***Graphical Processing System***

***Project***

***Documentation***

* *Rus Rares Tudor*
* *Group 30432*

*1. Contents:*

*1. Contents .* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

*2. Subject specification* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

*3. Scenario. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4*

*3.1. Scene and objects description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*

*3.2. Functionalities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*

*4. Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9*

*4.1. Special functions and algorithms. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*

*4.2. Graphics model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*

*4.3. Data structures. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*

*4.4. Class hierarchy. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*

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

*6. Conclusions and further developments. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14*

*7. References. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14*

2. 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.

- 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

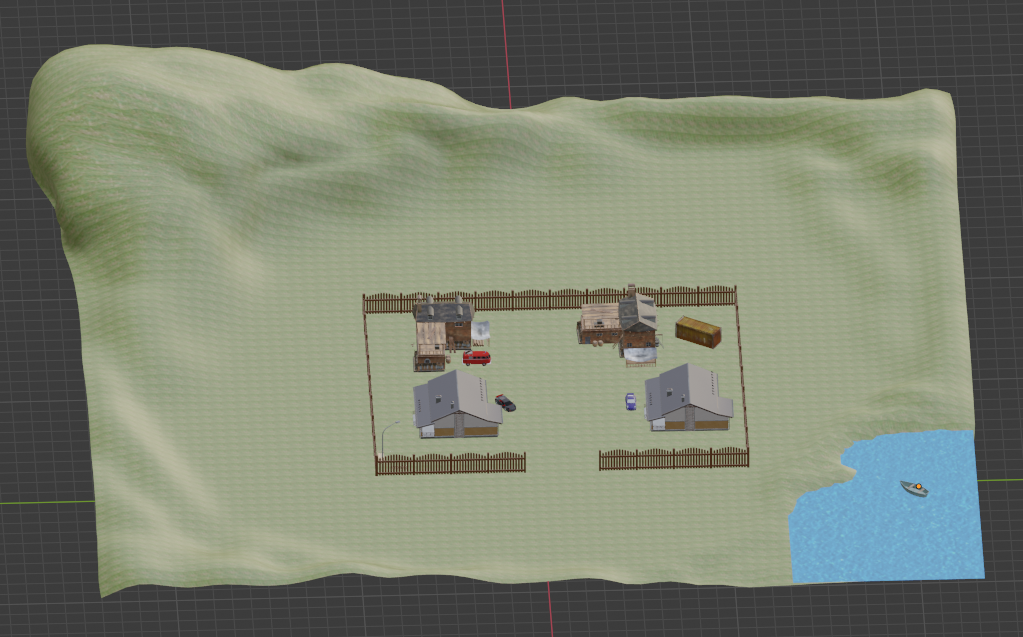
- exemplify animation of object components

- 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)

- documentation (mandatory)

