Graphical Processing Systems
Project Documentation

Rad Lidia

Group 30434

3rd year

Computer Science

Assistant Teacher: Oros Bogdan

Table of Content

1. Subject specification

The purpose of this project is to build a photorealistic representation of 3D objects using the OpenGL library. The user can directly manipulate the scene of objects using the keyboard or the mouse. The scene must contain minimum two light sources, which also allow for shadow computation, multiple objects with applied textures with the possibility of viewing solid, wireframe, polygonal and smooth surfaces. It should also display a certain level of photo-realism, including fog/rain/wind, object generation and collision detection. This project should make use of different algorithms.

2. Scenario

I chose to implement a view from a camping point. The user manipulates the object scene directly by entering the mouse and keyboard and can move around it.

2.1 Scene and object description

The scene contains one tent with a fire near. Right in front of these there is a river in which there is couple of ducks swimming and four rocks three forming a bridge to cross the river. The path goes to a little house with a bench near it.

2.2 Functionalities

The scene has two modes, the normal mode, which means that the lighting is provided by the directional source, and the fog mode. In the last one, the skybox changes to a foggy light and the light source loses its brightness.

All objects, except for the bees, cast shadows on the ground.

3. Implementation details

3.1 Functions and special algorithms

The program has one main loop (inside the main function), which calls the rendering functions: *renderScene()* which with the use of DrawObjects generate the whole scene.

First functionality of the project that can be observed is the camera movement: The camera can be controlled using keys from the keyboard. The camera also creates a preview of the scene by rotating around it when the key ‘1’ is being pressed.

The lighting of the scene comes from two sources: a directional light which acts like a global light, over the whole scene, and a light coming from far away.

Shadow computation is implemented from the global (directional) light source.

Another functionality is viewing the wireframe objects by pressing the ‘G’ key and disabled by pressing the ‘H’ key. Also, antialiasing is implemented when the ‘P’ key is pressed.

The textures are mapped to their object components with the help of the .mtl files attached to each object file. All the objects in the scene are correctly textured, no objects remain untextured.

Also, fog appears or disappear by the choice of the user. The fog is implemented over the scene but also over the skybox so that the passing from the ground to the skybox would be seamless.

3.1.1 Possible solutions

For **fog computation** I used the square exponential formula, which computes the fog color (a value between 0.0f and 1.0f) taking into account the current position of the camera.

float fogDensity = 0.05f;

float fragmentDistance = length(fragPosEye);

*float fogFactor = exp(-pow(fragmentDistance*fogDensity, 2));*

return clamp(fogFactor, 0.0f, 1.0f);

The final colour of the object is given by: *fColor = fogColor*(1 - fogFactor) + vec4(color, opacity)*fogFactor;*

The **direct light computation** is done with respect to the light direction, which is sent to the shader via a uniform.

The final colour of the object is given by the three coefficients: ambient (the universal light), diffuse (which depends on the direction of the light) and specular (the reflection, which depends on the light direction and the viewer’s position). These three values are computed using the following formulas:

*vec3 normalEye = normalize(normalMatrix * normal);*

*vec3 lightDirN = normalize(lightDirMatrix * lightDir);*

vec3 viewDirN = normalize(cameraPosEye - fragPosEye.xyz);

vec3 halfVector = normalize(lightDirN + viewDirN);

*ambient = ambientStrength * lightColor;*

*ambient *= vec3(texture(diffuseTexture, fragTexCoords));*

*diffuse = max(dot(normalEye, lightDirN), 0.0f) * lightColor;*

*diffuse *= vec3(texture(diffuseTexture, fragTexCoords));*

float specCoeff = pow(max(dot(halfVector, normalEye), 0.0f), shininess);

*specular = specularStrength * specCoeff * lightColor;*

*specular *= vec3(texture(specularTexture, fragTexCoords));*

The preview of the scene is done by changing the lookAt matrix of the camera. Instead of moving on the given coordinates, it will rotate at a given radius around the scene. In order to be the same speed on every computer, the glfwGetTime() function is used.

The shadow computation is done by using a depth buffer. Inside the shader, the shadow value is computed using

float bias = max(0.05(1.0-dot(normal, lightDir)), 0.01);*

float shadow = currentDepth - bias> closestDepth ? 1.0f : 0.0f;

3.1.2 The motivation of the chosen approach

3.2 Data structures

This project uses four shaders, the shaderStart for all computations regarding the light/shadow and fog, the skyBoxShader for placing the skybox in the scene, the lightCube shader for the objects that represent light sources and the depthMapShader for shadow computation. The data is sent to these shaders via uniforms:

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

*normalMatrix = glm::mat3(glm::inverseTranspose(view * model));*

glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));

or FBOs (using the *initFBOs* function).

3.3 Class hierarchy

The project has 6 classes, one of which being the main class.

Camera class contains all the data needed to model the camera, i.e. *cameraPosition, cameraTarget*, *cameraDirection* and *cameraRightDirection*. This class contains the functions move and rotate responsible for the movement of the user around the scene. The movement is performed by adding/subtracting from the current camera position the value obtained by the multiplication *cameraDirection*cameraSpeed.* For left and right step, the vector with which the speed is multiplied is *cameraRightDirection*. The *cameraSpeed* coefficient represents the number of positions the camera moves in one step. The rotation is performed by modifying the direction towards which the camera faces using the two angles: pitch and yaw. The final direction of the camera is given by the following three equations:

*cameraDirection.x = cos(glm::radians(pitch)) * cos(glm::radians(yaw));*

cameraDirection.y = sin(glm::radians(pitch));

*cameraDirection.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));*

Model3D class contains the methods necessary for correctly reading an object from the input file. This is done by the methods: *ReadOBJ*, *LoadTexture* and *ReadTextureFromFile*. The method *Draws* uses the shader given as parameters and draws the object. Another four important methods are get *getMinCoords, getMaxCoords, updateMinCoords, updateMaxCoords.* These methods are used in collision detection, the minimum and maximum coordinates are computed during the reading of the object. The two update functions take a model matrix as argument and multiply the vertices containing the min and max values, obtaining the new values.

SkyBox class contains all the necessary functions for loading and drawing a skybox.

Shader class contains all the necessary functions for loading and using a shader.

Mesh class contains the functions for setting up and drawing meshes.

The main class contains the functions for model/ shader parameters initializations as well as for the user movement around the scene.

4. Graphical user interface presentation/user manual

The user can move around the scene using the keys: E (rotate camera to the right), Q (rotate camera to the right, W (move camera forward), S (move camera backward), A (move camera left), D (move camera right).

L – rotate the directional light source;

F– enable fog mode;

G– disable fog mode;

P – enable/disable wireframe view;

1 – rotate around the scene

5. Conclusions and further developments

A development would be the addition of more objects to the scene. Also, the shadows could be improved (the coefficient values should be changed to accommodate the current scene). The movement of object components could also be improved.

6. References

The GPS laboratories

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-10-transparency/

https://learnopengl.com/Lighting/Multiple-lights

The objects/skyboxes were taken from:

https://sketchfab.com/feed

https://free3d.com/3d-models/

https://www.cgtrader.com/