Technical University of Cluj-Napoca

Graphic Processing
Project Documentation

Laboratory Assistant:

Adrian Sabou

Student:

Enoiu Diana-Cristina

Group 30432

1. Contents

2. Subject specification

The main purpose of this project is to develop a photo-realistic, interactive scene with the help of OpenGL library, along with GLM and GLEW. OpenGL is mainly considered an API that provides us with a lot of functions with which we can create graphic scenes and it also helps us communicate with the GPU.

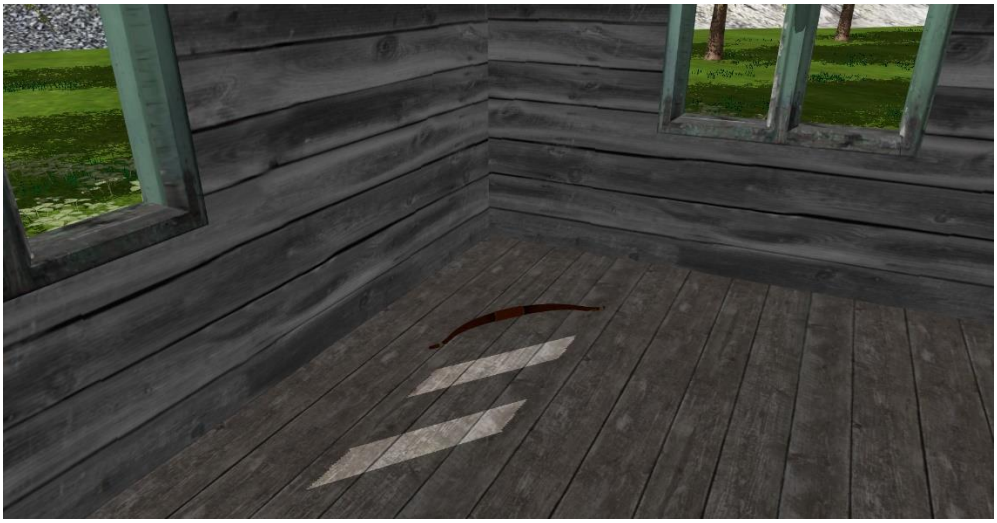
I have chosen to implement an archery simulator, where you can shoot the bow at a target in a scene with mountains, trees, and a small cottage. The final scene should have the main things that I have learnt during the Graphics Processing laboratory, like camera movement, shadows, texture mapping, loaded 3D objects, fog, etc.

3. Scenario

3.1. Scene and objects description

The scene shows a place in a forest, surrounded by mountains. There is also a lake near the mountains, and we can also see a clear sky. Near the forest there is a small cottage, and near the cottage we can see a target. The target is meant for a shooting bow.

We can see a bow inside a cottage, and once acquired we can also shoot with it at a target, seeing the actual movement of the arrow. If we aimed well, the target will disappear and reappear back a bit to the left.



We can also see our scene form different perspectives like night mode, fog enabled, with a flashlight on, or even simulate a day night cycle view.

The trees have some leaves that have some portions that are transparent and not shown, and the same applies for the grass. Otherwise, we will see instead some black portions around the leaves that would not be realistic. The shadow generation also takes into consideration that transparency.

3.2. Functionalities

We can move around the scene, seeing things from a first-person camera perspective. We cannot go through ground but once we reach the stairs of the cottage the camera will go up, simulating the climbing of the stairs. We cannot go through the walls of the cottage as well.

For the moment the bow is not in our hands, so we should go inside the cottage to find it. Once we found the bow, we can then press a special key to show it in our hands and then we can shoot with it. We can position in front of the target and once the arrow that is shot intersects the target, the target will disappear and reappear back a bit to the left.

The setting of different modes to our scene like show shadows, show fog, or even a night mode is also possible. In night mode, we can walk around with a flashlight, and the sky changes to black. For a more realistic view of the scene a simulation of a day night cycle is also implemented, where the light form the sun moves across the sky, as seen from the shadow and at the sunset we enter in night mode.

Different key bindings were also added for the visualization of line view and point view.

