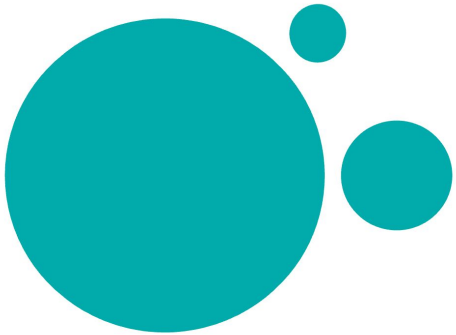
Shadow Mapping implementation in WebGL

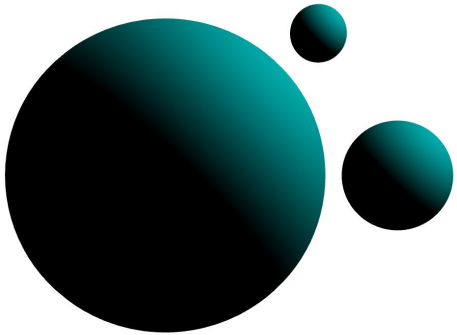
Daniele S. Cardullo - 2127806

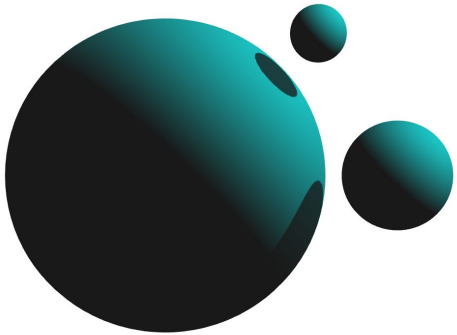
The need for shadows

Shadows are important for increasing the realism and depth of the scene.

It also enhance the visual perception of the spatial relationships between the objects in the scene.







How to implement shadows

- RayTraced Shadows: fire a primary ray then fire another ray directed towards the light source from the point hit by the primary ray, if this ray intersect an object than the point is in shadow.
  - this requires GPU intensive computations!
- **Shadow Mapping:** decide whether the light can or cannot see a point, if the light can’t see you then you’re in shadow.
  - Much faster and less computationally intensive than RayTracing
  - Less accurate
  - More oftenly used for static scenes

Computing Shadow Maps

To create shadow maps we need to find a way to render the scene as seen from the light source: we require a **Light POV Matrix**.

This matrix is computed as the product between an orthographic projection matrix and a *“LookAt”* matrix from the light position to the center of the scene.

Depth Map

To just compute the vertices from the light pov is not sufficient, we need to render the scene outputting the **depth values** for these light pov vertices. To do so two new shaders are needed.

```
#version 300 es
layout(location=0) in vec3 a_position;

uniform mat4 u_light_mvp;
void main() {
    gl_Position = u_light_mvp * vec4(a_position, 1);
}
```

```
#version 300 es
precision mediump float;

out float fragDepth;
void main() {
    fragDepth = gl_FragCoord.z;
}
```

Dealing with the Frame Buffer

The shown fragment shader will output the fragment depth (Fragment z coordinate as seen from the light source) to the **Frame Buffer**, this creates a depth map that is passed to a Texture 2D.

```
const frame_buffer = gl.createFramebuffer();
gl.bindFramebuffer(gl.FRAMEBUFFER, frame_buffer);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.TEXTURE_2D, shadow_map, 0);

gl.viewport(0, 0, depthmap_size[0], depthmap_size[1]);
gl.drawElements(gl.TRIANGLES, scene.indices.length, gl.UNSIGNED_SHORT, 0);

gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```

Texture?

As seen in the previous snippet the frame buffer content is directed to a 2D texture. This texture is what is actually called **Shadow Map**. It is then sampled in the main frame buffer to decide whether a certain point z is higher or lower than what the depth map contains.


```
const shadow_map = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, shadow_map);
gl.texStorage2D(gl.TEXTURE_2D, 1, gl.DEPTH_COMPONENT32F, depthmap_size[0], depthmap_size[1]);

gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_COMPARE_MODE, gl.COMPARE_REF_TO_TEXTURE);

gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_COMPARE_FUNC, gl.LEQUAL);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
```

Rendering Shadows

The final step is to finally render shadows to the screen, this happens in the main fragment shader where the shadow map is sampled and comparison are performed.

```
vec3 light_pov_position = (v_light_pov * 0.5 + 0.5).xyz;
float bias = 0.04;
light_pov_position = vec3(light_pov_position.xy, light_pov_position.z - bias);

for (int i = 0; i < 4; i++) {
  vec3 biased = vec3(light_pov_position.xy + adjacentPixels[i]/1500.0, light_pov_position.z);
  float hit = texture(u_shadowmap, biased);
  visibility *= max(hit, 0.85);
}
```