Graphics Processing Project
A walk in a desert

Bardi Bogdan, Group 30431/1

Contents

1 Subject specification

This project will implement a small cottage placed in a desert near an oasis which is patrolled by a helicopter.

2 Scenario

2.1 Scene

The scene starts with a large desert which has more elements placed on it. It also has a helicopter in the sky which patrols the area. The desert plane also has a few decorations such as a wooden house and a small oasis with a small lake and some palm trees next to it. The scene can also display a sandstorm scenario where the viewing distance is significantly reduced.

2.2 Functionalities

The project has a few functionalities such as:

- a semi-realistic water effect for the lake
- 2 light sources(a white light for the scene and a yellowish one for inside the house)
- animations for the helicopter with moving helicopter blades
- the aforementioned sandstorm.

3 Implementation Details

3.1 Functions and algorithms

3.1.1 Sandstorm

The sandstorm was implemented using a fog effect with a yellowish tint inside the fragment shader. This approach was chosen due to ease of implementation and almost 0 performance penalty.

3.1.2 Animation

The animation speed varies based on the render speed, as such the animation does not skip around when framerate dips.

```
void updateDelta(double elapsedSeconds)
{
    deltaMov = movementSpeed * elapsedSeconds * 500;
    deltaAngle = angularSpeed * elapsedSeconds * 300;
}
...
(inside the helicopter rendering)
double currentTimeStamp = glfwGetTime();
```

```
updateDelta(currentTimeStamp - lastTimeStamp);
lastTimeStamp = currentTimeStamp;
```

The helicopter is composed of two models, the helicopter body and the helicopter main blade. This allows the helicopter blades to have their own model matrix which can then be rotated based on the rotation angle independently of the helicopter body's movements. This approach was the easiest to implement while it required separating the models and ensure that they are placed correctly relative to each other.

3.1.3 Water effect

To achieve the realistic water effect with a refraction and reflection effect, it was required to create 2 FBOs and render the scene 2 more times: one with everything below the water for the refraction and one with everything above the water. The two textures are then mixed together in the final render pass in the water texture to give the effect.

In order to improve performance clip planes where used in order to avoid rendering anything above the water for the refraction and anything below the water for the reflection.

For the refraction texture the same Camera properties are used as the usual render. However the reflection texture involved inverting the camera pitch and moving the camera below the water at the same level as the refraction.

3.2 Data structures

3.2.1 Camera

This class handles the scene camera, which can be rotated using Euler coordinates(pitch and yaw) and can be moved in 5 directions(forward,backward,left, right, down(used only for the water reflection)). The class can generate the view matrix which can be used for all the transformations necessary in computer graphics.

3.2.2 Shader

This class handles the OpenGL shaders. This implements methods for loading fragment and vertex shaders and allows enabling them to be used in further graphics calls.

3.2.3 Window

This class holds the window information such as its size and allows creating GLFW windows which can be used to display the output of the GPU.

3.2.4 Mesh

This class holds the information about a specific mesh such as coordinates, textures and material properties like ambient, specular and diffuse properties. It also holds the OpenGL specific information about it such as the Vertex Array Object, Vertex Buffer Object and Element Buffer Object. It is used internally by the 3D model class.

3.2.5 3D Model

This class handles the loading and rendering of 3D Models. It loads the model from .obj files and it's corresponding materials from its .mtl file, and translates them into a vector of Meshes which can be rendered.

3.2.6 SkyBox

This class handles the loading and rendering of a skybox.

3.3 Shaders

This project uses a number of shaders in order to achieve the desired results.

3.3.1 Skybox shader

This shader is designed to render the skybox as a box with the vertices infinitely far and to texture it according to the loaded images from the skybox class. This implements the fog effect, it uses an uniform boolean which can be used to override the texture color with the yellow color of the fog.

3.3.2 Water

This shader is responsible with rendering a plane as the water surface and is responsible with merging the reflection and refraction textures and mapping them to the surface.

3.3.3 Main

This shader implements the main program and does most of the heavy lifting in the rendering such as:

- Setting the gl_clipdistance to set the clipping plane
- Setting the vertex positions
- Computing the fog
- Computing the light values for the 2 light sources

The fog is straight forward to implement in the shader, the fog factor can be computed in this manner and then the final color of the fragment is then blended with the color fog based on the factor.

```
float computeFog()
{
    float fogDensity = 0.05f;
    float fragmentDistance = length(view * model * vec4(fPosition, 1.0f));
    float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));

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

The lights are computed using this algorithm and then the color is determined based on the specular and ambient textures, which are then forwarded to be blended with the fog and then send to be rendered.

