James Milne  
S1513160  
Games Programming 2 (M3I622943-17-A)

*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*.



Contents

1: Display Game Class- page 3

2: Camera Class- page 3

3: Mesh Handler Class- page 3/4

4: Model Transform Class- page 4

5: Shader Handler Class- page 4

6: Texture Handler Class- page 4

7: Sound Manager Class- page 4/5

8: Main Class- page 5

9: Game Script Class (includes controls)- page 5

10: Shaders- page 6

11: References- page 7

1: Display Game Class

The game display class is responsible for drawing the game to the screen. The header component of this class creates the constructor, destructor, Change Buffers method, Clear Game Screen method, get Screen Width method, get Screen Height method, and error report method. It also creates variables for the GL context, the SDL window, the screen height, and screen width. The cpp component of this class contains the code for these established methods. The constructor initialises the screen height and width as well as setting up the SDL window as null for use in error reporting. The destructor deletes the GL context, the SDL window and quits the program. There are two getters that return the screen width and height that are declared in this cpp file. The start display method Initialises SDL and its attributes. It then creates the game window and checks that it was created successfully, displaying an error if it was not. The method also does the same for the GL context. Finally, the method creates the background colour. The error report method prints and error to the screen if one occurred then quits the program one a key is pressed. The clear game screen method clears the screen of and re-sets it colour. The Change buffers method swaps the buffers being used to send data to the graphics card.

2: Camera Class

The camera class only consists if a header file. 4 variables are used in this class a mat4 for the projection area, a glm vec3 for the position, one for the direction its facing, and one for what direction is up relative to the camera. The constructor for this class does nothing. The Create Game Camera method initialises the 4 variables used in this class. The Forward Back, Right Left, Pitch and Turn methods all take in a magnitude when called and adjust the cameras position or rotation based on that magnitude. The Get Cameras Projection method calculates the cameras view area using the 3 other variables in the class.

3: Mesh Handler Class

The Mesh Handler class is responsible for creating meshes and sending their data to the shaders. It is split up into 3 separate classes, The Vertex class, the Collision sphere class, and the Mesh Handler class. The Vertex class exists entirely in the header file. It had a variable for position, texture coordinates, and the normal along with 3 methods that return these when called. The constructor takes in a position and set of texture coordinates and sets these variables to the ones passed in. The Collision Sphere class also exists completely inside the header file. It creates a variable for the position and one for the radius of the circle. It also creates 4 methods to get and set the position and radius. The constructor takes a position and sets the position variable to it. In the header file the mesh handler script creates and instance of the collision sphere, an array of vertex buffers, a vertex array object, and enumerated types for position, texture coordinates, normal, and index vertex buffers, along with the number of buffers. The header file also creates the constructor, destructor, Initialise Mesh, draw Mesh, Load Mesh From File, Initialise Model, update Sphere Data, get Sphere Position, and get Sphere Radius methods. The get position and radius methods return the position and radius of the collision sphere. In the cpp file the Initialise Mesh method takes the data for a mesh consisting of positions, texture co-ordinates and normal values and passes it to the Initialise Model method. The initialise model method then binds this this data to the vertex buffers. The constructor sets the draw count to 0, The destructor deletes the vertex buffer arrays and object. The load model from file method takes a file name, loads data from the file and sends it to the initialise model method. It also creates a collision sphere for the model. The draw model method binds the vertex object to the vertex array and draws the mesh to the screen. The update sphere data method updates the collision spheres position and radius.

4: Model Transform Class

The model transform class is responsible for holding position, rotation, and scale data for each model in the game, it consists of only a header file. The class declares variables for position rotation and scale. The model transform method takes in values for position, rotation, and scale and sets them to the corresponding variables. The get model method creates matrices for position rotation and scale, multiplies them together and returns them. The class also contains getters and setters for position, rotation, and scale.

