

**Faculty of Automation and Computer Science**

**Computer Science Department**

**Graphical Processing Systems**

~Project Documentation~

*Moldovan Horia-Andrei*

*Group 30434*

1. **Contents**

2. Subject specification……………………………………………………………………,…3

3. Scenario…………………………………………………………………………………,…3

3.1. Scene and objects description……………………………………………………4

3.2. Functionalities……………………………………………………………………7

4. Implementation………………………………………………………………………,…….7

4.1. Special functions and algorithms…………………………………………………7

4.2. Graphics model…………………………………………………………………..11

4.3. Data structures……………………………………………………………………11

4.4. Class hierarchy…………………………………………………………………..12

5. Graphical user interface presentation and User manual……………………………………13

6. Conclusions and further developments…………………………………………………….14

7. References………………………………………………………………………………….15

1. **Subject specification**

The subject of the project consists in the photorealistic presentation of 3D objects using OpenGL library. The user directly manipulates by mouse and keyboard inputs the scene of objects. The detailed list of requirements is given in the following list:

* visualization of the scene: scaling, translation, rotation, camera movement
* using keyboard and mouse,
* using animation.
* specification 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.
* 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**
   1. **Scene and objects description**

My scene represents a residential property of a, most likely, rich person. It is placed on a field surrounded by a lot of nature and the contact with the surrounding environment is emphasized by the presence of a nearby freeway. The scene has a distinctive area, surrounded by a fence, which represents a garden filled with tall grass, flowers, trees and a small castle. Moreover, the scene is also filled with trees to put the user under the impression that he is located in a natural environment. The trees also confer complexity and realism to the scene since they are spread among the entire scene and not placed in straight lines or any kind of order.

However, in the micro-scene of the castle, the trees are specifically placed so that they offer a sense of symmetry to the viewer. The flowers are of two types, roses and pink daisies. They have been chosen in this way to emphasize the contrast with the grass.

The scene also has a country-style house with a nearby well. The rocks, the wood pieces and the well are placed in the scene to highlight the intervention of the human in the scene. Therefore, it is not a nature-based only scene and it contains many objects that involve human-intervention.

The following is a more detailed explanation on the purpose of each group of objects.

1. **Plants**

The scene contains mainly two types of trees which can be classified from the point of view of the castle micro-scene as inner trees and outer trees. The inner trees have a thinner image, and are placed very symmetrically. This contrasts with the outer trees which seem to be very disorganized. Therefore, the inner trees are considered to have been placed there by humans, while the outer trees are considered to have grown there naturally.



**Figure 1: Inner tree(left) and Outer tree(right)**

Another type of plants present in the scene are flowers and grass. Together, they form an eye-pleasant view of the garden scene. The grass is very detailed, being composed of many threads and with a light green color which completes the scene chromatically when the light source is above the grass. The flowers are of two types, spread among the garden in random places. The flower objects are the ones that make the scene a little bit more colored and cheerier.



**Figure 2: Grass and flowers**

1. **Buildings, fence, gates and the freeway**

The two buildings present in the scene are a house and a castle. The castle is small-sized and is the central point of the garden. It has brick-like texture that gives a medieval look to the scene. The same idea was taken into consideration when choosing the house. The house is made of stone and in grey tones so that it resembles a country house, rather than a modern house which would have not made sense to be nearby the castle.



**Figure 3: Stone house**

The gate has the same medieval look as the castle. It was chosen accordingly so that it completes the entire fence structure. The fence, besides the entrance, is made of classical wood.



**Figure 4: Medieval castle**

The freeway is present because it satisfies the intention of connecting the scene to the surrounding world.

1. **Rocks, logs and well**

In order to provide some other objects that can complete such a scene, some basic natural elements where chosen. Since the entire material-structure of the scene so far comprises organic materials (such as grass, stone or wood), the auxiliary objects where chosen from the same area.

Therefore, the rocks give the sense of a nature-surrounded environment. The rock block is situated next to the garden. The wood logs are an auxiliary to the stone house and represent the fuel for warming up the buildings. Moreover, the stone well is the water source for the buildings.



