Eurographics/SIGGRAPH Symposium on Computer Animation (2003)
D. Breen, M. Lin (Editors)

Constrained Animation of Flocks

Matt Anderson, Eric McDaniel and Stephen Chenney†

University of Wisconsin – Madison

Abstract

Group behaviors are widely used in animation, yet it is difficult to impose hard constraints on their behavior. We describe a new technique for the generation of constrained group animations that improves on existing approaches in two ways: the agents in our simulations meet exact constraints at specific times, and our simulations retain the global properties present in unconstrained motion. Users can position constraints on agents' positions at any time in the animation, or constrain the entire group to meet center of mass or shape constraints. Animations are generated in a two stage process. The first step finds an initial set of trajectories that exactly meet the constraints, but which may violate the behavior rules. The second stage samples new animations that maintain the constraints while improving the motion with respect to the underlying behavioral model. We present a range of animations created with our system.

Keywords: *flocking, constraints, sampling, plausible simulation*

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

1. Introduction

Simulated group behaviors are widely used in animation, with many production tools available, such as Weta Digital's MASSIVE[11] and AI.implant[1], a plug-in for Maya. Groups typically consist of individual agents, each with a set of parameterized rules governing its behavior (originally described by Reynolds[21]). The global behavior emerges from the actions of the individuals, producing large-scale effects that are otherwise hard to model. However, emergent behavior makes it very difficult to place constraints on the final motion. In this paper we describe a new technique for generating constrained group animations.

There are many situations in which constrained group behaviors are necessary. In film, a director might script the path of a particular character within the group (for dramatic effect or as part of a story), while retaining the overall group motion. Advertisers might wish to use the shape or motion of the flock to convey a message. We present examples, among others, of an agent constrained to come from behind to win and an entire flock forming letters. In the past such scenes would be hand animated.

The simplest method for generating constrained groups is to key-frame the agent of interest then simulate the other agents while the constrained agent follows its path (Figure 1). The key-framed agent in this situation is unable to respond to the actions of the other agents. One of two cases generally results: occasionally, the key-framed agent finds itself leading the group, but most often the constrained agent is unable to influence the group enough to avoid breaking off, as in Figure 1.

An alternate approach adds a constraint enforcement behavior rule to the agents, such as the *path following* behavior described by Reynolds[22] or the planning-based behaviors of Bayazit et. al.[4] Additional rules can guide the flock in real-time, as in gaming applications, but they must also interact with other behavior rules, such as collision avoidance and flock coherence rules. A significant parameter tuning problem is thus introduced: the constraint behavior must be strong to meet the constraints, yet not so

† email: {manderso,chate,schenney}@cs.wisc.edu

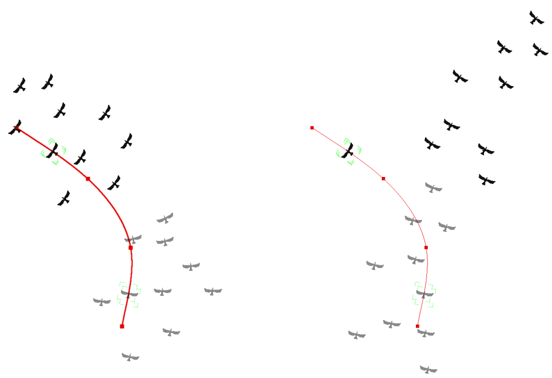

Figure 1: *A comparison between our system (left) and a naively constrained agent (right). The same flock is rendered in gray and black, representing earlier and later timesteps. In the naive case, the key-framed agent follows the path while the other agents follow their regular behavior rules. Despite the flock's ability to see the constrained agent, the result shown on the right is typical – the unconstrained agents fly off leaving the key-framed agent behind. The standard fix for this problem is a rule that tells the flock to follow the constrained agent, but that requires the constrained agent to lead. Our system solves for trajectories for all of the agents such that the constraints are satisfied and the global motion looks plausible, even if the constrained agent is embedded in the flock.*

strong that it overrides other, visually important behaviors. In an off-line production system, the tuning may need to be repeated for every constrained shot, resulting in inconsistency in the flock's motion across scenes. Duncan[11] describes the extensive behavior tweaking for flocking scenes in the *Lord of the Rings*, which ultimately required the addition of invisible geometry elements to guide agents. Our method works with existing, tuned models without modification to the rule-set or any other parameters.

Timing constraints pose particular difficulties for existing methods. Rules can guide a character along a path, but the speed at which it moves is governed by the interplay of many factors, such as the motion of flock-mates. Cuts between shots are typically used to address this problem, but they impose potentially undesirable artistic limitations on a director. For example, the movie *Spirit*[9], was hand animated to achieve both good control and a long, continuous shot. Furthermore, with existing approaches it remains very difficult to constrain multiple agents at the same time, enforce global constraints on all members of the flock (such as shape constraints), or enforce constraints on only part of the motion. Our system handles all of these cases.

Our major contribution is a method for imposing hard constraints on the paths of agents at specific times while retaining the global characteristics of an unconstrained flock. Our approach has several advantages for animators: it works with existing, tuned models; it can enforce timing constraints on long sequences without imposing cuts; and it offers new types of constraints not previously available (e.g. global shape constraints). We use a sampling approach to explore the space of animations that meet the constraints, and offer one or more to the user. Our algorithms are applicable to an off-line production environment.

2. Related Work

Many techniques have been proposed to generate animations that satisfy constraints and respect some underlying procedural model. Witkin and Kass[26] worked with articulated figures, and found joint torques over time that produced the desired motion while minimizing energy. This work, dubbed *Spacetime Constraints*, its extensions[8, 18], and similar work[2, 6, 14, 15], all use some form of optimization to find the "best" solution for the trajectories. They require derivatives of the objective function with respect to the control parameters, which are difficult to extract from flocking models. Specifically, the standard methods for perception and inter-agent forces introduce significant discontinuities. Ngo and Marks[19] used genetic algorithms in the absence of derivatives, and later extended their approach to collision intensive systems[23], while Popović et. al.[20] used derivatives where possible and a random search in other cases. However, these techniques do not scale well as the numbers of agents grows large.

