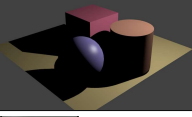

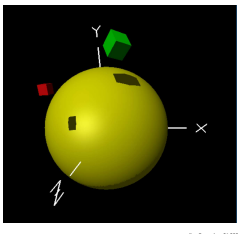



Casting Shadows in OpenGL








Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

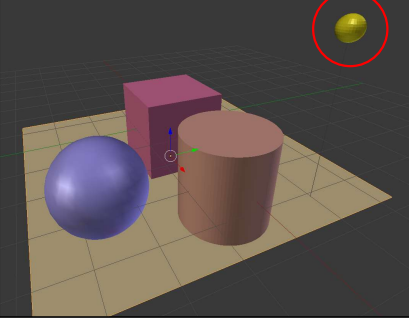






1

Identify the Light Source Casting the Shadow

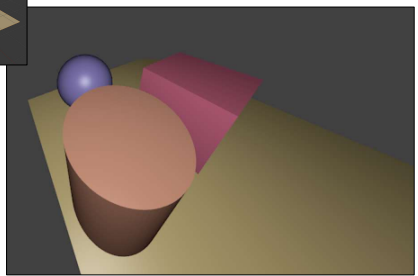







2

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

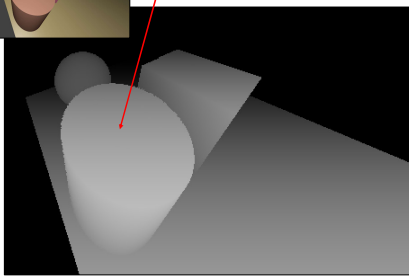







3

Use the Z-buffer as a Depth Shadow Map

2. Generate a depth view from the light source

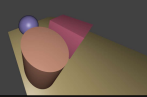
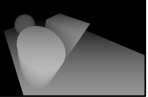


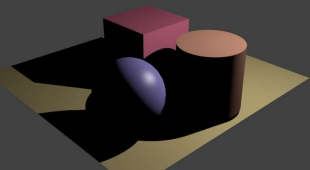





4

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.



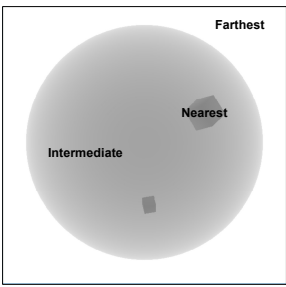
5



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.



6

OpenGL Demo Program: Creating the Off-screen Depth Shadow Map Framebuffer 7

```


// 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_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
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 framebuffer as 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:
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
    
```

In shadows.cpp:
InitGraphics()



University
Computer Graphics

7

OpenGL Demo Program: Rendering into the Depth Shadow Map 8

```

//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);

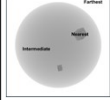
// these matrices are the equivalent of projection and view matrices
glm::mat4 lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, 1.f, 20.f);
glm::vec3 lightPos(lightX, lightY, lightZ);
glm::mat4 lightView = glm::lookAt(lightPos, glm::vec3(0, 0, 0), glm::vec3(0, 1, 0));

//this matrix is the transformation matrix that the vertex shader will use instead of glm::mat4 viewProjectionMatrix
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->UnUse();
glBindFramebuffer(GL_FRAMEBUFFER, 0);
    
```

In shadows.cpp:
Display(), I



University
Computer Graphics

8

OpenGL Demo Program: Rendering using the Depth Shadow Map 9

```

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, 8);
glm::vec3 look = glm::vec3(0, 0, 0);
glm::vec3 up = glm::vec3(0, 1, 0);
glm::mat4 modelview = glm::lookAt(eye, look, up);

if (Scale < MINSCALE) Scale = MINSCALE;
glm::vec3 scale = glm::vec3(Scale, Scale, Scale);
modelview = glm::scale(modelview, scale);
glm::vec3 xAxis = glm::vec3(1, 0, 0);
glm::vec3 yAxis = glm::vec3(0, 1, 0);
modelview = glm::rotate(modelview, glm::radians(Yrot), yAxis);
modelview = glm::rotate(modelview, glm::radians(Xrot), xAxis);
RenderWithShadows->SetUniformVariable(char*"uModelView", modelview);

glm::mat4 proj = glm::perspective(glm::radians(75.f), 1.f, 1.f, 100.f);
RenderWithShadows->SetUniformVariable(char*"uProj", proj);
DisplayOneScene(RenderWithShadows);
RenderWithShadows->UnUse();
    
```

