MVD: Advanced Graphics 1

12 - LightCasters

alunthomas.evans@salle.url.edu

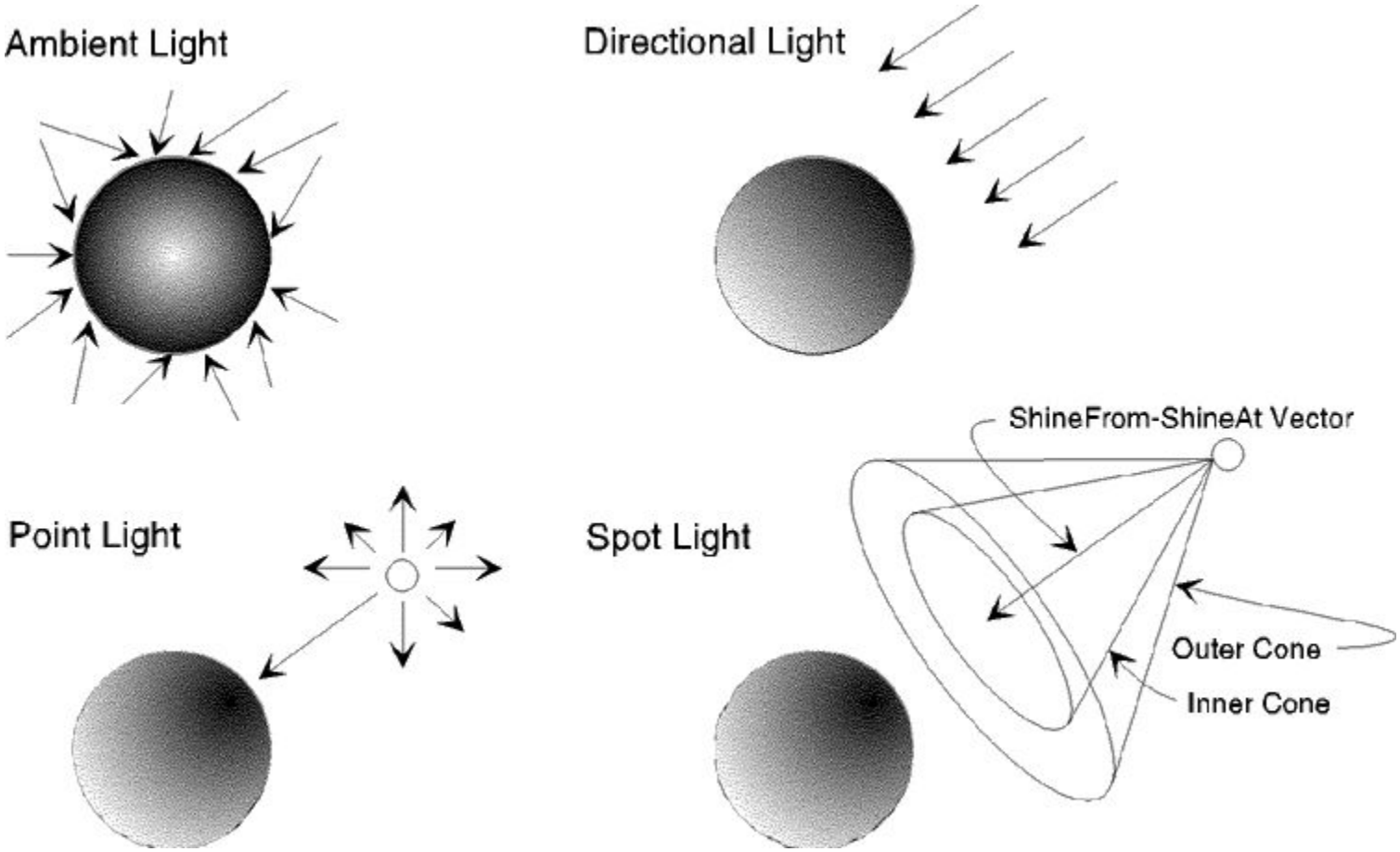
laSalle ENG
Universitat Ramon Llull

There are 4 basic types of light used in game engines

Can you think of them?