Group behavior models generally don't have an optimal behavior. Randomness is frequently used to increase the variability in each agent's behavior, which improves the appearance of the motion. Any one of a large number of potential animations is thus considered acceptable, with no principled way to choose a single best one. Barzel, Hughes and Wood[3] noted that randomness makes many types of simulation appear more plausible, and that randomness can help to solve constrained problems. Following, Chenney and Forsyth[7] solved collision intensive constraint problems by sampling from a distribution describing plausible outcomes. We exploit the same ideas in our system. However, unlike previous approaches we explicitly enforce the constraints in a pre-sampling step, and then maintain them throughout the process.

3. Constraining Flocks

Our system works on group behavior models that produce agent trajectories and provide a way to evaluate the plausibility of a given set of trajectories. In our implementation we use a variant of Reynolds' flocking model[22] (Section 3.1) that produces the positions of agents over time, which in turn implicitly encodes the velocities and orientations.

We aim to produce plausible animations that satisfy constraints while retaining the realistic properties of the underlying behavior model. The flocking model we use incorporates a *wander* behavior rule[22] that adds a random acceleration to each agent on each discrete simulation timestep. Many group behavior models include such a random component to make the motion appear less mechanical. Random components provide a natural way to assess the plausibility of an animation[3, 7]: a plausible motion is one in which the random components used to generate the motion are reasonably distributed. Our technique does not *require* random components – we could use any measure of plausibility that can be expressed as an objective function.

Our implementation supports three types of constraints: specific agents constrained to pass through a location at a given time; the center-of-mass (COM) of the group constrained to a point at a time; and the members of the flock constrained to lie within a given shape at a specific time. Constrained animations are produced in a two stage process. The first stage enforces the constraints without explicit regard to the plausibility of the result (Section 3.3). It uses a combination of methods: *Forward simulation* that enforces a particular set of initial conditions and generates unconstrained forward motion; *Backward simulation* that generates a simulation that ends at particular positions; *Path transformations* that take group motion along one path and transform it to follow another path; and *Path blending* that interpolates the motion of agents between two paths. The system combines these methods to produce a set of trajectories that satisfy the constraints.

The path transformation and blending techniques do not take into account the behavior rules – they are purely geometric operations – and hence the resulting paths may not be plausible. The second stage of our solution method samples new animations to enforce plausibility requirements while maintaining the constraints (Section 3.4). A sampling process is used with the output from the first phase as the initial sample. The sampler repeatedly proposes a new animation based on the old one, evaluates its plausibility, and decides to accept it as the new sample or reject it and stay with the current sample. Animations that are more plausible than the current one will always be accepted, while poorer examples will occasionally be used. The process is similar to simulated annealing but without the temperature reduction component. After running the sampler until the average plausibility stabilizes, or for a fixed number of steps, the last sample is presented as a solution. Multiple samples could be kept to give a user several options, or multiple instances of the process could be run.

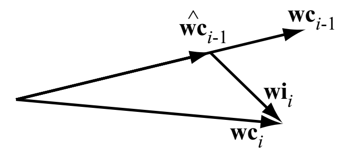

Figure 2: *The wander contribution for step i, $\mathbf{wc}_i$ is computed based on the contribution for the previous step and a randomly sampled wander impulse, $\mathbf{wi}_i$, by adding the impulse to the previous contribution (normalized first to prevent it growing too large, as recommended by Reynolds). Note that the wander impulse is randomly chosen on each timestep and is uncorrelated across steps, while each wander contribution is derived from the previous one and hence is correlated. Given a set of trajectories and the wander contributions, we can infer $\mathbf{wi}_i$ and use it to evaluate the plausibility of an animation (Section 3.4.2).*

The remainder of this section discusses each component of our algorithm in detail. Section 4 presents several animations produced with our system and discusses its limitations. We conclude in Section 5 with a look at potential extensions and improvements.

3.1. Behavior Model

Our underlying behavior model is based on Reynolds'[22] work. The internal state of each agent for each timestep consists of three vectors: its position, velocity and *wander contribution* (described below). For subsequent processing we keep only the position and wander contribution, as the velocity can be inferred from the sequence of positions.

Each agent has a set of rules that contribute acceleration requests on each timestep. The requests are combined through a weighted average and the result is applied as an impulse to the agent. The behavior of any agent at any timestep depends only on the positions of all the agents and obstacles at that timestep, and the wander contribution from the previous timestep. We use the following behavior rules:

Separation: Move away from your nearby neighbors.

Cohesion: Move toward the average position of your neighbors.

Alignment: Align your velocity with that of your neighbors.

Collision Avoidance: Steer away from impending collisions with static objects.

Speed Target: Try to attain a user defined speed.

Wander: Add a random *wander contribution* acceleration based on the previous timestep's wander contribution and a random vector (Figure 2). The wander contribution is designed to improve realism by adding random variations to the paths of the agents.

The weights used in the averaging process are defined as part of the behavior model and are a significant factor in determining the appearance of the flock. A user sets these weights in the absence of any constraints to get the desired flock appearance, independent of our algorithm, and our technique does not manipulate them further.

Behavior models in our system must support three operations: **Forward simulation:** The simulation of the behaviors forward from a set of initial conditions; **Plausibility Assessment:** Given a set of trajectories and a timestep, the model must be able to assess the plausibility of the state at the timestep; **Backward Simulation:** Simulation of the behaviors backward in time from a set of end conditions. In practice, we simulate forward using the regular behavior rules, but with sense of perception reversed so that agents see those behind them. The resulting sequence of states is then reversed. In effect, the agents react to those behind them while simulating, but after reversal they appear to be reacting to those in front, as they normally would. Our technique uses backward simulation in an intermediate step, so it is not essential that it be high quality motion. In Section 4 we discuss ways to remove the requirement for backward simulation.

