Computer Sciences Department

Data-Driven Group Animation

Yu-Chi Lai
Stephen Chenney
Shaohua Fan

Technical Report #1538

October 2005

UNIVERSITY OF WISCONSIN MADISON

Data-Driven Group Animation

Yu-Chi Lai Stephen Chenney
Shaohua Fan
University of Wisconsin, Madison

October 2, 2005

Abstract

We introduce Data-Driven Groups (DDGs), an animation technique for groups of discrete agents, such as flocks, herds, or small crowds. DDGs are a natural extension of human motion graphs to groups. They create motion by piecing together a sequence of recorded motion clips. The graph structure identifies clips that can be appended while maintaining continuity in the motion. We discuss a method for building DDGs for discrete agents and algorithms for extracting motion from the graph to meet environment constraints. The resulting animations show realistic motion at significantly reduced computational cost compared to simulation.

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 [18], to commercial systems like MASSIVE [7] and AI.implant [1], to any number of crowd animation systems. Agent models must be highly efficient for applications in computer games and interactive systems, particularly when used for 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 DDGs, 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). DDGs record the motion of an agent group as a whole, including the configuration of agents within the group. As with human motion graphs [3, 11, 14], 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 extracting clips from agent-based simulations that can be pieced together in a graph structure. We demonstrate uses of the resulting graph, including random motion restricted to a region.

There are three principle advantages to DDGs: efficiency, style control, and constraint satisfaction. The run-time CPU cost of DDGs 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.

This report describes a method for building motion graphs for groups of discrete agents. The primary component is a solution to the problem of finding good transition positions to connect the clips seamlessly.

DDGs 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.

2 Overview

A DDG 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 DDG requires identifying transition points and, possibly, modifying the clips to achieve seamless transitions.

All of the graphs in this paper are based on the flocking model introduced by Reynolds [18]. 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 neigh-

bors. The output state of the system, $S(t)$, is the position and velocity of each agent at each timestep, and it is this that we store in group motion clips. At a high level, DDGs could be applied to other group animation systems, even motion-captured groups. However, effective DDGs 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.

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 [17] 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.

DDGs are constructed by finding common configurations in an input animation sequence and using them directly as transition nodes. This is a direct application of motion graph construction for human motion (see Section 3). The primary problem to be solved is creating a comparison metric between group configurations, which we address in Section 4.

A variety of graph search algorithms can be applied to the graphs to synthesize new motion with particular properties, as we describe in Section 6. We demonstrate random walk on the graph, 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. Hence, we describe a random walk with look-ahead that restricts transitions to keep the group within a region.

3 Related Work

Motion capture [15] is the prominent application of data-driven animation. The formalism of connecting clips into a graph structure was independently developed by Arikan and Forsyth [3], Kovar et al. [11] and Lee et al. [14]. 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. [8], 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.

Other applications of data-driven synthesis for animation range from synthetic motion capture of fish body motion [21], to capturing the response of grass to a wind field [16]. In the former case, the state-space was the pose of the fish, and

