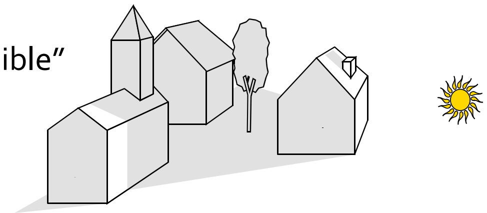
EECS 487: Interactive Computer Graphics

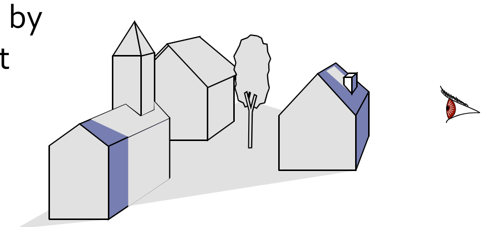
Lecture 32: Interactive Visual Effects
- Shadow Map
- Ambient Occlusion

Shadow Mapping

A point is lit if it is "visible" from the light source
- similar to visible surface determination



Shadow computation by simulating eye at light position

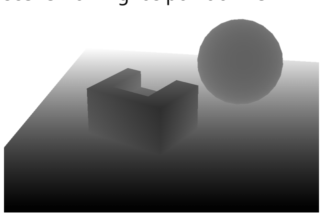


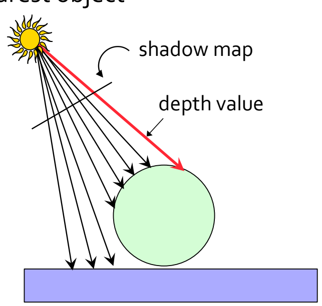
Durand

Shadow Mapping

Requires 2 passes through the pipeline
First pass: compute shadow map
(depth of closest pixels to the light)
- render scene from light
- populate z-buffer that we'll use as our shadow map
  - store the distance from light to nearest object
  - white is far, black is near

scene from light's point of view

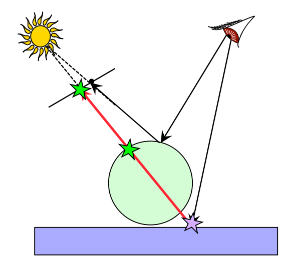




Foley et al., Akenine-Möller02,Durand

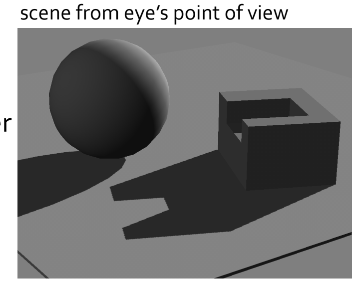
Shadow Mapping

Second pass: shadow determination using the shadow map



For each pixel
- do normal z-buffer computation to check visibility from eye
- if visible, look up distance to light
  - perspective project visible pixels from eye space back to light space
  - lookup depth ★ stored in shadow map
- if distance to light ☆ is (epsilon) greater than stored depth, pixel is in shadow

scene from eye's point of view



Foley et al., Akenine-Möller02,Durand

Shadow Mapping with OpenGL

Create the shadow map:

- render the scene with eye at light position
- save the resulting z-buffer and projection matrix, these are the shadow map and shadow projection matrix
  - `glReadPixels(…)` to read back z-buffer
  - `glGetDoublev(GL_MODELVIEW_MATRIX, …)`, `glGetDoublev(GL_PROJECTION_MATRIX, …)`, `glGetIntergerv(GL_VIEWPORT, …)`

Render scene:

- render the scene from the actual viewpoint
- save both color and z-buffers and viewpoint projection matrix

Yu

Shadow Mapping with OpenGL

Determine eye-space, light-space correspondences:

- unproject every pixel in the z-buffer to obtain object coordinates, using `gluUnproject(…)` and the viewpoint projection matrix
- project the object coordinates into the shadow map using `gluProject(…)` and the shadow projection matrix

Shadow rendering:

- compare the resulting z values with the corresponding content of the shadow map and update the color buffer to draw the projected shadows
- write back the modified color buffer using `glDrawPixels(…)`

Yu

Limitations of Shadow Maps

1. Limited field of view, no omni-directional light
2. Limited depth resolution
3. Shadow map aliasing
4. Hard shadows only (with jaggies) (or soft shadows only if PCF is used)

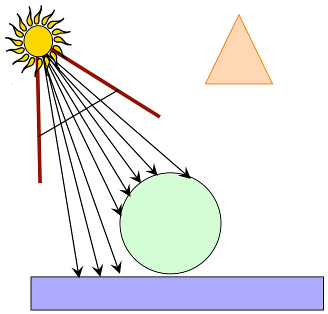
Durand

Limited Field of View

What if a point/object is outside field of view of the shadow map?

Use six shadow maps, to form a cube enclosing the light

- requires a separate rendering pass for each shadow map!

