



**NVIDIA®**

## **Performance Tools**

**Jeff Kiel, NVIDIA Corporation**  
**Sim Dietrich, Composite Studios**



# Performance Tools Agenda



- Problem statement
- GPU pipelined architecture at a glance
- NVPerfKit 2.0: Driver and GPU Performance Data
  - GLExpert: OpenGL API Assistance
  - NVPerfHUD: The GPU Performance Accelerator
  - NVPerfSDK: Integrated into your application
    - NVPerfAPI
    - PDH, NVDevCPL
  - NVIDIA plug-In for Microsoft PIX for Windows
- Solutions to common bottlenecks
- NVShaderPerf: Shader Performance



# What's The Problem?



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



# GPU architecture at a glance

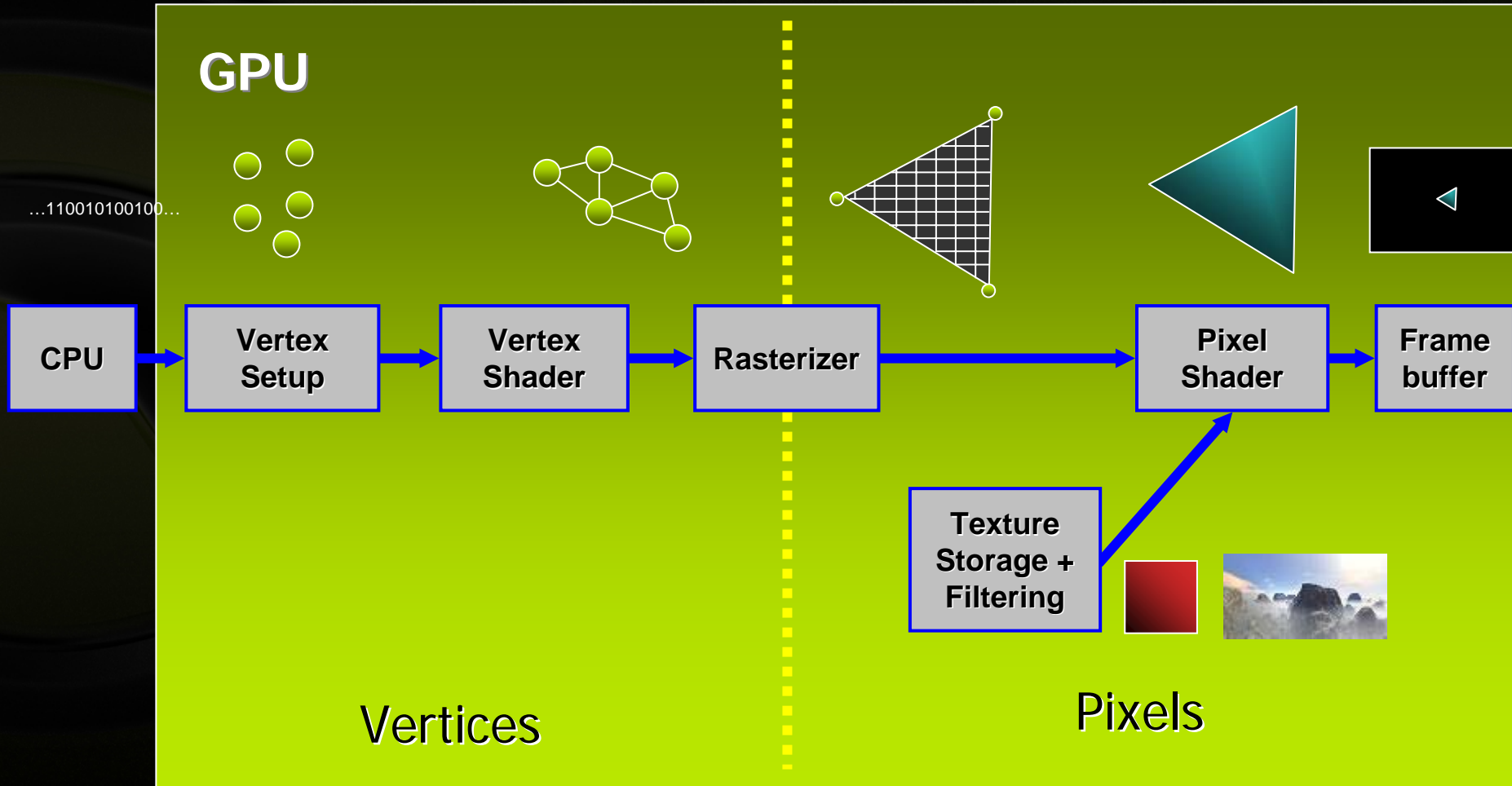


- **Pipelined architecture: each unit needs the data from the previous unit to do its job**
- **Method: Bottleneck identification and elimination**
- **Goal: Balance the pipeline**



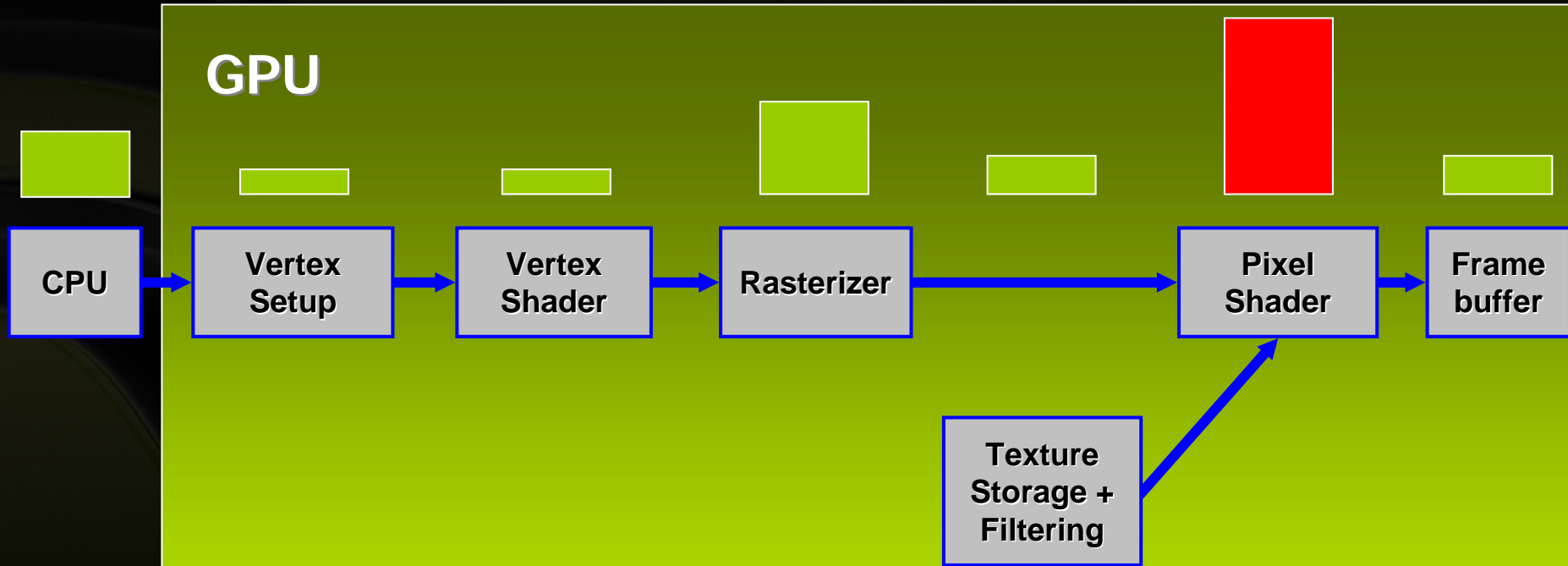


# GPU Pipelined Architecture (simplified view)





# GPU Pipelined Architecture (simplified view)



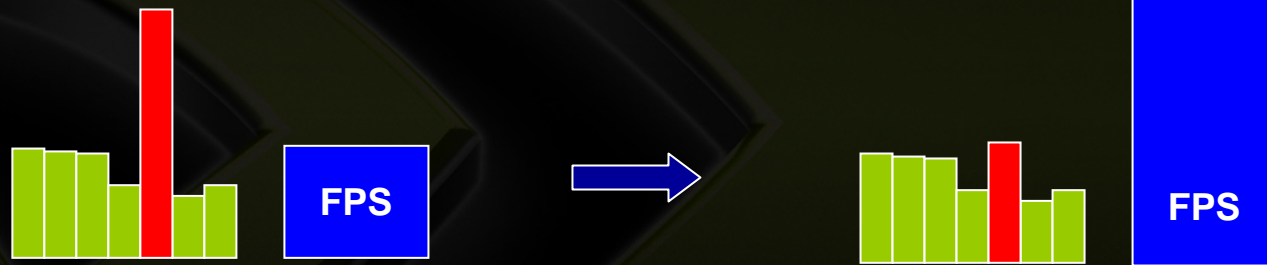
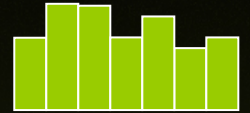
One unit can limit the speed of the pipeline...



# Classic Bottleneck Identification



Modify target stage to decrease workload



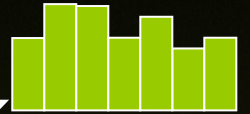
If performance/FPS improves greatly, this stage is the bottleneck  
Careful not to change the workload of other stages!



# Classic Bottleneck Identification



Rule out other stages, give them little or no work



FPS

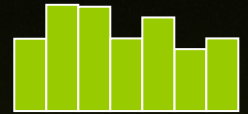


FPS

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 at different points along the pipeline while rendering**
  - Compare amount of work done to maximum work possible
  - Query the GPU for unit bottleneck information
- **The answer? NVPerfKit!**
  - NVPerfHUD: The GPU Performance Accelerator
  - NVPerfAPI: Integrated in your application

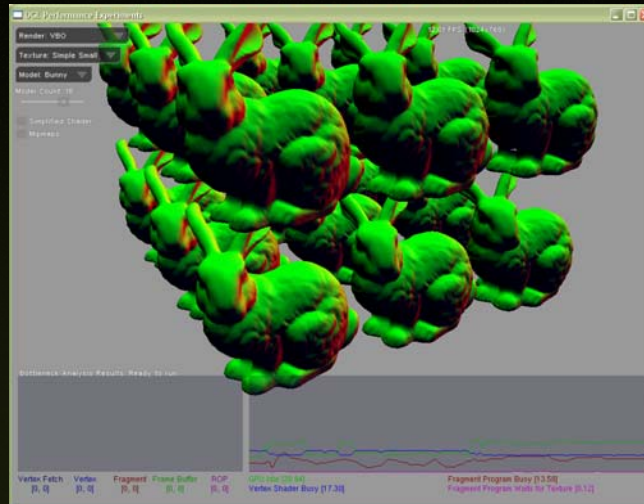
**Analyze your application like an NVIDIA Engineer!**



# NVPerfKit



- **NVPerfKit 2.0**
  - The package
  - Software/driver counters
  - GPU counters
  - Simplified Experiments
- **NVPerfHUD demo with Sim Dietrich**
- **Integrating NVPerfKit with NVPerfAPI**
- **Associated tools**





