Shadow

UCSD CSE 167
Tzu-Mao Li
Shadow

whether the light is blocked or not
Easiest way to compute shadow: ray traced shadow

```python
if d < ||light.position - shadow_ray.org||:
    color = 0
else:
    color = ...
```

Q: what if the shadow ray hits the red ball itself?
Easiest way to compute shadow: ray traced shadow

```python
if d > eps and d < ||light.position - shadow_ray.org|| * (1-eps):
    color = 0
else:
    color = ...
```

Q: what if the shadow ray hits the red ball itself?

use a small epsilon to avoid self intersection
Ray traced shadow in Cyberpunk 2077

https://twitter.com/Dachsjaeger/status/1493652010985562113/photo/1
Ray traced shadow in Cyberpunk 2077
Shadow map: shadow using rasterization
[Lance Williams 1978]
Why do we want rasterization shadow while we have raytraced shadow?

- modern AAA games still use shadow mapping
- for deformable objects, ray tracing is much slower than rasterization (at least 3x-5x)
- for soft shadow, there are tricks in shadow mapping to soften the shadow ray tracing usually has to rely on denoising
- it’s a very cool algorithm anyway!!!
Shadow mapping

assuming directional light first

idea: record the closest surfaces for the light sources
Shadow mapping

1. render a depth map from the light source
Shadow mapping

1. render a depth map from the light source
2. project the shading point to the depth map
   if distance_to_light > depth:
     // in shadow
   else:
     // not in shadow
Shadow mapping

1. render a depth map from the light source
2. project the shading point to the depth map
   if distance_to_light > depth:
     // in shadow
   else:
     // not in shadow
Rendering depth map in shadow mapping

usually use an orthogonal projection since the light rays are parallel

area we can see

image plane
Rendering depth map in shadow mapping

usually use an orthogonal projection since the light rays are parallel

build two matrices: lightView and lightProjection
  analogous to the view matrix  analogous to the projection matrix
Rendering depth map in shadow mapping

usually use an orthogonal projection since the light rays are parallel

build two matrices: \texttt{lightView} and \texttt{lightProjection}

given a vertex \( p \) on an object with a model matrix \texttt{model}, we do
\[
\texttt{lightProjection} \times \texttt{lightView} \times \texttt{model} \times p
\]
in the vertex shader
interpolate depth in fragment shader
Rendering depth map in shadow mapping

usually use an orthogonal projection since the light rays are parallel

build two matrices: lightView and lightProjection

given a vertex \( p \) on an object with a model matrix \( \text{model} \), we do

\[
\text{lightProjection} \times \text{lightView} \times \text{model} \times p
\]

in the vertex shader

to interpolate depth in the fragment shader

store depth in a texture
Shadow mapping

1. render a depth map from the light source
2. project the shading point to the depth map
   if distance_to_light > depth:
     // in shadow
   else:
     // not in shadow
Shadow mapping

given a shading point $p_s$ in world space, we do the projection to light space using

$$p_{\text{light}} = \text{lightProjection} \ast \text{lightView} \ast p_s$$
given a shading point $p_s$ in world space, we do the projection to light space using

$$\mathbf{p}_{\text{light}} = \mathbf{lightProjection} \ast \mathbf{lightView} \ast p_s$$

look up the depth map texture using $\mathbf{p}_{\text{light}.xy}$

$$\text{depth} = \text{texture(shadow_map, } p_{\text{light}.xy)}$$
Shadow mapping

given a shading point \( p_s \) in world space, we do the projection to light space using

\[
p_{\text{light}} = \text{lightProjection} \times \text{lightView} \times p_s
\]

look up the depth map texture using \( p_{\text{light}}.xy \)

\[
\text{depth} = \text{texture(shadow\_map, p_{\text{light}}.xy)}
\]

\[
\text{distance\_to\_light} = \| p_s - \text{light\_pos} \|
\]
Shadow mapping

1. render a depth map from the light source
2. project the shading point to the depth map
   if distance_to_light > depth:
     // in shadow
   else:
     // not in shadow
Shadow map for point lights

render 6 depth maps and form a cube
(using perspective projection)

or

render two “paraboloid” depth maps

Shadow map artifacts

- aliasing
- shadow acne

Shadow map artifacts

- aliasing
- shadow acne

Peter Panning

Shadow acne

caused by limited depth resolution & floating point precision
similar to Z-fighting
Solution to shadow acne:

shadow map bias

1. render a depth map from the light source
2. project the shading point to the depth map
   if distance_to_light - eps > depth:
     // in shadow
   else:
     // not in shadow

this is called the “shadow map bias” (or depth offset)

