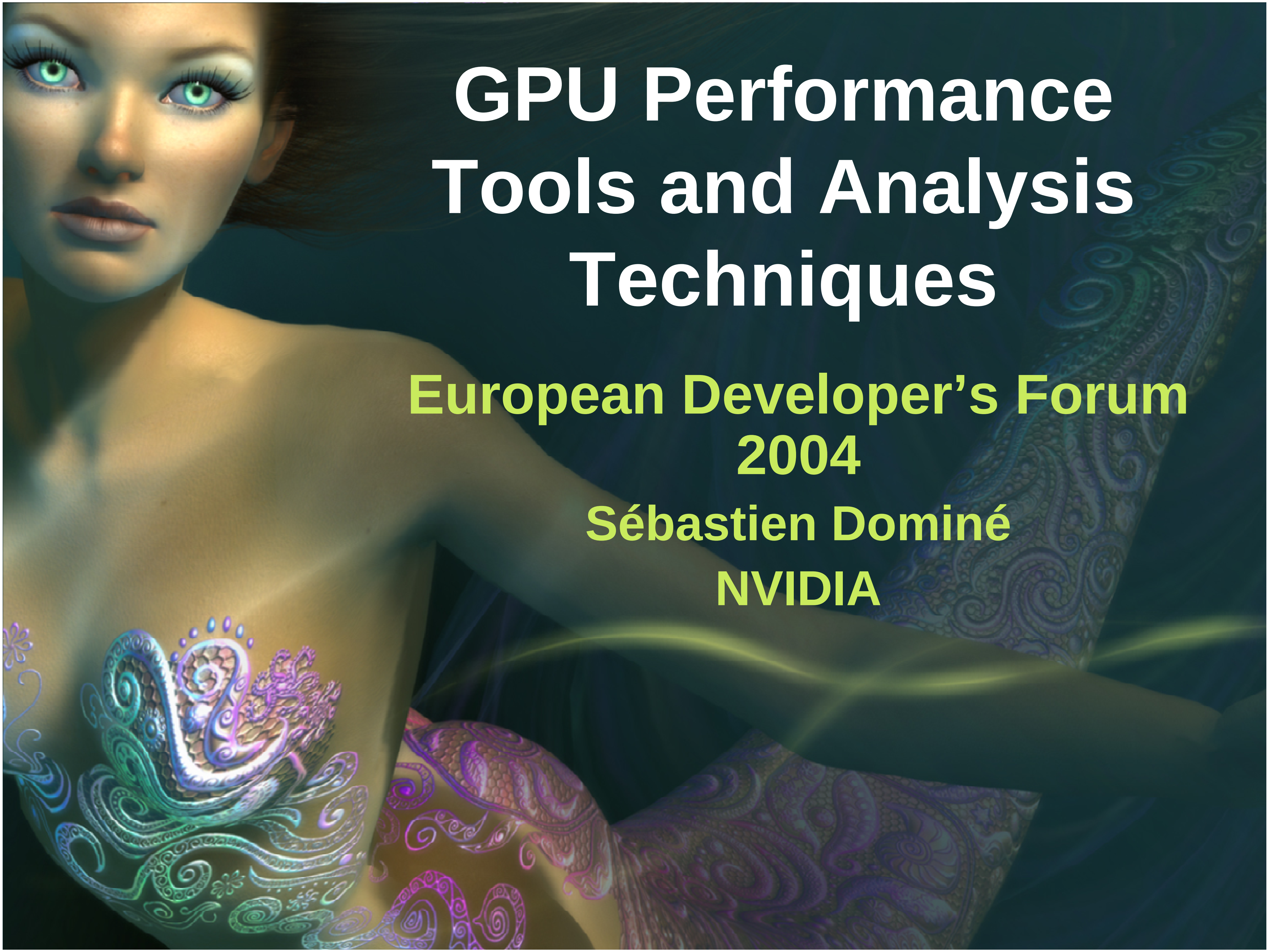


GPU Performance Tools and Analysis Techniques

European Developer's Forum 2004

Sébastien Dominé

NVIDIA

Agenda

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

Agenda

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

Performance Tools Survey

- NVPerfHUD
  - Direct3D9 Performance HUD
- NVShaderPerf
  - Offline 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 API calls to assist with performance analysis
- Only works on your own application

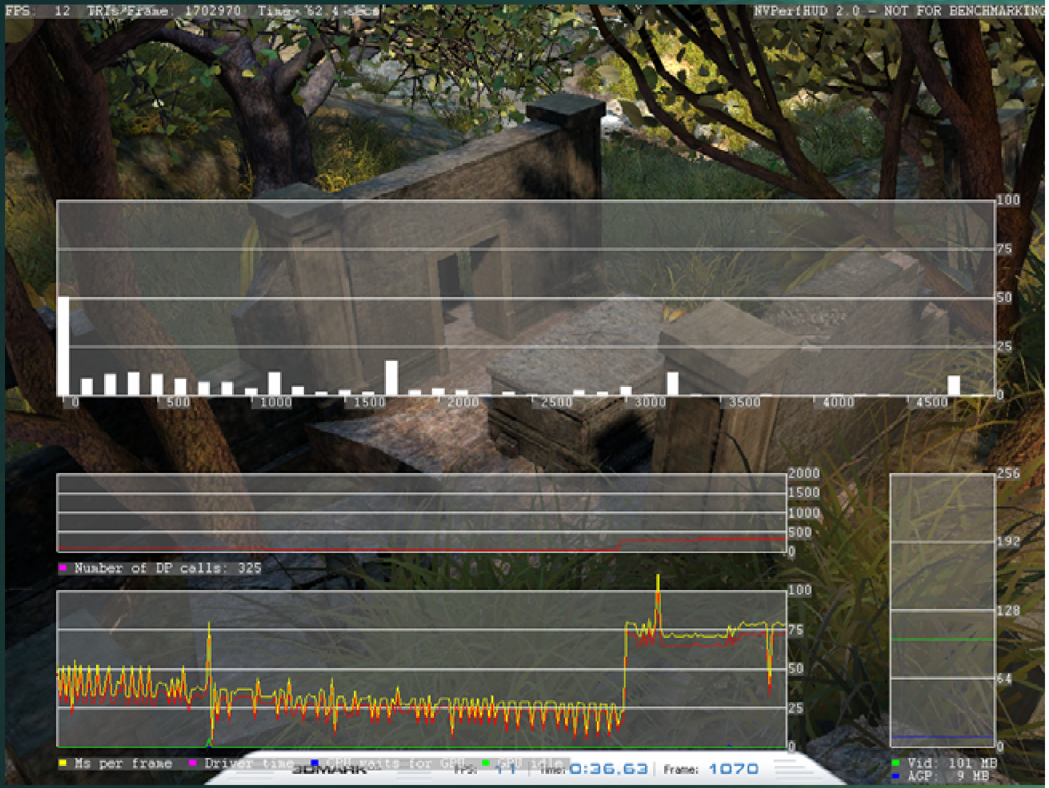
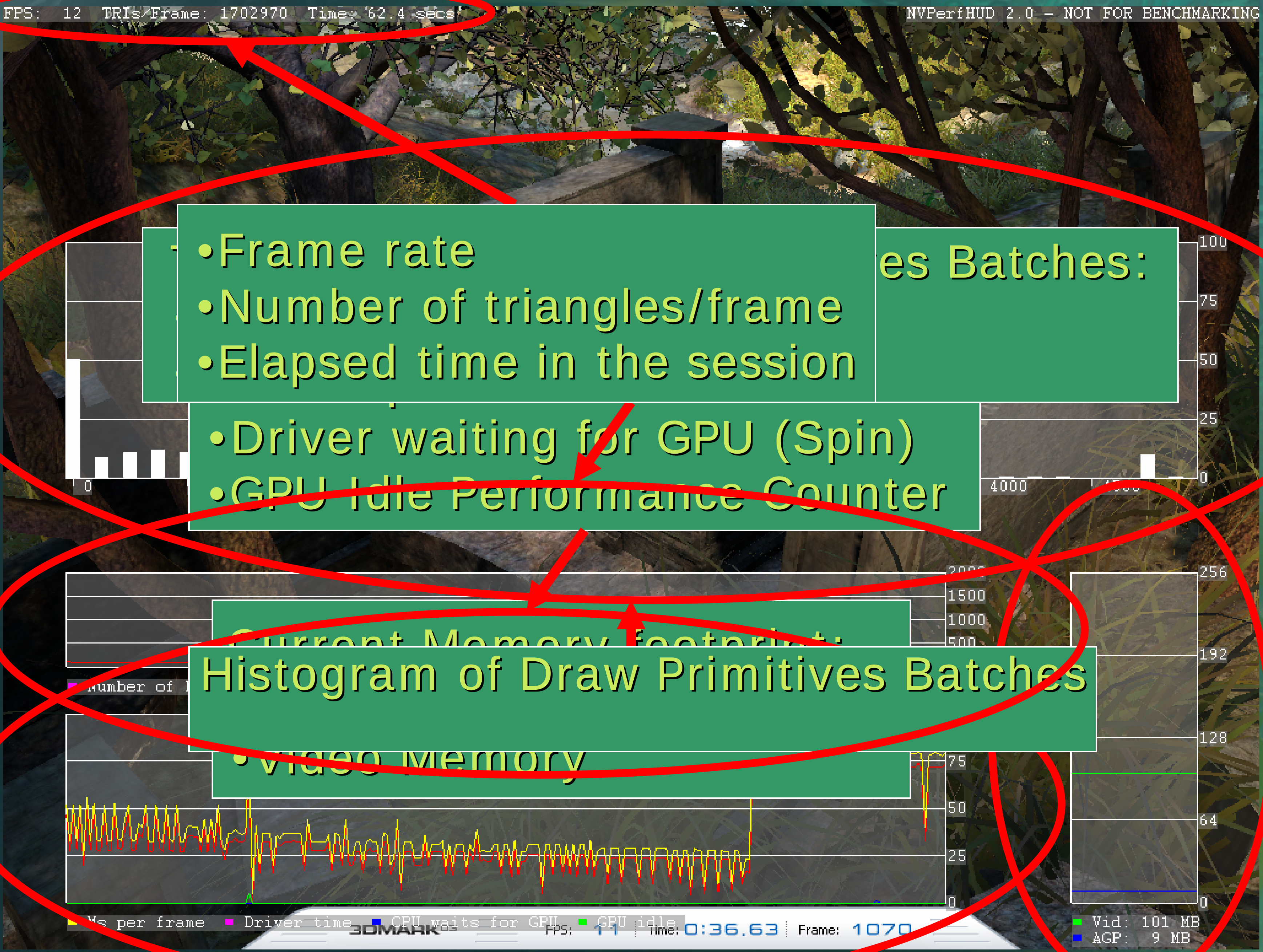


Image courtesy of FutureMark Corp.



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

es Batches:

- Driver waiting for GPU (Spin)
- GPU Idle Performance Counter

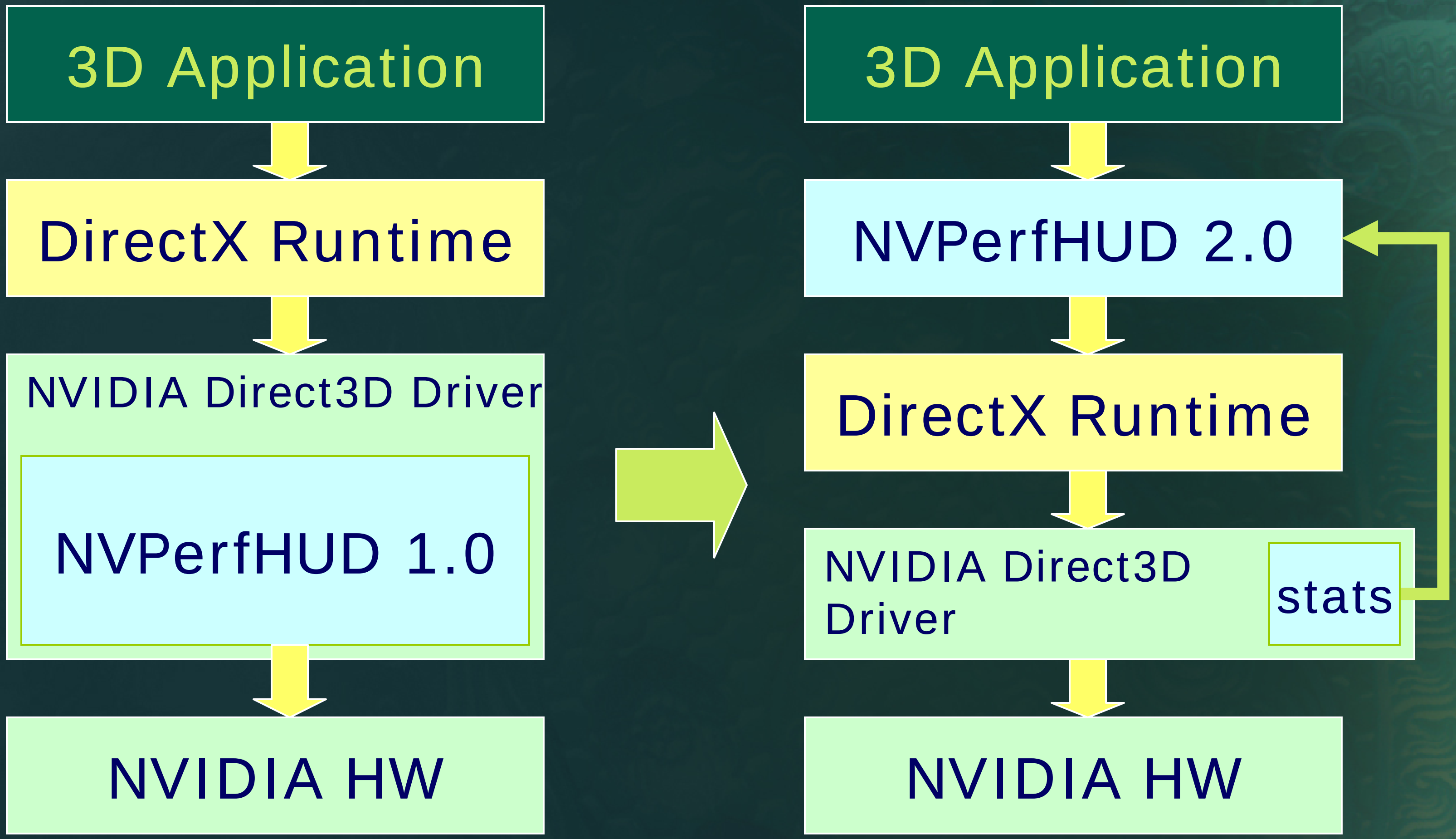
Current Memory footprint:

- Video Memory

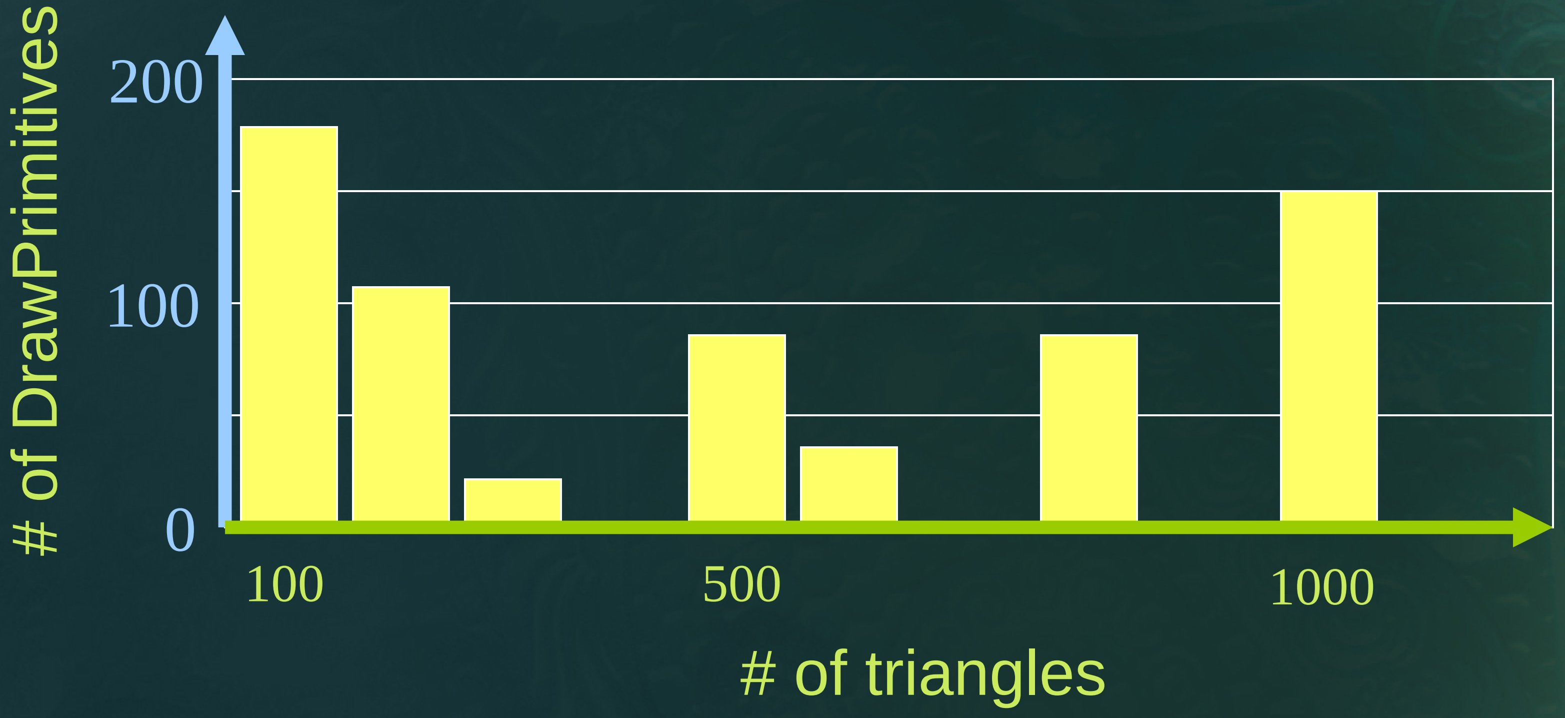
Histogram of Draw Primitives Batches

Image courtesy of FutureMark Corp.

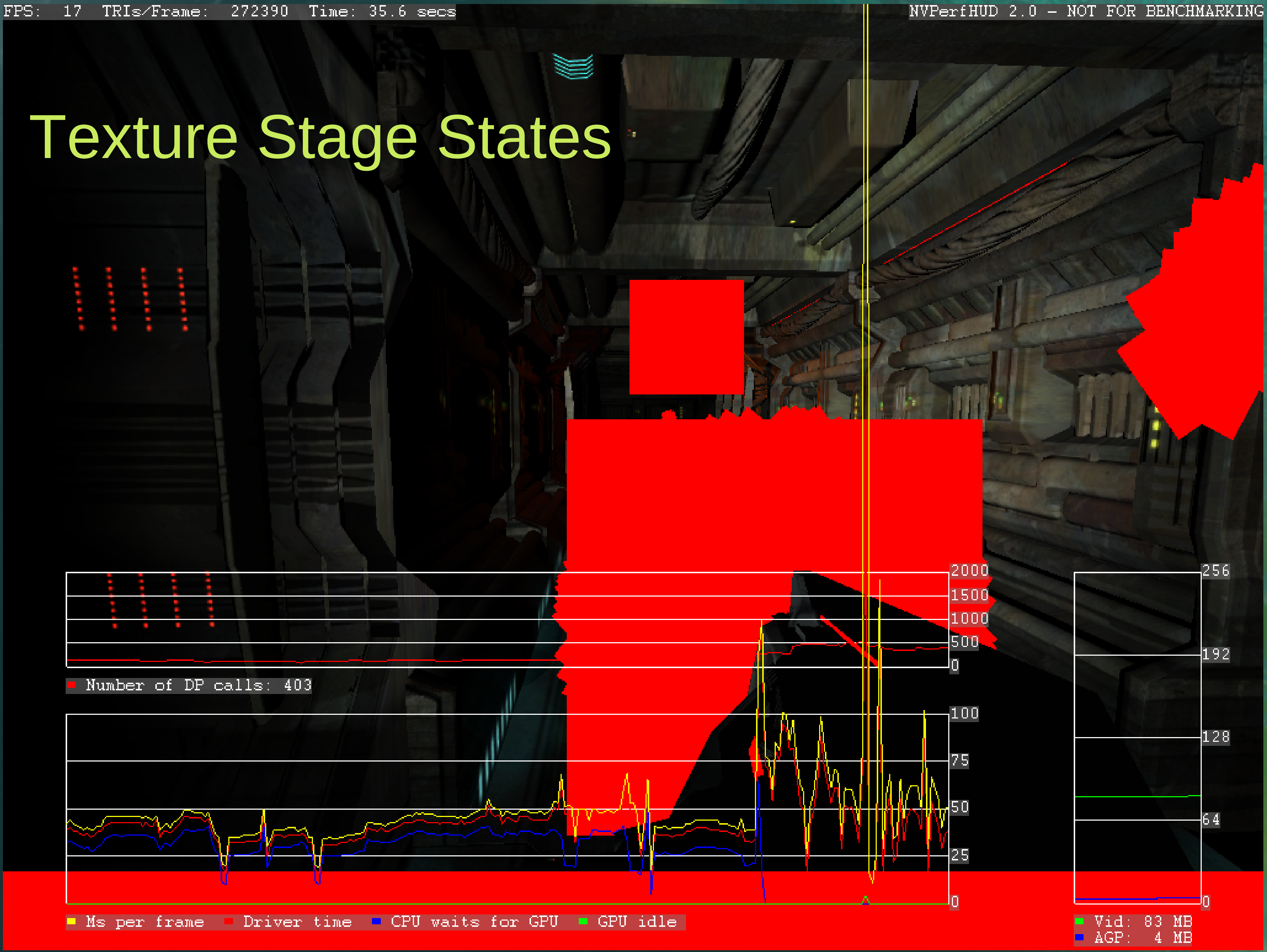
What’s new in 2.0?



- DrawPrimitives/DrawIndexedPrimitives Histogram



Texture Stage States



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

NVPerfHUD 2.0 - NOT FOR BENCHMARKING

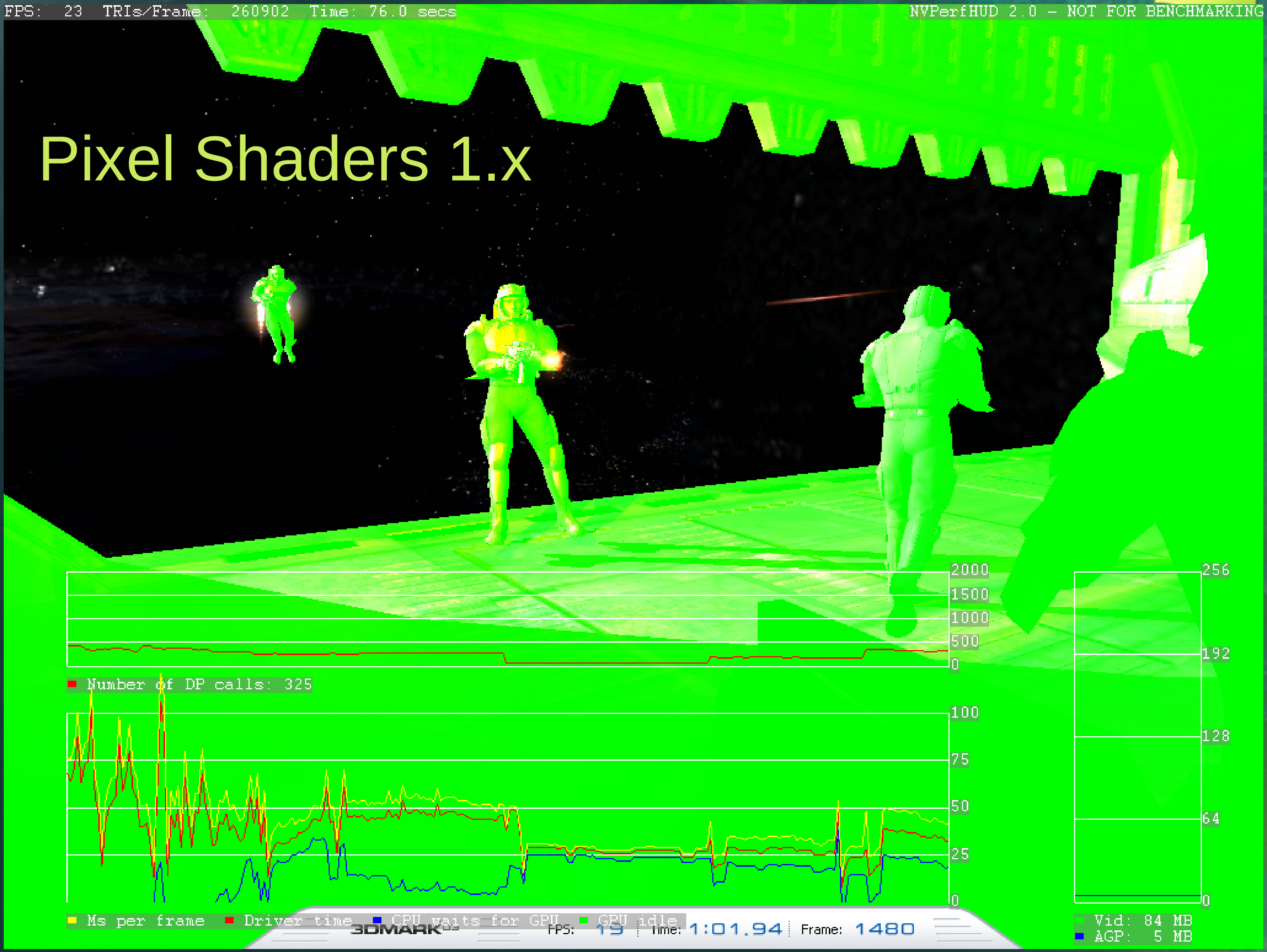
🟥 Number of DP calls: 403

🟨 Ms per frame 🟥 Driver time 🟦 CPU waits for GPU 🟩 GPU idle

🟩 Vid: 83 MB
🟦 AGP: 4 MB

Image courtesy of FutureMark Corp.

Pixel Shaders 1.x



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

NVPerfHUD 2.0 - NOT FOR BENCHMARKING

Number of DP calls: 325

Vid: 84 MB
AGP: 5 MB

Image courtesy of FutureMark Corp.

Pixel Shaders 2.0

Pixel Shaders 3.0

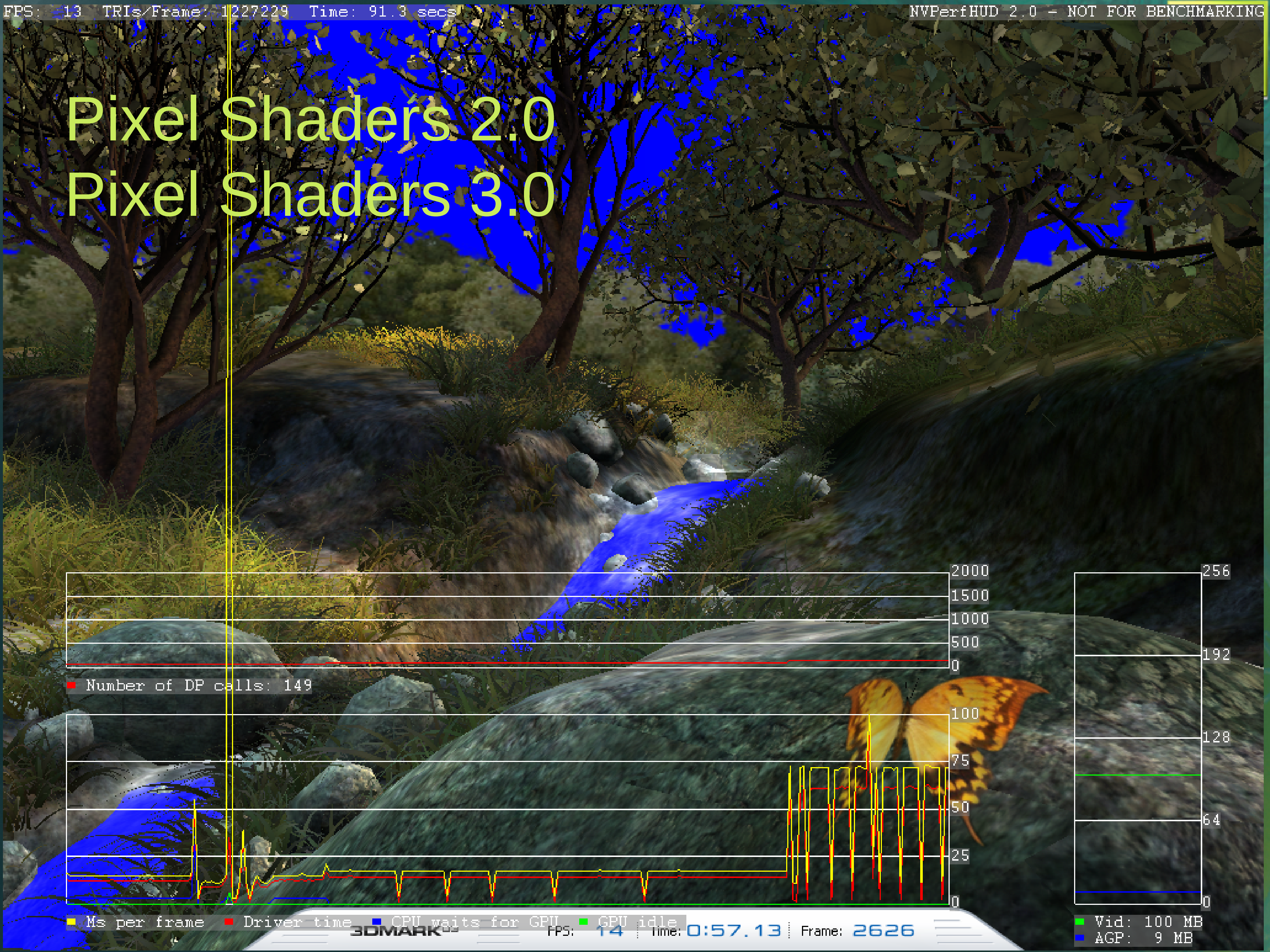


Image courtesy of FutureMark Corp.

