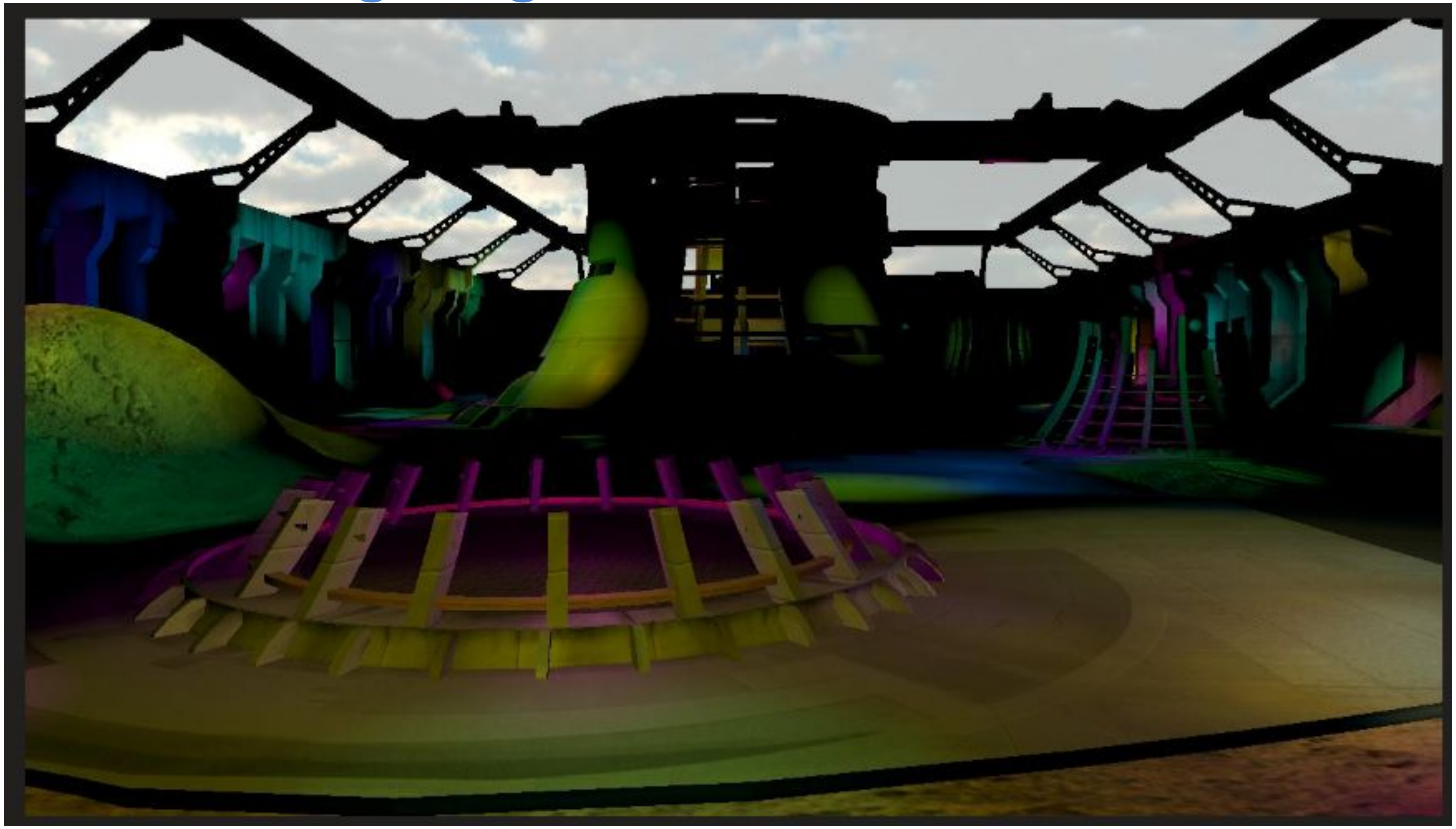
MVD: Advanced Graphics 1

17 - Light Volumes

alunthomas.evans@salle.url.edu

laSalle ENG
Universitat Ramon Llull

Deferred Lighting is famous for LOTS OF LIGHTS



e.g. 256 lights

Forward rendering

```
For every object in scene:
    For every fragment corresponding to that object:
        For every light (including shadows):
            Calculate shading
```

Inefficient because **multiply objects may share same pixel**

Our current deferred shader

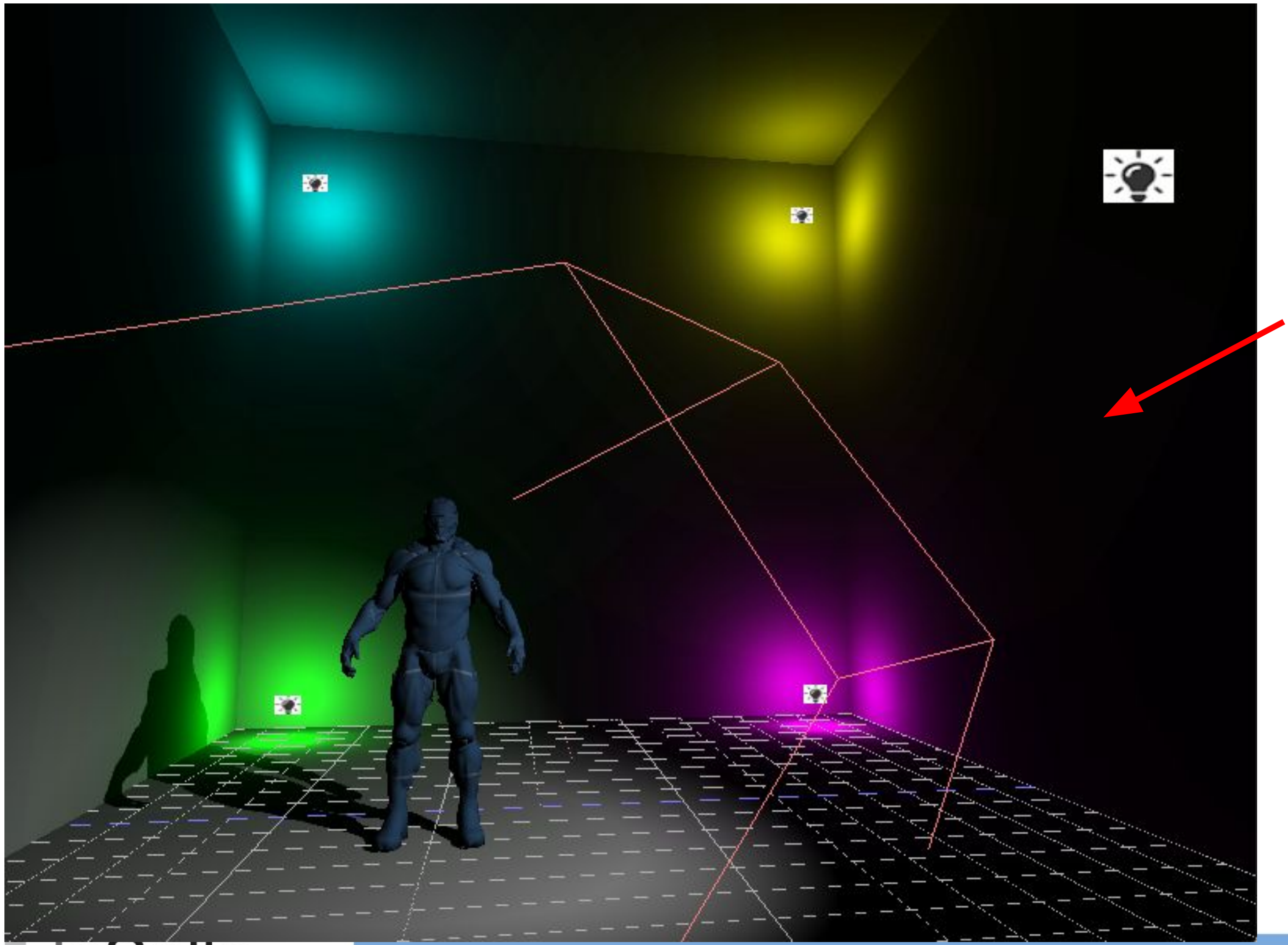
```
For every fragment in gbuffer (dimensions of screen):
    for every light:
        calculate shading
```

