Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2005)
K. Anjyo, P. Faloutsos (Editors)

Fast and accurate goal-directed motion synthesis for crowds

Mankyu Sung, Lucas Kovar and Michael Gleicher

Department of Computer Sciences
University of Wisconsin – Madison

Abstract

This paper presents a highly efficient motion synthesis algorithm that is well suited for animating large numbers of characters. Given constraints that require characters to be in specific poses, positions, and orientations in specified time intervals, our algorithm synthesizes motions that exactly *satisfy these constraints while avoiding inter-character collisions and collisions with the environment. We represent the space of possible actions with a motion graph and use search algorithms to generate motion. To provide a good initial guess for the search, we employ a fast path planner based on probabilistic roadmaps to navigate characters through complex environments. Also, unlike existing algorithms, our search process allows for smooth, continual adjustments to position, orientation, and timing. This allows us both to satisfy constraints precisely and to generate motion much faster than would otherwise be possible.*

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

1. Introduction

This paper presents a highly efficient algorithm for synthesizing realistic goal-directed motion for large numbers of characters. We focus on the important problem of character navigation, and introduce an algorithm for creating motions that are collision-free and that precisely satisfy constraints on duration, position, orientation, and body pose. For example, we might require multiple characters to meet (i.e., face each other) at a specified position and time, or we might require a collection of characters to navigate through a building into a theater and then sit down in an array of seats. Our algorithm is capable of animating entire groups of characters at better than real-time rates, i.e., the motion for the group takes less time to generate than it takes to play. Given a higher-level control mechanism for directing behavior [HM95, HFV00, FBT99, MT01, BC97, SCG04], our algorithm is well suited as a back end for efficiently generating detailed motion for each individual character that meets specifications on where and when desired actions should occur.

To generate high-quality motion with low computational cost, we represent the space of possible motions with a motion graph generated through the method of Gleicher et al. [GSKJ03]. This motion graph is a directed graph where each edge contains a clip of motion and nodes correspond to character poses that are shared by the end of all incoming clips and the beginning of all outgoing clips (Figure 1). By construction, any clip entering a node can be seamlessly connected to any clip leaving that node simply by concatenating them. This makes synthesis highly efficient, and it reduces the problem of creating a specific motion to finding an appropriate sequence of edges on the motion graph.

To create motions that avoid collisions and satisfy user-defined constraints, we proceed in two steps. We first use a fast path planner based on probabilistic roadmaps (PRMs) [KL94] to navigate through complicated environments and produce motions that approximately satisfy the constraints. The result is then refined through a randomized search algorithm that yields a motion which exactly conforms to the constraints. Many existing graph-based synthesis techniques also use search algorithms to generate motions that satisfy constraints [AF02, AFO03, GSKJ03, HGP04, KGP02, LCR*02, LL04]. However, because new motions can only consist of static clips from a fixed set, in general this approach cannot satisfy constraints exactly. For example, if a graph only contains clips where a character turns in 30 degree increments, then that character can never

end up facing a direction that is not a multiple of 30 degrees. This can lead to fruitless, time-consuming searches for motions that cannot exist. Rather than limiting synthesis to attaching fixed clips, we modify the search to allow for a continual, gradual adjustment of the character's position, orientation, and speed in the synthesized motion. This has two important advantages:

1. The search algorithm has greater flexibility to accurately satisfy constraints.
2. Motions can be constructed more quickly because the search need not compute *optimal* motions, but rather motions that are "close enough".

To provide guarantees on motion quality, we restrict the amount that a synthesized motion can be adjusted. The adjustment tolerance provides a natural mechanism for striking a balance between efficient synthesis and guarantees on motion quality. However, in practice, adjustments that are small enough to be difficult to discern are sufficient to make constraint satisfaction reliable and efficient.

The remainder of this paper begins with a review of related work in Section 2, then describes our synthesis algorithm in detail in Section 3 and presents results in Section 4, and concludes with a brief discussion in Section 5.

2. Related Work

Most previous work in graph-based motion synthesis has created new motions by connecting existing clips from a database with simple interpolation or displacement mapping methods [AF02, AFO03, GSKJ03, HGP04, KGP02, LCR*02, LL04]. Because these methods create new motions strictly by attaching existing clips, in general constraints on the generated motion are not exactly satisfied, and finding a solution that minimizes deviation from the constraints can be time consuming. In contrast, our work incorporates adjustments to a character's position, orientation, and speed directly into the search process. This allows the search to terminate whenever a sufficiently close (but perhaps suboptimal) motion is found, and it allows position, orientation, and timing constraints to be satisfied precisely rather rather than approximately. The most similar existing algorithm to our own is that of Choi et al. [CLS03]. Given a motion graph containing locomotion data, they randomly sampled footprint configurations in the environment (i.e., foot positions and orientations on the ground) and identified pairs of footprints which, within a tolerance, matched the configuration of footprints within at least one clip in the motion graph. The result was a probabilistic roadmap that associated short paths in the environment directly with motion clips. Character paths were computed by using Dijkstra's algorithm to find an appropriate sequence of footprints in the PRM, and motion was generated by attaching and modifying the corresponding motion clips so as to pass through the footprints. Our algorithm instead uses a PRM to obtain approximate motions which are then refined through a fast randomized search algorithm. This allows us to only add in the adjustments necessary to satisfy the user's constraints, rather than requiring synthesized motions to pass through a relatively dense set of sampled footprints.

Procedural motion synthesis is an alternative to graph-based synthesis that can also be well-suited for real-time applications. In particular, several methods have been developed for procedurally generating controllable walking motion [BC89, BMTT90, SM01, BUT04]. These methods can create motions that exactly follow effectively arbitrary trajectories, eliminating the need for additional search mechanisms. However, it is quite difficult to reproduce the realism of motion capture data with procedural methods. Also, existing procedural algorithms can only produce neutral walking motions, whereas graph-based methods trivially extend to more stylized and/or unusual forms of locomotion [KGP02].

