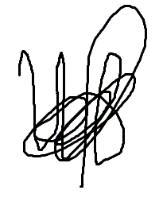
Games Programming 3 Coursework

Ivan Konstantinov

Student ID: S1829087

*I confirm that the code contained in this file (other than that provided or authorized) is all my work and has not been submitted elsewhere in fulfillment of this or any other award*.



Contents

[1. Overview - 2 -](#_Toc93606201)

[2. Collision - 3 -](#_Toc93606202)

[3. Camera - 4 -](#_Toc93606203)

[4. Movement - 5 -](#_Toc93606204)

# Overview

The coursework is to develop 3D asteroid game using OpenGL and C++. The game should contain 3 different models, with textures and shaders.

In the game there is a spaceship that is rendered using OpenGL. It can move using SDL and it can collide with the asteroids and it will play a sounds once it has collided. The movement of the player has been added to the game as extension material. As well as collision detection and camera to follow the player.

# Collision

This paragraph explains the collision detection which is implemented in the game

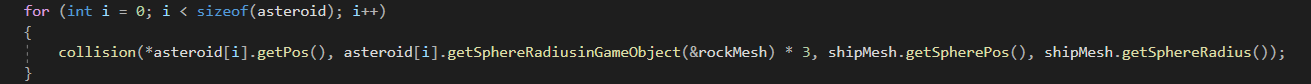


Figure.1.1

The first method that was implemented is for the collision. This function takes the position of each asteroid using for loop, the sphere position of each asteroid multiplied by 3, the position of the ship and the sphere radius.

Text

Description automatically generated

Figure 1.2

The function itself takes the position of both objects as a glm::vec3 and the sphere radius as a float. Then it calculates the distance and checks if the first object has collided with the second. If collision is in place, a sound is being played and the program is printing “collision” for testing purposes only. If there was no collision the function returns false and the program continues onwards.

Text

Description automatically generated

Figure 1.3

For the collision to work, two functions had to be added to the GameObject headerfile. The first one getSphereRadiusinGameObject takes pointer to the mesh and it uses it to get to the getSphereRadius() method which is on the mesh. Then it returns the sphere position that is used to check for the collision. The second function is to retrieve the position of the GameObject.

# Camera

This paragraph is about the movement of the camera. The camera has been created as non existing Game Object witch is in the game just to show to the player what is happening in the game.

Graphical user interface, text, website

Description automatically generated

Figure 2.1

The moveCamera() function was created to set the look at the ship mesh. As well as to update the camera position. The function is responsible for the following effect on the camera. The moveCamera() function is called in drawGame() in each frame, so it updates the camera frequently.

Graphical user interface

Description automatically generated with medium confidence

Figure 2.2

The setLook(glm::vec3 modelPos) function is taken from the lab. It sets the forward of the camera to the normalized vec3 modelPos – the current camera position.

Text

Description automatically generated

Figure 2.3

The setPos(glm::vec3 modelPos) sets the current camera position to the modelPos given in the function. The camera is following the space ship but as the ship moves further the camera is drawing the space ship further away. It needs to be moved using the Upper Arrow back to the ship.

# Movement

In this section the movement mechanic is described.

“W”, “A”, “S”, “D”, “Q”, “E” are used for the movement of the player

The arrow keys are used to control the camera position and rotation.

Text

Description automatically generated

Figure 3.1

For the space ship to move, the program is using transformPositions function, which takes the ship position, rotation and scale. The function updates the ship position by adding a vector3 with the speed that is defined in the MainGame header file.

For the left movement the vector3 used is (speed, 0.0, 0.0) where speed is the x coordinate, 0.0 is the y coordinate and 0.0 is the z coordinate.

For moving up the vector used is (0.0, speed, 0.0).

For moving right the vector used is (-speed, 0.0, 0.0).

For moving down the vector used is (0.0, -speed, 0.0).

For moving further on the z axis in the game it is (0.0, 0.0, speed)

For moving back on the z axis it is (0.0, 0.0, -speed);



Figure 3.2

getPosition() was added in order to get the transform position so it can be used in the movement function.

All the movement functions are multiplied by deltaTime so it can show the same movement on different machines and with different framerate.