What properties do we need to store for each one?

There are 4 basic types of light used in game engines



Properties for each light

Ambient light: *color*

Point light: *color, position*

Directional light: *color, direction*

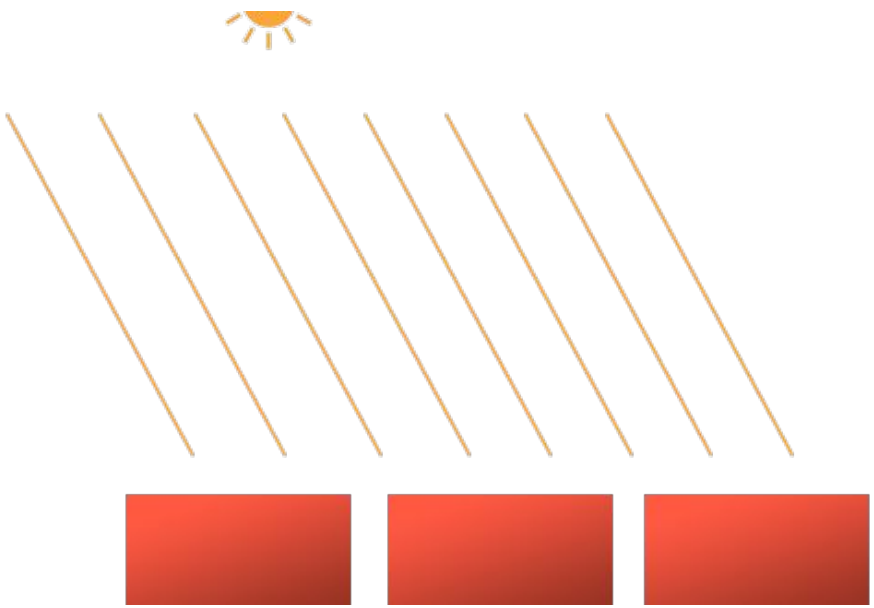
Spot light: *color, direction, spread (inner/outer)*

Directional Light

Easiest kind of light to implement. Like the sun.

No position - pass direction directly to shader

in shader: vec3 L = - light_direction



Task:

1. Uncomment light 1

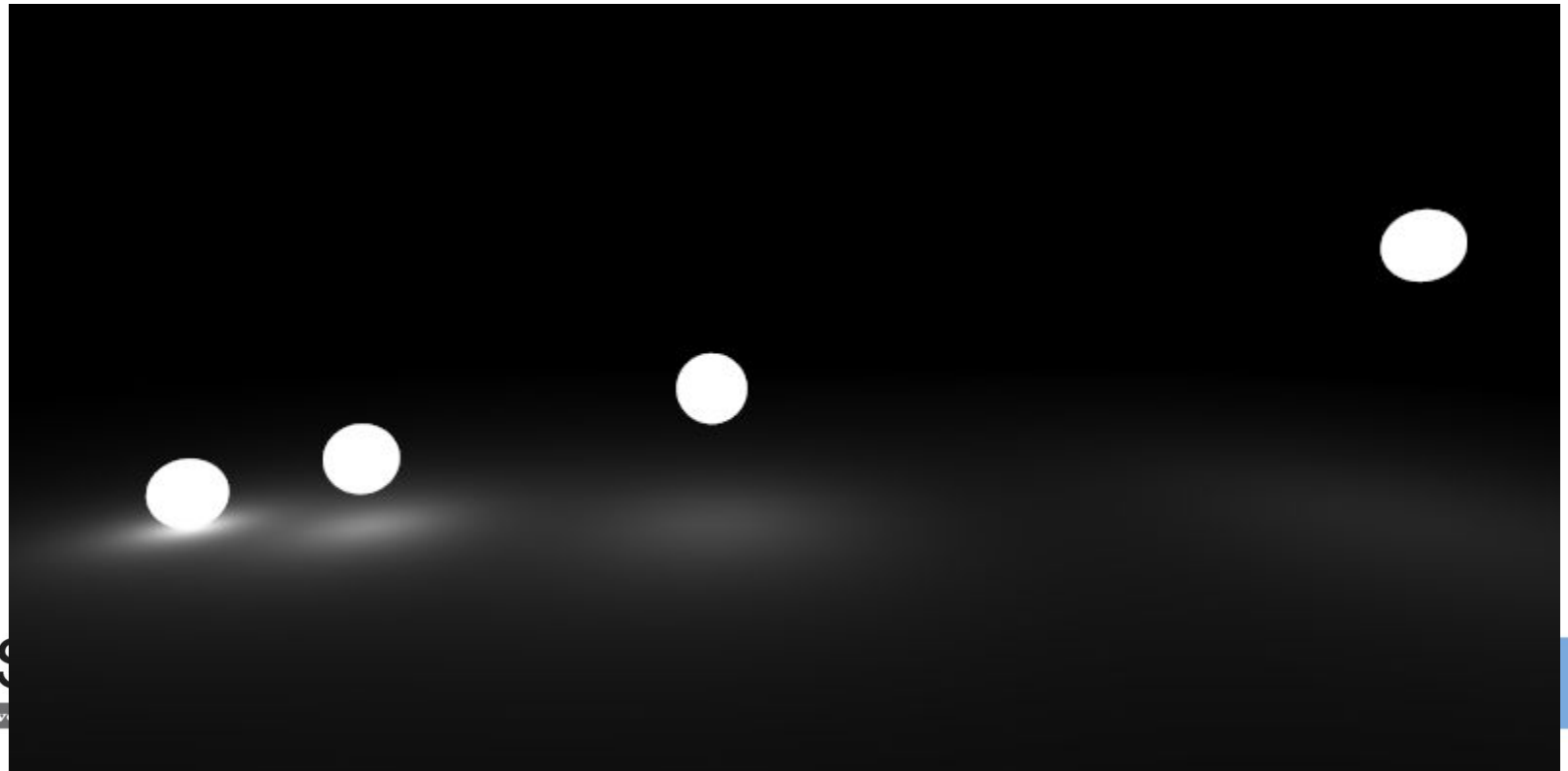
2. Modify phong shader:
   a. add ‘type’ (int) and direction (vec3) to Light struct
   b. calculate L = - direction

3. Modify setMaterialUniforms to send light type and direction

Point light

We already had implemented a simple point light.

but in the real-world, one of the key factor of points lights in attenuation



Light attenuation

Light loses energy the further you are from the source

Physically this is mapped as:

$$L_{distance} = \frac{L_1}{distance^2}$$

L_1 = strength of light at distance = 1

Unfortunately, simulating this in graphics looks bad, because as distance approaches 0, light reaches **infinity**!

Simulating light attenuation

Thus we can mix linear and quadratic attenuation with the following formula

$$F_{att} = \frac{1.0}{K_c + K_l * d + K_q * d^2}$$

It features three constants:

K_c = always kept at one to make sure $F_{att} > 1$

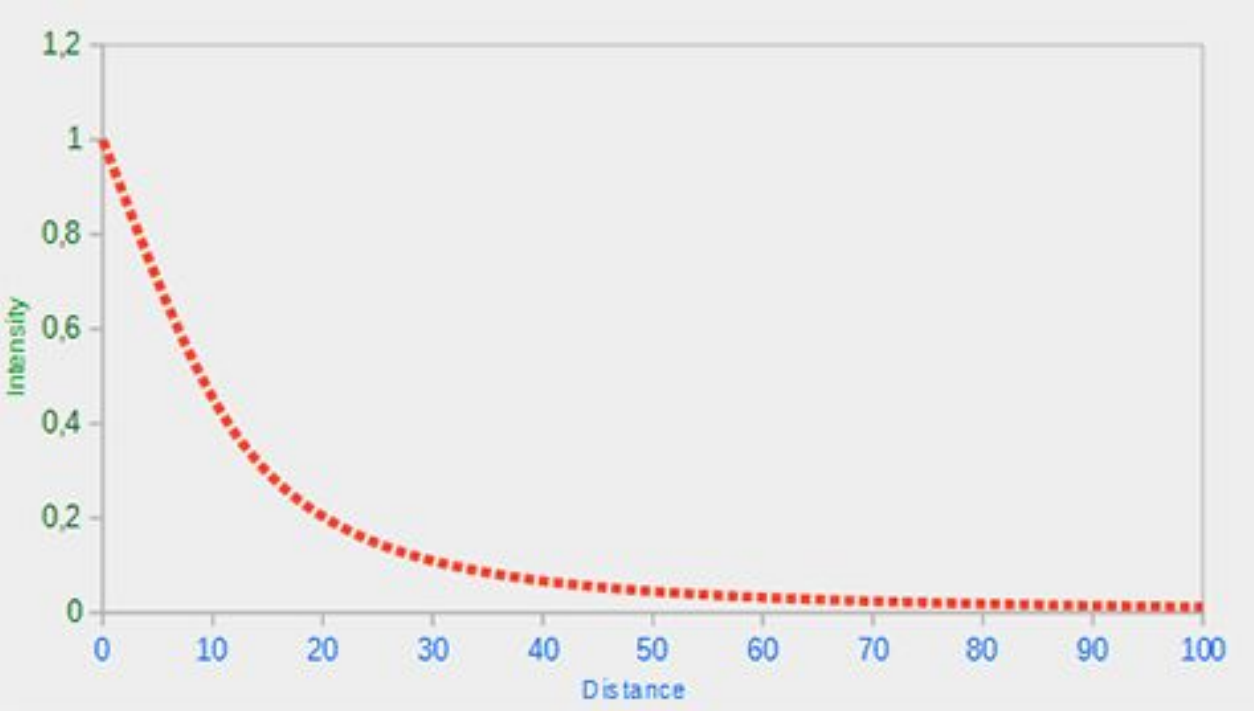
K_l = linear distance

K_q = quadratic distance

K_l is dominant when close, K_q is dominant further away

Graphical result

$$F_{att} = \frac{1.0}{K_c + K_l * d + K_q * d^2}$$



Attenuation constant values

The problem now is that we have to provide values for these constants.

The Ogre3D engine suggests these values.

In practice, you can use the ones for e.g. distance = 50, and vary light color

Distance	Constant	Linear	Quadratic
7	1.0	0.7	1.8
13	1.0	0.35	0.44
20	1.0	0.22	0.20
32	1.0	0.14	0.07
50	1.0	0.09	0.032
65	1.0	0.07	0.017
100	1.0	0.045	0.0075
160	1.0	0.027	0.0028
200	1.0	0.022	0.0019
325	1.0	0.014	0.0007
600	1.0	0.007	0.0002
3250	1.0	0.0014	0.000007

Shader implementation

Create a float value for attenuation and set its default value to 1.

```
float attenuation = 1.0;
```

