University of Xi'an Jiaotong-Liverpool

School of Advanced Technology

CPT205 Assessment 2 Report

ID:	*2034675*
Name:	Kaijie Lai
Email Address:	Kaijie.Lai20@student.xjtlu.edu.cn

Dec. 18, 2022

1 Introduction

Computer Graphics has rapid development in recent decades and has been applied in many places in our lives, from Animation Film to Electronic-Game, even the formation of Meta Verse. This Assignment is a 3D Modeling Project based on knowledge which we learned in this semester. As required, I created a scene of Football Field and designed a Physics Engine based on the real environment so that the objects in it are constrained by gravity.

The game starts with First-Person View of a player who is able to run around the pitch with the ball together, and when in shooting mode the ball can be kicked in a customised size in a specified direction to simulate the movement of a football in a real-life scenario. Players can also go further into Penalty Mode, where they need to kick the ball in the goal from a specified position to score the corresponding points. This report will be divided into two parts: a guide to the operation of the program and the design details of the program.

2 Instruction

1. Start: After entering the game, Hold Left Button and Move, you will can see a football on the pitch, like any FPV game, you can use W/S/A/D to control your character's movement and orientation, the football will always be ahead of your view.
2. Addition: Scroll Wheel to adjust the FOV (recommended 70). Hold Shift+W/S for up and down movements. The top left corner of the screen represents different game modes and the top right corner shows some real-time game parameters.
3. Mode Shoot: Press 1 to enter shooting mode, where the football will be static and you can continue to move freely. Click Right Button to begin storing power. Longer bar represents more power to kick. Click again to stop storing and the ball will be kicked at an angle depending on your orientation. Also, you can Press R to reset football's position and press 1 again back free mode. (It is recommended that you take a few steps back away from the ball and keep the pitch angle at around 20-30 degrees to get the perfect arc).

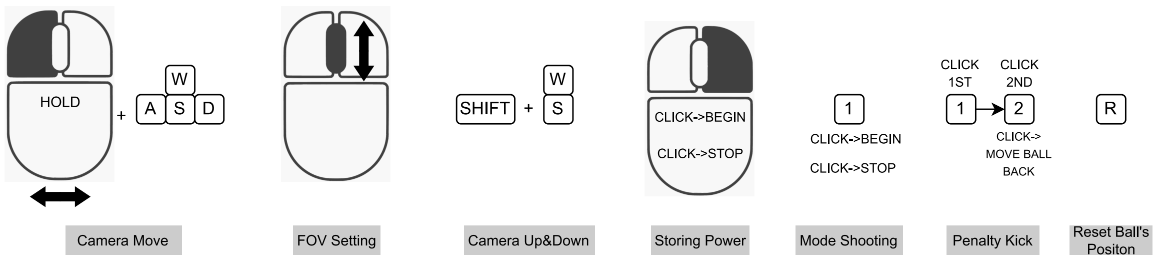


Figure 1: Instructions for some Operations

4. Penalty Kick: After Shoot Mode ON, Press 2 to enter the penalty mode automatically. You will have 5 yellow targets in to kick down by the same operation, one target stands one point.

3 First-Person View

First-Person View (FPV), also known as first-person point of view (POV), is the ability of some users of some technology to see from a particular visual perspective other than one's actual location. It is common in video games, allowing for more immersive play as a character.

Based on the **glutLookAt** function in the OpenGL freeglut library we can simulate camera as human eyes successfully. It creates a viewing matrix derived from an eye point, a reference point indicating the center of the scene, and an up vector.

```
gluLookAt (eye_x_, eye_y_, eye_z_, center_x_, center_y_, center_z_,
 0.0f, 1.0f, 0.0f);
```

Listing 1: Actual Implementation of Uploading

In the real world, kicking a ball away is more closer to launching a missile. And the direction of missile launch is based on the offset angle of the Pitch and Yaw axes. Therefore, We can use the model and geometric solution in *Figure 2*, to find the Pitch and Yaw angles, for kicking it to the direction we expected.

