



GPU Jackpot

Performance Tools and Performance Analysis Techniques

Sebastien Domine
Manager of Developer Technology Tools



Agenda

- Performance Tools Survey
- Performance Methodologies and Practice
- Next generation Performance Tools
- Conclusion
- Q & A



Agenda

- Performance Tools Survey
- Performance Methodologies and Practice
- Next generation Performance Tools
- Conclusion
- Q & A



Performance Tools Survey

- NVPerfHUD
 - Direct3D9 Performance HUD
- NVShaderPerf
 - Shader Performance Analysis
- FX Composer
 - HLSL Shader Editor IDE

NVPerfHUD 2.0



- Overlay graph that displays stats from :
 - Direct3D9 API interception layer
 - Direct3D Driver
 - Requires NVIDIA GPU
- Able to bypass and inject some API calls to assist with performance analysis
- Only works on your own application

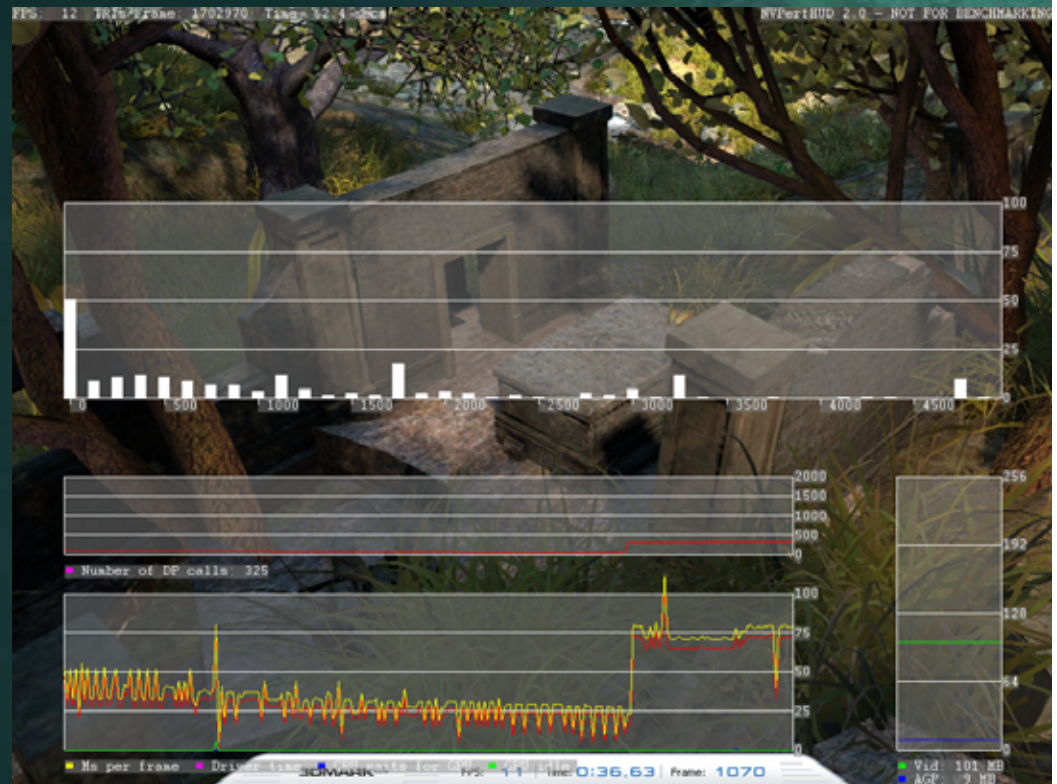
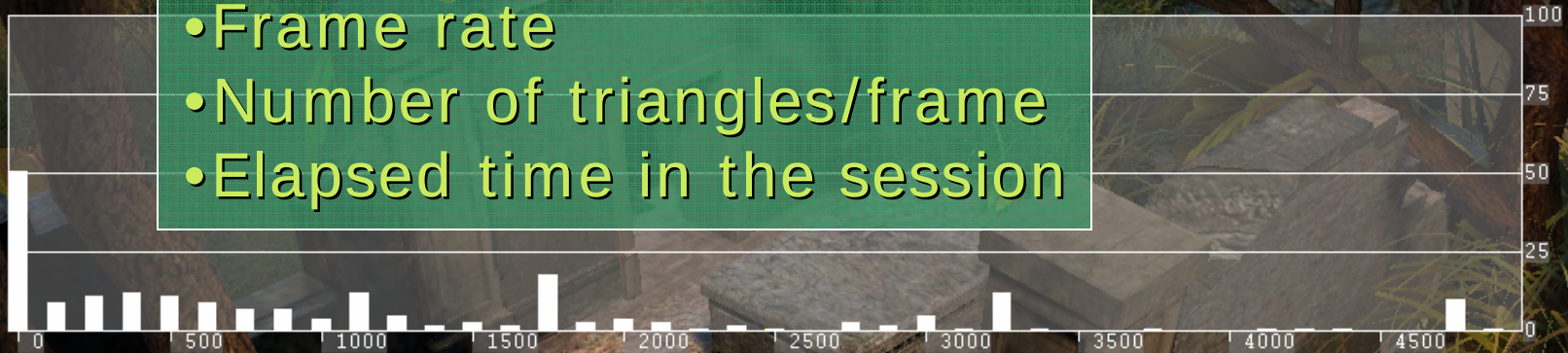


Image courtesy of FutureMark Corp.

FPS: 12 TRIs/Frame: 1702970 Time: 62.4 secs

NVPerfHUD 2.0 - NOT FOR BENCHMARKING

- Frame rate
- Number of triangles/frame
- Elapsed time in the session

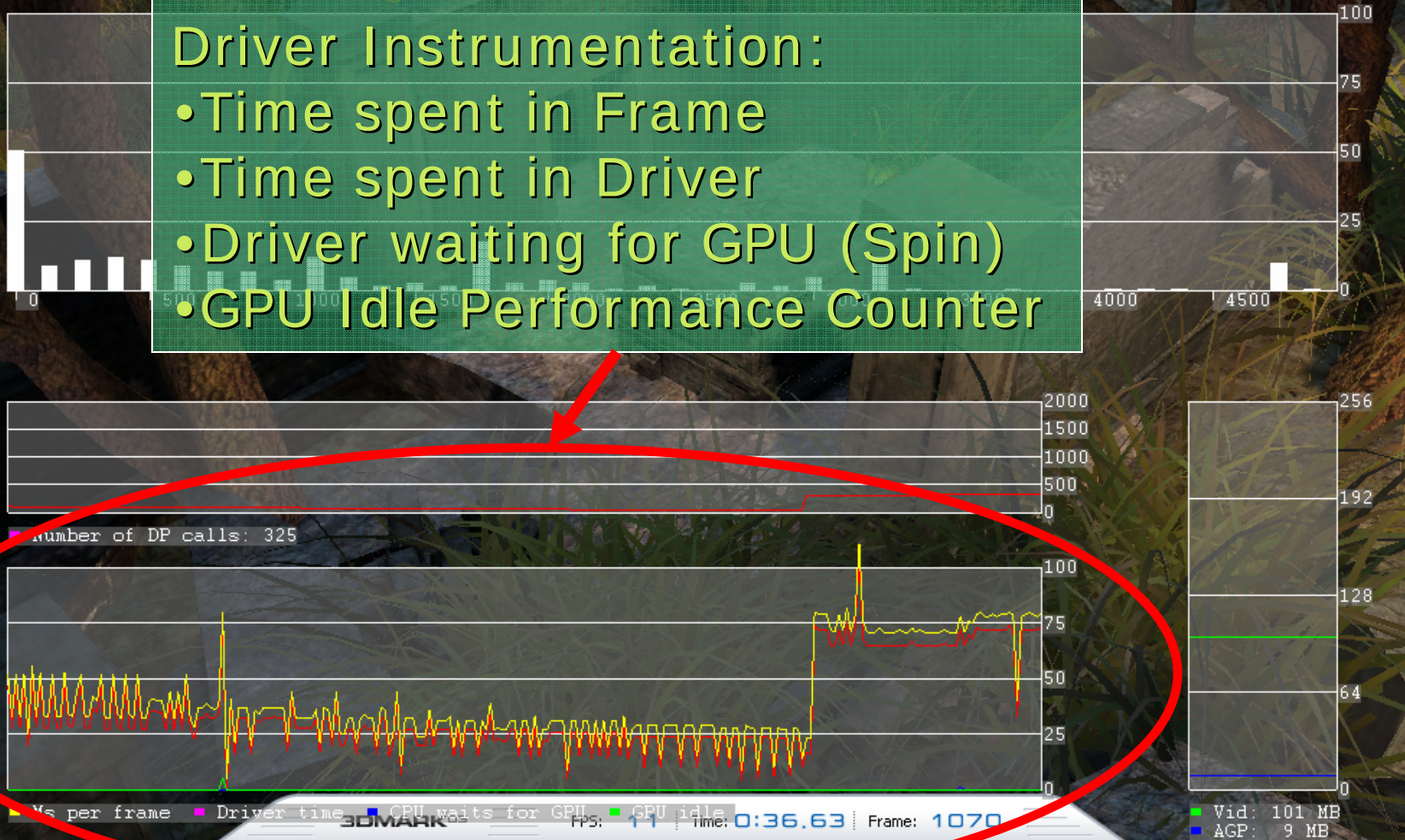


FPS: 12 TRIs/Frame: 1702970 Time: 62.4 secs

NVPerfHUD 2.0 - NOT FOR BENCHMARKING

Driver Instrumentation:

- Time spent in Frame
- Time spent in Driver
- Driver waiting for GPU (Spin)
- GPU Idle Performance Counter

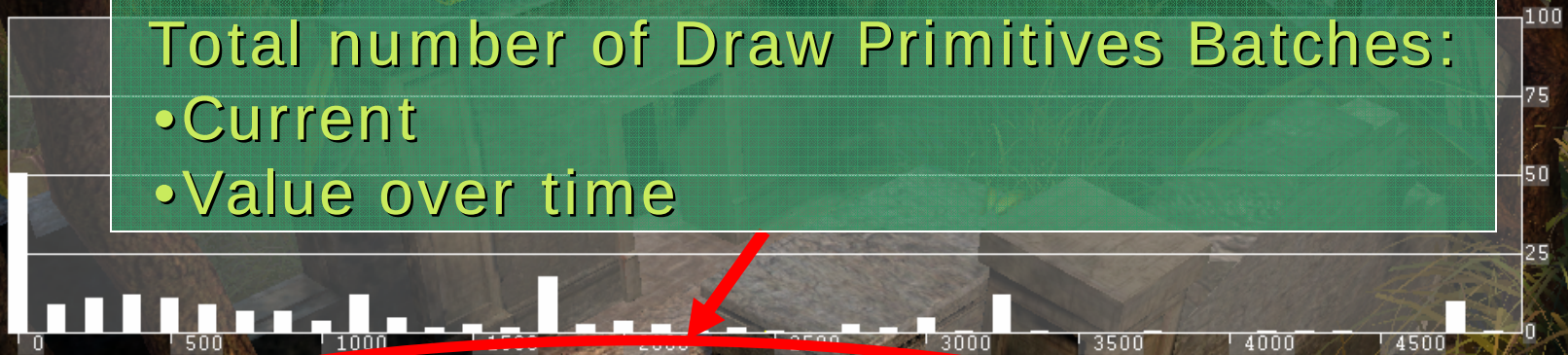


FPS: 12 TRIs/Frame: 1702970 Time: 62.4 secs

NVPerfHUD 2.0 - NOT FOR BENCHMARKING

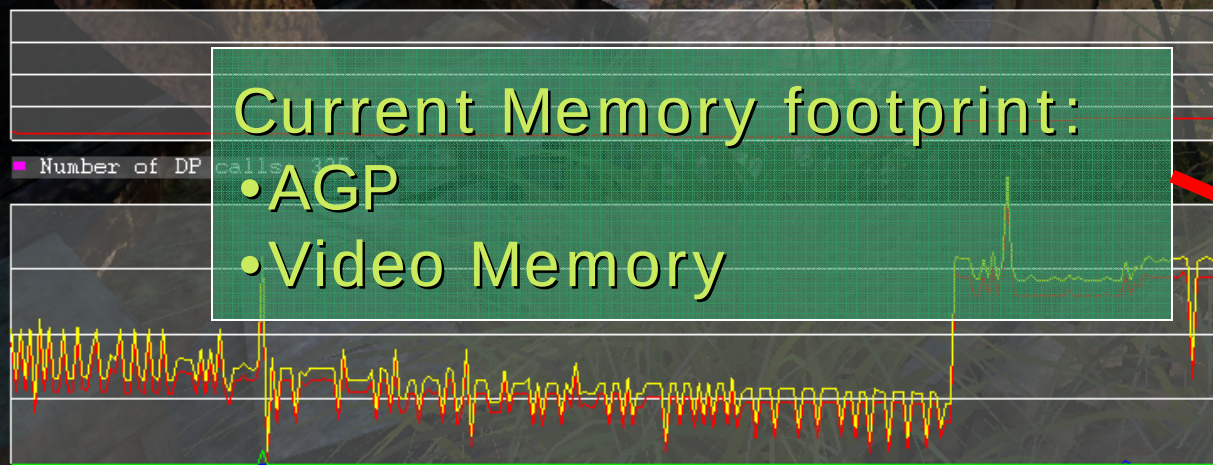
Total number of Draw Primitives Batches:

- Current
- Value over time



FPS: 12 TRIs/Frame: 1702970 Time: 62.4 secs

NVPerfHUD 2.0 - NOT FOR BENCHMARKING



3DMARK FPS: 11 Time: 0:36.63 Frame: 1070



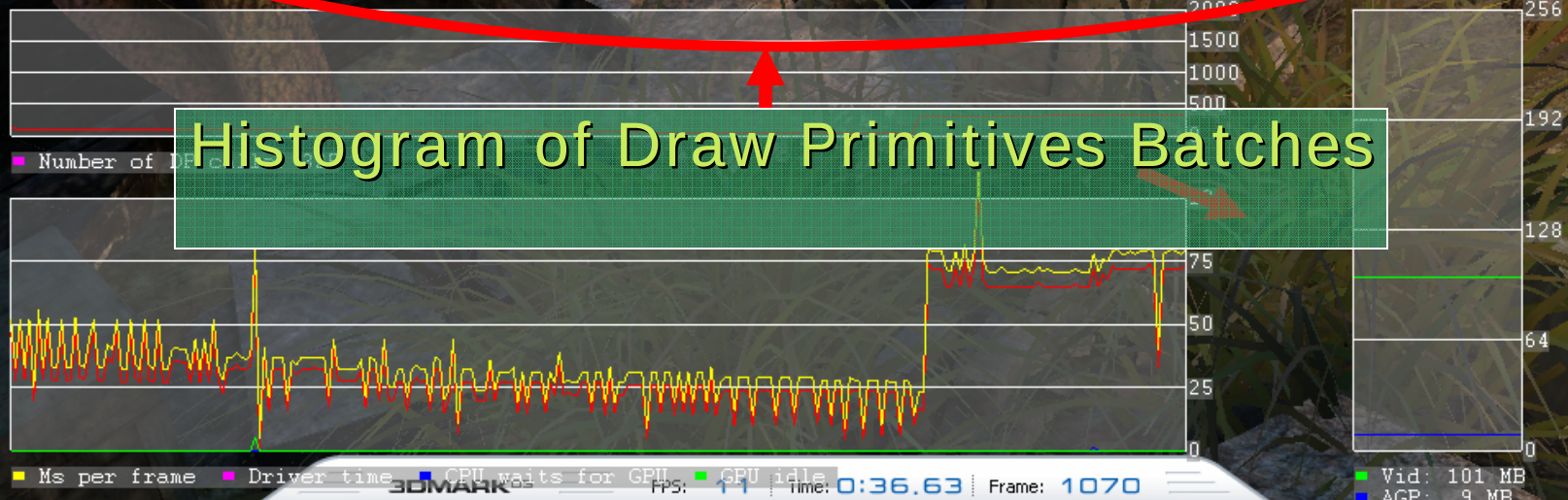
Vid: 101 MB
AGP: 9 MB

FPS: 12 TRIs/Frame: 1702970 Time: 62.4 secs

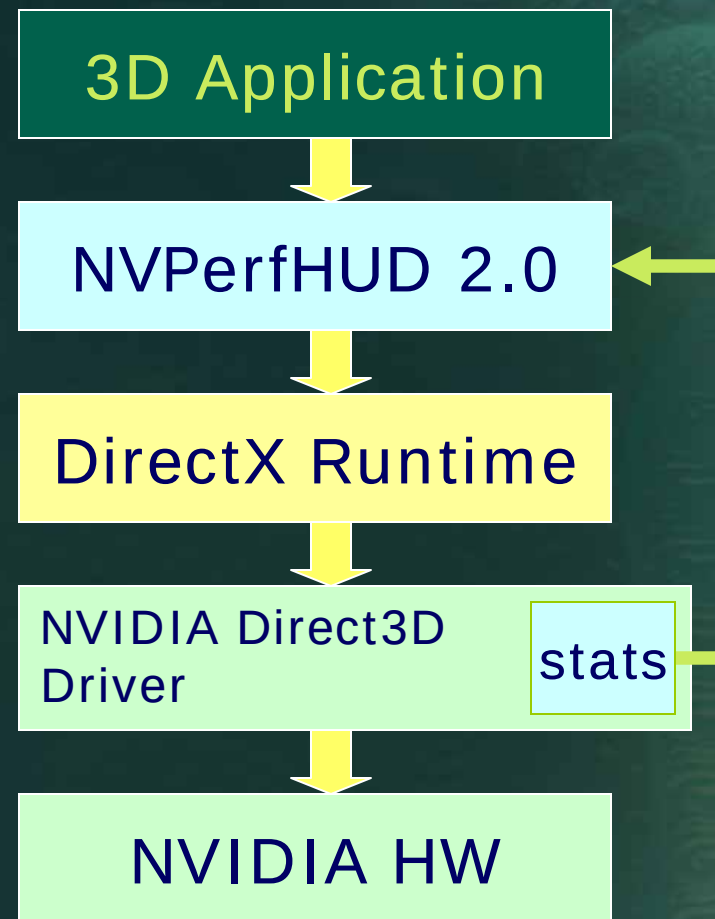
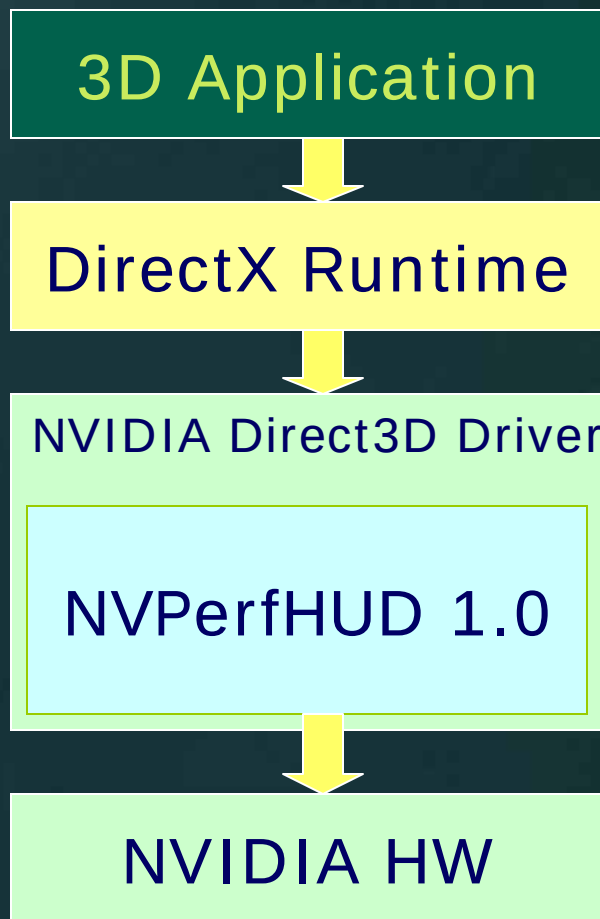
NVPerfHUD 2.0 - NOT FOR BENCHMARKING



Histogram of Draw Primitives Batches

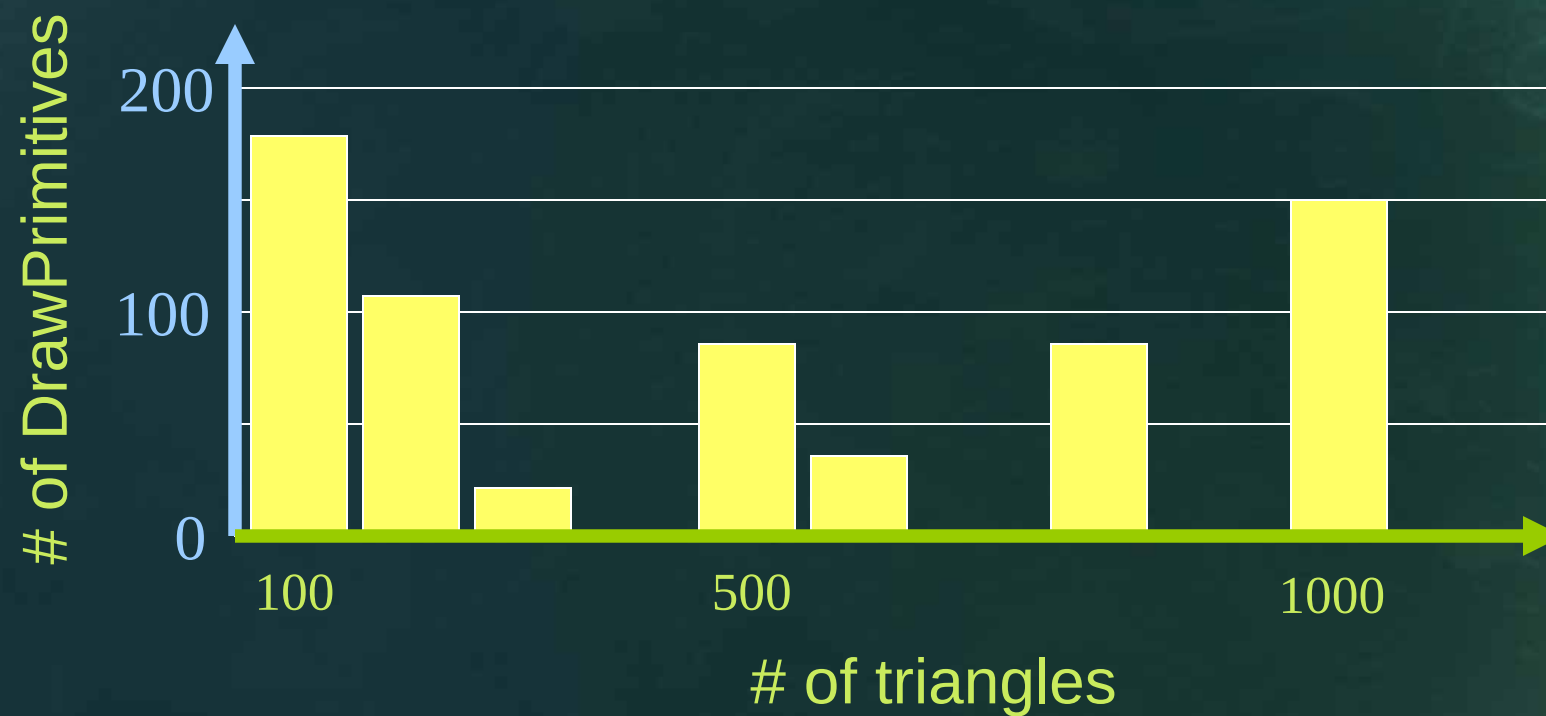


What's new in 2.0?





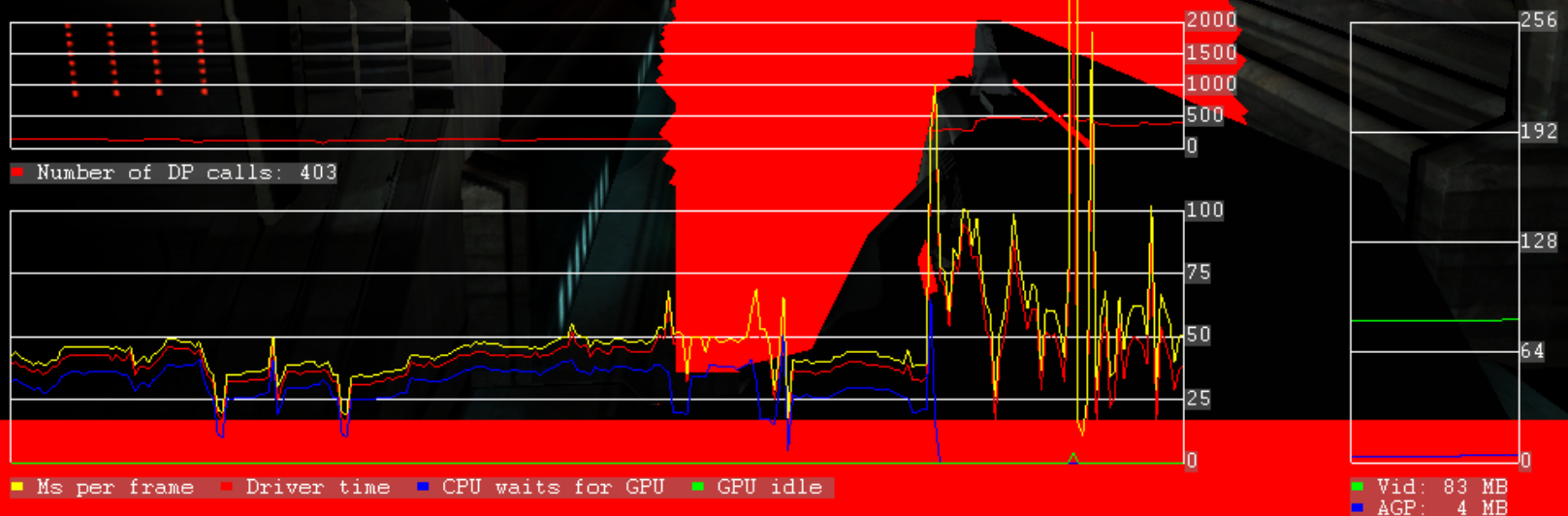
● DrawPrimitives/DrawIndexedPrimitives Histogram



