Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)
R. Boulic, D. K. Pai (Editors)

Evaluating Motion Graphs for Character Navigation

P. S. A. Reitsma[1] and N. S. Pollard[2]

[1]Brown University, Providence, RI — psar@cs.brown.edu
[2]Carnegie Mellon University, Pittsburgh, PA — nsp@cs.cmu.edu

Abstract

Realistic and directable humanlike characters are an ongoing goal in animation. Motion graph data structures hold much promise for achieving this goal. However, the quality of the results obtained from a motion graph may not be easy to predict from the input motion segments. This paper introduces the idea of assessing a data structure such as a motion graph for its utility in a particular application. We focus on navigation tasks and define metrics for evaluating expected path quality and coverage for a given environment. One key to evaluating a motion graph for navigation tasks is to first embed it into the environment in a way that captures all possible paths that might result from "playing back" the motion graph within that environment. This paper describes an algorithm for accomplishing this embedding that preserves the flexibility of the original motion graph. We use the metrics defined in this paper to compare motion datasets and to highlight areas where these datasets could be improved.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism:Animation

1. Introduction

One of our goals is to make realistic animation of human motion accessible to a broad range of users. Motion graphs have a particular appeal—the ideal vision is that a user can load a set of motion clips, automatically generate a graph structure, and instantly begin to direct the character using intuitive interfaces such as mouse gestures, full body demonstration, or specification of a set of constraints that should be satisfied.

However, the reality is often very different, and results depend on many parameters, such as the specific motion clips selected and thresholds chosen for transitions between motion segments. One open question is how much motion data should be included in a motion graph. Insufficient data is especially problematic, because it can result in a character with limited capabilities. For example, as Figure 1 illustrates, there may be paths through an environment that are difficult for the character to follow.

Much of the focus in constructing motion graphs to date has been on the local quality of the motion—creation of good blends and transitions and elimination of common artifacts such as footskating. These local quality metrics are critical for smoothly joining disparate motion segments to produce believable motion. However, more global metrics such as ability to effectively follow paths are also important for producing believable character performance.

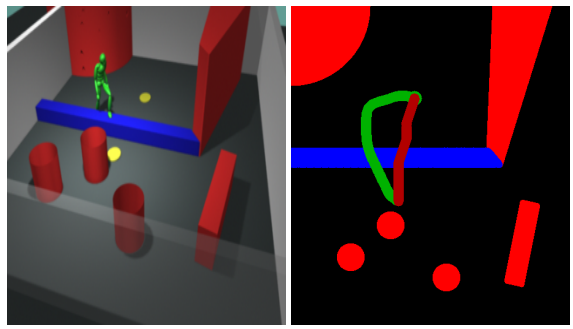

Figure 1: *A task that cannot be accomplished in a natural way using our test motion graph. The red (dark) path is the desired path. The green (pale) path is the shortest path available using this motion graph.*

This paper introduces the problem of evaluating more global properties of a motion graph data structure. The appropriate metrics for evaluation depend on the task, and this paper focuses on navigation, where the goal is to create a character that can travel through the environment while

maintaining natural looking motion. For navigation, quality of paths traveled through the environment and coverage—the ability to reach any position and orientation—are important and we suggest evaluation metrics related to these goals. Using these evaluation metrics, we compare several variations on a small motion dataset and highlight problems with this dataset in the context of a given environment.

To tie the motion to the environment and allow coverage to be evaluated, we introduce a new algorithm for embedding the motion graph into a particular environment. This algorithm places the motion graph into a global reference frame and reveals the extent to which the character will be capable of avoiding obstacles and achieving task goals within this particular environment. Our embedding algorithm has the advantage over previous work (e.g., [LCR*02][CLS03]) of embedding an *entire* motion graph into a given environment. Our goal is to capture the full range of motion variations that can be displayed by the character. The embedded graph may be useful for planning and real-time character animation as well as for the evaluation applications described in the current paper.

2. Background

Relatively little work has been done on evaluation of character capabilities. A number of researchers have explored user perception of animated human motion (e.g., [HOT98] [OHJ00] [RP03] [WB03] [HRv04]). In contrast, our goal is to evaluate more global properties of a character's behavior, such as the ability to perform a suite of tasks.

Interpolation approaches to motion generation (e.g., [WH97] [RCB98] [SRC01] [RSC01] [ZH02] [DJM04] [KG04]) ensure that motion can be generated for any point in a parameter space. However, interpolation may perform poorly for certain motions, such as those containing a flight phase [RP03].

Motion graph approaches to motion generation provide complementary benefits to interpolation. In particular, they encode natural transitions between behaviors and can generate motion with good local quality by constraining the motion to be very close to captured examples (e.g., [MTH00] [LCR*02] [KGP02] [AF02] [LWS02] [KPS03] [AFO03]). We use motion graph data structures for our experiments and focus on good global quality metrics, such as the ability of a character to follow paths through an environment when it is constrained to the collection of motion segments within a motion graph.

