Shaders and Procedural Modeling

For the Birds, Pixar, 2000
Last Time?

- Planar Shadows
- Projective Texture Shadows
- Shadow Maps
- Shadow Volumes
  - Stencil Buffer

Today

- Worksheet on Real-Time Shadows
- Texture Mapping & Other “Mapping” Techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping
- Programmable Shader Examples
  - Modern Graphics Hardware
  - Per-Pixel Shading
- Procedural Textures & Modeling
- Papers for Today
- Papers for Next Time
Today

- Worksheet on Real-Time Shadows
- **Texture Mapping** & Other “Mapping” Techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping
- Programmable Shader Examples
  - Modern Graphics Hardware
  - Per-Pixel Shading
- Procedural Textures & Modeling
- Papers for Today
- Papers for Next Time

Texture Mapping

For each triangle in the model establish a corresponding region in the phototexture

During rasterization interpolate the coordinate indices into the texture map
Texture Mapping Difficulties

- Tedious to specify texture coordinates
- Acquiring textures is surprisingly difficult
  - Photographs have projective distortions
  - Variations in reflectance and illumination
  - Tiling problems

Common Texture Coordinate Mappings

- Orthogonal
- Cylindrical
- Spherical
- Perspective Projection
- Texture Chart
Projective Textures

- Use the texture like a slide projector
- No need to specify texture coordinates explicitly

Projective Texture Example

- Modeling from photographs
- Using input photos as textures

Figure fromDebevec, Taylor & Malik
http://www.debevec.org/Research
Texture Chart

- Pack triangles into a single image

Today

- Worksheet on Real-Time Shadows
- Texture Mapping & Other “Mapping” Techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping
- Programmable Shader Examples
  - Modern Graphics Hardware
  - Per-Pixel Shading
- Procedural Textures & Modeling
- Papers for Today
- Papers for Next Time
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.

Bump Mapping / Normal 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
Normal Mapping

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

![original mesh](http://en.wikipedia.org/wiki/File:Normal_map_example.png)

<table>
<thead>
<tr>
<th></th>
<th>simplified mesh</th>
<th>simplified mesh and normal mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>original mesh</td>
<td>4M triangles</td>
<td>500 triangles</td>
</tr>
<tr>
<td>simplified mesh</td>
<td>500 triangles</td>
<td>500 triangles</td>
</tr>
</tbody>
</table>

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

• Worksheet on Real-Time Shadows
• Texture Mapping & Other “Mapping” Techniques
  – Bump Mapping
  – Displacement Mapping
  – Environment Mapping
  – Light Mapping
• Programmable Shader Examples
  – Modern Graphics Hardware
  – Per-Pixel Shading
• Procedural Textures & Modeling
• Papers for Today
• Papers for Next Time

Displacement Mapping

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

Originally a CPU-only, post-user-modeling step
Displacement Mapping

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

note the accurate and detailed
shadows cast by the stones

Procedural Displacement Mapping

Ken Musgrave
www.kenmusgrave.com
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

• Worksheet on Real-Time Shadows
• Texture Mapping & Other “Mapping” Techniques
  – Bump Mapping
  – Displacement Mapping
  – Environment Mapping
  – Light Mapping
• Programmable Shader Examples
  – Modern Graphics Hardware
  – Per-Pixel Shading
• Procedural Textures & Modeling
• Papers for Today
• Papers for Next Time

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?

Environment Mapping Example

Terminator II
Texture Maps for Illumination

- Also called "Light Maps"

Questions?

Image by Henrik Wann Jensen
Environment map by Paul Debevec
Today

• Worksheet on Real-Time Shadows
• Texture Mapping & Other “Mapping” Techniques
  – Bump Mapping
  – Displacement Mapping
  – Environment Mapping
  – Light Mapping
• Programmable Shader Examples
  – Modern Graphics Hardware
  – Per-Pixel Shading
• Procedural Textures & Modeling
• Papers for Today
• Papers for Next Time

Modern Graphics Hardware

• Increased parallelism
• Programmable geometry and pixel/fragment stages
• General-purpose computation on GPU (GPGPU)
GLSL example: hw4_shader.vs

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

// Output data
out vec3 vertexPosition_worldspace;
out vec3 vertexNormal_worldspace;
out vec3 EyeDirection_cameraSpace;
out vec3 myColor;

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

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

    // Position of the vertex, in world space : M * position
    vertexPosition_worldspace = M * vec4(vertexPosition_modelspace,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_cameraSpace = ( V * M * vec4(vertexPosition_modelspace,1))).xyz;
    EyeDirection_cameraSpace = vec3(0,0,0) - vertexPosition_cameraSpace;
    vertexNormal_worldspace = normalize(M * vec4(vertexNormal_modelspace,0)).xyz;

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

GLSL example: hw4_shader.fs

```cpp
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;

// a shader for a black & white checkerboard
vec3 checkerboard(vec3 pos)
{
    // determine the parity of this point in the 3D 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(0,1,0);
    } else
    {
        return vec3(1,1,1);
    }
}

void main()
{
    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 normal;
        MaterialDiffuseColor = orange(vertexPosition_worldspace,surface_normal);
    } else if (whichshader == 3) {
        MaterialDiffuseColor = wood(vertexPosition_worldspace,surface_normal);
    }

    vec3 total_diffuse = dot(normal,normalize(MaterialDiffuseColor));
    color = total_diffuse * LightColor;
}
```
Phong Reflection/Lighting Model

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

Color & Normal Interpolation

- It’s easy in OpenGL to specify different colors and/or normals at the vertices of triangles:
- Why is this useful?

