Photorealistic Presentation of 3D Objects

OpenGL

Linca Paul Tudor

E_30432

1. Contents

1. Contents

2. Subject Specification

3. Scenario

   3.1 Scene and objects description

   3.2 Functionalities

4. Implementation details

   4.1 Functions and special algorithms

   4.2 Graphical model

   4.3 Data structures

   4.4 Class hierarchy

5. Graphical User Interface / User Manual

6. Conclusions and further developments

7. References

2. Subject specification

The subject of the project consists in the photorealistic presentation of 3D using the OpenGL library. Through the graphical interface, the user should be able to control and visualize the scene using the keyboard and mouse as well as opt for an automated view of the scene.

3. Scenario

3.1 Scene and objects description

I chose the theme of my project to be an abandoned medieval village that has been infested by zombies.

Thus, the scene consists of numerous houses placed along the main road, a cathedral with a cemetery from which the zombies originate, stone walls that form the perimeter of the scene and a castle that overlooks the village. Additionally, the whole scene is enclosed in a skybox that provides the background, making the scene look nicer and bigger than it really is.

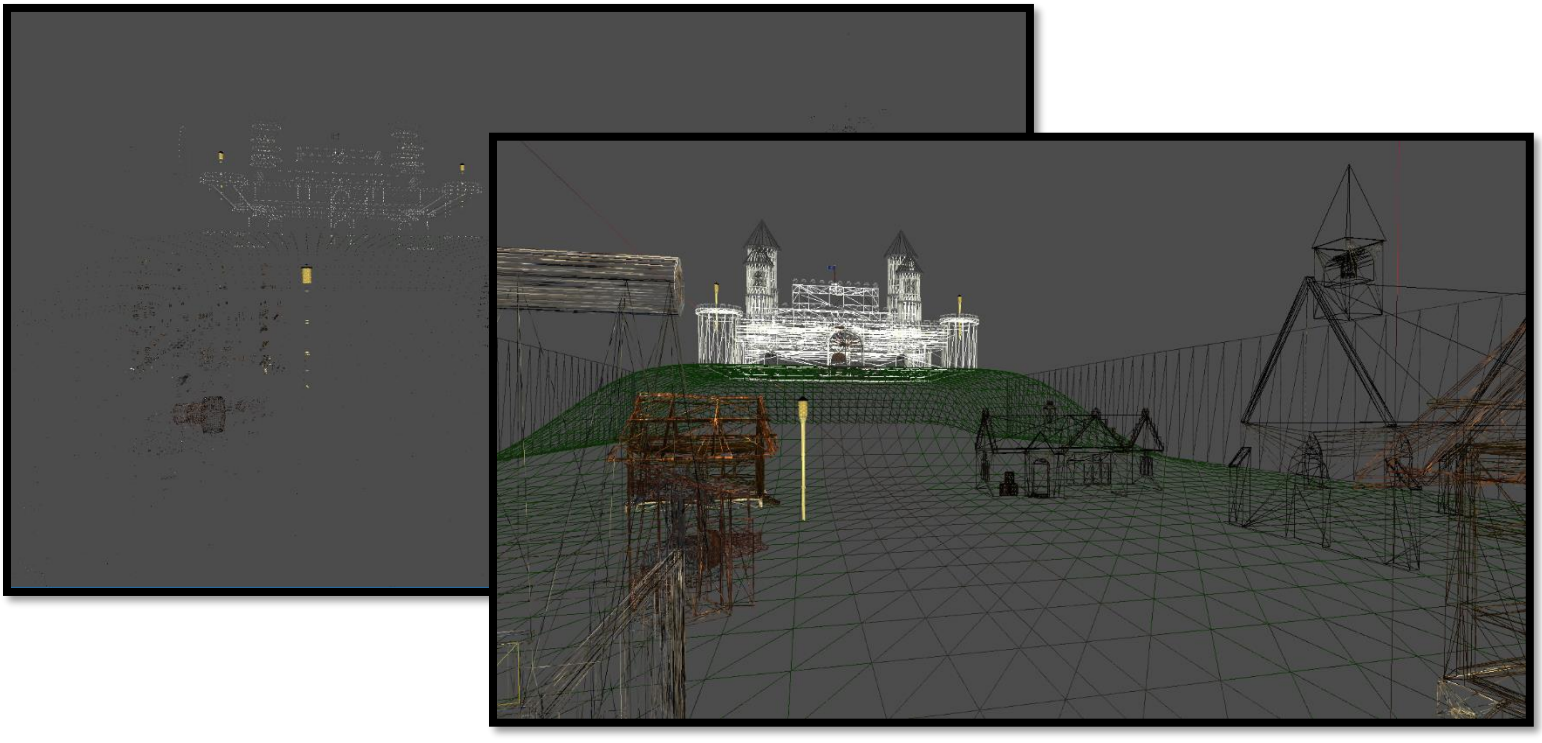
3.2 Functionalities

All of the objects in the scene are fixed except for the zombie. The zombie moves automatically, being trapped in a loop: it comes out of a grave, goes to the center of the village and returns to the cemetery entering a different grave.

The user can navigate the scene using the ‘WASD’ keys and the mouse. Additionally, there is an option to get a bird’s-eye view of the scene by pressing the ‘C’ key.

Another functionality that has been added is the generation of fog. By pressing the ‘F’ key, the user will be able to observe the scene become foggy, thus making the objects that are far away from the camera harder to see.

Finally, the user can get a wireframe or point view of the objects by pressing ‘O’ and ‘P’ respectively.



4. Implementation details

4.1 Functions and special algorithms

Initially, the window for the project is created through the initOpenGLWindow() function. The height and width of the window can be set by the programmer (1600x900) in my case. The glGenTextures(), glBindTextures() and glTextImage2D() functions are used to create the depth textures for the objects. WindowsResizeCallback() handles the event in which the user desires to resize the window.

The models and shaders are loaded to the application through the initModels() and initShaders() functions and the uniforms needed to transfer data from the application to the shaders are initialized in the initUniforms() functions.

