HW3

0856631 陳立軒

1 Introduction

In computer animation, inverse kinematics is the mathematical process of calculating the variable joint parameters needed to place the end of a kinematic chain, such as a robot manipulator or animation character's skeleton.

2 Fundamentals

Kinematics is the study of motion without considering the cause of the motion, such as forces and torques. Inverse kinematics is a method that helps define the motion of a robot to reach a desired location.

Although we can adopt analytic method such as cosine low to solve IK, the complexity increases dramatically and the formula should be derived again when the linage become complicated. In this homework, we are asked to implement IK via inverse-Jacobian method.

$$\theta = f^{-1}(p)$$
$$V = J(\theta)\dot{\theta}$$
$$J^T V = J^T J\dot{\theta}$$
$$(J^T J)^{-1} J^T V = (J^T J)^{-1} J^T J\dot{\theta}$$
$$\because (J^T J)^{-1} J^T = J^+$$
$$\therefore \dot{\theta} = J^+ V$$
$$\Longrightarrow \theta_{k+1} = \theta_k + \Delta t * J^+(\theta_k) V$$

3 Implementation

3.1 Pseudo Inverse Solver

```cpp
math::VectorNd_t PseudoinverseSolver::Solve(
        const math::MatrixN_t &coef_mat,
        const math::VectorNd_t &desired_vector
        ) const
{
    VectorNd_t angularVector(coef_mat.cols());
    MatrixN_t JacobianInverse(coef_mat.cols(), coef_mat.rows());

    Eigen::JacobiSVD<math::MatrixN_t> svd(coef_mat, Eigen::ComputeThinU | Eigen::ComputeThinV);
    Eigen::MatrixXd sigma(3, 3);

    sigma.setIdentity();
    sigma(0, 0) = svd.singularValues()[0];
    sigma(1, 1) = svd.singularValues()[1];
    sigma(2, 2) = svd.singularValues()[2];

    JacobianInverse = svd.matrixV() * sigma.inverse() * svd.matrixU().transpose();
    angularVector = JacobianInverse * desired_vector;

    return angularVector;
}
```

3.2 Inverse Jacobian IK Solver

```cpp
// Compute Jacobian
while (tempBoneIdx != skeleton_->bone_ptr(start_bone_idx)->parent->idx) {
    // (p - ri)
    math::Vector3d_t r = temp[end_bone_idx].end_pos() - temp[tempBoneIdx].start_pos();
    bool x = false, y = false, z = false;
    for (int colIdx = 0; colIdx < skeleton_->bone_ptr(tempBoneIdx)->dof;) {
        if (skeleton_->bone_ptr(tempBoneIdx)->dofx && !x) {
            Jacobian.col(colIdx + dofInc) = temp[tempBoneIdx].rotation().col(0).cross(r);
            colIdx++;
            x = true;
        }
        else if (skeleton_->bone_ptr(tempBoneIdx)->dofy && y != true) {
            Jacobian.col(colIdx + dofInc) = temp[tempBoneIdx].rotation().col(1).cross(r);
            colIdx++;
            y = true;
        }
        else if (skeleton_->bone_ptr(tempBoneIdx)->dofz && z != true) {
            Jacobian.col(colIdx + dofInc) = temp[tempBoneIdx].rotation().col(2).cross(r);
            colIdx++;
            z = true;
        }
    }
    dofInc += skeleton_->bone_ptr(tempBoneIdx)->dof;
    tempBoneIdx = skeleton_->bone_ptr(tempBoneIdx)->parent->idx;
}
```

```
while (tempBoneIdx != skeleton_->bone_ptr(start_bone_idx)->parent->idx){
    math::Vector3d_t angularVector = bodyPos6d[tempBoneIdx].angular_vector();
    bool x = false, y = false, z = false;
    for (int colIdx = 0; colIdx < skeleton_->bone_ptr(tempBoneIdx)->dof;) {
        if (skeleton_->bone_ptr(tempBoneIdx)->dofx && x != true) {
            angularVector[0] += deltaSita[0 + dofInc] * step_;
            colIdx++;
            x = true;
        }
        else if (skeleton_->bone_ptr(tempBoneIdx)->dofy && y != true) {
            angularVector[1] += deltaSita[1 + dofInc] * step_;
            colIdx++;
            y = true;
        }
        else if (skeleton_->bone_ptr(tempBoneIdx)->dofz && z != true) {
            angularVector[2] += deltaSita[2 + dofInc] * step_;
            colIdx++;
            z = true;
        }
    }
    bodyPos6d[tempBoneIdx].set_angular_vector(angularVector);
    dofInc += skeleton_->bone_ptr(tempBoneIdx)->dof;
    tempBoneIdx = skeleton_->bone_ptr(tempBoneIdx)->parent->idx;
}
```

4 Discussion

4.1 Effects of step

$$\theta_{k+1} = \theta_k + \Delta t * J^{+}(\theta_k)V$$

If increasing the step, that is to say, increasing the value of Δt, the pose of the skeleton may be unreal since the system equation may diverge.

4.2 Effects of distance_epsilon

If increasing the distance_epsilon, the error will increase since the distance epsilon is the terminal condition for iterative pseudo inverse kinematics.

5 Conclusion

Inverse kinematics is more complex than forward kinematics due to its properties such as redundancy and matrix singularity, but it plays an indispensable part in computer animation.