Faster because we have **removed one loop.**

-> **guarantee that each pixel will only be shaded once**

BUT!!!

Light attenuation means we are still calculating shading for lights that may not even touch an object

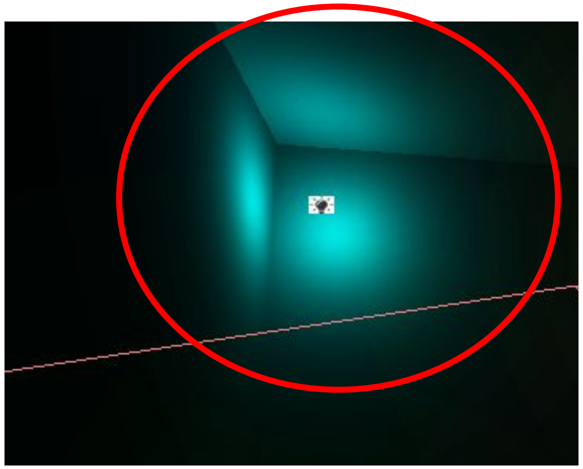


We are doing light calculations in this dark region for no reason!

Light volumes

The solution to this, and the reason that deferred shading is so popular, as it allows us to restrict the fragments of the screen that we use to draw.

e.g. Only do lighting calcs for blue light in volume surrounding it!



Calculate volume radius of point light

For point light, we can calculate the radius based on the attenuation equation (remember that)

$$F_{light} = \frac{I}{K_c + K_l * d + K_q * d^2}$$

Want to solve for *d* when light is 0

But it's a quadratic equation so we must set F_{light} = 0.01ish (many examples use 5/256)

Calculating radius

Solve the quadratic equation to get radius, *x*

$$\frac{5}{256} * Attenuation = I_{max}$$

$$5 * Attenuation = I_{max} * 256$$

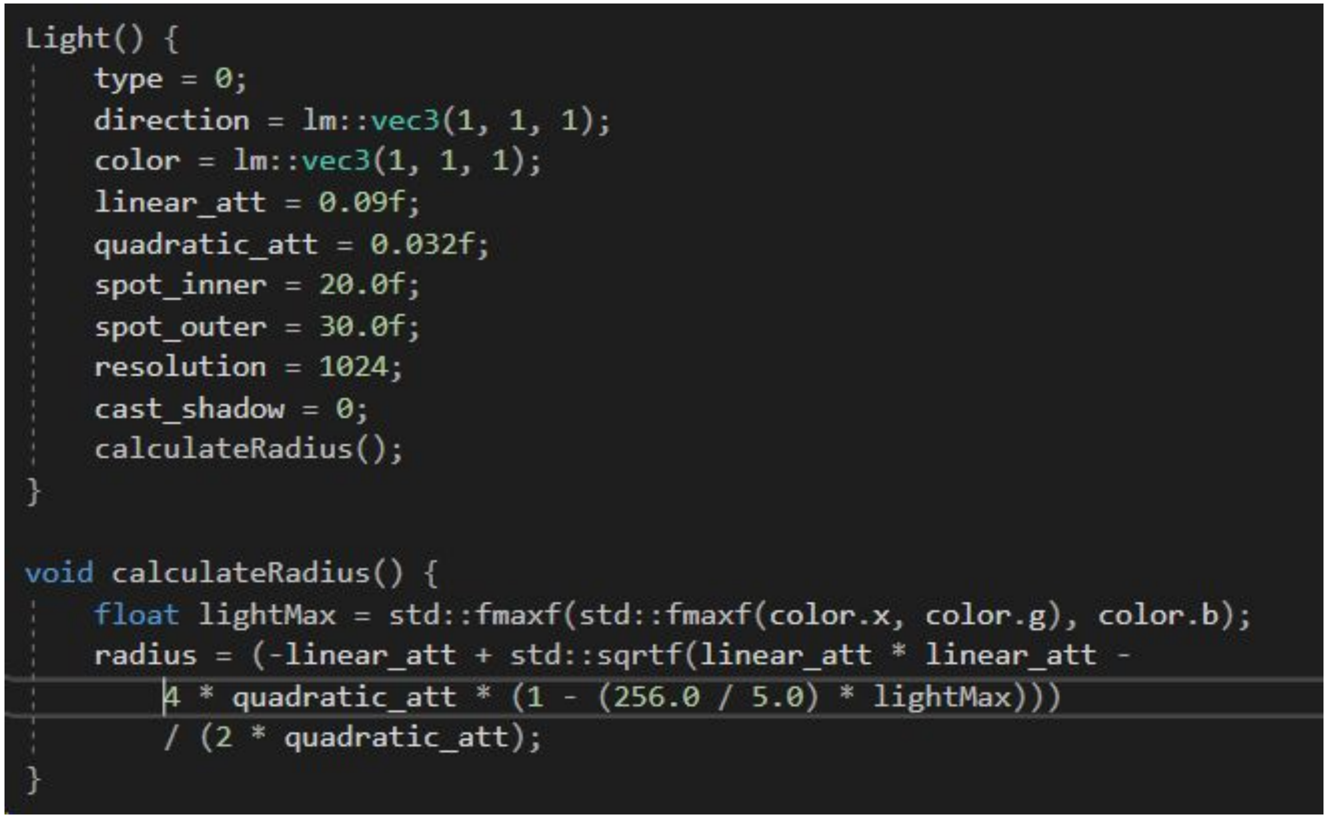
$$Attenuation = I_{max} * \frac{256}{5}$$

$$K_c + K_l * d + K_q * d^2 = I_{max} * \frac{256}{5}$$

$$K_q * d^2 + K_l * d + K_c - I_{max} * \frac{256}{5} = 0$$

$$x = \frac{-K_l + \sqrt{K_l^2 - 4 * K_q * (K_c - I_{max} * \frac{256}{5})}}{2 * K_q}$$

