Generating Freestyle Group Formations in Agent-Based Crowd Simulations

Qin Gu and Zhigang Deng * University of Houston

An interactive, scalable framework generates freestyle group formations and transitions via natural and flexible sketching interaction. It computes a plausible agent distribution in the target formation and agent correspondences between keyframes. Two-level formation trajectory control lets users intuitively guide agents' transition paths from the initial formation to the target formation.

In most crowd simulation systems, each agent intelligently moves toward its destination through navigational pathfinding algorithms and avoids collisions with other agents and obstacles through local behavior control models. However, research has paid little attention to simulating the collective behaviors of the entire crowd or a certain subgroup of the crowd. In many scenarios, these collective behaviors play a significant role in exhibiting the crowd's mass attributes. From the perspective of visual simulation, a group's collective behaviors are usually demonstrated as their general formations. For instance, when a computer game simulates a large-scale battle, the two army's formations directly reflect their controllers' strategic commands.

When simulating a crowd, one intuitive way to form a target formation is to provide each agent's desired position at a particular moment[1] and generate transitions between that position and the destination. However, users must manually specify many spatial, temporal, and correspondence constraints, which is time-consuming and nontrivial, particularly when the crowd includes many agents that change location frequently. Similarly, when a group of people performs a collective action simultaneously in the real world, it's generally impossible for their commander or team leaders to convey detailed movement information such as every group member's position at every time instance.

We propose an interactive, scalable framework to simulate freestyle group formation through intuitive user interfaces (see Figure 1). Specifically, each agent in a group first evaluates its optimal destination in a user-specified target formation constrained by formation patterns. Then, the framework employs *formation coordinates*, a relative-agent-position representation, to evaluate the final agent distribution in the target formation and the correspondence between the initial and target states. During the runtime transition from the initial formation to the target formation, each agent will move along a path guaranteed to avoid local collisions, follow user-sketched transition paths, and reach the target formation position.

This research provides two main contributions. The first is automated generation of freestyle group formations. Using a set of free-form user inputs, our approach automatically evaluates the corresponding target agent distribution with relatively few parameter tunings. The formation coordinates enable our approach to robustly handle any variation of formation shape, orientation, and scale.

The second contribution is agent motion trajectory control at both the local and global levels. This is especially useful when users try to simulate strategic scenarios such as "a military squad is forming a circle formation while bypassing any enemies or static obstacles."