The processMovement() functions, as its name would suggest, is used to perform all the changes to the models and camera in order to visualize the scene in the user's desired way. The rotateCamera() and moveModel() functions move the camera and zombie 3D model respectively around the scene. The mouse and keyboard events are handled through the keyboardCallback() and mouseCallback() functions.

Finally, the renderScene() function sends all the data to the shaders and, through scaling, translating and rotating, it computes how and where the models, lights and shadows should be rendered.

4.2 Graphics model

The scene has been created using blender, by binding together numerous 3D models downloaded online and exporting it as an .obj file so that it can be used in the project.

4.3 Data structures

A 3D object is represented using vertices and topological information. For each vertex we need attributes such as position, color, texture coordinate and other relevant information. Vertex data is stored in Vertex Buffer Objects. A series of VBOs can be stored in a Vertex Array Object.

In order to be able to render 3D objects, we need to write at least a vertex shader and a fragment shader. I have used multiple shaders: depth map shader, light shader, skybox shader etc. Uniforms were used to transfer data from the application to the sharers.

4.4 Class hierarchy

Camera.cpp is the class that computes the camera coordinates, how it should move and what it should look at.

Mesh.cpp and Mesh3D.cpp are used to define the objects and to apply textures on them.

Shader.cpp compiles the shaders and links them to the main application.

Skybox.cpp creates the skybox and defines how it should act.

5. Graphical User Interface / User manual

Upon compiling and building the program, the user will be presented with this window, in which the scene is rendered:

From here, the user can choose how view the scene and where to position the camera, using these keyboard bindings:

Mouse	Camera rotation
W, A, S, D	Camera movement (Forward, Left, Backwards, Right)
C	Turn Bird's-eye view on
V	Turn Bird's-eye view off
J	Rotate light left
L	Rotate light right
F	Turn fog on
G	Turn fog off
O	Wireframe view
P	Point view
I	Normal (texture) view
Space	Stop/Start model animation

6. Conclusions and further development

I believe that this project was chose well for students to be introduced to the OpenGL library and learn how the graphics card processes data in order to render what we see on our computes screens.

My project has a lot of room for further improvement. The animation of the zombie can be enhanced, more animations and different lights can be added, more effects such as rain and snow can be implemented, the skybox can be integrated better, camera collision can be implemented and object interaction could be an interesting functionality to be developed.

7. References

- https://moodle.cs.utcluj.ro/course/view.php?id=126
- https://www.turbosquid.com/
- https://free3d.com/
- https://www.cgtrader.com/
- https://learnopengl.com/
- https://www.blender.org/support/tutorials/
- https://www.youtube.com/