Village Square

Alexandru Bondor, group 30433

Contents

1. Subject specification

The subject of the project consists in the photorealistic presentation of 3D objects using OpenGL library. The object should be an aggregate that provides for individual deformation and animation of its components. Through the graphical interface the user directly manipulates by mouse and various controls on 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
  - Projected shadows
  - Volume shadows

> Exemplify animation of object components

> Photo-realism, scene complexity, detailed modeling, algorithms development and implementation (object generation, collision detection, shadow generation, fog, rain, wind), animation quality, different types of light sources (global, local, spotlights)

2. Scenario

The purpose of the project is to present a village square. It is meant to demonstrate the visual effects that can be obtained by using OpenGL library. It also demonstrates that scene and objects can be represented as close to reality as possible.

2.1. Scene and object description

As I have already mentioned, the scene is located in a village square. In the center of the square there is a dried fountain and pretty close to it there is a still burning fireplace. The four houses near the center seem to be all closed and the square is definitely ruled by the imposing chapel. To the right of the chapel there is a small cemetery. All the scene is surrounded by tall trees and in the far plane there can be seen some hills and the sun setting down being obscured by some huge rocks.

More than the story the scene is composed of actual objects:

> 4 houses which look the same because the house object is multiplied and translated 4 times;
> 1 chapel;
> 1 dry fountain;
> 1 fireplace + animated fire;
> 1 set of huge rocks obscuring the sun;
> 30 trees which look the same because the tree object is multiplied and translated 30 times;
> 1 crow composed of 2 triangles passing above the scene.

2.2. Functionalities

The camera can be moved around to see all the details of the scene. There is the crow flying above the scene which exemplifies the translation and rotation of an object. Moreover there is the fire which is animated to demonstrate the same and a spotlight above the fireplace simulates the light given by fire. Furthermore there are some hidden features whose activation will be described at point 4. These features include fog and shadows.

3. Implementation details

In the next section I will try to underline some important aspects regarding the implementation of the 3d scene. As you have probably guessed, I will not cover all the algorithms, code snippets or OpenGL library functions as this is beyond the scope of this documentation.

3.1. Functions and special algorithms

I have tried to keep the code as clean as possible and reuse the code where I could. I am sure that his is not the best solution but at least it is a working one.

“Special algorithms”:

> Basic collision detection;
> Fire animation

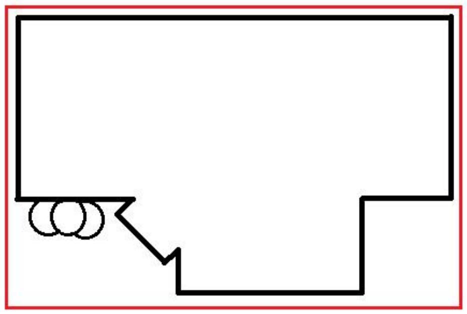
In the next section I will exemplify the found solutions.

3.1.1. Possible solutions

Basic collision detection

There isn’t much logic in writing down the method that makes the collision check or the one that computes the “collision quad” for each object. It would be more useful to explain the found solution.

In order to explain the method I will exemplify it on a house object. On the left there is the house object with material on it and on the right there is a sketch of the house viewed from top. I will make references to the image on the right when explaining the method.



The black line sketch is the outline of the house and the red line rectangle represents the approximated house in order to ease the process of collision detection.

How it works?

For every object in the scene I have attached a surrounding rectangle/square as exemplified in the above picture and stored it in an objBox array.

When the user moves in the scene, the collisionDetected method is called. As parameters, the methods requires the current position of the camera as well as the computed values for X and Y axis rotation and the direction of movement.

Inside the method the parameters are checked to see whether they lay inside any ”forbidden objBox”. If the method finds that the camera collides with any object it returns true, else it returns false.

Fire animation

The solution found to this problem exemplifies the power of glut library. There is a very usefull method called glutTimerFunc that can call a method after a specific period of time. Therefore, by calling a method that has inside its body glutTimerFunc which calls the same function you can obtain an infinite loop which might be useful. For example you can use it to animate a fire.

The pictures below were used to animate the fire.

By switching the fire image at every 150ms I have obtained a pretty decent fire effect.

The code below represents all the code required to swap the texture which will be redrawn every frame.

```
void pulse(int value)
{
        switch (value)
        {
        case 1:
                fireTexture = fire1Texture;
                break;
        case 2:
                fireTexture = fire2Texture;
                break;
        case 3:
                fireTexture = fire3Texture;
                break;
        case 4:
                fireTexture = fire4Texture;
                break;
        }
        if (value == 4)
        {
                value = 1;
        }
        else
        {
                value++;
        }
        glutTimerFunc(fire_delay, pulse, value);
}
```

3.1.2. The motivation of the chosen approach

Basic collision detection

I have chosen to implement collision that way because it is a fast algorithm for small number of objects and it is enough for the purpose of this project.

Fire animation

Looks good and it didn't require much time to be implemented.

3.2. Graphics model

3.3. Data structures

The main data structure used is the vector. Other custom structures were created just to compress variables. An example of custom structure is given below:

```
typedef struct c
{
	float x, y, z;
}Camera;
```

Instead of making 3 variables and then referring them so, I have compressed them into an object and then usage is more logical.

3.4. Class hierarchy

Improper called class hierarchy. The whole code is spread along several .cpp files:

> VillageSquare.cpp
  - Represents the entrance of the application. It is the main part of the application that makes use of the other .cpp files in order to generate and animate the desired scene. It is responsible for calling the drawing methods each frame and processing the input which translates into camera movement or activating/deactivating features.

> MapGenerator.cpp
  - This also has an important role in the application. Here are implemented the methods that load the objects together with their textures and the ones which actually draw the objects in the scene. Moreover there is the method which checks for collision.

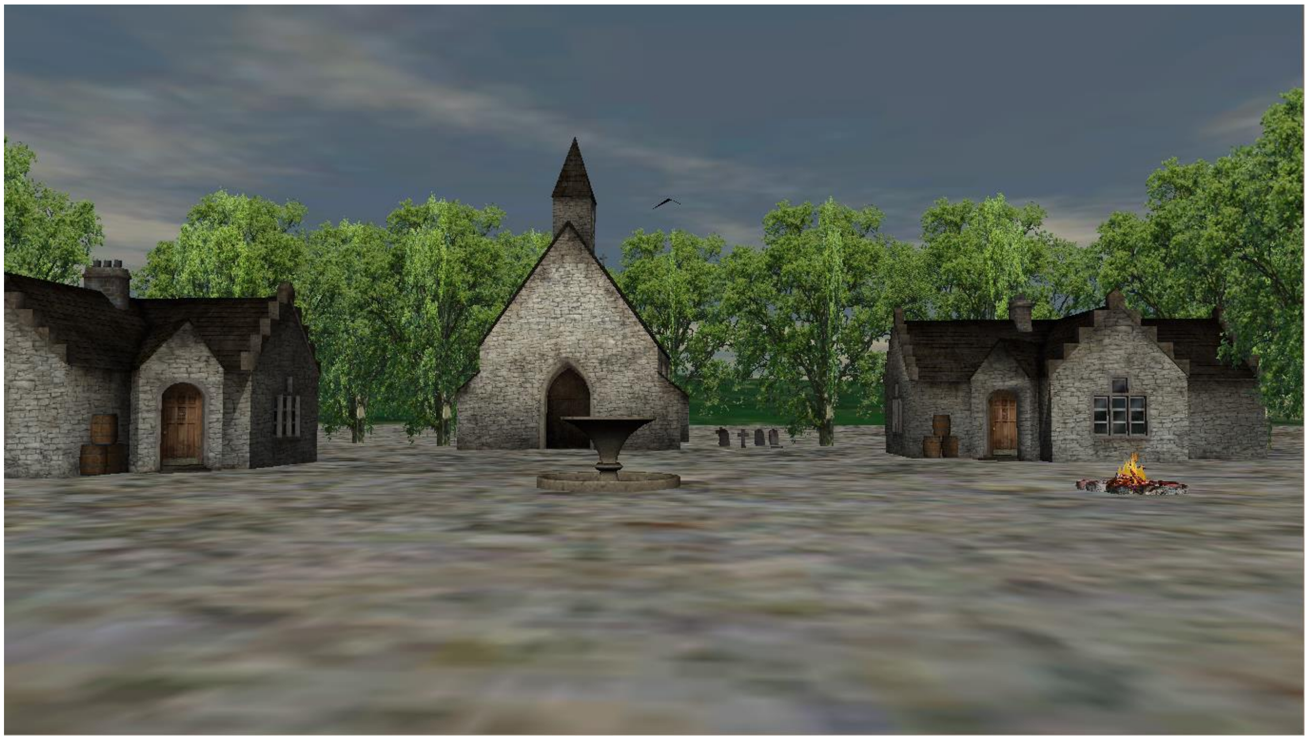
> Skybox.cpp
  - Loads skybox textures and draws the skybox.

> Shadows.cpp
  - Holds the methods used for computing the shadows.

> TextureLoader.cpp
  - Does what its name says.

4. Graphical user interface presentation / user manual

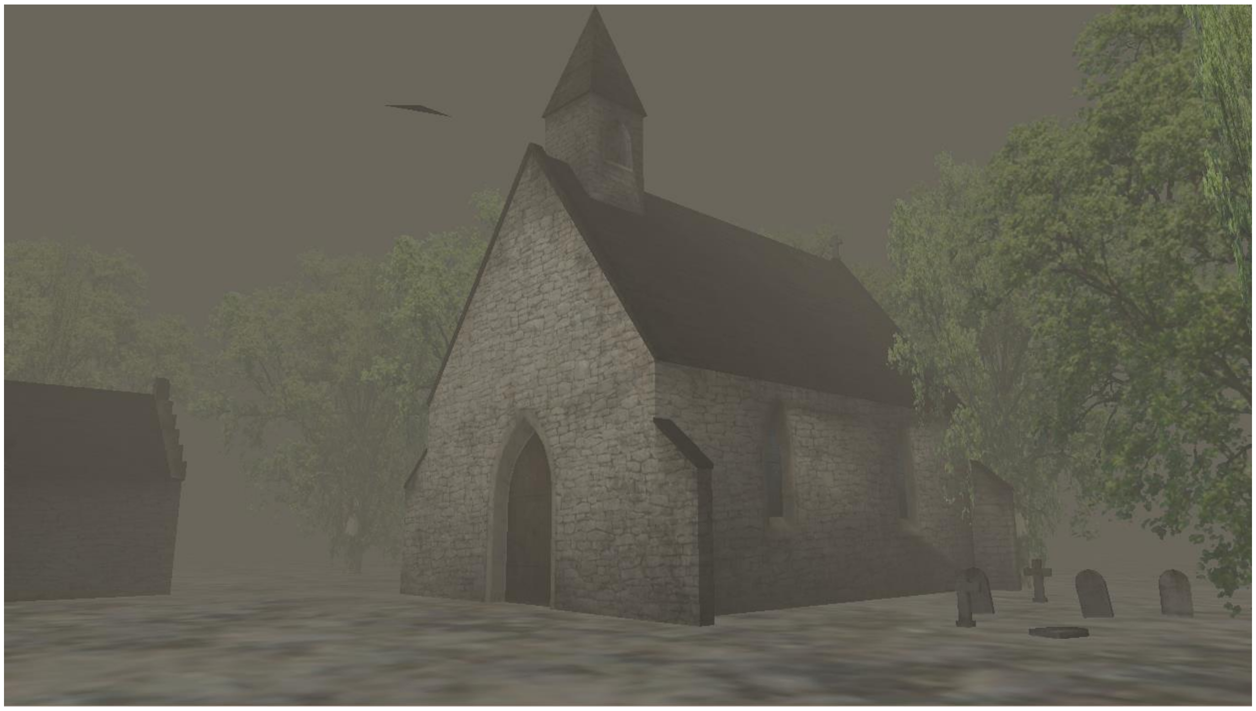
First view of the scene can be seen in the snapshot below. Right in front there is the dry fountain, in the back there is the chapel, on the mid right there is the fire and the rest of view if filled with the houses, trees, skybox and ground. The crow can be spotted to the right of the chapel tower. It is animated to fly from the right to the chapel, it will rotate 3 times around the tower then flies away to the left.



