Computer Graphics HW2

110550085 房天越

1. Implementation

I. Create Shader

```
unsigned int createShader(const string& filename, const string& type)
{
    unsigned int shader;
    if (type == "vert") {
        shader = glCreateShader(GL_VERTEX_SHADER);
    }
    else {
        shader = glCreateShader(GL_FRAGMENT_SHADER);
    }

    ifstream shaderFile(filename.c_str());
    stringstream shaderStream;
    shaderStream << shaderFile.rdbuf();
    shaderFile.close();
    string shaderCode = shaderStream.str();
    const char* shaderSource = shaderCode.c_str();

    glShaderSource(shader, 1, &shaderSource, NULL);
    glCompileShader(shader);

    return shader;
}
```

In this function, I create the shader according to the given type, and open the corresponding vertex file, then return it.

II. Create Program

```
unsigned int createProgram(unsigned int vertexShader, unsigned int fragmentShader)
{
    unsigned int shaderProgram = glCreateProgram();

    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);

    glLinkProgram(shaderProgram);

    glDetachShader(shaderProgram, vertexShader);
    glDetachShader(shaderProgram, fragmentShader);

    return shaderProgram;

}
```

In this function, I create the shader program by linking the vertex shader and the fragment shader.

III. Load Texture

```cpp
unsigned int loadTexture(const char* tFileName) {
    unsigned int texture;

    glEnable(GL_TEXTURE_2D);
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);


    int w, h, nrchannels;
    stbi_set_flip_vertically_on_load(true);
    unsigned char* data = stbi_load(tFileName, &w, &h, &nrchannels, 0);
    if (data) {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
        //glGenerateMipmap(GL_TEXTURE_2D);
    }
    else {
        cerr << "Load Texture Failed" << endl;
    }

    glBindTexture(GL_TEXTURE_2D, 0);
    stbi_image_free(data);
    return texture;
}
```

In this function, I load the texture of the given name, and return that to the caller.

IV. Setting VAO and VBO

```cpp
unsigned int modelVAO(Object& model)
{
    unsigned int VAO, VBO[3];

    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    glGenBuffers(3, VBO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT) * model.positions.size(), &(model.positions[0]), GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GL_FLOAT), 0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    glBindBuffer(GL_ARRAY_BUFFER, VBO[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT) * model.normals.size(), &(model.normals[0]), GL_STATIC_DRAW);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GL_FLOAT), 0);
    glEnableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    glBindBuffer(GL_ARRAY_BUFFER, VBO[2]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT) * model.texcoords.size(), &(model.texcoords[0]), GL_STATIC_DRAW);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(GL_FLOAT), 0);
    glEnableVertexAttribArray(2);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    glBindVertexArray(0);

    return VAO;

}
```

In this function, I set the VAO of a model by using the functions in the hint, the implementation of this function is inspired by (?) HW1.

V. Data Connection

```cpp
int modelLoc = glGetUniformLocation(shaderProgram, "M");
int viewLoc = glGetUniformLocation(shaderProgram, "V");
int projectionLoc = glGetUniformLocation(shaderProgram, "P");

int squeezeFactorLoc = glGetUniformLocation(shaderProgram, "squeezeFactor");
int useGrayscaleLoc = glGetUniformLocation(shaderProgram, "useGrayscale");
```

Here I connect the variables defined in the vertex shader and fragment shader to the main code by calling glGetUniformLocation function.

VI. Rendering the Board and the Penguin

```
glm::mat4 boardModelMatrix = glm::mat4(1.0f);


boardModelMatrix = glm::rotate(boardModelMatrix, glm::radians(swingAngle), glm::vec3(0.0f, 1.0f, 0.0f));
boardModelMatrix = glm::translate(boardModelMatrix, glm::vec3(0.0f, -0.5f, swingPos));
boardModelMatrix = glm::rotate(boardModelMatrix, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
boardModelMatrix = glm::scale(boardModelMatrix, glm::vec3(0.03f, 0.03f, 0.03f));



glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(boardModelMatrix));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(perspective));
glUniform1f(squeezeFactorLoc, 0);
glUniform1i(useGrayscaleLoc, useGrayscale);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, boardTexture);
glUniform1i(glGetUniformLocation(shaderProgram, "ourTexture"), 0);

glBindVertexArray(boardVAO);

glDrawArrays(GL_TRIANGLES, 0, boardModel.positions.size());
glBindVertexArray(0);
```

```
glm::mat4 penguinModelMatrix = glm::mat4(1.0f);

penguinModelMatrix = glm::rotate(penguinModelMatrix, glm::radians(swingAngle), glm::vec3(0.0f, 1.0f, 0.0f));
penguinModelMatrix = glm::translate(penguinModelMatrix, glm::vec3(0.0f, 0.0f, swingPos));
penguinModelMatrix = glm::rotate(penguinModelMatrix, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
penguinModelMatrix = glm::scale(penguinModelMatrix, glm::vec3(0.025f, 0.025f, 0.025f));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(penguinModelMatrix));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(perspective));
glUniform1f(squeezeFactorLoc, squeezeFactor);
glUniform1i(useGrayscaleLoc, useGrayscale);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, penguinTexture);
glUniform1i(glGetUniformLocation(shaderProgram, "ourTexture"), 0);

glBindVertexArray(penguinVAO);

glDrawArrays(GL_TRIANGLES, 0, penguinModel.positions.size());

glBindVertexArray(0);
```

Here is the part that I render the board and the penguin, first, I construct the matrix, do translation, rotation, and scaling like the method in HW1. Then, for the following part, I take the functions defined in drawModel in HW1 here to do rendering, also, this is done by checking the squeezeFactor and useGrayscale. They are needed according to the spec.

