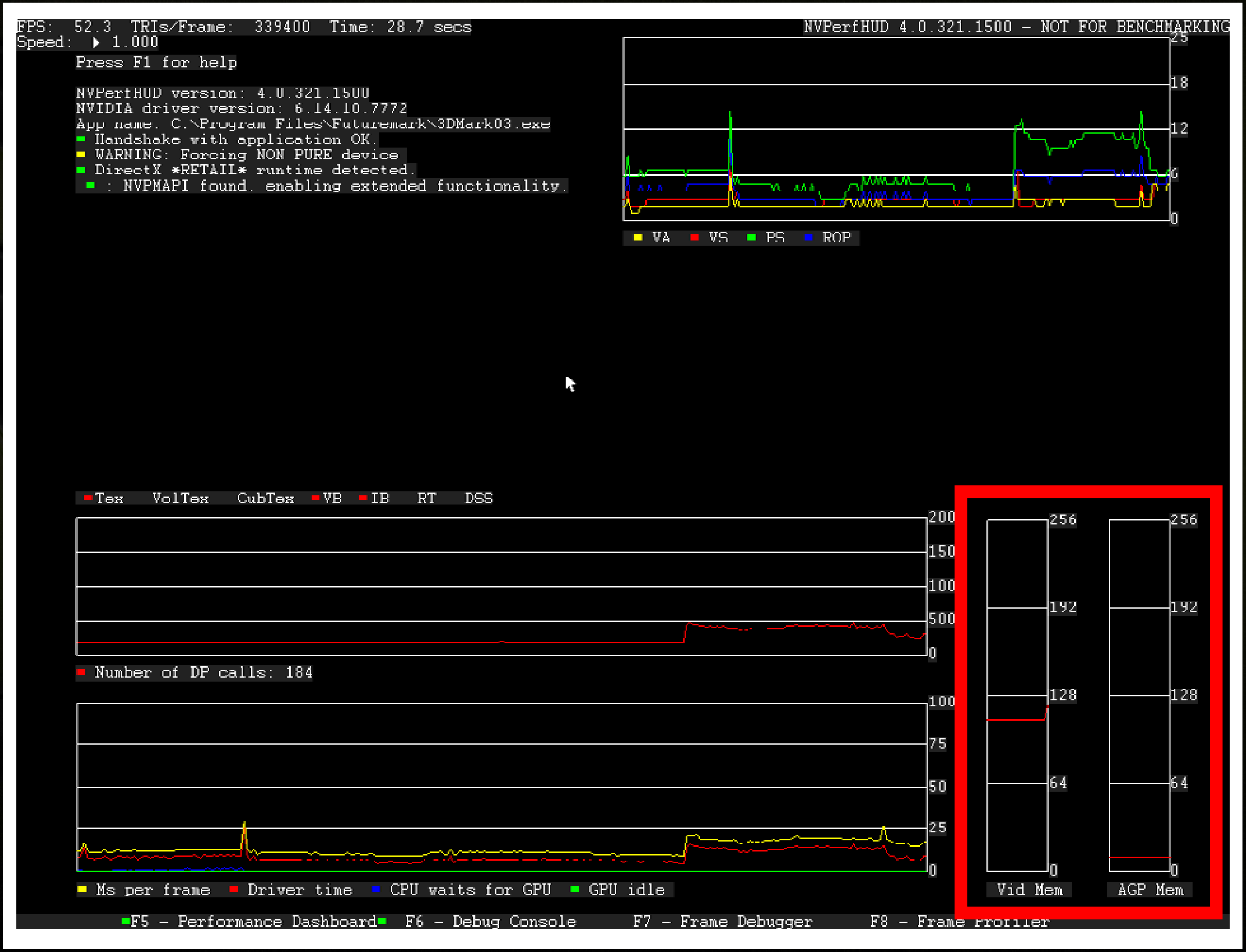
Performance Dashboard



```
FPS:  52.3  TRIs/Frame:  339400  Time: 28.7 secs
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\3DMark03.exe
        🟩 Handshake with application OK.
        🟨 WARNING: Forcing NON PURE device
        🟩 DirectX *RETAIL* runtime detected.
         🟩 : NVPMAPI found. enabling extended functionality.
```

NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING
25
18
12
6
0
🟨 VA 🟥 VS 🟩 PS 🟦 ROP
Tex Vertex CubeTex VB IB RT DSS
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
100
75
50
25
0
🟨 Ms per frame 🟥 Driver time 🟦 CPU waits for GPU 🟩 GPU idle
256
192
128
64
0
Vid Mem
256
192
128
64
0
AGP Mem
🟩F5 - Performance Dashboard🟩 F6 - Debug Console F7 - Frame Debugger F8 - Frame Profiler

Performance Dashboard

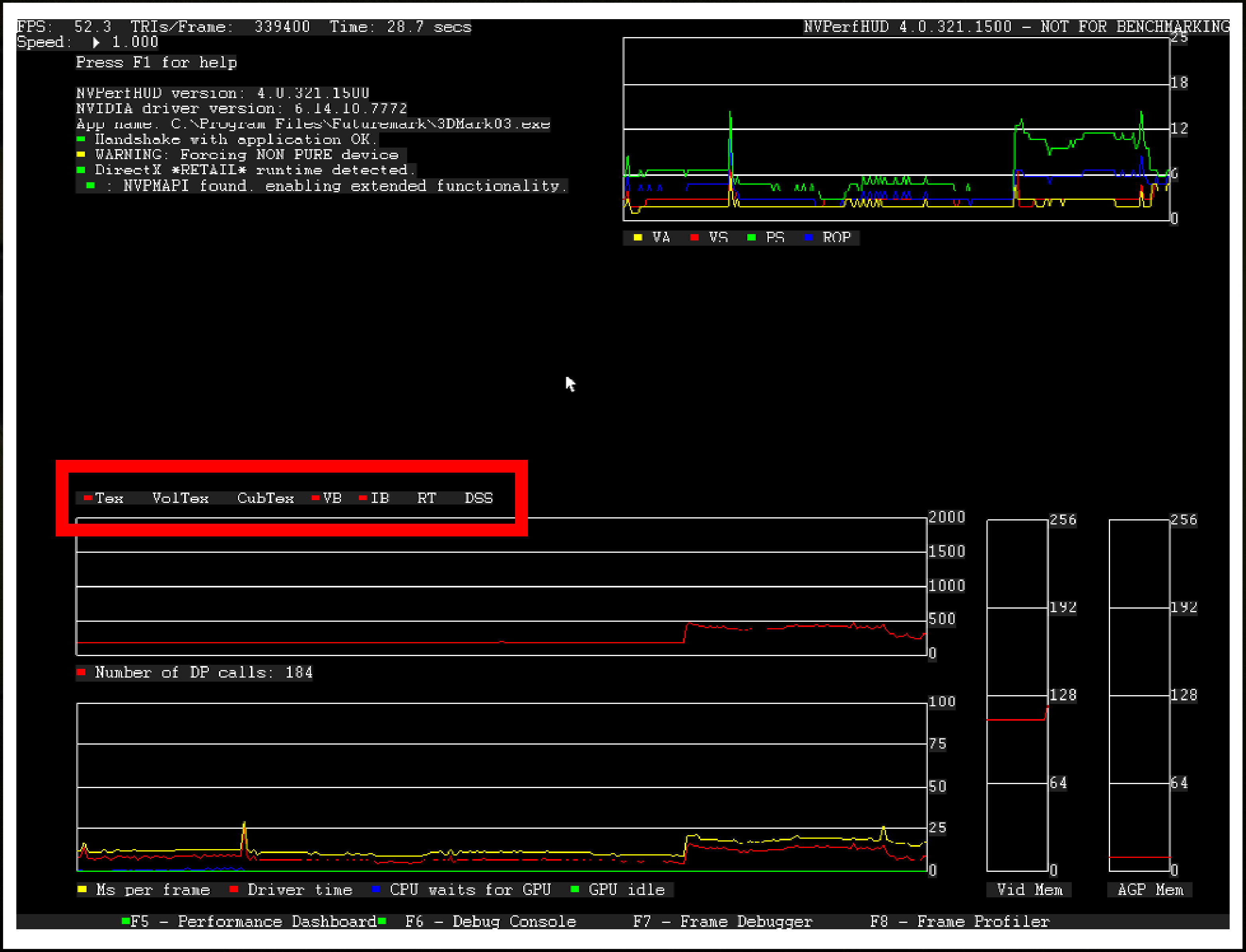


```
FPS:  52.3  TRIs/Frame:  339400  Time: 28.7 secs
Speed:  ▶ 1.000
        Press F1 for help

        NVPerfHUD version: 4.0.321.1500
        NVIDIA driver version: 6.14.10.7772
        App name: C:\Program Files\Futuremark\3DMark03.exe
        🟩 Handshake with application OK.
        🟨 WARNING: Forcing NON PURE device
        🟩 DirectX *RETAIL* runtime detected.
         🟩 : NVPMAPI found, enabling extended functionality.
```

```
NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING
```

```
🟥 Number of DP calls: 184
```

Performance Dashboard



```
FPS:  52.3  TRIs/Frame:  339400  Time: 28.7 secs          NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING
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\3DMark03.exe
        🟩 Handshake with application OK.
        🟨 WARNING: Forcing NON PURE device
        🟩 DirectX *RETAIL* runtime detected.
         🟩 : NVPMAPI found, enabling extended functionality.
```

