**Graphical Processing Systems**

**Project Documentation**

Loga Darius

Group 30432

Technical University of Cluj - Napoca

Contents

1. **Subject Specification**
2. **Scenario**

2.1. Scene and objects description

2.2. Functionalities

**3. Implementation**

3.1. Functions and special algorithms

3.1.1. Possible solutions

3.1.2. Motivation

3.2. Graphics model

3.3. Data structures

3.4. Class hierarchy

**4. Graphical User Interface**

**5. Conclusions and further developments**

**6. References**

1. **Subject Specification**

The subject that is to be specified is one project in photorealistic presentation using the OpenGL library for 3D objects. One can navigate through the scene using input devices and also manipulate them. The maximum requirements for this project are taken from the moodle page of this course and are as follows:

* Visualization of the scene: scaling, translation, rotation, camera movement using input devices and animation
* specifcation of light sources (minimum two different 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, rain, wind), animation quality, different types of light sources (global, local, spotlights).

1. **Scenario**

2.1. Scene and objects description

The scene from this project resembles stories from the fairy tails or sci-fi domain. It is the middle of nowhere, the texture from the skybox indicates that are castles nearby, close to the sand dunes and other trees. The window application opens with a nice view of one cozy farmhouse, surrounded by two dogs, an iron with bricks fence, two lines, symmetrically oriented, with three different species of trees, street lamps and above us it passes through an airplane. One of the dogs is moving towards the other one and the street lamps have a yellow-white colour at night time because it exists day/night cycle. The airplane disappears at certain time, but it can be restarted and placed at the original position. Below I will provide pictures with the objects and day/night cycle. There is also fog in the scene where it is getting dense during nighttime and also the ground it has a rocky aspect.



**Fig. 1** –Farmhouse, one of the dogs and rocky ground.



**Fig. 2** – Fence from nighttime.

**Fig. 3** – The airplane seen from below during nighttime.

**Fig. 4** – The set of three trees.





**Fig. 5** – Thin fog, castle from the skybox, view with all elements.



**Fig. 6** – Street lamp during the day. **Fig. 7** – Street lamp during the night

2.2. Functionalities

There are all sort of functionalities, not only the one from the input device, like keyboard and mouse where we press a key to move the camera left, right, forwards, backwards, rotate with certain radians the generated shadows from the directional light source, reset the animations for the walking dog and also for the train, start the animation of a walk simulation around the house without intersecting other objects and also to view the wireframe of objects.

Despite these, the scene also consists of day/night cycle. This cycle happens because of the change in the direction of the sun light source. There is also fog with different density levels and street lamps that illuminate on night, collision detection for not going into other objects, but also the field of view can change with the scroll of the mouse.

**3. Implementation**

3.1. Functions and special algorithms

3.1.1. Possible solutions

* Shadow computation

One of the most commonly used method for computing is Shadow mapping. Out there exists a lot of possible solutions for this problem with their advantages and disadvantages.

The Shadow Mapping computation is a multi-passes technique that uses depth texture for deteming whether a point is in shadow or not, from the light’s point of view, not by final location. One, out of the two main passes, is used for rendering the shadows firstly from light’s view and then rendering from the camera’s position.

* Light implementation

Here we can implement three possible solutions to this problem. The first algorithm is based on the Gouraud model for enlighten the vertices, in the vertex shader, and exists three types of light: **ambient light** that does not come from any other source and it does not depend on viewer’s position, being all over the scene. **Diffuse light** is evenly distributed in the scene and depends on the surface normal and the direction of the light without the viewer’s dependency and **specular light** that is reflected by surfaces that tend to have properties that replicate mirror effects, property of shininess.

The second one is the Phong model, similar to the Gouraud’s model, but the computation of lights is done in the fragmant shader. This is a significant change, but the amount of calculation to more elements, than in the vertex shader, result in a slow approach, still letting Gouraud’s model to shine. **[3]**

But an improved version of the Phong model is the Blinn-Phong model, present in the fragment shader, similar to the Gouraud’s one, is available using a halfway vector between view and light direction instead of an reflection one so that if it is closer to the surface normal, the higher the specular contribution it is for an easier computation. **[3]**

* Animation

There are several ways to generate animations, most noticiable, for a better realism, is to divide objects in smaller parts and compute them separately. But for the camera standpoint, the two most important methods are by points interpolation. For the linear points, it is sufficient to have liner interpolation, but when we want to take corners or follow a nonrectiliniar trajectory, we need to use the De Casteljau algorithm that have a smooth path along the Bezier curves.

