Graphic Processing

Student: Moldovan Alexandru Cristian

Group: 30433

3rd year

Assistant Teacher: Victor Ioan Bâcu

Contents

2. Subject specification

The subject of this project is the implementation of a 3D graphics application developed in C++ with OpenGL support.

The scene that I implemented is a military shooting range with moving targets, weapons, a truck and two tents. The user can move around the scene see the features implemented such as lights, shadows, animations, physics and collisions. The initial teapot model was kept for the physics demonstration.

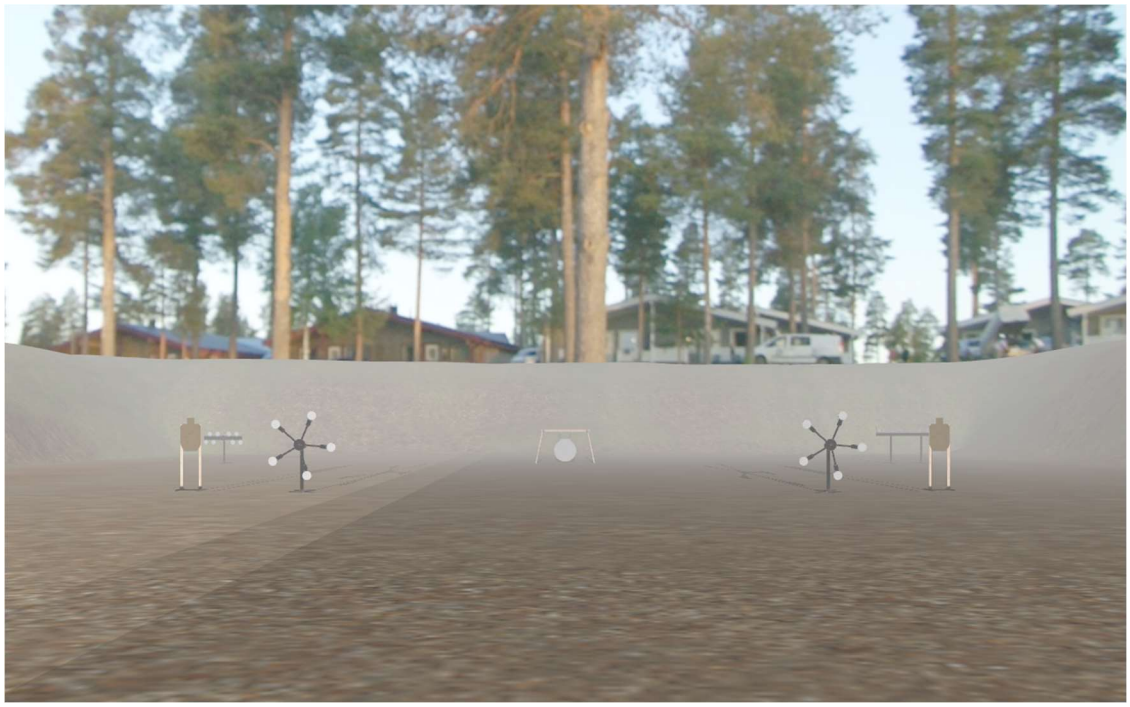
3. Scenario

3.1. Scene and object description

The scene is a militay shooting range. It contains 5 types of targets. Two of those are moving one of them is also rotating to give the shooters a challenge. Behind the player there are two benches with weapons – two automatic rifles, a pistol and a sniper rifle, two tents, one of them has the light on, a truck with the headlights on and the teapot that was unloaded from the truck and a tunel that provides access to the shooting range because for safety measures the shooting range is in a small valley sorounded by little dirt hills.

