

**Coursework Report**

**Games Programming 2**

**Module Name:** Games Programming 2

**Module Code:** M3I625656

**Module Leader:** Bryan Young

**Student ID:** S1717551

**Student Name:** Dawid Piotr Kubiak

*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*.

*Dawid Kubiak*

<https://github.com/dejwkubikson/GP2-Coursework>

Contents

[1. Audio class 3](#_Toc29560762)

[Header file 3](#_Toc29560763)

[Source file 4](#_Toc29560764)

[2. Camera class 4](#_Toc29560765)

[Header file 4](#_Toc29560766)

[3. Display class 5](#_Toc29560767)

[Header file 5](#_Toc29560768)

[Source file 5](#_Toc29560769)

[4. GameObject class 6](#_Toc29560770)

[Header file 6](#_Toc29560771)

[Source file 6](#_Toc29560772)

[5. Main class 6](#_Toc29560773)

[6. MainGame class 6](#_Toc29560774)

[Header file 6](#_Toc29560775)

[Source file 7](#_Toc29560776)

[7. Mesh class 7](#_Toc29560777)

[Header file 7](#_Toc29560778)

[Source file 8](#_Toc29560779)

[8. Shader class 8](#_Toc29560780)

[Header file 8](#_Toc29560781)

[Source file 8](#_Toc29560782)

[9. Skybox class 9](#_Toc29560783)

[Header file 9](#_Toc29560784)

[Source file 9](#_Toc29560785)

[10. Texture class 10](#_Toc29560786)

[Header file 10](#_Toc29560787)

[Source file 10](#_Toc29560788)

[11. Transform class 10](#_Toc29560789)

[12. External resources 11](#_Toc29560790)

[1. Audio: 11](#_Toc29560791)

[2. 3D models: 11](#_Toc29560792)

[3. Textures 11](#_Toc29560793)

# Audio class

Audio class is used in order play sounds in the scene.

## Header file

Audio header file holds data and vector3 structures, ALC context and device, functions to check if the computer stores bytes in big endian or small endian format (explained below), converts data to integers, loads WAV audio files, loads sounds to the scene, sets the audio’s listener, plays and stops the sounds.

## Source file

### Audio()

The constructor opens the sound card and creates the audio context.

### ~Audio()

The destructor deletes all audio sources and buffers, destroys the context and closes the audio device.

### isBigEndian()

This function returns a Boolean value. It checks if the computer stores bytes in big endian or small endian format. A big endian format is that the significant values are stored first, the small endian the other way around which is significant values are stored at the end.

### convertToInt(char\* buffer, int length)

Converts the data to integer value.

### loadWAV(const char& fn, int& chan, int& samplerate, int& bps, int& size)

The function loads a WAV audio file using input stream class. A char value is returned that contains the sound’s data which is used later in loadSound() function to determine for example what format should be set (Mono / Stereo) to play the sound.

### loadSound(const char\* filename)

This function tries to load a sound. Sets the audio’s format, adds the audio data to a vector list containing the data structure. Returns an integer holding the ID of the audio file.

### playSound(unsigned int id, glm::vec3& pos)

Plays the sound at specified vector3 position.

### stopSound(unsigned int id)

Stops playing the sound.

### setlistener(glm::vec3& pos, glm::vec3& camLookAt)

Sets the listener of the audio.

# Camera class

Camera class is used to display the scene to the player.

## Header file

Camera does not have a source file, therefore all functions are declared and instructions are set. The Camera.h file contains functions to initialize the camera, return view projection, manipulate the camera to simulate movement or rotate it. Furthermore the camera stores variables that store the camera’s projection, position, forward and up vectors and a Mesh class.

### Camera()

Constructor.

### initCamera(const glm::vec3& pos, float fov, float aspect, float nearClip, float farClip)

Initializes the camera at given vector3 position, sets the forward and up vectors and the projection. A collider is created and its sphere data is set to current cameras position with a radius equal to two.

### GetViewProjection()

The function returns the current view matrix.

### MoveForward(float amt)

Responsible for moving the camera forward and backward (when a negative amount is set). Adds the amount to the current position vector. The Y axis is blocked so that the camera does not fly away – for example when looking up – and is always set to 0. The collider’s sphere is updates with new position.

### MoveRight(float amt)

Moves the camera right and left (when a negative amount is set). Adds the amount to the current position vector using a cross product of up and forward vectors. The Y axis is blocked so that the camera does not fly away – for example when looking up – and is always set to 0. The collider’s sphere is updates with new position.

### RotateY(float angle)

Rotates the camera on Y axis (up and down). Sets the new forward and up vectors.

### RotateX(float angle)

Rotates the camera on X axis (left and right). Sets the new forward and up vectors.

### getPos()

Returns a vector3 containing current position of the camera.

# Display class

Handles the actual window of the game and display settings.

## Header file

The header file contains variables that hold the context, pointer to the window, screen height and width and arrays that hold light settings. Initializes and clears the display, swaps buffers, displays fog and light effect. Moreover, holds functions to return the screen width and height as the variables storing the values are private.

