GRAPHICAL PROCESSING SYSTEMS

IOANA BOZDOG, GROUP 30431

Contents

1. Subject specification

This project entails the creation of a 3D photorealistic scene in Visual Studio using OpenGl and C++ using the skills that we acquired during this semester's Graphical Processing laboratory and course. The scene must comprise of a multitude of objects, as well as a display of different techniques, such as fog computation or collision detection.

2. Scenario

a. scene and objects description

The scene itself was created in Blender 2.79 using the documentation found on Moodle and various videos. It symbolises a post-apocalyptic scene comprised of different objects such as buildings, rocks, trees, ivy and of course, the nanosuit from the labs. Since we strive for a photorealistic scene, I also put a fence around the whole scene so as to not spoil the illusion, because otherwise, the viewer could easily see the whole object if he were to backtrack a little using the keyboard and mouse.

b. functionalities

The scene has multiple functionalities, so the user can do a multitude of things:

1. They can navigate through the scene (W, A, S, D)
2. They can change directions and go up and down (arrows)
3. They can visualize the scene as wireframe and as points and go back to solid (Q, E, R)
4. They can also visualize a demo animation of the entire scene (M)
5. They can also display grass (G, H)
6. They can switch from day mode to horror mode (N, B)
7. Lastly, the user can rotate the light source (J and L)

3. Implementation Details

a. functions and special algorithms

The scene was created using a variety of algorithms, applied to the scene in order to get certain effects. Some of these techniques were explained in the GP laboratory [1].

The project has a multitude of effects, so in the following lines there will be an explanation of how they were implemented.

The grass was imported as an individual object since if I were to put it together with the main object it would be hard to walk through the scene because of the collision detection (that applies to the entire scene). Because of this I thought it would be more fun to create the option of it being shown or not. I basically coded the object on a key.

The Camera Animation was implemented by hardcoding some values and manipulating the camera position. I initially thought of using the time library to make the program move the camera for x seconds, but I didn't succeed. Because of that, the next solution that I thought of was hardcoding some values. For rotation, I rotated the camera using an angle variable. The camera stops when the angle reaches a certain value. The animation also goes forward, which I implemented by using the function for controlling the camera movement (`myCamera.move(gps::MOVE_FORWARD, cameraSpeed);`) and it stops when the camera reaches a certain value on the z axis. This is done twice, and after everything stops, the variable that is used to verify if the user pressed the key for the demo is set to false (just to be safe).

The “horror mode” was implemented by basically making the “lightColor” variable global so that I could change its value whenever the user chooses to press the key.

The rotation of the wheel was obtained by importing only the wheel object and then using an angle to that controls the position of the wheel that increases by 0.1f. When that angle reaches 360 then it is “reset” to -360.

The dove animation was created by having a variable “movement” increase with a certain increment. That variable needed to be added on the z position of the vect3 from the translate function. If that variable reaches a certain value (in my case, +/-10.0) it needed to be reset (basically if the dove is too far , rotate it and change the increment so it’s the negative of before).

1. possible solutions

The fog computation, the point lights and the global light were implemented as in the laboratory guide (also from the laboratory) (although the global light did suffer some modifications – the function was modified to return a vec3).

The fog is generated by manipulating the attributes associated with the fog effect. The colour of the fog is achieved by using a clear colour. The actual fog computation is obtained by a square exponential formula given in the lab.

The global light was implemented like the light from the shadow demo, but in order to achieve point lights I needed to make a modification, namely to make the function return a vec3.

The point light was implemented as a variation of the global light; the modification was the attenuation coefficient, which was computed as per the instructions from the lab. There was also a table that I made use of when it came to modifying the intensity and the dimensions of the point light. The modification that was needed for the global light comes in handy when we call the “computePointLight” function because we add the resulting vec3 from that (“computeLightComponents”) function and add the point light function. The result is saved into the initial vec3.

