Brock Barlow
ID #1113
Assessment ADGP 201 - Graphics

Graphics Assessment Documentation for “Textures”

Purpose:
Introduce the steps needed to apply a texture to a plane and apply uv mapping.

Learning Outcomes:
1) UV mapping.
2) Texture applied to plane.

Evidence:
This project has the ability to apply a texture to a plane. In the app.cpp file under the start function, three variables with a value of zero is declared: imageWidth, imageHeight and imageFormat. A unsigned char data variable is declared using stbi_load. The texture is then generated and binded. In the draw function, a int loc variable is declared. This variable gets the uniform location. The texture becomes active and is bond. In the fragment shader, in the main function: the fragColor equals the texture value, which is the diffuse and vTexCoord.

```
int imageWidth = 0, imageHeight = 0, imageFormat = 0;

unsigned char* data = stbi_load("data/textures/crate.png", &imageWidth, &imageHeight, &imageFormat, STBI_default);
glGenTextures(1, &m_texture);
glBindTexture(GL_TEXTURE_2D, m_texture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imageWidth, imageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
stbi_image_free(data);
```

```
int loc = glGetUniformLocation(m_programID, "ProjectionView");
glUniformMatrix4fv(loc, 1, GL_FALSE, glm::value_ptr(m_projectionViewMatrix));
```

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, m_texture);
```

```
loc = glGetUniformLocation(m_programID, "diffuse");
glUniform1i(loc, 0);
```

*Pictures of app.cpp

```
#version 410

layout(location=0) in vec4 Position;
layout(location=1) in vec2 TexCoord;
out vec2 vTexCoord;
uniform mat4 ProjectionView;
void main()
{
        vTexCoord = TexCoord;
        gl_Position= ProjectionView * Position;
}
```

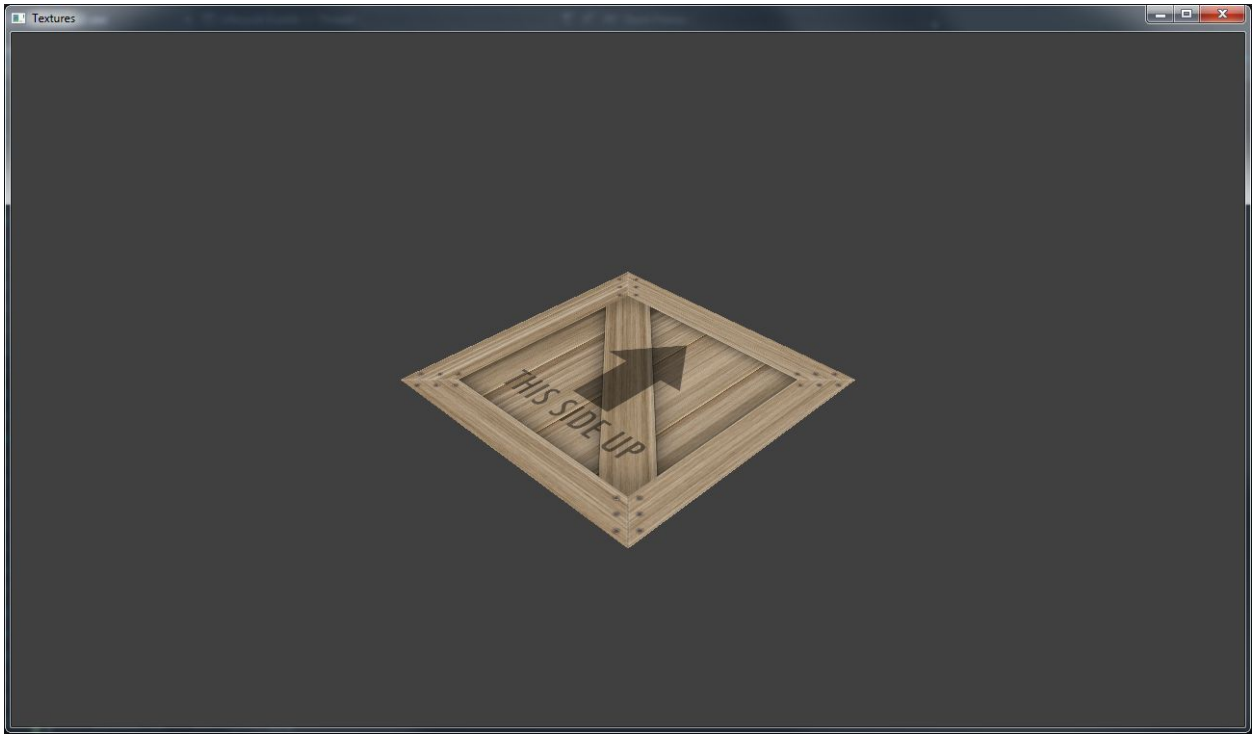
```
#version 410

in vec2 vTexCoord;
out vec4 FragColor;
uniform sampler2D diffuse;
uniform sampler2D white;

