Shadow Mapping

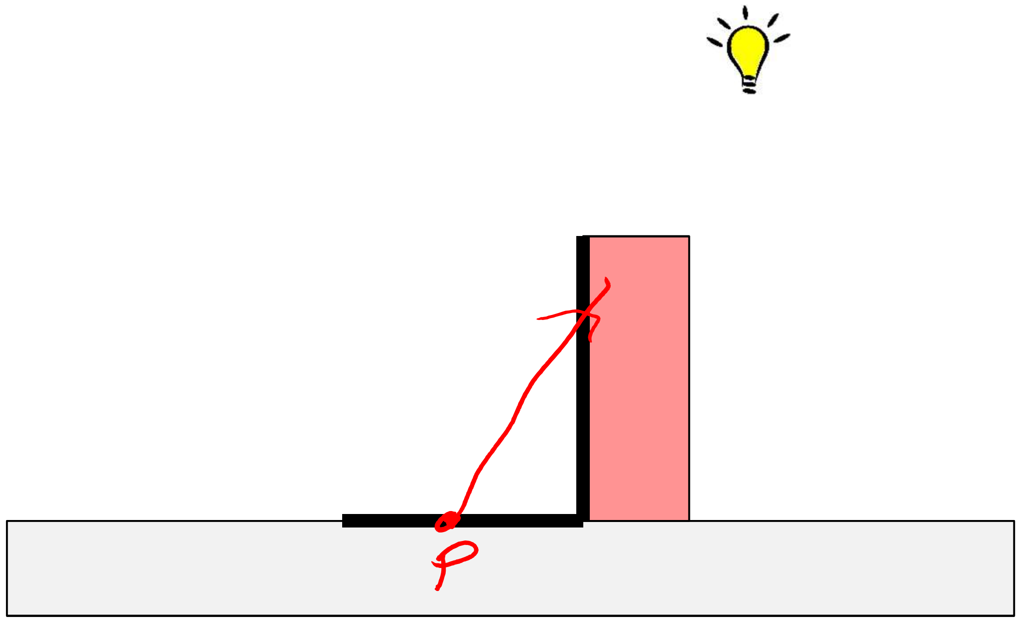
Edward Zhang & Brian Curless

CSE 557

Autumn 2017

Shadows

What does it mean for a point to be in shadow, relative to some light?



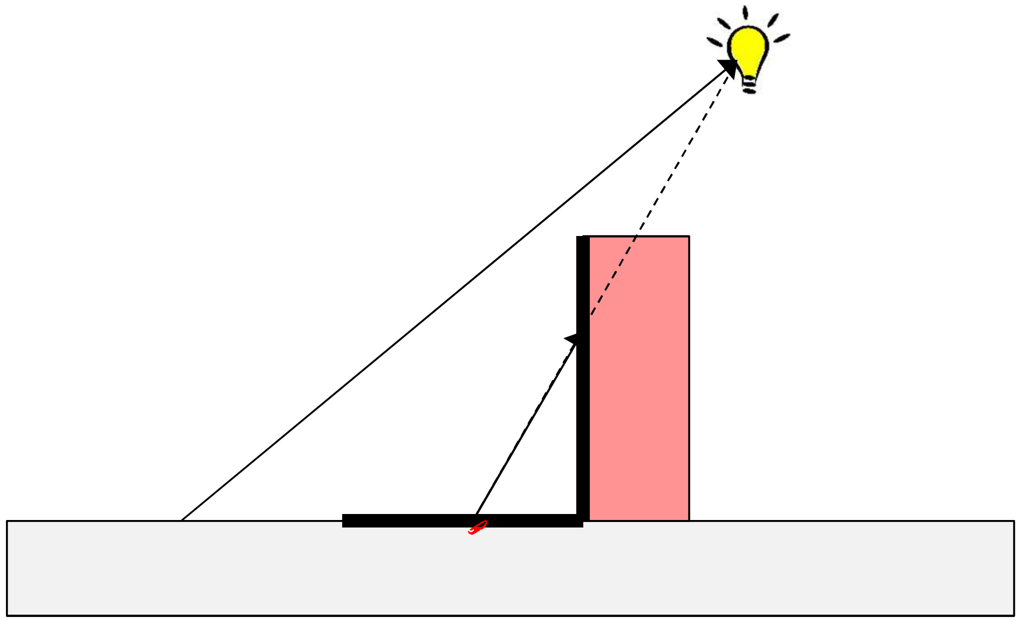
Shadows

What does it mean for a point to be in shadow, relative to some light?

