CIS 462/CIS562 Computer Animation (Fall 2018)
Homework Assignment 6
(Inverse Kinematics)

Due: Tuesday, Nov. 6, 2018 (must be submitted to Canvas before midnight)

Please note the following:

- No late submissions or extensions for this homework.
- This is a programming assignment. You will need to use the *AnimationToolkit* code framework from the previous CIS462/562 BVH Player assignment
- Double click on the file " Demo-IKViewer.exe" in the AnimationToolkit\bin directory to see a demo of the Inverse Kinematics assignment functionality
- Commit and push the updated project files to your CIS462-562 GitHub repository.
- When you are finished with this assignment, update your GitHub project files.
- Work within the *AnimationToolkit* code framework provided. Feel free to enhance the GUI interface if you desire.
- You only need to implement the functions in the aIKController.cpp files marked with "TODO" to complete this assignment.
- **NOTE: THIS IS AN INDIVIDUAL, NOT A GROUP ASSIGNMENT. That means all code written by you for this assignment should be original! Although you are permitted to consult with each other while working on this assignment, code that is substantially the same as that submitted by another student will be considered cheating.**

ANIMATION TOOLKIT – INVERSE KINEMATICS

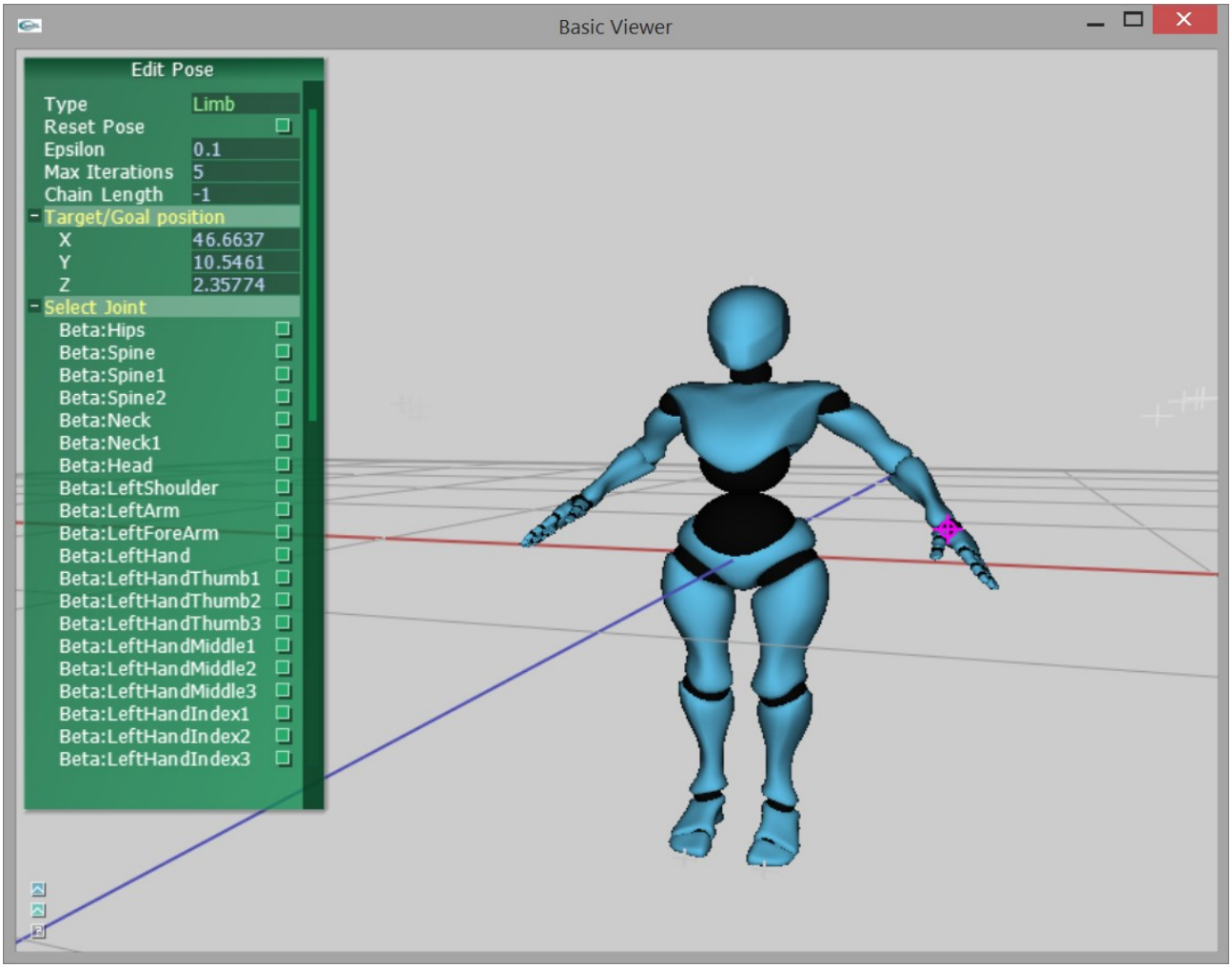
This is the third of a three-part assignment. In part 2a of the assignment you implemented various orientation conversion functions as well as quaternion interpolation using linear and cubic splines. In part 2b of the assignment you implemented forward kinematics using homogeneous transformations which allowed loading a BVH motion capture file and playing back the associated animation on the Beta character. In this part of the assignment you will implement the analytic (i.e. limb-based) and CCD (i.e. cyclic coordinate descent) forms of inverse kinematics in order to control the position of the hands and feet of the Beta character.

