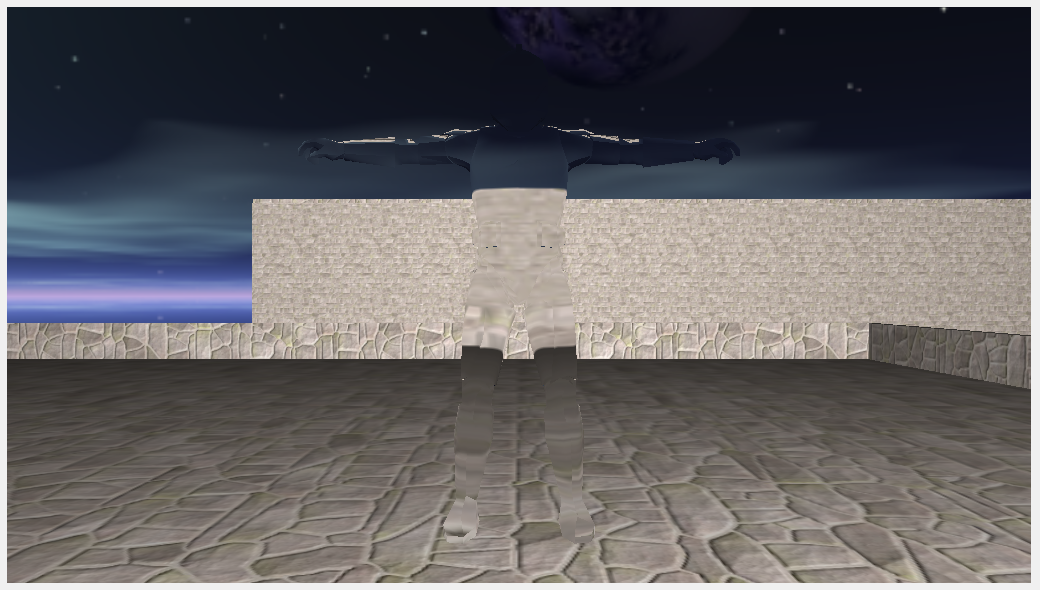
Lab 06 – Dynamic Environment Refraction – Joshua Kauer

In this lab I use a framebuffer to make a cubemap which I then map to an object that has the refraction shader making it look like it has a cloaking device on it. I do this in a very similar matter to making the shadow cube map, but instead of overriding the shader being used, I override the projection and the camera view. I do this six times from the objects position, with the camera facing a different direction for each one, afterwards I draw everything normally. One thing to consider is the cubemap’s faces can’t be any larger than the window’s size otherwise black bars appear near the bottom and top faces. The end result is an object that looks like it’s cloaked. Lastly, during my time trying to figure this out, I have discovered that this framebuffer doesn’t work if you have another framebuffer, I have yet to find a solution to fix this.

//Basic trooper without the shader so you know where to look for it in the next picture







//Framebuffer Code

glGenTextures(1, &refractCubeMap);

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, refractCubeMap);

for (GLuint j = 0; j < 6; ++j)

{

glTexImage2D(GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + j, 0, GL\_RGB, REFRACT\_WIDTH, REFRACT\_HEIGHT, 0, GL\_BGR, GL\_UNSIGNED\_BYTE, 0);

}

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_R, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glGenFramebuffers(1, &refractMapFBO);

glBindFramebuffer(GL\_FRAMEBUFFER, refractMapFBO);

//Optional? depth buffer

glGenRenderbuffers(1, &depthrenderbuffer);

glBindRenderbuffer(GL\_RENDERBUFFER, depthrenderbuffer);

glRenderbufferStorage(GL\_RENDERBUFFER, GL\_DEPTH\_COMPONENT, REFRACT\_WIDTH, REFRACT\_HEIGHT);

glBindRenderbuffer(GL\_RENDERBUFFER, 0);

//Attach it

glFramebufferRenderbuffer(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_RENDERBUFFER, refractMapFBO);

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X, refractCubeMap, 0);

if (glCheckFramebufferStatus(GL\_FRAMEBUFFER) != GL\_FRAMEBUFFER\_COMPLETE)

{

LOG(Error, "Framebuffer not complete!");

}

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

//Draw

glm::mat4 refractProj = glm::perspective(glm::radians(90.0f), (GLfloat)REFRACT\_WIDTH / (GLfloat)REFRACT\_HEIGHT, 0.1f, FarPlane);

refractTransforms[0] = (glm::lookAt(stealthPos, stealthPos + glm::vec3(1.0, 0.0, 0.0), glm::vec3(0.0, -1.0, 0.0)));

refractTransforms[1] = (glm::lookAt(stealthPos, stealthPos + glm::vec3(-1.0, 0.0, 0.0), glm::vec3(0.0, -1.0, 0.0)));

refractTransforms[2] = (glm::lookAt(stealthPos, stealthPos + glm::vec3(0.0, 1.0, 0.0), glm::vec3(0.0, 0.0, 1.0)));

refractTransforms[3] = (glm::lookAt(stealthPos, stealthPos + glm::vec3(0.0, -1.0, 0.0), glm::vec3(0.0, 0.0, -1.0)));

refractTransforms[4] = (glm::lookAt(stealthPos, stealthPos + glm::vec3(0.0, 0.0, 1.0), glm::vec3(0.0, -1.0, 0.0)));

refractTransforms[5] = (glm::lookAt(stealthPos, stealthPos + glm::vec3(0.0, 0.0, -1.0), glm::vec3(0.0, -1.0, 0.0)));

glBindFramebuffer(GL\_FRAMEBUFFER, refractMapFBO);

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

{

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + j, refractCubeMap, 0);

RenderEngine::Draw(&refractProj[0][0], &refractTransforms[j][0][0]);

}

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

TextureInfo texture;

texture = render[5].getTextureInfo();

texture.textureID = refractCubeMap;

texture.isCubeMap = true;

render[5].setTextureInfo(texture);

//Refract/Reflect Shader

#version 430 core

layout (location = 0) in vec4 position;

layout (location = 1) in vec3 normal;

out vec3 Normal;

out vec3 Position;

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

void main()

{

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

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

Position = vec3(model \* position);

}

#version 430 core

in vec3 Normal;

in vec3 Position;

out vec4 color;

uniform vec3 cameraPos;

uniform samplerCube skybox;

//Reflect

//void main()

//{

// vec3 I = normalize(Position - cameraPos);

// vec3 R = reflect(I, normalize(Normal));

// color = texture(skybox, R);

//}

//Refract

void main()

{

float ratio = 1.00 / 1.01;

vec3 I = normalize(Position - cameraPos);

vec3 R = refract(I, normalize(Normal), ratio);

color = texture(skybox, R);

}