 Published by the IEEE Computer Society

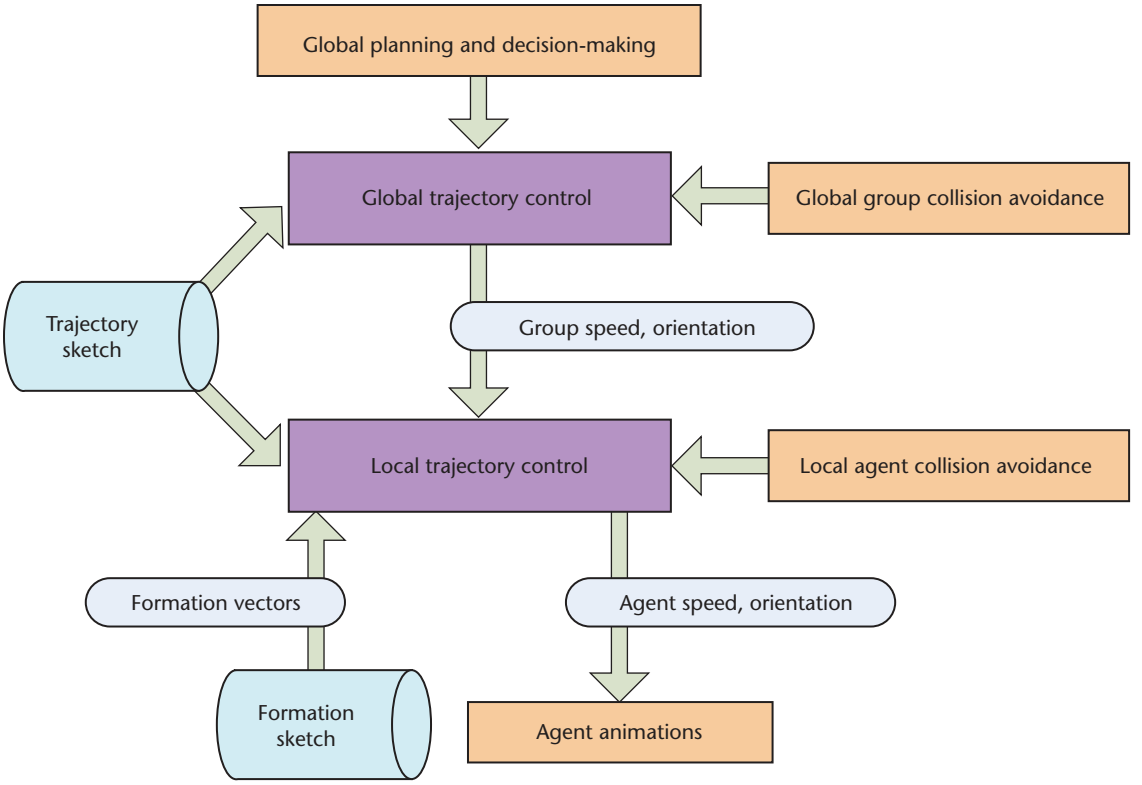


Figure 1. Embedding group formation control into a complete crowd simulation system. Orange indicates the components of a traditional crowd simulation system; blue and purple indicate user interactions and formation control, respectively, in our framework.

Target Formation Specification and Generation

Our group formation specification and evaluation efficiently convert an intuitive user input or description of a group formation into a numerical representation of agent distribution. A group formation's realism derives from the common observation that a group member in a real-world crowd is less likely to move across the formation to reach a far-off position if a much closer one is reachable. This is because humans tend to save energy unless special constraints exist, which is beyond this article's scope.

Formation Shape Representation

Interfaces for generating group formations should be intuitive enough for users to easily depict any formation without significant parameter tuning or prior mathematical knowledge. Meanwhile, they should be accurate enough for the crowd simulation algorithm to understand the users' purpose. In reality, a group formation's most salient visual feature is generally its contour shape, such as a square or circle. However, directly specifying and quantifying a geometric shape is nontrivial for most users.

To simplify this task, we propose a unified formation shape representation called a *formation template*—an oversampled point space (relative to the number of agents) with a roughly even distribution. Our approach converts the initial user input to the formation template; in this way, various types of user input can share the rest of the simulation pipeline. Our framework currently can automatically convert three common types of user input: *brush painting*, *texture maps*, and *boundary sketches*.

Brush painting. Two-dimensional image-processing tools such as Adobe Photoshop commonly use brush-painting input. Branislav Ulicny and his colleagues proposed an efficient brush tool to author desired crowd specifications.[2] To generate a unified representation, we employ a 2D painting's pixel intensities as a convenient reference to evaluate its target agent distribution. In the discrete simulated crowd, we sample a position as a template point only if its pixel intensity matches the brush's color within a certain threshold. Brush painting (see Figure 2a) works well for specifying simple formation shapes, which often require only a few painting components (for example, brush sizes).

Texture maps. For complicated formations, brush painting might be inaccurate and time-consuming. Instead, we can treat the target shape as a 2D