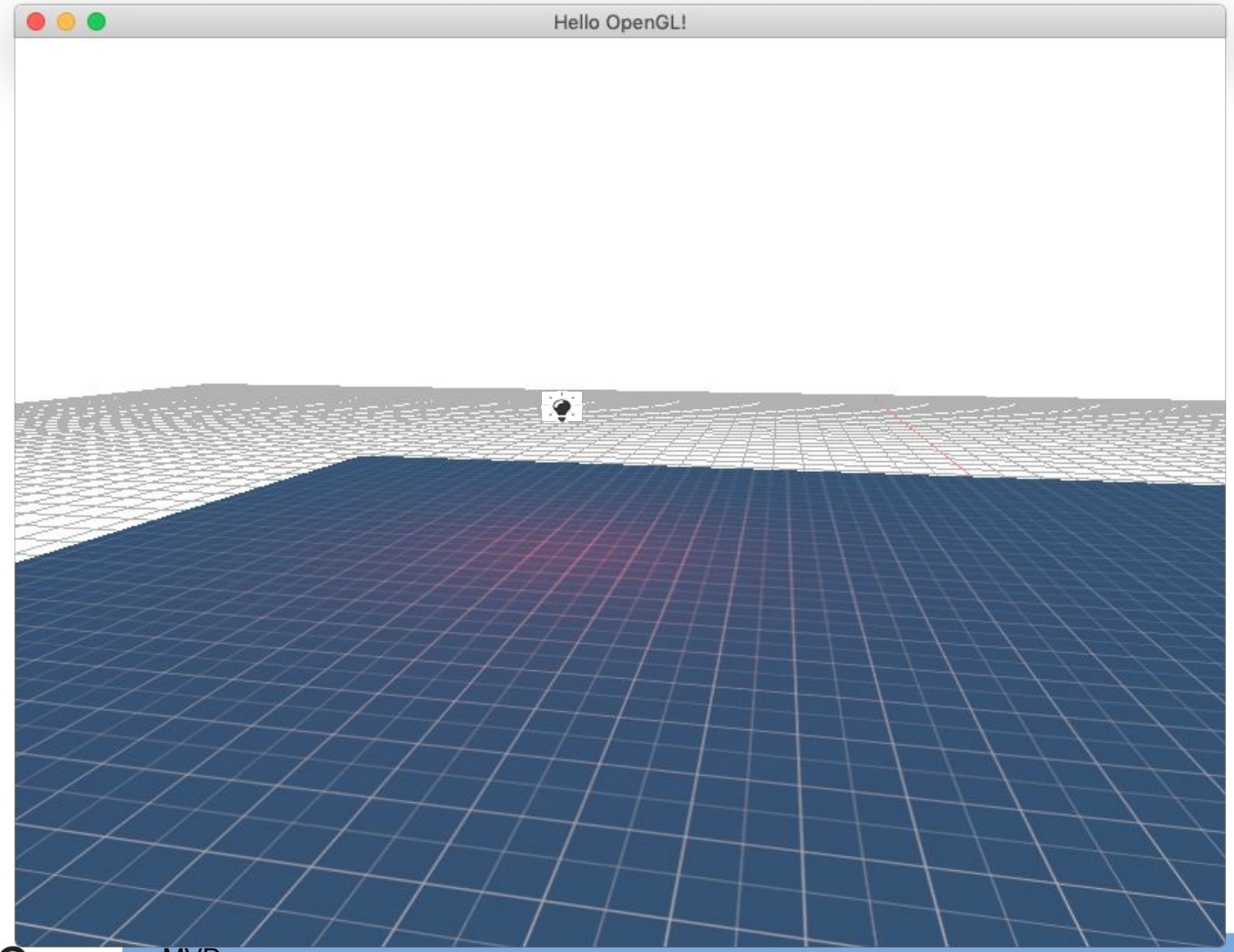
if light is a point light, modify this value

