camera.cpp

```
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
     */
    right = glm::vec3(0, 0, 0);
    viewMatrix = glm::identity<glm::mat4>();
    front = glm::normalize(glm::vec3(0, 0, 0) - position);
    right = glm::normalize(glm::cross(front, original_up));
    viewMatrix = glm::lookAt(position, glm::vec3(0, 0, 0), up);
}
```

front 設定為 camera 的前方並 normalize 降低 camera 移動速度

right 為 front 和 up 的 cross product

viewMatrix 用 lookat(camera 位置, 看向的位置, up)算出

```
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::identity<glm::mat4>();
    projectionMatrix = glm::perspective(FOV, aspectRatio, zNear, zFar);
}
```

projecttionMatrix 用 perspective 算出(照 hint 網址給的參數輸入就好)

main.cpp

```
    GLFWwindow* window = OpenGLContext::getWindow();
    /* TODO#0: Change window title to "HW1 - `your student id`"
     *         Ex. HW1 - 311550000
     */
    glfwSetWindowTitle(window, "HW1 - 311551144");
    glfwSetKeyCallback(window, keyCallback);
    glfwSetFramebufferSizeCallback(window, resizeCallback);
    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
```

改學號

```
void drawUnitCylinder() {
    /* TODO#2-1: Render a unit cylinder
     * Hint:
     *       glBegin/glEnd (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glBegin.xml)
     *       glColor3f (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glBegin.xml)
     *       glVertex3f (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glVertex.xml)
     *       glNormal (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glNormal.xml)
     *       glScalef (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glScale.xml)
     * Note:
     *       You can refer to ppt "Draw Cylinder" page and `CIRCLE_SEGMENT`
     *       You should set normal for lighting
     */

    float innerAngle = 360.f;
    float innerRadian;

    glPushMatrix();
    glTranslatef(target_pos.x, target_pos.y, target_pos.z);
    glScalef(TARGET_DIAMETER/2, TARGET_HEIGHT, TARGET_DIAMETER/2);
    glBegin(GL_TRIANGLE_STRIP);
    glColor3f(RED);
    for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
        innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
        float z = cos(innerRadian);
        float x = sin(innerRadian);
        innerRadian = ANGEL_TO_RADIAN(innerAngle * (i - 0.5) / CIRCLE_SEGMENT);
        float nz = cos(innerRadian);
        float nx = sin(innerRadian);
        glNormal3f(nx, 0.0f, nz);
        glVertex3f(x, 0.5f, z);
        glVertex3f(x, -0.5f, z);
    }
    glEnd();
    glPopMatrix();
```

innerAngle/ innerRadian為多邊形中心對頂點的角度(正方形即0、90、180、270、360度)用來計算各頂點的位置

畫target外環順序

1. Loop畫不同頂點的三角形
2. 縮放物件
3. 轉移物件

```
glPushMatrix();
glTranslatef(target_pos.x, target_pos.y, target_pos.z);
glScalef(TARGET_DIAMETER / 2, TARGET_HEIGHT, TARGET_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(0.0f, 1.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float x = sin(innerRadian);
    glVertex3f(x, 0.5f, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glTranslatef(target_pos.x, target_pos.y , target_pos.z);
glScalef(TARGET_DIAMETER / 2, TARGET_HEIGHT, TARGET_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(0.0f, -1.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float x = sin(innerRadian);
    glVertex3f(x, -0.5f, z);
}
glEnd();
glPopMatrix();
```

畫target上/下蓋順序

1. Loop畫多邊形
2. 縮放物件
3. 轉移物件

```
drawUnitCylinder();
/* TODO#2: Render a cylinder at target_pos
*       1. Translate to target_pos
*       2. Setup vertex color
*       3. Setup cylinder scale
*       4. Call drawUnitCylinder
* Hint:
*       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:
*       The coordinates of the cylinder are `target_pos`
*       The cylinder's size can refer to `TARGET_RADIUS`, `TARGET_DIAMETER` and `TARGET_DIAMETER`
*       The cylinder's color can refer to `RED`
*/
```