FPS: 17 TRIs/Frame: 272390 Time: 35.6 secs

NVPerfHUD 2.0 - NOT FOR BENCHMARKING

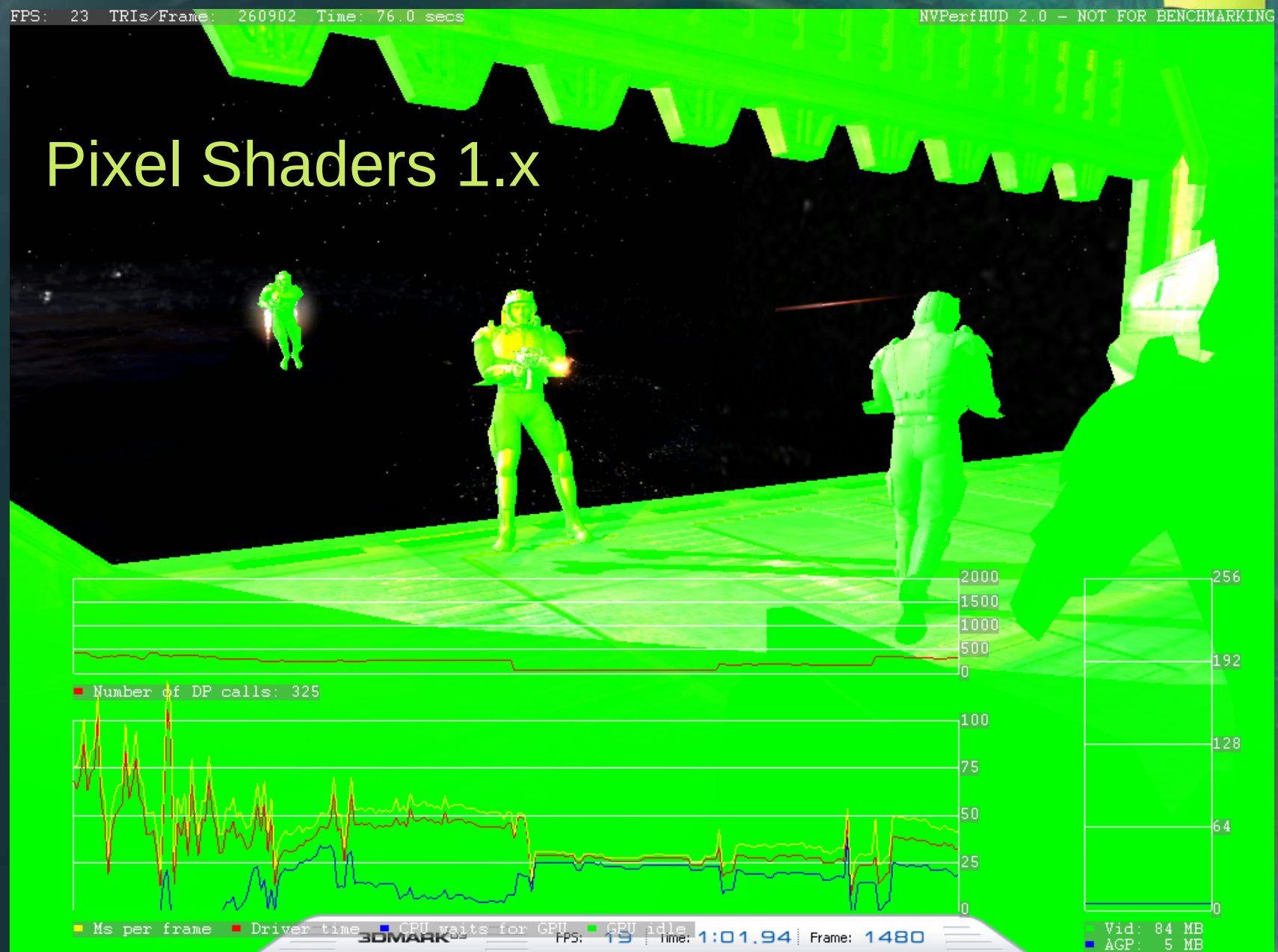
Texture Stage States



FPS: 23 TRIs/Frame: 260902 Time: 76.0 secs

NVPerfHUD 2.0 - NOT FOR BENCHMARKING

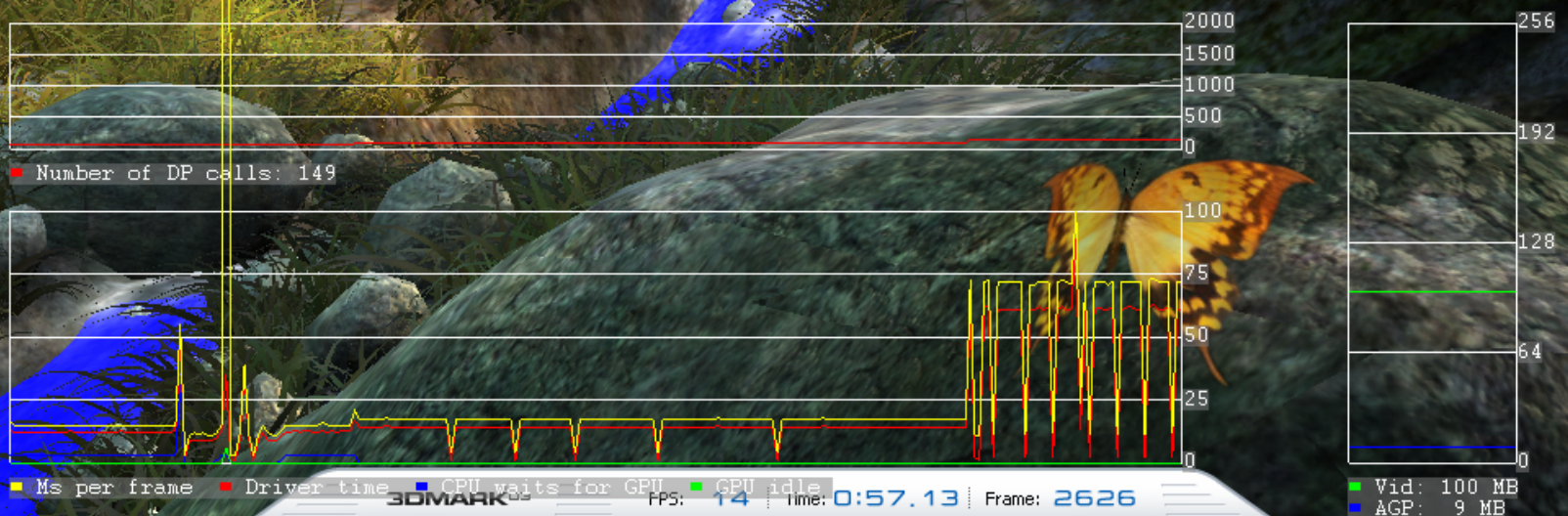
Pixel Shaders 1.x



FPS: 13 TRIs/Frame: 1227229 Time: 91.3 secs

NVPerfHUD 2.0 - NOT FOR BENCHMARKING

Pixel Shaders 2.0 Pixel Shaders 3.0



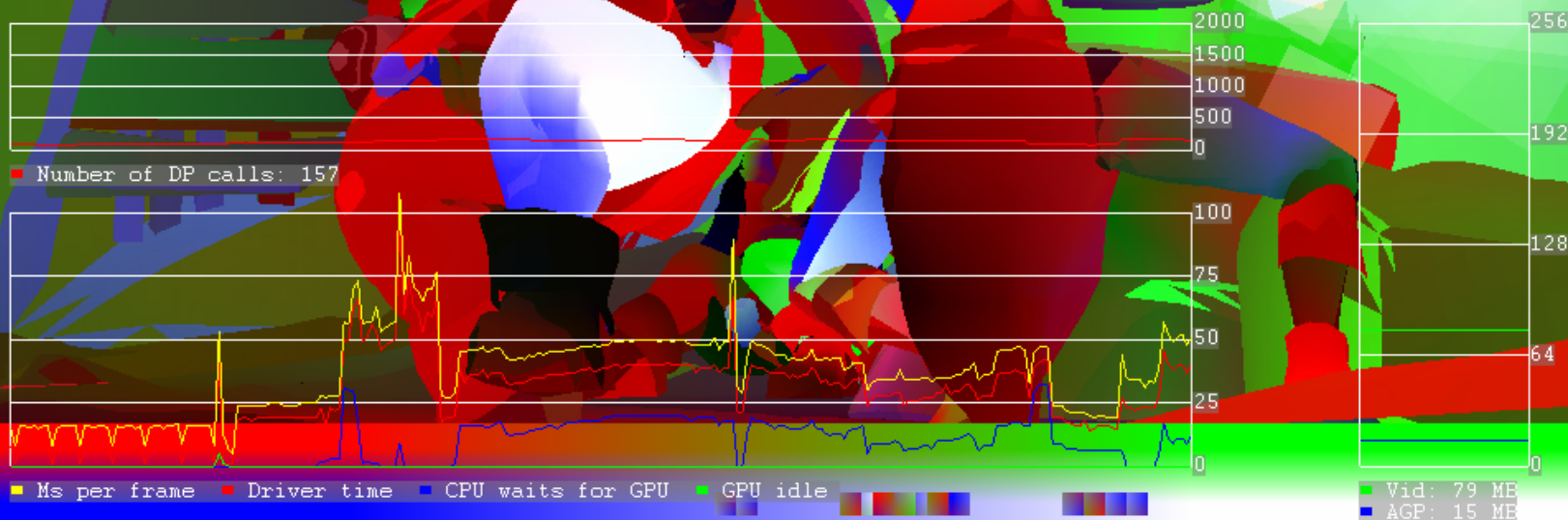
©2004 NVIDIA Corporation. All rights reserved.

Image courtesy of FutureMark Corp.

FPS: 19 TRIs/Frame: 344384 Time: 78.2 secs

NVPerfHUD 2.0 - NOT FOR BENCHMARKING

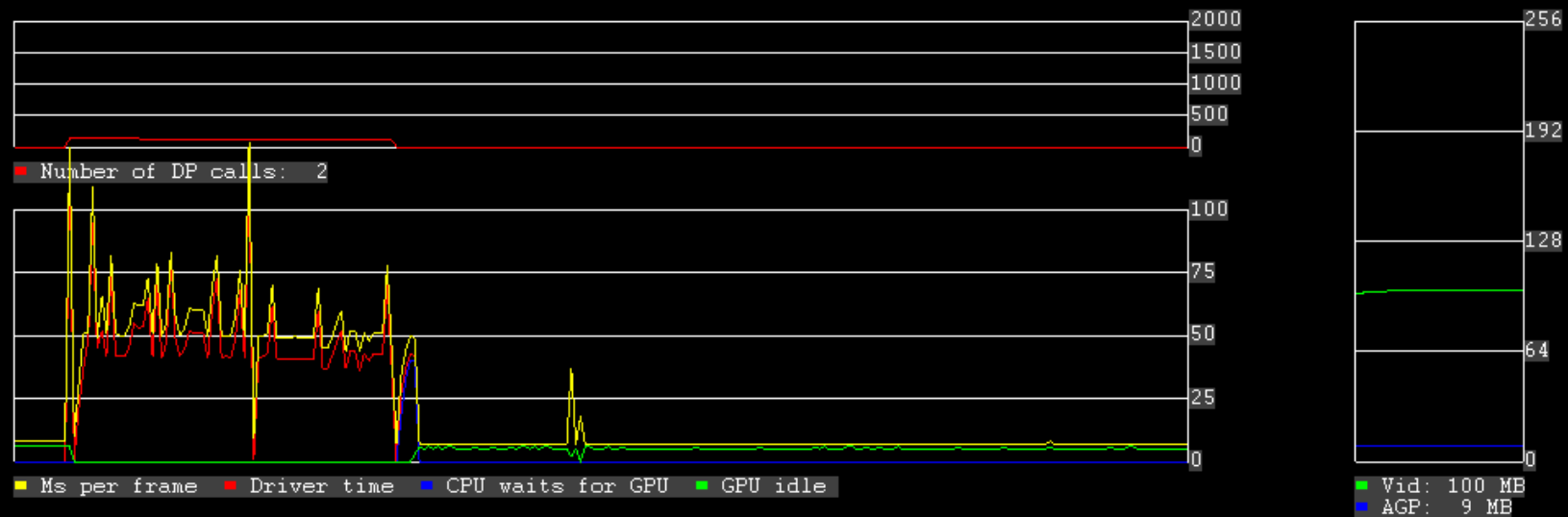
2x2 Texture replacement



FPS: 132 TRIs/Frame: 781650 Time: 60.9 secs

NVPerfHUD 2.0 - NOT FOR BENCHMARKING

- Null DrawPrimitive mode
- Null Viewport mode

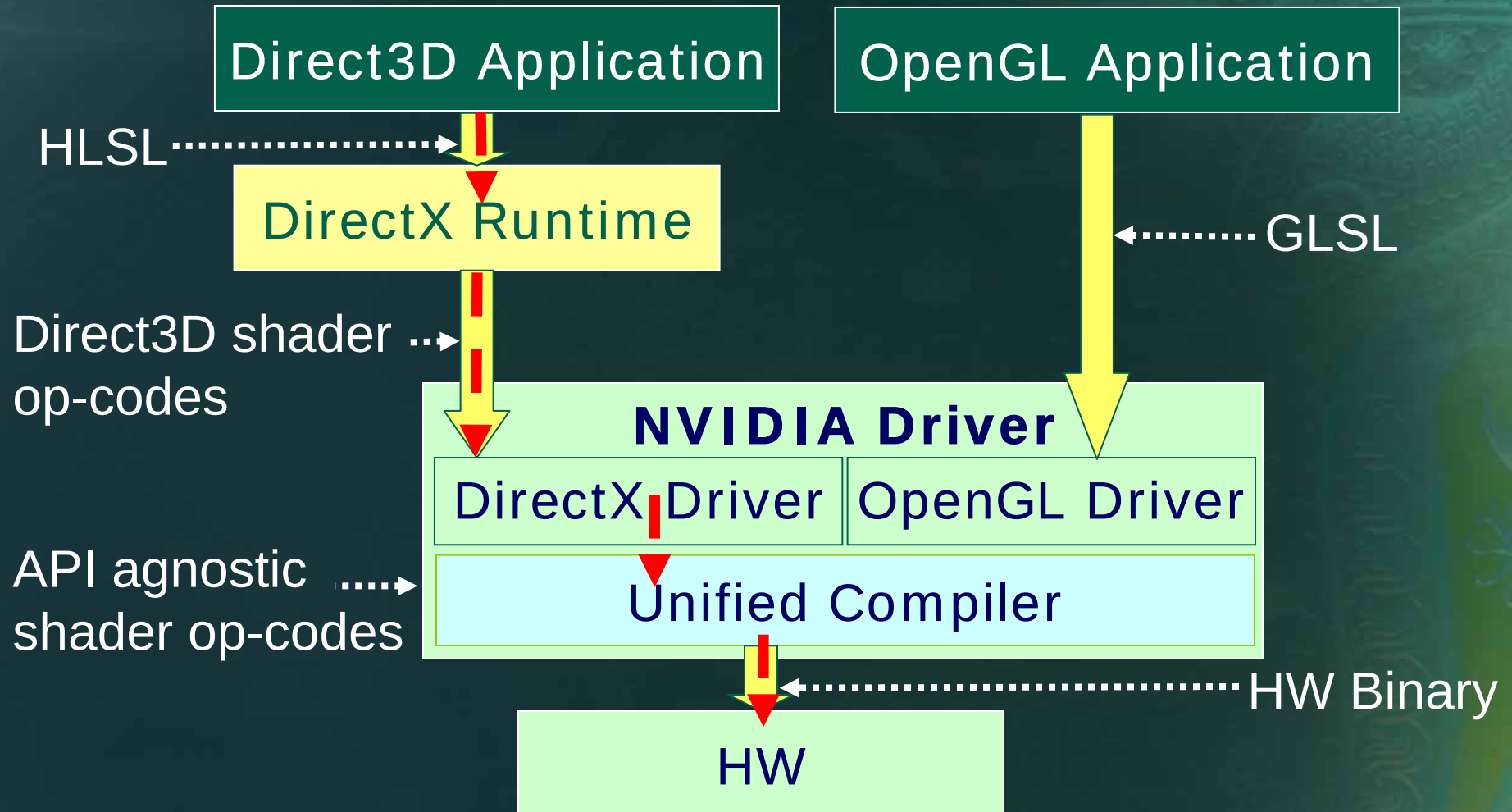




NVPerfHUD - Overhead

- NVPerfHUD is fairly lean but...
- Overlay graph and DLL interception can cost up to 1.3%
- Driver instrumentation can cost up to 6%
- Upper bound for total cost: 7%