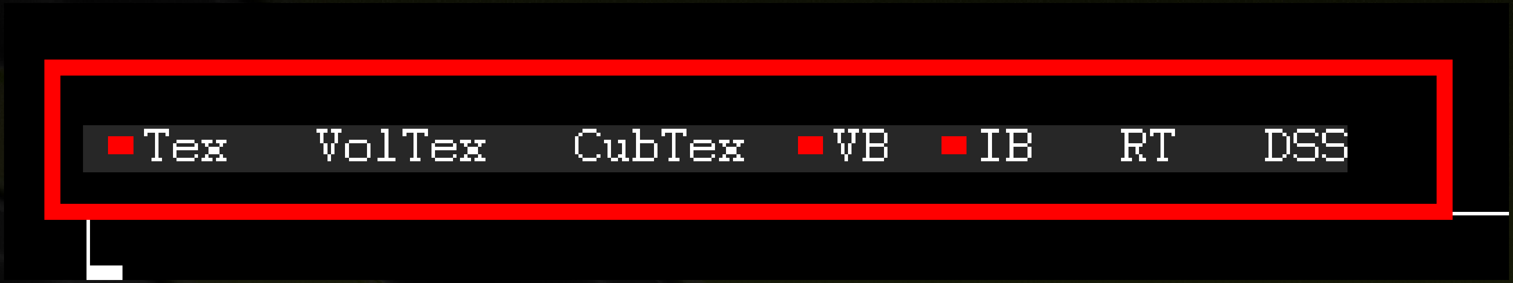
🟥 Number of DP calls: 184

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

25
18
12
6
0
🟨 VA 🟥 VS 🟩 PS 🟦 ROP
🟥Tex VolTex CubTex 🟥VB 🟥IB RT DSS
2000
1500
1000
500
0
100
75
50
25
0
🟨 Ms per frame 🟥 Driver time 🟦 CPU waits for GPU 🟩 GPU idle
256
192
128
64
0
Vid Mem
256
192
128
64
0
AGP Mem

Performance Dashboard

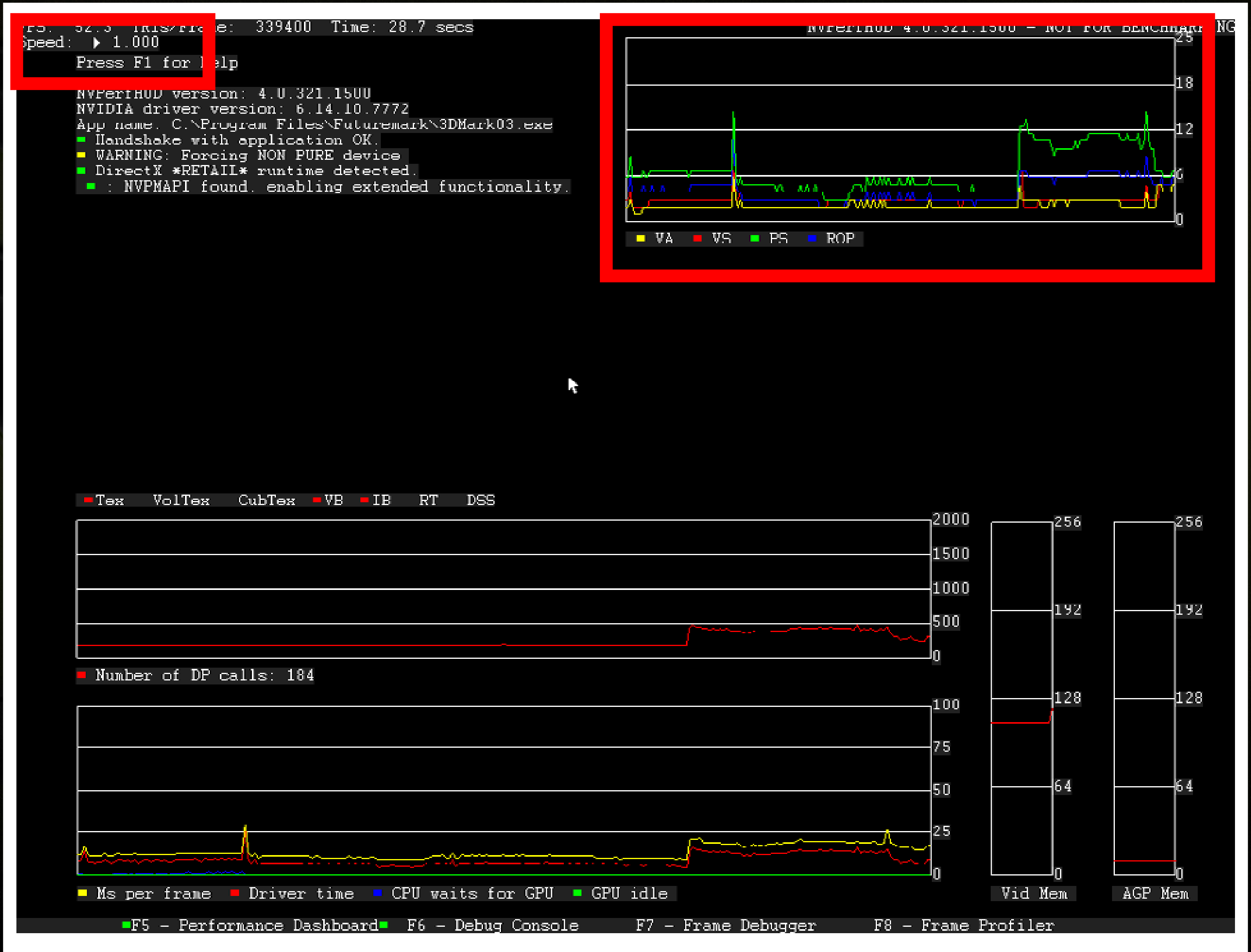
- Resource monitor



```
🟥Tex    VolTex    CubTex  🟥VB  🟥IB    RT    DSS
```

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

Performance Dashboard



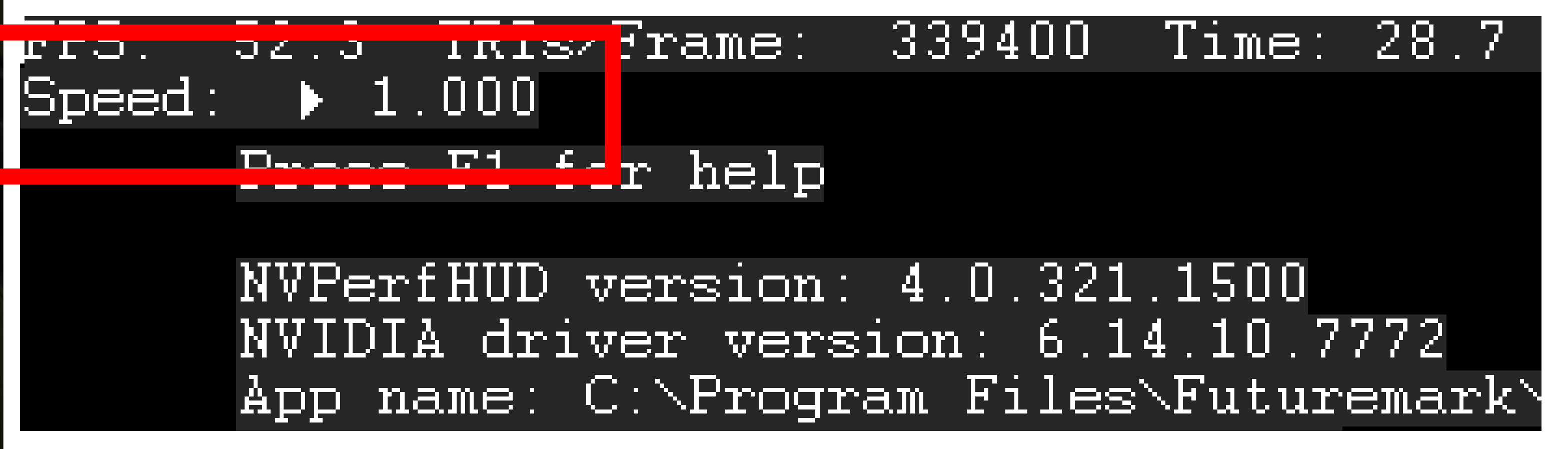
```
FPS: 52.3  TRIS/Frame:  339400  Time: 28.7 secs
Speed:  ▶ 1.000
        Press F1 for Help

        NVPerfHUD version: 4.0.321.1500
        NVIDIA driver version: 6.14.10.7772
        App name. C:\Program Files\Futuremark\3DMark03.exe
        🟩 Handshake with application OK.
        🟨 WARNING: Forcing NON PURE device
        🟩 DirectX *RETAIL* runtime detected.
         🟩 : NVPMAPI found. enabling extended functionality.
```

NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING

🟥 Number of DP calls: 184

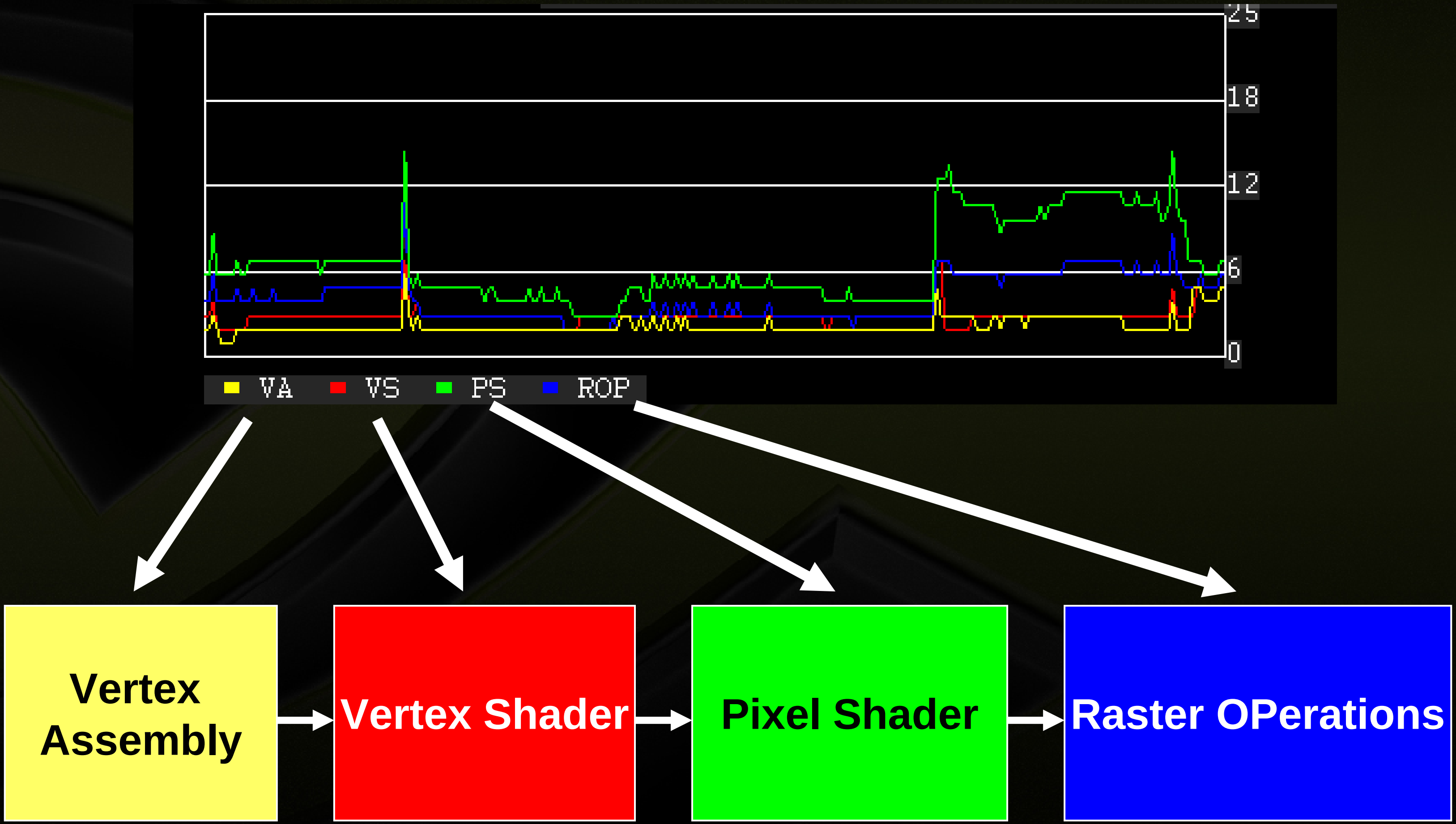
Performance Dashboard

- Speed control



```
FPS:  52.3  TRIs/Frame:   339400  Time: 28.7
Speed:   ▶ 1.000
      Press F1 for help

      NVPerfHUD version: 4.0.321.1500
      NVIDIA driver version: 6.14.10.7772
      App name: C:\Program Files\Futuremark\
```

The simplified graphics pipeline

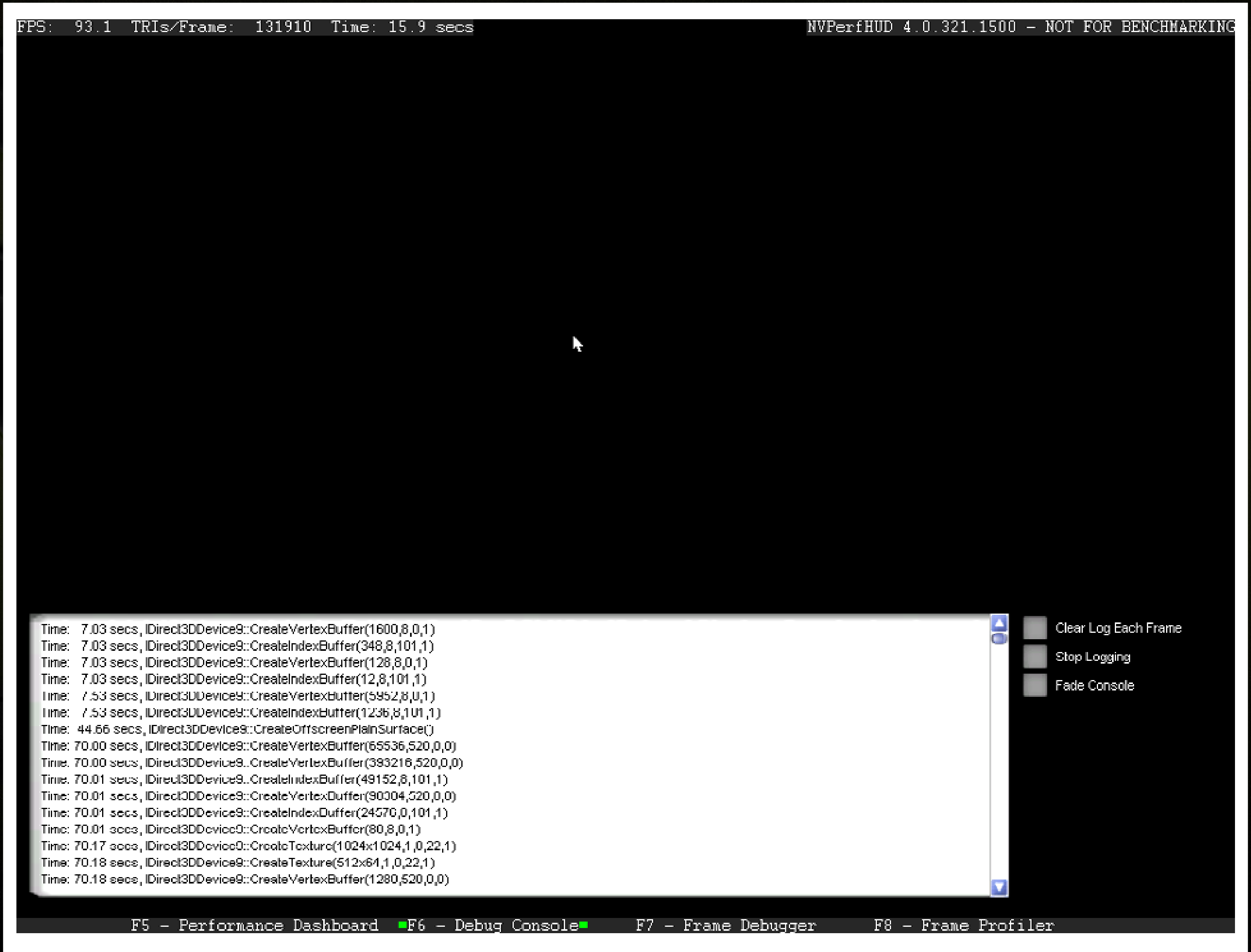


Vertex Assembly → Vertex Shader → Pixel Shader → Raster OPerations

Performance Dashboard Demo

- Install
- Configure
- Drag & Drop

Debug Console



```
FPS:   93.1   TRIs/Frame:   131910   Time: 15.9 secs                    NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING
```

```
Time:   7.03 secs, IDirect3DDevice9::CreateVertexBuffer(1600,8,0,1)
Time:   7.03 secs, IDirect3DDevice9::CreateIndexBuffer(348,8,101,1)
Time:   7.03 secs, IDirect3DDevice9::CreateVertexBuffer(128,8,0,1)
Time:   7.03 secs, IDirect3DDevice9::CreateIndexBuffer(12,8,101,1)
Time:   7.53 secs, IDirect3DDevice9::CreateVertexBuffer(5952,8,0,1)
Time:   7.53 secs, IDirect3DDevice9::CreateIndexBuffer(1236,8,101,1)
Time:  44.66 secs, IDirect3DDevice9::CreateOffscreenPlainSurface()
Time: 70.00 secs, IDirect3DDevice9::CreateVertexBuffer(65536,520,0,0)
Time: 70.00 secs, IDirect3DDevice9::CreateVertexBuffer(393216,520,0,0)
Time: 70.01 secs, IDirect3DDevice9::CreateIndexBuffer(49152,8,101,1)
Time: 70.01 secs, IDirect3DDevice9::CreateVertexBuffer(90304,520,0,0)
Time: 70.01 secs, IDirect3DDevice9::CreateIndexBuffer(24576,8,101,1)
Time: 70.01 secs, IDirect3DDevice9::CreateVertexBuffer(80,8,0,1)
Time: 70.17 secs, IDirect3DDevice9::CreateTexture(1024x1024,1,0,22,1)
Time: 70.18 secs, IDirect3DDevice9::CreateTexture(512x64,1,0,22,1)
Time: 70.18 secs, IDirect3DDevice9::CreateVertexBuffer(1280,520,0,0)
```

Frame Debugger

- Demo

Frame Debugger, advanced view

- Demo

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

Frame Profiler Demo, advanced view

About freezing the application

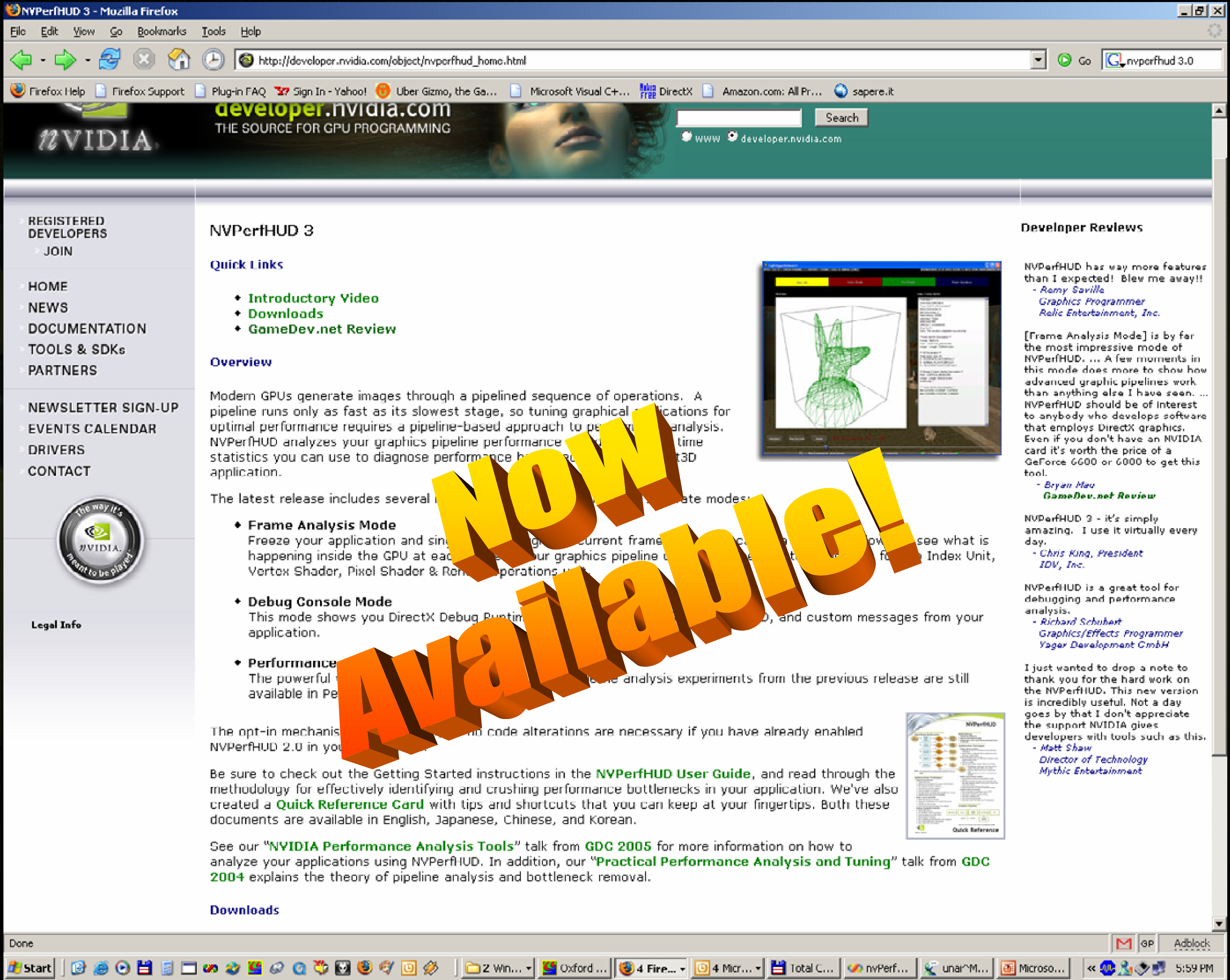
- Only possible if the application uses time-based animation