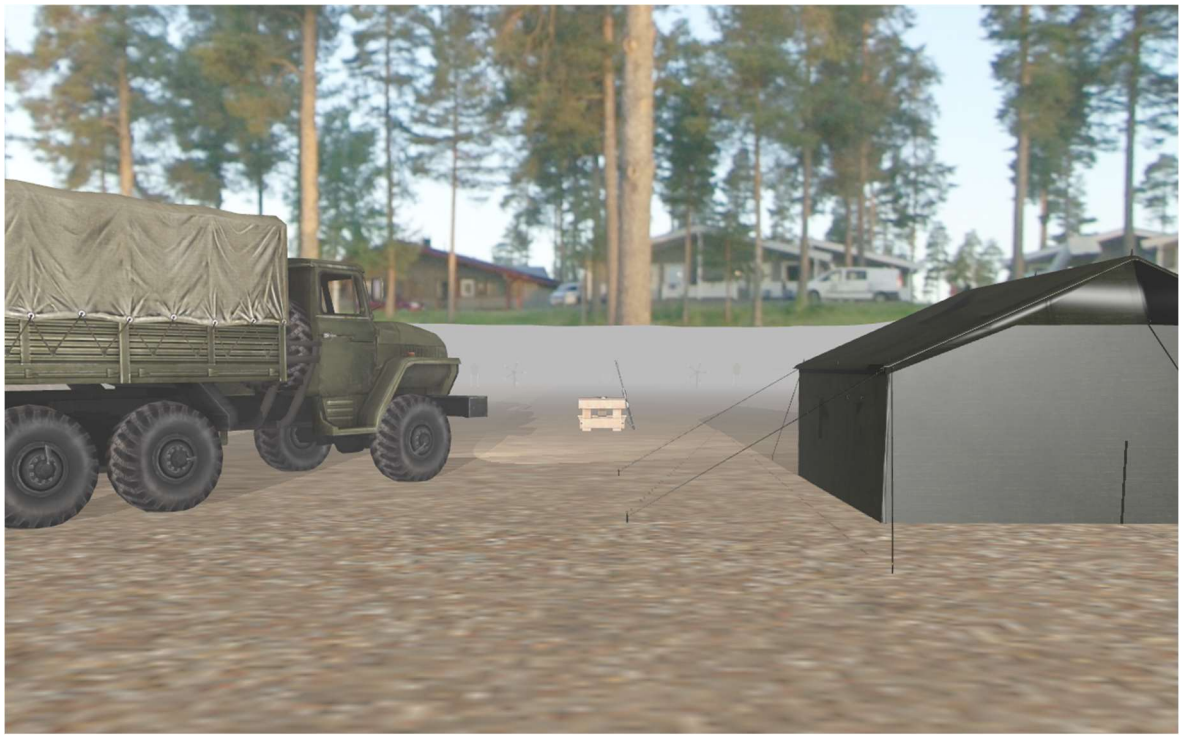
There are 3 types of lights that are used to illuminate the scene – a directional light for the sun, a point light that illuminates the interior of a tent and two directional lights for the headlights of the truck.

