COMS W4160— Computer Graphics Spring 2017

Programming Assignment 5

out: April 22, Saturday
due: May. 12, Friday, 10:00AM. Note: the due time is in the morning!

This assignment is about forward and inverse kinematics for computer animation. It is an assignment that we have decided to merge with the final project. Thus, it has a larger scope than the previous programming assignments. You are expected to code more, and of course this assignment will be weighted more in terms of your final grade.

1 Skeleton Structure

A skeleton has a tree structure, as illustrated in Figure 1. The skeleton consists of links connected by revolute joints. There is always a root link. One end of the root link might be fixed by a revolute joint (refer to Joint2D.java). Each link may connect with one or more child links. Note that a joint always connects one parent link with one child link (1:1). If a link is connected with multiple child links, then there are multiple joints, one connection for each child link (refer to LinkConnection2D.java). If one end of a link is free, that end is referred as an *end effector* (see red circles in Figure 1).

In an inverse kinematic system, the user should be able to select an end effector and drag it around, and your program needs to keep updating the angles of revolute joints to move the end effector to the desired position.

2 Starter Code

Our starter code is built on the code of programming assignment 1. The files that you need to finish with your code are under the c2g2/kinematics folder.

Starter code Structure. We provide a starter code framework for 2D skeletons. The skeleton is maintained in the class of Skeleton2D. A skeleton consists of a set of RigidLink2D instances organized in a tree structure. Each link might be connected with multiple child links. Each pair of parent/child links are connected through a revolute joint (RevoluteJoint2D), and such a connection is maintained in the class of LinkConnection2D.

Skeleton specification. We provide a simple XML parser to specify a skeleton structure (see Skeleton2D.java). An example of the skeleton XML file is shown here:

```
<?xml version="1.0"?>
<skeleton>
  <root x1="0.0" y1="0.4" x2="0.0" y2="0.2">
      <joint1 x="-0.1" y="-0.1">
        <joint2 x="0.7" y="-0.6">
          <joint3 x="0.7" y="-0.9">
          </joint3>
        </joint2>
```

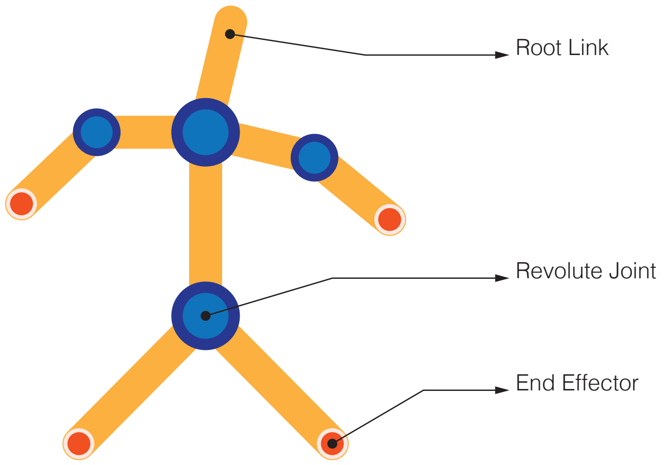

Figure 1: **Skeleton structure.**

```
        </joint1>
        <joint1 x="0.1" y="-0.1">
          <joint2 x="-0.3" y="-0.6">
          </joint2>
        </joint1>
        <joint1 x="-0.3" y="-0.1">
        </joint1>
    </root>
</skeleton>
```

Here the root link is specified by its two end positions. The second end position is connected with child links. All the child links are specified by the positions of joints. The other end of each child link is connected to its parent link. You can visualize an example skeleton by running the starter code directly (see the class c2g2.kinematics.engine.Main).

3 Programming Requirements

Your implementation consists of the following components: (i) forward kinematics on a 2D skeleton, (ii) inverse kinematics on a 2D skeleton, (iii) forward kinematics in 3D, and (iv) setting keyframes using inverse kinematics and creating animations using Cubic Bézier interpolation.

3.1 Forward Kinematics in 2D

You have two sub-tasks in this step. (i) You need to create a user interface (e.g., by pressing certain keys or adjusting slide bars) to set the rotational angles of all joints. You can bind any keys or design your user interface for updating the joint angles. Please refer to the starter code of previous assignments (e.g., PA-4) for the codes of binding specific keys. (ii) Implement the forward kinematics algorithm (in c2g2/kinematics/ForwardKinematics.java) that we discussed in class to update the positions of the joints. In general, you need to travel through the tree structure of the skeleton, and concatenate transformation matrices brought on by each revolute joint. You need to repeat the tree traversal and update the position of each joint.