NVShaderPerf





NVShaderPerf - Example

```
v2f BumpReflectVS(a2v IN,
    uniform float4x4 WorldViewProj,
    uniform float4x4 World,
    uniform float4x4 ViewIT)
{
    v2f OUT;
    // Position in screen space.
    OUT.Position = mul(IN.Position, WorldViewProj);
    // Tangent to object space.
    OUT.TangentToObjSpace = mul(IN.Tangent, WorldViewProj);
    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.Tangent.y, IN.Tangent.z);
    TangentToObjSpace[1] = float3(IN.Tangent.x, IN.Tangent.y, IN.Tangent.z);
    TangentToObjSpace[2] = float3(IN.Tangent.x, IN.Tangent.y, IN.Tangent.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.TexCoord2.x = dot(World[0].xyz, TangentToObjSpace[1]);
    OUT.TexCoord2.y = dot(World[1].xyz, TangentToObjSpace[1]);
    OUT.TexCoord2.z = dot(World[2].xyz, TangentToObjSpace[1]);
    OUT.TexCoord3.x = dot(World[0].xyz, TangentToObjSpace[2]);
    OUT.TexCoord3.y = dot(World[1].xyz, TangentToObjSpace[2]);
    OUT.TexCoord3.z = dot(World[2].xyz, TangentToObjSpace[2]);
    float4 worldPos = mul(IN.Position, WorldViewProj);
    // compute the eye vector (going from world pos to view pos)
    float4 eyeVector = worldPos - ViewIT.ViewPos;
    OUT.TexCoord1.w = eyeVector.x;
    OUT.TexCoord2.w = eyeVector.y;
    OUT.TexCoord3.w = eyeVector.z;
}
```

```
////// C:\WINDOWS\system32\cmd.exe
```

```
C:\Temp\NVShaderPerf_61_77>
```

```
float4 worldPos = mul(IN.Position, WorldViewProj);
```

```
Running performance on file goochy_HLSL.fx
```

```
*****
```

```
Technique <Untextured>, Pass <p0>
```

```
-----
```

```
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 performance using all FP32
```

```
Cycles: 51 :: # R Registers: 4
```

```
C:\WINDOWS\system32\cmd.exe
```

```
C:\Temp\NVShaderPerf_61_77>NVShaderPerf.exe -a nv40 -allprec -v 0 goochy_HLSL.fx
```

```
Running performance on file goochy_HLSL.fx
```

```
*****
```

```
Technique <Untextured>, Pass <p0>
```

```
-----
```

```
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
```

```
-----
```

```
C:\Temp\NVShaderPerf_61_77>
```



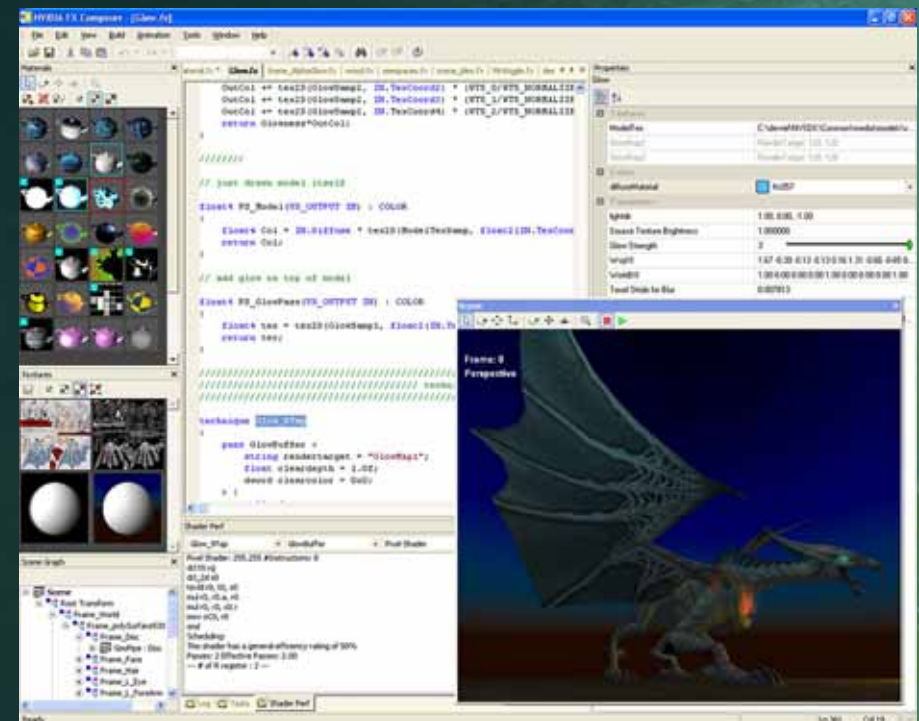
NVShaderPerf Next

- Support for:
 - GLSL (vertices)
 - !!VP1.0
 - !!ARBvp1.0
 - Cg
- Vertex Throughput
- Published COM Interfaces for easy integration into third party tools

FX Composer 1.5



- IDE for HLSL authoring, debugging and optimization
- Vertex and Pixel Shader scheduling
- Direct3D9 VS/PS op-code disassembly
- Advanced texture generation for baking Look Up Tables
- Visualization of RenderTargets

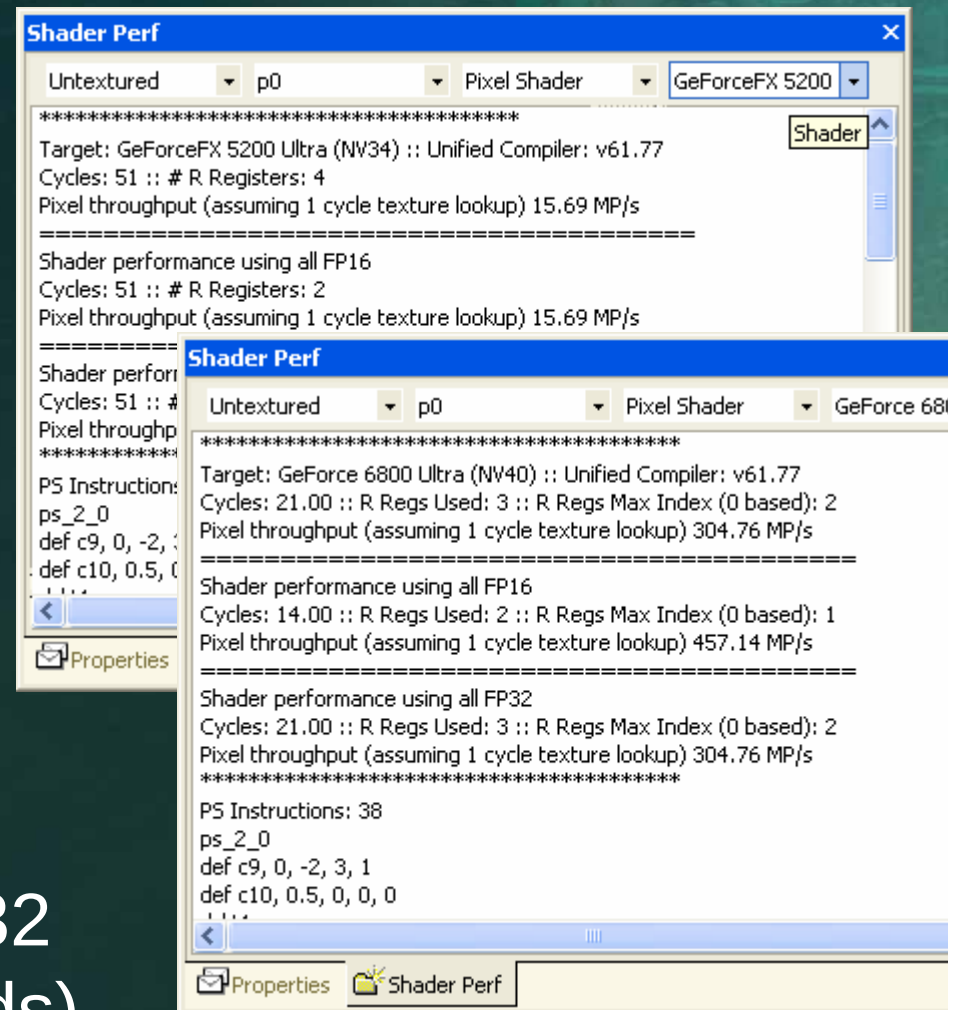


EverQuest® content courtesy Sony Online Entertainment

FX Composer – 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)





Agenda

- Performance Tools Survey
- Performance Methodologies and Practice
- Next generation Performance Tools
- Conclusion
- Q & A



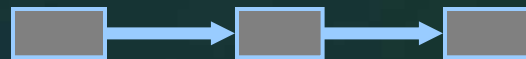
Performance Methodologies

- Basic principles
- Practice identifying the problems
- Learn how to fix the problems using our tools

