Programmable GPUs

Last Time?

- Planar Shadows
- Projective Texture Shadows
- Shadow Maps
- Shadow Volumes
  - Stencil Buffer
Reading for Today

• "Rendering Fake Soft Shadows with Smoothies", Chan & Durand, EGSR 2003

Reading for Today

“Hardware-Accelerated Global Illumination by Image Space Photon Mapping” McGuire & Luebke, HPG 2009

Figure 1: Image-space photon mapping can compute global illumination at interactive rates for scenes with multiple lights, caustics, shadows, and complex BSDFs. This scene renders at 26 Hz at 1920 × 1080. (Indirect and ambient intensity are amplified for comparison in this image.)
Reading for Today

• "Deep Opacity Maps", Yuksel and Keyser, Eurographics 2008

<table>
<thead>
<tr>
<th>No shadows</th>
<th>Opacity Shadow Maps</th>
<th>Opacity Shadow Maps</th>
<th>Density Clustering</th>
<th>Deep Opacity Maps</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>8 layers</td>
<td>256 layers</td>
<td>4 layers</td>
<td>3 layers</td>
</tr>
<tr>
<td>(104 fps)</td>
<td>(65 fps)</td>
<td>(0.5 fps)</td>
<td>(37 fps)</td>
<td>(50 fps)</td>
</tr>
</tbody>
</table>

Semester Status….

• HW4:
  – Will be posted this weekend… (sorry it’s late)
  – Will be due… in 1.5-2 weeks (it’s smaller than HW3)

• Final Project
  – Proposals due Monday evening, will get you feedback ASAP
  – Your work timeline for the project starts now!
    The first couple weeks are lighter, since you will do HW4 in parallel.
Final Project Presentations

• 34 students @ 10 minutes each
  (20 min for team of 2) = 6 very full hours

• 2 options:
  • Presentations on 3 days, 2 hours each 2-4pm:
    Fri Apr 19th, Tues Apr 23, & Fri Apr 26th
  OR
  • Presentations on 2 days, 3 hours each 2-5pm:
    Tues Apr 23, & Fri Apr 26th
    *Fri Apr 19th is an optional, work-in-class day*

• Project Reports due… Thursday Apr 25 (maybe)

Today

• Modern Graphics Hardware
• Shader Programming Languages
• Gouraud Shading vs. Phong Normal Interpolation
• Many “Mapping” techniques
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
  - 4-6 geometry units, 16 fragment units
  - Deep pipeline (~800 stages)
- NVIDIA GeForce 9 (Feb 2008)
  - 32/64 cores, 512 MB/1GB memory
- ATI Radeon R700 (2008)
  - 480 stream processing units
- NVIDIA GeForce GTX 480 (2010)
  - 480 cores, 1536 MB memory
  - 2560x1600 resolution
- ATI Radeon HD 7900 (2012)
  - 2048 processors, 3GB memory
- NVIDIA GeForce GTX 680 (2012)
  - 1536 cores, 2040 MB memory
- NVIDIA GeForce GTX 980 (2014)
  - 2048 core, 4 GB RAM
- AMD Radeon R9 275x2 (2014)
  - Water cooling, 1200 power supply
  - 5,632 processors, 8 GB RAM
- NVIDIA GeForce 10 (2016)
  - Almost 4,000 (shaders, texture map, render output)
- 2019... same?

Today

- Modern Graphics Hardware
- Shader Programming Languages
  - Cg design goals
  - GLSL examples
- Gouraud Shading vs. Phong Normal Interpolation
- Many “Mapping” techniques
Emerging & Evolving Languages

- Inspired by Shade Trees [Cook 1984] & Renderman Shading Language [1980’s]:
  - RTSL [Stanford 2001] – real-time shading language
  - Cg [NVIDIA 2003] – “C for graphics”
  - GLSL [OpenGL ARB 2004] – OpenGL 2.0
  - Optix [NVIDIA 2009] – Real time ray tracing engine for CUDA

- 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 [2003]… 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…)

Today

• Modern Graphics Hardware
• Shader Programming Languages
  – Cg design goals
  – GLSL examples
• Gouraud Shading vs. Phong Normal Interpolation
• Many “Mapping” techniques
GLSL example: hw4_shader.vs

```c
// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertexPosition_worldspace;
layout(location = 1) in vec3 vertexNormal_worldspace;
layout(location = 2) in vec3 vertexColor;

// Output data
out vec3 vertexPosition_worldspace;
out vec3 vertexNormal_worldspace;
out vec3 vertexColor;
out vec3 EyeDirection_camerspace;
out vec3 myColor;

// Values that stay constant for the whole mesh.
uniform mat4 MVP;
uniform mat4 V;
uniform mat4 N;
uniform vec3 LightPosition_worldspace;

void main(){
    // Output position of the vertex, in clip space : MVP * position
    gl_Position = MVP * vec4(vertexPosition_worldspace,1);

    // Position of the vertex, in worldspace : N * position
    vertexPosition_worldspace = (N * vec4(vertexPosition_worldspace,1)).xyz;

    // Vector that goes from the vertex to the camera, in camera space.
    // In camera space, the camera is at the origin (0,0,0).
    vec3 vertexPosition_camerspace = (V * N * vec4(vertexPosition_worldspace,1)).xyz;

    // Texture coordinate for 2D mapping
    EyeDirection_camerspace = vec3(0,0,1) - vertexPosition_camerspace;
    vec3 vertexNormal_worldspace = normalize((N * vec4(vertexNormal_worldspace,0)).xyz);

    // pass color to the fragment shader
    myColor = vertexColor;
}
```