# What is in the NVPerfKit package?



**Instrumented Driver**

**GLExpert**

**NVPerfHUD**

**NVPerfSDK**

**NVPerfAPI**

**Sample Code**

**Helper Classes**

**Documentation**

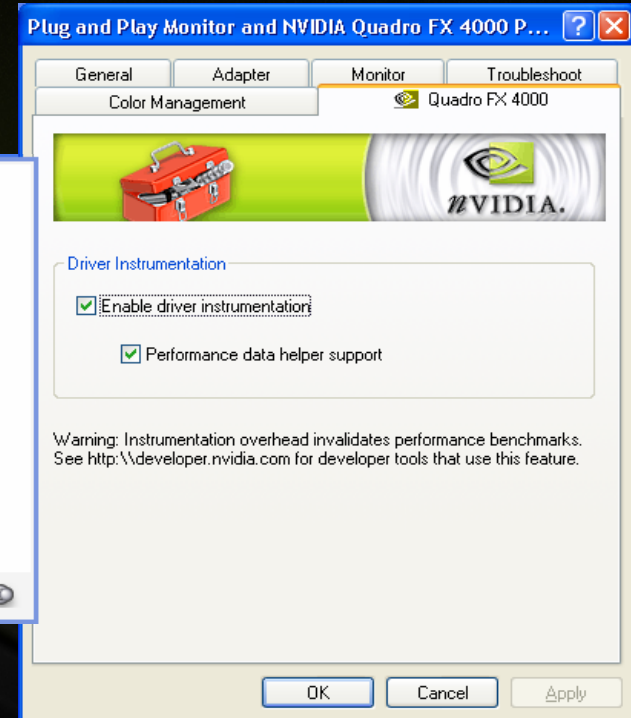
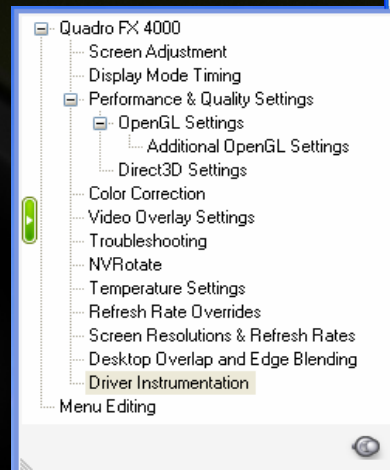
**Tools**

**NVIDIA Plug-In for**

**Microsoft PIX for Windows**

**gDEDebugger**

**NVDevCPL**





# NVPerfKit Instrumented Driver



- **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%**



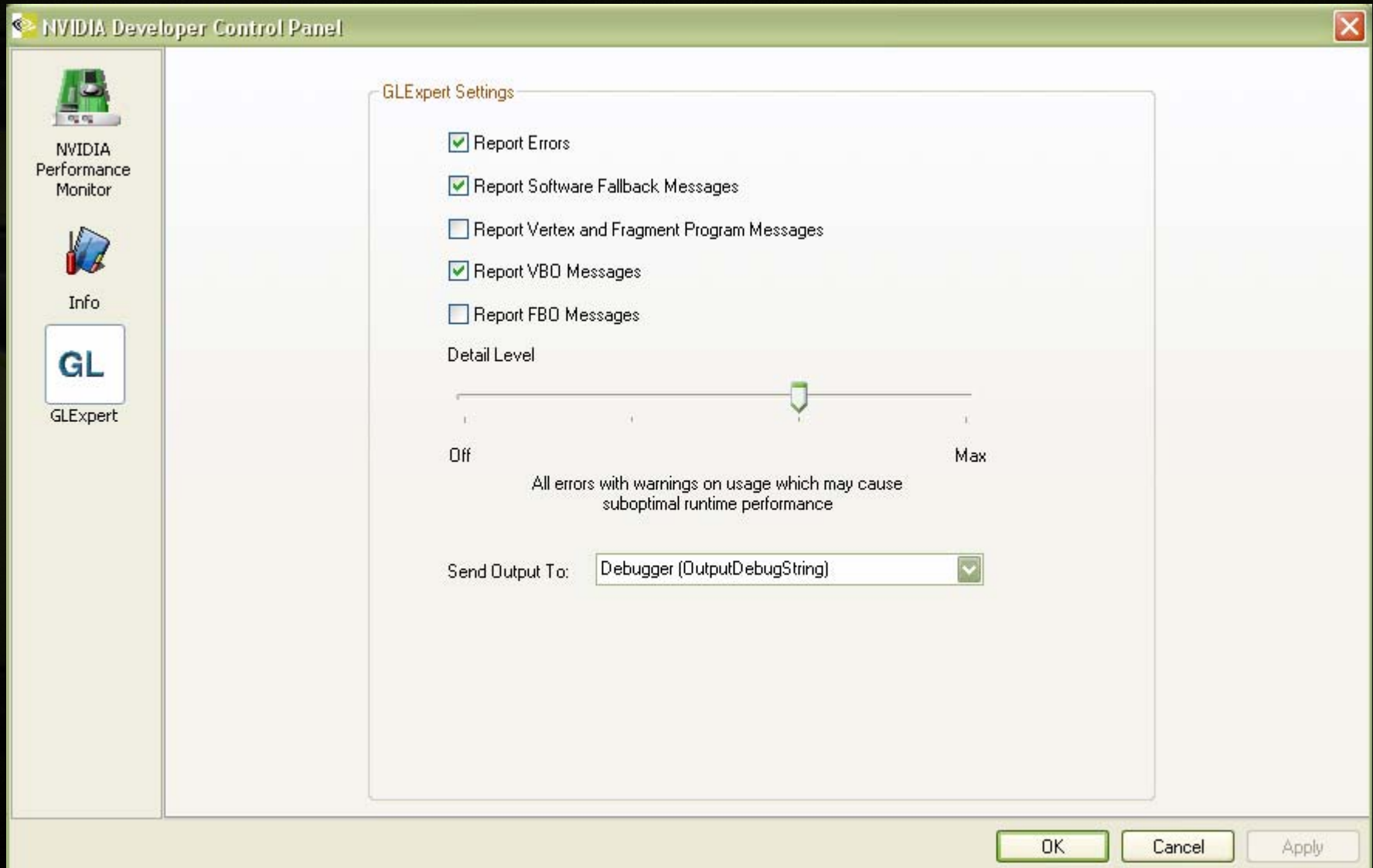
# GLExpert: What is it?



- **Helps eliminate performance issues on the CPU**
- **OpenGL portion of the Instrumented Driver**
  - Output information to console/stdout or debugger
  - Different groups and levels of information detail
- **Controlled using tab in NVDevCPL**
- **What it can do (today)**
  - GL Errors: print when they are 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 a configuration is unsupported
- **Feature list to grow with future drivers**



# GLExpert: NVDevCPL tab





# Project Status



- Shipping with NVPerfKit 2.0
- Windows for now, Linux to follow
- Supports NV3x, NV4x, and G7x architectures
- Integrated into Graphic Remedy's gDEDebugger
- What types of things are interesting?

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



# NVPerfKit: Direct3D Counters



## ● General

- FPS
- ms per frame

## ● Driver

- Driver frame time (total time spent in driver)
- Driver sleep time (waiting for GPU)

