Casting Shadows in OpenGL

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

Identify the Light Source Casting the 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.

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

- In shadows.cpp:
  ```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);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

  // attach texture to the current framebuffer as a depth buffer:
  glBindFramebuffer(GL_FRAMEBUFFER, DepthFramebuffer);
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, DepthTexture, 0);

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

  // first pass, render from light’s perspective, store depth of scene in texture
  glBindFramebuffer(GL_FRAMEBUFFER, DepthFramebuffer);
  glClear(GL_DEPTH_BUFFER_BIT);
  glDrawBuffer(GL_NONE);
  glReadBuffer(GL_NONE);

  glEnable(GL_DEPTH_TEST);
  glShadeModel(GL_FLAT);
  glDisable(GL_NORMALIZE);

  // these matrices are the equivalent of projection and view matrices:
  glm::mat4 lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, 1.0f, 20.0f);
  glm::vec3 lightPos(LightX, LightY, LightZ);
  glm::mat4 lightView = glm::lookAt(lightPos, glm::vec3(0., 0., 0.), glm::vec3(0., 1., 0.));

  // this matrix is the transformation matrix that the vertex shader will use instead of glModelViewProjectionMatrix:
  glm::mat4 lightSpaceMatrix = lightProjection * lightView;

  glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
  GetDepth->Use();
  GetDepth->SetUniformVariable((char*)"uLightSpaceMatrix", lightSpaceMatrix);
  glm::vec3 color = glm::vec3(0., 1., 1.);
  GetDepth->SetUniformVariable((char*)"uColor", color);
  DisplayOneScene(GetDepth); GetDepth->Use(0);

  glBindFramebuffer(GL_FRAMEBUFFER, 0);
  ```

In shadows.cpp:

- In shadows.cpp: Display, 1

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

OpenGL Demo Program: Rendering into the Depth Shadow Map

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

The rendering is done into a texture and only renders the depth, not any colors. (Normally, we would render both, but in this case, we only care about the depth.)

In this grayscale depth image, dark colors are nearest to the eye, light colors are farther away.
OpenGL Demo Program: Rendering into the Depth Shadow Map

```c
// GetDepth.vert
uniform mat4 uLightSpaceMatrix;
void main( )
{
    gl_Position = uLightSpaceMatrix * uAnim * gl_Vertex;
}

// GetDepth.frag
uniform vec3 uColor;
void main( )
{
    gl_FragColor = vec4(uColor, 1.);  // really doesn't matter...
}
```

OpenGL Demo Program: Rendering using the Depth Shadow Map

```c
// RenderWithShadows->Use(
Declare uniforms:
uniform mat4 uLightSpaceMatrix;
uniform mat4 uAnim;
uniform mat4 uModelView;
uniform mat4 uProj;
uniform float uLightX;
uniform float uLightY;
uniform float uLightZ;

Out variables:
out vec4 vFragPosLightSpace;
out vec3 vNs;
out vec3 vLs;
out vec3 vEs;

void main( )
{
    vec3 LightPosition = vec3(uLightX, uLightY, uLightZ);
    vec4 ECposition = uModelView * uAnim * gl_Vertex;
    vec3 tnorm = normalize( mat3(uAnim) * gl_Normal );
    vNs = tnorm;
    vLs = LightPosition - ECposition.xyz;
    vEs = vec3( 0., 0., 0. ) - ECposition.xyz;
    vFragPosLightSpace = uLightSpaceMatrix * uAnim * gl_Vertex;
    gl_Position = uProj * uModelView * uAnim * gl_Vertex;
}
```
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( ! 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. );
}

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

In shadows.cpp

```
Note that this is not a required part of a shadows program. I just did that to show you what it would look like.
```
How Did the Demo Program Render the 2D Shadow Map?

```cpp
void main( )
{
    vST = gl_MultiTexCoord0.st;
    gl_Position = uProj * uView * uModel * gl_Vertex;
}
```

```cpp
void main( )
{
    float gray = texture(uShadowMap, vST).r;
    gl_FragColor = vec4(gray, gray, gray, 1.);
}
```

DisplayShadowMap.vert

DisplayShadowMap.frag