2x2 Texture replacement

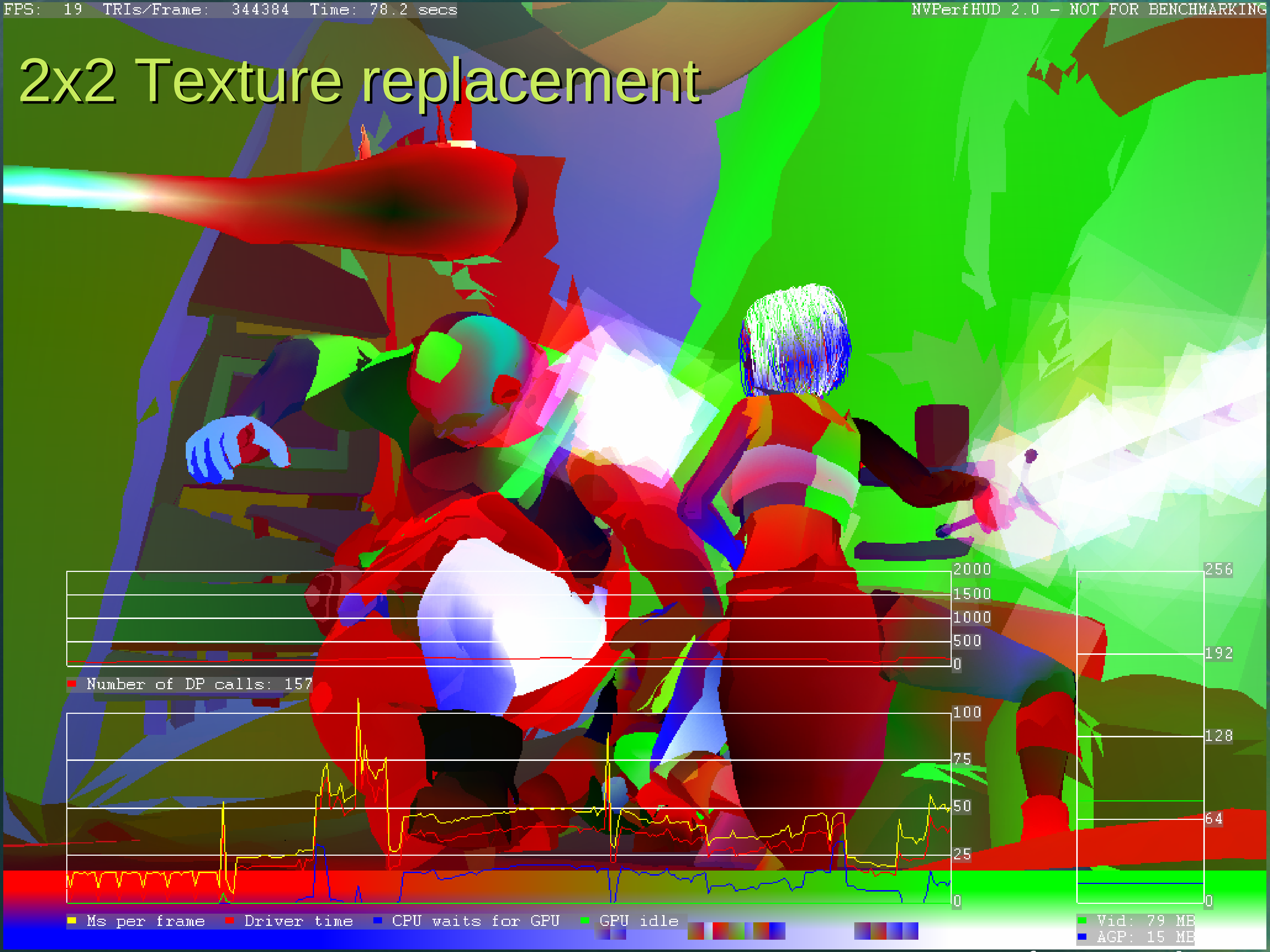
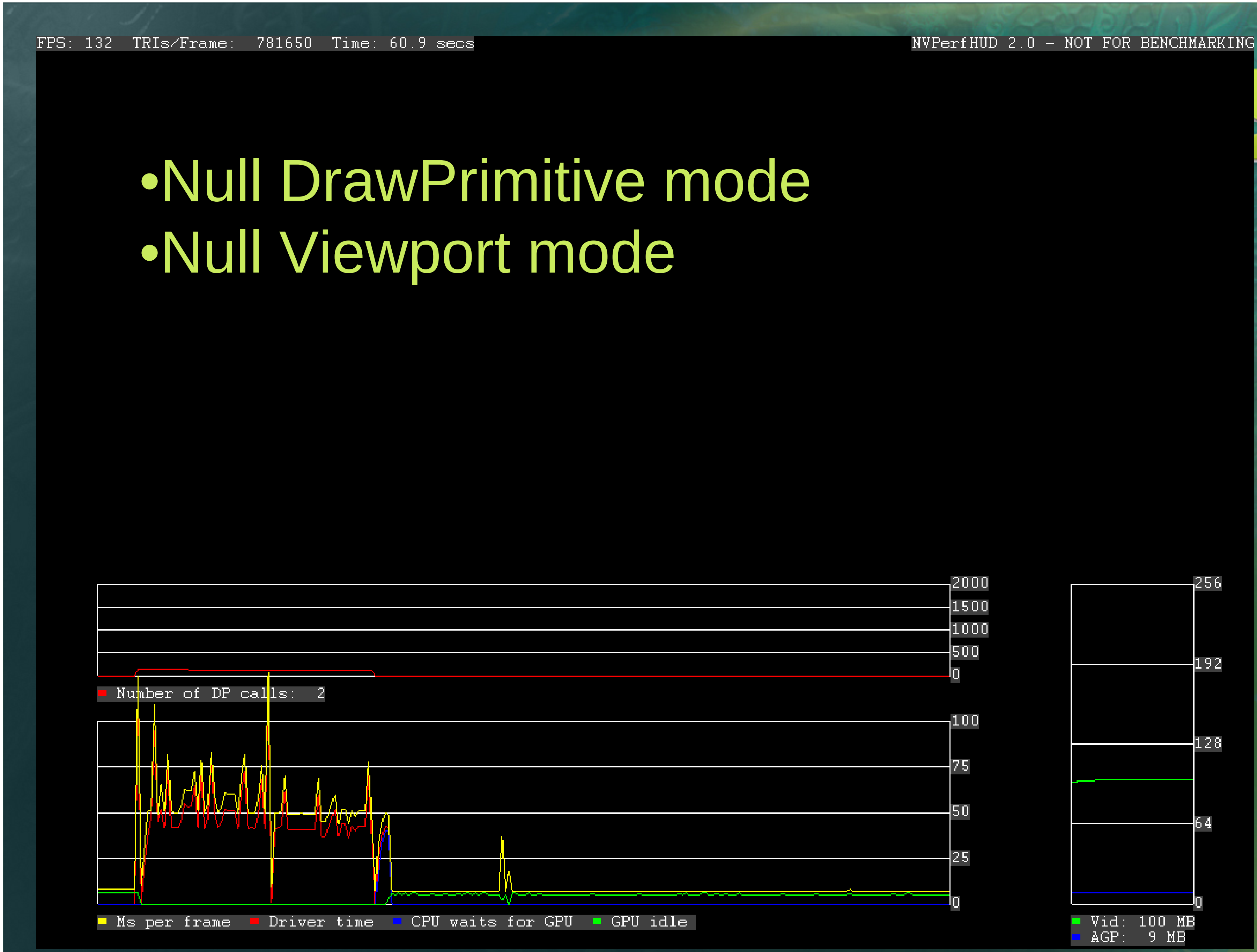


Image courtesy of FutureMark Corp.



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

NVPerfHUD 2.0 – NOT FOR BENCHMARKING

- Null DrawPrimitive mode
- Null Viewport mode

🟥 Number of DP calls: 2

🟩 Vid: 100 MB
🟦 AGP: 9 MB

Image courtesy of FutureMark Corp.

NVPerfHUD - Overhead

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

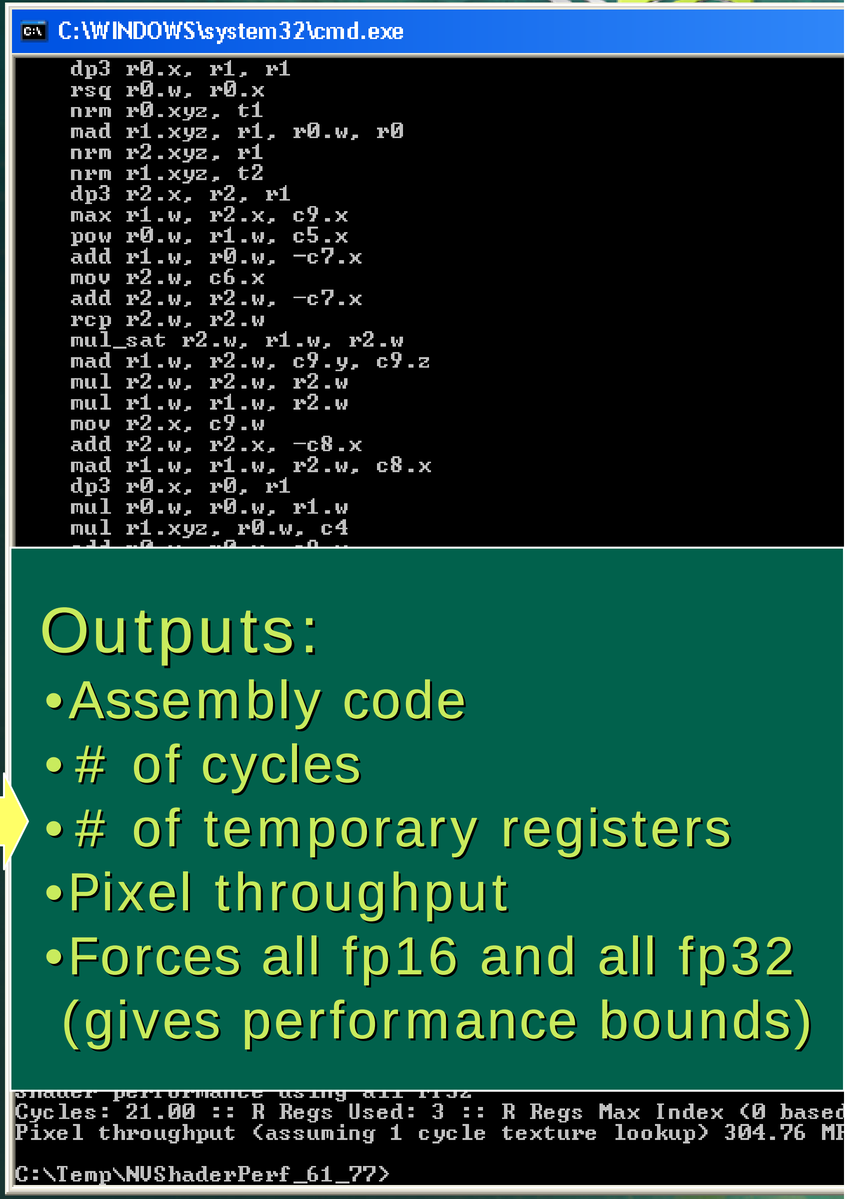
NVShaderPerf

Inputs:
- HLSL
- PS1.x,PS2.x,PS3.x
- VS1.x,VS2.x, VS3.x

GPU Arch:
- GeForce FX (NV3X)
- GeForce 6 Series (NV4X)
- Quadro FX (NV3X+NV4X)

