Department of Computer Science
Technical University of Cluj-Napoca

DOCUMENTATION

GP OPENGL PROJECT

GP LABORATORY

CURAC MIHAI – IONUȚ
30431

CONTENTS

1. SUBJECT SPECIFICATION

The subject of the project consists in the photorealistic presentation of 3D objects using the OpenGL library. OpenGL is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. Applications use it extensively in the fields of computer-aided design (CAD), virtual reality, scientific visualization, information visualization, flight simulation, and video games.

I chose to make a countryside scene in a mountainous area surrounded by trees. A road goes through the centre of this area, where there can be found a house and a fishing place as well. The user directly manipulates the scene of objects by mouse and keyboard inputs.

2. SCENARIO

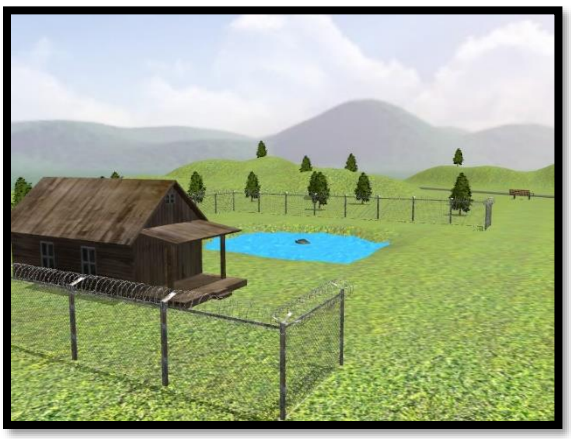
2.1. SCENE AND OBJECTS DESCRIPTION

The scene is set in a village from a mountainous area, with trees that surround the road which passes through the centre of the village. Near the road there are 2 pairs of opposite wooden benches, a house in which a family lives, and a fishing place.

A wooden fence encloses the family house’s yard on all 4 sides. In the backyard we find two big trees located in the corners. On one side of the yard there are two outdoor dining tables and an outdoor swing. All of them have wooden textures. On the other side of the yard there is a little playground with a yellow slide and a white and black soccer ball on top of it. In front of the yard, there is a metallic street light pole that can be turned on and off whenever more light is desired in the scene.

Opposite the house, on the other side of the road, there is a fishing place consisting of a wooden cottage where fishermen or tourists can stay, and a lake that a fish is happily swimming in. The fishing area is surrounded by a tall and secure metal fence. Moreover, on each of the road’s two endings there is a barrier announcing that we can’t get further.

Finally, I chose to implement a skybox with a cloudy blue sky and other mountains in the far distance, that look similarly to the close ones in my scene.



2.2. FUNCTIONALITIES

The users can view and manipulate the scene in multiple ways with the use of the mouse and several keyboard inputs. They can move the camera freely in any direction they want using the mouse and the keyboard, or they can press a key to enter a preview mode, where the camera is animated to show a panoramic, 360 degree visualization of the scene. They can also rotate in two directions the entire scene with all the objects in it.

The scene can also be viewed with an additional fog effect, that can be trigged via the keyboard. There are also keys for visualizing the scene in solid, point, or wireframe view.

I implemented a directional light that can be rotated by the user in two directions, and a point light situated in the street light pole that can be turned on and off at any time using the keyboard as well.

There are two objects that can be animated by the user using the keyboard: the ball on top of the slide and the fish in the lake. The ball can be moved up and down the slide at any time, and the fish can be moved in 4 directions: right, left, front, back.

3. IMPLEMENTATION DETAILS

3.1. FUNCTIONS AND SPECIAL ALGORITHMS

3.1.1. POSSIBLE SOLUTIONS

There are various functions and algorithms that can help in implementing the lights, the effects, the shadows, and the animation of the camera and objects.

For the user input, like the key bindings and the mouse movement, I used the *keyboardCallback(GLFWwindow* window, int key, int scancode, int action, int mode)*, *mouse_callback(GLFWwindow* glWindow, double xpos, double ypos)*, and *void processMovement()* functions. The keyboard functions check whether a key has been pressed by the user with the help of GLFW_PRESS and GLFW_RELEASE.