There is a view to the back of the scene where we can observe the pile of rocks and the reddish spotlight that is right above the fire lightning the fireplace.

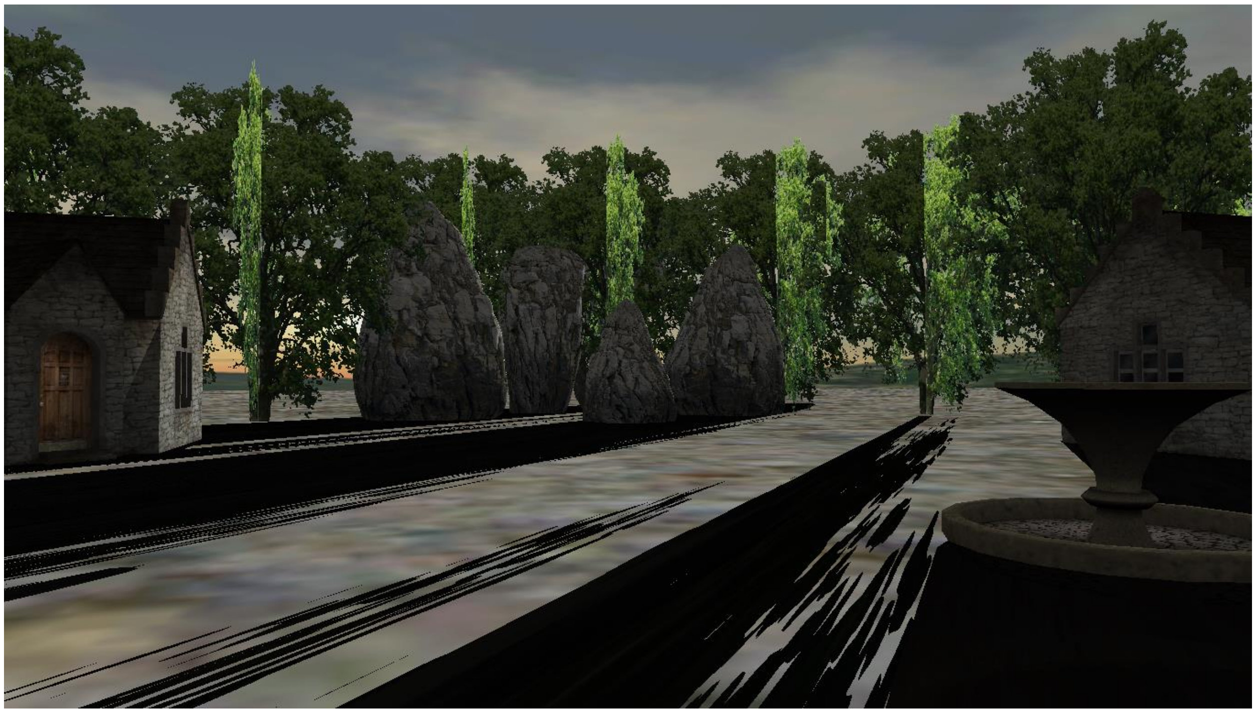
Fireplace close-up. The flame I was talking about at section 3.1.1. can be seen here.

Chapel view when fog is on.



Shadows exemplification. The shadows are so extended because the sunlight comes from far away.

The shadows are projected so there aren't any shadows on the objects.



User manual (mind CAPSLOCK because commands are case sensitive):

> Camera move
  - Use mouse to look around
  - Move forward with ‘w’, backward with ‘s’, to the left with ‘a’, to the right with ‘d’

> Fog
  - Toggle fog by pressing ‘f’

> Shadows
  - Toggle shadows by pressing ‘p’

5. Conclusions and further developments

To sum up, I consider that I have managed to create a realistic scene and with animations that can fool an eye. There are many further developments to be considered some of which I will mention:

> Volume shadow
> Particles fire
> 3d trees
> Possibility to enter buildings

6. References

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

www.google.com