John Reid S1511280

Computer Games Development  [Company address]

Games Programming 2 Document

# Camera

The code in the camera header file sets up everything needed to make a 3d camera. The initialise camera method actually creates the camera. It takes in five variables: a vector 3 for the position of the camera and four floats that are used for the field of view, aspect ratio, the near clipping plane and the far clipping planes. The method itself simply sets the variables in the header file to each of the variables just discussed.

# Game Display

The Game Display header and cpp files are used to create the window that the game will be run on. The .cpp file has both a constructor and deconstructor. The constructor sets the sdl\_window to null and sets the height and width variables to something which are the dimensions for the window that will be created. The deconstructor wipes all the context of the window, deletes the window and closes down the program.

GainWidth() and GainHeight() are used to return the width and height set in the previous method.

The ChangeBuffer() method simply changes the buffer being used for the window. WipeDisplay() is used to clear the window display. This is used every frame as it wipes the screen and updates it with the new position of the game objects etc.

InitialiseDisplay() is what is used to actually create the window. Firstly, it initialises the display and sets the colour and depth attributes. It then creates the window once this has been done and checks that there have been no errors and everything has been initialised. Z buffering and back-face culling is then enabled.

# GameObjects

The gameobject header and .cpp files are used to create meshes/game objects that are used in the game. The Initialise method is used to create an array of all the information required for the model. E.g. the position, the texture coordinates. Initialise model is then called once this has been completed.

The initialiseModel() method is what creates the model that is been requested. The method first creates a vertex array which is then bound to the game object. buffers are then created based on the array data which is then bound to the buffer so it can be moved to the GPU. OpenGL is then told what attributes the object holds and how to read the data attached.

CreateModel() is used to get access to the model needed which is then initialised using the initialise model method. A sphere is also attached to it at this point which will be used for collision detection later.

The draw() method takes the arrays of all the information previously created and draws them to the display so the model can be seen by the user.

The attach sphere collider method() takes the sphere previously attached and sets its position and radius so it can appropriately act as a collider.

# Textures

The GamObjectTexture header and cpp files are what is used to texture the models that were created using the gamobject files. The gameobjectTexture method starts by setting three variables that are used to get the width, height and number of components in the texture. It then loads the image and stores the data in imageData. Once it has checked that the data has been loaded it then binds the texture and wraps the texture using the width and height assigned earlier. Once this has been done it is filtered to either minify or magnify the texture depending on what its been textured to.

The Bind() method checks that one of the 32 possible textures that can be used is being used and then sets the texture and binds it to the model.

# Shader

The GameObjectShader method is what is used to initialise the shaders that will be used. It starts by creating the shader program and creating the vertex and fragment shaders. Once this has been done it adds all shaders created to the shader program. The method then sets the vector3 attributes to be equal to the position and texture coordinates. Once all this has been done the program that the GPU will run is created and the whole thing is checked for any errors. The deconstructor method detaches the shader, deletes the shaders and then deletes the shader program. The attach method is used to initiate the shader program. The Update method is used to update the models on the screen. It does this by recalculating the mvp matrix with the new position of the model and the texture of the model.

The CreateGameobjectShader method does as the name implies, it creates the shaders that the game will use. It begins by creating a shader based on the specified type passed into it. It then checks to ensure that the shader has been created. Once it is ensured that the shader has been created it converts the text passed into the method into a list of c\_strings which will allow the specifications of the shader to be read. Once the list has been created the shader is then complied and checked to ensure that no errors have occurred.

The loadGameobjectshader method does something similar except instead of creating the shader it loads the shader from a file. Firstly it gets the file name of the shader file and opens it, converting it to c\_string once it’s open. The method then checks to see if the file has been open and if it has enters a while loop that essentially says while the file is open use it. If the file has not been opened it returns an error.

# Sound

The Sound header and cpp files are what is used to create audio in the game. The Sound() method essentially opens the sound card, checks that the sound card has been opened and sets it so the sound card is the default thing that will be used when dealing with audio.

The determineEdian method determines if the computer being used uses big or little Endian so the sound can be played properly on every computer.

The ChangetoInt method is used to convert the character array to and int array as the wav files are stored as char arrays and they need to converted to int arrays as that is how openAL reads sound files.

LoadSoundFile() loads the wav file in a way that allows openAL to read it and load it.

LoadAudio() is the main method that is used within these files. It loads the file from the pathway supplied and determines all the information needed to play the file (sample rate etc). The information is then saved into a buffer that is ready to be used.

PlayAudio() This plays the audio calling the id previously created using the load audio method and the position that the user wants the sound to be played at.

StopSound() simply stops the sound that is being played.

The SetListener() method is also an important method being used in these files. The listener is what will essentially be “listening” for the audio and will allow the player to hear the audio being played. The pos of the listener affects how the player will hear the audio.

# MainGame

The MainGame header and cpp files contain the bulk of the code that is used to create the game.

The MainGame method is used as the initial setup and simply declares all the basic things needed before the rest of the code is called. This is simply changing the game state, so they know the game is playing, creating all the initial game objects that will be used in the game and creating the audio device so sound can be used.

CreateEverything() is again what is used for set up, it creates the game display, assigns the pathways to all the gameobject models so they can be drawn later, assigns the pathways for the sounds, creates the camera and sets the initial position for the player.

The GameUpdate() method is essentially an update method that runs the methods in it every frame until the gamestate is not playing. In other words, it updates whenever the game is open.

The player inputs method is used to check the player’s inputs. It checks if the player has pressed specific keys and acts accordingly. It checks if the player has pressed any of the arrow keys and moves them in the appropriate direction. It also checks if the player has pressed the space bar and if so it changes a variable to true that will fire a bullet and it also sets the position of the bullets.

The collisionDetection method checks to see if any of the spheres attached to the game objects have intersected with each other and returns a Boolean. In other words, it checks to see if the game objects have collided with each other.

The playAudio() method plays the audio. This method checks if the audio is already playing and if it isn’t play the audio.

DrawTheGame() method is what actually draws all the models onto the display and textures them. The method starts by wiping the display so that the game objects from the previous frame, it then creates the shaders and textures loading them from their respected pathways. It then sets the position, rotation and scale of each of the game objects, draws them and textures them. It also checks if the player has fired and if so, it draws a bullet.

AsteroidHitsPlayer checks to see if any of the asteroids has collided with the player and if so it deconstructs the asterioid exits the game.

BulletHitsAsteroid does the same thing except it checks if the bullet has collided with the asteroid. If the bullet has collided with the asteroid it deconstructs the asteroid, sets the playerfired Boolean back to false so it stops rendering. It also increases the players score by one and plays a bang sound.

EndGame Exits the game if the players score is equal to 5. i.e. if all the asteroids have been destroyed.

DrawSkybox() is used to create a skybox and texture it. The method creates a shader and texture using the appropriate pathways. It essentially draws a cube taking in a float as a parameter that will act as the area of the cube. Each of the face of the cube is then drawn and textured.

# Appendix

Spaceship - <https://free3d.com/3d-model/low-poly-spaceship-37605.html>

Asteroids - https://www.cgtrader.com/3d-models/space/planet/low-poly-asteroids

Skybox Tutorial - https://www.youtube.com/watch?v=imCNAWMC1Xs