Department of Computer Science
Technical University of Cluj-Napoca

Graphics Processing Systems

Laboratory activity 2020-2021

Project: FPP Shooter Game in OpenGL

Documentation

Name: Sandor Marian
Group: 30431
Email: marianshandor@gmail.com

Chapter 1: Content

Chapter 2

Subject specification

The subject of this project is the photorealistic presentation of 3D objects using OpenGL library. The user directly manipulates by mouse and keyboard inputs the scene of objects.

My idea for the project is to create somewhat of a minimalistic copy of a very popular game, Counter Strike [1]. The main characteristics that I want to incorporate in my project are that there will be a map of a small city and the player will control a character being able to shoot.

Chapter 3

Scenario

Scene and object description

The scene resembles a small city having a number of buildings, roads, cars and street lamps. This creates the world for our character. The character is a special police force troop carrying a rifle and will be able to move freely inside the city.

Functionalities

The user of the application can:

- Control the character movement (move, jump, duck, fire the gun, turn on flashlight).
- Use the free cam to navigate the scene.
- Change from day time to night time and reverse.
- Turn on and off the lights (car lights and street lamps).
- Play sounds effects.
- View the scene in different modes(ex: polygonal)

The scene also has some functionalities that are independent of the user:

- The intro animation in which the scene is presented and the transition to the FPP camera is done.

Chapter 4

Implementation

Functions and special algorithms

There are two libraries that provide vital functionalities to our project, namely GLM [2] and GLFW [3].

OpenGL Mathematics (GLM) is a header only C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specifications.

GLFW is an Open Source, multi-platform library for OpenGL, OpengGL ES and Vulkan development on the desktop. It provides simple API for creating windows, contexts and surfaces, receiving input and events.

Another important library that I have to mention is irrKlang [4] with the help of which I was able to add sound effects to the application.

irrKlang is a high level 2D and 3D cross platform (Windows, macOS, Linux) sound engine and audio library which plays WAV, MP3, OGG, FLAC, MOD, XM, IT, S3M and more file formats, and is usable in C++ and all .NET languages (C#, F#, etc.) It hass all the features known from low level audio libraries as well as lots of useful features like a sophisticated streaming engine, extandable audio reading, single and multithreading modes, 3d audio emulation for low end hardware, a plugin system, multiple rolloff models and more. All this can be accessed via an extremely simple API.

Starting from the main.cpp file and then moving to the shaders we identify a few essential functions such as: renderScene(), renderCharacter() and processIntroAnimation().

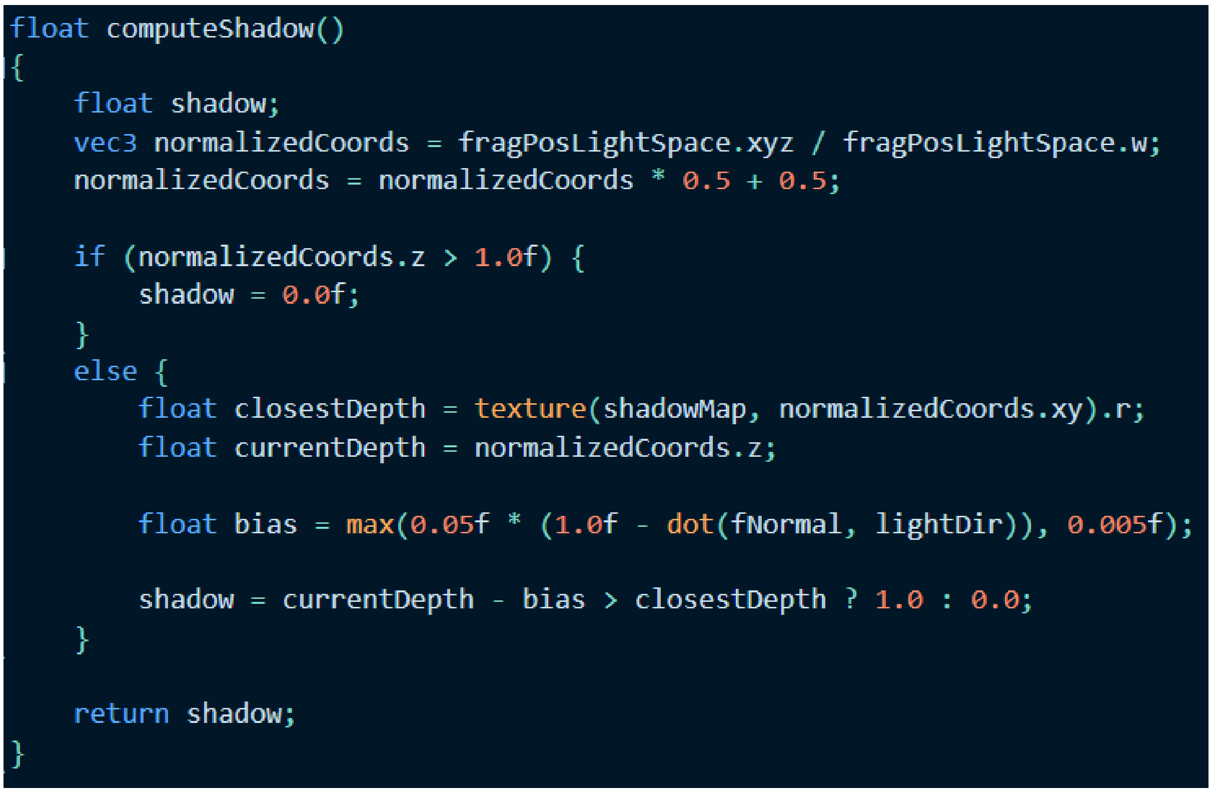
The processIntroAnimation() is responsible for the intro animation in which the camera takes a path presenting the scene and finishing in the FPP camera position from which the player is given the control. The animations is performed in 5 steps, each describing the motion that the camera has to do, ending with the step in which the application sets the necessary things such that the player can take control of the character.

The renderCharacter() function is also important for that it takes care of the character animations such as when it ducks or when it jumps. Through means of some flag variables, it detects weather it should render the standing position model, the duck position model or play the jumping animation which consists of changing the model between two positions of the jump.

Even though this function are vital for displaying frames to the user, the application has to go through a sequence of initializations which are performed by the functions: initOpenGLState(), initCrosshair(), initFBO(), initSkyBox(), initModels(), initShaders(), initUniforms() which make sure that the environment is all set for rendering.