呼叫drawUnitCylinder

```
/* TODO#3: Render the robotic arm
 *       1. Render the base
 *       2. Translate to top of the base
 *       3. Render an arm
 *       4. Translate to top of the arm
 *       5. Render the joint
 *       6. Translate and rotate to top of the join
 *       7. Repeat step 3-6
 * Hint:
 *       glPushMatrix/glPopMatrix (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/
 *       glRotatef (https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glRotate.xml)
 * Note:
 *       The size of every component can refer to `Components size definition` section
 *       Rotate degree for joints are `joint0_degree`, `joint1_degree` and `joint2_degree`
 *       You may implement drawBase, drawArm and drawJoin first
 */

float innerAngle = 360.f;
float innerRadian;

//BASE=============================================================
glPushMatrix();
glScalef(BASEE_DIAMETER / 2, BASE_HEIGHT, BASEE_DIAMETER / 2);
glBegin(GL_TRIANGLE_STRIP);
glColor3f(GREEN);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float x = sin(innerRadian);
    innerRadian = ANGEL_TO_RADIAN(innerAngle * (i - 0.5) / CIRCLE_SEGMENT);
    float nz = cos(innerRadian);
    float nx = sin(innerRadian);
    glNormal3f(nx, 0.0f, nz);
    glVertex3f(x, 1.0f, z);
    glVertex3f(x, 0.0f, z);
}
glEnd();
glPopMatrix();
```

畫BASE外環順序

1. Loop畫不同頂點的三角形
2. 縮放物件

```
glPushMatrix();
glScalef(BASEE_DIAMETER / 2, BASE_HEIGHT, BASEE_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(0.0f, 1.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float x = sin(innerRadian);
    glVertex3f(x, 1.0f, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glScalef(BASEE_DIAMETER / 2, BASE_HEIGHT, BASEE_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(0.0f, -1.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float x = sin(innerRadian);
    glVertex3f(x, 0.0f, z);
}
glEnd();
glPopMatrix();
```

畫BASE上/下蓋順序

1. Loop畫多邊形
2. 縮放物件

```
// Arm1========================================================
glPushMatrix();
glTranslatef(0.0f, BASE_HEIGHT, 0.0f);
glScalef(ARM_DIAMETER / 2, ARM_LEN, ARM_DIAMETER/ 2);
glBegin(GL_TRIANGLE_STRIP);
glColor3f(BLUE);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float x = sin(innerRadian);
    innerRadian = ANGEL_TO_RADIAN(innerAngle * (i - 0.5) / CIRCLE_SEGMENT);
    float nz = cos(innerRadian);
    float nx = sin(innerRadian);
    glNormal3f(nx, 0.0f, nz);
    glVertex3f(x, 1.0f, z);
    glVertex3f(x, 0.0f, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, BASE_HEIGHT, 0.0f);
glScalef(ARM_DIAMETER / 2, ARM_LEN, ARM_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(0.0f, 1.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float x = sin(innerRadian);
    glVertex3f(x, 1.0f, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, BASE_HEIGHT, 0.0f);
glScalef(ARM_DIAMETER / 2, ARM_LEN, ARM_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(0.0f, -1.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float x = sin(innerRadian);
    glVertex3f(x, 0.0f, z);
}
glEnd();
glPopMatrix();
```

畫ARM1外環與上/下蓋順序

1. Loop畫不同頂點的三角形/多邊形
2. 縮放物件
3. 轉移物件到BASE上

```
// Joint1═══════════════════════════════════════════════
glPushMatrix();
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint0_degree, 0, 1, 0);
glScalef(JOINT_WIDTH, JOINT_DIAMETER / 2, JOINT_DIAMETER / 2);
glBegin(GL_TRIANGLE_STRIP);
glColor3f(GREEN);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float y = sin(innerRadian);
    innerRadian = ANGEL_TO_RADIAN(innerAngle * (i - 0.5) / CIRCLE_SEGMENT);
    float nz = cos(innerRadian);
    float ny = sin(innerRadian);
    glNormal3f(0.0f, ny, nz);
    glVertex3f(-0.5f, y, z);
    glVertex3f(0.5f, y, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint0_degree, 0, 1, 0);
glScalef(JOINT_WIDTH, JOINT_DIAMETER / 2, JOINT_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(1.0f, 0.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float y = cos(innerRadian);
    float z = sin(innerRadian);
    glVertex3f(0.5f, y, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint0_degree, 0, 1, 0);
glScalef(JOINT_WIDTH, JOINT_DIAMETER / 2, JOINT_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(-1.0f, 0.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float y = sin(innerRadian);
    glVertex3f(-0.5f, y, z);
}
glEnd();
glPopMatrix();
```