```
void computeDirLight()
{
    //compute eye space coordinates
    vec4 fPosEye = view * model * vec4(fPosition, 1.0f);
    vec3 normalEye = normalize(normalMatrix * fNormal);

    //normalize light direction
    vec3 lightDirN = vec3(normalize(view * vec4(lightDir, 0.0f)));

    //compute view direction (in eye coordinates, the viewer is situated at the origin
    vec3 viewDir = normalize(- fPosEye.xyz);

    //compute ambient light
    ambient = ambientStrength * lightColor;

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

    //compute specular light
    vec3 reflectDir = reflect(-lightDirN, normalEye);
    float specCoeff = pow(max(dot(viewDir, reflectDir), 0.0f), 32);
    specular = specularStrength * specCoeff * lightColor;
}
```

4 User Manual

When the user starts the app he is shown a window which shows him the scene. Using the mouse he can look around the scene and using the WASD keys he can move forward, left, backward, right in the direction he is looking.

He can also press the F key which will toggle the sandstorm and significantly reduce the viewing distance with a yellowish fog.

4.1 Program screenshots

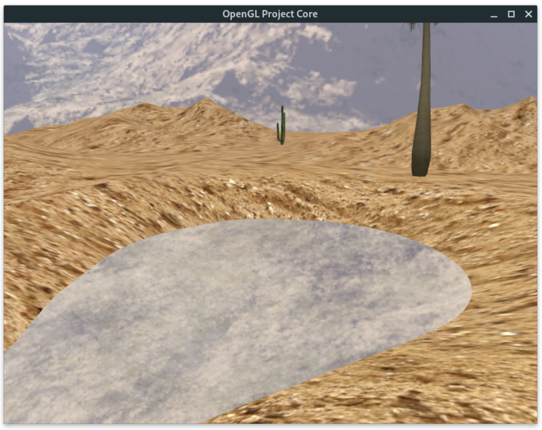

Figure 1: Water

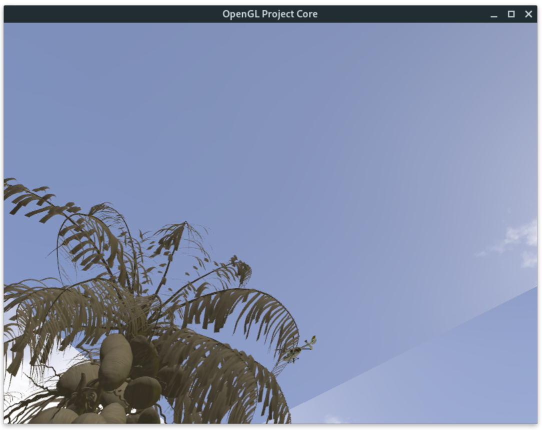

Figure 2: Chopper and a part of the tree

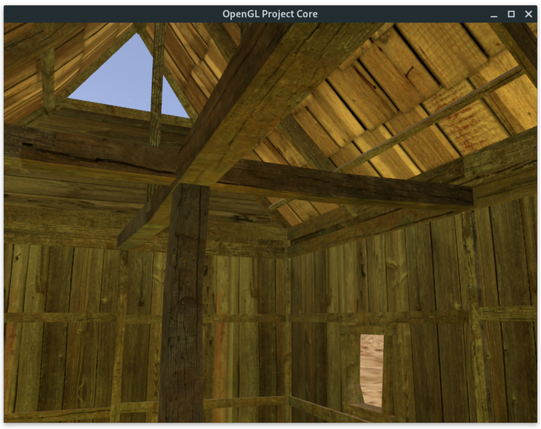

Figure 3: Inside the house

5 Conclusion and further developments

This project was a fun way to discover some functionality on the OpenGL especially discovering the method to create a semi-realistic water reflection and animating object component.

This project could use some further improvements such as:

- Adding shadow mapping
- Create a better water effect with simulated movement and specular highlighting

6 References

- https://learnopengl.com/Lighting/Multiple-lights
- https://learnopengl.com/Advanced-OpenGL/Framebuffers