In order to provide minimal realism, the scene should take in account the light sources and respect the basics of the shadows. For the sake of simplicity, we will only take into account the shadows given by the directional light (the light that comes from the sun or moon in our case).

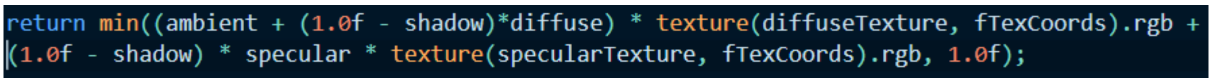
To be able to compute the shadows we first need to generate a shadow depth map texture that will contain the shadow information as seen from the directional light perspective. Once we have it, we can compute the shadow component in the fragment shader.



```
float computeShadow()
{
    float shadow;
    vec3 normalizedCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    normalizedCoords = normalizedCoords * 0.5 + 0.5;

    if (normalizedCoords.z > 1.0f) {
        shadow = 0.0f;
    }
    else {
        float closestDepth = texture(shadowMap, normalizedCoords.xy).r;
        float currentDepth = normalizedCoords.z;

        float bias = max(0.05f * (1.0f - dot(fNormal, lightDir)), 0.005f);

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

    return shadow;
}
```

Figure 1 Shadow Computation

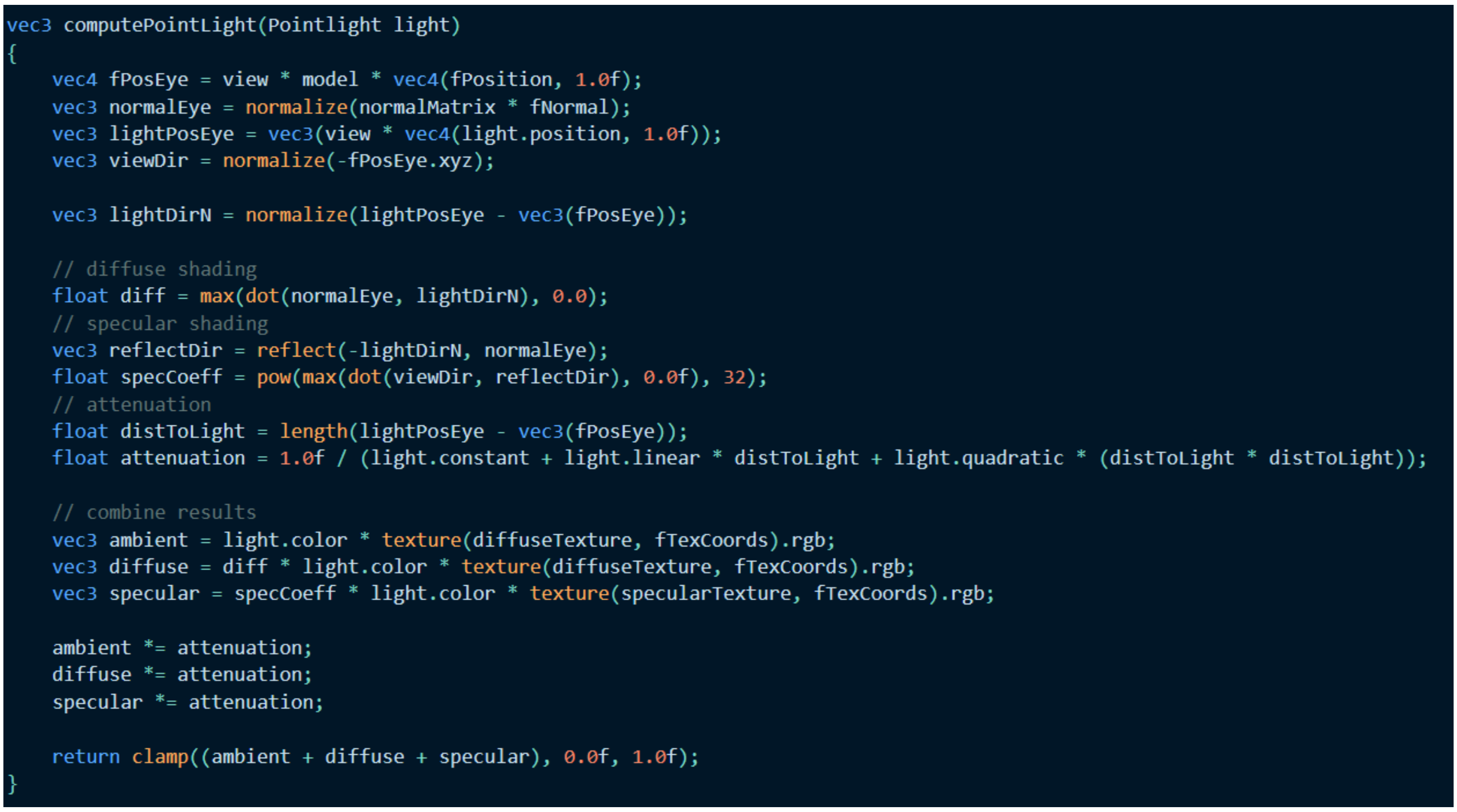
We then incorporate it in the final computation of the color.



```
return min((ambient + (1.0f - shadow)*diffuse) * texture(diffuseTexture, fTexCoords).rgb +
(1.0f - shadow) * specular * texture(specularTexture, fTexCoords).rgb, 1.0f);
```

