2019/2020

*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*.

*Marco McAlindin*.

Games Programming 2

TECHniCAL DOCUMENT

Marco Mcalindin S1718202

Contents

[**1.** **Application entry point (main())** 2](#_Toc29609333)

[**2.** **Game class** 3](#_Toc29609334)

[2.1. Game.h 3](#_Toc29609335)

[2.1.1. Public 3](#_Toc29609336)

[2.1.2. Private 3](#_Toc29609337)

[2.2. Game.cpp 3](#_Toc29609338)

[2.2.1. Void Game() 3](#_Toc29609339)

[2.2.2. Void Run() 4](#_Toc29609340)

[2.2.3. Void InitializeSystems() 4](#_Toc29609341)

[2.2.4. Void MainGameLoop() 4](#_Toc29609342)

[2.2.5. Void Input() 4](#_Toc29609343)

[2.2.6. Void DrawGame() 5](#_Toc29609344)

[2.2.7. Bool CollisionDetection(vec3,float,vec3,float) 5](#_Toc29609345)

[2.2.8. Void AudioPlay(unsigned int, vec3) 5](#_Toc29609346)

[**3.** **GameDisplay class** 6](#_Toc29609347)

[3.1. GameDisplay.h 6](#_Toc29609348)

[3.1.1. Public 6](#_Toc29609349)

[3.1.2. Private 6](#_Toc29609350)

[3.2. GameDisplay.cpp 6](#_Toc29609351)

[3.2.1. Void InitializeGameDisplay() 6](#_Toc29609352)

[3.2.2. Void swapBuffer() 7](#_Toc29609353)

[3.2.3. Void ResetDisplay(float,float,float,float) 7](#_Toc29609354)

[**4.** **Texture class** 7](#_Toc29609355)

[4.1. Texture.h 7](#_Toc29609356)

[4.1.1. Public 7](#_Toc29609357)

[4.1.2. Private 7](#_Toc29609358)

[4.2. Texture.cpp 7](#_Toc29609359)

[4.2.1. Void Texture() 7](#_Toc29609360)

[4.2.2. Void Bind() 8](#_Toc29609361)

[4.2.3. Void ¬Texture() 8](#_Toc29609362)

[**5.** **Shader class** 8](#_Toc29609363)

[5.1. Shader.h 8](#_Toc29609364)

[5.1.1. Public 8](#_Toc29609365)

[5.1.2. Private 9](#_Toc29609366)

[5.2. Shader.cpp 9](#_Toc29609367)

[5.2.1. Void Shader(const string) 9](#_Toc29609368)

[5.2.2. GLuint CreateShader(const string, unsigned int) 9](#_Toc29609369)

[5.2.3. String LoadShader() 9](#_Toc29609370)

[5.2.4. Void Update() 9](#_Toc29609371)

[5.2.5. Void ¬Shader() 10](#_Toc29609372)

[**6.** **Mesh class** 10](#_Toc29609373)

[6.1. Mesh.h 10](#_Toc29609374)

[6.1.1. Public 10](#_Toc29609375)

[6.1.2. Private 10](#_Toc29609376)

[6.2. Mesh.cpp 10](#_Toc29609377)

[6.2.1. Void InitializeMesh(Vertex, unsigned int, unsigned int\*, unsigned int) 10](#_Toc29609378)

[6.2.2. Void InitializeModel(const IndexModel&) 10](#_Toc29609379)

[6.2.3. Void LoadModel(const string) 10](#_Toc29609380)

[**7.** **Vertex class** 11](#_Toc29609381)

[**8.** **Sphere class** 11](#_Toc29609382)

[**9.** **Camera.h** 11](#_Toc29609383)

[9.1. Public 11](#_Toc29609384)

[**10.** **Transform.h** 11](#_Toc29609385)

[**11.** **SkyBox class** 12](#_Toc29609386)

[**12.** **References** 12](#_Toc29609387)

# **Application entry point (main())**

The first script encountered in our program is the applications entry point which is a class that returns an integer value. The class will return 0 when the application has to quit.

In this class our variables my game is declared and with it we execute all the code that is present in the Game class constructor. Once the constructors code is executed we call the Run method of the Game class that will start running our game.

# **Game class**

## Game.h

In our header file we declare classes and variables that will be used in our class.

### Public

Our public methods in this case are three, the constructor, destructor and Run.

