Computer Graphics Homework 1 Report

Created	@October 15, 2022 12:20 PM
Last edited time	@October 15, 2022 12:28 PM
Related to 記事 (Column)	
Tags	
URL	

Name	Student ID
許凱傑	0816117

Implementation

Camera

I used `glm::lookAt()` to calculate viewMatrix because it is convient and correct.

Unit Cylinder

I split unit cylinder into 3 parts: top, down and side.

To render top and down circle, I draw many small triangles to make them look like a circle. To achieve this, I calculate each triangles coordinate by circle's degree.
Their normals are `+y`, `-y`.

```
// draw top side of cylinder
glBegin(GL_TRIANGLE_STRIP);
  for (int i = 0; i <= CIRCLE_SEGMENT; ++i) {
    float rad = ANGEL_TO_RADIAN(360 * (float)i / CIRCLE_SEGMENT);
    float nRad = ANGEL_TO_RADIAN(360 * (float)(i + 1) / CIRCLE_SEGMENT);
    glNormal3f(0.0f, 1.0f, 0.0f);
    glVertex3f(0.0f, 1.0f, 0.0f);
    glVertex3f(cos(rad), 1, -sin(rad));
    glVertex3f(cos(nRad), 1, -sin(nRad));
  }
glEnd();
```

Drawing side of cylinder is a little complicated because each side segement has different normal.

```
// draw side of cylinder
for (int i = 0; i <= CIRCLE_SEGMENT; ++i) {
    glBegin(GL_TRIANGLE_STRIP);
    float rad = ANGEL_TO_RADIAN(360 * (float)i / CIRCLE_SEGMENT);
    float nRad = ANGEL_TO_RADIAN(360 * (float)(i + 1) / CIRCLE_SEGMENT);
    float mRad = (rad + nRad) / 2.0f;
    glNormal3f(cos(mRad), 0.0f, -sin(mRad));
    glVertex3f(cos(rad), 0.0, -sin(rad));
    glVertex3f(cos(nRad), 0.0, -sin(nRad));
    glVertex3f(cos(rad), 1.0, -sin(rad));
    glVertex3f(cos(nRad), 1.0, -sin(nRad));
    glEnd();
  }
```

Render the robotic arm

This part is easy, just notice that matrics is post-multiply in OpenGL. Then, use `glTranslatef()`, `glScalef()`, `glRotatef()` with proper parameter. Besides, be careful to use `glPushMatrix()`, and `glPopMatrix()` to control matrics hierachy.

Detect key-events, perform rotation or catch target object

To control the robotic arm and catch object, just add key-events that add or minus `joint<i>_degree` and set `isCatching` variable to `TRUE` if space bar is hold.

Catch the target object with robotic arm

I calculated catch position in `getCatchPosition()` function, this function return the catch Position, and then we can calculate the distance to `target_pos`.

If the current state is `isCatching == True` which means space bar is hold and the distance between `target_pos` and catch position is less than `TOLERANCE`, just assign catch position to `target_pos`;

```
glm::vec3 getCatchPostion() {
  glm::vec4 pos(glm::vec3(0.0f), 1.0f);
  static const glm::mat4x4 identity = glm::mat4x4(1.0f);
  static const glm::mat4x4 baseTrans = glm::translate(identity, glm::vec3(0, BASE_HEIGHT, 0));
  static const glm::mat4x4 armTrans = glm::translate(identity, glm::vec3(0, ARM_LEN, 0));
  static const glm::mat4x4 jointTrans = glm::translate(identity, glm::vec3(0, JOINT_RADIUS, 0));
  static const glm::mat4x4 offsetTrans = glm::translate(identity, glm::vec3(0, CATCH_POSITION_OFFSET, 0));
  const glm::mat4x4 jointRotate0 = glm::rotate(identity, ANGEL_TO_RADIAN(joint0_degree), glm::vec3(0, 1, 0));
  const glm::mat4x4 jointRotate1 = glm::rotate(identity, ANGEL_TO_RADIAN(joint1_degree), glm::vec3(0, 0, -1));
  const glm::mat4x4 jointRotate2 = glm::rotate(identity, ANGEL_TO_RADIAN(joint2_degree), glm::vec3(0, 0, -1));
  pos = jointRotate0 * baseTrans *
        armTrans *
        jointTrans * jointRotate1 * jointTrans *
        armTrans *
        jointTrans * jointRotate2 * jointTrans *
        armTrans *
        offsetTrans * pos;
  return glm::vec3(pos.x/pos.w, pos.y/pos.w, pos.z/pos.w);
}
```