畫JOINT1外環與上/下蓋順序

1. Loop畫不同頂點的三角形/多邊形
2. 縮放物件
3. 旋轉物件(Y軸)
4. 轉移物件到ARM1上

```
// Arm2========================================
glPushMatrix();
glRotatef(joint0_degree, 0, 1, 0);
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint1_degree, 1, 0, 0);
glTranslatef(0.0f, JOINT_RADIUS, 0.0f);
glScalef(ARM_DIAMETER / 2, ARM_LEN, ARM_DIAMETER / 2);
glBegin(GL_TRIANGLE_STRIP);
glColor3f(BLUE);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
  innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
  float z = cos(innerRadian);
  float x = sin(innerRadian);
  innerRadian = ANGEL_TO_RADIAN(innerAngle * (i - 0.5) / CIRCLE_SEGMENT);
  float nz = cos(innerRadian);
  float nx = sin(innerRadian);
  glNormal3f(nx, 0.0f, nz);
  glVertex3f(x, 1.0f, z);
  glVertex3f(x, 0.0f, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glRotatef(joint0_degree, 0, 1, 0);
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint1_degree, 1, 0, 0);
glTranslatef(0.0f, JOINT_RADIUS, 0.0f);
glScalef(ARM_DIAMETER / 2, ARM_LEN, ARM_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(0.0f, 1.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
  innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
  float z = cos(innerRadian);
  float x = sin(innerRadian);
  glVertex3f(x, 1.0f, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glRotatef(joint0_degree, 0, 1, 0);
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint1_degree, 1, 0, 0);
glTranslatef(0.0f, JOINT_RADIUS, 0.0f);
glScalef(ARM_DIAMETER / 2, ARM_LEN, ARM_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(0.0f, -1.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
  innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
  float x = cos(innerRadian);
  float z = sin(innerRadian);
  glVertex3f(x, 0.0f, z);
}
glEnd();
glPopMatrix();
```

畫ARM2外環與上/下蓋順序

1. Loop畫不同頂點的三角形/多邊形
2. 縮放物件
3. 轉移物件(joint半徑)
4. 旋轉物件(X軸)
5. 轉移物件到JOINT1上
6. 旋轉物件(Y軸)

```
// Joint2=========================================
glPushMatrix();
glRotatef(joint0_degree, 0, 1, 0);
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint1_degree, 1, 0, 0);
glTranslatef(0.0f, ARM_LEN + JOINT_DIAMETER, 0.0f);
glRotatef(joint2_degree, 1, 0, 0);
glScalef(JOINT_WIDTH, JOINT_DIAMETER / 2, JOINT_DIAMETER / 2);
glBegin(GL_TRIANGLE_STRIP);
glColor3f(GREEN);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float y = sin(innerRadian);
    innerRadian = ANGEL_TO_RADIAN(innerAngle * (i - 0.5) / CIRCLE_SEGMENT);
    float nz = cos(innerRadian);
    float ny = sin(innerRadian);
    glNormal3f(0.0f, ny, nz);
    glVertex3f(-0.5f, y, z);
    glVertex3f(0.5f, y, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glRotatef(joint0_degree, 0, 1, 0);
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint1_degree, 1, 0, 0);
glTranslatef(0.0f, ARM_LEN + JOINT_DIAMETER, 0.0f);
glRotatef(joint2_degree, 1, 0, 0);
glScalef(JOINT_WIDTH, JOINT_DIAMETER / 2, JOINT_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(1.0f, 0.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float y = cos(innerRadian);
    float z = sin(innerRadian);
    glVertex3f(0.5f, y, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glRotatef(joint0_degree, 0, 1, 0);
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint1_degree, 1, 0, 0);
glTranslatef(0.0f, ARM_LEN + JOINT_DIAMETER, 0.0f);
glRotatef(joint2_degree, 1, 0, 0);
glScalef(JOINT_WIDTH, JOINT_DIAMETER / 2, JOINT_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(-1.0f, 0.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float y = sin(innerRadian);
    glVertex3f(-0.5f, y, z);
}
glEnd();
glPopMatrix();
```

