NVIDIA OpenGL Update

Simon Green
Overview

- SLI
  - How it works
  - OpenGL Programming Tips
  - SLI Futures
- New extensions
  - NVX_instanced_arrays – OpenGL instancing!
  - EXT_timer_query
What is SLI?

- Allows scaling graphics performance by combining multiple GPUs in a single system
- Works best with NVIDIA nForce motherboards
- Improves rendering performance up to 2x with two GPUs
SLI-Ready PCs affordable for Everyone

nForce4 SLI Motherboard +
Dual GeForce 6800 GTs

$2200

nForce4 SLI Motherboard +
GeForce 6600 LE

$799

Copyright © NVIDIA Corporation 2004
Quad SLI

4 GPUs is better than 2!
2 cards, each with 2 GPUs
SLI Notebooks
SLI Game Performance Scaling

GeForce 7800 GTX  |  GeForce 7800 GTX SLI

- Project Snowblind
- Brothers in Arms
- Thief: Deadly Shadows
- Code Creatures
- Chronicles of Riddick
- Splinter Cell Chaos Theory
- Ground Control 2
- X2
- Doom 3
- 3DMark05
- 3DMark03
- Painkiller: Battle Out of Hell
- Colin McRea 5

Tests run on AMD FX 55 with 1GB memory 1600x1200, 4X/8X Power

Copyright © NVIDIA Corporation 2004
How SLI Works

- Plug multiple GPUs into motherboard
  - Have to be same model currently
- NVIDIA driver reports as one logical device
  - Video memory does NOT double
- Video scan out happens from one board
  - Bridge connector transmits digital video between boards
SLI and Game Development

- Developing a game now takes 2 years or more
  - CPU performance doubles (or less)
  - GPU performance quadruples

- CPU / GPU balance shifts
  - Worse: CPU-hungry modules are developed later
  - AI, physics, full game play

- SLI allows you to preview future GPU performance now
SLI Rendering Modes

- Compatibility mode
  - Only uses one GPU
  - No SLI benefits
- Alternate frame rendering (AFR)
- Split frame rendering (SFR)
- SLI AA
- SLI Stereo?
GPUs work on alternate frames in parallel

Scan-out toggles which framebuffer to read from
AFR Advantages

Advantages

- All work is parallelized
  - Scales geometry and pixel fill performance
- Preferred SLI mode

Disadvantages

- Requires pushing data to other GPU if frame is not self-contained
  - For example, if application updates a render-to-texture target only every other frame
GPUs work on the same frame

For two GPUs
- GPU 0 renders top region
- GPU 1 renders bottom region

Scan-out combines framebuffer data
SFR Advantages

- Driver load-balances by changing region size
  - Based on time each GPU took to render

- Driver clips geometry to regions
  - Avoids both GPUs processing all vertices
  - But not perfect

- Still requires sharing data between GPUs
  - E.g., render to texture
SFR Compared to AFR

- SFR works even when few frames are buffered
- Or when AFR otherwise fails

- In general, SFR has more communications overhead

- Applications with heavy vertex load benefit less from SFR
Overview: Things Interfering with SLI

- CPU-bound applications
  - Or vertical-sync enabled
- Applications that limit the number of frames buffered
- Communications overhead
CPU-Bound Applications

- SLI cannot help
- Reduce CPU work
- Move CPU work onto the GPU
  - See [http://www.gpgpu.org](http://www.gpgpu.org)
- Don’t deliberately throttle frame-rate
V-Sync

Enabling vertical-sync limits frame rate to multiples of the monitor refresh rate.
Limiting Number of Frames Buffered

Some apps allow at most one frame buffered
  To reduce lag
  Via occlusion queries
  Don’t read back-buffer - this causes CPU stall

Breaks AFR SLI

SLI is faster anyway
  e.g. 2 GPU SLI systems ~2.0x less lag
Why Reading the Back Buffer Is Bad

Back buffer read:
wait for GPU to finish rendering

Frame n  Frame n+1  ...

CPU GPU CPU GPU

Copyright © NVIDIA Corporation 2004
OpenGL SLI Tips

- Limit OpenGL rendering to a single window
  - child windows shouldn’t have OpenGL contexts
- Request pixel format with PDF_SWAP_EXCHANGE
  - tells driver that app doesn’t need the back buffer contents after SwapBuffers()
- Avoid rendering to FRONT buffer
  - use overlays instead on Quadro GPUs
Offscreen Rendering and Textures

- Limit P-buffer usage
  - Often requires broadcasting rendering to both GPUs
- Use render-to-texture rather than `glCopyTexImage2D`
  - `glCopyTexImage2D` requires texture to be copied to both GPUs
  - Use FBO or P-buffers instead
- Limit texture working set
  - Textures have to be stored on both GPUs
  - Don’t download new textures unnecessarily
Geometry

- Use Vertex Buffer Objects or display lists to render geometry
  - Don’t use immediate mode
  - Reduces CPU overhead
- Render the entire frame
  - Don’t use use glViewport or glScissor
  - Disables load balancing in SFR mode, and hurts performance in AFR mode
More OpenGL SLI Tips

- **Limit read-backs**
  - e.g. `glReadPixel`, `glCopyPixels`
  - causes pipeline to stall

- **Never call `glFinish()`**
  - doesn’t return until all rendering is finished
  - prevents parallelism

- **Avoid `glGetError()` in release code**
  - Causes sync point
How Do I Detect SLI Systems?