VII. Adjust the parameters

```
if (isAnglePlus) {
    swingAngle += 20.0f * static_cast<float>(dt);
    if (swingAngle>20) {
        isAnglePlus = !isAnglePlus;
    }
}
else {
    swingAngle -= 20.0f * static_cast<float>(dt);
    if (swingAngle < -20) {
        isAnglePlus = !isAnglePlus;
    }
}

if (isPlus) {
    swingPos += 1.0f * static_cast<float>(dt);
    if (swingPos > 2) {
        isPlus = !isPlus;
    }
}
else {
    swingPos -= 1.0f * static_cast<float>(dt);
    if (swingPos < 0) {
        isPlus = !isPlus;
    }
}
if (squeezing) {
    squeezeFactor += glm::radians(90.0f) * static_cast<float>(dt);
}
```

Here I adjust the parameters according to the demands in spec.

VIII. Vertex Shader and Fragment Shader

```
vec3 squeezedPos = aPos;
squeezedPos.y += aPos.z * sin(squeezeFactor) / 2.0;  // Squeeze effect for y coordinate
squeezedPos.z += aPos.y * sin(squeezeFactor) / 2.0;  // Squeeze effect for z coordinate
//squeezedPos.y *= squeezeFactor;
//squeezedPos.z *= squeezeFactor;

worldPos = M * vec4(squeezedPos, 1);

gl_Position = P * V * worldPos;
mat4 normal_transform = transpose(inverse(M));
normal = normalize((normal_transform * vec4(aNormal, 0.0)).xyz);

texCoord = aTexCoord;
```

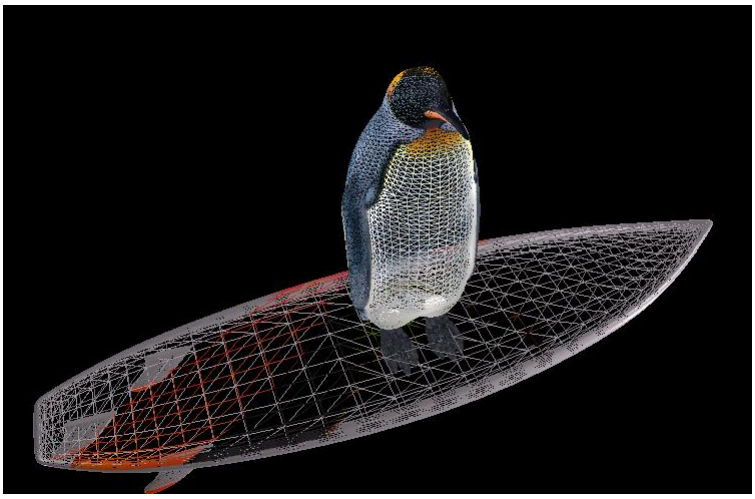
```
vec4 color = texture(ourTexture, texCoord);
if(useGrayscale){
    float grayscale = dot(color.rgb,vec3(0.299, 0.587, 0.114));
    FragColor = vec4(grayscale, grayscale, grayscale, color.a);
}
else{
    FragColor = color;
}
```

Here are the vertex shader code and the fragment shader code.
The vertex shader is affected by squeezing, if squeezing is true, then do the actions according to the spec.
The fragment shader is affected by using grayscale, if useGrayscale is true, then do inner dot between color.rgb, and the given vector.

IX. Special Effect!

This part, I designed a special effect by **pressing W**, this would

set the rendering mode from filling to wireframe, and you can press again to switch back. The effect is like this:



Here are the related parts of code:

```
if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
    wireframeMode = !wireframeMode;
    glPolygonMode(GL_FRONT_AND_BACK, wireframeMode ? GL_LINE : GL_FILL);
    cout << "KEY W PRESSED - Wireframe: " << (wireframeMode ? "On" : "Off") << endl;
}
```

When W is pressed, change the mode from GL_FILL to GL_LINE, and press again to switch back.

2. **Problems that I met and solutions:**

   I find it difficult to understand the way to implement the vertex shader and fragment shader, I didn't understand how to do that at first, but later I found that I could find reference in HW1, then it went on better.

   Another problem that I met is that I did not know how to implement the squeezing effect, at first I tried to do that in the main code, but it failed, then after asking my friends, I finally know that I should do it in the vertex shader.

   Overall, this assignment really took me some time to implement that, but I got to know about the shader codes better, the time I spent was not in vain!