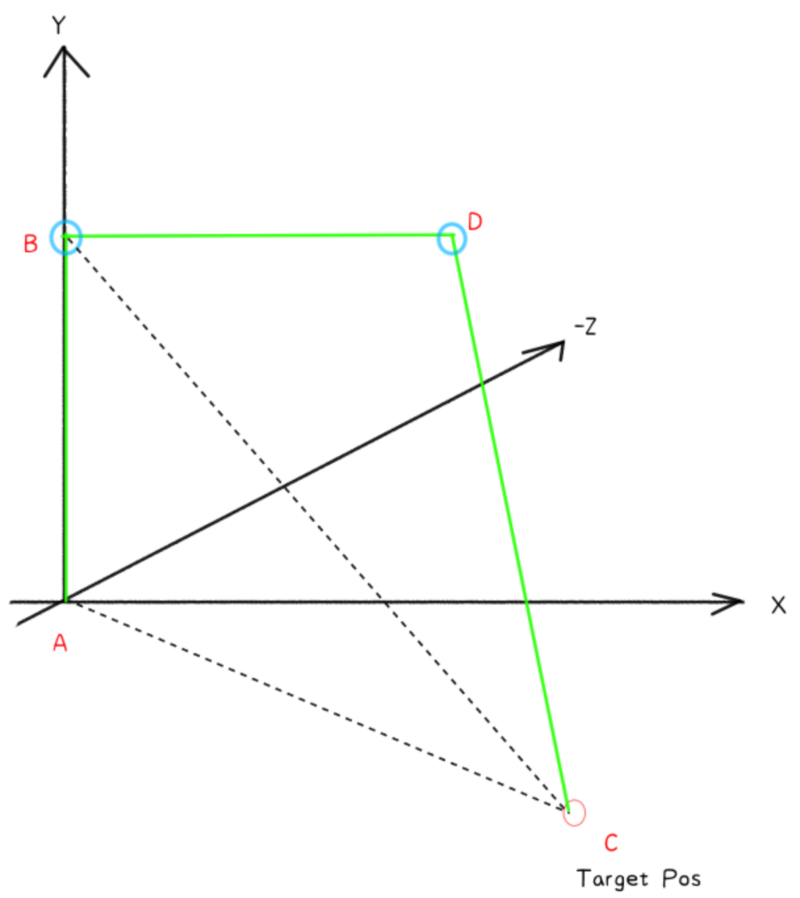
Problems you encountered

I didn't notices that in OpenGL, the camera is look at -Z axis, so when I were coding `drawUnitCylinder()` function. I gave reversed Z coordinate to OpenGL, leading to incorrect result.

Bonus

Robotic Arm Auto Solver

Press **M** to activate the magic funtion automatically adjust robotic arm to target position！



To implement the magic function, we need:

- joint0: $\angle XAC$
- joint1: $180 - \angle ABD$
- joint2: $180 - \angle BDC$

What we have:

- $\overrightarrow{AB}$
- $\overrightarrow{AC}$
- $\overrightarrow{BC}$
- $||\overrightarrow{BD}||$
- $||\overrightarrow{CD}||$

By Cosine Law $cosC = \frac{a^2+b^2-c^2}{2ab}$

- $\angle ABD = \angle ABC + \angle CBD$
    - $\angle ABC = arccos(\frac{|AB|^2+|BC|^2-|AC|^2}{2|AB||BC|})$

- $\angle CBD = arccos(\frac{|BC|^2+|BD|^2-|CD|^2}{2|BC||BD|})$

- $\angle BDC = arccos(\frac{|BD|^2+|CD|^2-|BC|^2}{2|BD||CD|})$

Joint 0

Calculate the angle between $\overrightarrow{AC}$ and x and z axis. We need its two angle with 2 direction because we need to decide rotate clockwise or counterclockwise.

```
// rotate joint0 to same 2D plane
glm::vec2 AC(target_pos.x, target_pos.z);
float xAngle = RADIAN_TO_ANGEL(glm::angle(glm::vec2(1,0), glm::normalize(AC)));
float zAngle = RADIAN_TO_ANGEL(glm::angle(glm::vec2(0, 1), glm::normalize(AC)));
if (xAngle < 90) {
  j0 = -90 + zAngle;
} else {
  j0 = -90 - zAngle;
}
```

Joint 1 and Joint 2

```
glm::vec2 A(0, 0);
glm::vec2 B(0, BASE_HEIGHT + ARM_LEN + JOINT_RADIUS);
glm::vec2 C(glm::length(glm::vec2(target_pos.x, target_pos.z)), target_pos.y+TARGET_HEIGHT/2.0);
float ABC = RADIAN_TO_ANGEL(glm::angle(glm::normalize(A - B), glm::normalize(C - B)));

float BD_distance = ARM_LEN + 2 * JOINT_RADIUS;
float DC_distance = ARM_LEN + JOINT_RADIUS + CATCH_POSITION_OFFSET;
float BC_distance = glm::distance(B,C);

auto getCosTheta = [](const float& a, const float& b, const float& c) {
    // cosine law
    return (a * a + b * b - c * c) / (2 * a * b);
};

float cosCBD = getCosTheta(BD_distance, BC_distance, DC_distance);
float CBD = RADIAN_TO_ANGEL(glm::acos(cosCBD));
float ABD = ABC + CBD;
j1 = (180.0 - ABD);

float cosBDC = getCosTheta(BD_distance, DC_distance, BC_distance);
float BDC = RADIAN_TO_ANGEL(glm::acos(cosBDC));
j2 = (180 - BDC);
```