Casting Shadows in OpenGL with Some Help from GLSL

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

Second, Render the Scene as Normal, but Consult the Depth

5. 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 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 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.
// force opengl to accept a framebuffer that doesn’t have a color buffer in it:
// attach texture to the current framebuffer as a depth buffer:

// create a framebuffer object and a depth texture object:

void

uniform vec3 uColor;

uniform mat4 uLightSpaceMatrix;
uniform mat4 uAnim;

void

out vec3 vEs; 
out vec4 vFragPosLightSpace;
uniform float  uLightZ;
uniform float  uLightX;
uniform mat4  uProj;
uniform mat4  uAnim;
uniform mat4  uLightSpaceMatrix;

vec3 LightPosition = vec3(uLightX, uLightY, uLightZ);

vec3 Up = vec3(0., 1., 0.);
vec3 XAxis = vec3(1., 0., 0.);
vec3 YAxis = vec3(0., 1., 0.);

vLs = LightPosition - ECposition.xyz;
vNs = tnorm;

float angle = (float)(45.f * 2.f * sin(M_PI * Time));

anim = glm::rotate(anim, glm::radians(angle), glm::normalize(glm::vec3(1.0, 0.0, 1.0)));

anim = glm::scale(anim, glm::vec3(0.5f));
anim = glm::translate(anim, glm::vec3(-1., 2.5 + 2.f * sin(M_PI * Time), 6.f));

color = glm::vec3(0., 1., 0.);


drawPF();
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 = vFragPosLightSpace.xyz / vFragPosLightSpace.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( ! 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. );