```
cgra::gl_mesh Application::cylinder(int subdiv, float top_radius, float bottom_radius, float height, bool is_cylinder_fill_top, bool is_cylinder_fill_bottom,
                                    float x_position, float y_position, float z_position, float x_rotation, float y_rotation, float z_rotation) {
    mesh_builder mb;
    mesh_vertex mv;

    float x_point;
    float y_point;
    float z_point;

    glm::vec3 centroid(x_position, y_position, z_position);

    glm::vec3 BOTTOM_POLE(centroid.x, centroid.y - (height * 0.5), centroid.z);
    glm::vec3 TOP_POLE(centroid.x, centroid.y + (height * 0.5), centroid.z);
    glm::vec3 RIGHT_POLE(centroid.x + ((top_radius + bottom_radius) * 0.5), centroid.y, centroid.z);
    glm::vec3 LEFT_POLE(centroid.x - ((top_radius + bottom_radius) * 0.5), centroid.y + (height * 0.5), centroid.z);

    int vert_count = 0;

    // iterate theta_subdiv x phi_subdiv to generate all vetices, then explicitly add the bottom pole, then top pole
    for (int i = 0; i < subdiv * 4; i++) {  // <= accounts for the last column of vertices to complete the shape -> 5 lines creates 4 rows
        float curr_theta = ((2 * pi<float>()) / subdiv) * i;

        if (i < subdiv) {  // bottom vertices for side
            x_point = centroid.x + (bottom_radius * cos(curr_theta));
            y_point = BOTTOM_POLE.y;
            z_point = centroid.z + (bottom_radius * sin(curr_theta));

            vec3 curr_point(x_point, y_point, z_point);
            mv.pos = curr_point;
            mv.norm = normalize(curr_point - centroid);
        }
        else if (i >= subdiv && i < (subdiv * 2)) {      // top vertices for side
            x_point = centroid.x + (top_radius * cos(curr_theta));
            y_point = TOP_POLE.y;
            z_point = centroid.z + (top_radius * sin(curr_theta));

            vec3 curr_point(x_point, y_point, z_point);
            mv.pos = curr_point;
            mv.norm = normalize(curr_point - centroid);
        }

        else if (i >= (subdiv * 2) && (i < subdiv * 3)) {  // bottom vertices for bottom
            x_point = centroid.x + (bottom_radius * cos(curr_theta));
            y_point = BOTTOM_POLE.y;
            z_point = centroid.z + (bottom_radius * sin(curr_theta));

            vec3 curr_point(x_point, y_point, z_point);
            mv.pos = curr_point;
            mv.norm = normalize(vec3(centroid.x, BOTTOM_POLE.y, centroid.z) - centroid);
        }
        else if (i >= (subdiv * 3)) {      // top vertices for top
            x_point = centroid.x + (top_radius * cos(curr_theta));
            y_point = TOP_POLE.y;
            z_point = centroid.z + (top_radius * sin(curr_theta));

            vec3 curr_point(x_point, y_point, z_point);
            mv.pos = curr_point;
            mv.norm = normalize(vec3(centroid.x, TOP_POLE.y, centroid.z) - centroid);
        }

        mb.push_vertex(mv);

    }


    // bottom pole
    mv.pos = BOTTOM_POLE;
    mv.norm = BOTTOM_POLE - centroid;
    mv.uv = vec2(0.0, 0.0);
    mb.push_vertex(mv);

    // top pole
    mv.pos = TOP_POLE;
    mv.norm = TOP_POLE - centroid;
    mv.uv = vec2(0.0, 1.0);
    mb.push_vertex(mv);


    // ---- Rotation start ----------------------------
    // convert degrees to radians
    float x_rotation_radians = glm::radians(x_rotation);
    float y_rotation_radians = glm::radians(y_rotation);
    float z_rotation_radians = glm::radians(z_rotation);

    glm::mat4 x_rotation_matrix = glm::rotate(glm::mat4(1.0f), x_rotation_radians, glm::vec3(1.0f, 0.0f, 0.0f));
    glm::mat4 y_rotation_matrix = glm::rotate(glm::mat4(1.0f), y_rotation_radians, glm::vec3(0.0f, 1.0f, 0.0f));
    glm::mat4 z_rotation_matrix = glm::rotate(glm::mat4(1.0f), z_rotation_radians, glm::vec3(0.0f, 0.0f, 1.0f));


    // Apply the rotation to each vertex
    for (cgra::mesh_vertex& vertex : mb.vertices) {
        vertex.pos -= centroid;   // bring position back to origin for rotation
        vertex.pos = glm::vec3(x_rotation_matrix * glm::vec4(vertex.pos, 1.0f));
        vertex.pos = glm::vec3(y_rotation_matrix * glm::vec4(vertex.pos, 1.0f));
        vertex.pos = glm::vec3(z_rotation_matrix * glm::vec4(vertex.pos, 1.0f));
        vertex.pos += centroid;   // return to current position
    }
    //----- Rotation end -------------------------




    //-----push indices - start ------------------------------------------------------------

    int BOTTOM_POLE_INDEX = subdiv * 4;
    int TOP_POLE_INDEX = subdiv * 4 + 1;

    // push each respective vertices index in proper order to generate triangles
    for (int i = 0; i < subdiv; i++) {   // iterate through each row
        if (i < subdiv - 1) {
            // top left triangle for this respective row and column
            mb.push_index(i);
            mb.push_index(i + subdiv);
            mb.push_index(i + subdiv + 1);

            // bottom right triangle for this respective row and column
            mb.push_index(i);
            mb.push_index(i + subdiv + 1);
            mb.push_index(i + 1);

            if (is_cylinder_fill_bottom) {
                // bottom triangle
                mb.push_index(i);
                mb.push_index(BOTTOM_POLE_INDEX);
                mb.push_index(i + 1);
            }

            if (is_cylinder_fill_top) {
                // top triangle
                mb.push_index(i + subdiv);
                mb.push_index(TOP_POLE_INDEX);
                mb.push_index(i + subdiv + 1);
            }
        }
        else {    // when i == subdiv; this is the last iteration so starts back at beginning vertex
            // top left triangle for this respective row and column
            mb.push_index(i);
            mb.push_index(i + subdiv);
            mb.push_index(i + 1);

            // bottom right triangle for this respective row and column
            mb.push_index(i);
            mb.push_index(i + 1);
            mb.push_index(0);

            if (is_cylinder_fill_bottom) {
                // bottom pole triangle
                mb.push_index(i);
                mb.push_index(BOTTOM_POLE_INDEX);
                mb.push_index(0);
            }

            if (is_cylinder_fill_top) {
                // top pole triangle
                mb.push_index(i + subdiv);
                mb.push_index(TOP_POLE_INDEX);
                mb.push_index(i + 1);
            }
        }

    }


    //-----push indices - end --------------------------------------------------------------

    m_model.mesh = mb.build();

    return m_model.mesh;
}
```