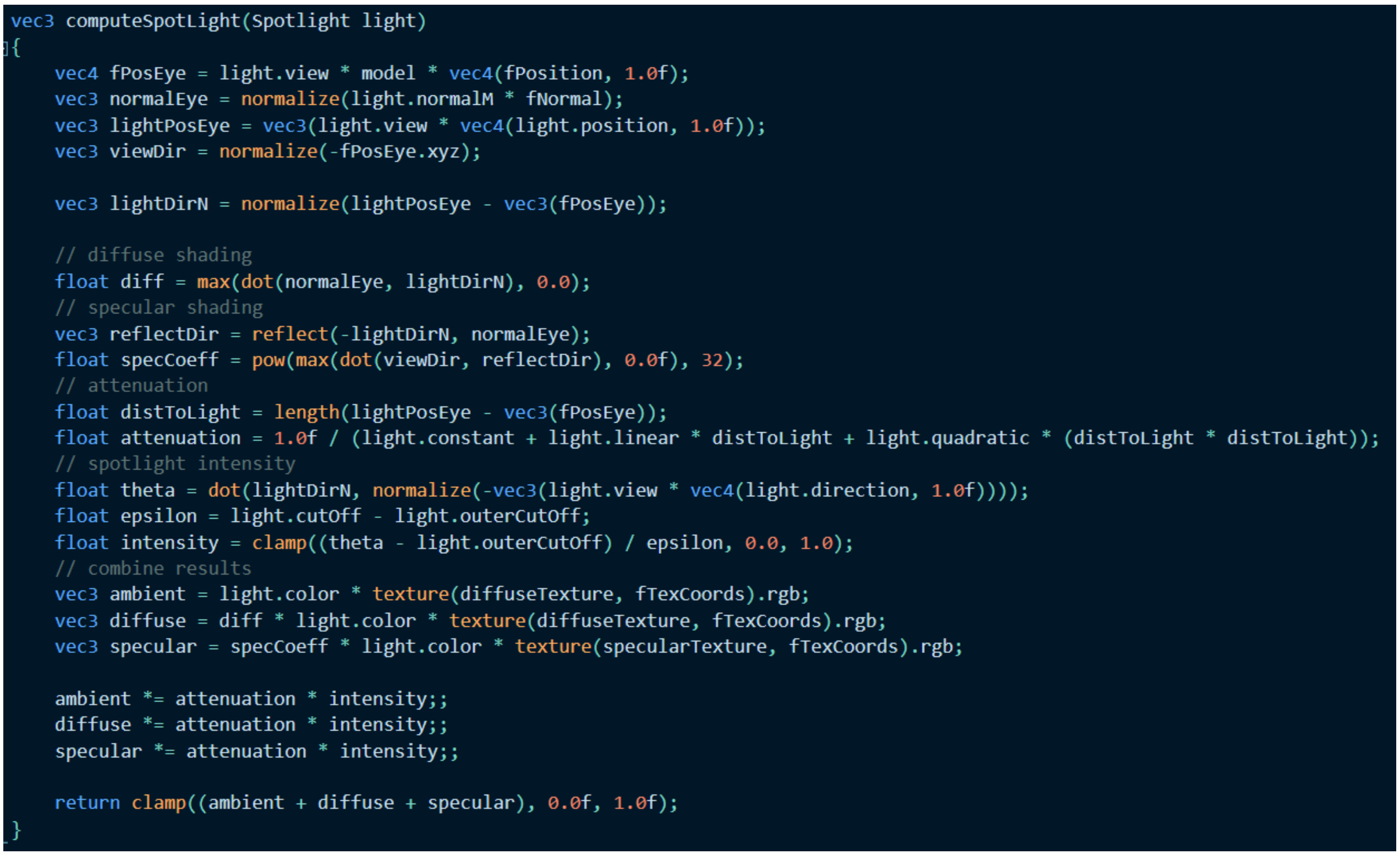
Figure 2 Directional Light Component

Moving on, there are two more important functions for computing the light given by a point source or a spotlight.



```
vec3 computePointLight(Pointlight light)
{
    vec4 fPosEye = view * model * vec4(fPosition, 1.0f);
    vec3 normalEye = normalize(normalMatrix * fNormal);
    vec3 lightPosEye = vec3(view * vec4(light.position, 1.0f));
    vec3 viewDir = normalize(-fPosEye.xyz);

    vec3 lightDirN = normalize(lightPosEye - vec3(fPosEye));

    // diffuse shading
    float diff = max(dot(normalEye, lightDirN), 0.0);
    // specular shading
    vec3 reflectDir = reflect(-lightDirN, normalEye);
    float specCoeff = pow(max(dot(viewDir, reflectDir), 0.0f), 32);
    // attenuation
    float distToLight = length(lightPosEye - vec3(fPosEye));
    float attenuation = 1.0f / (light.constant + light.linear * distToLight + light.quadratic * (distToLight * distToLight));

    // combine results
    vec3 ambient = light.color * texture(diffuseTexture, fTexCoords).rgb;
    vec3 diffuse = diff * light.color * texture(diffuseTexture, fTexCoords).rgb;
    vec3 specular = specCoeff * light.color * texture(specularTexture, fTexCoords).rgb;

    ambient *= attenuation;
    diffuse *= attenuation;
    specular *= attenuation;

    return clamp((ambient + diffuse + specular), 0.0f, 1.0f);
}
```

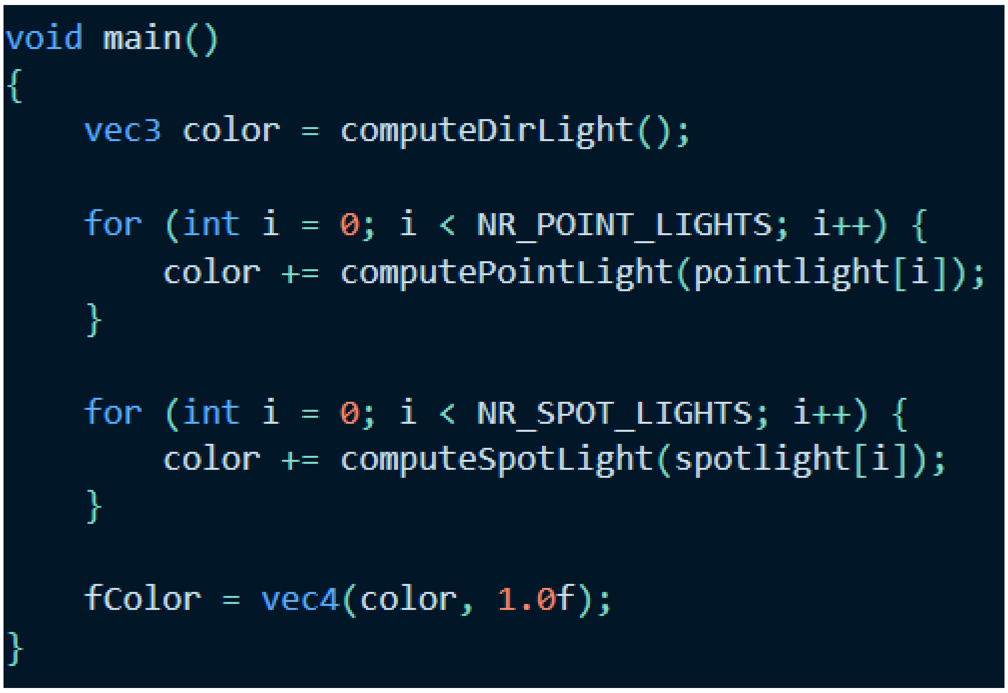
Figure 3 Point Light Computation



```
vec3 computeSpotLight(Spotlight light)
{
    vec4 fPosEye = light.view * model * vec4(fPosition, 1.0f);
    vec3 normalEye = normalize(light.normalM * fNormal);
    vec3 lightPosEye = vec3(light.view * vec4(light.position, 1.0f));
    vec3 viewDir = normalize(-fPosEye.xyz);

    vec3 lightDirN = normalize(lightPosEye - vec3(fPosEye));

    // diffuse shading
    float diff = max(dot(normalEye, lightDirN), 0.0);
    // specular shading
    vec3 reflectDir = reflect(-lightDirN, normalEye);
    float specCoeff = pow(max(dot(viewDir, reflectDir), 0.0f), 32);
    // attenuation
    float distToLight = length(lightPosEye - vec3(fPosEye));
    float attenuation = 1.0f / (light.constant + light.linear * distToLight + light.quadratic * (distToLight * distToLight));
    // spotlight intensity
    float theta = dot(lightDirN, normalize(-vec3(light.view * vec4(light.direction, 1.0f))));
    float epsilon = light.cutOff - light.outerCutOff;
    float intensity = clamp((theta - light.outerCutOff) / epsilon, 0.0, 1.0);
    // combine results
    vec3 ambient = light.color * texture(diffuseTexture, fTexCoords).rgb;
    vec3 diffuse = diff * light.color * texture(diffuseTexture, fTexCoords).rgb;
    vec3 specular = specCoeff * light.color * texture(specularTexture, fTexCoords).rgb;

    ambient *= attenuation * intensity;;
    diffuse *= attenuation * intensity;;
    specular *= attenuation * intensity;;

    return clamp((ambient + diffuse + specular), 0.0f, 1.0f);
}
```

