An Event-Driven Approach for Crowd Simulation with Example Motions

Ho Kyung Kim Jun Kyu Oh Min Gyu Choi
Hyun Joon Shin Hyung Woo Kang Sung Yong Shin

CS/TR-170

January 14, 2002

K A I S T
Department of Computer Science

An Event-Driven Approach for Crowd Simulation with Example Motions

Ho Kyung Kim Jun Kyu Oh Min Gyu Choi Hyun Joon Shin
Hyung Woo Kang Sung Yong Shin

Abstract

In this paper, we present a novel approach for simulating the behavior of each individual member of a crowd efficiently, of which aggregation eventually evolves into a crowd behavior. The backbone of our crowd simulation is the event-driven approach for collision detection among spheres with unknown trajectories. On top of the events for collision detection, we introduce new types of events to facilitate crowd simulation. We take into account two types of events: extern and internal events. Guided by the external events according to a given scenario, the internal events are mainly due to the interactions among the members themselves and their interactions with the surrounding environment. We invoke a member to react on an event, based on the behavior rules together with the information perceived from neighbors and the surrounding environment. To generate a natural-looking motion, we exploit live-captured motion clips. We obtain the motion by blending example motion clips and retarget it to a specific character model and situation in an online, real-time manner.

1 Introduction

1.1 Motivation

A large number of people in streets, parks, or public squares form a crowd to show complicated scenes that have frequently appeared in applications such as computer games, animations, and movie films. Each of these scenes evolves from the interaction among the members of the crowd. The interaction of a member with others determines its individual behavior, and the aggregate motion of the members yields highly dynamic and complex behaviors of a crowd that are hard to describe interactively.

It is not easy to synthesize visually-convincing motion for each member of the crowd. Although interactive motion editing is a reasonable solution to describe the motion for a small crowd, it is tedious and time-consuming for a large crowd. As the size of the crowd grows, the complexity issue also arises. Suppose that each member is required to interact with the rest of members to determine its behavior. Then, we have a quadratic growth rate of time complexity, which is prohibitive for a large crowd. As Reynolds [28] pointed out, each member pays little attention to the others that are not perceived. Based on this observation, there have been various schemes [3, 4, 25, 29, 34] exploiting

localized perception models. However, most of them check all the members to find their neighbors at every time step.

In this paper, we present a novel approach for simulating the behavior of each individual member of a crowd efficiently, of which aggregation eventually evolves into a crowd behavior. For efficient simulation, we take an event-driven approach. Guided by a given scenario, the events are mainly due to the interactions among the members themselves and their interactions with the surrounding environment. To generate natural-looking motions, we exploit live-captured motion clips.

1.2 Previous Works

Crowd Simulation Reynolds [28] introduced reactive behaviors to simulate groups of simple creatures such as flocks of birds, herds of land animals, and schools of fishes. He also mentioned the possibility of dynamic spatial partitioning to efficiently find the candidate pairs of interaction. Tu and Terzopoulos [34] simulated reactive behaviors of artificial fishes with synthetic visions and exhibited realistic behaviors such as avoiding collision, mating, wandering, and schooling. Brogan and Hodgins [3] proposed an algorithm for modeling group behaviors, in which each individual is forced to move toward a position computed based on group velocity, neighbors, and visible obstacles. They also applied their algorithm to simulating the Border collies and the Olympic bicycle racing [4]

Musse and Thalmann [25] adopted a sociological model to build relations among individuals and presented an adaptive collision avoidance scheme for a pair of individuals based on their distance from the camera. They also presented the ViCrowd model to simulate crowds with different levels of autonomy [26]. Each member is controlled by scripts, behavioral rules, or external controls. In addition, they formed a hierarchy composed of crowd, groups, and individuals. Perlin *et al.* [27] developed the Improv system for scripting interactive actors. They used a blackboard for actors to communicate with each other and introduced a layered behavior model to create complex behaviors from simpler scripts and actions. To produce the animation film, AntZ, PDI [1] developed a crowd simulation system that takes into account a combination of physical forces and procedural rules for generating realistic behaviors of ants.

Collision Detection Efficient collision detection among stationary or moving objects has received increasing attention in computational geometry and computer graphics. Space subdivision techniques [15, 24, 35, 37] have been widely used to localize collision checks. To simplify the shapes of objects and their moving trajectories for efficient collision detection, bounding volumes such as spheres [12] and axis-aligned boxes [7, 21, 23] have been used. Hierarchical bounding volumes such as OBB-trees [10], sphere-trees [13], and BV-trees [17] were also proposed to enclose objects more tightly. Collisions are usually checked at a each time step [2, 7, 10, 11, 24] under the assumption that the interval between time steps is relatively small with respect to the velocities of objects. To detect collisions among polyhedral objects moving along ballistic trajectories, Mirtich [21, 23] exploited the axis-aligned bounding box containing an object during a time interval. Kim *et al.* [15] presented an

event-driven approach with a space subdivision scheme to localize collision checks among moving spheres. Mirtich [22] adopted Jefferson's timewarp algorithm [14] to reduce unnecessary synchronization by rolling back the simulation of the colliding objects to the collision time. Milenkovic and Schmidl [20] gave an optimization-based animation to generate plausible motions for convex objects.

Provided with the maximum norm of the acceleration of every object, Hubbard [12] proposed a collision detection algorithm for moving spheres with unknown trajectories. He defined a four-dimensional bounding volume, called a parabolic horn, that contains a moving sphere for a time interval. To reduce the computational cost for finding the intersections among the horns, he simplified them with hypertrapezoids. To detect collisions among multiple moving spheres at an interactive rate, Kim *et al.* [16] interpreted the parabolic horn as a volume swept by a moving sphere of variable radius, called a time-varying bound, and extended an event-driven approach [15] to cope with the time-varying bounds.

Motion Editing There have been a bunch of research results for generating realistic motion with prescribed motions. Bruderlin and Williams [5] adopted signal processing techniques to modify animated motions. They also introduced displacement mapping to edit motions while preserving their detailed characteristics. Witkin and Popovic [39] presented a motion warping technique for the same purpose. Rose *et al.* [31] generated seamless transitions between motion clips using spacetime constraints. Gleicher [9] simplified the spacetime problem for motion retargeting, that is, adapting a motion of an articulated character to the target character of an identical structure but different size and proportions. Employing an optimization technique, he was able to achieve an interactive performance. To solve the same problem more efficiently, Lee and Shin [18] presented a hierarchical approach based on the multilevel B-spline approximation and a fast inverse kinematics solver. Shin *et al.* [32] proposed a computer puppetry technique that retargets motions in an online, real-time manner based on importance analysis.

