UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Graphic Processing

– OpenGL Semester Project –

Student: Chețe Doru-Gabriel
Group: 30433

Table of Contents

1. Subject specification

The subject of this project is to produce a scene that looks as realistic as possible using 3D objects with the help of the OpenGL library. The user can view and move through the scene using keyboard and mouse inputs and can change certain aspects of the scene, such as adding fog.

2. Scenario

2.1. Scene and Objects Description

The scenario I chose for the scene is one inspired by a fantasy medieval setting, with castle walls, battlements, towers, old-looking houses and dragons. I thought of a place that could visually represent the many fantasy worlds that are now so popular in movies, books, video games and TV series.

In the center of the scene there is a motte, a fortified tower surrounded by water, which flows from a nearby river. The remnants of an old wall lay in the back of the scene. Near the moat there are some houses and a mill. Close to the river, another couple of towers are guarded by dragons, which look upon the center of the scene. There is a red glow that suggests an approaching dusk and a fog can appear in order to add to the fantasy element of the setting. The tower in the middle proudly displays a lord’s banners and the bridge over the water is guarded by two knights.

I aimed to find and add objects that befit this setting while trying to only add textures that keep the realism of the scene as good as possible, while also managing the polygon count to preserve a good performance. Some of the objects, such as the mill, were edited to look more realistic and in order to be animated properly.

I modeled the ground of the scene myself using the tools provided by Blender. I wanted to obtain the effect of a valley hidden by hills on the sides and cut across by a river. The river is obtained by placing a flat plane right underneath the ground object and using the specific texture. This plane will show only in the parts where the ground object is carved, that is where I intended the river bed to be. The deep green texture together with the scattered trees add to the forest, wild setting of this small fortification.

The houses are the objects with the largest number of textures:

It is an object with a higher level of detail and has a number of materials with different textures. Different kinds of woods for the various parts of the wooden frame of the building, different types of doors and the textures used for the bricks and walls are also different. The textures I chose for this object, as well as the ones for some of the other objects in the scene are darker, in order to add to the medieval setting.

Figure 1. House object

Figure 2. Tower in the center of the scene

The central object of the scene is this little fortification, surrounded by water, with some banners. The flag at the top is animated can be hoisted to the top of the pole with user input. The texture used for this tower is also used on the towers and wall in the back of the scene.

The two knights guarding the bridge have plate armor that is textured to look like dark steel plates.

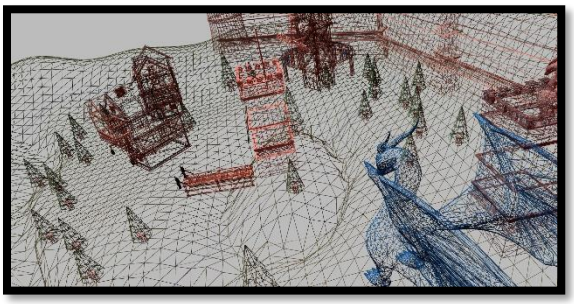

Figure 3. Wireframe view of the scene

Figure 4. Knights guarding bridge

2.2. Functionalities

The first functionality when starting the application is the preview. This is done automatically when starting the application. The preview presents the scene by rotating around the center, where the tower with the flag is located. This preview can be stopped at any time. There are 2 sources of light: one directional and a red-colored point light. Fog effect can be turned on. The light angle of the directional light can be also modifed, as well as the camera speed. The whole scene can be rotated and there are 3 viewing modes for the scene: the normal one, the wireframe viewing mode, and the vertex viewing mode.

3. Implementation

3.1. Functions and special algorithms

For initializing the fog, the fogOn uniform is used to enable the effect. Its color and intensity is set in the fragment shader.

