111550006 林庭寫

# Homework 1 – 2024 Computer Graphics

This report includes implementation detail corresponding to the instruction and encountered problems during the experiment. Since I used Visual Studio Code on Mac as IDE for this homework, the directory of include files had been modified.

## IMPLEMENTATION DETAILS

1. **Viewing Transformation**

First, I updated front and up vectors through rotation * original vector, then computed right vector as cross product of front and up. Additionally, I recomputed up as cross product of right and front vectors to make sure up vector was perpendicular to front and right after rotation. Finally, I directly used glm::lookAt to calculate view matrix, the three parameters are: position, focal point and up vector.

2. **Projection Transformation**

Simply use glm:: perspective to calculate the perspective projection matrix with provided parameters. The perspective projection matrix defines how 3D points are projected onto a 2D screen; FOV is set to 45 degrees (in radians) for a standard field of view. aspectRatio specifies the aspect ratio of the screen or window, ensuring the projection scales correctly along the x-axis. zNear and zFar define the near and far clipping planes, respectively, which set the range of distances from the camera that will be rendered. Anything closer than zNear or farther than zFar is clipped out.

## 3. Render Putter

The Putter was constructed with one vertical cylinder and one horizontal cylinder. The following contents are about how to complete drawUnitCylinder( ):

**Draw the Side of the Cylinder**:

- **Vertex Calculation**:
  - For each segment (from i = 0 to CIRCLE_SEGMENT), two angles (angle1 and angle2) define two positions on the circular edge.
  - x1, z1 and x2, z2 represent points on the circle's circumference.
- **Draw Two Triangles per Segment**:
  - Each segment is represented by two triangles, forming a rectangular section (or quad) of the cylinder.
  - **First Triangle**: Vertices are defined by glVertex3f(x1, 0.5f, z1), glVertex3f(x1, -0.5f, z1), and glVertex3f(x2, 0.5f, z2).
  - **Second Triangle**: Vertices are defined by glVertex3f(x1, -0.5f, z1), glVertex3f(x2, -0.5f, z2), and glVertex3f(x2, 0.5f, z2).
  - **Normals**: glNormal3f(x1, 0.0f, z1) and glNormal3f(x2, 0.0f, z2) provide outward normals, crucial for lighting.

**Draw the Top / Bottom Circle**:

- **Set Normal**: glNormal3f(0.0f, ±1.0f, 0.0f); to face upward / downward.
- **Loop through Circle Segments**:
  - Each triangle connects the center point (0.0f, -0.5f, 0.0f) with two points on the circle edge.
- **Vertices**:
  - The center vertex glVertex3f(0.0f, -0.5f, 0.0f), and vertices glVertex3f(x1, -0.5f, z1) and glVertex3f(x2, -0.5f, z2) form each triangle segment on the circle's base.

**Render Putter in main():**

Using glPushMatrix to save the current matrix state, which allowed applying transformations specific to the putter without affecting other objects in the scene. The pivotOffsetX/Y was set up for maintaining the relative position of hitting part and rod part; furthermore, the pivot was the swing rotation pivot. Apply glRotatef for initial rotation for the hitting part and glScalef for setting the given radius.

## 4. Render Golfball

Most of the render method of golfball in main() was identical to rendering putter. The only extra setup was glMultMatrixf, which apply the current rotation matrix to simulate spinning. This golf ball is rendered with unit sphere, the following step is how I set up sphere geometry.

- Setting Up Sphere Geometry
  - The sphere is divided into vertical (latitude) segments, called STACKS, and horizontal (longitude) segments, called SECTORS.
  - Each point on the sphere's surface is calculated using trigonometric functions(GL_TRIANGLE_STRIP) based on the stack and sector angles, which correspond to the latitude and longitude.
- Nested Loop implementation
  - Outer Loop (Stacks):

    The outer loop iterates through STACK levels, moving from the top (π/2 radians) to the bottom (-π/2 radians). For each stack level, two vertices are computed for the current stack and the next stack, creating a "strip" that wraps around the sphere.

