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

Group Motion Graphs

Yu-Chi Lai†, Stephen Chenney and ShaoHua Fan

University of Wisconsin, Madison

Abstract

We introduce Group Motion Graphs, a data-driven animation technique for groups of discrete agents, such as flocks, herds, or small crowds. Group Motion Graphs are conceptually similar to motion graphs constructed from motion-capture data, but have some important differences: we assume simulated motion; transition nodes are found by clustering group configurations from the input simulations; and clips to join transitions are explicitly constructed via constrained simulation. Graphs built this way offer known bounds on the trajectories that they generate, making it easier to search for particular output motions. The resulting animations show realistic motion at significantly reduced computational cost compared to simulation, and improved control.

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

1. Introduction

Rule-based agent techniques are commonly used to animate groups of virtual creatures in both real-time environments and off-line production. Examples range from the flocking models of Reynolds [Rey87], to commercial systems like MASSIVE [Dun02] and AI.implant [AI.03], to any number of crowd animation systems. Agent models must be highly efficient for use in computer games and interactive systems, particularly when acting as secondary animation to add realism to an environment. Furthermore, agent models should offer two forms of control: over what the group looks like and what the group does. For instance, an appearance goal might be resemblance to a particular animal herd, while an action goal might be following a particular path through the environment. In this paper we present Group Motion Graphs, a data-driven agent animation technique that addresses efficiency and both forms of control.

Data-driven methods record motion in a pre-production stage, and then play back the data to drive run-time motion (the most common example is human motion capture). Group Motion Graphs (GMGs) record the motion of an agent group as a whole, including the configuration of agents within the group. As with human motion graphs [KGP02, AF02, LCR*02], the data clips are stored in a graph structure that encodes which clips can be appended while retaining realistic, continuous motion. We describe techniques for constructing clips from agent-based simulations that can be pieced together in a dense graph structure. We demonstrate several uses of the resulting graph, including path following and random motion restricted to a region.

There are three principle advantages to GMGs: efficiency, style control, and constraint satisfaction. The run-time CPU cost of unconstrained motion generation from GMGs is essentially the time taken to set animation state from the clips. In comparison, a rule-based group simulation requires some mechanism for tracking relationships between agents and evaluating rules, which leads to super-linear cost with large constants. Motion clips encode a particular style, or an implicit set of constraints on the appearance of the motion. This style is maintained by the playback scheme, so a designer can be certain that motion generated from their clips will retain their style. Finally, once a graph of clips has been built it can be searched with standard techniques to produce constrained trajectories. This is cheaper than searching within a continuous simulation state space.

Our primary contribution is a method for building motion graphs for groups of discrete agents. This includes solutions to the problems of finding good transition configurations, a

† yu-chi@cs.wisc.edu

novel graph structure that gives guarantees on the synthesized motion, and techniques for computing transition probabilities on the graph. We also describe a new technique for forcing a flocking simulation to a specific group configuration while moving along a specific path.

GMGs are suitable for applications where the group moves through the environment as a cohesive unit, and individual agents do not interact with objects external to the group. We thus see the primary application as simulations that add realistic but previously expensive elements to large virtual environments. For example, outdoor game environments could cheaply add a roaming herd or circling flock, without incurring the cost of a large-scale agent simulation. Another application is constrained animation in which a group must follow a specific trajectory or meet other constraints.

2. Overview

A motion graph is a directed graph in which edges correspond to pre-recorded animation clips, and nodes represent places where clips can be joined. Animation generated from the graph can be thought of as a point that follows an edge and makes a choice at transitions as to which edge to follow next. Construction of a motion graph requires identifying transition points and the creation of clips to achieve seamless transitions. Others have defined motion graphs as the dual of ours, with clips at nodes and edges as transitions; the representations are functionally the same.

All of the graphs in this paper are based on the flocking model introduced by Reynolds [Rey87]. When simulating to generate motion, each agent at each time-step adjusts its trajectory by evaluating rules based on its own state and that of its neighbors. The output state of the system is the position and velocity of each agent at each timestep — this is what we store in group motion clips. At a high level, the techniques we describe could be applied to other group animation systems, even motion-captured groups. However, effective GMGs require group configurations that naturally repeat themselves often or the ability to force such repetition, properties that may not be available in all group models. Drawing extreme examples from sports, volleyball would be a good candidate for group motion graphs because it has a repeated starting configuration, while football (soccer) would pose greater difficulties.

A good motion graph has many options for transitions from one clip to another. This increases variety in the resulting motion. Equally important, the clips should sample the space of possible motions well [RP04] so that control algorithms have the flexibility to meet a wide range of goals. The clips should be short to enable frequent control choices, but not so short that the frequent transitions create artifacts. Short clips, well distributed over the range of motion, also save memory because variety can be obtained by rich combinations of clips, rather than individually complex clips.