Assuming point lights,

A point is in shadow if the ray from the point to the light (along L), intersects the scene.

Why is this hard to do in a shader?



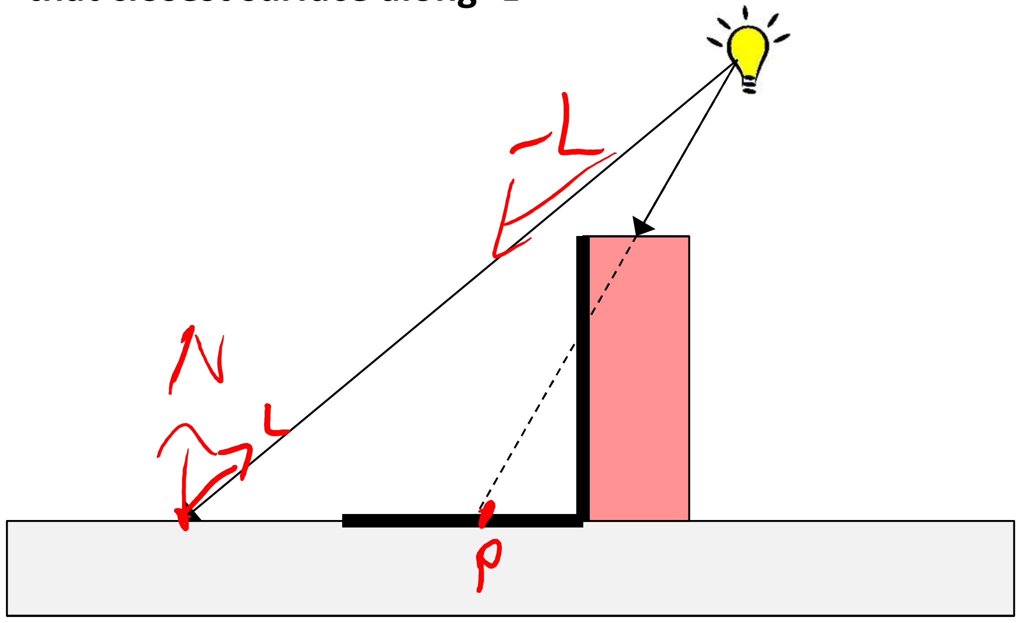
Shadows

What does it mean for a point to be in shadow, relative to some light?

Alternatively…

A point is unshadowed if it is the closest surface to the light along -**L**, so…

