Video Game Programming (15-493)
Spring 2005
Assignment 2: Motion Capture

Due: Monday, February 14, 2005

1 Overview

In this assignment, you will use motion capture data to automatically generate motion for a virtual human character. The idea is to generate a computer animation by connecting different pieces of motion, as in the motion graph approach discussed in class. We will provide you with a number of segments of mocap data as your raw material, and you will build a graph data structure to represent the connections between different pieces of these segments. Each frame in the motion can be treated as a node, and an edge is created between two nodes if the character's poses in the two frames are close enough. By randomly traversing the graph, you can generate a continuous animation whose content is randomly chosen from the original motion clips. Your assignment is to implement a motion graph. Once you have the graph, you need to:

1. generate long sequences of motions for the character (in theory this can be indefinite)
2. have the user sketch a path that the character then follows as best as possible

You will first need to define *good* transitions by defining an evaluation function to determine how similar the two poses that you want to transition between are. You will also need to design a fading/blending algorithm to make these transitions look smooth. To represent these transitions you will need to build a graph data structure.

You may find these two papers helpful. However, you do not need to follow the details of the approaches in these papers. They may help you in understanding the main ideas for building a motion graph. See the next section for details of what you should do.

- www.cs.wisc.edu/graphics/Papers/Gleicher/Mocap/mograph.pdf
- http://graphics.cs.cmu.edu/projects/Avatar/avatar.pdf

2 Requirements and Implementation

2.1 Evaluation Function

Define an evaluation function to determine how similar two poses are, i.e., a distance to measure the difference between two poses. Because a character's pose is described by all of his/her joint angles and the position and orientation of his/her root node, a simple way to compare two poses is to calculate the sum of the distances between each joint angle and the position and orientation of the root node. Of course, this would not be a good pose metric because the weighting of each joint is the same whereas some joints are probably more important than others in choosing a good transition in the motion. You should experiment with different weightings to get a good metric.

You should look for good transitions within each individual segment as well as good transitions between different motion segments. Also, to generate a large number of transitions you should allow transitions

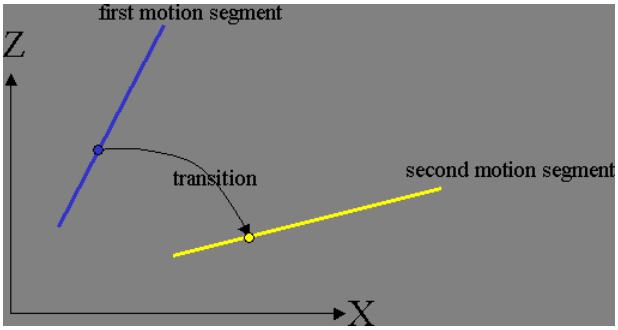

Figure 1: Blue and yellow lines represent the trajectory of the root node projected into XZ plane. Because we give a zero weight to the X and Z components of the root position and to the yaw orientation of the root, the transition shown in the figure will be declared *good* if all other joint angles are similar.

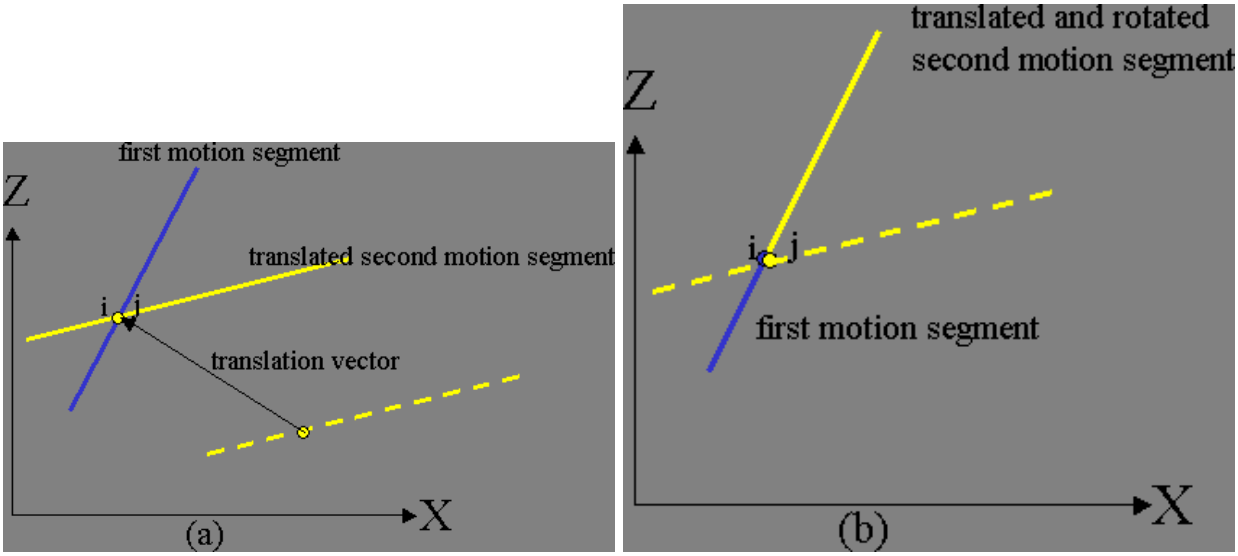

Figure 2: *Left*(a) translate second motion segment to preserve C_0 continuity of the root position. *Right*(b) rotate second motion to preserve C_0 continuity of the root orientation