  - Inner Loop (Sectors):

The inner loop iterates over `SECTOR` divisions around the sphere's circumference. It calculates the vertices for each point in the current stack and the next stack, effectively creating a strip of triangles to form part of the sphere's surface.

## 5. Putter Control

I used switch && case to control the putter, the detail explanation is as follows.

**Switch on Key Input**:

- The switch statement is used to check which key is pressed, enabling various actions based on specific keys.
- Each case handles one key and defines the behavior when the key is pressed (GLFW_PRESS) and released (GLFW_RELEASE).

**Key Event Cases**:

- **GLFW_KEY_UP**:
  - If the key is pressed, delta_xzpos is set to scalar::PLUS, indicating forward movement along the XZ plane.
  - When released, delta_xzpos is reset to scalar::NONE, stopping the movement.
- **GLFW_KEY_DOWN**:
  - Similar to the UP key but in the opposite direction: pressing sets delta_xzpos to scalar::MINUS for backward movement.
  - Releasing resets it to scalar::NONE.
- **GLFW_KEY_LEFT**:
  - When pressed, delta_y_rotate is set to angle::COUNTERCLOCKWISE for a counterclockwise rotation along the Y-axis (yaw).
  - When released, delta_y_rotate is reset to angle::NONE, stopping the rotation.

- **GLFW_KEY_RIGHT**:
    - Similar to the LEFT key, but rotates clockwise: pressing sets delta_y_rotate to angle::CLOCKWISE.
    - Releasing it resets delta_y_rotate to angle::NONE.
- **GLFW_KEY_SPACE**:
    - Used to simulate the putter swing.
    - Pressing sets delta_x_rotate to angle::CLOCKWISE (swing back).
    - Releasing it reverses delta_x_rotate to angle::COUNTERCLOCKWISE, simulating a forward swing.

**Control of Global Variables**:

The code modifies global variables (delta_xzpos, delta_y_rotate, and delta_x_rotate), which likely affect the rendering update loop. These variables define the putter's movement and rotation, so updating them triggers changes in its rendered position and orientation.

## 6. Hitting detection

The detection included several steps. The follows are the implementation logic:

- **Transform Putter's Hitting Part to World Space:**

Apply translation ( use glm::translate to create translate matrix ) and rotations to find the exact world position of the putter's hitting part.

**Translation (`xzpos`):** Moves the putter to its current position in the world.

**Yaw Rotation (`y_rotate`):** Rotates the putter horizontally around the Y-axis.

**Swing Rotation (`x_rotate`):** Rotates the putter vertically around the X-axis to simulate swinging.

- **Calculate Distance to Ball:**

Use glm::length to calculate the distance between ballpos and hitting part's world position.

- **Check Hit Conditions:**

Ensure the putter is swinging forward and the ball is within a tolerable distance. Use a boolean value isHit to monitor the condition.

7. **Ball Movement**

If conditions are met, move the ball forward, simulate rolling, and update its rotation with provided values. Stop the ball after it has traveled a specified distance. Directly modified ballpos with correspondin direction and speed for movement; use glm::rotate and align the direction with `ball_rotate_normal`:

```
ball_rotate_normal = glm::normalize(glm::cross(glm::vec3(0, 1, 0),
      ball_forward));
```

8. **Bonus**

In this part, I create a hole on the plane and the ball can fall into the hole. I create a hole with a cylinder for visualization ( drawHoleCylinder ), anduses the stencil buffer to create a "hole" effect in a plane by selectively masking out a specific region, in this case, a circular shape.

Additionally, I use a bool: isFalling to detect if the ball is over the hole. The ball will fall into the hole with speed and then reset to the original position.

## ENCOUNTERED PROBLEMS

1. The light penetrated the solid hitting part, add the following lines to tackle the issue.

   Use: glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glEnable(GL_DEPTH_TEST);

2. It is hard to create a hole in a plane. I try to simulate the hole with cylinder rather than directly dig a hole on the plane.