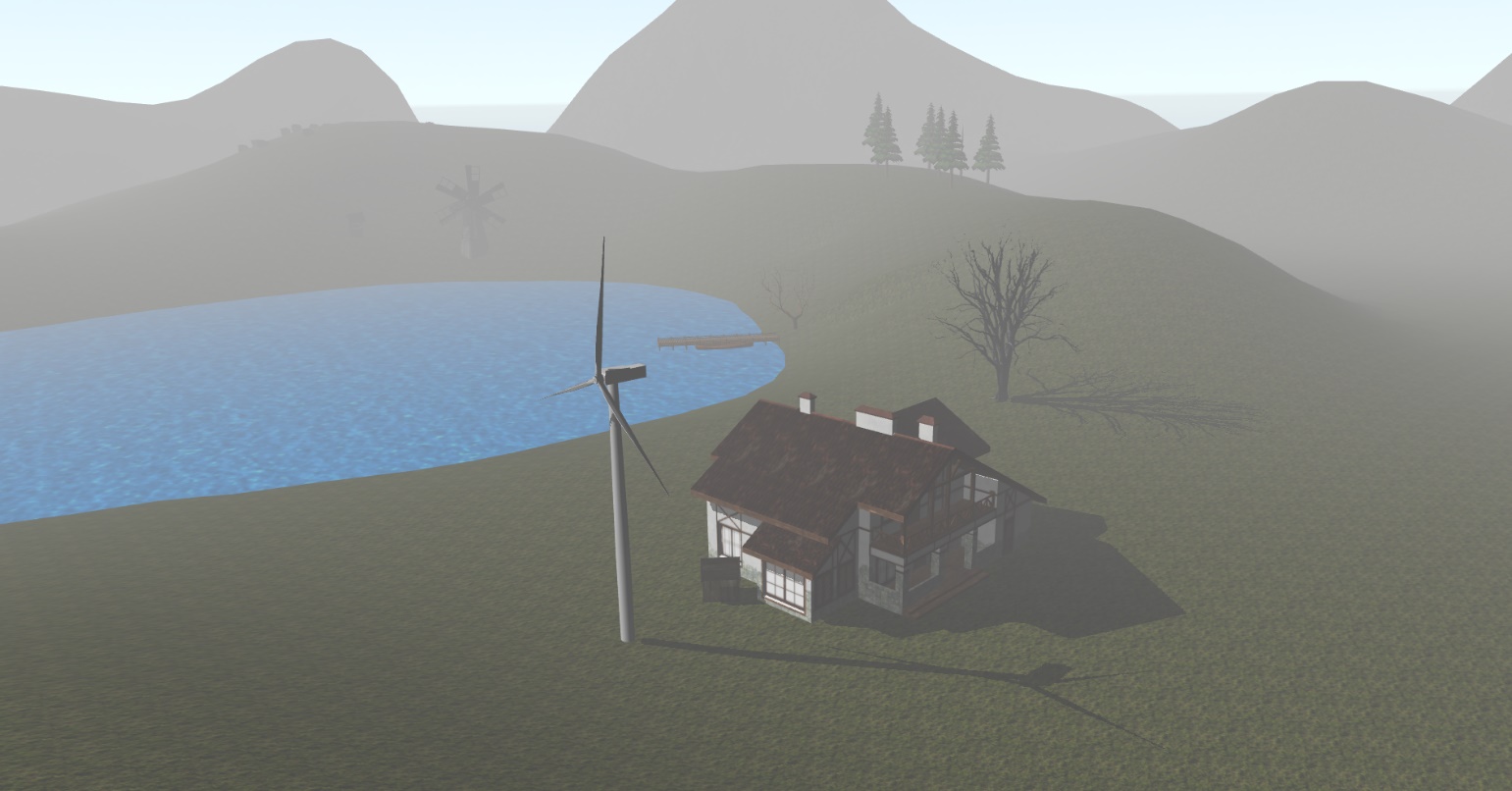
**Graphic Processing Project**

****

STUDENT: Bodea Razvan-Marius

Table of Contents

[1.Subject Specification 3](#_Toc92884965)