between frames that have very different character orientation (looking direction) and root positions as long as they have similar joint angles (see Fig. 1). To preserve C_0 continuity of the root position in the generated motion, you will need to translate the second segment (the segment you are about to transition to) by the difference between the two transition frames in XZ plane (see Fig. 2). To preserve C_0 continuity in the character orientation (looking direction) you will also need to rotate the second motion (see Fig. 2).

You can also include velocity or other information related to the continuity of a motion into the metric. You can also take into account the frames before and after the transition when defining the metric. Experiment with several different options and find a metric that gives you a good measurement of the perceptible difference of the motion in the two frames.

If you are using euler angles for rotation representation, you may want to exclude root orientation from your evaluation function completely as the same orientation is often represented by very different rotations around XYZ axes (this is especially noticeable for root node orientation because it varies greatly). Here is a pointer to code doing Euler angle conversion: vered.rose.utoronto.ca/people/david_dir/GEMS/GEMS.html. In order to rotate the root orientation, you may switch the Euler angle representation from X-Y-Z (this is what it's currently is) to Y-X-Z. This puts the yaw value first. You can then use this value when rotating the pieces of motion, but you should not include this value in the evaluation function.

Current Frame	Transition From		Transition To	
	Motion #	Frame #	Motion #	Frame #
171	0	171	0	367
193	0	388	0	1007
204	0	1017	0	1205
233	0	1233	0	315
247	0	328	0	1062
264	0	1078	0	153
278	0	166	0	362
280	0	363	0	789
315	0	823	0	593
322	0	599	0	950
343	0	970	2	100
344	2	100	5	5
352	5	12	3	23
443	3	113	2	126
444	2	126	5	67
460	5	82	0	785

Figure 3: Shows transitions taken for one example. Current Frame - frame number (of resulting amc) when transition happened. Transition From - a number representing an amc data file and the frame where the transition takes place. Transition To - a number representing am amc data file and the frame where the transition jumps to

2.2 Directed Graph

Construct a directed graph to represent the transitions. Using the evaluation function, you can calculate the distance between pair of nodes (poses) and determine if an edge should be created between these two nodes. Because each mocap file contains many frames, it would be quite time consuming to calculate the distance between for each pair of frames. You may want to downsample the data (only allow transitions only every few frames). You may also ignore links that would cause the motion to jump just a few frames ahead or behind of a current frame. In addition, you may want to prune a transition if the probability of transitioning is below some threshold. Finally, you should think about other approaches that would lead to a better transition graph.

2.3 Fading/Blending Algorithm

Design a fading/blending algorithm to make the transitions look smooth. Although the distance function helps you find similar poses, the difference between two poses will usually be noticeable and the motion will be discontinuous at the transition point. You can reduce this discontinuity by blending the ending frames of the first motion with the starting frames of the second motion.

2.4 Generate Long Sequences of Motion

Once you have the graph, you can start at a random node and traverse the graph randomly. You can then generate long sequences of motions (you are not required to generate really long ones here). Specifically, you should generate three examples to demonstrate your results. Each one must be at least 500 frames in length, and should contain at least 15 transitions. For each result, provide a transition table that shows all the transitions in that example (see Fig 3) and submit the .amc file.

- Results 1 and 2. Use database with forward motion (see below). For the first example, allow only a small number of transitions (set your parameters so that there are a small number of transitions). For the second example, allow more transitions.
- Result 3. Use the second database with forward walking motions, and different kinds of turning motions.

