Rusu Carla-Maria
Group: 30431

Graphical Processing Systems Project

Contents

Rusu Carla-Maria
Group: 30431

1.Subject specification

The purpose of this project is to create a 3D scene using OpenGL 4.0. The scene must contain animated objects and effects such as shadow and light. The photorealism, complexity of the scene and other factors are to be considered.

2.Scenario

2.1. Scene and objects description

The scene was built in Blender 2.81a by importing and placing objects individually. Two advantages of using Blender are the ability to map textures onto objects (upon exporting the .obj file, a. mtl will be automatically created with the respective textures) and the ease with which the objects can be arranged.

The scene and animated objects were then loaded into OpenGL; They are as follows: the whole scene (a single .obj which contains all of the stationary objects), a boat and its wheel, a crab, a lake, a cloud, a car and a light cube.

The scene contains an island (the terrain), surrounded by water, a skydome and numerous objects placed on it.

A directional (global) light and three point-lights were implemented. The project also contains fog and transparent objects.

2.2. Functionalities

The functionalities included are varied.

The scene can be visualized through the keyboard and mouse or through an animation (a tour of the scene). Objects can be viewed as solids, wireframes or points objects, texture mapping and materials.

Shadow computation was implemented through shadow mapping. Other features are collision detection (the boat can move inside a defined area, namely on the lake's surface and the camera cannot go through the car), fog, multiple light sources and object animations.

3.Implementation details

A demo project provided in the laboratory was used as the basis of the project. Project-specific functionalities were added to this existing foundation in order to obtain the final product.

3.1. Functions and special algorithms

The algorithms implemented in the project are the following: shadow mapping, collision detection, fog generation, scene tour, object generation, and debouncer.

3.1.1. Possible solutions

Shadow generation was done using shadow mapping as explained in the laboratories. It involves rendering the shadows from a directional light's point of view. An alternative would have been point shadows algorithm which renders the shadows from the perspective of a point light.

Collision detection can be done between a point and an object or between two objects. The main idea is to encase the object with which we don't want to collide inside a bounding box or circle. The box is defined by the minimum and maximum vertices of the object on each axis. If the next movement will cause collision with the object, then the movement will not take place.

Fog generation was done as explained in the laboratories, by blending the colour of the fog with the fragment colour based on the distance of the fragment.

The scene tour was implemented as such: choosing a few positions in the scene from which to view specific target points and then moving in a certain way with respect to the targets (go to point, rotate around it or combine the two for an arc-like movement). The tour was done in the rendering part, by moving for a fixed number of renders. Another way to describe the camera movement of the tour is more mathematically, for example through curves.

Object generation is achieved at the press of a button. The renderer draws as many objects of a kind (with slightly modified positions) as indicated by the button.

The debouncer was implemented specifically for the object generation algorithm as the button is sensitive and unreliable. It relies on accepting the object generation command only once every number of renders.

3.1.2. The motivation of the chosen approach

Shadow mapping was chosen because of the existence of a directional light in the scene. Collision detection between a point and an object based on a bounding box has been implemented, as the collision between the camera and a car and between the boat and the margins of a lake area are desired. The scene tour algorithm was chosen due to its simplicity and the debouncer was a necessity.

3.2. Graphics model

The graphics model was rendered using the Blinn-Phong illumination model, using ambient, diffuse and specular lighting. Shadow mapping, fog and transparency (for the water and clouds) were also used in the rendering process. Animations were realised with matrix transformations translate, scale and rotate.

3.3. Data structures

The OpenGL data structures used are: VAO, VBO, EBO AND FBO.

A Vertex Buffer Object (VBO) is a buffer that can also store information such as vertices, normals, texcoords, indicies, etc.

A Vertex Array Object (VAO) is an object which contains one or more Vertex Buffer Objects and is designed to store the information for a complete rendered object.

Element Buffer Object is a type of buffer that lets us re-use vertices to create multiple primitives out of them

The FBO (frame buffer object) is a combination of a colour buffer, a depth buffer and a stencil buffer.

The GLM data structures used are: vec2, vec3, vec4, mat3, mat4.

