# Computer Graphics Assignment 1

Student ID: 312551077
Name: 薛祖恩

## Implementation

1. TODO #1-1

```
void Camera::updateViewMatrix() {
    constexpr glm::vec3 original_front(0, 0, -1);
    constexpr glm::vec3 original_up(0, 1, 0);
    /* TODO#1-1: Calculate lookAt matrix
     *       1. Rotate original_front and original_up using this->rotation.
     *       2. Calculate right vector by cross product.
     *       3. Calculate view matrix with posision.
     * Hint:
     *       You can calculate the matrix by hand, or use
     *       glm::lookAt (https://glm.g-truc.net/0.9.9/api/a00247.html#gaa64aa951a0e99136bba9008d2b59c78e)
     * Note: You must not use gluLookAt
     */
    auto rotatedFront = this->rotation * original_front;
    auto rotatedUp = this->rotation * original_up;
    viewMatrix = glm::lookAt(position, position + rotatedFront, original_up);
    right = glm::normalize(glm::cross(rotatedUp, rotatedFront));
    front = rotatedFront;
    up = rotatedUp;
}
```

First, we calculate the new front and up, named by “rotatedFront” and “rotatedUp”. We can use the method glm::lookAt() to easily generate the view matrix. The last step is to update the value of right, front, up. We can get the camera right vector by calculating the cross product of front and up. These three variables are necessary to be updated since the camera is always moving and we want to get the current information of the camera.

2. TODO #1-2

```
void Camera::updateProjectionMatrix(float aspectRatio) {
    constexpr float FOV = glm::radians(45.0f);
    constexpr float zNear = 0.1f;
    constexpr float zFar = 100.0f;
    /* TODO#1-2: Calculate perspective projection matrix
     * Hint: You can calculate the matrix by hand, or use
     *       glm::perspective (https://glm.g-truc.net/0.9.9/api/a00243.html#ga747c8cf99458663dd7ad1bb3a2f07787)
     * Note: You must not use gluPerspective
     */
    projectionMatrix = glm::perspective(FOV, aspectRatio, zNear, zFar);
}
```

By using the method glm::perspective, we can easily generate the project matrix.

3. TODO #2

```
void drawCylinder() {
  // set height of the cylinder
  const float height = 4.0f;

  // Body of the cylinder
  glBegin(GL_QUAD_STRIP);
  const float radius = 0.5f;
  const float angle = 2.f * M_PI / CIRCLE_SEGMENT;
  for (int i = 0; i <= CIRCLE_SEGMENT; i++) {
    float x = radius * cos(i * angle);
    float y = radius * sin(i * angle);
    glNormal3f(x, 0, y);
    glVertex3f(x, -height / 2, y);
    glVertex3f(x, height / 2, y);
  }
  glEnd();

  // Top circle
  glBegin(GL_TRIANGLE_FAN);
  for (int i = 0; i <= CIRCLE_SEGMENT; i++) {
    float x = radius * cos(i * angle);
    float y = radius * sin(i * angle);
    glNormal3f(0, 1, 0);
    glVertex3f(-x, height / 2, y);
  }
  glEnd();

  // Bottom circle
  glBegin(GL_TRIANGLE_FAN);
  for (int i = 0; i <= CIRCLE_SEGMENT; i++) {
    float x = radius * cos(i * angle);
    float y = radius * sin(i * angle);
    glNormal3f(0, -1, 0);
    glVertex3f(x, -height / 2, y);
  }
  glEnd();
}

void airplaneBody() {
  glPushMatrix();
  glColor3f(BLUE);
  // translate y+0.5, since the bottom of the cylinder is (0, 0, 0)
  glTranslatef(0.0f, 0.5f, 0.0f);
  // rotate 90 degrees around x-axis
  glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
  drawCylinder();
  glPopMatrix();
}
```