## ● Counts

- Triangles
- Instanced triangle
- Batches
- Locked render targets

## ● Memory

- AGP memory used in MB and bytes
- Video memory used and total in MB and bytes



# NVPerfKit: OpenGL 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

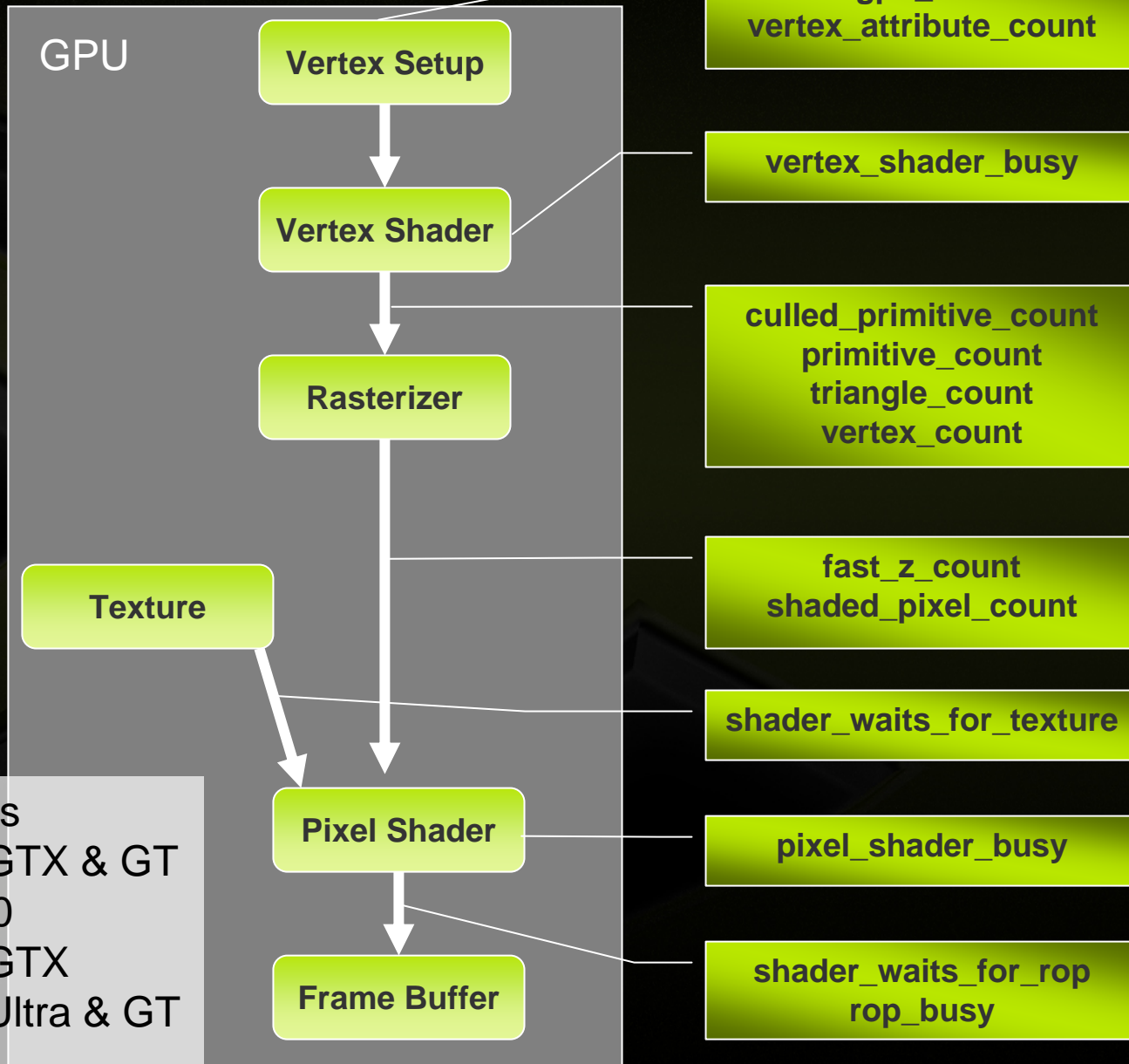
- Batches
- Vertices
- Primitives

## ● Memory

- AGP memory used in MB and bytes
- Video memory used and total in MB and bytes



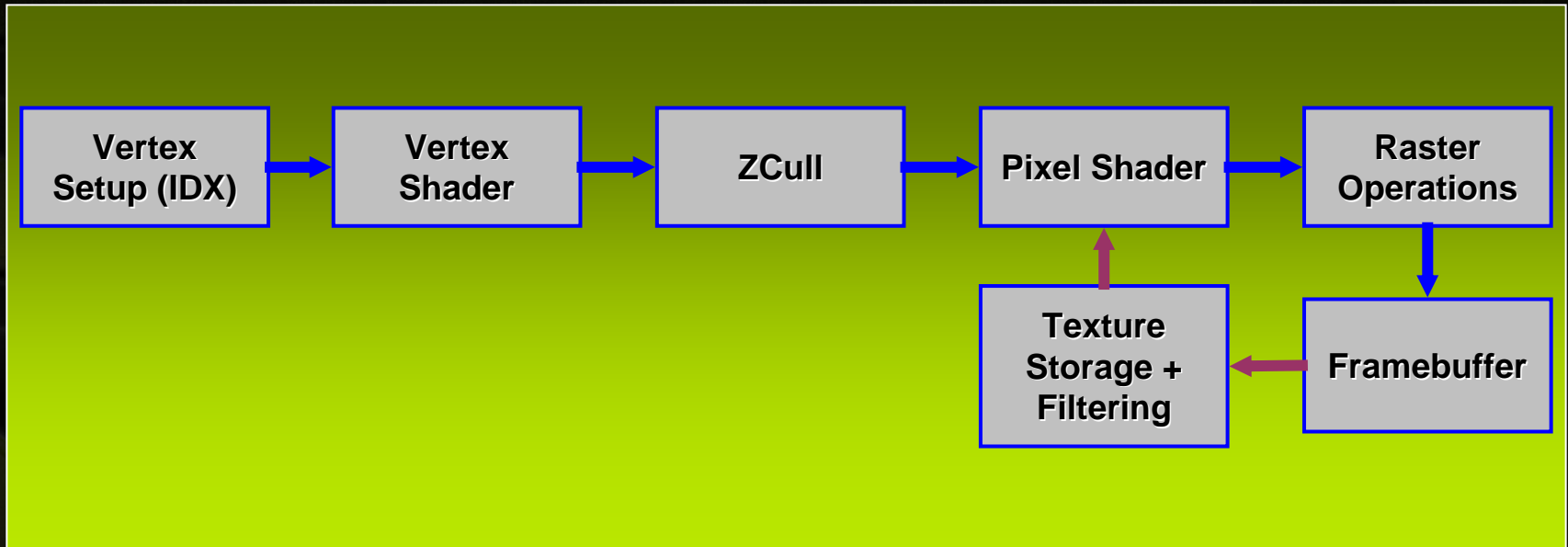
# NVPerfKit: GPU Counters



Supported GPUs  
GeForce 7900 GTX & GT  
Quadro FX 4500  
GeForce 7800 GTX  
GeForce 6800 Ultra & GT  
GeForce 6600



# NEW! Simplified Experiments



- Utilization and bottleneck experiments for each unit
- GPU Bottleneck experiment
  - Adds all bottleneck and utilization experiments
  - Expert system analyzes the results
- Exposed via NVPerfAPI



# What is NVPerfHUD?



- **Direct3D PERFormance Heads Up Display**
  - Overlay graphs and debugging tools on top of your application
  - Interactive tools for debugging and performance tuning
- **4 different HUDs**
  - 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