A second real-time alternative to graph-based synthesis is motion blending [GR96, PSS02, PLS03], which (as with procedural synthesis) allows continuous control over character trajectories. However, blending-based models are more restricted since individual clips must be blendable, whereas graph-based models only require clips to be similar at isolated points. Moreover, blending-based synthesis incurs additional computational overhead since each generated character pose must be formed by combining example poses. Hybrid graph/blending methods [RCB98, KPS03, PSS04] share similar concerns.

Crowd animation has been the subject of many previous research efforts. Much of this work has focused on behavior modelling and has used only abstract character representations such as point particles [HFV00, HM95, BC97]. Other work [MT01, UT02, FBT99] has used more explicit representations of human figures, but has focused on high-level control and hence has not been concerned with constructing detailed movement for individual characters. Our work, in contrast, is concerned with constructing detailed individual motions for groups of characters given constraints on these motions. This paper thus complements these previous efforts by providing a back end for synthesizing motions that meet constraints issued by a higher-level behavior modelling module. Other research efforts have, as with our work, used motion graphs to model detailed character motion [SCG04]. However, as noted earlier, with existing graph-based synthesis algorithms constraints cannot be exactly satisfied. This paper provides a related but alternative synthesis procedure that uses smooth, continuous motion adjustments to provide exact and efficient character control.

Our algorithm draws upon tools that are well established in the existing literature. PRMs have been used extensively in previous work to perform path planning for character navigation [PLS03, CLS03]; alternative techniques, such as rapidly-exploring random trees [LK00, LK99, KL00], could also be used. Our randomized search algorithm is related to

the ones suggested by Shödl and Essa [SE02] and Arikan and Forsyth [AF02]. Finally, our method for adjusting motions to better satisfy constraints is a version of displacement mapping [BW95, WP95], and it is similar to the algorithm used by Reitsma and Pollard to force motion paths to start and end on predefined grid cell locations [RP04]. This paper combines these techniques into a strategy for efficiently and controllably animating large numbers of characters.

3. Synthesizing Motion

3.1. Overview

Given a group of characters and a set of constraints on each character's configuration, our goal is to synthesize motions for the individual characters such that all constraints are satisfied and no collisions occur. Each constraint can specify a character pose, a position $\mathbf{p}$ and orientation θ for this pose, and a time interval $[t^a, t^b]$ in which this configuration must be obtained (possibly $t^a = t^b$). The time constraint can either be absolute (i.e., the character must arrive at a spot 3 seconds from now) or relative to another character's motion (i.e., the character must arrive at a spot within 1s another character). Not all of these components need to be specified — for example, one might require a character to move to a particular spot without specifying a pose, orientation, or time interval.

To avoid inter-character collisions, individual characters are processed sequentially, with characters whose motions have already been planned treated as moving obstacles. This limits the size of the search space and as a result is considerably more efficient and scalable than processing all characters simultaneously. While in principle sequential processing may result in artificially unsatisfiable constraints, typically there are many possible motions that satisfy a given set of constraints, and we have found that in practice sequential processing does not prevent sophisticated group animations from being generated. The specific processing order is arbitrary except insofar as is necessary to satisfy timing relationships — for example, if character A must arrive somewhere 1*s* after character B, then B is processed first. We assume that the timing constraints have no circular dependencies, so a feasible ordering always exists.

We represent the actions available to a character with a motion graph (Figure 1). Each edge corresponds to a clip of motion, and any sequence of connected edges yields a seamless motion composed of the corresponding sequence of clips. Every pose that the character can attain is contained within the motion graph, and so we require each constraint pose to correspond with the pose at some node $\mathcal{N}$. Traditional synthesis methods based on motion graphs [AF02, KGP02, LCR*02] are inherently discrete in that they search for a sequence of available clips that meets user-defined criteria. As noted in Section 1, this precludes constraints on continuous properties (such as position, orientation, and duration) from being exactly satisfied, and finding a clip sequence with minimal deviation can require an expensive

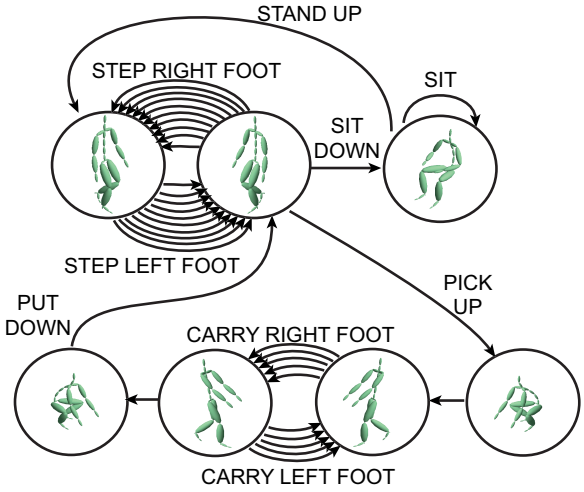

Figure 1: *An example motion graph. Edges are motion clips and nodes indicate frames where clips have the same joint orientations and velocities.*

search. We instead allow the clips from the motion graph to be continuously adjusted to alter a character's position, orientation, and speed. This strategy allows the constraints defined above to be exactly satisfied, and it yields shorter search times since a raw sequence of clips need only sufficiently reduce constraint deviation, rather than minimize it.

To produce a motion that satisfies a sequence of configuration constraints, it is sufficient to consider the problem of constructing segments of motion that start in a specified configuration $(\mathcal{N}_s, \theta_s, \mathbf{p}_s)$ and end in a specified configuration $(\mathcal{N}_e, \theta_e, \mathbf{p}_e)$ within a given time interval $[t^a, t^b]$. The full constraint sequence can then be satisfied by iteratively generating motion that travels from the current configuration to the next configuration. To satisfy the constraints at an iteration, we use a fast approximate planner to construct motions that navigate through the environment and then refine the result to produce a motion that exactly satisfies the constraints. More specifically, the algorithm first constructs two "seed" motions: a *forward* motion $\mathbf{M_f}$ that starts at $(\mathcal{N}_s, \theta_s, \mathbf{p}_s)$ and ends near (but in general not exactly at) $(\mathcal{N}_e, \theta_e, \mathbf{p}_e)$, and a *backward* motion $\mathbf{M_b}$ that ends precisely in $(\mathcal{N}_e, \theta_e, \mathbf{p}_e)$ and starts approximately at $(\mathcal{N}_s, \theta_s, \mathbf{p}_s)$. A randomized search procedure then adjusts these motions such that 1) they can be connected with adjustments to position, orientation, and timing that are below a user-defined threshold and 2) the resulting motion satisfies all constraints. This process is illustrated in Figure 2. The remainder of this section provides details on how $\mathbf{M_f}$ and $\mathbf{M_b}$ are created (Section 3.2) and then how they are adjusted and connected (Section 3.3).