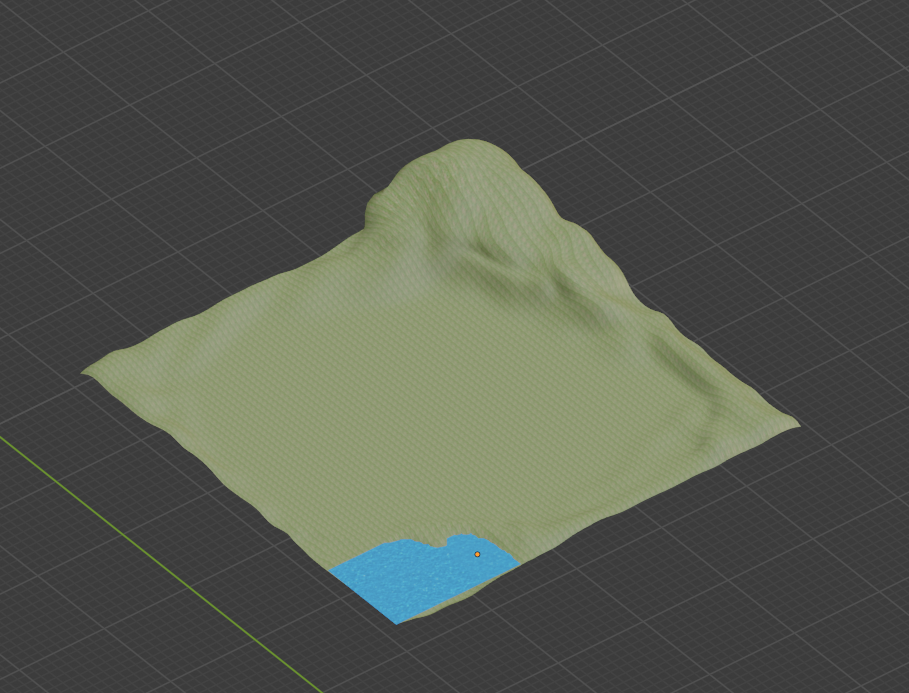
3. Scenario

3.1 Scene and object description

My scene is a wooden fenced yard, inside which are four houses and three cars. The scene was made in Blender, in which I could import any .obj file. Firstly, I added a plan that I textured in such a way as to create the impression of hills and depressions, by adding a grass texture to it, after that I added 2 more plains where the depressions, in order to create the impression of water. Secondly, I searched the model for the fence, houses and cars, the models of these objects were imported from the site free3d.com after that the objects were scaled and transposed in order to respect the real-world proportions. Here is a general view of the scene: 

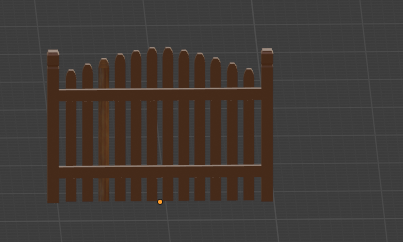
The following section, I will present each object in particular:

Ground:



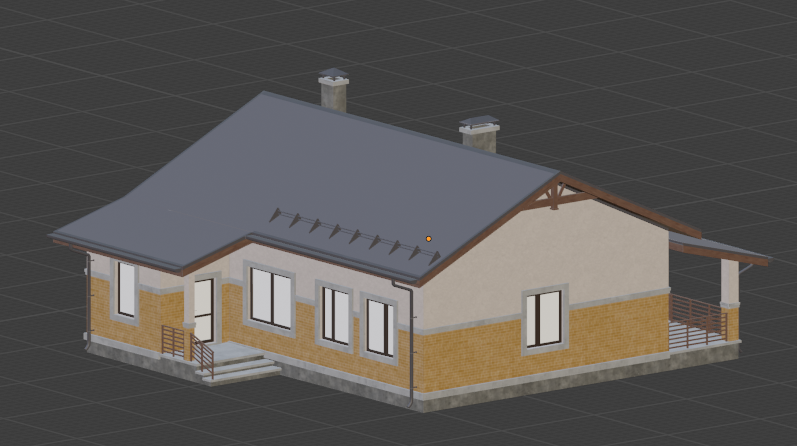
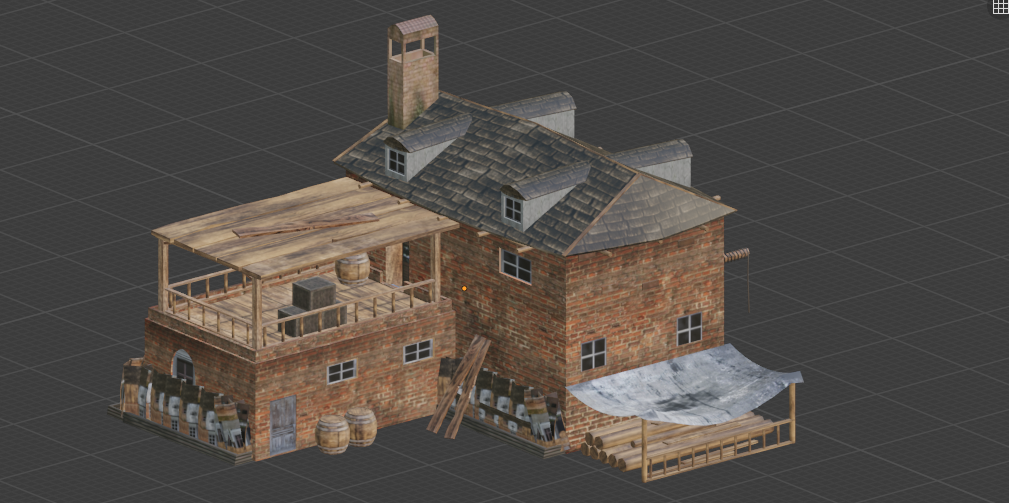
It represents three planes form the Add, Mesh menu of Blender. The big plane was modeled in such a way to create a real look of a hill next to a small lake, the geographic elements were created using the Sculpt Mode, after that all the plane was textured with a grass look. The two plans used for the water are only textured.

Fence:



A basic type of wood fence, it is an independent object, textured with a wood aspect.

Houses:

The scene contains four houses, with two different models, to create the impression of diversity I changed the angle of the seats in the scene. Each house contains a lot of different objects, each textured independently.