The shadow was also implemented using the laboratory guide and also using the given code from the respective laboratory work. The shadows demo code was really helpful here and I took inspiration from it.

The hardest thing implemented was the collision resolution as it wasn’t described in the laboratory guide. The final technique (the one that was implemented) was comprised of computing the bounding box of the camera and also the bounding box of the chosen objects (in this case, the entire scene) and comparing them to see whether the next position of the camera (in any given direction) is entering the bounding box of the scene. If that is not the case, the camera should be able to move freely. If by mistake the camera manages to enter the object, it will automatically be pushed back.

2. the motivation of the chosen approach

The main motivation of the chosen techniques was that they were described in the laboratory guide, if not even completely implemented (the shadows).

However, this wasn’t the case with collision resolution, as it wasn’t even mentioned in the labs. The main technique was explained to us by our lab professor. It encompassed using spheres to get the general volume of the objects; if the distance from the centre of a sphere to the other was smaller than the sphere’s diameter, then the two objects would collide. This method was easy to understand but hard to implement. The next choice was creating bounding boxes for the objects, which I ended up implementing. Basically, every object needed a

minimum and a maximum (vec3) to understand its volume. The camera also needed the same treatment. Then, every time the camera would change its position there was needed a verification to check if the next position of the camera would enter an object (which was implemented by comparing their minimums and maximums).

b. graphics model

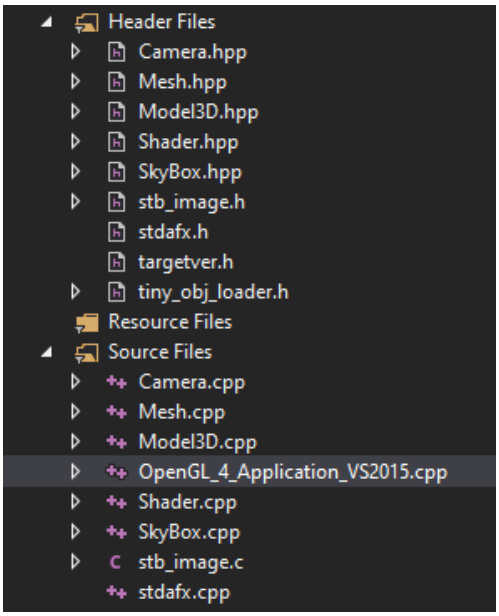
The project is created using the Blinn-Phong Lighting Model.

c. data structures

There were several data structures used in order to complete this project, the most important being vec3, mat4, GLuint VAO (vertex array object), VBO (vertex buffer obj.), FBO (frame buffer obj.), Vertex, Texture, Material.

d. class hierarchy

The project hierarchy can be seen below (both the header files and the source ones)



4.Graphical user interface presentation / user manual

The following pictures are taken from the program to display the scene and its capabilities. The functionalities of this scene have been presented above, so if the user wishes to see the possible actions they can take or how they could control the movement, section 2 takes care of that.

I have to mention that I haven't been able to decide between two of the versions of the scene that I have created. One is more of a post-apocalyptic scene, whilst the other is more of an abandoned city with lots of grass. The only problem is that initially I thought of displaying both scenes depending on which key the user would press, but the scenes are too large and I cannot fit them both.

Version 1 snapshots:

