HORROR GAME SCENE PROTOTYPE

Student Name: Cristea Tudor

Group: 30433

Coordinating Professor: Bâcu Victor

CONTENT

1) SUBJECT SPECIFICATION

The purpose of this project is to get familiar with the basic functions of the OpenGL rendering pipeline and then utilize it in order to implement a scene of objects in which various algorithms and functionalities are implemented such as lighting, shadows and camera movement ([5]). The main objective is represented by realism (interaction with the objects of the scene) and photo-realism (the look and feel of the scene).

2) SCENARIO

2.1) SCENE AND OBJECT DESCRIPTION

The scene that I implemented in my application represents a house in the middle of the forest, surrounded by trees and hills, during the winter season, hence the presence of a snowfall. The directional light is a bit dimmer since it is happening at nighttime. Near the starting position of the "player", there is a nice, reflective/refractive, sparkly, murky and dynamic lake in the middle of which there is a muddy rubber duck, as well as a rusty, old lamp post, which emits an orange-y light. In front of the "player" there is a big house, to which they can walk to, being guided by a dirt road. Then, the "player" can enter the house only if they open the door. However, in order to open the door, the "player" has to find the key, which is somewhere between the trees that are nearby the house. After they have acquired the key, they can approach the door and open it. Once the "player" is inside the house, they notice that the ground floor is empty, with the exception of a trapdoor that was left wide open, inviting the "player" to an unknown, underground room. Two flights of stairs guide the "player" down to the basement of the house, where a cute little cat, which is standing on a dusty mattress, is being held hostage against its own will by the owner of the house. Besides the cat, there are numerous other non-living objects in this basement such as a shovel, a crate, an explosive barrel, an arcade game, a vacuum cleaner, a washing machine, and a shelf onto which there are various cleaning objects and empty cans of food, as well as a tub of paint. This area is illuminated by a flickering, green(-ish) light. By the looks of things, this surely is the house of a serial kidnapper (but the "player" has no reason to worry, as the owner of the cottage will not be back home any time soon). Moreover, the "player" is equipped with a complementary horror game flashlight which can be activated at any time by pressing the "F" key on the keyboard and which can be used in order to better highlight a certain part of the scene or a certain object.

2.2) FUNCTIONALITIES

By using the "W", "A", "S" and "D" keys on the keyboard, the "player" can move throughout the scene of objects while having the "height" of the camera locked to a certain value so that they cannot go below or above the map. In addition, by using the mouse, the "player" can look around the scene ([4]).

By holding the "LEFT SHIFT" key on the keyboard, the "player" can sprint throughout the map, since its movements are sped up by double of the normal amount. By releasing the key, the speed turns back to normal.

By pressing the "F" key on the keyboard, the "player" can activate/deactivate the flashlight that has the same position and orientation as the camera. The flashlight itself is not just a circle, but it rather has cut-off zones in order to make it seem more realistic by smoothing out the edges of the light that is projected by it. Moreover, it also includes an attenuation factor

which again helps with the realism of the scene because the light is brighter on objects that are closer to the camera and dimmer on objects that are further away from the "player".

By pressing the "O" key on the keyboard, the "player" activates an automatic animation, which moves the camera through the scene of objects, highlighting the important aspects of the scene. After, the camera stops moving, signaling the fact that the animation is over, the "player" can press the "O" key again, bringing them back to the initial position. In addition, throughout the entire duration of this animation, the "W", "A", "S" and "D" keys of the keyboard, as well as the mouse, will be deactivated in order to ensure that the "player" does not create unnecessary visual bugs. This animation can only be triggered once per gameplay session.