void main() {
        FragColor = texture(diffuse, vTexCoord) * texture(white, vTexCoord);
}
```

*Pictures of the shader files

This is the result:



This project has the ability to apply a uv map onto a 3d object. In the app.cpp file under the start function, three variables with a value of zero is declared: imageWidth, imageHeight and imageFormat. A unsigned char data variable is declared using stbi_load. The texture is then generated and binded. This is done twice: once for the diffuse and one for the normal. A new fbx file is created in this function as well. In the draw function, a int loc variable is declared. This variable gets the uniform location. The texture becomes active and is bond. A for loop is used for a fbx mesh node. The createOpenGLBuffers function sets up the fbx variable. The createData function sets up the vertices and indices and is called in the start function. The vertex shader sets up the vTexCoord, vNormal, vTangent, and vBiTangent. In the fragment shader, in the main function: the fragColor equals the texture value, which is the diffuse and vTexCoord.

```
createData();

m_fbx = new FBXFile();
m_fbx->load("data/FBXmodels/stanford/Dragon.fbx");
createOpenGLBuffers(m_fbx);

int imageWidth = 0, imageHeight = 0, imageFormat = 0;

unsigned char* data;

data = stbi_load("data/textures/rock_diffuse.tga", &imageWidth, &imageHeight, &imageFormat, STBI_default);
glGenTextures(1, &m_texture);
glBindTexture(GL_TEXTURE_2D, m_texture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imageWidth, imageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
stbi_image_free(data);

data = stbi_load("data/textures/rock_normal.tga", &imageWidth, &imageHeight, &imageFormat, STBI_default);
glGenTextures(1, &m_normal);
glBindTexture(GL_TEXTURE_2D, m_normal);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imageWidth, imageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
stbi_image_free(data);
```

```
xtures::createOpenGLBuffers(FBXFile* fbx)

(unsigned int i = 0; i < fbx->getMeshCount(); ++i)

FBXMeshNode* mesh = fbx->getMeshByIndex(i);
unsigned int* glData = new unsigned int[3];

glGenVertexArrays(1, &glData[0]);
glBindVertexArray(glData[0]);
glGenBuffers(1, &glData[1]);
glGenBuffers(1, &glData[2]);
glBindBuffer(GL_ARRAY_BUFFER, glData[1]);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, glData[2]);
glBufferData(GL_ARRAY_BUFFER, mesh->m_vertices.size() * sizeof(FBXVertex), mesh->m_vertices.data(), GL_STATIC_DRAW);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, mesh->m_indices.size() * sizeof(unsigned int), mesh->m_indices.data(), GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, sizeof(FBXVertex), 0);
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 4, GL_FLOAT, GL_TRUE, sizeof(FBXVertex), ((char*)0) + FBXVertex::NormalOffset);
glBindVertexArray(0);
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
mesh->m_userData = glData;
```

```
Textures
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glUseProgram(m_programID);

    int loc = glGetUniformLocation(m_programID, "ProjectionView");
    glUniformMatrix4fv(loc, 1, GL_FALSE, &(myCamera.getProjectionView()[0][0]));

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, m_texture);

    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, m_normal);

    loc = glGetUniformLocation(m_programID, "diffuse");
    glUniform1i(loc, 0);

    loc = glGetUniformLocation(m_programID, "normal");
    glUniform1i(loc, 1);

    glm::vec3 lightAD(sin(glfwGetTime()), 1, cos(glfwGetTime()));
    loc = glGetUniformLocation(m_programID, "LightDir");
    glUniform3f(loc, lightAD.x, lightAD.y, lightAD.z);

    glBindVertexArray(m_VAO);
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, nullptr);
```

```
Textures                                    draw()

for (unsigned int i = 0; i < m_fbx->getMeshCount(); ++i) {
    FBXMeshNode* mesh = m_fbx->getMeshByIndex(i);
    unsigned int* glData = (unsigned int*)mesh->m_userData;
    glBindVertexArray(glData[0]);
    glDrawElements(GL_TRIANGLES, (unsigned int)mesh->m_indices.size(), GL_UNSIGNED_INT, 0);
}

glBindVertexArray(m_VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, nullptr);

myCamera.update(delta, window);

glfwSwapBuffers(window);
glfwPollEvents();
```

```
void Textures::createData()
{
    Vertex vertexData[] = {
        { -5, 0, 5, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1 },
        { 5, 0, 5, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1 },
        { 5, 0, -5, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0 },
        { -5, 0, -5, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0 },
    };
    unsigned int indexData[] = {
        0, 1, 2,
        0, 2, 3,
    };
    glGenVertexArrays(1, &m_VAO);
    glBindVertexArray(m_VAO);

    glGenBuffers(1, &m_VBO);
    glBindBuffer(GL_ARRAY_BUFFER, m_VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertex) * 4, vertexData, GL_STATIC_DRAW);

    glGenBuffers(1, &m_IBO);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_IBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(unsigned int) * 6, indexData, GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, sizeof(Vertex), 0);
```