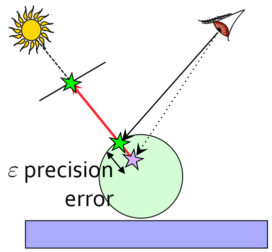


Durand

Limited Depth Resolution

Due to z-fighting, distance to light and stored depth may not compare correctly

- add an ε to depth in shadow map to prevent unintended (self-)shadowing
  - in computing shadow map move geometry away from light by a small amount
  - choosing correct ε value is tricky

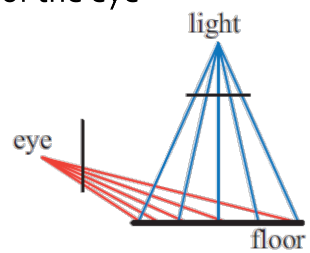


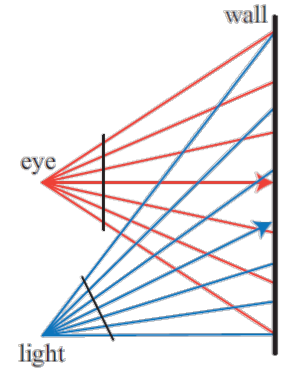
Precision error also possible with projection shadows

Merrell

Shadow Map Aliasing

Results from sampling rates mismatch (re-sampling problem): sampling rate (pixel size) of light is mismatched to that of the eye





Kilgard

Shadow Map Aliasing

Least aliasing when light frustum is reasonably well aligned with the eye's view frustum: the ratio of sample sizes is close to 1

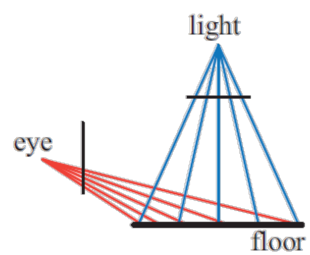
- best case if eye and light frusta are nearly identical ("miner's lamp" case)
  - but only limited scene setups satisfy this
- worst case is when light is shining at the viewer ("deer-in-the-headlights" case)
  - also known as the "dueling frusta" problem

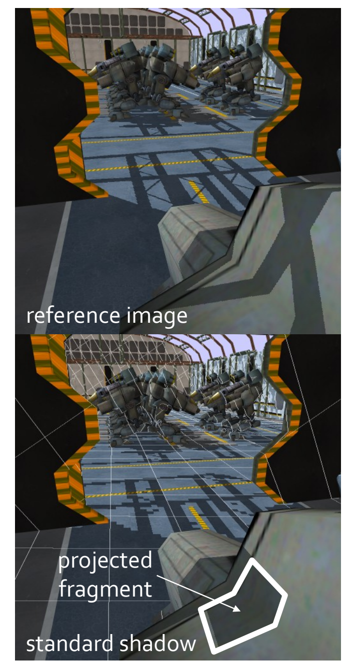
Kilgard

Shadow Map Aliasing

Shadow "fragment" is also stretched when eye is close to the surface but light is far away

- surfaces that are nearly edge-on to the eye face the light directly
- results in under-sampling of near field in shadow map



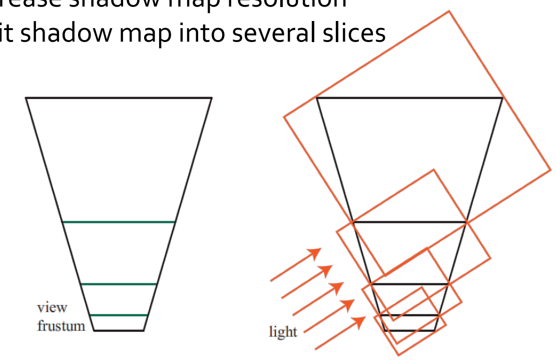


Durand,Merrell,RTR3

Shadow Map Anti-Aliasing

Possible solutions:

1. increase shadow map resolution
2. split shadow map into several slices

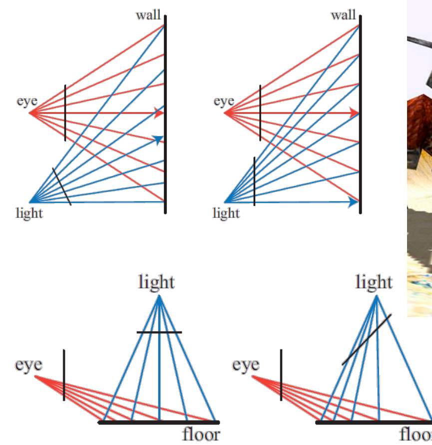


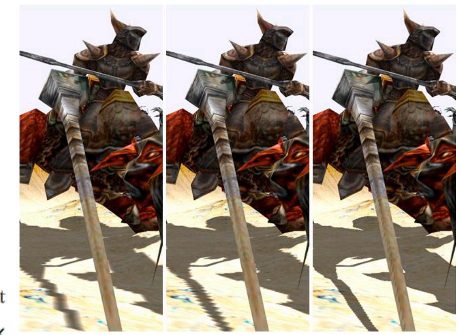
3. use asymmetric frustrum for shadow map rendering
4. average nearby pixels