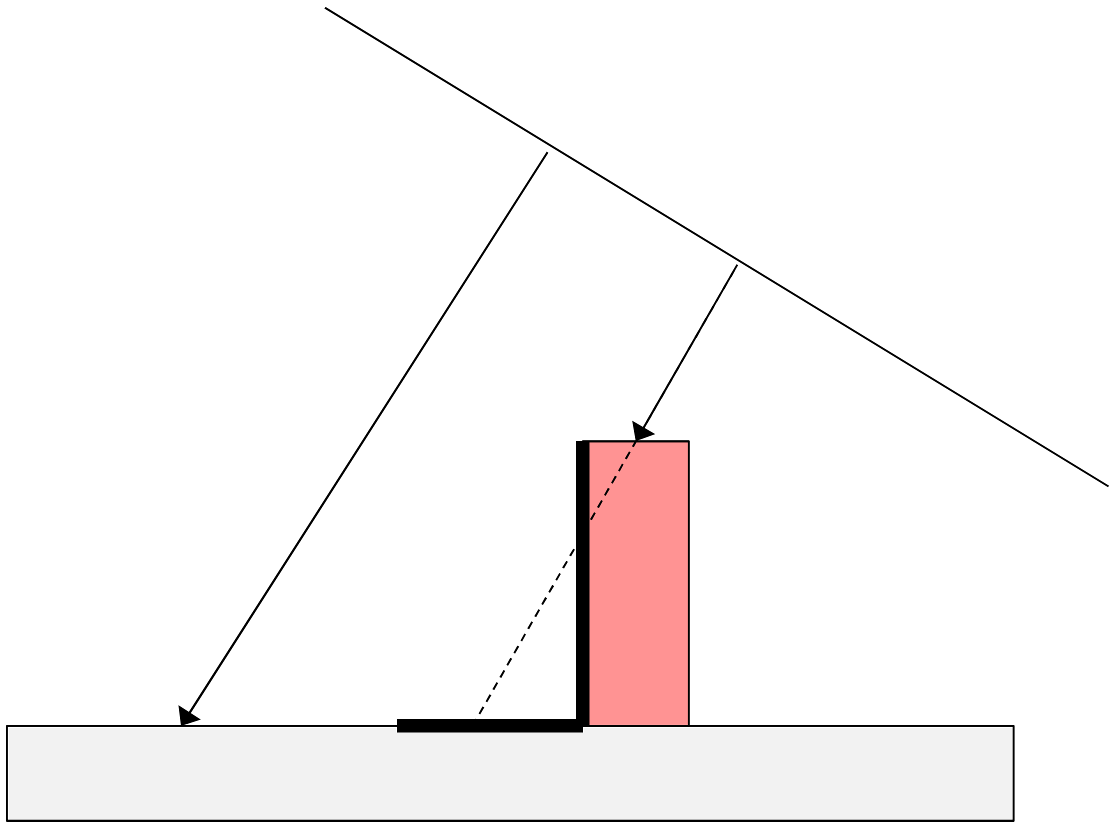
A point is in shadow if it is farther from the light than that closest surface along -L



Shadows

A point is in shadow if it is farther from the light than the closest surface along the ray

For directional lights, pick a plane perpendicular to L, but outside the scene bounds, as your source



Parameterizing shadow rays

A ray is parameterized by $\boldsymbol{P} = \boldsymbol{O} + \boldsymbol{v}t$ 9

$O=(0,0,0)$

$V=(1,0,0)$

$V=(2,0,0)$

How many degrees of freedom does this have in 3D (i.e. how many numbers uniquely determine a ray)?

3 for position

5

2 for direction

How many degrees of freedom are there for a shadow ray from a point light?

2 for direction

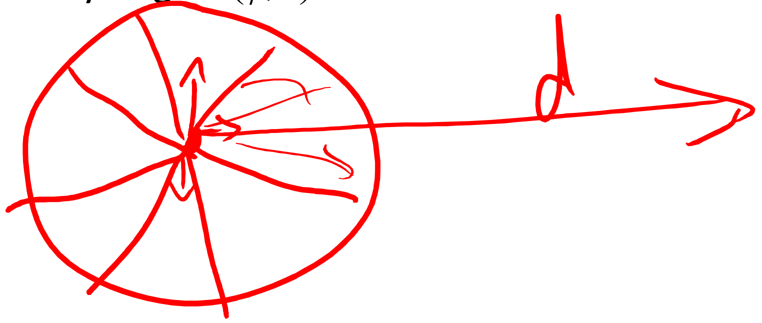
How many degrees of freedom are there for a shadow ray from a directional light?

2 for position

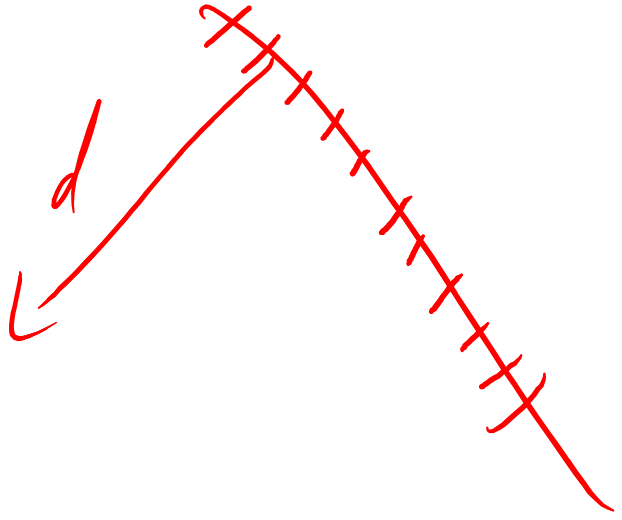
Precomputing shadow rays

A shadow map is a 2D lookup table containing the distance to the closest scene point along each possible shadow ray