It can be advantageous to embed a motion graph into a particular environment (e.g., see Figure 2). Work in this area has been pioneered by Lee and his colleagues [LCR*02] [CLS03]. In one approach, the motion graph is constructed using a coordinate frame fixed to the environment. This approach can capture character interaction with objects within the constrained capture area [LCR*02]. However, it does not

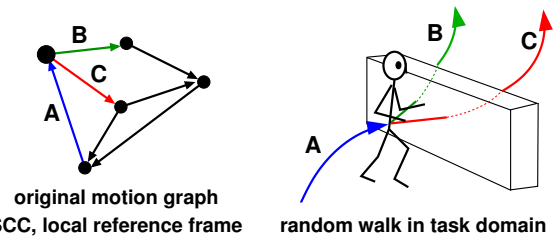

Figure 2: *(Left) A motion graph may be constructed using a reference frame local to the character to preserve flexibility in the character's motion. (Right) Using this motion graph to drive the character through an environment with obstacles can result in dead ends, however, which may thwart a local planner. Because obstacle information is not available in the local reference frame (left), extracting a strongly connected component (SCC) from the original motion graph does not guarantee that the character can wander through a given environment indefinitely.*

allow motion clips to be reused in different parts of the environment. In a second approach, Choi, Lee, and Shin [CLS03] show how a motion graph can be embedded into the environment by unrolling it onto a roadmap [KL98] constructed in that environment. This algorithm embeds a portion of the original graph into the environment, fixing it to the roadmap. We contribute to this previous work a novel embedding algorithm that maintains the advantages of both of these techniques; we embed an entire motion graph into a given environment in a way that (1) allows the same motion clip to be reused in different parts of the environment, and (2) allows us to choose among all available motion clips (i.e., we are not constrained to a roadmap). We note that while the algorithm of Choi et al. [CLS03] is elegant and efficient for path planning, we need a more complete representation of the motions available to the character to evaluate character capabilities in an equitable manner.

To make our embedding algorithm tractable, we use a grid-based approach, inspired by grid-based algorithms used in robot path planning (e.g., [LPO91] [Lat91] [DXCR93]). However, these algorithms cannot be adapted in a straightforward way to our problem, because they are designed to achieve a different objective: find a single good path for the robot to move from start to goal. In contrast, our embedding algorithm represents all possible paths through the environment for the purpose of evaluating character capabilities.

Finally, we mention the work of Gleicher and his colleagues to develop tools for editing motion graphs [GSKJ03]. Our work shares their goal of improving graph utility, for example to create poses that are "hubs," having a large branching factor. Our algorithms could assist the user in evaluating the utility of adding new motions to increase character capabilities or the utility of rearranging the graph by making different hub choices, for example.

3. Overview

The primary contributions of the paper are

- **A statement of the problem of evaluating global properties of motion generation algorithms.** We propose that it should be possible to quantitatively evaluate a character's ability to perform a suite of tasks and to use these results to compare motion graphs or other algorithms for motion generation.
- **A concrete scheme for evaluating motion graphs for navigation applications** (Section 4). In particular, we suggest metrics to capture expected path quality and ability of a character to cover an environment.
- **Benchmarking results for variations on a typical walking dataset downloaded from the web.** We show that good control of walking behavior is easy to achieve, but additional behaviors such as stepping over obstacles must be introduced with some care (Section 7).
- **A new algorithm to embed a motion graph into a given environment.** In contrast with previous research, our algorithm embeds the entire graph into the environment, capturing the variability inherent in the character's motion (Section 5).

The paragraphs below review our evaluation metrics (Section 4), embedding algorithm (Section 5), and results (Sections 6 and 7).

4. Evaluation Metrics

The appropriate metric for evaluating character capabilities depends on the task. For reaching or punching, the task may be to contact a target with an appropriate velocity profile; for some dance forms, the task may be to string together different dance moves while navigating within the constrained space of a dance floor. This paper focuses on navigation (primarily walking), and proposes two different metrics for evaluation: (1) expected quality of paths traveled through that environment and (2) coverage, or ability to reach any collision-free position and orientation in that environment.

4.1. Path Quality

The path quality metric is designed to capture the ability of the character to move through the environment efficiently. To estimate expected path quality, we must make an assumption about how paths for the character will be specified. In this paper, we consider the problem of point to point navigation. In point to point navigation, a start and end position are specified, and a path must be found for the character to travel between these two points. In the ideal case, near-minimal-length paths would exist for all start and end positions. The metric for relative path length in point to point navigation is expressed as follows:

$$E_P = \frac{pathLength}{minPathLength} \tag{1}$$

where *pathLength* is the result of integrating the change in position of the projection of the character root onto the ground plane from start to goal, and *minPathLength* is the minimum length path that would avoid collisions between the character and objects in the environment.

For benchmarking, we work with the embedded motion graph described in Section 5. Working with the embedded graph has two advantages: it improves the efficiency of shortest path calculations, and it also ensures that only paths that do not lead to dead ends are considered.

Given this embedded graph, we use Monte Carlo sampling to estimate the expected value of relative path length E_P. Start and goal positions are selected randomly from the set of collision-free positions in a given environment. Parameter *pathLength*, the shortest path length available to the character, is computed using Dijkstra's algorithm [CLRS01] on the embedded graph. Parameter *minPathLength* is found using breadth-first search through a grid placed over the ground plane. Path length metric E_p is then computed as in Equation 1.

4.2. Environment Coverage

The environment coverage metric is designed to capture the ability of the character to reach every portion of its workspace. For navigation, the workspace will typically be three-dimensional, and a point in this space will represent the position of the character root on the ground plane (x,z) and the yaw angle or facing direction (θ).