Our constraint solution process views an animation as the set of trajectories of all the agents through the time interval of interest. For all our examples, these trajectories are the positions and wander components of the agents sampled at 10Hz from the simulator. Velocities are implied by the positions. It is common practice to sample behaviors at such frequencies, and it produces good results. The system can readily accommodate variations in sampling rate, with computational cost increasingly roughly linearly with the number of samples. Agents move in a piecewise linear manner between sample points, and are always aligned with their velocities.

In principle our approach could handle other components of agent state, such as explicit orientation, but we have not experimented with such cases. Furthermore, our approach should work with any group behavior model that supports the forward, backward and evaluation operations. We have, however, only experimented with the flocking model.

3.2. Constraints

Our system supports three types of constraints: *point constraints* for which a specific agent must pass through a given point at a given time; *center-of-mass (COM) constraints* in which the COM, or average position, of a subset of the flock must be at a specific point at a specific time; and *shape constraints* for which a subset of the flock must lie inside a polygonal shape. Constraints can be mixed in any sensible way. For instance, different agents can be constrained at different times, or one agent can have a series of points to pass through, effectively key-framed with flocking motion interpolation. The COM constraint mimics the traditional "fly to point" constraint, but with precise control over timing.

Velocities must be associated with each constraint to aid in the generation of an initial sample. A user explicitly provides these for shape constraints, while for point and COM constraints the velocities are inferred by grouping contiguous constraints and fitting a Bspline path. The arc length and time between constraints gives a velocity. Isolated point or COM constraints have their velocities provided by the user. Velocities are only of temporary use and are not enforced by the system as currently implemented, although they could be enforced with very little modification to our approach.

Point constraints are explicitly enforced when an initial sample solution is generated, as described in the next section, and then maintained throughout the sampling process. COM and shape constraints are precisely satisfied by the initial sample process, but in the subsequent sampling phase they are enforced through their contribution to the plausibility of the animation, as detailed in section 3.4.2.

3.3. Enforcing Constraints with an Initial Solution

With a set of constraints defined, the system finds a set of trajectories to use as the initial sample for the subsequent refinement process. Our aim with this phase is to create trajectories that exactly meet all the constraints. We would also like the resulting animations to be plausible, but that is not an over-riding concern – the sampling phase can fix problems.

We define a *window* to be an interval between two sequential constraints. Constraints must be satisfied only at the start and end of any window (Figure 3),

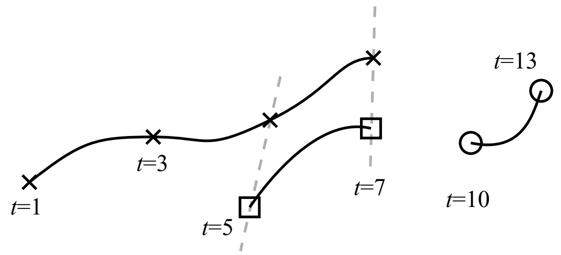

Figure 3: *The constrained interval is broken into windows for generating initial trajectories. The windows in this example are the intervals* $[1, 5]$*,* $[5, 7]$*,* $[7, 10]$ *and* $[10, 13]$*. Every constrained time defines a new window boundary, unless the constraint is part of a contiguous sequence of point or COM constraints for a particular set of agents (such as at* $t = 3$*). In this figure, the different symbols indicate different agents.*

except that a new window is not defined for point or COM constraints that are part of a contiguous set (our techniques meet these constraints when trajectories are found within a window). We then roll forward through the sequence of windows, generate an initial and final configuration for each window, and join them with plausible trajectories that are cheap to compute.

3.3.1. Generating Constrained Configurations

The initial flock configuration for the first window must satisfy all the constraints at that time and be a reasonable snapshot of the flock's motion. Likewise, the end configuration of the first window must also satisfy all the constraints and be plausible. For subsequent windows, the final configuration from the previous window provides a set of initial conditions that satisfy the constraints, while a satisfactory end configuration must still be created.

The process of finding a constrained configuration always begins with a snapshot of simulated motion, either the final configuration of a short unconstrained simulation or the outcome of a forward simulation from the start of a window. To ensure that all the constraints are satisfied, we then modify this snapshot according to the following rules:

- For each point constrained agent, the system applies a translation and rotation to move it to its target position and align its velocity with that implied by the constraint sequence. The transformations that were used are stored.
- If a subset of agents is COM constrained, the group is rigidly translated and rotated from the snapshot to make the COM meet the constraint and its velocity. We are also free to re-assign agents among the snapshot positions, because that will not change the COM. If we are finding a configuration for the end of a window, we sample a matching[10] that reduces the sum of squared distances between the location of each agent at the start of the window and its position at the end. This reduces the degree to which the agents must mix in getting from one constraint to another, which results in more plausible motion.
- If a subset of agents is shape constrained, we sample one point inside the shape for each agent. The samples are initially chosen uniformly at random within the shape, and then a few relaxation steps are performed on the point set to push very close points apart. We then do a re-assignment of agents to points as described for COM constraints.
- If the agent is unconstrained, the closest constrained agent in the snapshot is found, and that agent's transformation is applied. Alternately, we could choose a constrained agent at random; which option is better depends on the users preferences. Choosing the closest point produces smoother, but sometimes less interesting, results.

The result is a flock that tends to look plausible but is not assured to be so. If there is only one constrained agent, or a COM constraint, the result is generally reasonable because it represents a rigidly transformed version of an unconstrained forward simulated flock. Otherwise, the results depend on the number of constrained agents and the plausibility of their constrained positions, which presumably arise from authoring considerations and are out of the system's control.

3.3.2. Finding Initial Trajectories

The initial constraint enforcement process must find a set of trajectories within each window that meet the initial and final constraints for that interval. Depending on the constraints to be met, some or all of the following operations are performed:

Operation 1: Simulate forward to meet initial conditions. An unconstrained forward simulation is always run, starting with the initial conditions for the window, for the duration of the window.