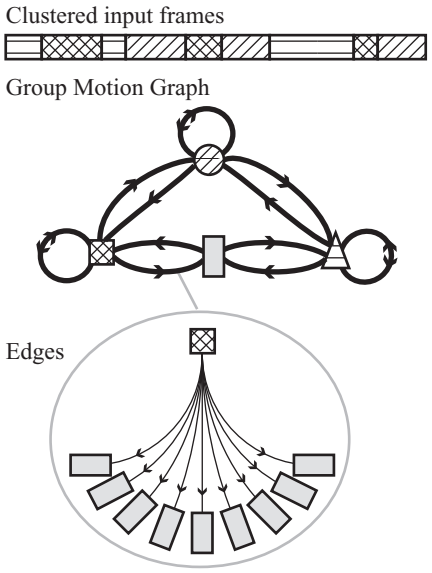

Figure 1: *The first step in building a GMG is clustering input configurations (top). The representative configurations from each cluster are used as transition nodes in a dense graph, where each edge represents a family of clips that follow a specific circular trajectory. Link nodes (the rectangular node) are inserted to break edges that are too long.*

GMGs use a novel construction method that produces graphs with a controllable density and transition frequency. Our construction method also assigns probabilities to transitions; these are used to improve the realism of synthesis. The stages in constructing a GMG are:

- Find a small set of *cluster configurations* that are representative of all the configurations seen in the input. Link them in a complete graph, with several edges of varying curvature between each configuration (Figure 1 and Section 4). Explicitly creating edges in this way gives guarantees on the trajectories that the graph can produce — in our case piecewise circular.
- Determine the required duration of the motion clips corresponding to each edge in the graph, based on how much the arrangement of agents must change while traveling along the edge. If the clips are too long, insert additional *link nodes* to place an upper bound on the time between each transition (Section 4.3).
- Gather *statistics* on the transitions between the nodes along each graph edge. This allows us to synthesize motion with statistical properties similar to the input.
- Use *constrained simulation* to construct the clips for each edge, as described in Section 5. New rules are added to the flocking behaviors to force the group along a specific path and into a specific arrangement of agents.

A variety of graph search algorithms can be applied to motion graphs to synthesize new motion with particular

properties, as we describe in Section 7. We demonstrate random walk on the graph guided by transition probabilities, which produces unconstrained motion that is very similar in style to the input motion. When used for secondary motion, it may be desirable to keep the group within the extents of the world. We describe a random walk with look-ahead that restricts transitions to keep the group within a region. Finally, the group may be required to follow a specific path or hit certain way-points. We give greedy and A^* search algorithms that support such constraints. Because GMGs are constructed to densely sample trajectories, we can be sure of following any path within curvature limitations.

3. Related Work

Motion capture [Men99] is the prominent application of data-driven animation. The formalism of connecting clips into a graph structure was independently developed by Arikan and Forsyth [AF02], Lee et al. [LCR*02], and Kovar et al. [KGP02]. Each group differed in how they matched frames and generated motion from the graph, but all found matches between poses in different frames and inserted transitions into the graph at these points. To improve the responsiveness and predictability of motion synthesis, Gleicher et al. [GSKJ03], constructed graphs with only a few transition nodes but many links; in an interactive application any motion is reachable from any other in only a very short period of time. GMGs use a similar idea to ensure that any trajectory can be generated from the graph. Lee and Lee [LL04] obtained interactive control using reinforcement learning to pre-compute the optimal action given a target state. While successful at planning for short-term goals, such as a boxing punch, their approach does not extend well to situations with a broad range of possible targets.

Other applications of data-driven synthesis for animation range from synthetic motion capture of fish body motion [YT99], to capturing the response of grass to a wind field [PC01]. In the former case, the state-space was the pose of the fish, and parameters for a periodic motion model were extracted from the simulation to speed up the run-time simulation. In the latter case, the bending of grass for varying wind speeds was pre-computed. Data-driven techniques have also been used to model the impulse response of dynamic systems such as cloth and plant models [JF03], however the size of the state space severely limited the possible impulses that could be applied. Notably, this method also pre-computed rendering parameters to provide interactive global illumination. None of these prior systems deal with the coordinated motion of groups.

GMGs provide a means of controlling the trajectory of a group as a whole. Previous techniques for guiding flocks include Reynolds' steering behaviors [Rey99] and the roadmap techniques of Bayazit et al. [BLA02]. While these techniques are sufficient for guiding a flock along some general path, the interactions between rules typically require compromise between constraint satisfaction and the style of the motion. The degree of control we offer encompasses these previous methods and adds additional tools. Anderson et al. [AMC03] describe an algorithm for global control of a flock that can meet hard constraints, but the method is not suitable for on-line control. GMGs simplify constrained animation by reducing the problem to one over a discrete search space (walks on the graph). A similar approach was taken by Go et al. [GVK04] for controlling single vehicles, but they did not work with an explicit graph structure.

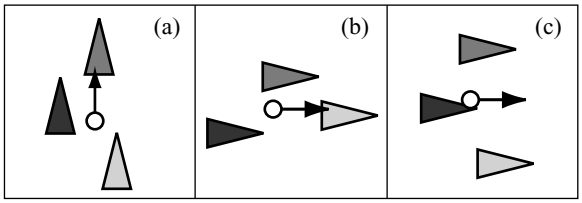

Figure 2: *Configurations (a) and (b) are similar, because agents are arranged the same way with respect to the average velocity, despite that velocity being different in world coordinates and despite some agents switching locations. Group (c) is not similar because the arrangement is different with respect to the group's velocity, despite being the same as (a) in world coordinates. Also shown is the origin and principle axis of the* configuration space *coordinate system attached to each group.*

We use Markov chain statistical techniques to control motion synthesis from GMGs. Brand and Hertzmann [BH00], Li et al. [LWS02] and Grochow et al. [GMHP04] are just some recent examples of the application of Markov models to human motion synthesis. Our techniques are simpler because we are only concerned with the sequence of path segments given the group's configuration (the relationship between pose and trajectory for human motion), not the relationship between the group's members (human pose itself).