Unuma *et al.*. [36] adopted the Fourier analysis to interpolate and extrapolate various motions in mood and style. Bruderin and Williams [5] suggested a scheme for multitarget motion interpolation to blend a number of sample motions in a frequency domain. They also gave a technique for dynamic time-warping to synchronize the sample motions. Using trilinear motion interpolation, Wiley and Hahn [38] produced a desired motion that satisfies a given spatial goal. Rose *et al.* [30] introduced the notion of verbs and adverbs to classify and parameterize the example motions. Based on multi-dimensional scattered data interpolation, they synthesized a new motion by blending the example motions in the parameter space.

1.3 Overview

Given a scenario for crowd animation, our objective is to generate a sequence of motions for each member in accordance with the user-specified behavioral rules. Differently from conventional crowd animation schemes [3, 4, 25, 26, 28, 34] that explicitly specify the movement of every member at each

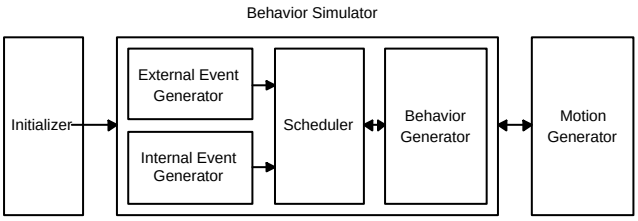


Figure 1: System overview

time step, we propose an event-driven scheme that redirects the behaviors of the members involved in an event whenever it occurs. In addition, to generate visually-convincing motion in real time, we employ an on-line motion blending scheme with live-captured example motions and retarget a blended motion to the specific character model and situation.

In our crowd simulation scheme, an event triggers the members to take certain motions in accordance with the behavioral rules together with the information perceived from neighbors and the environment. Those motions may again give rise to other events among the crowd. The backbone of our crowd simulation is the event-driven approach for collision detection among spheres with unknown trajectories [16]. On top of the events for collision detection, we introduce new types of events to facilitate crowd simulation. We take into account two types of events: external and internal events. An external event is designed to attract attentions from the crowd or to broadcast news among the crowd at a specific time instance as specified in the animation scenario. An internal event is to deal with spatial, temporal and emotional relationships among the crowd. We maintain the candidates for events in a priority queue regardless of their types to generate them in the order of their occurring times.

To efficiently trace the movements of the members and their interactions, we subdivide the whole space into a hierarchy of uniform cells and attach a local billboard at each cell for the cell-specific information such as interesting locations, obstacles, and members in the cell. The information related to the whole crowd and space is stored in a global billboard. When an event occurs, we derive the responses of the members involved in that event, based on their individual behavioral rules together with the information perceived from other members, the surrounding environment, and billboards. To deal with a complex behavior, we introduce a hierarchy of behavior rules. The lowest level of the hierarchy represents a primitive motion, such as walking, running, touching, and handshaking.

To generate a natural-looking motion, we blend example motion clips and retarget the motion, if necessary, to the specific character model and situation. In particular, we blend motions of various speed, gyration, and style to generate human locomotion, which plays a crucial role in the crowd animation. As preprocessing, we parameterize the example motion clips to place them in the parameter space for later on-the-fly blending. Given a parameter vector, we perform multi-dimensional scattered data interpolation to obtain the corresponding motion, and then adapt it to a given trajectory while

avoiding artifacts such as foot sliding and penetration.

The remainder of the paper is organized as follows. After describing an event-driven approach for collision detection in Section 2, we present our behavior simulation model for a crowd in Section 3. In Section 4, we present how to generate a desired motion for each member using example motion clips. We show experimental results to demonstrate the capability of our crowd animation scheme in Section 5. Finally, we conclude this paper and describe future work in Section 6.

2 Event-Driven Collision Detection

Our crowd simulation is based on an event-driven approach for collision detection. Most internal events are generated due to the collisions among the members. The way to avoid the collisions naturally induces the interaction paradigm among the members and also with their environment for crowd simulation.

2.1 Time-Varying Bound

For a small crowd, the collisions can be detected within a reasonable time by repeating the collision check between every pair of the members at each time step. However, for a large crowd, such pairwise collision checks are extremely time-consuming. For efficient collision detection, we decompose the space into a set of subspaces and then trace the movements of the members across the subspaces. The members of a crowd are approximated by their bounding volumes. For the types of crowd consisting of human-like figures, we approximate their members as cylinders, whose projections on the ground plane are circles. For the other types, we approximate their members by spheres. Therefore, our collision detection problem is reduced to that of collision detection among moving 2D or 3D spheres.

We assume that the trajectory of a sphere is not known. Therefore, one cannot predict the future position $\mathbf{x}(\mathbf{t})$ and velocity $\mathbf{v}(\mathbf{t})$ of the sphere. However, the exact position and velocity of any sphere at given time are available immediately after that time. We also assume that the maximum magnitude of the acceleration $\mathbf{a}(\mathbf{t})$ of an object is bounded by a known value A, that is,

$$||\mathbf{a}(\mathbf{t})|| \leq \mathbf{A}. \tag{1}$$

In practice, this makes sense since the speed of an object cannot be arbitrarily large. Let t_c be the current time. Then, a bound on $\mathbf{x}(\mathbf{t})$ can be derived as shown in [12]:

$$||\mathbf{x}(\mathbf{t}) - (\mathbf{x}(\mathbf{t_c}) + \mathbf{v}(\mathbf{t_c})\boldsymbol{\Delta}\mathbf{t})|| \leq \frac{\mathbf{A}}{\mathbf{2}}\boldsymbol{\Delta}\mathbf{t^2}, \tag{2}$$

where $\Delta t = t - t_c$ for $t \geq t_c$. This inequality states that the center $\mathbf{x}(\mathbf{t})$ of the sphere is within a distance of $\frac{A}{2}\Delta t^2$ from the known current position $\mathbf{x}(\mathbf{t_c}) + \mathbf{v}(\mathbf{t_c})\boldsymbol{\Delta}\mathbf{t}$.

Let r be the radius of the sphere approximating the member. Then, the sphere at time t is contained

in a bounding sphere of radius $\hat{r}(t)$ centered at the position $\hat{\mathbf{x}}(t)$:

$$\hat{r}(t) = \frac{A}{2}\Delta t^2 + r, \tag{3}$$

$$\hat{\mathbf{x}}(t) = \mathbf{x}(\mathbf{t_c}) + \mathbf{v}(\mathbf{t_c})\boldsymbol{\Delta}\mathbf{t}. \tag{4}$$

These two equations give a parabolic horn, whose cross section at the time t is the sphere of radius $\hat{r}(t)$ centered at $\hat{\mathbf{x}}(t)$ [12]. This horn can be interpreted as the volume swept by a time-varying sphere of variable radius $\hat{r}(t)$ moving from $\mathbf{x}(\mathbf{t_c})$ to $\mathbf{x}(\mathbf{t_c}) + \mathbf{v}(\mathbf{t_c})\boldsymbol{\Delta}\mathbf{t}$. We call this sphere as a time-varying bound. The collision between two time-varying bounds can be calculated efficiently because of their simplicity in shape and motion. If a pair of spheres collide with each other, then their time-varying bounds also collide beforehand. Hence, we check a collision between two spheres whenever their time-varying bounds collide with each other.