Every time you update the parameters of the joints, make sure that the OpenGL visualization re-renders the current skeleton structure. Please refer to the starter code for the OpenGL visualization, and feel free to improve it.

3.2 Inverse Kinematics in 2D

You have two sub-tasks in this step. (i) You need to follow the starter code and allow the user to choose an end effector of the skeleton through mouse clicking on the end effector. Then, the user should be able to drag the mouse and thereby specify the desired end effector position in a time continuous fashion. To this end, you need to travel through the skeleton structure, find all the end effectors, and identify the end effector over which the mouse is currently dragging. (ii) Implement the inverse kinematics algorithm (in c2g2/kinematics/InverseKinematics.java) that we discussed in class to find joint angles that realize the desired end effector position. Then, you need to update the skeleton structure with the resulting joint angles, and render it in OpenGL visualization. You should perform the inverse kinematic solve when the user is dragging the end effector, so the structure is updated continuously.

Solving a linear system. In the inverse kinematic algorithm, you need to solve linear systems (recall the derivation in class and slides of lecture-20 on CourseWorks) during Newton's iterations. There exist many Java libraries for solving a linear system. For example, we recommend the Efficient Java Matrix Library (EJML). You are allowed to choose other libraries, but make sure that all the external libraries are included in your submission. It is your responsibility to make sure that your code can be compiled successfully when the TAs grade it.

3.3 Forward Kinematics in 3D

Next, you need to implement a forward kinematics algorithm for 3D. We don't provide starter code for a 3D skeleton. You will need to write your own code by augmenting the 2D code. In 3D, you can use revolute joints to connect 3D links. Unlike the joints in 2D skeletons, those 3D revolute joints have revolution axis which might point along an arbitrary direction. You need to take into account the revolution axis of each joint when constructing the transformation matrices, as they might point in different directions.

Similar to the 2D forward kinematics, you need to implement a user interface to set the rotational angle of each joint. You also need to implement a simple 3D visualization to display the 3D skeleton on the screen.

3.4 Creating Character Animation by Interpolating Keyframes

Your last task is to create a skeleton animation by interpolating keyframes. You can set up key frames via inverse kinematics. For example, when setting up a key frame, the user can drag the end effectors to pose the skeleton. You need to implement a user interface to select a time frame and use inverse kinematics to specify the pose at that time frame. Then, you need to use the Cubic Bézier curve interpolation to construct an animation. You can implement this simple character animation system in 2D (using 2D forward and inverse kinematics), although we will give (10%) bonus points if you implement this in 3D.

Each animation should include **at least five** key frames. The total time length should be at least 4 seconds (assuming the animation has 30 frames per second). In your report, for each animation you submit (see below), please include a screen capture of each of the key frame poses and clearly label them.

4 Submission

Submission Checklist: Submit your assignment as a zip file via CourseWorks. Your submission must consist of the following parts:

1. **Documented code:** Your code should be reasonably documented and be readable/understandable by the TAs. If the TAs are not able to run and use your program, they will be unable to grade it. To ensure the TAs can grade your assignment, please make sure your code can be compiled successfully. It is your responsibility to make sure the code can be correctly compiled. Otherwise, you'll lose some points.

2. **Skeleton file:** You need to create your own skeletons. Please include the XML files of your skeletons in your submission.

3. **Video demos:** You need to include **five** videos, each has a length less than 15 seconds: **(1)** a video of screen capture demonstrating the 2D forward kinematics; **(2)** a video of screen capture demonstrating the 2D inverse kinematics; **(3)** a video demonstrating 3D forward kinematics; **(4)** a video demonstrating the interpolated skeleton animation. In this demo, you should use only a chain of links (with at least 5 links, no branches) to demonstrate the inverse kinematics (to set up key frames) and the interpolated animation; and **(5)** another video demonstrating the interpolated skeleton animation. In this demo, you should use more complex skeletons with branches (i.e., a character-like skeleton) to demonstrate the inverse kinematics (to set up key frames) and the interpolated animation.

4. **Brief report:** Include a description of what youve attempt, special features you have implemented, and any instructions on how to run/use your program. **Please also include the skeleton key frames used in your video (4) and (5)**. In compliance with Columbia's Code of Academic Integrity, please include references to any external sources or discussions that were used to achieve your results.