About the AnimationToolkit Basecode

This assignment builds on the previous three assignments by adding a new library and application. If you did not complete or had problems with the implementations in assignments 1 or 2a, you can edit the current IKViewer project file to use the associated library file solutions for assignments 1 or 2a. To do this, right click on the IKViewer project in VS2015, select Properties, then expand the Linker options. Select "Input" and then edit the "Additional Libraries" line so it includes either libAssignment1-soln.lib, libAssignment2a-soln.lib and/or libAssignment2a-soln.lib (for the release version) and the same files with libAssignmentx–soln**d**.lib for the debug version.

IKViewer User Interface Overview

The AnimationToolkit basecode for assignment 2c includes a simple interface and 3D viewer as shown in the screenshot below for selecting individual joints (which are known as the "end joint") and moving a red circle (known as the "target") which represents the desired location of the end joint. The default joint selected in the IKviewer is the LeftHand.



The camera control in the IKViewer is the same as in the BVHViewer:

- Left-button drag with the mouse to rotate
- Right-button drag with the mouse to pan
- Middle-button drag with the mouse to zoom
- 'f" button will focus the view on the character

The control panel in the top left-hand corner of the viewer window can be used to select different IK methods (i.e. Limb, CCD, PseudoInv and Other), set the error threshold (epsilon) and the max

number of iterations for the CDD method. The control panel at the bottom left can be used to select different end joints for the IK control of the Beta character. As mentioned previously, the target (i.e. goal) location of the end joint is represented as a red circle. The position of the target can be modified by holding down the control key and left clicking on the red circle with the mouse.

Forward and Inverse Kinematics Implementation Overview

The AnimationToolkit basecode includes a framework for organizing transforms into a hierarchy and animating them using curves. The core data structure for supporting this functionality is **AJoint**, which maintains pointers to parent and child transforms. **AJoint** primarily contains an **ATransform** which keeps track of the joints position and orientation relative to its parent. It also stores its transform relative to the world coordinate system for convenience.

Hierarchies of **AJoint** joint objects can be created using the **ASkeleton** class. The root joint of the **ASkeleton** hierarchy is positioned relative to the world coordinate system, while all other joints are positioned and oriented relative to its parent in the skeleton joint hierarchy. Although the **ASkeleton** class can be used to animate any group of transforms, for character animation it is natural to think of the **ASkeleton** as the base class of an **AActor** class, which also contains objects called "Controllers" that are used to create and/or set joint transform values as a function of time. The **IKController** class implements various IK methods that allow selected end joints of the character to be interactively positioned in space. In this assignment, you will need to complete the functions `computeLimbIK` and `computeCCDIk` which implement the Analytical (i.e. Limb-based) and CCD methods. You will also need to complete the implementation of the `createIKchain` function which finds the chain of joints starting at the end joint of a desired length.

To summarize, the organization of the class containment in this assignment is the following:

AActor – holds the character skeleton and IKController
ASkeleton – holds the hierarchy of joints of the character
AJoint – holds the local and global joint transforms
ATransform
IKController
ATarget - holds the desired position and orientation of the end joint
AIKChain - holds a vector of joint pointers and a vector of weight values

Assignment Details

1. (20 points) **createIKchain.** In this part of the assignment you need to complete the implementation of the `createIKchain` function which finds a sequence of joints between the end joint and root joint of a desired length.

2. (40 points) **computeLimbIK.** In this part of the assignment you need to complete the `computeLimbIK` function which implements the Analytical IK method.

2. (40 points) **computeCCDIK.** In this part of the assignment you need to complete the `computeCCDIK` function which implements the CCD IK method.

3. ***Extra Credit*** (25 points) **IKSolver_PseudoInv.** For this part of the assignment you need to implement a pseudo inverse-based IK method. The matrix math functions and operators in aMatrix.h can be used to construct the Jacobean matrix and compute its pseudo inverse. Note: currently the maximum matrix dimension is 25. To change this, modify the MATDIM parameter accordingly.

4. ***Extra Credit*** (25 points) **IKSolver_Other.** In this part of the assignment you can combine the Limb and CCD methods to allow the joints between the shoulder and root to contribute to the IK solution when the location of the target is out of reach of an arm.