CIS 462/CIS562 Computer Animation (Fall 2020)
Homework Assignment 4
(Forward and Inverse Kinematics)

Due: **Part 0 - Monday, Oct. 12, 2020** (must be submitted before midnight)
Part 1 - Thursday, Oct. 15, 2020 (must be submitted before midnight)
Part 2 - Tues., Oct. 20, 2020 (must be submitted before midnight)
Part 3 - Thurs., Oct. 22, 2020 (must be submitted before midnight)

Please note the following:

- No late submissions or extensions for this homework.
- Since this is a programming assignment, you will need to download the code framework from GitHub.
- Use MS Visual Studio 2017 (available on CETS computers; or download the software from the Microsoft website: https://visualstudio.microsoft.com/vs/older-downloads/).
- Use Unity (the versions we have tested: 2019.2, 2020.1) to test your plugin.
- Use CMake 3.16 or above to configure and generate the Visual Studio solution.
- When you are finished with this assignment, update your GitHub project files before the assignment deadline and submit the **commit link** to Canvas.
- **Copy the modified files in your Curve Editor assignment to the directory /src/animation including:**
    - **aSplineVec3.cpp**
    - **aSplineQuat.cpp**
    - **aRotation.cpp**
- Set **CurvePlugin** and **FKIKPlugin** as the startup Project separately and build them to compile the dynamic libraries in the directory /UnityPlugin/Assets/Plugins, as described later in this handout. Then, set **FKViewer** as the startup project to check your OpenGL. We recommend you build the project early to make sure your Visual Studio and Unity work as expected. It is important to check compatibility, especially when you are using a different version of Visual Studio or Unity.
- You only need to implement the functions in the following files in the directory */src/animation* marked with "TODO" to complete this assignment.
    - **aTransform.cpp**
    - **aJoint.cpp**
    - **aSkeleton.cpp**
    - **aBVHController.cpp**
    - **aIKController.cpp**
    - **aActor.cpp**
- In this assignment, you may need to implement and debug some functions only used in Unity. Please check the tutorial to learn how to debug the Unity plugin.
- **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.**