### Private

#### Methods()

We have various private methods. InitializeSystems() where we will load and initialize the base systems needed for our game. Input() method will process any input involved in the game. MainGameLoop() will run our game based on the state our game is in and exit appropriately. DrawGame() that will take care of how are game is drawn onto the screen (position, rotation and scale of our in game objects). AudioPlay() mnages the audio in the scene taking in two parameters an unsigned int for the source index and a vector3 for the spatial position of the sound. CollisionDetection() by taking in four parameters, the two positions of our collision spheres and the two desired radiuses of our collision spheres.

#### Variables

In our variables in this class we hold the most important variables for the front end of the game. There is a GameDisplay variable that will take care of what is shown on our screen, a mainGameState variable that holds the state our game is currently in, the audioDevice that controls the audio in the game, the mainCamera of the game and the mesh variables for our 3D models and some float variables needed to control these models.

## Game.cpp

### Void Game()

When we create our Game object the first code to execute is the constructor, in the constructor of this class we assign the mainGameState variable to PLAY, create a GameDisplay object and assign it to appropriate variable, which will execute its constructor code (Code will be explained in future paragraph), and then we assign the pointer variable of the audio device that will execute the code in the constructor of the GameAduio class (Code will be explained in future paragraph).

### Void Run()

This method only contains two lines of code which are two method calls, InitializeSystems() to initialize our base systems for our game, for example loading our 3D models and the second method being the game loop, MainGameLoop() which runs until our game state changes to QUIT.

### Void InitializeSystems()

Using the method InitializeGameDisplay() of our gameDisplay variable we initialize our game display, we also use the InitializeCamera() method to set our camera values.

After initializing our display and camera we load our models and our music for the game (As seen in the

A screen shot of a social media post

Description automatically generatedimage below) using the appropriate methods described in the appropriate future paragraphs.

Model and Sound loading

### Void MainGameLoop()

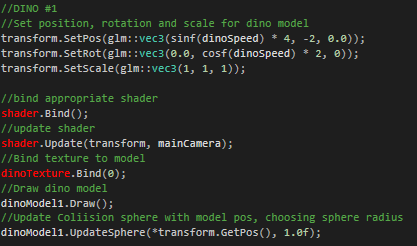
In our game loop we check for input with our Input() method, draw our game with DrawGame(), check if our wanted models collide using CollisionDetection() passing in the parameters of the calculated collision sphere of our meshes and then we play our background music for the scene which in this case is the helicopter sound of the drones in the scene.

There are only two sounds in the scene, which are the background helicopter sound mentioned above and a dinosaur roar sound that plays when the two dinosaur models collide.

### Void Input()

For input detection we create a SDL\_Event and using a switch statement check what button is pressed and act upon. In this application there is only one button taken in consideration which is the quit button to quit the application.

### Void DrawGame()

In this method we draw our game starting with clearing the screen setting the rgb values to zero then we load the textures needed for our models and the shader. After loading textures and shaders we set positions, rotation and scale for every object we want in our scene. In this case we use sine and cosine for appropriate oscillating motions. After setting our transform values for the mesh we bind and update the shader, bind the texture to the model and finally draw and update our sphere collision position.

Code written for every 3d model in game

We repeat these lines of code for every mesh in our scene assigning wanted values for the variables.

At the end of the function we swap the buffers to actually display to screen the code that has just been executed.

### Bool CollisionDetection(vec3,float,vec3,float)

We assign to a distance float variable the distance between two points using Pythagoras formula to find the hypotenuse using vectors passed in as parameters. Once calculated the distance we check it is less than the sum of the two radiuses, if it is return

### Void AudioPlay(unsigned int, vec3)

Check that our sound is not already playing and we play it using the appropriate sound from the index passed as a parameter at the vec3 parameter position.

# **GameDisplay class**

## GameDisplay.h

In our header file we declare classes and variables that will be used in our class.

### Public

#### Methods()

Aside from our constructor and our destructor we have three void methods and two float methods, also known as getters.

#### Variables

We only have one public variable which is our SDL\_GL context needed to display our scene on screen.

### Private

#### Methods()

We have one void variable to check if some of the SDL\_GL systems do not load properly in order to print an error message, which is our string parameter, and exit our application.

#### Variables

There are two float variables to set the application screen width and height and also have a pointer variable to hold our game window.

