Creating soft shadows

Soft vs. hard shadows

Common sense: binary status of shadow
But it looks very unrealistic

Real picture
Why?

Soft vs. hard shadows.

In real life light sources are not points.

Soft shadow creation

Caused by extended light sources

**Umbra**
Source completely occluded

**Penumbra**
Source partially occluded

**Fully lit**
Some assumptions

- One light source
- Monochromatic light source
- No special objects (clouds, hair)
- Many occluder shadows

Shadow map algorithm

- Point of view of the light source
- Method:
  - Z-buffer from light source is stored to shadow map buffer
  - Z-buffer from spectator
  - Comparison distance to light source with shadow map

Shadow volume algorithm

- Geometrical representation
- Extruding of silhouettes creates shadow volume
- Method:
  - Find silhouettes of occluders
  - Extruding silhouettes to shadow volumes
  - For every pixel number of crossed faces of shadow volumes counted
  - If number of total number of faces if positive we are in shadow

Soft shadow algorithms

- Image-based approach (based on shadow map algorithm)
- Object-based approach (based on shadow volume algorithms)

Image-based approach

- Combining some shadow maps from point samples
- Layered shadow maps instead of shadow map
- Some shadow maps take from point samples and computing percentage of light source visibility
- Using standard shadow map with techniques to compute soft shadow

Sampling the Light Source

- Use arbitrary hard shadow algorithm
- Select point sample on area light source
- Render hard shadows
- Sum up weighted result
  (e.g. accumulation buffer/FBO/Multipass)
Combining point light sources

The simplest method by Herf (1997)

Method

For every sample compute binary occlusion map
Computing attenuation map storing for every pixel how many light source samples occluded

Combining point light sources

Time complexity (N/σNp for attenuation map)

With fewer than 9 samples user sees number of hard shadows
Parallelizable

Sampling the Light Source

Example: Ground plane shadow texture

1. Initialize FB (white)
2. For each sample point do
   2a. Render scene
   2b. Subtract 1/N from FB only once for each pixel (stencil)!

Image from ATI Developer’s Site
Layered shadow map
Extension of previous method
Method
Z-buffer for light point samples
Warp to center of light source

Precomputation
1st viewpoint

Precomputation
1st viewpoint
Warped 2nd viewpoint

Layered shadow map
Build attenuation map
Store layer and distance to light source in shadow map
While rendering render color of pixel according to layer and attenuation map

Visibility channel
For linear light sources by Heifrich (2000)
For polygonal sources by Ying (2002)
Method
Low number of samples (usually 2)
Detect discontinuities in shadow maps
Polygon linking
Gouraud shading: 1 for farthest 0 for closer
Single sample soft shadow

Method:
- Standard shadow map from center of light source
- While rendering
  - If pixel is lit, find nearest shadowed pixel
  - If pixel is shadowed, find nearest lit pixel

Calculating f:
- Intensity of light:
  - 0.5*(1+f) for outer penumbra limited to [0.5, 1]
  - 0.5*(1-f) for inner penumbra limited to [0, 0.5]

Disadvantages:
- Bottleneck: to find nearest lit/shadowed pixel
- Doesn’t depend on size of light source, only from distances

Object based approach

Combining some hard shadows
- Extending shadow volume by heuristic
- Computing penumbra volume for each edge

Combining hard shadows

The simplest method to produce soft shadow
Method:
- Several light source samples
- Build shadow volumes for each sample
- Average received pictures

Stenciled Shadow Volumes for Simulating Soft Shadows

Cluster of 12 dim lights approximating an area light source.
Generates a soft shadow effect; careful about banding. 8 fps on GeForce4 Ti 4600.

The cluster of point lights.
Plateau Soft Drop Shadows
Say we want to paint a soft shadow for an image.

