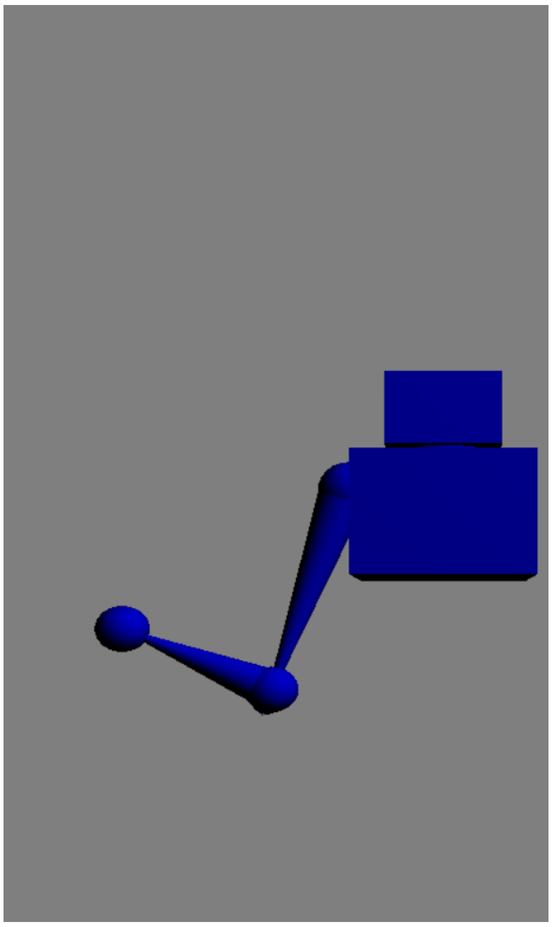
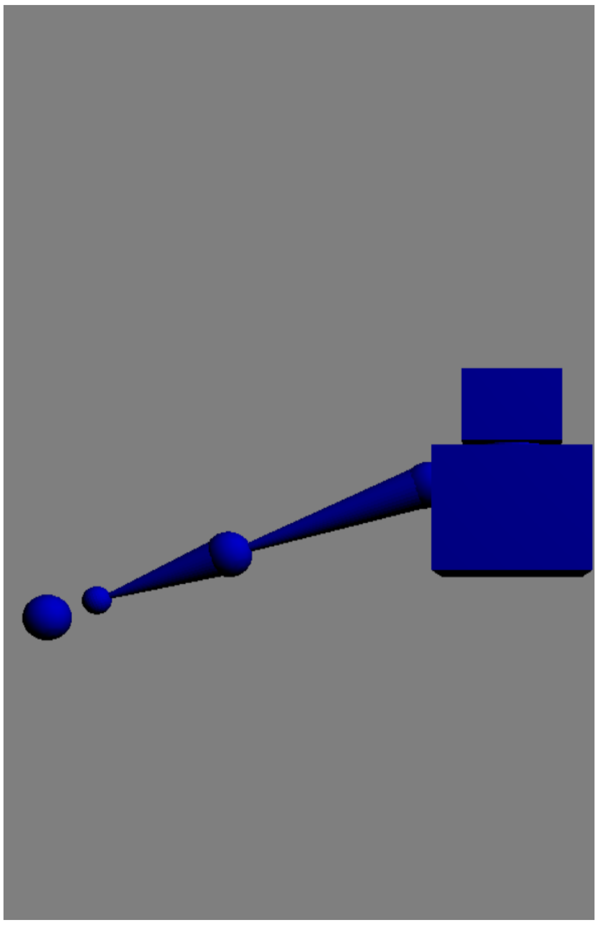
CS7GV5 Report Assignment 1

