Announcements

- HW 1 graded
- HW 2 due tomorrow
  - Turn in code AND classes in jar file
  - Do NOT hard-code parameters
  - Examples with noise have been posted

Next-Event Estimation

- How does it work?

Shadows

Methods for fast shadows:

- Shadow Maps
- Shadow Volumes

Using the Shadow Map

- When scene is viewed, check viewed location in light's shadow buffer
  - If point's depth is (epsilon) greater than shadow depth, object is in shadow

For each pixel, compare distance to light \$d\$ with the depth \$z\$ stored in the shadow map

Shadow Mapping: Pass 1

- Depth testing from light's point-of-view
  - Two pass algorithm

  - First, render depth buffer from light's point-of-view
    - Result is a "depth map" or "shadow map"
    - A 2D function indicating the depth of the closest pixels to the light
    - This depth map is used in the second pass
Shadow Mapping: 2nd pass

- 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

Shadow Map Issues

- Can only cast shadows over a frustum
  - Use 6 (like a cube map)

- Get speckling because of floating point errors
  - Use triangle ids
  - Use bias
    * If \(B > A + \text{bias}\) p in shadow

Properties of Shadow Maps

- One shadow map per light
- Render scene twice per frame
  - If static, can reuse

- Advantages
  - Fast
  - Easy to implement

- Disadvantages
  - Bias
  - Aliasing
  - Hard shadows

Why does Aliasing arise?

- Light
- Shadow Map
- Image Plane
- Eye View
- Projected Area
  - Eye View
  - Projected Area
  - Shadow Map

Shadow Volumes

- Clever counting method using stencil buffer
- Can cast shadows onto curved surfaces

Algorithm

- Finding volumes
  - Project out shadow volumes

- Rendering
  - Render scene into z-buffer, freeze z-buffer
  - Draw front-facing volumes in front/back of pixel
    * increment stencil
  - Draw back-facing volumes in front/back of pixel
    * decrement stencil
  - If \(\text{cnt} == 0\) lit else shadow
Z-fail Approach

- frontfacing
- backfacing

Performance

- Have to render lots of huge polygons
  - Front face increment
  - Back face decrement
  - Possible capping pass
- Uses a LOT fill rate
- Gives accurate shadows
  - IF implemented correctly
- Need access to geometry if want to use silhouette optimization

Comparison

- Shadow Maps
  - Adv: Fixed resolution, fast, simple
  - Disadv: Bias, aliasing
- Shadow Volumes
  - Adv: Accurate, high-quality
  - Disadv: Fill-rate limited, hard to implement robustly

Approaches to Improve Shadows

- Hard Shadows
  - Adaptive Shadow Maps [Fernando, Fernandez, Bala, Greenberg]
  - Shadow Silhouette Maps[Sen, Cammarano, Hanrahan]
- Hard and Soft Shadows
  - Edge-and-Point Rendering [Bala, Walter Greenberg]
- Soft Shadows

Adaptive Shadow Maps: Motivation

- Fernando, Fernandez, Bala, Greenberg [SIG01]

- Shadow maps require too much tweaking
  - Where to place light?
  - What resolution to use?

- Goals:
  - Address the aliasing problem
  - No user intervention
  - Interactive frame rate

Adaptive Shadow Maps

- Idea:
  - Refine shadow map on the fly
- Goal:
  - Shade each eye pixel with a different shadow map pixel
- Implementation:
  - Use hierarchical structure for shadow map
  - Create/delete pieces of shadow map as needed
  - Exploit fast rendering and frame buffer read-backs
Results: Images (Mesh)

- **Conventional Shadow Map** (2048 x 2048 pixels)
  - 16 MB Memory Usage
- **Adaptive Shadow Map** (Variable Resolution)
  - 16 MB Memory Usage

Equivalent Conventional Shadow Map Size: 65,536 x 65,536 Pixels

---

Results: Images (Mesh Close-Up)

- **Conventional Shadow Map**
  - 16 MB Memory Usage
- **Adaptive Shadow Map**
  - 16 MB Memory Usage

---

Edge-and-Point Rendering [Bala03]

- Edges: important discontinuities
  - Silhouettes and shadows
- Points: sparse shading samples

---

Edge-and-Point Image

- Alternative display representation
- Edge-constrained interpolation preserves sharp features
- Fast anti-aliasing

---

Edge-and-Point Image (EPI)

- **Goal**: compact and fast
  - Store at most one edge and one point per pixel
  - Limited sub-pixel precision
  - Pre-computed tables give fast anti-aliasing

---

Edge Reconstruction

- Rasterize edges onto image plane
- Record their intersections with pixel boundaries
- Can handle high complexity objects
Edge Finding

- Hierarchical trees: fast edge finding
  - Fraction of a second

Soft Shadow Edges

Black: silhouettes, Red: umbral edges, Blue: penumbral edges

Results

- Fast edge finding
- Accurate shadow reconstruction (similar to shadow volume quality)
- Pre-computed tables give fast anti-aliasing

Silhouette Shadow Map

- Shadow maps with silhouettes for precision and low fill rate
- Silhouette map: texture map (depth + silhouette)
  - Texel represents (x,y) of point on silhouette
  - At most one pt per texel: at most 1 silhouette
- Render with silhouette map
- Overall 3 passes

Rendering with Silhouette Map

Implementation

- ATI Radeon 9700 Pro
Results

- Relatively simple scenes: 1k-14k triangles
- Little slower than shadow volumes
  - but lower overdraw

Soft Shadows

- Soft shadows appear natural
- Hard to get soft shadows in hardware
- Slow in software

Heckbert and Herf

- Use accumulation buffer
- Render shadows from multiple point lights over the area light (like MC)
- Accumulate shadows

Heckbert/Herf Soft Shadows

- Advantage: gives true penumbra
- Limitations: overlapping shadows are unconvincing unless a lot of passes are made

Images courtesy of Michael Herf and Paul Heckbert
Soft Shadow Approximations

- Approximations
  - People can’t tell the difference
  - Good for games
- Convolution
- Penumbra Maps
- Penumbra Wedges

Haines: Shadow Plateaus

- Compute soft shadows on a plane
- Start with umbra from light’s center
- Blur outward from umbra to get penumbra

Figure 16: A single 128 × 128 shadow map was computed for the sheet of silhouettes, and used to shade shadows on each individual silo according to its location in space.

Haines: Shadow Plateaus

- Find silhouettes and draw cones & sheets
- Apply rendering as texture

Plateau Limitations

- Overstated umbra
- Penumbra not physically correct

Soler and Sillion

- Shadows as convolution

Plateau Shadows (1 pass) Heckbert/Herf (256 passes)
Penumbra Maps

- Wyman and Hansen
- Use shadow map and Haines technique for soft shadows on arbitrary surfaces
- Penumbra map
- Stores intensity of shadow
- Overall:
  - 3 pass: shadow map and penumbra map
  - Render image using depth from shadow map and intensity from penumbra map

Method Details: Visualization

Computing Penumbra Map Values

Uses fragment program

\[ I = \frac{Z_p - Z_F}{Z_p - Z_i} \]

- \( Z_p \) Distance to vertex \( v \)
- \( Z_F \) Distance to cone/sheet fragment
- \( Z_i \) Depth of shadow map pixel
- \( I \) Intensity in the scene

Rendering

- Render from camera’s viewpoint
- If occluded in shadow map, in umbra
- Else, modulate with value from penumbra map

Results
Assumptions

• Umbra from center is the real umbra; full penumbra visible from center
• Umbra is fixed size irrespective of size of light: over-stated umbra
• Silhouette stays fixed over light