3.2. Creating the Seed Motions

For simplicity, we only discuss the construction of $\mathbf{M_f}$; $\mathbf{M_b}$ is handled identically, except time flows in reverse. As in

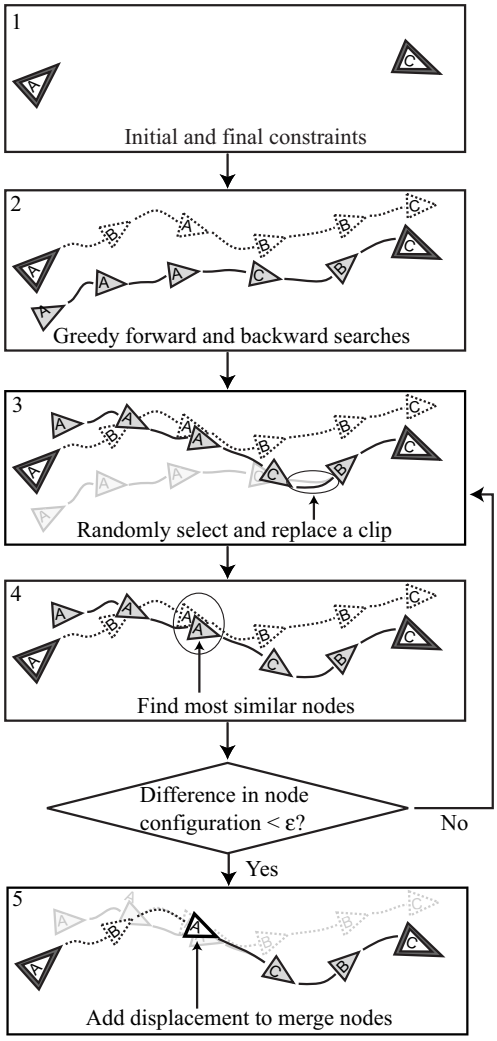

Figure 2: *An individual character's motion is generated by computing motions that satisfy, respectively, constraints on the initial and final configuration (steps 1 and 2) and then iteratively adjusting these motions so they can be seamlessly connected while satisfying all constraints (steps 3 – 5).*

previous work [PLS03], we start by building a probabilistic roadmap (Figure 3) to aid in navigating through the environment. Nodes in the roadmap are created by randomly sampling positions on the ground that are outside of obstacles, and edges are created between nodes that are within a threshold distance of each other and are mutually visible (i.e., the connecting line segment does not intersect any obstacles). Given desired starting and ending configurations $(\mathcal{P}_s, \theta_s, \mathbf{p}_s)$ and $(\mathcal{P}_e, \theta_e, \mathbf{p}_e)$, nodes are added to the road map at $\mathbf{p_s}$ and $\mathbf{p_e}$ and edges are added to all visible neighbors. A shortest path from $\mathbf{p_s}$ to $\mathbf{p_e}$ is then found with Dijkstra's algorithm, resulting in a sequence of way points $\mathbf{w}_1, \ldots, \mathbf{w}_n$ (Figure 3).

Once a sequence of way points is determined, a fast

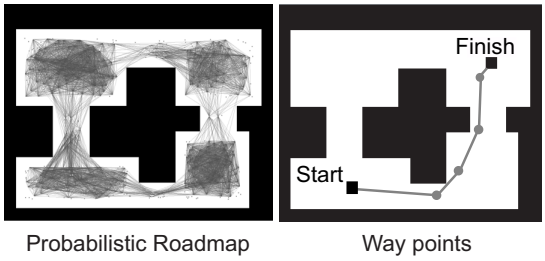

Figure 3: ***Left:*** *An example probabilistic roadmap.* ***Right:*** *A series of way points that connects a pair of positions.*

greedy planner guides the character through successive way points such that it travels from the initial configuration toward the final configuration. This planner continues until a local error measure does not decrease, and hence while it is guaranteed to terminate, in general it will not produce optimal motion. This is acceptable because only a rough initial motion is needed for the randomized search, and a fast approximate planner is preferable to a slower planner that produces optimal (but still necessarily inexact) results.

The planner proceeds as follows. First, assume that the current target way point is $\mathbf{w}_i$, with $i < n$. The planner iteratively selects the edge in the motion graph that brings the character closest to $\mathbf{w}_i$ without incurring collisions with the environment or higher-priority characters. This edge is required to be within the subgraph corresponding to the character's current locomotion state — for example, in Figure 1, a character who has just picked up a box would be required to use one of the "carry" edges. This subgraph is determined from annotations associated with the originally data, which may be added semi-automatically [AFO03] as a preprocess. Collisions are detected by placing minimum bounding cylinders around each character. Because the set of all possible character poses is encoded in the motion graph, these bounding cylinders can be precomputed for greater efficiency. If all possible edges increase the distance to $\mathbf{w}_i$, then $\mathbf{w}_i$ is set to the current way point and the planner attempts instead to reach $\mathbf{w}_{i+1}$.