Point Lights: Discretize by angles (φ,θ)



Directional Lights: Discretize by coordinates (x,y) of point on perpendicular plane



Shading with Shadows

Each shader gets one lookup table (shadow map) for each shadow-casting light in the scene.

When shading point $\mathbf{P}$, for each light source:

1. Compute light direction $\boldsymbol{L}$ and shadow ray $\boldsymbol{R}(t) = \boldsymbol{O} - \boldsymbol{L}t$
2. Compute distance t from $\mathbf{P}$ to light source
3. Look up ray $\boldsymbol{R}$ in the shadow map to get distance d
   - If $t > d$ point is shadowed
   - Otherwise, point is unshadowed

If $\mathbf{P}$ is unshadowed, $A_j^{\text{shadow}} = 1$

If $\mathbf{P}$ is shadowed, $A_j^{\text{shadow}} = 0$

$$I_{\text{direct}} = k_e + \sum k_d I_{La,j} + A_j^{\text{shadow}} A_j^{\text{dist}} I_{L,j} B_j \left(k_d(\mathbf{N}\cdot\mathbf{L}_j) + k_s(\mathbf{N}\cdot\mathbf{H}_j)_+^{n_s} \right)$$

Computing Shadow Maps

How do we compute shadow maps?

For each ray in our lookup table, raycast into the scene and find the nearest surface

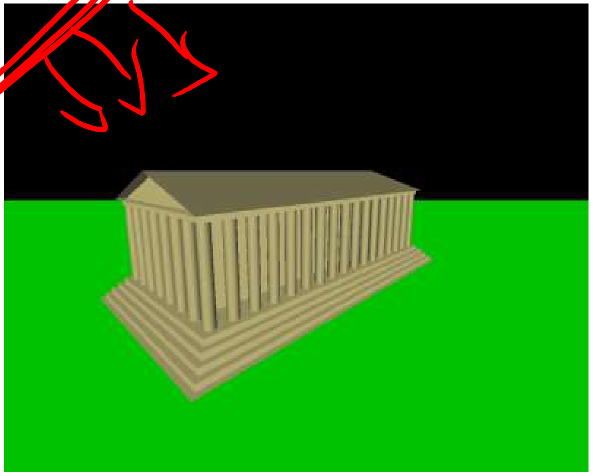
But raycasting is slow!

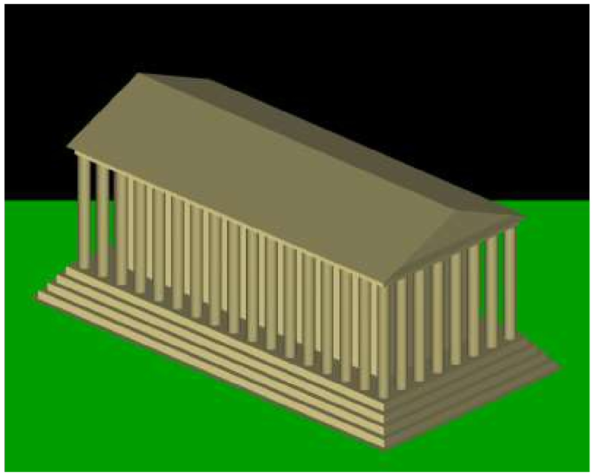
For each triangle, find which rays in the lookup table intersect the triangle (and only update the lookup table entry if the triangle is closer to the light than the current value)