The time-varying bounds of the members play a crucial role in our crowd animation scheme. When two time-varying bounds for a pair of members collide, these members exchange their status such as positions, velocities, and so on. Their neighbors are also identified by exploiting the coherence in time and space. Based on the gathered information for their status and neighbors, the members adjust themselves to their possible collision in future. As they are getting closer, their time-varying bounds collide more frequently. Accordingly, the members are getting ready to the collision by changing their behaviors gradually in accordance with the sequence of signals, that is, the collisions between their time-varying bounds. While moving toward their goals, the members contribute these local variations to the global behavior of the crowd.

2.2 Event Generation

To localize the collision checks among the time-varying bounds, we subdivide the whole space containing the spheres into a set of uniform cubical subspaces. We assume that the diameter of the biggest sphere is a constant multiple of that of the smallest one and that the length of an edge of the cubical subspace is bounded by some constant multiple of the diameter of the biggest sphere. Then, each non-empty subspace has a constant number of intersecting spheres. For every non-empty subspace, we keep a list of time-varying bounds occupying the subspace. These subspaces are maintained in a subspace tree, which is a binary search tree of bounded balance [8].

We keep track of the trajectories of time-varying bounds and their spatial distribution by identifying six types of events: entering, leaving, resetting, colliding, contacting, and separating events. Entering and leaving events are caused when time-varying bounds cross subspaces, and a resetting event is for preventing a time-varying bound from growing excessively. A colliding event is triggered by a collision between two time-varying bounds occupying the same subspace. A contacting event is due to a pair of spheres that become in contact, and a separating event occurs when two spheres in contact separate from each other. We compute the candidate events for each time-varying bound by allowing penetration with other bounds and maintain them in a priority queue.

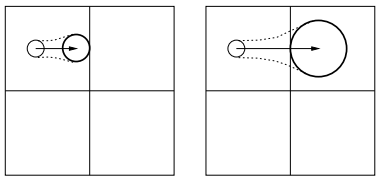


Figure 2: An entering (left) and a leaving (right) event for a time-varying bound.

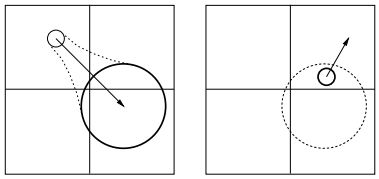


Figure 3: A resetting event occurs when the diameter of a time-varying bound reaches to the length of an edge of a subspace (left). Its radius is reset to that of its original sphere (right).

With penetration among time-varying bounds allowed, the candidate events seem to have nothing to do with the actual events at first glance. However, any pair of time-varying bounds do not penetrate with each other until the earliest event takes place. Moreover, the earliest event is guaranteed to be an actual event. Whenever the earliest event occurs, we modify the subspace tree and event tree in accordance with the type of the event to generate the next earliest event. Repeating this process, all events can be identified in their occurring time sequence.

2.3 Event Handling

Given a time-varying bound $B(s_i)$ of a sphere s_i lying in a subspace, an entering event occurs when $B(s_i)$ touches a face of the subspace to enter its adjacent subspace sharing the face. A leaving event occurs when $B(s_i)$ moves apart from the touching face. These types of events change the list of time-varying bounds for the subspace. Whenever $B(s_i)$ enters a subspace, we add $B(s_i)$ to the list for the subspace. Accordingly, we compute the new candidate colliding events of $B(s_i)$ with the other bounds intersecting the subspace to add them to the event queue. We also compute the candidate events of $B(s_i)$ for entering and leaving from the subspace. In the symmetrical way, we handle a leaving event.

Initially, $B(s_i)$ is identical to s_i. As time passes, it grows and may intersect many subspaces. Accordingly, the candidate events for $B(s_i)$ would increase excessively to degrade the efficiency. We circumvent this problem by restricting the diameter of $B(s_i)$ not to be larger than the length of an edge of a subspace. This restriction forces a time-varying bound to intersect at most eight subspaces, and thus the total number of non-empty subspaces is $O(n)$, where n is the number of the

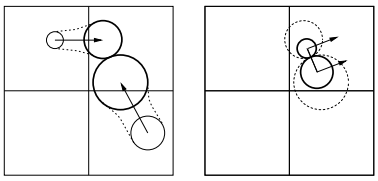


Figure 4: A colliding event for two time-varying bounds (left). A contacting event for two spheres (right).

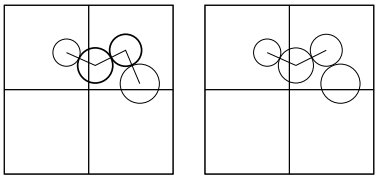


Figure 5: A separating event for two spheres occurs when their relative velocity becomes non-zero (left), and the contact graph is updated (right).

time-varying bounds. Whenever the diameter of $B(s_i)$ reaches to the length of an edge of a subspace, a resetting event occurs. Then, we make $B(s_i)$ be the same as s_i. For each subspace not intersecting $B(s_i)$ anymore, we remove $B(s_i)$ from the list of time-varying bounds. We also renew the trajectory of $B(s_i)$ using its current position and velocity, and then compute its candidate entering, leaving, colliding and resetting events. These events are inserted in the event queue after removing obsolete candidate events of $B(s_i)$ from the event queue.

A colliding event takes place when $B(s_i)$ collides with another time-varying bound $B(s_j)$ intersecting the same subspace. Then, we check the actual collision between the spheres s_i and s_j. We also check whether the two spheres become in contact. If the spheres are not in contact, we reset both of $B(s_i)$ and $B(s_j)$ in the same way as on a resetting event to resume the collision check. Otherwise, we generate the contacting event between s_i and s_j.

Two spheres are in contact if their distance is less than a given threshold and their relative velocity is zero. For two spheres in contact, colliding events between their time-varying bounds would occur repeatedly. To avoid such unnecessary events, we maintain a contact graph, which has a node for every sphere and has an edge between two nodes if their corresponding spheres are in contact. A connected component of this graph yields a contact group of spheres. For a pair of spheres in the same contact group, we do not check collisions between their time-varying bounds even though those bounds belong to the same subspace. When a contacting event between two spheres s_i and s_j occurs, we insert an edge between the two nodes corresponding to the spheres in the contact graph, and then reset both of $B(s_i)$ and $B(s_j)$ in the same way as on a resetting event.