To evaluate coverage, we must first embed the motion graph into the environment, as described in Section 5. Without such an embedding, it would be difficult to identify coverage holes. Once the motion graph has been embedded into the environment, each frame of motion in the embedding can be associated with a 3D data point (x,z,θ). To measure coverage, we regularly sample (x,z,θ) space and compute for each sample the distance to the nearest data point.

From this distance information, coverage is estimated as follows:

$$C = \frac{\sum_i covered(i)}{\sum_i collisionFree(i)} \tag{2}$$

Where the numerator contains a count of all samples for which there is at least one data point within a given distance ε, and the denominator contains a count of all samples where the character would not collide with obstacles in the environment. In other words, coverage C is an estimate of the fraction of the collision-free (x,z,θ) workspace through which the character can pass.

To help the user diagnose potential problems in a dataset, we use a brushfire algorithm (e.g., [LRDG90]) to identify local maxima in terms of distance to the nearest data point in (x,z,θ) space.

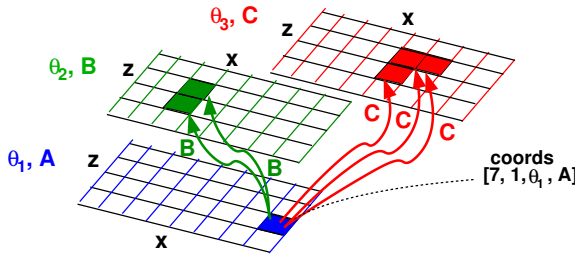

Figure 3: *Example of forming links for an embedding. The grid lies in a 4D space, indexed by 3D workspace configuration (x,z,θ) and by incoming motion clip (A, B, or C). Links from cell $[7,1,\theta_1,A]$ are formed by looping over all segments that follow from A and finding the destinations that can result from small modifications of the original motion.*

Example path quality and coverage measurements are presented in Section 7.

5. Embedding into the Environment

To answer the question of where the character can travel within a given environment, we must "unroll" the original motion graph into that environment. The primary challenge in performing this unrolling, or embedding, is to form a compact description of the space of feasible character trajectories.

To achieve this goal, we approximate the environment using a regular grid of cells. The original motion graph is then embedded into the environment by unrolling it from all cell centers. Unrolling from cell centers is not by itself sufficient to represent character capabilities well, however. Without further processing, the result is a collection of disconnected trees. To link these trees, we force motion segment endpoints onto cell centers; each motion segment must originate and terminate at a cell center. Section 5.1 describes an efficient one-step embedding process that follows from these assumptions.

Generating clips that originate and terminate at cell centers requires some editing of the original motion segments. If no editing is permitted, the embedded graph will be quite sparse, because motion segments would rarely end exactly at a cell center. In many applications, however, some amount of editing is acceptable, and introducing this flexibility when creating the embedded graph provides a more practical estimate of character capabilities. We assume that motion can be edited in proportion to distance traveled. As a result, longer clips may gain the ability to terminate at a collection of cell centers, all of which fall within the allowed "edit region." Section 5.2 describes in more detail the very simple motion editing model assumed in this paper.

Obstacle and other constraints are handled efficiently as part of the embedding process (Section 5.3). Section 5.4 discusses the correctness of the embedding algorithm.

5.1. One-step Unrolling into a Finite Space

When motion clips must originate and terminate at grid centers, embedding an entire motion graph into a task domain can be accomplished using a one-step unrolling process. First, a discretized 4D state space is defined. Each cell represents a state of the character and is indexed by position and orientation within the environment (x,z,θ) as well as by incoming motion clip. Cell $[7,1,\theta_1,A]$, shown in Figure 3, for example, indicates that after playing motion segment A, the character has arrived at position $(7,1)$ and orientation θ_1.

For each cell in the 4D state space, all possible links out of that cell are created. Figure 3 illustrates this process. To form all possible links out of a cell, we first identify outgoing motion segments by examining the original motion graph. We assume for this example that incoming motion segment A can link to outgoing motion segments B and C. One-step unrolling from cell $[7,1,\theta_1,A]$, then, will involve forming links from that cell to cells that can be reached by playing motion clips B and C from the starting point $(7,1,\theta_1)$. Some amount of motion editing may be acceptable, and so the number of destination cells associated with each motion clip may be greater than one. Section 5.2 describes how we constrain the amount of motion editing used.

Note that repeating this process just once for each cell results in forming all possible links in the graph. All starting configurations, all motion clips, and all of the editing variations that we model are considered. We also mention that although the character is only able to enter the highlighted cell in Figure 3 along motion segment A, this cell may have many incoming links at the end of the one-step unrolling process. The character may be able to enter cell $[7,1,\theta_1,A]$ by playing motion segment A from several different locations.

After the one-step unrolling process has been performed for all cells, we find the largest strongly connected component (SCC) in the directed graph [CLRS01]. This SCC is the desired embedded graph. Extracting only the SCC ensures that the character can traverse the graph without encountering a dead end, which is important for ensuring the character will always be in a state within which autonomous or interactive control will be viable.

