float LightX = -2.;

float LightY = 15.;

float LightZ = 10.;

**void**

**Display( )**

{

glutSetWindow( MainWindow );

//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.f, 1000.f);

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 glViewProjectionMatrix:

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);

// second pass:

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

glDrawBuffer(GL\_BACK);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glEnable(GL\_DEPTH\_TEST);

glShadeModel(GL\_SMOOTH);

glEnable(GL\_NORMALIZE);

GLsizei vx = glutGet(GLUT\_WINDOW\_WIDTH);

GLsizei vy = glutGet(GLUT\_WINDOW\_HEIGHT);

GLsizei v = vx < vy ? vx : vy; // minimum dimension

GLint xl = (vx - v) / 2;

GLint yb = (vy - v) / 2;

glViewport(xl, yb, v, v);

RenderWithShadows->Use();

//RenderWithShadows->SetUniformVariable((char\*)"uShadowMap", 0 );

RenderWithShadows->SetUniformVariable((char\*)"uLightX", LightX);

RenderWithShadows->SetUniformVariable((char\*)"uLightY", LightY);

RenderWithShadows->SetUniformVariable((char\*)"uLightZ", LightZ);

RenderWithShadows->SetUniformVariable((char\*)"uLightSpaceMatrix", lightSpaceMatrix);

glm::vec3 eye = glm::vec3(0., 0., 10.);

glm::vec3 look = glm::vec3(0., 0., 0.);

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

glm::mat4 view = glm::lookAt(eye, look, up);

if (Scale < MINSCALE)

Scale = MINSCALE;

glm::vec3 scale = glm::vec3(Scale, Scale, Scale);

view = glm::scale( view, scale);

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

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

view = glm::rotate(view, glm::radians(Yrot), yaxis );

view = glm::rotate(view, glm::radians(Xrot), xaxis);

RenderWithShadows->SetUniformVariable((char\*)"uView", view);

glm::mat4 proj = glm::perspective(glm::radians(90.f), 1.f, .1f, 1000.f);

RenderWithShadows->SetUniformVariable((char\*)"uProj", proj );

DisplayOneScene(RenderWithShadows);

// possibly draw the axes:

RenderWithShadows->Use(0);

if (AxesOn != 0)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(90., 1., .1, 1000.);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(0., 0., 10., 0., 0., 0., 0., 1., 0.);

glScalef((GLfloat)Scale, (GLfloat)Scale, (GLfloat)Scale);

glRotatef(Yrot, 0., 1., 0.);

glRotatef(Xrot, 1., 0., 0.);

glColor3f(1., 1., 1.);

glCallList(AxesList);

}

glutSwapBuffers( );

glFlush( );

}

**void**

**DisplayOneScene(GLSLProgram \* prog )**

{

//render a sphere:

glm::mat4 model = glm::mat4(1.f);

prog->SetUniformVariable((char\*)"uModel", model);

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

prog->SetUniformVariable((char\*)"uColor", color );

glCallList(SphereList);

//Render cubes:

model = glm::mat4(1.f);

model = glm::translate(model, glm::vec3(-1., 2.5 + 2.f \* sin(M\_PI \* Time), 10.));

model = glm::scale(model, glm::vec3(0.5));

prog->SetUniformVariable((char\*)"uModel", model);

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

prog->SetUniformVariable((char\*)"uColor", color );

glutSolidCube(1.);

model = glm::mat4(1.f);

model = glm::translate(model, glm::vec3(2.0f, 6.0f, 3.0));

float angle = (float)(45.f \* 2.f \* sin(M\_PI \* Time));

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

model = glm::scale(model, glm::vec3(0.5f));

prog->SetUniformVariable((char\*)"uModel", model);

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

prog->SetUniformVariable((char\*)"uColor", color);

glutSolidCube(2.);

prog->Use(0);

}

**GetDepth.vert:**

#version 330 compatibility

uniform mat4 uLightSpaceMatrix;

uniform mat4 uModel;

void main()

{

gl\_Position = uLightSpaceMatrix \* uModel \* gl\_Vertex;

}

**GetDepth.frag:**

#version 330 compatibility

uniform vec3 uColor;

void main()

{

gl\_FragColor = vec4(uColor, 1.); // really doesn't matter...

}

**RenderWithShadows.vert:**

#version 330 compatibility

uniform mat4 uLightSpaceMatrix;

uniform mat4 uModel;

uniform mat4 uView;

uniform mat4 uProj;

uniform float uLightX;

uniform float uLightY;

uniform float uLightZ;

out vec4 vFragPosLightSpace;

out vec3 vNs;

out vec3 vLs;

out vec3 vEs;

void main()

{

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

vec4 ECposition = uView \* uModel \* gl\_Vertex;

vec3 tnorm = normalize( mat3(uModel) \* gl\_Normal );

vNs = tnorm;

vLs = LightPosition - ECposition.xyz;

vEs = vec3( 0., 0., 0. ) - ECposition.xyz;

vFragPosLightSpace = uLightSpaceMatrix \* uModel \* gl\_Vertex;

gl\_Position = uProj \* uView \* uModel \* gl\_Vertex;

}

**RenderWithShadows.frag:**

#version 330 compatibility

uniform vec3 uColor;

uniform sampler2D uShadowMap;

in vec4 vFragPosLightSpace;

in vec3 vNs;

in vec3 vLs;

in vec3 vEs;

out vec4 fFragColor;

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;

//if(projection.z > 1.0)

//{

//isInShadow = false;

//}

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;

//float test = texture(uShadowMap, gl\_FragCoord.xy/vec2(1024,-1024)+vec2(0,1)).r;

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;

}

}

}

//lighting.b = test;

//lighting.rg \*= 0.2;

fFragColor = vec4( lighting, 1. );

}