LINK TO VIDEO DEMONSTRATION →https://youtu.be/lDo0Eb9rqVs

Name:	Liam Byrne
Student ID:	18326579

Required feature: Simple 2-bone IK in 2D

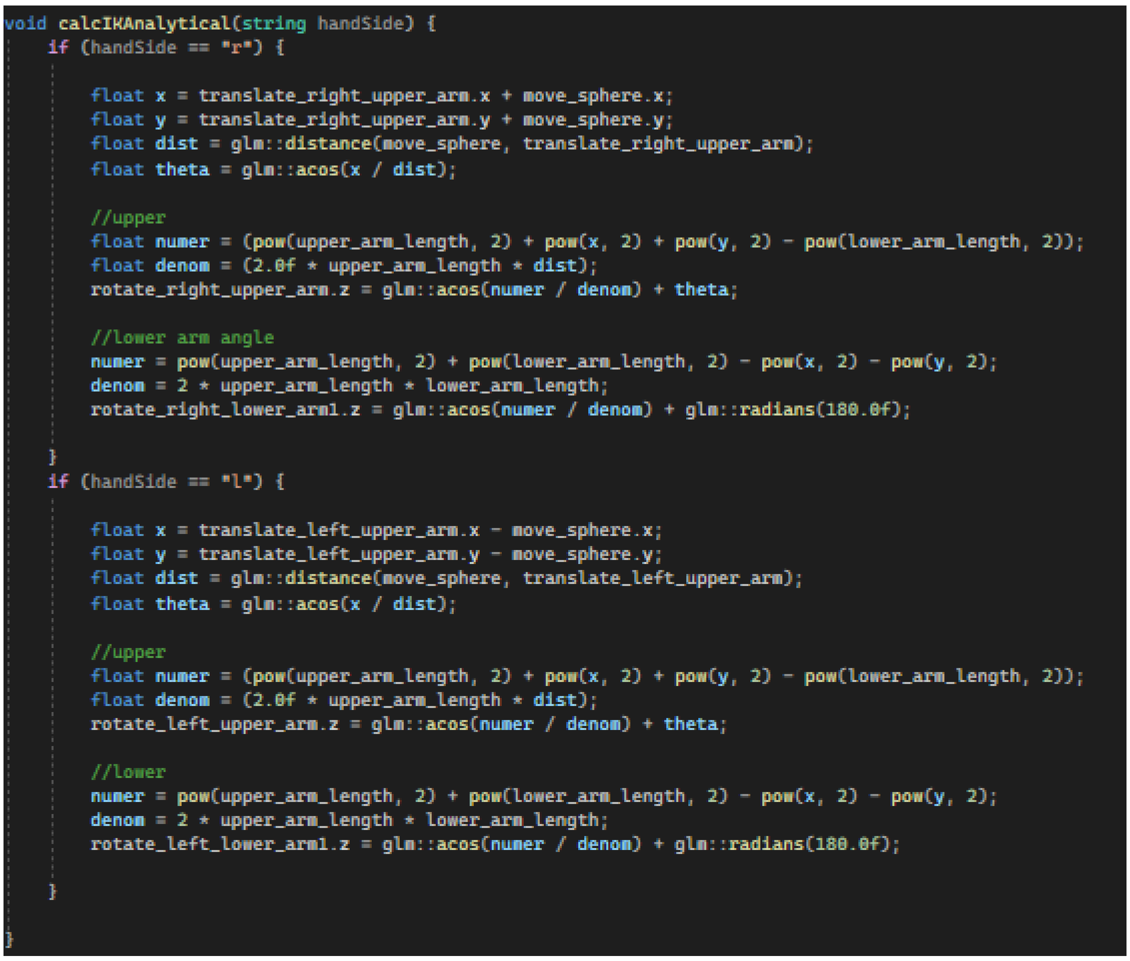
Screenshot(s) of feature:

Describe your implementation:
The code performs inverse kinematics by calculating the angles required for the upper and lower arms of a robotic arm to reach a specific point in space. It starts by computing the coordinates (x, y) of the target point relative to the upper arm's base. The distance from the upper arm's base to the target point is also computed using the glm::distance() function. Next, the angle theta is calculated using the arccosine function, which is used to find the angle between the line connecting the upper arm's base to the target point and the x-axis.

The upper arm's angle is then computed using the law of cosines, given the lengths of the upper and lower arms and the distance to the target point. Finally, the angle for the lower arm is calculated using the same law of cosines formula, but with the lengths of the upper and lower arms swapped. The calculated angles are stored in rotation variables for the upper and lower arms of the specified hand, in this case, "rotate_right_upper_arm.z" and "rotate_right_lower_arm1.z".

Code Snippet:



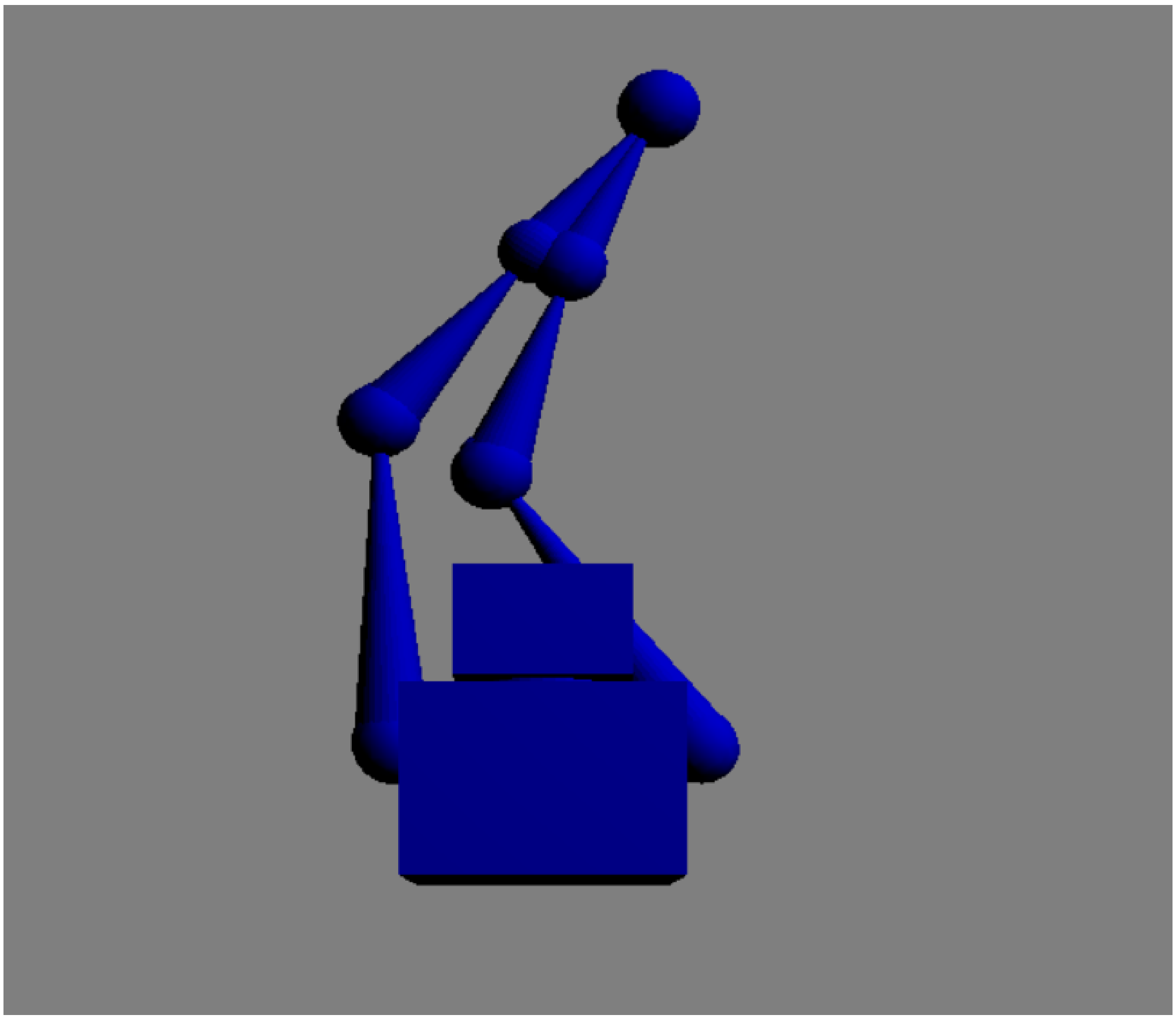
```
void calcIKAnalytical(string handSide) {
    if (handSide == "r") {

        float x = translate_right_upper_arm.x + move_sphere.x;
        float y = translate_right_upper_arm.y + move_sphere.y;
        float dist = glm::distance(move_sphere, translate_right_upper_arm);
        float theta = glm::acos(x / dist);

        //upper
        float numer = (pow(upper_arm_length, 2) + pow(x, 2) + pow(y, 2) - pow(lower_arm_length, 2));
        float denom = (2.0f * upper_arm_length * dist);
        rotate_right_upper_arm.z = glm::acos(numer / denom) + theta;

        //lower arm angle
        numer = pow(upper_arm_length, 2) + pow(lower_arm_length, 2) - pow(x, 2) - pow(y, 2);
        denom = 2 * upper_arm_length * lower_arm_length;
        rotate_right_lower_arm1.z = glm::acos(numer / denom) + glm::radians(180.0f);

    }
    if (handSide == "l") {

        float x = translate_left_upper_arm.x - move_sphere.x;
        float y = translate_left_upper_arm.y - move_sphere.y;
        float dist = glm::distance(move_sphere, translate_left_upper_arm);
        float theta = glm::acos(x / dist);

        //upper
        float numer = (pow(upper_arm_length, 2) + pow(x, 2) + pow(y, 2) - pow(lower_arm_length, 2));
        float denom = (2.0f * upper_arm_length * dist);
        rotate_left_upper_arm.z = glm::acos(numer / denom) + theta;

        //lower
        numer = pow(upper_arm_length, 2) + pow(lower_arm_length, 2) - pow(x, 2) - pow(y, 2);
        denom = 2 * upper_arm_length * lower_arm_length;
        rotate_left_lower_arm1.z = glm::acos(numer / denom) + glm::radians(180.0f);

    }

}
```

