Matthew Cadden

S1508111

Game Programming (Software Development)

Games Programming 2

*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*.

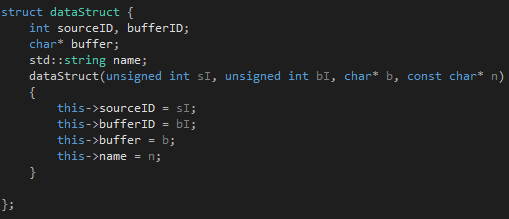
*Signature*. Matthew Cadden

Contents

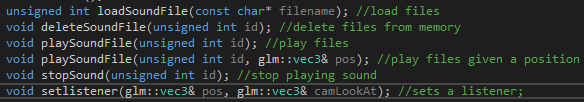
1. Audio
   1. Audio Header File (page 3)
   2. Audio .cpp (page 3 -4)
2. Create Mesh
   1. Create Mesh Header (page 4)
   2. Init Mesh method (page 4)
   3. Init The Model method (page 4)
   4. Load The Model method (page 4)
   5. Draw Mesh method (page 4)
   6. Update Sphere Data method (page 5)
3. Create Shaders
   1. Create Shaders Header (page 5)
   2. Init Shaders method (page 5)
   3. Create Shaders destructor (page 5)
   4. Update Shaders method (page 5)
   5. Make Shaders method (page 6)
   6. Other Methods (page 6)
   7. Issues with Shaders (page 6 - 8)
4. Get Textures
   1. Get Textures Class (page 8)
5. The Camera Structure
   1. Init Camera method (page 8 - 9)
   2. Get View Projection method (page 9)
   3. Camera Controls (page 9)
6. The Display Class
   1. The Display Constructor (page 9)
   2. Initialise Display method (page 9 - 10)
   3. Other Methods (page 10)
7. Transform Maths Structure (page 10 - 11)
8. Main Game Class
   1. Main Game Constructor (page 11)
   2. Run Game method (page 11)
   3. Initialise Systems method (page 11 - 12)
   4. Game Loop method (page 12)
   5. Input Handler method (page 12 - 13)
   6. Play Audio method (page 13)
   7. Collision method (page 14)
   8. Draw To Screen Method (page 14 - 15)
9. Appendix (page 16)
10. Audio

1.1 Audio Header File

Data Struct

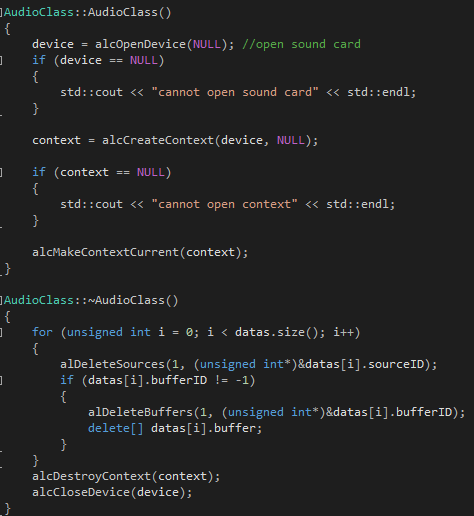


This will store all the data needed to read the music file.



These are the methods that can be called from other scripts to use audio in the game

1.2 Audio .cpp



The constructor for the audio class will attempt to open the soundcard create a context for it. In the destructor the code will clear out all of the buffers and close the soundcard. To load in a sound file it will determine the format of the file and then load in the data the correct way based on which format the audio is in.

1. Create Mesh

2.1 Create Mesh Header

In the header file we set up a class for the vertex which will hold the position, texture coordinates and normal of the mesh. This also contains the struct for the collision sphere and the mesh class itself.

2.2 Init Mesh method

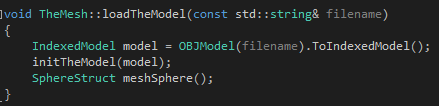
This method takes in a model and will get the position, texture coordinates and normal for every vertex. It will also get the indices of the model and the call the initTheModel method passing in the model.

