HW1_Report_109550017

以下將本次作業分為Camera、Plane Rendering與Motion三個部分進行介紹。

Camera

Viewing transformation

在`Camera::updateViewMatrix`中，使用`newFront`與`newUp`計算轉向後相機的front與up向量，並用`glm::cross`計算右向量，接著用`glm::lookAt`計算新的viewMatrix。

```
glm::vec3 newFront = rotation * original_front;
glm::vec3 newUp = rotation * original_up;

right = glm::cross(newFront, newUp);

viewMatrix = glm::lookAt(position, position + newFront, newUp);
```

Projection transformation

在`Camera::updateProjectionMatrix`中，使用設定好的FOV、zNear、zFar透過`glm::perspective`計算projectionMatrix。

```
constexpr float FOV = glm::radians(45.0f);
constexpr float zNear = 0.1f;
constexpr float zFar = 100.0f;

projectionMatrix = glm::perspective(FOV, aspectRatio, zNear, zFar);
```

Plane Rendering

我將飛機的rendering分為四個部分，分別為機身(`draw_body`)、右機翼(`draw_right_wing`)、左機翼(`draw_left_wing`)與機尾(`drawTail`)，如下方code所示。

```
void drawAirplaneComponents() {
  draw_body();
  draw_right_wing();
  draw_left_wing();
  drawTail();
}
```

Body Rendering

機身的rendering由`draw_body`完成，需要建立圓柱體的表面與圓柱體前後方的兩個圓形。

機身圓柱面

由於無法在OpenGL繪製真正的圓柱面，我們使用`glBegin(GL_QUAD_STRIP)`在OpenGL宣告一個四邊形條帶，以四邊形條帶的方式去逼近圓柱面，其中，該圓柱體擁有64個四邊形條帶，並把顏色設為藍色。

接著以創建64個四邊形條帶為目標，沿+z軸旋轉的方向，繪製65個邊，便能繪製圓柱體。

```
/*宣告接下來將繪製GL_QUAD_STRIP*/
glBegin(GL_QUAD_STRIP);
/*宣告顏色為藍色*/
glColor3f(BLUE);

/*高為4 半徑為0.5*/
float height = 4.0f;
float radius = 0.5f;

/*開始繪製圓柱面*/
for (int i = 0; i <= CIRCLE_SEGMENT; i++) {

    /*繪製當前邊的角度*/
    float angle = 2.0f * M_PI * (static_cast<float>(i) / 64.0f);

    /*找出兩端點連線*/
    float x1 = radius * cos(angle);
    float y1 = radius * sin(angle);
    float z1 = height / 2.0f;

    float x2 = radius * cos(angle);
    float y2 = radius * sin(angle);
    float z2 = -height / 2.0f;

    y1 += radius;
    y2 += radius;

    /*當前邊的法向量*/
```

```
    float nx = x1;
    float ny = y1;
    float nz = 0.0f;

    /*定義法向量與沿線兩端點*/
    glNormal3f(nx, ny, nz);
    glVertex3f(x1, y1, z1);
    glVertex3f(x2, y2, z2);
}
glEnd();
```

機身圓形

圓柱體兩端由圓形組成，由於無法在OpenGL繪製真正的圓形，我們使用`GL_TRIANGLE_FAN`在OpenGL宣告三角形扇形，以多個三角形扇形的方式去逼近圓形，其中，該圓形擁有64個三角形扇形，並把顏色設為藍色。