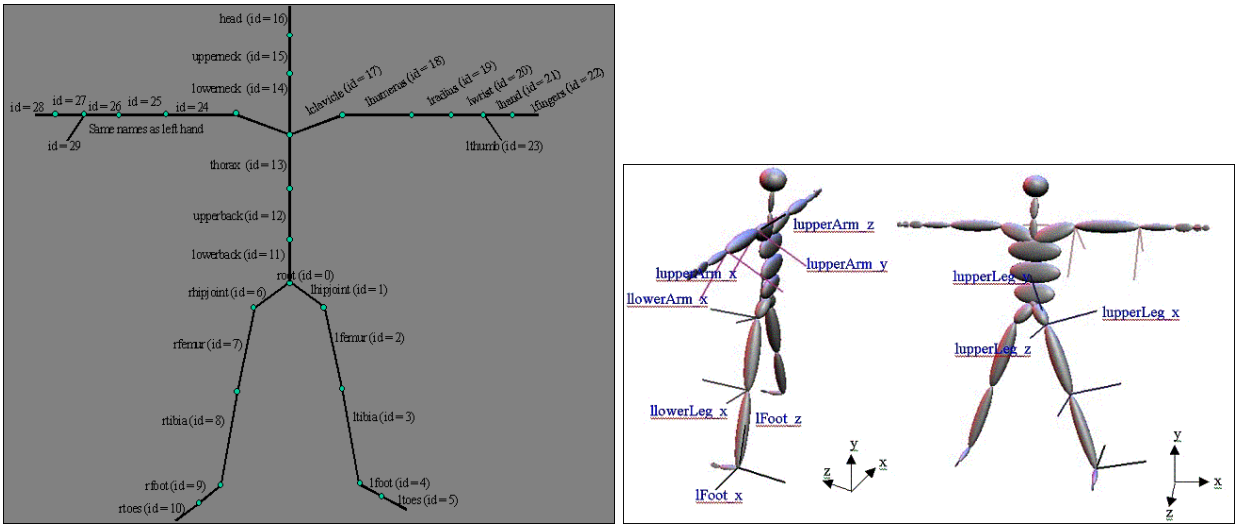

Figure 4: *Left* (a) Shows the skeleton hierarchy, in particular name and id for each bone as defined in .ASF file. *Right* (b) Shows local coordinate system for some of the bones.

2.5 Have the character follow user-sketched path

We will soon be providing an interface in the amcviewer that allows you to sketch a path on the floor using the mouse.

Your assignment here is to find some motion in the graph that allows the character to follow the path as best as possible. See the two papers for examples. No obstacles avoidance is needed. The key here is to find some heuristic that allows the character to follow the path.

3 Databases of Motion

3.1 Background on ASF/AMC files

Our human model is represented using a hierarchy (Figure 4). The optical mocap system generates two data files: ASF file, which encodes the skeleton kinematics (length of various links, degrees of freedom, etc.) and AMC files, which stores the motion (root translation, euler joint angles) over time. The world coordinate system in these data files is Y up. Figure 4 (a) shows the skeleton hierarchy as defined in ASF file (length of the bones is not to scale). Figure 4 (b) shows rotation axes and names of some bones.

You do not actually have to deal with the ASF file. We have a skinned character of one of our skeleton models which correspond to the given *16.asf* file. The model is the *woman42* model in the amcviewer code. You will have to deal with the AMC files.

3.2 Databases

We provide you with two databases of motion capture files. The first database contains motion data of human walking forward (6 files total). The second database contains motion data of human walking in different styles (forward, and different kinds of turns). We recommend you first test your code on the forward database. It should be easier to get your code to work on this database because you can probably get away with using a simpler metric. The data has a rate of 120 frames per second. You should downsample to 30, because this should not affect the quality of the motion and there is less data to deal with. Remember then to change the frame rate in the amcviewer code.

4 Starter Code

mocap.py - reads in skinned character representing the skeleton defined in 16.asf. Functions for reading in amc data, and displaying each frame on the skeleton. Used in amcviewer.py. The *applyMocapFrame* subroutine might be useful for you to display individual frames. You can import this code elsewhere and use it.

amcviewer.py - run this one as *ppython amcviewer.py filename.amc*. This is a viewer of the amc data. Loads the character and the floor. Might want to change camera settings and other parameters here. Sets up keyboard controls. *Left arrow* displays each frame in reverse. *Right arrow* displays each frame forward. *Space bar* plays and stops the motion.

amcReader.py - has a class to read amc file into a data structure (a list of lists). See comments in code for more description. Has methods for setting the values of the root positions and orientations, and also retrieving these values. The root positions and orientations are important for translating and rotating the pieces of motion in the motion graph. *testcode1.py* contains an example of how to use amcReader.

amcWriter.py - has a class to write a list of lists into an amc file. *testcode2.py* contains an example of how to use amcWriter.

vector.py and *matrix.py* - classes with vector and matrix operations that might be useful. *testcode3.py* and *testcode4.py* contain examples. Python also has its own way to deal with vectors and matrices.

You are welcome to add code to these classes. You will need to create your own files to implement the motion graph.

5 Grading

- If your code works on the forward database: 50%
- If your code works on the database with forward walking and turns: 15%
- If your character can follow a user-sketched path: 20%
- Written Report: 15%

6 Hand In

- Your source code.
- Three results of long sequences of motion generated by your code. Submit 3 amc files, and corresponding transition tables (see above).
- For the user-sketched path part, be specific about how we can reproduce the results that you have.
- Written Report. Turn in a writeup of what you did. Tell us what algorithms you tried for each of the points above. Tell us what worked, what didn't work, and what you would like to try if you have more time.