畫JOINT2外環與上/下蓋順序

1. Loop畫不同頂點的三角形/多邊形
2. 縮放物件
3. 旋轉物件(X軸)
4. 轉移物件到ARM2上
5. 旋轉物件(X軸)
6. 轉移物件到JOINT1上
7. 旋轉物件(Y軸)

```
// Arm3============================================================
glPushMatrix();
glRotatef(joint0_degree, 0, 1, 0);
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint1_degree, 1, 0, 0);
glTranslatef(0.0f, ARM_LEN + JOINT_DIAMETER, 0.0f);
glRotatef(joint2_degree, 1, 0, 0);
glTranslatef(0.0f, JOINT_RADIUS, 0.0f);
glScalef(ARM_DIAMETER / 2, ARM_LEN, ARM_DIAMETER / 2);
glBegin(GL_TRIANGLE_STRIP);
glColor3f(BLUE);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float x = sin(innerRadian);
    innerRadian = ANGEL_TO_RADIAN(innerAngle * (i - 0.5) / CIRCLE_SEGMENT);
    float nz = cos(innerRadian);
    float nx = sin(innerRadian);
    glNormal3f(nx, 0.0f, nz);
    glVertex3f(x, 1.0f, z);
    glVertex3f(x, 0.0f, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glRotatef(joint0_degree, 0, 1, 0);
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint1_degree, 1, 0, 0);
glTranslatef(0.0f, ARM_LEN + JOINT_DIAMETER, 0.0f);
glRotatef(joint2_degree, 1, 0, 0);
glTranslatef(0.0f, JOINT_RADIUS, 0.0f);
glScalef(ARM_DIAMETER / 2, ARM_LEN, ARM_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(0.0f, 1.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float z = cos(innerRadian);
    float x = sin(innerRadian);
    glVertex3f(x, 1.0f, z);
}
glEnd();
glPopMatrix();

glPushMatrix();
glRotatef(joint0_degree, 0, 1, 0);
glTranslatef(0.0f, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS, 0.0f);
glRotatef(joint1_degree, 1, 0, 0);
glTranslatef(0.0f, ARM_LEN + JOINT_DIAMETER, 0.0f);
glRotatef(joint2_degree, 1, 0, 0);
glTranslatef(0.0f, JOINT_RADIUS, 0.0f);
glScalef(ARM_DIAMETER / 2, ARM_LEN, ARM_DIAMETER / 2);
glBegin(GL_POLYGON);
glNormal3f(0.0f, -1.0f, 0.0f);
for (float i = 0; i <= CIRCLE_SEGMENT; i += 1.0f) {
    innerRadian = ANGEL_TO_RADIAN(innerAngle * i / CIRCLE_SEGMENT);
    float x = cos(innerRadian);
    float z = sin(innerRadian);
    glVertex3f(x, 0.0f, z);
}
glEnd();
glPopMatrix();
```

畫ARM3外環與上/下蓋順序

1. Loop畫不同頂點的三角形/多邊形
7. 縮放物件
8. 轉移物件(joint半徑)
2. 旋轉物件(X軸)
3. 轉移物件到ARM2上
4. 旋轉物件(X軸)
5. 轉移物件到JOINT1上
6. 旋轉物件(Y軸)

```
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;
  }
  /* TODO#4-1: Detect key-events, perform rotation or catch target object
   *       1. Use switch/case to find the key you want.
   *       2. Define and modify some global variable to trigger update in rendering loop
   * Hint:
   *       glfw3's key list (https://www.glfw.org/docs/3.3/group__keys.html)
   *       glfw3's action codes (https://www.glfw.org/docs/3.3/group__input.html#gada11d965c4da13090ad336e030e4d11f)
   * Note:
   *       You should finish your robotic arm first.
   *       Otherwise you will spend a lot of time debugging this with a black screen.
   */
  switch (key) {
    case GLFW_KEY_U:
      joint0_degree += 10 * ROTATE_SPEED;
      break;
    case GLFW_KEY_J:
      joint0_degree -= 10 * ROTATE_SPEED;
      break;
    case GLFW_KEY_I:
      joint1_degree -= 10 * ROTATE_SPEED;
      break;
    case GLFW_KEY_K:
      joint1_degree += 10 * ROTATE_SPEED;
      break;
    case GLFW_KEY_O:
      joint2_degree -= 10 * ROTATE_SPEED;
      break;
    case GLFW_KEY_L:
      joint2_degree += 10 * ROTATE_SPEED;
      break;
    case GLFW_KEY_G:
      if (action == GLFW_RELEASE) {
        if (g_down) {
          printf("down disable\n");
        } else {
          printf("down enable\n");
        }
        g_down = !g_down;
      }
      break;
    case GLFW_KEY_SPACE:
      switch (action) {
        case GLFW_PRESS:
          space_down = true;
          break;
        case GLFW_RELEASE:
          space_down = false;
          break;
        default:
          break;
      }
      break;
    default:
      break;
  }
}
```