Operation 2: Transform the forward simulated trajectories to meet point constraints. If the window is part of a sequence of point constraints on one or more agents, we fit a Bspline curve through the sequence and transform the result of Operation 1 such that the constrained agents follow the Bspline paths. The forward simulated path is time parameterized, as are the constrained agents' Bsplines. At each timestep the translation required to move each constrained agent from its forward simulated position onto its Bspline path position is computed and applied. Similarly, if the window is part of a sequence of COM constraints, we perform a path translation to move the COM of the flock onto the interpolated path. Unconstrained agents (with no point, COM or shape constraints) receive the translation for the agent to whom they were closest at the start of the window, or the COM translation. At the end of this step, all the agents with contiguous constraints that span the window are on their required trajectory, and the window is done if there are no additional constraints at the end of the window.

Operation 3: Simulate backward to meet end constraints. End constraints will not be satisfied by Operation 2 for agents that are shape constrained, or point or COM constrained only at the end of the window. The configuration generation technique of Section 3.3.1 is used to generate a final configuration for the window, but agents already in the correct position are left unchanged. A backward simulation is run from the resulting set of end conditions. The backward trajectory is path transformed as in Operation 2 to enforce any contiguous point and COM constraints. Finally, the positions of agents from the forward and backward simulations are blended to generate the final set of trajectories. The blend function we use is $\mathbf{x} = (1-u)\,\mathbf{x}_{forward} + u\mathbf{x}_{backward}$ with $u = -\left(2t^3 - 3t^2\right)$ for scaled time t running from 0 to 1 within the window. This blending function[17] retains the velocities from the forward trajectories at the start of the interval, and the velocities from the backward simulation at the end of the interval.

Cases occur in which no agent is part of a sequence of point constraints within the window, and hence there is no Bspline path to use as a reference path for path transformations. In this case we construct a Hermite curve that joins the center of masses for the initial and final conditions and has tangents aligned with the agents' average velocities. This curve, and the center of mass for the flock throughout the window are used to compute a translation that moves the center of mass of the flock along the Hermite curve joining the initial and final conditions.

After applying these operations in every window, the system has a set of constraint satisfying trajectories for all the agents throughout the entire constrained interval. The trajectories are joined to form one continuous set. They may not, however, be plausible: our methods for building them, including setting constrained configurations, translating agents and blending positions may result in collisions, tight loops or other implausible motions. The sampling phase, described next, refines the initial result to generate a plausible final animation.

3.4. Sampling Plausible Solutions

The trajectory generation process provides initial trajectories for all of the agents through the constrained interval. A sampling process, akin to Markov chain Monte Carlo or simulated annealing, is used to refine these paths to produce more plausible results. The high-level algorithm is:

generate initial trajectories, A_0
while not done
  propose candidate trajectories, A_c, based on A_i
  $u \leftarrow \text{random}(0,1)$
  if $u < P_{accept}$
    $A_{i+1} \leftarrow A_c$
  else
    $A_{i+1} \leftarrow A_i$

The sampling chain is run for either a fixed number of iterations or until the plausibility of the animations stabilizes. Two components are essential to the operation of the sampling process: the proposal mechanism that generates a candidate animation, A_c, based on the current one, A_i; and the acceptance criteria, P_{accept}, used to determine if the candidate is accepted.

We use an acceptance criteria motivated by the Markov chain Monte Carlo (MCMC) algorithm[13]:

$$P_{accept} = \frac{g(A_c)}{g(A_i)}$$

where $g(A)$ is an objective function evaluated on the animation, A. The criteria differs from that of MCMC in two ways: we do not require that the objective function g be a valid probability distribution, and we ignore the transition probability between states. The result is a sampling strategy that balances switching between different samples and favoring plausible animations, without getting stuck in local minima or accepting too many poor results. In the next section we discuss the methods for proposing new animations, while in Section 3.4.2 we describe the objective function $g(A)$.

3.4.1. Proposal Strategies

The system randomly chooses one of three strategies to propose candidate animations. Each has access to the current animation (the trajectories for every agent), the constraints, and the windows used to propose the initial sample.

Propose completely new trajectories: This strategy re-runs the process in Section 3.3 to generate entirely new trajectories within a subset of the constrained period. With 50% probability this proposal regenerates all of the trajectories using the procedure in Section 3.3. Otherwise, 1, 2 or 3 contiguous windows are chosen to be replaced, with uniform probability of generating any subset. When regenerating within a subset of the windows, additional, temporary constraints are placed at the ends of the subset to enforce continuity with neighboring windows (which are unchanged by the process). This proposal strategy is useful throughout the refinement process to encourage good mixing – the creation of significantly different animations – because it tends to propose animations that are locally different but not so bad as to be rejected. This strategy is chosen with probability 0.1.

Propose bumps: This strategy adds 1 to 3 randomly positioned and shaped bumps to the current trajectories. The bumps can be any length up to twice the length of the trajectory, although the random choice of bump size is biased toward those that cover 1 to 10 steps. The bumps are allowed to be longer than the trajectory, in which case the unused portion is thrown away. In that case, however, the endpoints of the trajectory will be changed. The center of the bump is placed at a timestep chosen at random, with the probability of choosing any point inversely proportional to its plausibility. Particularly poor sections of animation are therefore more likely to be modified. A specific agent's trajectory is chosen uniformly at random from among the agents, but agents for whom the bump would break constraints are excluded. This ensures that the point constraints are maintained.

The bump is added to the trajectory in a multi-resolution manner in order to have the bump "follow" the trajectory as it is added. We first define the bump in 3D in a plane that contains the x-axis, but is otherwise randomly oriented. The bump is parameterized by the distance along the x-axis, from its start at $x = 0$ to its end at $x = 1$. The distance from the axis is given by $d = 16s(x^2 - 2x^3 + x^4)$, which results in a curve with zero derivative at the points $x = 0$ and $x = 1$, as shown in Figure 4. The parameter s defines the size of the bump; it is samples from a normal distribution with mean 0 and standard deviation 0.05.