multiply final calculation by attenuation value

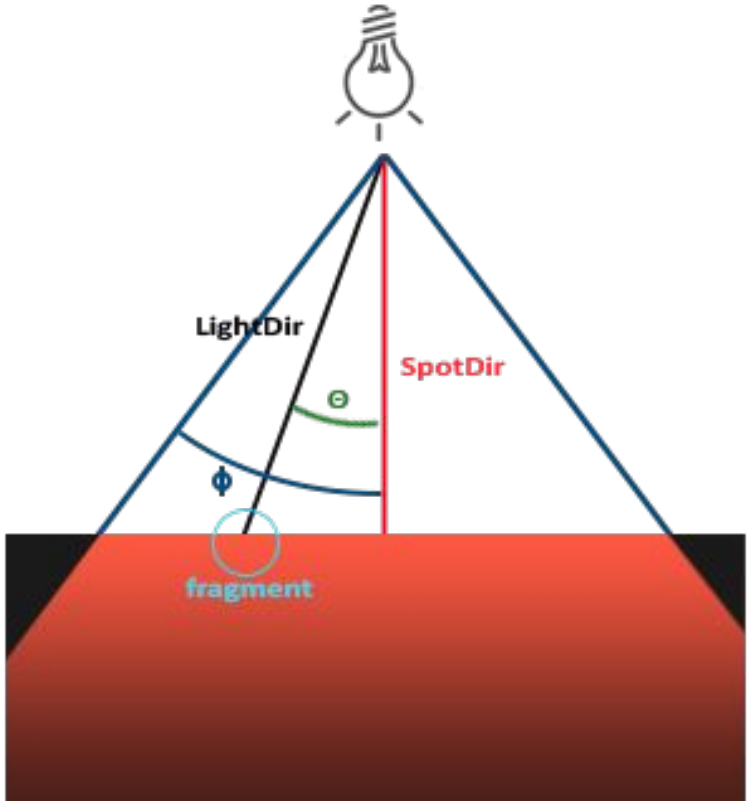
Task: Implement point light attenuation

1. In game.cpp, uncomment light 2:
   a. check type = 1
   b. check attenuation numbers
2. In shader
   a. add linear_att and quadratic_att to light struct
   b. for each light: if light.type > 0:
      i. get vector to light
      ii. L = normalize(distance_to_light)
      iii. implement attenuation using distance to light (use length())
3. In setMaterialUniforms:
   a. set new attenuation uniforms
4. Experiment with distance!

Should look like this



Spotlights



LightDir: vector **from Light to Fragment** (i.e. - L)

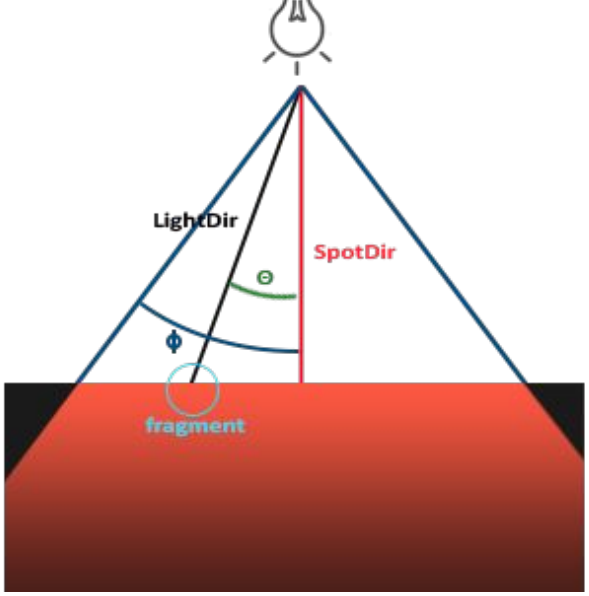
SpotDir: the direction of light

Phi ϕ: the cutoff angle that specifies the spotlight's radius. Everything outside this angle is not lit by the spotlight.

