Department of Computer Science
Technical University of Cluj-Napoca

Graphical Processing Systems

Laboratory project 2019-2020

Project title: Dreamy Nature Scene

Name: Aldea Roxana Ioana
Group: 30431
Email: aldearoxanaioana@gmail.com

Contents

Chapter 1

Subject specification

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

The main functionalities of the project are:

- visualization of the scene: scaling, translation, rotation, camera movement using keyboard and mouse and using an animation
- specification of light sources (directional and point lights)
- viewing solid, wireframe objects, polygonal and smooth surfaces
- texture mapping and materials
- textures quality and level of detail
- textures mapping on objects
- exemplify shadow computation
- exemplify animation of object components
- photo-realism, scene complexity, detailed modeling, algorithms development and implementation (objects generation, collision detection, shadow generation, fog), animation quality, different types of light sources (global, local)

Chapter 2

Scenario

2.1 Scene and objects description

My scene describes a place in nature, the main attraction being a lake with some swans, surrounded by trees, white flowers, roses and different plants. On the one side of the lake, is presented a house surrounded by a fence, a deck to the lake, a fountain and some barrels in the backyard. Also, flowers, mossy rocks and bushes fill the space. On the other side of the lake, I have designed a forest and some pieces of a mountain in the background. This fairy-like nature piece is animated by two deer, one which moves forwards and backwards, and one which is controlled from keyboards, to move forward and to rotate, left and right. Also, the tail of this deer is animated.

Figure 1: General presentation of the scene

2.2 Functionalities

The user can visualise the scene by keyboards and mouse. With 'W', 'A', 'S', 'D', you can move forward, to the left, backward or to the right and with mouse, you can change the direction of camera.

The scene can be visualised also by a camera animation, by pressing 'I' key. It is presenting a part of the scene and shows its beauty from relevant points.

As I have presented above, one deer can be controller from keyboards, with left, right, up arrows.

The scene can also be viewed as solid, wireframe objects and point objects using '1', '2' and '3' keys.

To increase the photo realism of the scene, I have added shadows to some objects. Also, the light that causes shadows, can be moved by pressing L and J keys, this way, it can be observed that the shadows are modifying as light direction changes.

Besides the directional light, the scene is lighted by five different coloured point lights, positioned in different places, in order to create some fairy effects. In order to highlight the point lights, I have created a night mode which is activated by N key.

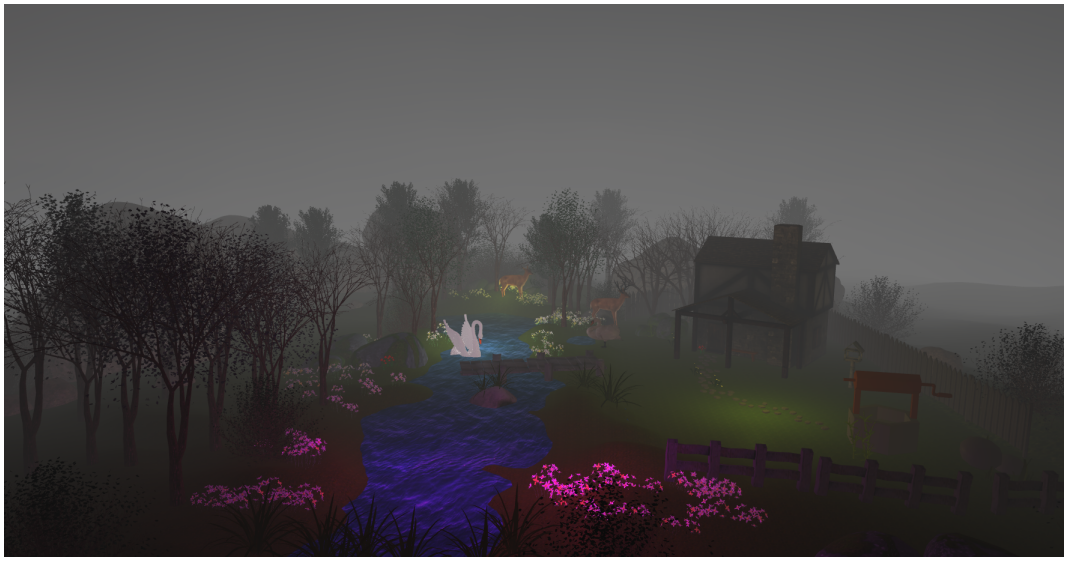

Figure 2: Scene in night mode(1)

