12 January 2021

Graphics Processing

Student: Todea Tudor-Stefan

Lab coordinator: Oros Bogdan Ioan

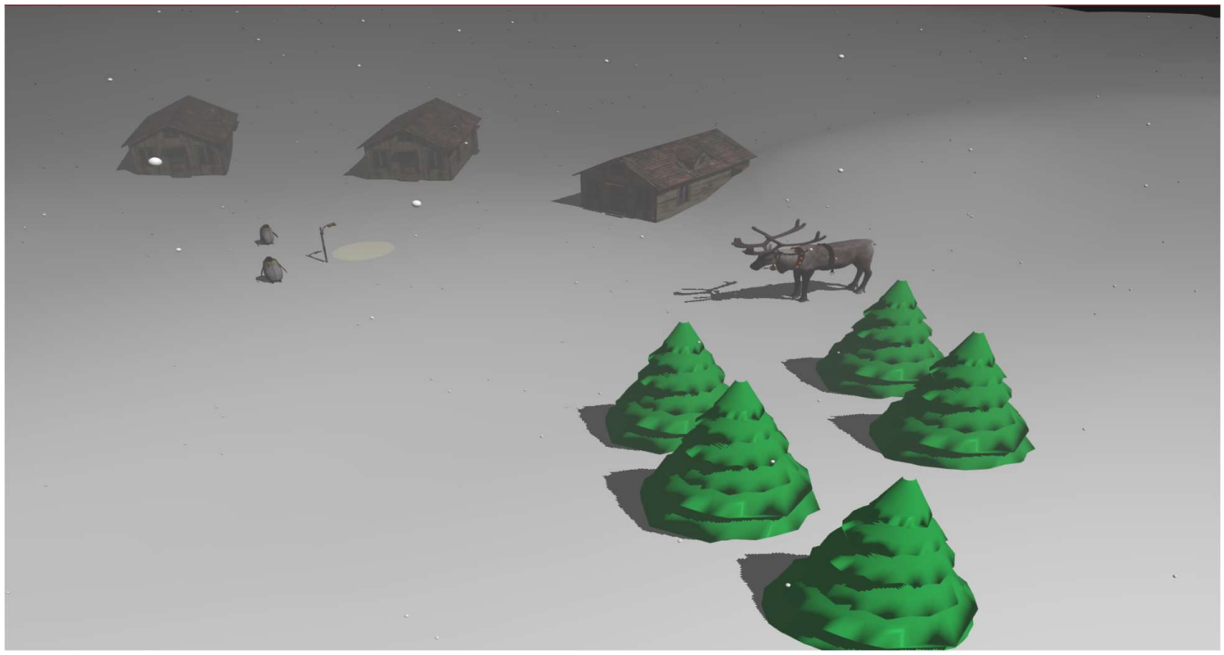
Contents

1. Subject Specification

We were required to design a scene displays a photorealistic presentation of 3D objects using OpenGL. The scene can be manipulated by the user through a series of inputs that will be described later in the documentation. The development was done by using the materials taught throughout the semester during the laboratory and by further research on the internet.

2. Scenario

2.1 Scene and Objects Description



The project is a simple scene made to highlight certain functionalities which will be presented in the next subsection. The scene contains multiple objects taken from various websites that provide free resources [1]. The penguins are used to exemplify two different types of animation: animation of a whole object and animation of object components. The light pole is used as a source for a spotlight and spheres with a white texture are used as snowflakes. The terrain has been tweaked in Blender. As for the background we have used a textured Sky Dome. The other objects such as the trees, cabins and the reindeer are present just to add to the complexity of the scene.

2.2 Functionalities

We have implemented the following functionalities:

- The camera can be manipulated by the user using the mouse and the keyboard. [2]
- The scene cand also be visualized using a predefined camera animation.
- There are two different light sources: a “global” directional light and a spotlight coming from a fixed source.
- Through keyboard commands the user can view the surfaces by using the default visualization mode, enabling AA to view smooth surfaces or enabling the wireframe mode to view the vertices and polygons of the objects.
- Each object has its own detailed and unique mapped texture.
- Each object aside from the snowflakes casts a shadow determined by the directional light.
- To exemplify animation of object components, one of the penguins moves its wings.
- We have also implemented a snowfall environmental effect with keyboard adjustable windspeed and randomly generated snowflake coordinates. There is also a fog environmental effect present.
- We have exemplified collision detection by building a bounding box around the moving penguin.