判別key為什麼按鍵然後改變joint_degree的值，空白鍵的畫就判定action是按下還是鬆開，然後再改變space_down的bool值

g_down是target是否墜落的bool值

```
/* TODO#4-2: Update joint degrees
 *       1. Finish keyCallback to detect key events
 *       2. Update jointx_degree if the correspond key is pressed
 * Note:
 *       You can use `ROTATE_SPEED` as the speed constant.
 *       If the rotate speed is too slow or too fast, please change `ROTATE_SPEED` value
 */

/* TODO#5: Catch the target object with robotic arm
 *       1. Calculate coordinate of the robotic arm endpoint
 *       2. Test if arm endpoint and the target object are close enough
 *       3. Update coordinate fo the target object to the arm endpoint
 *          if the space key is pressed
 * Hint:
 *       GLM fransform API (https://glm.g-truc.net/0.9.4/api/a00206.html)
 * Note:
 *       You might use `ANGEL_TO_RADIAN`
 *       and refer to `CATCH_POSITION_OFFSET` and `TOLERANCE`
 */
glm::vec4 arm_endpoint(0.0f, 0.0f, 0.0f, 1.0f);
glm::mat4 trasformMatrix_arm_endpoint(1.0f);

trasformMatrix_arm_endpoint = glm::translate(trasformMatrix_arm_endpoint, glm::vec3(0.0f, BASE_HEIGHT, 0.0f));
trasformMatrix_arm_endpoint = glm::rotate(trasformMatrix_arm_endpoint, ANGEL_TO_RADIAN(joint0_degree), glm::vec3(0.0f, 1.0f, 0.0f));
trasformMatrix_arm_endpoint = glm::translate(trasformMatrix_arm_endpoint, glm::vec3(0.0f, ARM_LEN, 0.0f));
trasformMatrix_arm_endpoint = glm::translate(trasformMatrix_arm_endpoint, glm::vec3(0.0f, JOINT_RADIUS, 0.0f));
trasformMatrix_arm_endpoint = glm::rotate(trasformMatrix_arm_endpoint, ANGEL_TO_RADIAN(joint1_degree), glm::vec3(1.0f, 0.0f, 0.0f));
trasformMatrix_arm_endpoint = glm::translate(trasformMatrix_arm_endpoint, glm::vec3(0.0f, JOINT_RADIUS, 0.0f));
trasformMatrix_arm_endpoint = glm::translate(trasformMatrix_arm_endpoint, glm::vec3(0.0f, ARM_LEN, 0.0f));
trasformMatrix_arm_endpoint = glm::translate(trasformMatrix_arm_endpoint, glm::vec3(0.0f, JOINT_RADIUS, 0.0f));
trasformMatrix_arm_endpoint = glm::rotate(trasformMatrix_arm_endpoint, ANGEL_TO_RADIAN(joint2_degree), glm::vec3(1.0f, 0.0f, 0.0f));
trasformMatrix_arm_endpoint = glm::translate(trasformMatrix_arm_endpoint, glm::vec3(0.0f, JOINT_RADIUS, 0.0f));
trasformMatrix_arm_endpoint = glm::translate(trasformMatrix_arm_endpoint, glm::vec3(0.0f, ARM_LEN, 0.0f));
trasformMatrix_arm_endpoint = glm::translate(trasformMatrix_arm_endpoint, glm::vec3(0.0f, CATCH_POSITION_OFFSET, 0.0f));

arm_endpoint = trasformMatrix_arm_endpoint * arm_endpoint;
float distance_target = powf(arm_endpoint.x - target_pos.x, 2.0f);
distance_target += powf(arm_endpoint.y - target_pos.y, 2.0f);
distance_target += powf(arm_endpoint.z - target_pos.z, 2.0f);
distance_target = sqrtf(distance_target);
if (space_down && distance_target <= TOLERANCE)
{
    target_pos = glm::vec3(arm_endpoint.x, arm_endpoint.y, arm_endpoint.z);
} else if (g_down && target_pos.y > 0) {
    target_pos.y = (target_pos.y - 0.005f < TARGET_HEIGHT / 2) ? TARGET_HEIGHT / 2 : target_pos.y - 0.005f;
}
```

從原點根據joint_degree0-3計算出手臂端點的位置
如果space_down等於true(按著空白鍵時)且端點位置與target的距離小於TOLERANCE時target位置設定為端點位置

BONUS: Target墜落(按一下G開啟/關閉，預設關閉)

遭遇問題:

- Camera.cpp中rotation不知道怎麼用(HINT1-1. Rotate original_front and original_up using this->rotation.)
- 繪圖順序(轉移、旋轉、縮放)
  多嘗試幾次就可以發現要從最遠做到最近