Set radius in Light Component constructor



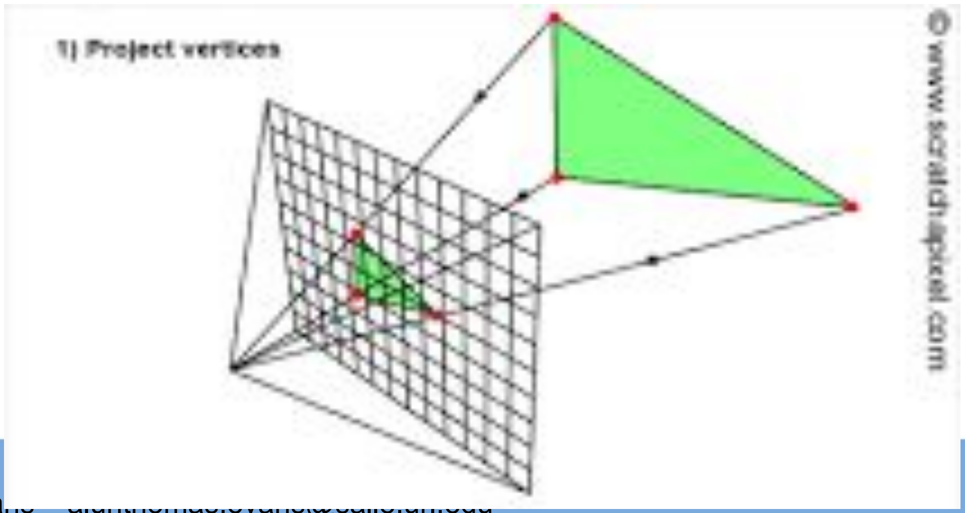
```
Light() {
    type = 0;
    direction = lm::vec3(1, 1, 1);
    color = lm::vec3(1, 1, 1);
    linear_att = 0.09f;
    quadratic_att = 0.032f;
    spot_inner = 20.0f;
    spot_outer = 30.0f;
    resolution = 1024;
    cast_shadow = 0;
    calculateRadius();
}

void calculateRadius() {
    float lightMax = std::fmaxf(std::fmaxf(color.x, color.g), color.b);
    radius = (-linear_att + std::sqrtf(linear_att * linear_att -
        4 * quadratic_att * (1 - (256.0 / 5.0) * lightMax)))
        / (2 * quadratic_att);
}
```

Using radius

We could use the radius as a conditional in the shader but we want to try to avoid big *if* blocks as they are difficult for the GPU to optimise.

We have a bigger conditional that we can use - rasterisation!

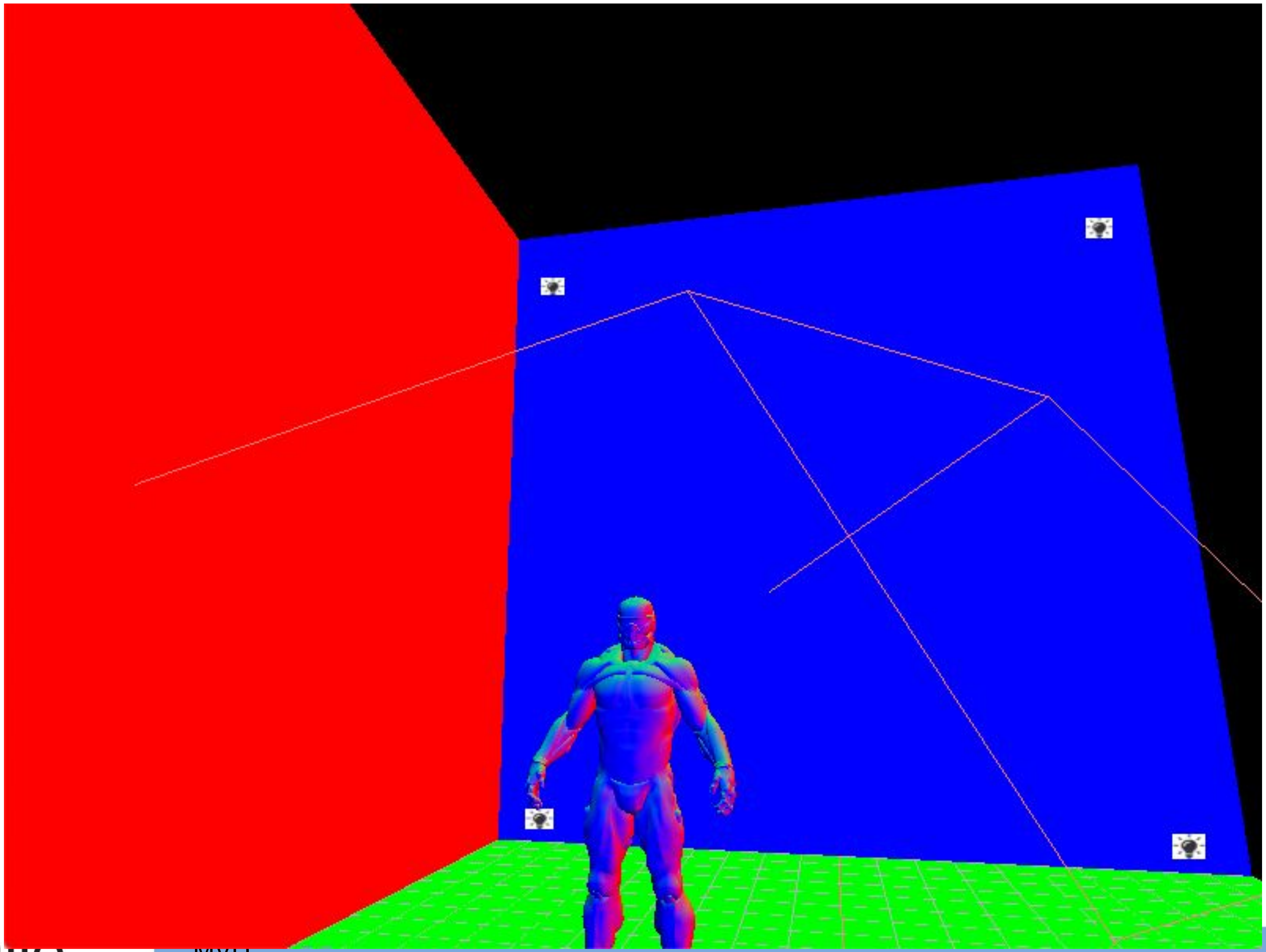


Light volume

Lets render a sphere with radius *x* around the position of a light. (Use Model-View-Projection matrix as normal)