3. Implementation

3.1 Functions and Special Algorithms

3.1.1 Snowfall and Wind

For the snowfall we took 10000 spheres with a white texture and placed them in random positions across the scene. The data for each snowflake is encapsulated in a data structure that looks like this:

```
struct Snowflake {
	glm::vec3 startingPosition;
	glm::vec3 currentPosition;
};
```

Basically, we generate 10000 random starting positions for each snowflake. During the scene rendering phase we check the movement of each snowflake to see if we hit the ground or the bounding box of the penguin or, since we have wind on the x axis, we also check if the snowflake is out of the scene on the x axis. If we are in such a case, we place the snowflake in its starting position on the x axis, the position on the z axis remains unchanged and the position on the y axis is reset to the top of the scene (we don’t place the snowflake at the starting position on the y axis since that value is random and we don’t want

snowflakes appearing out of the void in our scene). If there is no problem, we move the snowflake in the next position, which is decreased on the y axis by a constant variable and on the x axis by the speed of the wind. The final algorithm looks like this:

```
for (int i = 0; i < 10000; i++) {
    if (!checkBBCollision(penguinBB, glm::vec3(snowflakeArray[i].currentPosition.x + windSpeed, snowflakeArray[i].currentPosition.y - yFall, snowflakeArray[i].currentPosition.z))) {
        model = glm::translate(glm::mat4(1.0f), glm::vec3(snowflakeArray[i].currentPosition.x + windSpeed, snowflakeArray[i].currentPosition.y - yFall, snowflakeArray[i].currentPosition.z));
        model = glm::scale(model, glm::vec3(0.5f));
        glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE, glm::value_ptr(model));

        normalMatrix = glm::mat3(glm::inverseTranspose(view * model));
        glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));

        snowflake.Draw(shader);
        if (snowflakeArray[i].currentPosition.x > 50.0f || snowflakeArray[i].currentPosition.x < -50.0f) {
            snowflakeArray[i].currentPosition.x = snowflakeArray[i].startingPosition.x;
        }
        else {
            snowflakeArray[i].currentPosition.x = snowflakeArray[i].currentPosition.x + windSpeed;
        }
        if (snowflakeArray[i].currentPosition.y < -1.0f) {
            snowflakeArray[i].currentPosition.y = 70.0f;
        }
        else {
            snowflakeArray[i].currentPosition.y = snowflakeArray[i].currentPosition.y - yFall;
        }

    }
    else {
        snowflakeArray[i].currentPosition.y = 70.0f;
    }
```

3.1.2 Spotlight Lightning

The algorithm [3] for this type of lightning is quite simple. We take the source of light (which in our scene is a point on the head of the light pole) and a destination, which is a point on the ground. We pass those two variables to our fragment shader through uniforms:

```
spotLightPos = glm::vec3(0.3f, 1.3f, 0.0f);
spotLightPosLoc = glGetUniformLocation(myCustomShader.shaderProgram, "spotLightPos");
glUniform3fv(spotLightPosLoc, 1, glm::value_ptr(spotLightPos));

spotLightDir = glm::vec3(1.0f, 0.0f, 0.0f);
spotLightDirLoc = glGetUniformLocation(myCustomShader.shaderProgram, "spotLightDir");
glUniform3fv(spotLightDirLoc, 1, glm::value_ptr(spotLightDir));
```

We also need to compute the fragment position in the eye coordinate system:

```
fPosSpotEye = model * vec4(vPosition, 1.0f);
```

And finally, the last computation to determine the angle between the normalized vector determined by the fragment position and the spot light position, and the normal of the spot light direction vector.

```
float theta = dot(normalize(spotLightPos - fPosSpotEye.xyz), normalize(-(spotLightDir - spotLightPos)));
```

We set the cut off angle of the spotlight to 25 degrees. If theta is larger than the cut off angle, it means that the fragment is in the spotlight. We apply and orange color to the points inside the spotlight:

```
if (theta >= cos(radians(25.0f))) {
    ambient +=vec3(0.9f, 0.35f, 0.0f);
    diffuse += vec3(0.9f, 0.35f, 0.0f);
    specular += vec3(0.9f, 0.35f, 0.0f);

}
```

3.1.3 Collision Detection

To detect collisions with a certain object we must create a bounding box around that object. In this project we have selected a moving penguin object on which to exemplify this algorithm. To create the bounding box we iterate through each mesh of the object. Each mesh contains a vector of Vertex structures which contain the position of the certain object. We go through every vertex and select the maximum and minimum values for x, y, z between all the vertices of the object. We store these values in a custom BoundingBox structure:

```
struct BoundingBox {
    glm::vec3 mins;
    glm::vec3 maxs;
};
```

Now we have a function that checks the collision between a vec3 and the bounding box of the penguin. If the coordinates of the vec3 are between the minimum and maximum values of the bounding box on all three coordinates axis, then the point collides with the bounding box. We check the collision between each snowflake and the bounding box, as well as collisions between the camera and the bounding box.

3.1.4 Animations

All the animations are bound by frame rate, meaning that they are dependent on the number of scene renditions that happen each second.