[2. Scenario 4](#_Toc92884966)

[2.1. Scene and Objects Description 4](#_Toc92884967)

[2.2. Functionalities 7](#_Toc92884968)

[3. Implementation Details 8](#_Toc92884969)

[3.1. functions and special algorithms 8](#_Toc92884970)

[3.2. Graphics model 11](#_Toc92884971)

[3.3. Class Hierarchy 11](#_Toc92884972)

[4. User Manual 12](#_Toc92884973)

[5. Conclusions and further developments 13](#_Toc92884974)

[6. References 14](#_Toc92884975)

# 1.Subject Specification

The goal of the project is to create a scene that looks realistic, with 3D models and textures using the OpenGL library. The user can navigate through the scene using the mouse and keyboard.

Project Requirements:

* **(2p)** visualization of the scene: scaling, translation, rotation, camera movement
  + using keyboard and mouse
  + using animation
* **(1p)**specification of light sources (minimum two different lights)
* **(0.5p)** viewing solid, wireframe objects, polygonal and smooth surfaces
* **(1p)** texture mapping and materials  
  + textures quality and level of detail
  + textures mapping on objects
* **(1p)** exemplify shadow computation
* **(0.5p)** exemplify animation of object components
* **(3p)** 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)
* **(1p)** documentation (mandatory)