4. GMG Construction

The following sections describe the steps in building a GMG, beginning with a discussion of what it means for two group configurations to be similar.

4.1. Group Configurations

We make two assumptions about the group motion to maximize the self-similarity of groups within a cluster (Figure 2). First, we assume that the group's configuration depends on the direction of travel, but not how this direction of travel is embedded in the world (a typical assumption for motion graphs). Second, we assume that all the agents are evaluating the same set of rules, and hence can fulfill any role within the group. For comparing two configurations, C_X and C_Y, this means that for every agent in C_X there must be some agent near its location in C_Y, but not necessarily the *same* agent.

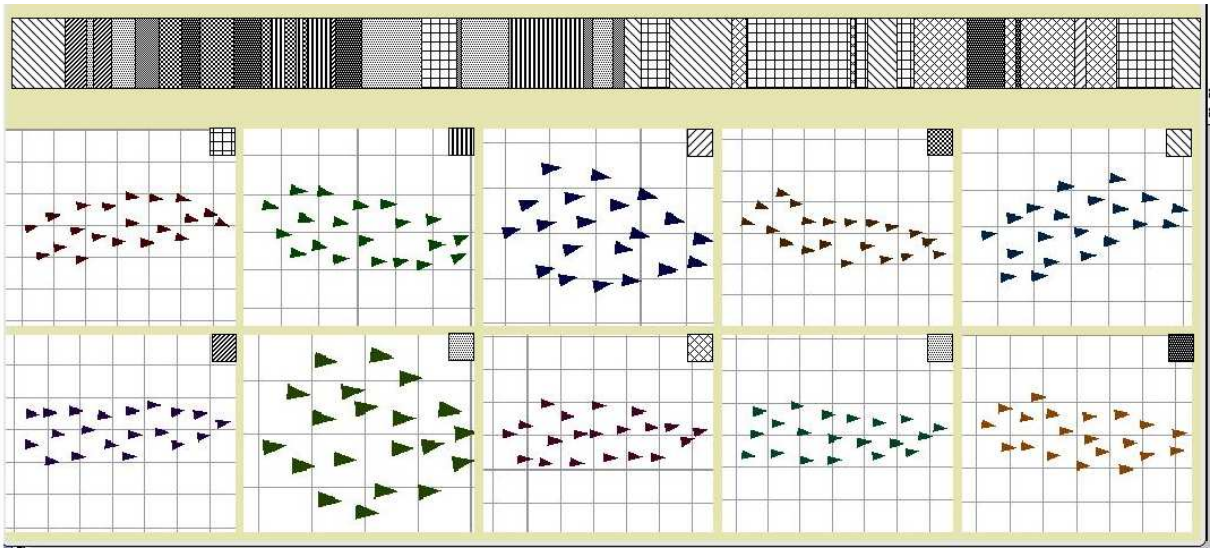

Figure 3: *The strip at the top represents the frames of a flocking simulation classified into configuration clusters, each indicated with a different hatch pattern. The corresponding set of cluster configurations is shown below.*

To precisely describe a configuration, we define a local, moving *configuration space* coordinate system (Figure 2). We use this coordinate system at various stages of the construction algorithm to provide a common reference frame between groups. Assume the group consists of N agents, each with world position $\mathbf{x}^{(i)}$ and velocity vector $\mathbf{v}^{(i)}$. At any instant, t, the origin of configuration space is the center of mass (COM) of the agents, $\mathbf{O}_c(t)$, and the x axis, $\mathbf{X}_c(t)$, is aligned with the average agent velocity, $\dot{\mathbf{O}}_c(t)$:

$$\mathbf{O}_c(t) = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}^{(i)}(t)$$

$$\dot{\mathbf{O}}_c(t) = \frac{1}{N}\sum_{i=1}^{N}\dot{\mathbf{x}}^{(i)}(t)$$

$$\mathbf{X}_c(t) = \frac{\dot{\mathbf{O}}_c(t)}{\|\dot{\mathbf{O}}_c(t)\|}$$

$\mathbf{O}_c$ and $\mathbf{X}_c$ are sufficient for a 2D coordinate system, while in 3D we require another axis to define roll about $\mathbf{X}_c$: $\mathbf{Y}_c = \widehat{\mathbf{up} \times \mathbf{X}_c}$ where **up** is an arbitrary world up direction. We refer to the transformation from world to configuration coordinates as at time t as $\mathcal{X}_{c \leftarrow w}(t)$.

The assumptions on group motion could be removed if the group behavior made them invalid (for instance, there was a designated leader). Note that removing the identical behaviors assumption makes construction simpler because we could use metrics that measured the difference between individual agents, rather than the metric we use that assumes no correspondences between agents. Also observe that we could handle subsets of agents with the same behaviors by using our metric within each subset. Working in world rather than configuration coordinates would require that velocity be considered when comparing agents.