In order to animate the moving penguin we have created an array of 3600 successive positions in which the penguin is translated. Alongside translations, the penguin performs a series of multiple rotations, different for each stage of movement and different for each foot, in order to simulate a somewhat realistic movement for a penguin.

```
for (int i = 0; i < 3600; i++) {
    penguinPath.push_back(glm::vec3(-2.0f, -0.45f, 0.002f * i));
}
```

The movement on each foot is done in 120 renditions determined by the variable penguinMovementIndex in which the body of the penguin performs multiple rotations. During these rotations we translate the penguin a little further by incrementing the penguinIndex variable during the stage in which the penguin rotates alongside the y axis.

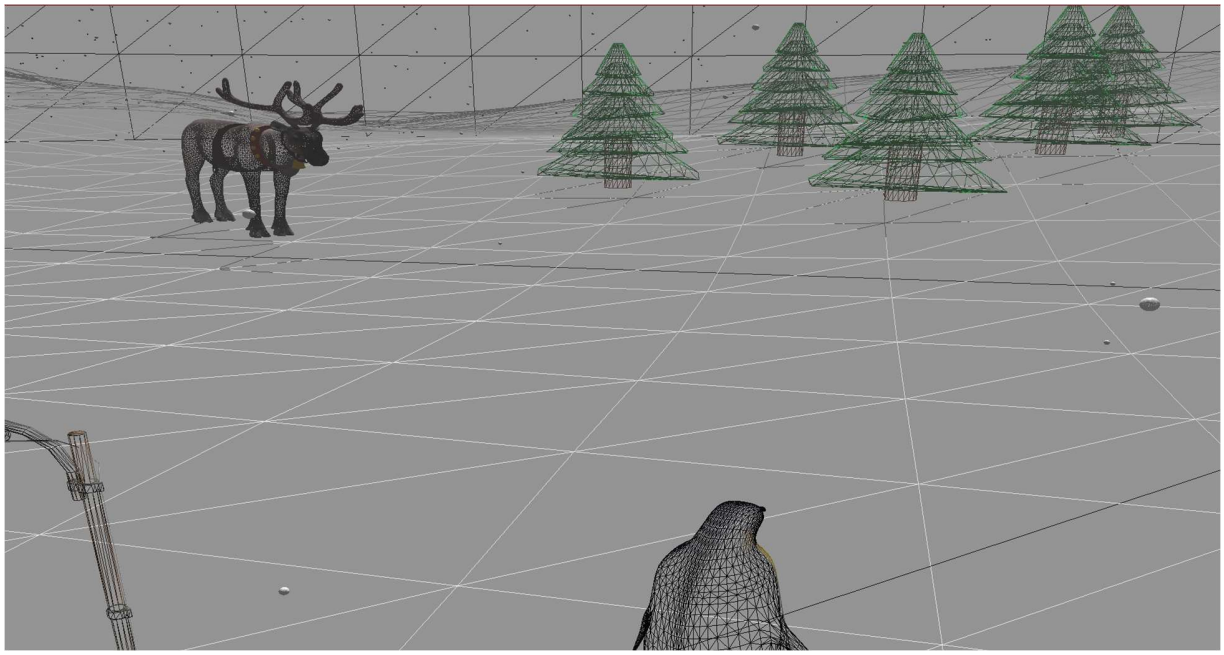
```
if (penguinIndex < 3599) {
    penguinMovementIndex++;
    if (penguinMovementIndex < 40) {
        if (foot == 0) {
            modelWholePenguin = glm::rotate(modelWholePenguin, glm::radians(0.12f * penguinMovementIndex), glm::vec3(0.0f, 0.0f, 1.0f));
        }
        else {
            modelWholePenguin = glm::rotate(modelWholePenguin, glm::radians(-0.12f * penguinMovementIndex), glm::vec3(0.0f, 0.0f, 1.0f));
            modelWholePenguin = glm::rotate(modelWholePenguin, glm::radians(-24.0f), glm::vec3(0.0f, 1.0f, 0.0f));

        }
    }
    if (penguinMovementIndex < 80 && penguinMovementIndex >= 40) {
        if (foot == 0) {
            modelWholePenguin = glm::rotate(modelWholePenguin, glm::radians(4.8f), glm::vec3(0.0f, 0.0f, 1.0f));
            modelWholePenguin = glm::rotate(modelWholePenguin, glm::radians(-0.6f * (penguinMovementIndex - 40)), glm::vec3(0.0f, 1.0f, 0.0f));

        }
        else {
            modelWholePenguin = glm::rotate(modelWholePenguin, glm::radians(-4.8f), glm::vec3(0.0f, 0.0f, 1.0f));
            modelWholePenguin = glm::rotate(modelWholePenguin, glm::radians(-24.0f + 0.6f * (penguinMovementIndex - 40)), glm::vec3(0.0f, 1.0f, 0.0f));
        }

        if (penguinMovementIndex % 5 == 0) {
            penguinIndex++;
        }
    }


    if (penguinMovementIndex < 120 && penguinMovementIndex >= 80) {
        if (foot == 0) {
            modelWholePenguin = glm::rotate(modelWholePenguin, glm::radians(4.8f - 0.12f * (penguinMovementIndex - 80)), glm::vec3(0.0f, 0.0f, 1.0f));
            modelWholePenguin = glm::rotate(modelWholePenguin, glm::radians(-24.0f), glm::vec3(0.0f, 1.0f, 0.0f));
        }
        else {
            modelWholePenguin = glm::rotate(modelWholePenguin, glm::radians(-4.8f + 0.12f * (penguinMovementIndex - 80)), glm::vec3(0.0f, 0.0f, 1.0f));

        }

    }


    if (penguinMovementIndex == 119) {
        penguinMovementIndex = 0;
        if (foot == 0) {
            foot = 1;
        }
        else {
            foot = 0;
        }
    }
}
```