## GameDisplay.cpp

### Void InitializeGameDisplay()

In this method we start setting the SDL attributes needed to display our game reserving the number of bits for each primary color(rgb) and setting our double buffer.

We continue by assigning to our window variable pointer an SDL window that we create using the appropriate SDL method which takes in six parameters, our window text, the location on the devices screen to initialize the window, the size of the window and a window that can be used by our OpenGL context.

Next the SDL context is created using the window previously created and assigned to our glcontext variable.

We initiate glew.

For the last three steps, every time we create we use the ReturnError(string) to check if our component are created successful and if not our error method will exit the application.

A screenshot of a cell phone

Description automatically generated

Initialize Game Display code

### Void swapBuffer()

A screen shot of a person

Description automatically generatedSwap the buffer using the SDL\_GL\_SwapWindow(SDL\_window) because we need to continuously swap between two buffers in order to not have code draw directly on the screen that is displayed by the user.

Swap buffer method

### Void ResetDisplay(float,float,float,float)

We clear the screen display colors and clear our buffer using the methods glClearColor() and glClear().

# **Texture class**

## Texture.h

### Public

#### Methods()

We have two public methods, the constructor which takes in a string parameter with the name of the texture file name, and the Bind() method to attach our texture taking in an unsigned int for the unit to attach the texture to.

### Private

#### Variables

GLuint texture handler.

## Texture.cpp

### Void Texture()

We start by declaring the width height and number of image components followed by declaring and assigning an image data variable using a method contained in the stb\_image.h script supplied by an external source. Once we have our image data we check it is not null, if it is then we print a texture load error.

A screenshot of a cell phone

Description automatically generatedFollowing we use method from the GL library to generate the texture, bind the texture defining its type, wrapping its dimensions and other necessary actions to initialize a texture.

Generating texture

### Void Bind()

In order to bind a texture we use methods from the GL library to check if we are working with the appropriate texture activate the texture unit.

A picture containing indoor, wall

Description automatically generated

Binding texture to model

### Void ¬Texture()

In the texture destructor we destroy the texture, because a texture has to be deleted when not needed in order to avoid memory leaks and over usage. The glDeleteTextures() method is used.

# **Shader class**

## Shader.h

### Public

#### Methods()

There are seven public methods in this class, two being the constructor and destructor. The other methods have the function of loading, creating and updating shaders and one method that checks for errors.

### Private

#### Variables

We have variables for the shader program, an array containing the shaders, an unsigned int for the number of shaders and an array containing our shader uniforms.

## Shader.cpp

### Void Shader(const string)

We start by creating the shader program using method from the GL library and then create the shader and loading it using our public methods CreateShader() and LoadShader().

Then we have a loop to attach all of the shaders to the shader program. Next we bind the attribute locations of the shaders to the program and then link and validate the shaders to the program, checking for errors with our CheckShadeError() method. A screen shot of a computer

Description automatically generatedFinishing off with getting the uniform location of the program.

Code to link shader to program

### GLuint CreateShader(const string, unsigned int)

To create a shader we have an unsigned int parameter to identify the type of shader that we use with the glCreateShader method. Next we check if the shader has been created. Then we need to convert the strings of the shader into a list of c-strings and then send and compile the code before checking for an error using the CheckShaderError(). If no error occurs our method will return the loaded shader.

### String LoadShader()

To load a shader we need to open a stream to read the shader file and read the GLSL code and return the string containing the appropriate data in order for the CreateShader() method to create the loaded shader.

### Void Update()

The update method uses a matrix formed by the multiplication of the transform of a model and the view projection of the scene camera to run the glUniformMatrix4fv() method. A close up of a screen

Description automatically generated

Shader update method

### Void ¬Shader()

There is a for loop that deletes all the shaders in our program in order to avoid memory leaks and over usage of memory.

# **Mesh class**

## Mesh.h

### Public

Methods()

There are seven public methods in this class including the constructor, destructor and two getters that use the Sphere class methods to get the position and radius of the sphere used to update the sphere using the UpdateSphere() for our collision detection. We also have methods to initialize the mesh and then methods to load a model, initialize it and draw it to the screen.

### Private

#### Variables

We have four variables, two for our VAO(VertexArrayObject) and VBO(VertexArrayBuffers[]) and then an int for the draw count and a Sphere for collision.