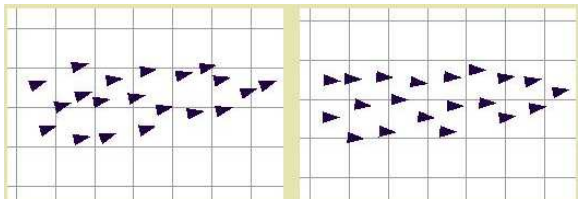

Figure 4: *Left is an example input frame and right is its corresponding cluster configuration.*

4.2. Configuration Clustering

The problem of finding representative configurations is one of clustering: partition all of the configurations in the input into subsets such that elements within each subset are similar. The most representative configuration for each cluster is then used as a cluster configuration node in the graph, as we can safely say that similar configurations are common in the input. Figure 3 shows the result of clustering on a sample segment of input.

We cluster using the standard k-means algorithm [FP03]. To begin, a set of k configurations are chosen uniformly at random from within the input sequence. These will be the representative configurations for each cluster. Each configuration in the input is then classified into the particular cluster to which it is closest, using the distance metric defined below. The algorithm iterates, updating the representative configuration to better reflect the cluster and reclassifying configurations based on the new set of representatives.

We choose the Hausdorff distance between the two point sets as our similarity metric, because it is fast to evaluate and works with our assumption that all agents are functionally

the same:

$$E_h(t_X, t_Y) = \max\{ \max_i \left[\min_j d(\mathbf{x}_c^{(i)}(t_X), \mathbf{x}_c^{(j)}(t_Y)) \right] \max_j \left[\min_i d(\mathbf{x}_c^{(i)}(t_X), \mathbf{x}_c^{(j)}(t_Y)) \right] \}$$

where t_X is the time corresponding to arrangement C_X, t_Y is the time corresponding to configuration C_Y, $\mathbf{x}_c^{(i)}(t)$ is the position in configuration space of agent i at time t, and $d(\mathbf{x}_c^{(i)}(t_X), \mathbf{x}_c^{(j)}(t_Y))$ is the Euclidean distance. The Hausdorff distance penalizes point sets with many points in X that are not close to points in Y, or vice versa, but it does not require a one-to-one correspondence between agents. Figure 4 shows the closest cluster configuration for an example input configuration.

An update scheme must be defined for the representative configuration of each cluster. We do not require this to be one of the configurations from the input — it is an abstract configuration used to represent the cluster's shape that we will later map back onto a configuration from the input. To update a representative, for each agent in it we find the closest agent in each configuration in the cluster. The average position of these agents is then the new representative agent's position.

The clustering algorithm terminates after a fixed number of iterations, 50 in our experiments. For each cluster representative we use the Hausdorff distance metric to find its closest input configuration and use this as the cluster configuration (Figure 3).

4.3. Joining Configurations

Once cluster configurations have been found, between every pair of configurations we construct a family of new motion clips. Explicitly constructing the paths in this way precisely defines the trajectories that can be generated from the graph: they are piece-wise circular with an upper bound on the length of each piece and an known upper bound on curvature.

In 2D, each clip in the family joins the start and end configurations along a circular arc, with one clip for each subtended angle, θ, between -90° and 90° in 15° increments (Figure 1). The start (end) configuration's velocity is aligned with the tangent at the start (end) of the clip. In 3D we have a choice for the plane of curvature: we take the 2D set of edges and replicate them in 15° increments about the direction of travel to produce a 3D fan of paths, parameterized by the angle ϕ. Each clip in the family is of near-equal arc length. We also create a family of self-loops for each configuration.

The arc length for a family of edges is computed based on the difference between the cluster configurations that it joins, as this is an indicator of how much change must take place in the agent arrangement. Working in configuration space, we use the bipartite matching procedure from Section 5.1 to find correspondences between the agents in the node configurations, and assume that each agent in the start configuration must move to its corresponding position in the end configuration while the center of mass of the flock roughly follows the edge.

We place an upper bound, Δv, on the speed of an agent in configuration space (the speed relative to the other agents), typically some fraction of the group's average velocity. The higher the fraction, the faster the group mixes. We can then define the minimum time and arc length required to rearrange the group:

$$T_{XY} = \max\left(T_{min}, \max_i \frac{\|\mathbf{x}_c^{(i)}(t_X) - \mathbf{x}_c^{(\mathcal{M}^{Y\leftarrow X}(i))}(t_Y)\|}{(1-\varepsilon)\Delta v} \right) \quad (1)$$

$$l_{XY} = T_{XY}\mathbf{v}_{target} \quad (2)$$

where T_{XY} is the estimated clip duration for joining C_X to C_Y, T_{min} is the minimum transition length we allow, $\mathcal{M}_{Y\leftarrow X}(i)$ is a mapping from agent IDs in C_X to those in C_Y (Section 5.1) and l_{XY} is the estimated clip length. The target velocity of the group, $\mathbf{v}_{target}$ is taken from the flocking simulation rules, while ε is a constant, around 0.2, used because the behavior rules for re-arranging the group (Section 5) compete with other rules (such as avoiding agent collisions) and so cannot produce the maximum possible relative speed within the group.

If T_{XY} is greater than a user defined maximum clip length, T_{max}, we insert additional *link nodes* into the graph between the configuration clusters. The new nodes are connected in sequence along the original edge, and a complete family of clips is defined between each link node. The number of link nodes, $\lfloor T_{XY}/T_{max} \rfloor$, is chosen to keep the clip length below T_{max}.

The configurations for the link nodes are found via constrained simulation. Consider an edge joining cluster configuration C_X to C_Y in time T_{XY}. We constrain a flock to follow a near-straight path, starting in configuration C_X and finishing in configuration C_Y after time T_{XY}. The details of this step are described in Section 5. From the resulting animation we take $\lfloor t_{XY}/t_{max} \rfloor$ equally spaced configurations.

4.4. Statistics Collection

During synthesis, we wish to have a probability distribution that tells us the relative likelihood of moving between two configurations with a particular edge. These probabilities are not uniform. For instance, a long stretched out flock is highly unlikely to transition to a circular group along a straight path because the behavior rules tend to discourage it, but such a transition is more likely if the flock is moving on a sharply curved path. During synthesis, the probabilities can be used in two ways: for random synthesis they are the transition

probabilities for a Markov process on the graph, or they can be used to form objective functions for search on the graph.

At each node C_i, we store the probability of transitioning to node C_{j^+} along a path of curvature θ_+ given that we entered C_i from node C_{j^-} along a path of curvature θ_-. The probabilities are tabulated at each node, indexed by incoming clip curvature and the graph node it connects to. In 3D we must also associate a probability with changes in the plane of curvature. In our graphs we found that the plane of curvature is about twice as likely to remain the same as it is to take a new value, and that any new value is equally likely. We hard code this result into our sampling process, although it could also be tabulated at the cost of larger tables.

We estimate the probabilities by stepping through configurations in the input and finding each one's closest cluster configuration in the graph. For every combination of incoming and outgoing edges from the graph node (representing a small segment of potential output motion), we compute a weight based on how closely the graph path matches the input path and configurations. This weight is accumulated in the probability table, which we normalize after processing all the input frames.

5. Constrained Simulation

Graph construction relies on the ability to construct clips that take the group from one configuration to another along a specific trajectory. We achieve this with three flocking rules working in combination with the default rules. The input to these rules is the target trajectory and initial and final configuration for the group. We also require a mapping, or correspondence, that tells us which agent in the input configuration maps to which in the output. This is due to our assumption that all the agents are following the same rules, so they can be substituted for each other in an arrangement. We discuss this matching procedure first.

5.1. Agent Matching

A match between two configurations, C_X at time t_X and C_Y at time t_Y, consists of a permutation, $\mathcal{M}_{Y\leftarrow X}(i)$, that maps agent identities in C_X into those in C_Y. We find the optimal matching with maximum weight bipartite matching, which finds the matching that minimizes the total difference in position between corresponding agents while enforcing a one-to-one mapping. A bipartite graph is constructed with N nodes on the left hand side labeled $1 \leq i \leq N$ and N nodes on the right indexed by j. Every node on the left is connected to every node on the right and each edge is given the weight

$$-\sum_{k=-N_w/2}^{N_w/2} w_k \left|\mathbf{x}_c^{(i)}(t_X) - \mathbf{x}_c^{(j)}(t_Y)\right|^2$$

The matching finds the set of edges in the graph that maximizes the sum of their weights subject to the constraint that every node i is connected to a single node j. Agents i and j are then in correspondence: $\mathcal{M}_{Y\leftarrow X}(i) = j$. We use the Kuhn-Munkres algorithm [Kuh55], also called the Hungarian method, to solve this problem, which runs in time $O(N^3)$.

The matching associated with each clip is stored with the clip. During synthesis from the graph, the currently active matching is accumulated as clips are traversed (Section 6), so at any time it is possible to identify which agent belongs to which path in the clip.

5.2. Constrained Flocking Rules

We define three new rules that are applied in combination with the standard flocking rules (cohesion, avoidance, etc.). The result is plausible flocking motion that comes very close to meeting the required constraints.

Follow path: This rule aims to step the COM of the group along the required trajectory, at a speed approximating the group's unconstrained speed. The actions of other rules will add variation to this target speed. During every time step, we predict the target position of the COM by advancing a constant length along the target path. The acceleration applied to every agent is $\mathbf{a} = \frac{\mathbf{x}_{target} - \mathbf{O}_c}{\delta t^2} - \frac{\dot{\mathbf{O}}_c}{\delta t}$ where $\mathbf{x}_{target}$ is the target location and δt is the time step.

Match configuration: We modify Reynolds' seeking rule [Rey99] so that each agent attempts to move toward its target location in configuration space, $\mathbf{x}_{target}^{(i)}$, transformed back into world space. The "Follow path" equation above can be used to convert this target into an acceleration, but this results in agents flying backwards within the flock early in the clip, as they rush to their final configuration positions. To avoid this, we limit the maximum speed the agent can move within the flock, based on the time remaining to the destination and the distance the agents must go within the configuration.

Target orientation: This rule attempts to orient the configuration correctly relative to the direction of travel. The acceleration to apply is derived from the desired change in COM velocity: $\mathbf{a} = (\mathbf{v}_{target} - \dot{\mathbf{O}}_c)/\delta t$.

At the end of the alloted clip length there may be some small deviation from the required configuration. We correct this by standard displacement map editing of the paths.