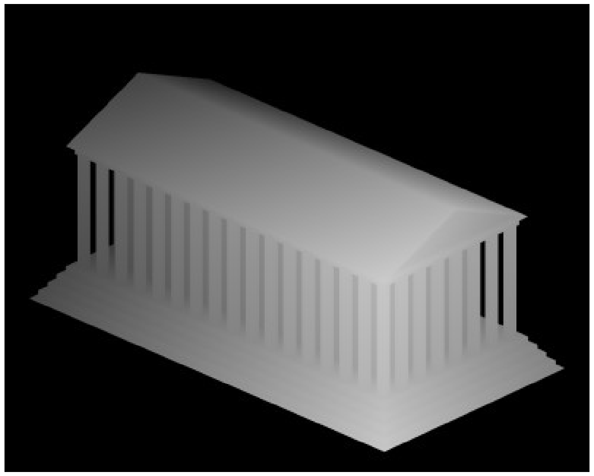
This sounds familiar…

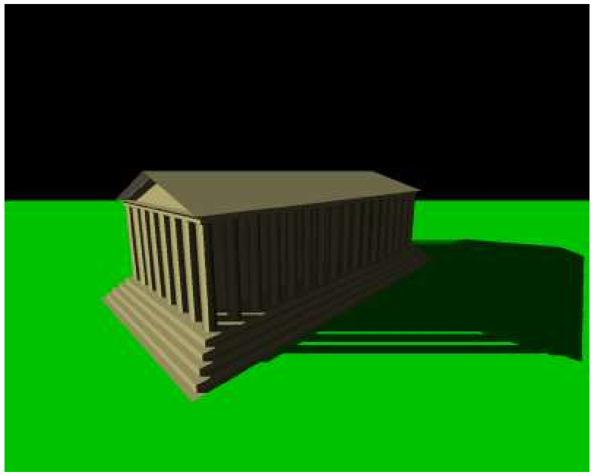
Computing Shadow Maps

To compute shadow maps, render the scene from the point of view of the light.


