

MoDB: Database System For Synthesizing Human Motion

Timothy Edmunds S. Muthukrishnan Subarna Sadhukhan Shinjiro Sueda
Rutgers University

{tedmunds,muthu,sadhuka,sueda}@cs.rutgers.edu

Abstract

Enacting and capturing real motion for all potential scenarios is prohibitively expensive; hence, there is a great demand to synthetically generate realistic human motion. However, it is a central challenge in character animation to synthetically generate a large sequence of smooth human motion.

We present a novel, database-centric solution to address this challenge. We demonstrate a method of generating long sequences of motion by performing various similarity-based “joins” on a database of captured motion sequences.

This demo illustrates our system (MoDB) and showcases the process of encoding captured motion into relational data and generating realistic motion by concatenating sub-sequences of the captured data according to feasibility metrics. The demo features an interactive character that moves towards user-specified targets; the character’s motion is generated by relying on the real time performance of the database for indexing and selection of feasible sub-sequences.

1. Introduction

The increasing accessibility of high-quality motion-capture systems that record real character actions has provided an inexpensive alternative to hand-editing for the creation of large pools of motion data for character animation. Even so, it is often intractable to capture all motions that a character may be required to perform (particularly in interactive applications); an attractive option, therefore, is to synthesize the desired motions from a pool of captured motion.

When recombining captured motion frames to form novel sequences for character animation, there are two considerations that must be addressed: the feasibility of the motion, and the degree to which it accomplishes the objective desired by the animator.

Whereas a captured motion sequence guarantees feasibility (in that it was actually performed by a human), that

guarantee is lost when sub-sequences are concatenated. Previous work[1, 3, 4] has addressed the feasibility concern by pre-processing the captured data into a directed graph in which the nodes are sub-sequences of recorded motions (“snippets”) and an edge indicates that the corresponding concatenation yields a feasible motion. While this “motion-graph” approach has illustrated the utility of recombining motion snippets, it suffers from the inability to tradeoff a motion’s feasibility against how well the motion accomplishes the desired objective.

When synthesizing character animations from motion graphs, it is necessary to incorporate an objective function that, for a given set of feasible transitions, quantifies the degree to which the resulting motion would accomplish the goals of the animator. The specific function is dictated largely by the application; it can be as concrete as the Euclidean distance between the result and some trajectory[3, 4], or as abstract as matching high-level motion annotations (such as “running”, “jumping”, or “happy”)[2]. In the motion graph approach, the choice of transitions is limited to those deemed feasible at the time of pre-processing; while there might exist a combination that would better accomplish the objective, it will not be found if the corresponding edge is absent from the graph.

Our approach is to select transitions from *all* of the snippets in the data pool (without a crushing quadratic space or time cost) by leveraging the capabilities of a relational database management system.

2. MoDB

At the high level, our approach works as follows. We acquire small sequence of motion data using appropriate tools, and decompose the motion data into a set of contiguous sequences (“snippets”). We load the snippets and their attributes into a database. In order to generate synthetic motions, we phrase both the feasibility constraint as well as the animation objective function (discussed in §1) as queries over the database. By adjusting the parameters of the query, we can control the tradeoff between the feasibility of the motion and how well the motion accomplishes the animator’s objective. We balance database retrieval time against

motion execution delays to create motions that respond interactively to user input.

The motions synthesized from our data set are composed by stringing together short (~ 0.5 s) snippets of the original motion capture data. The feasibility of a snippet concatenation is based on a comparison of the configuration of the character's (simplified) skeleton at the beginning and end of each snippet.

A character configuration is represented as a relation (*MotionTerminator*) whose attributes describe the joint parameters in a 17 joint skeleton. The joints are either free joints with a full 6 degrees of freedom, or ball joints with 3 degrees of rotational freedom. 7 attributes for each free joint and 4 for each ball joint (orientations are stored as normalized *quaternions*) thus allows for the detection of C^0 continuity. In order to also allow for consideration of C^1 continuity, an extra 3 attributes encode angular velocity and 3 attributes encode linear velocity. In total, the *MotionTerminator* relation contains 137 double format attributes, and 1 primary key attribute.

In our application, the objective is to move the character toward some target in the character's environment. Thus, we need to be able evaluate the effectiveness of a given snippet at moving toward the target position. Our database contains a relation (*SnippetDescriptor*) that describes the pertinent aspects of an entire motion snippet; as well as references to a starting and an ending *MotionTerminator*, it contains 3 attributes that give the 3 dimensional net translation of the character that results from playing a snippet. To keep track of the character's local coordinate frame, we also record the snippet's final orientation (4 attributes). In total, the *SnippetDescriptor* contains 7 double format attributes, 2 foreign key attributes, and 1 primary key attribute.

When serialized in the most convenient format for driving an animated character in realtime, each motion snippet is ~ 100 KB in size. For a non-trivial data set, retaining the snippets in memory rapidly becomes infeasible. Since a DBMS is expected to be at least as fast at retrieval from disk as ordinary application I/O, we opted to store the snippets serialized as BLOBs in a third database relation.

When composing a query to select a snippet for concatenation, the feasibility criterion is expressed by a WHERE clause, and the objective-accomplishment criterion is expressed by an ORDER BY clause.

The WHERE clause of the query uses inequality comparisons to reject motion snippets whose starting configurations have joints that deviate more than a specified amount (according to one of a variety of available metrics) from the ending configuration of the previous snippet.

The ORDER BY clause that prefers snippets that take the character closer to its goal is simpler than the WHERE clause. After the application has transformed the target po-

sition into the character's current local coordinates, all that is necessary is to define an attribute in the output relation that contains the difference between the target location and the snippet's net translation. As with the WHERE clause, the choice of distance metric determines the attribute aggregation that is necessary.

Given the query structure described above, there is a great deal of flexibility in the actual query issued to select motion snippets.

The thresholds used to parameterize the similarity tests on the joint configurations can be varied to increase the number of candidate snippets at the cost of degrading the motion's feasibility (and vice-versa). In addition to tuning the query to trade motion quality off against the objective function, the query can be modified to improve the query execution time (also at the expense of motion quality). The choice of distance metric in both the WHERE and ORDER BY clauses affects how much per-joint attribute aggregation is necessary. In the WHERE clause, some joints may be ignored entirely to reduce the join complexity. Similarly, joint velocity and angular velocity may be ignored for some or all of the joints.

Reducing the complexity of the WHERE clause is not without its cost (as well as the degradation of the motion quality). Fewer constraints on the continuity match results in a larger number of selected records, which results in a larger sorting operation for the ORDER BY clause.

3. Demo

We implemented the MoDB system using MySQL for database technology, OpenGL for animation and Java for the application logic and database/graphical glue. The demo renders, in real-time, a character that responds to user input — the user interactively selects a goal for the avatar to pursue. Each time a new motion snippet is required, the system queries the DBMS for a snippet that will take the character as close to the goal as possible. We show the effect that the parameters controlling the selection of motion snippets have on query response time in the DBMS.

References

- [1] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. In *Proceedings of SIGGRAPH 2002*, July 2002.
- [2] O. Arikan, D. A. Forsyth, and J. F. O'Brien. Motion synthesis from annotations. In *Proceedings of SIGGRAPH 2003*, July 2003.
- [3] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of SIGGRAPH 2002*, July 2002.
- [4] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars with human motion data. In *Proceedings of SIGGRAPH 2002*, July 2002.