5.3. Clip Storage

Clip storage is the largest memory consumer within the graph structure. Rather than storing densely point sampled agent state, we observe that most groups tend to move smoothly and hence use B-splines to represent the trajectories. The spline for each agent is parameterized by frame number and gives the agent's location in the configuration coordinate system of the first frame in the clip. In other words, every stored clip starts off with the group at the origin heading along the positive X axis. We also keep a spline for

the trajectory of the center of mass of the group because it is helpful in synthesis and rendering.

If memory consumption was a major concern, clip storage could be further optimized by storing the agents' positions in configuration coordinates. Presumably these vary relatively little over the course of the clip (depending on how fast the agents mix), and hence could be efficiently compressed with differential coding techniques or vector quantization.

6. Synthesis Algorithms

The process of synthesizing from a GMG is identical to that for human motion graphs with the exception that we must track agent correspondences. A cumulative correspondence, $\mathcal{M}_{current}(i)$ is maintained as synthesis progresses. The agent that started as agent i uses agent $\mathcal{M}_{current}(i)$'s state from the currently active clip. Initially, $\mathcal{M}_{current}(i) = i$. At each transition, $\mathcal{M}'_{current}(i) = \mathcal{M}_{CC'}(\mathcal{M}_{current}(i))$, where $\mathcal{M}_{CC'}$ is the correspondences stored for the transition.

The synthesis process is independent of the method for choosing the sequence of edges to be followed. In this section we discuss various graph walk algorithms, each designed to produce a particular target motion: random walk using the computed transition probabilities; constraining the group to a region; following a designated path; and planning to meet discrete goals while avoiding obstacles.

6.1. Random Walk

Random synthesis from a GMG is a Markov process on the graph, with the transition probabilities given by the tables built during graph construction. Each time a transition point is reached, we sample a new outgoing edge given the incoming edge, using standard sampling techniques for tabulated discrete distributions. While random synthesis produces reasonable group motion, it offers no control over the group.

6.2. Path Following

Path following with a GMG can be done with greedy search thanks to the well defined trajectories available out of each transition. Paths to follow can be input in any form that enables evaluation of points on the path at a given time. At each transition point, we choose the minimum cost outgoing edge, where cost is defined as the sum over frames of the difference between the flock's center of mass and the target path.

6.3. Region Constrained

Most virtual worlds are finite in extent, and we would like to constrain the flock to stay within the world. With traditional flocking simulations this would be done either with collision avoidance for the virtual walls of the world, or with other specific rules. We don't use those in GMGs because, while a virtual wall around the world may be invisible, the group's collision response to it will reveal its presence. With GMGs we can give the impression that the group stays within the region purely by chance.

The region constraint restricts the random walk on the graph to edges that remain within the region. At each transition node during synthesis, we choose an outgoing node at random, then conduct depth first search to find the first future path that remains inside the region (we test the center of mass of the group for inclusion in the region, but more stringent tests could be used). If no such path can be found, we choose another clip and try again. The dense connectivity and controlled transition interval is important in this application because it limits the depth of the search to about 4 clips in practice.

Descr.	#Conf.	#Link	#Edges	Mem.	Time
20-2D	10	0	1300	8	100
20-3D	10	0	13600	103	1000
100-2D	2	3	104	5	1100

Table 1: *Data for the GMGs we have constructed. We give a descriptive label, the number of configurations, link nodes and edges, the total memory consumption of the graph in MB, and the total time to construct the graph in seconds.*

6.4. Path Planning

There are many ways to plan paths based on a motion graph – here we present just one that takes as input a set of destination points to be hit by the center of mass of the flock and a set of points with radii to be avoided. We divide the problem into n destination search problems solved in succession. We use A^* path planning to solve each sub-problem. The estimated cost to reach the destination from the current node is simply Euclidean distance. The sub-path cost function favors short paths as well as those that contain likely transitions:

$$g(Path_n) = g(Path_{n-1}) - L(n) * \ln p_n$$

where $Path_n$ is the nth node in the path, $L(n)$ is the length of edge $Path_nPath_{n-1}$, and p_n is the probability of choosing edge $Path_nPath_{n-1}$, taken from the tables stored in the graph. To begin, $g(Path_0) = 0$ and $\ln p_1$ is computed assuming the group came from the $\theta = 0, \phi = 0$ incoming edge. We set $g(Path_n) = \inf$ when the path hits an obstacle.

7. Results

We have built three demonstration GMGs, with data summarized in Table 1. Each flock uses different rule parameters to produce a different style of flock motion. Memory usage is determined by several factors. It is roughly linear in the number of agents in the group, because we need to store trajectories for each agent. The number of configuration nodes

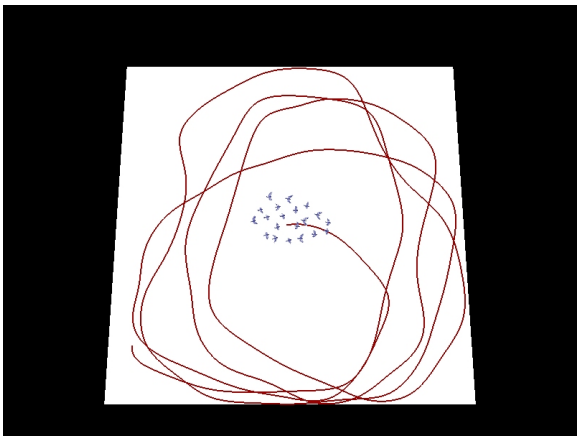

Figure 5: *Constraining a flock to a region. The trajectory of the flock is shown by the line.*

