Real Time Animation – Assignment 1

Cormac Doyle

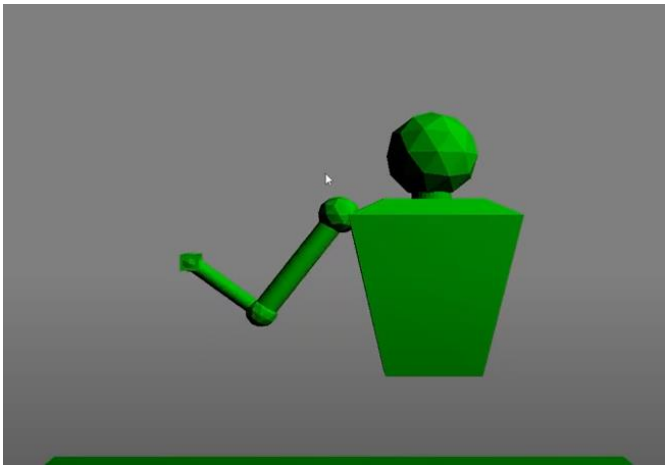
17334477

Youtube Demo

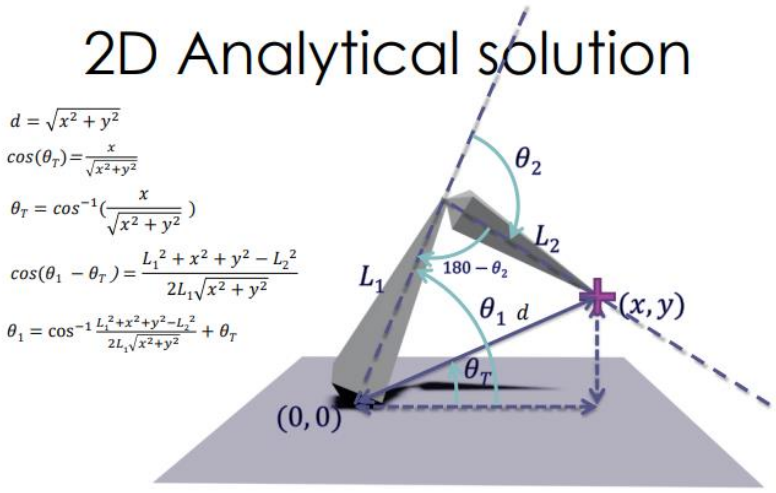
GitHub Repo

All Models were made by myself using blender

Required Feature: Simple 2-bone IK in 2D



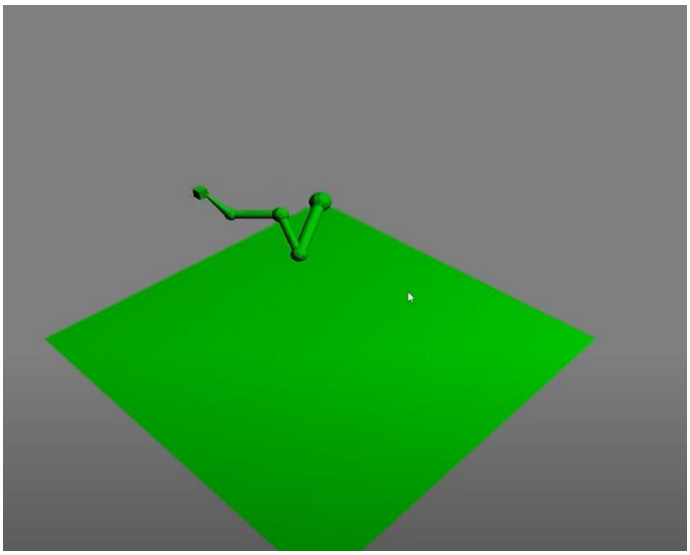
Following the equations shown in lectures:



$$\cos(180 - \theta_2) = -\cos\theta_2 = \frac{L_1^2 + L_2^2 - (x^2 + y^2)}{2L_1L_2} \qquad \theta_2 = \cos^{-1}\frac{L_1^2 + L_2^2 - x^2 + y^2}{2L_1L_2}$$

```
void calcIKAnalytical() {
    float x = translate_upper_arm.x - transform_target.x;
    float y = translate_upper_arm.y - transform_target.y;
    float dist = glm::distance(transform_target, translate_upper_arm);
    float thetaT = glm::acos(x / dist);

    //upper arm angle
    float numerator = (pow(upper_arm_length, 2) + pow(x, 2) + pow(y, 2) - pow(lower_arm_length, 2));
    float denominator = (2.0f * upper_arm_length * dist);

    rotate_upper_arm.z = glm::acos(numerator / denominator) + thetaT;

    //lower arm angle
    numerator = pow(upper_arm_length, 2) + pow(lower_arm_length, 2) - pow(x, 2) - pow(y, 2);
    denominator = 2 * upper_arm_length * lower_arm_length;

    rotate_lower_arm1.z = glm::acos(numerator / denominator) + glm::radians(180.0f);

}
```

Required feature: Multi-bone IK in 3D



A CCD algorithm is used.

Starting at the end of the chain, the angle between the current link and the end of chain is directed at the transform of the target vector. This is repeated for every link in the chain.