Figure 4 Spot Light Computation

To obtain the final color of the fragment, the color can be influenced by more than one source of light, we add all the light components and return the result.



```
void main()
{
    vec3 color = computeDirLight();

    for (int i = 0; i < NR_POINT_LIGHTS; i++) {
        color += computePointLight(pointlight[i]);
    }

    for (int i = 0; i < NR_SPOT_LIGHTS; i++) {
        color += computeSpotLight(spotlight[i]);
    }

    fColor = vec4(color, 1.0f);
}
```

Figure 5 Final Fragment Color

Graphics model

In order to bring the scene to live, I used multiple 3D models that I have downloaded free from the internet. There are a few things that I kept in mind while searching for objects and those are that they should have obj format and contain a .mtl file that includes textures. Most of the objects can be found on these three websites: turbosquid [5], cgtrader [6] and free3D [7]. As for the textures, I got the ones I used from textures website [8]. Some of the objects I had to modify a bit to suit my taste and for that I used blender. The most demanding part was to find a proper character model that also has skeleton attached in order to be able to animate it in blender.

Data structures

The data structures used in the application are the one presented and discussed at the laboratory. The main ones are: Model3D (for loading objects into the scene), SkyBox(for renderng the skybox), shaders (for rendering the scene – vertex shader and fragment shader), Framebuffer Objects (used in shadow computation), uniforms (for sending data to the shaders), OpenGL (specific datatypes and functions: vec3, mat4, lookAt(), glfwGetTime() etc.).

Class hierarchy

The project's class hierarchy is defined by the libraries used and by the shaders.