If we assume the character is to navigate continuously throughout the environment, this embedded SCC is convenient for computing coverage of that environment. It is also useful for computing path quality. For example, consider evaluating path quality between two points using search techniques on the original motion graph. Potentially, all of the best paths between the points could lead unavoidably to states where the character has no motions available (e.g., the only available motions take the character through obstacles). Using standard search techniques to discover that these paths are dead ends would cause us to incur additional computational expense.

Moreover, since estimating an expected level of path qual-

ity requires sampling path quality in many places, there is the risk that computationally expensive path-validation will be duplicated over many samples. Failing to discover that these paths are dead ends, however, may lead to incorrect estimates of path quality. While it has a high one-time computational cost, using an embedded graph structure to compute path quality allows efficient and correct computation of the many local path quality tests required to gain an accurate measure of global path quality available in the environment using the original motion graph.

5.2. Motion Editing Constraints

The number of links created in the embedded graph and the resulting estimate of coverage (Equation 2) depend on the extent to which we are willing to edit the motion segments in the original motion graph. It is not well understood how to quantify the amount of motion editing that is acceptable for motions such as walking; perceptual experiments are needed to develop a well grounded model. To illustrate the effect of editing on coverage, we assume a very simple model, based on the intuition that the amount that a motion can be changed should grow with distance traveled. In particular, we specify the amount that the root position and orientation can be adjusted as a linear function of distance traveled:

$$abs\,(p(a) - p_M(a)) < a \begin{bmatrix} r_x \\ r_z \\ r_\theta \end{bmatrix} \qquad (3)$$

where $(p(a) - p_M(a))$ is the vector difference between the original and new root configurations, a is the arclength of the original motion clip, and $a[r_x \;\; r_z \;\; r_\theta]^T$ is the size of the ellipsoid representing allowable edits to root configuration at arclength a. In our experiments, which had a grid spacing of $20cm$ in position and $20degrees$ in orientation, we used the value $25cm/m$ for r_x and r_z and $25degrees/m$ for r_θ. Figure 4 shows a 2D sketch to illustrate the idea of this "edit region."

This linear model serves as a simplified abstraction of the different motion editing techniques one could employ; the edit region is our estimate of the points to which we could possibly warp the endpoint of a motion, given its starting point, the motion editing techniques available, and limitations on motion editing that are imposed to maintain motion quality. Our model of motion editing estimates that warping the motion to end at any of the grid points outside the edit region will result in an edited motion that may contain unacceptable artifacts.

5.3. Obstacle Avoidance and Annotation Constraints

Many links will not be possible due to collisions with obstacles in the environment or other constraint violations (e.g., attempting to walk across a chasm). For obstacle avoidance, constraint checking is done at the time the links are formed. We do not process cells where the character would collide with obstacles; links that would result in collisions between the character and the environment are culled from the graph before the final SCC is computed.

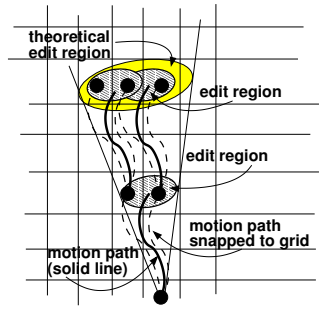

Figure 4: *In our grid-based algorithm, the end of each motion segment is snapped to the centers of cells based on a region of acceptable destinations computed for that motion segment. This figure compares how growth in this theoretical "edit region" compares to the regions actually used in the grid-based algorithm.*

Other types of constraint violations are handled using annotation constraints, where the environment is annotated to indicate regions where a specific type of motion should or should not be performed. This constraint type was used to map stepping motion to the blue obstacle in Figure 1. The same procedure could also be used for more freeform annotations, such as placing a particular dance move in front of the judges.

5.4. Correctness of the Embedding Algorithm

Our embedding algorithm is resolution complete in the sense that a sufficiently fine grid will capture all significant variations in character paths that are possible given our motion graph, task domain, and motion editing model. However, at practical grid resolution, some possible paths will be lost. Figure 4 shows an example. Consider a straight walking motion that can repeat. Figure 4 shows a sketch of edit regions grown using our approach, which requires forcing motion clips to terminate at cell centers. The theoretical edit region that would result from playing the same motion clip twice in succession is also shown. Eventually, (after playing the motion clip three or four times in succession), the theoretical edit region will contain cell centers that are not captured by our algorithm. As the grid is made finer, the practical effect of this approximation will decrease.

The embedding algorithm can be made conservative by making connections only to grid centers within the edit region associated with a motion segment. If the algorithm is implemented in this way, no paths can be generated that are not possible given the motion graph, task domain, and motion editing model.

We chose to use a slightly different approach, however,

and always make a connection between the end of a motion segment and the nearest grid center, regardless of whether that center is in the interior of the edit region. Our approach preserves the connectivity of the original motion graph. However, if the size of the edit region is small compared to grid size, it can introduce paths that are not possible given our model. To avoid introducing such paths, grid resolution should be kept sufficiently small that all edit regions will contain at least one grid point. (Looking ahead to the results, Table 2 shows how poor coverage can result from mismatching grid size and edit size.)

6. Example Scenario

Our primary test scene is a cluttered environment through which the character must navigate (Figure 1). The size of the room was 7m by 8m. The room was divided into cells with a resolution of 20cm, and a grid spacing of 20 degrees was used for the character's orientation about the vertical axis.

