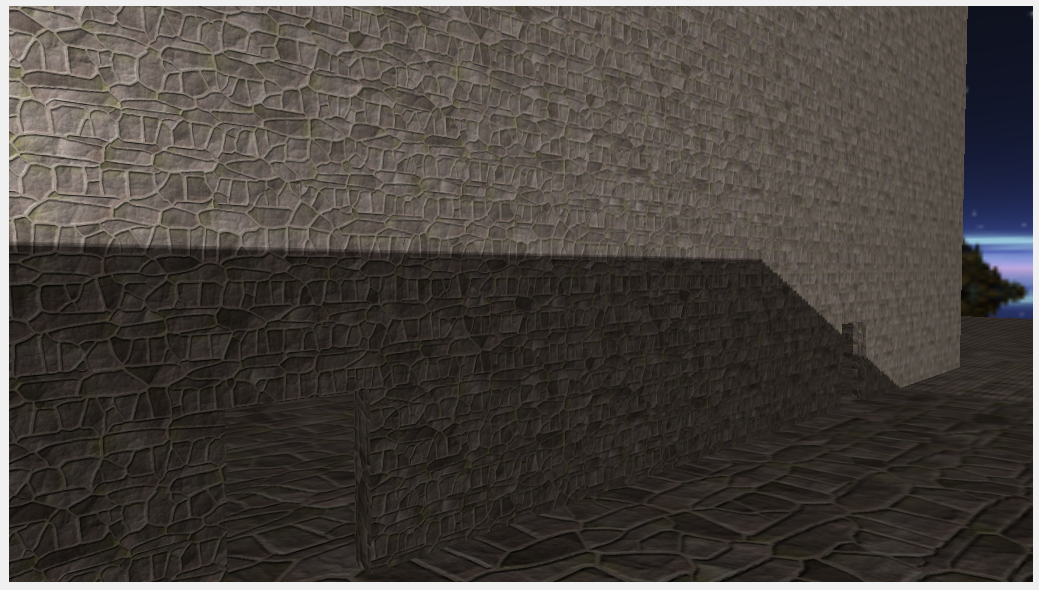
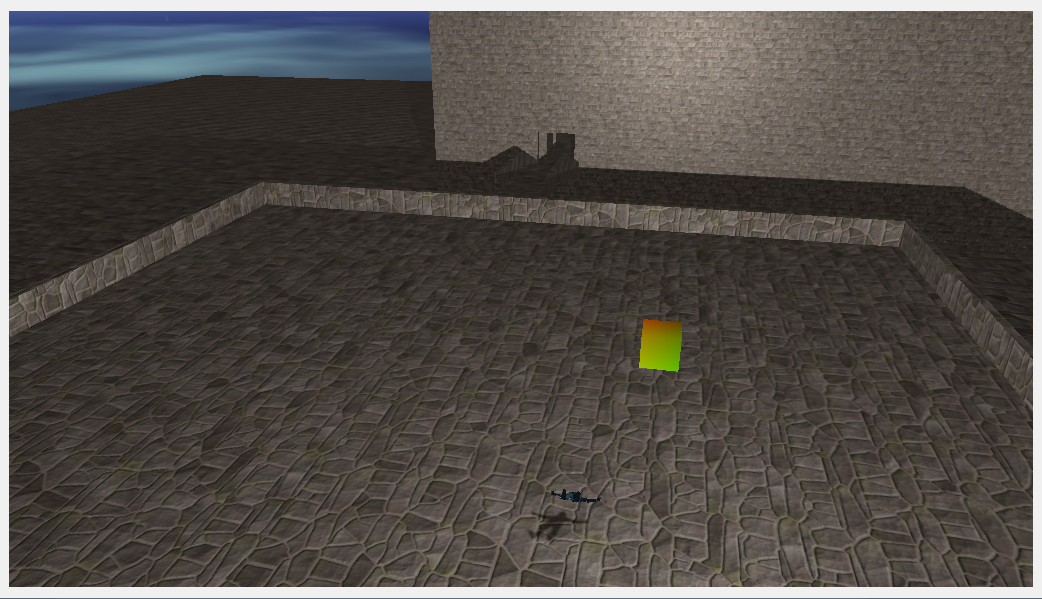
Lab 05 – Shadows– Joshua Kauer

For this lab I have made two shaders that make shadows. The first shader requires a frame buffer and runs a depth test from that light’s position in all six directions of the light to make sure the light is being projected omnidirectional for the shadow. I then use that shader and framebuffer to write to a cube map on how much light should be where in the scene. I then use the second shader to actually have the light be projected in a way to make it look like shadows are working. Also the projection of shadows will work with any object but projecting shadows onto an object will only work with objects that have the second shader.

P.S. The cube in the second picture is the light.