## Mesh.cpp

### Void InitializeMesh(Vertex, unsigned int, unsigned int\*, unsigned int)

To initialize a mesh we start by placing in vectors of a IndexModel variable the vertices for our position and normal values and indices. In the end we call the Initialize model method.

### Void InitializeModel(const IndexModel&)

We start by assigning the number of indices to the draw count variable. Then we generate and bind the vertex array object. Then using the appropriate methods from the GL library we generate buffers and tell opengl what type of data it is and then send it to the GPU by using the positions normals and indices.

### Void LoadModel(const string)

The IndexModel variable is assigned by getting the file name of the obj file and is used to call the InitializeModel(IndexModel). We then create the sphere object.

* + 1. Void Draw()

A screenshot of a cell phone

Description automatically generatedMethods from the GL library are used to draw our model to the screen using the VAO.

Code used to draw mesh onto screen

* + 1. UpdateSphere(vec3, float)

In this method we set the sphere pos using the models current position and setting the radius length.

# **Vertex class**

The vertex class is used to place a vertex in our 3D game. The methods involved in this class are the constructor to assign position, coordinates and vertex normal. We also have getters to be able to use the data at any time in our scripts.

# **Sphere class**

The sphere class is intended to create a sphere using a radius and a vector3 variable in order to use it as a collider for our game objects. This class has getters and setters for our position and radius variables.

# **Camera.h**

### Public

#### Methods()

Initialize camera takes in parameters for the field of view, the position, aspect ratio, near clip and far clip of the game camera. The fov, aspect, near clip and far clip compose the projection. Then we have a forward variable and a up variable used in the MoveForward() and MoveUp() methods to move the camera of one unit in the appropriate direction. There are also another two methods, one that rotates using the same process as the movement but using a matrix mat4 and an angle parameter.

#### Variables

#### Private

The private variable consist of the projection, position and the forward and up values.

# **Transform.h**

In this class we have three private variables position, rotation an scale of a game object. With setter and getter methods these variables can be used. There is also a GetModel() method that returns a spatial matrix of an appropriate model.

This class is used to manipulate position, rotation and scale of an object in our 3D game world.

# **SkyBox class**

Disclaimer: The skybox class does not work and is an attempt to an extra feature of the project that could not be completed by lack of time. The code is taken from learn opengl website.

So far the code in the class would set arrays for the cube map and skybox coordinates then uses such coordinates and GL library methods to set up the VAO and VBO buffers.

# **References**

Ground Obj:

<https://free3d.com/3d-model/-rectangular-grass-patch--205749.html>

Dinosaur Obj:

<https://free3d.com/3d-model/deinonychus-70337.html>

Skybox Class, skybox shader and cubemap shader:

<https://learnopengl.com/Advanced-OpenGL/Cubemaps>

Drone Obj:

<https://free3d.com/3d-model/drone-66571.html>

Dinosaur Texture:

<https://www.google.com/search?q=dinosaur+texture&rlz=1C1CHBF_en-GBGB814GB814&sxsrf=ACYBGNSEz5HPthmbttM8GR2SVINp0suNpA:1578715729614&source=lnms&tbm=isch&sa=X&ved=2ahUKEwi7gI-y1vrmAhUHiFwKHfliD-cQ_AUoAXoECA4QAw&biw=1920&bih=920#imgrc=cizBJg2H2W5m4M:>

Ground Texture:

<https://www.google.com/search?q=rock+ground+texture&rlz=1C1CHBF_en-GBGB814GB814&sxsrf=ACYBGNScohmcLfK3UBpoKZ92h2_rgSW0AA:1578715772327&source=lnms&tbm=isch&sa=X&ved=2ahUKEwi-9L3G1vrmAhUmQ0EAHY1hCtAQ_AUoAXoECA0QAw&biw=1920&bih=920#imgrc=lkH9wfnHbov9zM:>

Drone Texture:

<https://www.google.com/search?q=metal+texture&rlz=1C1CHBF_en-GBGB814GB814&sxsrf=ACYBGNTpJgV5uw64nRSvZh_TyNMfugiepw:1578715824123&source=lnms&tbm=isch&sa=X&ved=2ahUKEwi3npff1vrmAhVMi1wKHdyHBVUQ_AUoAXoECA0QAw&biw=1920&bih=920#imgrc=Un2RaeJ72zX7WM:>