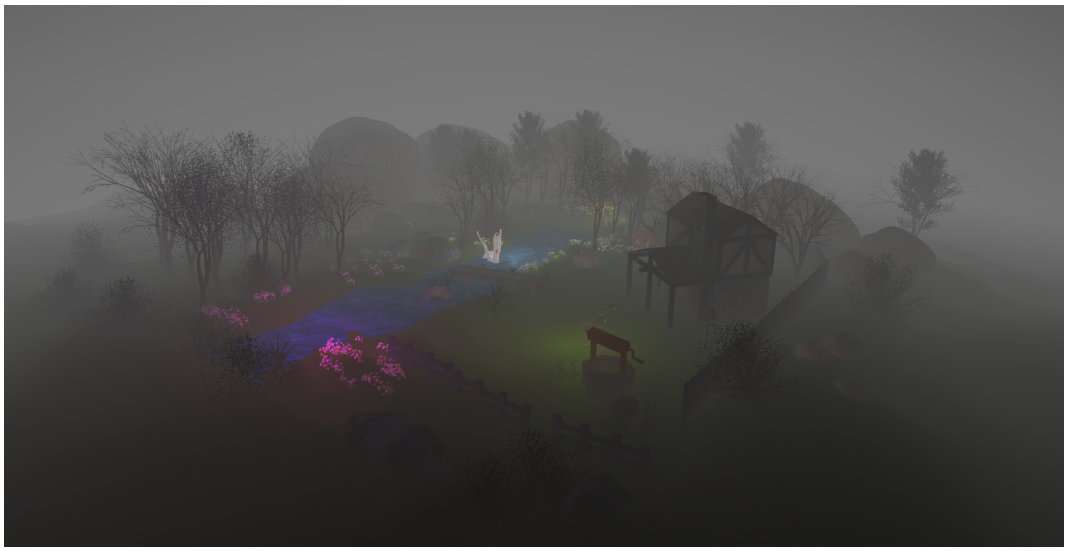

Figure 3: Scene in night mode(2)

Another effects which contribute to the realism of the scene is the fog effect which beautifully covers the nature land at day and at night and collision detection, implemented on camera, in order not to pass through objects.

Another effect is the transparency of water, which can be activated by pressing 'T' key.

Chapter 3

Implementation details

3.1 Functions and special algorithms

The algorithms used are Shadow Mapping technique, fog computation, collision detection, directional and point light computation, deer animations and animation on the entire scene.

For computing the shadow, I have followed the algorithm presented in the laboratory documents. Shadow mapping technique uses depth textures to decide whether a point lies in shadow or not. This way, it consists in two passes. First time, the scene is rendered from the light point of view and the main purpose is to store the depth values of objects in a depth map. This is achieved by creating a depth texture and attaching it to a framebuffer. The second pass consists in rendering the scene from final point of view and in the shader, the current depth values are compared with the depth values from depth map. If they have a depth value grater than from the one from the depth map, it means that object is in shadow.

The fog is computed as in the laboratory, I have a function that computes the fog, then this value is interpolated with the color of the fragment. The fog depends on distance of the view position, the fog effect becoming more powerful as the distance increases.

The transparency is computed by using glEnable() and glBlendFunc(), in order interpolate to the object's color and the current color from the framebuffer. The amount of transparency is defined by the alpha channel from the color representation, simply setting fColor.a with a value between 0 and 1.

The light is computed using Blinn-Phong model, by calculating ambient, diffuse, specular component and adding them, both for the directional light and point lights.

For computing the animations, I have used the transformation functions, glm::translate and glm::rotate. Also, for the scene tour animation, I have used besides camera functions, MOVE FORWARD, BACKWARD, RIGHT, rotate and a function to display the position of the camera, the glfwGetTime() function, in order to move the camera around the scene a limited number of seconds. In order to place the camera in some positions, I have printed the camera position as I move in the scene and choose the desired one.