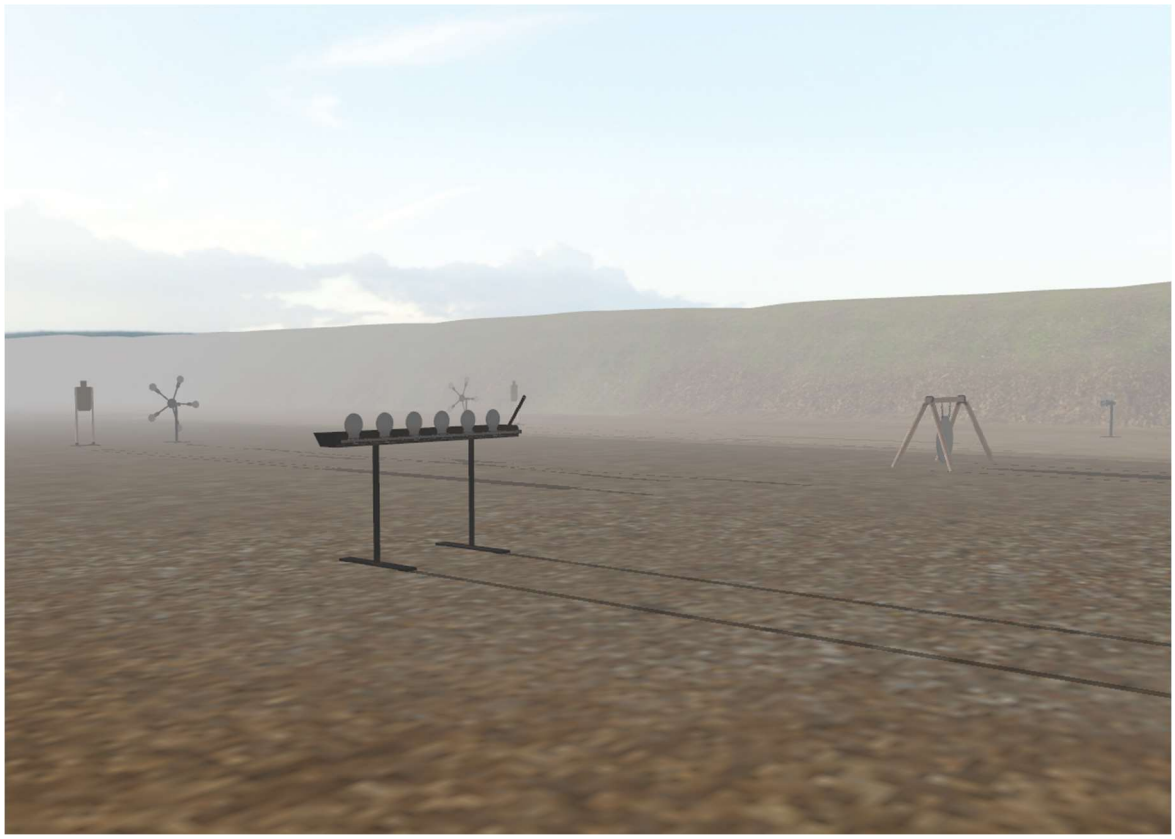
Other features of the scene are the fog that is visible but not to dense so the targets are still visible and the cubemap that represents a sunny day.



This is the front view of the scene in which you can see the targets.

This is the other part of the scene with the tents, the truck, the weapons and the two benches and also the teapot.





Some other photos from different angles from the scene.

3.2. Functionalities

The physics in the project are implemented using the Bullet Physics Library that provides collision between objects, physics calculation and many more.

Every object in the scene has a box collider that is used for collision detection and is a kinematic object with mass 0 meaning that is stationary and can't be moved beside the terrain that uses a triangle mesh collider and the teapot that is a rigidbody and is affected by gravity and can be pushed around.

The player is also a rigidbody and uses a capsule collider so it can interact with the objects in the scene. It can be controlled using the keyboard. The camera is attached to the player is following it and can be used to look around with the mouse. The camera also can be moved using an animation that is activated with a press of a key and shows the entire scene.

Lighting as mentioned above is also implemented with 3 types of lights – point light, spot light and directional light and also shadows for the objects that are affected by the lights.

Fog is also present with a small density but enough to be visible.

Animations are for objects and object components are implemented to move and rotate the targets. The animations are looping ones and run indefinitely and are implemented using keyframes so positions and rotations can be added to change the animation.

4. Implementation details

4.1. Functions and special algorithms

4.1.1. Possible solution

For the OpenGL implementation I used the functions that are presented in the laboratory works. Some of them include - glBindTexture(…), glGenTexure (…), glfwCreateWindow (…), glViewport (…), glGetUniformLocation(…), glUniformMatrix4fv(…), glGenFramebuffers(…), glTexParameteri(…).

Some custom functions implemented in main that I modified are as follows:

- initBullet() - used for initializing the physics library
- initSkyBox() - used for initializing the skybox
- initShaders() - used for initializing the shaders
- initUniforms() - used for getting the uniform locations and setting the data
- initFBO() - used for initializing the buffers and textures for the shadow computation
- initModels() - used for initializing the 3D models
- computeDelta() - used for computing delta as the difference between current frame and last frame
- animateCamera() - function that contains the code for the camera animation
- animateObjectLocation(Object3D& object, const std::vector<btVector3> keyframes) – function used to animate the location of any 3D object and uses keyframes to store locations and interpolates between them
- animateObjectRotation(Object3D& object, const std::vector<btQuaternion> keyframes) – function used to animate the rotation of any 3D object and uses keyframes to store rotations and interpolates between them
- renderScene() – contains the code that is run every frame to render the scene
- mouseCallback(GLFWwindow* window, double xpos, double ypos) – contains the code that is run when the mouse is moved
- scrollCallback(GLFWwindow* window, double xoffset, double yoffset) – contains the code that is run when the scroll wheel is used
- processMovement() - contains the code that is run when the keyboard is used
- drawObjects(gps::Shader shader, bool depthPass, GLint modelLoc) – contains the code that is used to render the objects