By pressing the "P" key on the keyboard, the "player" can pick up the key from the floor, but only if their location is very close to that of the key's. After acquiring the key and going near the entrance of the house, the "player" can press the "P" key again in order to trigger the animation that opens the door of the house. This animation implies three moving parts, each having their own animation, while two of them also have an animation that depends on the third part. The first and the most important part is the door itself, which performs a rotation around the axis that goes through its supposed hinge. The second part is the door handle which also performs the same rotation as the door, but in addition, performs another rotation around its own center, thus mimicking the action of pressing and then releasing the door handle ([11]). The third and final object is the door lock (the little block that slides in and out of the door frame in order to let the door be opened or not) which also performs the same rotation as the door, but in addition, performs a translation towards the inside of the door and then back out, in order to mimic its motion when a door is getting opened. In addition, the player's position will be locked (the mouse can still be used though) throughout the entirety of the animation, in order to ensure that there are no bugs being created by the interaction between the "player" and the moving door. This animation can only be triggered once per gameplay session.

By pressing the "Y" key on the keyboard, the scene turns from polygonal objects to their wireframe line counterparts. By pressing the key a second time, the scene turns to the wireframe point mode. By pressing it a third time, the scene turns back to normal.

By pressing the "N" key on the keyboard, the scene turns from smooth shading (default) to flat shading, meaning that the vertex normals are not interpolated when they are transmitted from the vertex shader to the fragment shader ([15]). By pressing the key again, the scene turns back to normal.

By pressing the "M" key on the keyboard, the depth map of the scene is shown on the screen. By pressing the key again, the scene turns back to normal.

By using the "I" and "J" keys on the keyboard, the "player" can translate the entire scene of objects on the Z axis. Similarly, by using the "J" and "L" keys, the scene can be translated on the X axis and by using the "G" and "H" keys, the scene can be translated on the Y axis.

By pressing the "Z" and "X" keys on the keyboard, the "player" can scale the scene up or down.

By pressing the "Q" and "E" keys on the keyboard, the "player" can rotate the scene to the left or right.

Navigating throughout the scene, most objects (house, walls, door, lake, invisible surrounding barrier) have collision detection (the trees are an exception). In addition, when going up and down the stairs, the "height" of the camera increases and decreases accordingly.

To add another degree of realism, the scene is surrounded by a nighttime skybox which gives the impression of an extension of the scene. Furthermore, fog is also present in the scene, but only in the "outside" areas (so, there is no fog inside the house or the basement) ([5], [10]). Another element that is visible only in the "outside" areas is represented by the snowfall.

In addition to the more general, directional light, which, in this case, is considered to be represented by the full moon, there are also several point lights (eight to be exact) surrounding the lamp post's glass components and another one in the basement, the attenuation coefficients

of which have been configured as appropriately as possible, in order to further add to the photo-realism of the entire scene. Moreover, there is a spotlight (the flashlight) present in the scene as well, which was already mentioned and explained above ([2], [3]). Furthermore, the directional light, as well as the point light from the basement, generate shadows ([12], [14]).

When it comes to the objects themselves, I tried to find the highest quality objects and corresponding textures for the lowest possible cost (i.e., for free) ([8], [9], [10]), but by using Blender, I used the "Decimate" modifier wherever I considered necessary, in order to reduce the number of polygons of an object while still maintaining that high quality feel ([6], [7]). This was done in order to obtain the least possible impact on the framerate.

3) IMPLEMENTATION DETAILS

For the automatic scene animation, Bézier curves were used, namely one Bézier curve that has five control points and then three other Bézier curves, each of which only have two control points ([1]). This decision was made because of the layout of the house and the basement (for example, the camera had to go straight down the stairs while looking ahead; this could have been done only with a Bézier curve that has only two control points). By using this method, instead of manually making the camera go between one point and the next or by using another algorithm, the presentation of the scene is smoother and free of jitter or unnatural transitions. The parameters that were used for the Bézier curves are the three coordinates of the camera position (x, y and z) for moving the camera, as well as the yaw and pitch for rotating the camera.

For the collision detection algorithm, several methods were used together depending on the object that the "player" would collide with. For the invisible barrier surrounding the map of the scene, several key points were chosen and by taking them in pairs of consecutive points, their corresponding equations of lines were computed and if one of the coordinates of the "player" (either the x or the z) would place the camera to the left of one of these lines, then that move would not be executed. Thus "illegal" moves are prevented rather than corrected. For the lake or the house, rectangular shapes are considered and if the so-called "future position" of the camera would lie inside one of these rectangles, then that move would not be made. For the basement stairs, in which the "height" of the camera is changing continuously, only two lines were considered, in between which the "player" is allowed to move freely (on the other two axes).

For the movement on stairs, rectangles were considered again, only this time, if the "player" lies inside one of them, then the y coordinate of the camera **must** have a certain value. Therefore, for multiple, successive stairs, a constant value is incremented or decremented to the initial one (the one that the player starts the game with) and then that resulted value is attributed to the y coordinate of the camera's position. Another method which I have tried before this one, but was inferior, was to instead consider so-called "barriers" which the "player" would walk through and that would trigger an increment or decrement of the current position of the camera (so, instead of setting the y coordinate of the camera to a certain, predefined value, it would directly be modified). This yielded various bugs and an unexpected behavior, which led me to seek an alternative, better method. Another advantage of this method over the other one is that if the "player" chooses to stay in between two consecutive stairs, the camera's y coordinate will not go back and forth between the two values corresponding to those two stairs, but instead it will stick to one of them depending on the precise location of the "player".

In my application, there are mainly two scenes which are rendered together, namely the exterior one and the interior one (to which the door, the door handle and the door lock are added separately since they are the moving parts of the object animation). Each of them has a separate shader that is being rendered with. This decision was made because of the differences in

lighting and photo-realism between the two. On the one hand, the objects of the exterior scene need to be influenced by the light of the lamp post that is outside (mainly the objects that are in its immediate vicinity) and need to have a layer of fog on top of them. On the other hand, the objects of the interior scene need not to be influenced by the lamp post's light, nor do they need that layer of fog, but instead, they need to be influenced by the light of the basement (mainly the objects that are in the basement and not the ones outside it).

For the implementation of shadows, a depth map texture was used for the directional light, while a cube map texture was used for the point light in the basement. For both of these, the scene was rendered to a separate framebuffer object (which has one of these textures attached to it) instead of the default one (which is used for rendering on the screen), and in a different coordinate space, namely the one defined as if the position of the camera would coincide with the position of the light. In order to make the shadows of the exterior scene seem smoother, a bit of percentage-closer filtering (PCF) was performed on them. Similarly, this procedure was also applied on the interior shadows. In order to obtain these interior shadows, a separate shader that included a geometry shader, besides the usual vertex shader and fragment shader ([13]), had to be created. This new shader comes in between the two existing ones and is responsible with transforming the vertices from world space to the 6 different light spaces (which coincides with the number of faces of a cube).

For the snowfall, I have constructed a particle system, which consists of a number of small, spherical objects that continuously change their positions from high up in the air and heading towards the ground/roof of the house, with a random but constant speed. When they reach a certain height, they are reset to a random starting position and they are given a new, random speed. In order for their movement to be smooth, Bézier curves are used once again, the control points of which are computed randomly but within a certain range based on the starting position coordinates.

For the lake near the starting position of the player, I have made it such that it is a realistic lake meaning that it has a reflection, as well as a refraction. This was done using two additional framebuffer objects, each of which have a (color) texture attached to them, and the one used for refraction also has a depth texture attached to it. In addition, I have implemented a small functionality that enables this new texture of the water to move ever so slightly in order to give the illusion of a realistic lake. This was done by using a "DuDv" map, which can essentially be used to add a distortion to the texture of an object, and this distortion is adjusted based on a time variable in order to create waves in a continuous and dynamic manner. Moreover, in order to increase the degree of realism of the lake, the Fresnel effect was implemented meaning that the if the "player" looks down at the lake (makes an acute angle with the normal of the lake) then the refraction part will be more visible and if the "player" looks at it from its side (makes an obtuse angle with the normal of the lake), then the reflection part will be more visible. Furthermore, through the use of a "normal" map, I have also added specular highlights to the water in order to add the effect of (moon) light hitting the small waves of the lake. Using the depth texture of the refraction FBO, I have added a murky water effect, such that the lake's water is clearer towards the surface and the sides, and dirtier at the bottom ([16]).