NVShaderPerf

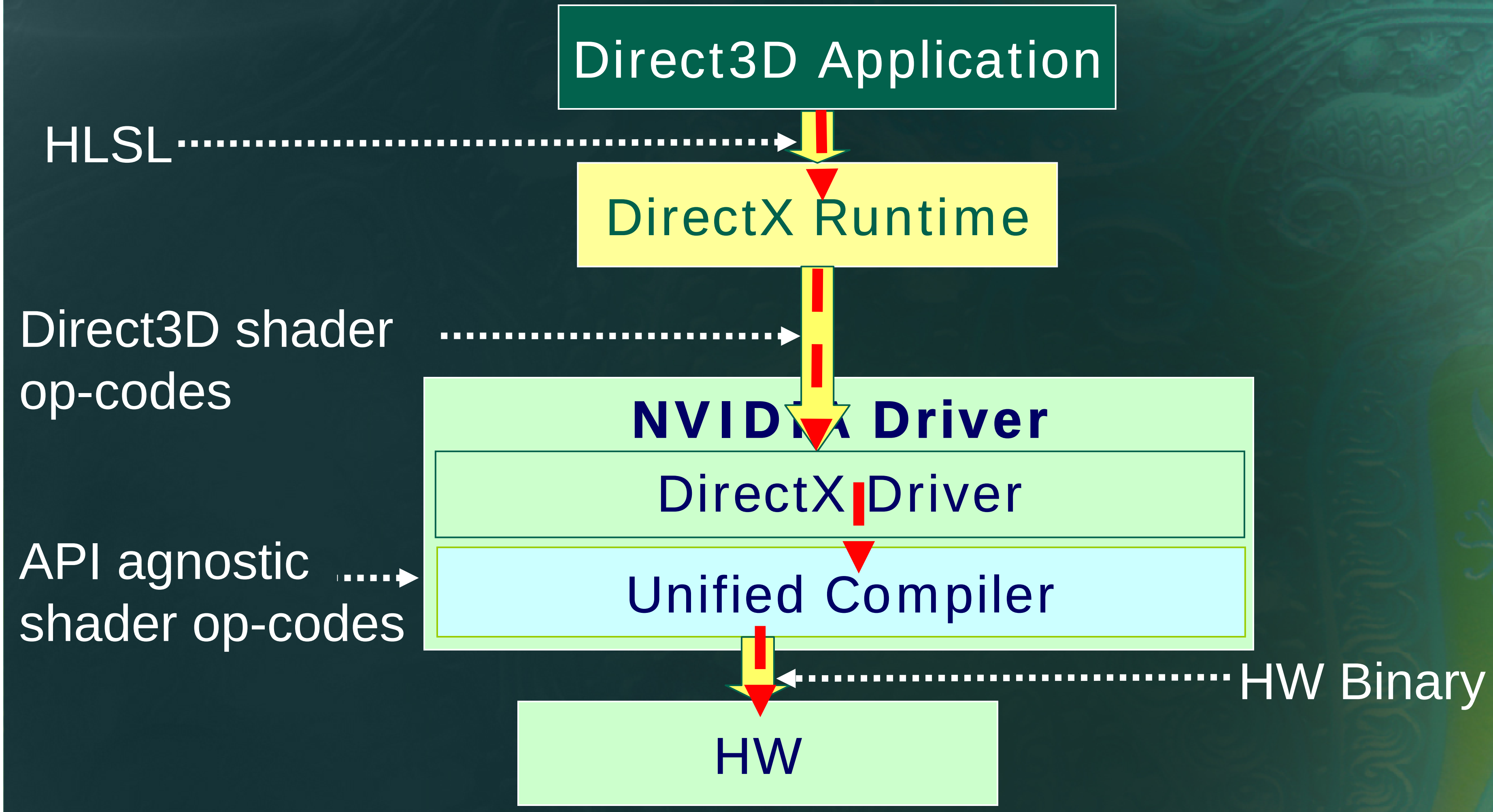
Outputs:
- Assembly code
- # of cycles
- # of temporary registers
- Pixel throughput
- Forces all fp16 and all fp32 (gives performance bounds)



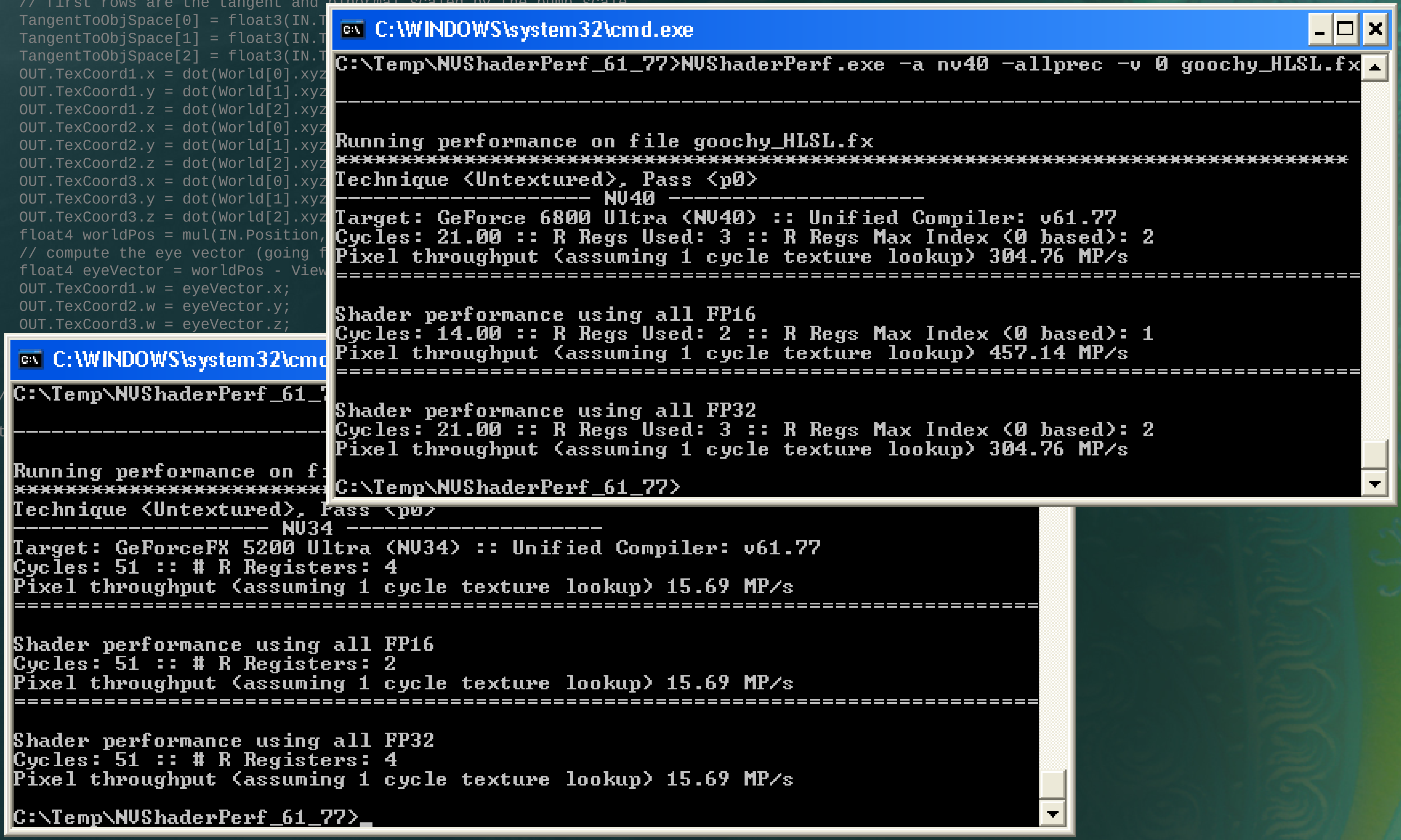
```
    dp3 r0.x, r1, r1
    rsq r0.w, r0.x
    nrm r0.xyz, t1
    mad r1.xyz, r1, r0.w, r0
    nrm r2.xyz, r1
    nrm r1.xyz, t2
    dp3 r2.x, r2, r1
    max r1.w, r2.x, c9.x
    pow r0.w, r1.w, c5.x
    add r1.w, r0.w, -c7.x
    mov r2.w, c6.x
    add r2.w, r2.w, -c7.x
    rcp r2.w, r2.w
    mul_sat 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
```

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

C:\Temp\NVShaderPerf_61_77>
```

NVShaderPerf



NVShaderPerf - Example

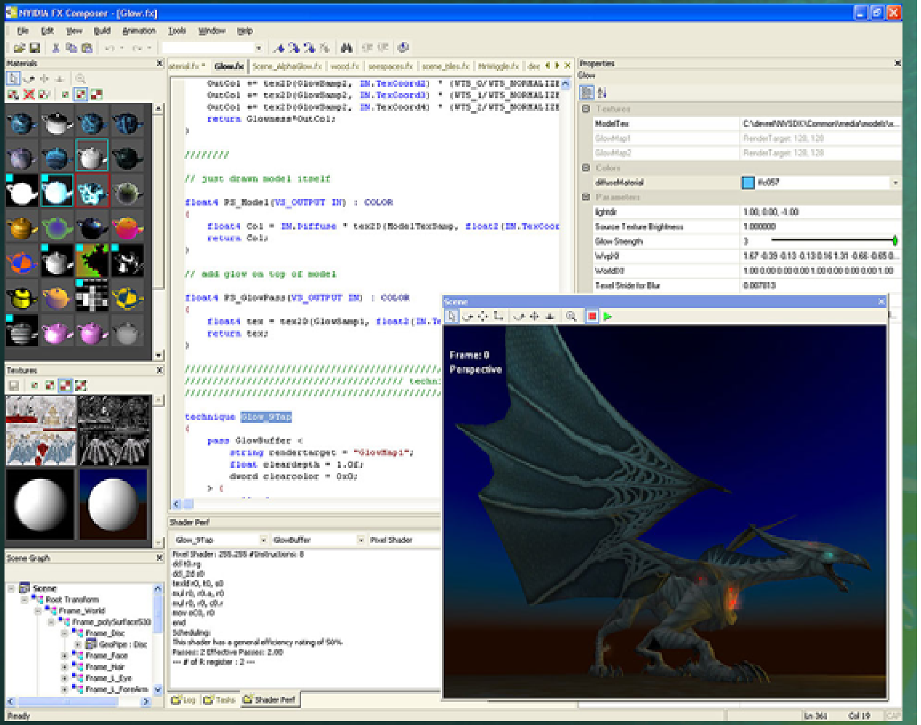


```
C:\Temp\NVShaderPerf_61_77>NVShaderPerf.exe -a nv40 -allprec -v 0 goochy_HLSL.fx

-------------------------------------------------------------------------------

Running performance on file goochy_HLSL.fx
*******************************************************************************
Technique <Untextured>, Pass <p0>
-------------------- NV40 --------------------
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>
```

```
C:\Temp\NVShaderPerf_61_7

-------------------------

Running performance on f
*************************
Technique <Untextured>, Pass <p0>
-------------------- NV34 --------------------
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
Pixel throughput (assuming 1 cycle texture lookup) 15.69 MP/s

C:\Temp\NVShaderPerf_61_77>_
```

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



```
    OutCol += tex2D(GlowSamp2, IN.TexCoord2) * (WT5_0/WT5_NORMALIZE
    OutCol += tex2D(GlowSamp2, IN.TexCoord3) * (WT5_1/WT5_NORMALIZE
    OutCol += tex2D(GlowSamp2, IN.TexCoord4) * (WT5_2/WT5_NORMALIZE
    return Glowness*OutCol;
}

////////

// just drawn model itself

float4 PS_Model(VS_OUTPUT IN) : COLOR
{
    float4 Col = IN.Diffuse * tex2D(ModelTexSamp, float2(IN.TexCoor
    return Col;
}

// add glow on top of model

float4 PS_GlowPass(VS_OUTPUT IN) : COLOR
{
    float4 tex = tex2D(GlowSamp1, float2(IN.Te
    return tex;
}

//////////////////////////////////////////
////////////////////////////////////// techn
//////////////////////////////////////////

technique Glow_9Tap
{
    pass GlowBuffer <
        string rendertarget = "GlowMap1";
        float cleardepth = 1.0f;
        dword clearcolor = 0x0;
    > {
```

Properties	
Textures	
ModelTex	C:\devrel\NVSDK\Common\media\models\...
GlowMap1	RenderTarget: 128, 128
GlowMap2	RenderTarget: 128, 128
Colors	
diffuseMaterial	#c0f7
Parameters	
lightdir	1.00, 0.00, -1.00
Source Texture Brightness	1.000000
Glow Strength	3
WvpXf	1.67 -0.39 -0.13 -0.13 0.16 1.31 -0.66 -0.65 0...
WorldXf	1.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 1.00
Texel Stride for Blur	0.007813

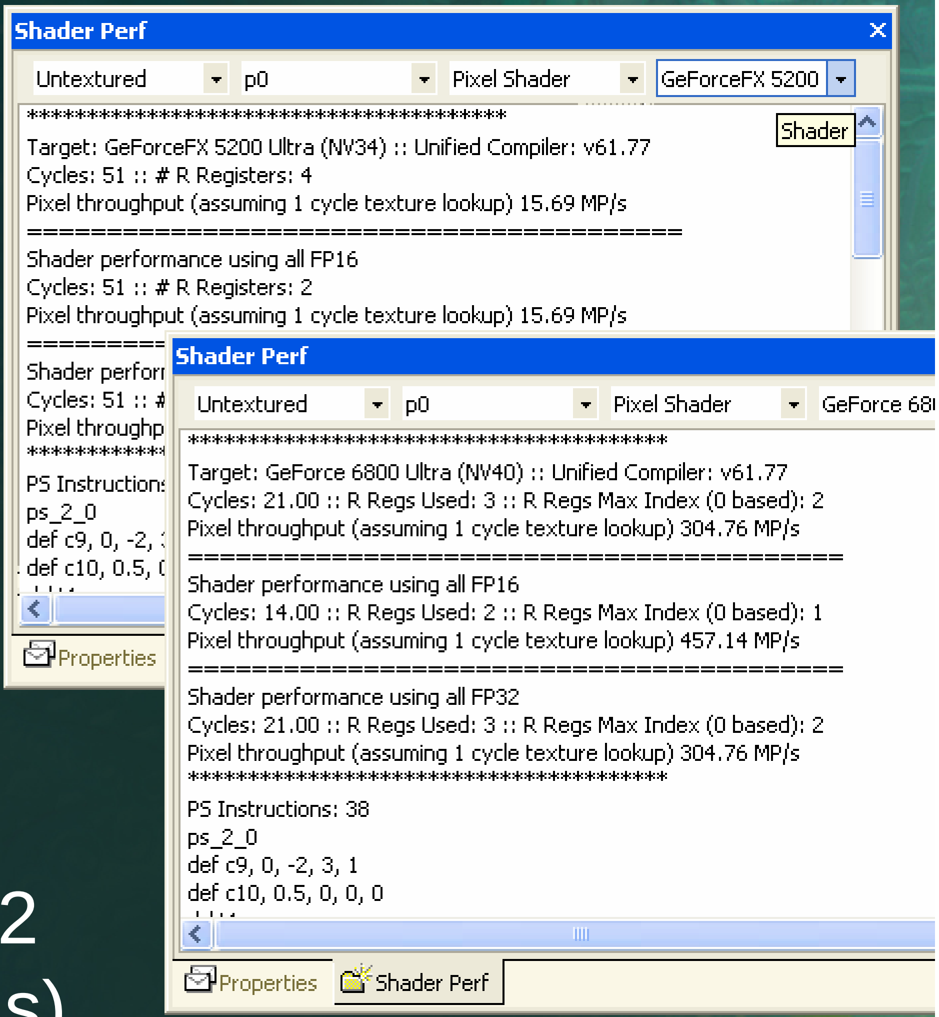
Glow_9Tap · GlowBuffer · Pixel Shader

```
Pixel Shader : 255.255 #Instructions: 8
dcl t0.rg
dcl_2d s0
texld r0, t0, s0
mul r0, r0.a, r0
mul r0, r0, c0.r
mov oC0, r0
end
Scheduling:
This shader has a general efficiency rating of 50%
Passes: 2 Effective Passes: 2.00
--- # of R register : 2 ---
```

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)