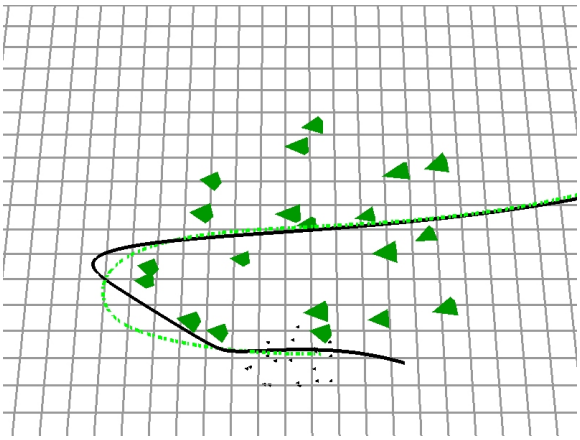

Figure 6: *A top-down snapshot of a flock following a designed path constraint in 3D. The dotted green line is the trajectory of the flock and the solid black line is the designed path.*

used and the number of link nodes required determines the number of edges in the graph (along with the discretization of curvature for the clips). In turn, the number of edges gives the number of clips required and the size of the probability tables.

The construction time of a graph depends on the number of clusters desired – more configurations require more time – and the total number of edges that must be simulated to produce clips. For example, in the 20-agent graphs we built, clustering accounted for about 70% of the time, whereas in the 100-agent graph the clustering took about 30%. This is primarily due to the link nodes, and hence larger number of edges, required for the 100-agent graph. We used fewer configurations for the 100-agent because the required link nodes are additional configurations that add variety to the synthesized motion without affecting the ability to satisfy

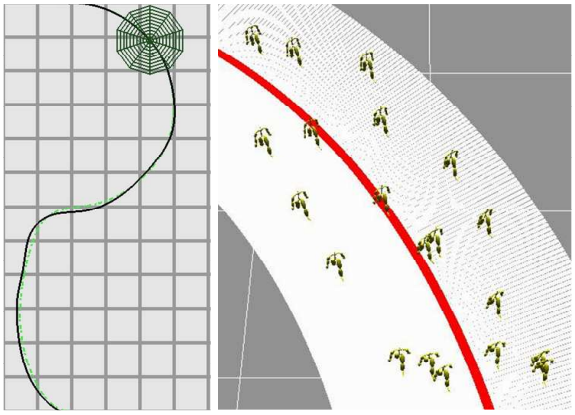

Figure 7: *The left shows the trajectory of the flock in dotted green and the designed path in solid black. The right shows a snapshot of a human crowd with the character poses driven by a human motion graph.*

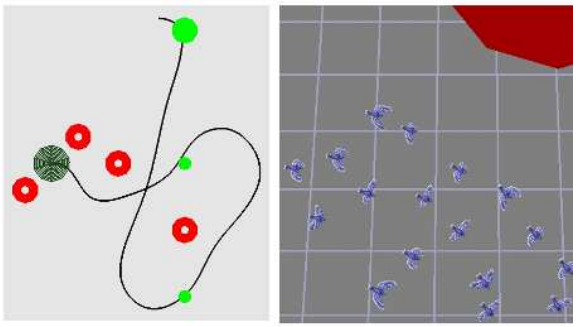

Figure 8: *The overview of an entire flock motion is shown in the left. The black line is the trajectory of the flock. The red circles with holes are the obstacles and the solid green circles are the destinations. Right is a snapshot of the flock avoiding the obstacle.*

constraints. This indicates that good variety can be obtained with only a small number of dissimilar cluster configurations.

The accompanying video includes animations generated from our graphs, and demonstrates the various synthesis techniques. Figures 5 through 8 show frames from each demonstration. We applied the graphs to both bird models and human figures. The birds use a fixed wing-flap animation cycle while moving along the paths. To animate the humans, we use a walk cycle that is time-warped and edited to follow the path.

The time to synthesize motion from the graph depends on the particular method, with random synthesis being fastest and planning the slowest. Random synthesis for 100 agents took about 7.5ms per virtual second on a 2.4GHz P4. In other words, for real-time playback it would consume about 0.75% of the CPU. This compares to about 500ms, or 50%, for sim-

ulating the same group. While the exact numbers are likely to vary based on the precise implementation of the flocking simulation and GMGs, the difference is practically significant: less than 1% of CPU is a reasonable price to pay for secondary group motion that adds realism to a virtual environment; 10% is not. The trade-off is in memory consumption, but for secondary motion applications a small graph with few configurations is likely to be acceptable.

Other forms of synthesis are more expensive. Synthesizing 20 agents constrained to a 2D region takes around 15ms per virtual second, with the additional time being spent on look-ahead. Following a path with a 20 agent group in 3D consumes about 229ms per virtual second in an unoptimized implementation (it is cheaper in 2D). Finally, the time taken for planning is very dependent on the specific constraints, as is typical of A^* implementations. The example we present took 4.4s to plan 70s of output motion.