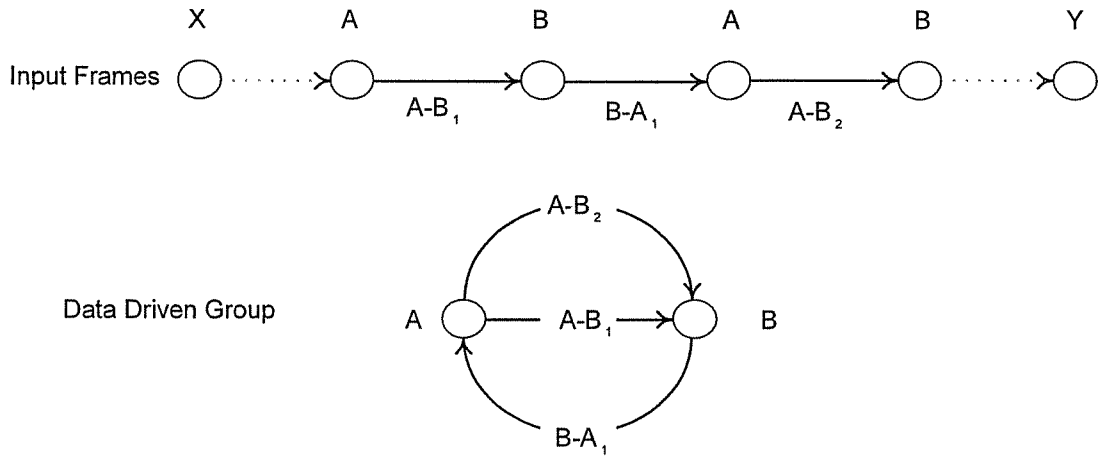

Figure 1: *An input animation with two configurations, A and B, that each appear twice in the sequence, and the DDG that results.*

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 [10], 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.

DDGs provide a means of controlling the trajectory of a group as a whole. Previous techniques for guiding flocks include Reynolds' steering behaviors [19] and the roadmap techniques of Bayazit et al. [4]. While these techniques are sufficient for guiding a flock along some general path, they cannot guarantee the correct outcome because they rely on rules that could be superseded by other rules. The degree of control we offer encompasses these previous methods and adds additional tools. Anderson et al. [2] describe an algorithm for global control of a flock that can meet hard constraints, but the method is not suitable for on-line control. DDGs 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. [9] for controlling single vehicles, but they did not work with an explicit graph structure.

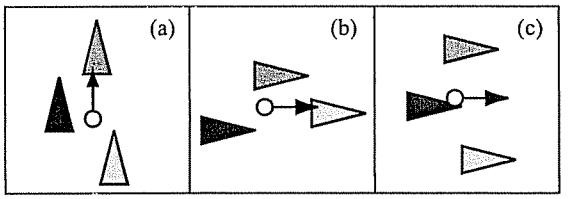

Figure 2: *Configurations (a) and (b) are similar, because agents are arranged the same way with respect to the average velocity, despite that velocity being different in world coordinates and despite some agents switching locations. Group (c) is not similar because the arrangement is different with 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.*

4 Data-Driven Groups

A clip of motion is defined as a sequence of regular sampling vectors represented all agents' positions and velocities. A DDG is a directed graph where all edges correspond to clips of motion. Vertices in the graph, or nodes, serve as choice points connecting these clips, i.e., each clip corresponding to an outgoing edge is potentially the successor to any incoming edge's clip. We use the term *node configuration* to refer to the configuration of the group at a graph node. For a node to have multiple outgoing edges, there must be multiple clips that can follow the clip(s) leading into the node.

Our strategy for constructing a graph is to search for node configurations in the input data. Good configurations for nodes are those that recur in the input (Figure 1): the clips preceding each appearance of the node configuration in the input are incoming edges in the graph, and succeeding clips are output edges. The instances of a particular node configuration do not need to be exactly identical – simple blending techniques can reliably generate a transition if two configurations are "close" to each other.

The remainder of this section is divided into two parts. First, we describe our comparison metric for identifying similar configurations. We then explain how to construct the DDG.

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.

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 $\dot{\mathbf{x}}_i$. At any instant, the origin of configuration space is the center of mass of the agents and the x axis is aligned with the average agent velocity:

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

$\mathbf{O}_c$ 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 $\mathbf{up}$ 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.

4.2 A Discrete Agent Comparison Metric

The metric used for comparing two configurations should give a small distance when agents in one configuration can be blended to the next without visual artifacts. As discussed above, we allow a rigid transformation to re-orient one configuration onto the other, and we allow agents to swap identities during the blend. However, we must have a one-to-one correspondence between agents to enable those in the initial blend configuration to switch roles for the final configuration.

Say we have N agents in the group. At the time, t_A, that configuration C_A appears, each agent has a location, $\mathbf{x}_i(t_A)$ for $1 \leq i \leq N$. Similarly, at time t_B the agents are in configuration C_B in positions $\mathbf{x}_i(t_B)$. During the blend from C_A to C_B, each agent must move from position $\mathbf{x}_i(t_A)$ to position $\mathbf{x}_{\mathcal{M}_{B \leftarrow A}(i)}(t_B)$, where $\mathcal{M}_{B \leftarrow A}(i)$ is a one-to-one mapping that tells us which slot in C_B will be occupied by agent i from C_A.

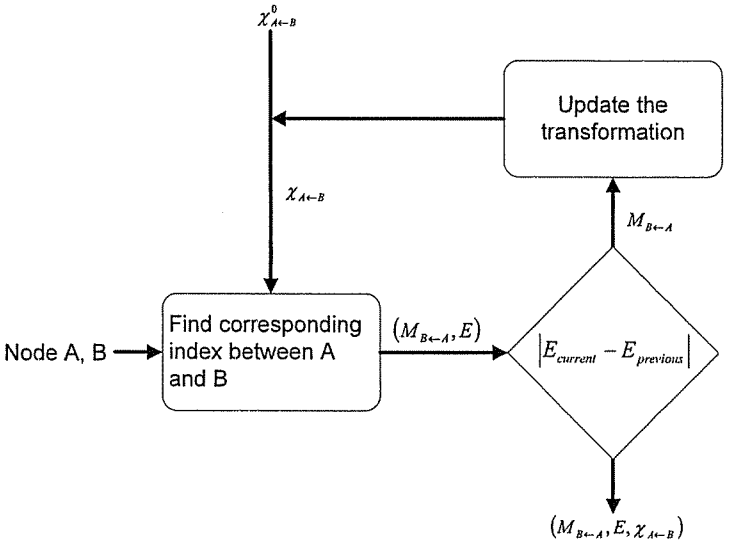

Figure 3: *The process used to find a matching between configurations* **a** *and* **b**

Let $\mathcal{X}_{A\leftarrow B}$ be a transformation intended to align C_B with C_A. Our metric, $E(t_A, t_B)$ is defined as

$$E(t_A, t_B) = \min_{\mathcal{M}_{B\leftarrow A}, \mathcal{X}_{A\leftarrow B}} \frac{1}{N} \sum_{i=1}^{N} \left(\mathbf{x}_i(t_A) - \mathcal{X}_{A\leftarrow B}\mathbf{x}_{\mathcal{M}_{B\leftarrow A}(i)}(t_B)\right)^2$$

We solve the minimization problem using a combination of the iterated closest points algorithm [5] and bipartite graph matching (Figure 3(b)).

- Optimizing $\mathcal{M}_{B\leftarrow A}$: We construct a bi-partite graph, with indexes i in one partition and j in the other, with every vertex in one partition connected to every vertex in the other. The edges are labeled with the distance $-\left(\mathbf{x}_i(t_A) - \mathcal{X}_{A\leftarrow B}\mathbf{x}_j(t_B)\right)^2$. We then run the Kuhn-Munkres [12] algorithm to find the maximal weight bi-partite matching. Each edge from a node i to node j in the maximal matching provides $\mathcal{M}_{B\leftarrow A}(i) = j$.

- Optimizing $\mathcal{X}_{A\leftarrow B}$: With $\mathcal{M}_{B\leftarrow A}$ set, we can update $\mathcal{X}_{A\leftarrow B}$ by applying standard point set registration techniques [5].

The sequence of optimizing $\mathcal{M}_{B\leftarrow A}$ and $\mathcal{X}_{A\leftarrow B}$ is repeated until the distance ceases to change. The process must converge because the sub-steps never increase the distance, and the minimum distance is 0. A starting value for $\mathcal{X}_{A\leftarrow B}$ is required. We use the transformation that aligns the configuration coordinates (Section 4.1).

4.3 Constructing a Data-Driven Group

With the method described in previous section, we can construct the graph with a method similar to [11]. First, we insert a set of candidate configurations into a long sequence of simulation at a constant frequency. We then set up a difference matrix whose $(i,j)th$ element is the value $E(t_i, t_j)$. A set of configuration nodes is extracted by locating local minima in the difference matrix. We take some number of the lowest local minima as the configuration nodes. In our case we chose a number of nodes proportional to the input clip length. Once the configuration nodes have been chosen, the clips joining them are formed into edges of the graph, and the strongly connected components algorithm is run to trim dead-ends (see Kovar et al. [11]).