GLSL example: hw4_shader.fs

```c
in vec3 vertexNormal_worldspace;

// Output data
out vec3 color;

// Values that stay constant for the whole mesh.
uniform vec3 LightPosition_worldspace;
uniform int colormode;
uniform int whichshader;

void main(){
    vec3 checkerboard(vec3 pos) {
        // Determine the parity of this point in the 2D checkerboard
        int count = 0;
        if (mod(pos.x,0.3) > 0.15) count++;
        if (mod(pos.y,0.3) > 0.15) count++;
        if (mod(pos.z,0.3) > 0.15) count++;
        if (count == 1 || count == 3) {
            return vec3(1,1,1);
        } else {
            return vec3(1,1,1);
        }
    }

    vec3 LightColor = vec3(1,1,1);
    float LightPower = 4.0;

    // Surface normal
    vec3 surface_normal = vertexNormal_worldspace;

    // Material properties
    vec3 MaterialDiffuseColor = myColor;
    if (whichshader == 1) {
        MaterialDiffuseColor = checkerboard(vertexPosition_worldspace);
    } else if (whichshader == 2) {
        vec3 normal2;
        MaterialDiffuseColor = orange(vertexPosition_worldspace,surface_normal);
    } else if (whichshader == 3) {
        MaterialDiffuseColor = wood(vertexPosition_worldspace,surface_normal);
    }
```
Remember Gouraud Shading?

- Instead of shading with the normal of the triangle, we’ll shade the vertices with the average normal and interpolate the shaded color across each face
  - Gives the illusion of a smooth surface with smoothly varying normals

Phong Normal Interpolation

- Interpolate the average vertex normals across the face and compute per-pixel shading
  - Normals should be re-normalized (ensure length=1)

- Before shaders, per-pixel shading was not possible in hardware (Gouraud shading is actually a decent substitute!)
Today

- Modern Graphics Hardware
- Shader Programming Languages
- Gouraud Shading vs. Phong Normal Interpolation
- Many “Mapping” techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping

Phong Reflection/Lighting Model

\[ I_p = k_a i_a + \sum_{m \in \text{lights}} (k_d (\hat{L}_m \cdot \hat{N})i_{m,d} + k_s (\hat{R}_m \cdot \hat{V})^\alpha i_{m,s}) \]
The Phong Model

• Sum of three components:
  diffuse reflection + specular reflection + “ambient”.

Another GLSL example: orange.vs

```glsl
varying vec3 normal;
varying vec3 position_eyespace;
varying vec3 position_worldspace;

// a shader that looks like orange peel
void main(void) {

    // the fragment shader requires both the world space position (for
    // consistent bump mapping) & eyespace position (for the phong
    // specular highlight)
    position_eyespace = vec3(gl_ModelViewMatrix * gl_Vertex);
    position_worldspace = gl_Vertex.xyz;

    // pass along the normal
    normal = normalize(gl_NormalMatrix * gl_Normal);

    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```
Another GLSL example: orange.fs

```glsl
varying vec3 normal;
varying vec3 position_eyespace;
varying vec3 position_worldspace;

// a shader that looks like orange peel
void main (void) {
    // the base color is orange!
    vec3 color = vec3(1.0, 0.5, 0.1);
    // high frequency noise added to the normal for the bump map
    vec3 normal2 = normalize(normal + 0.4 * noise3(76.0 * position_worldspace));
    // direction to the light
    vec3 light = normalize(gl_LightSource[1].position.xyz - position_eyespace);
    // direction to the viewer
    vec3 eye_vector = normalize(-position_eyespace);
    // ideal specular reflection
    vec3 reflected_vector = normalize(-reflect(light, normal2));
    // basic phong lighting
    float ambient = 0.4;
    float diffuse = 0.4 * max(dot(normal2, light), 0.0);
    float specular = 0.2 * pow(max(dot(reflected_vector, eye_vector), 0.0), 10.0);
    vec3 white = vec3(1.0, 1.0, 1.0);
    color = ambient * color + diffuse * color + specular * white;
    gl_FragColor = vec4(color, 1.0);
}
```

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

Another Bump Map Example

Cylinder w/Diffuse Texture Map

Cylinder w/Texture Map & Bump Map
Normal Mapping

- Variation on Bump Mapping: Use an RGB texture to directly encode the normal

What's Missing?

- There are no bumps on the silhouette of a bump-mapped or normal-mapped object

- Bump/Normal maps don’t allow self-occlusion or self-shadowing
Today

- Modern Graphics Hardware
- Shader Programming Languages
- Gouraud Shading vs. Phong Normal Interpolation
- Many “Mapping” techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping

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
**Parallax Mapping a.k.a. Offset Mapping or Virtual Displacement Mapping**

- Displace the texture coordinates for each pixel based on view angle and value of the height map at that point
- At steeper view-angles, texture coordinates are displaced more, giving illusion of depth due to parallax effects

“Detailed shape representation with parallax mapping”, Kaneko et al. ICAT 2001

**Parallax Occlusion Mapping**

- Brawley & Tatarchuk 2004
- Per pixel ray tracing of the heightfield geometry
- Occlusions & soft shadows

Today

- Modern Graphics Hardware
- Shader Programming Languages
- Gouraud Shading vs. Phong Normal Interpolation
- Many “Mapping” techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping

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.
What's the Best Chart?

Box Map

Latitude Map

Gl. Map

Environment Mapping Example

Terminator II
Texture Maps for Illumination

• Also called "Light Maps"

Questions?

Image by Henrik Wann Jensen
Environment map by Paul Debevec
Reading for Next Time \( (\textit{pick one}) \)


- Chris Wyman, "An Approximate Image-Space Approach for Interactive Refraction", SIGGRAPH 2005