**Welcome Sim Dietrich!!**

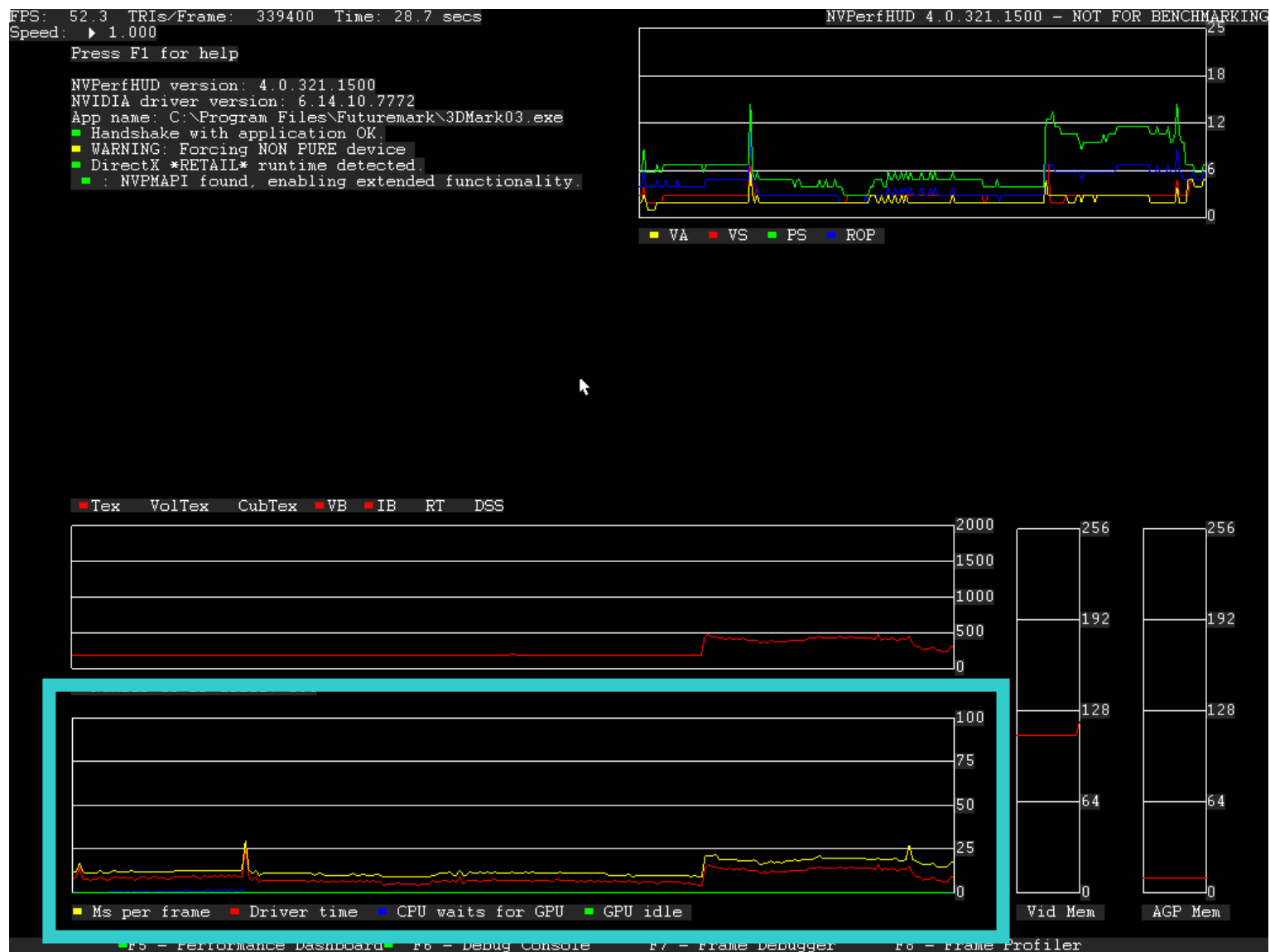


# Demo: Performance Dashboard



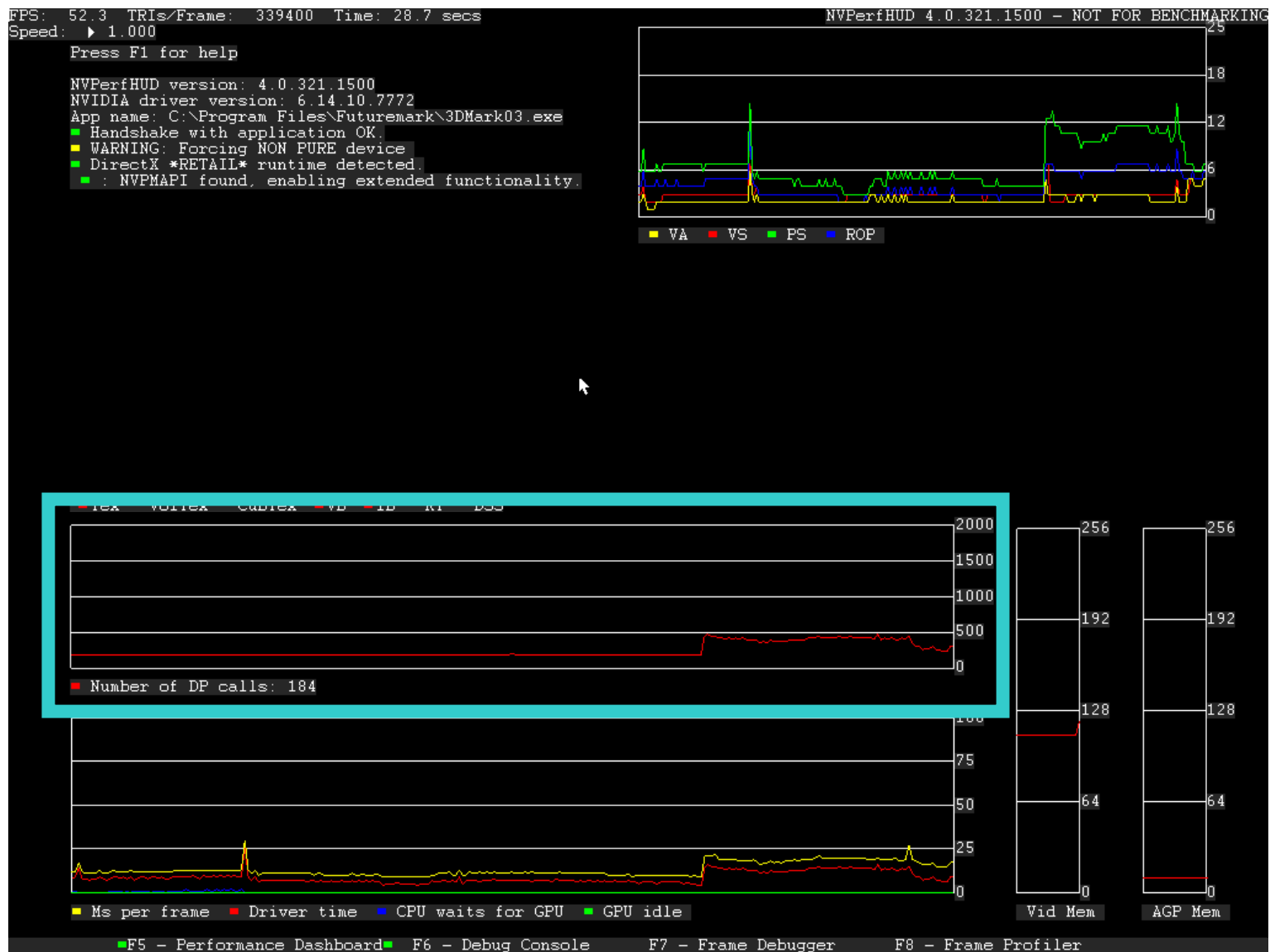


# Demo: Performance Dashboard



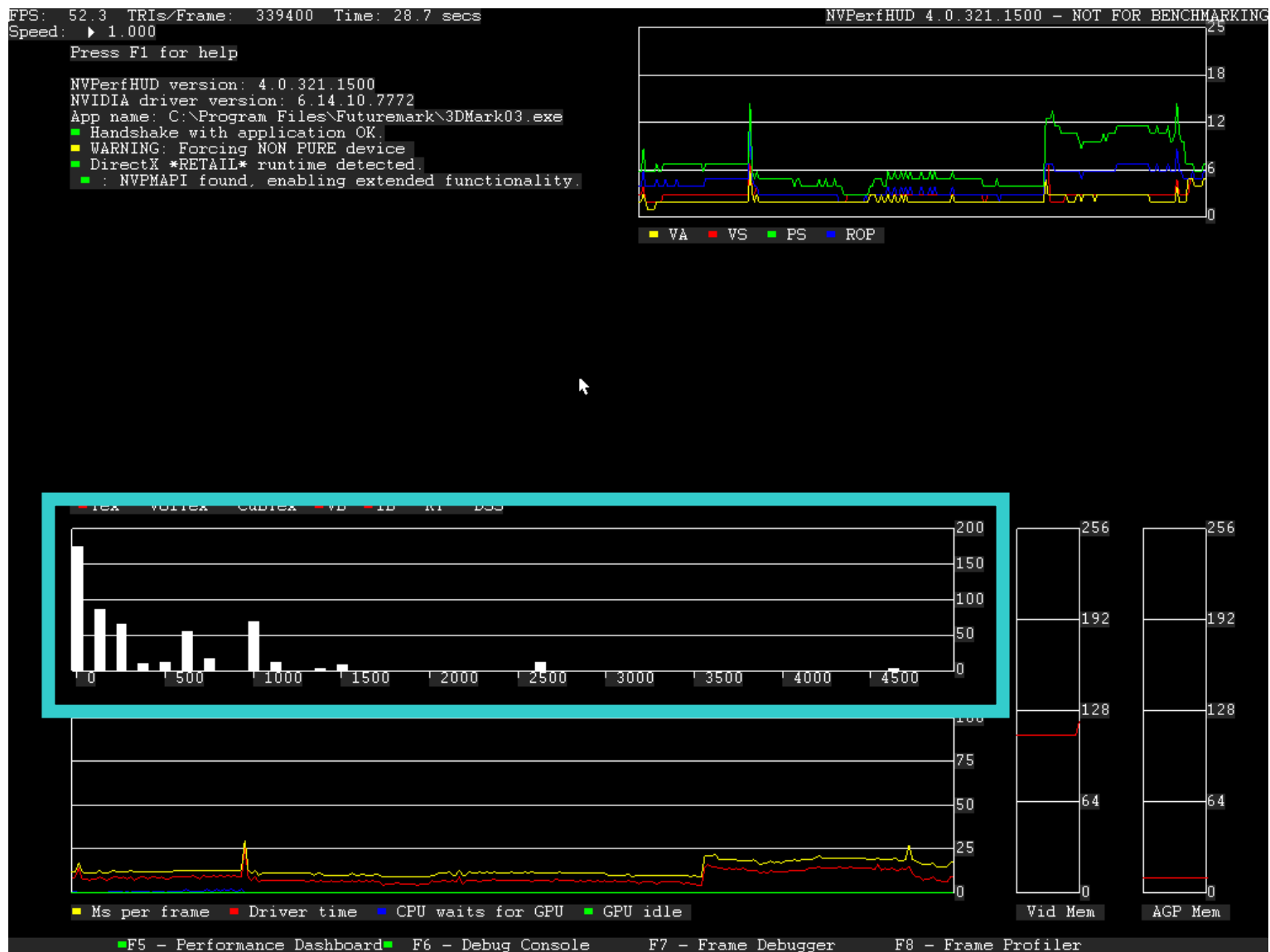


# Demo: Performance Dashboard



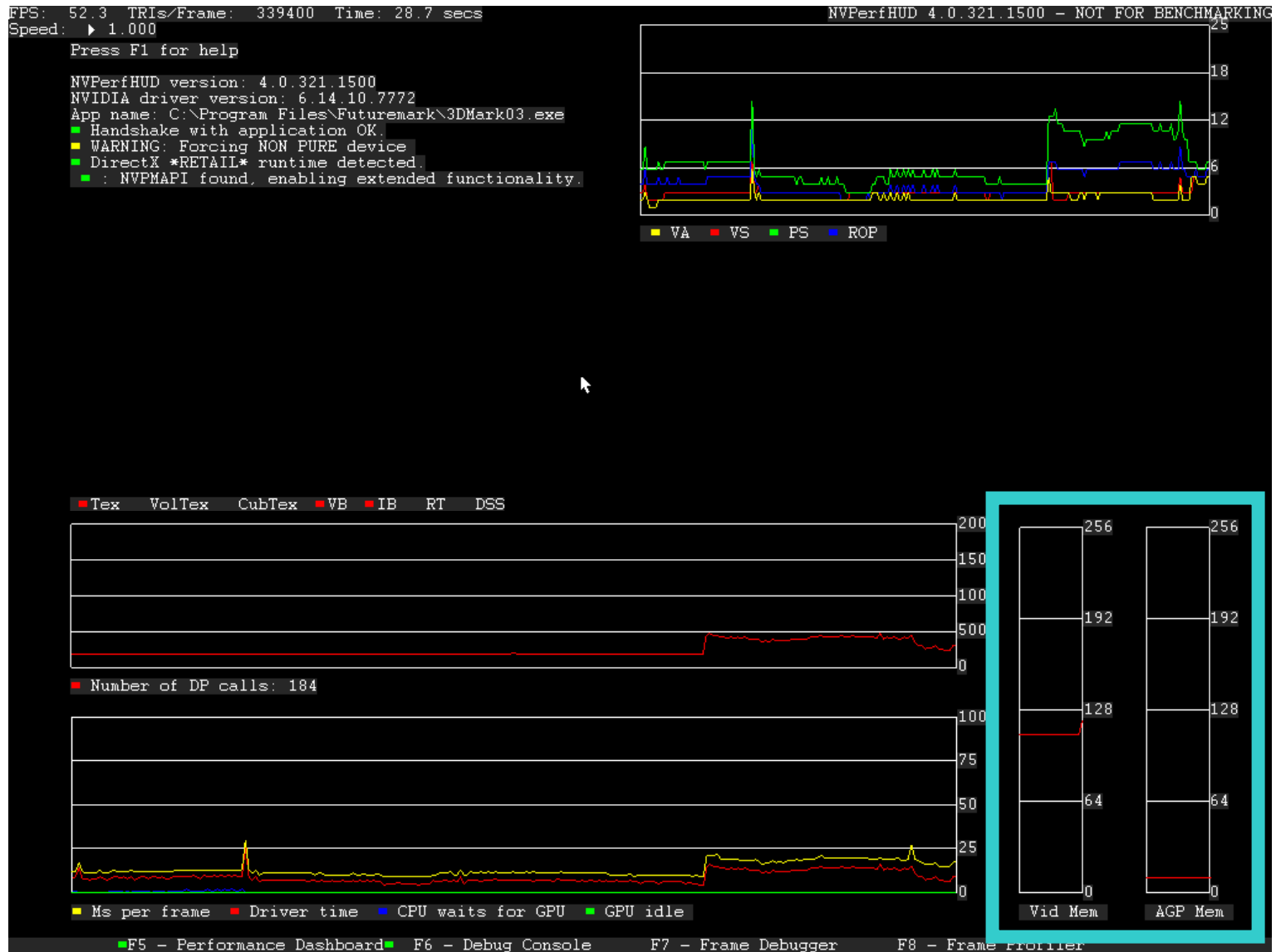


# Demo: Performance Dashboard





# Demo: Performance Dashboard





# Demo: Performance Dashboard

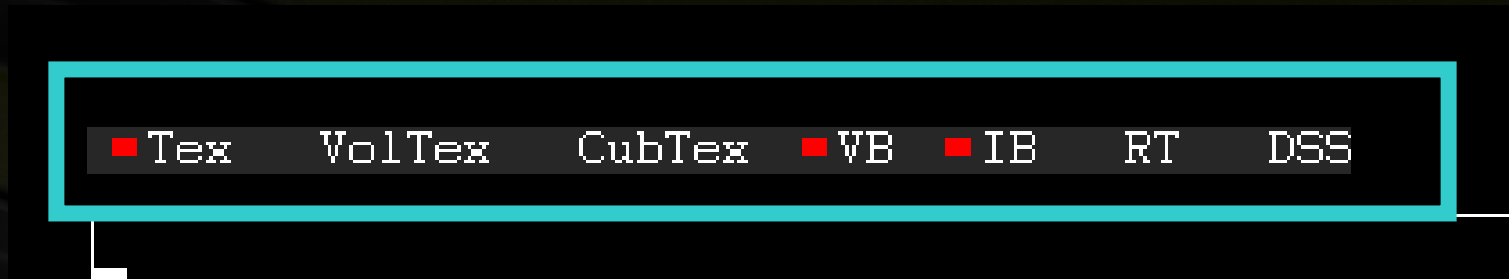




# Demo: Performance Dashboard



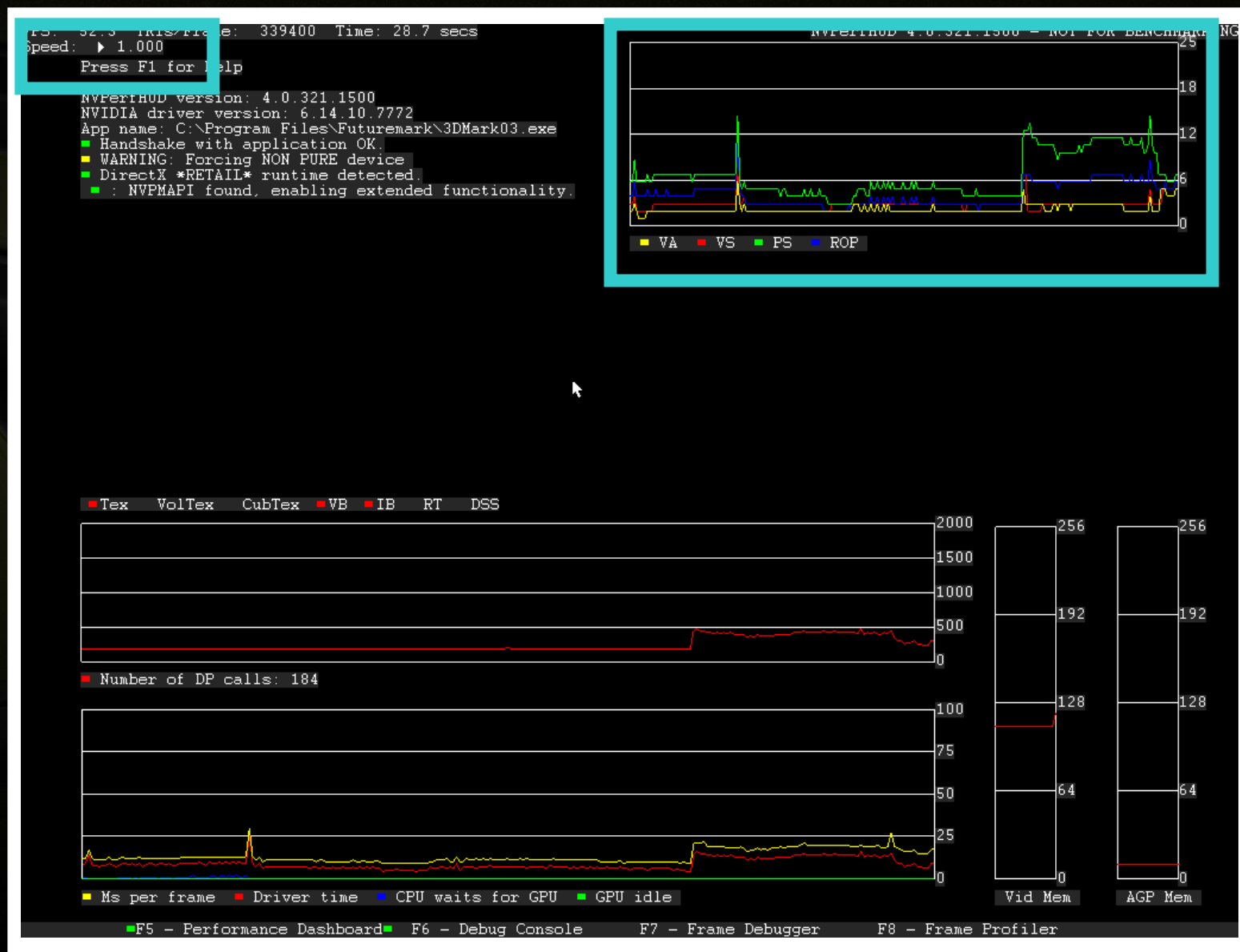
Resource creation monitor



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



# Demo: Performance Dashboard





# Demo: Performance Dashboard

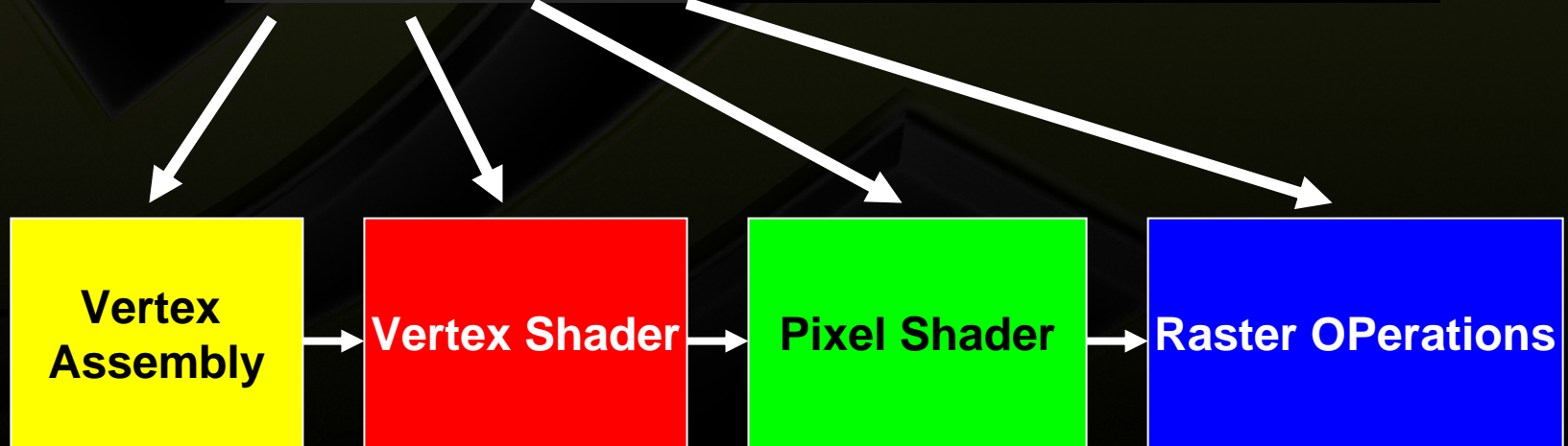
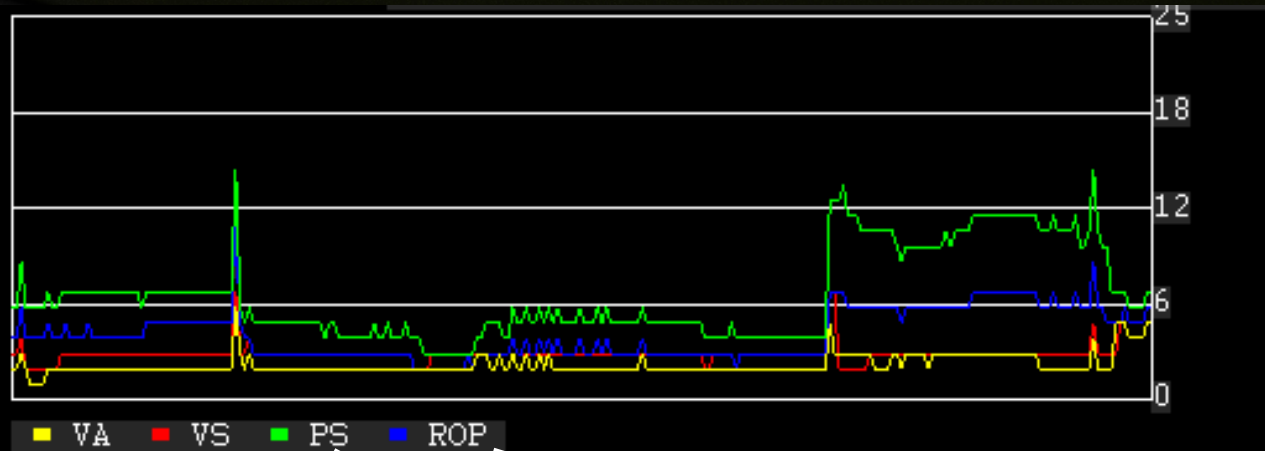


Speed control

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



# Demo: The simplified graphics pipeline





# Demo: Debug Console



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

Time: 7.03 secs, IDirect3DDevice9::CreateVertexBuffer(1600,8,0,1)  
Time: 7.03 secs, IDirect3DDevice9::CreateIndexBuffer(348,8,101,1)  
Time: 7.03 secs, IDirect3DDevice9::CreateVertexBuffer(128,8,0,1)  
Time: 7.03 secs, IDirect3DDevice9::CreateIndexBuffer(12,8,101,1)  
Time: 7.53 secs, IDirect3DDevice9::CreateVertexBuffer(5952,8,0,1)  
Time: 7.53 secs, IDirect3DDevice9::CreateIndexBuffer(1236,8,101,1)  
Time: 44.66 secs, IDirect3DDevice9::CreateOffscreenPlainSurface()  
Time: 70.00 secs, IDirect3DDevice9::CreateVertexBuffer(65536,520,0,0)  
Time: 70.00 secs, IDirect3DDevice9::CreateVertexBuffer(393216,520,0,0)  
Time: 70.01 secs, IDirect3DDevice9::CreateIndexBuffer(49152,8,101,1)  
Time: 70.01 secs, IDirect3DDevice9::CreateVertexBuffer(98304,520,0,0)  
Time: 70.01 secs, IDirect3DDevice9::CreateIndexBuffer(24576,8,101,1)  
Time: 70.01 secs, IDirect3DDevice9::CreateVertexBuffer(80,8,0,1)  
Time: 70.17 secs, IDirect3DDevice9::CreateTexture(1024x1024,1,0,22,1)  
Time: 70.18 secs, IDirect3DDevice9::CreateTexture(512x64,1,0,22,1)  
Time: 70.18 secs, IDirect3DDevice9::CreateVertexBuffer(1280,520,0,0)

☐ Clear Log Each Frame  
☐ Stop Logging  
☐ Fade Console

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