We began by forming a motion graph in the character's local reference frame. The motion set for this graph was 23 motions downloaded from the CMU motion capture database (mocap.cs.cmu.edu). These motions included 6 straight walks, 4 gradual left turns, 4 gradual right turns, 4 sharp left turns, 4 sharp right turns, and 1 stepping motion. The motions were relatively short clips and the total amount of motion was approximately 77 seconds. Costs for transitions between motion segments were computed using the technique of Lee et al. [LCR*02]. The map of transition costs was then processed to identify local maxima and to enforce a user-specified minimum clip length. The Appendix contains some details of our motion graph algorithm. The minimum clip length was enforced to help ensure that snapping motion segment endpoints to grid centers did not create unreasonable artifacts. When identifying transition locations that were local maxima, we used an objective function that attempted to preserve the largest possible number of connections to other motion segments, with each connection weighted by quality of the transition; our goal was to preserve as much flexibility in the character's motion as possible given our clip length restrictions. Forming a motion graph in the character's local reference frame on our motion set resulted in an SCC with 64 nodes and 154 links.

The motion graph in the character's local reference frame was then embedded into the environment as described in Section 5. The code was written in Java, and run on a 2.2GHz Xeon computer. Simple collision checking was done by giving the character a uniform radius in the horizontal directions. The embedded graph had 195K nodes and 963K links, and required approximately 5 minutes to compute.

When playing back motion through the embedded graph, transitions between motion clips were smoothed by using quaternion splines to gradually close the gap formed at the transition point. Motion editing to warp clips to grid centers was done using displacement splines on the root translation and yaw angle. This editing technique of course creates foot sliding artifacts, which should be cleaned up in postprocessing.

Motion Set	Source Frames	MoGraph Frames	Coverage Fraction XZ	XZA
Full Set	2304	1177	0.951	0.904
Half Motions	1424	1010	0.955	0.905
No Duplicates	733	537	0.922	0.879
No Sharp Turns	1551	948	0.942	0.893
No Gradual Turns	1543	682	0.958	0.896

Table 1: *Frames of source motion, frame size of the motion graph, and coverage comparison for the different motion datasets. XZ coverage indicates the fraction of the collision-free ground plane that can be reached at some orientation. XZA coverage is the fraction of all collision-free 3D configurations* (x, z, θ) *that can be reached.*

7. Results

Motion Dataset Comparison. We begin by comparing different motion data sets. How much motion is really needed to allow the character to perform reasonably in the environment? To explore the value of having duplicate motions and different types of turns, we examined results from 5 different motion sets, starting from the original set of 23 motions containing straight walks, gradual turns, sharp turns, and stepping:

1. the original dataset: 23 motions
2. half of the motions of each type removed: 12 motions
3. one motion of each type retained: 6 motions
4. all sharp turns removed: 15 motions
5. all gradual turns removed: 15 motions

The number of frames in the resulting motion graphs is shown in Table 1. Note that the number of frames in these motion sets does not vary as much as might be expected. Because we enforce a minimum clip length of 0.333 seconds when forming the motion graph, doubling the number of motions does not always lead to doubling the number of frames in the motion graph. Through experimentation, we found that when enforcing a minimum clip length, short source motions (2–3s) tended to provide far fewer usable clips as compared to longer source motions (4–5s) than their relative sizes would suggest. The *Half Motions* and *No Duplicates* sets of source motions do not contain these shorter motions, accounting for their more efficient utilization of the available frames.

Coverage. Table 1 also shows the differences in coverage obtained from different motion sets. Coverage was quite

Editing Size	Coverage Fraction			
	Always connect		Respect edit bounds	
	XZ	XZA	XZ	XZA
5	0.557	0.256	0.000	0.000
10	0.772	0.538	0.241	0.145
15	0.882	0.737	0.624	0.405
20	0.938	0.875	0.897	0.821
25	**0.951**	**0.904**	0.949	0.896
30	0.969	0.943	0.969	0.942
35	0.974	0.954	0.974	0.954

Table 2: *Effect of motion editing size on coverage. Editing size is the value of r_x and r_z in $cm/meter$ and also the value of r_θ in $degrees/m$ in Equation 3. "Always connect" means to always connect to at least one cell center. "Respect edit bounds" means only connect to cell centers within the edit region. XZ coverage and XZA coverage are as in Table 1. Numbers in bold show our experimental settings. Results depend on grid spacing, which was 20cm for x and z and 20 degrees for* θ.

high for all examples, and only pruning the motion set all the way down to 6 clips had a sizable effect.

How much motion editing is required to obtain good coverage? Table 2 shows how coverage for the full dataset varies with the amount of motion editing permitted. An editing size of 25 (cm/m in position and $degrees/m$ in orientation) was used in our experiments. When editing size was smaller than 20, we observed degradation in ability to form high quality paths at the grid resolution used in our experiments.

While allowing more motion editing permits greater coverage of the environment, in practice there are limits on the extent to which a motion can be edited while remaining of high enough quality to meet the requirements of the application. When editing size is larger than 25, we observe unacceptable footsliding artifacts in the resulting motion. Although our current estimate of a reasonable amount of motion editing is subjective, it is our hope that it will be possible to develop perceptually based algorithms to automatically compute such estimates.