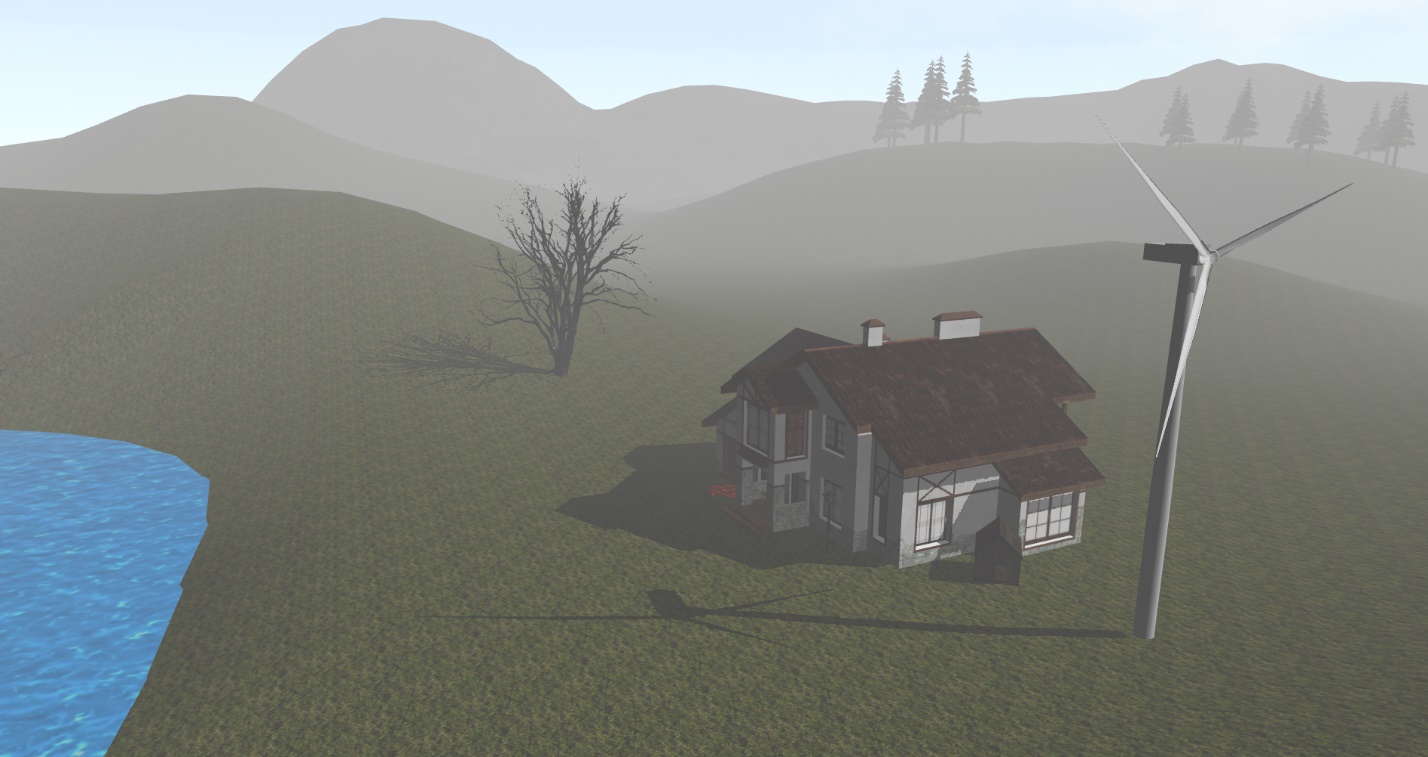
# 2. Scenario

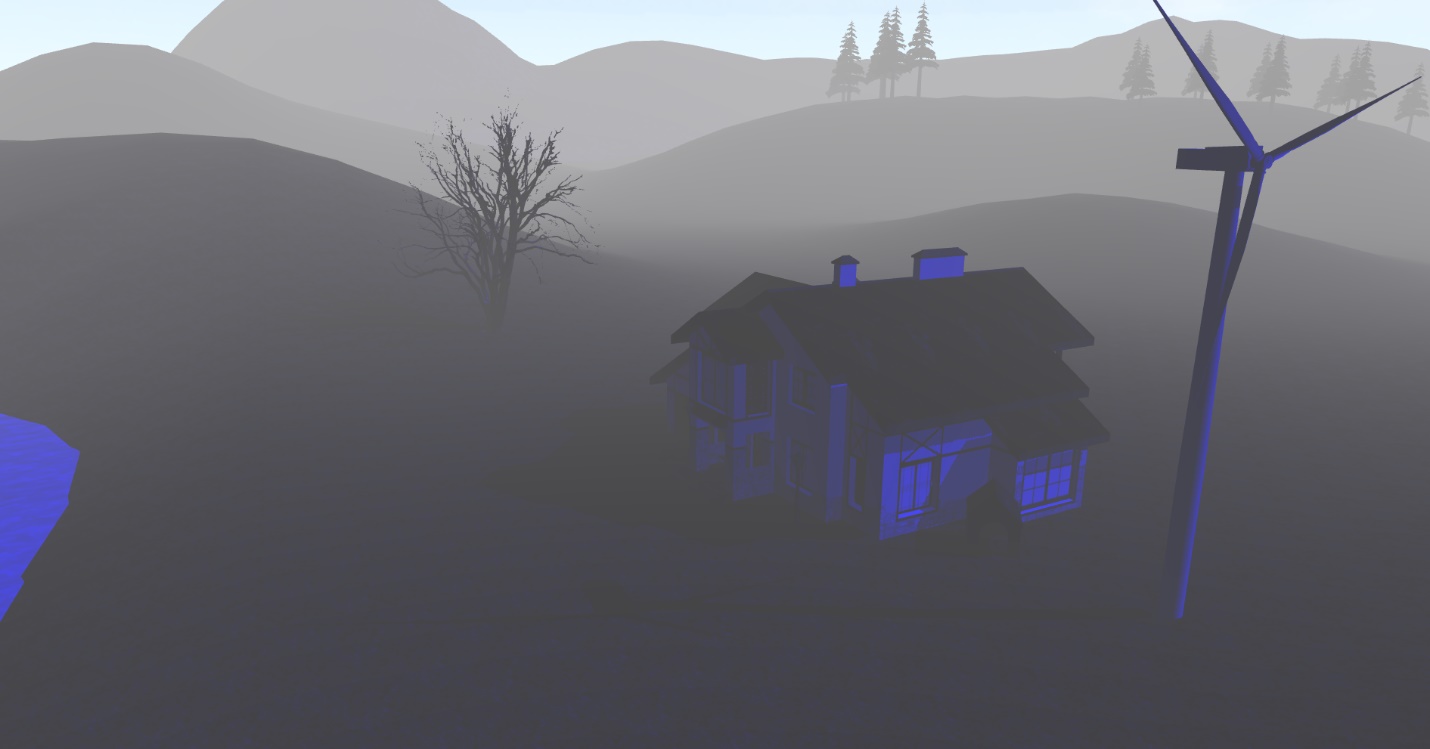
## 2.1. Scene and Objects Description

The idea of my scene was to create a remote place, somewhere in the mountains, a place where you would go to just relax after a hard month of non-stop work on projects. The main goal was to have a scene with objects that fit together, in order to create a realistic look. Alos the scaling ot the objects is extremely important for creating a realistic look.

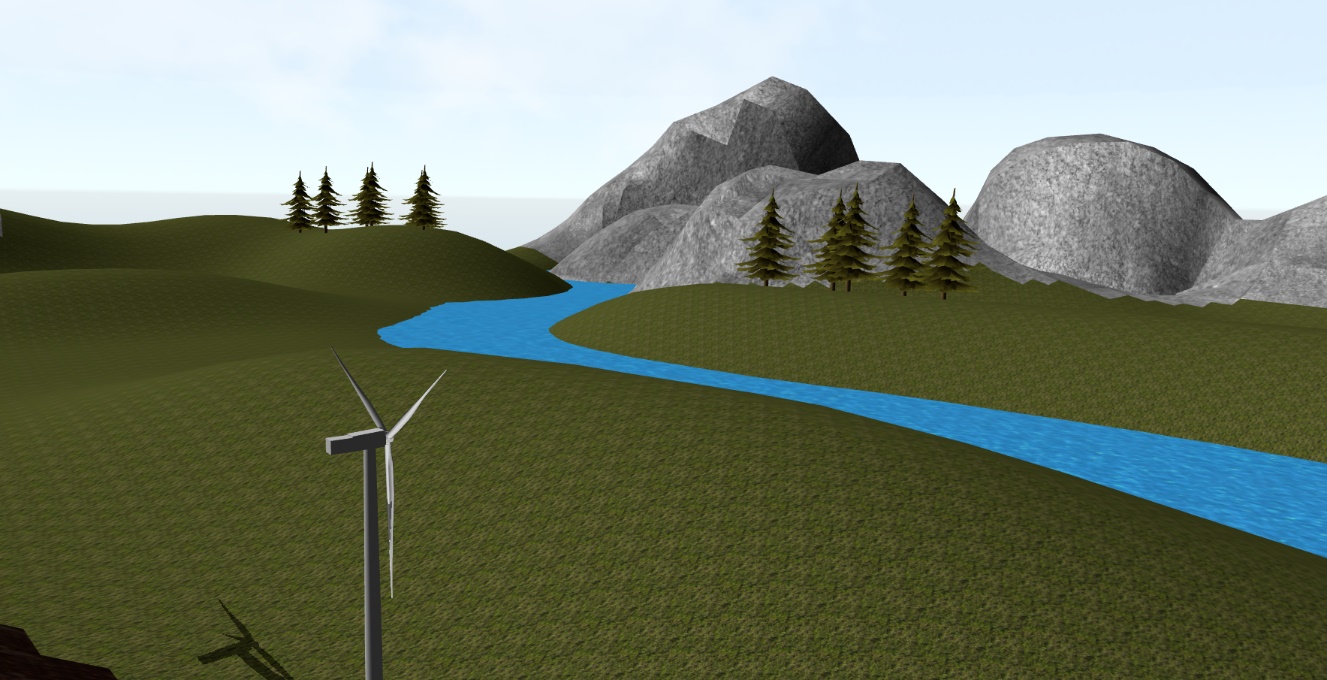
I tried adding some effects in order to get a better representation (fog, rain, light, shadows).

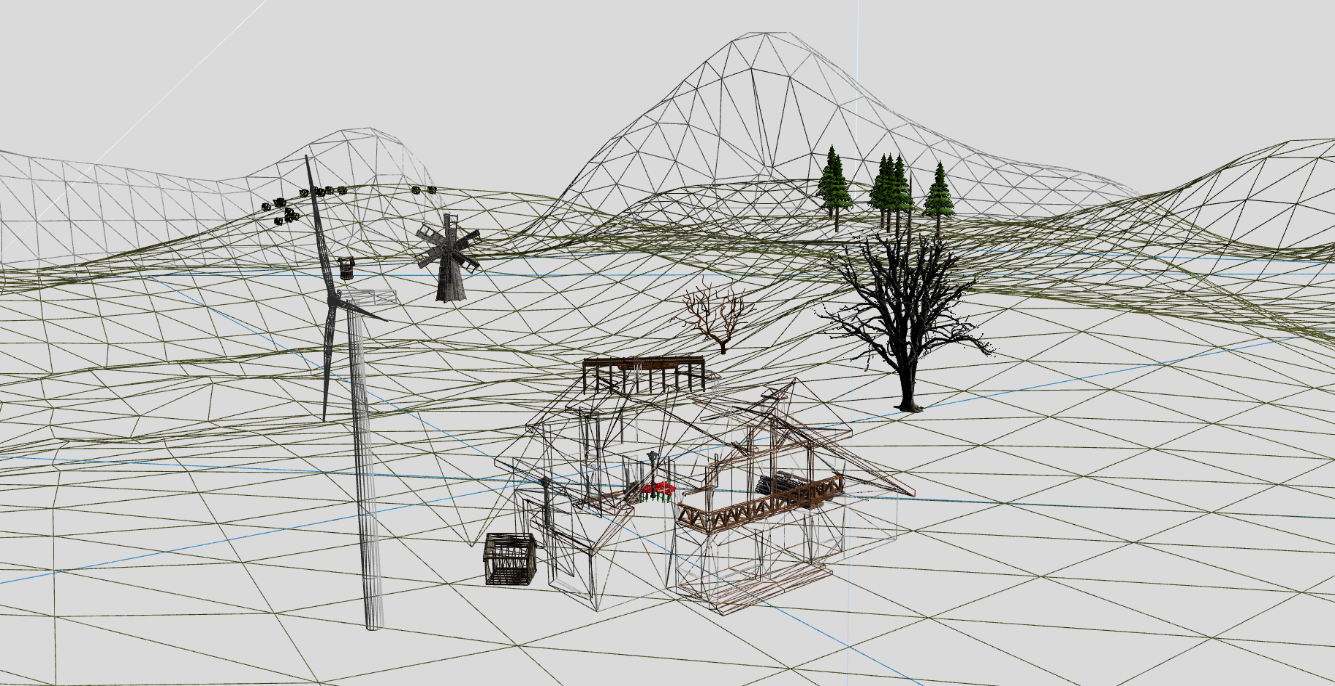
The main objects of the scene were 3D models downloaded from different sites (see bibliography), and some smaller objects that that I created and textured (flowers).











## 2.2. Functionalities

The main functionality of the project is navigating throug the map. This can be done by using the WASD keys and by clicking and dragging with the mouse to look around the scene.