In case of collision detection, I have added to the Model3D class, other six fields, which keep the minimums and maximums of each dimension. Then, when I want to move the camera, firstly I test if the future position of camera does not make it collide with other object (house, water, swan). The objects for which the collision is tested are stored in an array.

3.1.1 Possible solutions

In case of collision detection, the approach can be improved by storing the minimums, maximums in a Bounding Box class and creating an object for every mesh of the Model3D objects. This way, the collision detection will be more accurate.

3.1.2 The motivation of the chosen approach

Shadow computation, fog generation and light computation algorithms could be found in the laboratory, hence they were implemented in this way (for simplicity). Collision detection was quite easy and straightforward to implement and for camera intro animation, the code may be a little bit lengthy.

3.2 Graphics model

For rendering the scene, it has been used the Blinn-Phong illumination model, besided with shadow mapping technique, fog effect and transparency.

3.3 Data structures

- OpenGL library:
  - Vertex Buffer Object (VBO)
  - Vertex Array Object (VAO)
  - Frame Buffer Object (FBO)
- GLM library:
  - vec3, vec4
  - mat3, mat4
- GLSL library
  - sampler2D
  - samplerCube
- Model3D
- Camera
- Mesh
- Shader
- Skybox

3.4 Class hierarchy

- Camera.hpp/Camera.cpp
- Mesh.hpp/Mesh.cpp - defines the faces of each 3D object
- Model3D.hpp/Model3D.cpp - handles a 3D object
- Shader.hpp/Shader.cpp
- SkyBox.hpp/SkyBox.cpp
- glm.hpp - GLM header
- GLEW.h/GLFW.h - OpenGL headers

Chapter 4

Graphical user interface presentation / user manual

The scene has been designed in Blender. Most of the scene consists in just one object, while the trees, lake, house, deer and swan are independent objects.

As I have previously mentioned, the scene can be visualised in two modes, a day mode and a night mode. This switch is done by pressing N key.