Extra Feature 1: : Multi-bone IK in 3D

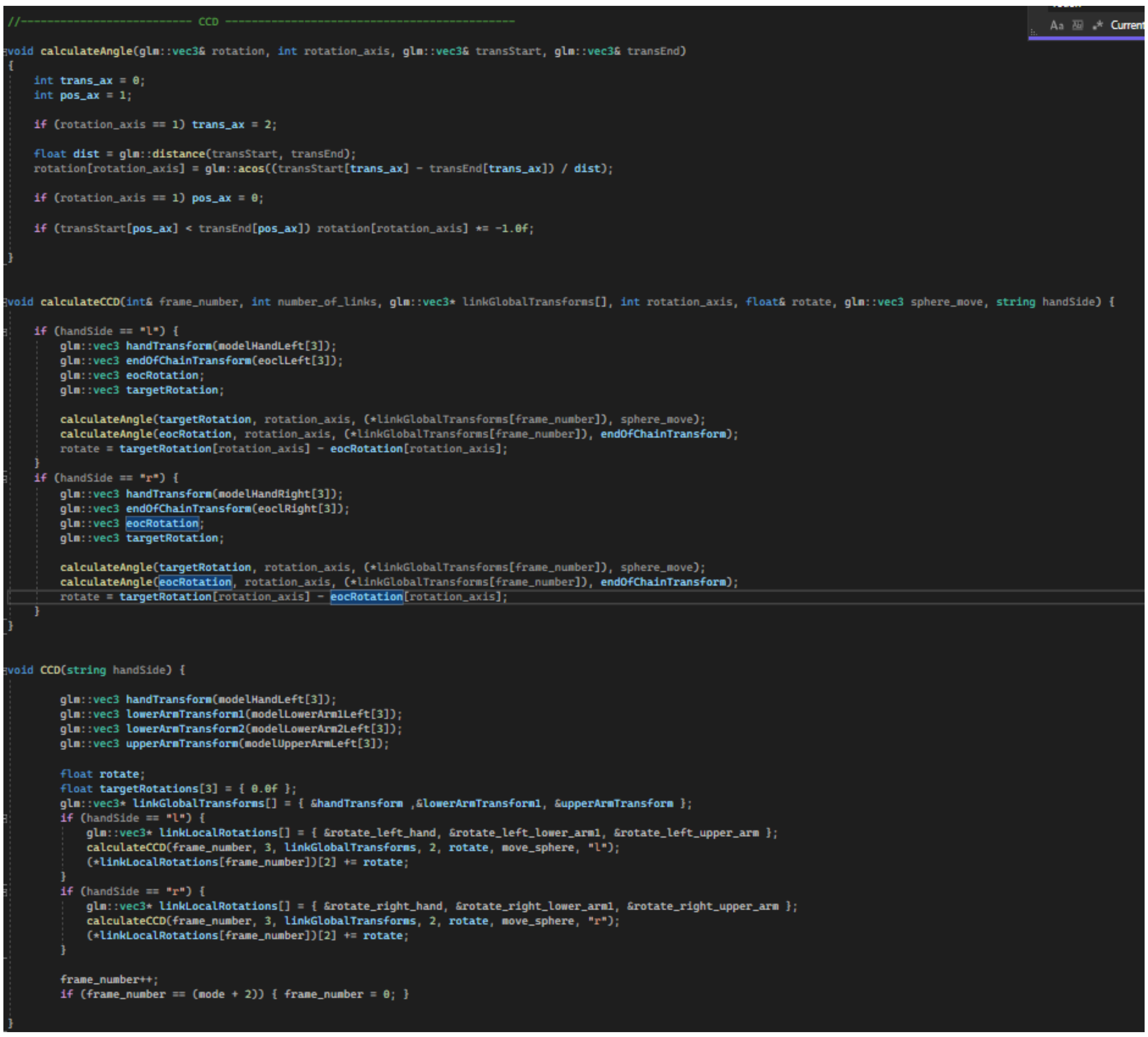
Describe your implementation:

The input parameters of the CCD function are the side of the hand ("l" for left or "r" for right).The calculateAngle function is used to calculate the angle that each joint needs to rotate to align its axis with the vector that connects its current position to the target position. The calculateCCD function uses calculateAngle to calculate the rotation angle for the joint. It then subtracts the current joint orientation from the target orientation to get the rotation angle. The CCD function updates the rotation angles for each joint in the chain by calling calculateCCD.

Screenshot(s) of feature:



Code Snippet(s):



```
//------------------------ CCD ----------------------------------------

void calculateAngle(glm::vec3& rotation, int rotation_axis, glm::vec3& transStart, glm::vec3& transEnd)
{
    int trans_ax = 0;
    int pos_ax = 1;

    if (rotation_axis == 1) trans_ax = 2;

    float dist = glm::distance(transStart, transEnd);
    rotation[rotation_axis] = glm::acos((transStart[trans_ax] - transEnd[trans_ax]) / dist);

    if (rotation_axis == 1) pos_ax = 0;

    if (transStart[pos_ax] < transEnd[pos_ax]) rotation[rotation_axis] *= -1.0f;

}


void calculateCCD(int& frame_number, int number_of_links, glm::vec3* linkGlobalTransforms[], int rotation_axis, float& rotate, glm::vec3 sphere_move, string handSide) {

    if (handSide == "l") {
        glm::vec3 handTransform(modelHandLeft[3]);
        glm::vec3 endOfChainTransform(eoclLeft[3]);
        glm::vec3 eocRotation;
        glm::vec3 targetRotation;

        calculateAngle(targetRotation, rotation_axis, (*linkGlobalTransforms[frame_number]), sphere_move);
        calculateAngle(eocRotation, rotation_axis, (*linkGlobalTransforms[frame_number]), endOfChainTransform);
        rotate = targetRotation[rotation_axis] - eocRotation[rotation_axis];
    }
    if (handSide == "r") {
        glm::vec3 handTransform(modelHandRight[3]);
        glm::vec3 endOfChainTransform(eoclRight[3]);
        glm::vec3 eocRotation;
        glm::vec3 targetRotation;

        calculateAngle(targetRotation, rotation_axis, (*linkGlobalTransforms[frame_number]), sphere_move);
        calculateAngle(eocRotation, rotation_axis, (*linkGlobalTransforms[frame_number]), endOfChainTransform);
        rotate = targetRotation[rotation_axis] - eocRotation[rotation_axis];
    }
}


void CCD(string handSide) {

        glm::vec3 handTransform(modelHandLeft[3]);
        glm::vec3 lowerArmTransform1(modelLowerArm1Left[3]);
        glm::vec3 lowerArmTransform2(modelLowerArm2Left[3]);
        glm::vec3 upperArmTransform(modelUpperArmLeft[3]);

        float rotate;
        float targetRotations[3] = { 0.0f };
        glm::vec3* linkGlobalTransforms[] = { &handTransform ,&lowerArmTransform1, &upperArmTransform };
        if (handSide == "l") {
            glm::vec3* linkLocalRotations[] = { &rotate_left_hand, &rotate_left_lower_arm1, &rotate_left_upper_arm };
            calculateCCD(frame_number, 3, linkGlobalTransforms, 2, rotate, move_sphere, "l");
            (*linkLocalRotations[frame_number])[2] += rotate;
        }
        if (handSide == "r") {
            glm::vec3* linkLocalRotations[] = { &rotate_right_hand, &rotate_right_lower_arm1, &rotate_right_upper_arm };
            calculateCCD(frame_number, 3, linkGlobalTransforms, 2, rotate, move_sphere, "r");
            (*linkLocalRotations[frame_number])[2] += rotate;
        }

        frame_number++;
        if (frame_number == (mode + 2)) { frame_number = 0; }

}
```