- Stop the clock
  - Intercept: QueryPerformanceCounter(), timeGetTime()
  - NO RDTSC!!

- Pos += V * DeltaTime

Schedule

- Beta: August
- Release : September



NVPerfHUD 3

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 ...ications for optimal performance requires a pipeline-based approach to pe... analysis. NVPerfHUD analyzes your graphics pipeline performance ... time statistics you can use to diagnose performance b... ...t3D application.

The latest release includes several ... ...ate modes:

- **Frame Analysis Mode**
  Freeze your application and sin... ...g ...current frame ... ...ow ... see what is happening inside the GPU at eac... ...ur graphics pipeline ... fo... Index Unit, Vertex Shader, Pixel Shader & Ren... ...perations ...

- **Debug Console Mode**
  This mode shows you DirectX Debug Runtim... ...D, and custom messages from your application.

- **Performance**
  The powerful ... ...e analysis experiments from the previous release are still available in Pe...

The opt-in mechanis... ...no code alterations are necessary if you have already enabled NVPerfHUD 2.0 in you...

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!!
- Romy 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 6000 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

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

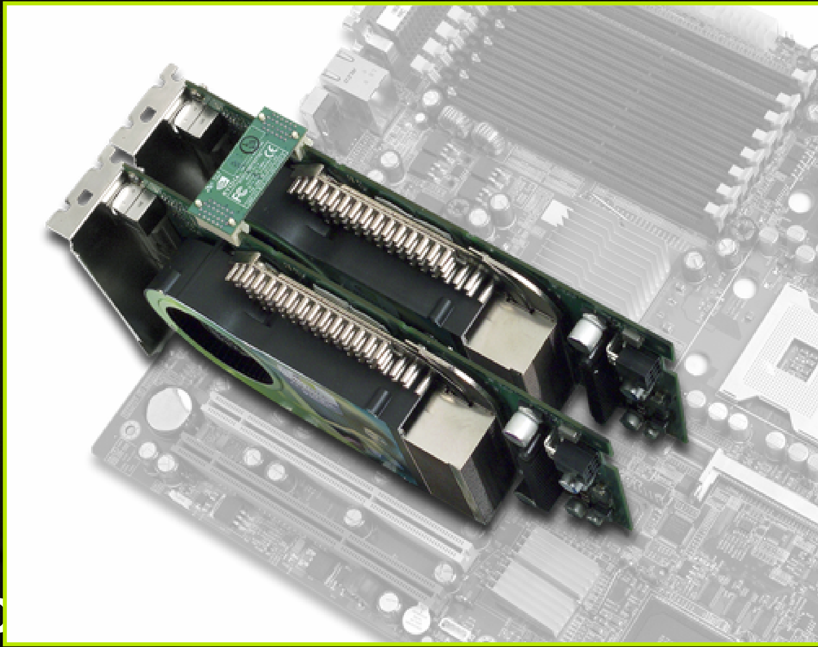
FXComposer@nvidia.com

SLI

Matthias M Wloka
NVIDIA Corporation

SLI: Scalable Link Interface

- Plug 2 identical GPUs into PCI-E motherboard
- Driver still reports only one (logical) device
  - Renders up to 1.9x faster
- Video memory does NOT do



Don’t Care For High-End Niche Markets

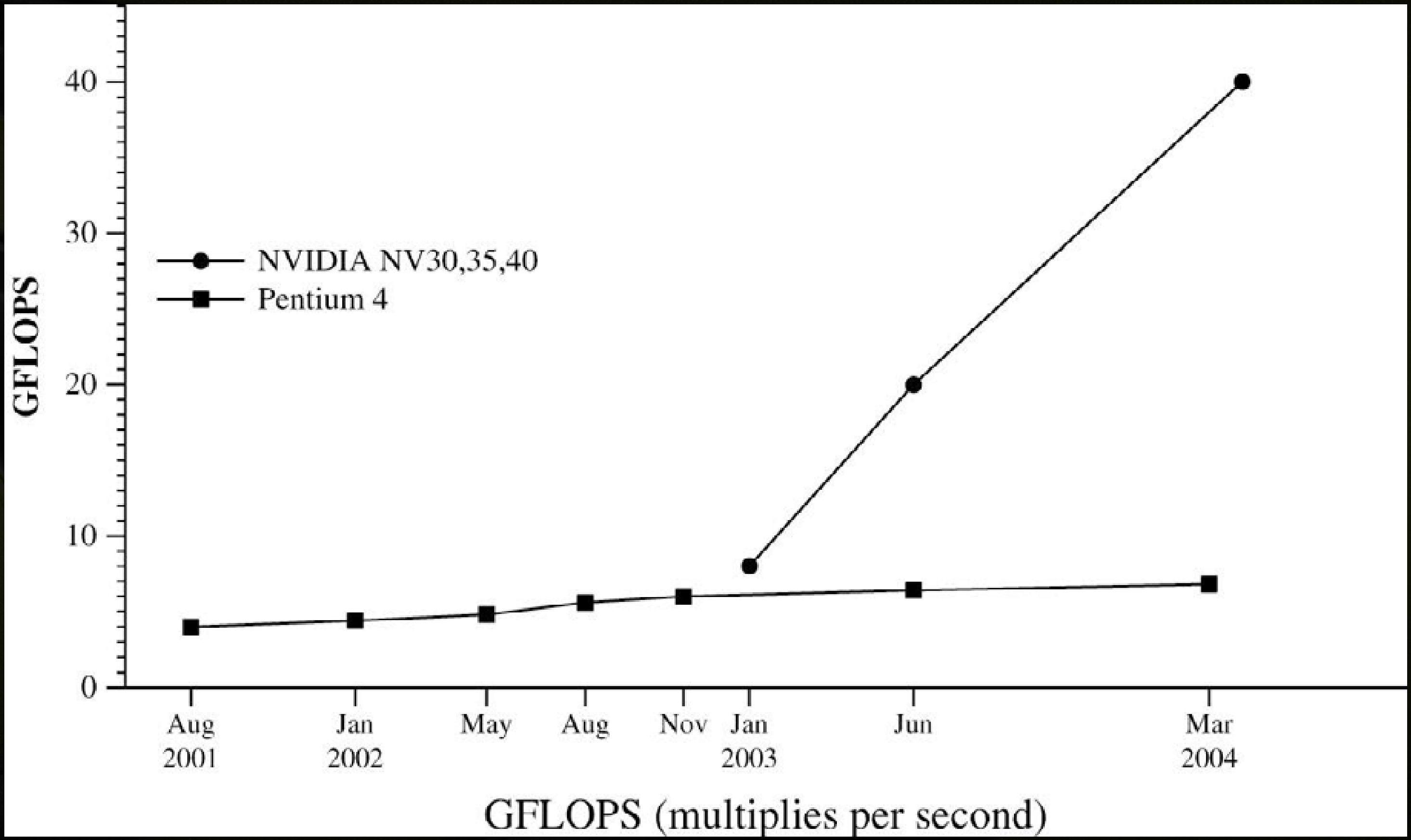
- SLI becoming mainstream:
  - GeForce 6600 GT SLI
  - In addition to 6800 GT and 6800 Ultra
- Dual core boards
  - Gigabyte 3D1:
    Dual 6600 GT

- SLI motherboards
  sold to date: > 350,000 units
  - That’s > 25% of total nForce 4

Game Development Cycle

- 2 years (or more)
  - CPU performance doubles (or less)
  - GPU performance quadruples
- CPU/GPU balance shifts!
  - Worse: CPU-hungry modules come later: AI, physics, full game play
- SLI hints at future GPU vs. CPU balance
  - For target ‘mainstream’ spec

The Last Couple of Years



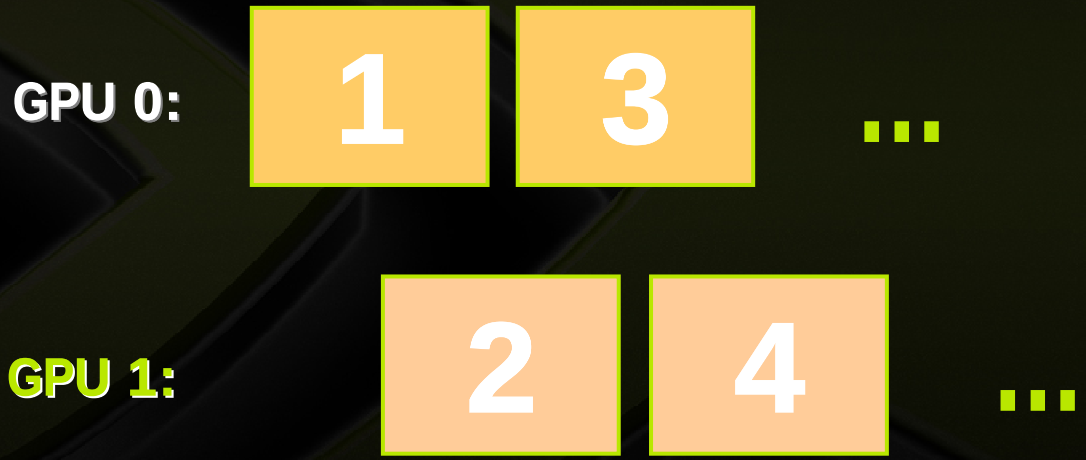
Courtesy Ian Buck, Stanford University

Ok, How Does SLI Work?

