GPU Performance Tools

Sébastien Dominé
Manager of Developer Technology Tools
NVIDIA Corporation
Agenda

• Tools of Today
  – NVPerfHUD 2.0
  – NVShaderPerf
  – FX Composer 1.1

• Tools of Tomorrow
  – Instrumented Driver
  – FX Composer 1.5

• Conclusion and Q&A
The Tools of Today
NVPerfHUD 2.0

- Overlay graph that displays stats from:
  - Direct3D9 API interception layer
  - Direct3D Driver
- Able to bypass and inject some API calls to assist with performance analysis

Image courtesy of FutureMark Corp.
What’s new in 2.0?

3D Application → DirectX Runtime → DirectX Driver → NVPerfHUD 1.0 → HW

3D Application → DirectX Runtime → DirectX Driver → NVPerfHUD 2.0 → stats → HW
• DrawPrimitives/DrawIndexPrimitives

Histogram

# of DrawPrimitives

# of triangles
Texture Stage States

Image courtesy of FutureMark Corp.
Pixel Shaders 1.x

Image courtesy of FutureMark Corp.
2x2 Texture replacement

Image courtesy of FutureMark Corp.
Null DrawPrimitive mode

Image courtesy of FutureMark Corp.
NVPerfHUD - Overhead

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

Demo running FutureMark’s 3DMark2003
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);
    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]);
    xzy, TangentToObjSpace[0]);
    xzy, TangentToObjSpace[1]);
    xzy, TangentToObjSpace[1]);
    xzy, TangentToObjSpace[2]);
    xzy, TangentToObjSpace[2]);
    worldPos = mul(IN.Position, World);
    // compute the eye vector (going from shaded point to eye) in cube space
    eyeVector = worldPos - ViewIT[3]; // view inv. transpose contains eye
    OUT.TexCoord1.w = eyeVector.x;
    OUT.TexCoord2.w = eyeVector.y;
    OUT.TexCoord3.w = eyeVector.z;
    return OUT;
}

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));
    // then use the transformed normal and eye vector to compute a reflection vector
    // used to fetch the cube map
    float3 lookup = reflect(eyevec, worldNorm);
    return texCUBE(EnvironmentMap, lookup);
}

Inputs:
• HLSL
• !!FP1.0
• !!ARBfp1.0
• PS1.x
• PS2.x

GPU Arch:
• NV3X
• ...and more
NVShaderPerf

Direct3D Application

DirectX Runtime

DirectX Driver

Unified Compiler

HW

HLSL

Direct3D shader op-codes

API agnostic shader op-codes

HW Binary
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]);
    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 in last row.
    OUT.TexCoord1.w = eyeVector.x;
    OUT.TexCoord2.w = eyeVector.y;
    OUT.TexCoord3.w = eyeVector.z;
    return OUT;
}

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 brightness)
    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);
    float4 lookup = reflect(eyevec, worldNorm);
    return lookup;
}
FX Composer 1.1

- IDE for HLSL authoring, debugging and optimization
- Pixel Shader scheduling
- Direct3D9 VS/PS op-code disassembly
- Advanced texture generation for baking Look Up Tables
FX Composer – Shader Perf

• Target GPU
• Driver version match
• Number of Cycles
• Number of Registers
• GPU Utilization

![Shader Perf](image)
• Vertex Shader
• Pixel Shader 1.x, 2.x
FX Composer – LUT Optimization

• Bake your own texture for function look up:
  – Normalization cubemaps
  – Lighting computation
  – Expensive math
  – Functions that can be artist controlled...
FX Composer - Demo

Overview of a performance tutorial
The Tools of Tomorrow
Instrumented Driver

• Perfect companion for Intel VTune, MSFT PIX for Windows, Perfmon, etc...
• Allows 3D applications to monitor:
  – Resource available (AGP, etc...)
  – Driver counters (Spins, etc...)
  – Hardware counters (bottlenecks, etc...)
Driver Instrumentation Architecture

VTune
PIX for Windows
Game Engine

Windows Performance Data Helper (PDH)

NVIDIA Developer Control Panel

OpenGL Driver
Direct3D Driver

NVPMAPI.DLL
NV Dev Control Panel
Instrumented Driver - Demo

- Direct3D9
- OpenGL
- HW Counters
FX Composer 1.5

• Vertex Shader Scheduling
• Texture Anisotropic Wizardry
• Support for next generation GPU
• …and much more to come!
Conclusion

• API Performance
• Unified Compiler Performance
• Driver Performance
• HW Performance
• What else do you need?
Other talks of interest...

- Wed: Practical Performance Analysis and Tuning [4:00pm – 5:00pm] by Ashu Rege and Clint Brewer
Credits

• FutureMark for letting us use 3DMark2003
• Special thanks to Raul Aguaviva, Jeffrey Kiel and Christopher Maughan for making these tools!
Q&A

• Questions?
  Sébastien Dominé - sdomine@nvidia.com
Graphics Pipeline

Vertices → Geometry Storage → Geometry Processor → Rasterizer → Fragment Processor → Frame buffer

CPU → CPU/Bus → GPU

Texture Storage + Filtering

CUDA
PC Driver Model

3D Application

DirectX Runtime

DirectX Driver

Unified Compiler

OpenGL Runtime

OpenGL Driver

HW