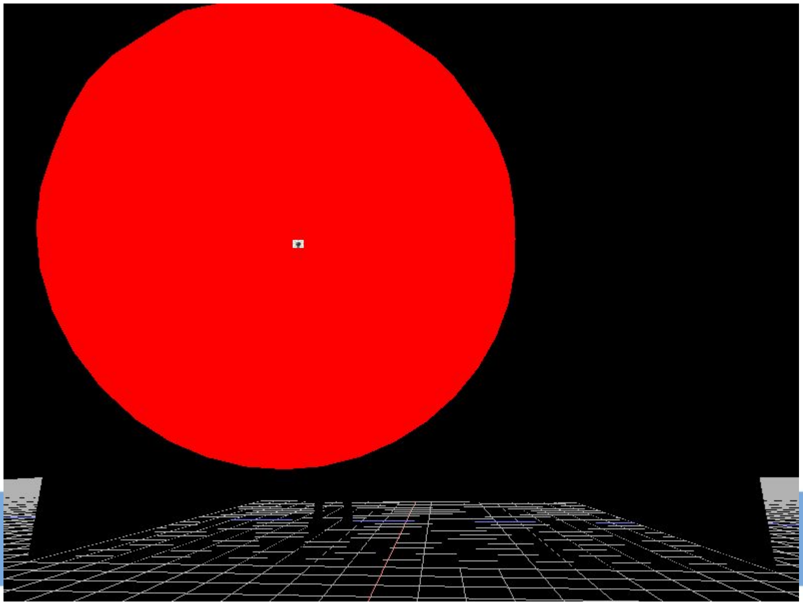
Colour of the sphere is going to be the colour of shaded G-buffer texture. i.e. we **only do lighting calculations for each light, in the fragments that surround that light**

For example, let's take any texture in gbuffer



...and render a sphere around light aka *light volume*

- Create 'deferred volume' shader
  - vertex: simple MVP shader
  - fragment: debug red!
- Load sphere mesh as geometry
- For each (point) light:
  - Create MVP from light radius and light position
  - draw sphere!

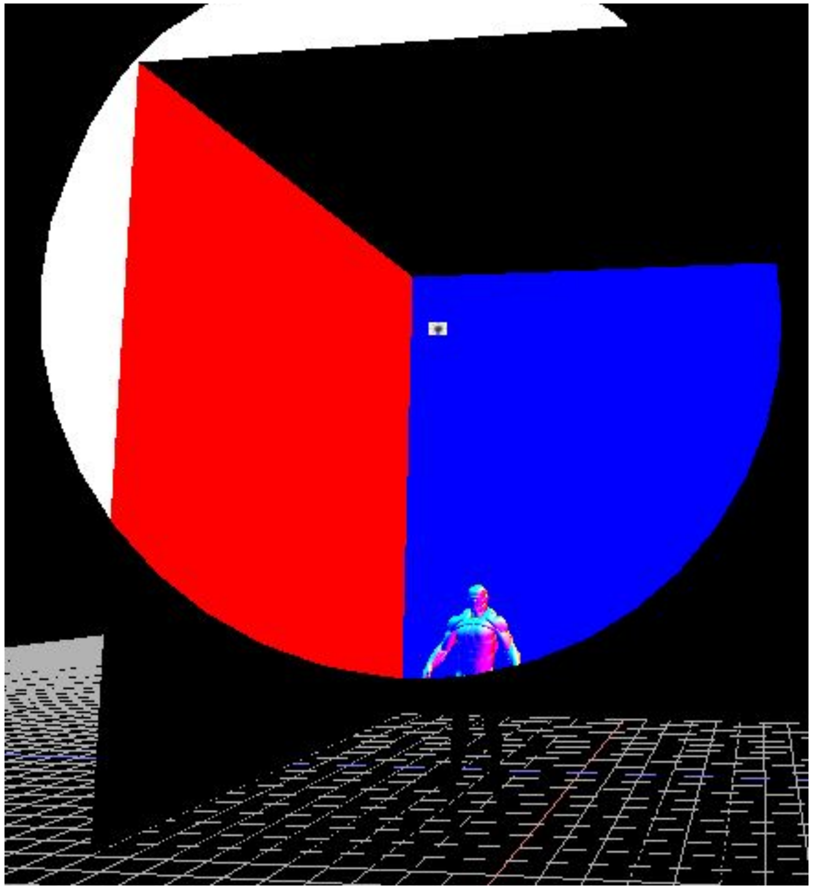


Now combine the two

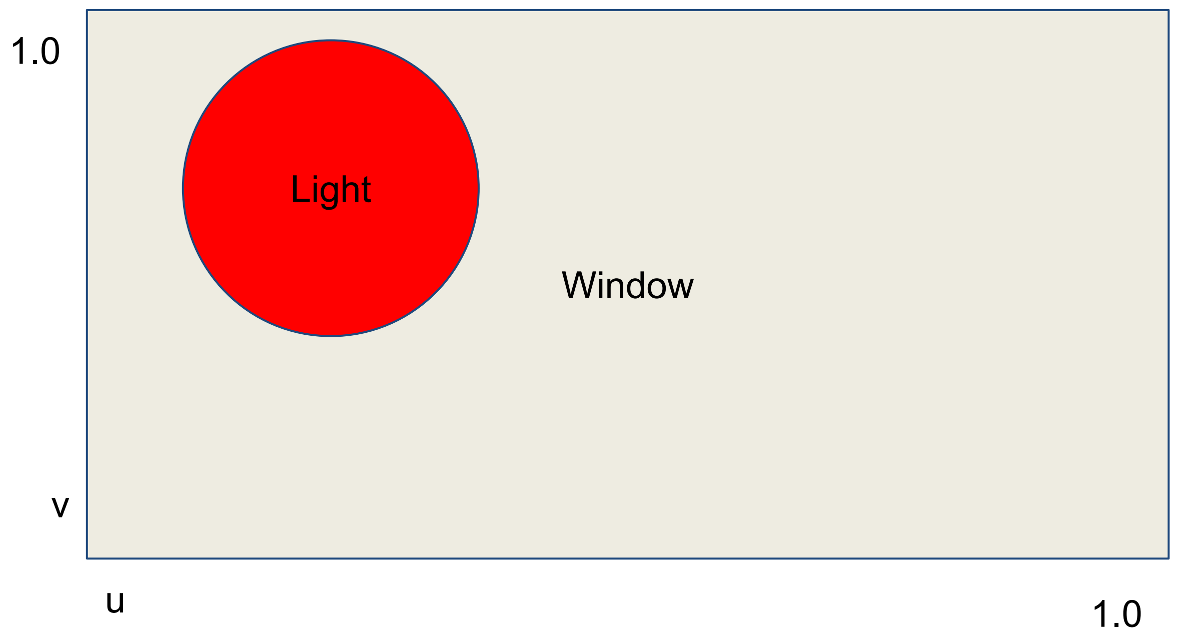
Sphere geometry is like a 'window' into the Gbuffer

Set colour of rendered sphere to be colour of a texture from the Gbuffer

But what are the texture coordinates?