Unshadowed scene with one directional light

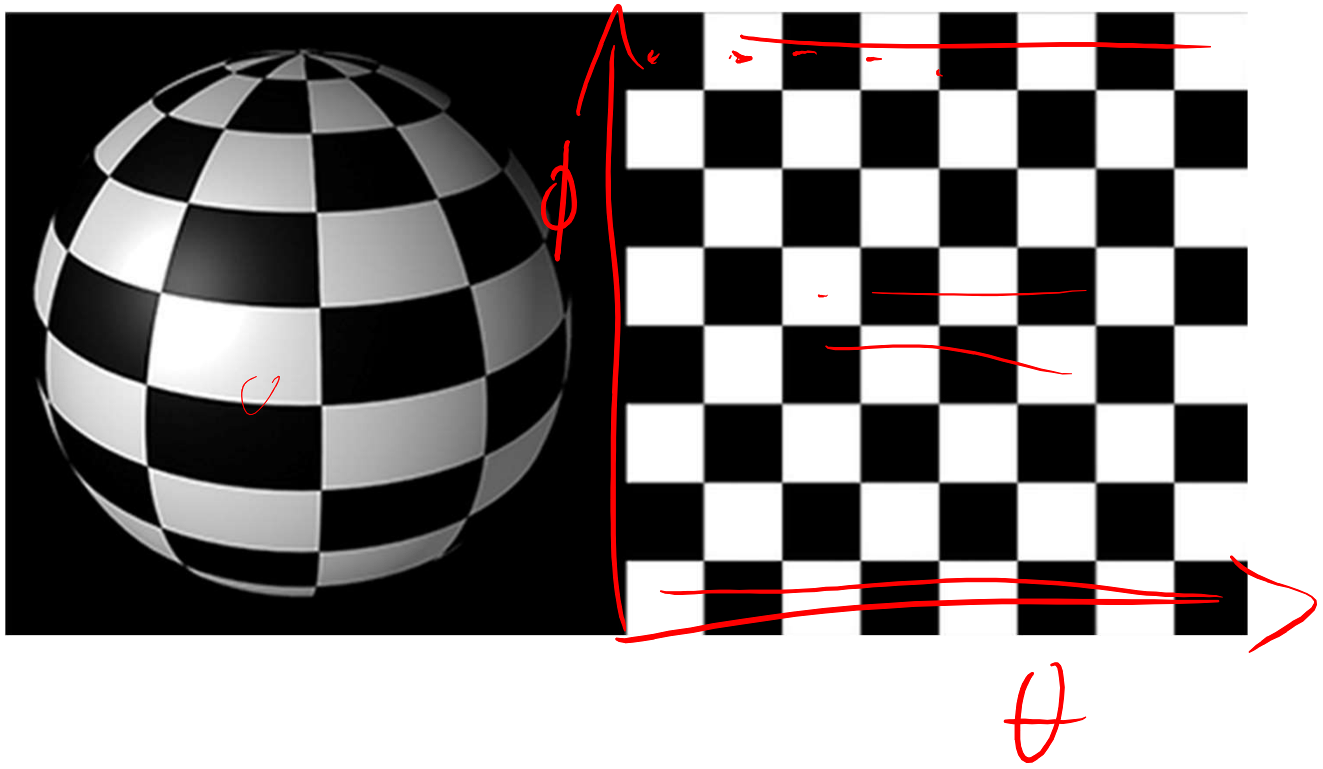

Scene rendered onto directional light plane


Depth map of scene from directional light plane


Shadowed Scene

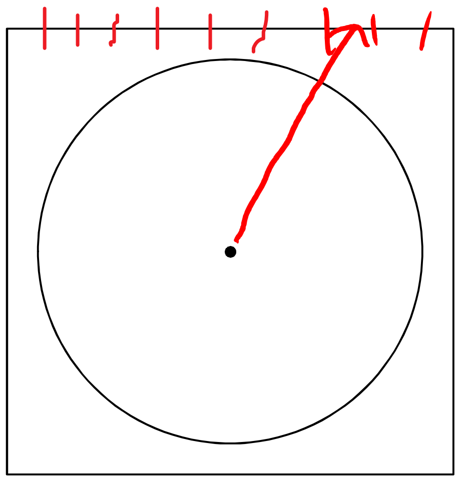
Shadow Map Format

For point lights, our shadow map needs to cover the sphere of directions. A direct mapping of (φ,θ) is not ideal since there is a singularity at the poles, and not enough resolution at the equator.



Computing Shadow Maps

Instead, we use cubemaps: six square textures mapped to the faces of a cube



To look up a direction in a cubemap, you figure out which face the ray intersects, and then look up the appropriate texel in that face texture.

OpenGL does this for you – all you have to do is give it a ray direction!

