Department of Computer Science
Technical University of Cluj-Napoca

Graphics Processing Systems

Laboratory activity 2019-2020

Project: Star Wars scene in OpenGL

Documentation

Name: Anda Papita
Group: 30431
Email: papitaanda@yahoo.com

Chapter 1: Content

Chapter 2 3
Subject specification 3
Chapter 3 3
Scenario 3
  Scene and object description 3
  Functionalities 3
Chapter 4 4
Implementation 4
  Functions and special algorithms 4
  Graphics model 5
  Data structures 5
  Class hierarchy 5
Chapter 5 6
Graphical user interface presentation / user manual 6
Chapter 6 8
Conclusions and further developments 8
Chapter 7 8
References 8

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.

The scene I created represents a Star Wars scenery which I intended to look like the planet Tatooine. The camera can be moved around the scene by using the keyboard, while some of the objects can also be manipulated by using the keyboard

Chapter 3

Scenario

Scene and object description

The scene is intended to incorporate a scenery from the Star Wars movie franchise, more precisely the planet Tatooine. Thus, resemblances can be found in the two suns, the sand dunes, the pink colored sky, homes with antennas, the crashed ship and the characters of course. On the one side of the scene we can observe the rebel alliance while on the other side there is an army of droids, storm troopers and their space fleet.

Functionalities

The user of the application can:

- Navigate through the scene by the help of keys and mouse
- Go on a virtual tour
- Control the Millennium Falcon ship by making it turn invisible then disappear
- Control the x-wing ship movement
- Control the movement of bb8 droid in all directions by keys
- Control movement of the light cube that produces shadows
- Generate/delete fog

- View the scene in different modes(ex: polygonal)

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

- The movement by rotation of an imperial tie bomber
- The movement by translation of the millennium falcon ship
- A point of light separate of the light cube

Chapter 4

Implementation

Functions and special algorithms

OpenGL Mathematics (GLM) is a header only C++ mathematics library for graphics software based on the specifications of the OpenGL Shading Language (GLSL), well suited for any development context that requires a simple and convenient mathematics library. All the solutions in this project are implemented with the help of this library and its predefined functions such as glViewport(…) used for setting Viewport transform, glfwCreateWindow(…) used for creating the window, glUniformLocation (…), glBindTexture(…).

The renderScene() function is used to generate all the details of the scene and all the animations applied on the objects (both the ones that are independent of the user and the ones that are controlled by the user) such as rotations, translations and scaling.

In addition to this, other useful functions were initUniforms(), where the lights' details are all set and all the uniforms sent to the shaders are initialized. The function void initShaders() is a function where the shaders are instantiated and the function void initModels() is a function where all the 3D Models are instantiated as well.

Moreover, for generating effects such as shadow, fog, light point I implemented special algorithms in the fragment shader.

The fog computation is added in the fragment shader in order to make the scene appear spooky:

```
float computeFog() {
        float fogDensity = 0.01f;
        float fragmentDistance = length(fragPosEye);
        float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));
        return clamp(fogFactor, 0.0f, 1.0f);
}
```

The shadow computation is also added in the fragment shader (computeShadow() function) by using the shadow mapping technique. The scene is rendered from the light's point of view and the depth information is stored in a depth buffer, which is a 2D texture containing the depth of each fragment.

The separate light point is also computed in the fragment shader, which is source of light that lightens the scene uniformly in all directions.

```
vec3 lightDirN = normalize(cameraPosEye - fragPosEye.xyz);
```

Graphics model

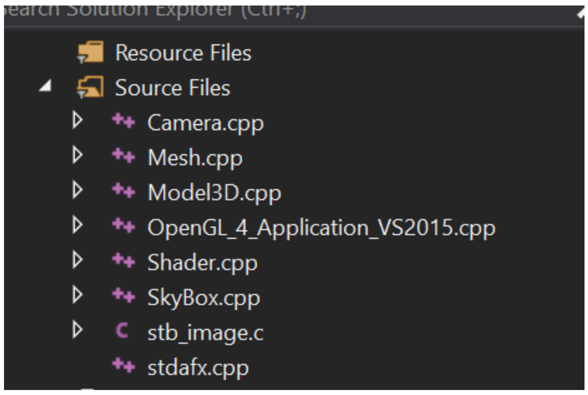
The objects from the scene are 3D Models under the .obj extension which were found online on webpages that allow free downloading of such models. The scene was constructed and designed by using Blender, after which it was exported as a single .obj, except from the separate 3D objects that I wanted to animate.

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), 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.



- Resource Files
- Source Files
  - Camera.cpp
  - Mesh.cpp
  - Model3D.cpp
  - OpenGL_4_Application_VS2015.cpp
  - Shader.cpp
  - SkyBox.cpp
  - stb_image.c
  - stdafx.cpp