5 Synthesis Algorithms

The process of synthesizing from a DDG is identical to that for human motion graphs with the added detail of tracking 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, $\mathcal{M}'_{current}(i) = \mathcal{M}_{C' \leftarrow C}(\mathcal{M}_{current}(i))$, where $\mathcal{M}_{C' \leftarrow C}$ 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 two graph walk algorithms, each designed to produce a particular target motion: random walk and constraining the group to a region.

5.1 Random Walk

Random synthesis is simply random graph walk on a DDGs. Each time a transition point is reached, we randomly pick an outgoing edges from that node. While random synthesis produces reasonable group motion, it offers no control over the group.

5.2 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.

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

Descr.	Trans.	Mem.	Time
20-2D	7200	50	2200
20-3D	7200	50	2400

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

remains inside the region (we test the center of mass of the group for inclusion in the region). If no such path can be found, we choose another clip and try again.

6 Results

We have built two demonstrations with data summarized in Table 1. Each flock uses the same rule parameters but one in 2D and the other in 3D. Memory usage is determined by the total animation frames stored, and total number of agents involved. It is linear with the number of agents in the group. We choose a fixed portion of local minima in the difference matrix for configuration nodes, so the number of transition edges is roughly fixed (the process to trim dead-ends may reduce the number of nodes and edges). Figure 4 is the snap shot of the constrained flock inside a fixed region.

Random synthesis took about 7.5ms per virtual second on a 2.4GHz P4. This compares to about 100ms for simulating the same group. The difference is practically significant: less than 0.1ms per frame is a reasonable price to pay for secondary group motion that adds realism to a virtual environment; 10ms 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.

The primary limitation of DDGs, 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.

7 Conclusion

DDGs offer efficient and controllable motion for small to medium sized groups. Open problems include creating graphs directly from captured motion and further reduc-