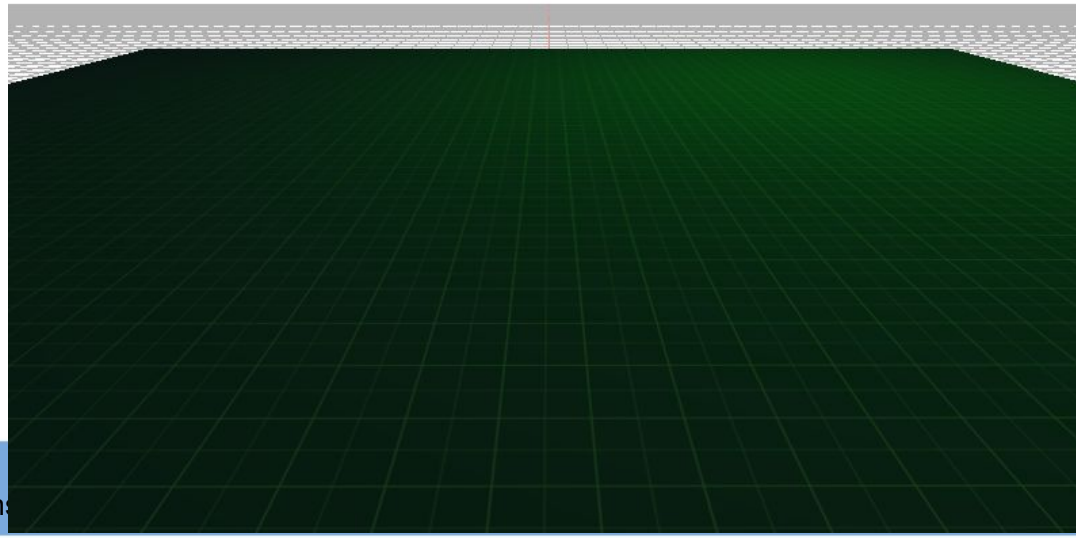
Theta θ: the angle between the LightDir vector and the SpotDir vector. The θ value should be smaller than the Φ value to be inside the spotlight.

Spotlight calculations

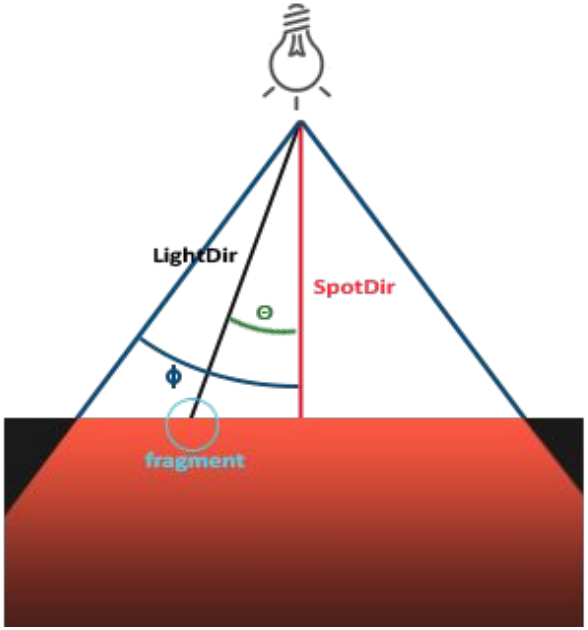
We have to calculate the dot product of lightdir (L) and the spot direction D



cosine θ = DdotL



Spotlight calculations



Need to cut off angle based on field of view of cone

send to shader:

```
cos( spotcone_fov_radians / 2)
```

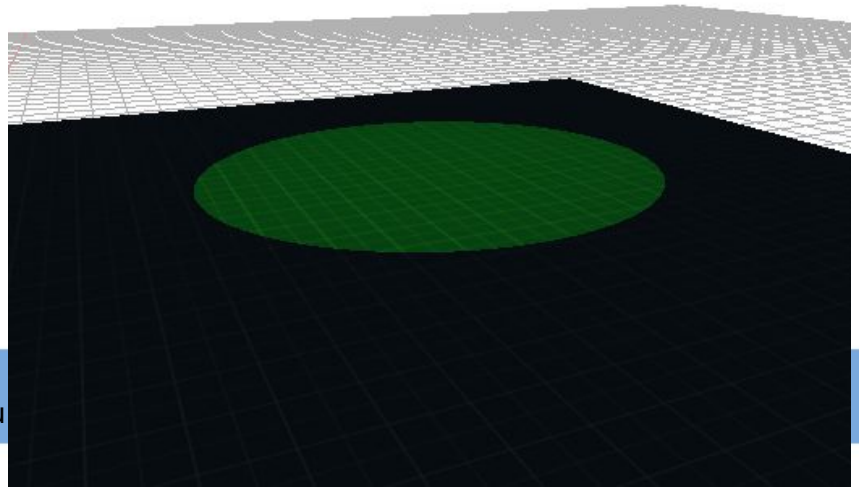
e.g. if fov is 30 degrees, need to send

```
cos (30*DEG2RAD / 2)
```