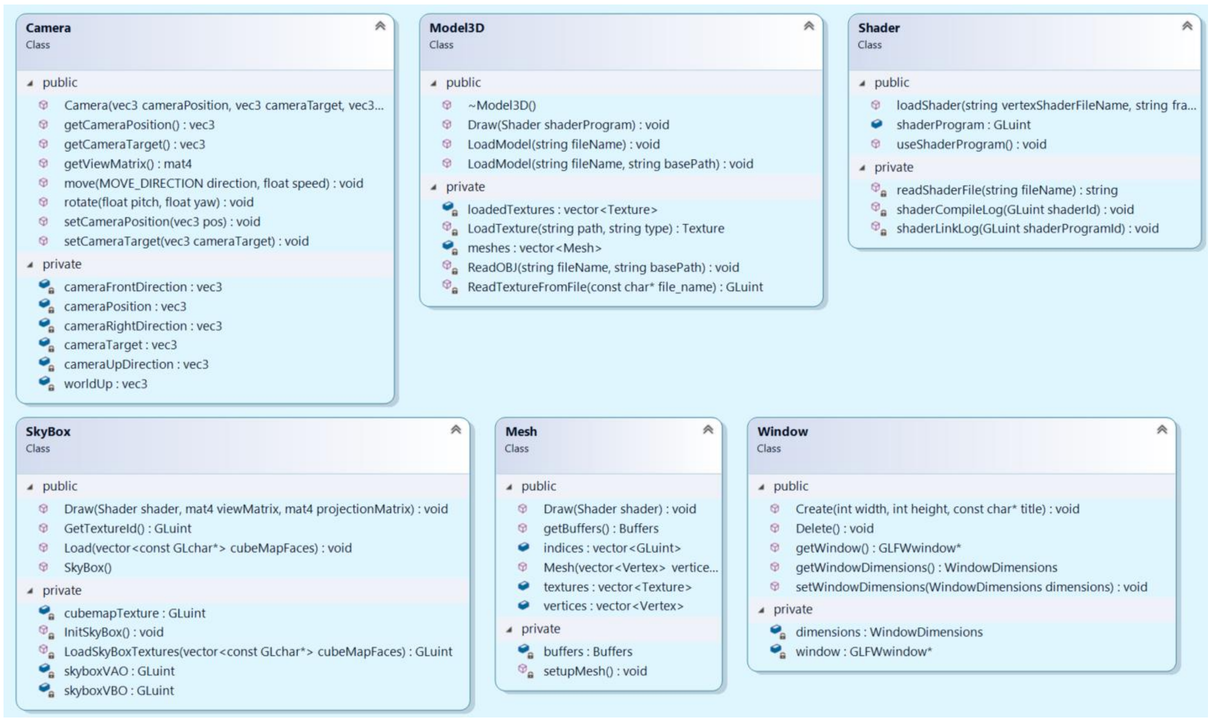

Camera
Class
public
Camera(vec3 cameraPosition, vec3 cameraTarget, vec3...
getCameraPosition() : vec3
getCameraTarget() : vec3
getViewMatrix() : mat4
move(MOVE_DIRECTION direction, float speed) : void
rotate(float pitch, float yaw) : void
setCameraPosition(vec3 pos) : void
setCameraTarget(vec3 cameraTarget) : void
private
cameraFrontDirection : vec3
cameraPosition : vec3
cameraRightDirection : vec3
cameraTarget : vec3
cameraUpDirection : vec3
worldUp : vec3
Model3D
Class
public
~Model3D()
Draw(Shader shaderProgram) : void
LoadModel(string fileName) : void
LoadModel(string fileName, string basePath) : void
private
loadedTextures : vector<Texture>
LoadTexture(string path, string type) : Texture
meshes : vector<Mesh>
ReadOBJ(string fileName, string basePath) : void
ReadTextureFromFile(const char* file_name) : GLuint
Shader
Class
public
loadShader(string vertexShaderFileName, string fra...
shaderProgram : GLuint
useShaderProgram() : void
private
readShaderFile(string fileName) : string
shaderCompileLog(GLuint shaderId) : void
shaderLinkLog(GLuint shaderProgramId) : void
SkyBox
Class
public
Draw(Shader shader, mat4 viewMatrix, mat4 projectionMatrix) : void
GetTextureId() : GLuint
Load(vector<const GLchar*> cubeMapFaces) : void
SkyBox()
private
cubemapTexture : GLuint
InitSkyBox() : void
LoadSkyBoxTextures(vector<const GLchar*> cubeMapFaces) : GLuint
skyboxVAO : GLuint
skyboxVBO : GLuint
Mesh
Class
public
Draw(Shader shader) : void
getBuffers() : Buffers
indices : vector<GLuint>
Mesh(vector<Vertex> vertice...
textures : vector<Texture>
vertices : vector<Vertex>
private
buffers : Buffers
setupMesh() : void
Window
Class
public
Create(int width, int height, const char* title) : void
Delete() : void
getWindow() : GLFWwindow*
getWindowDimensions() : WindowDimensions
setWindowDimensions(WindowDimensions dimensions) : void
private
dimensions : WindowDimensions
window : GLFWwindow*