接著以創建64個三角形扇形為目標，沿+z軸旋轉的方向，繪製65個邊，便能繪製圓柱體。

> 面對使用者的圓與背對使用者的圓在算法上相同，不過須注意，背對使用者的圓在創建vertex時，不同於大部分面向使用者的物體，因其法向量與旋轉軸相反，須以沿**旋轉軸順時鐘**的方向繪製，而非逆時針，才能使物體出現在螢幕中。

```
/*宣告接下來將繪製GL_TRIANGLE_FAN*/
glBegin(GL_TRIANGLE_FAN);
/*宣告顏色為藍色*/
glColor3f(BLUE);

/*開始繪製面對使用者的圓形*/
for (int i = 0; i <= CIRCLE_SEGMENT; i++) {

  /*當前邊的對應角度*/
  float angle = 2.0f * M_PI * (static_cast<float>(i) / CIRCLE_SEGMENT);

  /*端點位置*/
  float x = radius * cos(angle);
  float y = radius * sin(angle);
  float z = height / 2.0f;
  y += radius;

  /*圓的對應法向量*/
  float nx = 0.0f;
  float ny = 0.0f;
```

```
  float nz = 1.0f;

  /*定義邊的法向量與端點*/
  glNormal3f(nx, ny, nz);
  glVertex3f(x, y, z);
}
glEnd();
```

Wing Rendering

機翼由四方體組成，共有6個面，我將四方體頂面的左下角頂點設為基準點，以該點配合長、寬、高延伸其他點的位置。

每個面的繪製邏輯相同，左機翼與右機翼的繪製邏輯也相同，以`GL_QUADS`宣告繪製四邊形後，定義法向量，並以`glVertex3f`定義四方體的四個頂點。

> 須注意四方體底面、背面、左面因法向量與旋轉軸相反，因此要以沿**旋轉軸順時鐘**方向才能使物體出現在畫面中。

```
/*四方體的長、寬、高*/
float length = 4.0f;
float width = 1.0f;
float height = 0.5f;

/*圓柱體的半徑*/
/*因機翼(四方體)附著在機身(圓柱體)上，須考慮圓柱體半徑長*/
float cylinder_radius = 0.5f;

/*基準點，其他點會以本點配合長寬高做延伸*/
float x = 0.0f;
float y = cylinder_radius + height/2.0f;
float z = width/2.0f;

/*四邊形法向量預設值*/
float normalX = 1.0f;
float normalY = 1.0f;
float normalZ = 1.0f;


/***頂面***/
glBegin(GL_QUADS); /*宣告四邊形*/
glColor3f(RED); /*定義四邊形為紅色*/
glNormal3f(0.0f, normalY, 0.0f); /*定義四邊形法向量*/

/*定義四邊形頂點*/
/*定義方向沿+Y軸逆時鐘*/
glVertex3f(x, y, z); /*四邊形左下角*/
glVertex3f(x + length, y, z); /*四邊形右下角*/
```

```
glVertex3f(x + length, y, z - width); /*四邊形右上角*/
glVertex3f(x, y, z - width); /*四邊形左上角*/
glEnd(); /*結束四邊形宣告*/
```

Tail Rendering

機尾為四面體，四面體由三角形組成，以`GL_TRIANGLES`宣告完三角形後，設定顏色與法向量，並依據法向量使否與旋轉軸方向相同，決定需順時鐘或逆時鐘繪製頂點。此外，由於四面體的左右面並非垂直旋轉軸，法向量須由頂點求出平面上的兩個向量做外積得到。

```
/*圓柱體的半徑*/
/*因機尾(四面體)附著在機身(圓柱體)上，須考慮圓柱體半徑長*/
float cylinder_radius = 0.5f;

/*四面體邊長與三角形高*/
float bottomEdge = 2.0f;
float height1 = 1.0f;
float height2 = 0.5f;

/*基準點，其他點會以本點配合長與高做延伸*/
float x = 0.0f;
float y = cylinder_radius;
float z = 2.0f;

/*四邊形法向量預設值*/
float normalX = 1.0f;
float normalY = 1.0f;
float normalZ = 1.0f;

/***頂面***/
glBegin(GL_TRIANGLES); /*宣告三角形*/
glColor3f(GREEN); /*定義三角形為綠色*/
glNormal3f(0.0f, normalY, 0.0f); /*定義四邊形法向量*/

/*定義三角形的點*/
glVertex3f(x, y, z);
glVertex3f(x - bottomEdge / 2, y, z + height1);
glVertex3f(x + bottomEdge / 2.0f, y, z + height1);
glEnd(); /*結束三角形宣告*/
...
/***背面***/
glBegin(GL_TRIANGLES); /*宣告三角形*/
glColor3f(GREEN); /*定義三角形為綠色*/
glNormal3f(0.0f, 0.0f, normalZ); /*定義四邊形法向量*/

/*定義三角形的點*/
glVertex3f(x + bottomEdge / 2.0f, y, z + height1);
glVertex3f(x - bottomEdge / 2, y, z + height1);
```

```
glVertex3f(x, y - height2, z + height1);
glEnd(); /*結束三角形宣告*/
...
/***右面***/
glBegin(GL_TRIANGLES); /*宣告三角形*/
glColor3f(GREEN); /*定義三角形為綠色*/
...
/*定義三角形法向量*/
glm::vec3 vec1(0, -height2, height1); /*平面上第一個向量*/
glm::vec3 vec2(bottomEdge / 2.0f, 0.0f, height1); /*平面上第二個向量*/
glm::vec3 normal = glm::cross(vec1, vec2); /*平面上兩個向量做外積*/
glNormal3f(normal[0], normal[1], normal[2]); /*定義法向量*/

/*定義三角形的點*/
glVertex3f(x, y, z);
glVertex3f(x + bottomEdge / 2.0f, y, z + height1);
glVertex3f(x, y - height2, z + height1);
glEnd(); /*結束三角形宣告*/
```