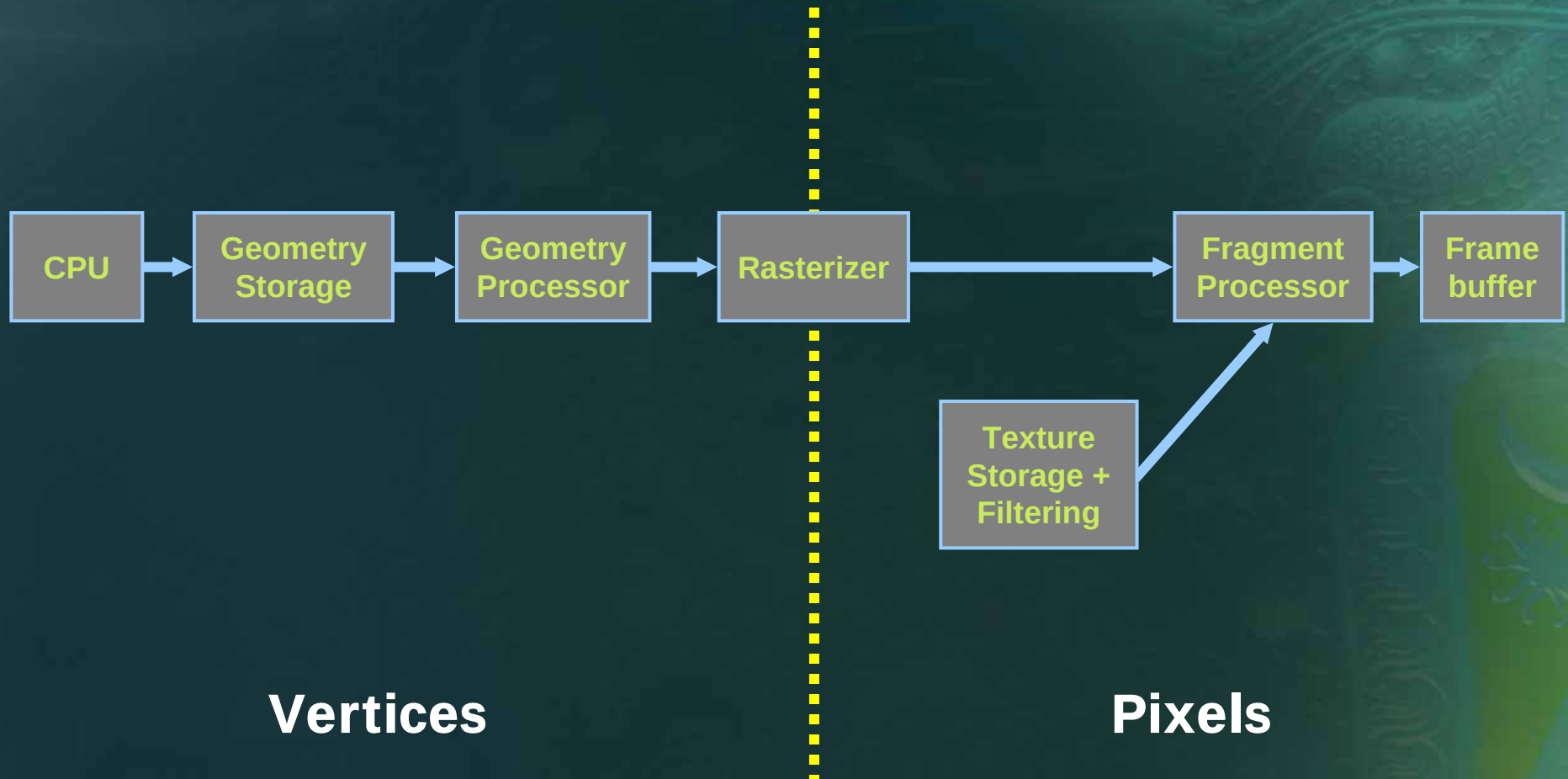


Basic Principles

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



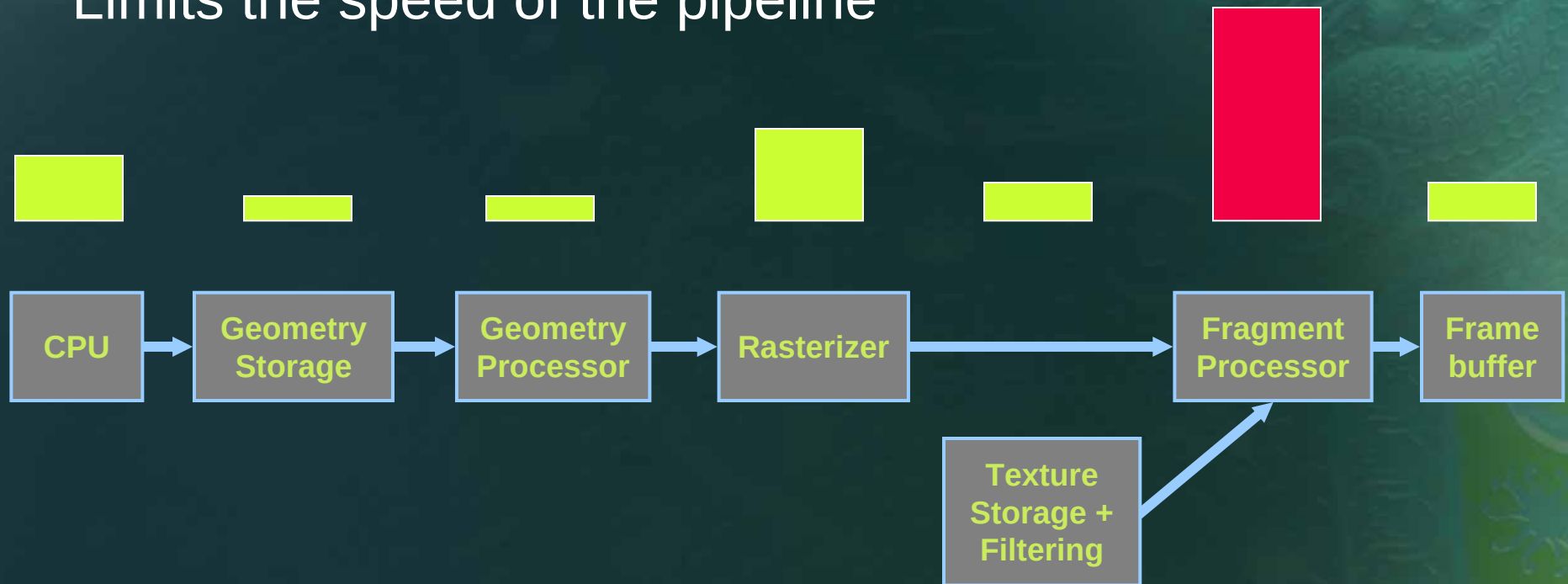
Pipelined Architecture (simplified view)





The Terrible Bottleneck

Limits the speed of the pipeline



Bottleneck Identification

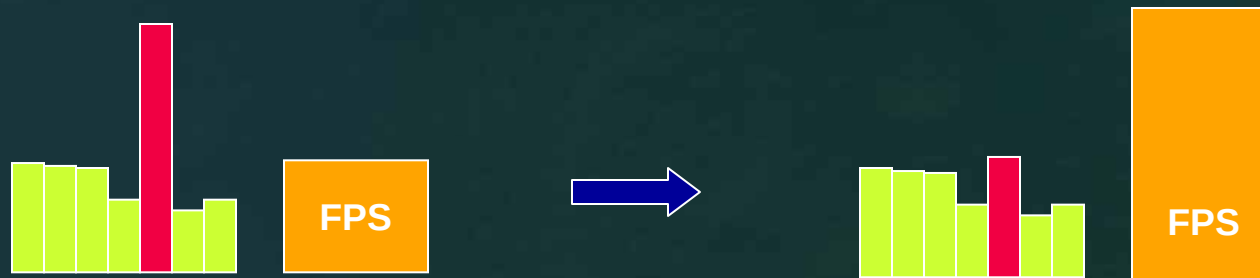


- Need to identify it quickly and correctly
 - Guessing what it is without testing can waste a lot of coding time
- Two ways to identify a stage as the bottleneck
 - Modify the stage itself
 - Rule out the other stages

Bottleneck Identification



- Modify the stage itself
 - By decreasing its workload

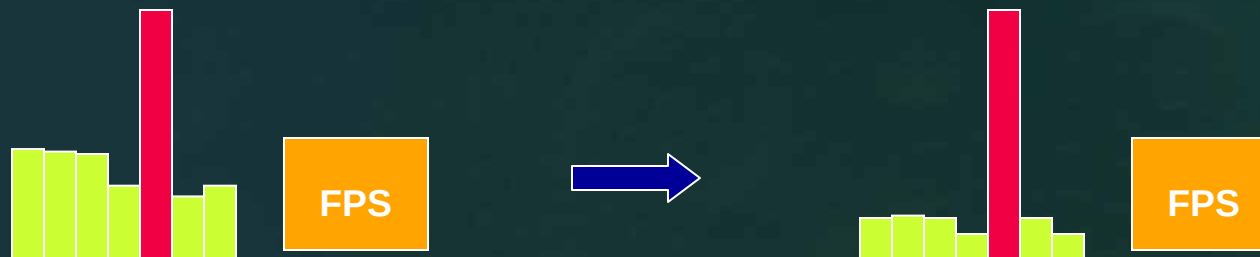


- If performance improves greatly, then you know this is the bottleneck
- Careful not to change the workload of other stages!

Bottleneck Identification



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



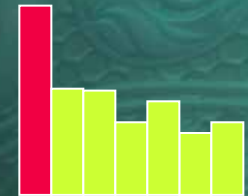
- If performance doesn't change significantly, then you know this is the bottleneck
- Careful not to change the workload of this stage!

Bottleneck Identification



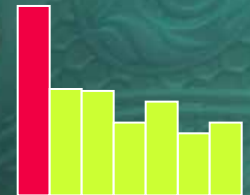
- Most changes to a stage affect other stages as well
- Can be hard to pick what test to do
- Let's go over some tests

Bottleneck Identification: CPU



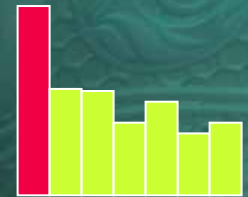
- CPU workload
 - What could the problem be?
 - Could be the game
 - Complex physics, finite element analysis, tessellation
 - Memory management
 - Data structures
 - Could be incorrect usage of API
 - Check debug runtime output for errors and warnings
 - Could be the display driver
 - Too many batches

Bottleneck Identification: CPU



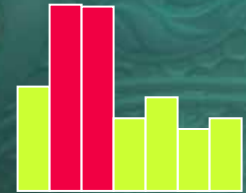
- Reduce the CPU workload
 - Temporarily turn off
 - Complex physics
 - Game logic/AI
 - Dynamic Tessellation
 - Fancy animation controllers
 - Any other thing you know to be expensive on the CPU as long as it doesn't change the rendering workload

Bottleneck Identification: CPU



- Rule out other stages
 - Kill the DrawPrimitive calls
 - Set up everything as you normally would but when the time comes to render something, just do not make the DrawPrimitive* call
 - Problem: you don't know what the runtime or driver does when a draw primitive call is made
 - Use VTUNE or NVPerfHUD
 - These let you see right away if the CPU time is in your app or somewhere else

Bottleneck Identification: Vertex

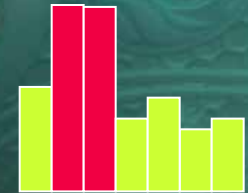


● Vertex Bound

● What could the problem be?

- Transferring the vertices and indices to the card
- Turning the vertices and indices into triangles
- Vertex cache misses
- Using an expensive vertex shader

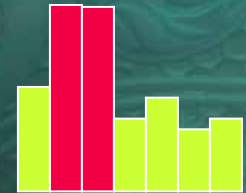
Bottleneck Identification: Vertex



- Reduce vertex overhead

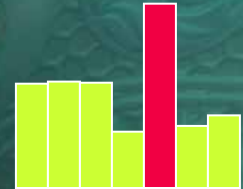
- Use simpler vertex shader
 - But still include all the data for the pixel shader
- Send fewer Triangles??
 - Not good: can affect pixel shader, texture, and frame buffer
- Decrease AGP Aperture??
 - Maybe not good: can affect texture also, depends on where your textures are
 - Use NVPerfHUD to see video memory
 - » If it's full then you might have textures in AGP
- Try to change the Vertex Format with a smaller footprint
 - Bus Bandwidth consideration – unlikely though

Bottleneck Identification: Vertex



- Rule out other stages
 - Render to a smaller backbuffer or null viewport;
this can rule out
 - Texture
 - Frame buffer
 - Pixel shader
 - Test for a CPU bottleneck

Bottleneck Identification: Texture



- Texture Bound
 - What could the problem be?
 - Texture cache misses
 - Huge Textures
 - Bandwidth
 - Texturing out of AGP

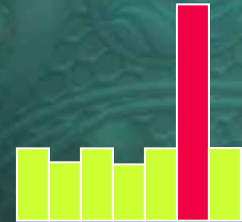
Bottleneck Identification: Texture



● Reduce Texture bandwidth

- Use tiny (2x2) textures
 - Good, but if you are using alpha test with texture alpha, then this could actually make things run slower due to increased fill. It is still a good easy test though
- Use mipmaps if you aren't already
- Use compressed textures (DXT1/3/5)
 - Also, increases the number of texels in the cache.
 - For normal maps, use DXT5 with y in the G channel (6bit), x in the alpha (8bit) and use shader for hemispherical projection (Also See [Mipmapping Normal Maps](#))
- Turn off anisotropic filtering if you have it on

Bottleneck Identification: Fragment

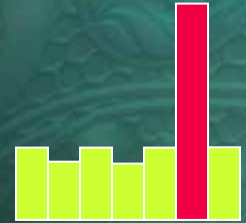


- Fragment Bound

- What could the problem be?

- Expensive pixel shader
 - Rendering more fragments than necessary
 - High depth complexity
 - Poor z-cull

Bottleneck Identification: Fragment



- Modify the stage itself
 - Just output a solid color
 - Good: does no work per fragment
 - But also affects texture, so you must then rule out texture
 - Use simpler math
 - Good: does less work per fragment
 - But make sure that the math still indexes into the textures the same way or you will change the texture stage as well

Bottleneck Identification: FB



- Frame Buffer bandwidth
 - What could the problem be?
 - Touching the buffer more times than necessary
 - Multiple passes
 - Tons of alpha blending – too many read/modify/writes
 - Using too big a buffer
 - Allocating stencil when you don't need it
 - A lot of time dynamic reflection cube-maps can get away with r5g6b5 color instead of x8r8g8b8

Bottleneck Identification: FB



- Modify the stage itself
 - Use a 16 bit depth buffer instead of a 24 bit one
 - Use a 16 bit color buffer instead of a 32 bit one

Bottleneck Identification



- Now we have a bunch of practical ideas to find out if each stage is a bottleneck or not

- Questions on Bottleneck Identification?



Practice

- Now lets look at some sample problems and see if we can find out where the problem is
- NVPerfHUD to the rescue!



