Programmable GPUS

Last Time?
• Planar Shadows
• Projective Texture Shadows
• Shadow Maps
• Shadow Volumes
  – Stencil Buffer

Today
• Modern Graphics Hardware
• Shader Programming Languages
• 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)

Misc. Stats on 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)
• NVIDIA® GeForce® GTX 480 (2010)
  – 480 cores
  – 2560x1600 resolution
  – 1536 MB memory
Today

• Modern Graphics Hardware
• Shader Programming Languages
• Gouraud Shading vs. Phong Normal Interpolation
• Bump, Displacement, & Environment Mapping

Emerging Languages

• Inspired by Shade Trees [Cook 1984] & Renderman Shading Language:
  – RTSL [Stanford 2001] – real-time shading language
  – Cg [NVIDIA 2003] – C for graphics

• General Purpose GPU computing
  – CUDA [NVIDIA 2007]
  – OpenCL (Open Computing Language) [Apple 2008]
    for heterogeneous platforms of CPUs & GPUs

Cg Design Goals

• Ease of programming
• Portability
• Complete support for hardware functionality
• Performance
• Minimal interference with application data
• Ease of adoption
• Extensibility for future hardware
• Support for non-shading uses of the GPU

“Cg: A system for programming graphics hardware in a C-like language”
Mark et al. SIGGRAPH 2003

Cg Design

• Hardware is changing rapidly… no single standard
• Specify “profile” for each hardware
  – May omit support of some language capabilities
    (e.g., texture lookup in vertex processor)
• Use hardware virtualization or emulation?
  – “Performance would be so poor it would be worthless for most applications”
  – Well, it might be ok for general purpose programming (not real-time graphics)

Cg compiler vs. GPU assembly

• Can inspect the assembly language produced by Cg compiler and perform additional optimizations by hand
  – Generally once development is complete (& output is correct)
  – Using Cg is easier than writing GPU assembly from scratch

(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…)
GLSL example: checkerboard.vs

```glsl
// a shader for a checkerboard

uniform vec3 camera;
uniform vec3 light;

float diffuse = dot(normal, light); // diffuse shading
float specular = max(0.0, 0.5 * pow(diffuse, 255.0)); // specular shading

void main() {
    vec3 color = mix(vec3(1.0, 1.0, 0.5), vec3(0.0, 0.0, 0.5), specular);
    gl_FragColor = vec4(color, 1.0);
}
```

GLSL example: checkerboard.fs

```glsl
// a shader for a checkerboard

void main() {
    vec3 color = mix(vec3(1.0, 1.0, 0.5), vec3(0.0, 0.0, 0.5), specular);
    gl_FragColor = vec4(color, 1.0);
}
```

Today

- Modern Graphics Hardware
- Shader Programming Languages
- 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](image)

Phong Normal Interpolation (Not Phong Shading)

- Interpolate the average vertex normals across the face and compute per-pixel shading.
  
  ![Must be renormalized](image)

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.

- Sphere w/Diffuse Texture
- Swirly Bump Map
- Sphere w/Diffuse Texture & Bump Map
Another GLSL example: orange.vs

Another GLSL example: orange.fs

Bump Mapping

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

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

Another Bump Map Example

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

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"

Terminator II
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

Readings for Friday:

Choose: