Textures

COMP3015 Lab 4

Single Texture

Setup

Before coding, download the ‘Additional files’ folder and unzip it. Inside, add the ‘stb_image.cpp’ file to your project’s helper/stb folder. Next, within the project’s ‘media’ folder, add a new folder called ‘texture’ & add the ‘Additional files’ folder’s texture & cube header & cpp files to it respectively. Lastly, add the stb_image.cpp, texture & cube files to the Visual Studio project’s solution.

Vertex Shader

The vertex shader will continue to make use of the Blinn-Phong model. The only addition necessary are the vertex texture coordinates. Add an input variable named vec2 indexed at location 2 called ‘VertexTexCoord’ & an out variable of type vec2 named ‘TexCoord.’ In the main function, set TexCoord’s value to VertexTexCoord:

```
layout (location = 2) in vec2 VertexTexCoord;

//In main:

TexCoord = VertexTexCoord;
```

Fragment Shader

Add an in variable of type vec2 named ‘TexCoord’ & a uniform layout with a binding index of 0 of type sampler2D. The sampler2D type is used for 2-dimensional textures.

```
in vec2 TexCoord;

layout(binding=0) uniform sampler2D Tex1;
```

In the Blinn-Phong function, acquire the colour for each fragment & set it to a vec3 variable named ‘texColor:’

```
vec3 texColor = texture(Tex1, TexCoord).rgb;
```

Lastly, calculate the ambient & diffuse with the use of the texColor variable.

Scenebasic Uniform Header

Add a variable of type Cube called ‘cube.’

Scenebasic Uniform CPP

Init Scene Function

Set the view and projection:

```
compile();

glEnable(GL_DEPTH_TEST);

view = glm::lookAt(vec3(1.0f, 1.25f, 1.25f), vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));

projection = mat4(1.0f);
```

Set your light uniforms & load the texture file:

```
GLuint texID = Texture::loadTexture("../Project_Template/media/texture/brick1.jpg");

glActiveTexture(GL_TEXTURE0);

glBindTexture(GL_TEXTURE_2D, texID);
```

Render Function

Set the ‘Light.Position,’ ‘Material.Ks’ and ‘Material.Shininess’ uniforms. Notably, don’t forget to call render on the cube.

Multiple Textures

Vertex Shader

When implementing multiple textures, the vertex shader is unaffected. Use the vertex shader implementation used prior for singular textures.

Fragment Shader

Use the same fragment shader used for singular textures, but add another uniform of type sampler2D. Make sure you set its layout index to 1. Now there are 2 textures present in the fragment shader:

```
layout(binding=0) uniform sampler2D BrickTex;

layout(binding=1) uniform sampler2D MossTex;
```

In the Blinn-Phong function, extract the colour for each texture, similarly to how this was done for one texture previously. When doing so, combine the colours with the GLSL mix function, using the alpha channel of the ‘mossTexColor’ as an interpolator:

```
vec3 col = mix(brickTexColor.rgb, mossTexColor.rgb, mossTexColor.a);
```

The rest of the shader stays the same; just use col for any light calculations.

Scenebasic Uniform

In the scenebasic_uniform files, we need to make use of 2 textures in the initScene() function. Bind them to GL_TEXTURE 0 & 1 respectively:

```
// Load brick texture file into channel 0
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, brick);


// Load moss texture file into channel 1
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, moss);
```

Alpha Test (Discard)

Vertex Shader

The vertex shader remains the same as implemented for multiple textures.

Fragment Shader

The same fragment shader implemented for multiple textures is necessary, however it needs some changes. Bind 2 textures named ‘BaseTex’ & ‘AlphaTex.’ Next, unpack the ‘texColour’ variable for the base texture in the Blinn Phong function & do all the light calculations with it:

```
vec3 texColor = texture(BaseTex, TexCoord).rgb;
```

In the main function, keep or discard fragments based on an alpha value with the use of ‘alphaTexture.’ First, unpack ‘AlphaTex’ into a new variable named ‘alphaMap’ of type vec4. Then, use an if statement to check for any value under 0.15. If any value is below 0.15, the fragment is discarded, otherwise it is rendered:

```
if(alphaMap.a < 0.15 )

discard;

 else

{

        FragColor = vec4( blinnPhong(Position,normalize(Normal)), 1.0 );

}
```

.This works on face value, however some issues do arise. The back of the fragments that point away from the light are in the dark. Therefore, lighting must be calculated for them as well. Use gl_FrontFacing with an if statement to check for this. Anything found to be front facing should have its lighting calculated in the same way as just implemented. Anything found to be back facing must use the same formula, but with the normal reversed.

Scenebasic Uniform

Use a teapot model for the object that is rendered, since it is easier to see the effect that has been applied with the teapot:

```
teapot(14, glm::mat4(1.0f))
```

Normal Mapping

Vertex Shader

The vertex shader is similar to the implementation used in the previous implementations. First, add two new layout variables named ‘vertexNormal’ & ‘vertexTangent’ at the location indexes of 2 & 3 respectively. The existing ‘VertexPosition’ & ‘VertexNormal’ variables should be indexed at locations 0 & 1 respectively:

```
layout (location = 0) in vec3 VertexPosition;

layout (location = 1) in vec3 VertexNormal;

layout (location = 2) in vec2 VertexTexCoord;

layout (location = 3) in vec4 VertexTangent;
```

Next, define a light uniform struct and all the uniforms for the matrices used in transformations. Use the previous examples of the vertex shaders as references. We will need 3 out variables named ‘LightDirection,’ ‘TexCoord’ & ‘ViewDir:’

```
out vec3 LightDir;

out vec2 TexCoord;

out vec3 ViewDir;
```

In the main() function, transform normal and tangent to eye space:

```
// Transform normal and tangent to eye space

vec3 norm = normalize( NormalMatrix * VertexNormal );

vec3 tang = normalize( NormalMatrix * vec3(VertexTangent) );
```

Compute the binormal:

```
// Compute the binormal

vec3 binormal = normalize( cross( norm, tang ) ) * VertexTangent.w;
```

Set the matrix for transformation to tangent space:

```
// Matrix for transformation to tangent space

mat3 toObjectLocal = mat3(

        tang.x, binormal.x, norm.x,

        tang.y, binormal.y, norm.y,

        tang.z, binormal.z, norm.z

);
```

Transform the light direction & view direction to tangent space:

```
vec3 pos = vec3( ModelViewMatrix * vec4(VertexPosition,1.0) );

LightDir = toObjectLocal * (Light.Position.xyz - pos);

ViewDir = toObjectLocal * normalize(-pos);
```

Lastly, pass the value of ‘VertexTexCoord’ to ‘TexCoord’ so the fragment shader can use it for its calculations. Notably, don’t forget to set ‘gl_position’ for the next stage.

Fragment Shader

3 in vectors are needed, which will be called ‘LightDir’, ‘TexCoord’ & ‘ViewDir:’

```
in vec3 LightDir;

in vec2 TexCoord;

in vec3 ViewDir;
```

Declare 2 uniforms for textures named ‘ColorTex’ & ‘NormalMapTex:’

```
layout(binding=0) uniform sampler2D ColorTex;

layout(binding=1) uniform sampler2D NormalMapTex;
```

Ensure that your material & light uniform struct is declared & that your out vector is ‘FragColor.’ Then, in the main function unpack the normal & set it to a range between 0 & 1. Note that normals are sometimes within the range of -1 to 1 instead:

```
vec3 norm = texture(NormalMapTex, TexCoord).xyz;

norm.xy = 2.0 * norm.xy - 1.0;
```

Pass the result of the Blinn-Phong function to FragColor. In the Blinn-Phong function itself, the standard Blinn-Phong model is calculated, however the light & view direction calculations can be omitted. This is because they have already been determined in the vertex shader.

Scenebasic Uniform Header

The scenebasic_uniform header file is similar to that of the previous files, however this time the model of an ogre head will be used. In order to use it, add the ‘bs_ears’ file to your project’s media folder. Next, add an include for the ‘objmesh.h’ file so the ogre model’s can be created as type ‘ObjMesh.’ Lastly, create a private object variable as shown below:

```
std::unique_ptr<ObjMesh> ogre
```

Scenebasic Uniform CPP

Load the ogre model in the constructor:

```
ogre = ObjMesh::load("../Project_Template/media/bs_ears.obj", false, true);
```

In the initScene() function, set the view to the following:

```
view = glm::lookAt(vec3(-1.0f, 0.25f, 2.0f), vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));
```

Set the uniforms for ‘Light’ & ‘material,’ & load & bind the textures for diffuse (“ogre_diffuse.png”) & normal (“ogre_normalmap.png”). Lastly, render the model in the render function.

Skybox with Cube Maps

Vertex Shader

Create a layout variable indexed at location 0 of type vec3 named ‘VertexPosition:’

```
layout (location = 0) in vec3 VertexPosition;
```

Create an out variable of type vec3 named ‘Vec’ for passing ‘VertexPosition’ to the fragment shader. The MVP uniform is also needed, as it is added to gl_Position. In the main function, assign the ‘VertexPosition’ to ‘Vec’ & multiply ‘MVP’ with ‘vec4(VertexPosition, 1.0).’

Fragment Shader

Create a layout uniform at the binding index of 0 of type samplerCube named ‘SkyBoxTex:’

```
layout(binding=0) uniform samplerCube SkyBoxTex;
```

Create a vec3 input variable named ‘Vec,’ & an out variable of type vec4 named ‘FragColor.’ Use previous examples as references for how to declare ‘FragColour.’ Then, in the main function extract the colour from the texture:

```
vec3 texColor = texture(SkyBoxTex, normalize(Vec)).rgb;
```

Lastly, assign ‘texColor’ to ‘FragColor.’

Scenebasic Uniform Header

Add the skybox header & cpp files to your project’s solution & create a variable named ‘sky’ of type ‘SkyBox:’

```
SkyBox sky;
```

Create 3 new private object variables of type float named ‘angle,’ ‘tPrev’ & ‘rotSpeed:’

```
float angle, tPrev, rotSpeed;
```

Scenebasic Uniform CPP

Constructor

```
SceneBasic_Uniform::SceneBasic_Uniform() : angle(0.0f), tPrev(0.0f), rotSpeed(glm::pi<float>() / 8.0f),

sky(100.0f)

{

        //

}
```

Init Scene Function

```
compile();

glEnable(GL_DEPTH_TEST);

projection = mat4(1.0f);

angle = glm::radians(90.0f); //set the initial angle

//extract the cube texture

GLuint cubeTex = Texture::loadHdrCubeMap("../Project_Template/media/texture/cube/pisa-hdr/pisa");

//activate and bindtexture

glActiveTexture(GL_TEXTURE0);

glBindTexture(GL_TEXTURE_CUBE_MAP, cubeTex);
```

Update Function

```
float deltaT = t - tPrev;

if (tPrev == 0.0f)

        deltaT = 0.0f;

tPrev = t;

angle += rotSpeed * deltaT;

if (angle > glm::two_pi<float>())

        angle -= glm::two_pi<float>();
```

Render Function

```
vec3 cameraPos = vec3(7.0f * cos(angle), 2.0f, 7.0f * sin(angle));

view = glm::lookAt(cameraPos, vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));

// Draw sky

prog.use();

model = mat4(1.0f);

setMatrices();

sky.render();
```

Additional Materials

Projected Textures

Vertex Shader

Inputs

Create two input variables of type vec3 named ‘VertexPosition’ & ‘VertexNormal’ at layout indexes 0 & 1 respectively.

Outputs

Additionally, create 3 out variables, one of type vec3 named ‘EyeNormal’ & two of type vec4 named ‘EyePosition’ & ‘ProjTexCoord.’

Uniforms

Create 5 uniforms, one of type mat3 named ‘NormalMatrix’ & the rest of type mat4, named ‘ProjectorMatrix,’ ‘ModelViewMatrix,’ ‘ModelMatrix’ & ‘MVP.’

Main Function

Alter ‘vertexPosition’ to be a vec4:

```
vec4 pos4 = vec4(VertexPosition,1.0);
```

Turn your normals from model coordinates into eye coordinates & apply the result to EyeNormal:

```
EyeNormal = normalize(NormalMatrix * VertexNormal);
```

Turn your pos4 from model coordinates into eye coordinates & pass the result to EyePosition. Notably, no normalisation is needed for this. Next, acquire the projected texture coordinates:

```
ProjTexCoord = ProjectorMatrix * (ModelMatrix * pos4);
```

Lastly, multiply gl_Position with ‘MVP.’

Fragment Shader

Inputs & Outputs

Create 3 input variables, two of type vec3 named ‘EyeNormal’ & ‘EyePosition’ & one of type vec4 named ‘ProjTexCoord.’ Also create one out variable named ‘FragColor.’

Uniforms

Create two uniform structs named ‘Light’ & ‘Material.’ The ‘Light’ struct needs to contain ‘Position,’ ‘L’ for diffuse & specular lighting & ‘La’ for ambient lighting. The ‘Material’ struct needs to contain ‘Ka,’ ‘Kd,’ ‘Ks, & ‘Shininess.’ One last uniform for the projected texture is also required, as shown below:

```
layout(binding=0) uniform sampler2D ProjectorTex;
```

Blinn-Phong Function

Note that a Blinn-Phong function is necessary for calculation of colour. If one does not currently exist in your project, implement it.

Main Function

Create a variable of type vec3 named ‘color’ & pass the result of the Blinn-Phong function to it:

```
vec3 color = blinnPhong(normalize(EyePosition.xyz), EyeNormal);
```

Create a variable of type vec3 named ‘projTexColour’ & use an if statement to check as to whether its z value is positive. If z is positive, the location is in front of the projector & therefore we proceed with the texture look up:

```
vec3 projTexColor = vec3(0.0);

if( ProjTexCoord.z > 0.0 )

        projTexColor = textureProj( ProjectorTex, ProjTexCoord ).rgb;
```

Apply the colours of both the model’s corresponding pixel & projected texture to FragColor:

```
FragColor = vec4(color + projTexColor * 0.5, 1);
```

7. That’s it with the fragment shader

Scenebasic Uniform Header

Use both a teapot & plane model. Additionally, if it is desired that the camera is to rotate, make use of the ‘angle,’ ‘tPrev’ & ‘rotSpeed’ floats in order to achieve this.

Scenebasic Uniform CPP

Constructor

Use the constructor to initialise all the variables that were declared earlier:

```
SceneBasic_Uniform::SceneBasic_Uniform(): angle(0.0f), tPrev(0.0f),

rotSpeed(glm::pi<float>() / 8.0f),

teapot(14, mat4(1.0f)),

plane(100.0f, 100.0f, 1, 1)

{

        //

}
```

Init Scene Function

Setup the projector matrix:

```
compile();

glEnable(GL_DEPTH_TEST);

projection = mat4(1.0f);

angle = glm::radians(90.0f);

//set up things for the projector matrix

vec3 projPos = vec3(2.0f, 5.0f, 5.0f);

vec3 projAt = vec3(-2.0f, -4.0f, 0.0f);

vec3 projUp = vec3(0.0f, 1.0f, 0.0f);

mat4 projView = glm::lookAt(projPos, projAt, projUp);

mat4 projProj = glm::perspective(glm::radians(30.0f), 1.0f, 0.2f, 1000.0f);

mat4 bias = glm::translate(mat4(1.0f), vec3(0.5f));

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

prog.setUniform("ProjectorMatrix", bias * projProj * projView);
```

Load and set up the projected texture:

```
// Load texture file
GLuint flowerTex = Texture::loadTexture("../Project_Template/media/texture/flower.png");
//set up and send the projected texture
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, flowerTex);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
```

Setup the light uniform:

```
prog.setUniform("Light.Position", glm::vec4(0.0f, 0.0f, 0.0f, 1.0f));
```

Update Function

If you want to rotate the camera, add the following:

```
float deltaT = t - tPrev;
if (tPrev == 0.0f)
        deltaT = 0.0f;
tPrev = t;
angle += rotSpeed * deltaT;
if (angle > glm::two_pi<float>())
        angle -= glm::two_pi<float>();
```

Render Function

Setup the camera:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
//set up your camera
vec3 cameraPos = vec3(7.0f * cos(angle), 2.0f, 7.0f * sin(angle));
view = glm::lookAt(cameraPos, vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));
```

Setup the material struct's uniforms:

```
//set up your material uniforms
prog.setUniform("Material.Kd", 0.5f, 0.2f, 0.1f);
prog.setUniform("Material.Ks", 0.95f, 0.95f, 0.95f);
prog.setUniform("Material.Ka", 0.1f, 0.1f, 0.1f);
prog.setUniform("Material.Shininess", 100.0f);
```

Setup & render the teapot:

```
//set up and render the teapot
model = mat4(1.0f);
model = glm::translate(model, vec3(0.0f, -1.0f, 0.0f));
model = glm::rotate(model, glm::radians(-90.0f), vec3(1.0f, 0.0f, 0.0f));
setMatrices();
teapot.render();
```

Setup the material struct's plane related uniforms & render the plane:

```
//set up the material uniforms for plane and render the plane
prog.setUniform("Material.Kd", 0.4f, 0.4f, 0.4f);
prog.setUniform("Material.Ks", 0.0f, 0.0f, 0.0f);
prog.setUniform("Material.Ka", 0.1f, 0.1f, 0.1f);
prog.setUniform("Material.Shininess", 1.0f);
model = mat4(1.0f);
model = glm::translate(model, vec3(0.0f, -0.75f, 0.0f));
setMatrices();
plane.render();
```

Set Matrices Function

Set your model matrix uniform:

```
mat4 mv = view * model;
prog.setUniform("ModelMatrix", model);
prog.setUniform("ModelViewMatrix", mv);
prog.setUniform("NormalMatrix", glm::mat3(vec3(mv[0]), vec3(mv[1]), vec3(mv[2])));
prog.setUniform("MVP", projection * mv);
```

Rendering to a Texture

Implementation

For a reference of an implementation of rendering to a texture, see the relevant lecture recording.

Setup & Scenebasic Uniform Header

Add the ‘spot’ folder from within the ‘Additional files’ folder to your project’s ‘media’ folder. First add “spot” folder from Additional files to “media” folder. Next, load the spot object, add a Gluint fboHandle & create the methods ‘setupFBO(),’ ‘renderToTexture()’ & ‘renderScene().’ Lastly, clear the teapot & plane from the constructor. The only variables used in the constructor that are required are ‘angle,’ ‘tPrev’ & ‘rotSpeed.’

Scenebasic Uniform CPP

Init Scene Function

```
void SceneBasic_Uniform::initScene()

{

        compile();

        glEnable(GL_DEPTH_TEST);

        //load the texture and the model

        GLuint spotTexture = Texture::loadTexture("media/spot/spot_texture.png");

        spot = ObjMesh::load("media/spot/spot_triangulated.obj");

        projection = mat4(1.0f);

        angle = glm::radians(140.0f);

        prog.setUniform("Light.L", vec3(1.0f));

        prog.setUniform("Light.La", vec3(0.15f));

        setupFBO(); //we call the setup for our fbo

        glActiveTexture(GL_TEXTURE1);

        glBindTexture(GL_TEXTURE_2D, spotTexture);

}
```

FBO Setup Function

Framebuffer & Texture Object

```
// Generate and bind the framebuffer
glGenFramebuffers(1, &fboHandle);
glBindFramebuffer(GL_FRAMEBUFFER, fboHandle);
// Create the texture object
GLuint renderTex;
glGenTextures(1, &renderTex);
glActiveTexture(GL_TEXTURE0); // Use texture unit 0
glBindTexture(GL_TEXTURE_2D, renderTex);
glTexStorage2D(GL_TEXTURE_2D, 1, GL_RGBA8, 512, 512);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
// Bind the texture to the FBO
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, renderTex, 0);
```

Depth Buffer & Fragment Output

```
// Create the depth buffer
GLuint depthBuf;
glGenRenderbuffers(1, &depthBuf);
glBindRenderbuffer(GL_RENDERBUFFER, depthBuf);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT, 512, 512);
// Bind the depth buffer to the FBO
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
GL_RENDERBUFFER, depthBuf);
// Set the targets for the fragment output variables
GLenum drawBuffers[] = { GL_COLOR_ATTACHMENT0 };
glDrawBuffers(1, drawBuffers);
GLenum result = glCheckFramebufferStatus(GL_FRAMEBUFFER);
if (result == GL_FRAMEBUFFER_COMPLETE) {
        cout << "Framebuffer is complete" << endl;
}
else {
        cout << "Framebuffer error: " << result << endl;
}
// Unbind the framebuffer, and revert to default framebuffer
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Render Function

```
void SceneBasic_Uniform::render()
{
    //bind the buffer
    glBindFramebuffer(GL_FRAMEBUFFER, fboHandle);
    //render to texture
    renderToTexture();
    //flush the buffer
    glFlush();
    //unbind the write buffer and bind the default buffer
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
    //render the scene using the newly written texture
    renderScene();
}
```

Render to Texture Function

```
void SceneBasic_Uniform::renderToTexture() {
    prog.setUniform("RenderTex", 1);
    glViewport(0, 0, 512, 512);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    view = glm::lookAt(vec3(0.0f, 0.0f, 2.5f), vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));
    projection = glm::perspective(glm::radians(50.0f), 1.0f, 0.3f, 100.0f);
    prog.setUniform("Light.Position", glm::vec4(0.0f, 0.0f, 0.0f, 1.0f));
    prog.setUniform("Material.Ks", 0.95f, 0.95f, 0.95f);
    prog.setUniform("Material.Shininess", 100.0f);
    model = mat4(1.0f);
    model = glm::rotate(model, angle, vec3(0.0f, 1.0f, 0.0f));
    setMatrices();
    spot->render();
}
```

Render Scene Function

```
void SceneBasic_Uniform::renderScene() {
        prog.setUniform("RenderTex", 0);
        glViewport(0, 0, width, height);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        vec3 cameraPos = vec3(2.0f * cos(angle), 1.5f, 2.0f * sin(angle));
        view = glm::lookAt(cameraPos, vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));
        projection = glm::perspective(glm::radians(45.0f), (float)width / height, 0.3f, 100.0f);
        prog.setUniform("Light.Position", glm::vec4(0.0f, 0.0f, 0.0f, 1.0f));
        prog.setUniform("Material.Ks", 0.0f, 0.0f, 0.0f);
        prog.setUniform("Material.Shininess", 1.0f);
        model = mat4(1.0f);
        setMatrices();
        cube.render();
}
```