Spotlight calculations

cos_theta = DdotL (make sure L is *light -> point*)

```
if (cos_theta > cos_outercone_div2)
    spot_cone_intensity = 0.0;
```



Task: implement spotlight

1. In game, for light 3:
   a. uncomment code
2. In setMaterialUniforms_, send cosine of spot cone to shader
3. In shader:
   a. add spot_inner_cosine to light struct
   b. if light[i].type == 2
      i. calculate D and L correctly
      ii. cos_theta = DdotL
      iii. if cos_theta > spot_cone_cosine: spot_cone_intensity = 1.0

Soft boundary spotlight

Need to interpolate between inner and outer boundary

spot_cone_intensity = 1.0 at inner boundary, 0.0 at outer boundary

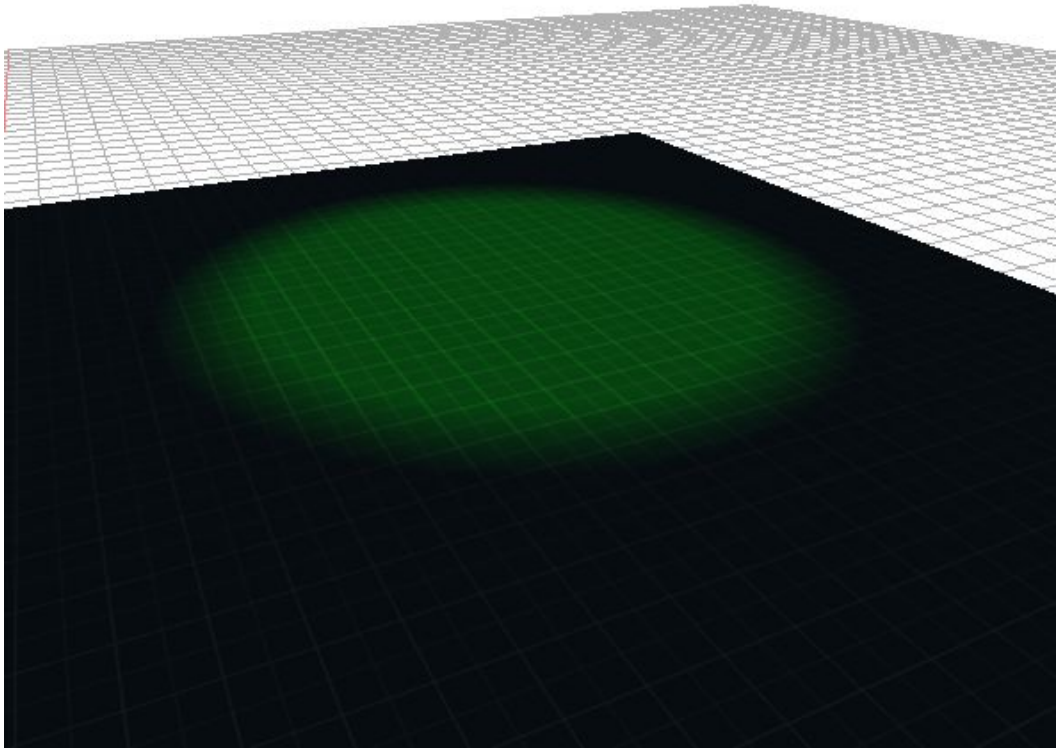
Soft boundary spotlight

$$\text{Intensity} = \frac{(\theta - \gamma)}{(\phi - \gamma)}$$

θ = *DdotL*

γ = *cosine outer cone*

ϕ = *cosine inner cone*



Clamp in shader

```
spot_cone_intensity = clamp(intensity, 0.0, 1.0);
```

Clamping makes sure we don’t over or under interpolate!

Task

1) Send outer cone to shader
2) Modify shader calculations to do interpolations
3) Put all three lights together