Shading 5: Shadows

Christian Miller
CS 354 - Fall 2011
Exam 1 results

- Mean 62, std. dev. 18, min 15, max 94
- Totally uncorrelated with project scores
Shadows are important

- They’re one of the best depth cues we have, even better than binocular vision
• Umbra is totally dark, penumbra fades into light
• Point light sources have no penumbra
Shadows in raytracing

- Handled very naturally by casting shadow rays
- Soft shadows require a sampling method
- Shadow rays are too expensive to do in realtime
In a rasterizer

- We generate pixels for one triangle at a time
- We don’t have any other geometry at hand
- We do have the texture unit and shaders to use...
Planar shadows

- Draw all the geometry again, in black, with a modelview matrix that flattens it onto a plane
- Ends up being like a perspective transform
Planar shadows

\[
M = \begin{bmatrix}
  n \cdot l + d - l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\
  -l_y n_x & n \cdot l + d - l_y n_y & -l_y n_z & -l_y d \\
  -l_z n_x & -l_z n_y & n \cdot l + d - l_z n_z & -l_z d \\
  -n_x & -n_y & -n_z & n \cdot l
\end{bmatrix}
\]
Stencil Buffer

- Another layer in the frame buffer, like color/depth
- Contains arbitrary integer values
- Writes into framebuffer can modify the stencil values, such as increasing or decreasing them
- Can also apply a stencil test to cull the pixels that are drawn onto the screen
- Frequently used to mask shapes
Stencil reflections

- Same operation can be used to clip planar shadows

[RTR]
Be careful

• Projection is defined even behind the light, so you get inverse shadows
Other Problems

• Handles just one plane
• Can’t reliably do transparent shadows
• Can’t do curved surfaces
• Can’t do self-shadowing
• No obvious way to do soft shadows
• It’s cheap and simple, what do you expect?
More general

• We need something that will tell us if any given pixel is visible from the light or not
• Shape of the occluded surface shouldn’t matter
• Would be nice if it could take advantage of texture hardware
Projective textures

- Recall from a few lectures ago that we can project textures using homogeneous texture coordinates.
Projective shadows

- Draw object silhouette from light’s point of view into a texture, then project onto scene
- Requires a separate shadow texture for each object, doesn’t handle self-shadowing

[RTR]
Shadow mapping

• Idea: render depth buffer from light’s perspective into a texture, instead of just a silhouette

• Project that texture onto the scene

• At each pixel, compute the pixel’s depth value relative to the light, compare against shadow map, and that tells you if the pixel is in shadow
Inaccuracy

- Depth buffer has finite numerical accuracy, and each shadow map pixel represents an angled area in the final image.
Biasing

- A cheap fix for this is adding a “bias” in the shadow comparison
- Can overdo it and end up moving shadow away from the object the generated it
Resolution mismatch

- If the shadow map is too low-res, the results are hideous
Skewing samples

- By rotating the shadow map near plane, you can skew the sampling pattern to get higher resolution near the eye.
Skewing samples
Cascaded shadow maps

• Have a bunch of shadow maps of different resolutions, use the tiny high-detail ones close to eye
Blurring shadow maps

- Naively averaging several depth values produces a nonsense value somewhere in between.
- Instead, do the shadow test on several shadow texels, then blend the results of those tests together:
  - Basically, average the number of texels in shadow.
- Called percentage closer filtering (PCF).
- Gives a very crude penumbra as a bonus.
PCF results
**Shadow map comments**

- Can capture all directions around a light with a cube shadow map
- The most common way to create shadows in games
- Low-res shadow maps are responsible for much of the ugly in real time rendering
Shadow volumes

• A different way to generate hard shadows that just uses geometry

• The basic idea: compute the 3D volumes that are in shadow, and check every pixel to see if they’re in one of those volumes

• There’s a cool trick to make this efficient
Shadow volumes

- Point
- Triangle
- Infinite pyramid
- Pyramid
- Shadow volume
Making shadow volumes

- Find the silhouette edges (one triangle is front facing the light, the other is back facing)
- Project those edges out to infinity using an infinite far-plane perspective matrix (lecture 7)
- Connect those to original edges with quads
Once you have shadow volumes, you can count how many times a ray enters and leaves them to determine if a point is in shadow.
Stencil buffer version

- First, render all scene geometry into depth buffer
- Next, render shadow volumes into stencil buffer
  - Have all front faces increment stencil value
  - Have all back faces decrement stencil value
- After this, all zeros in the stencil buffer are lit pixels, all nonzeros are shadowed
  - Render diffuse and specular using that info
Shadow volumes in use

- This is (almost) exactly the shadowing technique used in Doom 3
Caveats

• Counting trick needs to be modified if eye is in shadow (since this throws off the counts)

• Shadow volumes generate many polygons that cover many, many pixels on the screen
  
  • This makes the GPU re-render each pixel several times

• Result is per-pixel accurate
Shadow volume complexity:
- 1 to 5 fills
- 6 to 10 fills
- 11 to 20 fills
- 21 to 30 fills
- 31 to 40 fills
- 41 to 50 fills
- 51 to 70 fills
- 71 or more fills
FIGURES COURTESY...

- Real-Time Rendering, 3rd ed. [RTR]
  - Tomas Akenine-Moller, Eric Haines, Naty Hoffman
  - Eric Lengyel
  - Edward Angel, Dave Shreiner
- Wikipedia [WP]
- Light Space Perspective Shadow Maps, Wimmer et al., EGSR 2004 [LSPSM]
- Shadow Map Antialiasing, by Michael Bunnell, Ch. 11 in GPU Gems [SMA]
- Nvidia Projective Texture Mapping notes [PTM]