Performance Tools

Raul Aguaviva, Sim Dietrich, and Sébastien Dominé
Agenda

- NVPerfHUD 3.0
  - The tool
  - In real life
- NVPerfKIT 1.0
- NVShaderPerf 71.84
- FX Composer 1.6
- Conclusion
- Q&A
NVPerfHUD 3.0

FPS: 15.1, TRIs/Frame: 1746450, Time: 37.0 secs

NVPerfHUD version: 3.0, 208.1527
NVidia driver version: 6.14.10.7550
App name: C:\Program Files\Futuremark\3DMark05\3DMark05.exe

- Handshake with application OK.
- WARNING: Forcing NON PURE device in order to enable the shader visualization features.
- Direct3D RETAIL runtime detected.
- Connection with driver OK!

Number of D3 calls: 394
Debug Console
Advanced - Index Unit

Index/ Vertex Buffer

** Index Buffer Description **
- Format: D3DFMT_INDEX16
- Pool: D3DPool_MANAGED
- Usage: D3DUSAGE_WRITEONLY
- Length: 840 bytes

** VE Declaration **
- Total vertex size: 72
- 0: POSITION FLOATS3 DEFAULT
- 0: NORMAL FLOATS3 DEFAULT
- 0: COLOR RGB COLOR DEFAULT
- 0: TEXCOORD FLOATS2 DEFAULT
- 0: TEXCOORD FLOATS2 DEFAULT
- 0: TEXCOORD FLOATS3 DEFAULT

** IP Info **
- DrawIndexedPrimitive:
  - Type: D3DPT_TRIANGLELIST
  - BaseVertexIndex: 0
  - MinVertexIndex: 0
  - NumVertices: 763
  - StartIndex: 0
  - NumCount: 1484
  - HResult: 0x00000000
- Msg: OK
- Desc: The function completed successfully

DP Pointer 176 / 247

PS - Performance  P6 - Debug Console  P7 - Frame Analysis

GDC 2005
Advanced – Vertex Shader

```cpp
// v0 = (0, 0, 0, 0);
// v1 = (0, 0, 0, 0);
// v2 = (0, 0, 0, 0);
// v3 = (0, 0, 0, 0);
// v4 = (0, 0, 0, 0);

vertex Shader Constants:

float4 constant0 = float4(0.5, 0.5, 0.5, 0.5);
float4 constant1 = float4(0.6, 0.7, 0.8, 0.9);
float4 constant2 = float4(0.3, 0.4, 0.5, 0.6);
float4 constant3 = float4(0.7, 0.8, 0.9, 1.0);
float4 constant4 = float4(0.2, 0.3, 0.4, 0.5);
float4 constant5 = float4(0.1, 0.2, 0.3, 0.4);
float4 constant6 = float4(0.4, 0.5, 0.6, 0.7);
float4 constant7 = float4(0.7, 0.8, 0.9, 1.0);
float4 constant8 = float4(0.5, 0.6, 0.7, 0.8);
float4 constant9 = float4(0.6, 0.7, 0.8, 0.9);
float4 constant10 = float4(0.7, 0.8, 0.9, 1.0);
float4 constant11 = float4(0.8, 0.9, 1.0, 1.0);

DP Pointer 161 / 247
```
Advanced – Pixel Shader

DeferredShading

Pixel Shader

Registers:
- p0
- p1
- s0
- s1
- d0

Textures:
- Sampler 0
  - D0
  - D1

Pixel Shader Constants:
- Floating Point Constants:
  - C0: 0.000000
  - C1: 0.000000
- Integer Constants:
- Boolean Constants:

Step Back
Step Forward
Stop

Delta Pointer 164 / 247

GDC 2005
Advanced – Raster Operation
NVPerfHUD 3.0 – Freezing the game

- Only possible if game uses time-based animation

- Stop the clock
  - Intercept: QueryPerformanceCounter(...) timeGetTime(...)
  - No RDTSC

- Pos += V * DeltaTime

- When DeltaTime is 0
  - Don’t divide by DeltaTime
  - Don’t skip Presents()
Coming Soon!

- NVPerfHUD 4.0
  - Pipeline utilization graph
  - Automated Bottleneck Identification
NVPerfHUD Schedule

**NVPerfHUD 3.0**
- Registered developer Beta 2 : now
- Registered developer Beta 3 : 3/16/2005
- Final Release : 3/23/2005

**NVPerfHUD 4.0**
- Registered developer Beta: 5/1/2005
- Final Release : 5/15/2005
NVPerfHUD in real life...

FPS: 33.6, TRIS/Frame: 152997, Time: 29.2 sec
Press F1 for help

NVPerfHUD version: 3.0.215.1953
NVIDIA driver version: 6.14.10.7505
App name: C:\dev\Next\Next.exe
- Handshake with application OK
- WARNING: Forcing NON PURE device in order to enable the shader visualization features.
- DirectX *RETAIL* runtime detected.
- Connection with driver OK!

- Number of DP calls: 441

- Ms per frame - Driver time - CPU waits for GPU - GPU idle

- Press hotkey to activate NVPerfHUD -
NVPerfKIT

Complete Performance Instrumentation Solution
- Instrumented Driver
- NVIDIA Developer Control Panel
- Plug-in for Microsoft PIX for Windows
- Support for PDH (Performance Data Helper)
- Code samples for OpenGL and Direct3D
- Secure mechanism for authorizing applications to be instrumented
NVPerfKIT

- Instrumented Driver
  - Exposes GPU and Driver Performance Counters
  - Supports OpenGL and Direct3D
  - Supports SLI Counters
  - Requires GeForce FX or 6 Series
NVPerfKIT Performance Data Helper
NVPerfKIT

Schedule
- Registered Developer - Early Beta Access : 3/16/2005
- Registered Developer - Beta : 3/23/2005
- Registered Developer - Release : 4/7/2005
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);
    // 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.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, World);
    // compute the eye vector (going from shaded point to eye) in cube space
    float4 eyeVector = worldPos - ViewIT[3]; // view inv. transpose contains eye position in world space
    OUT.TexCoord1.w = eyeVector.x;
    OUT.TexCoord2.w = eyeVector.y;
    OUT.TexCoord3.w = eyeVector.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 from the cube map
    // we multiply by 2 only to increase brightness
    float3 eyevec = float3(IN.TexCoord1.w, IN.TexCoord2.w, IN.TexCoord3.w);
    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);
}
NVShaderPerf – Coming up...

- Vertex throughput
- GLSL vertex program
- Multiple driver versions
- What else do you need?
FX Composer 1.6 – 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)
Questions?

- NVPerfHUD@nvidia.com
- FXComposer@nvidia.com
- NVshaderPerf@nvidia.com
- NVPerfKIT@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.
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