# Demo: Frame Debugger



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

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

Nothing

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

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

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



# Demo: Advanced Frame Debug



FPS: 69.3 This Frame: 00410 Time: 11.2 secs RYEngine 4.0.321.1500 - NOT FOR BENCHMARKING

Vertex Assembly | Vertex Shader | **Pixel Shader** | Raster Operations

Vertex Shader

Index Buffer Description

Index Buffer Bounding Box

Step Back Step Forward Draw Call 32/59

FS - Performance Dashboard FS - Debug Console F7 - Frame Debugger F8 - Frame Profiler

FPS: 69.4 This Frame: 00410 Time: 12.0 secs RYEngine 4.0.321.1500 - NOT FOR BENCHMARKING

Vertex Assembly | Vertex Shader | **Pixel Shader** | Raster Operations

Vertex Shader

Textures (x4 to zoom)

Vertex Shader Constants

Integer Constants

Boolean Constants

Step Back Step Forward Draw Call 32/59

FS - Performance Dashboard FS - Debug Console F7 - Frame Debugger F8 - Frame Profiler

FPS: 65.6 This Frame: 00410 Time: 25.4 secs RYEngine 4.0.321.1500 - NOT FOR BENCHMARKING

Vertex Assembly | Vertex Shader | **Pixel Shader** | Raster Operations

Pixel Shader

Textures (x4 to zoom)

Pixel Shader Constants

Integer Constants

Boolean Constants

Step Back Step Forward Draw Call 66/59

FS - Performance Dashboard FS - Debug Console F7 - Frame Debugger F8 - Frame Profiler

FPS: 69.6 This Frame: 00410 Time: 15.2 secs RYEngine 4.0.321.1500 - NOT FOR BENCHMARKING

Vertex Assembly | Vertex Shader | **Pixel Shader** | Raster Operations

Render Targets and Render States

Render Targets

Render States Dump

Step Back Step Forward Draw Call 32/59

FS - Performance Dashboard FS - Debug Console F7 - Frame Debugger F8 - Frame Profiler



# 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...**
- 
- **NEED TO ADD INFO ABOUT THE GRAPHS**



# Demo: Frame Profiler measuring





# Demo: Frame Profiler





# Demo: Frame Profiler





# Demo: Advanced Frame Profiler



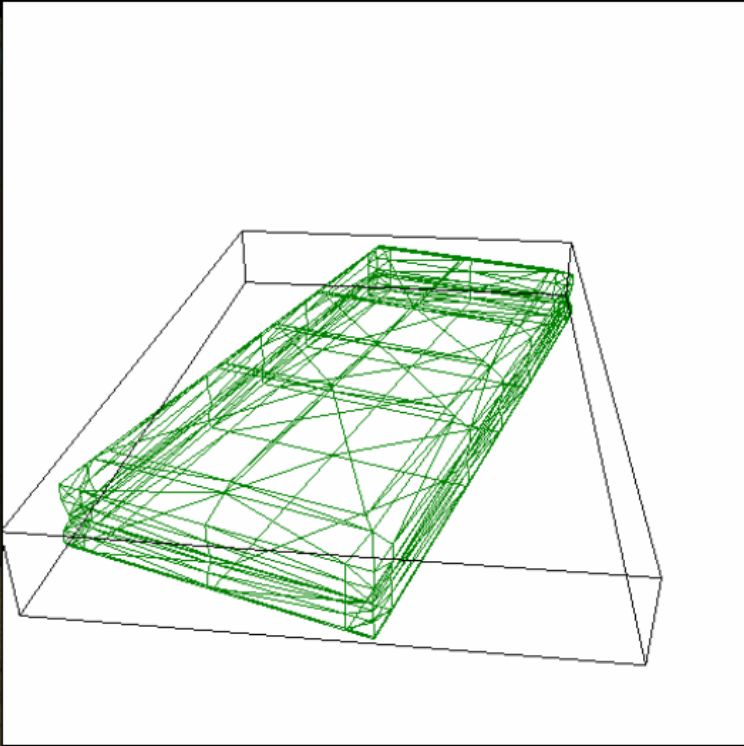
FPS: 59.1 TRIs/Frame: 88410 Time: 39.8 secs NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING

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

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

Vertex Assembly Vertex Shader Pixel Shader Raster Operations

Wireframe



Index / Vertex Buffer

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

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

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



# Freezing the application



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



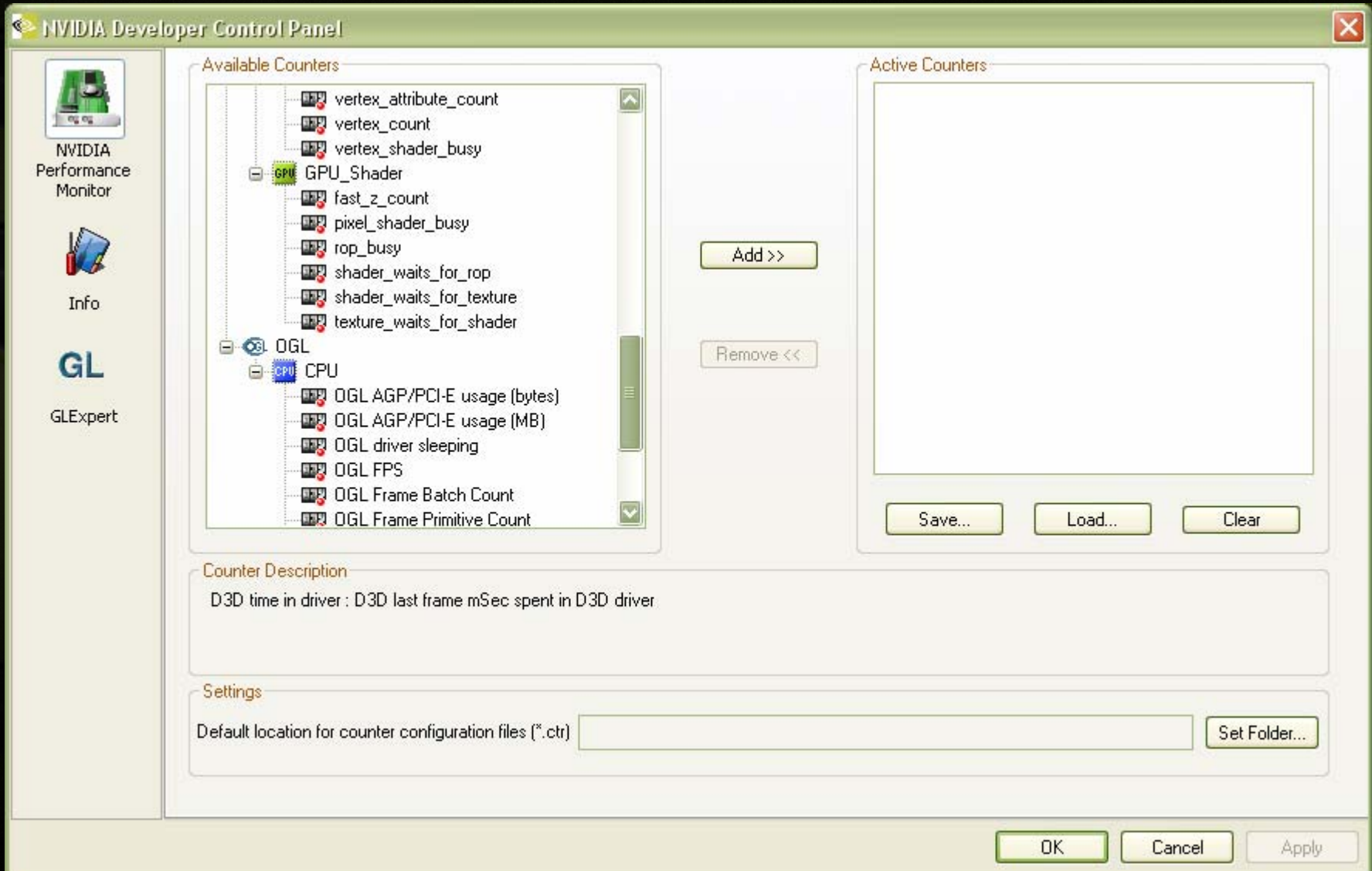
# How do I use NVPerfKit counters?