Figure 6 Class Diagram

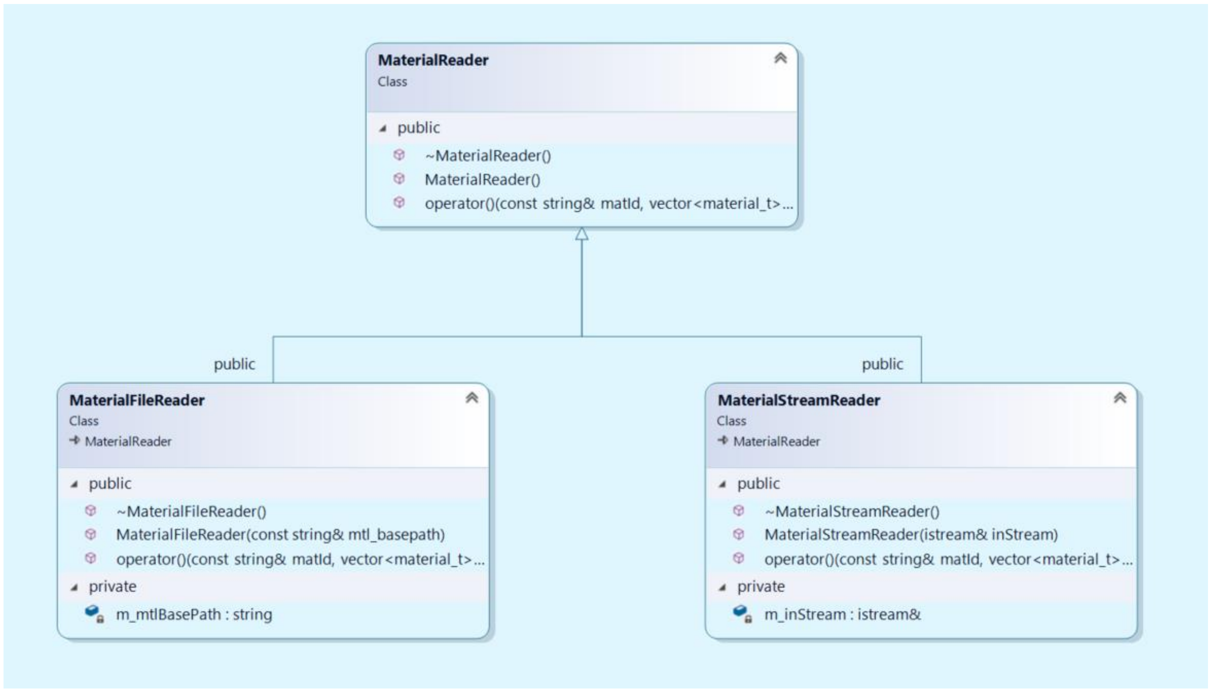


Figure 7 Class Diagram

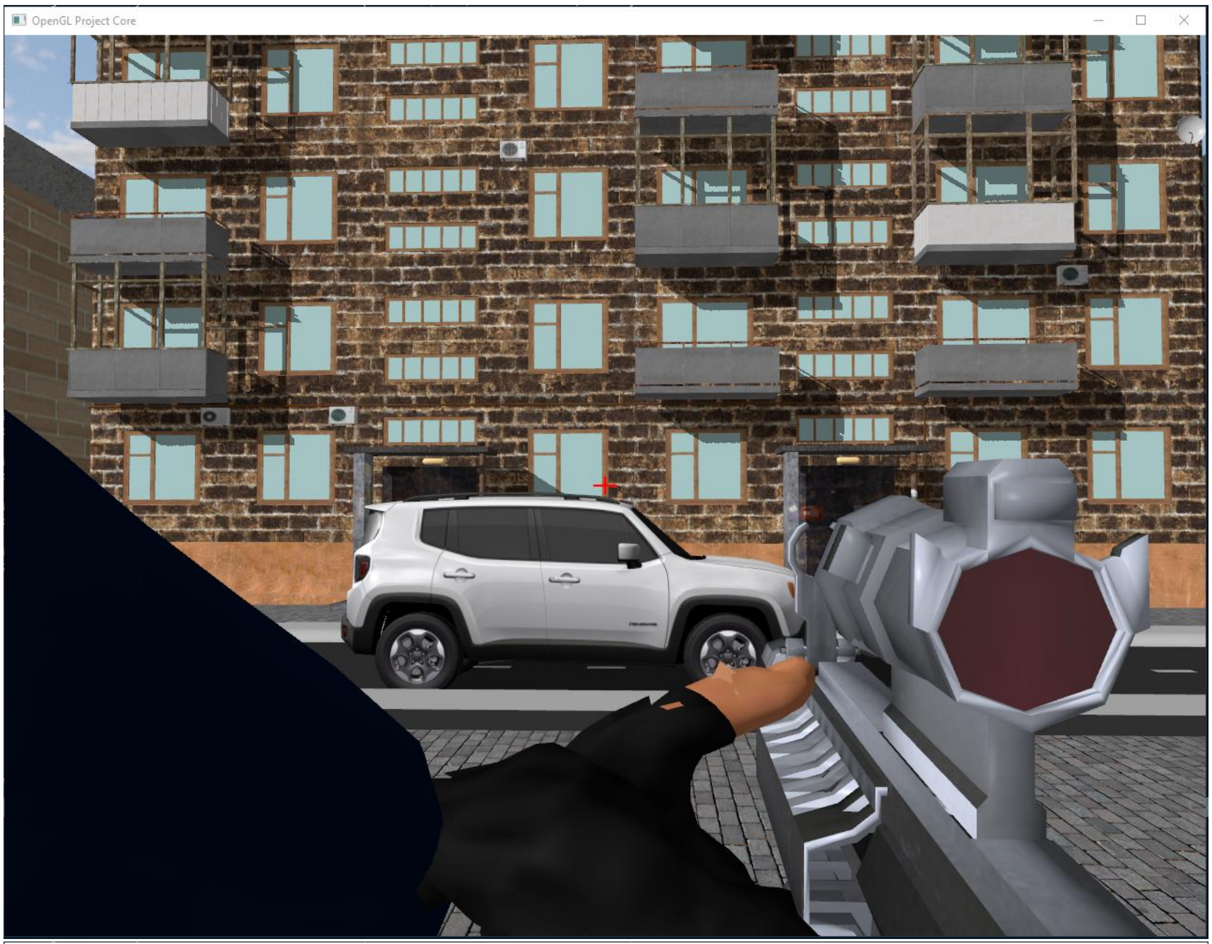
Chapter 5

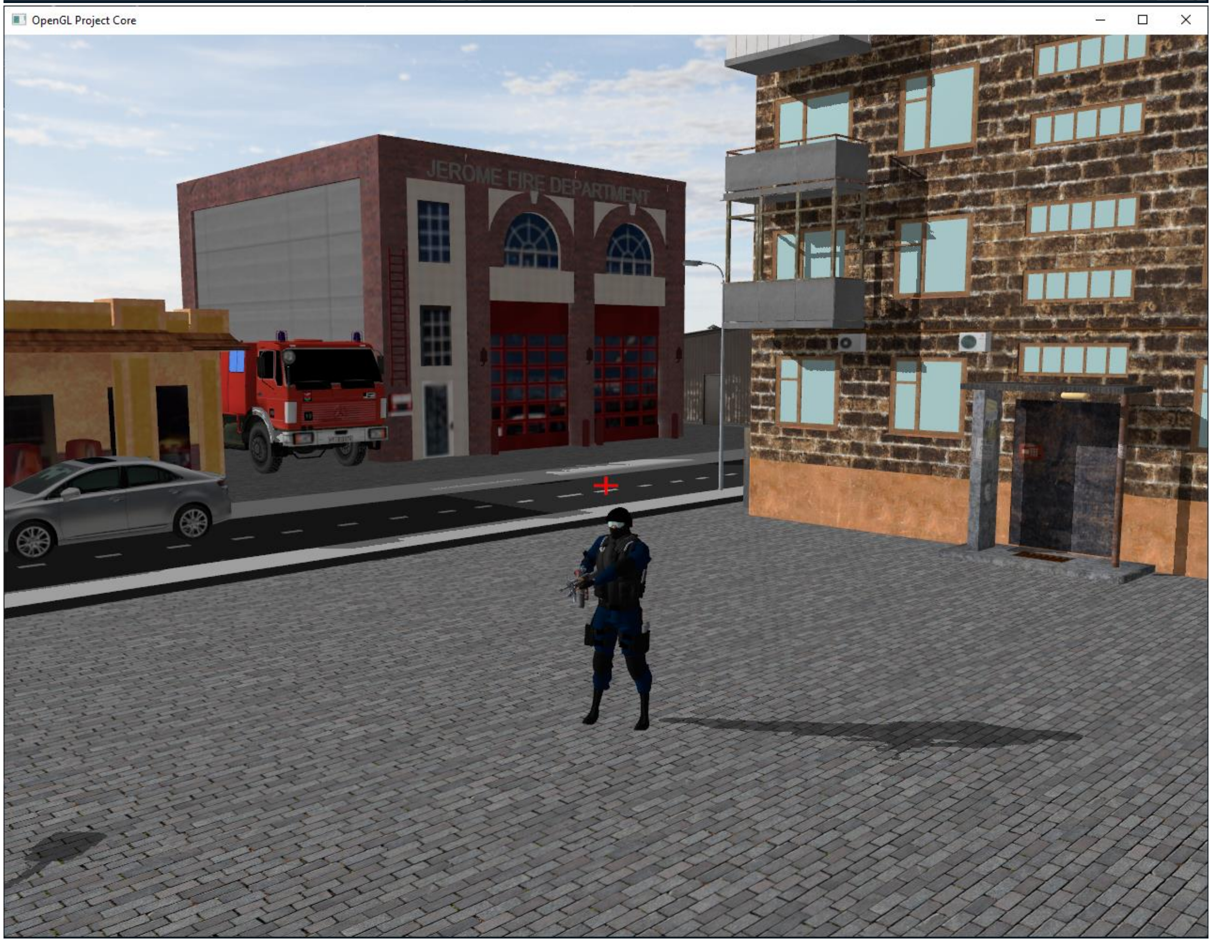
Graphical user interface presentation / user manual