**Figure 5: Rocks, logs and the stone well**

1. **The helicopter**

The helicopter is the most sophisticated object of the entire scene. Its detailed modelling makes it photo-realistic and the continual spinning of its propeller. Moreover, the helicopter is able to move in more dimensions (up, down, front, back) and rotate on the z axis to move around in the third dimension as well.



**Figure 6: The helicopter**

1. **Dog on the bridge**

In order to add some more dynamic animation to the scene a dog object was added. The dog is initially placed on a tall bridge, but when the appropriate keyboard is pressed, the dog starts running from one end of the bridge to the opposite end. When the end is reached, the dog steps on a platform, which has the purpose to bring the dog down from the bridge.

****

**Figure 7: Dog animation**

* 1. **Functionalities**

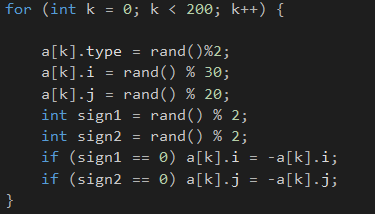
The functionalities of the application are numerous and provide the user the opportunity of having a better look at the scene, as well as playing with some of the object components. The following functionalities are available:

* Camera movement with the keyboards and automatic camera movement.
* Helicopter driving using keyboard.
* Calling the dog to you.
* Rain control with the keyboard.
* Camera restrictions (such as not entering the ground or not entering the castle).
* Light source control with keyboard for both lights.
* Random object generation (for flowers and rain). Every time the user starts the application, the flower positions and types are different from the previous execution of the program.

1. **Implementation**
   1. **Special functions and algorithms**
2. Functions for object generation

Solution

In order to generate the flowers in the garden, two functions were employed. The first function generates random positions on the XZ plane using the random library. It generates a random number between 0 and 30 for the X axis, a random number between 0 and 20 for the Z axis and a random number (0 or 1) for the type of flower, 0 being the code for rose and 1 for the pink flower. Moreover, the sign of the x and z coordinates are taken randomly so that the flowers are distributed in the front, back, left and right relative to the position of the castle. The following code section displays the generation:



**Figure 8: Flower position generation**

The function is called in the main program before rendering the scene. Then, in the render scene, we employ a function that translates the model matrix onto the generated position and the draw the object. This is performed 200 times, generating in this way 200 different flowers.

Motivation

This approach was chosen because it implies generating positions only one time

per execution, so that in the rendering function (which is called in the main loop), we always draw the flowers in the same positions which are through the whole execution present in the memory.

1. Functions for rain animation

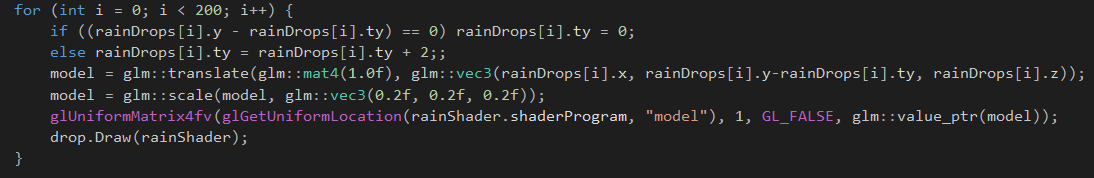
Possible solutions

The generation of rain drops could be performed in two ways:

1. Create an object which contains many drops (transparent) spheres and translate it using a global variable over and over again.
2. Create an object which contains only one drop, generate initial positions with high values on the Y axis and translate each one of them individually.

Motivation

I have chosen solution number 2 because it is more flexible and the generating problem had been already solved. The generation of the object positions is very similar to the one of the flowers, with different parameters. The initial position is stored in memory once and never changed afterwards so that the drops can always be brought back to the initial position and the rain effect can be continuous. The following code section represents the drawing strategy:



**Figure 9: Code snapshot for rain drops drawing**

1. Functions for camera collision

Possible solutions

The camera collision problem can be solved in two main ways:

1. Detect the margins for each object that can’t be passed through and use a memory location for storing all these values. Afterward, when moving the camera, check for a right translation.
2. Use a hardcoded version of the coordinates which block the movement of the camera. Make an elaborated function for checking all the possible coordinated.