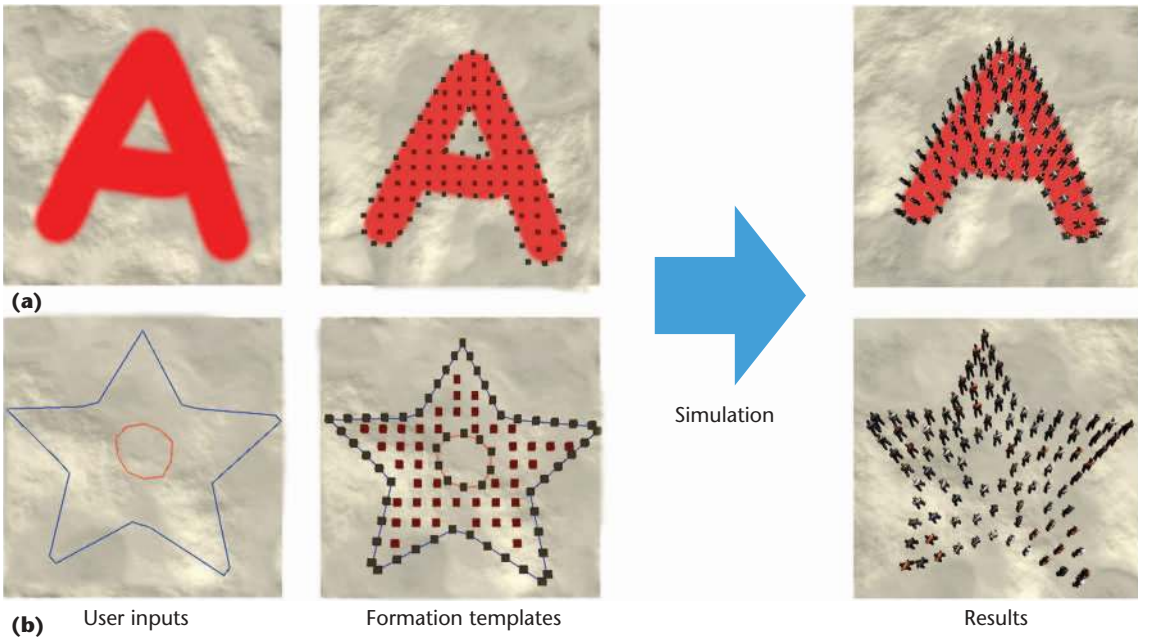


Figure 2. Generating unified formation templates with (a) brush painting and (b) boundary sketches. A formation template is an oversampled point space (relative to the number of agents) with a roughly even distribution.

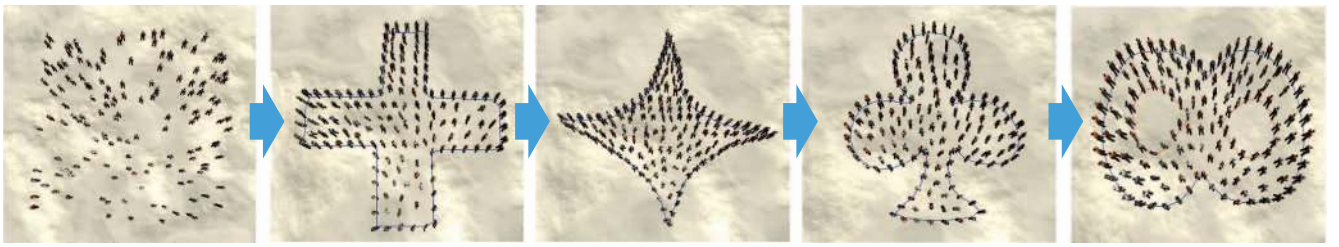


Figure 3. Generating freestyle group formations through inclusive curves (the blue boundaries) and exclusive curves (the red boundaries). The final formation has holes, which would otherwise be tricky to generate.

texture ground and directly retrieve the shape information by matching the points in the texture coordinates with those in the crowd simulation world coordinates.

Boundary sketches. The previous two types of user input provide fast solutions for sampling relatively simple or predefined formation shapes. However, in many scenarios, users might want to specify complicated formations on the fly. Brush painting can't specify subtle formation details such as sharp angles or continuous changes of the stroke size. Texture maps require the target formation to be well defined in advance and cannot be changed on the fly. Furthermore, both input types depend on color intensity information during sampling. If the environment has similar color information on the virtual ground, the sampling results would be inaccurate.

To complement those two input types, we developed a freehand-sketching interface using two types of boundary curves—*inclusive curves* and *exclusive curves*. With them, users can specify arbitrary target formations such as the examples in Figures 2b and 3.

