Performance Tools

Jeff Kiel, NVIDIA Corporation
Performance Tools Agenda

- Problem statement
- Overview of GPU pipelined architecture
- NVPerfKit 2.0: Driver and GPU Performance Data
  - NVPerfHUD: The GPU Performance Accelerator
  - NVPerfSDK: Performance data integrated into your application
- NVPerfHUD ES Sneak Preview
- gDEBugger: OpenGL performance analysis and debugging
- NVShaderPerf: Shader program performance
The Problem?

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

The Solution? NVPerfKit!

35% FPS improvement!*
11 Rendering bugs fixed!*

*Average of 35% FPS improvement and 11 bugs fixed reported by over 100 users of NVPerfHUD in recent Developer Survey
GPU pipelined architecture

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.

If performance doesn’t change significantly, this stage is the bottleneck.
Careful not to change the workload of target stage!
Sample performance data in each subunit of the GPU pipeline while rendering
- Compare amount of work done to maximum work possible
- Query the subunit for unit bottleneck information

**NVPerfKit: The Ideal GPU Performance Tool!**
- **NVPerfHUD: The GPU Performance Accelerator**
- **NVPerfAPI: Integrated in your application**

Analyze your application like an NVIDIA Engineer!
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
  - gDEBugger 2.4
  - NVDevCPL
NVPerfKit Instrumented Driver

- GLExpert functionality
- 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 to stdout or debugger
  - Different groups/levels of information detail
  - Controlled using environment variables in Linux, tab in NVDevCPL on Windows

Information provided:
- GL Errors: print when 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 for unsupported configuration
GLExpert: NVDevCPL tab

GLExpert Settings

- Report Errors
- Report Software Fallback Messages
- Report Vertex and Fragment Program Messages
- Report VBO Messages
- Report FBO Messages

Detail Level

- Off
- Max

All errors with warnings on usage which may cause suboptimal runtime performance

Send Output To: Debugger (OutputDebugString)
NVPerfKit: Counter Types

- **SW/Driver Counters (Instrumented Driver)**
  - Insight into OpenGL and Direct3D driver performance
  - Exposed via NVPerfAPI, PIX, and PDH

- **Raw GPU Counters (Instrumented GPU)**
  - Real time performance monitoring
  - Exposed via NVPerfAPI, PIX, and PDH

- **Simplified Experiments (Instrumented GPU)**
  - In depth performance analysis and bottleneck determination
  - Exposed via NVPerfAPI

- **Instrumented GPUs**
  - Quadro FX 5500 & 4500
  - GeForce 7900 GTX & GT
  - GeForce 7800 GTX
  - GeForce 6800 Ultra & GT
  - GeForce 6600
OpenGL/Direct3D 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
- (Direct3D) Triangles and instanced triangles
- (Direct3D) Locked render targets

Memory
- AGP memory used
- Video memory used and total
What is NVPerfHUD?

Direct3D Performance and Debugging Tool
- Overlay graphs and debugging tools on top of your application
- Interactive tools for debugging and performance tuning

4 different screens
- 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

3DMark06 used with permission from Futuremark corporation
Demo: Performance Dashboard

3DMark06 used with permission from Futuremark corporation
Demo: Performance Dashboard

3DMark06 used with permission from Futuremark corporation
Demo: Performance Dashboard

3DMark06 used with permission from Futuremark corporation
Demo: Performance Dashboard

3DMark06 used with permission from Futuremark corporation
Demo: Performance Dashboard

3DMark06 used with permission from Futuremark corporation
Demo: Performance Dashboard

Resources monitored

- Textures
- Volume Textures
- Cube textures
- Vertex Buffers
- Index buffers
- Stencil and depth surfaces
Demo: Performance Dashboard
Demo: Performance Dashboard

Speed control

FPS: 52.3  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

- Vertex Assembly
- Vertex Shader
- Pixel Shader
- Raster OPerations
Demo: Debug Console
Demo: Advanced Frame Debug

3DMark06 used with permission from Futuremark corporation
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…
Demo: Frame Profiler