Texture coordinates are window coordinates



Geometry buffer textures are *size of window* (in pixels).

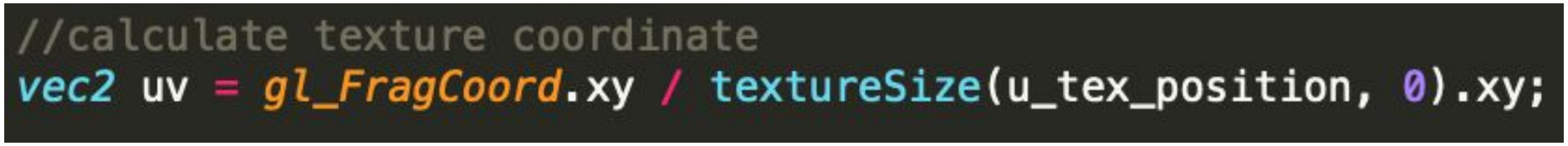
So need to take window texture coords and pass to light volume

Gettin window coordinates in shader

gl_FragCoord = *current coordinate of fragment in pixels*

textureSize = dimensions of texture

so

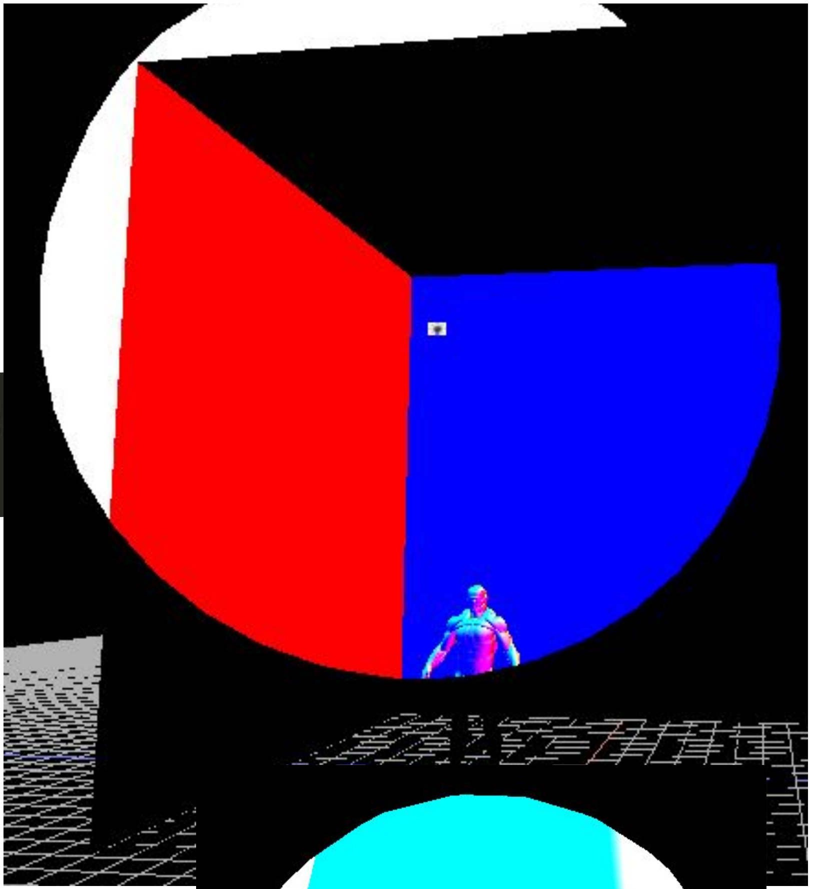


```
//calculate texture coordinate
vec2 uv = gl_FragCoord.xy / textureSize(u_tex_position, 0).xy;
```

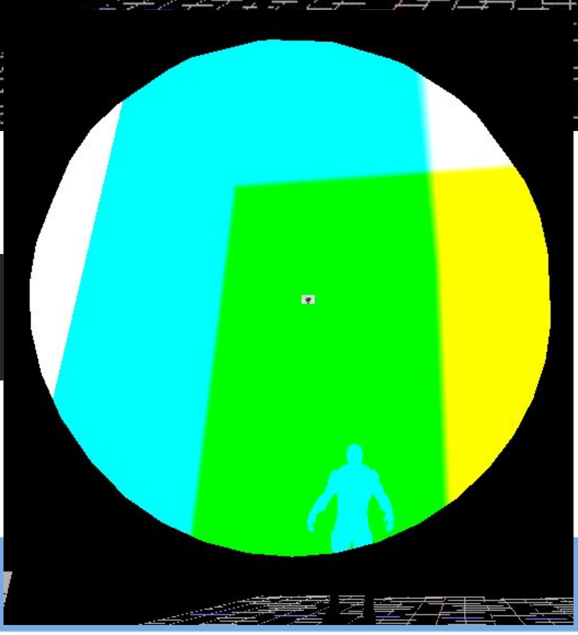
gives us texture coordinates (in 0->1) for gbuffer texture for current pixel of window

Sampling textures

```
vec3 N = texture(u_tex_normal, uv).xyz;
fragColor = vec4(N, 1.0);
```



```
vec3 position = texture(u_tex_position, uv).xyz;
fragColor = vec4(position, 1.0);
```