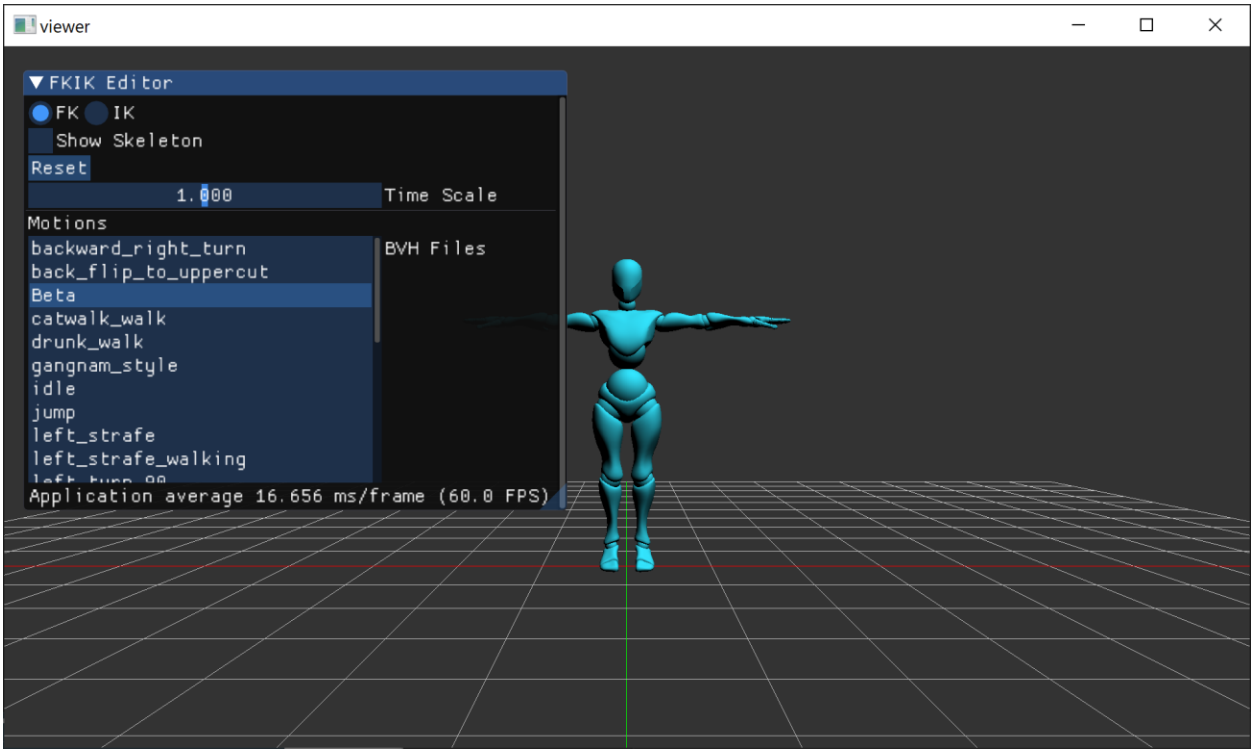
ANIMATION TOOLKIT FK/IK EDITOR

In this assignment you will develop an interactive FKIK editor and a Unity FKIK plugin by implementing various functions in the Animation Toolkit. The base code you receive for the Animation Toolkit will be used throughout the semester, with each subsequent programming assignment adding new features and functionality to the toolkit libraries. The Animation Toolkit base code includes the following directories:

- /3rdparty – some third-party libraries for OpenGL, GUI, and model processing
- /fbx – the FBX format models
- /motions – the BVH format motion files
- /obj – the OBJ format models
- /shader – the shaders used for rendering
- /src – header files and source files, the base code you will extend is here
- /UnityPlugin – the unity project where you can test your plugin

FKIK Editor Overview

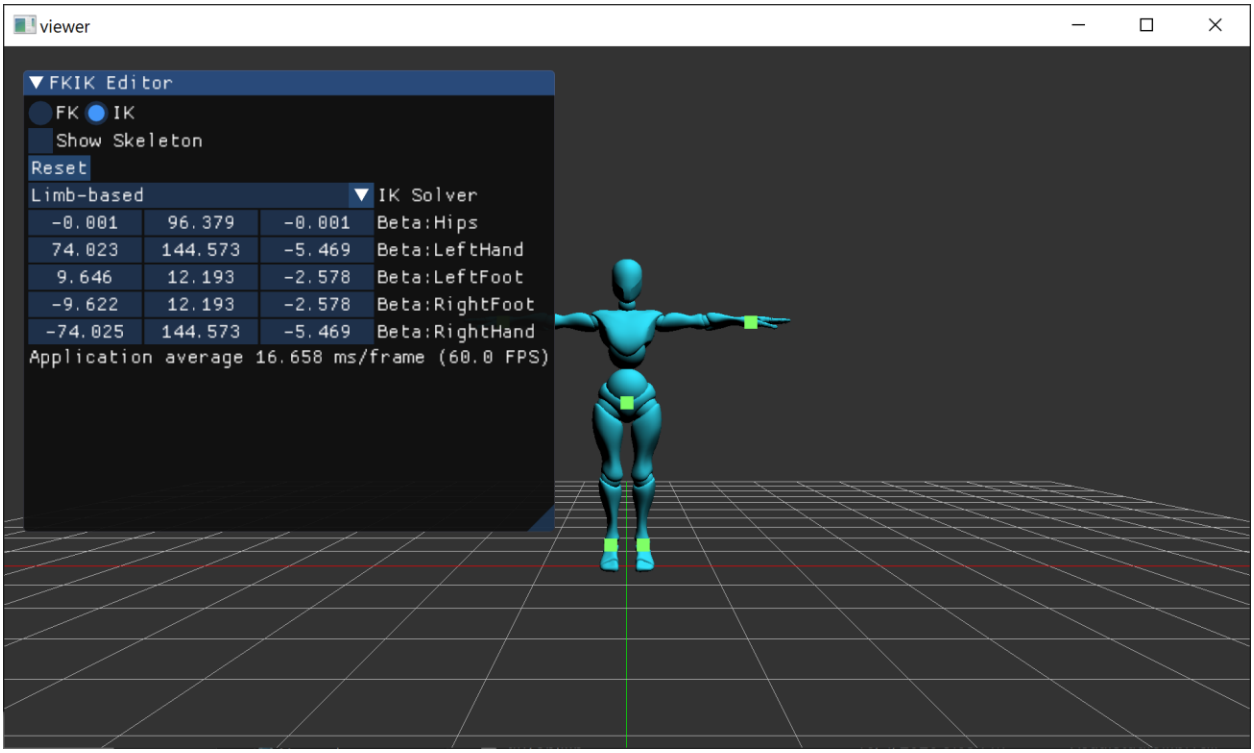
The FKIK Editor Viewer includes two demos which will allow you to test the FK and IK code being developed in this assignment.