Originally, all we could afford to do in hardware was interpolate colors.
Per-Pixel Shading!

- We are not just interpolating the color
- Phong Reflection/Lighting can be calculated per pixel, not just per vertex

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!)
Another GLSL example: orange.vs

```glsl
// 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
// 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(70.0*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 = 0.6;
    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);
}
```
Today

- Worksheet on Real-Time Shadows
- Texture Mapping & Other “Mapping” Techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping
- Programmable Shader Examples
  - Modern Graphics Hardware
  - Per-Pixel Shading
- Procedural Textures & Modeling
- Papers for Today
- Papers for Next Time

Texture Map vs. Solid Texture

“Solid Texturing of Complex Surfaces”, Peachey, SIGGRAPH 1985
Procedural Textures

\[ f(x,y,z) \rightarrow \text{color} \]

- Advantages:
  - easy to implement in ray tracer
  - more compact than texture maps (especially for solid textures)
  - infinite resolution

- Disadvantages
  - non-intuitive
  - difficult to match existing texture
Reading for Today


Perlin Noise

• Properties:
  – Looks “random”, but is deterministic (always returns the same answer for a specific coordinate)
  – Small memory footprint & fast to compute
  – Known amplitude & frequency
  – Smooth interpolation when zoomed in

• Can be combined/layered:
  – Add multiple noise functions w/ different frequencies and amplitudes
  – Simple arithmetic operations (thresholding, sine waves, etc.)
Grey: Delauney Triangulation
- “Best” triangulation of the red dots (most equilateral)
- A specific triangle is in the Delauney Triangle if and only if the circle defined by those 3 points does not contain any other red dot
- Note: Well defined when points are random. If points are on a uniform grid, we have ties…

Black: Voronoi Diagram
- Each cell is the set of all points in the plane that claim that cell’s red dot as the closest
- Note: The black edges perpendicularly bisect the grey edges
Voronoi Diagram/Cells/Regions

- How to re-district the Netherlands into provinces so that everyone reports to the closest capital
- Cell edges are the perpendicular bisectors of nearby points
- 2D or 3D
- Supports efficient Nearest Neighbor queries

http://ccc.inaoep.mx/~rodrigo/robotica/Trigui.pdf
Cellular Textures

www.worley.com

Questions?

Image by Justin Legakis
Optional 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 x 1080. (Indirect and ambient intensity are amplified for comparison in this image.)

Optional Reading for Today

"Geometry Images", Gu, Gortler, & Hoppe, SIGGRAPH 2002

3D shape is unrolled/flattened/stretched into a square image. Stored using existing image formats and compression methods.
Today

- Worksheet on Real-Time Shadows
- Texture Mapping & Other “Mapping” Techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping
- Programmable Shader Examples
  - Modern Graphics Hardware
  - Per-Pixel Shading
- Procedural Textures & Modeling
- Papers for Today
- Papers for Next Time
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping

L-Systems

alphabet: \{a,b\}
initiator: a
production rules:
  a -> b
  b -> ba
generations:
  a
  b
  ba
  bab
  babba
  babbabab
  babbabababababababab

Prusinkiewicz & Lindenmayer,
The Algorithmic Beauty of Plants, 1990
http://algorithmicbotany.org/
L-Systems

http://algorithmicbotany.org/

Animation of Plant Development
Prusinkiewicz et al., SIGGRAPH 1993

“Synthetic Topiary”, Prusinkiewicz, James, and Mech, SIGGRAPH 1994
Cellular Texturing for Architecture

“Feature-Based Cellular Texturing for Architectural Models”, Legakis, Dorsey, & Gortler, SIGGRAPH 2001

Procedural Modeling Advantages

• Small representation
• Generate detail as needed (“infinite”? resolution)
• Great for natural mathematical patterns and man-made engineering and design
• Trivial to make many duplicate objects with small variations
L-Systems for Cities

“Procedural Modeling of Cities”, Parish & Müller, SIGGRAPH 2001

Procedural Modeling of Buildings

Applications

- Entertainment – Gaming
- Education – Studying botanical variation
- Archeological reconstruction
- Realism for Training
- Predicting the future (how will things grow over time)
- Urban planning (preparing for traffic)
- Accommodate for that growth/change

Image-based Procedural Modeling of Facades

- Mueller, Zeng, Wonka, & Van Goo
  SIGGRAPH 2007

Input Photograph
Reconstructed 3D Geometry
Questions about Procedural Modeling

- Number of rules necessary?
- Cost in human designer time of creating procedural model?
- Re-useability of procedural model?
- Validation
- Can you build a procedural model that produces a specific target?
  - From a photo of a specific rare wood grain, can you create a procedural model that creates texture that looks like it came from a different location of the same/similar tree?