2.3 Init The Model method



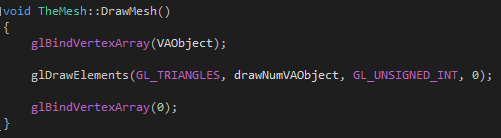
This will get the amount of the Vertex array object that needs to drawn. After this we generate a vertex array object and bind it so that it will be the VAO that any operations take place on. After this the positions, texture coords, normal and indices are passed in using the Vertex array buffers. Finally the VAO is unbound.

2.4 Load The Model method



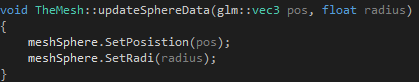
The model will be loaded by its filename and then passed into the method describes in section 2.3, a member of the SphereStruct is also created called meshSphere.

2.5 Draw Mesh method



This method will bind the VAO and then begin to draw out the mesh.

2.6 Update Sphere Data method



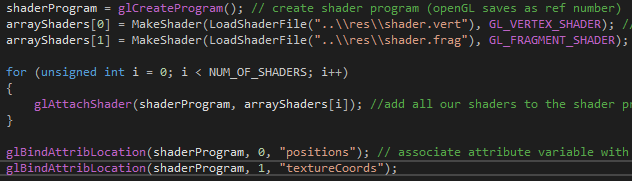
When this method is called it will update the collision sphere around the mesh so that is stays centred as the model is moved.

1. Create Shaders

3.1 Create Shaders Header

This method will also include the TransformMaths class and the camera class. In the shaders class a shader program is made and an array of the shaders.

3.2 Init Shaders method



This method will create the shader program and then load in shaders from file and put them into an array of shaders. These shaders are then attached to the shader program. Attribute locations are bound to the shader program for use with the shader files that were loaded in.

After this the shader program is linked so that it will be able to run on the GPU, then the program is validated. This method will also get the unifroms from the shader associated with the name given.

3.3 Create Shaders destructor

This will detach all of the shaders from the shader program and then delete the shaders. After this is done the shader program is also deleted.

3.4 Update Shaders method

This method will recalculate the Model View Projection matrix by multiplying the view projection of the camera by the model transform. It will also update the uniform matrix.

3.5 Make Shader method

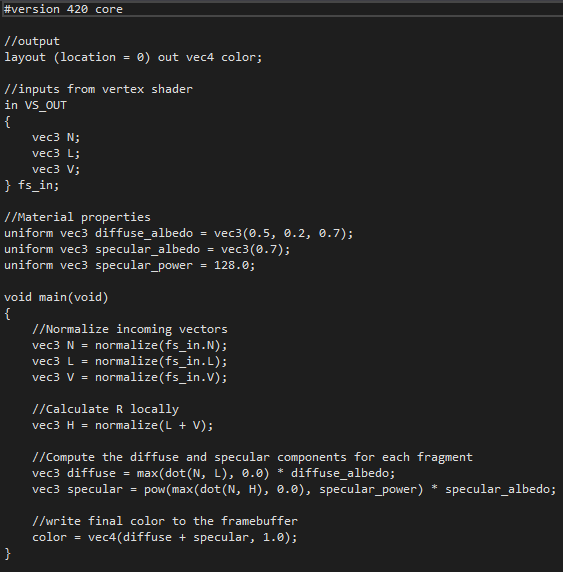
This method will take in a string and an unsigned int as parameters. It will then create a shader using the unsigned int to get the type of shader. The method will then convert the string into a list of c-strings. After this the shader code is sent to openGl and openGL is then used to compile the shader code.

3.6 Other Methods

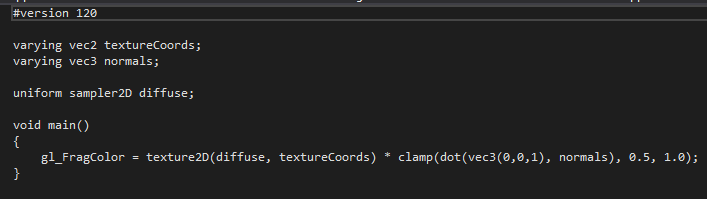
Other methods in this class are used to load in the shader files given a filename. Another method checks if there are any errors with the shader and is called from time to time. There is also a method to bind shaders which can be used to bind to the shader program.