The function “drawCylinder” and “airplaneBody” is a part of TODO#2, in “drawCylinder”, the basic concept is to separate the graph we want to render in many pieces, in “airplaneBody”, there are some set up for the cylinder, including translation on y since we want (0, 0, 0) be the point of the lower middle of the cylinder, and rotation since the original cylinder in “drawCylinder” is parallel to the y-axis.

```
void drawWings() {
    // avoid culling
    glDisable(GL_CULL_FACE);

    // translate and rotate for wings rotation
    glTranslatef(0.0, 0.25, 0.0);
    glRotatef(wingRotationAngle, 0.0, 0.0, 1.0);
    glBegin(GL_QUADS);

    glNormal3f(0.0, -1.0, 0.0);
    glVertex3f(0.0, 0.0, 0.5);
    glVertex3f(0.0, 0.0, -0.5);
    glVertex3f(-4.0, 0.0, -0.5);
    glVertex3f(-4.0, 0.0, 0.5);

    glNormal3f(0.0, 1.0, 0.0);
    glVertex3f(0.0, 0.5, 0.5);
    glVertex3f(-4.0, 0.5, 0.5);
    glVertex3f(-4.0, 0.5, -0.5);
    glVertex3f(0.0, 0.5, -0.5);

    glNormal3f(0.0, 0.0, 1.0);
    glVertex3f(0.0, 0.0, 0.5);
    glVertex3f(0.0, 0.5, 0.5);
    glVertex3f(-4.0, 0.5, 0.5);
    glVertex3f(-4.0, 0.0, 0.5);

    glNormal3f(0.0, 0.0, -1.0);
    glVertex3f(0.0, 0.0, -0.5);
    glVertex3f(-4.0, 0.0, -0.5);
    glVertex3f(-4.0, 0.5, -0.5);
    glVertex3f(0.0, 0.5, -0.5);

    glNormal3f(-1.0, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.5);
    glVertex3f(0.0, 0.5, 0.5);
    glVertex3f(0.0, 0.5, -0.5);
    glVertex3f(0.0, 0.0, -0.5);

    glNormal3f(1.0, 0.0, 0.0);
    glVertex3f(-4.0, 0.0, 0.5);
    glVertex3f(-4.0, 0.0, -0.5);
    glVertex3f(-4.0, 0.5, -0.5);
    glVertex3f(-4.0, 0.5, 0.5);

    glEnd();
```

This function is quite big, this is the first part of this function, and this is for the left wing (cuboid). It takes some time to find the correct parameter.

```
    // -2 since the left and right wings must be symmetric
    glRotatef(-2 * wingRotationAngle, 0.0, 0.0, 1.0);
    glBegin(GL_QUADS);

    glNormal3f(0.0, -1.0, 0.0);
    glVertex3f(4.0, 0.0, 0.5);
    glVertex3f(4.0, 0.0, -0.5);
    glVertex3f(0.0, 0.0, -0.5);
    glVertex3f(0.0, 0.0, 0.5);

    glNormal3f(0.0, 1.0, 0.0);
    glVertex3f(4.0, 0.5, 0.5);
    glVertex3f(0.0, 0.5, 0.5);
    glVertex3f(0.0, 0.5, -0.5);
    glVertex3f(4.0, 0.5, -0.5);

    glNormal3f(0.0, 0.0, 1.0);
    glVertex3f(4.0, 0.0, 0.5);
    glVertex3f(4.0, 0.5, 0.5);
    glVertex3f(0.0, 0.5, 0.5);
    glVertex3f(0.0, 0.0, 0.5);

    glNormal3f(0.0, 0.0, -1.0);
    glVertex3f(4.0, 0.0, -0.5);
    glVertex3f(0.0, 0.0, -0.5);
    glVertex3f(0.0, 0.5, -0.5);
    glVertex3f(4.0, 0.5, -0.5);

    glNormal3f(-1.0, 0.0, 0.0);
    glVertex3f(4.0, 0.0, 0.5);
    glVertex3f(4.0, 0.5, 0.5);
    glVertex3f(4.0, 0.5, -0.5);
    glVertex3f(4.0, 0.0, -0.5);

    glNormal3f(1.0, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.5);
    glVertex3f(0.0, 0.0, -0.5);
    glVertex3f(0.0, 0.5, -0.5);
    glVertex3f(0.0, 0.5, 0.5);
    glEnd();
}

void airplaneWings() {
    glPushMatrix();
    glColor3f(RED);
    drawWings();
    glPopMatrix();
}
```

This is the second part of this function, and this is for the right wing (cuboid). It takes some time to find the correct parameter. The function “airplaneWings” setup the color of the wings.

```
void drawTail() {
    glBegin(GL_TRIANGLES);

    // four points: (0, 0.5, 2), (-1, 0.5, 3), (1, 0.5, 3), (0, 0, 3)
    glVertex3f(0.0f, 0.5f, -2.0f);
    glVertex3f(-1.0f, 0.5f, -3.0f);
    glVertex3f(1.0f, 0.5f, -3.0f);

    glVertex3f(0.0f, 0.5f, -2.0f);
    glVertex3f(-1.0f, 0.5f, -3.0f);
    glVertex3f(0.0f, 0.0f, -3.0f);

    glVertex3f(0.0f, 0.5f, -2.0f);
    glVertex3f(1.0f, 0.5f, -3.0f);
    glVertex3f(0.0f, 0.0f, -3.0f);

    glVertex3f(-1.0f, 0.5f, -3.0f);
    glVertex3f(1.0f, 0.5f, -3.0f);
    glVertex3f(0.0f, 0.0f, -3.0f);

    glEnd();
}

void airplaneTail() {
    glPushMatrix();
    glColor3f(GREEN);
    drawTail();
    glPopMatrix();
}

void drawAirplane() {
    airplaneBody();
    airplaneWings();
    airplaneTail();
}
```

There are four points in the Tetrahedron, in this case, that is: (0, 0.5, 2), (-1, 0.5, 3), (1, 0.5, 3), (0, 0, 3). Then we do an easy set up at “airplaneTail”, setup the color of the tail, and then call three functions to start drawing.

