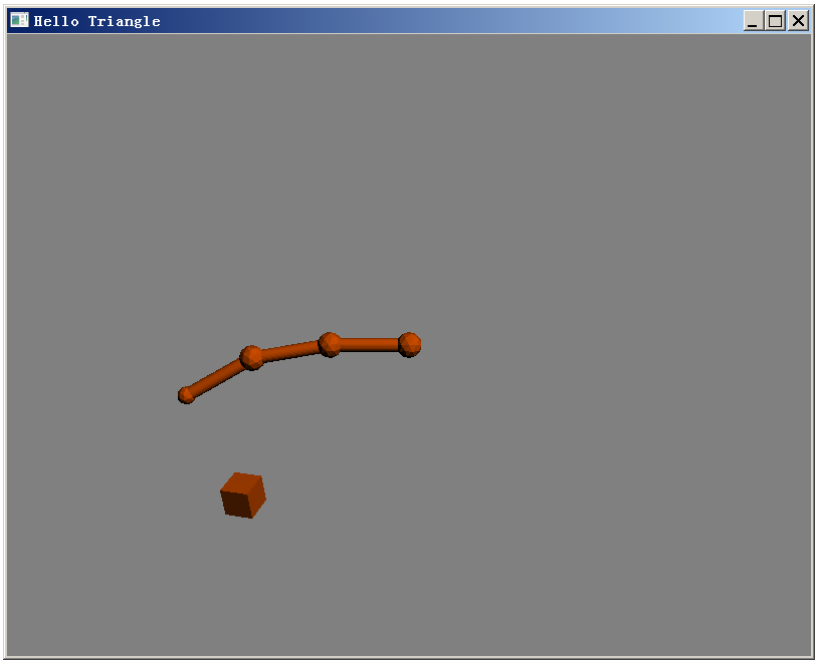
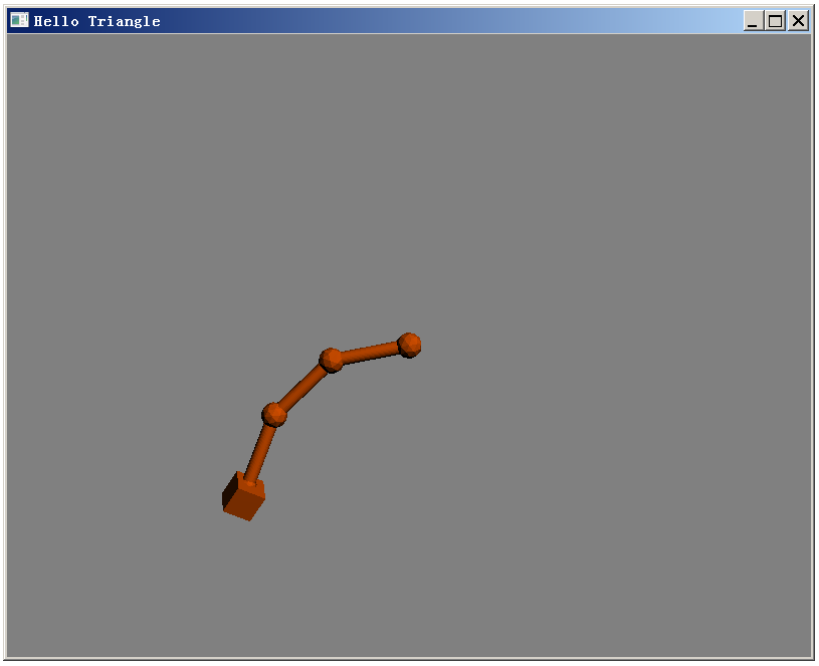
CS7GV5 Report Example

Name:	Zhiqiang Cheng
Student ID:	22322639
Youtube link:	https://youtu.be/AzgBT6INkzA

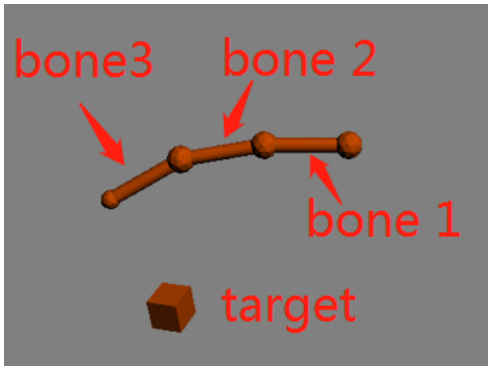
Required feature: Simple 2-bone IK in 2D

Screenshot(s) of feature:





Describe your implementation:



1. press key up, down, left, right to move target

2. forward kinematics
   press z or x, to rotate bone 1
   press a or s, to rotate bone 2
   press q or w, to rotate bone 3

3. IK with CCD
   press c

Code Snippet:

```
void display() {
    int currentFrame = glutGet(GLUT_ELAPSED_TIME);
    deltaTime = currentFrame - lastFrame;
    lastFrame = currentFrame;

    // tell GL to only draw onto a pixel if the shape is closer to the viewer
    glEnable(GL_DEPTH_TEST); // enable depth-testing
    glDepthFunc(GL_LESS); // depth-testing interprets a smaller value as "closer"
    glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    shader.use();

    mat4 view = identity_mat4();
    //mat4 persp_proj = perspective(45.0f, (float)width / (float)height, 0.1f, 1000.0f);
    mat4 persp_proj = perspective(camera.Zoom, (float)scr_width / (float)scr_height, 0.1f,
100.0f);
    //view = look_at(vec3(0, 0, 0), vec3(0, 0, -1), vec3(0, 1, 0));
    view = camera.GetViewMatrix();
    shader.setMat4("proj", persp_proj);
    shader.setMat4("view", view);

    // box, target
    modelTarget = identity_mat4();
    modelTarget = scale(modelTarget, vec3(0.2,0.2,0.2));
    modelTarget = rotate_x_deg(modelTarget, rotate_x);
    modelTarget = rotate_y_deg(modelTarget, rotate_y);
    modelTarget = rotate_z_deg(modelTarget, rotate_z);
    //modelTarget = translate(modelTarget, vec3(0.0, -3.5, 0.0));
    modelTarget = translate(modelTarget, vec3(target_x, target_y, target_z));
    shader.setMat4("model", modelTarget);
```

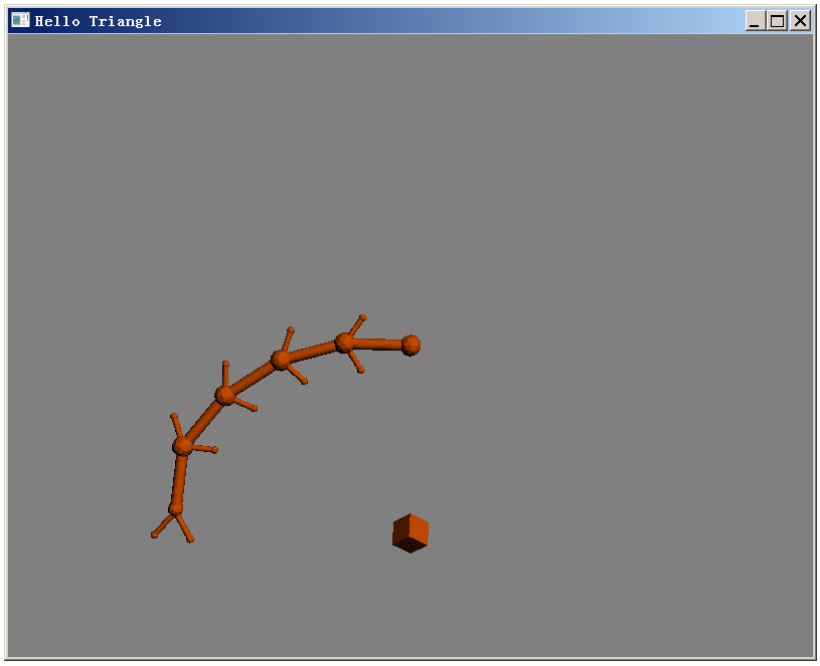
```
    renderCube();

    // bone 1, root
    model1 = identity_mat4();
    model1 = rotate_z_deg(model1, bone1_start_z_deg + bone1_rotate_z);
    shader.setMat4("model", model1);
    renderBone();

    // bone 2
    model2 = identity_mat4();
    model2 = rotate_z_deg(model2, bone2_start_z_deg + bone2_rotate_z);
    model2 = translate(model2, vec3(-1.05, 0.0, 0.0));
    model2 = model1 * model2;
    shader.setMat4("model", model2);
    renderBone();

    // bone 3
    model3 = identity_mat4();
    model3 = rotate_z_deg(model3, bone3_start_z_deg + bone3_rotate_z);
    model3 = translate(model3, vec3(-1.05, 0.0, 0.0));
    model3 = model2 * model3;
    shader.setMat4("model", model3);
    renderBone();

    // end point
    model4 = identity_mat4();
    model4 = scale(model4, vec3(0.5, 0.5, 0.5));
    model4 = rotate_z_deg(model4, bone4_start_z_deg + bone4_rotate_z);
    model4 = translate(model4, vec3(-1.05, 0.0, 0.0));
    model4 = model3 * model4;
    shader.setMat4("model", model4);
    //renderBone();

    glutSwapBuffers();
}


void SpecialKeys(int key, int x, int y) {

    GLfloat stepSize = 0.025f;

    switch (key) {
        case GLUT_KEY_UP:
            target_y += stepSize;
            break;
        case GLUT_KEY_DOWN:
            target_y -= stepSize;
            break;
        case GLUT_KEY_LEFT:
            target_x -= stepSize;
            break;
        case GLUT_KEY_RIGHT:
            target_x += stepSize;
            break;
        default:
            break;
    }
}

void CCD(int frame) {
    static int i = 0;
```

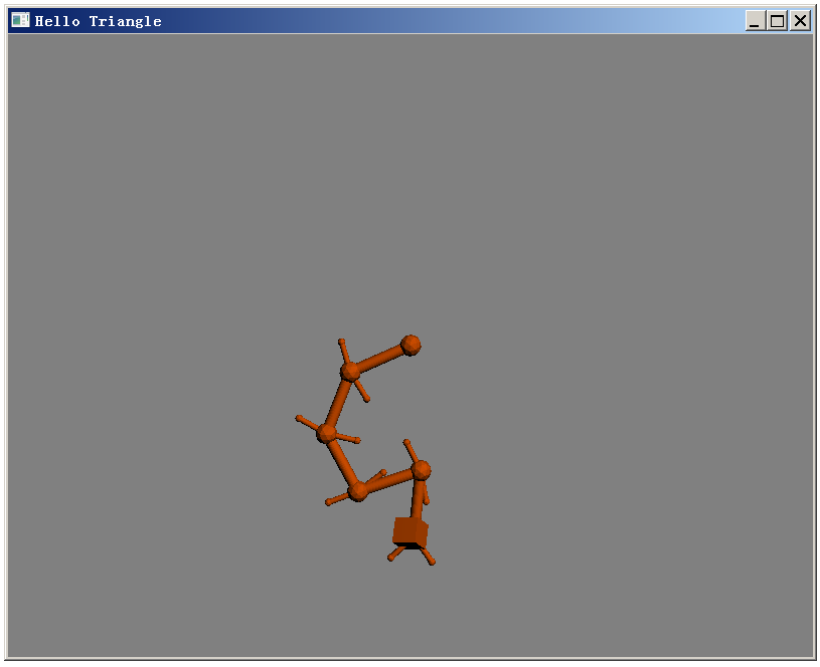
```
    if (++i > 100) {
        i = 0;
        CCDRunning = false;
        return;
    }

    vec3 endPos = getPos(model4);
    vec3 tarPos = getPos(modelTarget);
    vec3 curPos;
    vec3 e_c;
    vec3 t_c;
    vec3 r;
    float cos_theta;
    float angle;

    float distance = sqrt(pow((tarPos.v[0] - endPos.v[0]), 2) + pow((tarPos.v[1] -
endPos.v[1]), 2));
    float tolerance = 0.1;

    if (distance < tolerance) {
        solve = true;
        CCDRunning = false;
    }
    else
    {
        if (frame == 1) {
            endPos = getPos(model4);
            tarPos = getPos(modelTarget);
            curPos = getPos(model3);
            e_c = normalise(endPos - curPos);
            t_c = normalise(tarPos - curPos);
            r = cross(e_c, t_c);
            cos_theta = clamp(dot(e_c, t_c));
            angle = acos(cos_theta) * ONE_RAD_IN_DEG;

            if (r.v[2] < 0) {
                angle *= -1;
            }
            bone3_rotate_z += angle;
        }
        else if (frame == 2) {
            endPos = getPos(model4);
            tarPos = getPos(modelTarget);
            curPos = getPos(model2);
            e_c = normalise(endPos - curPos);
            t_c = normalise(tarPos - curPos);
            r = cross(e_c, t_c);
            cos_theta = clamp(dot(e_c, t_c));
            angle = acos(cos_theta) * ONE_RAD_IN_DEG;

            if (r.v[2] < 0) {
                angle *= -1;
            }

            bone2_rotate_z += angle;
        }
        else if (frame == 3) {
            endPos = getPos(model4);
            tarPos = getPos(modelTarget);
            curPos = getPos(model1);
            e_c = normalise(endPos - curPos);
```

```
            t_c = normalise(tarPos - curPos);
            r = cross(e_c, t_c);
            cos_theta = clamp(dot(e_c, t_c));
            angle = acos(cos_theta) * ONE_RAD_IN_DEG;

            if (r.v[2] < 0) {
                angle *= -1;
            }
            bone1_rotate_z += angle;
        }
    }
}
```

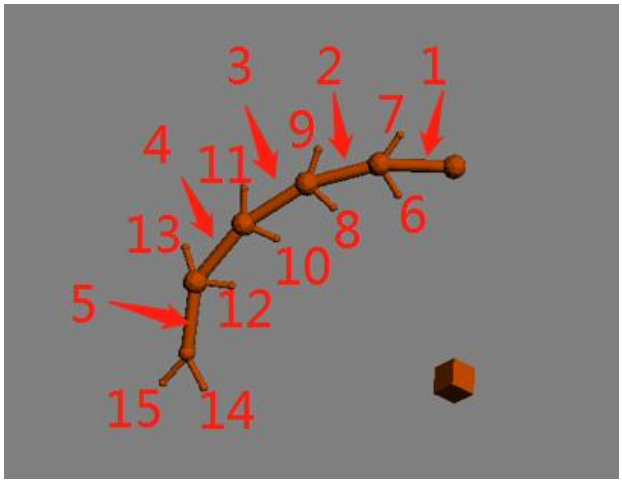
Required feature: Multi-bone IK in 3D

Screenshot(s) of feature:





Describe your implementation:



1. There are total 15 bones, contains five big bone (1,2,3,4,5), 10 small bones (6-15)

2. press key up, down, left, right to move target

3. forward kinematics
   press z or x, to rotate bone 1
   press a or s, to rotate bone 2
   press q or w, to rotate bone 3
   press d or f, to rotate bone 4
   press e or r, to rotate bone 5

4. IK with CCD
   press c

Code Snippet:

```
void display() {
    int currentFrame = glutGet(GLUT_ELAPSED_TIME);
    deltaTime = currentFrame - lastFrame;
    lastFrame = currentFrame;
```

```
    // tell GL to only draw onto a pixel if the shape is closer to the viewer
    glEnable(GL_DEPTH_TEST); // enable depth-testing
    glDepthFunc(GL_LESS); // depth-testing interprets a smaller value as "closer"
    glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    shader.use();

    mat4 view = identity_mat4();
    //mat4 persp_proj = perspective(45.0f, (float)width / (float)height, 0.1f, 1000.0f);
    mat4 persp_proj = perspective(camera.Zoom, (float)scr_width / (float)scr_height, 0.1f,
100.0f);
    //view = look_at(vec3(0, 0, 0), vec3(0, 0, -1), vec3(0, 1, 0));
    view = camera.GetViewMatrix();
    shader.setMat4("proj", persp_proj);
    shader.setMat4("view", view);

    // box, target
    modelTarget = identity_mat4();
    modelTarget = scale(modelTarget, vec3(0.2,0.2,0.2));
    modelTarget = rotate_x_deg(modelTarget, rotate_x);
    modelTarget = rotate_y_deg(modelTarget, rotate_y);
    modelTarget = rotate_z_deg(modelTarget, rotate_z);
    //modelTarget = translate(modelTarget, vec3(0.0, -3.5, 0.0));
    modelTarget = translate(modelTarget, vec3(target_x, target_y, target_z));
    shader.setMat4("model", modelTarget);
    renderCube();

    // bone 1, root
    model[0] = identity_mat4();
    model[0] = rotate_z_deg(model[0], 0 + bone_rotate_z[0]);
    //model1 = translate(model1, vec3(0.0, 0.0, 0.0));
    shader.setMat4("model", model[0]);
    renderBone();

    for (int i = 1; i < numsOfBones; i++)
    {
        model[i] = identity_mat4();
        model[i] = rotate_z_deg(model[i], 0 + bone_rotate_z[i]);
        model[i] = translate(model[i], vec3(-1.05, 0.0, 0.0));
        model[i] = model[i-1] * model[i];
        shader.setMat4("model", model[i]);
        renderBone();
```

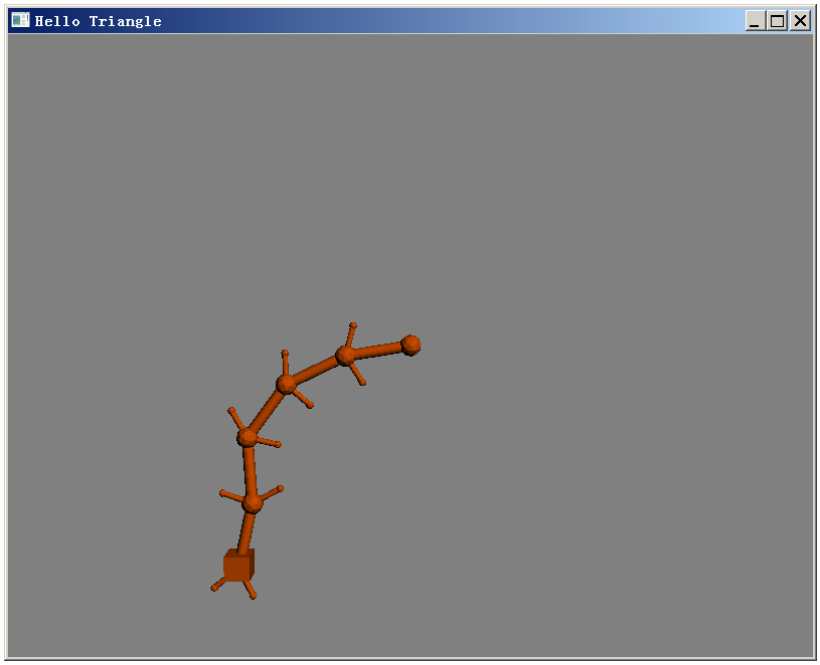
```
        // bone left
        mat4 model_left = identity_mat4();
        model_left = scale(model_left, vec3(0.5, 0.5, 0.5));
        model_left = rotate_z_deg(model_left, 90 + bone_left_rotate_z);
        model_left = translate(model_left, vec3(-1.05, 0.0, 0.0));
        model_left = model[i-1] * model_left;
        shader.setMat4("model", model_left);
        renderBone();

        // bone right
        mat4 model_right = identity_mat4();
        model_right = scale(model_right, vec3(0.5, 0.5, 0.5));
        model_right = rotate_z_deg(model_right, -90 + bone_right_rotate_z);
        model_right = translate(model_right, vec3(-1.05, 0.0, 0.0));
        model_right = model[i-1] * model_right;
        shader.setMat4("model", model_right);
        renderBone();

    }

    // top1
    modelTop1 = identity_mat4();
    modelTop1 = scale(modelTop1, vec3(0.5, 0.5, 0.5));
    modelTop1 = rotate_z_deg(modelTop1, boneTop1_start_z_deg + boneTop1_rotate_z);
    modelTop1 = translate(modelTop1, vec3(-1.05, 0.0, 0.0));
    modelTop1 = model[numsOfBones-1] * modelTop1;
    shader.setMat4("model", modelTop1);
    renderBone();

    // top2
    modelTop2 = identity_mat4();
    modelTop2 = scale(modelTop2, vec3(0.5, 0.5, 0.5));
    modelTop2 = rotate_z_deg(modelTop2, boneTop2_start_z_deg + boneTop2_rotate_z);
    modelTop2 = translate(modelTop2, vec3(-1.05, 0.0, 0.0));
    modelTop2 = model[numsOfBones-1] * modelTop2;

    shader.setMat4("model", modelTop2);
    renderBone();

    glutSwapBuffers();
}



void CCD(int frame) {
```