3.7 Issues with shaders

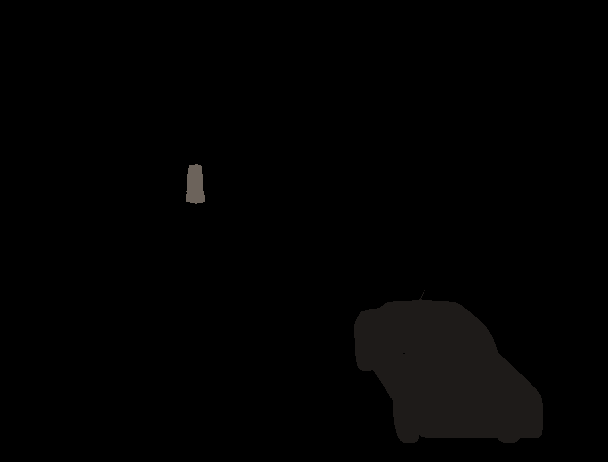
Many issues were encountered with shaders and these became unable to be used.



The Blinn-Phong Lighting was attempted but could not be made to correctly work a snippet of this can be seen. Fog was also attempted but again the fog shader could not be made to work. Another type of shader attempted was a simple lighting shader.



This is the fragment shader of the lighting shader. When this shader was attempted the results ranged from a black screen to unrecognisable textures as can be seen below.



For this reason, it was decided to not use this shader.

4.0 Get Textures

4.1 Get Textures class

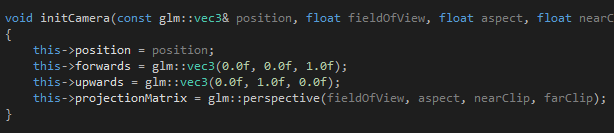


The initTextures method will load the image and get the width, height and number of components for the image. It will then bind the texture and define the type. It will then set the texture up for use. The destructor will delete all textures and the address. There is also a bind textures method which is called to bind textures to meshes.

5.0 The Camera Structure

5.1 Init Camera method

The header contains the initCamera method will set up the camera.

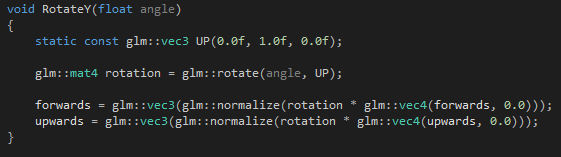


This will set the position to be whatever is passed in as well as setting up forwards and up. The projection matrix is also set up here using values that are passed into the method.

5.2 Get View Projection method

This method will just return the view projection matrix which is done by multiplying the projection matrix with the cameras view.

5.3 Camera controls

The camera controls are handled in a few different methods. The ones used in this coursework are the MoveForward method, MoveRIght method and the RotateY method. The MoveForward method just zooms the camera in or out while the MoveRight method can be used to move the camera to the right or left. Both these methods require a float value to be passed in so that they are given an amount to move by. The RotateY method will take in a float for an angle value and then rotate on the y-axis.  A method is also used in here to return the cameras position.

6.0 The Display Class

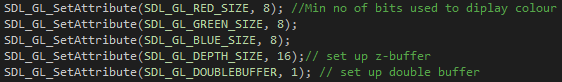
6.1 The Display Constructor

The constructor will set the window to be a null pointer and set the width and height of the screen.

6.2 Initialise Display method



This will initialise everything that we need for the display.



This will set the attributes so that 8 bits are used for every colour. The double buffer is used so that there are 2 windows constantly being swapped. These are the window the player sees and the window currently being drawn to and these are swapped every game frame. This is done so that the player cannot see the drawing take place.

