CIS 462/CIS562 Computer Animation (Fall 2019)
Homework Assignment 5
(Forward and Inverse Kinematics)

Due: Monday, Nov. 4, 2019 (by 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 Rotations assignment
- Double click on the files " Demo-BVHViewer.exe" and Demo-IKViewer.exe in the AnimationToolkit\bin directory to see a demo of the Forward and 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 aTransform.cpp, aJoint.cpp, aSkeleton.cpp, aBVHController.cpp and 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

This is the second of a three-part assignment.

Part 2a – Rotations

Previously, in Part 2a of the assignment you implemented various orientation conversion functions as well as quaternion interpolation using linear and cubic splines.

Part 2b – Forward Kinematics

In Part 2b of the assignment you will implement forward kinematics using homogeneous transformations. The features you implement will lay the groundwork for animating characters using motion capture data and keyframe splines in the subsequent assignments. Specifically, after completing this part of the assignment, you will be able to load a BVH motion capture file and play back the animation on a character.

Part 2c – Inverse Kinematics

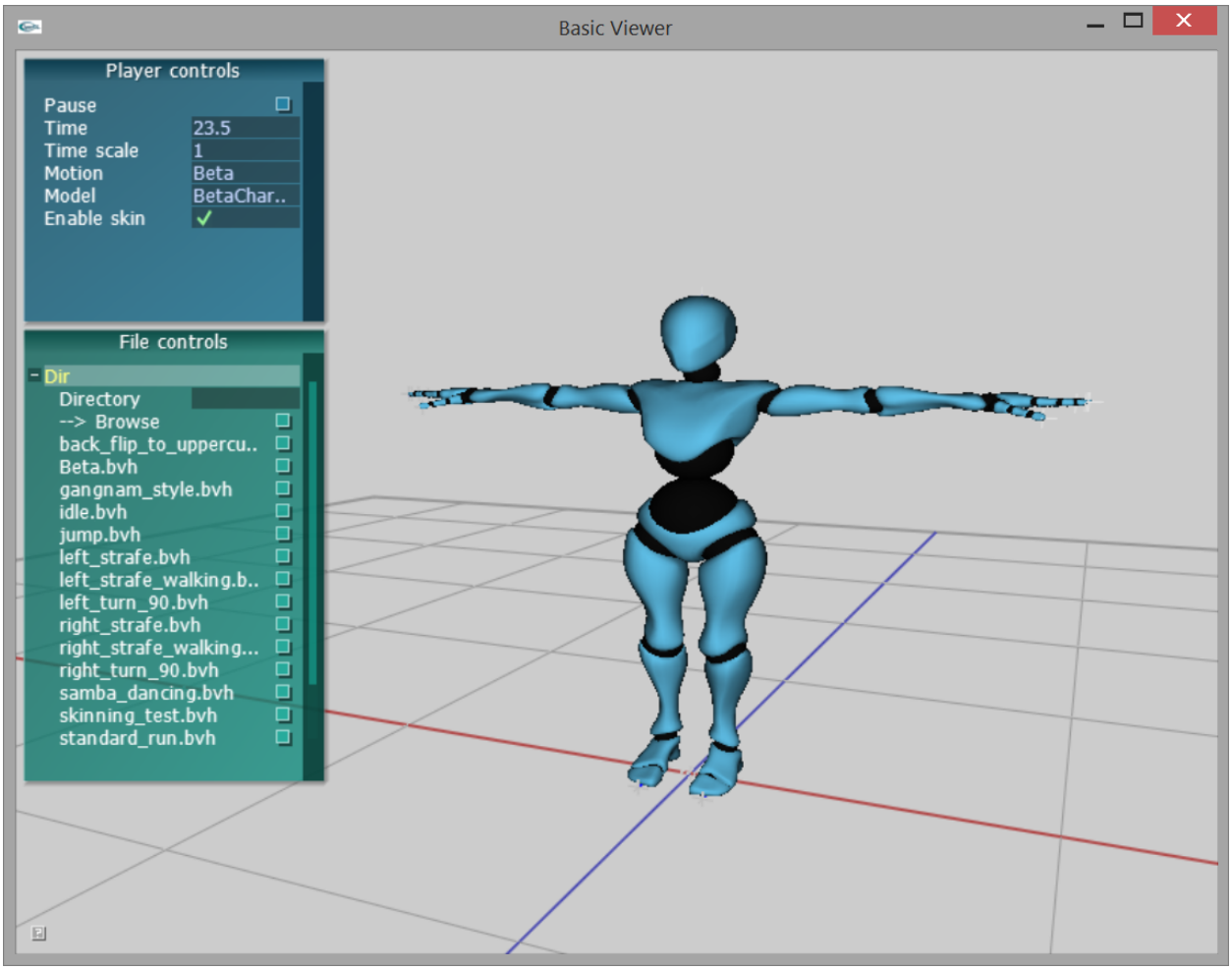
In Part 2c of the assignment you will implement the limb-based (i.e. geometric) and CCD (i.e. cyclic coordinate descent) forms of inverse kinematics in order to control the position of the hands and feet of the Beta character.