In shadows.cpp:
Display(), II

University
Computer Graphics

9

OpenGL Demo Program: Rendering using the Depth Shadow Map 10

```

#version 330 compatibility
uniform mat4 uLightSpaceMatrix;
uniform mat4 uAnim;

void
main()
{
    gl_Position = uLightSpaceMatrix * uAnim * gl_Vertex;
}

#version 330 compatibility
uniform vec3 uColor;

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

GetDepth.vert

GetDepth.frag

University
Computer Graphics

10

OpenGL Demo Program: Rendering using the Depth Shadow Map 11

```

#version 330 compatibility
uniform mat4 uLightSpaceMatrix;
uniform mat4 uAnim;
uniform mat4 uModelView;
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 EPosition = uModelView * uAnim * gl_Vertex;
    vec3 Inorm = normalize(mat3(uAnim) * gl_Normal);
    vNs = Inorm;
    vLs = LightPosition - EPosition.xyz;
    vEs = vec3(0, 0, 0) - EPosition.xyz;
    vFragPosLightSpace = uLightSpaceMatrix * uAnim * gl_Vertex;
    gl_Position = uProj * uModelView * uAnim * gl_Vertex;
}
    
```

RenderWithShadows.vert

University
Computer Graphics

11

OpenGL Demo Program: Rendering using the Depth Shadow Map 12

```

#version 330 compatibility
uniform vec3 uColor;
uniform sampler2D uShadowMap;
uniform int uShadowDir;

in vec4 vFragPosLightSpace;
in vec3 vNs;
in vec3 vLs;
in vec3 vEs;

const float BIAS = 0.01;
const vec3 SPHERICAL_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 isShadow = (currentDepth - BIAS) > closestDepth;
    return isShadow;
}
    
```

RenderWithShadows.frag, I

University
Computer Graphics

12

OpenGL Demo Program: Rendering using the Depth Shadow Map

```

void main()
{
    vec3 normal = normalize(Na);
    vec3 light = normalize(Ls);
    vec3 eye = normalize(VEs);

    float d = 0.;
    float s = 0.;
    vec3 lighting = KA * uColor;

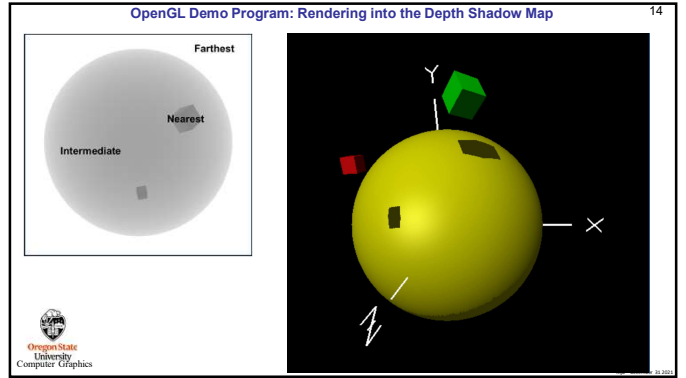
    bool isShadow = isShadow(FragPosLightSpace);
    if( uShadowOn != 0 )
        isShadow = false; // if in ShadowOff mode, nothing should be considered in a shadow
    if( !isShadow )
    {
        d = dot(normal,light);
        if( d > 0 )
        {
            vec3 diffuse = KD * uColor;
            lighting += diffuse;

            vec3 refl = normalize( reflect(-light, normal) );
            float dd = dot(eye,refl);
            if( dd > 0. )
            {
                s = pow( dd, SHININESS );
                vec3 specular = KS * SPECULAR_COLOR;
                lighting += specular;
            }
        }
    }
    gl_FragColor = vec4( lighting, 1. );
}
    
```

RenderWithShadows.frag, II

13

13



14

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.);
glm::vec3 look = glm::vec3(0., 0., 0.);
glm::vec3 up = glm::vec3(0., 1., 0.);
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, .1f, 100.f);
DisplayShadowMap->SetUniformVariable((char*)"uProj", proj);

glBegin(GL_QUADS);
glTexCoord2f(0., 0.);
glVertex3f(-1., -1., 0.);
glTexCoord2f(1., 0.);
glVertex3f( 1., -1., 0.);
glTexCoord2f(1., 1.);
glVertex3f( 1.,  1., 0.);
glTexCoord2f(0., 1.);
glVertex3f(-1.,  1., 0.);
glEnd();

DisplayShadowMap->UnUse();
    
```

15

15

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

```

#version 330 compatibility
uniform mat4 uModel;
uniform mat4 uView;
uniform mat4 uProj;

out vec2 vST;

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

#version 330 compatibility
uniform sampler2D uShadowMap;

in vec2 vST;

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

16

16