```
****************************************
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 perfor
Cycles: 51 :: #
Pixel throughp
***********
PS Instructions
ps_2_0
def c9, 0, -2, 3
def c10, 0.5, 0
```

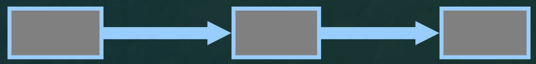
```
****************************************
Target: GeForce 6800 Ultra (NV40) :: Unified Compiler: v61.77
Cycles: 21.00 :: R Regs Used: 3 :: R Regs Max Index (0 based): 2
Pixel throughput (assuming 1 cycle texture lookup) 304.76 MP/s
============================================
Shader performance using 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
```

Agenda

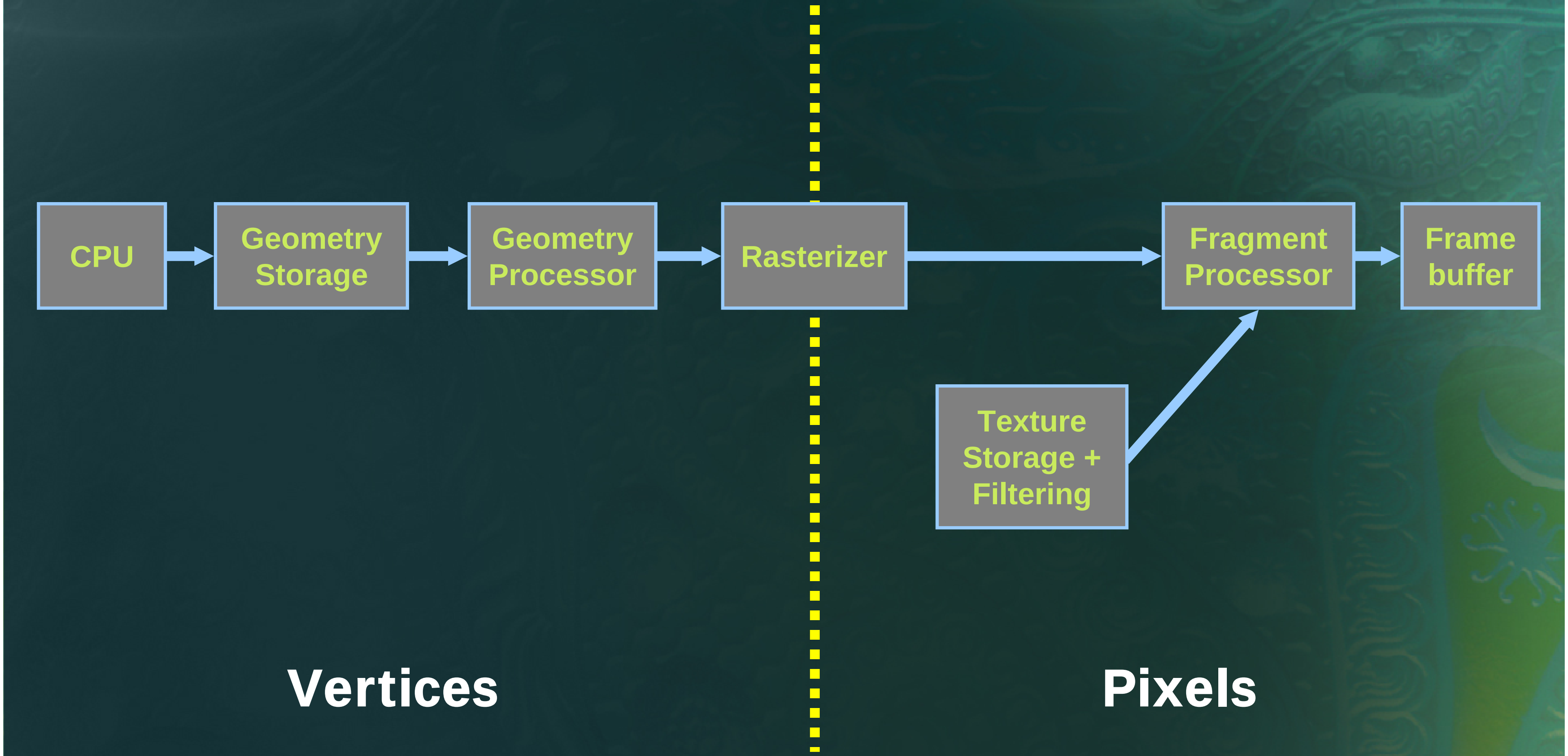
- Performance Tools Survey
- Practice
- Next generation Performance Tools
- Conclusion
- Q & A

Basic Principles

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

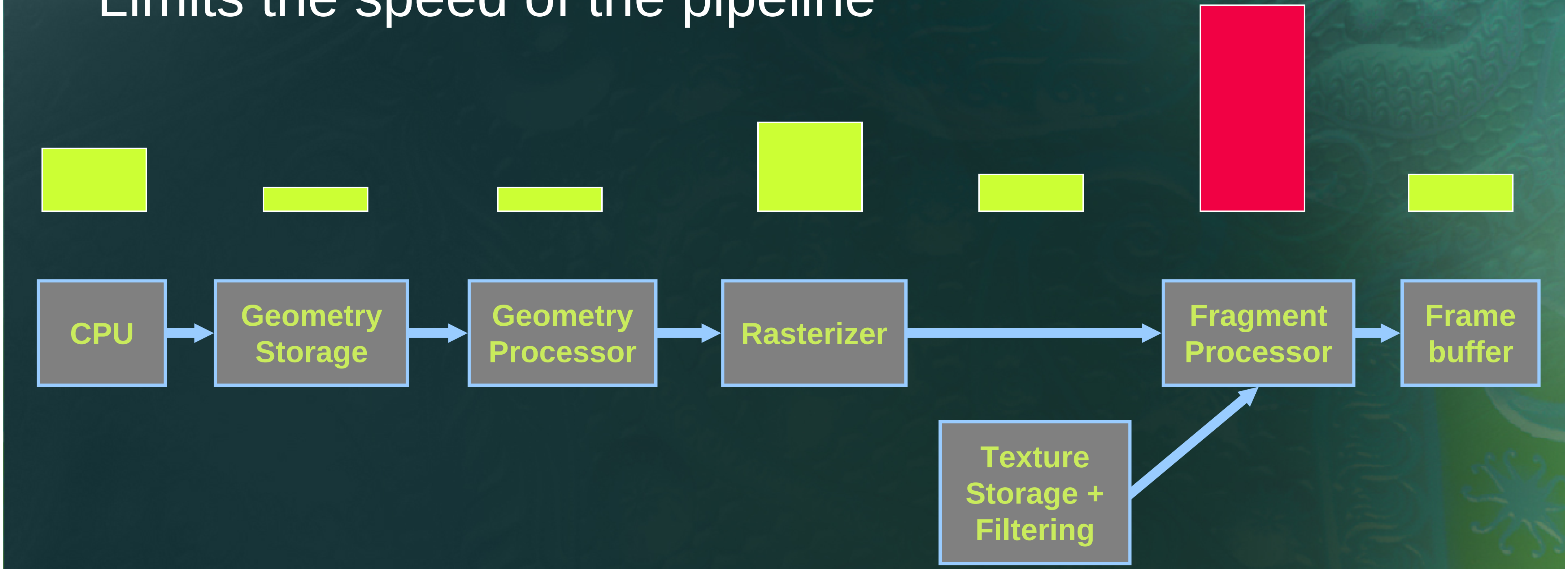


Pipelined Architecture (simplified view)



The Terrible Bottleneck

Limits the speed of the pipeline

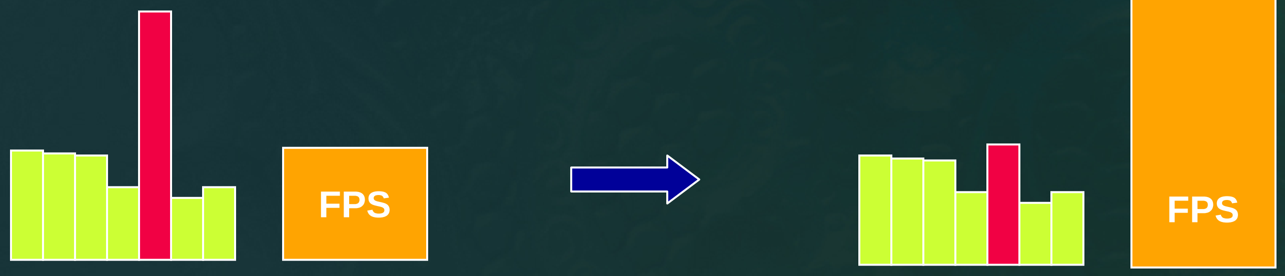


Bottleneck Identification

- 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

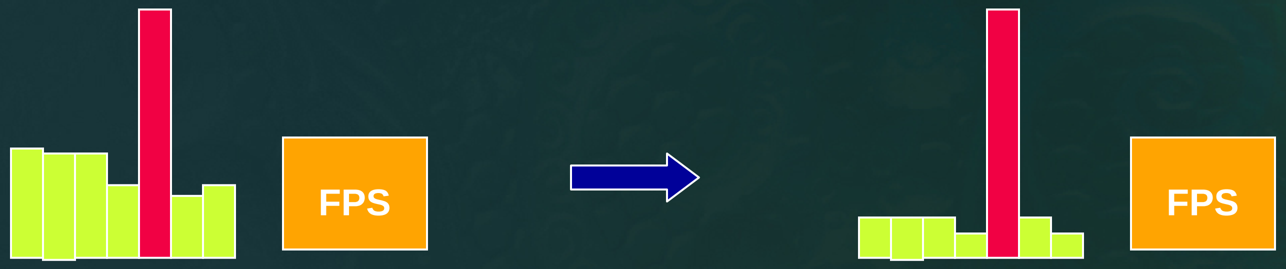
- Modify the stage itself
  - By decreasing its workload



  - 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



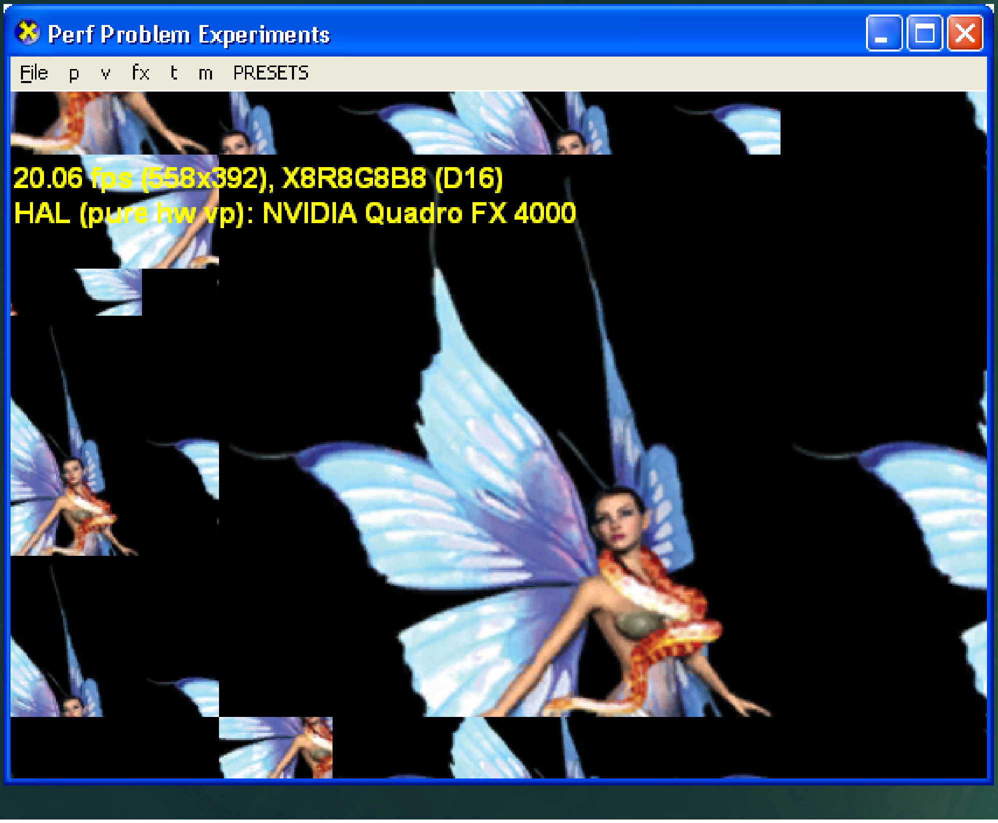
  - If performance doesn’t change significantly, then you know this is the bottleneck
  - Careful not to change the workload of this stage!

Practice

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

Practice: Example 1

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



```
20.06 fps (558x392), X8R8G8B8 (D16)
HAL (pure hw vp): NVIDIA Quadro FX 4000
```

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

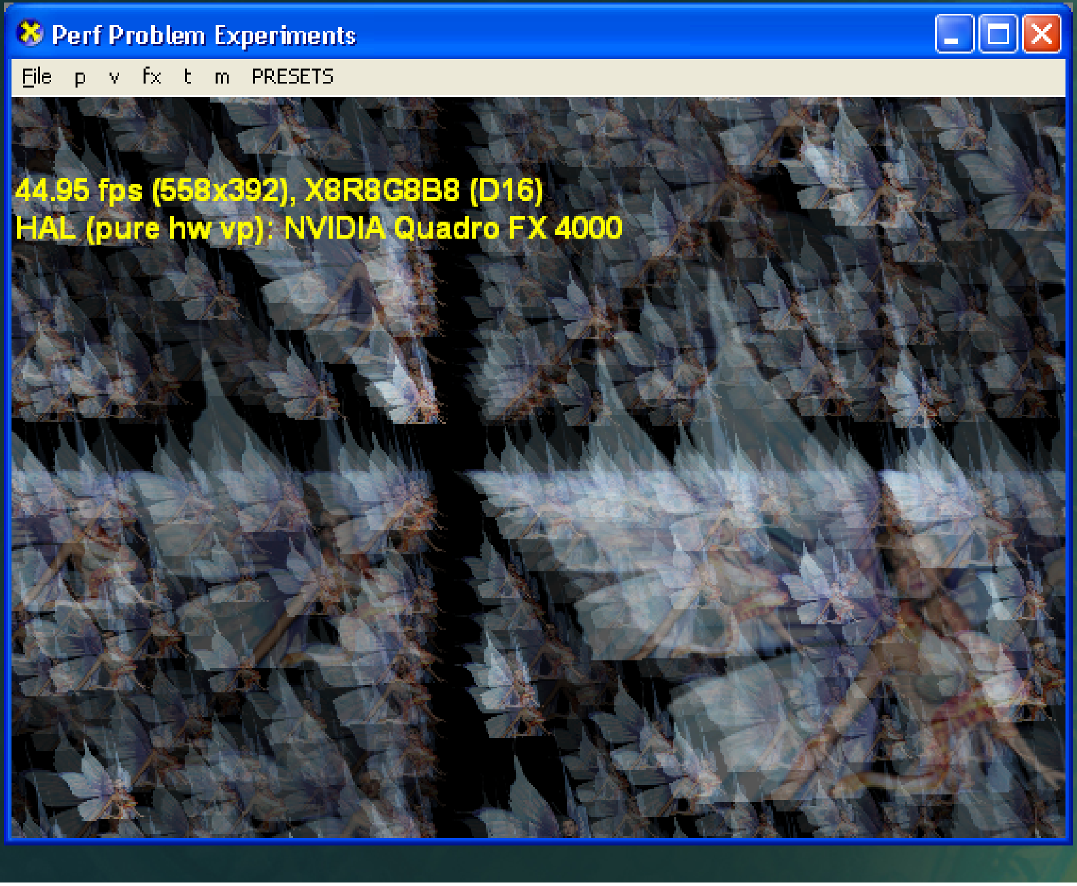
- Dynamic Vertex Buffer
  - GOOD Lock flags

