計算機圖學 HW1 Report

1. 實作

1.1 TODO#0 更改視窗標題

把glfwSetWindowTitle()的第二個參數改成"HW1 - 311552013"。

```
glfwSetWindowTitle(window, "HW1 - 311552013");
```

1.2 TODO#1-1 計算view matrix

更新front, up , right三個方向的向量，並使用glm::lookAt()函數計算出view matirx。

```
  front = original_front * rotation;
  up = original_up * rotation;
  right = glm::cross(front, up);
  viewMatrix = glm::lookAt(position, position + front, up);
```

1.3 TODO#1-2 計算projection matrix

使用glm::perspective()函數計算出projection matirx。

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

1.4 TODO#2-1 畫出單位圓柱

因為本次作業是使用64邊形來近似圓柱，因此先算出一個切片的角度為多少(單位:弧度)，也就是變數slice。接著使用slice搭配迴圈畫出圓柱，迴圈的每一圈都會畫出4次三角形(頂面1個+底面1個+側面2個)。

```
    glBegin(GL_TRIANGLES);
    float slice = ANGEL_TO_RADIAN(360.0f / CIRCLE_SEGMENT);
    for (int i = 0; i < CIRCLE_SEGMENT; i++) {
        // bottom
        glNormal3f(0.0f, -1.0f, 1.0f);
        glVertex3f(0.0f, 0.0f, 0.0f);
        glVertex3f(std::sin(slice * (CIRCLE_SEGMENT - i)), 0.0f,
std::cos(slice * (CIRCLE_SEGMENT - i)));
        glVertex3f(std::sin(slice * (CIRCLE_SEGMENT - (i + 1))), 0.0f,
std::cos(slice * (CIRCLE_SEGMENT - (i + 1))));

        // top
        glNormal3f(0.0f, 1.0f, 0.0f);
        glVertex3f(0.0f, 1.0f, 0.0f);
        glVertex3f(std::sin(slice * i), 1.0f, std::cos(slice * i));
        glVertex3f(std::sin(slice * (i + 1)), 1.0f, std::cos(slice * (i +
```

```
1)));

        // side
        glNormal3f(std::sin(slice * (i + 0.5)), 0.0f, std::cos(slice * (i
+ 0.5)));
        glVertex3f(std::sin(slice * i), 0.0f, std::cos(slice * i));
        glVertex3f(std::sin(slice * (i + 1)), 0.0f, std::cos(slice * (i +
1)));
        glVertex3f(std::sin(slice * i), 1.0f, std::cos(slice * i));
        glVertex3f(std::sin(slice * (i + 1)), 1.0f, std::cos(slice * (i +
1)));
        glVertex3f(std::sin(slice * i), 1.0f, std::cos(slice * i));
        glVertex3f(std::sin(slice * (i + 1)), 0.0f, std::cos(slice * (i +
1)));
    }

    glEnd();
```

1.5 TODO#2 畫出目標圓柱

先將座標移到target_pos的位置，調整好顏色(開啟bonus時為米白色，未開啟時為紅色)以及scale，再使用剛剛寫好的drawUnitCylinder()函數畫出目標圓柱。

```
    glPushMatrix();
    glTranslatef(target_pos.x, target_pos.y, target_pos.z);
    if (bonus)
        glColor3f(WHITE);
    else
        glColor3f(RED);
    glScalef(TARGET_RADIUS, TARGET_HEIGHT, TARGET_RADIUS);
    drawUnitCylinder();
    glPopMatrix();
```