## Source file

### Display()

Display constructor initializes the game window with a null pointer to generate null access violation for debugging. Moreover, it sets the screen’s width and height.

### ~Display()

Deletes the context and game window. Cleans up all initialized SDL subsystems.

### initDisplay()

Initializes all SDL subsystems. Sets minimum number of bits to use to display each colour and what buffer to use. Creates the game window in the centre of the screen. After that, enables z-buffering and setts faces that are not visible to the camera to not render. Does simple error handling and clears (sets) the screen colour to black.

### swapBuffer()

Uses SDL\_GL\_SwapWindow function to update the window with OpenGL rendering.

### clearDisplay(float r, float g, float b, float a)

Clears colour and depth buffers.

### getScreenWidth()

Returns a float value containing the screen width.

### getScreenHeight()

Returns a float value containing the screen height.

### displayFog()

Uses GL\_MODELVIEW to create a fog with nicest (visual) settings. The fog is set to display as an exponential formula meaning that the further away from the camera the foggier it gets. Then the fog density and colour (grey) are set.

### displayLight()

Uses GL\_MODELVIEW to create a light effect. Setting the ambient, diffuse and specular intensity of the light and its position.

### returnError(string errorString)

Error handling. Display error message and cleans up all initialized SDL subsystems.

# GameObject class

The class is defined to make it easier to access all game objects in the scene.

## Header file

The header file contains the classes constructor and destructor, functions to load model and texture and references to Mesh, Texture and Transform classes.

## Source file

### GameObject()

Simple constructor.

### ~GameObject()

The destructor deletes the mesh and texture.

### LoadModel(const string& filename)

Loads model using Mesh’s class loadModel function.

### LoadTexture(const string& filename)

Loads texture using Texture’s class Initialize function.

# Main class

The class creates MainGame class that runs the whole game.

### Main(int argc, char\*\* argv)

The argument is used to call SDL main. Creates MainGame class and runs it using its run() function.

# MainGame class

This class contains the main game loop therefore it is responsible for running the whole game.

## Header file

The header file contains the constructor, destructor and other functions to run the game, initialize all the systems, the main game loop, processing input, a function to draw the whole game, collision checking function and last but not least a function that is responsible for playing the audio clips. Holds references to Display, Camera, GameObject, Shader, Skybox and Audio classes. Moreover, contains an enum class GameState that holds one of two game states – play and exit. Finally, holds two unsigned integers that hold the background music and collision sound effect.

## Source file

### MainGame()

Locks the mouse in the centre of the screen, sets the game state to play and creates new display, shader, audio device and sky box.

### ~MainGame()

The destructor does not hold any instructions.

### run()

Starts the initSystems and gameLoop() functions.

### initSystems()

Initializing all components of the display, creating objects and loading textures, each object’s collision sphere is set. Shader is initialized and loaded from a file, Skybox’s and Camera’s classes initialization functions are called. Finally, both sounds are loaded.

### processInput()

Uses SDL events to determine what keys the user presses and to track the mouse movement. A switch statement checks for W, A, S, D keys (on press) that execute Camera’s MoveForward() and MoveRight() classes depending on which button was pressed. Furthermore, the user can quit the game using Escape button. If the event is a mouse motion then the RotateY() and RotateX() Camera’s functions are executed with a parameter equal to the distance the mouse travelled.

### gameLoop()

Holds the main while loop that stops only if the GameState changes to Exit. Start by processing input, playing background music to drawing the game using drawGame() function. Finally, checks if a collision occurred between the Camera’s sphere and any of the objects in the scene.

### collision(glm::vec3 m1Pos, float m1Rad, glm::vec3 m2Pos, float m2Rad)

Returns a Boolean value whether a collision occurred between the passed objects.

### playAudio(unsigned int Source, glm::vec3 pos)

Uses the Audio class to play a sound.

### drawGame()

The function starts by clearing the display and displaying fog and light effects. Drawing skybox and binding the shader. Each object’s position, rotation and scale are set, the shader updated, texture bind and the object are then drawn. Finally, the function swaps the buffer.

# Mesh class

Mesh class is responsible for object’s position and 3D model.

## Header file

The header file holds Vertex and Sphere structures. Vertex structure is used to generate position, texture coordinates and normal. Sphere structure is used to create a collider by setting its position and radius. Apart from the constructor and destructor the Mesh class has a draw, initialize, load model, initialize model methods. Moreover, the class has functions to update the sphere data. Finally, the header stores an enumerator holding details of the objects, sphere, vertex array and vertex buffers variables.

## Source file

### Mesh()

The constructor sets the drawCount variable to null.

### ~Mesh()

Deletes the vertex array.

### init(Vertex\* vertices, unsigned int numVertices, unsigned int\* indices, unsigned int numIndices)

The function creates a model using position, texture and normals vertices. The model is then initialized using initModel() function.

### initModel(const IndexedModel& model)

The method generates a vertex array and stores it in VAO. The VAO is bind and buffers are generated based on the vertex buffers array. The data is passed to OpenGL and moved to the GPU – this is done for the position, texture coordinates, normals and indices of the model. At the end the vertex array is unbind from VAO.