* Fog Computation

Rain and snow need to have particle generation in order to be well computed for a higher realism, but fog is just a simple function in the fragment shader. The three methods for creating a fog efect are: liner, exponential and square exponential. Liner method uses linear interpolation for blending object’s colour with fog colour, the exponential method adds the viewing distance a key factor in the computation, and the last one is just much faster and quicker than the classical one.

* Skybox generation

The background of a scene is so important in most cases, and by using one that uses a cubemap, where anyone can draw inside a cube and this map can apply different textures to those parts to obtain a 3D background that can be very easily sampled by using a direction vector.

* Collision detection and camera correction

To have such restrictions as not intersecting an object with the camera view is by constantly checking the camera movement but also, we can store the bounding boxes of different objects not to interesct themselves.

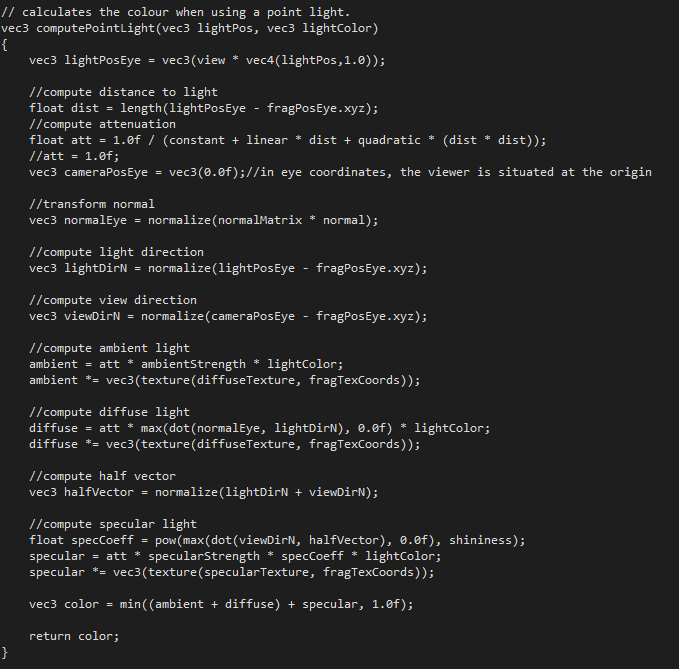
3.1.2. Motivation

* Shadow computation

One of the most common and efficient way was presented in my laboratories session’s in which shadows are rendered bases on the light’s point first and then on camera’s position. A depth map is created to determine the vertices in the shadow by comparing them with depth values in the map. Textures are obtained via FBO, framebuffer object. Not that hard to implement and also, for the day/night cycle, the shadows are generated based on the camera target and directional light source making them move at any time the light source is moving. Shadows can be seen in the images posted above.

* Light implementation

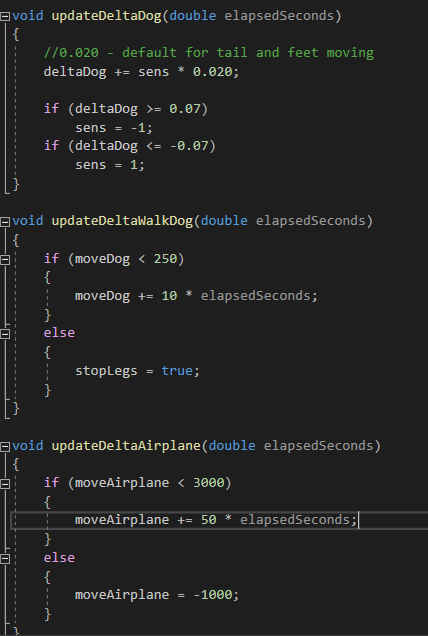
Even though the light implementation using Gouraud method remains the most efficient and practical one, the Blinn-Phong method is the one that takes the realism part seriously. In this project I used two types of light sources: directional and positional using the Blinn-Phong model. The directional one, which is the sun, is located at infinity, having parallel rays and we can generate the shadows for each object and two positional lights represented by the street lamps that are static and can be seen at their peek in the night time. Images with those lights are in the above chapter.