Adding lighting calculations

Can use exactly the same code for previous lighting calculations

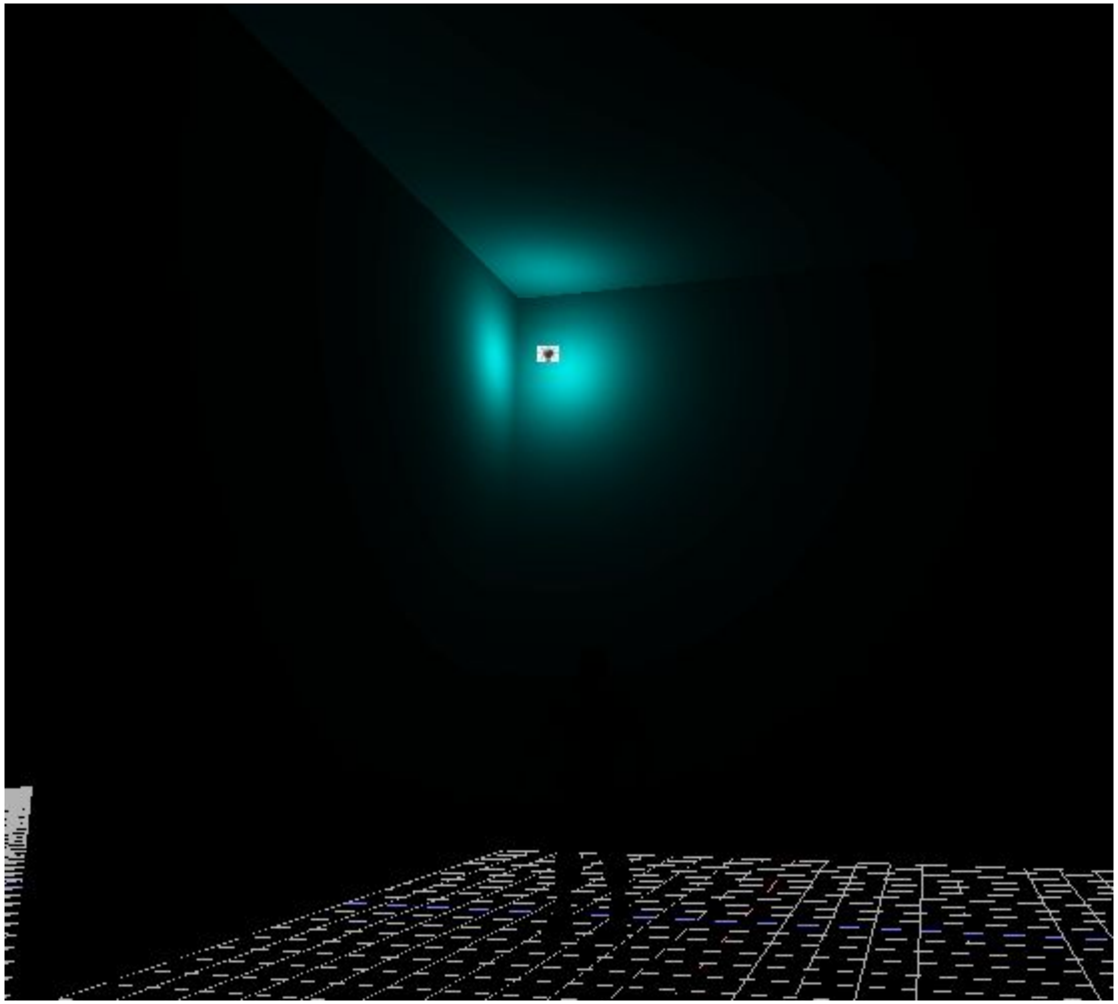
Same UBO. Same shadowmaps

Only difference there is only ONE light in the shader.

