Real-Time Shadows

Last Time?
- The Rendering Equation
  \[ L(x', \omega') = E(x', \omega') + \int_{\omega} \rho(\omega, \omega') L(x, \omega) G(x, x') V(x, x') \, dA \]
- Progressive Radiosity
- Adaptive Subdivision
- Discontinuity Meshing
- Hierarchical Radiosity

Today
- Why are Shadows Important?
- Planar Shadows
- Projective Texture Shadows
- Shadow Maps
- Shadow Volumes

Why are Shadows Important?
- Depth cue
- Scene Lighting
- Realism
- Contact points

Shadows as a Depth Cue

For Intuition about Scene Lighting
- Position of the light (e.g. sundial)
- Hard shadows vs. soft shadows
- Colored lights
- Directional light vs. point light
Today

• Why are Shadows Important?
• Planar Shadows
• Projective Texture Shadows
  – Shadow View Duality
  – Texture Mapping
• Shadow Maps
• Shadow Volumes

Cast Shadows on Planar Surfaces

• Draw the object primitives a second time, projected to the ground plane

Limitations of Planar Shadows

• Does not produce self-shadows, shadows cast on other objects, shadows on curved surfaces, etc.

Shadow/View Duality

• A point is lit if it is visible from the light source
  • Shadow computation similar to view computation

Texture Mapping

• Don't have to represent everything with geometry

Fake Shadows using Projective Textures

• Separate obstacle and receiver
• Compute b/w image of obstacle from light
• Use image as projective texture for each receiver

Figure from Moller & Haines "Real Time Rendering"
Projective Texture Shadow Limitations

- Must specify occluder & receiver
- No self-shadows
- Resolution

Questions?

Figure from Moller & Haines: ‘Real-Time Rendering’

Today

- Why are Shadows Important?
- Planar Shadows
- Projective Texture Shadows
- Shadow Maps
- Shadow Volumes

Shadow Maps

- In Renderman
  - (High-end production software)

Shadow Mapping

- Texture mapping with depth information
- Requires 2 passes through the pipeline:
  - Compute shadow map (depth from light source)
  - Render final image, check shadow map to see if points are in shadow

Shadow Map Look Up

- We have a 3D point \((x,y,z)_WS\)
- How do we look up the depth from the shadow map?
- Use the 4x4 perspective projection matrix from the light source to get \((x',y',z')_LS\)
- \(\text{ShadowMap}(x',y') < z'\)?
Limitations of Shadow Maps

1. Field of View
2. Bias (Epsilon)
3. Aliasing

1. Field of View Problem

- What if point to shadow is outside field of view of shadow map?
  - Use cubical shadow map
  - Use only spot lights!

2. The Bias (Epsilon) Nightmare

- For a point visible from the light source
  \[ \text{ShadowMap}(x',y') \approx z' \]
- How can we avoid erroneous self-shadowing?
  - Add bias (epsilon)

2. Bias (Epsilon) for Shadow Maps

\[ \text{ShadowMap}(x',y') + \text{bias} < z' \]
Choosing a good bias value can be very tricky

3. Shadow Map Aliasing

- Under-sampling of the shadow map
- Reprojection aliasing – especially bad when the camera & light are opposite each other

3. Shadow Map Filtering

- Should we filter the depth? (weighted average of neighboring depth values)
- No... filtering depth is not meaningful
3. Percentage Closer Filtering

- Instead filter the result of the test (weighted average of comparison results)
- But makes the bias issue more tricky

Projective Texturing + Shadow Map

Images from Cass Everitt et al., “Hardware Shadow Mapping”
NVIDIA SDK White Paper

Shadows in Production

- Often use shadow maps
- Ray casting as fallback in case of robustness issues

Hardware Shadow Maps

- Can be done with hardware texture mapping
  - Texture coordinates u,v,w generated using 4x4 matrix
  - Modern hardware permits tests on texture values

Questions?
Today

- Why are Shadows Important?
- Planar Shadows
- Projective Texture Shadows
- Shadow Maps
- **Shadow Volumes**
  - The Stencil Buffer