1.6 TODO#3 畫出機械手臂

利用matrix疊加的特性，從底座開始一個一個向上畫到尖端，注意最外面要用glPushMatrix()和glPopMatrix()包起來，避免不同部分的幾何操作互相干擾。另外，因為每個圓柱的有自己的scale，每次畫完時都乘上倒數把scale復原。

```
// joint 0 (base)
    glPushMatrix();
    glTranslatef(0.0f, 0.0f, 0.0f);
    glRotatef(joint0_degree, 0.0f, 1.0f, 0.0f);
    glColor3f(GREEN);
    glScalef(BASE_RADIUS, BASE_HEIGHT, BASE_RADIUS);
```

```
    drawUnitCylinder();
    glScalef(1.0f / BASE_RADIUS, 1.0f / BASE_HEIGHT, 1.0f / BASE_RADIUS);

    // arm

    // move to arm 1 position
    glTranslatef(0.0f, BASE_HEIGHT, 0.0f);

    // arm 1
    glTranslatef(0.0f, 0.0f, 0.0f);
    glColor3f(BLUE);
    glScalef(ARM_RADIUS, ARM_LEN, ARM_RADIUS);
    drawUnitCylinder();
    glScalef(1.0f/ARM_RADIUS, 1.0f / ARM_LEN, 1.0f / ARM_RADIUS);

    // move to joint 1 position
    glTranslatef(0.0f, ARM_LEN + JOINT_RADIUS, 0.0f);

    // joint 1
    glTranslatef(0.0f, 0.0f, -ARM_RADIUS);
    glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
    glRotatef(joint1_degree, 0.0f, 1.0f, 0.0f);
    glColor3f(GREEN);
    glScalef(JOINT_RADIUS, JOINT_WIDTH, JOINT_RADIUS);
    drawUnitCylinder();
    glScalef(1.0f / JOINT_RADIUS, 1.0f / JOINT_WIDTH, 1.0f /
JOINT_RADIUS);
    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
    glTranslatef(0.0f, 0.0f, ARM_RADIUS);

    // move to arm 2 position
    glTranslatef(0.0f, JOINT_RADIUS, 0.0f);

    // arm 2
    glColor3f(BLUE);
    glScalef(ARM_RADIUS, ARM_LEN, ARM_RADIUS);
    glRotatef(0.0f, 1.0f, 0.0f, 0.0f);
    drawUnitCylinder();
    glScalef(1.0f / ARM_RADIUS, 1.0f / ARM_LEN, 1.0f / ARM_RADIUS);

    // move to joint 2 position
    glTranslatef(0.0f, ARM_LEN + JOINT_RADIUS, 0.0f);

    // joint 2
```

```
    glTranslatef(0.0f, 0.0f, -ARM_RADIUS);
    glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
    glRotatef(joint2_degree, 0.0f, 1.0f, 0.0f);
    glColor3f(GREEN);
    glScalef(JOINT_RADIUS, JOINT_WIDTH, JOINT_RADIUS);
    drawUnitCylinder();
    glScalef(1.0f / JOINT_RADIUS, 1.0f / JOINT_WIDTH, 1.0f /
JOINT_RADIUS);
    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
    glTranslatef(0.0f, 0.0f, ARM_RADIUS);

    // move to arm 3 position
    glTranslatef(0.0f, JOINT_RADIUS, 0.0f);

    // arm 3
    glColor3f(BLUE);
    glScalef(ARM_RADIUS, ARM_LEN, ARM_RADIUS);
    drawUnitCylinder();
    glScalef(1.0f / ARM_RADIUS, 1.0f / ARM_LEN, 1.0f / ARM_RADIUS);

    glPopMatrix();
```

1.7 TODO#4-1 偵測按鍵

針對每個按鍵都有相對應事件的flag，按下按鍵就會改變flag的值。在手臂移動的部分，按下按鍵和放開都會改變xxx_is_rotating的值，移動手臂的時候就會根據這些flag決定是否更新手臂座標。在開啟bonus的部分，因為想要按下之後可以繼續維持狀態，因此只偵測按下按鍵(action==GLFW_PRESS)的訊號。

```
  switch(key)
  {
    case GLFW_KEY_U:
        joint0_is_rotating = (joint0_is_rotating + 1) % 2;
        break;
    case GLFW_KEY_J:
        joint0_is_rotating = -(joint0_is_rotating + 1) % 2;
        break;
    case GLFW_KEY_K:
        joint1_is_rotating = (joint1_is_rotating + 1) % 2;
        break;
    case GLFW_KEY_I:
        joint1_is_rotating = -(joint1_is_rotating + 1) % 2;
        break;
    case GLFW_KEY_O:
```

```
        joint2_is_rotating = -(joint2_is_rotating + 1) % 2;
        break;
    case GLFW_KEY_L:
        joint2_is_rotating = (joint2_is_rotating + 1) % 2;
        break;
    case GLFW_KEY_SPACE:
        is_catching = !is_catching;
        break;
    // bonus features
    case GLFW_KEY_B:
        if (action == GLFW_PRESS)
            bonus = !bonus;
        break;
  }
```

1.8 TODO#4-2 更新關節旋轉角度

使用旋轉關節的flag以及預設的ROTATE_SPEED常數更新關節旋轉角度。

```
    joint0_degree += ROTATE_SPEED * joint0_is_rotating;
    joint1_degree += ROTATE_SPEED * joint1_is_rotating;
    joint2_degree += ROTATE_SPEED * joint2_is_rotating;
```

