Games Programming 2 Coursework

Ivan Konstantinov

Student ID: S1829087

*I confirm that the code contained in this file (other than that provided or authorized) is all my work and has not been submitted elsewhere in fulfillment of this or any other award*.

Contents

[1. Overview - 2 -](#_Toc63189639)

[2. Display - 3 -](#_Toc63189640)

[3. Camera - 4 -](#_Toc63189641)

[4. Mesh - 5 -](#_Toc63189642)

[5. Shader - 6 -](#_Toc63189643)

[6. Textures - 7 -](#_Toc63189644)

[7. Transform - 8 -](#_Toc63189645)

[8. Main Game - 9 -](#_Toc63189646)

[9. Extension - 10 -](#_Toc63189647)

[10. References - 11 -](#_Toc63189648)

# Overview

My coursework is to create 3D environment using OpenGL and C++. It should contain 3 different models, with their textures. I have decided to create a scene where a wolf must catch the cat with user control. The third model is a moon for decoration at the top right corner. After the cat has been caught, a collision sound is triggered. I have added lightning and user control of the model via input.

Link to github: <https://github.com/vankata1998/GP2_coursework>

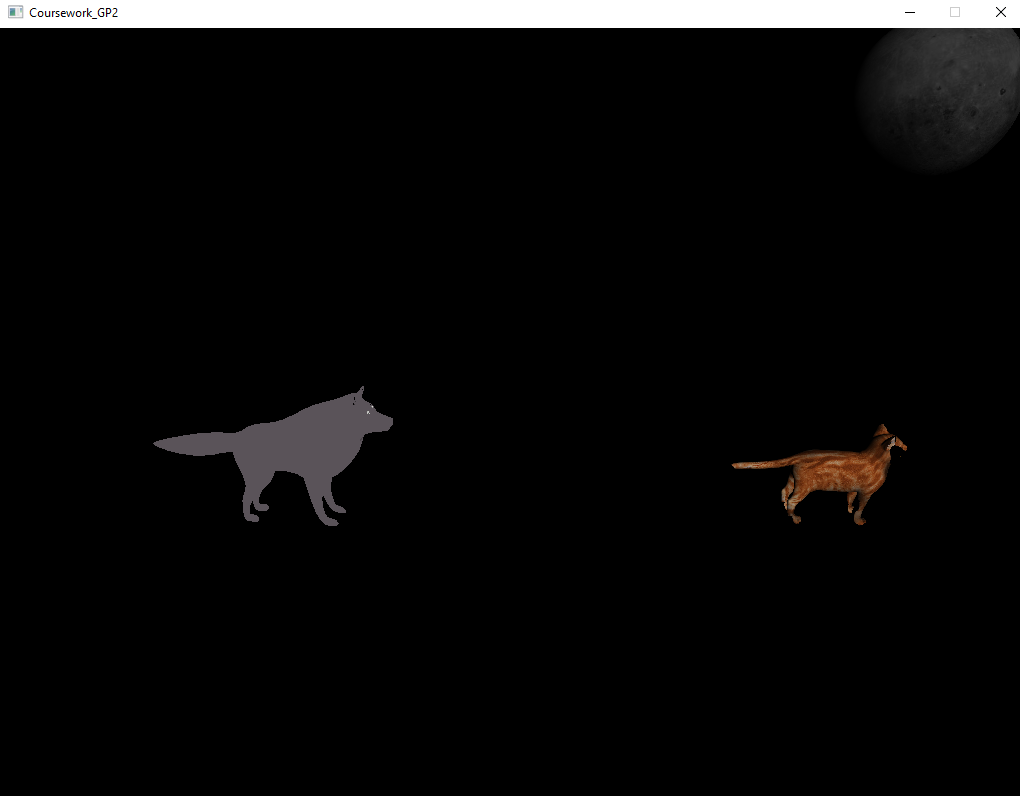


Figure 1.1

# Display

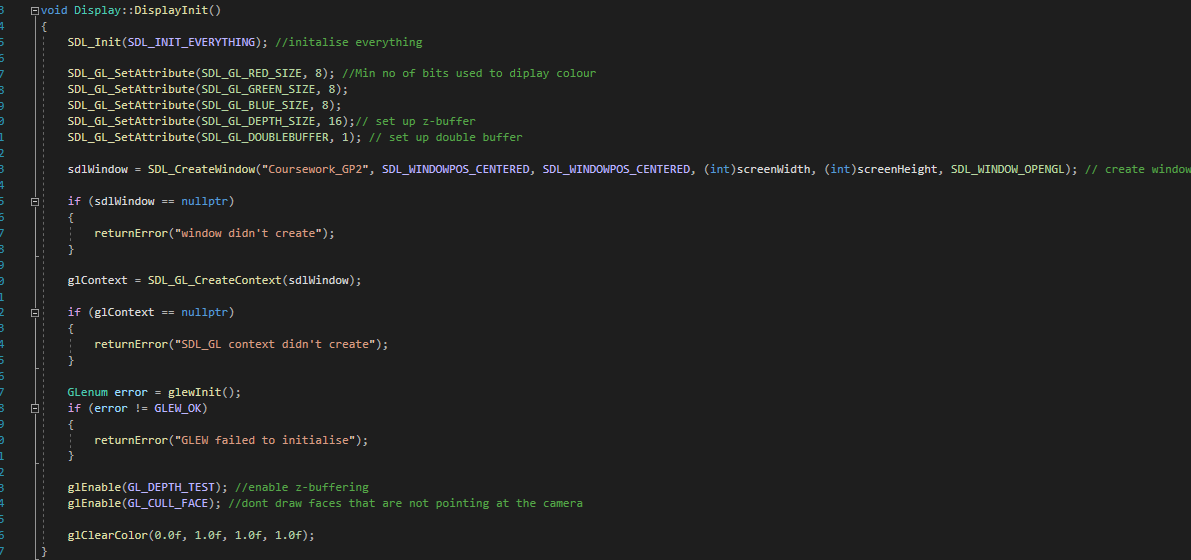


Figure 2.1

In Display.cpp the constructor sets the pointer to the display to be equal to NULL, screenWidth to be 1024 and screenHeight to be 768. In the destructor, the window and the glContext are being destroyed after the program is closed.

In the returnError() function the program is looking for an error and if any is found it is being returned to the console. In the swapBuffer() function the SDL\_GL\_SwapWindow (sdlWindow) is being used to swap between the display buffer and the draw buffer on which the GPU is drawing. In clearDisplay() function the Display is cleared and set to a specific color variable.

In the DisplayInit() function SDL is being initialized and a window is being created using SDL\_CreateWindow with the required parameters. The program also has some getters to get the screen Width and Height.