Stencil Buffer

- Tag pixels in one rendering pass to control their update in subsequent rendering passes
  - "For all pixels in the frame buffer" → "For all tagged pixels in the frame buffer"
- Can specify different rendering operations for each case:
  - stencil test fails
  - stencil test passes & depth test fails
  - stencil test passes & depth test passes

Stencil Buffer – Real-time Mirror

- Clear frame, depth & stencil buffers
- Draw all non-mirror geometry to frame & depth buffers
- Draw mirror to stencil buffer, where depth buffer passes
- Set depth to infinity, where stencil buffer passes
- Draw reflected geometry to frame & depth buffer, where stencil buffer passes

[See NVIDIA’s stencil buffer tutorial](http://developer.nvidia.com)
also discusses blending, multiple mirrors, objects behind mirror, etc...

Shadow Volumes

- Explicitly represent the volume of space in shadow
- For each polygon
  - Pyramid with point light as apex
  - Include polygon to cap
- Shadow test similar to clipping

Shadow Volumes

- If a point is inside a shadow volume cast by a particular light, the point does not receive any illumination from that light
- Cost of naive implementation:
  \#polygons \* \#lights

Shadow Volumes

- Shoot a ray from the eye to the visible point
- Increment/decrement a counter each time we intersect a shadow volume polygon (check z buffer)
- If the counter ≠ 0, the point is in shadow
Shadow Volumes w/ the Stencil Buffer

- Initialize stencil buffer to 0
- Draw scene with ambient light only
- Turn off frame buffer & z-buffer updates
- Draw front-facing shadow polygons
  - If z-pass → increment counter
- Draw back-facing shadow polygons
  - If z-pass → decrement counter
- Turn on frame buffer updates
- Turn on lighting and redraw pixels with counter = 0

If the Eye is in Shadow...

- "... then a counter of 0 does not necessarily mean lit"
- 3 Possible Solutions:
  1. Explicitly test eye point with respect to all shadow volumes
  2. Clip the shadow volumes to the view frustum
  3. "Z-Fail" shadow volumes

1. Test Eye with Respect to Volumes

- Adjust initial counter value

Expensive

2. Clip the Shadow Volumes

- Clip the shadow volumes to the view frustum
  - and include these new polygons

Messy CSG

3. "Z-Fail" Shadow Volumes

- Start at infinity
- Draw front-facing shadow polygons
  - If z-fail, decrement counter
- Draw back-facing shadow polygons
  - If z-fail, increment counter
- Introduces problems with far clipping plane
- Solved by clamping the depth during clipping
Optimizing Shadow Volumes

- Use silhouette edges only (edge where a back-facing & front-facing polygon meet)

Limitations of Shadow Volumes

- Introduces a lot of new geometry
- Expensive to rasterize long skinny triangles
- Limited precision of stencil buffer (counters)
  - for a really complex scene/object, the counter can overflow
- Objects must be watertight to use silhouette trick
- Rasterization of polygons sharing an edge must not overlap & must not have gap

Questions?

- From a previous quiz: Check the boxes to indicate the features & limitations of each technique

<table>
<thead>
<tr>
<th>Features / Limitations</th>
<th>Plane Shadows</th>
<th>Projective Shadows</th>
<th>Shadow Maps</th>
<th>Shadow Volumes</th>
<th>Ray Casting Shadows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows objects to cast shadows on themselves (self-shadowing)</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>Permits shadows on arbitrary surfaces (i.e. curved)</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>Resists geometry from the viewpoint of the light</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>Generates extra geometric primitives</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>Limited resolution of intermediate representation can result in jaggie shadow artifacts</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
</tbody>
</table>

Reading for Today:
Lokovic and Veach
Deep Shadow Maps
SIGGRAPH 2000

Reading for Friday 3/23:
Veatch & Guibas
"Optimally Combining Sampling Techniques for Monte Carlo Rendering"
SIGGRAPH 95

Local illumination only
Shadows are important!
Sampling the light source
Sampling the BRDF