Shadow map artifacts

- aliasing
- shadow acne

Peter panning

happens when the shadow map bias is too large
Adaptive shadow map bias: slope-scaled depth

\[ \text{eps} = \max(0.05 \times (1 - \text{dot(normal, lightDir)}), 0.005) \]

larger shadow map bias when the surface normal is perpendicular to light
small bias when normal is parallel to light

surface with high slope can have worse depth approximation on the depth map

Shadow map artifacts

- aliasing
- shadow acne

Shadow map aliasing

caused by limited depth resolution

solutions
1. increase shadow map resolution
2. filtering
Shadow map aliasing
caused by limited depth resolution

solutions
1. increase shadow map resolution
2. filtering
Cascaded Shadow Map

idea: render different shadow maps for different distance w.r.t. camera

for shadows closer to the camera, use higher resolution depth map
for shadows farther from the camera, use low resolution depth map
Cascaded Shadow Map

https://forum.babylonjs.com/t/cascaded-shadow-maps-csm-are-now-in/7970
Cascaded Shadow Map

CSM in Counter-Strike: Global Offense

https://learnopengl.com/Guest-Articles/2021/CSM
Clipmaps in Unreal Engine 5

similar to Cascaded Shadow Map: store different shadow maps for different regions in the scene

Modern game engines use crazy resolutions for shadow mapping

Unreal Engine 5 uses 16k x 16k shadow maps (one for each cubemap/clipmap!!)
Shadow map aliasing

causled by limited depth resolution

solutions
1. increase shadow map resolution
2. filtering
Standard texture filtering:
interpolation of depth values

depth = texture(shadow_map, p_light.xy)
Percentage closer filtering

idea: filter the visibility function, not the depth

1987

Rendering Antialiased Shadows with Depth Maps

William T. Reeves
David H. Salesin
Robert L. Cook
Without percentage closer filtering

from Christoph Peters’ fantastic slides: http://momentsingraphics.de/I3D2015.html
Without percentage closer filtering

from Christoph Peters’ fantastic slides: http://momentsingraphics.de/I3D2015.html
Percentage closer filter
filters visibility instead of depth

from Christoph Peters’ fantastic slides: http://momentsingraphics.de/I3D2015.html
With percentage closer filter

from Christoph Peters’ fantastic slides: http://momentsingraphics.de/I3D2015.html
Percentage closer filter can be used for simulating soft shadows

adjust the filter size based on the distance to the blocker

\[
W_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot W_{Light}}{d_{Blocker}}
\]

- Assumes that blocker, receiver, and light are parallel
Shadow mapping is messy

parameters for bias/filtering are usually highly scene dependent
Alternative: Shadow Volume

[Frank Crow 1977]

occluder

everything inside this polygonal volume is inside shadow
Alternative: Shadow Volume
[Frank Crow 1977]

shoot a ray into the shadow volume

occluder
Alternative: Shadow Volume
[Frank Crow 1977]

shoot a ray into the shadow volume
+1 or -1 when we intersect the ray with the volume polygon
Alternative: Shadow Volume

[Frank Crow 1977]

shoot a ray into the shadow volume

+1 or -1 when we intersect the ray with the volume polygon

count != 0 -> in shadow
Alternative: Shadow Volume

[Frank Crow 1977]

shoot a ray into the shadow volume

+1 or -1 when we intersect the ray with the volume polygon

count != 0 -> in shadow

can be implemented using “stencil buffers” in a rasterizer
Shadow volumes in a rasterizer

draw front-facing shadow polygons
  if Z pass -> +1 to stencil buffer

draw back-facing shadow polygons
  if Z pass -> -1 to stencil buffer

draw pixels with 0 in stencil buffer
Shadow volume in Doom 3 [2004]

super high quality shadow!!

Alternative: Distance field shadow

- use the distance to occluder to blur the shadow
- higher quality shadow and good antialiasing
- doesn’t quite work for deformable objects

https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/RayTracedDistanceFieldShadowing/
Next: antialiasing

https://blog.codinghorror.com/fast-approximate-anti-aliasing-fxaa/