Graphics Programming

Jamie Milne

S1513160

Computer Games (Software Development)

*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*.

Contents

[MainScene Class 3](#_Toc514161451)

[ModelTransform Class 3](#_Toc514161452)

[SceneCamera Class 3](#_Toc514161453)

[ObjectMesh Class 3](#_Toc514161454)

[SceneDisplay Class 4](#_Toc514161455)

[ShaderClass 4](#_Toc514161456)

[Shaders 4](#_Toc514161457)

`

# MainScene Class

In the header file of this class variables are established for the 5 models and the 5 shaders used in the scene. Variables for the timer and sceneState are also created, along with instances of the sceneDisplay and scene camera scripts. In the cpp file for this class, the constructor sets the games state to active, creates a new display, and creates objects for the models and shaders. The runProgram method is called from main.cpp and calls the initialise function and then the main loop function. The initialise function calls the initialise function in the sceneDisplay and the createCamera function in the sceneCamera. It also loads the models and shaders from their external files by calling their respective load functions. The timer is also initialised in this method by setting it to 1.

The mainLoop function contains a while loop that runs until the sceneState becomes Close. Within this loop, the QuitFunction and drawScene methods are called. This timer is also incremented by 0.01. The QuitFunction monitors an SDL event type, if it becomes SDL\_QUIT it sets the sceneState to Close, which in turn ends the mainLoop function. The drawScene function is responsible for drawing the game to the screen. It first clears the screen of what was previously on it. It then calls the modelDraw method and the drawModel methods for each model, passing in a different index for each model to the modelDraw method. The buffers are then switched. The drawModel method updates and binds different shaders depending on what index it receives, it also calls functions for each shader to pass variables such as the light direction and view and projection matrices. It also sets the transform position, rotation, and scale.

# ModelTransform Class

This class consists of only a header file. The file contains variables for the models’ position, rotation, and scale. Constructor takes in arguments for the position and rotation while setting the scale 2 1 and then applies these arguments to the respective variables. The GetModel function gets the position, scale and rotation matrices, multiplies them, then returns them. The class also contains getters and setters for position, rotation, and scale.

# SceneCamera Class

This class also consists of only a header file. It contains variables for the Cameras projection, position, forward, and up values. The class also contains getters for the camera view, projection, position, and view projection. The createCamera method takes arguments for the position, field of view, aspect ratio, near clipping plane, and the far clipping plane. It then uses these to set the position, forward, up and projection of the camera.

# ObjectMesh Class

This header file contains both the ObjectVertex struct and the ObjectMesh class. The ObjectVertex contains variables for position, texture co-ordinates, and normal. It also contains getters for these variables. This constructor takes in arguments for position and texture co-ordinates and applies them to the respective variables. The ObjectMesh contains variables for the vertex array object, the array of vertex buffers, and how much of the vertex array object needs to be drawn. There are also enumerated types for the position in the vertex buffer, the texture co-ordinates, normal, and index in the vertex buffer, and number of buffers. The initialise method takes the vertices of a model and takes their positions, texture co-ordinates, and normals and stores them as a mesh. It them calls the initaliseModel method. The initialiseModel method takes the data of a model. It generates buffers and uses them to pass the data from the model over to the GPU for use in the shaders. The loadModelFromFile method takes an external file and passes it to the initialiseModel method. The destructor deletes the vertexArrayObject. The drawModel method draws the model to the screen.

# SceneDisplay Class

In the header file variables are declared for the screen width, screen height, GL context, and SDL window. In the constructor the sdl window is given a null pointer for debugging purposes and the screen width and height are set. In the createDisplay method SDL is initialised and the sdl window and gl context are created, if wither of them fail an error message is displayed. Z- buffering and backface culling are enabled then the screen is cleared. In the destructor the sdl window and gl context are destroyed, then the SDL quit function is called. The class contains getters for the screen width and height. The changeBuffers and clearScreenDisplay methods switch the buffers and clear the screen respectively.

# ShaderClass

The header file of this class contains uniform functions. These take a variable and apss them to the shader. The types that have functions are bool, int, float, vec2, vec3, vec4, mat2, mat3, and mat4. The vector data types have overloaded methods to allow for both regular and glm vectors to be sent. The header file also contains variables for the number of shaders, the shader program, the array of shaders, and the array of uniforms. There are also enumerated types for the number of uniforms and the uniform transforms. The initaliseShader function takes a vertex shader file and a fragment shader file. It calls the CreateShader method using these files. It then adds these shaders to the shader program. The vertex position, texture co-ordinates, and normal are bound as attribute variables. The shader program is linked to the GPU. The update method updates the camera position and uniforms. The BindShader method binds the shader to the GPU. The LoadShaderFromFile method takes an external shader file and returns its contents as a string. The CreateShader method creates and compiles shaders based on the name and typed passed in.

# Shaders

There are 5 shaders used in the program. One is Rim Lighting. This shader simulates a light source coming from behind an object. It does this by taking the dot product of the surface normal and the view direction and using the inverse to set the lighting value. Another Shader used is Toon Shading. This shader uses the dot product between the surface normal and the light direction to choose a shade of a solid colour depending on the value of the dot product. The Toon Rim Shader combines these effects together by multiplying their outputs. The Toon Rim Fog shader adds a fog effect on top of these effects. It does this by applying a fog factor to the model based on how close or far it is to the camera. This value is then combined with the toon rim output to create the final value. The last shader used is the blob shader. This shader creates a spot light effect on the model. It does this by the inner and outer colour values and, using the mix function, uses the inner and outer radiuses to interpolate the lighting value over an area on the model.