When travelling to the final way point $\mathbf{w_n} = \mathbf{p_e}$, the planner must also account for possible constraints on target orientation θ_e and ending pose $\mathcal{P}_e$. The latter constraint reduces to requiring the motion to terminate in the node $N_{\mathcal{P}_e}$ that corresponds to $\mathcal{P}_e$. A similar greedy algorithm is used, except 1) the next edge is selected so as to minimize a weighted sum of the position and orientation errors, rather than just position errors, and 2) whenever the current motion terminates at $N_{\mathcal{P}_e}$, it is stored in a temporary variable $\mathbf{M}_{best}$. Whenever the error stops decreasing, the current $\mathbf{M}_{best}$ is returned. If no value for $\mathbf{M}_{best}$ exists (because no motion terminating in $N_{\mathcal{P}_e}$ has been encountered), then instead Dijkstra's algorithm is used to find the path in the motion graph to $N_{\mathcal{P}_e}$ containing the fewest edges. If multiple shortest paths exist, then ties are broken by greedily selecting edges that mini-

mize position/orientation error. Finally, if the orientation is unconstrained, then only position error is considered, and if the final pose is unconstrained, then the planner simply adds edges until the position/orientation error stops decreasing.

Note that the planner does not consider time constraints. These constraints are addressed in the next stage of the synthesis process.

3.3. Adjusting and Merging the Seed Motions

The seed motions $\mathbf{M}_f$ and $\mathbf{M}_b$ satisfy, respectively, the constraints on the character's initial and final configuration. Our goal is to merge them into a single motion that satisfies both of these configuration constraints and, if specified, has a duration within the desired interval $[t^a, t^b]$. We start by finding frames in $\mathbf{M}_f$ and $\mathbf{M}_b$ where the character is in the same pose and where the position and orientation of this pose are similar. Displacement maps are then added so the position and orientation are identical. Finally, the adjusted motions are spliced together at these frames to form a seamless new motion that begins and ends in the desired configurations, and the speed of this motion is altered to conform to the time constraints. While this algorithm guarantees that the constraints are satisfied, the result may look unrealistic if overly large changes are made. Our strategy is to use a randomized search algorithm to perturb $\mathbf{M}_f$ and $\mathbf{M}_b$ such that they are sufficiently similar at a pair frames and of sufficient duration that the necessary adjustments are below a user-controllable tolerance. Figure 2 graphically depicts this process. The remainder of this section explains our algorithm in greater detail.

Let $\mathbf{M}_f$ and $\mathbf{M}_b$ be composed, respectively, of n_f and n_b clips from the motion graph. The i^{th} clip of $\mathbf{M}_f$ is represented by a tuple $\{t_{f,i}, \mathcal{I}_{f,i}, \mathbf{p}_{f,i}, \theta_{f,i}\}$ containing the clip's duration $t_{f,i}$, the index $\mathcal{I}_{f,i}$ of its starting node in the motion graph, and the position $\mathbf{p}_{f,i}$ and orientation $\theta_{f,i}$ of the pose associated with this node. The j^{th} clip of $\mathbf{M}_b$ is represented similarly. We consider joining $\mathbf{M}_f$ and $\mathbf{M}_b$ at any point where they share a node from the motion graph. Let the subsection of a motion $\mathbf{M}$ consisting of the r_1^{th} clip through the r_2^{th} clip be $\mathbf{M}[r_1, r_2]$, and consider the motion formed by concatenating $\mathbf{M}_b[j, n_b]$ onto the end of $\mathbf{M}_f[1, i]$. We define the cost $C_{\mathbf{M}_f,\mathbf{M}_b}(i, j)$ of creating this motion as a sum of position, orientation, and time errors, normalized by the motion's duration:

$$C_{\mathbf{M}_f,\mathbf{M}_b}(i,j) = \frac{1}{N(i,j)}\left(E_{\mathbf{p}}(i,j) + \alpha_1 E_\theta(i,j) + \alpha_2 E_t(i,j)\right), \tag{1}$$

where

- $N(i, j)$ is the total duration of $\mathbf{M}_f[1, i]$ and $\mathbf{M}_b[j, n_b]$, computed as $\sum_{k=1}^{i} t_{f,k} + \sum_{k=j}^{n_b} t_{b,k}$.
- $E_{\mathbf{p}}(i, j)$ is the distance between the end of $\mathbf{M}_f[1, i]$ and the start of $\mathbf{M}_b[j, n_b]$, computed as $\|\mathbf{p}_{f,i+1} - \mathbf{p}_{b,j}\|$.
- $E_\theta(i, j)$ is the orientation difference, computed as $|\theta_{f,i+1} - \theta_{b,j}|$, with numerical values assigned to $\theta_{f,i+1}$ and $\theta_{b,j}$ such that this error is no greater than 180°.
- $E_t(i, j)$ is the time error. If the constraint time interval is $[t^a, t^b]$, then $E_t(i, j) = 0$ if $N(i, j) \in [t^a, t^b]$ and otherwise $E_t(i, j) = \min(|N(i, j) - t^a|, |N(i, j) - t^b|)$.
- α_1 and α_2 are scaling factors to relate the different error measures. In our implementation, $1cm \approx 1^\circ \approx \frac{1}{30}s$.

The cost $C'\left(\mathbf{M}_f, \mathbf{M}_b\right)$ of connecting $\mathbf{M}_f$ and $\mathbf{M}_b$ is defined as the minimum value of $C_{\mathbf{M}_f,\mathbf{M}_b}(i, j)$ over all indices where the terminating node of $\mathbf{M}_f[1, i]$ is the same as the starting node of $\mathbf{M}_b[j, n_b]$.

$$C'\left(\mathbf{M}_f, \mathbf{M}_b\right) = \min_{i,j:\mathcal{I}_{f,i+1}=\mathcal{I}_{b,j}} C_{\mathbf{M}_f,\mathbf{M}_b}(i,j) \tag{2}$$

If a pose constraint exists on the start and/or end of the desired motion (as opposed to, for example, just a position constraint), then $\mathbf{M}_f$ and $\mathbf{M}_b$ will always share at least one node. Otherwise, it is possible for them to share no nodes, in which case $C'\left(\mathbf{M}_f, \mathbf{M}_b\right) = \infty$.