The second type of animation is done by the second penguin. For this one we animated only some components of the object, the wings, which rotate alongside the z axis. We separated the wings from the body of the penguin creating three different objects and we rotate the wings upwards 110 degrees. When we reach 110 degrees, we switch the variable raiseWings to 0 and the wings begin to descend to their original position.

```
modelPenguin = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -0.45f, -3.0f));

modelPenguinLeftArm = glm::translate(modelPenguinLeftArm, glm::vec3(0.18826f, -0.45f + 0.66209f, -3.0f + 0.012794f));
modelPenguinLeftArm = glm::rotate(modelPenguinLeftArm, glm::radians(wingsAngle), glm::vec3(0.0f, 0.0f, 1.0f));

modelPenguinRightArm = glm::translate(modelPenguinRightArm, glm::vec3(-0.18826f, -0.45f + 0.66209f, -3.0f + 0.012794f));
modelPenguinRightArm = glm::rotate(modelPenguinRightArm, glm::radians(-wingsAngle), glm::vec3(0.0f, 0.0f, 1.0f));

if (wingsAngle == 110) {
    raiseWings = 0;
}
else if (wingsAngle == 0) {
    raiseWings = 1;
}

if (raiseWings == 0) {
    wingsAngle -= 0.5f;

}
else {
    wingsAngle += 0.5f;
}
```

3.2 Graphics Model

For this project we have used only polygonal graphics modelling. All objects are made up of polygons.



3.3 Data Structures

We created only two structures: for the snowflake object and the bounding box:

```
struct Snowflake {
	glm::vec3 startingPosition;
	glm::vec3 currentPosition;
};
```

```
struct BoundingBox {
    glm::vec3 mins;
    glm::vec3 maxs;
};
```

3.4 Class Hierarchy

Camera

Encapsulates the variables and methods related to the camera, such as the movement of the camera by changing the position (with the keyboard), the front direction (with the movement of the mouse) or the handling of the Zoom variable change, that is used to compute the projection for the scene. We also compute the view matrix of the camera.

Model3D and Mesh

The Model3D class handles the loading and the drawing of each object and its textures. It does so by splitting the object in its meshes and using the Mesh class to handle the representation of the said meshes by using certain data structures for the vertices, indices and textures of each mesh.

Shader

This class handles reading, linking, compiling and loading shader files. It houses all of the functionality related to working with shader files.

4. User Manual

The camera movement is done by the W, A, S and D keys, just like in any video game. The camera front direction is controlled by the movement of the mouse. Using the scroll wheel, we can zoom the scene in and out. The wind speed is increased by the UP key, while the speed is decreased by the DOWN key. We can manipulate the direction of the directional light using the J and L keys. We can move between the rendering modes using the 1 key for the normal rendering mode, the 2 key for enabling Anti Aliasing and the 3 key to switch to wireframe rendering mode. Lastly, we can start the camera animation using the 4 key and stop it with the 5 key.

5. Further Development and Conclusions

The project could be further improved by adding bounding boxes to multiple objects. Spotlight shadows would be another way to enhance the project. There is also room for the addition on of more objects. If the scene was larger there could have also been room for procedural generation of terrain and objects.

6. References

[1] https://free3d.com/3d-model/emperor-penguin-601811.html

https://free3d.com/3d-model/light-pole-32057.html

https://free3d.com/3d-model/snowy-terrain-46334.html

https://free3d.com/3d-model/abandoned-cottage-house-825251.html

https://free3d.com/3d-model/reindeer-v1--219610.html

https://free3d.com/3d-model/christmas-tree-891764.html

[2] https://learnopengl.com/Getting-started/Camera

[3] https://learnopengl.com/Lighting/Light-casters