//Depth Test Shader

//Vertex

#version 430

layout (location = 0) in vec4 position;

uniform mat4 model;

void main()

{

gl\_Position = model \* position;

}

//Geometry

#version 430

layout (triangles) in;

layout (triangle\_strip, max\_vertices=18) out;

uniform mat4 shadowMatrices[6];

out vec4 FragPos; // FragPos from GS (output per emitvertex)

void main()

{

for(int face = 0; face < 6; ++face)

{

gl\_Layer = face; // built-in variable that specifies to which face we render.

for(int i = 0; i < 3; ++i) // for each triangle's vertices

{

FragPos = gl\_in[i].gl\_Position;

gl\_Position = shadowMatrices[face] \* FragPos;

EmitVertex();

}

EndPrimitive();

}

}

//Fragment

#version 430

in vec4 FragPos;

uniform vec3 lightPos;

uniform float far\_plane;

void main()

{

// get distance between fragment and light source

float lightDistance = length(FragPos.xyz - lightPos);

// map to [0;1] range by dividing by far\_plane

lightDistance = lightDistance / far\_plane;

// Write this as modified depth

gl\_FragDepth = lightDistance;

}

//Shadow Mapping Shader

//Vertex

#version 430

layout (location = 0) in vec4 position;

layout (location = 1) in vec2 texCoords;

layout (location = 2) in vec3 normal;

out VS\_OUT {

vec3 FragPos;

vec2 TexCoords;

vec3 Normal;

} vs\_out;

uniform mat4 projection;

uniform mat4 view;

uniform mat4 model;

void main()

{

gl\_Position = projection \* view \* model \* position;

vs\_out.FragPos = vec3(model \* position);

vs\_out.TexCoords = texCoords;

vs\_out.Normal = transpose(inverse(mat3(model))) \* normal;

}

//Fragment

#version 430

out vec4 FragColor;

in VS\_OUT {

vec3 FragPos;

vec2 TexCoords;

vec3 Normal;

} fs\_in;

uniform sampler2D diffuseTexture;

uniform samplerCube depthMap;

uniform vec3 lightPos;

uniform vec3 viewPos;

uniform float far\_plane;

// array of offset direction for sampling

vec3 gridSamplingDisk[20] = vec3[]

(

vec3(1, 1, 1), vec3(1, -1, 1), vec3(-1, -1, 1), vec3(-1, 1, 1),

vec3(1, 1, -1), vec3(1, -1, -1), vec3(-1, -1, -1), vec3(-1, 1, -1),

vec3(1, 1, 0), vec3(1, -1, 0), vec3(-1, -1, 0), vec3(-1, 1, 0),

vec3(1, 0, 1), vec3(-1, 0, 1), vec3(1, 0, -1), vec3(-1, 0, -1),

vec3(0, 1, 1), vec3(0, -1, 1), vec3(0, -1, -1), vec3(0, 1, -1)

);

float ShadowCalculation(vec3 fragPos)

{

// Get vector between fragment position and light position

vec3 fragToLight = fragPos - lightPos;

// Get current linear depth as the length between the fragment and light position

float currentDepth = length(fragToLight);

// Test for shadows with PCF

float shadow = 0.0;

float bias = 0.15;

int samples = 20;

float viewDistance = length(viewPos - fragPos);

float diskRadius = (1.0 + (viewDistance / far\_plane)) / 25.0;

for(int i = 0; i < samples; ++i)

{

float closestDepth = texture(depthMap, fragToLight + gridSamplingDisk[i] \* diskRadius).r;

closestDepth \*= far\_plane; // Undo mapping [0;1]

if(currentDepth - bias > closestDepth)

shadow += 1.0;

}

shadow /= float(samples);

// Display closestDepth as debug (to visualize depth cubemap)

// FragColor = vec4(vec3(closestDepth / far\_plane), 1.0);

// return shadow;

return shadow;

}

void main()

{

vec3 color = texture(diffuseTexture, fs\_in.TexCoords).rgb;

vec3 normal = normalize(fs\_in.Normal);

vec3 lightColor = vec3(0.3);

// Ambient

vec3 ambient = 0.3 \* color;

// Diffuse

vec3 lightDir = normalize(lightPos - fs\_in.FragPos);

float diff = max(dot(lightDir, normal), 0.0);

vec3 diffuse = diff \* lightColor;

// Specular

vec3 viewDir = normalize(viewPos - fs\_in.FragPos);

vec3 reflectDir = reflect(-lightDir, normal);

float spec = 0.0;

vec3 halfwayDir = normalize(lightDir + viewDir);

spec = pow(max(dot(normal, halfwayDir), 0.0), 64.0);

vec3 specular = spec \* lightColor;

// Calculate shadow

float shadow = ShadowCalculation(fs\_in.FragPos);

vec3 lighting = (ambient + (1.0 - shadow) \* (diffuse + specular)) \* color;

FragColor = vec4(lighting, 1.0f);

}