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
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.

Second, Render the Scene as Normal, but Consult the Depth Map to Decide where Lighting Applies

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:
glBindFramebuffer(GL_FRAMEBUFFER, DepthFramebuffer);

// force opengl to accept a framebuffer that doesn’t have a color buffer in it:
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);

OpenGL Demo Program: Rendering into the Depth Shadow Map

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

glBindFramebuffer(GL_FRAMEBUFFER, DepthFramebuffer);
glClear(GL_DEPTH_BUFFER_BIT);

// these matrices are the equivalent of projection and view matrices:
glm::mat4 lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, 1.f, 20.f);

In shadows.cpp:
```cpp
InitGraphics();
```
```cpp
void main() {
    vec3 LightPosition = vec3(uLightX, uLightY, uLightZ);
    vec4 EPosition = uView * uModel * gl_Vertex;
    vNs = normals(mat4(uModel) * gl_Normal);
    vEs = vec3(0., 0., 0.) - EPosition.xyz;
    gl_Position = uProj * uView * uModel * gl_Vertex;
}
```
void main()
{
    vec3 normal = normalize(vNs);
    vec3 light = normalize(vLs);
    vec3 eye = normalize(vEs);
    float d = 0.0;
    float s = 0.0;
    vec3 lighting = KA * uColor;
    bool isInShadow = IsInShadow(vFragPosLightSpace);
    if( ! isInShadow )
    {
        d = dot(normal, light);
        if(d > 0.0)
        {
            vec3 diffuse = KD * d * uColor;
            lighting += diffuse;
            vec3 refl = normalize(reflect(-light, normal));
            float dd = dot(eye, refl);
            if(dd > 0.0)
            {
                s = pow(dd, SHININESS);
                vec3 specular = KS * s * SPECULAR_COLOR;
                lighting += specular;
            }
        }
    }
    gl_FragColor = vec4(lighting, 1.0);
}

OpenGL Demo Program: Rendering using the Depth Shadow Map

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

In shadows.cpp
DisplayShadowMap->Use();
DisplayShadowMap->SetUniformVariable((char*)"uShadowMap", 0);

glm::mat4 model = glm::mat4(1.f);
DisplayShadowMap->SetUniformVariable((char*)"uModel", model);

glm::vec3 eye = glm::vec3(0., 0., 1.);
DisplayShadowMap->SetUniformVariable((char*)"uEye", eye);

glm::vec3 up = glm::vec3(0., 1., 0.);
DisplayShadowMap->SetUniformVariable((char*)"uUp", up);

glm::mat4 view = glm::lookAt(eye, look, up);
DisplayShadowMap->SetUniformVariable((char*)"uView", view);

glm::mat4 proj = glm::ortho(-0.6f, 0.6f, -0.6f, 0.6f, 0.1f, 100.f);
DisplayShadowMap->SetUniformVariable((char*)"uProj", proj);

glBegin(GL_QUADS);
gTexCoord2f(0., 0.);
glVertex3f(-1., -1., 0.);
gTexCoord2f(1., 0.);
glVertex3f(1., -1., 0.);
gTexCoord2f(1., 1.);
glVertex3f(1., 1., 0.);
gTexCoord2f(0., 1.);
glVertex3f(-1., 1., 0.);
gEnd();