It can be helpful to display the locations of the largest gaps or holes; Figure 5 illustrates these results. Arrows indicate orientation at local maxima and typically point toward or away from the nearest obstacle. The most obvious problem that can be inferred from this figure is that there are no motions in the dataset that allow the character to stop and turn in place.

Path Quality. Table 3 shows the differences in ability to navigate from a start to an end point for the different test motion sets. In all cases, the median motion is quite good. Motions with no gradual turns show the poorest performance, especially in the number of paths more than 25% longer than the minimum ($E_P > 1.25$). Motions with no sharp turns produce a relatively large motion graph and good coverage (Table 1), but produce many poor paths due to the inability to take sharp corners. (This problem is especially apparent in the $E_P > 1.1$ column.) All of the motion sets tested have some combinations of start and end positions that produce poor results.

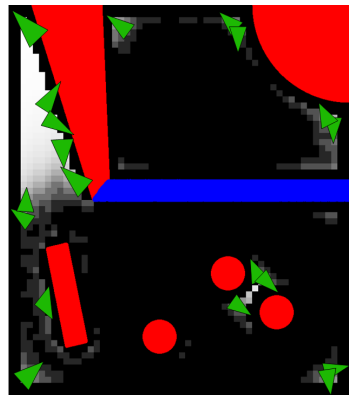

Figure 5: *Holes in coverage. The brightest regions are the farthest distance from covered, collision-free cells. This figure shows at each x,z position the maximum distance over all orientations* θ. *Arrows indicate orientations of some of the local maxima.*

Motion Set	Path Quality		% Poor Quality	
	Median	95%	$E_P > 1.1$	$E_P > 1.25$
Full Set	1.0066	1.124	7.0	2.0
Half Motions	1.0063	1.131	6.6	2.8
No Duplicates	1.0083	1.140	8.2	2.2
No Sharp Turns	1.0066	1.133	9.0	2.2
No Gradual Turns	1.0082	1.180	8.8	3.8

Table 3: *Path quality comparison for the different motion datasets. The median column lists median path length as a fraction of the minimum path length (E_P in Equation 1). The 95% column lists 95th percentile values for E_P. The "$E_P > 1.1$" column lists the percent of paths tested having a path length ratio greater than 1.1, and the "$E_P > 1.25$" column lists the percent of paths with path length ratios greater than 1.25.*

For the complete motion set, Figures 1 and 6 show examples of median and poor paths identified during Monte Carlo sampling. From this and other poor paths identified, we can infer that the character cannot easily pass between the two closest barrels and that the stepping motion is not sufficiently flexible to generate consistently good motions over the step obstacle. It would be useful to add additional stepping motions to the dataset, especially preceded and followed by turns.

Video. The video shows some additional paths, along with their path length ratios E_P. Note that only the positions of the

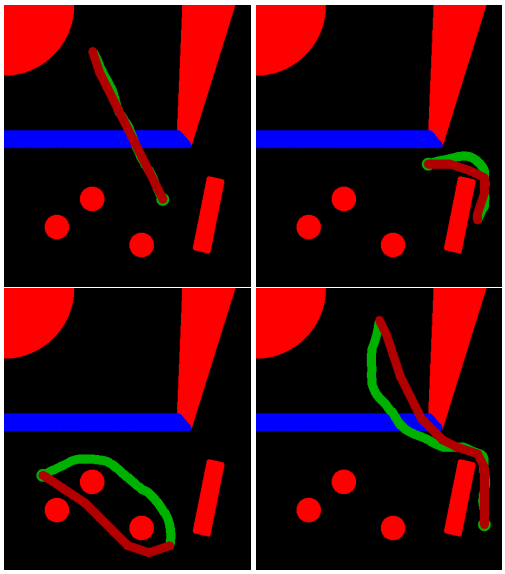

Figure 6: *Examples of paths found during Monte Carlo sampling. The upper left path has median value for path length ratio E_P. The other paths are all near the 95th percentile, meaning that 5% of paths are as poor or worse. Specific values for E_P are 1.002, 1.15, 1.16, and 1.11.*

start and end points were specified. Any initial orientation could be used, and the initial orientations shown are those that lead to the shortest paths. For these examples, the motion graph is lacking the motions or transitions that would make more direct paths possible.

8. Discussion

This is to our knowledge the first attempt to evaluate global properties of a motion graph data structure such as ability to reach all points in an environment and ability to follow a path. Embedding a motion graph into the environment and assessing global properties of this data structure allows us to compare motion datasets and identify weak points in these motion datasets.

In our exploration of a very small walking dataset, we were able to display evidence of two specific problems: (1) the lack of motions for stopping and turning made it difficult for the character to get into tight spaces, and (2) the single stepping motion we used was not sufficient to produce consistently natural looking behavior when moving between the two regions of the environment that are separated by the step obstacle. We also were intrigued by two other observations: (a) adding duplicate motions provided a relatively small improvement in path quality and coverage, and (b) extremely small motion datasets were capable of producing good behavior on average for this test application.

Design of Good Metrics. The metrics that we chose—path quality and coverage—are specific to the task of navigation, and many applications, even those relying on navigation, may benefit from a different choice of metrics. Much work remains to explore how the idea of evaluating a motion graph can be adapted to a variety of applications and used to improve algorithms for motion graph generation. Our goal in this paper was to introduce the idea, work through a concrete example, and show that quantitative evaluation techniques can provide insight into character capabilities captured in a motion graph data structure.