The primary limitation of GMGs, as with any data-driven method, is that situations not in the data cannot be reproduced. In the context of group animation, this problem is most apparent in environmental interactions. For instance, the group cannot split around an obstacle unless a clip with a similar sized obstacle is present in the pre-recorded data. Similarly, individual agents cannot modify their motion in response to a local environmental feature, such as another agent not part of the group.

We could, however, mix data-driven motion with simulation. If the group approaches an obstacle, it could be switched to simulation for a short while then forced back into a configuration from the graph. Our techniques for building the graph support this kind of constrained final configuration. Similarly, individual agents could be simulated while the others followed their fixed trajectories.

Our graphs are built from simulated data, and rely on simulation to produce clips with smooth transitions. We do this because group motion typically lacks recurring configurations – the equivalent of a single human's relaxed stance or periodic motion such as walking. Exceptions include groups playing sports where there is some start positioning (volleyball, for instance). For groups with recurring configurations, GMGs could be constructed more analogously to human motion graphs, by finding matching arrangements in the input and re-arranging the input clips to form the graph. We attempted this approach with our input simulation data, using a modified version of the matching algorithm of Section 5.1 to identify similar configurations (a more stringent metric than Hausdorff distance is required because the transitions are constructed by blending rather than simulation). However, we found that pairs of sufficiently similar configurations were very rare in the input data, resulting a poorly connected graph with very long clips.

8. Conclusion

Group motion graphs offer efficient and controllable motion for small to medium sized groups. Open problems include creating graphs directly from captured motion and further reductions in run-time cost. In particular, for large groups the cost of rendering starts to dominate the cost of simulating. Using ideas from video textures [SSSE00] and crowd impostors [DHOO05], it should be possible to pre-render the motion to textures that are billboarded into a scene. The primary challenge to overcome is view independence.

A second appealing concept is mixed simulation and motion graphs, which is more applicable for GMGs because we work from simulation in the first place. This could be viewed in a simulation level of detail framework, with simulation simplified to graphs simplified to billboards, with the latter offering the most promising way to get computational costs that do not depend on the number of agents.

Acknowledgments

Funding for this work was provided by NSF grant CCR-0204372. We used equipment donated by Intel and Nvidia. Many thanks to the UW Graphics Group and anonymous reviewers for comments and help.

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Transactions on Graphics 21*, 3 (2002), 483–490. 1, 3

[AI.03] AI.IMPLANT:, 2003. http://www.ai-implant.com. 1

[AMC03] ANDERSON M., MCDANIEL E., CHENNEY S.: Constrained animation of flocks. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), pp. 286–197. 3

[BH00] BRAND M., HERTZMANN A.: Style machines. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), pp. 183–192. 3

[BLA02] BAYAZIT O. B., LIEN J.-M., AMATO N. M.: Better flocking behaviors in complex environments using global roadmaps. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR'02)* (2002). 3

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: A real-time geometry/impostor crowd rendering system. In *Proceedings of the ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games* (2005), pp. 95–102. 9

[Dun02] DUNCAN J.: Ring masters. *Cinefex*, 89 (April 2002), 64–131. 1

[FP03] FORSYTH D. A., PONCE J.: *Computer Vision: A Modern Approach*. Prentice Hall, 2003. 4

[GMHP04] GROCHOW K., MARTIN S. L., HERTZMANN A., POPOVĆ Z.: Style-based inverse kinematics. *ACM Trans. Graph. 23*, 3 (2004). 3

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

[GVK04] GO J., VU T., KUFFNER J. J.: Autonomous behaviors for interactive vehicle animations. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), pp. 9–18. 3

[JF03] JAMES D. L., FATAHALIAN K.: Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph. 22*, 3 (2003). 3

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. In *SIGGRAPH 2002* (2002), pp. 473–482. 1, 3

[Kuh55] KUHN H. W.: The hungarian method for the assignment problem. *Naval Res. Logist. Quart. 2* (1955), 83–97. 6

[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. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 491–500. 1, 3

[LL04] LEE J., LEE K. H.: Precomputing avatar behavior from human motion data. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), pp. 79–87. 3

[LWS02] LI Y., WANG T., SHUM H.-Y.: Motion texture: a two-level statistical model for character motion synthesis. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 465–472. 3

[Men99] MENACHE A.: *Understanding Motion Capture for Computer Animation and Computer Games*. Morgan Kaufman, 1999. 3

[PC01] PERBET F., CANI M.-P.: Animating prairies in real-time. In *Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), pp. 103–110. 3

[Rey87] REYNOLDS C. W.: Flocks, herds, and schools: A distributed behavior model. In *Computer Graphics: SIGGRAPH '87 Conference Proceedings* (1987), vol. 21(4), pp. 25–34. 1, 2

[Rey99] REYNOLDS C. W.: Steering behaviors for autonomous characters. In *1999 Game Developers Conference* (1999), pp. 763–782. 3, 6

[RP04] REITSMA P. S. A., POLLARD N. S.: Evaluating motion graphs for character navigation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), pp. 89–98. 2

[SSSE00] SCHÖDL A., SZELISKI R., SALESIN D. H., ESSA I.: Video textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), pp. 489–498. 9

[YT99] YU Q., TERZOPOULOS D.: Synthetic motion capture: Implementing an interactive virtual marine environment. *The Visual Computer* (1999), 377–394. 3