In our system the trajectory is sampled at discrete intervals, and the bump is sampled to give the same number of samples as the trajectory has over the required length of the bump. A multi-resolution decomposition is performed on both the bump and the piece of trajectory to be modified. The decomposition recursively down-samples a curve by a factor of two, replacing every second point with its offset from the midpoint of its neighbors (Figure 4). The offset coordinates, $\mathbf{o}_i$, for point i, $\mathbf{p}_i$, at level l is computed as follows (level 0 is the curve consisting only of the

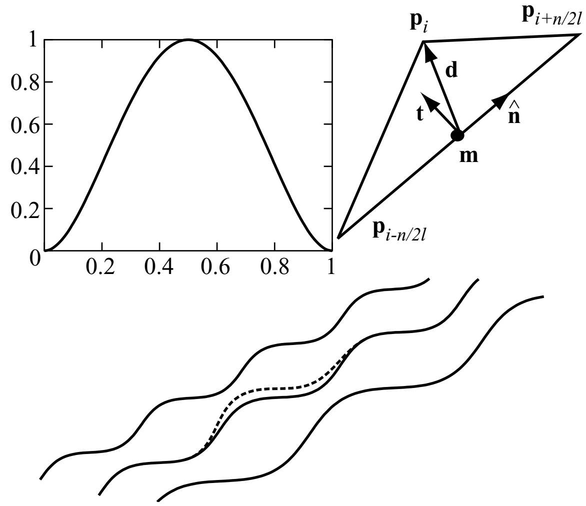

Figure 4: *One proposal strategy adds bumps to segments of an agent's trajectory. Top left: The bump is a quartic curve with zero value and derivative at its ends. We scale the vertical size of the bump by a normally distributed random amount with mean 0 and standard deviation 0.05. Top right: bumps are added in a multi-resolution manner. The position of a point is encoded according to its offset from its neighbors' midpoint, expressed in a coordinate frame determined by the neighbors' locations. See the text for details. Bottom: The result (dashed) of adding a bump of height 0.05 to a segment of curve (solid). Our multi-resolution strategy automatically accounts for the orientation off the curve segment and the scaling of the bump.*

endpoints of the original curve):

$$
\begin{aligned}
\mathbf{m} &= \frac{\mathbf{p}_{i+n/2^l} + \mathbf{p}_{i-n/2^l}}{2} \\
\mathbf{d} &= \mathbf{p}_i - \mathbf{m} \\
\mathbf{n} &= \mathbf{p}_{i+n/2^l} - \mathbf{p}_{i-n/2^l} \\
\mathbf{t} &= \mathbf{r} \times \hat{\mathbf{n}} \\
\mathbf{s} &= \hat{\mathbf{n}} \times \mathbf{t} \\
\mathbf{o}_i &= \left(\frac{1}{\|n\|} (\mathbf{d} \cdot \mathbf{n}), \frac{1}{\|n\|} (\mathbf{d} \cdot \mathbf{s}), \frac{1}{\|n\|} (\mathbf{d} \cdot \mathbf{t}) \right)
\end{aligned}
$$

The midpoint, $\mathbf{m}$, of the neighbors at level l is computed first, and then $\mathbf{p}_i$'s offset from it. A coordinate frame, $(\mathbf{n}, \mathbf{s}, \mathbf{t})$ is computed using the unit vector along the line joining $\mathbf{p}_i$'s neighbors, $\mathbf{n}$, and a global unit vector $\mathbf{r}$. The multi-resolution coordinates stored for $\mathbf{p}_i$ are the projections of its offset onto this coordinate frame, divided by the distance between the neighbors.

Our method is a modified version of a standard technique for multi-resolution decomposition of a curve, (producing results similar to those of Finkelstein and Salesin[12] and similar to the "standard" method mentioned in Guskov et. al.[16]) except that we work with 3D curves and have a scaling factor derived from the spacing of the samples. The move to 3D necessitated the use of the global vector **r**, which orients the reference frame about the line joining the neighbors. With 2D curves the orientation is implicit. There remains a potential degeneracy, if the line joining the neighbors is aligned with **r**, but we detect this case and use a different **r**. We have not found this to be a problem. The scaling of the coefficients by the distance between the neighbors automatically accounts for varying sample spacing in world coordinates, and also automatically scales the bump according to the length of the segment.

After decomposing both the bump and the segment of the trajectory, their multi-resolution coefficients are added. The new segment of trajectory is recomposed and put in place of the original segment. This proposal strategy is chosen with probability 0.6. It is significantly cheaper than the complete regeneration of trajectories, and is very effective at cleaning up local problems with a solution or mutating an already plausible solution.

Propose velocity changes: This proposal adjusts the speed with which some agents move along their trajectory. Qualitatively, the proposal increases the speed of an agent through one portion of its trajectory, then decreases it in a subsequent section, so that the agent travels the same overall distance and continuity of the trajectories is maintained. A procedure identical to that above is used for selecting trajectory segments to modify. The system computes the cumulative distance traveled by the agent at each sample point over the segment of interest, then adds an offset, o, to each distance given by $o = 16s(u^2 - 2u^3 + u^4)$, where u varies from 0 to 1 along the segment, and s is a scaling value chosen from a normal distribution with mean 0 and standard deviation of $0.025 \times r$, where r is the length of the segment to modify. The trajectory is re-sampled to give points spaced according to the modified distances. This strategy is chosen with probability 0.3, and its effects are like those of a bump addition: it is good for resolving local problems and mutating plausible solutions.

3.4.2. Evaluating Animations

The evaluation function takes the product over all timesteps in the constrained regions. Each timestep has terms for the plausibility of the wander impulses used, g_w, terms for any COM, g_c, or shape constraints, g_s, that may be active at that time, and a term that biases toward a single flock, g_f. Overall,

$$g(A) = \prod_{t\in[0,T]} g_w(A,t) \cdot g_c(A,t) \cdot g_s(A,t) \cdot g_f(A,t)$$

where the various component functions are defined in the following sections. Note that terms for the point constraints are not required because they are enforced in the initial trajectory generation step and the proposal mechanism maintains them.

The plausibility of a step in the flocking model is evaluated based on the wander contribution implied by the trajectories on that step. In a regular forward simulation, the wander contribution for step i, $\mathbf{wc}_i$ is computed based on the previous wander vector and a randomly sampled wander impulse, $\mathbf{wi}_i$ (Figure 2). This contribution is included in a weighted sum of all the behavior contributions to compute the instantaneous change in velocity of the agent at that timestep. The other behaviors depend only on all the agents' positions and velocities at the timestep in question.