About the AnimationToolkit Basecode

This assignment builds on the previous two programming assignments by adding two new libraries and applications. If you did not complete or had problems with the implementations in assignments 1 and 2a, you can edit the current BVHViewer project file to use the associated library file solutions for assignments 1 and 2a. To do this, right click on the BVHViewer project in Visual Studio, select Properties, then expand the Linker options. Select "Input" and then edit the "Additional Libraries" line so it includes libAssignment1-soln.lib and libAssignment2a-soln.lib.

BVHViewer User Interface Overview – Forward Kinematics

The AnimationToolkit basecode for the forward kinematics (part 2b) of the assignment includes a simple interface and 3D viewer for loading and playing BVH files as shown in the screenshot below. By default, the BVHViewer loads the beta character and a number of associated animations (stored in the motions directory in files using the BVH format). You can add your own BVH files to the motions directory if you like.



The camera can be controlled in the BVHViewer using the mouse:

- 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 load different characters (in the fbx format). The control panel at the bottom left can be used to load and select the associated character animation files (in the BVH format). The character animation can be turned on/off by pressing the play button.

Forward 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 joint's 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 could 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. In this assignment, the skeleton transform data will be read from a BVH file. This is accomplished using a **BVHController** which creates and initializes spline curves for each joint based on the motion data contained in *.bvh animation files. Since in this assignment the bones of the skeleton are of constant length, the BVHController contains only a single ASplineVec3 spline curve (stored in the member variable mRootMotion) for the root motion translations. Given that all the joints in the skeleton can rotate (including the root), the BVHController also stores ASplineQuat quaternion spline curves for every joint (in the member variable mMotion). During playback, the BVHController gets the values from the animation curves stored in mRootMotion and mMotion and updates the joint transforms of the skeleton accordingly.

To summarize, the organization of the class containment in part 2b of the assignment is the following:

AActor – holds the character skeleton and BVHController
  ASkeleton – holds the hierarchy of joints of the character
    AJoint – holds the local and global joint transforms
      ATransform
  BVHController
    ASplineVec3 - holds the root joint position spline
    ASplineQuat - holds the quaternion splines for each joint orienation

Part 2b - Forward Kinematics To Dos (*50 points*)

1. (25 points) **Transforms.** In this part of the assignment you need to complete the implementation of the ATransform class. These features will support a character skeleton whose joints are arranged in a hierarchy where each child is positioned and oriented relative to its parent.