**Fig. 8** – Point light implementation. **[2]**

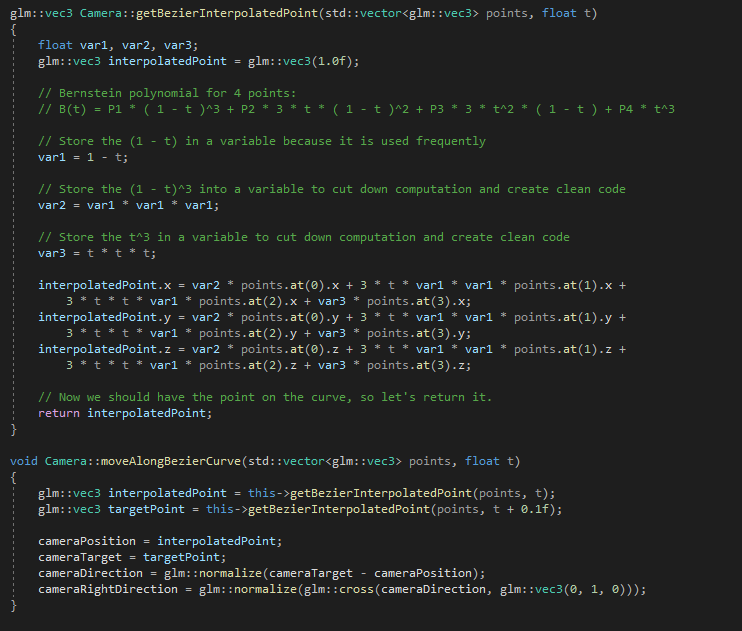
* Animation

Several different animations techniques I used for this project, for example: the airplain that updates every move using a variable that increments, translations and the variable can be somehow resetted. One more interesting animation is the one that the brown dog performs. Some parts were cut in **Blender** to increase the realism, and the two pair of feet, body and tail are different parts and move independently using translations and rotations. The tail moves constantly, when he stays, or even when he moves.

The other important animation method is the De Casteljau’s method using the Bezier curves to interpolate points. This is not a linear interpolation, so we need those curved points to have a smooth animation and not to intersect any objects. The algortihm needs to have some control points that can be obtained from the approximated corners’ location of the interior garden surrounding by the fence.



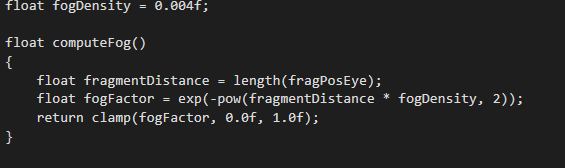
**Fig. 9** – Dog and airplane animation code. **[2]**



**Fig. 10** – Functions to obtain the points and move along the curve. **[1]**

* Fog Computation

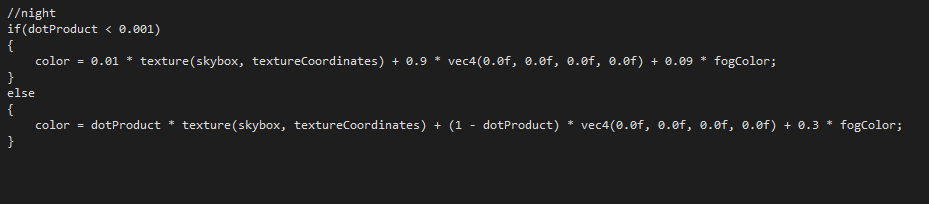
I only implemented the fog using the square exponential method based on the texture’s colour and dependency on viewing distance because when we take a significant distance from the house, we can see that the image is blurred just to increase the realism in the scene.



**Fig. 11** -Fog computation. **[2]**

* Skybox generation

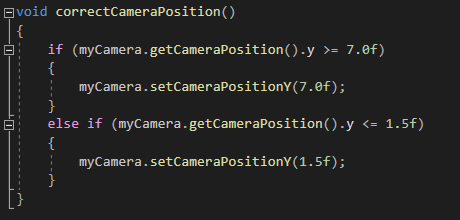
The background used in the project adds a little note to the realism part and also for a good presentation of the scene. Like I mentioned earlier, to create a magical space, we need a suitable texture for a skybox and also this changes the colour depending on one of the periods during this time cycle.