The sampling of wander impulses in a forward simulation produces plausible results. If the wander impulses implied by a set of constrained trajectories look like they were sampled from the same distribution as those sampled in the forward case, we can say that the trajectories represent a plausible animation. When given a set of trajectories and timestep to evaluate, the system computes the total impulse applied at the timestep (the difference in velocities from before and after the step). Treating the trajectories as the outcome of a forward simulation, this impulse is the weighted average of the wander contribution and the other behaviors. From the agents' positions we compute the contribution due to the other behaviors. The discrepancy between that and the total observed impulse is the wander contribution that would be used by a forward simulation to generate the trajectories. If these wanders look like they were sampled correctly, the animation is plausible.

To evaluate the function g_w, we compute the wander contribution from both the current and previous step, and hence the wander impulse for the current step $\mathbf{wi}_i$ (see Figure 2). We define g_w to be the probability of seeing the complete set of $\mathbf{wi}_i$s for an entire simulation. In the flocking model we use, the wander impulses are vectors in a uniformly random direction with a normally distributed length with mean 0 and standard deviation set by the user, typically to a value of $\sigma_w = 0.125$. Hence,

$$g_w(A,t) = \prod_i \frac{1}{\sigma_w\sqrt{2\pi}} e^{-\|\mathbf{wi}_i(A,t)\|/2\sigma_w^2}$$

where i ranges over the agents in the system.

The COM constraints are enforced with the follow-

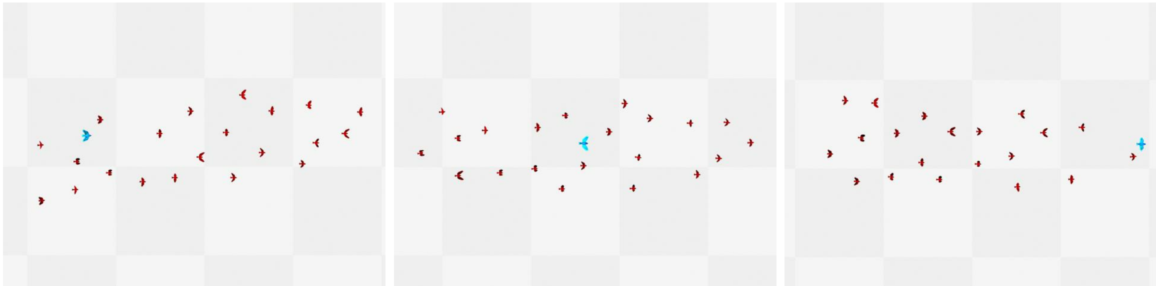

Figure 5: *Three frames from an example in which an agent (larger and brighter) is constrained to come from behind to win a race. Constraints are placed at the start and end time of the race. At the start time, the winning agent is point constrained to be behind the rest of the flock, which is COM constrained. At the end time, the agent is point constrained in front of the other COM constrained agents.*

ing term:

$$g_c(A,t) = \prod_{\mathbf{C}\in\mathcal{COM}(t)} \frac{1}{\sigma_{com}\sqrt{2\pi}} e^{-\|\mathbf{COM}(A,t)-\mathbf{C}_x\|/2\sigma_{com}^2}$$

where $\mathbf{C}$ is a constraint requiring the COM to be at $\mathbf{C}_x$, $\mathcal{COM}(t)$ is the set of COM constraints active at the timestep, $\mathbf{COM}(\mathbf{A},\mathbf{t})$ is the COM of the group at time t, and σ_{com} is a user defined parameter (set to 0.1 for all our examples).

Shape constraints contribute a term that depends on the sum of squared distances of each agent from its nearest point in the shape.

$$g_s(A,t) = \prod_{\mathbf{S}\in\mathcal{SHAPES}(t)} e^{-c_S d(\mathbf{S},A,t)}$$

where $\mathbf{S}$ is a shape, $\mathcal{SHAPES}(t)$ is the set of shape constraints active at the timestep, and c_S is a user defined constant, set to 10 in all our examples. Our implementation currently only supports 2D shapes defined as a set of triangles. This makes it simple to find distances to the shape. For 3D we could constrain the agents to fill a volume (with no change to our algorithm) or to form a shape when projected onto the image plane.

Under our plausibility model, a group of completely isolated agents are every bit as valid as a single flock, provided their wander impulses are reasonable. To keep the flock together, we employ a term that favors a particular number of flocks, where a flock is a set of agents with the property that every agent can see some other agent in the same flock, but all of the agents in one flock are unable to see any agent in another flock. The term is:

$$g_f(A,t) = e^{-c_F|F-F_c|}$$

where F is the number of flocks at time t, F_c is the preferred number as defined by a user, and c_F is a constant, set to 1000 for our examples. In all our examples we requested a single flock.

The system is relatively insensitive to the values of the constants c_S, c_F and σ_{com}. Values that range over an order of magnitude appear to perform equally well.

4. Examples

The simplest example for our system is a single constrained agent (Figure 1). The process for proposing initial samples does well in this case, and the sampling produces a range of variations. It is difficult to determine which agent is constrained unless the constraint is explicitly visualized. Figure 5 is an example of mixing position and COM constraints, in which an agent is constrained to be behind the flock at the beginning and in front by the end. This example highlights some of the problems with using rules to achieve constraints. The “winning” agent could be started at the back and given a higher target speed, but the parameters of the speed rule need to be balanced against the other flocking rules (particularly the *alignment* rule that attempts to match velocities within the flock). The rule balancing process is typically very time consuming and would need to be repeated if the constraint was changed, whereas our method meets the goals without any modification to the behaviors. Most of our examples are in 2D for easier visualization, but the system works equally well in 3D (Figure 6).