3DMark06 used with permission from Futuremark corporation
Demo: Advanced Frame Profiler

3DMark06 used with permission from Futuremark corporation
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
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

```c
NVPMGetCounterValueByName("IDX BNeck", 0, &nIDXBeckEvents, 
&nIDXBeckCycles);
NVPMGetCounterValueByName("IDX SOL", 0, &nIDXSOLEvents, 
&nIDXSOLCycles);
```
Associated Tools: NVIDIA Plug-In for Microsoft PIX for Windows
Associated Tools: NVIDIA Plug-In for Microsoft PIX for Windows
Project Status

- NVPerfKit 2.0 for Windows 32bit available now at developer.nvidia.com
- NVPerfKit 2.1 (August 2006)
  - NVPerfHUD 4.1
  - ForceWare Release 90 Driver
  - Windows 64 bit
  - Linux 32 bit and 64 bit
- Instrumented GPUs
  - Quadro FX 5500 & 4500
  - GeForce 7900 GTX & GT
  - GeForce 7800 GTX
  - GeForce 6800 Ultra & GT
  - GeForce 6600
  - GeForce 6600

Feedback and Support: NVPerfKit@nvidia.com
NVPerfHUD ES

- Developing similar performance tools for handheld developers using OpenGL ES
- Application runs on real hardware with Instrumented Driver
- IDE runs on host computer (Linux or PC)
- Same debugging and performance analysis tools that are available on NVPerfHUD 4.0!
Debugging Features
Graphic Remedy’s gDEBugger
Free gDEBugger Licenses!

OpenGL ARB and Graphic Remedy
Academic License Program

- Annual program for academic OpenGL developers
- One year license for full featured version, including all software updates
- Limited number of free licenses available for non-commercial, non-academic developers

Details: [http://academic.gremedy.com](http://academic.gremedy.com)
What is NVShaderPerf?

What’s coming with version 2.0?
v2f BumpReflectVS(a2v IN,
uniform float4x4 WorldViewProj,
uniform float4x4 World,
uniform float4x4 ViewIT)
{
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]);
OUT.TexCoord1.w = eyeVector.x;OUT.TexCoord2.w = eyeVector.y;OUT.TexCoord3.w = eyeVector.z;
return OUT;
}

///////////////// pixel shader //////////////////

float4 BumpReflectPS(v2f IN,
sampler2D NormalMap,
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;
// 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);
float3 lookup = reflect(eyevec, worldNorm);
return texCUBE(EnvironmentMap, lookup);
}
NVShaderPerf: In your pipeline

- Test current performance
- Compare with 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)
NVShaderPerf 1.8

- Support for GeForce 7800 GTX and Quadro FX 4500
- Unified Compiler from ForceWare Release 80 driver
- Better support for branching performance
  - Default computes maximum path through shader
  - Use –minbranch to compute minimum path
// 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;

float3 centerToSurfaceVec = objFlatNormal; // = normalize( v2f.objCoord )
float firstDot = centerToSurfaceVec.y; // = dot( centerToSurfaceVec, float3( 0, 1, 0 ) )
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 - objEyePos );
    float3 ray_origin = v2f.objCoord;
    // refract the ray before intersecting with the iris
    ray_dir = refract( ray_dir, objFlatNormal, g_refraction_u );
    // first, see if the refracted ray would leave the eye before hitting the Iris.
    float t_eyeballSurface = SphereIntersect( 16.0, ray_origin, ray_dir );
    float3 objPosOnEyeBall = ray_origin + t_eyeballSurface * ray_dir;
    float3 centerToSurface2 = normalize( objPosOnEyeBall );
    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.
NVShaderPerf – Version 2.0

- Improved vertex performance simulation and calculation of vertex throughput
- GLSL vertex program
- Multiple driver versions from one NVShaderPerf
- Much smaller footprint
- New programmatic interface
- Integration into FX Composer 2.0

Support and Feedback: 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:

Feedback and Support:
- NVPerfKit@nvidia.com
- NVPerfHUD@nvidia.com
- NVShaderPerf@nvidia.com
- FXComposer@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.

developer.nvidia.com