Computing Shadow Maps

Before rendering the scene to the framebuffer,

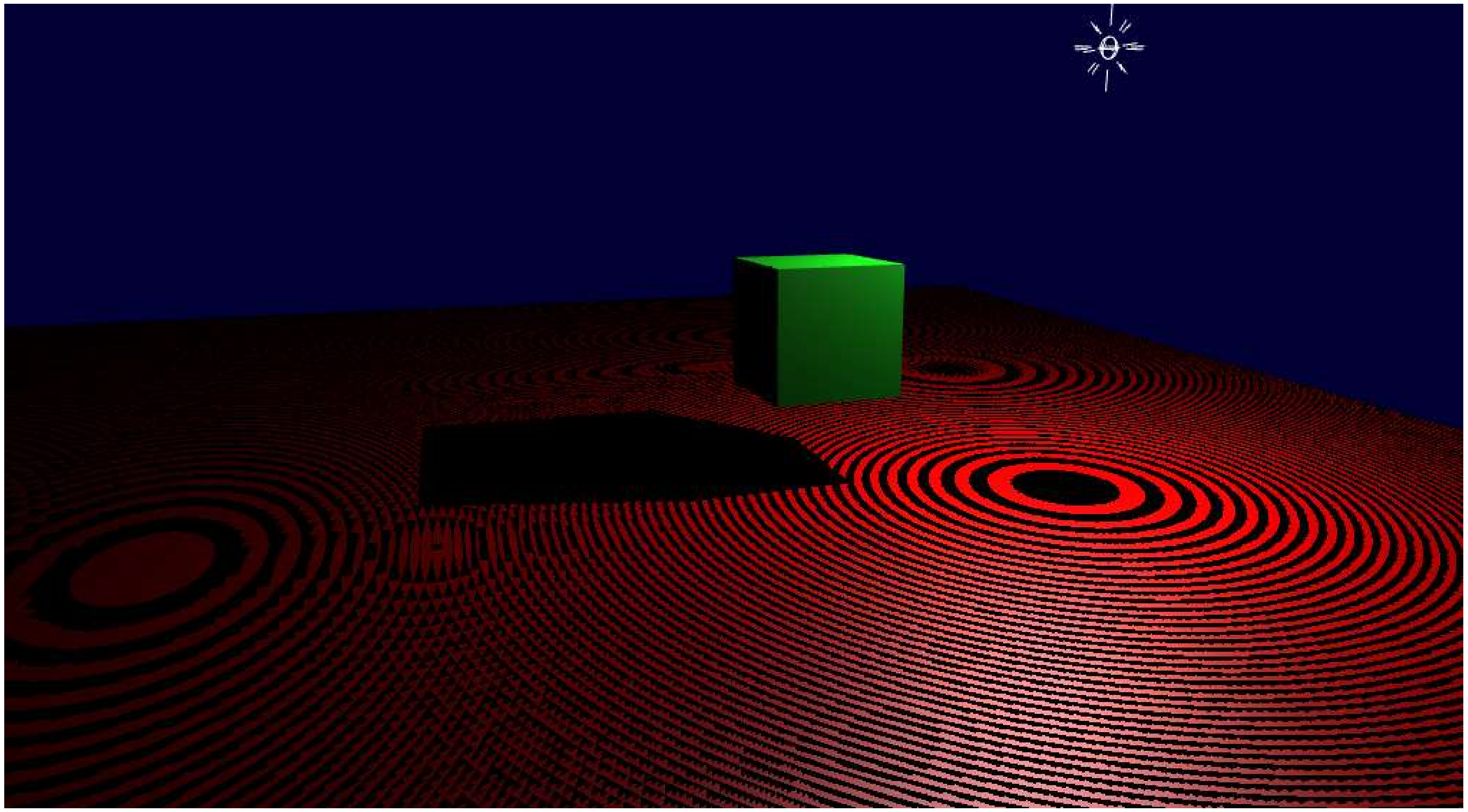
For each light:

1. Get a texture to hold the shadow map
2. Bind the texture as the framebuffer
3. Compute appropriate projection and view matrices for the shadow map (x6 for cubemaps)
4. Render the scene as a depth map (x6 for cubemaps)