```
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



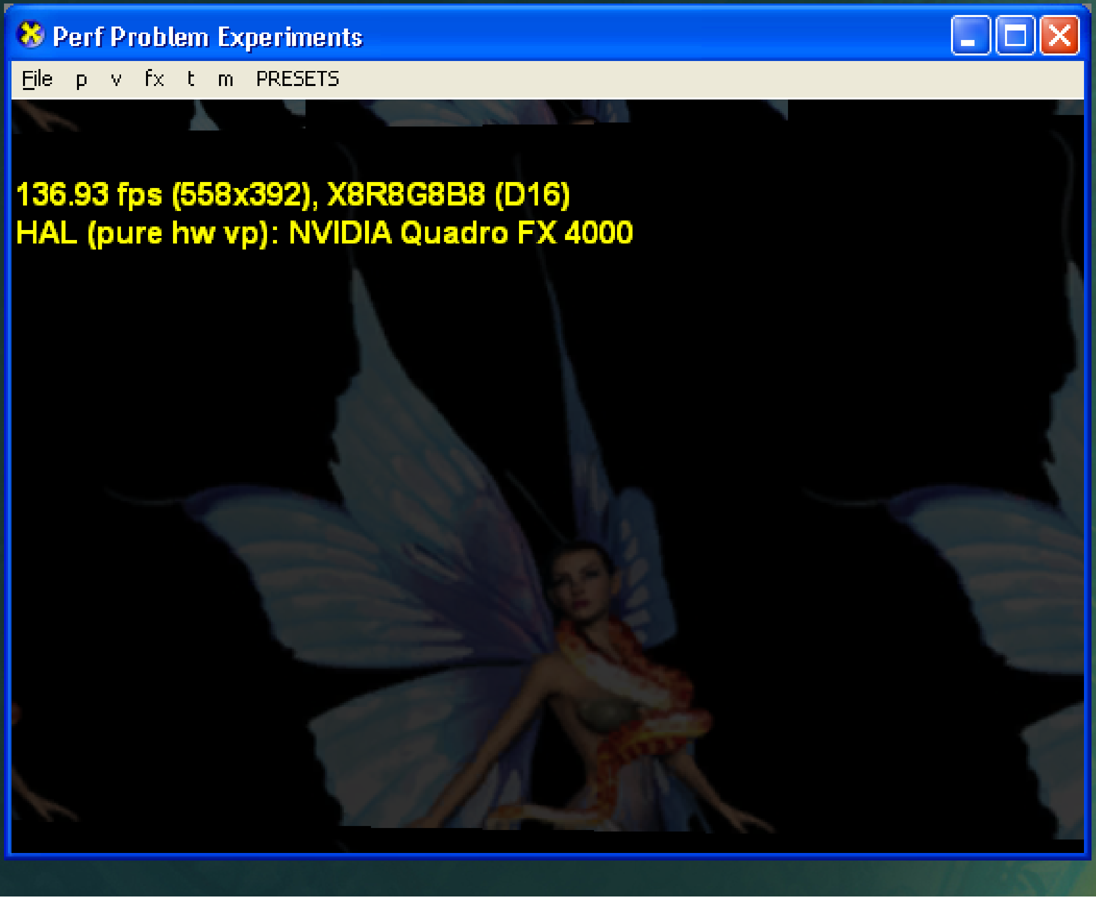
44.95 fps (558x392), X8R8G8B8 (D16)
HAL (pure hw vp): NVIDIA Quadro FX 4000

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

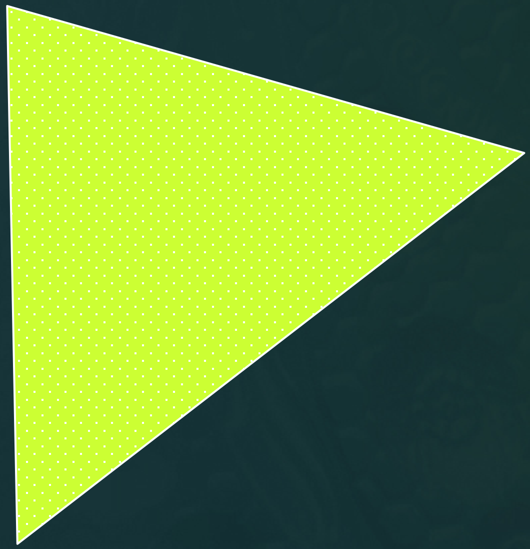
- Another slow scene
  - Who wants a prize?



```
136.93 fps (558x392), X8R8G8B8 (D16)
HAL (pure hw vp): NVIDIA Quadro FX 4000
```

Practice: Example 3

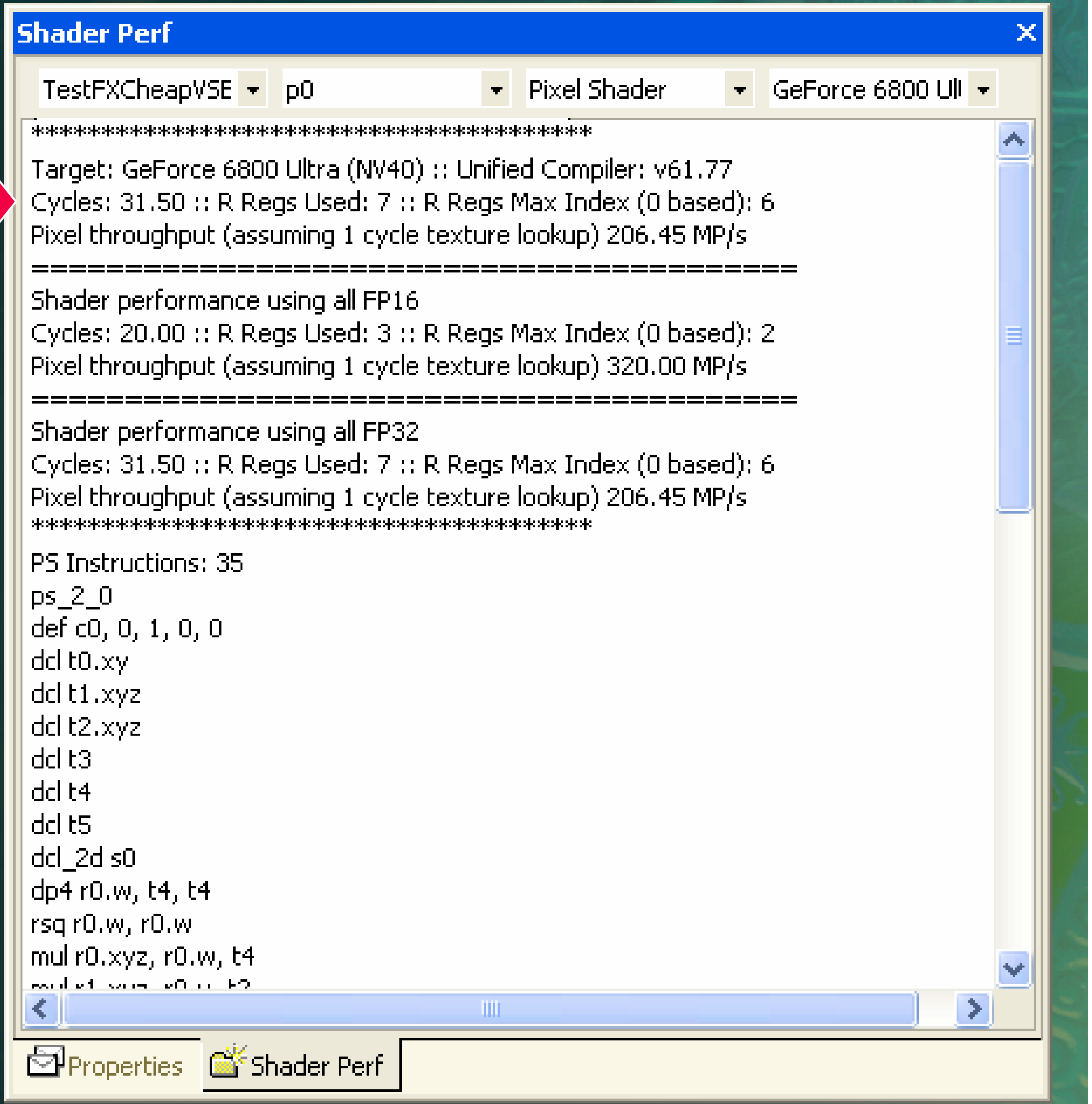
- Expensive pixel shader
  - Can have huge performance effect
  - Only 3 verts, but maybe a million pixels
    - That's only 1024x1024



Look at all the pixels!!

Practice: Example 3

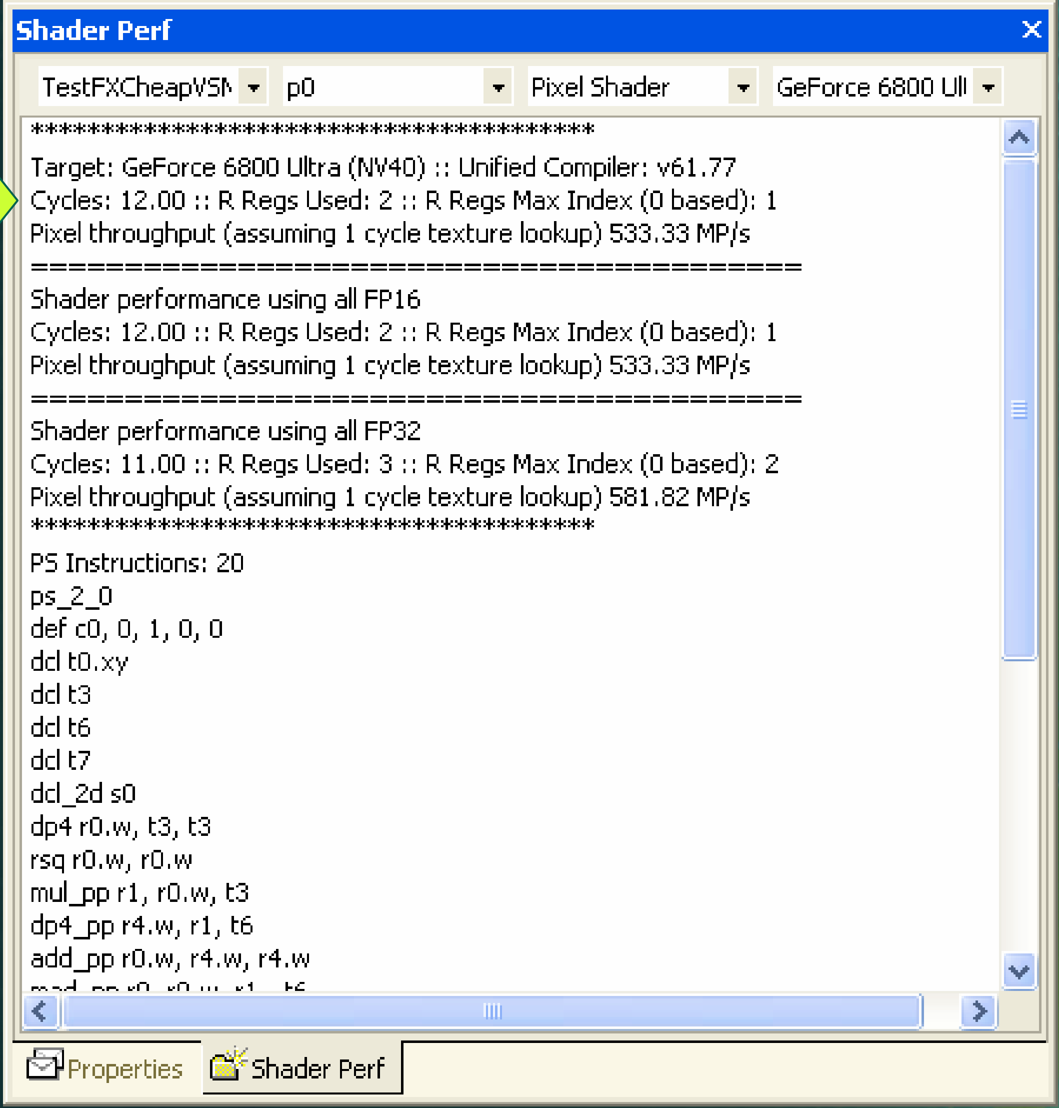
32 cycles BAD



```
****************************************
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 r1.xyz, r0.w, t3
```