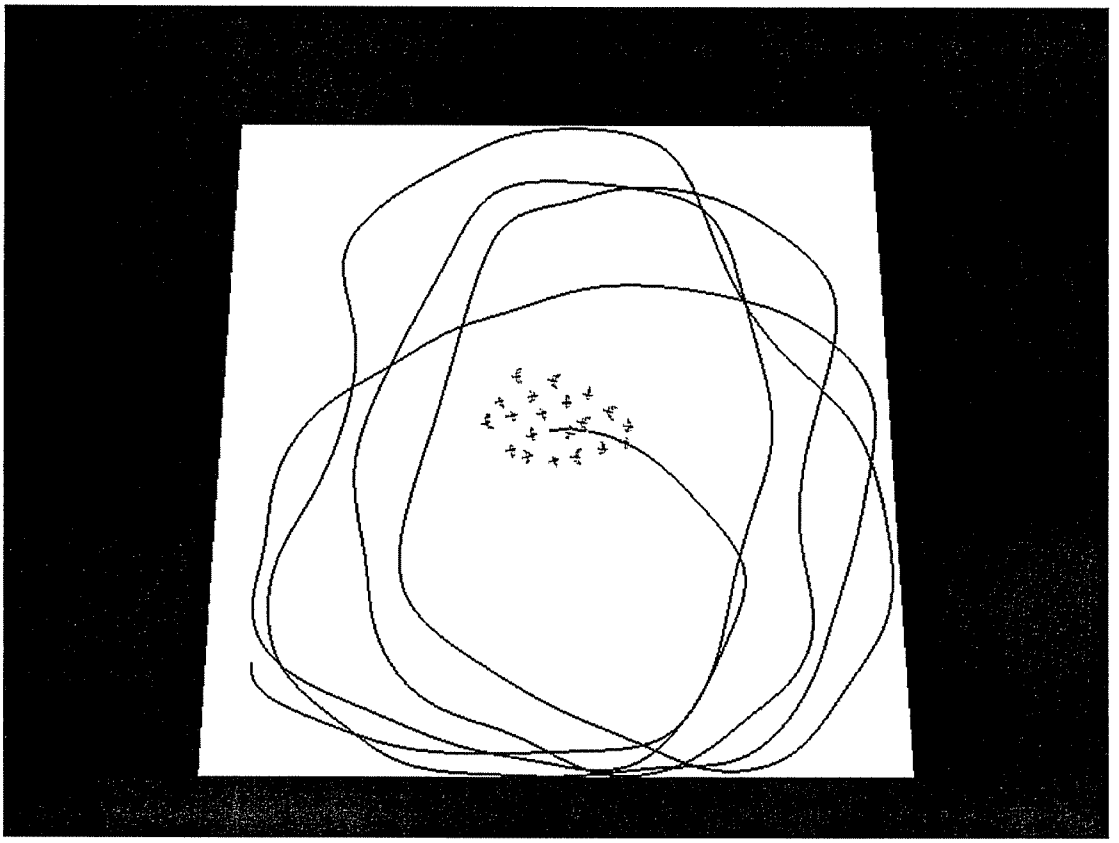

Figure 4: *Constraining a flock to a region. The trajectory of the flock is shown by the dark red line.*

tions 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 [20] and crowd impostors [6], 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. The result would be computational costs that do not depend on the number of agents.

We found that pairs of very similar configurations were rare in the input data, resulting a poorly connected graph with very long clips. This is due to a lack of recurring configurations because there is no such thing as a regular gait or resting pose for most groups, as there is in human motion. In addition, the complexity of the distance metric computation is $O(N^3)$. We have extended our work to create motion graphs using constrained simulation to build clips between configuration nodes [13]. With the help of new flock rules, we can construct the flock to smoothly transform from one configuration to another while following a designed path.

Acknowledgments

This work was partly funded by NSF grant CCR-0204372, and equipment donations from Intel.

References

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

[2] Matt Anderson, Eric McDaniel, and Stephen Chenney. Constrained animation of flocks. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 286–197, 2003.

[3] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Transactions on Graphics*, 21(3):483–490, 2002.

[4] O. Burchan Bayazit, Jyh-Ming Lien, and Nancy M. Amato. Better flocking behaviors in complex environments using global roadmaps. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR'02)*, 2002.

[5] Paul Besl and Neil McKay. A method for registration of 3-d shapes. *IEEE Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feburary/March 1992.

[6] Simon Dobbyn, John Hamill, Keith O'Conor, and Carol O'Sullivan. Geopostors: A real-time geometry/impostor crowd rendering system. In *Proceedings of the ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*, pages 95–102, 2005.

[7] Jody Duncan. Ring masters. *Cinefex*, (89):64–131, April 2002.

[8] Michael Gleicher, Hyun Joon Shin, Lucas Kovar, and Andrew Jepsen. Snap-together motion: assembling run-time animations. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 181–188, 2003.

[9] Jared Go, Thuc Vu, and James J. Kuffner. Autonomous behaviors for interactive vehicle animations. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 9–18, 2004.

[10] Doug L. James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph.*, 22(3), 2003.

[11] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *SIGGRAPH 2002*, pages 473–482, 2002.

[12] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955.

[13] Yu-Chi Lai, Stephen Chenney, and Shaohua Fan. Group motion graphs. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 281–290, 2005.

[14] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 491–500, 2002.

[15] Alberto Menache. *Understanding Motion Capture for Computer Animation and Computer Games.* Morgan Kaufman, 1999.

[16] Frank Perbet and Maric-Paule Cani. Animating prairies in real-time. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 103–110, 2001.

[17] P. S. A. Reitsma and N. S. Pollard. Evaluating motion graphs for character navigation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 89–98, 2004.

[18] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavior model. In *Computer Graphics: SIGGRAPH '87 Conference Proceedings*, volume 21(4), pages 25–34, 1987.

[19] Craig W. Reynolds. Steering behaviors for autonomous characters. In *1999 Game Developers Conference*, pages 763–782, 1999.

[20] Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 489–498, 2000.

[21] Qinxin Yu and Demetri Terzopoulos. Synthetic motion capture: Implementing an interactive virtual marine environment. *The Visual Computer*, pages 377–394, 1999.