A separating event between s_i and s_j, which are in contact, occurs when their relative velocity

becomes non-zero. We remove the edge between the nodes corresponding to the spheres s_i and s_j from the contact graph, and then reset both of $B(s_i)$ and $B(s_j)$. Suppose that the removal of the edge makes the contact group split into two contact groups. For a pair of spheres s_k and s_l intersecting the same subspace and belonging to the different contact groups, the event queue may not maintain the candidate colliding event for $B(s_k)$ and $B(s_l)$, since s_k and s_l were in the same contact group. Hence, for such a pair of $B(s_k)$ and $B(s_l)$, we calculate their colliding event to insert it in the event queue.

3 Behavior Simulation

To simulate the behaviors of the members, we adopt an event-driven approach. When an event occurs, the members involved in that event are invoked to react based on their behavioral rules together with the information perceived from other members, the surrounding environment, and billboards. In Section 3.1, we first describe the types of events that we consider for our crowd simulation scheme, and then show how to derive the response of each member to each type of events in Section 3.2.

3.1 Behavioral Events

For crowd simulation, we consider two types of events: external and internal events. An external event is for the behavior of a crowd as specified in the animation scenario or by the users. An internal event is for dealing with spatial, temporal and emotional relationships among the crowd. Every event has its occurring time, the members affected by the event, and the event-specific information such as the location where the event takes place. We maintain the candidates for the events in a priority queue to identify the actual event in the order of their occurring times.

In order to properly react on an event, each individual has its own internal status that determines its behavior. The internal status of a member contains its physical state and mental state. The physical state of a member includes its current position, velocity and its time-varying bound. The mental state of a member includes its goal, social state and emotional state. When an event occurs, each member involved in the event takes action by updating its internal status in accordance with its behavioral rules based on its current internal state, its neighbors, and news post on billboards.

We have three types of internal events: spatial, temporal, and emotional events. The spatial events are mainly for collision avoidance. They are also used for information propagation. To trace such events efficiently, we employ the collision detection scheme developed in Section 2. To localize the collision check among the time-varying bounds, we have subdivided the whole space into a hierarchy of uniform subspaces called cells. We attach a local billboard at each cell, which shows the cell-specific information such as locations of interest, obstacles, and members in the cell. Whenever a pair of time-varying bounds collide with each other, they exchange their internal status for collision avoidance and information propagation. In addition, upon entering or leaving a cell, a member updates its internal status by referring to the local billboard. The temporal events are for scheduling actions in

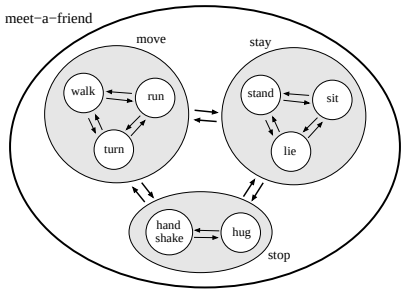


Figure 6: A hierarchical state-transition graph for example social behavior

future, for which a resetting event are a typical example. We can also schedule a sequence of display actions on the screen with such events. Finally, emotional events are for conveying the emotional states of a member to others. An example is to threaten others due to madness. An emotional event is due to the evolution of the internal state of a member directed by its interaction with others.

External events are for explicitly directing the behavior of the members. During the simulation, the members can be directed with external events in two ways as specified in an animation scenario. The behaviors of certain members can be activated independent of their interaction among themselves and with the environment. The external events can also be used to broadcast news to the members. For this purpose, we set up a global billboard for the whole space. On external event, we update the billboard for guiding the simulation.

3.2 Behavior Generation

When an event occurs, we generate the responses of the members involved in that event, based on their individual behavioral rules together with the internal status. For efficient handling of a complex behavior, we introduce a hierarchy of behavior rules. The hierarchy consists of three levels which, from top to bottom, represent social behaviors, spatial behaviors, and primitive behaviors, respectively.

The highest level of the hierarchy consists of social behaviors that are most general and abstract, for example, go-to-school, meet-a-friend, have-lunch, etc. The next level includes spatial behaviors, which are the building blocks for the social behaviors, e.g., move-to-where, stop-moving, stay-where-you-are, etc. The lowest level contains primitive behaviors, which are the building blocks for the spatial behaviors. The primitive behaviors correspond to the given example motions, such as walking, running, touching, or handshaking. In general, the external events govern the social behavior, and the internal events are related to the spatial behaviors.

A behavioral rule is represented as a hierarchy of state-transition graphs as shown in Fig. 6. A behavior at each level is described as a state-transition graph, in which a state (node) represents the behavior at the lower level, and an edge represents a transition from a state to another. A state-transition graph is a directed multigraph, since multiple transitions are allowed between two states. Each edge is

attached with a decision function. This function is a predicate to decide if its corresponding transition can be activated, given parameters such as the event type, the current state, and other information on the status of the member involved. A single node may have multiple outgoing edges produced by a single event. One of them is chosen as the link to the next state depending on the probabilities assigned to them. A state can also have an edge pointing to itself, which means no state transition at all.

When an event occurs, the current state of the graph for each member changes to another state accordingly, or stays where it is. Various factors should be considered in determining the next state, such as the information perceived from other members, the surrounding environment, and the billboards. When a state makes transition to a new state, the state transition graph at its lower level of the new state is initialized to continue behavior generation.

To define the behavior rule conveniently and specifically, we use both scripts and external functions. We use user-defined external functions to define primitive behaviors. The other two types of behaviors at higher levels are described with scripts.

4 Motion Generation

Based on the framework of multidimensional scattered data interpolation proposed by Rose *et al*. [30] and Sloan *et al*. [33], our approach for motion generation consists of two main parts: parameterization and motion blending. As preprocessing, we parameterize example motions to place them in a parameter space. Provided with a vector of parameters, we generate the corresponding motion at each frame by blending the parameterized example motions.

4.1 Motion Parameterization

For effective control over a variety of locomotion, we focus on three parameters: style, speed, and gyration. The rest of parameters for an example motion are specified interactively as discussed in [30]. We have three types of locomotion, walking, jogging, and running to which we assign zero, one, and two, respectively, as their values of the style parameter. Parameters such as speed and gyration will be computed from each example motion.

With some loss of generality, suppose that each example motion is short enough for its speed and gyration to be invariant over its duration. For a lengthy motion with non-homogeneous speed and gyration, we need some preprocessing to decompose it into short motions satisfying this assumption. Since a motion of constant speed and gyration traces a circular trajectory, we approximate the root trajectory of the motion as a circular arc that best fits the projected trajectory on the floor by least-squares fitting. Notice that a straight line is a circular arc of infinite radius. The speed and the gyration of the motion are computed from the length of the arc and its subtending angle by dividing them equally by the duration of the motion.