I found this video very helpful: CCD Demo

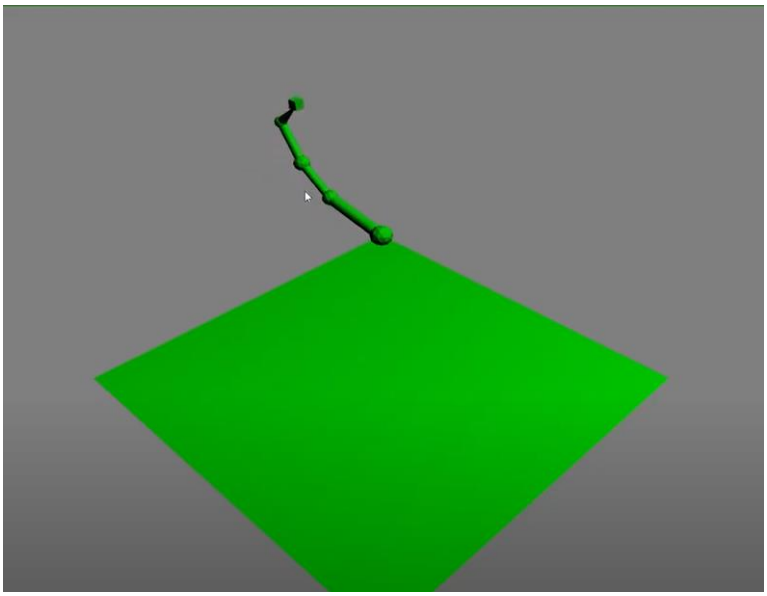
In my implementation I calculated the angle of one link per frame. This led to an interpolated effect of the chain following the target. However, because only one angle was being changed per frame, if the target vector was very far and CCD algorithm needed a lot of repetitions to cycle through, it can take a delayed amount of time for the chain to find the target.

For the 3D IK rotations were calculated for the Z and Y axis and quaternions were used to apply the rotations.

```cpp
//based off whats seen in this video: https://www.youtube.com/shorts/MvuO9ZHGr6k
//each frame a different link angle is calculated, this way no 'for' or 'while' loop is necessary and it results in a interpolated effect

void calcCCD(int& frame_number, int number_of_links, glm::vec3* linkGlobalTransforms[],int rotation_axis, float& rotateBy, glm::vec3 target_transform) {
    glm::vec3 handTransform(modelHand[3]);

    glm::vec3 endOfChainTransform(modelEndOfChain[3]);

    glm::vec3 endOfChainRotation;


    glm::vec3 targetRotation;
    calcAngle(targetRotation, rotation_axis, (*linkGlobalTransforms[frame_number]), target_transform);
    calcAngle(endOfChainRotation, rotation_axis, (*linkGlobalTransforms[frame_number]), endOfChainTransform);
    rotateBy = targetRotation[rotation_axis] - endOfChainRotation[rotation_axis];

}
```

```cpp
void calcAngle(glm::vec3& rotation, int rotation_axis, glm::vec3& transformStart, glm::vec3& transformEnd)
{
    int transform_axis = 0;
    if (rotation_axis == 1) {
        transform_axis = 2;
    }

    float dist = glm::distance(transformStart, transformEnd);
    rotation[rotation_axis] = glm::acos((transformStart[transform_axis] - transformEnd[transform_axis]) / dist);

    int position_axis = 1;
    if (rotation_axis == 1) {
        position_axis = 0;
    }

    if (transformStart[position_axis] < transformEnd[position_axis]) {
        rotation[rotation_axis] *= -1.0f;
    }

}
```

Required feature: Scripted animation



Following a parametric equation found here for ease-in ease-out curves. A smooth animation is calculated between points in a list of vectors. The direction or the x or y axis is changed accordingly depending on the next point in the path.

```
//parametric equation to join splines found here :
//https://math.stackexchange.com/questions/121720/ease-in-out-function/121755#121755

glm::vec3 animationPath[] = {
    glm::vec3(-5.0f,0,0),
    glm::vec3(0,4.0f,0),
    glm::vec3(4.0f,2.0f, 0.0f),
    glm::vec3(3.0f,-4.0f,0.0f),
    glm::vec3(-2.0f,1.0f,0.0f),
    glm::vec3(-5.0f,0,0)
};

void animateTarget()
{

    curveY = pow(curveX,2) / (2.0f * (pow(curveX, 2) - curveX) + 1.0f);

    lenX = (animationPath[spline_index].x - animationPath[spline_index - 1].x);
    lenY = (animationPath[spline_index].y - animationPath[spline_index - 1].y);

    if (animationPath[spline_index].y > animationPath[spline_index - 1].y) {
        transform_target.y = curveY * lenY + animationPath[spline_index-1].y;
    }
    else {
        transform_target.y = (1-curveY) * -lenY + animationPath[spline_index].y;
    }

    transform_target.x += 0.0015f * lenX;
    curveX += 0.0015f;

    if (abs(glm::distance(transform_target,animationPath[spline_index])) < 0.005f ) {
        moveToNextinPath();
    }

}
```