Programmable GPUS

Last Time:
- Graphics Pipeline
- Clipping
- Rasterization

Today
- Modern Graphics Hardware
- Cg Programming Language
- Gouraud Shading vs. Phong Normal
- Interpolation
- Bump, Displacement, & Environment Mapping

Modern Graphics Hardware
- High performance through
  - Parallelism
  - Specialization
  - No data dependency
  - Efficient pre-fetching

Programmable Graphics Hardware
- Geometry and pixel (fragment) stage become programmable
  - Elaborate appearance
  - More and more general-purpose computation (GPU hacking)

Modern Graphics Hardware
- 2005
  - About 4-6 geometry units
  - About 16 fragment units
  - Deep pipeline (~800 stages)
  - 600 million vertices/second
  - 6 billion texels/second
- NVIDIA GeForce 9 (Feb 2008)
  - ~1 TFLOPS
  - 32/64 stream processors
  - 512 MB/1GB memory
- ATI Radeon R700 (2008?)
  - ~480 stream processing units
<table>
<thead>
<tr>
<th><strong>Today</strong></th>
<th><strong>Emerging Languages</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Modern Graphics Hardware</td>
<td>• Inspired by Shade Trees [Cook 1984] &amp; Renderman Shading Language:</td>
</tr>
<tr>
<td>• <strong>Cg Programming Language</strong></td>
<td>– RTSL [Stanford 2001] – real-time shading language</td>
</tr>
<tr>
<td>• Gouraud Shading vs. Phong Normal Interpolation</td>
<td>– Cg [NVIDIA 2003] – C for graphics</td>
</tr>
<tr>
<td></td>
<td>• CUDA [NVIDIA 2007] – language for general purpose GPU computing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Cg Design Goals</strong></th>
<th><strong>Cg Design</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Ease of programming</td>
<td></td>
</tr>
<tr>
<td>“Cg: A system for programming graphics hardware in a C-like language” Mark et al. SIGGRAPH 2003</td>
<td></td>
</tr>
<tr>
<td>• Portability</td>
<td></td>
</tr>
<tr>
<td>• Complete support for hardware functionality</td>
<td></td>
</tr>
<tr>
<td>• Performance</td>
<td></td>
</tr>
<tr>
<td>• Minimal interference with application data</td>
<td></td>
</tr>
<tr>
<td>• Ease of adoption</td>
<td></td>
</tr>
<tr>
<td>• Extensibility for future hardware</td>
<td></td>
</tr>
<tr>
<td>• Support for non-shading uses of the GPU</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Cg was designed as a “hardware-focused general-purpose language rather than a domain-specific shading language”</td>
</tr>
<tr>
<td></td>
<td>• Multi-program model for Cg to match hardware:</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td></td>
<td>• Can inspect the assembly language produced by Cg compiler and perform additional optimizations by hand</td>
</tr>
<tr>
<td></td>
<td>– Generally once development is complete (&amp; output is correct)</td>
</tr>
<tr>
<td></td>
<td>– Using Cg is easier than writing GPU assembly from scratch</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Cg Design</strong></th>
<th><strong>Cg compiler vs. GPU assembly</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Hardware is changing rapidly… no single standard</td>
<td></td>
</tr>
<tr>
<td>• Specify “profile” for each hardware</td>
<td></td>
</tr>
<tr>
<td>– May omit support of some language capabilities (e.g., texture lookup in vertex processor)</td>
<td></td>
</tr>
<tr>
<td>• Use hardware virtualization or emulation?</td>
<td></td>
</tr>
<tr>
<td>– “Performance would be so poor it would be worthless for most applications”</td>
<td></td>
</tr>
<tr>
<td>– Well, it might be ok for general purpose programming (not real-time graphics)</td>
<td></td>
</tr>
</tbody>
</table>
(Typical) Language Design Issues

- Parameter binding
- Call by reference vs. call by value
- Data types: 32 bit float, 16 bit float, 12 bit fixed & type-promotion (aim for performance)
- Specialized arrays or general-purpose arrays
  - float4 x vs. float x[4]
- Indirect addressing/pointers (not allowed…)
- Recursion (not allowed…)

Data flow in Cg

- Sample vertex program:
  ```c
  void simpleTransform(float4 objectPosition)
  {
    float color;
    float decalCoord;
  
    uniform float brightness,
    uniform float4 modelViewProjection;
  
    clipPosition = mul(modelViewProjection, objectPosition);
    oColor = brightness * color;
    oDecalCoord = decalCoord;
  }
  ```

Today

- Modern Graphics Hardware
- Cg Programming Language
- Gouraud Shading vs. Phong Normal Interpolation
- Bump, Displacement, & Environment Mapping

Remember Gouraud Shading?

- Instead of shading with the normal of the triangle, shade the vertices with the average normal and interpolate the color across each face
  *Illusion of a smooth surface with smoothly varying normals*

Phong Normal Interpolation (Not Phong Shading)

- Interpolate the average vertex normals across the face and compute *per-pixel shading*

Bump Mapping

- Use textures to alter the surface normal
  - Does not change the actual shape of the surface
  - Just shaded as if it were a different shape
Bump Mapping

- Treat the texture as a single-valued height function
- Compute the normal from the partial derivatives in the texture

Another Bump Map Example

What's Missing?

- There are no bumps on the silhouette of a bump-mapped object
- Bump maps don’t allow self-occlusion or self-shadowing

Displacement Mapping

- Use the texture map to actually move the surface point
- The geometry must be displaced before visibility is determined

Displacement Mapping

Image from:

Geometry Caching for Ray-Tracing Displacement Maps
EGRW 1996
Matt Pharr and Pat Hanrahan

note the detailed shadows cast by the stones

Displacement Mapping

Ken Musgrave
Environment Maps

- We can simulate reflections by using the direction of the reflected ray to index a spherical texture map at "infinity".
- Assumes that all reflected rays begin from the same point.

Environment Mapping Example

Terminator II

What's the Best Chart?

Texture Maps for Illumination

- Also called "Light Maps"

Questions?

Image by Henrik Wann Jensen
Environment map by Paul Debevec

Reading for Today:

- Chris Wyman, "An Approximate Image-Space Approach for Interactive Refraction", SIGGRAPH 2005
Reading for Tuesday (3/24)

"Efficient BRDF Importance Sampling Using a Factored Representation"
Lawrence, Rusinkiewicz, & Ramamoorthi, SIGGRAPH 2004

1200 Samples/Pixel

Traditional importance function  Lawrence et al.