**NVCpl API:**
- NVIDIA-specific API supported by all NV drivers

**Function support for:**
- Detecting that NVCpl API is available
- Bus mode (PCI/AGP/PCI-E) and rate (1x-8x)
- Video RAM size
- SLI
NVCpl API SLI Detection

SDK sample and full documentation available

```c
HINSTANCE hLib = ::LoadLibrary("NVCPL.dll");

NvCplGetDataIntType NvCplGetDataInt;
NvCplGetDataInt =
    (NvCplGetDataIntType))::GetProcAddress(hLib,
    "NvCplGetDataInt");

long    numSLIGPUs = 0L;
NvCplGetDataInt(NVCPL_API_NUMBER_OF_SLI_GPUS,
    &numSLIGPUs);
```
Forcing SLI Support In Your Game

Use NVCpl
- NvCplSetDataInt() sets AFR, SFR, Compatibility mode
- See SDK sample

Modify or create a profile:
- [http://nzone.com/object/nzone_sli_appprofile.html](http://nzone.com/object/nzone_sli_appprofile.html)
- End-users can create profile as well
SLI Performance Tools

- NVPerfKit has support for SLI
- Provides performance counters for
  - Total SLI peer-to-peer bytes
  - Total SLI peer-to-peer transactions
- Above originating from
  - Vertex/index buffers: bytes and transactions
  - Textures: bytes and transactions
  - Render targets: bytes and transactions
What is Instancing?

- Rendering multiple instances of a given geometry
- Some attributes can vary across instances
  - Transformation matrix
  - Color
- Examples
  - Trees in a forest
  - Characters in a crowd
  - Boulders in a avalanche
  - Screws in an assembly
Instancing Methods in OpenGL

- Send transform as vertex program constants
  - Relatively slow
  - Can also pack several transforms into constant memory and index in vertex program

- Send transform using immediate mode texture coordinates ("pseudo instancing")
  - Usually much faster (glTexCoord calls are inlined)
  - Requires custom vertex program
  - Can use glArrayElement to set current texture coordinates from a vertex array (not efficient on NV hardware)

- NVX_instance_arrays
  - Single draw call
  - Fastest
NVX\_instanced\_arrays

- Allows rendering multiple instances of an object with a single draw call
- Similar to Direct3D instancing functionality
- OpenGL draw call cost is lower than Direct3D, but still gives a significant performance benefit
- Combined with render-to-vertex array, can be used for controlling object transformations on the GPU
- Performance is dependent on CPU speed, GPU speed, number of objects and number of vertices per object
- Will improve on next generation GPU hardware
## OpenGL Instancing Performance

<table>
<thead>
<tr>
<th>Verts/object</th>
<th>Constants (fps)</th>
<th>Texcoords (fps)</th>
<th>Instancing (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>205</td>
<td>323</td>
<td>560</td>
</tr>
<tr>
<td>24</td>
<td>200</td>
<td>266</td>
<td>440</td>
</tr>
<tr>
<td>60</td>
<td>183</td>
<td>190</td>
<td>246</td>
</tr>
<tr>
<td>120</td>
<td>138</td>
<td>135</td>
<td>155</td>
</tr>
<tr>
<td>220</td>
<td>72</td>
<td>77</td>
<td>77</td>
</tr>
</tbody>
</table>

8192 objects, Quadro FX 4500, P4 3.4 GHz
NVX\_instanced\_arrays

- Allows rendering an array of primitives multiple times, while stepping specified vertex attribute arrays only once per N objects.
- Only supports generic attribute arrays.
- No immediate mode.
- Warning – experimental extension -API may change!
- Typically 3 attribute arrays are used to store a 3x4 transformation matrix.
  - Attribute divisor is set to 1 for these arrays.
  - Custom vertex program transforms geometry from object to world space based on input attributes.
void VertexAttribDivisorNVX(uint attrib, uint divisor);

- Specifies rate at which to advance attribute per object
- 0 = disabled
- Attribute 0 (position) cannot be changed
- Future – fractional divisor to allow geometry amplification?

void DrawArraysInstancedNVX(enum mode, int start, sizei count, sizei primCount);

void DrawElementsInstancedNVX(enum mode, sizei count, enum type, const void *indices, sizei primCount);