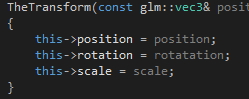


A window is then created using the width and height specified and is given a name. After checking for errors we use  to make sure we don’t draw the faces which the camera cannot see.

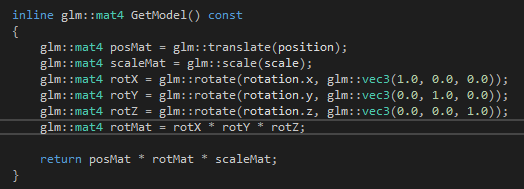
6.3 other methods

In this script there are getters which will return the width or height of the screen. The destructor will delete the openGL context, destroy the window and then quit the application. The swapDisplayedScreen method is used to swap between the current screen being displayed and the next screen which is the next frame that has finished drawing. There is also a method that can be used to clear the display.

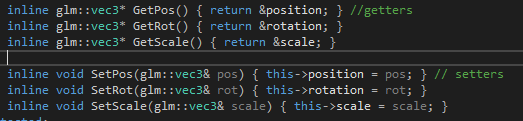
7.0 Transform Maths Structure



Stores the position, rotation and scale.



This will return the models transform by using the position matrix, scale matrix and rotation matrix.

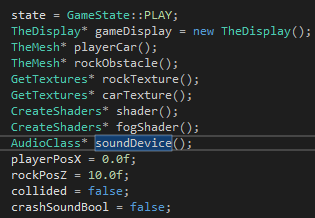


There is also getters and setters for position, rotation and scale so that you can call or set one of these manually at any time.

8.0 Main Game Class

8.1 Main Game Constructor

This will set up some of the variables we need to run the game. It will also set the GameState to play.



The collided bool is switched to true when there has been a collision, the crash sound bool is used so that the crash sound will only be played once.

8.2 Run Game method

This method will just call the initialiseSystems method and the gameLoop method.

8.3 Initialise Systems method

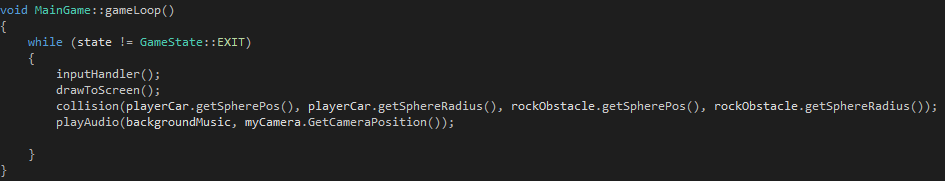
This method is used to load in all the resources we will need in the game along with initialising the game display and camera.



The counter here is set at 0 as this is what we use to change the rocks position in the z axis.

8.4 Game Loop method

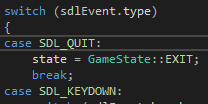
This method will run as long as the GameState does not change to EXIT.



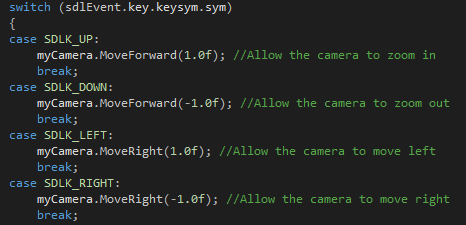
Here it will check for inputs from user then draw to the screen. It will also check for collisions between the car and the rock. While this is happening it will also play the background music.

8.5 Input Handler method

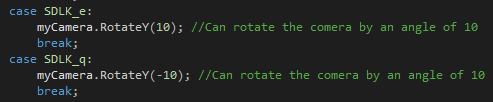
This method makes use of an SDL event to check for user input.



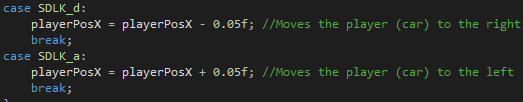
This checks what type of event has happened, if it was a quit event the application will exit otherwise if it is a keydown event then it will go on to check for which key was pressed.



These are the four keys which control the camera’s movement and are tied to the arrow keys on the keyboard. This allows the camera to zoom in and out and move right or left depending on the key pressed.