Scalability. Regarding timing, embedding the graph into the environment takes 5 minutes (including all program setup), determining coverage requires 32 minutes, and performing 500 path length runs requires 50 minutes for the full motion set.

As with any motion graph algorithm, the size of the original motion graph may scale poorly with the number of frames in the example motions, but in practice is typically found to scale well when some care is used to select transitions. Once the original motion graph has been formed, time required to run the embedding algorithm scales linearly with the number of links in the original motion graph. Path length computation time depends on the branching factor of the embedded graph, but over the range of our examples, it was strongly linear in the size of the original motion graph ($F = 239$, $p < 0.0006$). The number of edges in the embedded graph also scaled linearly in the size of the original motion graph over the range of our examples ($F = 157$, $p < 0.0002$). The coverage calculation scales linearly with the size of the embedded motion graph.

Practically, accommodating significantly larger motion sets should be feasible with the current implementation, and we are currently exploring the extent to which the basic algorithm will scale. To scale to very large motion databases will require either separating out distinct behaviors as suggested by Kovar and his colleagues [KGP02] or pursuing an approach that involves clustering and/or multiple levels of resolution, as in the work of Arikan and his colleagues [AFO03]. We are interested in investigating efficiency improvements such as these.

To scale to large environments, we envision a form of tiling. The environment would be represented as a collection of tiles—embedded graphs that are computed and analyzed separately, but that can be connected together in a seamless way. An ability to use the same tile in different parts of the environment would result in considerable computational savings.

The most extreme case is open space or a space that can be regularly tiled. In this case, the embedding algorithm is run on a single tile, and edges are allowed to wrap around and back onto that tile (i.e., the tile is topologically equivalent to a torus). Figure 7 shows an example.

One drawback of grid-based techniques is that they scale

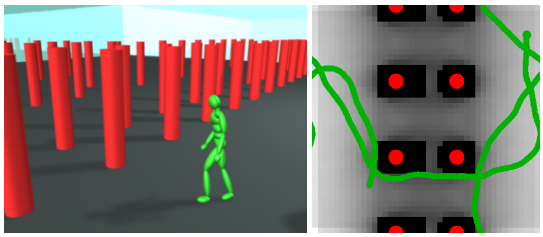

Figure 7: *(Left) The character is following a random path through a motion graph that has been embedded into an environment tiled with a repeating pattern. (Right) A random path wraps around from left to right, bottom to top, then right to left.*

exponentially with the dimensionality of the task space. While this cannot be avoided, Kovar et al. [KG04] demonstrate that interesting tasks can often be parameterized and controlled with only a very few degrees of freedom. We hope to explore techniques such as tiling for compositing tasks that are low-dimensional in isolation, allowing complex behaviors to be efficiently formed from layers of simpler behaviors.

Static vs. Dynamic Task Domains. The algorithms presented in this paper are appropriate for the portion of the task domain that is static. Accommodating moving obstacles is an area of future work. However, we note that good coverage and path quality within the static portion of the environment are necessary conditions for good performance when moving obstacles or other dynamic objects are added.

Building Good Motion Graphs. One of our longer term goals is to use tools such as these to direct a motion capture session. In some motion capture scenarios, the talent is only available for a short period of time. It is important to collect all of the motion that will later be necessary. It would be useful to be able to monitor the motion database as it is being collected, evaluate the intermediate results for an intended task, and suggest new motions that should be obtained to fill in any gaps (e.g., by extrapolating from existing motions).

Working with Other Motion Generation Algorithms. We note that the evaluation algorithms described in this paper could be applied to alternative techniques for generating motion. For example, a character controlled by a procedural algorithm could be run through its repertoire of actions, either systematically or by hand (i.e., under interactive user control). The resulting motion could be treated as motion capture data, embedded into a task domain, and evaluated to assess how the character might operate in that domain.

In summary, the techniques shown here provide a way to evaluate a character's capabilities that is more sound than the trial and error approach commonly used. Evaluation techniques such as these can help a user to compare alternative data structures and identify areas in which a motion graph or motion generation algorithm can be improved.

Acknowledgments

This research was supported in part by NSF grants IIS-0205224 and IIS-0326322.

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Transactions on Graphics 21*, 3 (2002), 483–490.

[AFO03] ARIKAN O., FORSYTH D. A., O'BRIEN J. F.: Motion synthesis from annotations. *ACM Transactions on Graphics 22*, 3 (2003), 402–408.

[CLRS01] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: *Introduction to Algorithms*. MIT Press, 2001.

[CLS03] CHOI M. G., LEE J., SHIN S. Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics 22*, 2 (2003), 182–203.

[DJM04] DRUMWRIGHT E., JENKINS O. C., MATARIĆ M. J.: Exemplar-based primitives for humanoid movement classification and control. In *Proc. IEEE Intl. Conference on Robotics and Automation* (2004).

[DXCR93] DONALD B., XAVIER P., CANNY J., REIF J.: Kinodynamic motion planning. *Journal of the ACM 40*, 5 (1993), 1048–1066.