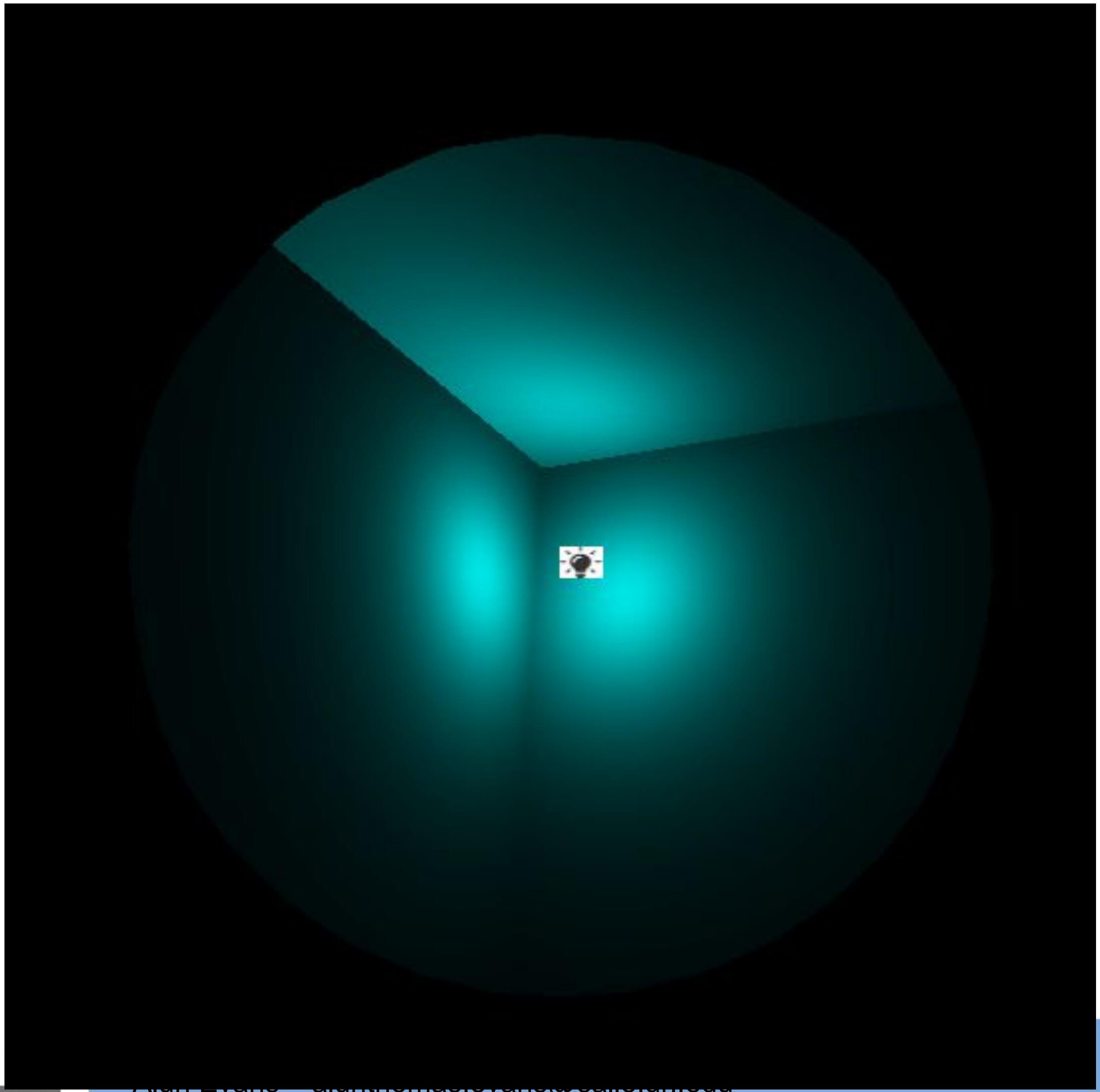
i.e. add a uniform u_light_id

and change `lights[i]` for `lights[u_light_id]`

One sphere = one light



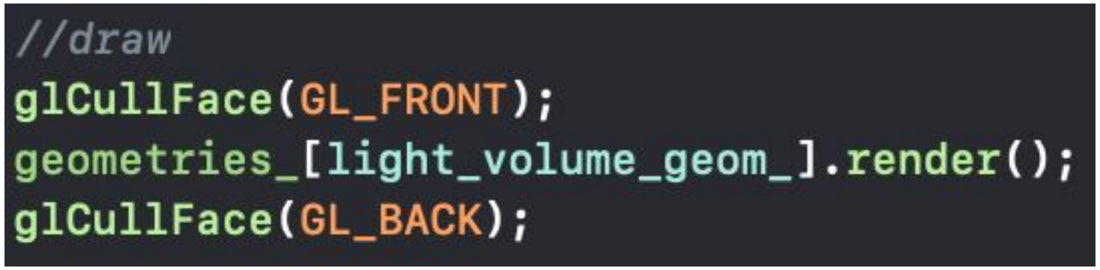
If we don’t set sphere scale correctly



What happens if camera goes into sphere?

Disappears!

Solve by culling front faces of sphere, only painting back faces



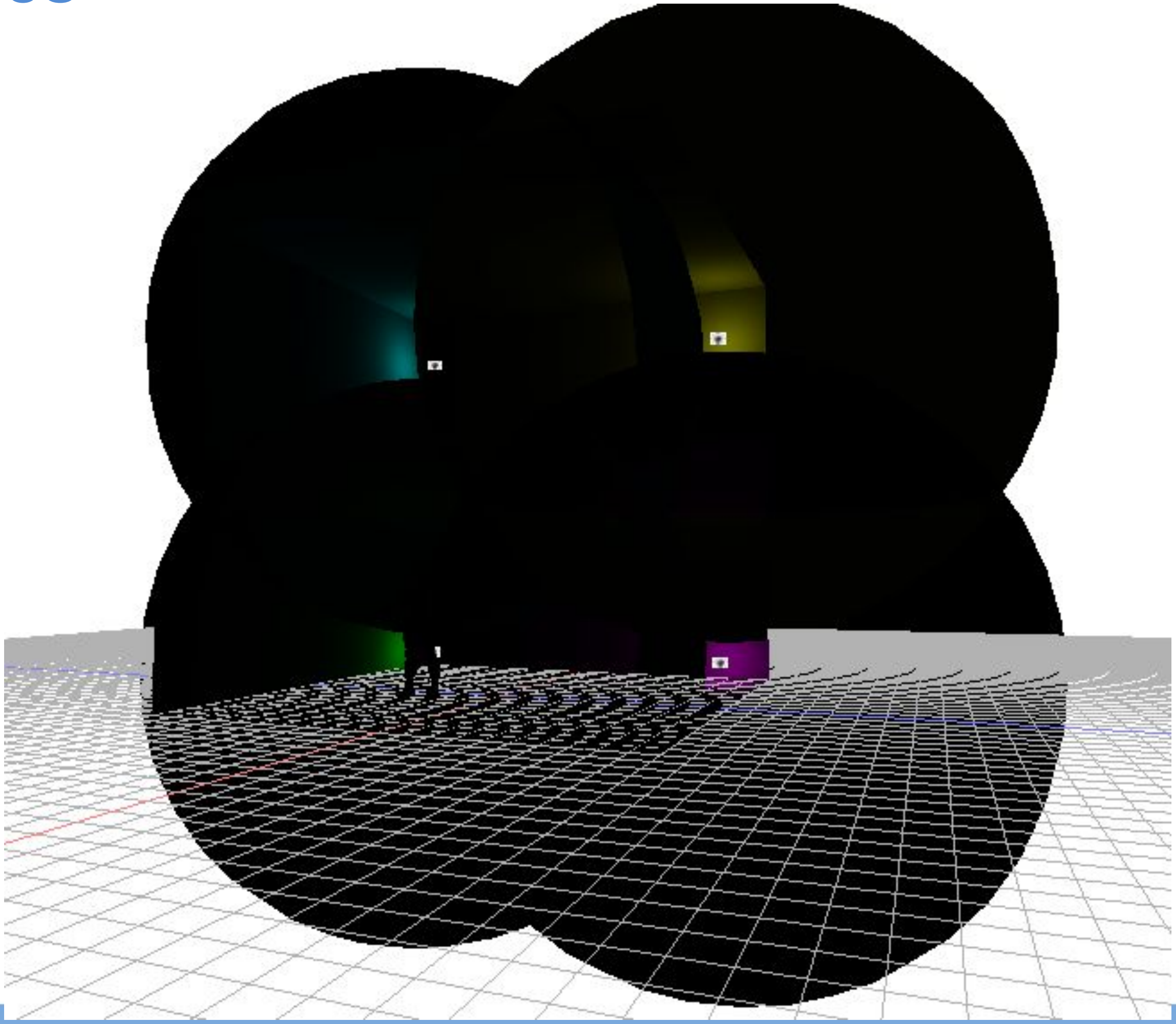
```
//draw
glCullFace(GL_FRONT);
geometries_[light_volume_geom_].render();
glCullFace(GL_BACK);
```

Multiple spheres

What's the problem?

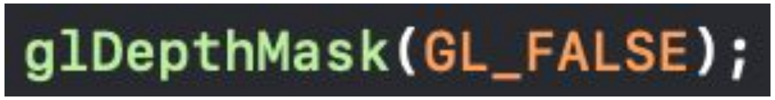
Sphere's are drawn on top of each other.

Occluding the lighting



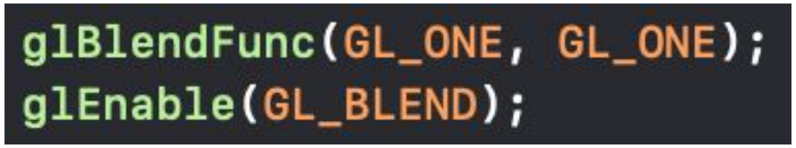
Solution: blending multiple spheres

We want to blend multiple light sources i.e. don’t want nearest sphere to occlude other spheres. First, disable depth test:



```
glDepthMask(GL_FALSE);
```

then enable blending with a 1 + 1 blend function



```
glBlendFunc(GL_ONE, GL_ONE);
glEnable(GL_BLEND);
```