Motion

飛機的動作包含三個，分別為機翼自動旋轉、飛機旋轉與飛機移動。

Wing Rotate

- 啟動方式：當使用者按住`空白鍵`不放，機翼即自動旋轉顯示拍打動作，放開空白鍵則停止。

將Wing Rotate定義在`drawAirplaneComponents`中，使得旋轉矩陣對機翼的影響早於整體飛機的控制。在`drawAirplaneComponents`中，首先透過`glPushMatrix`在Model-view Matrix上進行Rotation，其中，

透過使用 `glPopMatrix()` 恢復到先前保存的狀態。這樣可以防止不同的變換相互干擾，並使您能夠更輕鬆地管理複雜的場景中的多個物體的變換

```
void drawAirplaneComponents() {
...
  /*將旋轉套用在model-view matrix上*/
  glPushMatrix();

  /*使用者按下空白鍵*/
  if (spaceKeyPressed) {

    /*當機翼旋轉超過20度*/
    if (wing_angle > 20 || wing_angle < -20) {
      wing_angle -= wing_speed; /*減少旋轉角度*/
```

```
      wing_speed = -wing_speed; /*將wing_speed加上負號(與原先角度改變方向相反)
                                  以用於下方的wing_angle += wing_speed;*/
    }

    /*當機翼旋轉在20度內*/
    glRotatef(wing_angle, 0.0f, 0.0f, 1.0f); /*讓機翼旋轉wing_angle*/
    wing_angle += wing_speed; /*改變旋轉角度*/
  }
  draw_right_wing(); /*render右機翼*/
  glPopMatrix(); /*恢復到先前保存的狀態*/
...
}
```

Plane Rotate

飛機旋轉的邏輯在X軸、Y軸、Z軸上相同，都是建立旋轉矩陣套用在前進向量上，以改變位移方向。飛機本身旋轉則透過`glRotatef`依旋轉角度旋轉。

Pitch: X軸旋轉(Bonus)

- 啟動方式：
  - 飛機離地時，當使用者按住`向上鍵`不放，飛機即沿+X軸逆時鐘旋轉，放開向上鍵則停止。
  - 飛機離地時，當使用者按住`向下鍵`不放，飛機即沿+X軸順時鐘旋轉，放開向下鍵則停止。

將Pitch Rotation定義在`main`的while迴圈中，透過`glPushMatrix`在Model-view Matrix上進行Rotation。其中，`plane_dir`為飛機前進方向，當使用者按下`向上鍵`或`向下鍵`時，飛機會向上或向下旋轉；若放開`向上鍵`或`向下鍵`鍵，則飛機會往原先的前進方向墜落。

