Casting Shadows in OpenGL

Identify the Light Source Casting the Shadow

First, Render the Scene from that Light Source

1. Render a view from the light source – everything you cannot see must be in a shadow

Use the Z-buffer as a Depth Shadow Map

2. Generate a depth view from the light source
Second, Render the Scene as Normal, but Consult the Depth Map to Decide where Lighting Applies

3. Put the eye back where it really belongs. Render that view. Every time you create a pixel in the scene, compare its 3D location against the depth map. If the light-position camera could not see it before, don’t allow lighting to be applied to it now.

OpenGL Shadow Demo Program: The Depth Shadow Map

The depth shadow map is created from the point of view of the light source.

The rendering is done into an off-screen framebuffer and only renders the depth, not any colors.

In this grayscale image, dark colors are nearest to the eye, light colors are farther away.

OpenGL Demo Program: Creating the Off-screen Depth Shadow Map Framebuffer

```cpp
// create a framebuffer object and a depth texture object:
glGenFramebuffers(1, &DepthFramebuffer);
glGenTextures(1, &DepthTexture);

//Create a texture that will be the framebuffer’s depth buffer
glBindTexture(GL_TEXTURE_2D, DepthTexture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);

// attach texture to framebuffer as depth buffer:

// attach texture to framebuffer as depth buffer:
```

OpenGL Demo Program: Rendering into the Depth Shadow Map

```cpp
// first pass, render from light’s perspective, store depth of scene in texture

// these matrices are the equivalent of projection and view matrices

// this matrix is the transformation matrix that the vertex shader will use instead of glm::modelViewProjectionMatrix:
```
OpenGL Demo Program: Rendering using the Depth Shadow Map

```cpp
#version 330 compatibility
uniform mat4 uLightSpaceMatrix;
uniform mat4 uAnim;
void main()
{
    gl_Position = uLightSpaceMatrix * uAnim * gl_Vertex;
}
```

```cpp
#version 330 compatibility
uniform vec3 uColor;
void main()
{
    gl_FragColor = vec4(uColor, 1.);    // really doesn't matter...
}
```

```cpp
#version 330 compatibility
uniform vec3 uColor;
uniform sampler2D uShadowMap;
in vec4 vFragPosLightSpace;
in vec3 vNs;
in vec3 vLs;
in vec3 vEs;
const float BIAS = 0.01;
const vec3 SPECULAR_COLOR = vec3(1., 1., 1.);
const float SHININESS = 8.;
const float KA = 0.20;
const float KD = 0.60;
const float KS = (1. - KA - KD);

bool IsInShadow(vec4 fragPosLightSpace)
{
    // have to manually do homogenous division to make light space position in range of -1 to 1:
    vec3 projection = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // then make it from 0 to 1:
    projection = 0.5 * projection + 0.5;
    // get closest depth from light's perspective
    float closestDepth = texture(uShadowMap, projection.xy).r;
    // get current depth:
    float currentDepth = projection.z;
    bool isInShadow = (currentDepth - BIAS) > closestDepth;
    return isInShadow;
}
```

```cpp
#version 330 compatibility
uniform vec3 uColor;
in vec4 vFragPosLightSpace;
in vec3 vNs;
in vec3 vLs;
in vec3 vEs;
const float BIAS = 0.01;
const vec3 SPECULAR_COLOR = vec3(1., 1., 1.);
const float SHININESS = 8.;
const float KA = 0.20;
const float KD = 0.60;
const float KS = (1. - KA - KD);
bool IsInShadow(vec4 fragPosLightSpace)
{
    // have to manually do homogenous division to make light space position in range of -1 to 1:
    vec3 projection = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // then make it from 0 to 1:
    projection = 0.5 * projection + 0.5;
    // get closest depth from light's perspective
    float closestDepth = texture(uShadowMap, projection.xy).r;
    // get current depth:
    float currentDepth = projection.z;
    bool isInShadow = (currentDepth - BIAS) > closestDepth;
    return isInShadow;
}
```
void main( )
{
    vec3 normal = normalize(vNs);
    vec3 light = normalize(vLs);
    vec3 eye = normalize(vEs);
    float d = 0.;
    float s = 0.;
    vec3 lighting = KA * uColor;
    bool isInShadow = IsInShadow(vFragPosLightSpace);
    if( uShadowsOn != 0 )
        isInShadow = false;    // if in ShadowOff mode, nothing should be considered in a shadow
    if( ! isInShadow )
    {
        d = dot(normal, light);
        if(d > 0.)
        {
            vec3 diffuse = KD * d * uColor;
            lighting += diffuse;
            vec3 refl = normalize( reflect(-light, normal ) );
            float dd = dot(eye, refl);
            if( dd > 0. )
            {
                s = pow( dd, SHININESS );
                vec3 specular = KS * s * SPECULAR_COLOR;
                lighting += specular;
            }
        }
    }
    gl_FragColor = vec4( lighting, 1. );
}

OpenGL Demo Program: Rendering using the Depth Shadow Map

OpenGL Demo Program: Rendering into the Depth Shadow Map

How Did the Demo Program Render that 2D Shadow Map?

In shadows.cpp

DisplayShadowMap.vert

DisplayShadowMap.frag