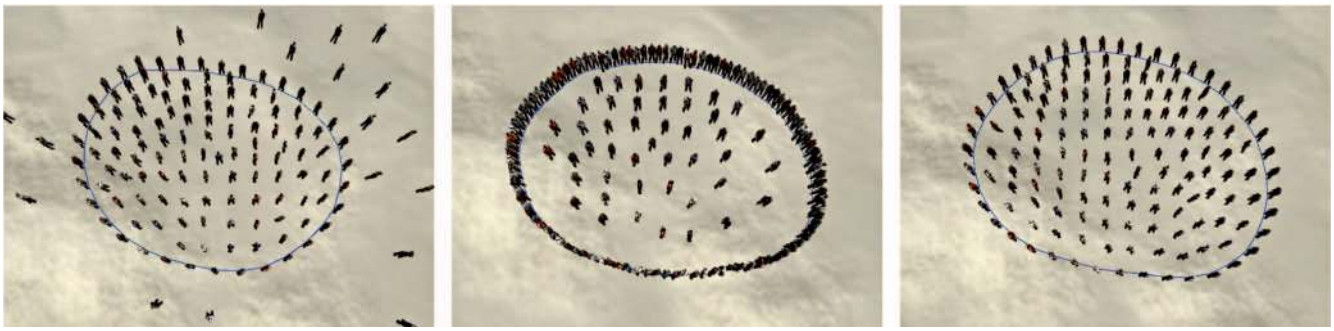
The initial user sketches are freehand curves or line stripes that might contain irregular point (shape vertex) distributions. To generate a visually balanced target formation template, we first evenly resample (that is, reevaluate) the points on the boundaries. We then fill the area between the inclusive and exclusive boundaries through an extended scanline flood-fill algorithm (see Figure 4), employing the same sampling unit used for boundaries. We adapted our filling algorithm to a discrete grid space so that its computation is faster than the traditional pixel-by-pixel sampling that most image-processing tools use.

Figure 4 is an extension of the traditional flood-fill algorithm in a continuous (pixel by pixel) space, such as the bucket-fill tool used in 2D painting programs. When checking the connected neighbors of each sampling point in the template, we use the `PointInBoundary()` function to

```
Input: bPoints, boundary points.
Input: unitX, a sampling unit to traverse the template.
Input: unitY, a sampling unit to traverse the template.
Output: fPoints, outputted filled points inside the boundaries.

 1: q = new empty queue;
 2: q.push(startingPoint);
 3: while q is not empty do
 4:   checkPoint = q.front();
 5:   left = checkPoint - unitX;
 6:   while PointInBoundary(left) && left has not been checked do
 7:     fPoints.push(left);
 8:     top = left + unitY;
 9:     if PointInBoundary(top) && top has not been checked then
10:       q.push(top);
11:     end if
12:     bottom = left - unitY;
13:     if PointInBoundary(bottom) && bottom has not been checked then
14:       q.push(bottom);
15:     end if
16:     left = checkPoint - unitX;
17:   end while
18:   perform the same steps for all right neighbors of the checkpoint ...
19:   queue.pop(checkPoint);
20: end while
21: return fPoints;
```

Figure 4. A discrete flood-fill algorithm. We adapted this algorithm to a discrete grid space so that its computation is faster than the traditional pixel-by-pixel sampling that most image-processing tools use.



(a) **(b)** **(c)**

Figure 5. Sampling-unit optimization. (a) An overly large sampling unit gives the template fewer points than agents. (b) An overly small sampling unit clusters too many agents on the boundaries. (c) Our adaptive sampling algorithm evaluates the unit for an optimal configuration.

check whether a point is inside the polygon formed by the boundary points. We could evaluate the point in several ways. We choose the angle sum solution because it is robust in 3D space. Furthermore, to avoid sampling points too close to the boundaries, `PointInBoundary()` checks four points with constant offsets (top, bottom, left, and right) to the current checkpoint.

This algorithm's most critical parameter is the sampling unit used for the boundaries and inside areas. An overly large sampling unit can reduce the computational load but might produce an unbalanced agent distribution in the target formation (see Figure 5a). On the other hand, an overly small sampling unit might break the balance between boundary and nonboundary agents (see Figure 5b). This is because the algorithm evaluates the boundary agents before the nonboundary agents (we discuss this in more detail later). A high density of boundary agents could lead to too few agents inside the formation shape.

To relieve users from repeatedly trying different sampling rates, we introduce an automatic algorithm to identify a balanced, optimal sampling unit (see Figure 5c). This algorithm is based on the observation that for any 2D polygon, if the number of vertices is large enough, the desired ratio between the boundary length and the area's