- **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**

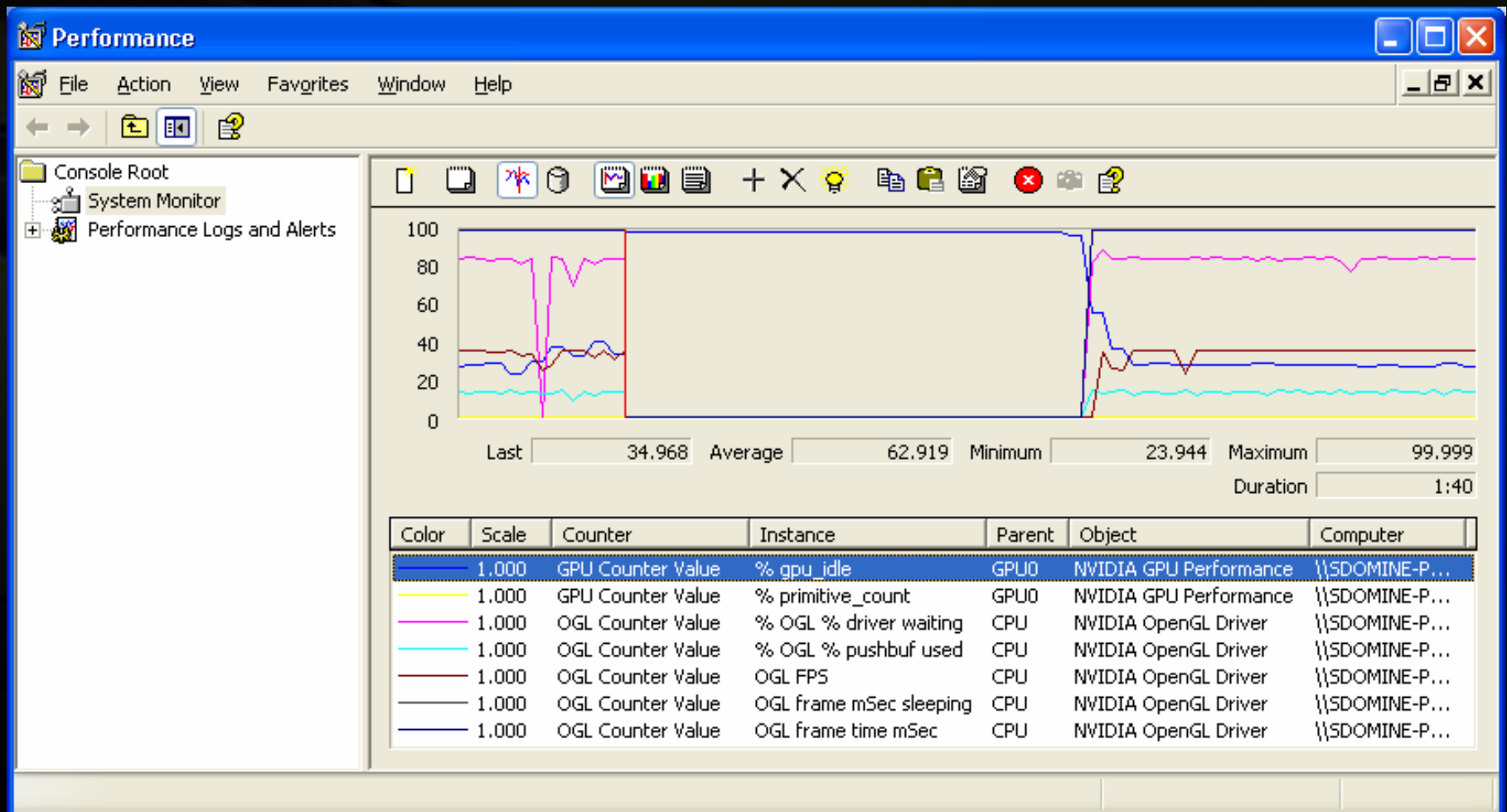


# Enable counters: NVDevCPL





# Graphing results: Perfmon





# NEW! 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

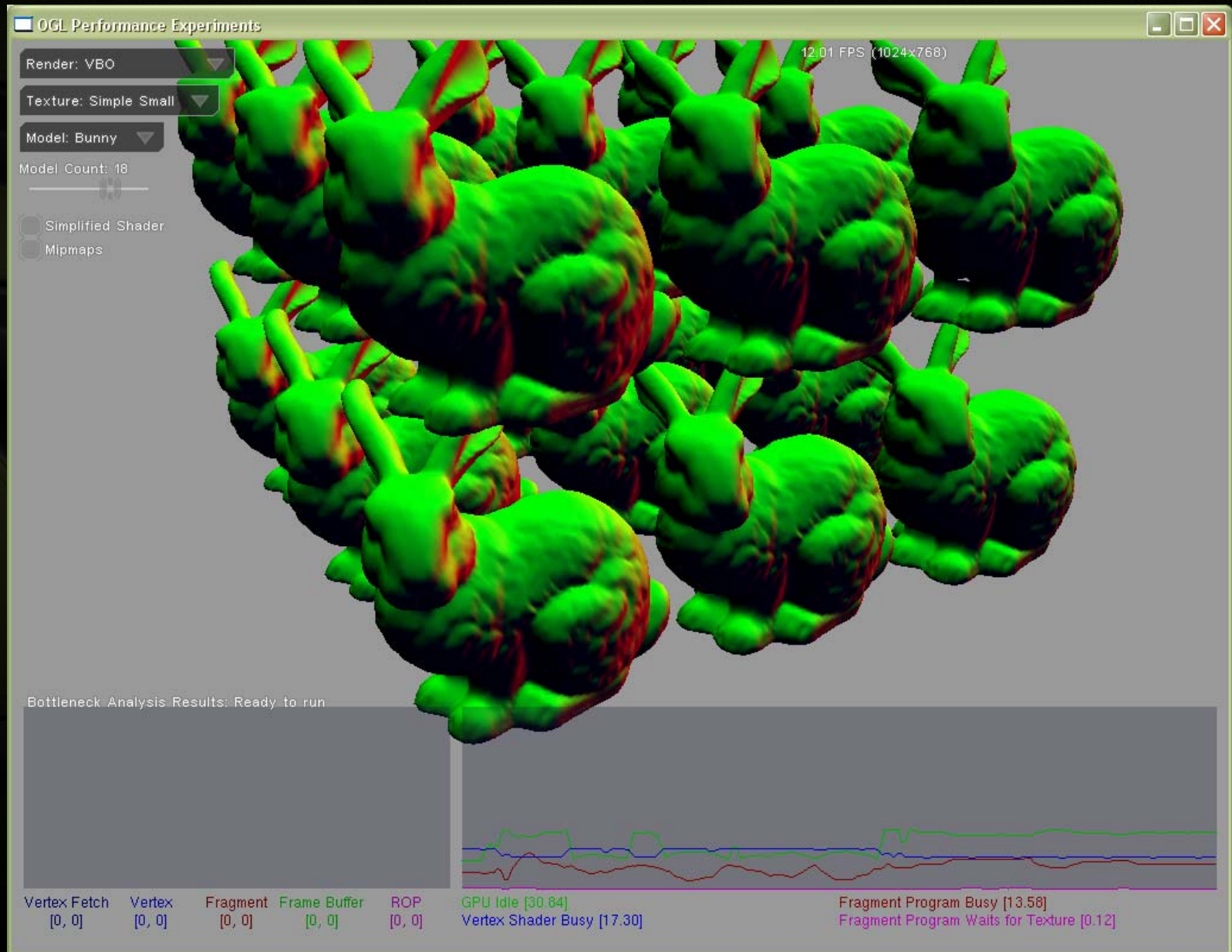


```
// Somewhere in setup
nVSBusy = NVPMGetCounterByName("vertex_shader_busy");
NVPMAddCounter(nVSBusy);
nPSBusy = NVPMGetCounterByName("pixel_shader_busy");
NVPMAddCounter(nPSBusy);
nWaitTexture = NVPMGetCounterByName("shader_waits_for_texture");
NVPMAddCounter(nWaitTexture);
nGPUIdle = NVPMGetCounterByName("gpu_idle");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

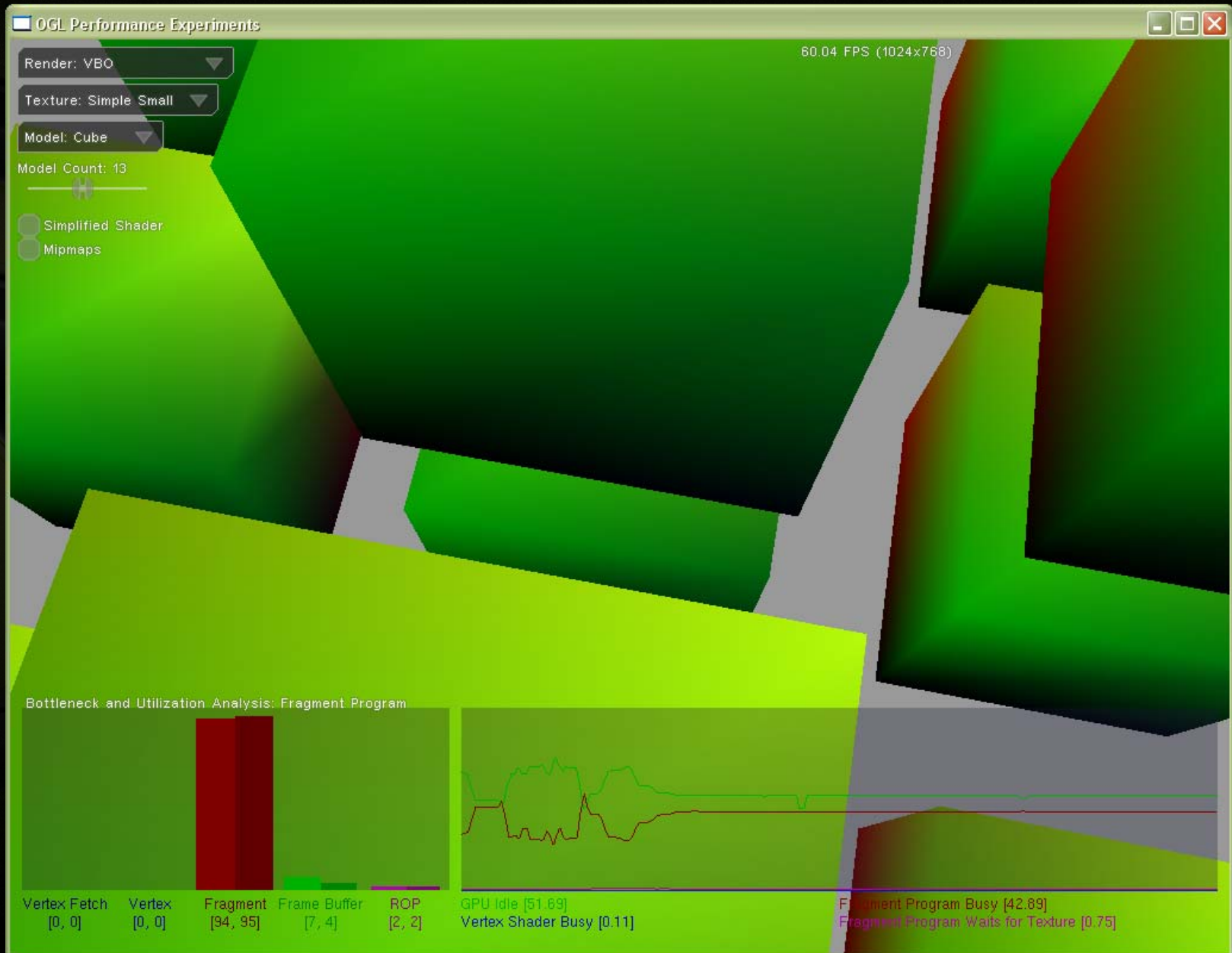


- **GPU Bottleneck experiment**
  - Run bottleneck and utilization experiments on all units
  - Process results to find bottlenecked unit
- Individual unit information can be queried
- Can run individual unit experiments
- Events: % utilization or % bottleneck...best way to visualize data
- Cycles: microseconds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

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

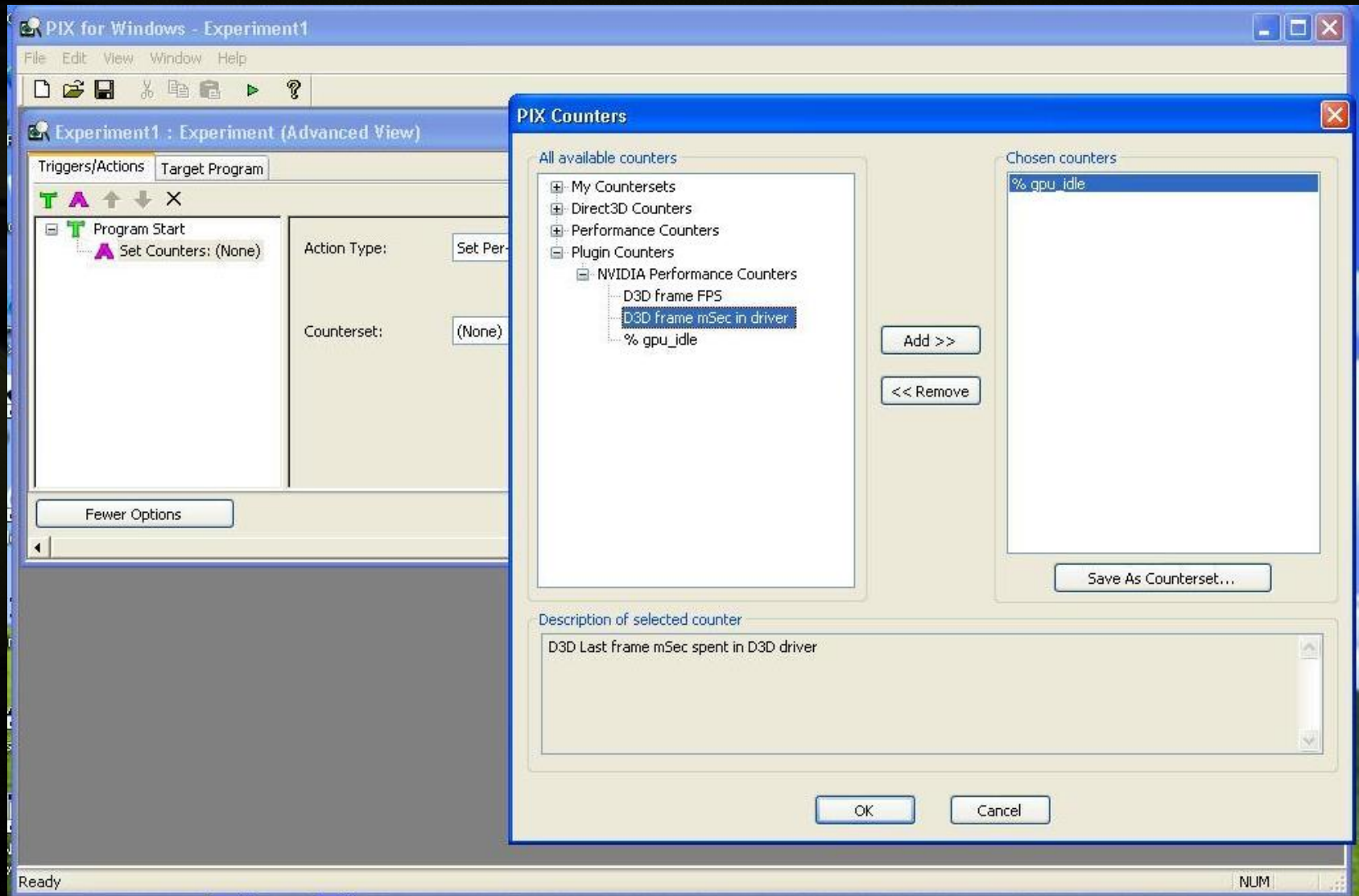


# NVPerfAPI: SimExp



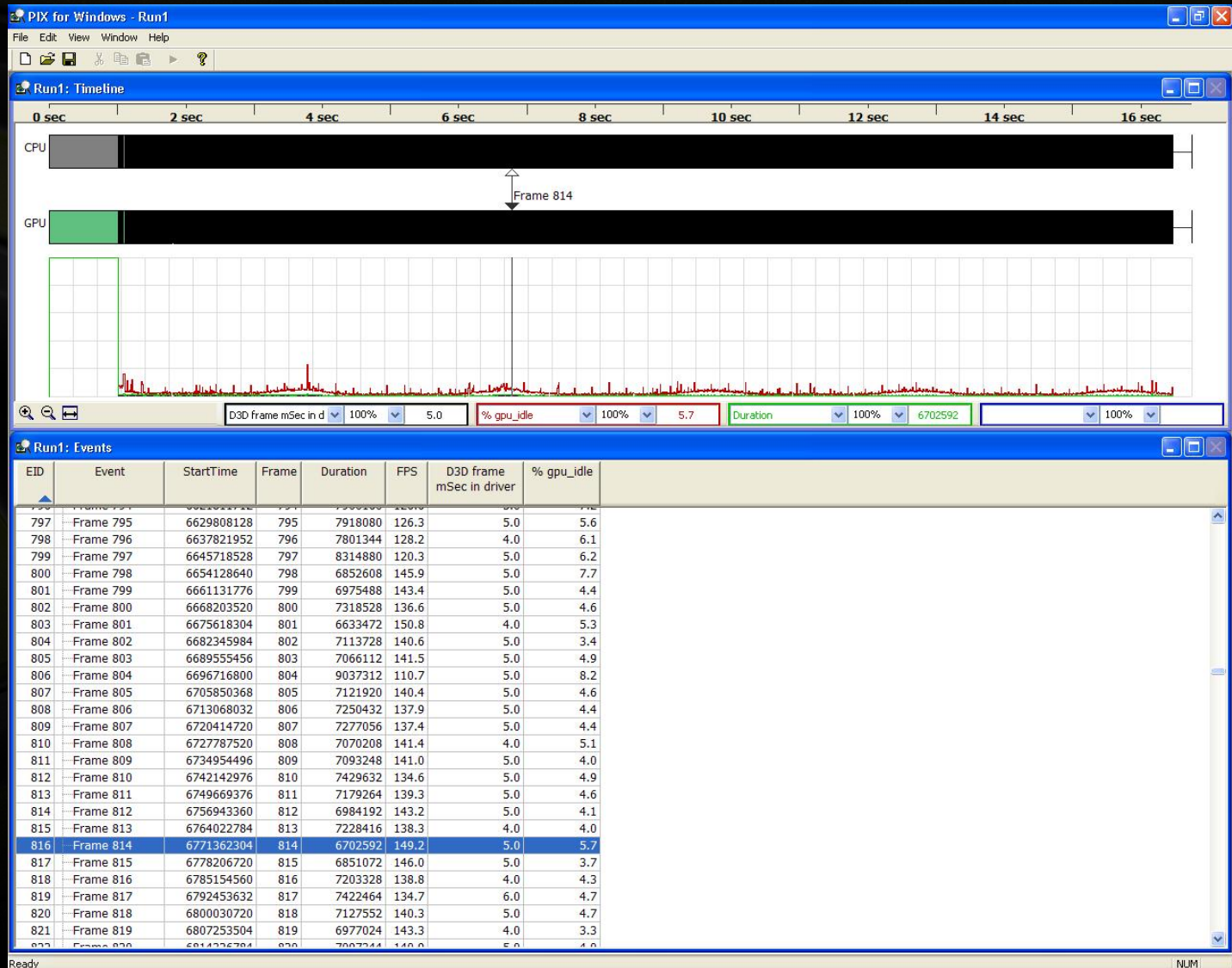


# Associated Tools: NVIDIA Plug-In for Microsoft PIX for Windows





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





# Graphic Remedy's gDEDebugger



The image displays the gDEDebugger interface for the GRTeaPot application. The main window shows a 3D rendered scene of a landscape with hills, a body of water, and a sky with clouds. The interface includes several panels for debugging and performance monitoring.

**FPS Monitor:**

- Triangle count: 181328
- Visible Cells: 32%
- Current FPS: 35

**Alpha, Visibility, Terrain, Sky:**

- Alpha Reference: 0.25
- Alpha Booster: 1.50
- Transparency AA: ☒

**OpenGL Function Calls History:**

- Context 1 - 23 OpenGL function calls
- glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL)
- glUseProgramObjectARB(3)
- glUniform1fARB(0, 0.70)
- glStringMarkerGREMEDY(Drawing scene objects)
- glBindTexture(GL\_TEXTURE\_2D, 6)
- glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, ...)

**OpenGL State Variables:**

OpenGL State Variable Name	Value
GL_VIEWPORT	(0, 0, 400, 400)
GL_PROJECTION_MATRIX	(2.00, 0.00, 0.00, 0.00)(0.00, ...)
GL_MODELVIEW_MATRIX	(1.00, 0.00, 0.00, 0.00)(0.00, ...)

**Calls Stack:**

- tpDrawScene - grteapotapplication.cpp, line 1206
- tpPaintWindow - grteapotapplication.cpp, line 1309
- tpTimerProc - grteapotapplication.cpp, line 1429
- GetDC - USER32.dll
- IsChild - USER32.dll
- DispatchMessageA - USER32.dll

**Properties:**

**Function name:** tpDrawScene  
**File path:** c:\program files\graphic remedy\gdebugger\examples\teapot\src\grteapotapplication  
**Line number:** 1206  
**Module name:** c:\program files\graphic remedy\gdebugger\examples\teapot\GRTeaPot.exe

**Performance Graph:**

Counter Name | Value

Counter Name	Value
Frames/sec: Context 1	64
CPU 0 Utilization	5
GPU0: % vertex_shader_busy	0
GPU0: % gpu_idle	92
GPU0: vertex_count	0

**Performance Dashboard:**

100 100 100 100

64 5 0 92

Fra... CPU... GPU... GPU... GPU...

**Function Calls Statistics:**

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



# Solutions to common bottlenecks



## ● CPU Bound?

### ● In your code:

- VTune...VTune...VTune... Don't assume!
- **LOD all calculations: Physics, animation, AI, you name it!**

### ● In driver code:

- **Create all resources up front: textures, VBs, IBs, shaders**
- **Reduce locking resources on the fly (discard VBs/IBs, don't write to a surface the GPU is reading from)**
- Create bigger batches: texture atlas, stitch strips together with degenerates
- Vertex shader constants = lookup table for matrices
- Instancing

### ● Transferring data to GPU

- Smallest vertex format possible
  - Remove unnecessary data
  - Use smallest data type possible
- Derive attributes in vertex shader
- 16 bit indices



# Solutions to common bottlenecks



- **IDX Bound, Vertex Shader Bound?**
  - Reduce vertex attribute count
    - Compute some attributes
    - Combine attributes (2 2D tex coords per attribute)
  - Use geometry LOD
  - Move invariant calculations to the CPU
  - **Use indexed primitives, more cache friendly**
  - Don't do unnecessary matrix multiplies
  - Use vertex shader branching to bypass expensive calculations
  - **Use NVShaderPerf!**



# Solutions to common bottlenecks



## ● Pixel Shader Bound?

- **Render depth first (no color writes = 2X speed)**
- **Prebake complex math into textures**
- **Move per pixel calculations to the vertex shader**
- **Use partial precision where possible, try it you may like the result**
- **Avoid unnecessary normalizations**
- **Use LOD specific pixel shaders**
- **Use NVShaderPerf!**



# Solutions to common bottlenecks



## ● Texture bound?

- Prefilter textures to reduce size
- Mipmap on any texture/surface that might be minified
- Compressed textures
- Use float textures only when needed



# Solutions to common bottlenecks



- **Frame buffer bound?**
  - **Render depth first (no color writes = 2X speed)**
  - **Only use alpha blending when necessary**
  - **Use alpha test**
  - **Disable depth writes when possible**
  - **Avoid clearing the color buffer if you touch every pixel (but do clear Z)**
  - **Render front to back to get better z culling**
  - **Use float textures only when needed**



# NVShaderPerf



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