```
    glGenVertexArrays(1, &m_VAO);
    glBindVertexArray(m_VAO);

    glGenBuffers(1, &m_VBO);
    glBindBuffer(GL_ARRAY_BUFFER, m_VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertex) * 4, vertexData, GL_STATIC_DRAW);

    glGenBuffers(1, &m_IBO);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_IBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(unsigned int) * 6, indexData, GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, sizeof(Vertex), 0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), ((char*)0) + 48);
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 4, GL_FLOAT, GL_FALSE, sizeof(Vertex), ((char*)0) + 16);
    glEnableVertexAttribArray(3);
    glVertexAttribPointer(3, 4, GL_FLOAT, GL_FALSE, sizeof(Vertex), ((char*)0) + 32);
    glBindVertexArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
}
```

*Pictures of app.cpp

nfo.txt vsAdvanceInfo.txt Textures.cpp

```
#version 410

layout(location=0) in vec4 Position;
layout(location=1) in vec2 TexCoord;
layout(location=2) in vec4 Normal;
layout(location=3) in vec4 Tangent;
out vec2 vTexCoord;
out vec3 vNormal;
out vec3 vTangent;
out vec3 vBiTangent;
uniform mat4 ProjectionView;
void main()
{
    vTexCoord = TexCoord;
    vNormal = Normal.xyz;
    vTangent = Tangent.xyz;
    vBiTangent = cross(vNormal, vTangent);
    gl_Position = ProjectionView * Position;
}
```

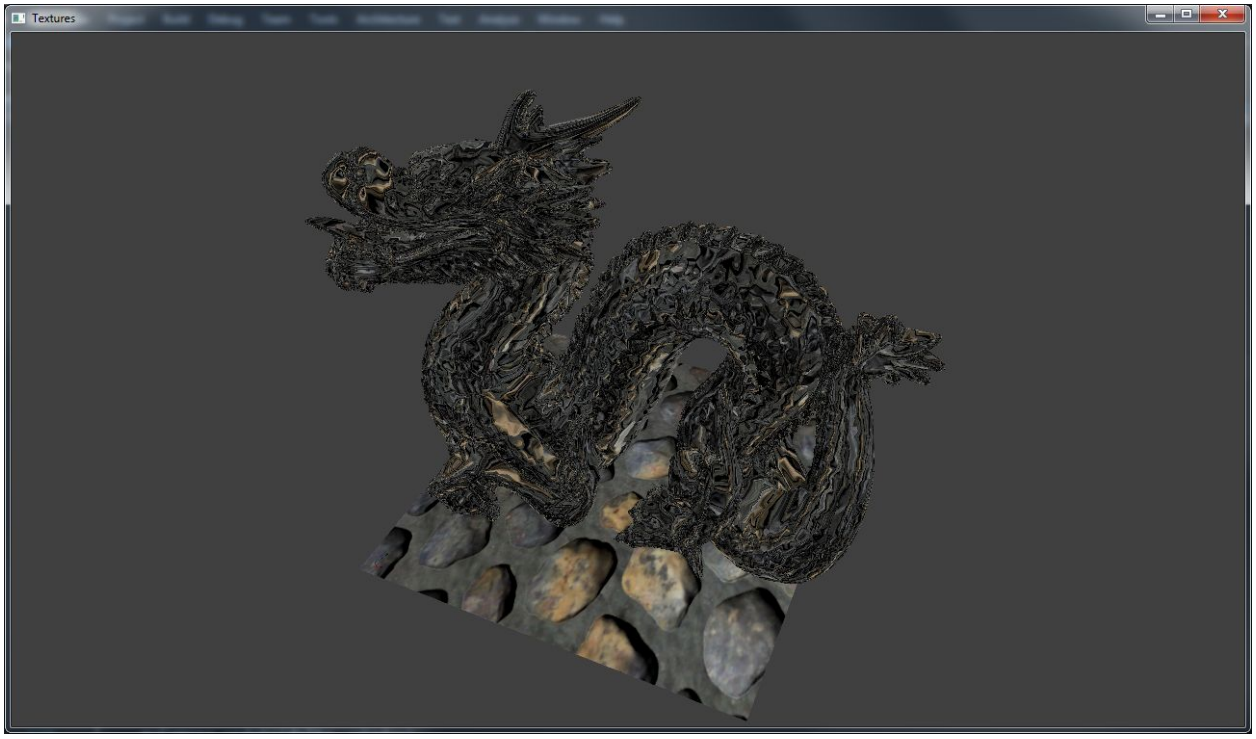
fo.txt vsAdvanceInfo.txt Textures.cpp

```
#version 410

in vec2 vTexCoord;
in vec3 vNormal;
in vec3 vTangent;
in vec3 vBiTangent;
out vec4 FragColor;
uniform vec3 LightDir;
uniform sampler2D diffuse;
uniform sampler2D normal;
void main()
{
    mat3 TBN = mat3(normalize( vTangent ), normalize( vBiTangent ), normalize( vNormal ));
    vec3 N = texture(normal, vTexCoord).xyz * 2 - 1;
    float d = max( 0, dot( normalize( TBN * N ), normalize( LightDir )));
    FragColor = texture(diffuse, vTexCoord);
    FragColor.rgb = FragColor.rgb * d;
}
```

*Picture of the shader files

This is the result:


Textures