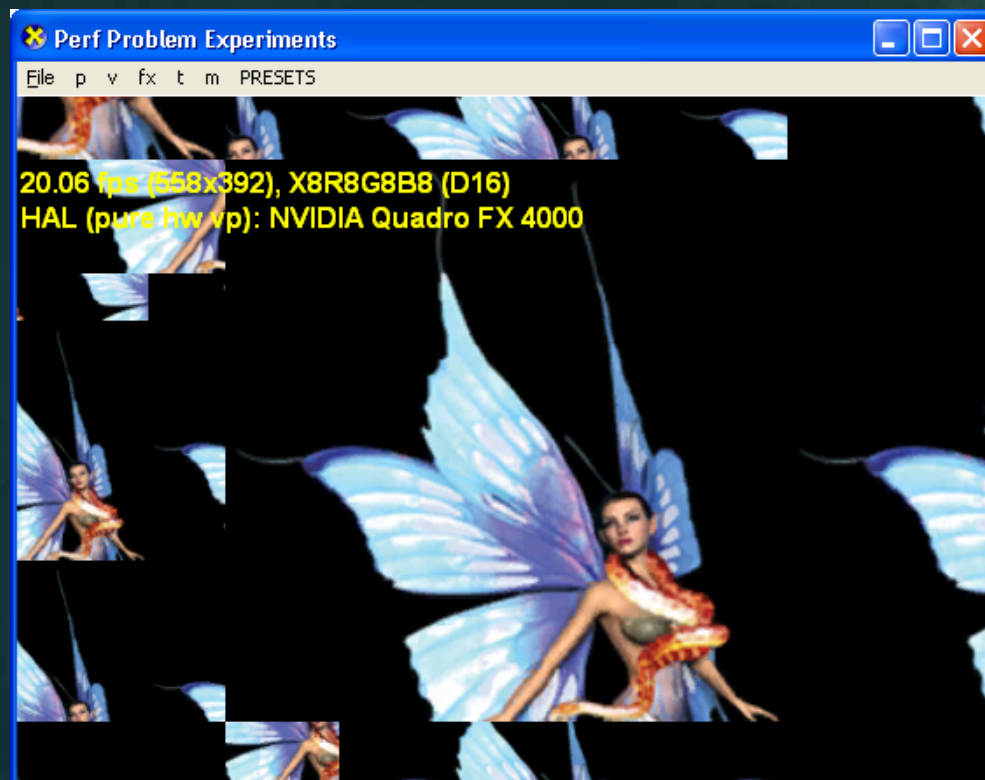
Practice: Clean the Machine

- Please use the exact target configuration!
- Make sure that your machine is ready for analysis
 - Make sure you have the right drivers
 - Use a release build of the game (optimizations on)
 - Check debug output for warnings or errors but.....
 - Use the **release** d3d runtime!!!
 - No maximum validation
 - No driver overridden anisotropic filtering or anti-aliasing
 - Make sure v-sync is off



Practice: Example 1

- A seemingly simple scene runs horribly slow
 - Narrow in on the bottleneck





Practice: Example 1

- Dynamic vertex buffer
 - BAD creation flags

```
HRESULT hr = pd3dDevice->CreateVertexBuffer(  
    6* sizeof( PARTICLE_VERT ),  
    0,    //declares this as static  
    PARTICLE_VERT::FVF,  
    D3DPOOL_DEFAULT,  
    &m_pVB,  
    NULL );
```



Practice: Example 1

- Dynamic vertex buffer
 - GOOD creation flags

```
HRESULT hr = pd3dDevice->CreateVertexBuffer(  
    6* sizeof( PARTICLE_VERT ),  
    D3DUSAGE_DYNAMIC |  
    D3DUSAGE_WRITEONLY,  
    PARTICLE_VERT::FVF,  
    D3DPOOL_DEFAULT,  
    &m_pVB,  
    NULL );
```



Practice: Example 1

- Dynamic Vertex Buffer
 - BAD Lock flags

```
m_pVB->Lock(0, 0,(void**)&quadTris, 0);
```

- No flags at all!?
 - That can't be good....



Practice: Example 1

- Dynamic Vertex Buffer
 - GOOD Lock flags

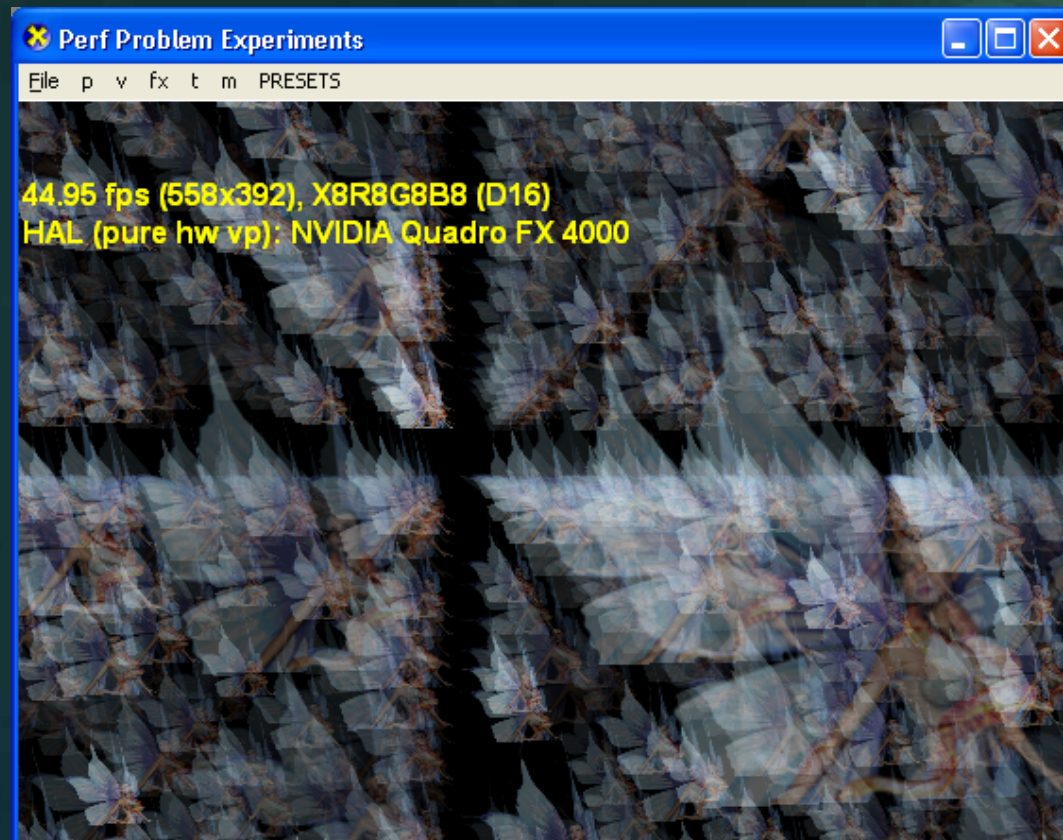
```
m_pVB->Lock(0, 0, (void**)&quadTris,  
D3DLOCK_NOSYSLOCK | D3DLOCK_DISCARD);
```

- Use D3DLOCK_DISCARD the first time you lock a vertex buffer each frame
 - And again when that buffer is full
 - Otherwise just use NOSYSLOCK

Practice: Example 2



- Another slow scene
 - What's the problem here





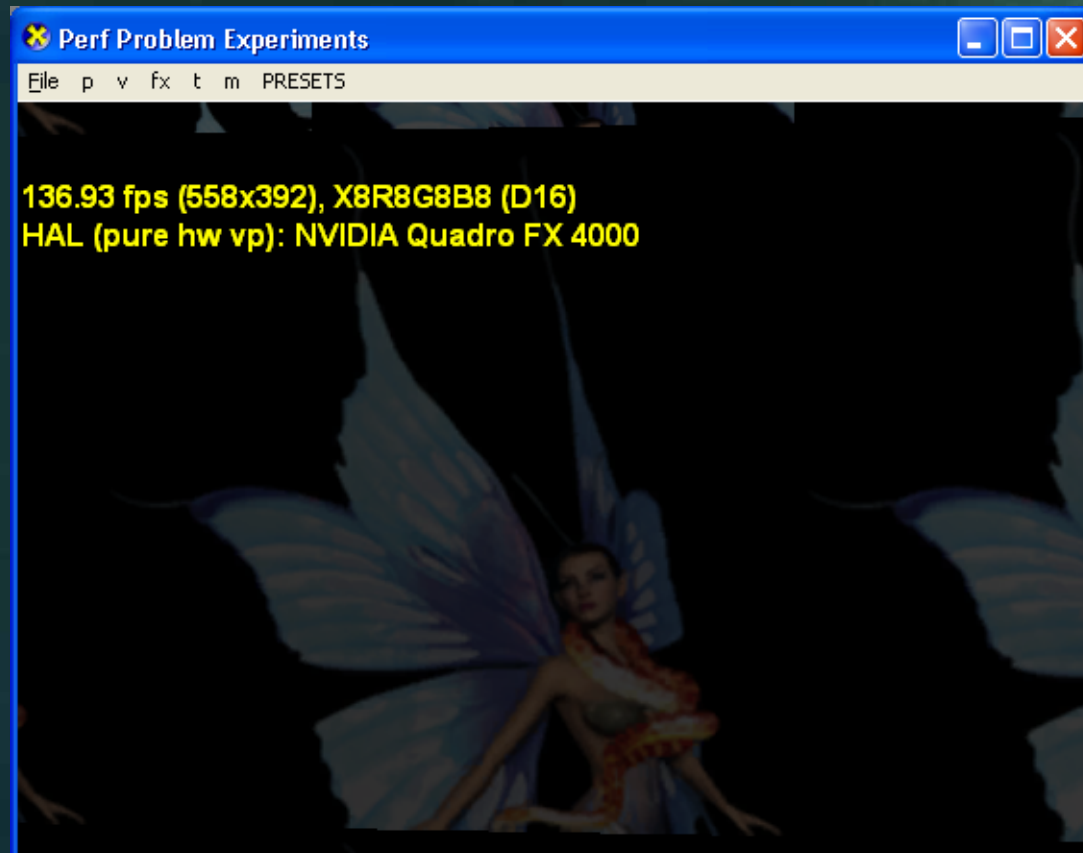
Practice: Example 2

- Texture bandwidth overkill
 - Use mipmaps
 - Use dxt1 if possible
 - Some cards can store compressed data in cache
 - Use smaller textures when they are fine
 - Does the grass blade really need a 1024x1024 texture?
 - Maybe

Practice: Example 3



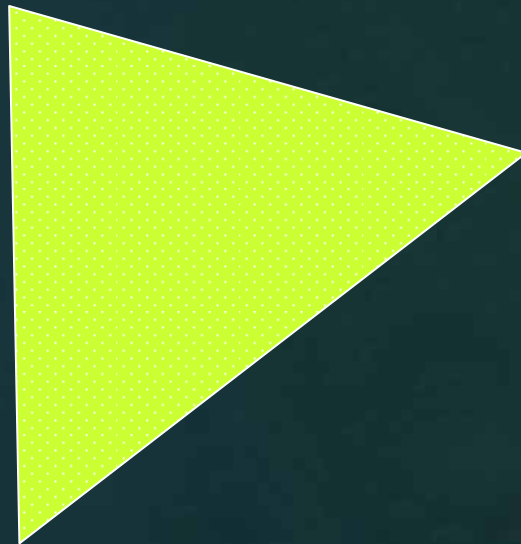
- Another slow scene
 - Who wants a prize?





Practice: Example 3

- Expensive pixel shader
 - Can have huge performance effect
 - Only 3 verts, but maybe a million pixels
 - That's only 1024x1024



Look at all the pixels!!

Practice: Example 3



32 cycles BAD



Shader Perf

TestFXCheapVSE p0 Pixel Shader GeForce 6800 ULI

Target: GeForce 6800 Ultra (NV40) :: Unified Compiler: v61.77
Cycles: 31.50 :: R Regs Used: 7 :: R Regs Max Index (0 based): 6
Pixel throughput (assuming 1 cycle texture lookup) 206.45 MP/s

=====

Shader performance using all FP16
Cycles: 20.00 :: R Regs Used: 3 :: R Regs Max Index (0 based): 2
Pixel throughput (assuming 1 cycle texture lookup) 320.00 MP/s

=====

Shader performance using all FP32
Cycles: 31.50 :: R Regs Used: 7 :: R Regs Max Index (0 based): 6
Pixel throughput (assuming 1 cycle texture lookup) 206.45 MP/s

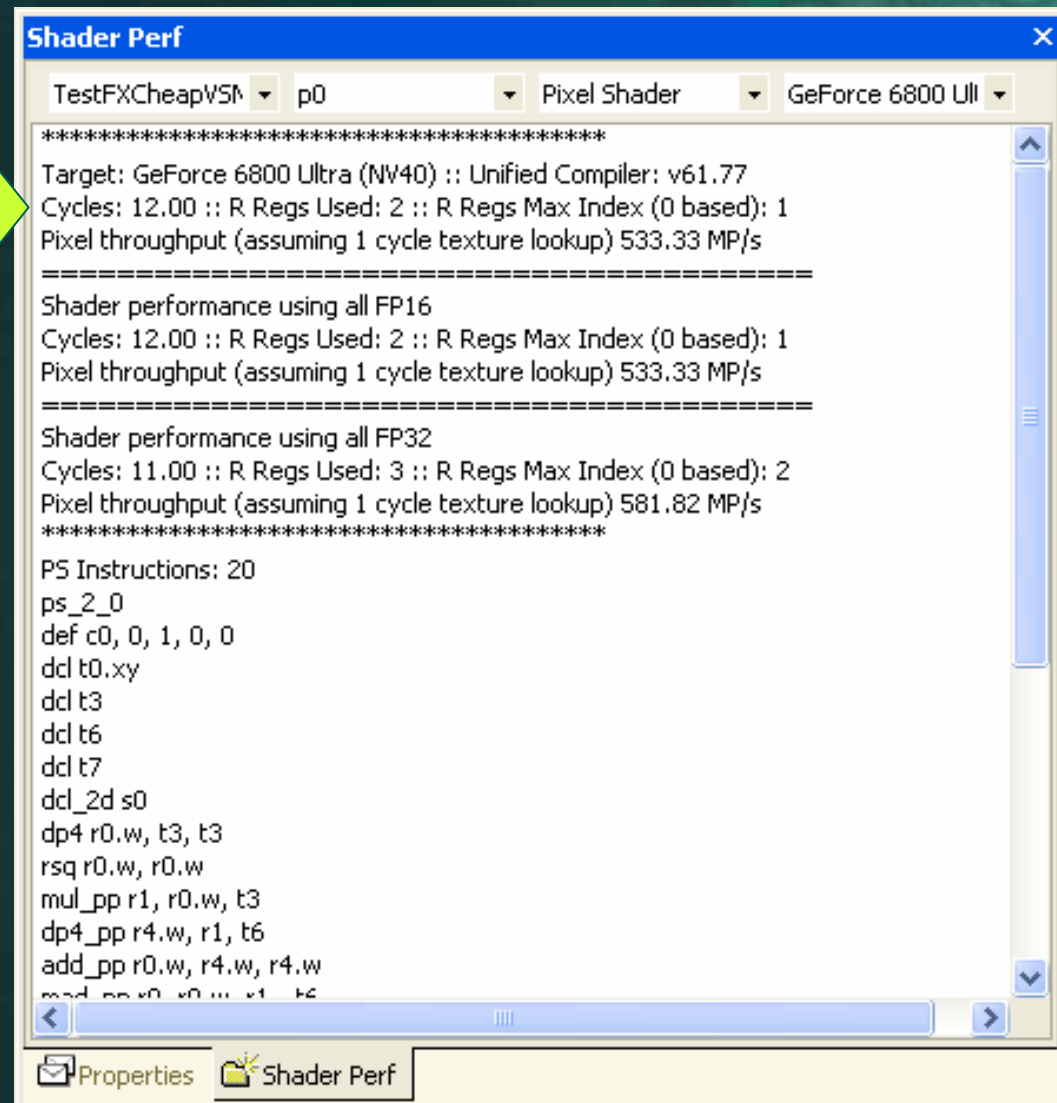
PS Instructions: 35
ps_2_0
def c0, 0, 1, 0, 0
dcl t0.xy
dcl t1.xyz
dcl t2.xyz
dcl t3
dcl t4
dcl t5
dcl_2d s0
dp4 r0.w, t4, t4
rsq r0.w, r0.w
mul r0.xyz, r0.w, t4
mul r0.w, r0.w, t2