Practice: Example 3

12 cycles GOOD



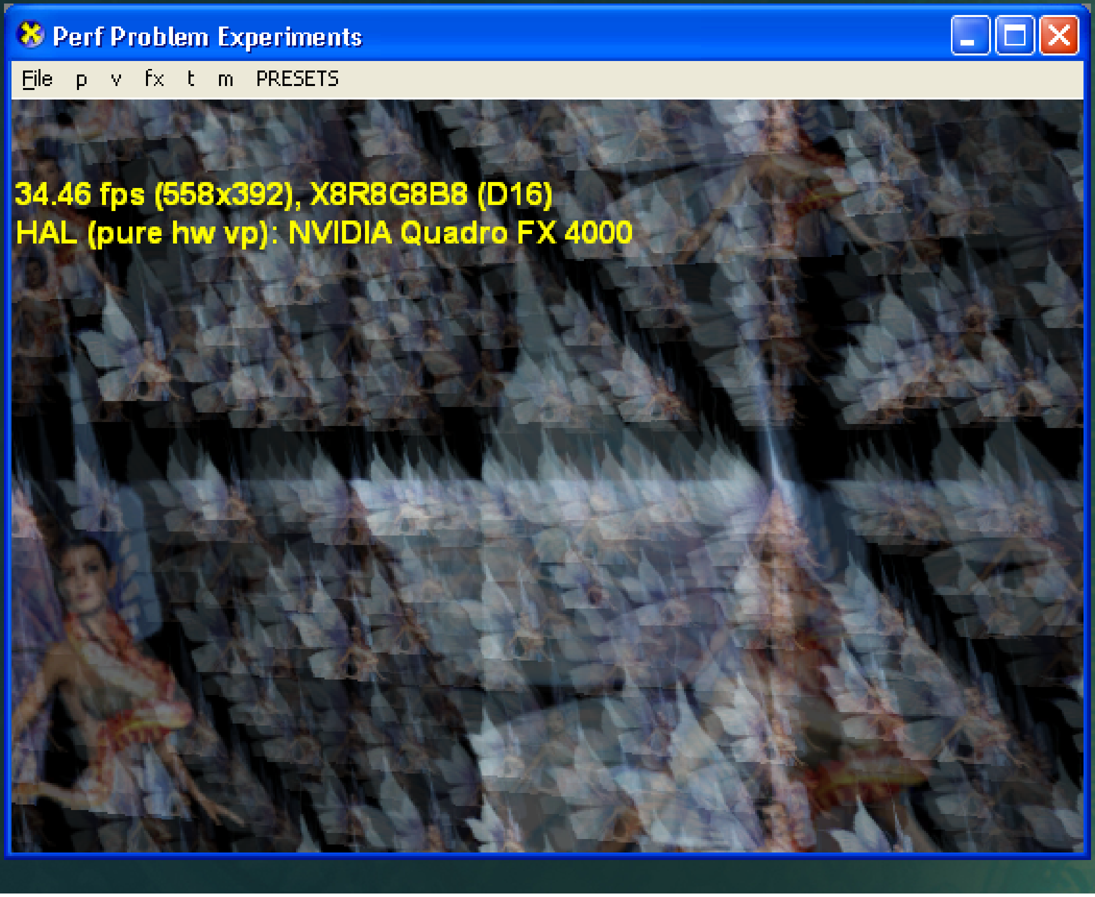
```
*****************************************
Target: GeForce 6800 Ultra (NV40) :: Unified Compiler: v61.77
Cycles: 12.00 :: R Regs Used: 2 :: R Regs Max Index (0 based): 1
Pixel throughput (assuming 1 cycle texture lookup) 533.33 MP/s
==============================================
Shader performance using all FP16
Cycles: 12.00 :: R Regs Used: 2 :: R Regs Max Index (0 based): 1
Pixel throughput (assuming 1 cycle texture lookup) 533.33 MP/s
==============================================
Shader performance using all FP32
Cycles: 11.00 :: R Regs Used: 3 :: R Regs Max Index (0 based): 2
Pixel throughput (assuming 1 cycle texture lookup) 581.82 MP/s
*****************************************
PS Instructions: 20
ps_2_0
def c0, 0, 1, 0, 0
dcl t0.xy
dcl t3
dcl t6
dcl t7
dcl_2d s0
dp4 r0.w, t3, t3
rsq r0.w, r0.w
mul_pp r1, r0.w, t3
dp4_pp r4.w, r1, t6
add_pp r0.w, r4.w, r4.w
```

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

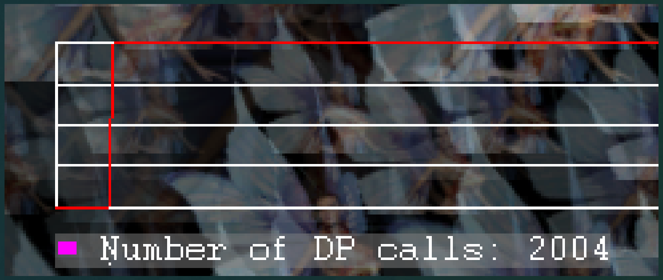
- The last one
  - Audience: there are no more prizes, but we've locked the doors



34.46 fps (558x392), X8R8G8B8 (D16)
HAL (pure hw vp): NVIDIA Quadro FX 4000

Practice: Example 4

- Too many batches
  - Was sending every quad as it’s own batch
  - Instead, group quads into one big VB then send that with one call



■ Number of DP calls: 2004

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

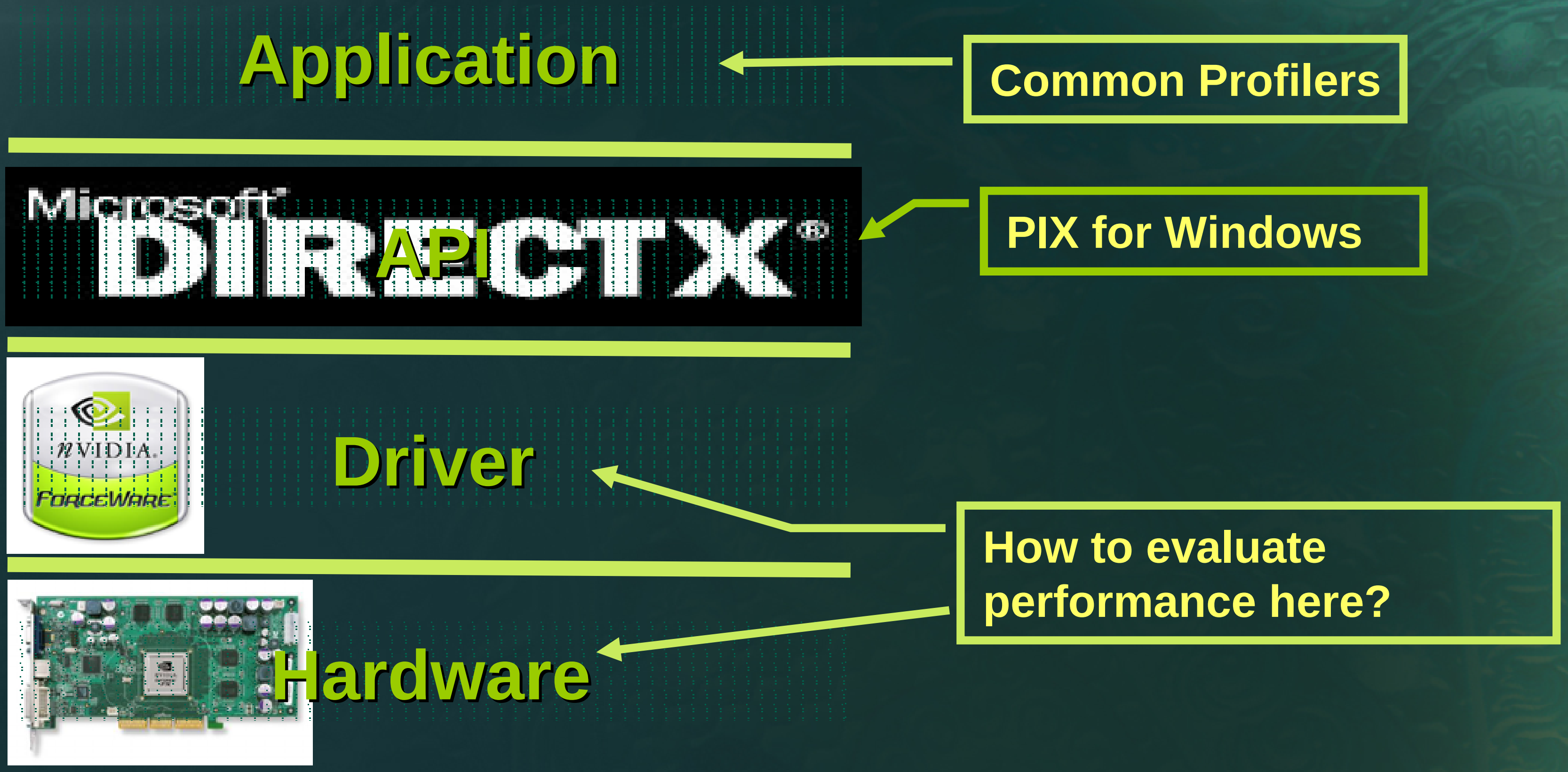
Agenda

- 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 DirectX

Problem



Application

API

Driver

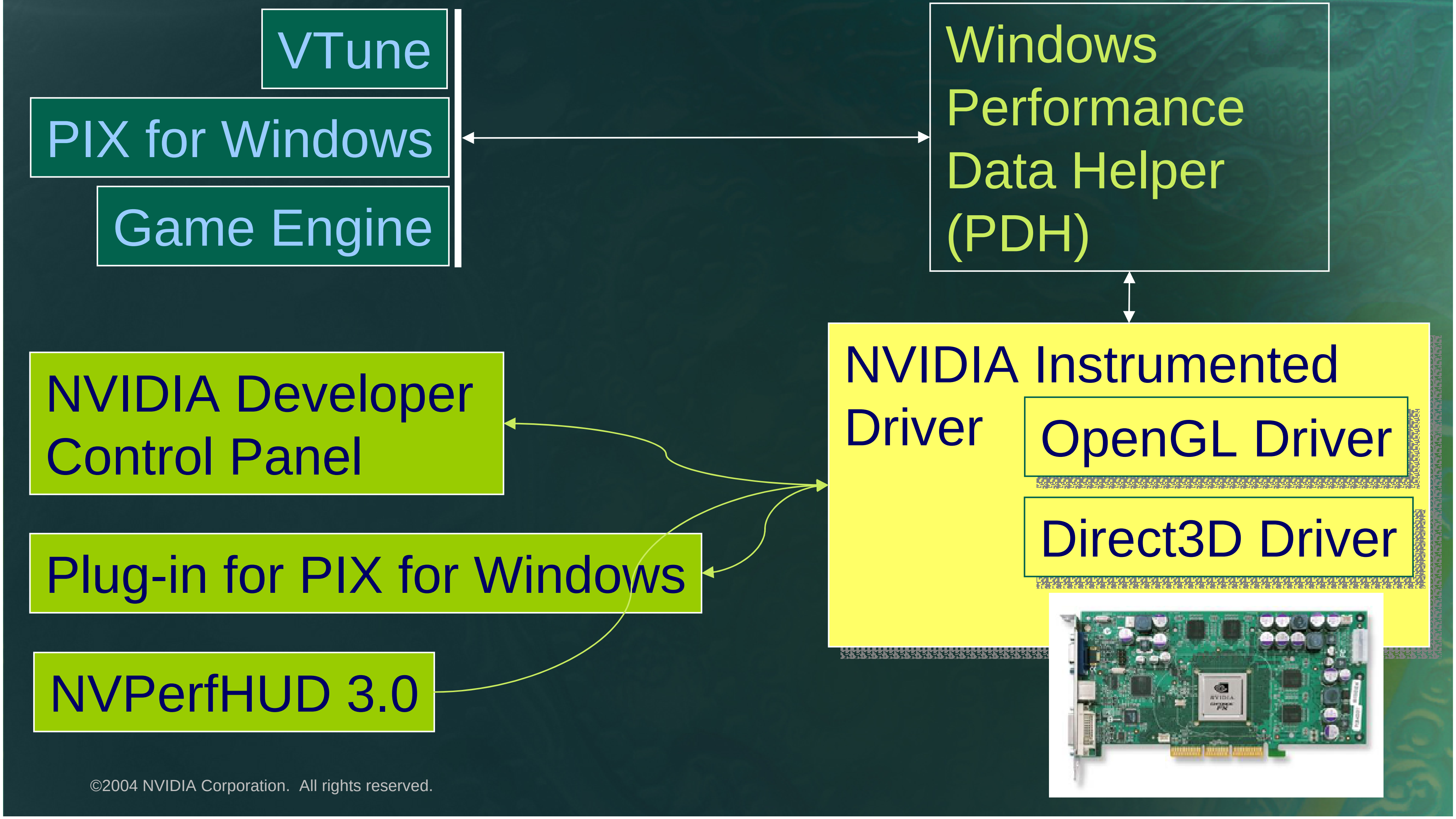
Hardware

Common Profilers

PIX for Windows

How to evaluate performance here?

Solution



VTune

PIX for Windows

Game Engine

Windows Performance Data Helper (PDH)

NVIDIA Instrumented Driver

OpenGL Driver

Direct3D Driver

NVIDIA Developer Control Panel

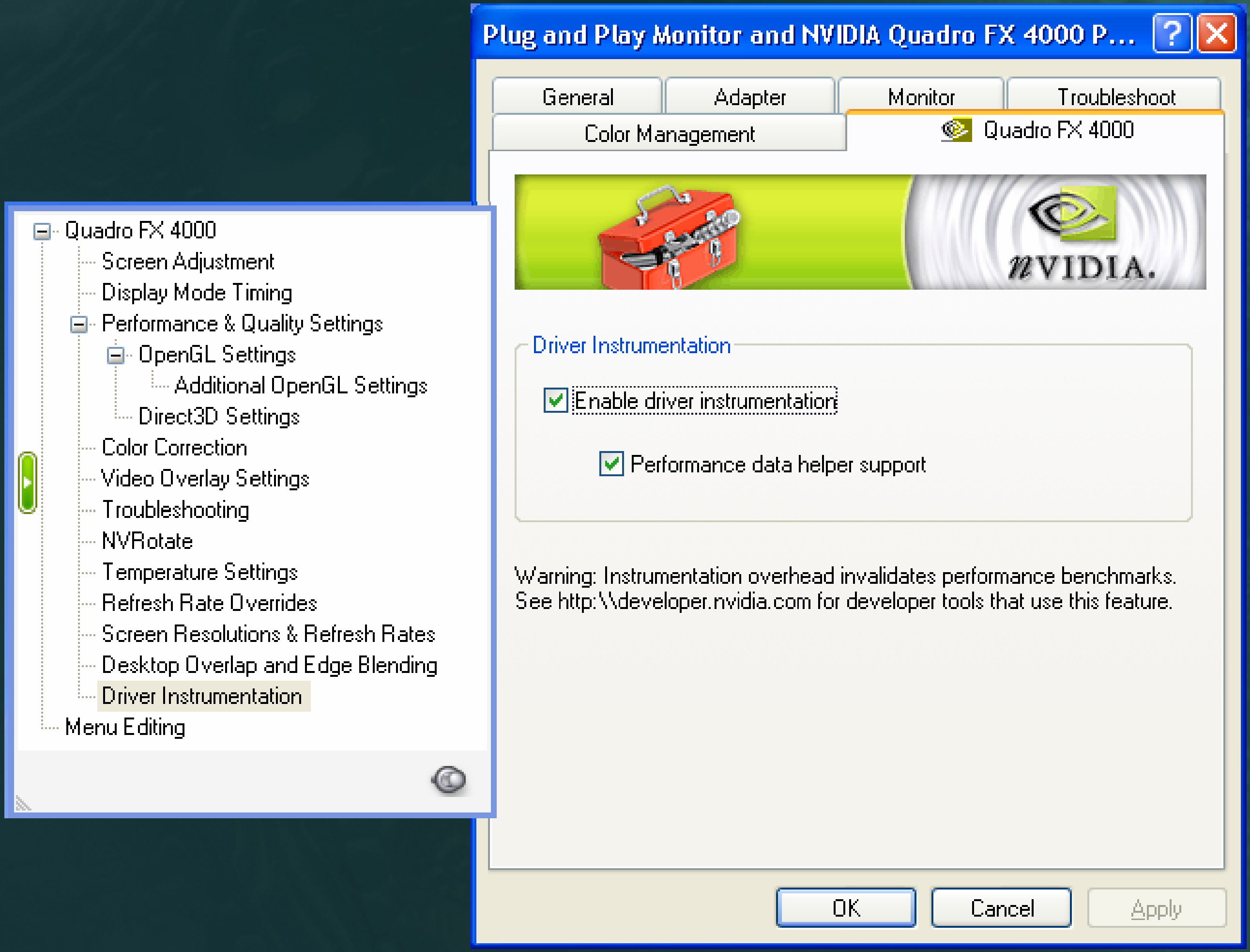
Plug-in for PIX for Windows

NVPerfHUD 3.0

Instrumented Driver

- Special Instrumented Driver
  - Built with regular drivers
  - Includes NVPMAPI.DLL
- Exposes Driver and HW Performance Counters
- Compatible with Windows WMI and PDH
- New Driver Instrumentation tab in NVIDIA Display Control Panel

