Shadows

Wolfgang Heidrich

Course News

Assignment 3 (project)
- Due April 1

Reading
- Chapter 11.8, 10

Next Friday
- Will be out of town for committee meeting
- Replacement TBD
Shadows

**What are shadows?**
- What distinguishes a point in shadow from a lit point?

---

**Types of light sources**
- Point, directional
- Area lights and generally shaped lights
  - *Not considered here*
  - *Later: ray-tracing for such light sources*

**Problem statement**
- A shadow algorithm for point and directional lights determines which scene points are
  - *Visible from the light source (i.e. illuminated)*
  - *Invisible from the light source (i.e. in shadow)*
- Thus: shadow casting is a visibility problem!
Types of Shadow Algorithms

**Object Space**
- Like object space visibility algorithms, the method computes in object space which polygon parts that are illuminated and which are in shadow
  - *Individual parts are then drawn with different intensity*
- Typically slow, $O(n^2)$, not for dynamic scenes

**Image Space**
- Determine visibility per pixel in the final image
  - *Sort of like depth buffer*
  - *Shadow maps*
  - *Shadow volumes*

---

Credits

- The following shadow mapping slides are taken from Mark Kilgard’s OpenGL course at Siggraph 2002.
**Shadow Mapping Concept (1)**

*Depth testing from the light’s point-of-view*

- Two pass algorithm
- First, render depth buffer from the light’s point-of-view
  - The result is a “depth map” or “shadow map”
  - Essentially a 2D function indicating the depth of the closest pixels to the light
- This depth map is used in the second pass

**Shadow Mapping Concept (2)**

*Shadow determination with the depth map*

- Second, render scene from the eye’s point-of-view
- For each rasterized fragment
  - Determine fragment’s XYZ position relative to the light
  - This light position should be setup to match the frustum used to create the depth map
  - Compare the depth value at light position XY in the depth map to fragment’s light position Z
The Shadow Mapping Concept (3)

The Shadow Map Comparison

- Two values
  - $A = Z$ value from depth map at fragment’s light XY position
  - $B = Z$ value of fragment’s XYZ light position
- If $B$ is greater than $A$, then there must be something closer to the light than the fragment
  - Then the fragment is shadowed
- If $A$ and $B$ are approximately equal, the fragment is lit

Shadow Mapping with a Picture in 2D (1)

The $A < B$ shadowed fragment case

- light source
- depth map image plane
- depth map $Z = A$
- eye position
- eye view image plane, a.k.a. the frame buffer
- fragment’s light $Z = B$
Shadow Mapping with a Picture in 2D (2)

The \( A = B \) lit fragment case

Visualizing the Shadow Mapping Technique (1)

A scene with fairly complex shadows
Visualizing the Shadow Mapping Technique (2)

Compare with and without shadows

with shadows

without shadows

Visualizing the Shadow Mapping Technique (3)

The scene from the light’s point-of-view

FYI: from the eye’s point-of-view again
Visualizing the Shadow Mapping Technique (4)

The depth buffer from the light's point-of-view

FYI: from the light's point-of-view again

Visualizing the Shadow Mapping Technique (5)

Projecting the depth map onto the eye's view

FYI: depth map for light's point-of-view again
Visualizing the Shadow Mapping Technique (6)

*Projecting light’s planar distance onto eye’s view*

Visualizing the Shadow Mapping Technique (6)

*Comparing light distance to light depth map*

Green is where the light planar distance and the light depth map are approximately equal. Non-green is where shadows should be.

Wolfgang Heidrich
Visualizing the Shadow Mapping Technique (7)

Complete scene with shadows

Notice how specular highlights never appear in shadows

Notice how curved surfaces cast shadows on each other

In Practice: Depth Map Precision Issues

Have to add a little offset to depth map values to account for limited precision

Too little bias, everything begins to shadow

Too much bias, shadow starts too far back

Just right
What is Projective Texturing?