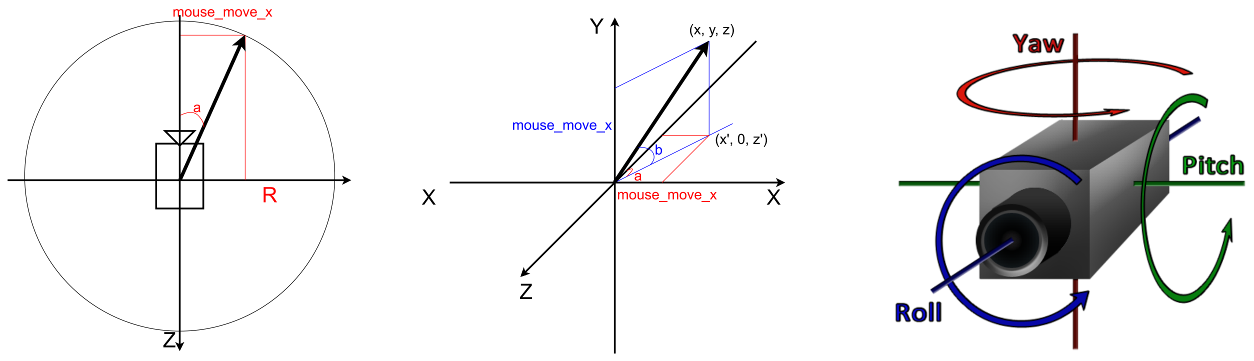


Figure 2: Illustration of the Camera View Calculation

Specifically, since the angle control in game is caused by the movement of mouse cursor on the screen, we can use the callback function **gluMotionFunc** to get the relative distances $mouse\_move\_x$ and $mouse\_move\_y$ per unit time of mouse movement on the screen. Given a unit circle radius of R and assume camera is rotating within this circle, then we can obtain the angle a of the Yaw axis, and similarly to obtain the angle b of the Pitch axis.

$$Angle\ a = arcsin(\frac{mouse\_move\_x}{R}) * \frac{180^\circ}{\pi} \tag{1}$$

$$Angle\ b = arcsin(\frac{mouse\_move\_y}{R}) * \frac{180^\circ}{\pi} \tag{2}$$

4 OOP Design

In order to achieve better maintainability, some objects are encapsulated in this project.

1. **Texture:** To make it easier and more efficient to use textures, I have set a object Texture so that only need to enter the corresponding file address in it to get information about the length, width and colour channel of the image. In addition, I have studyed and used the key point knowledge from *stb_image.h* to parse information about common image types such as JPG and JPEG, solving the disadvantage that BMP images are often too large.

2. **Lighting:** Lighting is also a very important part of the 3D scene and, in order to apply lighting more efficiently, it has set as class as well. **glLightfv** function is used to set the light for the parallel light in two positions and also set the corresponding ambient and diffuse lighting to make the scene more realistic.

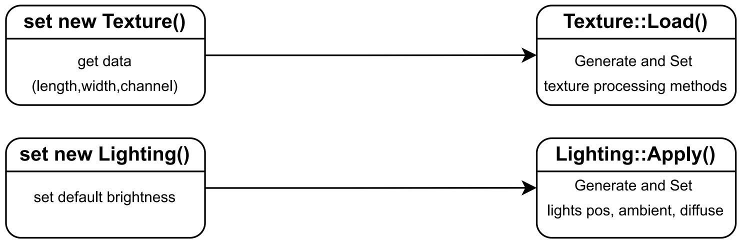
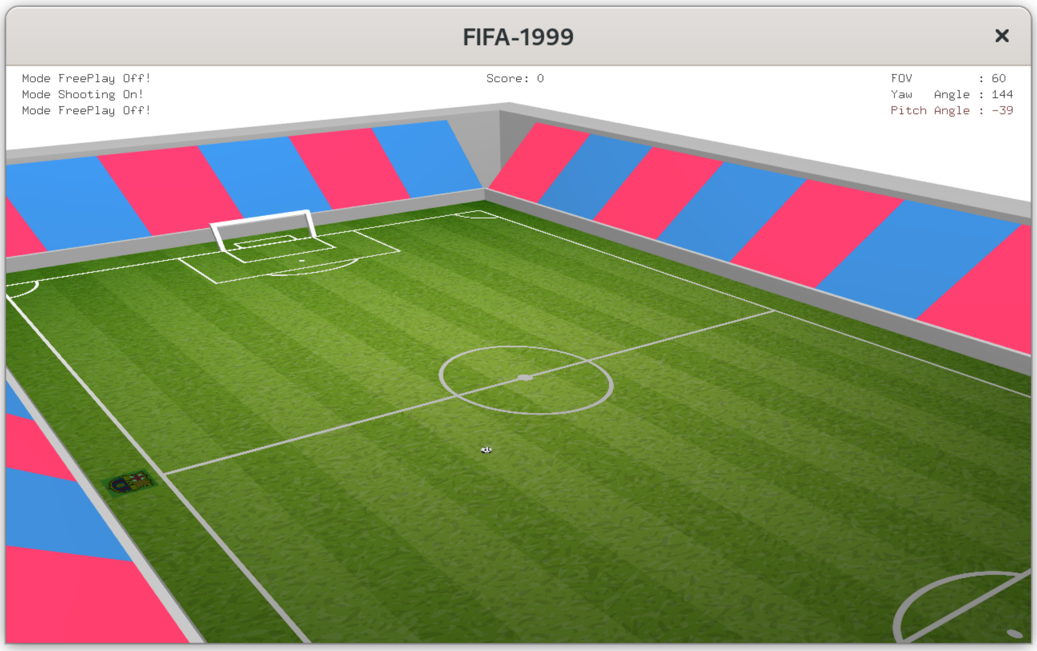


