Name: Jing Wang
ID: 21330849

Assignment 1

Module: CS7GV5
February 11, 2022

1 Youtube link

https://www.youtube.com/watch?v=KdTAo2uqvpg.

2 Required feature: Euler angle rotation

The rotation transformation was implemented using Euler angles. The following code snippet shows how a transformation matrix is created. The $euler_p$ is the pitch rotation angle, the $euler_y$ is the yaw rotation angle, the $euler_r$ is the roll rotation angle.

```
glm::mat4 trans = glm::scale(glm::mat4(1.0f), glm::vec3(1.0f, 1.0f, 1.0f))
    ; // the transformation matrix
trans = glm::rotate(trans, glm::radians(euler_r), glm::vec3(0.0f, 0.0f,
    1.0f));
trans = glm::rotate(trans, glm::radians(euler_y), glm::vec3(0.0f, 1.0f,
    0.0f));
trans = glm::rotate(trans, glm::radians(euler_p), glm::vec3(1.0f, 0.0f,
    0.0f));
```

Figure 1 to 4 shows the rotation using Euler angles. However, Euler angles will meet gimbal lock (see Figure 5). When we rotate the yaw angle until the rotation direction of roll aligns to the direction of pitch, gimbal lock happens.

Figure 1: Plane (no rotation).

Figure 2: Rotation (roll angle).

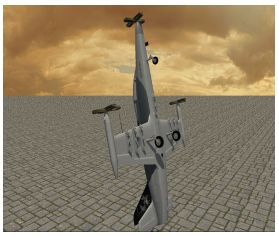

Figure 3: Rotation (yaw angle).

Figure 4: Rotation (pitch angle).

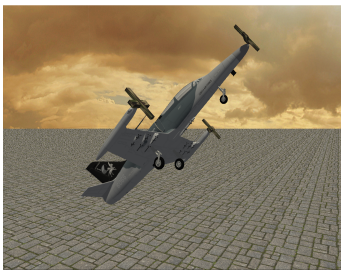

Figure 5: Gimbal lock. Rotations on roll angle (left) and pitch angle (right) are the same direction.

3 Extra feature 1: quaternion rotation

To overcome gimbal lock of using Euler angles, I used quaternion for rotation. After using quaternion, the plane can always rotate following the inputs of the viewer (please watch the video). The point of implementing quaternion is to memory the final quaternion and the next rotation should be left multiplied to the recorded quaternion to update the final quaternion. The following code shows how I implemented quaternion for this assignment.

```
glm::quat quaternion = glm::angleAxis(glm::radians(0.0f), glm::vec3(1., 0., 0.));
float old_eulerY = euler_y;
float old_eulerR = euler_r;
float old_eulerP = euler_p;
float delta_p = 0;
float delta_y = 0;
float delta_r = 0;

// the while loop is the loop to draw every frame
while (!glfwWindowShouldClose(window))
    delta_p = euler_p - old_eulerP;
    delta_y = euler_y - old_eulerY;
    delta_r = euler_r - old_eulerR;

    glm::vec3 rotationAxisX = glm::vec3(glm::toMat4(quaternion) * glm::vec4(1.0f, 0.0f, 0.0f, 1.0f)); // find the axis to rotate around
    glm::quat rot1 = glm::angleAxis(glm::radians(delta_p), rotationAxisX);
    quaternion = rot1 * quaternion; // update the final quaternion

    glm::vec3 rotationAxisY = glm::vec3(glm::toMat4(quaternion) * glm::vec4(0.0f, 1.0f, 0.0f, 1.0f)); // find the axis to rotate around
    glm::quat rot2 = glm::angleAxis(glm::radians(delta_y), rotationAxisY);
    quaternion = rot2 * quaternion; // update the final quaternion

    glm::vec3 rotationAxisZ = glm::vec3(glm::toMat4(quaternion) * glm::vec4(0.0f, 0.0f, 1.0f, 1.0f)); // find the axis to rotate around
    glm::quat rot3 = glm::angleAxis(glm::radians(delta_r), rotationAxisZ);
    quaternion = rot3 * quaternion; // update the final quaternion

    old_eulerP = euler_p;
    old_eulerY = euler_y;
    old_eulerR = euler_r;

    trans = glm::toMat4(quaternion);
```