### loadModel(const std::string& filename)

Loads the 3D model by file name. The model is then initialized using the initModel() method and a collision sphere is created for that model.

### draw()

Binds the vertex array and draws the model.

### updateSphereData(glm::vec3 pos, float radius)

Updates the sphere’s position and radius using Sphere’s class methods.

### getSpherePos()

Returns a vector3 value of the collider’s position.

### getSphereRadius()

Returns a float value equal to collider’s radius.

# Shader class

Shader class is used to render the game.

## Header file

Sets the GPU to use specified shaders, loads and creates shaders. The header file holds the number of shaders, tracks the shader program and creates array of shaders and number of uniform variables.

## Source file

### Shader()

Constructor with no instructions in it.

### ~Shader()

Removes the shader from the program then deletes them. At the end frees the memory and invalidates the shaders.

### Bind()

Installs the program object specified by the program as part of the rendering state.

### Update(const Transform& transform, const Camera& camera)

Refreshes the shader.

### Initialize(const string& filename)

Creates the shader program. Loads vertex and fragment shaders and adds them to the program. Creates executables that will run on the GPU shaders, checks for errors and then again checks if the entire program is valid (checks if the executables can execute). Associates the location of uniform variable within the program.

### CreateShader(const string&text, unsigned int type)

Creates the shader based on the specified type. Checks if the shader has been created and converts string into list of c-strings. Sends the source do to OpenGL and tells it to compile the shader’s code. Finally checks for an error and returns a GLuint shader value.

### LoadShader(const string& filename)

Loads the shader using a file name.

### CheckShaderError(GLuint shader, GLuint flag, bool isProgram, const string& errorMessage)

Displays the error message if occurs.

# Skybox class

Skybox class is used in order to display skybox in the game.

## Header file

Holds functions to initialize, draw sky box and load its texture. Has a declared enumerator that holds the walls’ names and a skybox array that holds the textures’ IDs.

## Source file

### Skybox()

Does not hold any instructions.

### ~Skybox()

Deletes the skybox’s textures.

### initSkyBox()

Loads textures using loadSkyBoxTexture() function and stores their ids in the skybox array.

### loadSkyBoxTexture(const char\* filename)

Loads a BMP image using SDL library. Uses Linear filtering for minification (when the texture is smaller than the object) and magnification (when the texture is larger than the object). Repeats the pixels in the texture’s edges in order to remove a white line at each edge of the cube. Then the texture is created and the texture is removed (freeing RGB surface as it is no longer needed). Finally an integer ID variable is returned.

### drawSkyBox(float size)

Lighting and z-buffering is disabled, and all faces of the cube are created using the loaded textures. Lighting and z-buffering is enabled back on at the end of the method.

# Texture class

Used to perform actions on textures – texturing and binding the elements.

## Header file

Contains a GLuint textureHandler variable and functions that initialize and bind the texture.

## Source file

### Texture()

No instructions stored in this method.

### ~Texture()

Deletes the textures.

### Initialize(const std::string& filename)

Creates variables to store width, height, number of components and loads the image to store its data. The texture is then defined, wrapped around the outside width and height, linear filtering is used for minification and magnification and the texture is created.

### Bind(unsigned int unit)

Binds the texture.

# Transform class

## Header file

Used to perform any transformations on objects. Contains a header file only. The class contains functions to set and get position, rotation and scale.

### Transform(const glm::vec3& pos = glm::vec3(), const glm::vec3& rot = glm::vec3(), const glm::vec3& scale = glm::vec3(1.0f, 1.0f, 1.0f))

Constructor that sets the position, rotation and scale to the specified.

### GetModel()

Returns a mat4 value containing the multiplication of position, rotation and scale matrixes.

### GetMVP()

Returns a mat4 value containing the multiplication of camera’s view projection and the returned value by GetModel() method.

### GetPos()

Returns vector3 value containing current position.

### GetRot()

Returns vector3 value containing current rotation.

### GetScale()

Returns vector3 value containing current scale.

### SetPos()

Sets a vector3 value of the current position.

### SetRot()

Sets a vector3 value of the current rotation.

### SetScale()

Sets a vector3 value of the current scale.

# External resources

Audio:  
Background music – <https://www.youtube.com/watch?v=Ihq64W33cyo>

Collision sound - <https://www.youtube.com/watch?v=Pfp9Pi04HwM>

## 3D models:

Barrier - <https://free3d.com/3d-model/police-fencing-151405.html>

Crate - <https://free3d.com/3d-model/crate-86737.html>

Plastic box - <https://free3d.com/3d-model/plastic-fruit-crate-v1--190535.html>

## Textures

Wood - <https://mokatana.wordpress.com/seamless-wood-texture-free-76>

Yellow plastic – [https://www.photos-public-domain.com/2013/11/07/bumpy-yellow-plastic-texture](https://www.photos-public-domain.com/2013/11/07/bumpy-yellow-plastic-texture/)

Barrier - [https://www.pinterest.com/pin/430375308118044625](https://www.pinterest.com/pin/430375308118044625/)