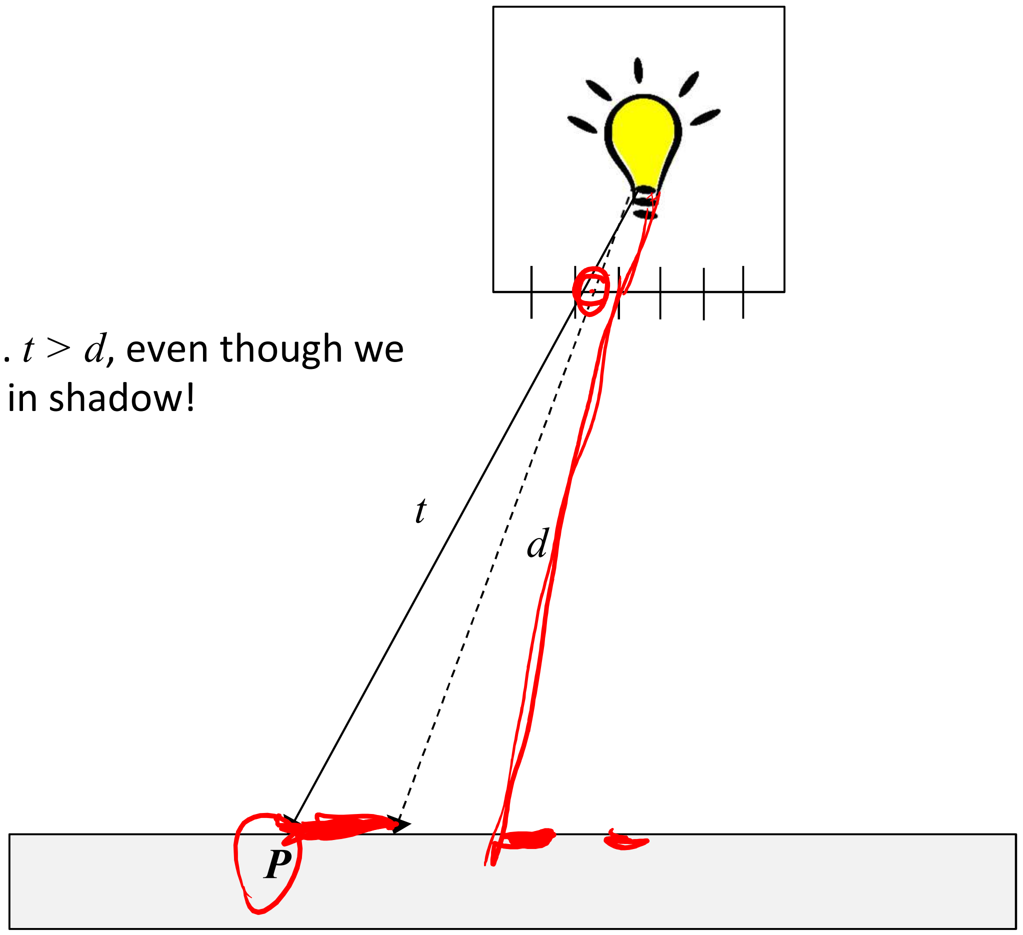
In Animator/Modeler, steps 1-3 are done for you (for point lights).

Your job is to write a shader that will output the distance to (rather than the shaded appearance of) the fragment being shaded.

Artifacts – “Shadow Acne”



Artifacts – "Shadow Acne"



Oops… $t > d$, even though we aren't in shadow!

Fix:

```
~~if (t > d) A_shadow = 0;~~
if (t > d + shadow_bias) A_shadow = 0;
```

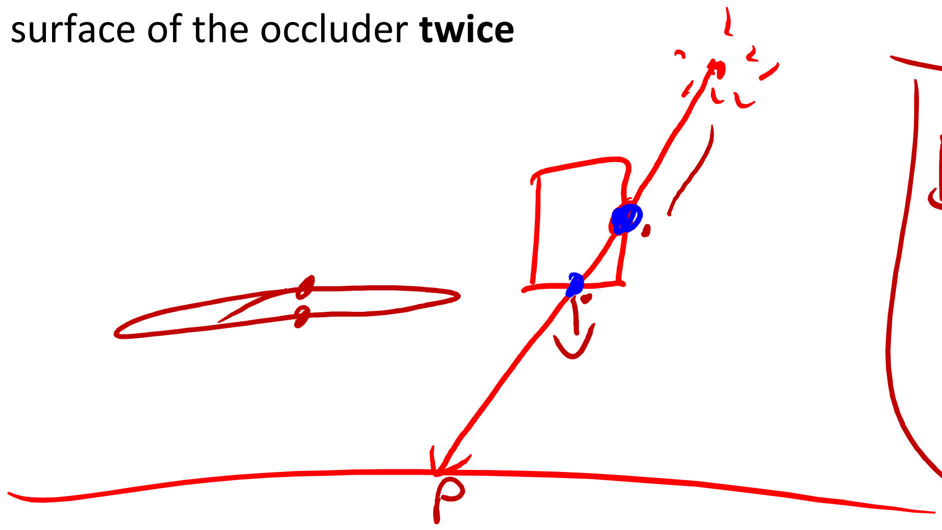
Artifacts – "Peter-Panning"

optimisticmitch|tumblr

Artifacts – “Peter-Panning”

Peter Panning happens when the shadow bias is too big.

Observation: An occluded shadow ray passes through the surface of the occluder **twice**



$B = N \cdot L$

$B < 0$ don't shade

Fix: Render shadowmaps using backfaces, not frontfaces. No shadow bias necessary!

But requires watertight geometry of some minimum thickness, and no intersecting objects