* Move forward(W)
* Move backwards(S)
* Move left (A)
* Move right (D)
* Move down (X)
* Move up (Z)
* Rotate whole scene left (Q)
* Rotate whole scene right (E)
* Look around (hold left click and drag the mouse)

Other functionalities:

* Animation of the wind turbine (rotation of a part around a point)
* Shadows
* Global light
* Toggle fog effect on and off (F,G)
* Toggle rain effect on and off(R,T)
* Wireframe view (N)
* Vertices view (M)

# 3. Implementation Details

## 3.1. functions and special algorithms

a. Fog

The fog is added to the computation of the final color when a uniform variable is 1. The base color of the fog is hard- coded inside the fogColor variable. The startFog uniform can be changed form 0 to 1 and the other way by pressing buttons on the keyboard.

uniform int startFog;

float computeFog()

{

float fogDensity = 0.3f;

float fragmentDistance = length(fPosEye);

float fogFactor = exp(-pow(fragmentDistance \* fogDensity, 2));

return clamp(fogFactor, 0.0f, 1.0f);

}

float fogFactor = computeFog();

vec4 fogColor = vec4(0.5f, 0.5f, 0.5f, 1.0f);

if(startFog == 1)

fColor = mix(fogColor, min(colorWithShadow \* vec4(light, 1.0f),1.0f), fogFactor);

else

fColor =min(colorWithShadow \* vec4(light, 1.0f),1.0f);

b. Rain

Inspired by this code <https://gist.github.com/thaenor/4d9531cc9a7d1c34b998> .

The idea of this algorithm is to create random objects with the rain model somewhere up in the scene and drop them down and delete them when they reach a point below the scene.

Similarely, this code can be changed to implement snowing and other weather effects by just changing the 3D model that is drawn.

The rainNr int is used to change the ammount of rain particles created.

int isRaining;

int rainNr = 1000;

if (pressedKeys[GLFW\_KEY\_R]) {

if (isRaining) return;

isRaining = 1;

raindrops.clear();

for (int i = 1; i <= rainNr; i++) {

raindrops.push\_back(generateRaindrop());

}

}

std::list< std::pair< glm::vec3, float> > raindrops;

std::pair< glm::vec3, float> generateRaindrop() {

return { glm::vec3(rand() % 66 - 20, rand() % 25 + 5, rand() % 76 - 38) , 10.0f / (rand() % 50 + 50.0f) };

}

void moveRaindrops() {

for (auto& it : raindrops) {

it.first.y -= it.second;

if (it.first.y <= -2.5) {

raindrops.push\_back(generateRaindrop());

}

}

raindrops.remove\_if([](auto it) {return it.first.y <= -2.5; });

}

if (isRaining) {

moveRaindrops();

for (auto it : raindrops) {

model = glm::translate(glm::mat4(1.0f), it.first);

glUniformMatrix4fv(modelLoc, 1, GL\_FALSE, glm::value\_ptr(model));

normalMatrix = glm::mat3(glm::inverseTranspose(view \* model));

glUniformMatrix3fv(normalMatrixLoc, 1, GL\_FALSE, glm::value\_ptr(normalMatrix));

rainDrop.Draw(shader);

}

}

c. Shadows

For calculating the shadows I had to create a map from the POV of the light source (if you look from the light towards the scene and hit a point, anything that is behind that point will be in the shadow and have a slightly darker color). The implementation of the shadow is similar to the one presentet in the lab guide.

float computeShadow() {

vec3 normalizedCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;

normalizedCoords = normalizedCoords \* 0.5 + 0.5;

float closestDepth = texture(shadowMap, normalizedCoords.xy).r;

float currentDepth = normalizedCoords.z;

//float shadow = currentDepth > closestDepth ? 1.0 : 0.0;

float bias=0.005f;

float shadow =currentDepth-bias>closestDepth ? 1.0f : 0.0f;

return shadow;