4.2 Motion Blending

Given a vector of parameters at each frame, we first determine the weights of the example motions. We then perform timewarping to synchronize the example motions. Next, we compute the target motion by blending the timewarped example motions with respect to their weights. Finally, given a trajectory on the floor, we adapt the blended posture to the target character and the environment to follow the trajectory through motion retargeting.

4.2.1 Weight Computation

We employ a multidimensional scattered data interpolation technique suggested by Sloan *et al.* [33]. Their approach is based on the interpolation scheme of Rose *et al.* [30]. Incorporating cardinal basis functions, Sloan *et al.*reformulated this scheme to provide a more efficient blending method. Differently from the original version which interpolates each degree of freedom at every frame, they computed the weights of the example motions for the given parameter vector and blended them with respect to these weights.

Cardinal basis functions consist of two terms: linear basis functions and radial basis functions. The linear basis functions provide an overall approximation of the example motions over the entire parameter space and thus facilitate motion extrapolation outside the convex hull constructed by the parameters of the example motions. The radial basis functions take care of the residuals of the example motions left by the linear basis functions, so that the examples are interpolated exactly.

4.2.2 Incremental Timewarping

To blend the example motions of a wide range of speed, we employ the notion of keytimes introduced by Rose *et al.* [30], that is, the important instances of a motion such as the moments of heel-strikes and toe-offs. All example motions consist of the same sequence of keytime phases, that is, the example motions start with the same foot and take the same number of steps.

Based on keytimes, a timewarping function is defined as a piecewise linear mapping of generic time onto actual time [30]. Once all example motions have reparameterized with their timewarping functions, we can compute the actual time of a blended motion at a given generic time. According to Sloan *et al.* [33], the actual time of the new motion is given by the weighted sum of the actual times of the example motions at a given generic time. This approach works well when the weight of each example motion is fixed during the whole cycle of the motion. In general, the weight may change dynamically from frame to frame, and the speed variation over the example motions may also be quite large. In this case, a blended actual time may go reversely, that is, go back to the past, with respect to the generic time.

To blend motions of various speed with time-varying weights, we propose an incremental time warping technique. The basic idea is to blend the change rates of the actual times with respect to the generic time rather than the actual times themselves and to accumulate the weighted change rate for

incrementally updating the current actual time of the blended motion. With this incremental approach, we can guarantee the monotonicity of the blended timewarping function since the change rates are always positive.

4.2.3 Posture Blending

We generate the target posture at a given generic time by blending the corresponding postures of example motions at the same generic time. The posture of an articulated body is defined by the position of the root segment, its orientation, and the joint angles. We blend not only the joint angles but also the root positions and orientations of the example postures to obtain the posture of the target character, which will be adapted to a given trajectory.

We take a simple weighted sum to blend the root positions. However, due to the non-linearity of the orientation space, this scheme can not be applied directly to blending orientation data such as root orientations and joint angles. Therefore, we provide a new orientation blending technique based on the orientation filtering scheme of Lee and Shin [19].

The basic idea for blending orientations is to transform the orientation data into their analogues in a vector space through the logarithm map, to compute their weighted sum, and then to transform the result back to the orientation space through the exponential map. Here, we employ the exponential and logarithm maps that facilitate a natural, non-singular parameterization for "small" angular displacements [19]. Thus, for mapping orientation data into their vector displacements, we choose the reference orientation that is as "close" to all example orientations as possible.

To choose the reference orientation, we first define the distance metric between two quaternions. Buss and Fillmore [6] used the geodesic norm $\theta = \| \log(\mathbf{q_1 q_2}) \|$ as their distance metric to find the spherical average of points on a d-dimensional sphere $\mathbb{S}$. However, they did not address the problem caused by the antipodal equivalence property of the unit quaternion space. Since $\mathbf{q}$ and $-\mathbf{q}$ represent the same orientation, the angular distance between two quaternions, $\mathbf{q_1}$ and $\mathbf{q_2}$ is

$$\mathrm{dist}(\mathbf{q_1}, \mathbf{q_2}) = \min \left(\| \log \left(\mathbf{q_1^{-1} q_2} \right) \|, \| \log \left(\mathbf{q_1^{-1}} (-\mathbf{q_2}) \right) \| \right). \tag{5}$$

Therefore, the reference quaternion $\mathbf{q}_*$ is obtained by minimizing the sum of squared distances:

$$E = \sum_{i=1}^{N_e} \| \mathrm{dist}(\mathbf{q}_*, \mathbf{q_i}) \|^2. \tag{6}$$

However, this distance metric is not differentiable at $\theta = \frac{\pi}{2}$. Thus, we introduce an alternative distance metric based on a sinusoidal function:

$$\mathrm{dist}(\mathbf{q_1}, \mathbf{q_2}) = \sin(\| \log(\mathbf{q_1^{-1} q_2}) \|). \tag{7}$$

This metric not only approximates the angular distance well but also is differentiable at any point in

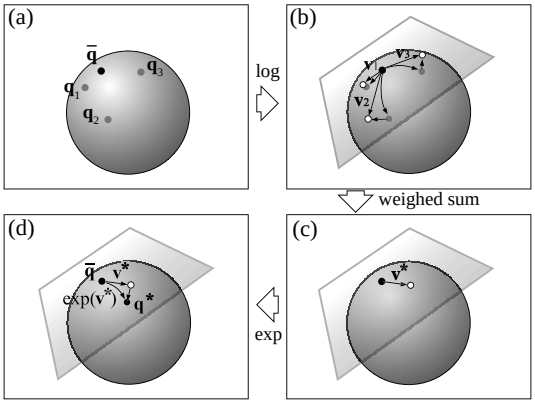


Figure 7: Unit quaternion blending.

$[0, \pi)$.