4) CONCLUSIONS AND FURTHER DEVELOPMENTS

In conclusion, the OpenGL rendering pipeline offers a very flexible and fast way to render a graphics scene of objects. However, the downside is that it is very complex and hard to debug, because there are many parts of the code (the code in main and the code in the shaders) which run in parallel. Additionally, only a small mistake can "break" the rendering process, thus making it a bit frustrating to work with this pipeline. However, and alternative would be the "glut" library, which is based on the OpenGL pipeline and is easier to understand, as well as more convenient to use, but it does not offer that much flexibility. My experience

working with it was not very good at the beginning, but I must admit that seeing the end result after I correctly and completely implemented an algorithm was very satisfying because it can literally be seen and interacted with. Overall, the satisfaction and the pleasure of working on this project kept increasing continuously, as I gained more and more experience with this pipeline.

It goes without saying, but the application can be further developed in the following ways:

- expanding the scene even further in order to make it more spacious
- adding other areas that can be explored and other objects that can be interacted with
- adding other natural elements that are linked to the weather such as rain or wind
- adding a time parameter according to which the lighting elements and the skybox would gradually change
- develop a better (more accurate) collision detection mechanism, especially for uneven terrains such as hills or pits
- add hair/fur to the cat and animate it so that it can walk around
- add a model/character for the "player" and animate it, so that they can see a moving body when they look down
- implement advanced graphics concepts such as bloom, anti-aliasing, motion blur, HDR, SSAO
- visually communicate to the "player", the tasks that they have to do or some hints that they might find useful
- tweak the existing control points of the Bézier curve and add new ones to the scene presentation animation
- add NPC-s to the scene that interact with the "player" or pose a threat to them

5) REFERENCES

- [1] "Bézier curve" [Online] https://en.wikipedia.org/wiki/B%C3%A9zier_curve#Linear_curves
- [2] Joey de Vries, "Light casters" [Online] https://learnopengl.com/Lighting/Light-casters
- [3] Joey de Vries, "Multiple lights" [Online] https://learnopengl.com/Lighting/Multiple-lights
- [4] Joey de Vries, "Camera" [Online] https://learnopengl.com/Getting-started/Camera
- [5] Graphics Processing Course - Laboratory Guides [Online] https://moodle.cs.utcluj.ro/course/view.php?id=613
- [6] Cosmin Nandra, "Blender – Quick Tutorial" [Online] https://docs.google.com/document/d/1njtWPMmOQNIaD_z9ve8iPRUqQTWdIV_PO-NvPD0nOuM/edit?pli=1#heading=h.7h28ckb0ku81
- [7] Cosmin Nandra, YouTube videos on Blender tutorials [Online] https://www.youtube.com/watch?v=ckGg7aUGoPQ&list=PLrgcDEgRZ_kndoWmRkAK4Y7ToJdOf-OSM&index=1
- [8] Free3D (for obj's and mtl's) [Online] https://free3d.com/
- [9] cgtrader (for obj's and mtl's) [Online] https://www.cgtrader.com/
- [10] Google (for seamless textures and the skybox) [Online] https://www.google.com/
- [11] "Find the coordinates of a point on a circle" [Online] https://math.stackexchange.com/questions/260096/find-the-coordinates-of-a-point-on-a-circle
- [12] Joey de Vries, "Shadow Mapping" [Online] https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

- [13] Joey de Vries, “Geometry Shader” [Online] https://learnopengl.com/Advanced-OpenGL/Geometry-Shader
- [14] Joey de Vries, “Point Shadows” [Online] https://learnopengl.com/Advanced-Lighting/Shadows/Point-Shadows
- [15] “"flat" qualifier in glsl?” [Online] https://stackoverflow.com/questions/27581271/flat-qualifier-in-glsl
- [16] “OpenGL Water Tutorials”, ThinMatrix [Online] https://youtube.com/playlist?list=PLRIWtICgwaX23jiqVByUs0bqhnalNTNZh&feature=shared