- Compatibility mode:
  - Only uses one GPU
  - No SLI benefits
- Alternate frame rendering (AFR)
- Split frame rendering (SFR)

AFR

- Each GPU works on its own frame



GPU 0: 1 3 ...

GPU 1: 2 4 ...

- Scan-out toggles where to read framebuffer from

General Rendering Case for AFR

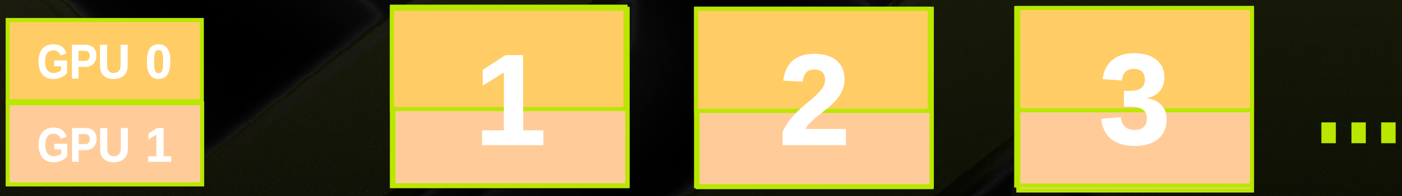
- If frame not self-contained:
  - Push necessary data to other GPU
  - E.g., updating render-to-texture targets only every other frame
- Pushing data to other GPU is overhead
  - Hence not 2x speed-up

AFR Advantages

- All work is parallelized
  - Pixel fill, raster, vertex transform
- Preferred SLI mode
- Works best when frame self-contained
  - No prior work is re-used
  - No communications overhead between GPUs

SFR

- Both GPUs work on the same frame
  - GPU 0 renders top portion
  - GPU 1 renders bottom portion



- Scan-out combines framebuffer data

General Rendering Case for SFR

- Load-balance ‘top’ vs. ‘bottom’
  - If one GPU took longer to render
  - Adjust load accordingly (make it work less)
- Clip vertices to top/bottom portions
  - Avoids both GPUs processing all vertices
  - But not perfect
- Still requires data sharing:
  - E.g., render to texture

SFR Compared to AFR

- SFR works even when limiting number of frames buffered
  - Or when AFR otherwise fails
- In general, SFR has more communications overhead
- Applications with heavy vertex load benefit less from SFR

How Do I Detect SLI Systems?

- NVCpl API:
  - NVIDIA-specific API supported by all NV drivers
- Function support for:
  - Detecting that NVCpl API is available
  - Bus mode (PCI/AGP/PCI-E) and rate (1x-8x)
  - Video RAM size
  - SLI

NVCpl API SLI Detection

- SDK sample and full documentation available

```
HINSTANCE hLib = ::LoadLibrary("NVCPL.dll");

NvCplGetDataIntType NvCplGetDataInt;
NvCplGetDataInt =
    (NvCplGetDataIntType)::GetProcAddress(hLib,
                    "NvCplGetDataInt");

long    numSLIGPUs  = 0L;
NvCplGetDataInt(NVCPL_API_NUMBER_OF_SLI_GPUS,
                           &numSLIGPUs);
```

Forcing SLI Support In Your Game

- Use NVCpl
  - NvCplSetDataInt() sets
    AFR, SFR, Compatibility mode
  - See SDK sample
- Modify or create a profile:
  - http://nzone.com/object/nzone_sli_appprofile.html
  - End-users can create profiles as well

Overview: Things Interfering with SLI

- CPU-bound applications
  - Or vsync enabled
- Limiting number of frames buffered
- Communications overhead

CPU-Bound Applications

- SLI cannot help
- Reduce CPU work or better:
- Move CPU work onto the GPU
  - See http://GPGPU.org
- Don’t throttle frame-rate

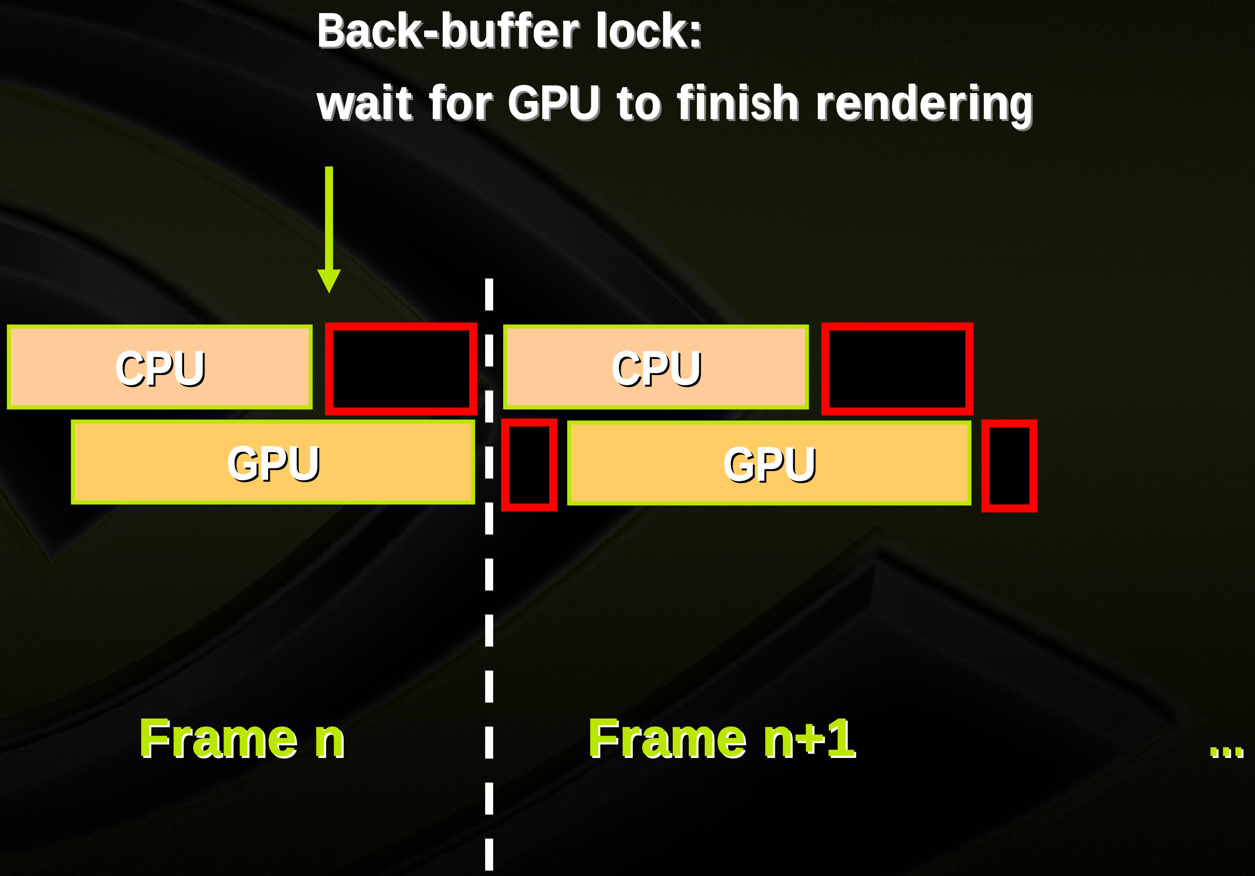
VSync Enabled

- Throttles frame-rate to monitor refresh
- Enabling triple-buffering does NOT offset enabling vsync:
  - If render-rate faster than monitor refresh,
  - Then vsync still gates GPU
- Worse, triple-buffering
  - Increases lag
  - Consumes (much) more video-memory

Limiting Number of Frames Buffered

- Some apps allow at most one frame buffered
  - To reduce lag
  - Via event queries
  - Don’t lock/read back-buffer: Causes CPU stall!
- Disables AFR SLI speed-up
- But SLI is up to ~1.9x faster
  - I.e., SLI systems ~1.9x less lag