Figure 3: Implementation of Texture and Lighting objects



```
Mode FreePlay Off!              Score: 0              FOV         : 60
Mode Shooting On!                                     Yaw   Angle : 144
Mode FreePlay Off!                                    Pitch Angle : -39
```

Figure 4: Aerial view of field

3. **Game State:** In the GameState object all callback functions like keyboard and mouse, view changes and screen rendering will be managed respectively and clearly. This also stores the real-time information about the *position_xyz* and $rotation_xyz$ of the football in the physics engine.

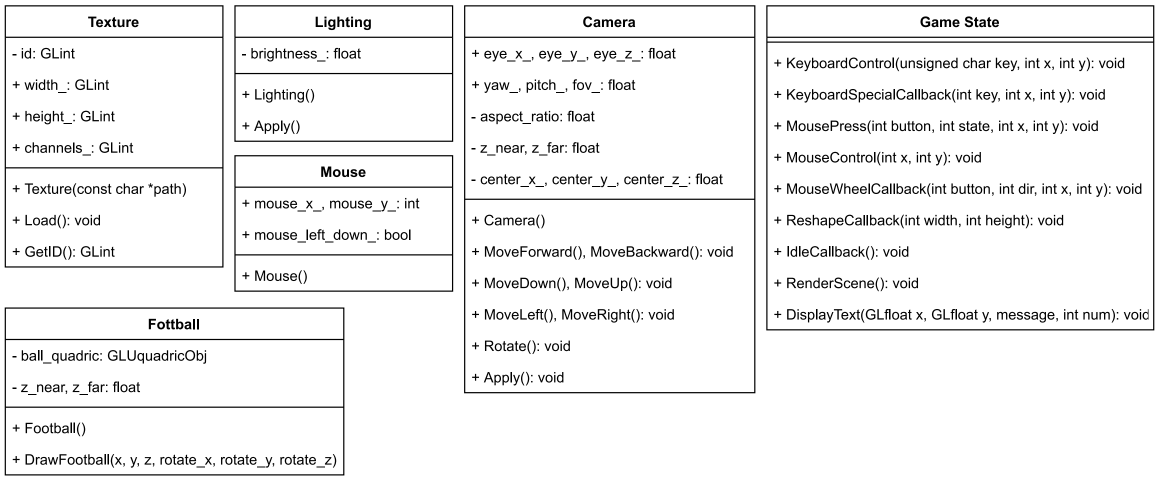
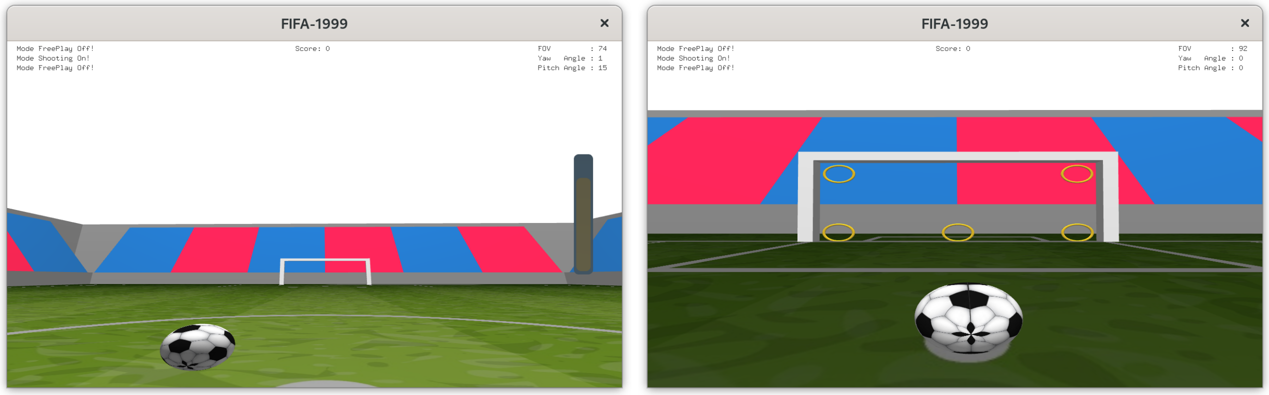