With the new distance metric, the reference orientation $\mathbf{q}_*$ can be found by minimizing the objective function:

$$E = \sum_{i=1}^{N_e} \sin^2 \theta_i, \tag{8}$$

where $\theta_i = \| \log(\mathbf{q}_*^{-1}\mathbf{q_i})\|$. Since $\sin^2 \theta_i = (1 - \cos^2 \theta_i)$ and $\cos \theta_i = \mathbf{q_i^T} \cdot \mathbf{q}_*$, we have

$$E = \sum_{i=1}^{N_e} \left(1 - (\mathbf{q_i^T} \cdot \mathbf{q}_*)^\mathbf{2}\right). \tag{9}$$

To find $\mathbf{q}_*$ that minimizes E subject to the unitariness constraint of the quaternion, we employ the lagrangian multiplier method:

$$\frac{\partial E}{\partial \mathbf{q}_*} = \lambda \frac{\partial C}{\partial \mathbf{q}_*}, \tag{10}$$

where $C = 1 - \|\mathbf{q}_*\|^2$ and λ is the lagrangian multiplier. Plugging Equation (9) into Equation (10), we have

$$\left(\sum_{i=1}^{N_e} \mathbf{q_i} \cdot \mathbf{q_i^T}\right) \mathbf{q}_* = \lambda \mathbf{q}_* \quad \text{or} \quad \mathbf{A}\mathbf{q}_* = \lambda \mathbf{q}_*, \tag{11}$$

where $\mathbf{q}_*$ is represented as a 4×1 vector, $\mathbf{A}$ is a 4×4 matrix, and λ is a real number. The problem of finding $\mathbf{q}_*$ in Equation (11) is a typical eigenvector problem. Since a 4×4 matrix has a maximum of four eigenvectors, we examine each solution for Equation(9) and choose the best one which minimizes the objective function E as given in Equation (9).

Figure 7 illustrates our unit quaternion blending scheme. Given the reference orientation $\mathbf{q}_*$ (Figure 7(a)), we transform each example orientation $\mathbf{q_i}$ into its corresponding displacement vector $\mathbf{v_i}$ through the logarithm map, that is, $\mathbf{v_i} = \log(\mathbf{q}_*^{-1}\mathbf{q_i})$ (Figure 7(b)). Then, we blend $\mathbf{v_i}$, $1 \leq i \leq N_e$

with respect to their weights w_i to obtain the displacement vector $\mathbf{v} = \sum_{\mathbf{i=1}}^{\mathbf{N_e}} \mathbf{w_i v_i}$, where N_e is the number of the example motions (Figure 7(c)). Finally, we compute the blended orientation $\mathbf{q}$ by transforming $\mathbf{v}$ back to the orientation space and apply it to $\mathbf{q}_*$, that is, $\mathbf{q} = \mathbf{q}_* \exp(\mathbf{v})$, (Figure 7(d)).

Now, we adapt the blended root position and orientation of a target character to a given trajectory. We first adjust the root position. Given a curved root trajectory on the floor, we compute the target root position, so that its projection on the floor is coincident with its trajectory while preserving its elevation. We then adjust the blended root orientation for the character to follow the trajectory. The direction for the character to move forward is obtained by blending the tangent vectors of the arcs that approximate the projected root trajectories of the example motions. We determine the target root orientation such that the forward direction coincides with the tangent vector of the given root trajectory: We rotate the root segment by the angular difference between the forward direction and the tangent vector of the given root trajectory about the unit normal vector of the floor.

4.2.4 Motion Retargeting

When the target character has a different size and proportion from the actual human performer, we are not able to simultaneously preserve the joint angles of the blended posture and its end-effector positions. Therefore, there may be artifacts such as foot sliding and penetration. To obtain a convincing locomotion of the target character, we need to adjust the blended posture in an online, real-time manner while keeping the characteristics of the original motion. For this purpose, we employ an importance-based approach for online computer puppetry introduced by Shin *et al*. [32].

To employ their approach, we have to provide a sequence of target foot positions as input data. We obtain them by blending the foot positions of the example motions. To do that, we first represent each of them in the local coordinate frame of its root segment. Then, a target foot position is obtained by blending those of the example motions with respect to their weights. Finally, we recover its corresponding target foot position in the global coordinate frame by applying the target root position and orientation to it.

A target foot position may vary from time to time in accordance with its weight change. Therefore, we force the target foot position to be fixed while the foot contacts the floor. The duration of its contact can be easily obtained from the keytimes. When the foot is approaching or contacting the floor, we change the joint angles to keep the foot position. Otherwise, we keep the joint angles to preserve the motion characteristics inherited from the example motions.

5 Experimental Results

Our crowd animation system is implemented in C++ on top of the Microsoft Windows 2000. Experiments are performed on an Intel Pentium PC (PIII 800 MHz processor and 512 MB memory) with commercially available motion clips. We use a human model of 40 DOFs: 6 DOFs for the pelvis position and orientation, 3 DOFs for the spine, 7 DOFs for each limb, and 3 DOFs for the neck. The

motion clips are sampled at the rate of 30 frames per second.

Our first experiment is for simulating human-like characters to walk around in the square while avoiding collision with the others. We approximate each of the characters by the bounding cylinder of radius 10 inches. The maximum norm of the acceleration is 50 to 100 inches/s^2. Figure 8 (a) shows the initial placements of the characters in the square. After simulating the characters for several seconds, we generate an external event for those in the upper-left corner of the square to direct a crowd motion such as an evacuation that all the members should move to the lower-right corner. The characters as drawn with red circles in Figure 8 (b) are received the external message. They distribute the message to others through local interactions. Figure 8 (c) shows a snapshot of the simulation. Eventually, all the members gather around the lower-right corner as shown in Figure 8 (d).

The second experiment exhibits complex behaviors of a crowd walking around a museum to see pictures on the wall as shown in Figure 9. Each character first lines up to enter the museum, and then moves from place to place guided by the directions on signposts while interacting with the others and the environment. The local billboard of a cell contains the information such as the members, the pictures, and the signposts lying in the cell. Each member stops in front of a picture for a while to see it and then moves to another until eventually leaving the museum.

Our third experiment is for simulating athletes competing in a long-distance race. Figure 10 (a) shows the race track on the ground and the initial placements of the runners. To start the race, we generate an external event corresponding to the starting signal. In modeling the behavioral rules for the runners, we take into account the competitive nature of the race as well as the level of fatigue due to their physical exercise. When a runner spurts abruptly at a full speed, an emotional event is generated to alert nearby runners so that they may not fall too far behind that runner. We also generate a temporal event to consider the fatigue of a runner accumulated during the race. The racer runs at a full speed until the temporal event occurs, and then gradually slows down the pace. Figure 10 (b) shows a snapshot of the race. To generate realistic motions of various speed and gyration, we take example motions such as “walking”, “jogging” and “running” with five different gyrations and four different speeds.

The final experiment is for a flock of birds flying along a corridor. Differently from the above experiments, each bird is approximated as its bounding sphere of radius 20 inches. The number of birds is 1000, the initial speed of the birds is 50 inches/s, and the maximum norm of acceleration is 50 to 100 inches/s^2. The corridor has the volume of $7200 \times 1000 \times 1000$ cubic inches with ten static columns within it. The radius of each column is 50 inches. Birds initially gather around one side of the corridor as shown in Figure 11(a), and move to the other side during the simulation. Figure 11(b) gives a snapshot of the simulation. It takes 3.0335 seconds on average for one second simulation of 1000 birds. To show the efficiency of our event-driven approach, we also simulate the same example using the fixed time step approach, which checks all pairs of the members to find the neighbors of each member at every time step. By varying the number of birds from 100 to 1000, we plot the simulation times for one second simulation as given in Figure 12. The rendering and motion generation times are

excluded to compare only the efficiency of collision detection. We can observe that the event-driven simulation outperforms the fixed time step simulation as expected.