Motivation

Although solution number 1 would have been a more elegant one, it would have meant interfering with the Model3D and Mesh classes and potentially altering their basic functionality. Therefore, I chose method number 2 and restricted the not allowed areas to the area below the ground and the interior of the castle.

1. Light sources implementation

Possible solutions

1. *Gouraud (per vertex lighting)* – compute the light influence for each vertex using the vertex normal, the light direction (or light location in case of the point light), the light color, the viewer position. Generate three different vector which represent ambient light (light of the environment), diffuse light (the light scatterd equally among all directions of the light source) and specular light (the light reflected by the surface). All the light computations for this model are performed in the vertex shader.
2. *Phong (per pixel lighting)* – compute the light influence for each pixel using the fragment normal instead of the vertex normal. The technique is the same as for Gouraud model, but all computations are employed in the fragment shader (at the fragment level which means each pixel is taken into consideration).
3. *Blinn-Phong* – this model is very similar to the Phong model but instead of computing the reflection direction using the reflection technique, we compute it as a half vector which is at the exact half between the normal and the light direction. All computations are performed in the fragment shader as well (it is a per-pixel lighting implementation).

Motivation

I chose the Blinn-Phong model. Although the Gouraud model performes better regardin the execution time (in the GPU, computations are performed only for vertices of each mesh), I chose a per-pixel implementation because it reflects the reality in a better manner and the quality of the colors is much more improved. The model was used for both light sources (point and direction light) because the difference between them represents just the computation of the light influence (the point light scatters light in all directions and the light attenuates over time and the direction light scatters light in only one direction and the light comes from all places on the given direction.

1. Shadow computation

Solution

For the shadow computation, the chosen solution was according to the one studied at the university: *Shadow maping*. The approach uses a depth map buffer to store depth values from the light perspective. The decision if an object is in shadow or not is taken getting values from the depth map buffer. The algorithm has two passes:

* The first pass renders the scene from the light point of view. By creating a depth texture attached to a frame buffer object, we render the entire scene saving only the depth values.
* The second pass renders the scene from the camera position point of view and compares the depth of each fragment to the values stored in the shadow map. Fragments that have a greater value in the depth map than the current stored, are in shadow.

Motivation

The solution is easy to understand because it doesn’t use any complicated notions. Moreover, the results are quite satisfying (although the shadows are quite pixelated and do not present an entire resemblance to a real-life shadow).

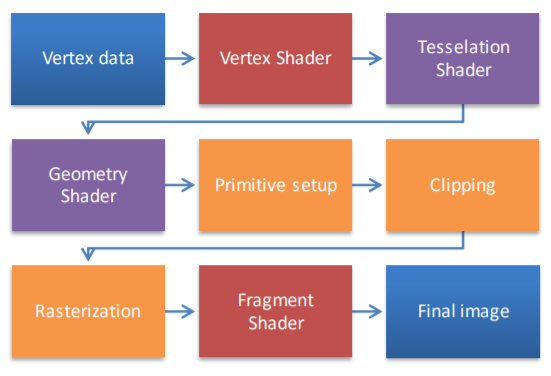
* 1. **Graphics model**

**Open Graphics Library (OpenGL)** is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU) to achieve hardware-accelerated rendering.

The system architecture is Client-Server where:

* Client: OpenGL application developed by programmer
* Server: OpenGL implementation provided by the manufacturer of your computer graphics hardware.

The following is the representation for the graphics pipeline:



**Figure 11: OpenGL graphics pipeline**

The chosen programing language is C++ and the environment is Visual Studio 2013.