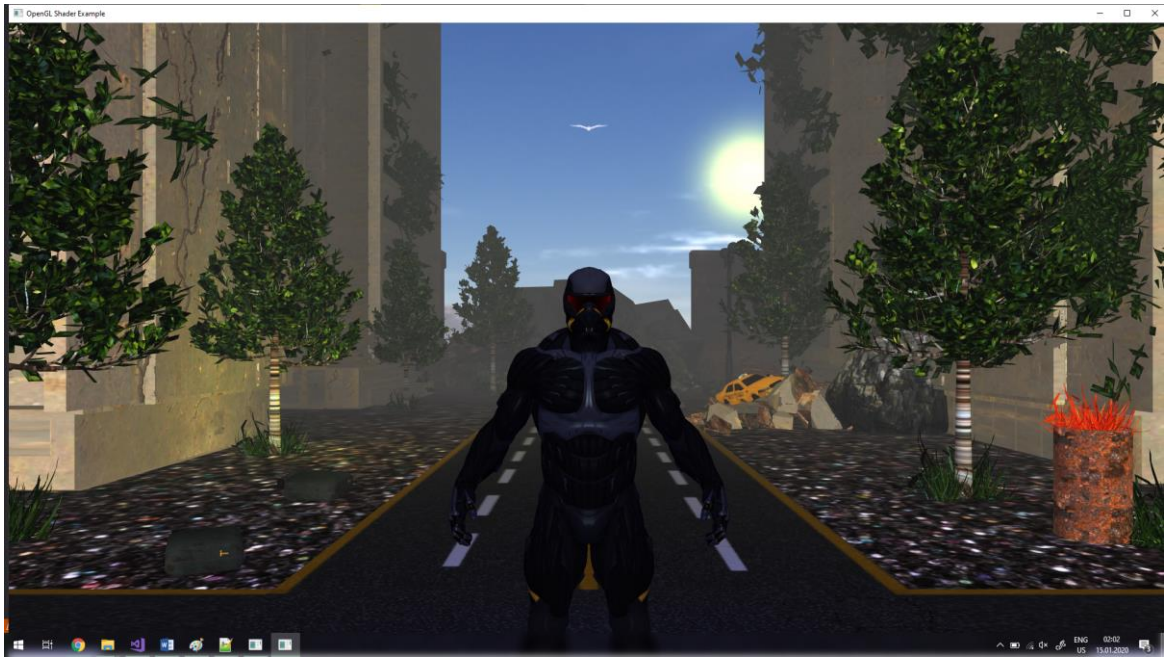


Fig. 1 The opening scene, fog

Fig. 2 Point lights, fog

Fig. 3 Shadow, point light, animation (the wheel rotates)

