Xi'an Jiaotong-Liverpool University

School of Advanced Technology

Assessment 2 - 3D Modelling Project

Author:
Zheyuan Cao

Student ID & Degree
2141805 & Information and Computing Science

A Report Submitted For

CPT205 Computer Graphics

December 15, 2023

1 Introduction

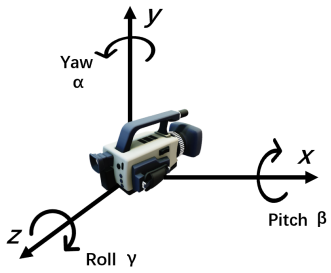
Computer Graphics, a branch of Computer Science that focuses on the rendering and analysis of 2D images and 3D models to assist diverse fields such as data visualization, virtual reality and computer-aided design (CAD). This assignment delves into 3D modelling, requiring a combination of theoretical knowledge with programming skills. Specifically, it involves utilizing OpenGL functions to implement graphical techniques like viewing, hierarchical modelling and texture mapping.

To fulfil modelling missions, this project produces an interactive 3D game that simulates the combat airplane training scene. The aircraft is equipped with a weapon system while users are capable of conducting relevant control such as manipulating the flight trajectory and switching perspectives.

2 Design Techniques

- **Object-Oriented Programming (OOP):** The project utilizes OOP principles to instantiate multiple classes and corresponding objects such as *Camera*, *PointLight* and *Texture*. These entities encapsulate associated data attributes and functionalities within predefined interfaces, hiding concrete implementation details to simplify the system maintenance. For example, *Texture* class defines methods like *ReadImage()* for image uploading through relative addresses while *Camera* initializes the position of the camera in the world co-ordinate.
- **Union Construction:** Union is a C++ operation that facilitates diverse access to fixed memory space. Specifically in this work, it provides a dual-access for the vector structure. For example, it allows for explicit direct access to individual components such as x, y and z in vector *Vec3*, concurrently enabling implicit indexed access like *value[3]* in *Vec3*, which guarantees the program efficiency and flexible data manipulation.
- **Overloading:** Overloading techniques employed in the program serve as distinct purposes. For instance, overloading for construction functions *Vec2()* and *Vec2(float inX, float inY)* to instantiate objects with default or specific values while the vector classes overload operators like *, += and -= to implement algebraic vector operations. Moreover, the keyword *inline* in C++ facilitates the integration of these overloaded vector operators into the calling context by the compiler, as opposed to adhering to the standard function invocation procedures. This combined strategy diminishes the function calling overhead and optimizes the application performance.
- **Creation of Geometry:** Various geometric primitives are applied in the modeling of realistic objects, such as spheres to embody enemies and cubes to represent wings of the aircraft.
- **Transformations:** Transformations are crucial for the verisimilitude of 3D scenes, encompassing both object and lighting manipulation. Specifically, fundamental operations such as translation, scaling and rotation are employed to position and mould the object as desired. For instance, transformation instructions with distinct arguments, wrapped around the *glPushMatrix()* and *glPopMatrix()*, are utilized to generate spherical targets with diverse locations and sizes. Similarly, model-view matrix contents like *glTranslatef()* are instrumental in the precise relocation of the light source within the virtual space, thereby contributing to the enhancement of the overall rendering effect.
- **Viewing and Projection:** The program renders a perspective view with the function *gluPerspective()*, resembling the authentic human visual system. More specifically, the OpenGL function

gluLookAt() forms a viewing matrix predicated upon a specified camera position, a scene-central focal reference point and a vector that denotes the upward orientation. Furthermore, the enumerable class *ECameraView* defines three distinct views from top, front and side respectively, which enables the transformation of camera co-ordinates. Moreover, the rotation of the camera emulates the 3-dimension axis in aircraft dynamics: Yaw, Roll and Pitch, as illustrated in figure 1 below. The *mouseMotion()* function calculates the yaw angle, contingent on mouse horizontal displacement within the viewport, simulating the horizontal yaw rotation.



$$Angles\ cameraRotation = -\frac{mousePosition}{windowWidth} \times 180^{\circ}$$