}

shadow = computeShadow();

vec3 color = min((ambient + (1.0f - shadow)\*diffuse) + (1.0f - shadow)\*specular, 1.0f);

vec4 colorWithShadow = vec4(color,1.0f);

if(startFog == 1)

fColor = mix(fogColor, min(colorWithShadow \* vec4(light, 1.0f),1.0f), fogFactor);

else

fColor =min(colorWithShadow \* vec4(light, 1.0f),1.0f);

depthShader.useShaderProgram();

glUniformMatrix4fv(glGetUniformLocation(depthShader.shaderProgram, "lightSpaceTrMatrix"),

1,

GL\_FALSE,

glm::value\_ptr(computeLightSpaceTrMatrix()));

glViewport(0, 0, SHADOW\_WIDTH, SHADOW\_HEIGHT);

glBindFramebuffer(GL\_FRAMEBUFFER, shadowMapFBO);

glClear(GL\_DEPTH\_BUFFER\_BIT);

drawObjects(depthShader, true);

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

void initFBO() {

//TODO - Create the FBO, the depth texture and attach the depth texture to the FBO

//generate FBO ID

glGenFramebuffers(1, &shadowMapFBO);

//create depth texture for FBO

glGenTextures(1, &depthMapTexture);

glBindTexture(GL\_TEXTURE\_2D, depthMapTexture);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_DEPTH\_COMPONENT,

SHADOW\_WIDTH, SHADOW\_HEIGHT, 0, GL\_DEPTH\_COMPONENT, GL\_FLOAT, NULL);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

float borderColor[] = { 1.0f, 1.0f, 1.0f, 1.0f };

glTexParameterfv(GL\_TEXTURE\_2D, GL\_TEXTURE\_BORDER\_COLOR, borderColor);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_BORDER);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_BORDER);

//attach texture to FBO

glBindFramebuffer(GL\_FRAMEBUFFER, shadowMapFBO);

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, depthMapTexture,

0);

glDrawBuffer(GL\_NONE);

glReadBuffer(GL\_NONE);

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

}

## 3.2. Graphics model

The majority of the objects that I used were downloaded from the internet and imported into the main scene in blender, where I positioned them.

For some objects, I had to create my own textures and also I had to reduce the number of vertices of some objects that had a complexity that was too high.

For creating the animation, I had to split an object in 2 parts: one that is inside the main scene and one that will be drawn separately in the application (the part that rotates). In order to find the center of the rotating object, I also used blender, but these changes on the axes have to be made in order to obtain a correct representation inside OpenGL :

x=x

y=z

z=-y

## 3.3. Class Hierarchy

**Camera**: moving the camera based on the keyboard and mouse

**Mesh**: initialize VAO, VBO, EBO, drawing

**Model3D**: proecss object and texture data

**Shader**: shader data processing

**Skybox**: skybox data processing

# 4. User Manual

ESCAPE -> exit app

Left click and drag -> look around

W -> move forward

A -> move left

S -> move back

D -> move right

Z -> move up

X -> move down

Q -> rotate scene left

E -> rotate scene right

B-> normal mode

N -> wireframe mode

M -> vertice mode

Up arrow -> change light position (+)

Down arrow -> change light position(-)

J -> change light angle (+)

L -> change light angle (-)

F -> start fog

G -> stop fog

R- > start rain

T -> stop rain

# 5. Conclusions and further developments

This project was intersesting and fun to make but I did not have a lot of time for it so it still has room for further improvements. The shadows need to be recalculated because right now they move with the mouse when it changes positions. Collisions and multiple light points also need to be added.

I will definitely work again on this project or a new one, maybe using a different library/ engine because I am interested in game design and this could be a start for creating a simple game.

# 6. References

LearnOpenGL -> extremely useful page that contains examples and implementations of all the aspects needed for this project

<https://learnopengl.com/>

3D models that I used for my scene

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

<https://free3d.com/>

Code for rain

<https://gist.github.com/thaenor/4d9531cc9a7d1c34b998>