Fig. 4 Displaying the grass, fog

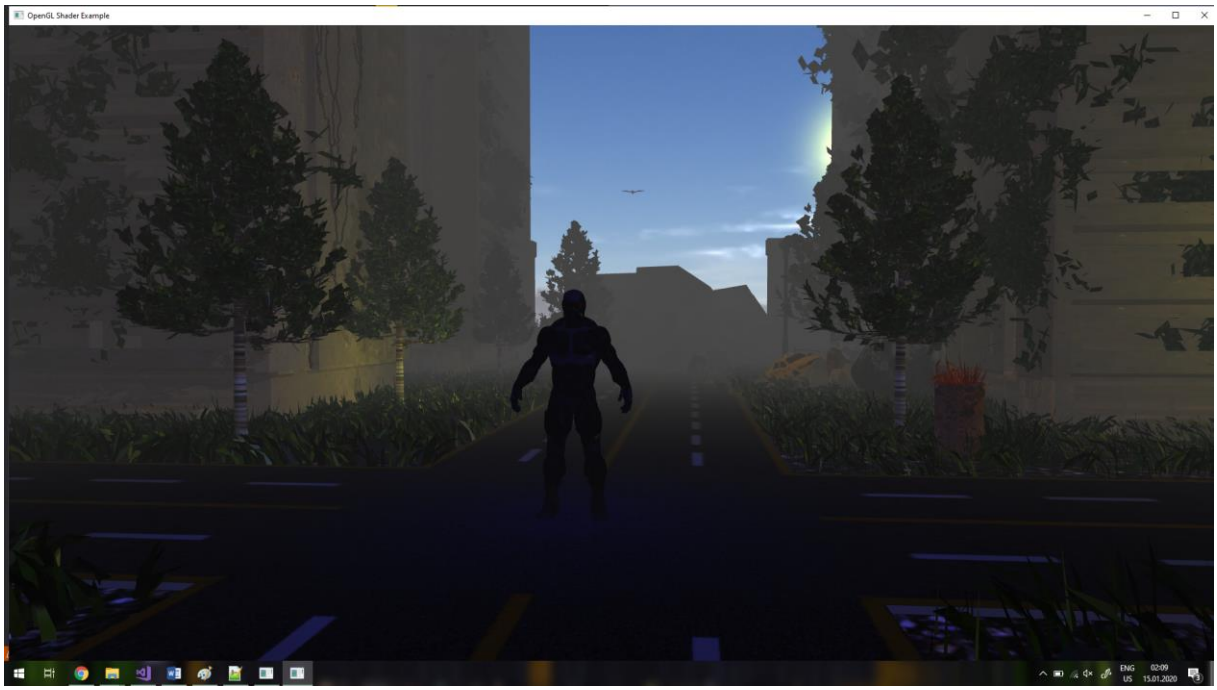


Fig. 5 Horror mode with grass on, fog