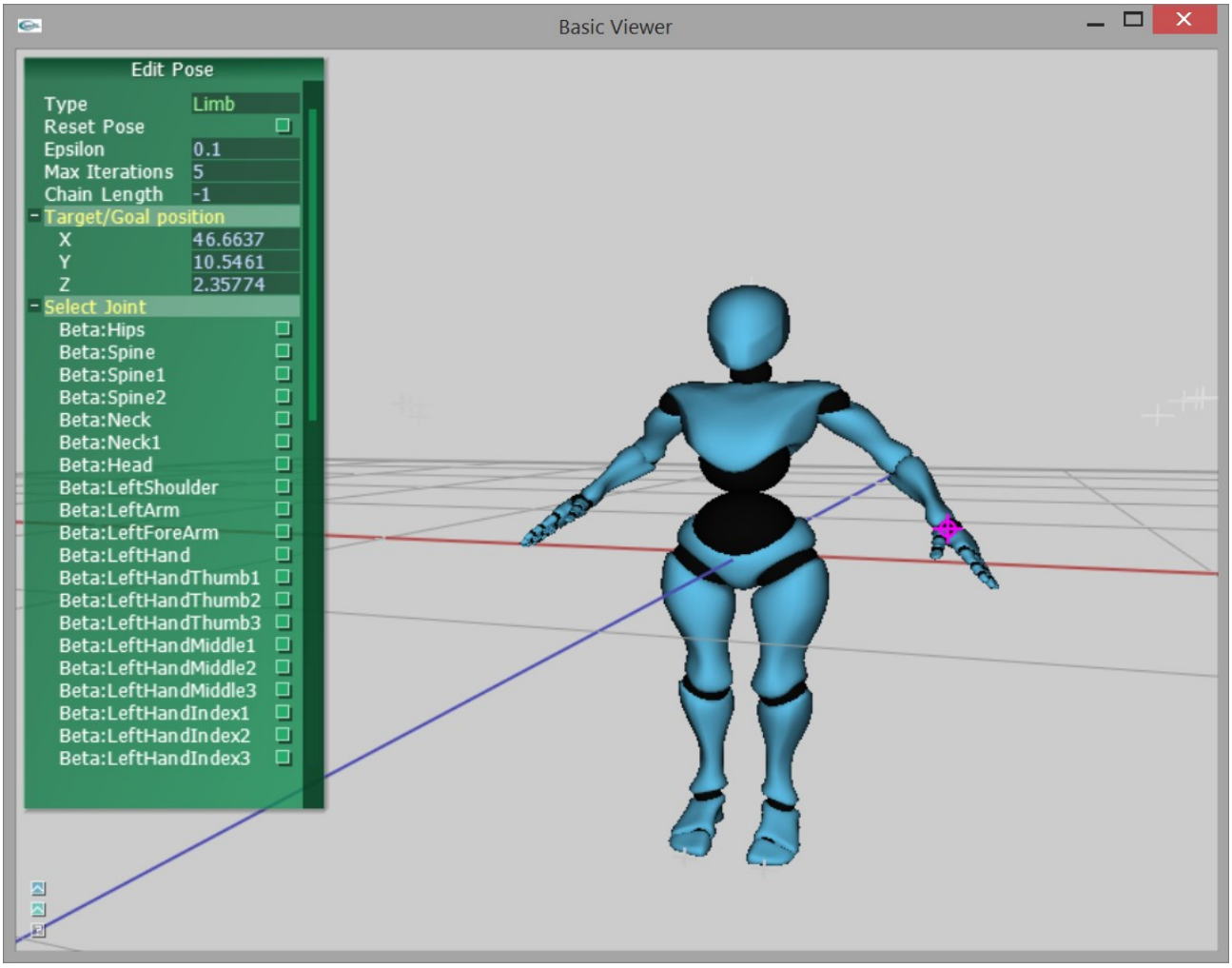
- (5 points) Implement ATransform::Inverse()
- (5 points) Implement ATransform::Translate()
- (5 points) Implement ATransform::Rotate ()
- (5 points) Implement ATransform::RotTrans ()
- (5 points) Implement operator* () for the ATransform class

2. (25 points) **Forward Kinematics** is the process of computing the position and orientation of each joint in a skeleton relative to the world given the local joint transformations. To do this, first the joint hierarchy (i.e. skeleton) of the character needs to be created and stored using the ASkeleton class, then each joint's transformation with respect to its parent (i.e. local2Parent) and the world (i.e. local2Global) needs to be computed using the associated Euler angle data stored in mMotion. A joint's global transformations (local2global) is computed using the local2global transform of its parent. In order to animate the character using the bvh motion data you need to compute the the forward kinematics by completing the implementation of the following functions:

- (5 points) Implement ASkeleton::update()
- (10 points) Implement AJoint::updateTransform()
- (10 points) Implement ABVHController::update()

IKViewer User Interface Overview

The AnimationToolkit basecode for the inverse kinematics (part 2c) of the assignment 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.

Inverse Kinematics Implementation Overview

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 Limb-based (i.e. geometric) 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 for the IK part of the 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

Part 2c - Inverse Kinematics To Dos (*100 points*)

1. (10 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. (45 points) **computeLimbIK.** In this part of the assignment you need to complete the `computeLimbIK` function which implements the Limb-based (i.e. geometric) IK method.

3. (45 points) **computeCCDIK.** In this part of the assignment you need to complete the `computeCCDIK` function which implements the CCD IK method.

4. ***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.