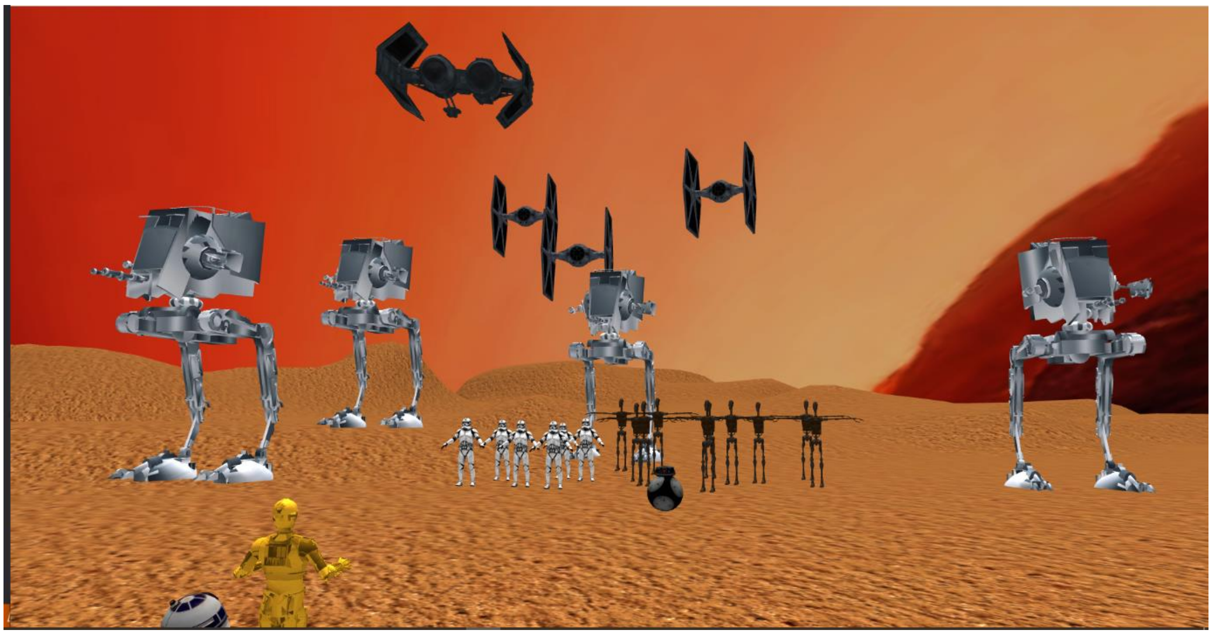
Chapter 5

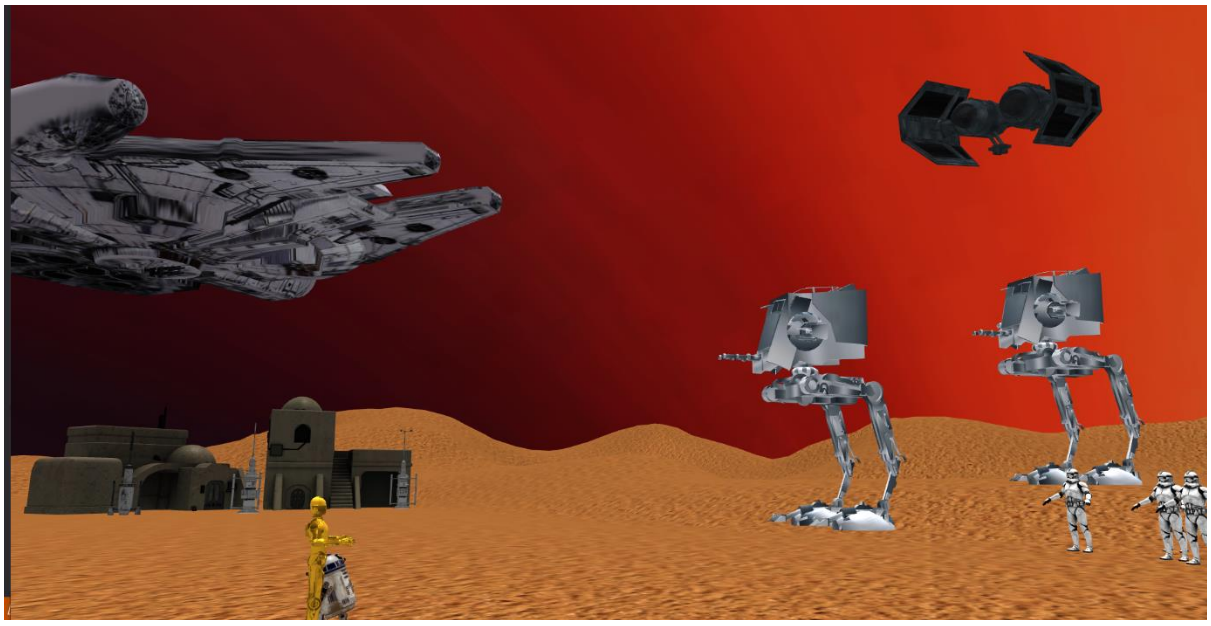
Graphical user interface presentation / user manual

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

- W – zoom in or move the camera forward
- S – zoom out or move the camera backward
- A – move the camera to the left
- D – move the camera to the right
- Q – rotate the camera to the right
- E – rotate the camera to the left
- J – move the light cube to the left
- L – move the light cube to the right
- F – make the fog appear
- G – make the fog disappear
- K – transform the scene in wireframe view and back
- Z – transform the scene in point view and back
- X – Go on a virtual tour
- M – make the falcon ship go with high speed, then turn invisible and disappear
- Up – move the x-wing backward
- Down – move the x-wing forward
- Left – move the x-wing to the left
- Right – move the x-wing to the right
- T,Y,U,I,O – the keys that control the movement of bb8, which can move in all possible directions

The camera can also be moved by using the mouse.





Chapter 6

Conclusions and further developments

The graphics project was a good way to introduce us into working with OpenGL and getting to understand rendering algorithms. In addition to this, it was also a nice introduction to Blender, as it helped us get familiar with using this design tool.

The project could be furthered developed by adding more animation to the scene and other complex effects to make it seem more authentic.

Chapter 7

References

- https://free3d.com/
- https://www.turbosquid.com/Search/3D-Models/free/low-poly
- https://www.cgtrader.com/free-3d-models
- https://sketchfab.com/features/download