Some other functions:

- Object3D::Render(gps::Shader shader, bool depthPass, GLint modelLoc, GLint &normalMatrixLoc, glm::mat4 view) – used to render the the 3D object and is called in the drawObjects for every object in the scene
- Object3D::Load(std::string fileName, std::string basePath, float mass, glm::vec3 position, glm::vec3 rotation, COLLIDER_TYPE colliderType, bool isKinematic) – used to load the mesh from the .obj file, create the object and add a collider
- Player::move(btVector3 velocity) – used to move the player by applying force to the rigidbody attached to it
- Model3D::ReadOBJ(std::string fileName, std::string basePath) – function used to read the data from .obj file, contains also code for computing the bounding box of the 3D model by getting the minimum and maximum position of the vertices

Regarding shaders, 3 shaders were used for the application

- basic - used for rendering the objects, contains functions for computing the light, shadows and fog
- depthMapShader – used for creating the shadow map
- skyboxShader - used for rendering the skybox

4.1.2. Motivation of the chosen approach

Most of those functions were implemented in the laboratory works and I used them as inspiration and as a base to implement my own functionality. There are functions in custom classes - Object3D and Player that were created to better incorporate the physics library and to organize and reuse the code as much as possible.

4.2. Graphics model

All the 3D models other than the terrain were downloaded from the internet. Those that were not the .obj format were imported in Blender and then converted to .obj and then the .mtl file was modified to contain the path to the textures.

The terrain was custom made in Blender and the texture was created by combining multiple textures with the use of texture paint and nodes and baked into an image file. The actual mesh of the terrain was created by hard surface modeling a base mesh and then adding details using sculpting. Because the mesh was very high poly, a decimate modifier was used to unsubdivided the mesh so it can be also used for a collider. As a final step it was exported as an .obj file.

4.3. Data structure

Some custom data structures besides the ones already implemented in the laboratory works and project core are the Object3D and Player classes that were designed to use the physics library. Other data structure used is a struct that contains data for the bounding box such as – minimum position, maximum position, origin, and size.

4.4. Class hierarchy

- Camera class – contains the logic for the camera, has function for moving and rotating the camera and computing directional vectors
- Mesh class – contains the logic for creating a mesh
- Model3D class – contains the logic for creating a 3D model from an .obj file and computing the bounding box
- Object3D class – contains the logic for creating a 3D object from the Model3D class data and adding collision to the object and rendering
- Player class – contains the logic for creating a player and controlling it
- Shader class – contains the logic for loading and compiling shaders
- Skybox class - contains the logic for creating and rendering the skybox
- Window class – contains the logic for the program window

5. Graphical user interface presentation / User manual

- WASD for moving the player
- Mouse for rotating the camera
- Scroll wheel for zooming in and out
- R – wireframe view
- T – solid view
- U – polygonal shading
- Y – smooth shading
- P – start camera animation
- I – print player position in console

6. Conclusions and further development

The project in the current state is a presentation of some features that can be implemented using OpenGL in a game like manner by allowing the user to move and look around the scene. It features real time rendering with lighting and shading effects achievable through OpenGL, texture mapping and animations.

For further development I would add the possibility of the user to interact with the objects in the scene and create a minigame of shooting the targets together with a scoring system and expand the scene by adding more objects and terrain.

7. References

- https://learnopengl.com
- Graphics Processing Course

- http://www.opengl-tutorial.org/miscellaneous/clicking-on-objects/picking-with-a-physics-library/
- https://pybullet.org/Bullet/BulletFull/
- https://chat.openai.com
- https://sketchfab.com/feed