4. Implementation details

4.1. Functions and special algorithms

4.1.1. Possible solutions

Main functions that are used in this project:

- `void renderScene()` – the main function that will render our scene every frame, it considers if we have a shadow computation or not (to render the shadows as well), and then it calls a render function for each imported object, that will render the object based on its translation rotation and scale transformations.
- `void initOpenGLWindow()` – to initialize the window dimensions to 1920 x 1080
- `void initOpenGLState()` – initialize for opengl
- `void initModels()` – loads the models from their specific path
- `void initShaders()` – initialization for the two shaders "basic" and "shadow"
- `void initSkyBoxShader()` – initialization for the third shader "skybox"
- `void initUniforms()` – establish the communication between the gpu and the cpu by sending data to the shaders from the application
- `void initFBO()` – FBO is used to compute the shadows
- `void processInputs()` – used to process the inputs from the keyboard
- `void setWindowCallbacks()` – used to call the main callback like, mouse callback, window callback etc.

For the loading of the 3D models I have chosen to use Assimp.

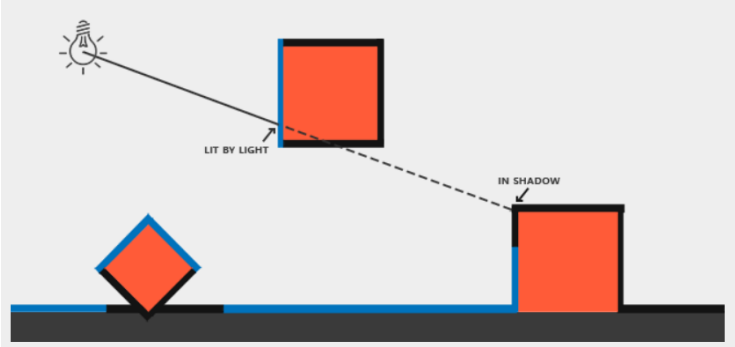
- `void Model3D::RenderModel(gps::Shader shaderProgram)` - the function that will call the mesh to draw the model with the corresponding texture
- `void Model3D::LoadModel(const std::string& fileName, const std::string path, bool transparentModel)` – the function that does the loading of a 3D model, it also takes into consideration if it is an transparent model, to load the texture with different parameters
- `std::vector<Texture> Model3D::loadTextures(aiMaterial* mat, aiTextureType type, std::string textureName)` – the function that will load the textures
- `GLuint Model3D::ReadTextureFromFile(const char* file_name)` – this function actually reads the texture from the specified file

To detect whether or not I am in a specific area, for the collision detection of the cottage I have used a simple formula to check whether or not a point is inside a rectangle i.e. 0 <= dot(AB,AM) <= dot(AB,AB) && 0 <= dot(AD,AM) <= dot(AD,AD)

- ```
  bool Collision::checkIfPointInsideRectangle(glm::vec2 A, glm::vec2 B, glm::vec2 D, glm::vec2 M) {
  //point M is inside a rectangle if (0 < AM.AB < AB.AB) && (0 < AM.AD < AD.AD)
  glm::vec2 AM = M - A;
  glm::vec2 AB = B - A;
  glm::vec2 AD = D - A;
  if (0 < glm::dot(AM, AB) && glm::dot(AM, AB) < glm::dot(AB, AB) && 0 < glm::dot(AM, AD) && glm::dot(AM, AD) < glm::dot(AD, AD)) {
  	return true;
  }
  else {
  	return false;
  }}
  ```