Code used in the fragment shader to enable fog:

```
float computeFog()
{
	float fogDensity = 0.03f;
	float fragmentDistance = length(fPosEye);
	float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));

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

// in the main function
float fogFactor = computeFog();
	vec4 fogColor = vec4(0.5f, 0.5f, 0.5f, 1.0f);
	if(fogOn == 1) {
```

```
            fColor = mix(fogColor, min(colorWithShadow * vec4(light,
1.0f), 1.0f), fogFactor);
      }
      else
      {
            fColor = min(colorWithShadow * vec4(light, 1.0f), 1.0f);
      }
```

For shadows, the algorithm presented in the labs is used, where we build a depth map looking from the direction of the light, in order to see which objects block the rays of light and where specifically. Once the map is built, the algorithm relies on the fact that if the Z coordinate of an object is higher than another, then that object is in the shadow. (The shadow is cast by the object first hit by the light ray)

This is also done in the fragment shader:

```
float computeShadow()
{
      // perform perspective divide
      vec3 normalizedCoords = fragPosLightSpace.xyz /
fragPosLightSpace.w;
      if (normalizedCoords.z > 1.0f)
            return 0.0f;
      // Transform to [0,1] range
      normalizedCoords = normalizedCoords * 0.5 + 0.5;
      // Get closest depth value from light's perspective
      float closestDepth = texture(shadowMap, normalizedCoords.xy).r;
      // Get depth of current fragment from light's perspective
      float currentDepth = normalizedCoords.z;
      // Check whether current frag pos is in shadow
```

```
    float bias = 0.001f;
    float shadow = currentDepth - bias > closestDepth ? 1.0 : 0.0;

    return shadow;
}
```

The FBO is initialized in the main file of the application:

```
void initFBO() {
    //generate FBO ID
    glGenFramebuffers(1, &shadowMapFBO);

    //create depth texture for FBO
    glGenTextures(1, &depthMapTexture);
    glBindTexture(GL_TEXTURE_2D, depthMapTexture);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT,
        SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

    float borderColor[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);

    //attach texture to FBO
    glBindFramebuffer(GL_FRAMEBUFFER, shadowMapFBO);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
GL_TEXTURE_2D, depthMapTexture, 0);
    glDrawBuffer(GL_NONE);
    glReadBuffer(GL_NONE);
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
}
```

3.2. Graphics model

The scene was put together using Blender. Most of the objects used were downloaded from the internet and edited to fit into the scene. Blender gave me the

tools to build the terrain and to put all of the objects and textures together. When using Blender however, I had to take in consideration the fact that the coordinate systems are different.

Blender		OpenGL
X	->	X
Y	->	Z
Z	->	-Y

Most of the objects in the scene are referred to using a single object in OpenGL. However for the objects that are animated (the flag, the blades of the windmill) I had to load them separately. Additional transformations have to be done to animated this objects. For example, the blades of the windmill first have to be translated into origin, then rotated with the desired angle and then translated back to their position.

When the H button is pressed the flag starts being hoisted up, this being equivalent to translating on the Y axis in OpenGL. This is done so until the coordinate at the top of the pole is reached.

3.3. Class hierarchy

The project makes use of 5 classes: Camera, Mesh, Model3D, Shader and Skybox. The Camera class is used for the movement inside of the scene and it has several methods for moving on the axes and rotating. It is instantiated in main.cpp as “myCamera” and used throughout the program. Mesh class is used for drawing. Model3D processes vertex data. The various models are loaded using Model3D objects in main. Shader class is used for processing shaders and it is also used in main for loading the shader files. Skybox class is used for adding the skybox feature to the project. It has functions for initializing it, for loading and for drawing.

4. User manual

The preview is initialized automatically on application startup.

Stop preview: V

The following buttons are for moving around the scene:

Move Forwards: W

Move Backwards: S

Move Left: A

Move Right: D

Rotate Scene: E and Q

Look: Mouse

Camera Speed Up:]

Camera Speed Down: [

Other buttons:

Smaller Light Angle: J

Bigger Light Angle: K

Activate Fog: F

Wireframe: 2

Vertices: 1

Normal viewing mode: 3

Hoist the flag on the tower up: H

5. Conclusions

Developing this application gave me more insight into how OpenGL works and how it can be used. It made me think about how a lot of the theoretical aspects of Graphic Processing are implemented and used in programming.

For future development, I would add more functionalities that allow the user to interact with the scene, such as enabling more object movements and animations. A better, more complex preview could be added, one that follows a certain path on the ground to present the scene at ground level. The animations could also be improved (I would try to animate the dragons' wings).

In conclusion, this application helped me tie all the information gathered throughout the semester and better understand the OpenGL library.