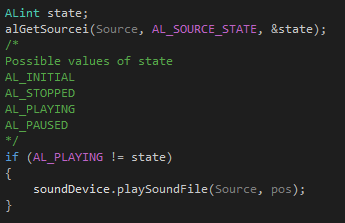


These keys control the cameras rotation in the y-axis and is set to a fixed amount of 10.



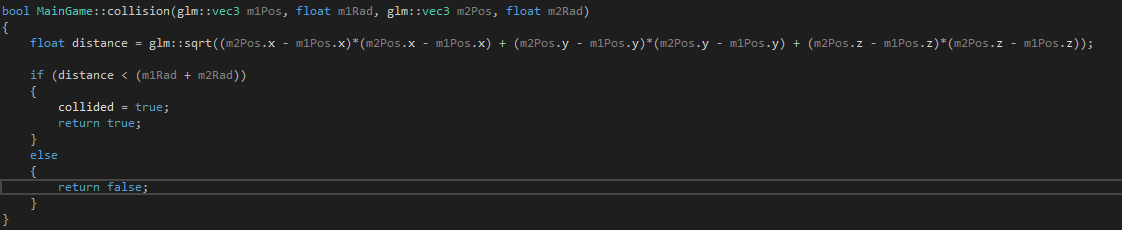
This is how player movement is handled with the player being able to move left or right by a fixed amount of 0.05 each key press.

8.6 Play Audio method



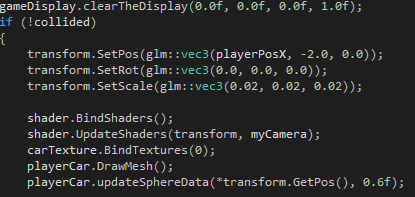
This method will take in a source and position so you can provide it with an audio file to play that was loaded during section 8.3 and then a position to play at.

8.7 Collision method

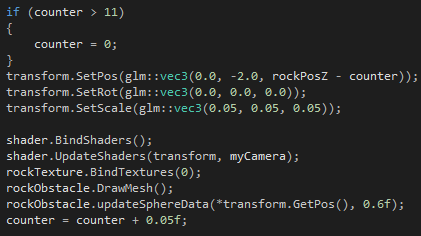


The collision method will calculate by using the positions of the sphere collision meshes in the x, y and z. After this it is checked if the distance between the meshes is less than both of their radius together as if this is true then there is a collision. When this happens the collided bool is set to true so that we can change the draw method.

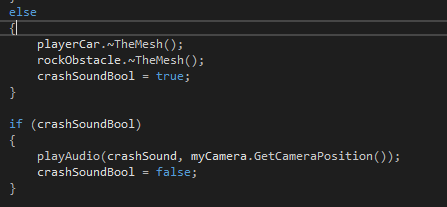
8.8 Draw To Screen method



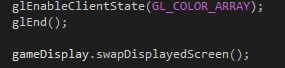
The player is only drawn if there has been no collision detected. The position is set and can be changed in the x-axis by using playerPosX. After this the shaders are bound and updated then we bind the correct texture for the model. After this the model is drawn and the collision sphere data is updated.



This is how the rock is set up and moves. It will move in the z-axis by constantly taking away from the counter after this goes outside the screen the counter is reset so the rock will spawn back at its original position and then continue moving towards the player. It is drawn in the same way as the player but with a different texture bound to it.



If there has been a collision both meshes are destroyed and the audio for a crash is played.



This is also where we swap the screen so we can constantly draw to the one screen that the player cannot see and then swap that for the one they were currently looking at.

1. Appendix

Car Texture - <https://img00.deviantart.net/1dc0/i/2013/334/3/c/car_paint_texture_by_doodlee_a-d4eqf2g.jpg>

Stone obstacle and texture - <https://www.turbosquid.com/FullPreview/Index.cfm/ID/787472>

Car model - <https://www.turbosquid.com/FullPreview/Index.cfm/ID/394290>

Tutorials used - https://www.youtube.com/user/thebennybox