- Renders primCount instances of specified geometric primitives, using attribute divisors
for (instance = 0; instance < primCount; instance++) {
    Begin(mode);
    for (vertex = 0; vertex < count; vertex++) {
        for (attrib = 1; attrib < MAX_ATTRIB; attrib++) {
            if (ArrayAttribEnabled[attrib]) {
                if (InstanceDivisors[attrib] > 0) {
                    offset = instance / InstanceDivisors[attrib];
                } else {
                    offset = start + vertex;
                }
                offset *= CookedAttribStride[attrib];
                VertexAttribvFunc[attrib](
                    VertexAttribPointers[attrib] + offset);
            }
        }
        if (ArrayAttribEnabled[0]) {
            offset = start + vertex;
            offset *= CookedAttribStride[0];
            VertexAttribvFunc[0](
                VertexAttribPointers[0] + offset);
        }
    }
    End();
}
Standard Rendering Loop

// load vertex arrays and transform data
...

for(int i=0; i<nobjects; i++) {
    // send transformation as texture coordinates
    glMultiTexCoord4fv(GL_TEXTURE0, &transform_data[0][i*4]);
    glMultiTexCoord4fv(GL_TEXTURE1, &transform_data[1][i*4]);
    glMultiTexCoord4fv(GL_TEXTURE2, &transform_data[2][i*4]);

    // draw instance
    glDrawElements(GL_TRIANGLES, nindices, GL_UNSIGNED_SHORT, indices);
}

Using Instancing

// set vertex array pointers
...

// enable transform attribute arrays and set divisors
glEnableVertexAttribArrayARB(8);  // texcoord0
glVertexAttribDivisorNVX(8, 1);
glEnableVertexAttribArrayARB(9);  // texcoord1
glVertexAttribDivisorNVX(9, 1);
glEnableVertexAttribArrayARB(10);  // texcoord2
glVertexAttribDivisorNVX(10, 1);

// draw all instances at once
glDrawElementsInstancedNVX(GL_TRIANGLES, nindices,
    GL_UNSIGNED_SHORT, indices, nobjects);

glDisableVertexAttribArrayARB(8);
glVertexAttribDivisorNVX(8, 0);
glDisableVertexAttribArrayARB(9);
glVertexAttribDivisorNVX(9, 0);
glDisableVertexAttribArrayARB(10);
glVertexAttribDivisorNVX(10, 0);
## HavokFX Instancing Results

<table>
<thead>
<tr>
<th></th>
<th>Readback (fps)</th>
<th>Instancing (fps)</th>
<th>Instancing / Readback</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096 bricks</td>
<td>240</td>
<td>280</td>
<td>1.17</td>
</tr>
<tr>
<td>8000 bricks</td>
<td>130</td>
<td>150</td>
<td>1.15</td>
</tr>
<tr>
<td>27000 bricks</td>
<td>40</td>
<td>46</td>
<td>1.15</td>
</tr>
<tr>
<td>5000 boulders</td>
<td>173</td>
<td>223</td>
<td>1.29</td>
</tr>
<tr>
<td>10000 boulders</td>
<td>90</td>
<td>114</td>
<td>1.27</td>
</tr>
<tr>
<td>30000 boulders</td>
<td>31</td>
<td>41</td>
<td>1.32</td>
</tr>
</tbody>
</table>
Timing is important for performance tuning

How can you improve something if you can’t measure it accurately?

Problem with timing the GPU is that it is asynchronous and has a deep pipeline

There’s no way to know if a particular command has completed before reading the timer

Usual solution is to insert glFinish() commands

Guarantees that all rendering commands have completed, but stalls pipeline and changes performance!
Provides a method for timing a sequence of OpenGL commands, without stalling the pipeline
Based on the query object mechanism introduced by the occlusion query extension

- `glBeginQuery()`
  - Timer starts when all prior commands have completed
- `glEndQuery()`
  - Timer stops when all prior commands have completed

Measures total time elapsed (driver + hardware)
- Measured in nanoseconds (10^{-9} seconds)
- 32 bit counter can represent about 4 seconds maximum

Introduces `GLuint64` type to allow 64 bit counters
Code Example

GLint queries[N];
glGenQueries(N, queries); // generate query objects

for(int i=0; i<N; i++) {
    glBeginQuery(GL_TIME_ELAPSED_EXT, queries[i]); // Start query
    // Draw object i
    glEndQuery(GL_TIME_ELAPSED_EXT); // End query
}

// Wait for all results to become available
// (should really only wait for previous frame’s results)
int available = 0;
while (!available) {
    glGetQueryObjectiv(GL_QUERY_RESULT_AVAILABLE, queries[N-1], &available);
}

// See how much time the rendering of object i took in nanoseconds
GLuint64EXT timeElapsed;
for (i = 0; i < N; i++) {
    glGetQueryObjectui64vEXT(queries[i], GL_QUERY_RESULT, &timeElapsed);
    // do something with result
}
Questions?

GPU Programming Guide:

http://developer.nvidia.com

Thanks: Matthias Wloka, Jason Allen, Michael Gold