An intuition for projective texturing
- The slide projector analogy

About Projective Texturing (1)

First, what is perspective-correct texturing?
- Normal 2D texture mapping uses \(s, t\) coordinates
- 2D perspective-correct texture mapping
  - Means \((s, t)\) should be interpolated linearly in eye-space
  - So compute per-vertex \(s/w, t/w, \) and \(1/w\)
  - Linearly interpolated these three parameters over polygon
  - Per-fragment compute \(s' = (s/w) / (1/w)\) and \(t' = (t/w) / (1/w)\)
  - Results in per-fragment perspective correct \(s', t'\)
So what is projective texturing?

- Now consider homogeneous texture coordinates
  - \((s, t, r, q) \rightarrow (s/q, t/q, r/q)\)
  - Similar to homogeneous clip coordinates where 
    \((x, y, z, w) = (x/w, y/w, z/w)\)
- Idea is to have \((s/q, t/q, r/q)\) be projected per-fragment

Assign light-space texture coordinates to polygon vertices

- Transform eye-space \((x, y, z, w)\) coordinates to the light’s view frustum (match how the light’s depth map is generated)
- Further transform these coordinates to map directly into the light view’s depth map
  - *Expressible as a projective transform*
- \((s/q, t/q)\) will map to light’s depth map texture
Shadow Map Operation

Next Step:

- Compare depth map value to distance of fragment from light source
- Different GPU generations support different means of implementing this
  - Today’s GPUs: pixel shader!
  - Earlier: special hardware for implementing this feature (e.g. SGI), or just using alpha blending [Heidrich’99]

Issues with Shadow Mapping (1)

Not without its problems

- Prone to aliasing artifacts
  - “percentage closer” filtering helps this
  - normal color filtering does not work well
- Depth bias is not completely foolproof
- Requires extra shadow map rendering pass and texture loading
- Higher resolution shadow map reduces blockiness
  - but also increase texture copying expense
Hardware Shadow Map Filtering Example

**GL_NEAREST:** blocky  **GL_LINEAR:** antialiased edges

*Low shadow map resolution used to heighten filtering artifacts*

---

Issues with Shadow Mapping (2)

**Not without its problems**

- Shadows are limited to view frustums
  - *could use six view frustums for omni-directional light*
- Objects outside or crossing the near and far clip planes are not properly accounted for by shadowing
  - *move near plane in as close as possible*
  - *but too close throws away valuable depth map precision when using a projective frustum*
More Examples

Complex objects all shadow

More Examples

Even the floor casts shadow

Note shadow leakage due to infinitely thin floor

Could be fixed by giving floor thickness

Wolfgang Heidrich
Combining Projective Texturing for Spotlights

Use a spotlight-style projected texture to give shadow maps a spotlight falloff

Combining Shadows with Atmospherics

Shadows in a dusty room

Simulate atmospheric effects such as suspended dust
1) Construct shadow map
2) Draw scene with shadow map
3) Modulate projected texture image with projected shadow map
4) Blend back-to-front shadowed slicing planes also modulated by projected texture image

Credit: Cass Everitt
**Shadow Maps**

**Approach for shadows from point light sources**

- Surface point is in shadow if it is not visible from the light source
- Use depth buffer to test visibility:
  - Render scene from the point light source
  - Store resulting depth buffer as texture map
  - For every fragment generated while rendering from the camera position, project the fragment into the depth texture taken from the camera, and check if it passes the depth test.

**Shadow Volumes**

**Use new buffer: stencil buffer**

- Just another channel of the framebuffer
- Can count how often a pixel is drawn

**Algorithm (1):**

- Generate silhouette polygons for all objects
  - Polygons starting at silhouette edges of object
  - Extending away from light source towards infinity
  - These can be computed in vertex programs
Shadow Volumes

Algorithm (2):

- Render all original geometry into the depth buffer
  - *I.e. do not draw any colors (or only draw ambient illumination term)*
- Render front-facing silhouette polygons while incrementing the stencil buffer for every rendered fragment
- Render back-facing silhouette polygons while decrementing the stencil buffer for every rendered fragment
- Draw illuminated geometry where the stencil buffer is 0, shadow elsewhere
Shadow Volumes

Discussion:
- Object space method therefore no precision issues
- Lots of large polygons: can be slow
  - High geometry count
  - Large number of pixels rendered
Coming Up:

Next Week:
- Ray-tracing
- Global illumination