Soft planar shadows
Haines (2001)
Planar receiver
Method
- Standard shadow volume algorithm
- Vertices of silhouette turned to cones
- Building edges around cones
Disadvantages:
- Planar surfaces
- Spherical light source
- Outer penumbra
- Penumbra depends only from distance occluder-receiver

Plateau Idea
"Paint" each shadow’s edge, blurring it as its height from the plane increases.

Forming Plateaus
Create the shadow object
Apply rendering as texture
Render from above

Plateau Result
Soft shadows, at the cost of finding the silhouette and drawing cones & sheets.

Plateau Limitations
Overstated umbrae, penumbrae are not physically correct.
Plateau Shadows (1 pass)   Heckbert/Herf (256 passes)
Penumbra Maps

Builds on simple idea of “shadow plateaus” introduced by Haines (’01)
Plausible soft shadows
Hard upon contact, soft with distance
Simple implementation on graphics hardware
Hides some aliasing
One sample per pixel

Penumbra Map Assumptions

A hard shadow is a reasonable approximation for a shadow’s umbra
Object silhouettes remain constant over light’s surface

Key Insight

When using a hard shadow as the umbra, all of the approximate penumbra is visible from the center of the light
Allows storage of penumbral intensity in a separate map called a penumbra map

Creating Penumbra Map

Compute shadow map (for hard shadow)
Compute object silhouette from light’s center
Compute cones at silhouette vertices
Compute sheets connecting vertices (along silhouette edges)

Computing Cones

For each silhouette vertex
Find distance from light’s center to vertex
Find distance from vertex to far plane
Using these distances and the light radius $L_r$, compute $C_r$ using similar triangles

Computing Sheets

Create quads at each silhouette edge tangent to the adjacent cones
May not be planar
Subdivide significantly non-planar quads for good results
Shadowing Complex Objects

Can not just draw quad with 0 (shadowed) at A and 1 (illuminated) at C
Result depends on current fragment F on quad and point P in the shadow map

Use Fragment Program to Generate Map

\[
\begin{align*}
\text{FragmentProgram} (Z_w, F, \text{Shadow}) \\
1. \quad Z_{\text{norm}} = \text{GetWindowCoord}(F) \\
2. \quad Z_F = \text{TextureLookup}(\text{Shadow}, \text{Coord}) \\
3. \quad Z_p = Z_{\text{norm}} \\
4. \quad \text{if } (Z_F < Z_p) \text{ DiscardFragment} \\
5. \quad Z_F = \text{ConvertToWorldSpace}(Z_F) \\
6. \quad Z_F = Z_F - Z_p \\
7. \quad f = (Z_F - Z_p) / (Z_F - Z_p) \\
8. \quad \text{OutputColor} = f \\
9. \quad \text{OutputDepth} = f \\
\end{align*}
\]

Rendering

Compare fragment’s depth to shadow map to determine if light is completely blocked

If not completely shadowed, index into penumbra map to determine percentage of light reaching surface

Multiple lights requires multiple shadow and penumbra maps

Video

Results

Framerates are ~18 Hz with 1024² shadow map, penumbra map, and image size
Note the pathtraced image uses the larger light
Use 10k triangle bunny to generate shadows

More Results

• Framerates are ~15 Hz for 1024² resolution
**Changing Penumbra Map Size**

<table>
<thead>
<tr>
<th>Map Size</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024²</td>
<td><img src="image1.png" alt="Image" /></td>
</tr>
<tr>
<td>512²</td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td>256²</td>
<td><img src="image3.png" alt="Image" /></td>
</tr>
<tr>
<td>128²</td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
</tbody>
</table>

**Problems**

- Shadows are not accurate
  - Less accurate as occluders move further away from shadowed objects
- Assume silhouettes constant over light
  - Noticeable pops on cube
  - No problems with other objects
- Blending overlapping penumbras
  - Occurs on a per-pixel basis
  - No geometric info in the hardware
  - Artifacts at silhouette concavities