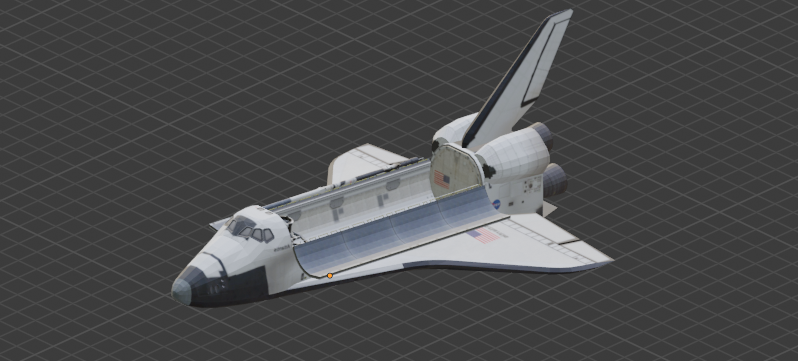
Vehicles:



Each of the objects presented in the photo above are composed of several objects each independently textured in such a way as to create the impression of a car, bus and a boat.

Container:



Shuttle: 

In order to add some more dynamic animation to the scene a new object was added. The shuttle is initially placed in the sky, but when the appropriate keyboard is pressed, the shuttle starts to cross the sky around, all types of an real shuttle are implemented.

3.2 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.
* Shuttle driving using keyboard.
* Calling the camera back in the initial place.
* Light source control with keyboard for the light.
* Camera restrictions.
* Viewing solid, wireframe objects, polygonal and smooth surfaces.

4. Implementation

3.1 Special functions and algorithms

Shadow computation

Solution

For the shadow computation, the chosen solution was according to the one studied at the university: Shadow mapping. 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).

Light sources implementation

Solutions

Blinn-Phong – this model is very similar to the Phong model (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).) 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, because 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.

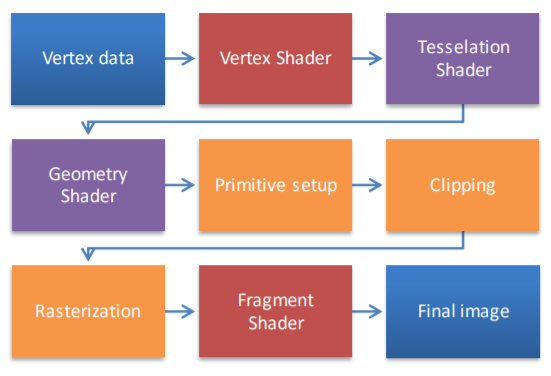
3.2 Graphics model

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:



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

3.3 Data structure

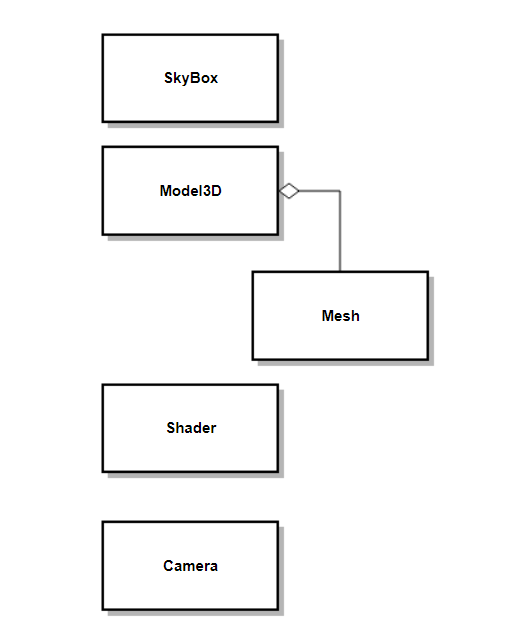
Classes

* 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).

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.

3.4 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.

5. Graphical user interface presentation and User manual



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.

Controls:

W, A, S, D – move the camera forward/ left/right/left.

↑, ↓, →, ← - move the shuttle forward/ left/right/left.

. , , - move the shuttle up/down.

J, L – rotate the direction light.

V – switch to wireframe representation, double-tap to switch to smooth surface representation.

B – switch to point representation double-tap to switch to smooth surface representation.

C – start the camera animation.

X – place the camera in the initial position.

6. Conclusions and further developments

Even if the project did not meet all the targets initially proposed, the result was close to expectations. I am satisfied because I managed to better understand the material taught in the laboratories and because I had the opportunity to work with tools provided by the libraries and to see how it works.

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.

Provide a night time view of the scene.

Provide object-collision detection.

Provide wind, rain, snow or fire effects.

Extend the scene of objects to a more complex one by adding more buildings.

7. References

Free 3D models

<https://free3d.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>