Properties Shader Perf

Practice: Example 3



12 cycles GOOD





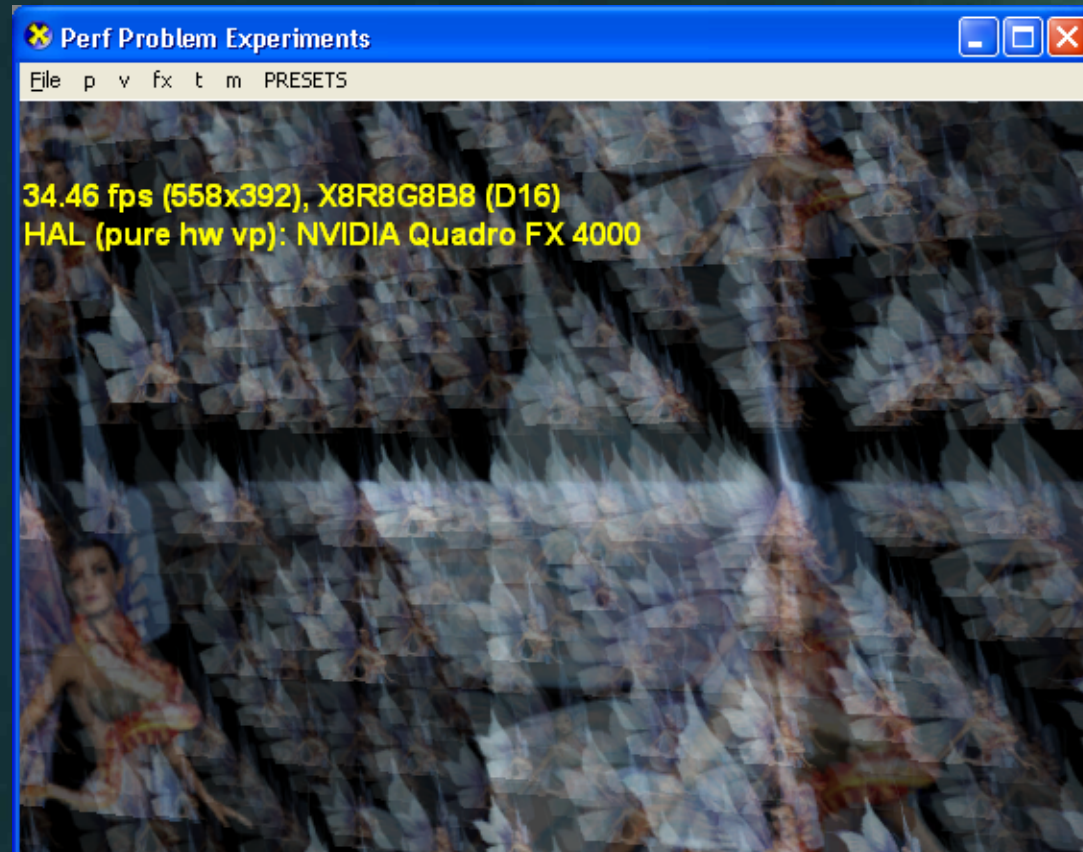
Practice: Example 3

- What changed?
 - Moved math that was constant across the triangle into the vertex shader
 - Used 'half' instead of 'float'
 - Got rid of normalize where it wasn't necessary
 - See Normalization Heuristics
 - <http://developer.nvidia.com>



Practice: Example 4

- The last one
 - Audience: there are no more prizes, but we've locked the doors





Practice: Example 4

- What if they use different textures?
 - Use texture atlases
 - Put the two textures into a single texture and use a vertex and pixel shader to offset the texture coordinates



Agenda

- Performance Tools Survey
- Performance Methodologies and Practice
- Next generation Performance Tools
- Conclusion
- Q & A

Next Generation Performance Tools



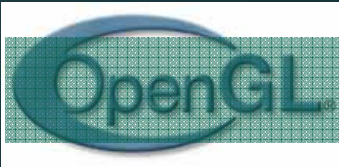
- NVIDIA Performance Kit (PerfKit)
 - Instrumented Driver
 - NVIDIA Developer Control Panel (NVDevCPL)
 - NVIDIA Plug-in for Microsoft PIX for Windows
 - Sample Code for OpenGL and DirectX
 - Special library/header for secure hand-shake
- Will be available to registered developers

Problem



Application

Common Profilers



Microsoft
API

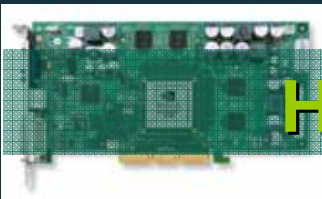
DIRECTX®

PIX for Windows
OGLDebug



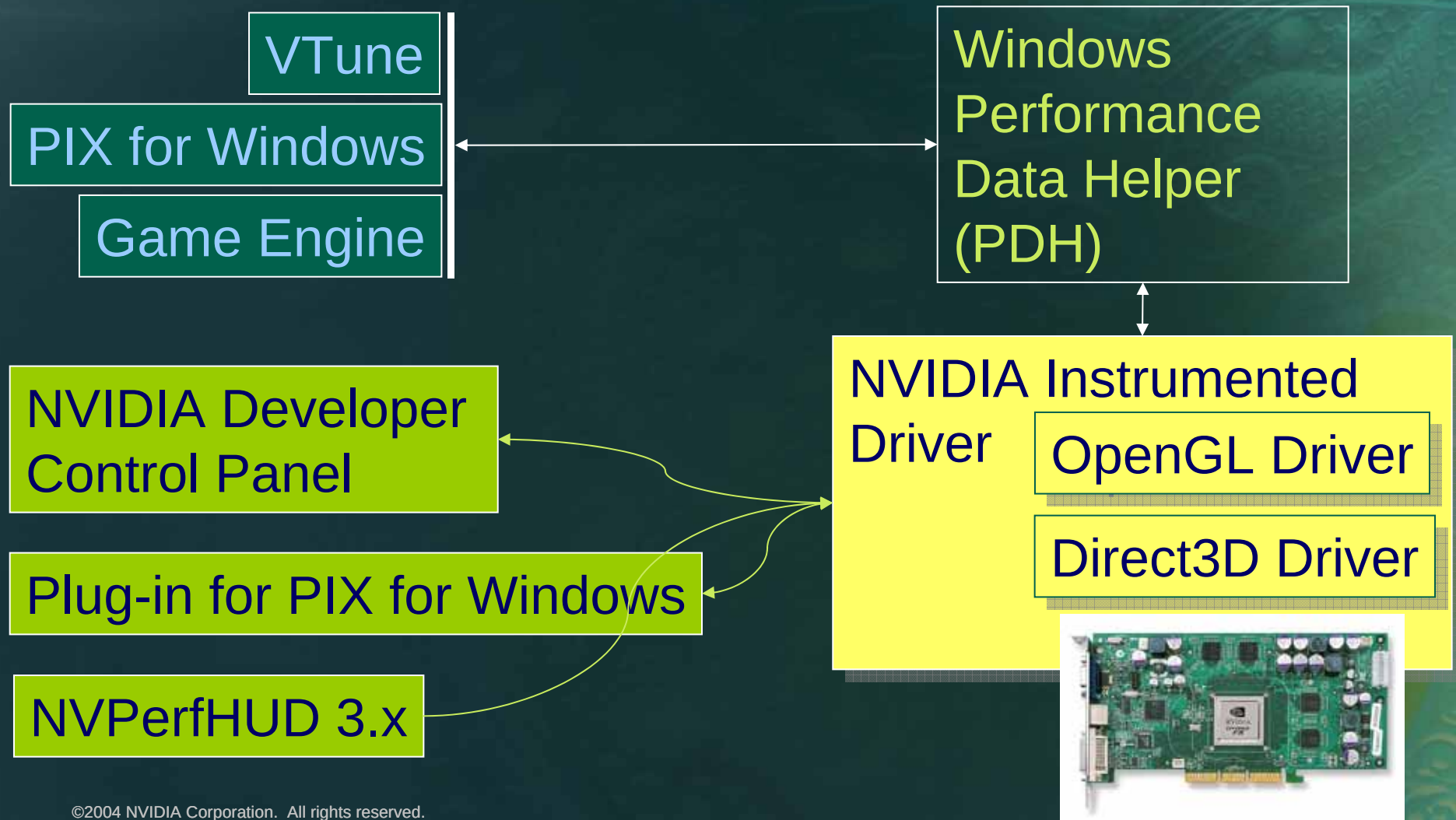
Driver

How to evaluate
performance here?



Hardware

Solution

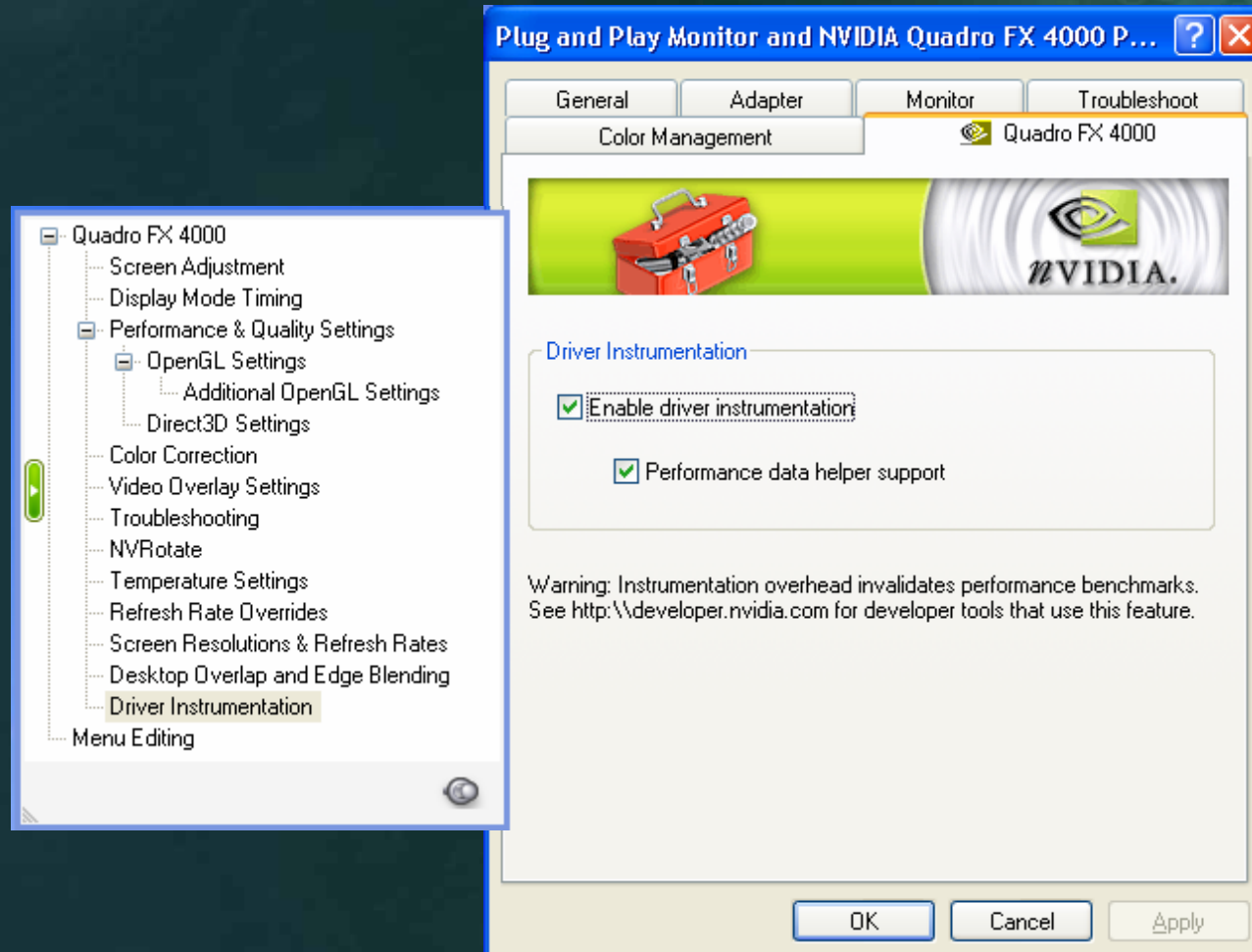




Instrumented Driver

- Special Instrumented Driver
 - Built with regular drivers
 - Includes NVPMAPI.DLL (PDH server)
- Exposes Driver and HW Performance Counters
- Compatible with Windows WMI and PDH
- New Driver Instrumentation tab in NVIDIA Display Control Panel
- Secure mechanism to prevent unauthorized performance analysis