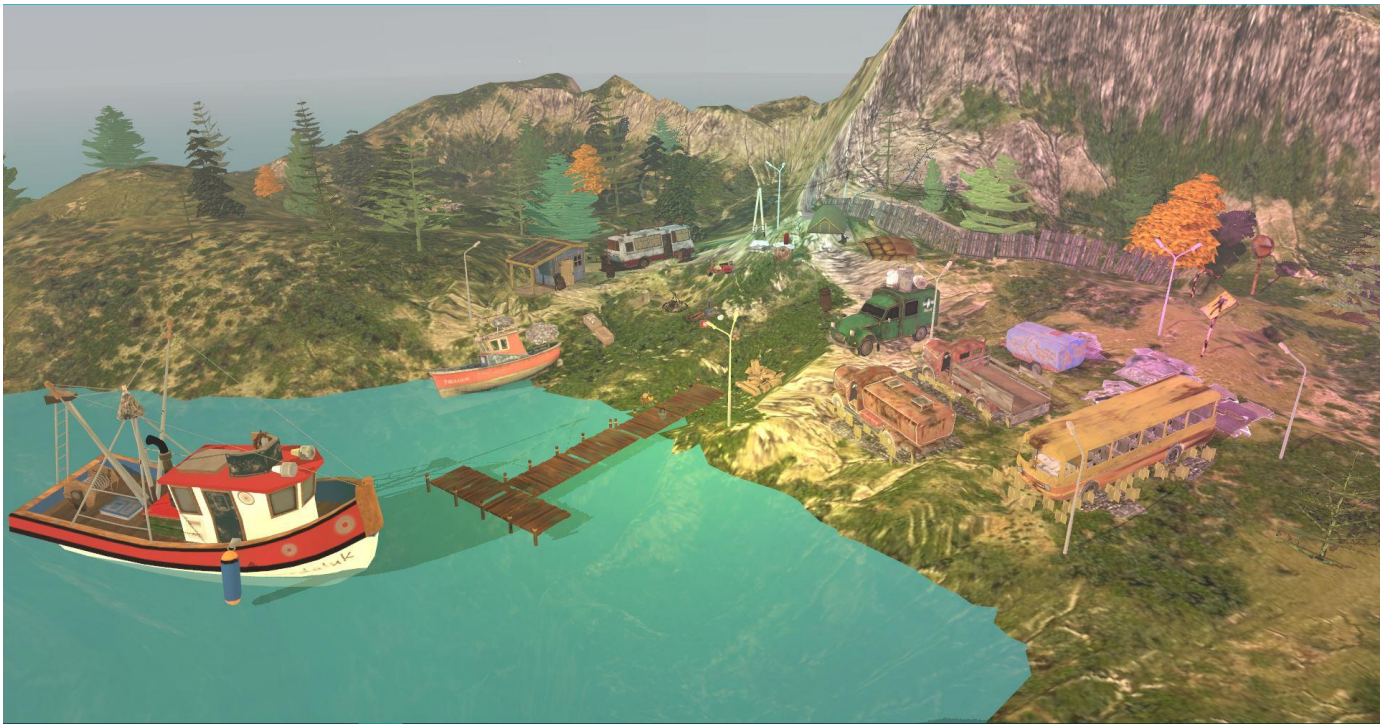
3.4. Class hierarchy

The OpenGL_4_Application_VS2015 is the source code which makes use of the Camera class (handles camera movement and attributes), the Mesh class (initializes buffers and applies textures), the Model3D class (the 3d object class), the Shader class (provides shader functionality).

Rusu Carla-Maria
Group: 30431

4. Graphical user interface presentation/ user manual

The scene contains a single .obj file of the static objects (including terrain, water, trees, defunct cars, shed, camping gear) and separate .objs for the dynamic objects (the boat, the wheel of the boat, the clouds, the crab) and for the objects on which effects are applied (the lake, the car).



The car, while not dynamic, is used to exemplify the collision detection algorithm. It is bounded by a bounding box which the camera cannot pass through.

The lake and the clouds are rendered with transparency, using the alpha channel, to obtain a better looking and more realistic final product. The clouds also move on the X and Z axis (diagonally) until a position is reached.

The crab is animated: it rotates several degrees left, jumps, rotates right, jumps and begins the cycle again.

The wheel of the boat is animated as part of the boat object. It rotates left and right on the Z axis, according to the angle the boat turns to.

The boat can only move in a predefined area on the lake, as collision detection is in place.

There is a directional light with which the shadow mapping is linked and three point-lights (one magenta, one yellow and one cyan).

Key mapping:

- T – enable tour
- Y – disable tour
- G – generate clouds (each will be generated to a slightly different position on the diagonal axis described by Z and X)
- 1 – switch to point view mode
- 2 – switch to solid view mode
- 3 – switch to wireframe view mode
- W – move camera forward
- A – move camera left
- S – move camera backward
- D – move camera right
- LEFT SHIFT – move camera up
- SPACE – move camera down
- M – switch to day mode
- N – switch to night mode
- O – rotate directional light left
- P – rotate directional light right
- I – move boat forward
- J – rotate boat left
- L – rotate boat right

5. Conclusions and further developments

The project involved learning to operate in Blender and OpenGL, thus developing skills in this regard. The result is a relatively complex scene, with a few animations and functionalities implemented with graphical processing algorithms learned from the GPS laboratory over the semester or from individual research.

Some further developments would be: gamma correction, point shadows, normal mapping, parallax mapping, addition of particles, effects such as wind, rain, 2D text interface, billboards, more complex animations and user interactivity (1st and 3rd person viewpoints), a synthetic physics engine (simulate gravity for instance).

6. References

[1] GPS laboratory. 2019

[2] Joey de Vries. LearnOpenGL. https://learnopengl.com/

[3] opengl-tutorial. http://www.opengl-tutorial.org/