Advanced Shading I: Shadow Rasterization Techniques
Shadow Terminology

**umbra**: light totally blocked

**penumbra**: light partially blocked

**occluder**: object blocking light
Shadow Terminology

**umbra**: light totally blocked

**penumbra**: light partially blocked

**occluder**: object blocking light

Point lights have no penumbra
Shadow Rendering

Hard shadows: umbra only
• easy with ray tracing (shadow rays)

Soft shadows: penumbra and umbra
• very difficult without global illumination

Today: rasterizing hard shadows
Projection Shadows

Easiest case:
- one object
- one light
- shadow cast on flat ground
Projection Shadows

Main idea: render scene twice
• once as usual
• once with object shaded black and flattened onto plane
Projection Shadows

projection of $v$ lies along ray $l + (v - l)t$
Projection Shadows

projection of $v$ lies along ray $l + (v - l)t$

$$[(l - b) + (v - l)t] \cdot \hat{n} = 0$$

$$t = \frac{(b-l) \cdot \hat{n}}{(v-l) \cdot \hat{n}}$$
Projection Shadows

projection of \( v \) lies along ray \( l + (v - l)t \)

\[
[(l - b) + (v - l)t] \cdot \hat{n} = 0
\]

\[
t = \frac{(b-l) \cdot \hat{n}}{(v-l) \cdot \hat{n}}
\]

\[
p = l + (v - l) \frac{(b-l) \cdot \hat{n}}{(v-l) \cdot \hat{n}}
\]
Projection Shadows

projection of v lies along ray $l + (v - l)t$

\[
[(l - b) + (v - l)t] \cdot \hat{n} = 0
\]

\[
t = \frac{(b - l) \cdot \hat{n}}{(v - l) \cdot \hat{n}}
\]

\[
p = l + (v - l) \frac{(b - l) \cdot \hat{n}}{(v - l) \cdot \hat{n}}
\]

how to write as affine transformation?
Projection Operation

Needed:

\[
M \begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix} = \frac{(x, y, z)}{(x, y, z) \cdot \hat{n}}
\]
Projection Operation

Needed:

\[
M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \frac{(x,y,z)}{(x,y,z) \cdot \hat{n}}
\]

\[
M = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
\hat{n}^T & 0 & 0 & 0 & 0
\end{bmatrix}
\]
Projection Shadows

projection of $v$ lies along ray $l + (v - l)t$

$$p = l + (v - l)\frac{(b-l) \cdot \mathbf{n}}{(v-l) \cdot \mathbf{n}}$$

how to write as affine transformation?
Projection Shadows

projection of \( v \) lies along ray \( l + (v - l)t \)

\[
p = l + (v - l) \frac{(b - l) \cdot \hat{n}}{(v - l) \cdot \hat{n}}
\]

\[
p = T_lS_{(b - l) \cdot \hat{n}}MT_{-l}v
\]
Projection Shadows

Pros:
• easy to code

Cons:
Projection Shadows

Pros:
• easy to code, fast

Cons:
• only draws shadows on flat surfaces
• no soft shadows
• no self-shadows
Projection Shadows

Multiple lights?
• fake with alpha blending

Final gotcha:
Shadow Maps

Render scene **from the light**
- record depth only

why useful?
Shadow Maps

To shade a pixel, calculate: 1) pixel to light distance; 2) shadow map value
Shadow Maps

compare pixel-to-light distance to shadow map

if equal, no shadow
if greater, in shadow
What’s The Bug? Part I

Floating point == is very dangerous
• numerical floating-point error
• depth buffer has finite precision
What’s The Bug? Part I

Floating point == is very dangerous

- numerical floating-point error
- depth buffer has finite precision

Fix by including tolerance ("bias") in check
What’s The Bug? Part II
What’s The Bug? Part II

Shadow map is too coarse
• shadows are aliased
What’s The Bug? Part II

Shadow map is too coarse
• shadows are aliased

Cascaded shadow maps: use higher-resolution shadow map near eye
What’s The Bug? Part II

Shadow map is too coarse
  • shadows are aliased

Cascaded shadow maps: use higher-resolution shadow map near eye
  • highest-quality fix
  • complicated
    • (have to partition shadow map)
Percentage Closer Filtering

Sample several nearby pixels on shadow map instead of only one pixel.

Set shadow intensity proportional to number of "shadow" votes.
Shadow Maps

Very common real-time technique
• used in Unreal Engine, etc

Pros:
• works with curved objects
• works with self-shadows

Cons:
Shadow Maps

Very common real-time technique
- used in Unreal Engine, etc

Pros:
- works with curved objects
- works with self-shadows

Cons:
- very prone to aliasing artifacts
- still no soft shadows
Shadow Volumes

Consider single light, single triangle
Shadow Volumes

Main idea:

• build 3D object representing the shadow volume

If pixel is inside volume, it is in shadow
Shadow Volumes

Main idea:
• build **3D object** representing the shadow volume

If pixel is inside volume, it is in shadow
• shoot ray from pixel to eye, count number of intersections
• odd = in shadow
Shadow Volumes

For more complicated object, look only at **silhouette edges**

- edges connecting front-facing to back-facing triangles
Shadow Volumes

Increasingly popular in real-time graphics
Shadow Volumes

Increasingly popular in real-time graphics

Pros:
• high-quality shadows (no aliasing)
• self-shadows automatically included

Cons:
Shadow Volumes

Increasingly popular in real-time graphics

Pros:
• high-quality shadows (no aliasing)
• self-shadows automatically included

Cons:
• slow and complicated
• shadows chunky if meshes too coarse
• still no soft shadows