* 1. **Data structures**

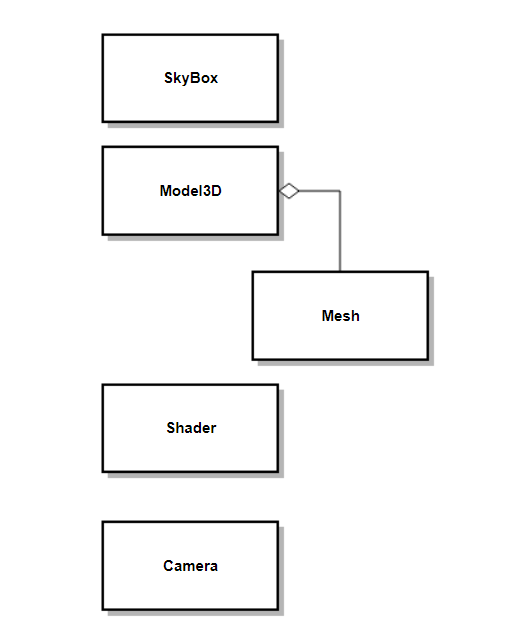
The main data structures employed in the implementation are either classes or C++ type structures.

1. Classses

* Camera: the class that holds the camera position, target and direction and has methods for retrieving them. Moreover, this class has methods for camera movement in directions: front, back, left and right, and camera rotation with pitch and yaw.
* SkyBox: the class that has methods for loading a sky box, drawing the sky box and retrieving the texture id.
* Mesh: the class that holds the vertices, textures and colors of a mesh (a mesh is the basic building block of the object, most likely a polygon) and has methods for setting up and drawing.
* Model3D: the class that holds the internal structure of the object. It has several functionalities (such as loading the object, reading textures and loading textures into the video memory).

1. Structures

* Vertex: characterized by the position, the normal and the texture coordinates.
* Texture: characterized by the id of the texture, the type (diffuse or specular) and the path to the texture.
* Material: characterized by the three vectors – ambient, diffuse and specular.
* flowerPos: characterized by - the type of flower (either rose or the pink flower), the coordinate of the X axis and the coordinate of the Y axis
* dropPos: characterized by – the coordinates on the X, Y and Z axis as well as by a variable called *ty* which holds the displacement from the initial position. This variable has the task of helping in the translation mechanism for the rain drops. By using this variable, we never loose the initial positon of the drop.
  1. **Class hierarchy**



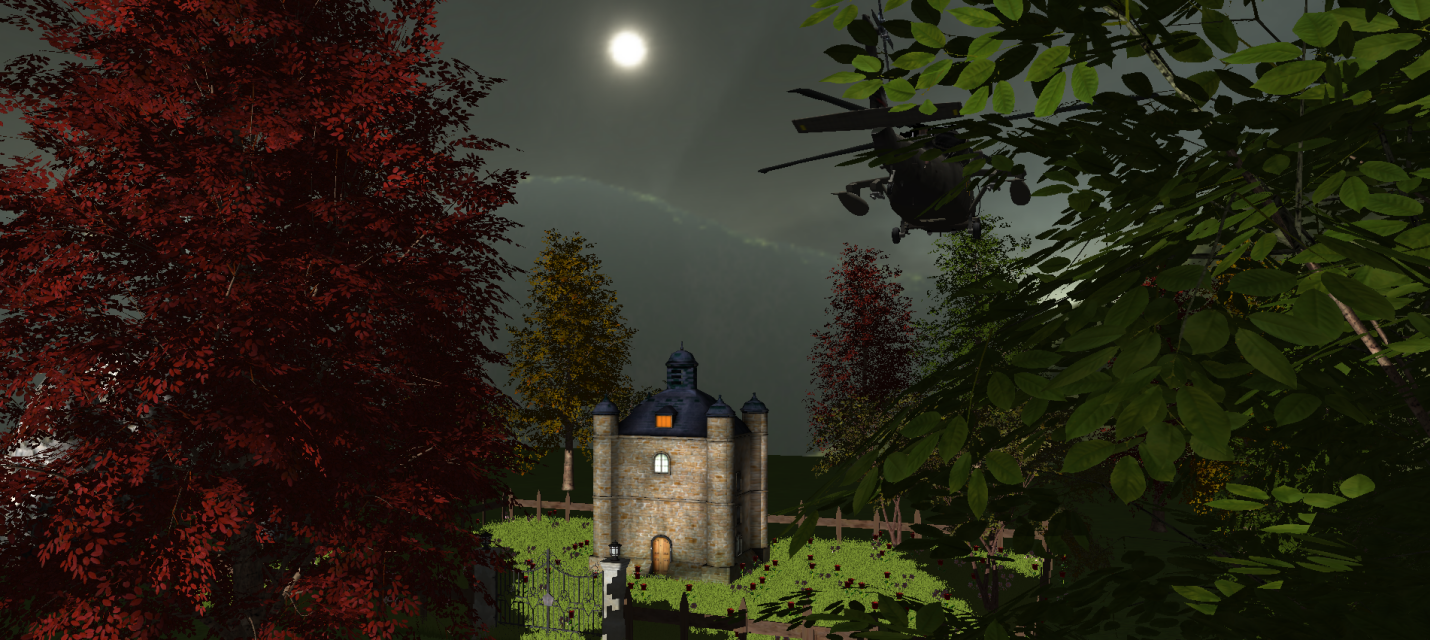
The class diagram reflects only the classes with which I had to deal in my project. The most relevant mention to be made is that the Model3D is composed of more meshes, hence the aggregation between the two classes.