**Fig. 12** – Skybox color during cycles.

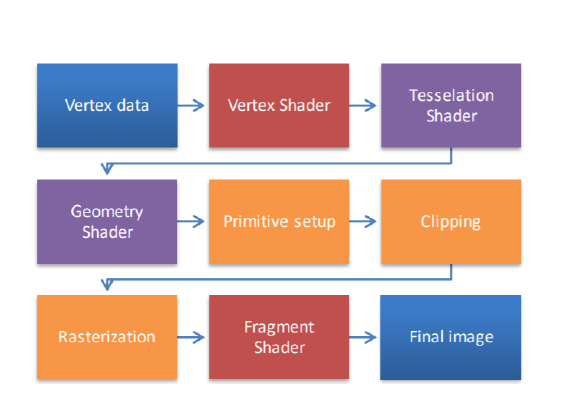
* Collision detection and camera correction

For this part the underneath function was implemented not allowing anyone to go below the ground level, and also not to fly, just to be a really tall person, it is closer to the real world and the collision detection was implemented using the bounding box coordinates restricting anyone the acess to the house when they collide.

**Fig. 13** –Function for collision detection with air and ground.

3.2. Graphics model

The project was implemented using C++ and the OpenGL library (version 3.2), in the Visual Studio IDE. OpenGL is considered an Application Programming Interface (API) and is used to develop graphical applications by accessing features available in the graphics hardware **[2]**. The first versions of OpenGL used a fixed function pipeline, but this version that I have used uses a programmable pipeline which is more effcient. Is with ease to develop on every operating system because of the cross-platform support. Writing small programms (shaders) we can extand the functionalities of our GPU and they are also attached to an OpenGL application. They use OpenGL Shading Language (GLSL) similar cu C language. [2]



**Fig. 14** – OpenGL graphics pipeline.

3.3. Data structures

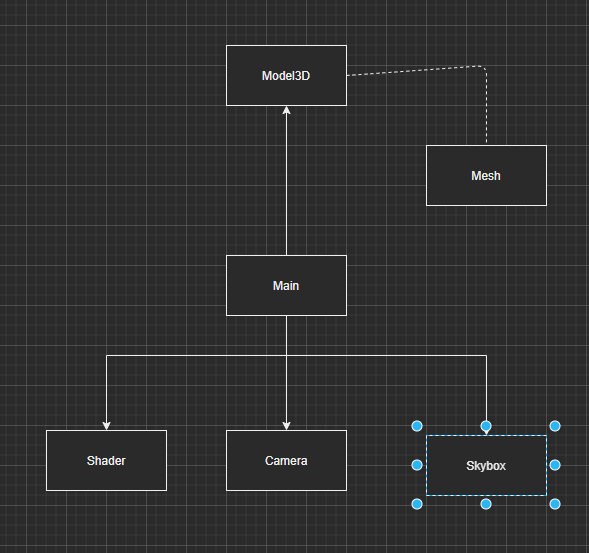
The implemented classes for data structures are: **Camera**, **Model3D**, **Shader**, **Skybox** and **Mesh**.

The **Camera** class has private instance variables for camera position, target, direction, right direction and up vector. I also use the mouseCallback() function for rotation and also the camera should move so smooth when we use the yaw and pitch values. In this class, I implemented functions for calculation of interpolated points and moving along them.

The **Model3D** class is used to load the .obj files of different models from various paths and also reads textures and maps them to objects. One boolean variable is present to see if a texture needs more attention on rendering some parts that need that “special attention”. For example: leaves on a tree or a light bulb from a street lamp to give them a little transparency.

The **Mesh** class is needed because a lot of objects are made from regular meshes. The **Skybox** class is used to load and initialize the skybox faces and textures, and as well to render the skybox. Lastly, the **Shader** class makes possible the loading and the usage of the shaders in the application. There are also some useful structures like: **Vertex**, **Texture** and **Material**.

3.4. Class hierarchy



**Fig. 15** – Class hierarchy.

**4. Graphical User Interface**

I will insert a final image of the project from that cycle. The keyboard inputs used are:

- Using the mouse can view the scene

**- B** key – animation starts

**- Q** key – airplane generated and starts moving

**- W, A, S, D** keys – work around the scene

**- X** key – wireframe view

**- Z** key – disable wireframe view

**- E** key – dog moves.

**Fig. 16** – View from the day.



**Fig. 17** – View from the night.

**5. Conclusions and further developments**

In conclusion, I think that I managed to develop some useful skills using the OpenGL library and I managed to understand better the concepts of light, object, animation, shadows and hope that also my photo-realism perceptions are a little bit more trained.

As for future development, I would like to have more objects, animations of any kind, other light sources and different scenes, to interchange them and also raining and snowing sounds challenging to implemenent, but firstly, I need to improve other functionalities from this initial project to increase realism such as better collition computation and animation of small parts.

**6. References**

[1] https://github.com/gametutorials/tutorials/blob/master/opengl/bezier

[2] Laboratory works

[3] https://learnopengl.com/.