Benjamin Lockhart

S2030347

Games Programming 2 (M316206039-22-A-GLAS)

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

*Benjamin Lockhart*

Contents

Main Class Pg. 3

MainGame Class Pg. 3

Mesh Class Pg. 4

SDLAudio Class Pg. 4

Texture Class Pg. 5

Transform Class Pg. 5

Main Class

This is run upon the execution of the code and is simply used to run the mainGame class.

MainGame Class

This is the primary class from which everything else is run. It begins by running the initSystems function which creates instances of the display and the camera, and loads in the 3 models used in the program as well as the background audio. It also initialises a new shader loaded from file.

Next the gameLoop function is run and continues to do so until the program is closed. Each time the function loops it plays the gameAudio track loaded in the initSystems function and then runs 3 functions: processInput, drawGame, collision.

The processInput function creates a variable to store the most recent input event, it then checks the input type against a switch function and executes the appropriate action.

The collision function takes in the position and radius of both mesh 1 and 2. It then finds the distance between the 2 center points of the meshes and if the the distance is less than the combined distance of the 2 radius’ then it writes to the log and returns true, otherwise it will return false.

The drawGame function starts by clearing the contents of the screen, it then sets the position, rotation and scale of the first mesh, the x position follows a sine function which is updated every loop. The 2 textures are then loaded from file before binding and updating the shader to the first mesh. Redrawing the mesh, and updating the data on the sphere position for collision detection. The same is then repeated for the second and third meshes.

Mesh Class

The mesh class begins by finding the position, texcoord and normal values of every vertice within the mesh as assigning it, then assigning the value of every index.

The initModel function begins by assigning a variable with the value of the total number of indices within the mesh. It then generates and binds the vertex array and generates the vertex buffer based on the array data. It then marks what kind of data is held within the buffer and marks it to be stored within the GPU. The same is then repeated for the texture coordinate buffer, the normal buffer, and the index buffer.

The loadModel function stores an instance of the model within the model variable, and runs the initModel function using this reference. It also creates the mesh sphere used for collision detection.

The draw function binds the vertex array and runs the glDrawElements function using a reference of the triangles used and the drawCount variable used to store the size of the indices array.

The updateSphereData function takes in a position and radius variable and set the position and radius of the mesh sphere based on the value of these variables.

SDLAudio Class

The SDLAudio class begins by initialising the auidio, and setting the value of the aurio\_rate, audio\_format, audio\_channels, and audio\_buffers variables. It then checks if the audio could be initialised and throws an error if it couldn’t.

The addSoundEffect function creates an instance of the audio file held within the path specified. It then checks if the variable is null, and if it isn’t it pushes back the instance of the audio.

The addAudioTrack again creates an instance of the background music held within the path specified. It checks if the variable is null and if it is it throws an error.

The playSoundEffect function checks if the value specified can be found within the sound effect bank, throwiung an error if it is out of range. If it is within range the sound effect is played.

The playAudioTrack function checks if there is already music playing, if there is none it plays the background music.

Texture Class

The texture class begins by creating variables for the width, height, and number of components of the texture image. The image is then loaded and stored within the imageData variable which is checked to see if it is null.

The function then binds the texture. The parameters for the texture are then set including, the wrap texture width and height, the linear filtering for both texture magnification and minification, as well as assigning all the texture data.

The Bind function first looks to see if we are working with one of the 32 textures then sets the active texture unit and the type of texture to bind.

Transform Class

The transform class first sets the local value of the position, rotation, and scale.

The GetModel function then sets the value of the material position, scale and rotation as well as the x, y, and z rotations. It then returns the values of the material position, rotation, and scale multiplied together.

The class also contains both getters and setters for the position, rotation, and scale.