Forward Kinematics

This viewer is used for loading and playing BVH files as shown in the screenshot above. By default, the viewer loads the beta character and several 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.

- Camera Control
  - Left CTRL + left-button drag with the mouse to rotate.
  - Left CTRL + middle-button drag with the mouse to zoom.
  - Left CTRL + right-button drag with the mouse to pan.
- The **Reset** button loads the default Beta.bvh motion.
- The **Time Scale** slider controls the speed of the animation.
- The **BVH Files** list shows all the BVH format files for the beta character for selection.



Inverse Kinematics

This viewer is used for selecting the IK solver and moving the targets (the green points) which represent the desired locations of the end joints and the root joint as shown in the screenshot above.

- The **IK Solver** drop-down menu specifies the IK method including Limb-based IK, CCD IK, and Pseudo Inverse IK.
- Move targets:
  - Use the input fields to edit the position of the targets for the root joint (Hips) and four end joints.
  - Or drag the green points to move targets.

Unity Plugin

The code you implemented can be compiled as a dynamic library *"__FKIKPlugin.dll"* and then be loaded in Unity as a native plugin. The plugin is at the following path:

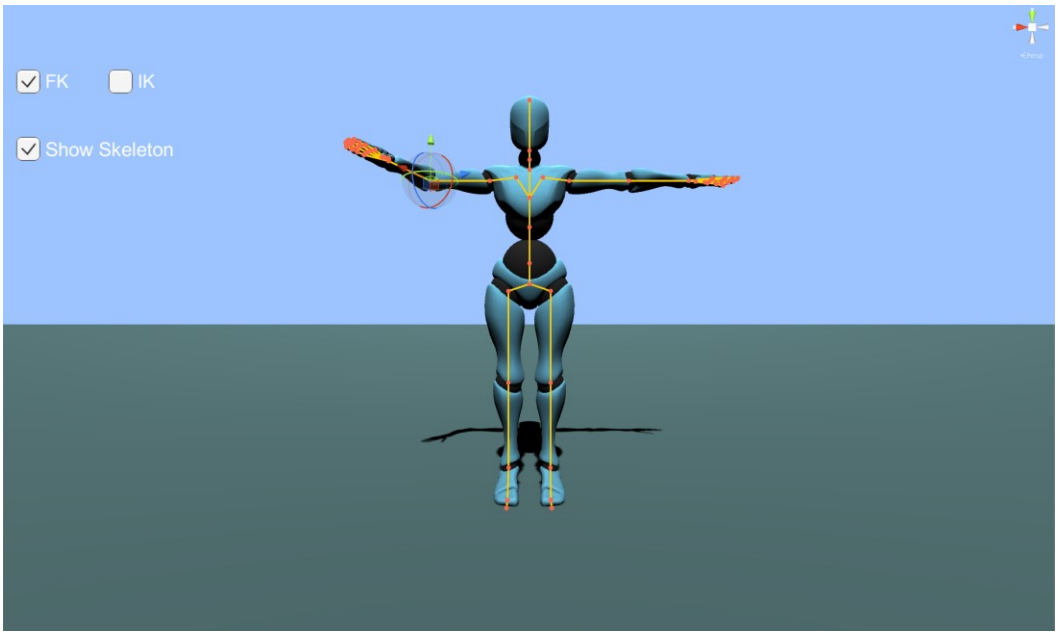
./UnityPlugin/Assets/Plugins

For this assignment, you need to **write extra code** to make it work. Open the Unity project in the directory */UnityPlugin* and open the Unity scenes */scenes/FKIK* and */scenes/BVH*.