Instrumented Driver

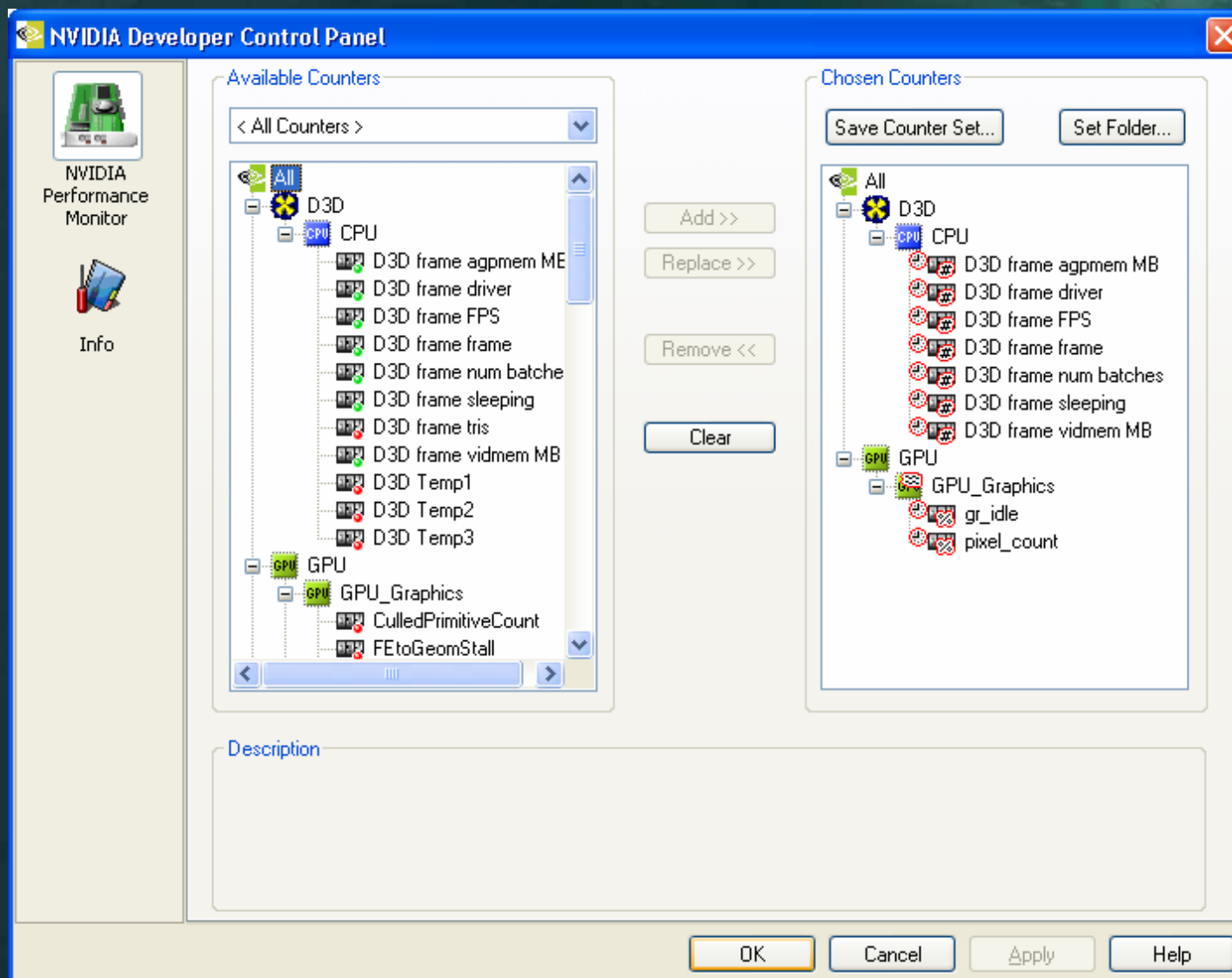




Developer Control Panel

- Control per-counter specifics
 - Enabled or not
 - Raw values or % values
 - Etc.
- Manage multiple counter-sets
 - Tray Icon: fast application of presets
- Provides HW specific information

Developer Control Panel: Demo



PIX for Windows



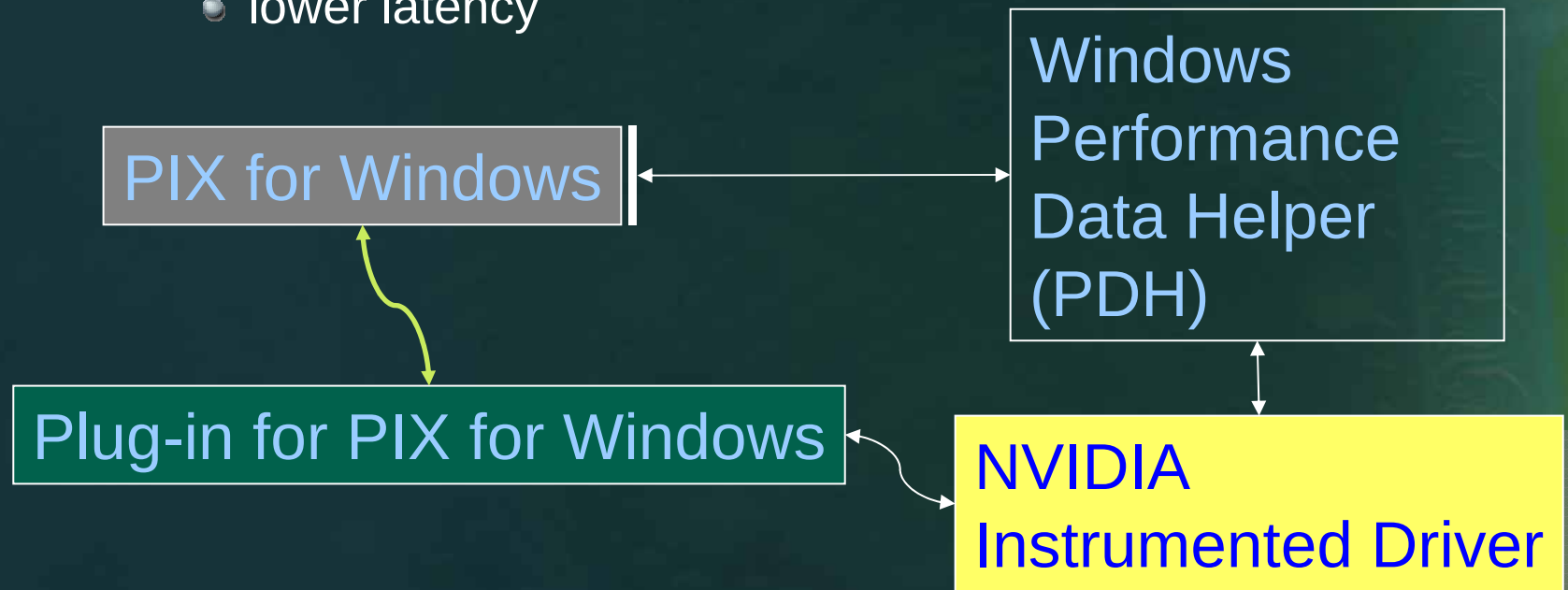
- New tools from Microsoft
 - Coming from Xbox success
 - Focused on Direct3D API instrumentation



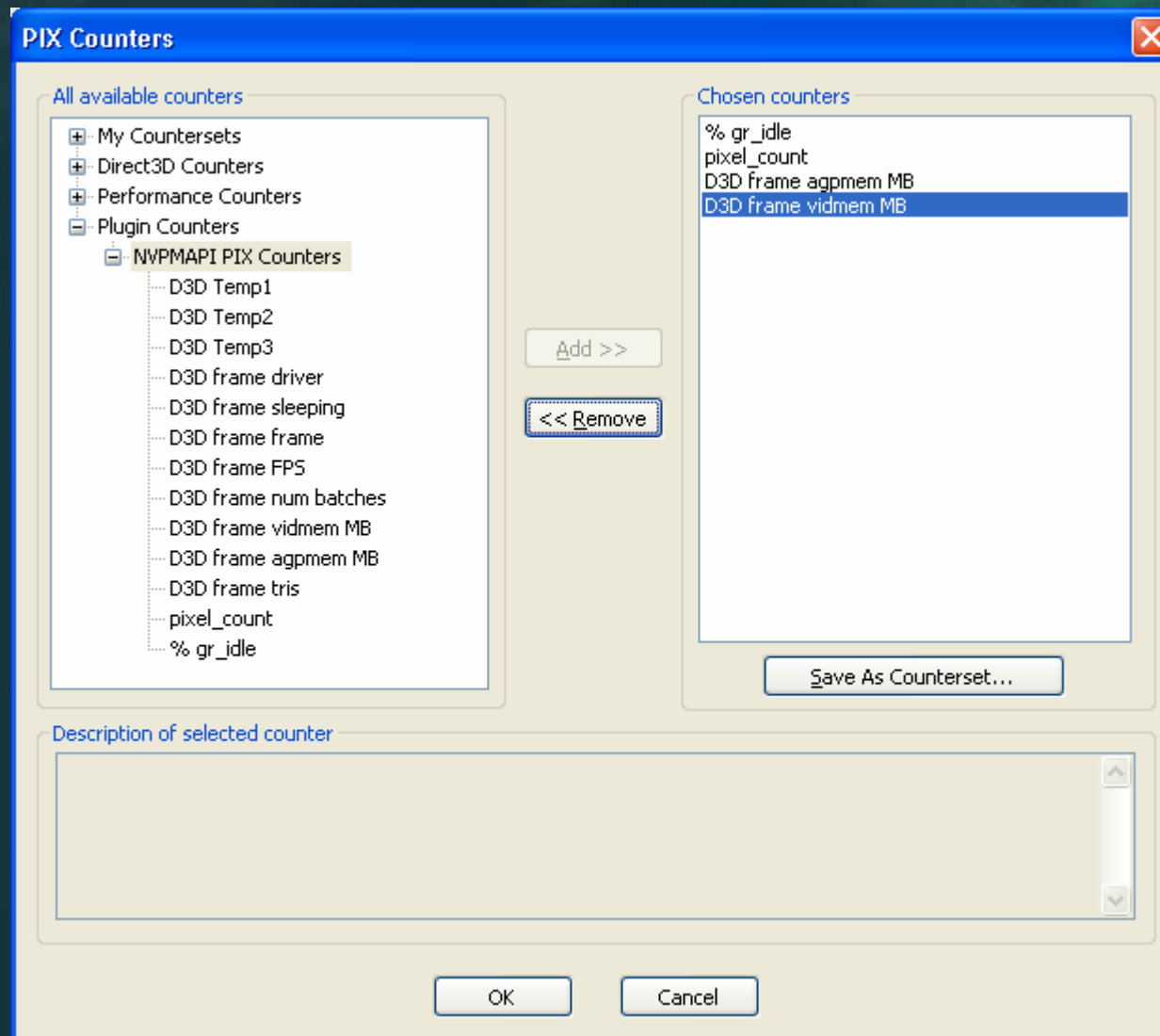


PIX for Windows Plug-in

- NVIDIA's performance counters
 - PIX's PDH adaptor
 - NVIDIA's Pix Plug-in
 - higher frequency
 - lower latency



Plug-in for PIX





NVPerfKit Code Samples

- Includes C++ helper classes for PDH access and display
 - PDHHelper
 - Trace<T>
 - TraceDisplay
 - Various display types
 - OpenGL and Direct3D implementations
- Sample Code and App
 - Identical usage code in Direct3D and OpenGL
 - Illustrates sampling issues and dynamic reconfigurability



NVPerfKit Code Samples

Members

- CPDHelper pdh;
- Ctrace<float> trace;
- CTraceDisplay tracedisplay(0,0,0.5,0.25);

Init

- pdh.add("\\NVIDIA GPU Performance(GPU0/%gr_idle)\\% GPU Counter Value");
- trace.name("idle");
- tracedisplay.insert(&trace,0,1,0,0.2);

Periodically

- pdh.sample();
- trace.insert(pdh.value(0).doubleValue);
- tracedisplay.display(CTraceDisplay::LINE_STREAM);



Next Gen NVPerfHUD

- Real time step-by-step frame rendering
 - visualize draw call individually
 - support render targets
- Automated bottleneck analysis
- New Advanced GUI system
- Visualize Direct3D Debug runtime and Application OutputDebug Streams

Next Gen NVPerfHUD



FPS: 160.6 TRIs/Frame: 0 Time: 81.6 secs [ON] NVPerfHUD 3.0 - NOT FOR BENCHMARKING

```
24) Enabled: idx_geom_attribute_count
25) Enabled: instruction_issued_vpe0
26) Enabled: FERRCount1
27) Enabled: FERRCount2
28) Enabled: texture_footprint_count
29) Enabled: OGL FPS
30) * Done
31) DirectInput8Create(00400000, 00000800, bf798030, 4da2483a, 645d99aa, )
32) Res = 00000000
33) Enabled: D3D frame driver
34) Enabled: D3D frame sleeping
35) Enabled: D3D frame frame
36) Enabled: D3D frame FPS
37) Enabled: D3D frame num batches
38) Enabled: D3D frame vidmem MB
39) Enabled: D3D frame agpmem MB
40) Enabled: D3D frame tris
41) Enabled: gr_idle
42) Enabled: shader_all_quads
43) Enabled: idx_geom_attribute_count
44) Enabled: instruction_issued_vpe0
45) Enabled: FERRCount1
46) Enabled: FERRCount2
47) Enabled: texture_footprint_count
48) Enabled: OGL FPS
```

DP Pointer 22 / 204
DrawIndexedPrimitive(D3DPT_TRIANGLELIST ,0,0,14,0)
RT:0x00183740 BB:0x00183740

Step Back Step Forward

☐ Depth Comp
☐ Wire Frame
☐ Force Clear



Conclusion

- Comprehensive Suite of Performance Tools
 - performance information at all levels
 - Direct3D API
 - Direct3D and OpenGL Driver
 - Hardware
- Through a variety of venues
 - Microsoft WMI/PDH
 - Microsoft PIX for Windows
 - User application
 - NVPerfHUD, FX Composer, NVShaderPerf,...
- Exciting Next Gen of Performance Tools



Questions?

- What else can we do for you?
 - sdkfeedback@nvidia.com



The Source for GPU Programming

developer.nvidia.com

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

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