[GSKJ03] GLEICHER M., SHIN H. J., KOVAR L., JAPSEN A.: Snap-together motion: Assembling run-time animations. In *Proceedings of 2003 Symposium on Interactive 3D Graphics* (2003).

[HOT98] HODGINS J. K., O'BRIEN J. F., TUMBLIN J.: Perception of human motion with different geometric models. *ACM Transactions on Graphics 4*, 4 (October 1998), 307–316.

[HRv04] HARRISON J., RENSINK R. A., VAN DE PANNE M.: Obscuring length changes during animated motion. *ACM Transactions on Graphics 23*, 3 (2004).

[KG04] KOVAR L., GLEICHER M.: Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics 23*, 3 (2004).

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Transactions on Graphics 21*, 3 (2002), 473–482.

[KGS02] KOVAR L., GLEICHER M., SCHREINER J.: Footskate cleanup for motion capture editing. In *Proceedings of ACM Symposium on Computer Animation 2002* (2002).

[KL98] KAVRAKI L. E., LATOMBE J.-C.: Probabilistic roadmaps for robot path planning. In *Practical Motion Planning In Robotics: Current Approaches and Future Directions*, Gupta K., del Pobil A., (Eds.). John Wiley, 1998, pp. 33–53.

[KPS03] KIM T., PARK S. I., SHIN S. Y.: Rhythmic-motion synthesis based on motion-beat analysis. *ACM Transactions on Graphics 22*, 3 (2003), 392–401.

[Lat91] LATOMBE J. C.: *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[LCR*02] LEE J., CHAI J., REITSMA P. S. A., HODGINS J. K., POLLARD N. S.: Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics 21*, 3 (2002), 491–500.

[LPO91] LOZANO-PÉREZ T., O'DONNELL P. A.: Parallel robot motion planning. In *Proc. IEEE Intl. Conference on Robotics and Automation* (1991).

[LRDG90] LENGYEL J., REICHERT M., DONALD B. R., GREENBERG D. P.: Real-time robot motion planning using rasterizing computer graphics hardware. *Computer Graphics (Proceedings of ACM SIGGRAPH 90) 24*, 4 (1990), 327–335.

[LWS02] LI Y., WANG T., SHUM H.-Y.: Motion texture: a two-level statistical model for character motion synthesis. *ACM Transactions on Graphics 21*, 3 (2002), 465–472.

[MTH00] MOLINA-TANCO L., HILTON A.: Realistic synthesis of novel human movements from a database of motion capture examples. In *Proc. IEEE Workshop on Human Motion 2000* (2000).

[OHJ00] OESKER M., HECHT H., JUNG B.: Psychological evidence for unconscious processing of detail in real-time animation of multiple characters. *Journal of Visualization and Computer Animation 11* (2000), 105–112.

[RCB98] ROSE C. F., COHEN M. F., BODENHEIMER B.: Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications September/October* (1998), 32–40.

[RP03] REITSMA P. S. A., POLLARD N. S.: Perceptual metrics for character animation: sensitivity to errors in ballistic motion. *ACM Transactions on Graphics 22*, 3 (2003), 537–542.

[RSC01] ROSE C. F., SLOAN P.-P. J., COHEN M. F.: Animation: Artist-directed inverse-kinematics using radial basis function interpolation. *Computer Graphics Forum 20*, 3 (2001).

[SRC01] SLOAN P.-P. J., ROSE C. F., COHEN M. F.: Shape by example. In *Proceedings of the 2001 ACM Symposium on Interactive 3D Graphics* (2001), pp. 135–143.

[WB03] WANG J., BODENHEIMER B.: An evaluation of a cost metric for selecting transitions between motion segments. In *Proceedings of ACM Symposium on Computer Animation 2003* (2003).

[WH97] WILEY D. J., HAHN J. K.: Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications Nov/Dec* (1997), 39–45.

[ZH02] ZORDAN V. B., HODGINS J. K.: Motion capture-driven simulations that hit and react. In *Proceedings of ACM Symposium on Computer Animation 2002* (2002), pp. 89–96.

Appendix: Transition Selection

Given an initial set of candidate transitions between frames of the input motions, we use the following iterative algorithm to compute the final transitions with an enforced minimum clip size:

Repeat
- Find window w with $count(w) > 1$, frame $c \in w$ that maximizes $value(w,c)$
- $coalesce(w,c)$

Until $count(w) \leq 1 \forall$ windows w

window(s, f, m): the set of m frames in source file s ranging from f to $f+m-1$. Here, m is always the minimum clip size, and w will always refer to a window of this sort.

count(w): the number of frames $f \in w$ s.t. $\exists$ frame $g \in DB$ s.t. $equivalency(f,g) > 0$.

DB: the set of frames of the input source motion files

value(w,c): $\sum_g equivalency(c,g) \quad \forall g$ s.t. $\exists f \in w$ s.t. $equivalency(f,g) > 0$. i.e., the value of window w if all transitions must go through frame c.

equivalency(f,g): the similarity of frames f and g, if that value is greater than the acceptance threshold, else 0.

coalesce(w,c): $\forall g \in DB$ s.t. $\exists f \in w$ s.t. $equivalency(f,g) > 0$ recompute $equivalency(c,g)$ and set $equivalency(f,g)$ to 0. i.e., force all transitions inside window w to use frame c or no frame at all.