4. TODO #3

```
/* TODO#3: Render the airplane
*       1. Render the body.
*       2. Render the wings.(Don't forget to assure wings rotate at the center of body.)
*       3. Render the tail.
* Hint:
*       glPushMatrix/glPopMatrix (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glPushMatrix.xml)
*       glRotatef (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glRotate.xml)
*       glTranslatef (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glTranslate.xml)
*       glColor3f (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glColor.xml)
*       glScalef (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glScale.xml)
* Note:
*       You may implement functions for drawing components of airplane first
*       You should try and think carefully about changing the order of rotate and translate
*/
// rotate first then translate
glTranslatef(airplanePosition.x, airplanePosition.y, airplanePosition.z);
glRotated(rotateValue, 0.0, 1.0, 0.0);
drawAirplane();
```

I calculated four different variables: airplanePosition.x, y, z, and rotateValue, the way to calculate them are shown in the next two TODO sections.

5. TODO #4-1

```
int changeStatus(int attribute) { return (attribute + 1) % 2; }

void keyCallback(GLFWwindow* window, int key, int, int action, int) {
  // There are three actions: press, release, hold(repeat)
  if (action == GLFW_REPEAT) return;

  // Press ESC to close the window.
  if (key == GLFW_KEY_ESCAPE) {
    glfwSetWindowShouldClose(window, GLFW_TRUE);
    return;
  }
  /* ... */
  if (action == GLFW_PRESS) {
    switch (key) {
      case GLFW_KEY_SPACE:
        IsFlyingUp = changeStatus(IsFlyingUp);
        IsFlyingForward = changeStatus(IsFlyingForward);
        break;
      case GLFW_KEY_LEFT:
        rotateValue += ROTATE_SPEED;
        break;
      case GLFW_KEY_RIGHT:
        rotateValue -= ROTATE_SPEED;
        break;
    }
  }
}
```

The function “changeStatus” is similarly to a lock for variables. It will have different process according to the value “isFlyingUp” and “isFlyingForward”, I will mention in the next TODO section. When the left and right key is clicked, increase and decrease the value “rotateValue”, respectively.

6. TODO #4-2

First, I would like to introduce the global variables.

```
struct Position {
    float x;
    float y;
    float z;
} airplanePosition, rotatePosition;

int IsFlyingUp = 0;
int IsFlyingForward = 0;
float rotateValue = 0;
float wingRotationAngle = 0;
```

There are two positions used in this program, airplanePosition and rotatePosition, which determines the rotation and the translation of the airplane. IsFlyingUp and IsFlyingForward is the value that records the current state of the airplane. rotateValue and wingRotateAngle is the variable to store the value related to rotation of the airplane and the wing rotation of the airplane.

```
// if the left or the right key is pressed down, keep rotating
if (glfwGetKey(window, GLFW_KEY_LEFT) == GLFW_PRESS) {
    rotateValue += ROTATE_SPEED;
} else if (glfwGetKey(window, GLFW_KEY_RIGHT) == GLFW_PRESS) {
    rotateValue -= ROTATE_SPEED;
}
rotatePosition.x = std::sin(ANGLE_TO_RADIAN(rotateValue));
rotatePosition.z = std::cos(ANGLE_TO_RADIAN(rotateValue));
// set up value for flyingUp, flyingForward, wingRotationAngle
if (glfwGetKey(window, GLFW_KEY_SPACE) == GLFW_RELEASE) {
    airplanePosition.y -= FLYING_SPEED;
    if (airplanePosition.y <= 0) {
        airplanePosition.y = 0;
    }
} else {
    airplanePosition.x += rotatePosition.x * 0.3;
    airplanePosition.y += FLYING_SPEED;
    airplanePosition.z += rotatePosition.z * 0.3;
    wingRotationAngle = 20.0f * sin(glfwGetTime() * 5.0);
}
// Render a white board
```

We check if the left key and right key is pressed, the rotate value will increase or decrease depends on what the user press. Then we calculate the rotate position of the airplane determined by the rotate value. The last part is to check if the space key is released. If the user released the key, the y of the airplane position should decrease to simulate gravity. If the user didn't release the space key, then the airplane position and

wing rotate angle must be updated.

**Problems Encountered**

1. Remote control to the computer might cannot debug the function related to the mouse, the value of dx, dy is not as expected.
2. Not familiar with the view matrix, so I spend some time implementing the function “updateViewMatrix” in camera.cpp.
3. About the wing rotation, in the function drawWings(), for the right wing, we need to rotate “wingRotateAngle” × 2, since if we only rotate 1 of it, the wing will stay instead of moving because the rotation discharged.
4. It takes some time to create the body, wings, and tail of the airplane.

**Suggestions**

There is no suggestion from me.