6 Conclusion

We have presented an event-driven approach for crowd simulation. When an event occurs, the response of each member involved in that event is derived from the internal status based on a hierarchy of behavioral rules together with the information perceived from other members and the environment. To generate natural-looking motions of the members, we exploit the example motion clips. Given a vector of parameters, we synthesize its corresponding motion by blending those example motions, and then retarget the motion to the specific character model and situation. As demonstrated in experimental results, our approach can simulate, at an interactive rate, the behaviors of a crowd consisting of several hundred human-like characters, each of which has 40 degrees of freedom and is represented by about 1,000 polygons.

References

[1] AntZ, 1998. http://www.antz.com/technology.

[2] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. In *Computer Graphics (Proc. of SIGGRAPH '90)*, pages 19–28, 1990.

[3] D. C. Brogan and J. K. Hodgins. Group behaviors for systems with significant dynamics. *Autonomous Robots*, 4(1):137–153, 1997.

[4] D. C. Brogan, R. A. Metoyer, and J. K. Hodgins. Dynamically simulated characters in virtual environments. *IEEE Computer Graphics and Applications*, 15(5):58–69, 1998.

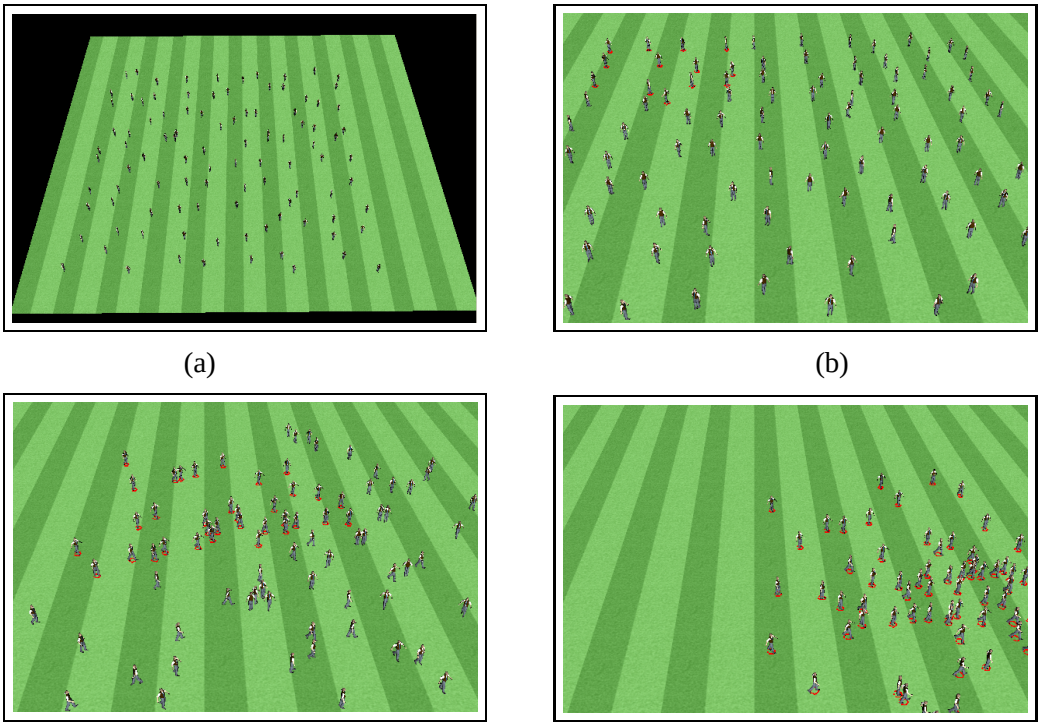
[5] A. Bruderlin and L. Williams. Motion siganl processing. *Computer Graphics (Proc. SIGGRAPH '95)*, 29:97–104, August 1995.

[6] S. R. Buss and J. P. Fillmore. Spherical averages and applicataions to spherical splines and interpolation. *ACM Transactions On Graphics*, 20(2):95–126, Apr. 2001.

[7] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 189–196, 1995.

[8] T. H. Cormen, C. H. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

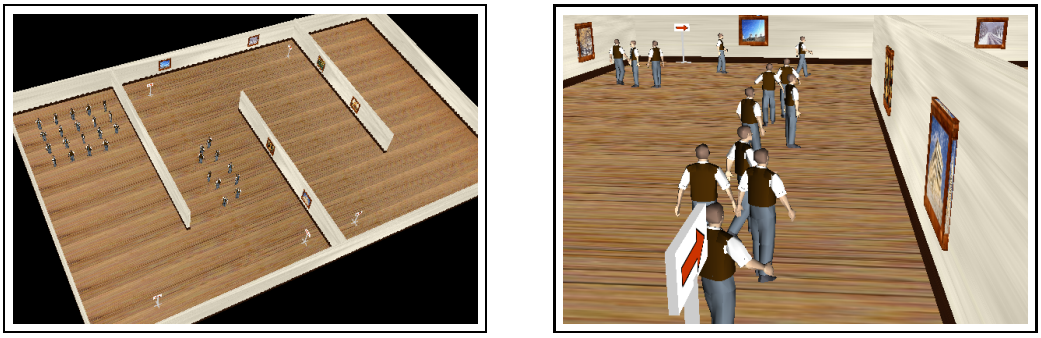
[9] Michael Gleicher. Retargeting motion to new characters. In *Computer Graphics (Proc. of SIGGRAPH '98)*, pages 33–42, 1998.