5: Shader Handler Class

The shader handler class is responsible for managing the shaders the program uses. In the header file a variable for the number of shaders is created, a variable for the shader program is created, an array for the shaders is created and an array for the uniforms is created. An enumerated type is also created containing the uniforms and the number of uniforms. In the cpp file the constructor creates the shader program and loads the shaders. It then adds the shaders to the shader program. It then associates the camera position with the shaders and links the shader program to the gpu. The destructor detaches the shaders from the shader program then deletes them. It also deletes the shader program. The update shaders method updates the shaders with the camera position and view area. The shader binding method binds the shaders to the gpu. The shader loading method loads shaders from external files and returns an error if it was unable to do so. The shader error report logs an error if the shader cannot be compiled.

6: Texture Handler Class

The texture handler class is responsible for applying textures to models and binding them to the shaders. In the header file an integer for the index used by the class is created. In the cpp file the constructor gets the dimensions and components of he image used for the texture and saves them. It then wraps and filters the image according to its size in relation to the area it will cover. The destructor deletes the texture. The bind texture method binds the texture to the texture index.

7: Sound Manager Class

The sound manager class is responsible for handling all the programs audio. In the header file a struct is created to hold the sound clips data. A struct is also created to hold the position the sound plays from. An array of sound data sets is created along with an AL context and AL device. In the cpp file the constructor sets the device to the sound card and creates the context. The destructor deletes the data sets, destroys the context, and closes the device. The to large method takes in an integer and returns if it is too big. The change var to int method takes data from the sound file and changes it into an integer. The load sound file method loads a sound frim an external file. The ready clip method makes the sound clip ready to be played. The activate and deactivate clip methods play and stop the sound clips.

8: Main class

All the main class does is create an instance of the game script and calls its run game method.

9: Game Script Class

The game script class is responsible for creating all the objects and running the game loop. This script is the core of the program. In the header file an enumerated class called game state is created. An instance of display screen, game state camera, model transform, and sound manager are created. 4 instances of mesh handler are created, one for each object in the game. Five Booleans are created for controlling the player object and a float for a counter is created. Two unsigned integers are created for the two sound clips and four vectors are created to keep track of each model’s previous position. In the cpp file the constructor sets the game state to playing. It also instantiates the player and bullet objects and creates the game scree. The run game method calls the initialise game systems method and run game loop method. The input processing method take in the keys the user has pressed and performs actions accordingly. When developing this section of the program a forum was used for assistance.(1) The left and right arrow keys set the left and right Booleans to true, space does the same for the space Boolean. The up and down keys call the cameras tilt function, the w and s keys call the cameras forward back function, the a and d keys call the right left function, and the up and down arrow keys call the tilt function. The run game loop method calls the input processing and draw screen methods. It sets the previous positions of the models to their current positions. It also plays the background music. The draw game screen method starts by clearing the screen, it then loads the textures and shaders. It then goes on to apply transforms to the objects. It does this by editing the instance of model transform created in the header then applying it to an object, then applying textures to the object and finally updating the collision sphere data before doing the same for the next object. The player character moves left if the left Boolean is true, right if the right Boolean is true, and stays still if neither is true. The bullet moves to the players position if its y value is above 4 and the space Boolean is true, otherwise it constantly move up. For the two enemies they move to the left until they reach the edge of the screen, then move right and changing again if they reach the right edge. Collision processing is called every frame to check if they come into contact with the bullet, if they do their position is re-set and the explosion sound is played. The method finished by swapping the buffers. The initialise game systems method loads the meshes and sound clips, It also creates the camera and sets the counter and initial positions of all the objects. The play sound method plays the sound passed in at the location passed in. The collision processing method takes the positions and collision spheres of two models and checks the distance between the positions, if its less than the combined length of the two collision sphere radiuses then it returns true, if not it returns false.

10: Shaders

The shaders were used to create a lighting effect on the meshes. To achieve this, an online tutorial was used to assist in the creation of these shaders(2). There was an attempt made at creating a fog effect using shaders however no objects would render to the screen and the issue could not be resolved so was abandoned. The remnants of this attempt can be found in the resources folder labelled as Failed Fog Shader.

11: References

The Model Loader and stb\_image.c classes along with the monkey head model, textures, and sound clips were provided by the module staff.

(1) <http://www.cplusplus.com/forum/windows/182214/>

(2) <https://www.youtube.com/watch?v=NS980twY1ZE&feature=youtu.be&t=23m>