Durand,RTR3

Shadow Map Anti-Aliasing

Asymmetric frustrum for shadow map rendering

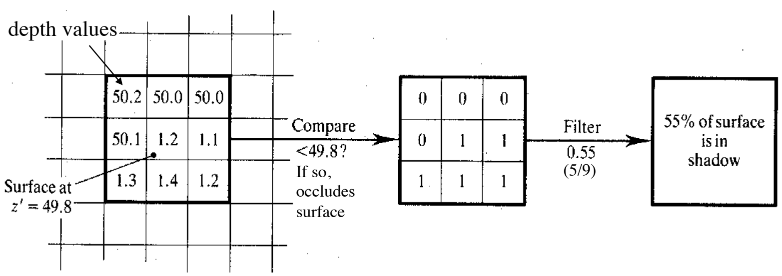




different projections, same resolution

RTR3,Martin&Tan

Percentage-Closer Filtering



Results in aliased shadow
Anti-aliasing by averaging several nearby shadow map fragments
What to average? Depth values?

- No, 1.2 < 49.8, but so is 22.9: still a binary result, no anti-aliased blurring

Instead, perform depth test for a neighborhood of pixels

- then compute percentage of lit pixels
- (this makes ε computation even trickier!)

Durand,Schulze

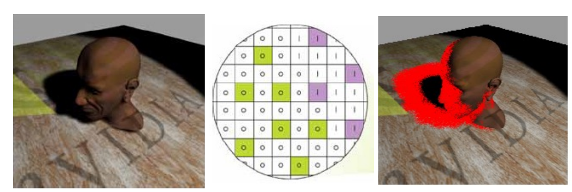
Soft-Shadows with Shadow Map

Look up several nearby shadow map fragments, not just one

Compute average shadow value for the neighborhood

- use immediate neighbors for anti-aliasing
- use neighbors further afield for soft-shadows

Sampling strategy: grid, jittered, or adaptive for improved performance



Percentage-Closer Filtering

Supported in hardware for small (2×2) filters

- shadow map coordinates generated using projection matrix
- shadow map stored as texture: modern hardware permits tests on texture values
- can use larger filters with additional rendering passes

Durand,Schulze

Ambient Occlusion

All the real-time shadow algorithms assume directional lighting only

Ambient occlusion is shadows due to global indirect/ambient lighting

Can be pre-computed using global illumination and baked into a texture
→ limited to static scenes

Yang et al., TP3

Ambient Occlusion

Two approaches:

- Screen-space Ambient Occlusion (SSAO)
- Screen-space Directional Occlusion (SSDO)

Basic idea:

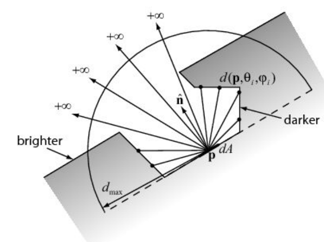
- approximate indirect lighting using a uniform, distant environment irradiance
- simulate the darkening effect where ambient light is blocked by other geometry in the scene
  - compute the portion of the hemisphere around a point that is blocked
  - purely geometric, independent of lighting conditions or viewing direction

TP3

SSAO

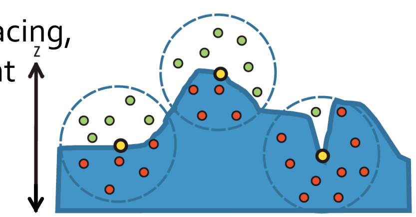
At each point find fraction of hemisphere that is occluded

- visible fraction: 1-occlusion
- modulate diffuse shading by $(1-\text{occlusion})$
  - $\mathbf{c}_d = m_d s_d \max((\mathbf{n}\cdot\mathbf{l}), 0)\ (1-\text{occlusion})$



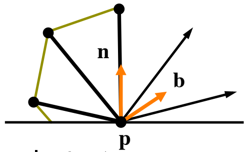
Alternately, to avoid ray tracing, sample a circle around point

- compare depth of samples against z-buffer content
- sample is not occluded if depth $<$ z-buffer content



Bala, Yang, Yip, X

SSDO

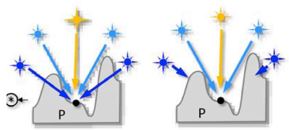


SSAO plus EM and indirect lighting:

- instead of modulating diffuse shading by $(1 - \text{occlusion})$, add up light contribution from unoccluded directions only
- compute average un-occluded direction ("bent normal", $\mathbf{b}$)
- lookup light/environment map using the "bent normal" ⇒ result in light-colored shadow instead of just grey
- (can add a single bounce off facing occluding surfaces)

Alternately, to avoid ray tracing,

- sample uniformly across the hemisphere above point, this gives N light vectors
- sample each light vector at random offsets from the point
- compare depth of samples against z-buffer content
- sample is not occluded if depth $< z$-buffer content



Ritschel,Grosch,Seidel09

Shadow and Environment Maps

Basic method to add realism to interactive rendering: instead of ray tracing, use image-based methods

- Shadow maps: image-based hard shadows [Williams78]
  - many recent extensions
  - widely used even in software rendering (RenderMan)
- Environment maps: image-based complex lighting [Blinn&Newell76]
  - huge amount of recent work

Together, give many "realistic" effects

- but cannot be easily combined!

Ramamoorthi