**Blending Issues**

In these three cases, overlapping penumbras should be handled differently

- No geometric information in the pixel program means no quick way to decide in hardware
- We always choose the darkest pixel (left image)

**Same as Smoothies….**

**Smoothies**

- Chan (2003)
- Method
  - Shadow map
  - Identify silhouette edges
  - Construct smoothies
  - Render smoothies

**Smoothies**

- Disadvantages
  - Outer penumbra only
  - There is always umbra
  - Connecting edges
Penumbra wedges

Akentine-Moller and Assarsson (2002-03)

Method
- Building silhouette from single sample
- Building penumbra wedges
- Shadow volume algorithm
- If point inside wedge algorithm uses fragment programs implemented in hardware

 PENUMBRA MAPS

<table>
<thead>
<tr>
<th>Penumbra Maps</th>
<th>Smoothies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry: cones and sheets</td>
<td>quads</td>
</tr>
<tr>
<td>Store depth: blockers only</td>
<td>blockers + smoothies</td>
</tr>
</tbody>
</table>

Smoothie depth:
- Extra storage + comparison
- Handles surfaces that act only as receivers
Penumbra wedges

Some Issue (need to ‘round’ corners)

Penumbra wedges

Penumbral Wedge Rendering

Soft Shadow Correction

Soft Shadow Correction

Penumbra wedges

Penumbra wedges

Soft Shadow Correction

Lighting pass for ordinary stencil shadows uses stencil test
0 in stencil buffer at a particular pixel means light can reach that pixel
Nonzero means pixel is in shadow
Soft Shadow Correction

For soft shadows, use alpha blending during lighting pass

Value in the alpha channel represents how much of the area light is covered

0 means entire light source visible from a particular pixel
1 means no part of light source is visible (fully shadowed)

How the visibility computation works:

How do we compute this fast with a pixel shader?

Precomputed contribution in 4D textures

4D textures used as look-up table

Enables
Fast computation
Textured light sources (e.g., fire)
Colored shadows.
Fire video

More examples using textured lights

Texture of 16 area lights
Texture of two colors

Colored Lights

Soft Shadow Correction
Render the shadow volumes into a 16-bit floating-point render target

Penumbral Wedge Rendering
In the vertex program, we compute the three outside bounding planes of a half-wedge
Send these planes to the fragment program in viewport space!
Allows us to do a quick test to determine whether a viewport-space point is outside the half-wedge

Penumbral Wedge Rendering
In the fragment program, we test the viewport-space position of the point in the frame buffer against three half-wedge bounding planes
We will use the depth test to reject points on the wrong side of the extruded silhouette edge
Penumbral Wedge Rendering

Sort half-wedges into two batches:
1) Those for which camera is on the positive side of the silhouette edge
2) Those for which camera is on the negative side of the silhouette edge

Extruded silhouette plane normal always points outward from shadow volume

Rendering Outer Half-wedges

Half-wedges for which camera is on positive side of silhouette plane
Render front faces when z test fails

Half-wedges for which camera is on negative side of silhouette plane
Render back faces when z test passes

Penumbral Wedge Rendering

If the value was greater than one, then it's saturated to one, corresponding to fully shadowed
Then render lighting pass, multiplying source color by one minus destination alpha

`glBlendFunc(GL_ONE_MINUS_DST_ALPHA, GL_ONE);`
Semi-penumbral Shadows
Method for speeding up penumbral wedge soft shadows
Only render outer half-wedges
Less correct, but still looks good
Lose the ability to cast shadows that have no point of 100% light occlusion

A Penumbral Wedge

Semi-penumbral Shadows
Instead of full penumbra:
Render outer half of penumbra only:

Summary
Hard vs. soft shadows
Existing algorithms for soft shadow creation
Advantages and disadvantages of each algorithms
Bibliography

Tomas Akenine-Möller and Ulf Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges.
Eric Chan and Fredo Durand. Rendering fake soft shadows with smoothies.