For the collision of the arrow with the target I simply checked depending on the position of the target at a certain point if the arrow is inside some boundaries.
  ```

For the computation of shadows, I have used the same algorithm as presented in the laboratories. We move the camera to see from the light perspective, everything that we see is bright and what we don't see is in shadow. This technique is called Shadow Mapping.



For the point light I had to compute the attenuation value, calculated by the formula presented in the laboratories.

$$Att = \frac{1.0}{K_c + K_l * d + K_q * d^2}$$

For the spotlights I have used the attenuation but also a theta value to determine whether the fragment is lit by the spotlight or not.

To calculate the fog I have used the squared exponential fog.

$$fogFactor = e^{-(fragmentDistance * fogDensity)^2}$$

To determine whether a fragment from the transparent object is shown or not I verify the alpha value and if the alpha value is smaller that 0.4f we discard that fragment.

4.1.2. The motivation of the chosen approach

For this project we have received a project core, that we had to use to implement our project. It already have some basic shaders implemented , a directional light and the Model3D and Mesh classes used to load a 3D model. However I have changed the Model3D class to use the Assimp library instead.

I have chosen to use Assimp as the main library to use to load 3D models because it allows me to load more then .obj models if I need to. It will also allow me to load animations with .dae format if I want to do so in the future. It was also challenging to get Assimp working, but now I have a better understanding of what happens when we want to load a 3D model.

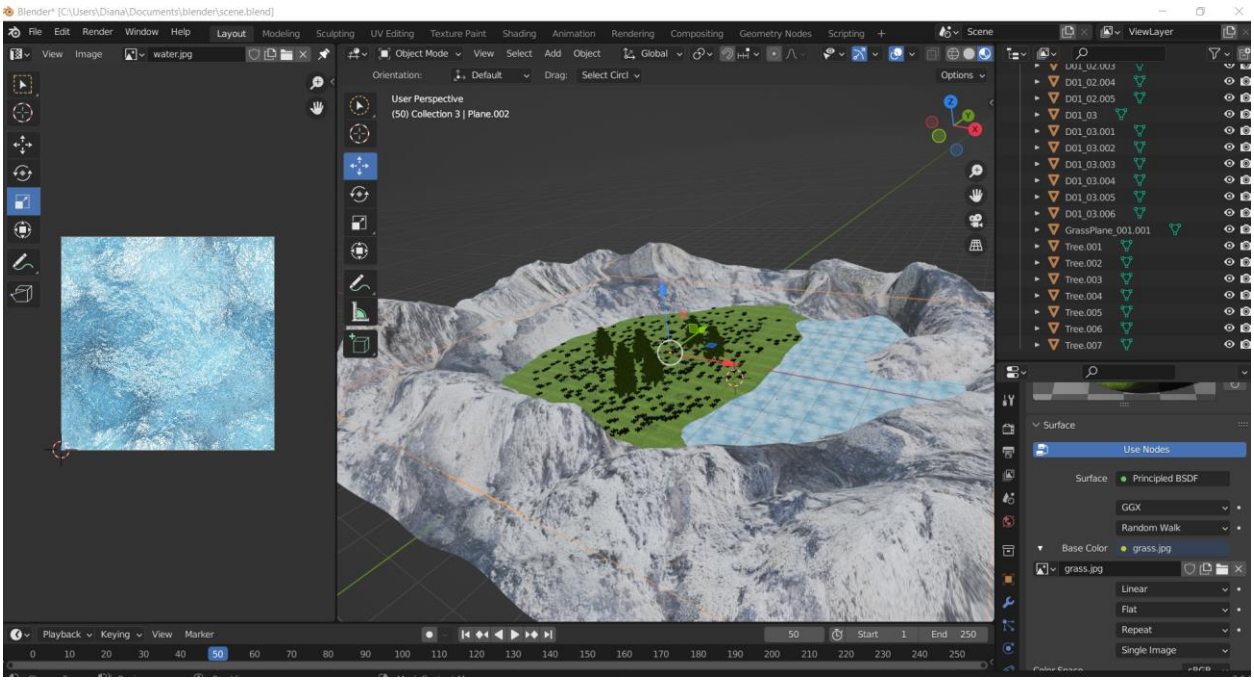
For the collision detection I have used the above algorithm because it was fast and it is also useful for static objects where I know the exact position of each wall. I could not use this algorithm for the target because the target is moving, so I needed a more relative way to detect the collision.

The rest of the main techniques to achieve shadows, fog, point lights, transparency, directional light etc. are the same as presented in the laboratories. I preferred to implement them this way because I have understood the techniques from the laboratories.

4.2. Graphics model

For the 3D models I have loaded some objects from the internet into the Blender, and then I exported them as .obj models, keeping also the position, scale, and rotation done in Blender.

The scene generation with ground, mountains, and lake, is done completely in Blender, I did not loaded anything from the internet. The textures for the scene were also custom.

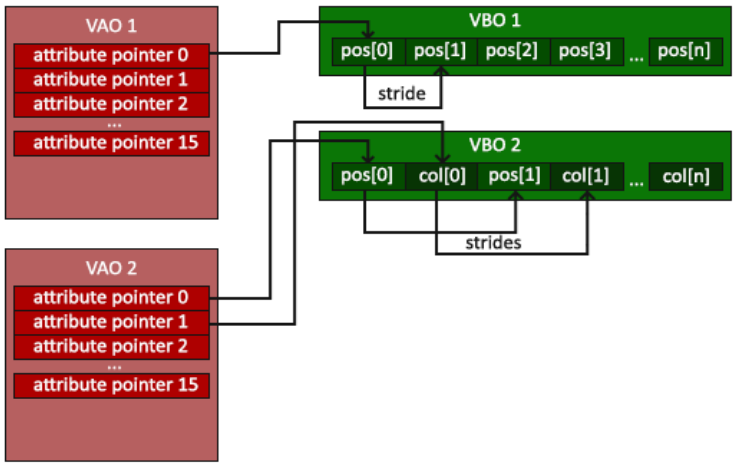


4.3. Data structures

The main data structure used is provided by GLM and it is a vec3. This is used to tell us the position of a point in a XYZ coordinate system and, also, to tell us the direction of a vector. We used it for almost everything that we want to generate in our scene.

Besides that we have used arrays and vectors to store multiple elements.

We have also used VAO and VBO when loading 3D models. A VAO holds the data about vertices positions, their normal and texture coordinates to send it to the shaders. A VBO makes the binding of this data. A FBO is also used to render the depth map.