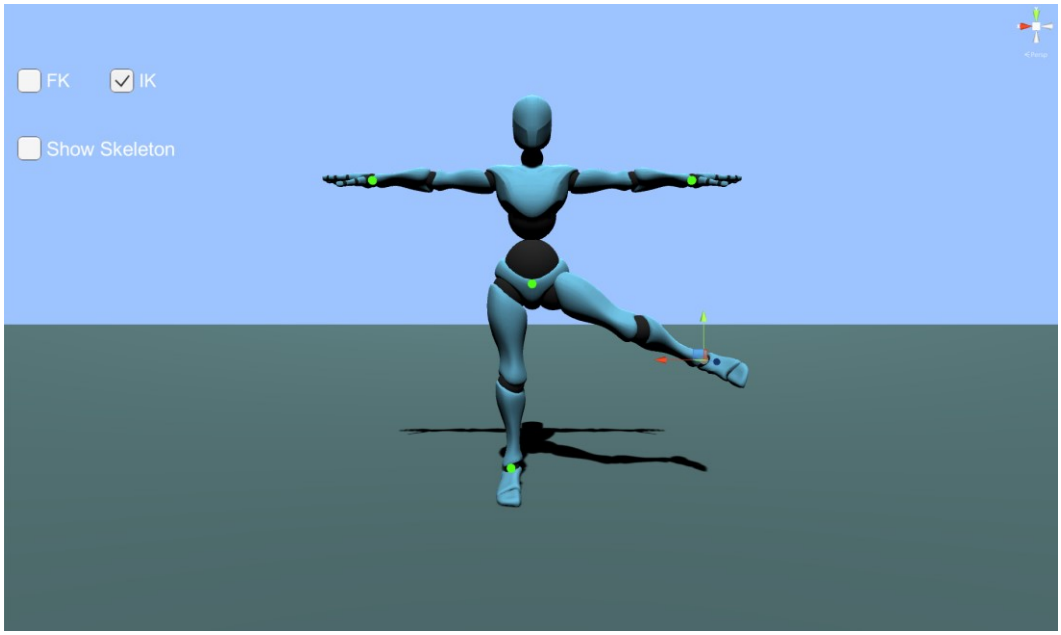
Camera Control:

- RMB + WASDQE - move camera forward, left, backwards, right, down and up respectively.
- RMB + MOUSE MOVE - rotate to look around.
- RMB + LALT + MOUSE MOVE - orbit around focus point. If the camera hasn't been focused, the default focus point is the world origin.
- MMB + MOUSE MOVE – pan.
- MOUSE SCROLL WHEEL - zoom in/zoom out.

For the FKIK scene, you can switch between the FK mode and the IK mode.