1. **Graphical user interface presentation and User Manual**

The following are some prints of the graphical user interface:



**Figure 13: View of the house, red balloon and trees**



**Figure 12: View of the castle**

**User Manual**

The user has as input devices the keyboard and the mouse.

The mouse enables the user to rotate the camera around so that it can change angle of view

(but not the camera position).

The following list explains the functionality of each key:

* W – move the camera forward.
* A – move the camera to the left.
* S – move the camera to the right
* D – move the camera to the right.
* H – helicopter moves forward.
* U – helicopter moves up.
* M – helicopter moves down.
* L – helicopter rotates.
* Q – rotate the direction light.
* 1 – switch to wireframe representation.
* 2 – switch to smooth surface representation.
* 3 – enable rain animation.
* 4 – disable rain animation.

1. **Conclusions and further developments**

The project development process represented a good way of studying the OpenGL graphics pipeline. Moreover, the mathematics behind the graphics programming became more clear after understanding how translations, rotations and other operations are being performed.

The small contact with the .obj and .mtl files was also good for future experiences with 3D modelling tools.

It was also a good programming task since the animations, object generation and camera movement required some programming skills. However, I was pleased that the implementation did not seem to be C++ dependent entirely and it wouldn’t be so hard for me to try to interface OpenGL with some other programming language.

Being restricted by the small amount of available free objects on the internet, I realized how important it would be as a programmer in the field of graphics to have alongside people with experience in 3D modelling mainly because when you imagine a scene, it is hard to reproduce it with what you are given and not with what you are capable of making yourself.

Further developments

There is a significant number of developments that would make the project much more interesting. The following list comprises just a few of them:

* Integrating characters and animating them (persons would be very important to make the scene look livelier).
* Provide a complete animation of the camera that drives the user through all the corners of the scene of objects.
* Provide a night time view of the scene.
* Provide object-collision detection for the helicopter and animate destruction when the helicopter is colliding with the other objects.
* Obtain more realistic objects (diversify the nature elements such as trees, flowers, stones).
* Provide wind, snow or fire effects.
* Extend the scene of objects to a more complex one by adding more buildings. Integrate the scene into a mountain object such that the user cannot notice the “end” of the scene where the ground disappears.

1. **References**

* Free 3D models

<https://free3d.com/>

<https://www.turbosquid.com/>

<https://www.cgtrader.com/>

<https://www.pixelsquid.com/>

* Theoretical aspects and source code

<http://cgis.utcluj.ro/teaching/>

<https://learnopengl.com/>

<https://stackoverflow.com/>

* OpenGL description

<https://en.wikipedia.org/wiki/OpenGL>