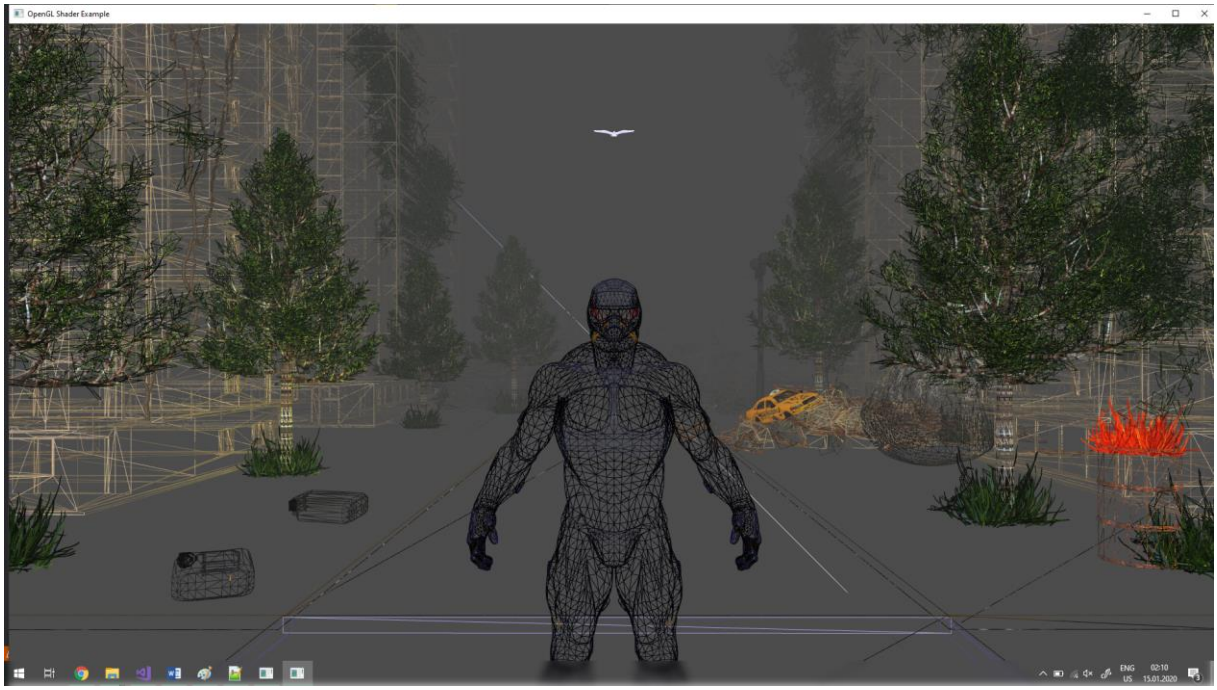


Fig. 6 Line mode

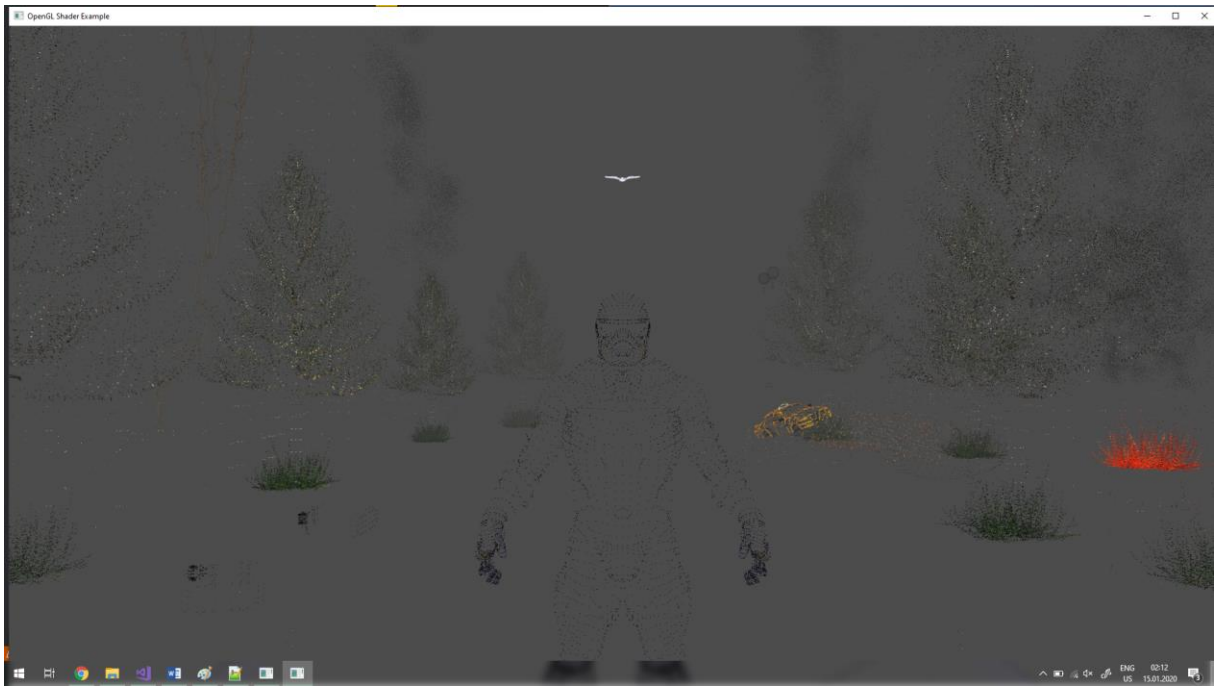


Fig. 7 Point mode

Version 2 snapshots:

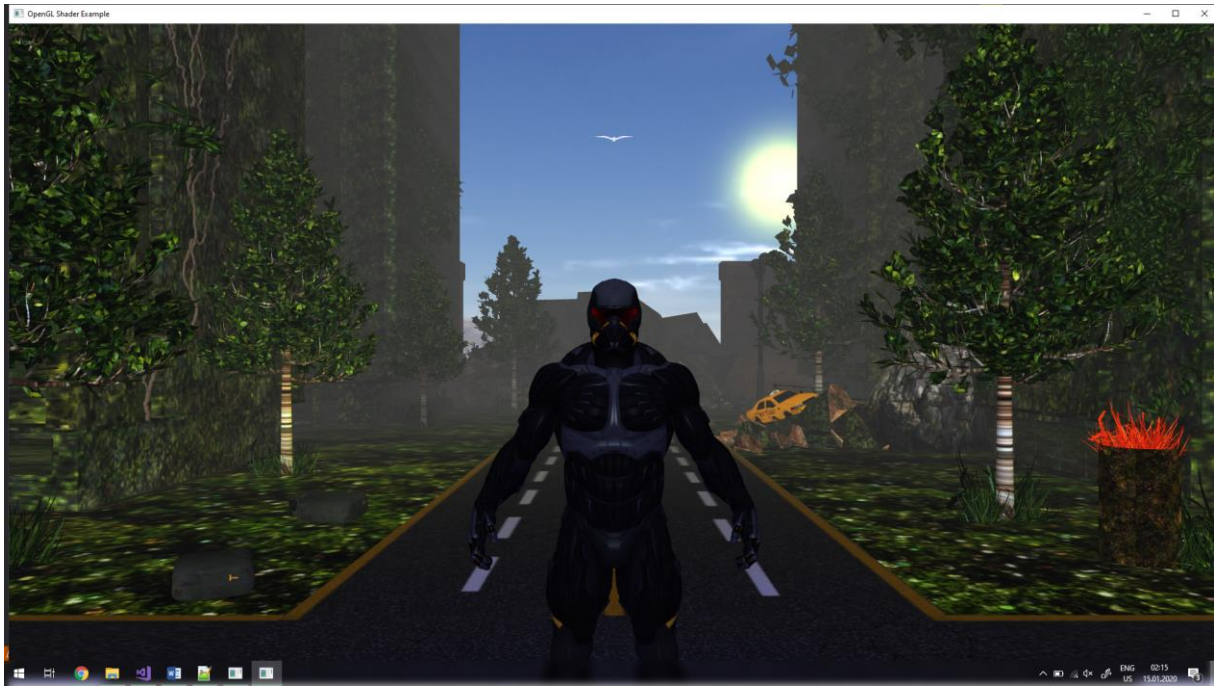


Fig. 8 The opening scene, fog

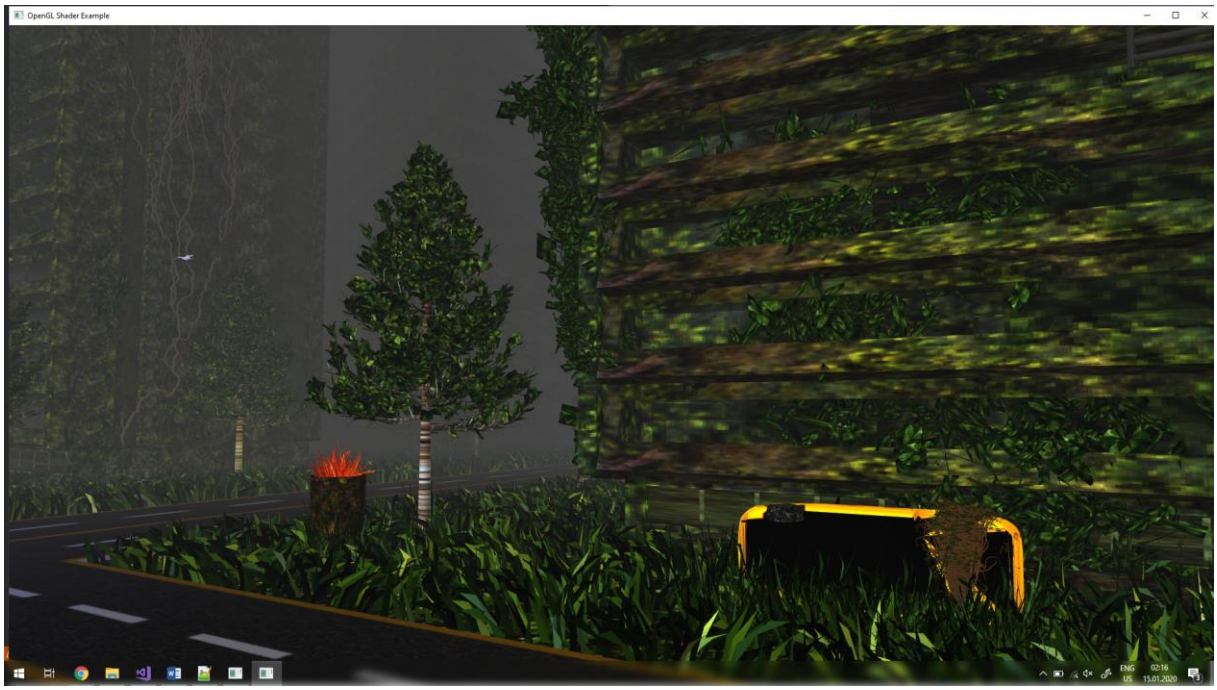


Fig. 9 Shadows, grass, point light and animation, fog, grass on

Fig. 10 Horror mode, point lights, fog, grass on

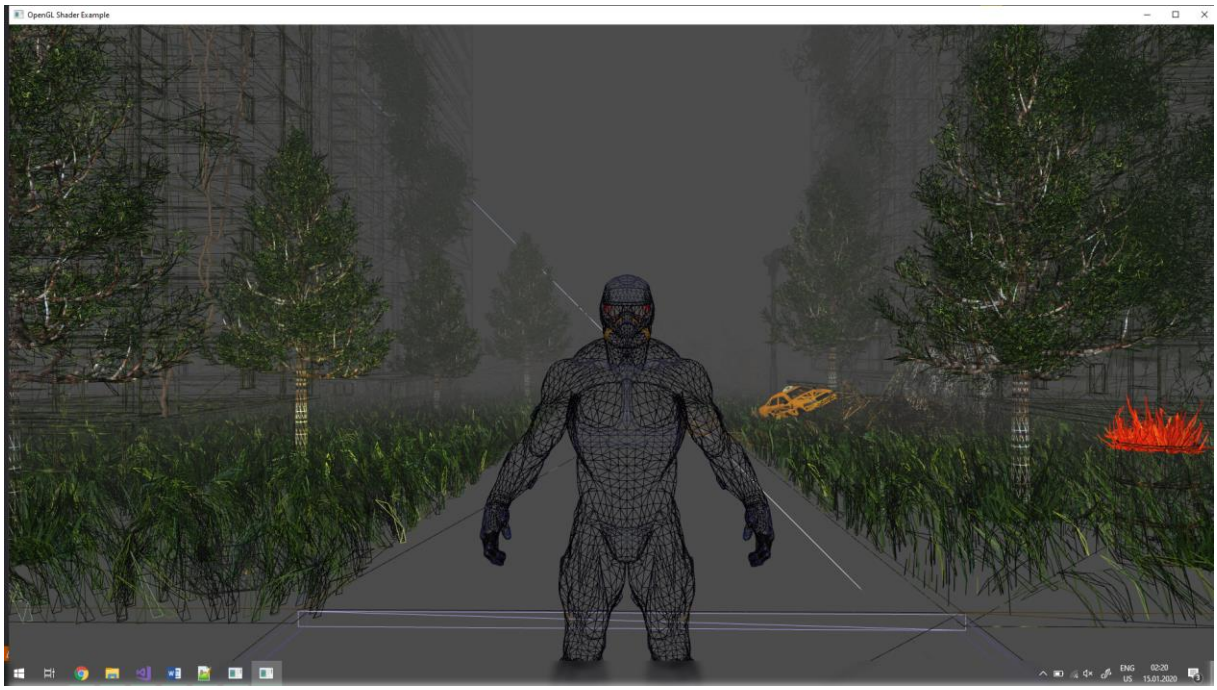


Fig. 11 Line mode, grass on

Fig. 12 Point mode, grass on

5. Conclusions and further developments

This project was quite the challenge, given that I had no previous experience in Blender or in OpenGl, but despite that I quite enjoyed creating the scene and adding effects such as the fog or the shadows.

For further development I would like to polish the scene in Blender and add a lot more functionalities and effects to make the scene a lot more photorealistic. I would also like to create more animations and add sound if possible.

6.References

[1] graphical processing systems laboratory work:
https://moodle.cs.utcluj.ro/course/view.php?id=188

7.Screenshots from previous versions of the scene