Constraints in the presence of static obstacles propose particular challenges for our system, because we completely ignore them in the initial trajectory generation procedure and correct them in the sampling phase. Experiments indicated that, when used to generate initial trajectories, obstacles introduced deviations that were inappropriately shifted to new locations by the path translation operation. If obstacles are ignored, the initial trajectories frequently contain

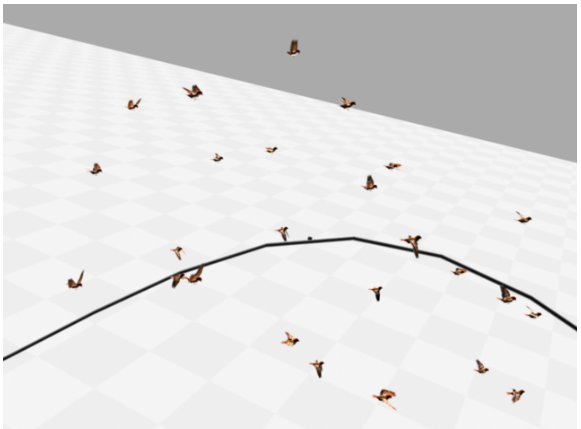

Figure 6: *Our system also operates in 3D, as illustrated by this frame from an animation in which the flock follows a looping path.*

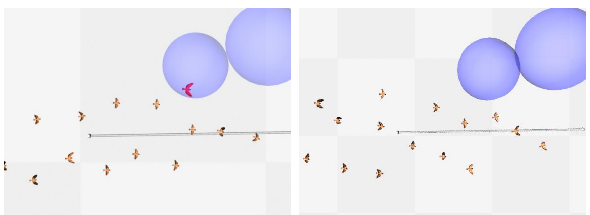

Figure 7: *An example in which the flock is forced, via center of mass constraints, to fly very close to a static obstacle. The left frame shows the initial paths generated by the system. The obstacles are ignored because we found they introduced artifacts in the motion. The subsequent sampling process removes the penetrations resulting in a plausible animation (right).*

agents that pass through obstacles, and the sampling process must make more iterations to correct the problems. The example in Figure 7 demonstrates static obstacles combined with COM constraints. This sequence is 5 seconds long and required 200 sampling iterations to generate the solution shown. This took about 10 seconds on a current generation PC.

The examples in Figure 8 demonstrate the effect of using our system with different flocking parameters, and hence different measures of plausibility. The flock on the left uses our standard parameters. The one on the right places a heavier weight on the cohesion behavior, causing the agents to flock much more tightly and circulate within the flock more rapidly. Results from identical constraint sets, with two constrained agents, are also shown. Note the constrained results resembles the unconstrained flocks. Each constrained example from Figure 8 took a few minutes to compute, and the sampling process stabilized around a plausible animation after about 800 iterations.

Cases containing multiple agents with point constraints are more difficult than single constrained cases because our initial trajectory generation process translates unconstrained agents according to only one of the constrained agents. This produces motion that initially is not very plausible. The sampling process generally manages to find solutions despite the limitations of the initial step, as Figure 8 illustrates.

The greatest limitation of our system is the process for generating initial trajectories. It could be improved by including more behavioral information into the process. The current system only reflects behavioral effects if they survive the path translation and blending operations, which they frequently do not. One promising approach is to set the trajectories of agents one at a time, using information obtained from the behavior rules to place each agent relative to those already positioned. Alternatively, additional behavioral rules could be used in the initial constraint generation process and then removed for the sampling phase. This would also remove the requirement for backward simulation and path blending.

It is possible for a user to specify constraints that a flock cannot reasonably meet. Our system will still enforce them, but the motion may not appear plausible. The spacing of constraints causes the most difficulties, either because agents may be forced to move too fast or slow, or because the flock may have insufficient time to re-arrange itself between constraints. Our interface for setting constraints provides hints for the spacing of constraints based on speed requirements. Otherwise we have found the cause of problematic motion to be visually obvious, and a user could adjust the constraints if that were a concern.

Our system produces solutions in seconds for small examples, up to hours for long samples that need to run for many sampling iterations. The running time of any one iteration is dominated by the time to simulate and evaluate the flock, which in turn is dominated by nearest neighbors computations. In our current implementation this scales quadratically, and we found that for the flock sizes we used (up to 100 agents) the use of spatial data structures did not improve the situation because most agents can see each other. Otherwise, the running time scales roughly linearly with the length of the constrained sequence, but it is hard to estimate up front the run time required for a particular constraint set. Our experience indicates that scenarios with multiple constrained agents and static obstacles are the most difficult to solve. Multiple instances of our system will produce different results, making it a good

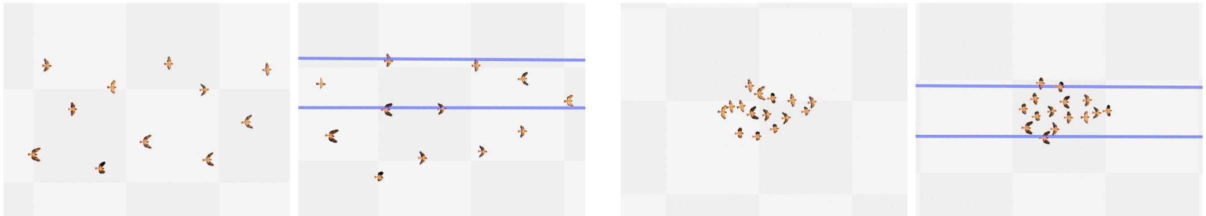

Figure 8: *Our approach retains the overall properties of the original behavior model. The two left images are of our standard flock, with the leftmost unconstrained and the center-left one required to satisfy simultaneous point constraints on two agents. On the right are corresponding snapshots of an insect-like flock, in which agents circulate more rapidly and fly closer together. The far-right flock was required to satisfy the same constraints as the center-left flock, but notice that it retains the insect-like properties.*

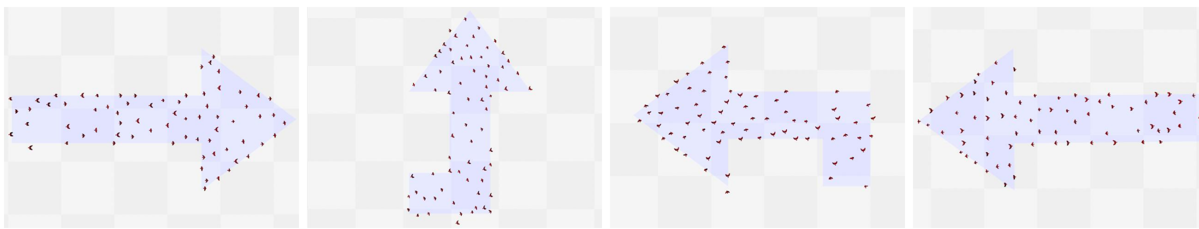

Figure 9: *An example of shape constraints solved with our system, in which 50 agents in a flock form each arrow in turn. The initial trajectory generation procedure is very effective for shape constraints. This example was run for only 1000 sampling iterations (taking two hours).*

candidate for cluster computing. Several jobs can be sent out and the results perused for the best solutions.