Intuitively, $C'(\mathbf{M}_f, \mathbf{M}_b)$ represents the amount of per-frame adjustment needed to seamlessly splice $\mathbf{M}_f$ and $\mathbf{M}_b$ and satisfy the time constraints. To preserve the realism of the original motion data, we require this to be below a user-defined threshold ε (optionally, one might also limit the total accumulated adjustment, but in our experience it is sufficient to consider the per-frame adjustment $C'(\mathbf{M}_f, \mathbf{M}_b)$; see Section 4). If $C'(\mathbf{M}_f, \mathbf{M}_b) > \varepsilon$, then $\mathbf{M}_f$ and $\mathbf{M}_b$ are perturbed through a randomized search algorithm to produce a new pair of motions with a below-threshold cost. The search proceeds by generating n_m perturbed motion pairs, checking as each pair is generated whether the individual motions are collision-free and whether the splicing cost (Equation 2) is below ε. If so, the search returns this motion pair. Otherwise, the k_m lowest-cost pairs are retained and the search continues. If the search fails to find a below-threshold motion pair after a user-defined maximum number of iterations N_{max}, then a warning is issued and the current lowest cost pair is returned.

During the search, perturbed motion pairs are generated as follows. First, one of the k_m motion pairs from the previous search iteration is selected at random, unless it is the first iteration, in which case only the original pair $(\mathbf{M}_f, \mathbf{M}_b)$ is available. Next, one of the two motions in this pair is selected. A clip $\mathbf{C}$ is chosen at random from this motion and replaced with a new clip $\mathbf{C}'$. In order to ensure that $\mathbf{C}'$ joins seamlessly with the rest of the motion, it is required to originate and terminate at the same nodes in the motion graph as $\mathbf{C}$ (this also means that the perturbed motion begins and ends in the same poses, so pose constraints automatically remain satisfied). Because the motion graph is constructed so as to have many more edges than nodes, in general there will be many possible replacement clips. Let the change in the character's position and orientation in $\mathbf{C}$ be $\delta\mathbf{p}_\mathbf{C}$ and $\delta\theta_\mathbf{C}$, and let $\mathbf{C}$'s duration be $\delta t_\mathbf{C}$. The actual replacement clip $\mathbf{C}'$ is

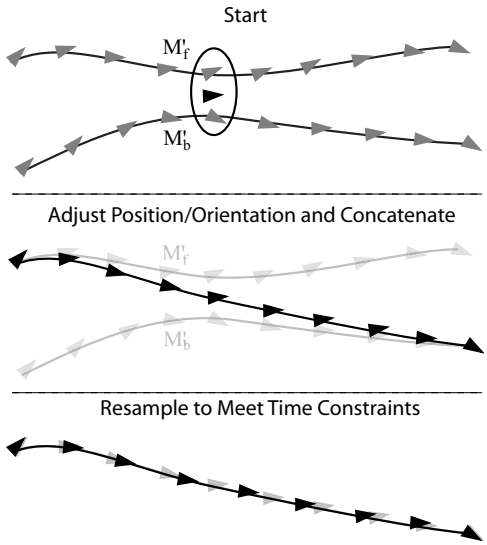

Figure 4: $\mathbf{M}'_f$ *and* $\mathbf{M}'_b$ *are joined at the nodes calculated in Equation 2 by adding displacements that compensate for differences in position and orientation, and the result is then resampled to meet the time constraints.*

chosen with a probability inversely proportional to its "distance" from $\mathbf{C}$, in terms of the relative change in position, orientation, and time:

$$\|\Delta\mathbf{p}_{\mathbf{C}} - \Delta\mathbf{p}_{\mathbf{C}'}\| + \alpha_1|\Delta\theta_{\mathbf{C}} - \Delta\theta_{\mathbf{C}'}| + \alpha_2|\Delta t_{\mathbf{C}} - \Delta t_{\mathbf{C}'}|$$

Once suitably perturbed forward and backward motions $\mathbf{M}'_f$ and $\mathbf{M}'_b$ are found, they are joined at the nodes calculated in Equation 2; see Figure 4. Let $\mathbf{M}'_f[1,i]$ and $\mathbf{M}'_b[j,n'_b]$ be the motion segments that will be joined, and let the difference in the final position and orientation of $\mathbf{M}'_f[1,i]$ and the initial position and orientation of $\mathbf{M}'_b[j,n'_b]$ be, respectively, $\delta\mathbf{p}$ and $\delta\theta$. Also, let N_f be the duration of $\mathbf{M}'_f[1,i]$, N_b be the duration of $\mathbf{M}'_b[j,n'_b]$, and δt be the smallest amount that must be added to $N_f + N_b$ such that $(N_f + N_b + \delta t)$ is inside the constraint time interval $[t^a, t^b]$. Lastly, define $\lambda_1 = \frac{N_f}{N_f+N_b}$ and $\lambda_2 = \frac{N_b}{N_f+N_b}$. The final motion is formed as follows:

1. The k^{th} frame of $\mathbf{M}'_f[1,i]$ has its position and orientation adjusted by $\frac{k-1}{N_f-1}\lambda_1\delta\mathbf{p}$ and $\frac{k-1}{N_f-1}\lambda_1\delta\theta$.
2. The k^{th} frame of $\mathbf{M}'_b[j,n'_b]$ has its position and orientation adjusted by $-\frac{k-1}{N_b-1}\lambda_2\delta\mathbf{p}$ and $-\frac{k-1}{N_b-1}\lambda_2\delta\theta$.
3. The adjusted $\mathbf{M}'_f[1,i]$ and $\mathbf{M}'_b[j,n'_b]$ are concatenated.
4. The result is resampled so its duration is $(N_f + N_b + \delta t)$.

By construction, the result is continuous and satisfies all constraints. A final test for collisions is made on this new motion and, if the test fails, then the motion is discarded and the search algorithm continues from where it left off.