```

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

```

# NVShaderPerf



```

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

    // first rows are the tangent and binormal scaled by the bump scale
    TangentToObjSpace[0] = float3(IN.Tangent.x, IN.Binormal.x, IN.Normal.x);
    TangentToObjSpace[1] = float3(IN.Tangent.y, IN.Binormal.y, IN.Normal.y);
    TangentToObjSpace[2] = float3(IN.Tangent.z, IN.Binormal.z, IN.Normal.z);
    OUT.TexCoord1.x = dot(World[0].xyz, TangentToObjSpace[0]);
    OUT.TexCoord1.y = dot(World[1].xyz, TangentToObjSpace[0]);
    OUT.TexCoord1.z = dot(World[2].xyz, TangentToObjSpace[0]);
    OUT.TexCoord1.w = dot(World[3].xyz, TangentToObjSpace[0]);
    OUT.TexCoord2.x = dot(World[0].xyz, TangentToObjSpace[1]);
    OUT.TexCoord2.y = dot(World[1].xyz, TangentToObjSpace[1]);
    OUT.TexCoord2.z = dot(World[2].xyz, TangentToObjSpace[1]);
    OUT.TexCoord2.w = dot(World[3].xyz, TangentToObjSpace[1]);
    OUT.TexCoord3.x = dot(World[0].xyz, TangentToObjSpace[2]);
    OUT.TexCoord3.y = dot(World[1].xyz, TangentToObjSpace[2]);
    OUT.TexCoord3.z = dot(World[2].xyz, TangentToObjSpace[2]);
    OUT.TexCoord3.w = dot(World[3].xyz, TangentToObjSpace[2]);
    float4 worldPos = mul(IN.Position, World);
    // compute the eye vector (eye pos - obj pos) in cube space
    float4 eyeVec = mul(worldPos, ViewIT);
    OUT.TexCoord1.w = eyeVec.x;
    OUT.TexCoord2.w = eyeVec.y;
    OUT.TexCoord3.w = eyeVec.z;
    return OUT;
}

// ===== pixel shader =====
float4 BumpReflectPS(v2f IN,
    uniform sampler2D NormalMap,
    uniform samplerCUBE EnvironmentMap,
    uniform float BumpScale) : COLOR
{
    // fetch the bump normal from the normal map
    float3 normal = tex2D(NormalMap, IN.TexCoord.xy).xyz * 2.0 - 1.0;
    normal = normalize(float3(normal.x * BumpScale, normal.y * BumpScale, normal.z));
    // transform the bump normal into cube space
    // then use the transformed normal and eye vector to compute a reflection vector
    // used to fetch the cube map
    // (we multiply by 2 only to increase precision)
    float3 eyevec = float3(IN.TexCoord1.w, IN.TexCoord2.w, IN.TexCoord3.w);
    float3 worldNorm;
    worldNorm.x = dot(IN.TexCoord1.xyz, normal);
    worldNorm.y = dot(IN.TexCoord2.xyz, normal);
    worldNorm.z = dot(IN.TexCoord3.xyz, normal);
    float3 lookup = reflect(eyevec, worldNorm);
    return texCUBE(EnvironmentMap, lookup);
}

```

## Inputs

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

## NVShaderPerf

## GPU Arch:

- GeForce 7X00
- GeForce 6X00
- Geforce FX series
- Quadro FX series

C:\WINDOWS\system32\cmd.exe

```

dp3 r0.x, r1, r1
rsq r0.w, r0.x
nrm r0.xyz, t1
mad r1.xyz, r1, r0.w, r0
nrm r2.xyz, r1
nrm r1.xyz, t2
dp3 r2.x, r2, r1
max r1.w, r2.x, c9.x
pow r0.w, r1.w, c5.x
add r1.w, r0.w, -c7.x
mov r2.w, c6.x
add r2.w, r2.w, -c7.x
rcp r2.w, r2.w
mul_sat r2.w, r1.w, r2.w
mad r1.w, r2.w, c9.y, c9.z
mul r2.w, r2.w, r2.w
mul r1.w, r1.w, r2.w
mov r2.x, c9.w
add r2.w, r2.x, -c8.x
mad r1.w, r1.w, r2.w, c8.x
dp3 r0.x, r0, r1
mul r0.w, r0.w, r1.w
mul r1.xyz, r0.w, c4
add r0.w, r0.x, c9.w
mul r0.w, r0.w, c10.x
mov r0.xyz, c0
add r0.xyz, r0, -c1
mov r1.w, c9.w
add r1.w, r1.w, c9.w
mad r2.xyz, r0.w, r2, c3
mad r2.xyz, r0.w, r2, c3
add r1.w, r1.w, c9.w
mov r1.w, c9.w
mov r0.w, c9.w

```

## Outputs:

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

```

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

```



# NVShaderPerf: In your pipeline



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



# FX Composer 1.8 – Shader Perf



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

The screenshot displays two instances of the 'Shader Perf' window. The top window is for a 'Pixel Shader' on a 'GeForceFX 5200' target. It shows performance metrics for a shader with 51 cycles and 4 registers, achieving a pixel throughput of 15.69 MP/s. The bottom window is for a 'Pixel Shader' on a 'GeForce 6800 Ultra' target. It shows performance metrics for a shader with 21.00 cycles and 3 registers, achieving a pixel throughput of 304.76 MP/s. The bottom window also displays the disassembly of the shader, showing instructions like 'ps\_2\_0', 'def c9, 0, -', and 'def c10, 0, 0, 0'.

**Shader Perf**

Untextured p0 Pixel Shader GeForceFX 5200

\*\*\*\*\*

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, .

....

Property

**Shader Perf**

Untextured p0 Pixel Shader GeForce 6800 Ull

\*\*\*\*\*

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 8X.XX driver**
- **Better support for branching performance**
  - **Default computes maximum path through shader**
  - **Use `-minbranch` to compute minimum path**



# NVShaderPerf 1.8



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

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

```
float tea;
```

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

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

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

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

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

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

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

```
}
```

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

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

Eye Shader from Luna

Maximum branch takes 674 cycles

Minimum branch takes 193 cycles.

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



# NVShaderPerf – Version 2.0



- Vertex throughput
- GLSL vertex program
- Multiple driver versions from one NVShaderPerf
- Much smaller footprint
- New programmatic interface
- Integration into FX Composer 2.0
- What else do you need?

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



# Questions?



- **Developer tools DVDs available at our booth**
  - NVPerfKit 2.0
  - NVPerfHUD 4.0 Overview Video
  - NVPerfHUD 4.0 Quick Reference Card
  - User Guides
- **Online:**
  - <http://developer.nvidia.com/NVPerfKit>
  - <http://developer.nvidia.com/NVPerfHUD>

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

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

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

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



# NVIDIA Session Raffle



- **GPU Gems 2 books**

- Also available at the GDC store

- **7900GT Board**

- Fast!
- Cool, silent, 1-slot solution

- **Didn't win?**

- Swipe your card at the entrance to enter our post-GDC raffle!



# The Source for GPU Programming

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

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



**nVIDIA**

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

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

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



# NVIDIA SDK

## The Source for GPU Programming



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

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

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

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

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

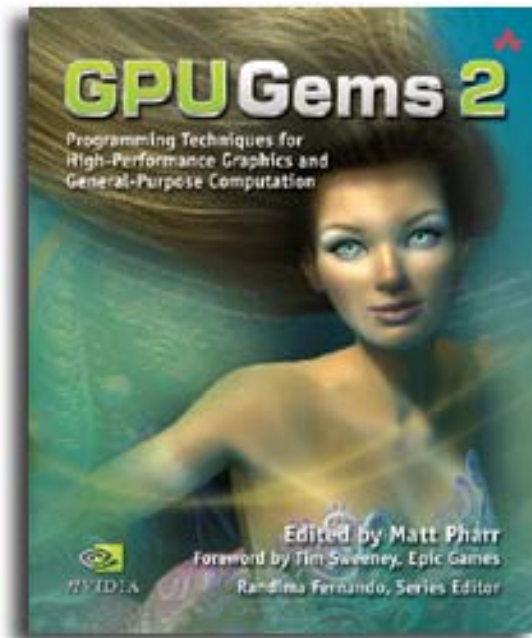




# GPU Gems 2

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

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



### Graphics Programming



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

### GPGPU Programming



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