Texture
- id: GLint
+ width_: GLint
+ height_: GLint
+ channels_: GLint
+ Texture(const char *path)
+ Load(): void
+ GetID(): GLint
Lighting
- brightness_: float
+ Lighting()
+ Apply()
Mouse
+ mouse_x_, mouse_y_: int
+ mouse_left_down_: bool
+ Mouse()
Camera
+ eye_x_, eye_y_, eye_z_: float
+ yaw_, pitch_, fov_: float
- aspect_ratio: float
- z_near, z_far: float
- center_x_, center_y_, center_z_: float
+ Camera()
+ MoveForward(), MoveBackward(): void
+ MoveDown(), MoveUp(): void
+ MoveLeft(), MoveRight(): void
+ Rotate(): void
+ Apply(): void
Game State
+ KeyboardControl(unsigned char key, int x, int y): void
+ KeyboardSpecialCallback(int key, int x, int y): void
+ MousePress(int button, int state, int x, int y): void
+ MouseControl(int x, int y): void
+ MouseWheelCallback(int button, int dir, int x, int y): void
+ ReshapeCallback(int width, int height): void
+ IdleCallback(): void
+ RenderScene(): void
+ DisplayText(GLfloat x, GLfloat y, message, int num): void
Fottball
- ball_quadric: GLUquadricObj
- z_near, z_far: float
+ Football()
+ DrawFootball(x, y, z, rotate_x, rotate_y, rotate_z)

Figure 5: Objects in program

4. **Football:** The football object is also an important object in the program. The key point to simulate a realistic animation in game is calculate different positions and draw it out per unit time.
Physics Engine: The initial velocity *init_v* of the corresponding size is generated according to the length of the accumulation time, the constant acceleration a and g in the horizontal and vertical directions of the given object in motion according to the knowledge of physics, and the position of the football in the spatial coordinate system in different units of time during a flat throwing motion is calculated through the formula of the kinematic theorem per unit of time.



```
Mode FreePlay Off!              Score: 0              FOV         : 74
Mode Shooting On!                                     Yaw   Angle : 1
Mode FreePlay Off!                                    Pitch Angle : 15
```

```
Mode FreePlay Off!              Score: 0              FOV         : 92
Mode Shooting On!                                     Yaw   Angle : 0
Mode FreePlay Off!                                    Pitch Angle : 0
```

Figure 6: Shoot Mode and Penalty Mode in game

$$init\_v\_vertical = init\_v * \sin(pitch) \quad (3)$$
$$init\_v\_horizontal = init\_v * \cos(pitch) \quad (4)$$

$$relative\_x = init\_v\_vertical * ty - \frac{g * ty^2}{2} \quad (5)$$
$$relative\_y = -(init\_v\_horiziontal * tx - \frac{u * tx^2}{2}) \quad (6)$$

$$football\_position\_x = init\_x + relative\_x * \sin(yaw) \quad (7)$$
$$football\_position\_y = init\_y + relative\_y \quad (8)$$
$$football\_position\_z = init\_z + relative\_x * \cos(yaw) \quad (9)$$

By the way, there are two functions **DisplayText** and **DisplayAccumulateBar** to achieve the effect that text and graphics are always displayed in 2D graphics in the 3D scene. And debug-mode is facilitated by the macro definition and the design of the Log in Helper.h .

```
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
gluOrtho2D(0.0, INIT_WINDOW_WIDTH, 0.0, INIT_WINDOW_HEIGHT);
glMatrixMode(GL_MODELVIEW);
```

Listing 2: Display 2D graphics in 3D scene

5 Decision Flowchart

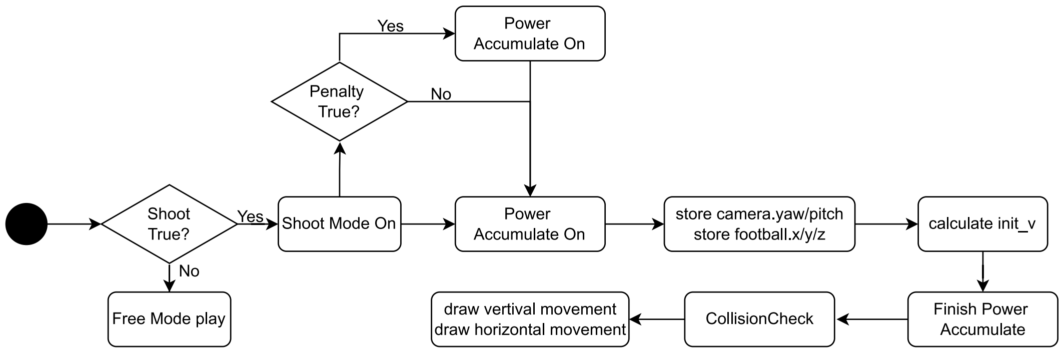


Figure 7: Decisions' flowchart in 3 game modes

Here is the end of this report.