(a) (b)

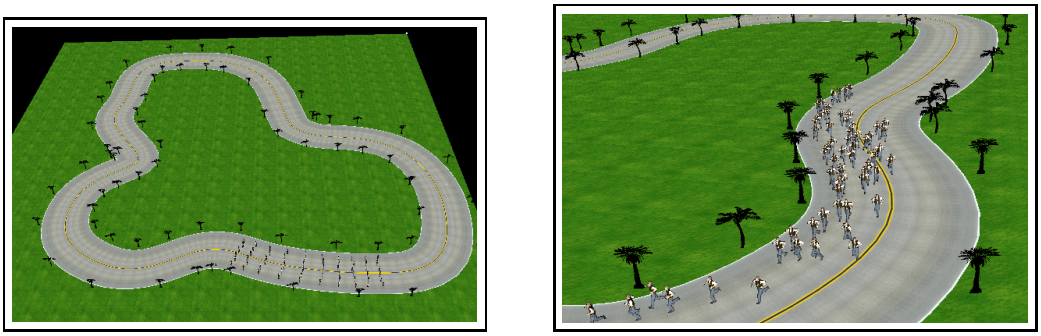
(c) (d)

Figure 8: Walking around in a square



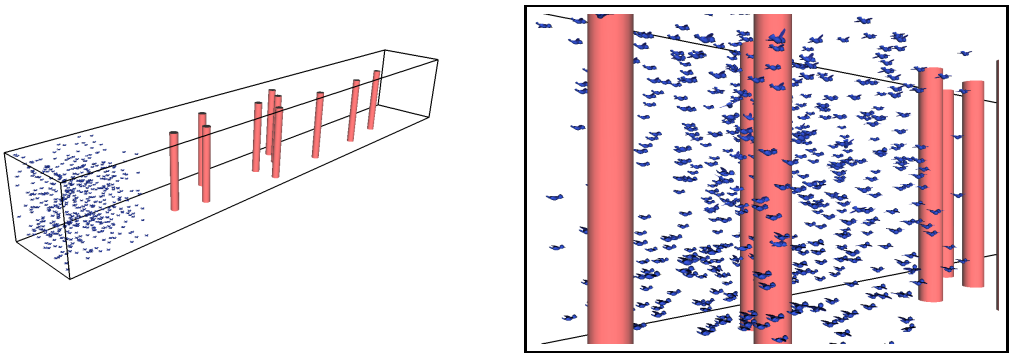
(a) (b)

Figure 9: The gallery in a museum



(a) (b)

Figure 10: A long-distance race



(a) (b)

Figure 11: A flock of free-flying birds

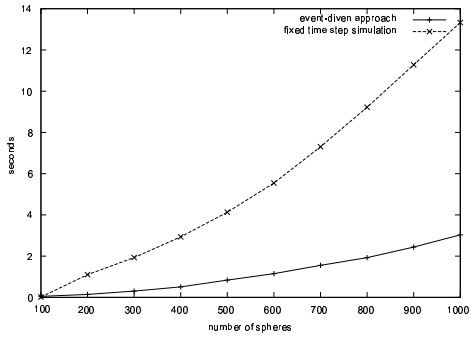


Figure 12: The actual processing time required to simulate birds for one second. The solid line is for fixed time step simulation, and the dotted line is for our event-driven simulation.

[10] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *Computer Graphics (Proc. of SIGGRAPH '96)*, pages 171–180, 1996.

[11] J. K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, August 1988.

[12] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, September 1995.

[13] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, July 1996.

[14] D. R. Jefferson. Virtual time. *ACM Tracsactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[15] D. Kim, L. J. Guibas, and S. Y. Shin. Fast collision detection among multiple moving spheres. *IEEE Transactions on Visualization and Computer Graphics*, 4(3):230–242, 1998.

[16] H. K. Kim, L. J. Guibas, and S. Y. Shin. Efficient collision detection among moving spheres with unknown trajectories. Technical Report CS/TR-2000-159, Dept. Electrical Engineering & Computer Science, Korea Advanced Institute of Science and Technology, 2000.

[17] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[18] J. Lee and S. Y. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Computer Graphics (Proc. of SIGGRAPH '99)*, pages 39–48, 1999.

[19] J. Lee and S. Y. Shin. General construction of time-domain filters for orientation data. *IEEE Transactions on Visualization and Computer Graphics*, 2001. To appear.

[20] V. J. Milenkovic and H. Schmidl. Optimization-based animation. In *Computer Graphics (Proc. of SIGGRAPH'2001)*, pages 37–46, 2001.

[21] B. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley, December 1996.

[22] B. Mirtich. Timewarp rigid body simulation. In *Computer Graphics (Proc. of SIGGRAPH'2000)*, pages 193–200, 2000.

[23] B. Mirtich. Efficient algorithms for two-phase collision detection. Technical report, TR-97-23, A Mitsubishi Electric Research Laboratory, Dec. 1997.

[24] M. Moore and J. Wilhelms. Collision detection and response for computer animation. In *Computer Graphics (Proc. of SIGGRAPH '88)*, pages 289–298, 1988.

[25] S. R. Musse and D. Thalmann. A model of human crowd behavior: Group inter-relationship and collision detection analysis. In *Computer Animation and Simulations '97, Proc. Eurographics workshop*, pages 39–51, 1997.

[26] S. R. Musse and D. Thalmann. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):152–164, 2001.

[27] K. Perlin and A. Goldberg. Improv: A system for scriping interactive actors in virtual worlds. In *Computer Graphics (Proc. of SIGGRAPH'96)*, pages 205–216, 1996.

[28] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics (Proc. of SIGGRAPH '87)*, pages 25–34, 1987.

[29] C. W. Reynolds. Steering behaviors for autonomous characters. In *Proc. of Game Developers Conference*, pages 181–188, 1999.

[30] C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics & Applications*, 18(5):32–40, 1998.

[31] C. Rose, B. Guenter, B. Bodenheimer, and M. F. Cohen. Efficient generation of motion transitions using spacetime constraints. *Computer Graphics (Proc. SIGGRAPH '96)*, 30:147–154, August 1996.

[32] H. J. Shin, J. Lee, M. Gleicher, and S. Y. Shin. Computer puppetry: An important-based approach. *ACM Transactions on Graphics*, 20(2):to appear, 2001.

[33] P. Sloan, C. F. Rose, and M. F. Cohen. Shape and animation by example. Technical Report MSR-TR-2000-79, Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, July 2000. Available at http://www.research.microsoft.com.

[34] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Computer Graphics (Proc. of SIGGRAPH'94)*, pages 43–50, 1994.

[35] G. Turk. Interactive collision detection for molecular graphics. Technical report, TR90-014, Dept. of Computer Science, Univ. of North Carolina at Chapel Hill, Jan. 1990.

[36] M. Unuma, K. Anjyo, and R. Takeuchi. Fourier principles for emotion-based human figure animation. In *Computer Graphics (Proc. of SIGGRAPH '95)*, pages 91–96, 1995.

[37] R. Webb and M. Gigante. Using dynamic bounding volume hierarchies to improve efficiency of rigid body simulation. *Visual Computing (Proceeding of Computer Graphics International)*, pages 825–842, 1992.

[38] D. J. Wiley and J. K. Hahn. Interpolation synthesis for articulated figure motion. In *Proc. of IEEE Virtual Reality Annual International Symposium '97*, pages 157–160, 1997.

[39] A. Witkin and Z. Popović. Motion warping. In *Computer Graphics (Proc. of SIGGRAPH '95)*, pages 105–108, 1995.