對機身做旋轉時，除了考慮機身本身的旋轉，還需考慮旋轉後飛機的行進方向，我使用`glRotatef`對飛機本身做旋轉，並將`rotationMatrix_x`作為旋轉矩陣，與目前的前進方向`plane_dir`相乘，對前進方向向量做旋轉，藉此改變未來Translation方向。

```
...
/*使用者按住向上鍵*/
else if (upKeyPressed) {
  /*旋轉矩陣*/
  glm::mat4 rotationMatrix_x = glm::rotate(glm::mat4(1.0f), glm::radians(angle),
                                           glm::vec3(1.0f, 0.0f, 0.0f));
  /*對前進向量進行旋轉*/
  plane_dir = glm::vec3(rotationMatrix_x * glm::vec4(plane_dir, 0.0f));
  /*更新當前轉向角*/
  rotate_x += angle;
}

/*使用者按住向下鍵*/
```

```
else if (rightKeyPressed) {
  /*旋轉矩陣*/
  glm::mat4 rotationMatrix_y = glm::rotate(glm::mat4(1.0f), glm::radians(-angle),
                                           glm::vec3(0.0f, 1.0f, 0.0f));
  /*對前進向量進行旋轉*/
  plane_dir = glm::vec3(rotationMatrix_y * glm::vec4(plane_dir, 0.0f));
  /*更新當前轉向角*/
  rotate_y -= angle;
}
...
glRotatef(rotate_x, 1.0f, 0.0f, 0.0f); /*對飛機進行旋轉*/
```

Yaw: Y軸旋轉

- 啟動方式：
  - 飛機離地時，當使用者按住`<`不放，飛機即沿+Y軸逆時鐘旋轉，放開向左鍵則停止。
  - 飛機離地時，當使用者按住`>`不放，飛機即沿+Y軸順時鐘旋轉，放開向右鍵則停止。

將Yaw Rotation定義在`main`的while迴圈中，透過`glPushMatrix`在Model-view Matrix上進行Rotation。其中，`plane_dir`為飛機前進方向，當使用者按下`<`或`>`時，飛機會向左或向右旋轉；若放開`<`或`>`鍵，則飛機會往原先的前進方向墜落。

對機身做旋轉時，除了考慮機身本身的旋轉，還需考慮旋轉後飛機的行進方向，我使用`glRotatef`對飛機本身做旋轉，並將`rotationMatrix_y`作為旋轉矩陣，與目前的前進方向`plane_dir`相乘，對前進方向向量做旋轉，藉此改變未來Translation方向。

```
/*定義前進方向向量初始值*/
glm::vec3 plane_dir(0.0f, 1.0f, -1.0f);
plane_dir = glm::normalize(plane_dir);
plane_dir = plane_dir * 0.05f;
...
/*當目前飛機離地*/
if (spaceKeyPressed || space_released) {
  /*使用者按住左鍵*/
  if (leftKeyPressed) {
    /*旋轉矩陣*/
    glm::mat4 rotationMatrix_y = glm::rotate(glm::mat4(1.0f), glm::radians(angle),
                                             glm::vec3(0.0f, 1.0f, 0.0f));
    /*對前進向量進行旋轉*/
    plane_dir = glm::vec3(rotationMatrix_y * glm::vec4(plane_dir, 0.0f));
    /*更新當前轉向角*/
    rotate_y += angle;
  }

  /*使用者按住右鍵*/
  else if (rightKeyPressed) {
```

```
    /*旋轉矩陣*/
    glm::mat4 rotationMatrix_y = glm::rotate(glm::mat4(1.0f), glm::radians(-angle),
                                             glm::vec3(0.0f, 1.0f, 0.0f));
    /*對前進向量進行旋轉*/
    plane_dir = glm::vec3(rotationMatrix_y * glm::vec4(plane_dir, 0.0f));
    /*更新當前轉向角*/
    rotate_y -= angle;
  }
...
glRotatef(rotate_y, 0.0f, 1.0f, 0.0f); /*對飛機進行旋轉*/
```

Row: Z軸旋轉(Bonus)