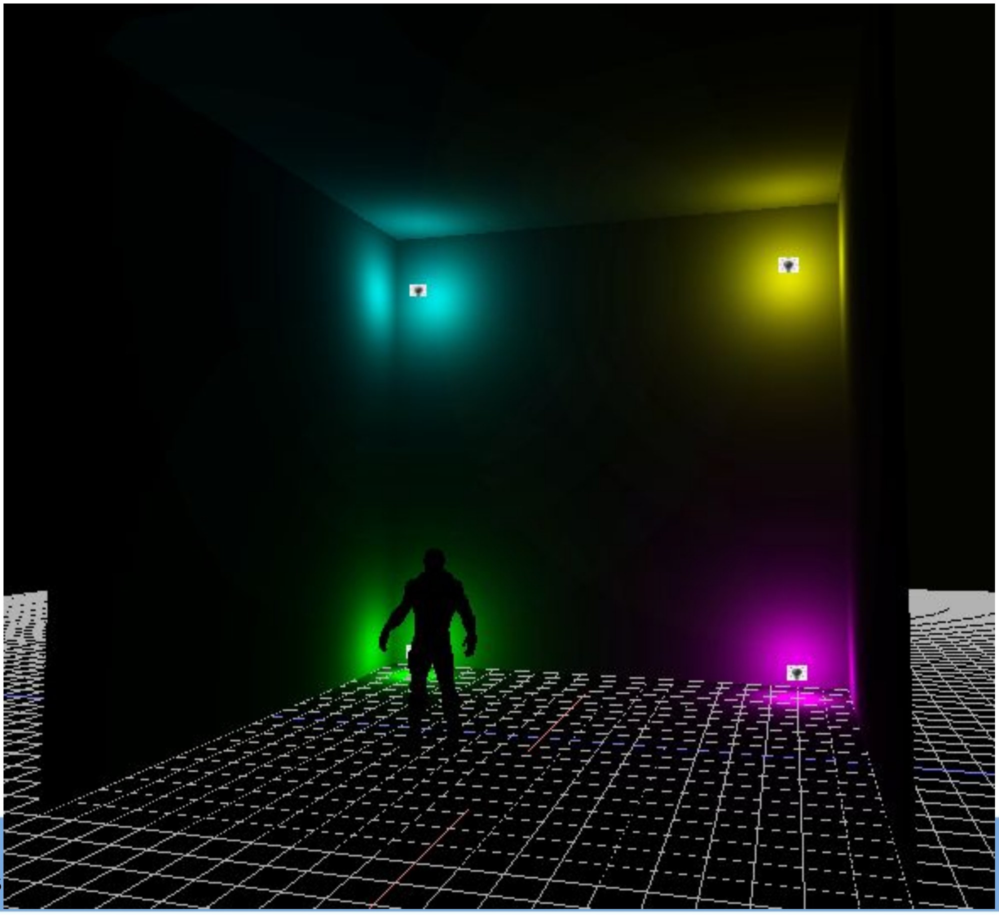
It’s super important that our background clear colour is black. *Why?*

Render multiple spheres

Render sphere for each light, one after another.

No depth test

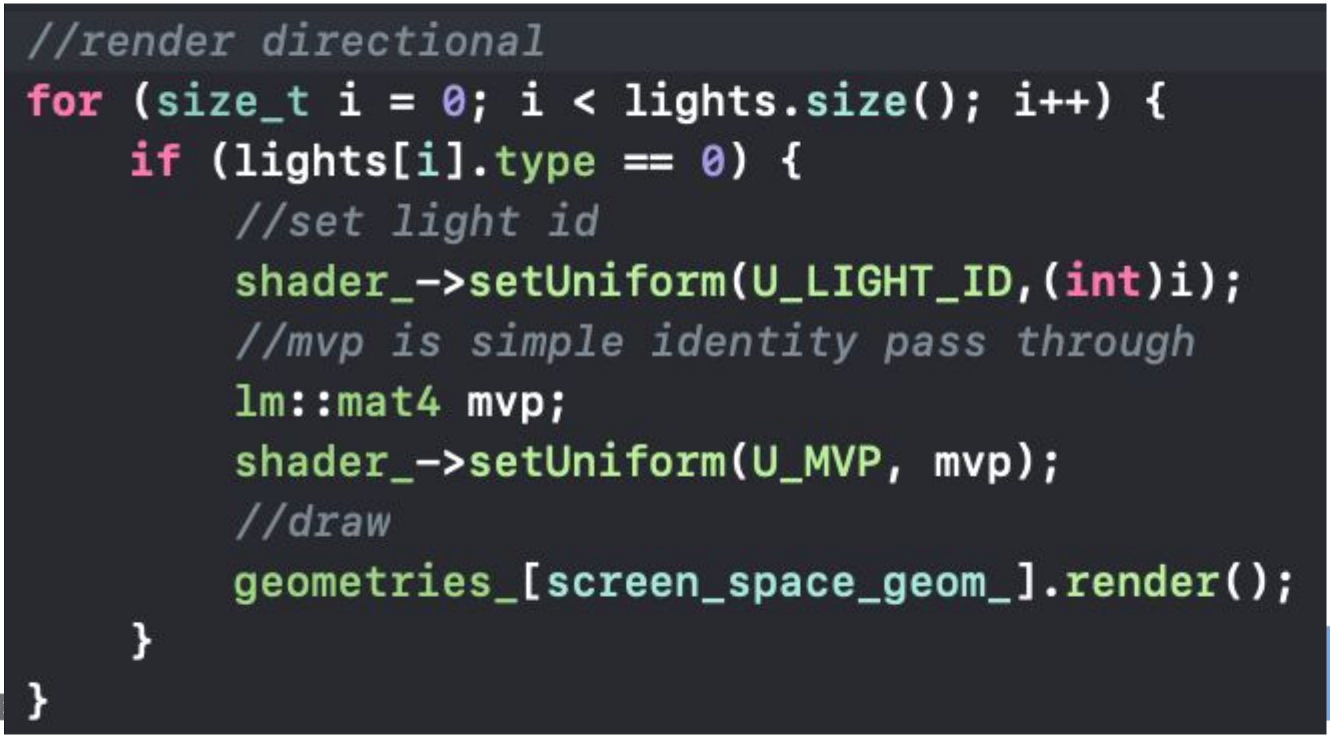
Blend all results



Adding a directional light

Directional light affects **ALL fragments** (unlike point or spot light) as it has **no attenuation**

So directional light has no volume, just draw screen space quad with ‘blank’ MVP -> use same shader as light volume



```
//render directional
for (size_t i = 0; i < lights.size(); i++) {
    if (lights[i].type == 0) {
        //set light id
        shader_->setUniform(U_LIGHT_ID,(int)i);
        //mvp is simple identity pass through
        lm::mat4 mvp;
        shader_->setUniform(U_MVP, mvp);
        //draw
        geometries_[screen_space_geom_].render();
    }
}
```

Adding a spot light

Load a cone geometry

Need to rotate to same angle as light

Also need to scale to correct dimensions