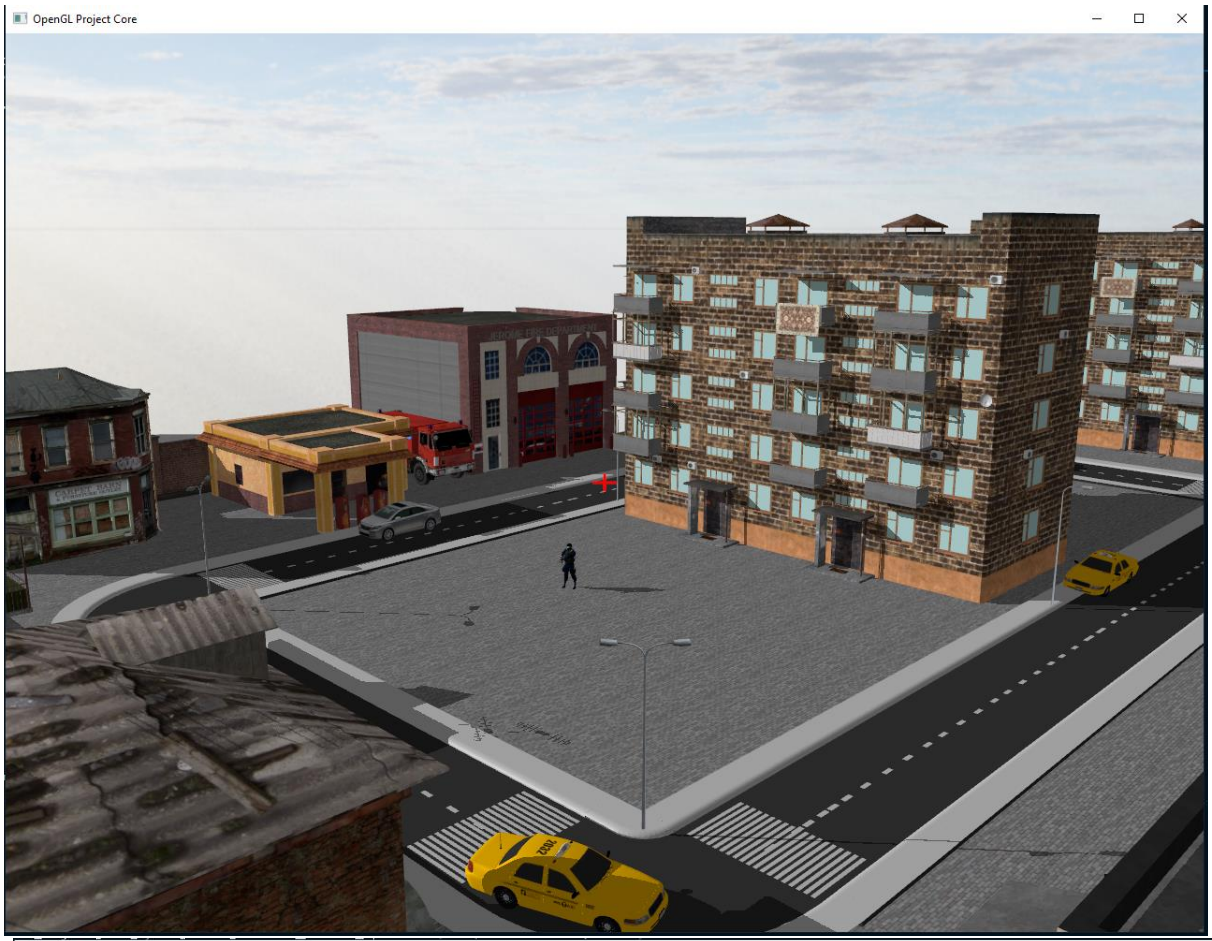
The user can perform the following functions on the scene or on the objects in the scene by using the following keyboard keys:

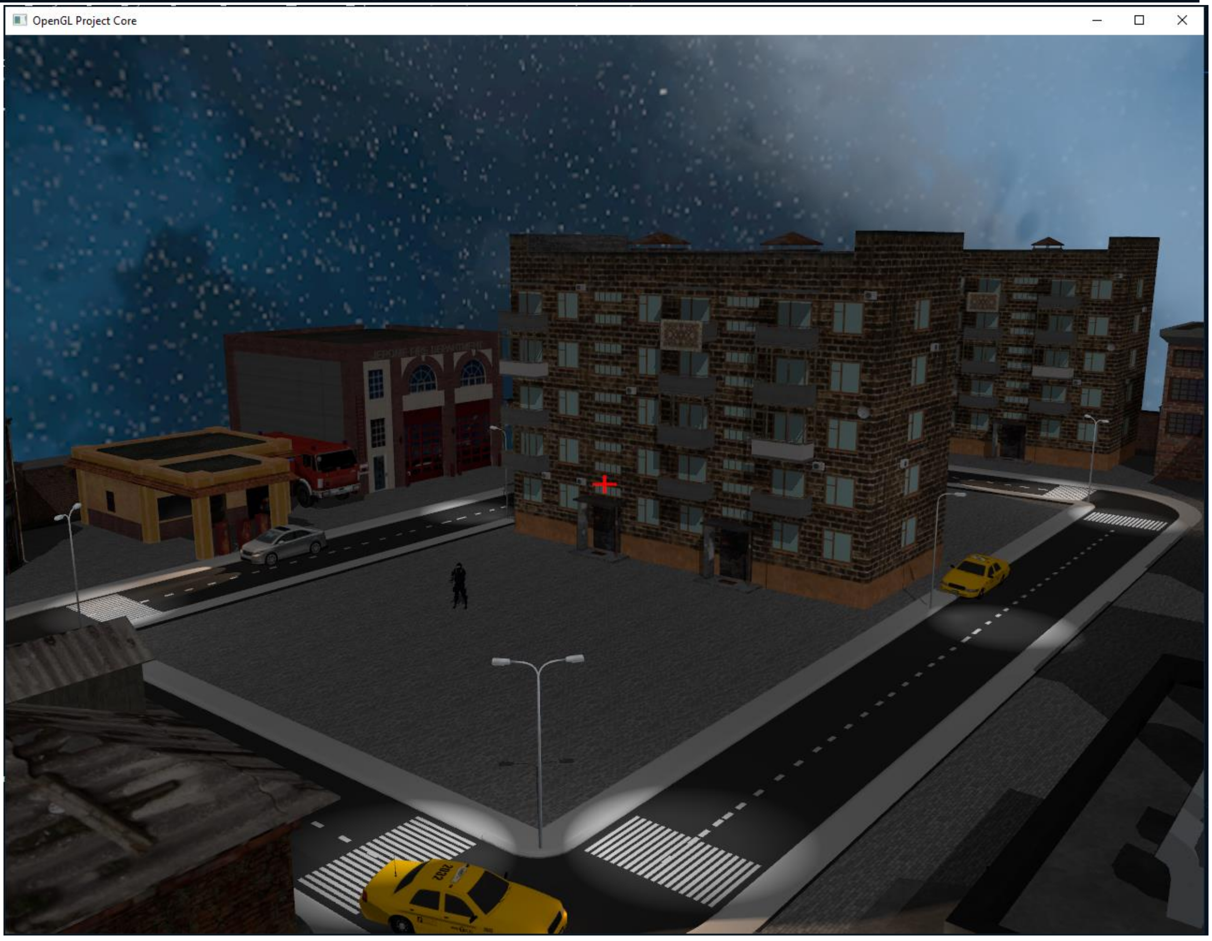
- W, A, S, D – Move the character (in FPP mode) or camera (in Free Cam mode)
- Left CTRL – Duck
- Space – Jump
- E – Move camera Up
- Q – Move camera Down
- B – Switch between FPP and Free Cam mode
- N – Switch between Night and Day
- L – Turn on and off the Lights
- F – Turn on and off flashlight
- M – Show Shadow Depth Map
- O – Switch wireframe mode on and off
- 1-9,0 – Play cool sounds
- Left Mouse Button – Shoot
-