# Camera

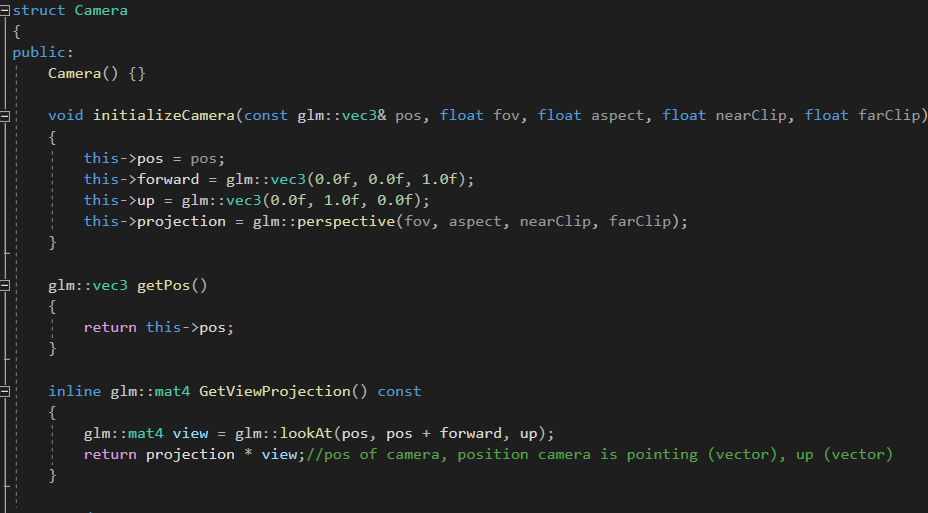


Figure 3.1

The camera struct is created and it is responsible for the camera position and movement. In InitializeCamera(), camera position, forward, up, and projections are being set.

Camera position is where the camera is located in the game space. Forward is in which direction the camera is looking. In getPos() the function returns the position of the camera when it is called. The projection is a glm::mat4 variable and the position, forward, and up are glm::vec3.

# Mesh

Mesh class is used to generate the mesh of the model by using its vertices.

The init() function initializes the mesh. The initModel() function draws the mesh by using the glGenVertexArrays, glBindVertexArray, glGenBuffers ,glBindBuffer, glBufferData, glEnableVertexAttribArray functions that are defined in OpenGL and they are responsible for generating a vertex array and moving the data which is in there to the GPU instead of the CPU. In the loadModel() function a model is being loaded by giving this method the name of the model obj file. The draw() function draws the mesh every second.

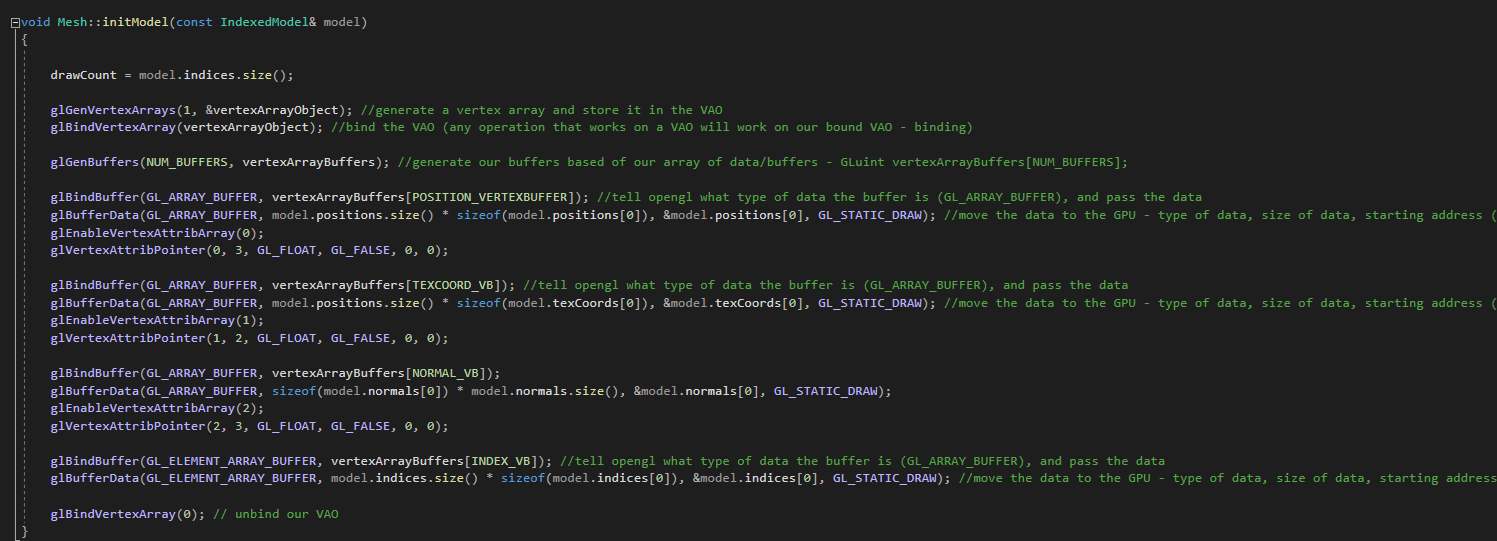


Figure 4.1

# Shader

Figure 5.1

In Shader Class, the vertex and the fragment shader files are added to the program. Then the program checks if there are any errors with that shaders implementation. In the destructor of that class, all shaders are being deleted.