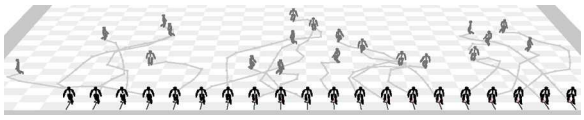

Figure 5: ***Experiment 1:*** *20 characters (grey) are required to arrive simultaneously in specified configurations (black).*

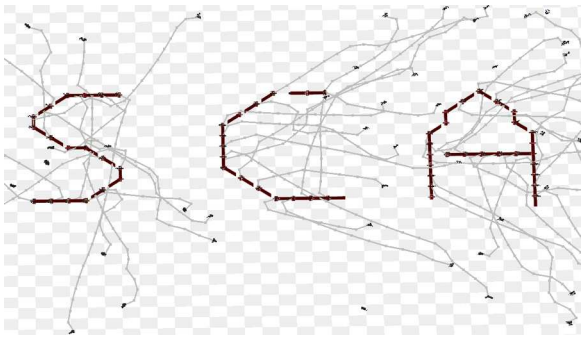

Figure 6: ***Experiment 2:*** *70 characters are required to arrive at a set of positions in which they spell out SCA.*

4. Results

We tested our algorithm using the motion graph shown in Figure 1, which allowed characters to walk in various directions; sit down into and get up from a chair; and pick up a box, carry it in various directions, and put it down. Altogether, the motion graph contained 50s of motion data sampled at 30Hz. In our experiments, we set $\varepsilon = 2cm/s = 2^\circ/s = \frac{1}{15}s/s$, and the parameters for the search algorithm (see Section 3.3) were $n_m = 200$, $k_m = 30$, and $N_{max} = 1000$; for these values the search algorithm was always able to find a motion pair that was within the adjustment tolerance and avoided collisions. We generated several animations that required a large number of characters to perform certain tasks at specified positions, locations, and times; see Figures 5 through 9 and Table 1 for a summary of results.

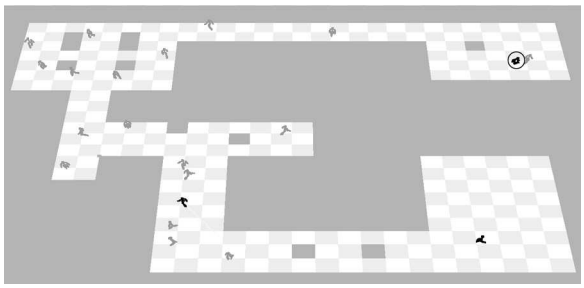

Figure 7: ***Experiment 3:*** *The character in black must start at the lower right, navigate through a set of rooms filled with obstacles and other characters, and lift a box in the room at the upper right. The location of the box is circled.*

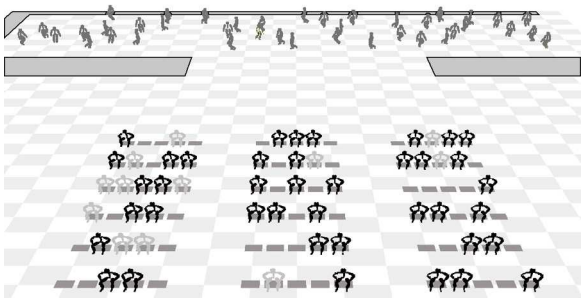

Figure 8: ***Experiment 4:*** *40 characters start in a theater lobby (top) and take their seats (bottom). Dark sitting characters represent target poses, and light sitting characters are agents that are already sitting down (i.e., obstacles).*

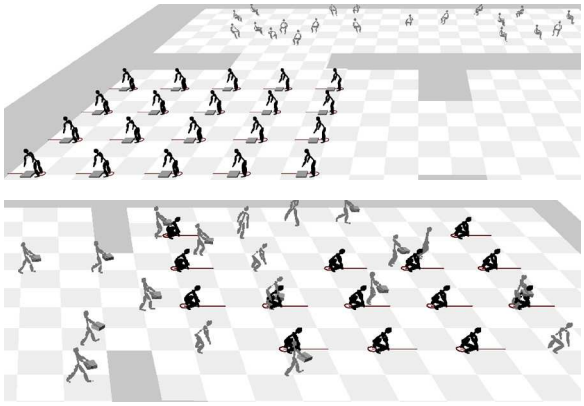

Figure 9: ***Experiment 5:*** *20 characters, initially sitting, must each pick up a designated box in the second room, place it in a designated position, and then return to their original seat. The top image shows initial positions (grey) and constraints poses (black). The lower image shows characters moving towards placement goals, with constraint poses again shown in black.*

In order to test our method on varying numbers of characters and densities of characters, for Experiment 6 we created a contrived scenario where a number of characters are placed on a uniform grid and are given target positions on a translation of this grid. The target for each character is chosen such that each character's path is of approximately the same length, but that the characters must interact in order to meet their goals, as illustrated in Figure 10. This experiment allowed us to test examples with large numbers of characters. When the spacing between characters is sufficiently large that collision avoidance does not unduly restrict movement, our algorithm can synthesize the motions for hundreds of characters faster than real time.

The speed of our algorithm depends upon the complexity of the scenario that is to be animated. In particular, higher-density groups of characters (or, equivalently, more cluttered environments) require additional time in order to avoid collision, both because collision detection is more time consuming and because the search will typically explore a greater number of paths before finding one that avoids collisions. To illustrate this, for Experiment 6 we created a series of scenarios wherein motion was synthesized for a group of 300 characters with different inter-characters spacings. Figure 11 shows the average amount of time needed to plan the motion for an individual character as a function of the grid spacing. At smaller grid spacings, the average synthesis time increases because more paths are rejected during the random search due to collisions.