5. Conclusion

Our system succeeds at producing constrained flocking motion for a wide variety of examples. Unlike previous systems, we enforce hard position constraints, we specify the times at which constraints must be met, and we generate solutions that respect the underlying behavioral model. While it sometimes produces poor results, one or two additional constraints were sufficient to improve the performance of our system when needed, and it was always clear how to place them. Furthermore, the system can produce many candidate solutions, from which a user can select the most appropriate. Overall we explored a problem of significant practical importance not comprehensively addressed previously in the literature.

We would like to make stronger claims that the animations produced by our system are representative of the underlying model. We could do this if we guaranteed that our sampling phase produced animations distributed identically to those of the unconstrained case. Markov chain Monte Carlo sampling would enable us to make such claims, but our current proposal strategies preclude this. In particular, the addition of multi-resolution bumps is not an easy operation to invert, and hence it is hard to compute the reverse transition probabilities required for MCMC[13]. Alternate proposal schemes could be designed, but those that we considered (adding bumps in world coordinates, for instance) would not produce proposals as good as our current choices.

While we have demonstrated our system for a flocking model, we are interested in testing its performance on other behavioral models. In principle, we require a simulator that implements only a few functions (Section 3.1), of which the ability to simulate backward is probably the most limiting. Of particular interest are the dynamically accurate models of Brogan and Hodgins[5] or Tu and Terzopoulos[25], or behavioral models based on particle system, such as those used for *Star Wars: Episode 1*[24].

Acknowledgments

This work was funded in part by NSF grant CCR-0204372. We appreciated the comments and suggestions of those who reviewed this paper.

References

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

2. Ronan Barzel and Alan H. Barr. A Modeling System Based on Dynamic Constraints. In *Computer Graphics (SIGGRAPH 88 Conf. Proc.)*, volume 22, pages 179–188, August 1988.

3. Ronan Barzel, John F. Hughes, and Daniel N. Wood. Plausible motion simulation for computer graphics animation. In *Computer Animation and Simulation '96*, pages 184–197, 1996. Proceedings of the Eurographics Workshop in Poitiers, France, August 31-September 1, 1996.

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. David Brogan and Jessica Hodgins. Group behaviors for systems with significant dynamics. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 528–534, 1995.

6. Lynne Shapiro Brotman and Arun N. Netravali. Motion interpolation by optimal control. *Computer Graphics (SIGGRAPH 88 Conference Proceedings)*, 22(4):309–315, August 1988.

7. Stephen Chenney and D.A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *SIGGRAPH 2000 Conference Proceedings*, pages 219–228. ACM SIGGRAPH, July 2000.

8. Michael F. Cohen. Interactive spacetime control for animation. In *Computer Graphics: Proceedings of SIGGRAPH 92*, volume 26(2), pages 293–302, 1992.

9. Doug Cooper. Challenges of the homeland pan in "Spirit", 2002. SIGGRAPH 2002 Conference Abstracts and Applications, page 156.

10. Frank Dellaert. *Monte Carol EM for Data-Association and its Applications in Computer Vision*. PhD thesis, Carnegie Mellon University, 2001.

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

12. Adam Finkelstein and David H. Salesin. Multiresolution curves. In *Computer Graphics: Proceedings of SIGGRAPH 94*, pages 261–268, 1994.

13. Walter R Gilks, Sylvia Richardson, and David J Spiegelhalter. *Markov chain Monte Carlo in Practice*. Chapman & Hall, 1996.

14. Michael Gleicher. Motion editing with spacetime constraints. In *Proceedings 1997 Symposium on Interactive 3D Graphics*, pages 139–148, April 1997. Providence, RI, April 27-30.

15. Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Computer Graphics, Proceedings of SIGGRAPH 98*, pages 9–20. ACM SIGGRAPH, 1998.

16. Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schrder. Normal meshes. In *SIGGRAPH 2000 Conference Proceedings*, pages 95–102, 2000.

17. Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 473–482, 2002.

18. Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. Hierarchical spacetime control. In *Computer Graphics: Proceedings of SIGGRAPH 94*, pages 35–42, 1994.

19. J. Thomas Ngo and Joe Marks. Spacetime constraints revisited. In *Computer Graphics: Proceedings of SIGGRAPH 93*, pages 343–350, 1993.

20. Jovan Popović, Steven Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive Manipulation of Rigid Body Simulations. In *SIGGRAPH 2000 Conference Proceedings*. ACM SIGGRAPH, July 2000. 209–218.

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

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

23. Diane Tang, J. Thomas Ngo, and Joe Marks. N-body spacetime constraints. *The Journal of Visualization and Computer Animation*, 6:143–154, 1995.

24. Marjolaine Tremblay and Hiromi Ono. Multiple creatures choreograhy on Star Wars: Episode I "The Phantom Menace". SIGGRAPH 99 Animation Sketch. In Conference Abstracts and Applications, page 205, August 1999.

25. Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Computer Graphics: Proceedings of SIGGRAPH 94*, pages 43–50. ACM SIGGRAPH, 1994.

26. Andrew Witkin and Michael Kass. Spacetime constraints. In *Computer Graphics: Proceedings of SIGGRAPH 88*, pages 159–168, 1988. Atlanta, Georgia, August 1-5.