Figure 1: Camera Rotation

- **Lighting and Materials:** Illumination constitutes an indispensable part of the graphical rendering pipeline, including three primary categories of light sources that are employed in this project: radially diverging point light, infinite directional light and focused spot lights. Specifically, to simulate the realistic sunshine, the *PointLight* class defines the source position, whereas *DirectionalLight* specifies the direction for parallel light rays. Particularly for *SpotLight*, it constructs an illumination cone with an angular limit. These configurations are methodically executed through iterative invocations of *glLightfv()*, paired with the definition of material properties such as ambient, diffuse, and specular colours via the function *glMaterialfv()*.
- **Texture Mapping:** Texture mapping, a crucial stage in the rendering pipeline, projects intricate patterns onto the surface of geometric objects, thereby facilitating virtual models to closely approximate corresponding real-world entities. The technique involves two principal stages: specification and application. Initially, images are loaded and defined as textures via *glTexImage2D()* with specific details such as mipmapping level. Subsequently, texture parameters like wrapping modes and

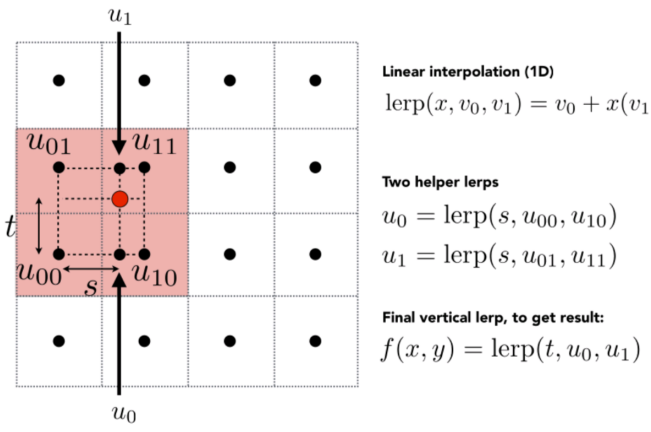


Figure 2: Bilinear Interpolation

filtering methods, are set through *glTexParameteri()*. For example, to optimize the texture magnification and minification, the bilinear interpolation whose algorithm is explained in figure 2 above, is utilized to implement the smooth shading on texel.

- **Hierarchical Modelling:** Complex 3D scenes, composed of multiple discrete entities, necessitate a structure beyond linear modeling. These entities are thus structured into a hierarchical scene graph, facilitating transformations via depth-first or breadth-first traversal with matrix stack. For example, figure 3 depicts the Direct Acyclic Graph (DAG) of an aircraft while figure 4 displays a hierarchical tree of the whole scene.

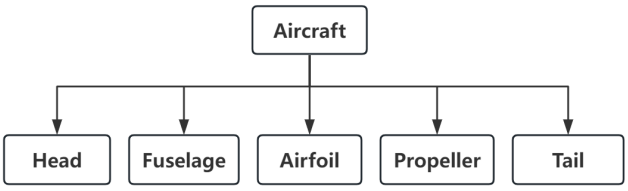


Figure 3: Aircraft DAG

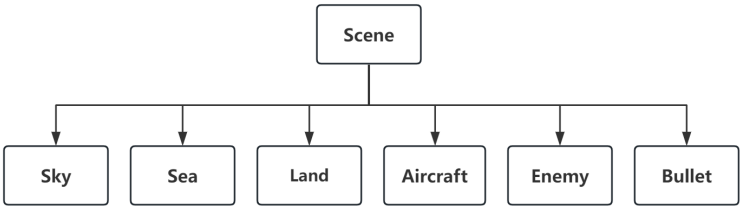


Figure 4: Scene Hierarchical Tree

- **Animation and Interactions:** The incorporation of static and animated elements is essential for modelling vivid 3D scene. In this project, animations are strategically applied using callback functions. For example, dynamic features like autonomous aircraft flight and the motion of bullets are achieved by linking to idle function *glutIdleFunc(update)*. Furthermore, custom callback functions such as *key()* and *mouse()* are designed to enhance user interactions. For instance, the player can control the flight direction and shoot at enemies to simulate pilot training. More interaction details will be explained in Instruction section.