Example	# Agents	Duration	Synthesis Time	
			Average	Total
1	20	14.8s	0.21s	4.2s
2	70	21.3s	0.18s	12.6s
3	1	83.3s	0.01s	0.01s
4	40	30.2s	0.35s	14.0s
5	20	23.6s	0.15s	3.0s
6a	300	25.3s	0.034s	0.86s
6b	500	25.6s	0.035s	0.90s

Table 1: *Information relating to the animations shown from Figure 5 to Figure 9. The second column states the total number of characters that were animated, the third column states the average duration of each character's motion, the fourth column states the average amount of time needed to synthesize the motion for a character, and the final column states the total amount of time needed to synthesize all motion for all characters. All experiments were performed on a PC with a 3.0Ghz processor and 1GB main memory. For Experiment 6 (described below) we report results for representative trials with (6a) 300 characters and (6b) 500 characters.*

Different values of ε provide different tradeoffs between motion quality and synthesis speed — larger values permit greater deviation from the raw graph-generated motions, but also allow the search to terminate more quickly because a larger range of motion pairs will be able to satisfy the constraints. Figure 12 shows the time needed to synthesize the motion for all 70 characters in scenario 2 at different values of ε; the shape of this graph was similar in the other scenarios. At very small tolerances, relatively few motions can be made to satisfy the constraints, but at higher tolerances a wider variety exists and the search hence can terminate sooner. However, eventually the speed benefit of increasing the tolerance decreases, because sufficient flexibility already exists to quickly find motions that satisfy the constraints,

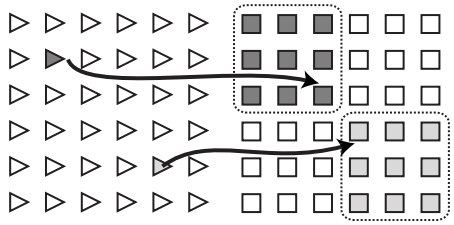

Figure 10: *The scenario tested in Experiment 6. Characters begin in grid formation (left) and are each assigned a target on a translated version of this grid (right). Targets are chosen to keep each character's path approximately the same length. Specifically, if a character begins at position* (i, j) *on the original grid, then it will be assigned a randomly perturbed position on the target grid. For example, the character indicated by the dark grey triangle is assigned one of the target positions indicated by a dark grey square.*

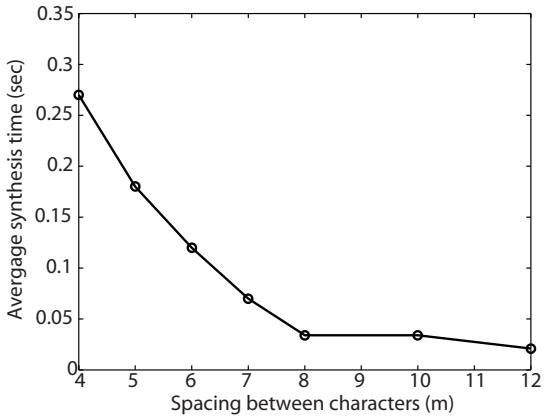

Figure 11: *The average time per character needed to generate motion for* 300 *characters in the scenario of Experiment 6, as a function of the spacing between characters.*

and the synthesis time becomes dominated by other concerns like collision detection.

We limit the amount of per-frame adjustment to ε based on the intuition that longer motions can tolerate larger overall adjustments if they are worked in gradually. However, if the total accumulated adjustment exceeds a certain amount, this approach breaks down. For example, a 180° change in orientation will yield unrealistic results now matter how gradually it is introduced, because eventually the character will face opposite the direction of travel. This can trivially be avoided by placing a second limit ε' on the total allowable adjustment. However, in our experiments we found this to be unnecessary, because the amount that motions needed to be altered to meet the constraints was effectively independent of the duration of the motions.

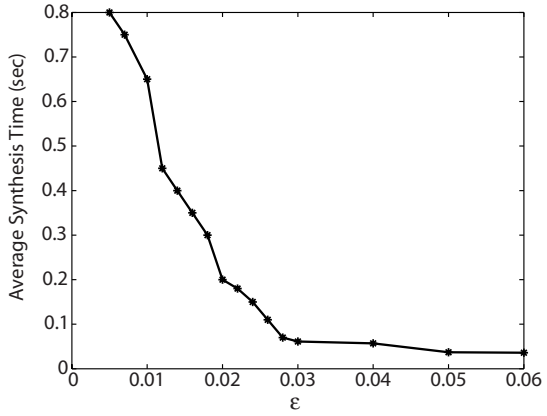

Figure 12: *The average time per character needed to generate motion for* 70 *characters in a typical scenario, as a function of the allowable amount of motion adjustment* ε*.*

5. Discussion

This paper has presented an efficient algorithm for generating realistic goal-directed motion for large numbers of characters. Our algorithm creates collision-free motion that precisely satisfies constraints on pose, position, orientation, and duration. A fast PRM-based path planner is used to construct rough motions that navigate through complex environments, and this is then refined through a randomized search over a motion graph that explicitly incorporates continual, gradual adjustments to a character's position, orientation, and speed. The flexibility provided by these adjustments allows us both to precisely satisfy constraints and to reduce the time needed to construct desired motions.

We have focused on the problem of character navigation, where the main technical challenge is to guide a character through an environment such that it ends up in a particular configuration. For stationary tasks, such as manipulating objects or gesticulating during conversation, we assume that a corresponding clip or known sequence of clips already exists. Also, to be able to reliably satisfy constraints without destroying motion quality, our algorithm relies on constraints being sufficiently sparse that the connecting motions can be meaningfully adjusted. For example, if the character were constrained to be in a specific pose every second, then it is likely that satisfying all of these constraints would require a very large value for ε, which would in turn probably result in unrealistic motion.

Our motion graph search algorithm does not generate optimal motion in the sense of minimizing deviation from the constraints. Instead, it finds motions that can be made to exactly satisfy constraints by being adjusted within a user-specified tolerance, and all motions within this tolerance are considered equally suitable. In principle, this involves trading off motion quality for computational speed, but in prac-

tice even a modest tolerance can significantly reduce searching times without appreciably altering motion quality. Our search algorithm also assumes that the constraints fully specify what a motion should do, and that any motion that satisfies the constraints is acceptable. This can sometimes lead to artifacts where characters, despite having smooth motion, exhibit unusual behavior. For example, a character might not take the most direct path towards a route, and indeed will intentionally wander if the time constraints require a lengthy motion. This is a common limitation that is shared by existing graph-based synthesis algorithms which rely on constrained minimization [AF02, KGP02, LCR*02] — if the objective function and constraints does not completely describe the desired properties of a motion, then undesirable behavior is inevitable. For scenes with many characters, however, we have found this to be less of a problem because a viewer's attention typically is not limited to a single individual, and hence longer-term behavioral oddities are less noticeable.

The constraints used by our synthesis algorithm are produced by an external process, and it is the responsibility of this process to ensure that these constraints are achievable. For example, it should not require two characters to be at the exact same place at the same instant of time.

The methods of this paper are offline and hence not directly applicable to applications like games. However, for more interactive assessment of the results the search can be broken into stages, first generating motion that satisfies the initial configuration/time constraint of each character, then the next constraint for each character, and so on. A more general extension to online applications is left for future work. Also, a straightforward extension of this work is to employ smooth, continual motion adjustments for collision avoidance, in addition to constraint satisfaction.

Acknowledgements

This work was made possible through motion data donations from House of Moves and financial support from NSF grants CCR-9984506 and CCR-0204372. We thank Hyun Joon Shin for assistance in creating the motion graphs.

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.

[BC89] BRUDERLIN A., CALVERT T.: Goal-directed, dynamic animation of human walking. In *Proc. of ACM SIGGRAPH 1989* (July 1989), Annual Conference Series, ACM SIGGRAPH, pp. 233–242.

[BC97] BOUVIER E., COHEN E.: From crowd simulation to airbag deployment:particle system, a new paradigm of simulation. *Journal of Electronic imaging* (1997), 94–107.

[BMTT90] BOULIC R., MAGNENAT-THALMANN N., THALMANN D.: A global walking model with real-time kinematic personification. *Visual Computer 6*, 6 (1990), 344–358.

[BUT04] BOULIC R., ULICNY B., THALMANN D.: Versatile walk engine. *Journal of Game Development 1*, 1 (2004).

[BW95] BRUDERLIN A., WILLIAMS L.: Motion signal processing. In *Proc. of ACM SIGGRAPH 95* (Aug. 1995), Annual Conference Series, ACM SIGGRAPH, pp. 97–104.

[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.

[FBT99] FARENC N., BOULIC R., THALMAN D.: An informed environment dedicated to the simulaion of virtual humans in urban context. In *Proc. of EUROGRAPHICS 1999* (1999), Annual Conference Series, pp. 309–318.

[GR96] GUO S., ROBERGE J.: A high-level control mechanism for human locomotion based on parametric frame space interpolation. In *Proc. of Eurographics Workshop on Computer Animation and Simulation '96* (Aug. 1996), pp. 95–107.

[GSKJ03] GLEICHER M., SHIN H. J., KOVAR L., JEPSEN A.: Snap-together motion: assembling run-time animations. In *Proc. of the 2003 Symposium on Interactive 3D graphics* (2003), pp. 181–188.

[HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating dynamics feature of escape panic. In *Nature* (2000), pp. 487–490.

[HGP04] HSU E., GENTRY S., POPOVIĆ J.: Example-based control of human motion. In *Proc.of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2004), pp. 69–77.

[HM95] HELBING D., MOLNAR P.: Social force model for pedestrian dynamics. In *Physical Review* (May 1995), pp. 4282–4286.

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

[KL94] KAVRAKI L., LATOMBE J. C.: Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. on Robotics and Automation* (1994), pp. 2138–2145.

[KL00] KUFFNER J., LAVALLE S.: Rrt-connect: An efficient approach to single-query path planning. In *Proc.*

IEEE Int. Conf. on Robotics and Automation (2000), pp. 995–1001.

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

[KSG02] KOVAR L., SCHREINER J., GLEICHER M.: Footskate cleanup for motion capture editing. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2002), pp. 97–104.

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

[LK99] LAVALLE S., KU J.: Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. on Robotics and Automation* (1999), pp. 473–479.

[LK00] LAVALLE S., KUFFNER J.: Rapidly-exploring random trees: Progress and prospects. In *In Workshop on the Algorithmic Foundations of Robotics.* (2000), pp. 293–308.

[LL04] LEE J., LEE K. H.: Precomputing avatar behavior from human motion data. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2004), pp. 79–87.

[MT01] MUSSE S. R., THALMANN D.: Hierarchical model for real time simulation of virtaul human crowds. In *IEEE Transaction on Visualization and Computer Graphics* (2001), pp. 152–164.

[PLS03] PETTRÉ J., LAUMOND J.-P., SIMÉON T.: A 2-stages locomotion planner for digital actors. In *SCA '03: Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), pp. 258–264.

[PSS02] PARK S. I., SHIN H. J., SHIN S. Y.: On-line locomotion generation based on motion blending. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2002), pp. 105–111.

[PSS04] PARK S. I., SHIN H. J., SHIN S. Y.: On-line motion blending for real-time locomotion generation. In *Computer Animation and Virtual Worlds 15* (2004), pp. 125–138.

[RCB98] ROSE C., COHEN M., BODENHEIMER B.: Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Application 18*, 5 (1998), 32–40.

[RP04] REITSMA P. S. A., POLLARD N. S.: Evaluating motion graphs for character navigation. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2004), pp. 89–98.

[SCG04] SUNG M., CHENNEY S., GLEICHER M.: Scalable behaviors for crowd simulation. In *Computer Graphics Forum (EUROGRAPHICS '04)* (2004), vol. 23, pp. 519–528.

[SE02] SCHÖDL A., ESSA I.: Controlled animation of video sprites. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2002), pp. 121–127.

[SM01] SUN H. C., METAXAS D.: Automating gait generation. In *Proc. of ACM SIGGRAPH 2001* (July 2001), Annual Conference Series, ACM SIGGRAPH, pp. 261–270.

[UT02] ULICNY B., THALMANN D.: Towards interactive real-time crowd behavior simulation. *Computer Graphics Forum* (Nov. 2002), 767–775.

[WP95] WITKIN A., POPOVIĆ Z.: Motion warping. In *Proc. of ACM SIGGRAPH 95* (Aug. 1995), Annual Conference Series, ACM SIGGRAPH, pp. 105–108.