CS6665 - Character Animation
Project 2 - Motion Graph

Duong Hoang and Nghia Truong

November 14, 2014

1 Introduction

In this project, we implement a basic motion graph, and test it on two different datasets. Given a set of motion data in BVH file format, our program constructs a directed graph where each node is a motion frame and each edge connects either adjacent frames in the original motions, or frames in which the character has similar poses. We use the similarity metric and interpolation scheme described in [KGP02]. This similarity metric does not use joint angles, thus we can avoid assigning weights to the joints. More importantly, it implicitly takes into account differences in not only positions but also velocities and accelerations. We interpolate using a window of frames to ensure smooth transitions. By walking randomly in the graph, we can generate random, arbitrarily long motions, comprising of different short motion sequences in the database. Our character never gets stuck because our graph is pruned to get rid of dead ends. Due to time constraint, we are not able summarize the graph by clustering, address the foot skating problem due to interpolation, nor implement any graph search algorithms to constraint the synthesized motions.

2 Motion Graph

Motion graph [AF02, KGP02, LCR$^+$02] is the family of techniques that organize short motion clips into a directed graph, and synthesize new motions by following paths in this graph. By stitching together similar frames across different motions, longer, more interesting motions can be created. It is also possible to generate motions that satisfy additional constraints by searching for these constraints in the graph. Thus motion graphs can generalize an existing motion capture database, making it appealing when compared to the costly alternative that is to simply capture a wide range of motions.

A good motion graph needs to be well-connected. Low connectivity makes it difficult to switch between motions, and the character can get stuck in some local component of the graph. But we cannot simply increase connectivity by connecting dissimilar poses, since that would produce bad, unnatural motions. A reasonable similarity metric and threshold for poses, therefore, are crucial to ensure the quality of the graph. In our system, the threshold parameter is tuned by hand for each set of input motions.

3 Implementation

The input to our system is a set of BVH files. We create an initial graph whose vertices are the original frames, and edges connect adjacent frames. For each pair of vertices, we compute a distance specifying how similar the corresponding poses are. An edge is then created between each pair of vertices having a distance smaller than a predifined threshold. The next step is to prune the graph of all dead-ends, so that the character will not be stuck. It is common to build a high-level summary of the graph by, for example, grouping the vertices into clusters to facilitate graph search. In our case, this step is not necessary since we only implement a random graph walk.

4 Frame Distance Computation

Consider the ith frame from motion A and the jth frame from motion B. To compute the distance between A_i and B_j, we form two windows consisting of k consecutive frames each. The first window starts at frame A_i ($A_i, \ldots, A_{i+k-1}$) and the second window ends at frame B_j ($B_{j-k+1}, \ldots, B_j$). k is set to 30 frames in our implementation, which corresponds to 1/3 second if the input motions are captured at 120Hz. We collect the positions of every joint on the character across k frames to form two point clouds in A and B respectively (the number of points in each cloud is kn, with n being the number of joints). The B point cloud is transformed rigidly using least square to match the A point cloud, and we sum over the squared Euclidian distances between all points in the A cloud and the corresponding points in the transformed B cloud to get the distance between frame A_i and frame B_j. Formally, this distance is

$$d(A_i, B_j) = \min_{\theta, x_0, z_0} \frac{1}{nk} \sum_i \left\| \mathbf{p}_i - \mathbf{T}_{\theta, x_0, z_0} \mathbf{p}'_i \right\|^2 \qquad (1)$$

where $\mathbf{T}_{\theta, x_0, z_0}$ is a rigid transformation that rotates a point $\mathbf{p}$ about the y (vertical) axis by θ and then translates it by (x_0, z_0). If we use x, z to denote the x and z coordinates of points in the A cloud and x', z' to denote the coordinates of points in the B cloud, closed-form solutions for θ, x_0, z_0 exist, as follow (recall that k is the window size and n is the number of joints):

$$\theta = \arctan \frac{\sum_i (x_i z'_i - x'_i z_i) - \frac{1}{nk}(\bar{x}\bar{z'} - \bar{x'}\bar{z})}{\sum_i (x_i x'_i + z_i z'_i) - \frac{1}{nk}(\bar{x}\bar{x'} + \bar{z}\bar{z'})} \qquad (2)$$

$$x_0 = \frac{1}{nk}(\bar{x} - \bar{x'} \cos\theta - \bar{z'} \sin\theta) \qquad (3)$$

$$z_0 = \frac{1}{nk}(\bar{z} + \bar{x'} \sin\theta - \bar{z'} \cos\theta) \qquad (4)$$

where $\bar{x} = \frac{1}{nk}\sum_i x_i$, similarly for $\bar{x'}, \bar{z}, \bar{z'}$.

Using joint positions instead of angles, we do not need to come up with a set of joint weights, which is difficult to get right. Joint velocities and accelerations are also automatically taken into account by comparing k-frame windows and not individual frames.

An edge from A_i to B_j is created whenever $d(A_i, B_j) < d_{threshold}$, where $d_{threshold}$ is some predefined constant. We actually use different thresholds for inter and intra-motion frames. Computing distances between all pair of frames can be prohibitively slow, so extra care is needed to optimize the distance computation code. We implement this step using multi-threading, compute and save the results (all the distances) to disk once, then only load up this data when we need to tune $d_{threshold}$.

Tuning for a balanced $d_{threshold}$ is important. In our implementation, it is chosen so that the number of new edges is at least half the number of original edges, to make sure the character can easily switch between different motions.

5 Graph Pruning

After new edges are added, the graph needs to be pruned to make sure there are no dead ends. We loop through all vertices in the graph, and if a vertex has no outgoing edge, we remove it (and all incoming edges to it) from the graph. This process is repeated until there are no more vertices to remove, at which point the graph is free of dead ends. It may, however, contain sub-graphs, or regions that are not well connected to the rest of the graph, implying the character may get stuck inside one such region for a long time. We hope to address this issue in the future.

6 Frame Interpolation

6.1 Alignment

6.2 Interpolation

7 Random Walk

8 Results

References

[AF02] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21(3):483–490, July 2002.

[BK04] Samuel R. Buss and Jin-Su Kim. Selectively damped least squares for inverse kinematics. *Journal of Graphics Tools*, 10:37–49, 2004.

[Bus04] Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. Technical report, IEEE Journal of Robotics and Automation, 2004.

[Gle98] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 33–42, New York, NY, USA, 1998. ACM.

[KGP02] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 473–482, New York, NY, USA, 2002. ACM.

[LCR$^+$02] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 491–500, New York, NY, USA, 2002. ACM.