3 Objects And Features

- **Aircraft:** The aircraft serves as a pivotal element within this work with construction stemming from geometry primitives such as cubes and cones, specifically through the utilization of *glutSolidCube()* and *glutSolidCone()* functions within the *drawPlane()* method. Relevant movements are accomplished via the function *updatePlane()*, involving the forward flight and horizontal rotation.
- **Bullet:** Bullets shot from the aircraft consist of spherical entities endowed with lighting effects, generated by the construction method *drawBullets()*. Furthermore, the *updateBullets()* encapsulated in *update()* function is responsible for defining corresponding dynamic attributes, encompassing directional vectors and velocity parameters. Users are capable of discharging them to shoot at enemies or any spatial co-ordinates within the virtual world.

- **Enemy:** The last primary component entails the instantiation of spherical enemies with various sizes, achieved via the combination of class *Enemy* with its related constructor *drawEnemies()*. Upon successfully shot by bullets, they will disappear from the screen to simulate the obliteration by the user, which implemented in *updateEnemies()*.

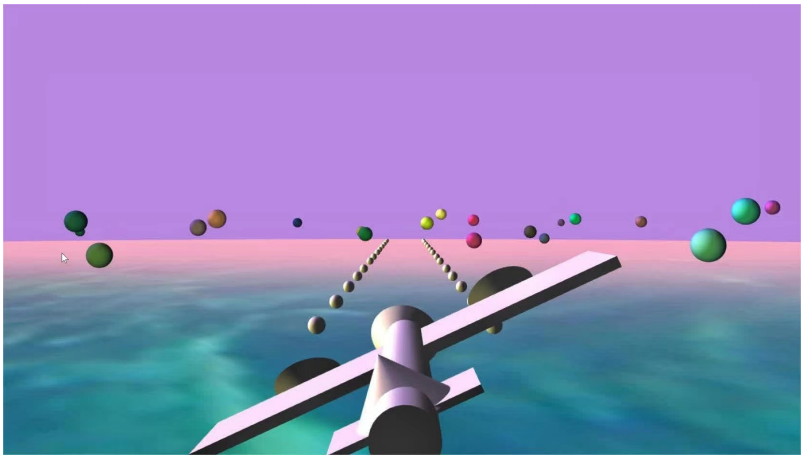


Figure 5: Flight Simulator Components

Figure 5 above displays an overall view of main components outlined.

4 Instructions

This section explains interaction instructions in detail, which are exhibited in figure 6 below.

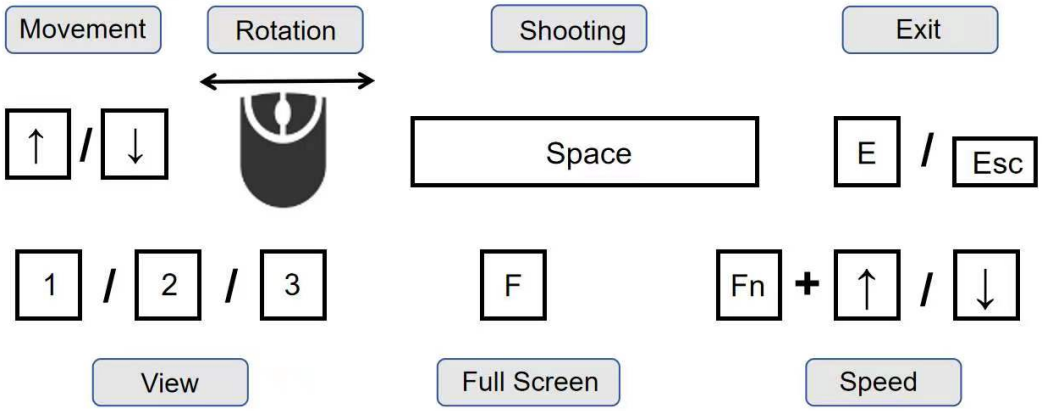


Figure 6: Interaction Instructions

- **Vertical Movement:** Press **Up Arrow** and **Down Arrow** to move up and down.
- **Horizontal Rotation:** Hold and swag the **Mouse** horizontally to turn left or right.
- **Shooting:** Press the **Space Key** to fire bullets.
- **Speed Control:** Press the **Page Up** and **Page Down** to speed up and slow down.
- **View Transformation:** Press **1** or **2** or **3** to change the camera view from top, front and side.
- **Full Screen:** Press **F** or **f** to enter the full screen mode.
- **Exit:** Press **E** or **e** or **Escape** to exit the program.

Here marks the end of the 3D modelling project report.