Inside the mesh class we have 3 data structures i.e. Vertex that stores position, normal and texture coordinates, Texture, to store id, texture type and path, and, also, the Buffers to store VAO, VBO and EBO.

4.4. Class hierarchy

- **Camera class** - used so that we can move across the scene. This actual implementation looks like we are moving in the scene but what it actually does is that it moves the scene accordingly to create this effect.
- **Model3D class** - this class is used to load the 3D models from the disk using the Assimp library
- **Mesh class –** used to draw the opbejcts depending on their position, normal and texture coordinates. Also used to setpu the VAO, VBO and EBO.
- **Shader class –** used to create the shaders.
- **Skybox class** – used to help us generate the skybox.
- **Window class** – used at the generation of the window.
- **Collision class** – used to implement the collision detection.
- **Main –** the main application where everything happens.

5. Graphical user interface presentation / user manual

The user can interact with the scene using the following keys:

- W, A, S, D – to move the camera forward, backwards, and left to right.
- KP_ADD – to show the bow, if acquired from the cottage
- KP_SUB – to not show the bow anymore
- KP_MULTIPLY – to use the flashlight

- KP_DIVIDE – to stop using the flashlight
- Z – to print the position of the camera
- O – to show the shadows
- P – to stop showing the shadows
- K – to show fog
- L – to stop fog
- N – to enter night mode
- M – to exit night mode
- KP_1 – to start day night cycle
- KP_2 - to stop day night cycle
- KP_7 – to show wireframe view
- KP_8 – to show point view
- KP_9 – to go back to solid, smooth view

Using the mouse the user can rotate the camera just by moving it, and it can also shoot the bow if it press the left-click button. To acquire the bow the user does not have to press any key, it just have to find the bow location and to cross over it.

6. Conclusions and further developments

This assignment taught me a lot of interesting things about how to generate a scene using OpenGL. I have realized that OpenGL is a powerful API with which we can do a lot of things and is also cross-platform. I have learned the basics of Graphic Processing, but also some more advanced notions like shadow generation. It might help me in the future if I want to develop games using Unity or Unreal Engine, because they both use OpenGL as their rendering pipeline. This assignment was also the first to teach me how to program a GPU. This project was also an introduction on using Blender.

This project can be further developed for example by generating some sort of particle effect when hitting the target with the arrow, or to implement rain and wind. I could also add more objects into the scene or make the lake have some reflective properties to reflect the sky. The day night cycle can also be improved by taking into consideration the sunset light or morning light colors and I could also add a smoother transition between the two skybox textures.

I conclusion, I have enjoyed a lot working at this project, and I am glad that I had the opportunity to learn OpenGL.

7. References

https://learnopengl.com/Introduction

https://www.youtube.com/ - Blender tutorials

https://www.humus.name/index.php?page=Textures

https://www.cgtrader.com/free-3d-models

https://www.turbosquid.com/Search/3D-Models

https://free3d.com/

https://sketchfab.com/features/free-3d-models

Courses and Graphic Processing laboratories.