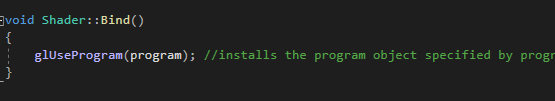


Figure 5.2

In Bind() the function installs a program as part of the current rendering state. In the Update() the shader is being updated each frame. In CreateShader(), it takes the shader text and returns the shader for the program to use., There are some error checks, to assure that the shader is created correctly. The LoadShadder() takes a string reference of a file name and it loads the shader from that file.

# Textures

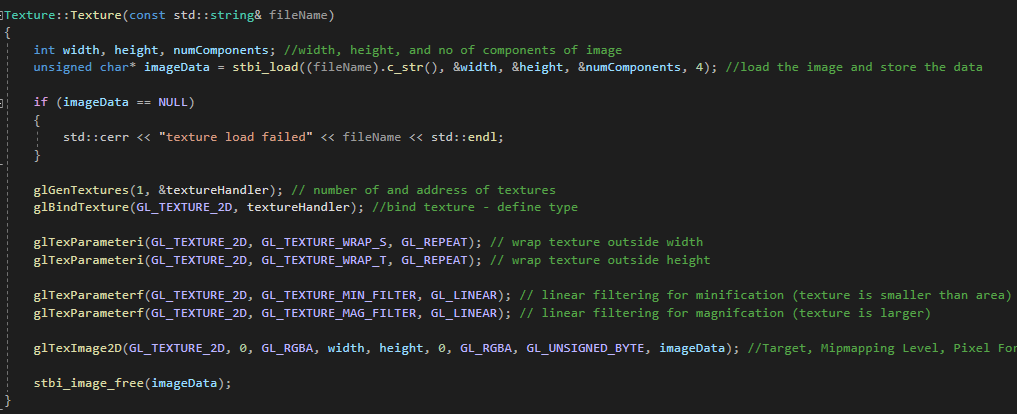


Figure 6.1

In Textures.cpp we have constructor, destructor, and a Bind() function. The constructor has stbi() load function that takes a filename and loads the texture file into the program. Then the program gives the address of our textures to glGenTextures. Then the program binds the texture to our texture handler and we specify how much is the minification and magnification of the texture. Then we load the Texture to the program using glTexImage2D(). In the destructor, the program deletes the reference to the texture.

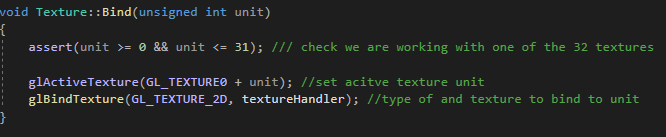


Figure 6.2

In the Bind() function, we check if the program is working with 1 of the 32 textures and then the program sets the following texture to active and bind the texture to the texture handler.

# Transform

Figure 7.1

In Transform.h the models that are already in the program are getting their transforms, or in other words positions within the world space. The constructor is setting their positions. The GetModel() function works like a getter, it let us take the exact position multiplied by the model's rotation and the scale of the model. This tells us exactly where is our model positioned in the game world and in what direction it is looking at and how big it is. In transform.h the program also has setters and getters for position.x, position.y, position.z, rotation, and scale to be able to access and change that value out of transform.h