4 Extra feature 2: hierarchical moving element

In addition to the plane model, three propellers were added. When rotating the plane, the propellers will be rotated following the rotation of the plane, meanwhile the propellers have their own rotations (Figure 6). The following code snippet shows the hierarchical moving.

```
glm::mat4 trans_leftPropeller = glm::translate(trans, glm::vec3(4.5f,
    0.095f, 1.85f)); // this transformation matrix is for the left
    propeller of the plane // third translate it back to its own position
trans_leftPropeller = glm::rotate(trans_leftPropeller, glm::radians(360.0f
    * (currentTime - startTime)), glm::vec3(0.0f, 0.0f, 1.0f)); // second
    rotate it
trans_leftPropeller = glm::translate(trans_leftPropeller, glm::vec3(-4.5f,
    -0.095f, -1.85f)); // first translate it to the origin
```

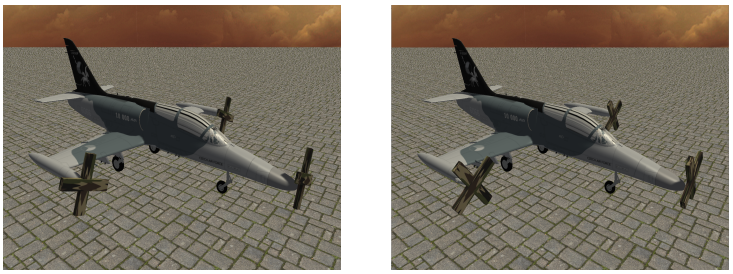

Figure 6: Hierarchical moving of the plane and its three propellers.

5 Extra feature 3: two camera view

Two camera view modes were implemented for this assignment. One is third person view (Figure 7, left). We can move the camera position to forward/back/left/right/up/down by pressing W/S/A/D/E/Q.

Another one is first person view (Figure 7, right). The view changes corresponding to the rotation of the plane.

6 Extra feature 4: better visual appearance

To achieve better visual appearance, a plane model is used for assignment instead of a simple cylinder. The rendering is using Blinn-Phong rendering model. You can see the diffuse and specular reflection. To enhance the feeling of the depth order of objects in the environment relative to the viewer, shadows were rendered using percentage closer filtering. On top of that, sky box was also implemented using cube map. Figure 8 shows the rendering results. Figure 9 shows the skybox.

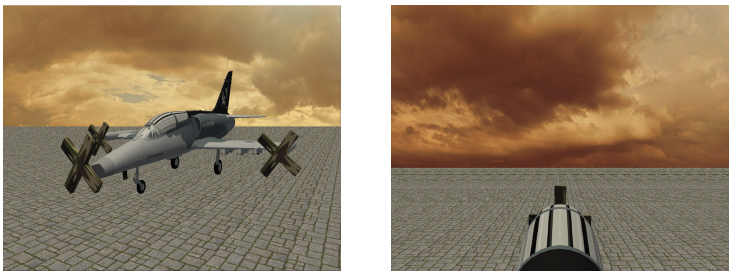

Figure 7: Two view modes. The left one is third person view, while the right one is first-person view.

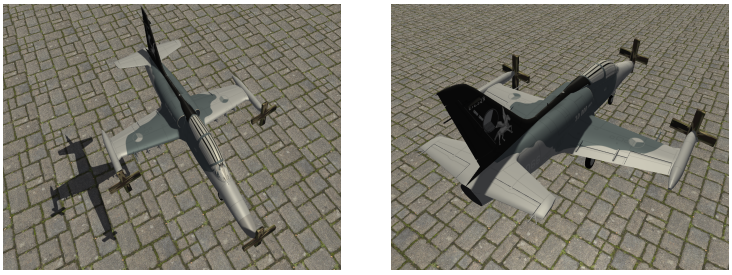

Figure 8: Blinn-Phong rendering and shadows.

Figure 9: Skybox.