Figure 4: Scene in night mode

Figure 5: Scene in Day mode

By pressing 1, the scene is presented as wireframe mode

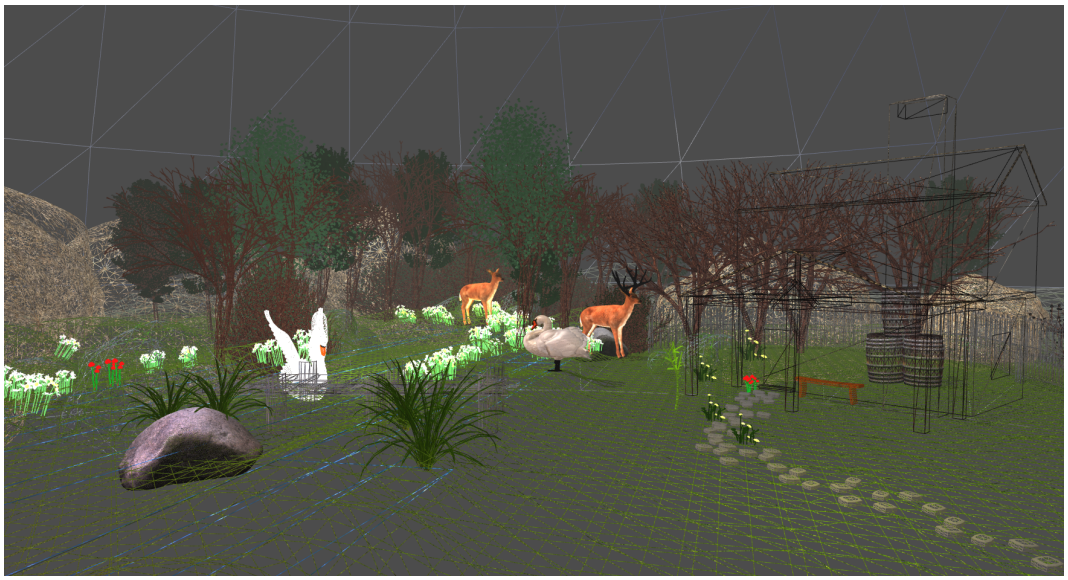

Figure 6: Scene in Wireframe mode

By pressing 2, the scene is presented as point objects

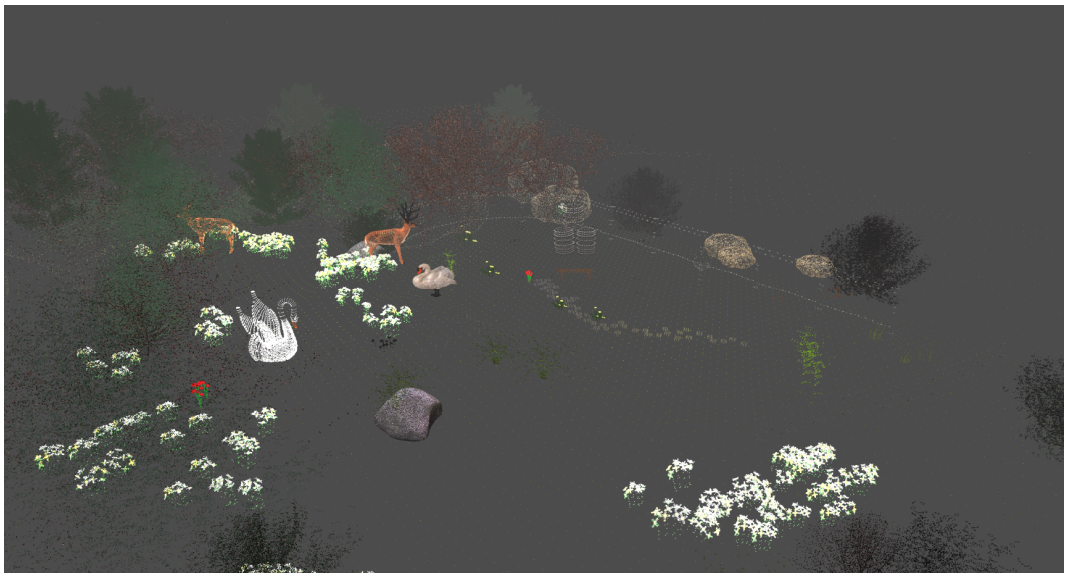

Figure 7: Scene in Point mode

By pressing T, the scene the transparency of water is activated.

Figure 8: Scene with transparency highlighted

4.1 List of all functionalities

- Key '1': Wireframe mode
- Key '2': Solid mode
- Key '3': Point mode
- Key 'W': Camera moves forward
- Key 'A': Camera moves left
- Key 'S': Camera moves backward
- Key 'D': Camera moves right
- Key 'L': Light source moves right
- Key 'J': Light source moves left
- Key 'I': Scene animation activated
- Key 'O': Scene animation stopped
- Key 'N': Day/Night mode switch
- Key 'T': Transparency of water switch
- Key Up arrow: Deer moves forward
- Key Left arrow: Deer rotates to the left
- Key Right arrow:Deer rotates to the right

Chapter 5

Conclusions and further developments

In conclusion, at start it was quite hard to decide on how to design the scene, but in time, adding more and more objects, the scene started to look good and become more and more complex. As I have already said, the scene was designed in Blender 2.7, a tool I have never used before, but after working on this project I have acquired some skills, for example, mapping a texture on an object from scratch, on any face, using sculpt mode, positioning, transforming objects etc. Also, when the scene has become complex and the number of vertices has increased very much, the program started to work hard and it was a burden sometimes. It was also challenging to add lights in OpenGL, to make animations, moreover when the origin of the object was not set in its center.

All in all, it was a nice project and I am quite satisfied with how it looks, but of course it can be improved. There can be added more objects, more animated animals, to increase the realism of the forest, maybe some flying fireflies. The shadows can be computed more accurately or the collision algorithm and animation on the whole scene improved.

Bibliography

[1] Learn OpenGl
https://learnopengl.com/

[2] Moodle, GPS Laboratory
https://moodle.cs.utcluj.ro/course/view.php?id=188

[3] Blender tutorial
https://docs.google.com/document/d/1njtWPMmOQNIaD$_z$9ve8iPRUqQTWdIV$_P$O − NvPD0nOuM/edit?fbclid = IwAR3Ixy2YZYjSZZiDkDryvGJoG − ZIRxPpdNfj383kpxQGgSBVPS0Sd0THA1U