Why Locking the Back-Buffer Is Bad

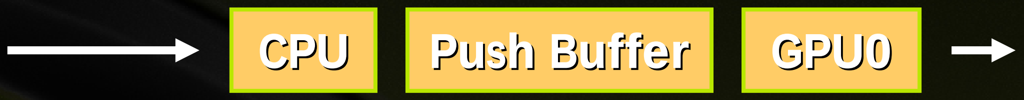


Limit Frames Buffered to Number of GPUs

- Single GPU system:
  Buffer at most 1 frame

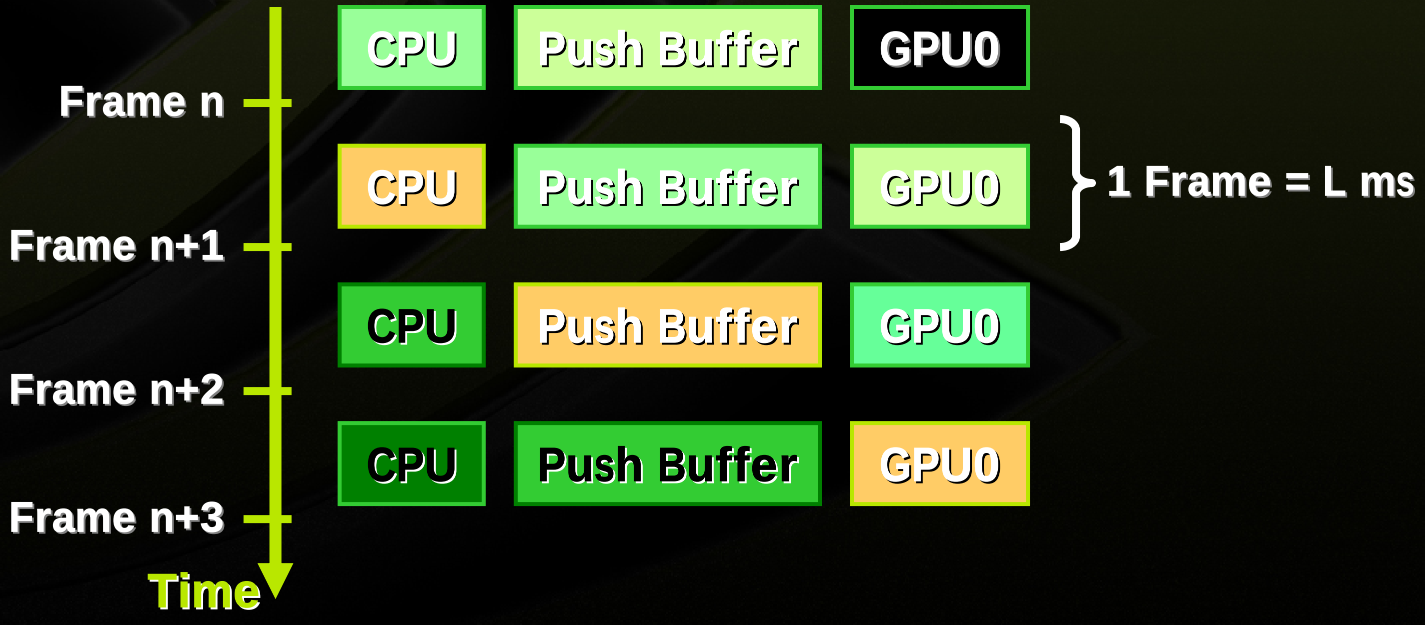
- When detecting SLI system:
  Buffer at most 2 frame

The Basic Pipeline



CPU → Push Buffer → GPU0 →

Frames flow through pipe over time:



CPU | Push Buffer | GPU0

Frame n

CPU | Push Buffer | GPU0

1 Frame = L ms

Frame n+1

CPU | Push Buffer | GPU0

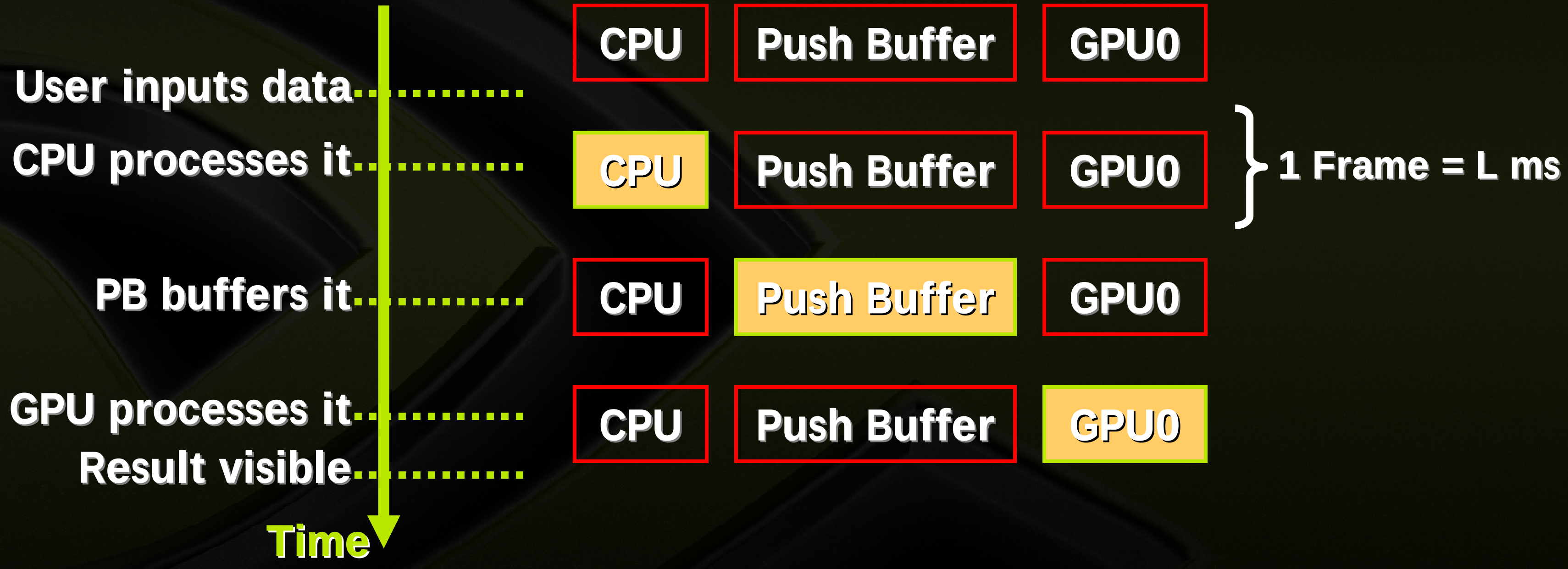
Frame n+2

CPU | Push Buffer | GPU0

Frame n+3

Time

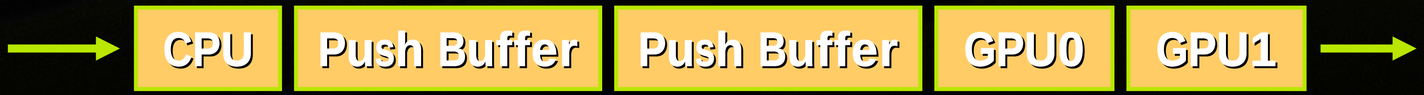
Single GPU Latency



Total latency: 3L ms

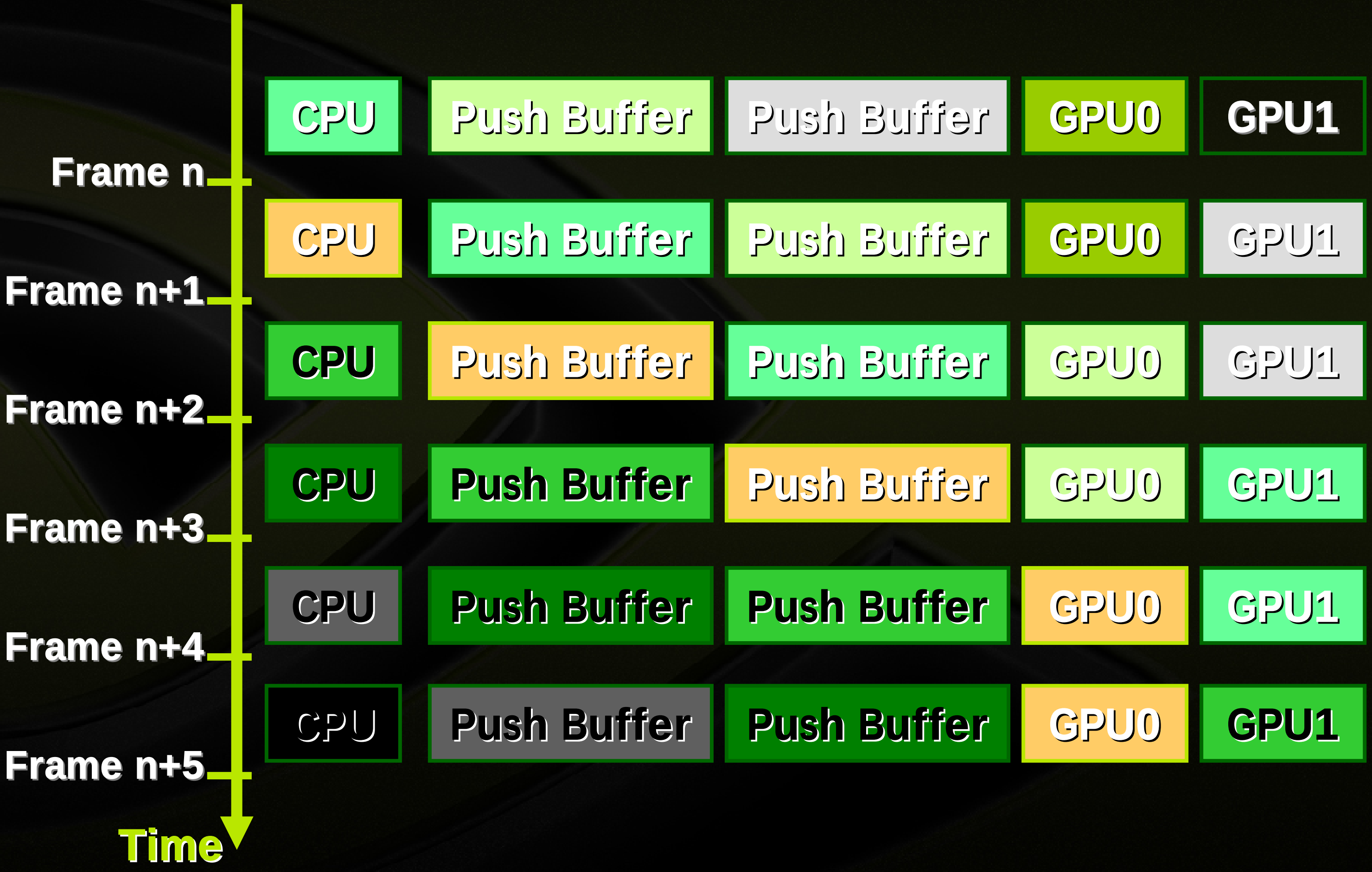
Latency Assumptions

- GPU limited
  - If not, then push buffer contains <1 frame
  - No point in limiting push buffer
- SLI is 2x faster
  - Can relax this later!
- Increase frames buffered to 2:



CPU → Push Buffer → Push Buffer → GPU0 → GPU1

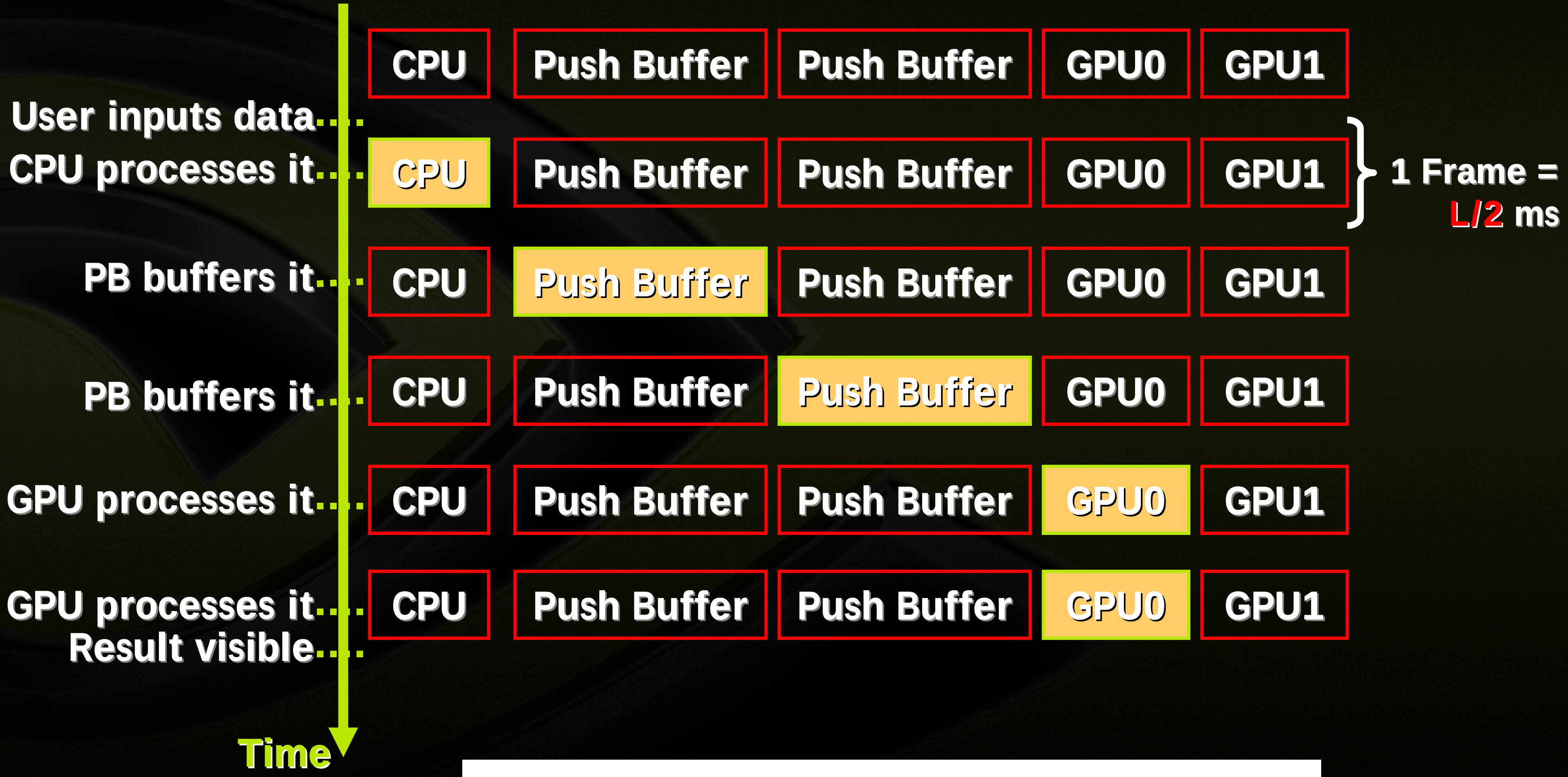
Frames Flowing Through AFR SLI



CPU	Push Buffer	Push Buffer	GPU0	GPU1
CPU	Push Buffer	Push Buffer	GPU0	GPU1
CPU	Push Buffer	Push Buffer	GPU0	GPU1
CPU	Push Buffer	Push Buffer	GPU0	GPU1
CPU	Push Buffer	Push Buffer	GPU0	GPU1
CPU	Push Buffer	Push Buffer	GPU0	GPU1

1 Frame = L/2 ms

AFR SLI Latency



	CPU	Push Buffer	Push Buffer	GPU0	GPU1
User inputs data CPU processes it	**CPU**	Push Buffer	Push Buffer	GPU0	GPU1
PB buffers it	CPU	**Push Buffer**	Push Buffer	GPU0	GPU1
PB buffers it	CPU	Push Buffer	**Push Buffer**	GPU0	GPU1
GPU processes it	CPU	Push Buffer	Push Buffer	**GPU0**	GPU1
GPU processes it Result visible	CPU	Push Buffer	Push Buffer	**GPU0**	GPU1

1 Frame = L/2 ms

Time

Total latency: 5·L/2 ms

Latency Comparison: Single vs. AFR

- Single GPU latency: 3L ms
  - 3 frames of length L ms
- AFR SLI GPU latency: 5 L/2 = 2.5L ms!
  - 5 frames of length L/2 ms
    (i.e., double frame rate)
  - Despite buffering twice as many frames
- SLI speed-up only needs to be 1.66!
  - 3L = 5L/x → x = 5L/3L = 1.66
  - Most games speed-up by ~1.8

SFR Latency?

- SFR unaffected by buffering one frame
- SFR speed-up directly reduces lag
  - If SFR 2x faster,
  - Then latency 2x shorter

Even Better: Limit Lag Based on FPS

- If your game runs at over 100 fps
  - Reasonable to buffer 3 frames
- If your game runs at less than 15 fps
  - Only allow one frame to buffer
- Faster SLI system gets automatic benefit
- Our drivers already do that
  - > 15 fps buffer 3 frames as usual
  - < 15fps reduce number of frames buffered

Overview: Things Interfering with SLI

- CPU-bound applications
  - Or vsync enabled
- Limiting number of frames buffered
- Communications overhead

Communications Overhead

- Peer to peer SLI memory transfers
  - Transfer itself costs bandwidth and time
  - GPU stalls waiting for transfer to complete
- Or replicate operations on both GPUs
  - For example, render to texture
- Relevant resources:
  - Vertex/index buffers
  - Textures
  - Render targets

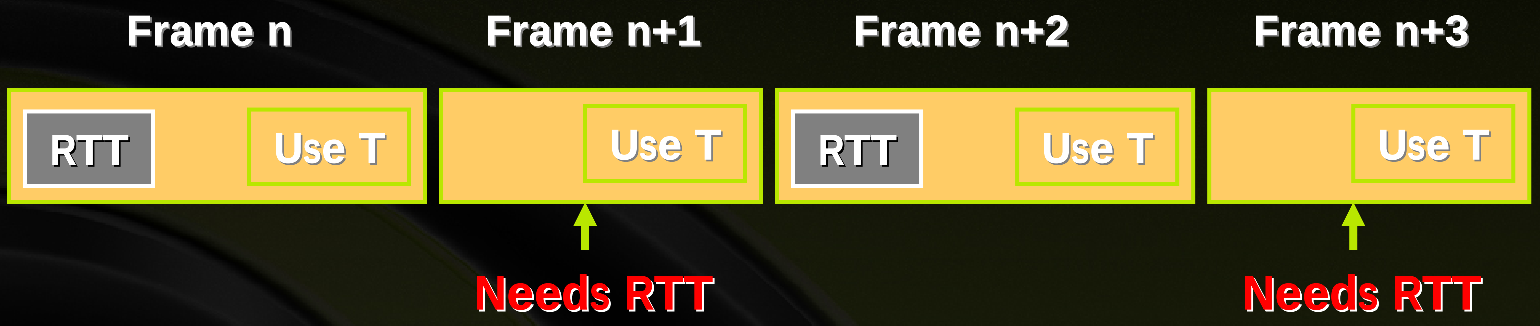
Uploading Resources On the Fly

- Remember video RAM is duplicated
- Need to transfer to both video RAMs
- Not much developers can do to avoid this
  - Oh well

Render Targets

- Clear Z
  - Always clear Z!
- Clear color when detecting SLI
  - Tells driver that the old data is irrelevant
  - No need to transfer old data across GPUs
- Don’t reuse data across frames
  - Make frames self sufficient, i.e., independent from one another

Update-Skipping “Optimization”



Frame n

Frame n+1

Frame n+2

Frame n+3

RTT

Use T

Use T

RTT

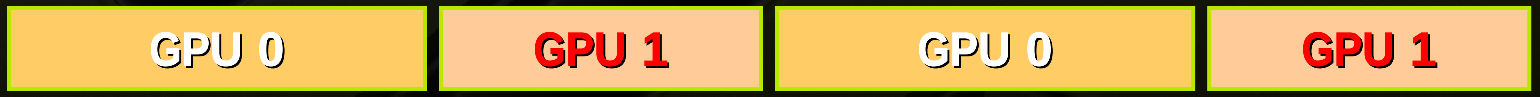
Use T

Use T

Needs RTT

Needs RTT

- Added SLI overhead:



GPU 0

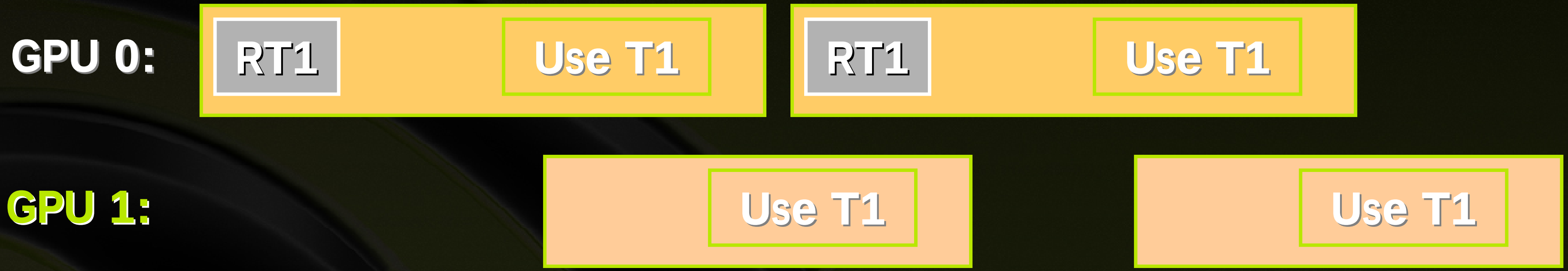
GPU 1

GPU 0

GPU 1

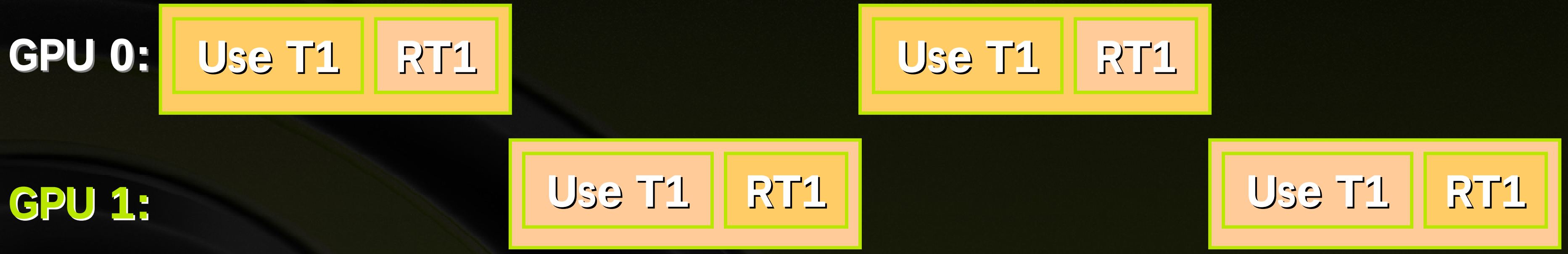
  - GPU 1 stalls until GPU 0 RTT finishes and transfers
  - Or GPU 1 duplicates RTT operation
  - Might as well do right thing when on SLI

Render Early, Use Late!



- Avoid sync-stalls
  - In AFR SLI as shown
  - And in single GPU mode
  - But still has communications overhead

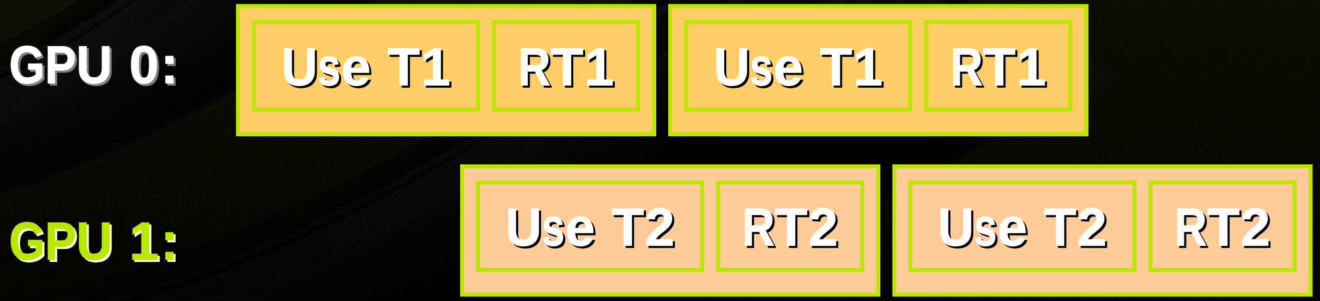
Really Bad: Use Early, Render Late



GPU 0: Use T1 RT1 Use T1 RT1

GPU 1: Use T1 RT1 Use T1 RT1

Instead: Ring-buffer textures when on SLI!



GPU 0: Use T1 RT1 Use T1 RT1

GPU 1: Use T2 RT2 Use T2 RT2

SLI Performance Debug Support

- SLI support in NVPerfKit:
  - Pluggable hardware and driver signals for
  - PIX
  - perfmon.exe
  - pdh (your game, VTune…)

- “NVIDIA Performance Analysis Tools”
  Today, 2:30pm - 3:30pm

Supported SLI Performance Signals

- Total SLI peer-to-peer bytes
- Total SLI peer-to-peer transactions

- Above originating from
  - Vertex/index buffers: bytes and transactions
  - Textures: bytes and transactions
  - Render targets: bytes and transactions

Questions?

- GPU Programming Guide, Chapter 8
  http://developer.nvidia.com/object/gpu_programming_guide.html

- http://developer.nvidia.com
  The Source for GPU Programming

- mwloka@nvidia.com

- Slides available online

NVIDIA SDK

The Source for GPU Programming

developer.nvidia.com
THE SOURCE FOR GPU PROGRAMMING
Antialiasing with Transparency
GEFORCE 7
DIRECTX
Whitepaper Video Files Run
Antialiasing with Post Processing
GEFORCE 3
DIRECTX
Whitepaper Video Files Run
Glow
GEFORCE 3
DIRECTX
Whitepaper Files Run
NVIDIA Control Panel API
GEFORCE 3
Utility

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

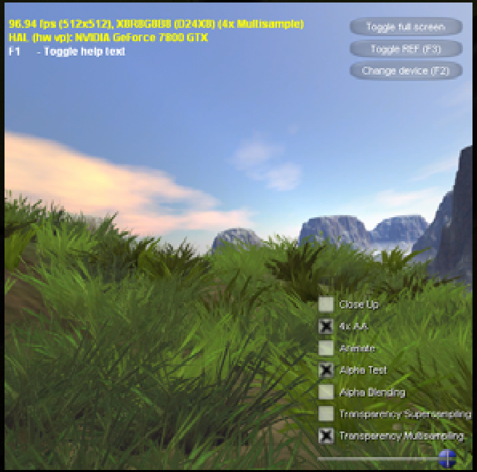
gamedeveloper FRONTLINE AWARD '04

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

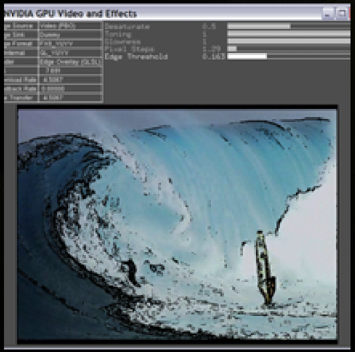
  Transparency AA, GPU Cloth, Geometry Instancing, Rainbow Fogbow, 2xFP16 HRD, Perspective Shadow Maps, Texture Atlas Utility, ...

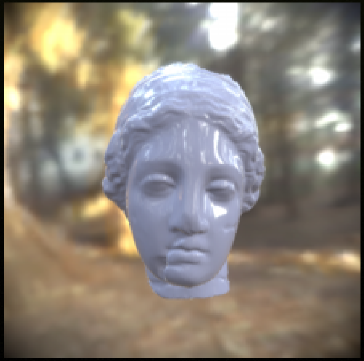
- Hundreds of effects, complete with custom geometry, animation and more:

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



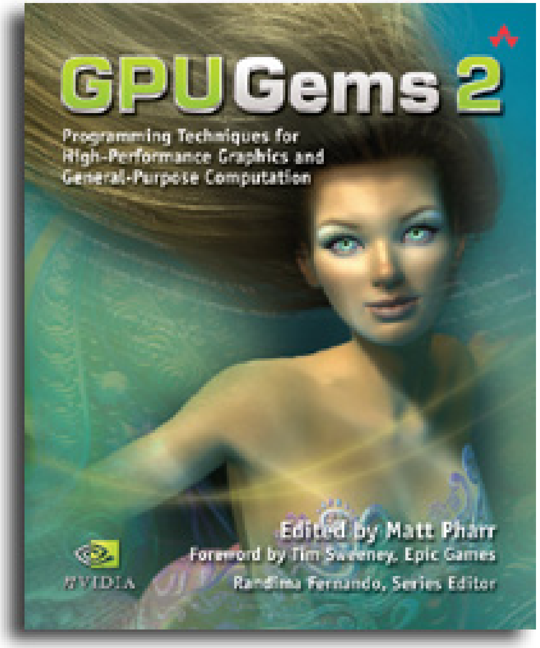
```
96.94 fps (512x512), X8R8G8B8 (D24X8) (4x Multisample)
HAL (hw vp): NVIDIA GeForce 7800 GTX
F1    - Toggle help text
```





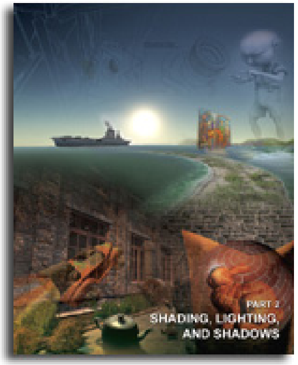
GPU Gems 2

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


GPU Gems 2
Programming Techniques for High-Performance Graphics and General-Purpose Computation
Edited by Matt Pharr
Foreword by Tim Sweeney, Epic Games
Randima Fernando, Series Editor
NVIDIA

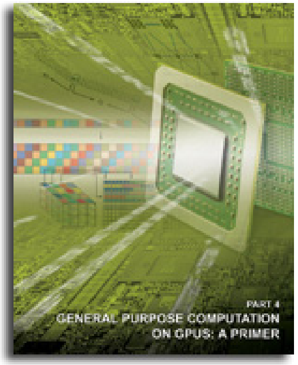
- 880 full-color pages
- 330 figures
- Hard cover
- $59.99
- Experts from universities and industry

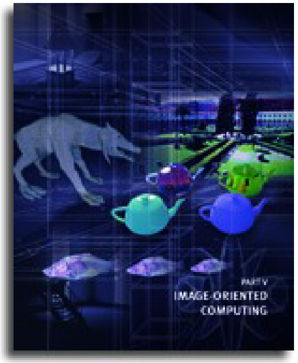
Graphics Programming

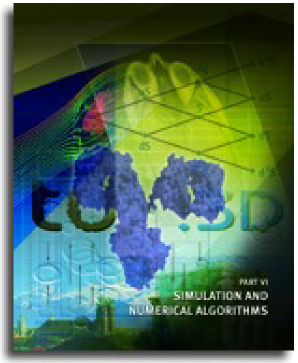


- Geometric Complexity
- Shading, Lighting, and Shadows
- High-Quality Rendering

GPGPU Programming







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