Project 2:
Inverse Kinematics for Motion Capture

Physics Based Animation
CSE 558, Spring 2000

Zoran Popović

Out: April 20, 2000
Due: Tuesday, May 9, 2000

1 Problem definition

In this assignment you will implement a differential inverse kinematics solver for the specific purpose of computing joint angles from intermittent motion capture data. Your challenge is to be able to compute the joint angles within the satisfactory accuracy, as fast as possible. At the same time, the resulting animation should not have any 'popping' artifact due to the disappearance of the markers at various times.

On the input you will have a motion capture data file. The result of your computation should be time-varying joint angles for you character. When displaying the final animation, in addition to displaying the character, you should also show the markers and their violations. The output should also measure the aggregate distance between the marker trajectories and their corresponding body points throughout the entire animation. You should be able to measure the speed of your IK code.

2 Implementation

The problem implementation is intentionally open-ended. You can choose to setup the problem either as a least-squares fit, or as a standard Lagrangian formulation, or as anything else you might think of. The main goal is to do it fast, without sacrificing accuracy, and to gracefully deal with incomplete data.

No matter which method you choose, you will need to compute the partial derivatives of any point on the body efficiently, which involves differentiating through transformation paths of the hierarchy. You will also need an efficient sparse linear equation solver. I suggest LSQR or conjugate gradient.

3 Motion capture data file format

As an input to your code you should use the *hop3.cme* file. The files in the cme format have the following first line:

```
frames = 960 markers = 26 Hz = 120
```

where frames is the number of captured time samples, markers is the total number of markers on the body, and Hz is the frequency of the motion capture. Therefore, the above motion capture data contains $960/120$ seconds of trajectories for 26 markers. The rest of the file contains $frames \times markers$ lines each containing 3 coordinates for a specific marker. First, there are 26 lines containing the coordinates for 26 markers at frame 1, followed by 26 lines with coordinates of 26 markers at frame 2, etc. If a position of a specific marker is unknown at a given frame it's coordinates will be given as

```
0.0 0.0 0.0
```

The motion capture file you are given is not clean, so there are many instances of unknown marker positions.

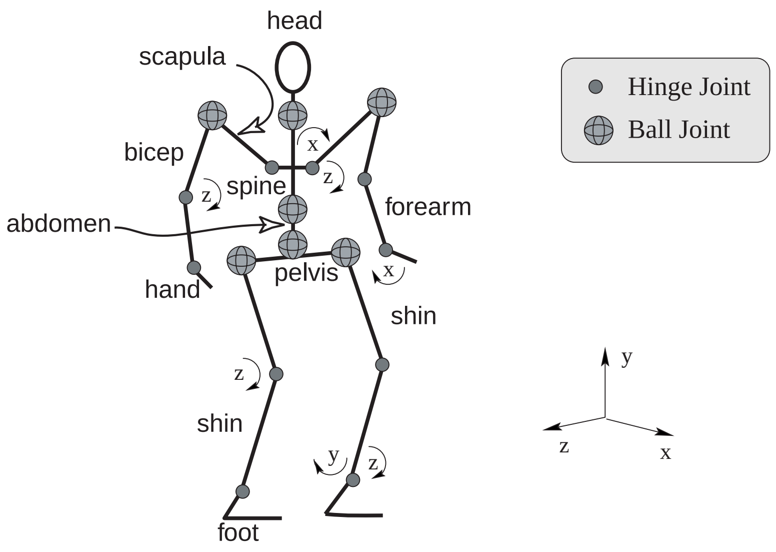

Figure 1: DOFs for the human character representation.

4 Character dimensions and marker placement

Ideally, you should be able to automatically infer the character's dimensions, kinematic properties, and marker placements. Since this is a hard problem you're allowed to look at human.cc file for help on the dimensions, marker placements, relative weights of different body parts, etc. This source file builds a transformation hierarchy shown in Figure1. I strongly suggest that you use the same hierarchy, and the same number of DOFs for the purposes of fair evaluation of the performance. You should implement ball joints either as 3 Euler angles or as quaternions.

5 Solver Interface

You best bet is to never actually build any matrices. Keep only the structures which will enable you to efficiently compute $\mathbf{A}\mathbf{v}$ and $\mathbf{v}^T\mathbf{A}$ for a given vector $\mathbf{v}$. Your sparse linear solver should need only these two functions to solve $\mathbf{A}\mathbf{x} = \mathbf{b}$. The termination criterion for you iterative solver should measure the amount of progress made at each step, and stop when little or no progress can be made. If you simply bound the number of solver iterations, you will have no guarantees of convergence, and most likely your accuracy will suffer.

6 Evaluation

You should be able to measure the frequency of your updates for each time frame of data. Be sure that you can turn the display off, so that you get this measurement independent of the cost to display the animation.

You accuracy measurement should be a sum of the marker violations for each time frame. At each frame, the violation is measured as a sum of distances between marker positions and the corresponding body points.

Every time a marker appears or disappears there is a perceptible discontinuity in the joint angle functions. This discontinuity produces an unsightly popping behavior. You should figure out a way to alleviate this problem. A successful solution would not contain any perceptible "pops" in the final animation.

7 Experiments

As always, there are numerous optional ways to extend the standard framework of this problem:

- Compute the mass matrix, or just the diagonal elements of the mass matrix in order to account for the different scales of various DOFs.

- Given the kinematics of the character, compute the best possible marker body positions. Don't worry about the speed here, it's considered a pre-processing step.
- Compute the kinematic dimensions and marker location automatically from the motion capture data, and the roughly estimated locations of the marker body positions.
- Could you improve the quality of your solution if you allowed yourself a lookahead of n frames? That is, you would compute the IK solution for frame f_i knowing the result for all previous frames as well as the frames $f_{i+1}, \cdots, f_{i+n}$? Implement a realtime IK implementation that would use this intrinsic lag to better handle the popping behavior.
- What if you had the full knowledge of all frames in the animation? Implement an IK solution that would solve this off-line version of the problem.
- Use the mass information for each body part to produce a more natural solution for character movement in light of missing marker data.