Below is the code from *keyboardCallback* for the initialization of the fog effect using the "F" and "G" keys:

```
// start fog
if (key == GLFW_KEY_F && action == GLFW_PRESS) {

        myCustomShader.useShaderProgram();
        foginit = 1;
        foginitLoc = glGetUniformLocation(myCustomShader.shaderProgram, "foginit");
        glUniform1i(foginitLoc, foginit);
}

// stop fog
if (key == GLFW_KEY_G && action == GLFW_PRESS) {
        myCustomShader.useShaderProgram();
        foginit = 0;
        foginitLoc = glGetUniformLocation(myCustomShader.shaderProgram, "foginit");
        glUniform1i(foginitLoc, foginit);


}
```

The *processMovement* deals specifically with the keys that increment and decrement values such as the directional light angle, the scene rotation angle, the camera movement, the fish movement animation, and the soccer ball animation. The code to move the camera forward, backward, right, and left is written below:

```
        if (pressedKeys[GLFW_KEY_W]) {
                myCamera.move(gps::MOVE_FORWARD, cameraSpeed);
        }

        if (pressedKeys[GLFW_KEY_S]) {
                myCamera.move(gps::MOVE_BACKWARD, cameraSpeed);
        }

        if (pressedKeys[GLFW_KEY_A]) {
                myCamera.move(gps::MOVE_RIGHT, cameraSpeed);
        }

        if (pressedKeys[GLFW_KEY_D]) {
                myCamera.move(gps::MOVE_LEFT, cameraSpeed);
        }
```

Next up, the *initObjects* function loads the object models (.obj files) from their specified locations using LoadModel. *initShaders* does the same for the shaders (.vert and .frag files), using loadShader. The *computeLightSpaceTrMatrix* returns the light-space transformation matrix after specifying parameters such as the values for the near plane, far plane, and orthographic projection to avoid any perspective deforming.

In the *initUniforms* function, I also specified the light direction, light colour, and point light position, using glGetUniformLocation and glUniform3fv.

The *drawObjects(gps::Shader shader, bool depthPass)* function is very important, since it is run two times using different shaders and it draws the objects in the scene. Here is also where I do the scaling, translation, and rotation operations on the objects in the scene, in order to move and rotate them when the user presses the appropriate keys.

Below is the code for drawing the ball and the fish objects (that are also animated):

```
shader.useShaderProgram();
        model = glm::rotate(glm::mat4(1.0f), glm::radians(angleY), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(0.0f, -delta/1.5, -delta));
        glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE,
glm::value_ptr(model));
        // do not send the normal matrix if we are rendering in the depth map

        if (!depthPass) {
                normalMatrix = glm::mat3(glm::inverseTranspose(view * model));
                glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));
        }

        ball.Draw(shader);

        shader.useShaderProgram();
        model = glm::rotate(glm::mat4(1.0f), glm::radians(angleY), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(moveX, 0.0f, moveZ));
        glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE,
glm::value_ptr(model));

        if (!depthPass) {
                normalMatrix = glm::mat3(glm::inverseTranspose(view * model));
                glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));
        }

        fish.Draw(shader);
```

The *renderScene* function is also very important. Here, we call *drawObjects* firstly with the depthPass parameter set on “true”, so with the depthMapShader parameter as well. Next, I either render the depth map on screen (when the user toggles it with the “M” key), or I do the final rendering pass with shadows, by binding the shadow map and the fog density, and then calling *drawObjects* with myCustomShader and the depthPass parameter set to “false”.

Additionally, in order to properly set up the point light from the street pole, I added the *computePointLight(vec4 lightPosEye)* function in shaderStart.frag, that returns the sum of the point light calculated parameters. This sum is added to the overall lighting of the scene in the main function. The fog colour is mixed with the colorWithShadow parameter in this function as well.

3.1.2. THE MOTIVATION OF THE CHOSEN APPROACH

I chose this approach when implementing the project since it was derived from the knowledge acquired over the semester at the GP laboratories, where we worked with light sources, shadows, camera movement, animation, skyboxes, and learnt how to integrate them into our projects. This proved to be a solid foundation when working on the project and helped me better understand how to tackle the other requirements, such as detailed modelling and effects.