1.9 TODO#5 抓取目標

使用glm::translate()和glm::rotate()算出手臂尖端位置，並計算與目標中心點的距離是否在容忍範圍之內，把結果存在can_catch。如果can_catch為真且is_catching(是否按下空白鍵)為真，持續更新目標的座標到手臂尖端的位置。

```
    glm::vec4 catch_detect_position(0.0f, 0.0f, 0.0f, 1.0f);
    glm::vec3 target_center = target_pos + glm::vec3(0.0f, TARGET_HEIGHT
/ 2, 0.0f);
    // (target_pos.x, target_pos.y + TARGET_HEIGHT / 2, target_pos.z);
    glm::mat4 trans(1.0f);

    // create transformation matrix
    trans = glm::rotate(trans, ANGEL_TO_RADIAN(joint0_degree),
glm::vec3(0.0f, 1.0f, 0.0f));
    trans = glm::translate(trans, glm::vec3(0.0f, BASE_HEIGHT, 0.0f));
    trans = glm::translate(trans, glm::vec3(0.0f, ARM_LEN, 0.0f));
    trans = glm::translate(trans, glm::vec3(0.0f, JOINT_RADIUS, 0.0f));

    trans = glm::rotate(trans, ANGEL_TO_RADIAN(joint1_degree),
glm::vec3(0.0f, 0.0f, 1.0f));
    trans = glm::translate(trans, glm::vec3(0.0f, JOINT_RADIUS, 0.0f));
```

```
    trans = glm::translate(trans, glm::vec3(0.0f, ARM_LEN, 0.0f));
    trans = glm::translate(trans, glm::vec3(0.0f, JOINT_RADIUS, 0.0f));

    trans = glm::rotate(trans, ANGEL_TO_RADIAN(joint2_degree),
glm::vec3(0.0f, 0.0f, 1.0f));
    trans = glm::translate(trans, glm::vec3(0.0f, JOINT_RADIUS, 0.0f));
    trans = glm::translate(trans, glm::vec3(0.0f, ARM_LEN, 0.0f));
    trans = glm::translate(trans, glm::vec3(0.0f, CATCH_POSITION_OFFSET,
0.0f));

    // get catch detect position
    catch_detect_position = trans * catch_detect_position;

    // check if the position of arm endpoint can catch target
    can_catch = (
        (catch_detect_position.x - target_center.x) *
(catch_detect_position.x - target_center.x) +
        (catch_detect_position.y - target_center.y) *
(catch_detect_position.y - target_center.y) +
        (catch_detect_position.z - target_center.z) *
(catch_detect_position.z - target_center.z)
        < TOLERANCE
    );

    // if can catch and is catch (pressing space), update the position of
target
    if (can_catch && is_catching)
        target_pos = glm::vec3(
            catch_detect_position.x,
            catch_detect_position.y - TARGET_HEIGHT / 2,
            catch_detect_position.z
        );
```

1.10 TODO#BONUS 新增目標掉落機制與彈跳效果

當目標不在地面上且非抓取狀態時，目標會往下掉落，並在落地時產生一個彈跳效果。預設落下速度=0，加速度=-0.001，時間單位為執行一次window loop的時間。每一次都會更新速度值與目標位置，當落地時，給予一個反向的速度來模擬彈跳，且速度為原速度值的一半，看起來就會有愈跳愈低的效果。

```
    // physical falling simulation
    if (bonus && !(can_catch && is_catching)){
        target_pos.y += velocity;
        if (target_pos.y < 0){
```

```
            target_pos.y = 0;
            velocity = -velocity*0.5;
        }
        velocity += acceleration;
    }
```

2. 操作

- WASD移動camera位置
- 滑鼠移動視角
- 空白鍵抓取物體(需按住不放)
- UJIKOL移動手臂關節
- B開啟bonus功能，再按一次關閉，效果:
    - 目標變成米白色
    - 非抓取狀態會掉落至地面並產生彈跳效果

3. 問題

3.1 圓柱體部分消失?

用glVertex3f()畫出來的三角形只會顯示一面，另一面不會顯示，因此需要注意畫的時候從外面看必須是逆時針方向。這當初困擾我一段時間，感謝助教在E3上的提醒，讓我解決這個bug。

3.2 圓柱體光影顯示問題?

在畫圓柱體時每一面都需設定glNormal3f()。