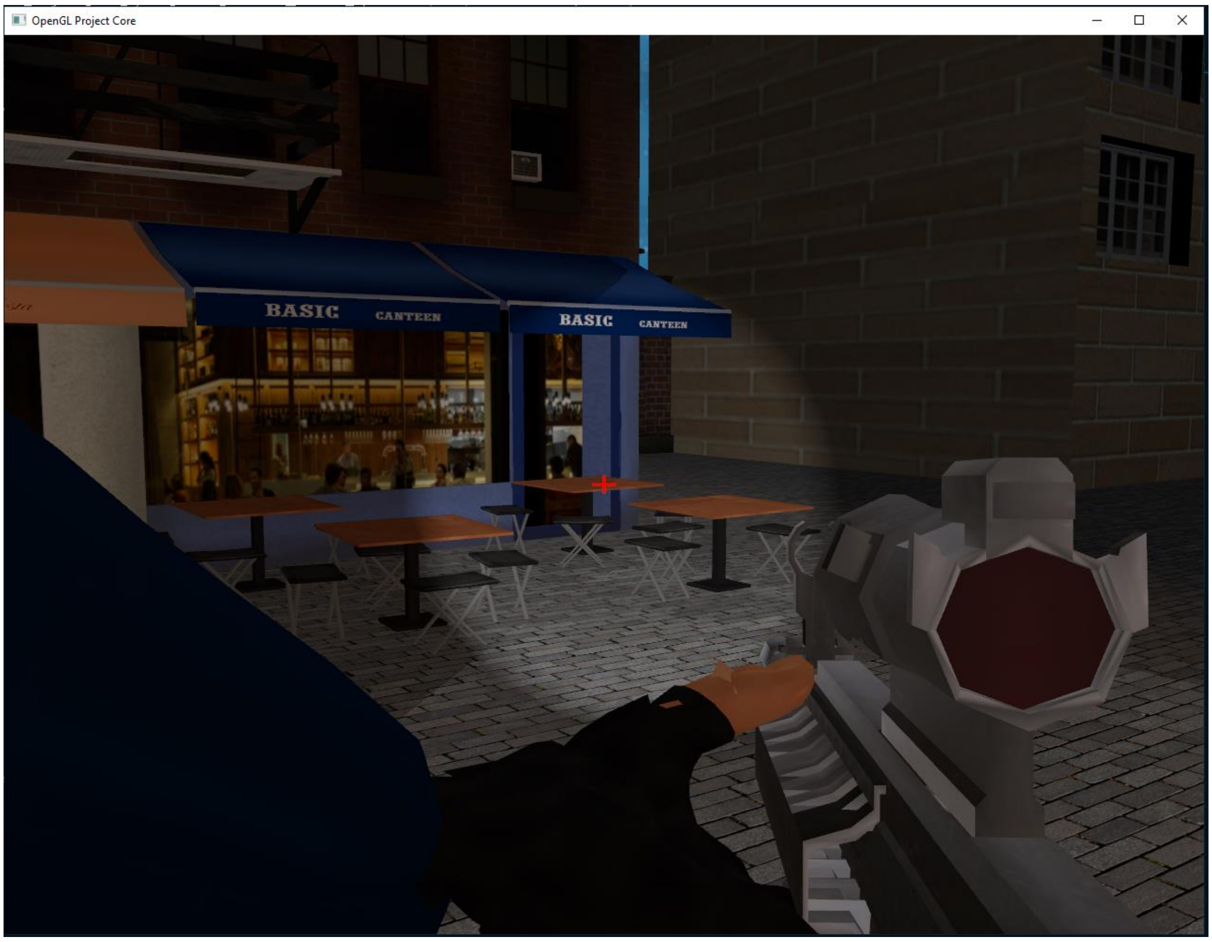
The camera can also be moved by using the mouse.


OpenGL Project Core


OpenGL Project Core
JEROME FIRE DEPARTMENT


OpenGL Project Core


OpenGL Project Core



Chapter 6

Conclusions and further developments

This project acts like a very good introduction to OpenGL development. It helped me familiarize with the steps of implementing an OpenGL application as well as getting used to the graphical pipeline and shaders. Besides this, I have experienced working with three useful libraries: GLM, GLFW and irrKlang. Another good factor was that I also worked in blender a bit which might serve me well in the future.

As for the application that I have implemented, there is enough room for improvement. I am quite satisfied with what I have accomplished but nonetheless, there are some things that should be improved. First of all, object collision is a must and should be taken cared of next. Next, the character should be animated a bit more such as fluid walking and jumping. Having all these mentioned improvements, the “game” would eventually start to resemble what I first dreamed of if to be, namely a simplistic copy of Counter Strike.

Chapter 7

References

- [1] https://store.steampowered.com/app/10/CounterStrike/
- [2] https://github.com/g-truc/glm
- [3] https://www.glfw.org/
- [4] https://www.ambiera.com/irrklang/
- [5] https://www.turbosquid.com/
- [6] https://www.cgtrader.com/
- [7] https://free3d.com/
- [8] https://www.textures.com/