3.2. GRAPHICS MODEL

The objects, textures and skybox were downloaded from the Internet, mostly from the website https://free3d.com/. I imported the objects in the Blender software after being done with the ground planes (mountains, water). Some of the objects’ components needed to be separated by texture material into different smaller objects, since they weren’t displayed correctly. I had to edit most of the textures for the objects, in order to add elements such as a better texture for the house roof. I also scaled the textures to dimensions that were powers of 2 and converted most of them to the .png extension. I made sure that the .obj, .mtl, and texture files were written correctly and added them to the Visual Studio project. The objects that can be animated were added separately from the full scene, as well as objects that I needed to reposition constantly, such as the light pole.

3.3. DATA STRUCTURES

I used arrays in the project, such as *bool pressedKeys[1024]* for specifying whether a key was pressed or released by the user. I also used many of the OpenGL specific data types, such as GLuint, GLfloat, GLchar, and GLenum. Most of them were already provided in the project skeleton that we received. I added new variables using them to accommodate more features and parameters, such as GLint foginitLoc, GLfloat fogDensity = 0.01f (for the fog effect), GLuint lightPosLoc (for the point light).

3.4. CLASS HIERARCHY

The class hierarchy for the project is the following:

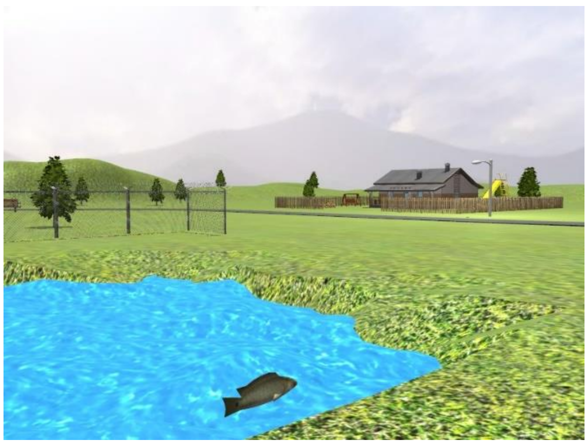
- Camera – contains the camera constructor, the update function for internal parameters following a camera move event, the update function for the camera internal parameters following a camera rotate event, and the *scenePreview* function;
- Mesh – contains the mesh constructor, the Mesh drawing function, and the function that initializes all the buffer objects/arrays;
- Model3D – contains the *LoadModel* functions, the function that draws each mesh from the model, the function that does the parsing of the .obj file and fills in the data structure, the function that retrieves a texture associated with the object, and the function that reads the pixel data from an image file and loads it into the video memory;
- Shader – contains the *readShaderFile* function, the *loadShader* function, and the compilation and linking info functions;
- SkyBox – contains the *Load* function, the *Draw* function, the *LoadSkyBoxTextures* function, the *InitSkyBox* function, and the *GetTextureId* function.

4. GRAPHICAL USER INTERFACE PRESENTATION / USER MANUAL

As previously stated, the users can visualize and manipulate the scene in multiple ways with the use of the mouse and several keyboard inputs. They are all presented in the following guiding table:

ESC	Close the application
1	Wireframe view of the scene
2	Point view of the scene
3	Normal view of the scene
5	Camera preview animation
8	Start point light
9	Stop point light
F	Start fog effect
G	Stop fog effect
M	Toggle depth map view
Q	Rotate entire scene to the left
E	Rotate entire scene to the right
J	Rotate directional light left
L	Rotate directional light right
W	Move camera forward
A	Move camera to the left
S	Move camera backward
D	Move camera to the right
Z	Move ball up the slide
X	Move ball down the slide
UP	Move fish forward
LEFT	Move fish to the left

RIGHT	Move fish to the right
DOWN	Move fish backward



5. CONCLUSIONS AND FURTHER DEVELOPMENTS

This project was a great opportunity to get hands-on experience with the presentation of 3D objects using the OpenGL API, since I have never worked with such complex visual systems before. It was also interesting to learn how to use Blender and what its uses are, even though I felt that the default shortcuts and mouse controls were a bit clunky. Modifying object texture files to fit the project's specifications was time consuming, as well.

The project can still be improved by adding, for example, more effects, like rain and wind. Further developments could also be the addition of more light sources, like a spotlight, and more objects to the scene.

6. REFERENCES

- https://free3d.com/
- https://learnopengl.com/Lighting/Light-casters
- https://open.gl/transformations
- https://www.turbosquid.com/
- http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/
- The laboratory materials provided on Moodle.