Instrumented Driver



Driver Instrumentation

☑ Enable driver instrumentation

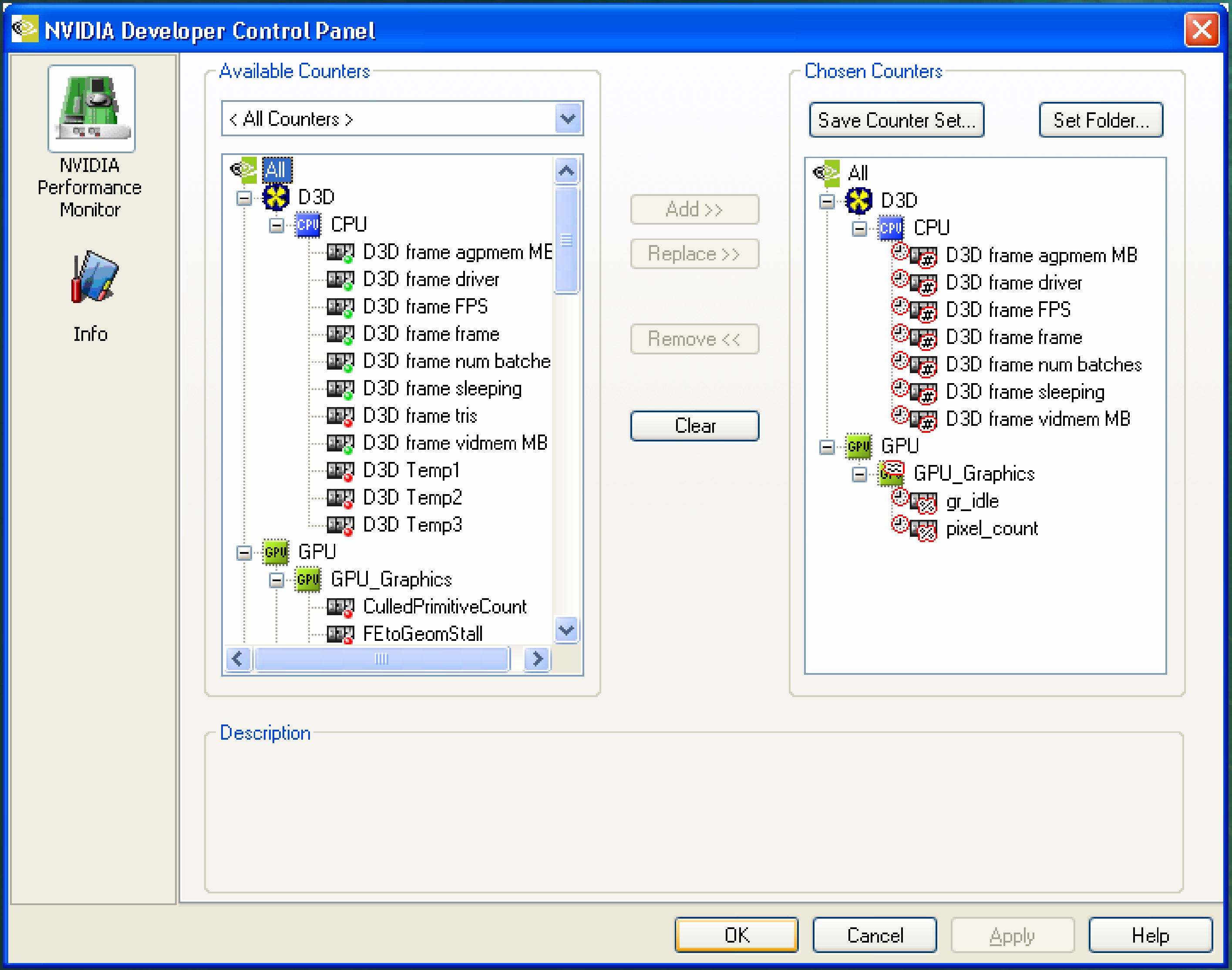
☑ Performance data helper support

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

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



Available Counters:

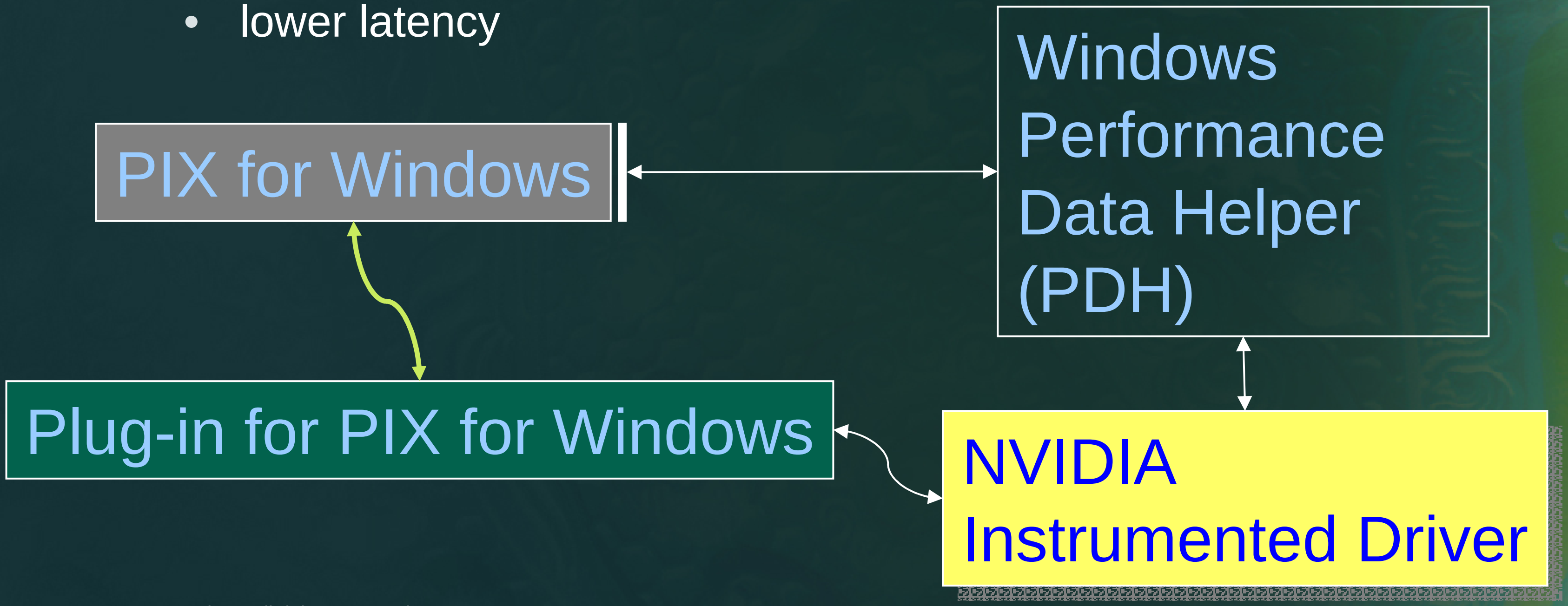
- All
  - D3D
    - CPU
      - D3D frame agpmem MB
      - D3D frame driver
      - D3D frame FPS
      - D3D frame frame
      - D3D frame num batche
      - D3D frame sleeping
      - D3D frame tris
      - D3D frame vidmem MB
      - D3D Temp1
      - D3D Temp2
      - D3D Temp3
  - GPU
    - GPU_Graphics
      - CulledPrimitiveCount
      - FEtoGeomStall

Chosen Counters:

- All
  - D3D
    - CPU
      - D3D frame agpmem MB
      - D3D frame driver
      - D3D frame FPS
      - D3D frame frame
      - D3D frame num batches
      - D3D frame sleeping
      - D3D frame vidmem MB
  - GPU
    - GPU_Graphics
      - gr_idle
      - pixel_count

PIX for Windows Plug-in

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



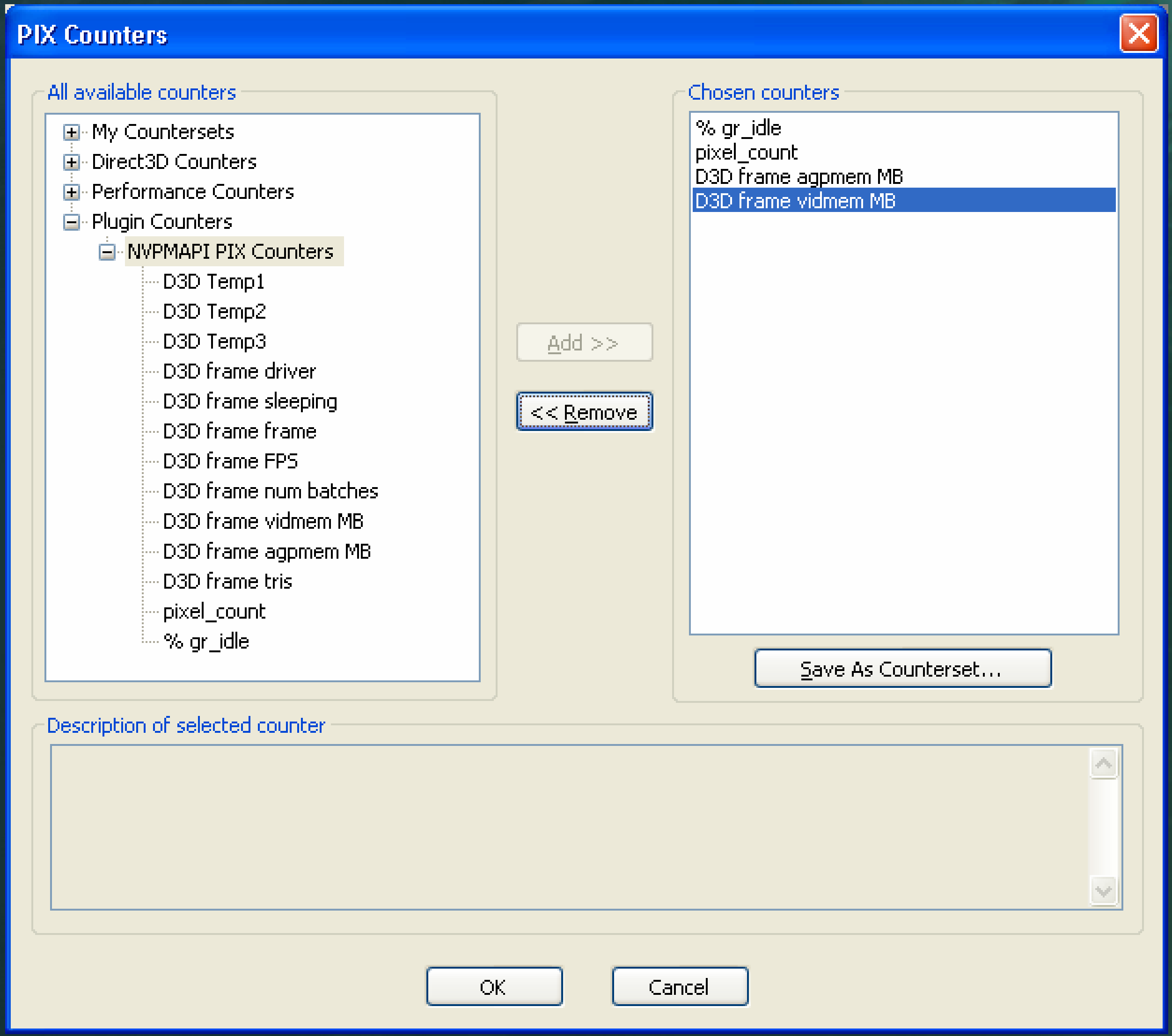
PIX for Windows

Windows Performance Data Helper (PDH)

Plug-in for PIX for Windows

NVIDIA Instrumented Driver

Plug-in for PIX: Demo



PIX Counters

All available counters

- My Countersets
- Direct3D Counters
- Performance Counters
- Plugin Counters
  - NVPMAPI PIX Counters
    - D3D Temp1
    - D3D Temp2
    - D3D Temp3
    - D3D frame driver
    - D3D frame sleeping
    - D3D frame frame
    - D3D frame FPS
    - D3D frame num batches
    - D3D frame vidmem MB
    - D3D frame agpmem MB
    - D3D frame tris
    - pixel_count
    - % gr_idle

Chosen counters

- % gr_idle
- pixel_count
- D3D frame agpmem MB
- D3D frame vidmem MB

Description of selected counter

NVPerfKit Code Samples

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

Conclusion

- Comprehensive Suite of Performance Tools
  - performance information at all levels
    - Direct3D API
    - Direct3D Driver
    - Hardware
- Provided in a variety of venues
    - Microsoft WMI/PDH
    - Microsoft PIX for Windows
    - User application
    - NVPerfHUD

Questions?

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