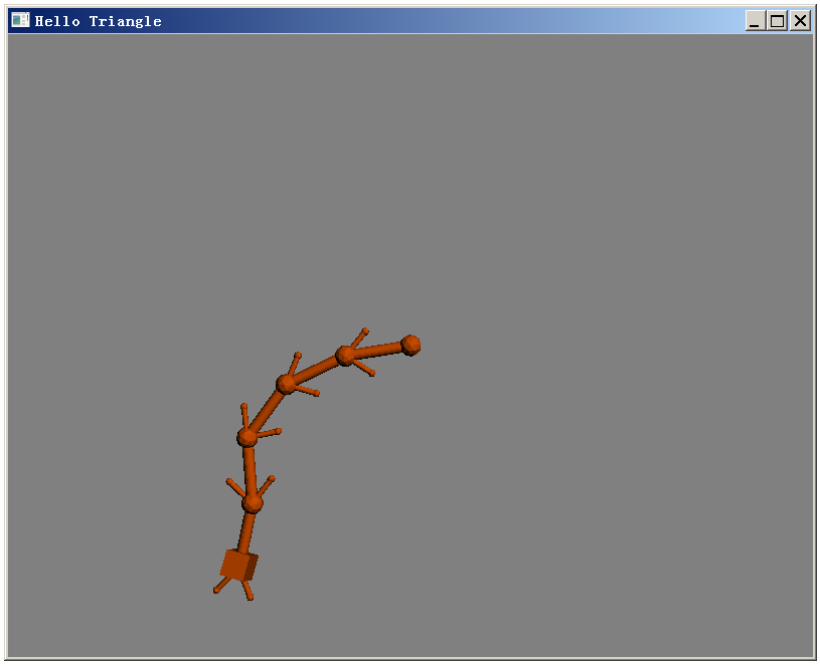
```
    static int i = 0;
    if (++i > numsOfBones * 30) {
        i = 0;
        CCDRunning = false;
        return;
    }

    vec3 endPos = getPos(modelTop1);
    vec3 tarPos = getPos(modelTarget);
    vec3 curPos;
    vec3 e_c;
    vec3 t_c;
    vec3 r;
    float cos_theta;
    float angle;

    float distance = sqrt(pow((tarPos.v[0] - endPos.v[0]), 2) + pow((tarPos.v[1] -
endPos.v[1]), 2));
    float tolerance = 0.1;

    if (distance < tolerance) {
        solve = true;
        CCDRunning = false;
    }
    else
    {

        int frameIndex = numsOfBones - frame - 1;

        for (int i = numsOfBones - 1; i >= 0; i--)
        {
            if (frameIndex == i) {
                endPos = getPos(modelTop1);
                tarPos = getPos(modelTarget);
                curPos = getPos(model[i]);
                e_c = normalise(endPos - curPos);
                t_c = normalise(tarPos - curPos);
                r = cross(e_c, t_c);
                cos_theta = clamp(dot(e_c, t_c));
                angle = acos(cos_theta) * ONE_RAD_IN_DEG;

                if (r.v[2] < 0) {
                    angle *= -1;
                }
```

```
                bone_rotate_z[i] += angle;
            }
        }
    }
}
```

Required feature: Scripted animation

Screenshot(s) of feature:





Describe your implementation:

1. The small bone（6 – 15），can animated at 45 degree.

2. The big bone（1，2，3，4，5），can animated at 360 fegree.

   press z or x, to animate bone 1
   press a or s, to animate bone 2
   press q or w, to animate bone 3
   press d or f, to animate bone 4
   press e or r, to animate bone 5

3. IK with CCD
   press c to animate

Code Snippet:

```
void updateScene() {

    static DWORD last_time = 0;
```

```
    DWORD curr_time = timeGetTime();
    if (last_time == 0)
        last_time = curr_time;
    float delta = (curr_time - last_time) * 0.001f;
    last_time = curr_time;

    // animte the target
    rotate_y += 20.0f * delta;
    rotate_x += 20.0f * delta;
    rotate_z += 20.0f * delta;

    // animte the bone
    boneTop1_rotate_z += 20.0f * delta;
    boneTop1_rotate_z = fmodf(boneTop1_rotate_z, 45.0f);
    boneTop2_rotate_z -= 20.0f * delta;
    boneTop2_rotate_z = fmodf(boneTop2_rotate_z, -45.0f);

    // animte the bone
    bone_left_rotate_z += 70.0f * delta;
    bone_left_rotate_z = fmodf(bone_left_rotate_z, 90.0f);
    bone_right_rotate_z -= 70.0f * delta;
    bone_right_rotate_z = fmodf(bone_right_rotate_z, -90.0f);

    // CCD and animation
    if (CCDRunning) {
        static int frame = 0;
        CCD(frame);
        frame++;
        if (frame > numsOfBones) frame = 0;
    }

    // Draw the next frame
    glutPostRedisplay();
}
```