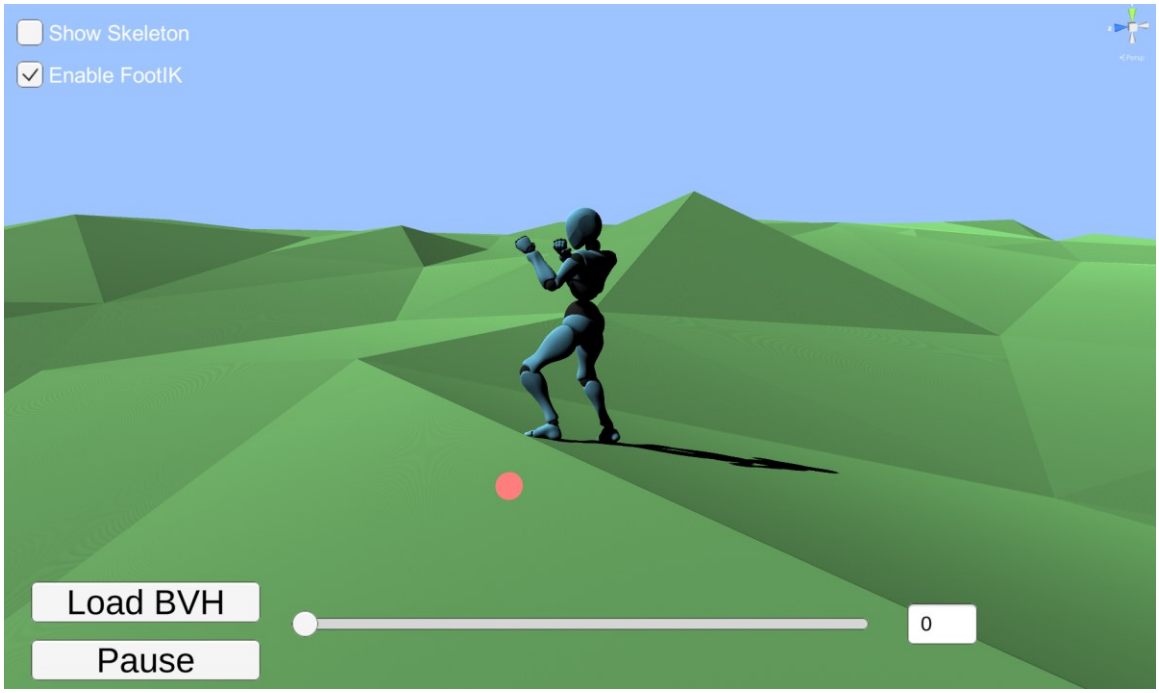


In the FK mode, you can select a joint (the red point) and use the rotation gizmo to rotate the joint.



In the IK mode, you can select a target (the green point) and use the translation gizmo to move the target as what you did in the OpenGL viewer.

For the BVH scene, you can load BVH format files to animate the beta character. The walking motion is loaded in default.



You can click the **Load BVH** button to select and load a BVH format file, click the **Pause** button to play or pause the animation, and use the **Slider** to choose a particular frame.

The red point in this scene is a target (not the targets used in IK) for the beta character, so the character will always walk towards this target. You can hold **LCTRL** and click the **Left Mouse Button** to set the target position.

When toggling **Enable FootIK** on, IK will be applied to the feet of the character so that it will always stands on the terrain.

Assignment Details

Part 0. Code Framework (10 points)

After downloading the assignment code framework from Github, it is recommended that you look through the FK/IK project codebase to better understand the object-oriented software design and the program execution model. In order to keep you honest, please answer the questions below:

1. (5 points): What is the data hierarchy (not the class hierarchy) of AJoint, ASkeleton, AActor, ATransform, BVHController, IKController? Please write it using the following format.

For example:

ClassA
    ClassB
        ClassC
    ClassD
        ClassE
        ClassF

In the example above, Class A stores data with the type of Class B and ClassD; ClassB stores data with type ClassC, etc.

2. (5 points) How many quaternion splines and vec3 splines are used for the Beta character animation? Which function is responsible for initializing these splines? Hint: Set break points in the debugger and step through the code to find the answer.

Please submit your Part 0 answers in the form of an MSWord or pdf doc to the HW4 Assignment folder in the CIS462/562 Canvas site.

Part 1. Forward Kinematics (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 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()

Part 2. Inverse Kinematics (*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.

Part 3. Unity Plugin (*50 points*)

1. (25 points) **Guide Joint**. After each walking cycle, the character will go back to the origin. To solve this problem, we add a guide joint in the AActor class. It is a virtual joint which you can consider as the parent joint of the root joint (the Hips). In other words, the transform of the character will be in the guide joint's space instead of the world space. Before each walking cycle, you need to update the position of the guide joint to where the character is. The process is:

1. *Set the global position of the guide joint to the global position of the root joint*
2. *Walking Cycle 1*
3. *Set the global position of the guide joint to the global position of the root joint*
4. *Walking Cycle 2*
5. *……*

Implement `AActor::updateGuideJoint`. You need to update the orientation of the guide joint as well so that the character can face towards the target position at the beginning of each walking cycle.

2. (25 points) **Foot IK**. In each frame, we will use ray cast in Unity to get the height (y value) and the normal of the terrain at the position of the left foot and the right foot of the beta character. Implement `AActor::solveFootIK` to update the character with Limb-based IK and update the

orientation of the feet based on the normal so that the character can always stand on the terrain. Note that the normal and the height given are in the world space. Besides, you need to first update the transform of the root\hips joint before applying IK.