# Main Game

This is the file responsible for the whole gameplay of the game. The first thing the program does in it is to give all the meshes transforms within the game space. After that, the program has the MainGame() constructor which creates mesh() for each mesh that the program has in the game. In the run function, the game calls initSystems ()and gameLoop(). InitSystems() creates a display, initializes it, loads all the audio, sounds effects, mesh models, and initializes the camera and the shader. Also, there is a variable called counter which is responsible for the movement of the characters. In the gameLoop() function the program has ProcessInput(), drawGame(), collision detection function and playAudio(). The ProcessInput checks If an event has occurred which in our case will be SDL\_QUIT which quits the whole program or SDL\_KEYDOWN, which looks if the specific button was pressed by the user, if it was, it does the following actions, which in our case, let’s the wolf to transform its position and chase the cat. In the collision, the program creates spheres around all meshes, and then with the radius and some mathematics the program checks if the two different mesh spheres have collided, and if they have a specific voice is played. In the drawgame() function Texture references are created and the texture jpg are added by their file name to the following references. Then the program binds the shader, updates the shader, draws the mesh, sets mesh rotation and scale, and updates the sphere data for the collision sphere for each mesh. All this happens on the drawgame(), once the function is done, the program swaps the buffer with the display one, and starts drawing the next scene. All that functions are used constantly within the gameLoop().

# Extension

For my extension material, I decided to add the user input controller and lightning to the game. For the user input, I used the SDL\_PollEvent() function which lets me track if a certain event has happened, and therefore I can check if a specific key on the keyboard was pressed or released, with that I have set W,A,S,D to be the movement keys for the wolf. I move the wolf by setting its mesh transform position using the SetPos() function with glm::vec3 formed from the x.position, y. position, and z.position added to counter, depending on what movement I want to get.

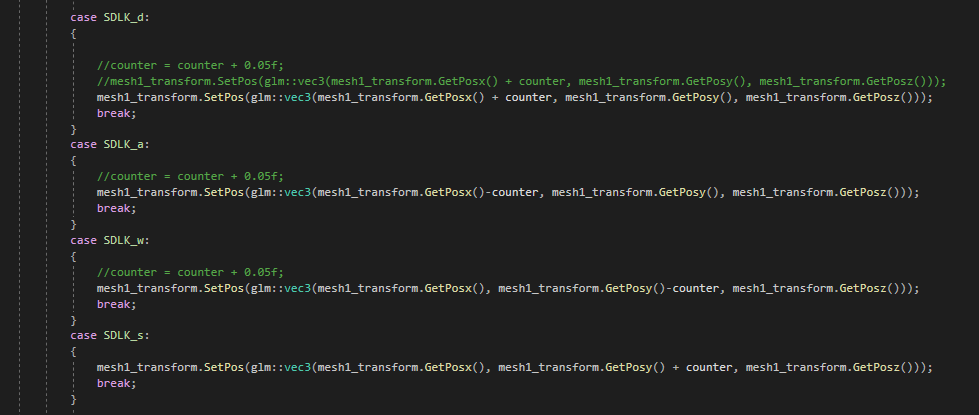


Figure 9.1

For the lightning, I use Lambertian Lightning. In the Fragment Shader I set the gl\_FragColor to be equal to texture2D(diffuse, texCoord0)\* clamp(dot(-vec3(0,0,1), normal0), 0.0, 1.0), which takes our texture and multiplies it by a certain value which is between 0 and 1 and this gives us the lightning

# References

I have used <https://youtu.be/ftiKrP3gW3k?list=PLEETnX-uPtBXT9T-hD0Bj31DSnwio-ywh> for the Lambertian lighting tutorial.

The moon: <https://free3d.com/3d-model/realistic-moon-65166.html>

The cat: <https://free3d.com/3d-model/cat-v1--522281.html>

The wolf: <https://free3d.com/3d-model/low-poly-wolf-432873.html>

For the soundtrack, I have used the short free version of :

<https://www.playonloop.com/2015-music-loops/countering-evil/>

The textures of the moon and the cat come with their models and for the wolf is used the brick texture that was provided in the labs. The collision sound is also from the labs.

Video of the coursework:

<https://youtu.be/a3iZXcefbEA>