- 啟動方式：
  - 飛機離地時，當使用者按住`E`不放，飛機即沿+Z軸逆時鐘旋轉，放開E鍵則停止。
  - 飛機離地時，當使用者按住`R`不放，飛機即沿+Z軸順時鐘旋轉，放開R鍵則停止。

將Row Rotation定義在`main`的while迴圈中，透過`glPushMatrix`在Model-view Matrix上進行Rotation。其中，`plane_dir`為飛機前進方向，當使用者按下`E`或`R`時，飛機會向左或向右旋轉；若放開`E`或`R`鍵，則飛機會往原先的前進方向墜落。

對機身做旋轉時，除了考慮機身本身的旋轉，還需考慮旋轉後飛機的行進方向，我使用`glRotatef`對飛機本身做旋轉，並將`rotationMatrix_z`作為旋轉矩陣，與目前的前進方向`plane_dir`相乘，對前進方向向量做旋轉，藉此改變未來Translation方向。

```
/*使用者按住E鍵*/
else if (eKeyPressed) {
  /*旋轉矩陣*/
  glm::mat4 rotationMatrix_z = glm::rotate(glm::mat4(1.0f), glm::radians(angle),
                                           glm::vec3(0.0f, 0.0f, 1.0f));
  /*對前進向量進行旋轉*/
  plane_dir = glm::vec3(rotationMatrix_z * glm::vec4(plane_dir, 0.0f));
  /*更新當前轉向角*/
  rotate_z += angle;
}

/*使用者按住R鍵*/
else if (rKeyPressed) {
  /*旋轉矩陣*/
  glm::mat4 rotationMatrix_z = glm::rotate(glm::mat4(1.0f), glm::radians(-angle),
                                           glm::vec3(0.0f, 0.0f, 1.0f));
  /*對前進向量進行旋轉*/
  plane_dir = glm::vec3(rotationMatrix_z * glm::vec4(plane_dir, 0.0f));
  /*更新當前轉向角*/
  rotate_z -= angle;
}
```

```
...
glRotatef(rotate_z, 0.0f, 0.0f, 1.0f); /*對飛機進行旋轉*/
drawAirplaneComponents(); /*render飛機*/
glPopMatrix(); /*恢復到先前保存的狀態*/
```

Plane Translate

- 啟動方式：當使用者按住`空白鍵`不放，機身即往前方與上方移動，放開空白鍵則往前方與下方移動。

將Plane Translate定義在`main`的while迴圈中，使移動矩陣最後作用在飛機上。透過`glPushMatrix`在Model-view Matrix上進行Translation。其中，使用`plane_pos`紀錄飛機位置，`plane_dir`為飛機前進方向，當使用者按下空白鍵時，飛機會往前進方向移動；若放開空白鍵，則飛機會往原先的前進方向墜落。

```
/*定義前進方向向量初始值*/
glm::vec3 plane_pos(0.0f, 0.0f, 0.0f);
glm::vec3 plane_dir(0.0f, 1.0f, -1.0f);
plane_dir = glm::normalize(plane_dir);
plane_dir = plane_dir * 0.05f;

glPushMatrix(); /*將旋轉套用在model-view matrix上*/
glTranslatef(plane_pos.x, plane_pos.y, plane_pos.z); /*使飛機移動*/

/*按住空白鍵*/
if (spaceKeyPressed) {
  space_released = false;
  plane_pos += plane_dir; /*往前進方向前進*/
}
/*放開空白鍵*/
if (space_released) {
  float cur_y = plane_pos.y - plane_dir.y;
  if (cur_y <= 0) {
    plane_pos.y = 0;
    space_released = false;
  } else {
    /*往前進方向墜落*/
    plane_pos.x += plane_dir.x;
    plane_pos.y -= plane_dir.y;
    plane_pos.z += plane_dir.z;
  }
  ...
}
```

遇到的問題

- 依旋轉軸方向判斷頂點該順時鐘還是逆時鐘繪製，否則會無法顯示在畫面上。
- 法向量要在繪製頂點前設置，否則顏色會無法正常顯示。