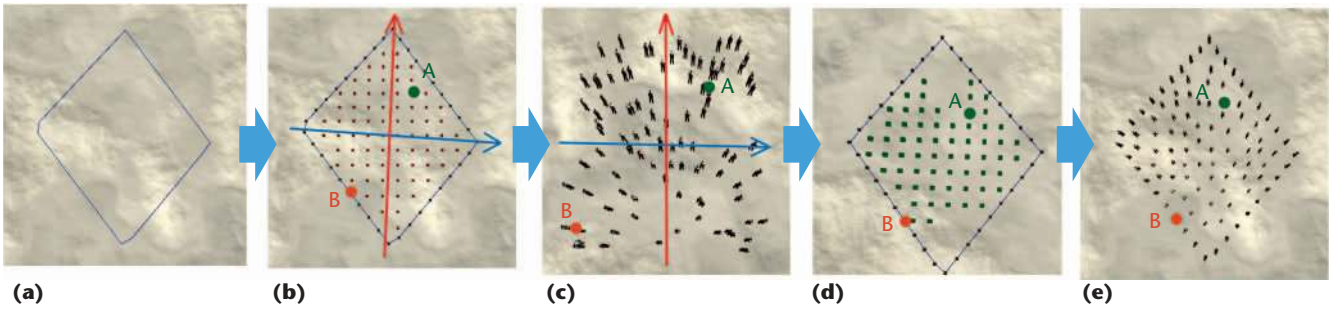


Figure 6. Evaluating the positions of agents A and B in the target formation. (a) Interactive user input and specifications. (b) Converting all the points in the formation template to formation coordinates. (c) Computing the formation coordinates for A and B in the initial agent distribution. (d) Finding the corresponding template points for A and B with the most similar formation coordinates. (e) Performing point-based relaxation to generate the final distribution in the target formation.

square root is approximately a constant. However, most numerical solutions for computing the area of an irregular (convex or concave) polygon, such as triangulation methods, aren't efficient enough.

To compute the optimal sampling unit, we start with a relatively small unit such as 100, as opposed to the world grid unit of 1,000. Then, we evaluate r, the ratio between the boundary and nonboundary agents, by counting the boundary points and inside points. Once we estimate r, the optimal sampling unit for generating the formation template is the ratio between the total boundary curve length and the number of boundary agents b. To obtain b, we solve

$$b^2 + r^2 \times b - r^2 \times n = 0,$$

where n is the total number of group agents.

Formation Coordinates

After creating the formation template, we must evaluate the best candidate position for each agent. However, it's nontrivial to find any deterministic relationship between the world coordinates of the original agent positions and the template point positions. This complexity is due to the target formations' large variety of shapes, positions, orientations, and scales. So, we need a comprehensive yet compact representation to correlate the original and target formations. Formation coordinates fulfill this function, translating the world coordinates to a scalable formation representation.

To eliminate the effect of entire group movements during formation generation, we first evaluate the weighted group center position of all the agents, p_c, using

$$p_c = \frac{1}{m}\sum_{i=1}^{m} \omega_i p_i,$$

where m is the number of agents, ω_i is the ith agent's weight, and p_i is the ith agent's world position.

Then, we compute two relative features and one absolute feature from each agent and each candidate point in the template (see Figure 6). First, we compute the *agent's direction to the formation center*: n_x, n_y. Both n_x and n_y range from 0 to 1. Any agent in a formation needs to know its relative direction with respect to the current group center, instead of the relative direction to the origin of the world coordinate system. We compute this by normalizing the direction vector from the group center to the evaluating point.

Second, we compute the *agent's relative distance to the formation center*: *dist*, which is between 0 and 1. Again, this feature must be relative to ensure the formation coordinate's scalability. We normalize the absolute distance between the current point to the group center by the formation shape's radius (that is, the distance from the evaluating point to a boundary point). However, this radius might vary significantly for different boundary points. Also, recomputing the exact radius for every point is impractical. Instead, we use only certain representative formation radii, and we choose the closest radius when performing normalization. In our experiments, evenly splitting the formation shape to eight sectors by four vectors provided visually acceptable results and performance.

Finally, we compute the *absolute formation orientation*: θ_x, θ_y. The previous two relative features enclose each agent's general position with respect to the group center, assuming the whole group always keeps its orientation. For example, an agent in a group formation's southwest region will normally always be in that region. However, this isn't always valid when the whole group changes its orientation. For example, if the entire group's orientation changes to 180 degrees according to the world horizontal axis, the same agent's absolute direction will also rotate. So, from the world coordinate system's perspective, that agent is now in the group formation's northeast area (see Figure 6).

Therefore, the formation coordinate includes θ_x, θ_y. By default, this feature will be the same for all the agents in one group to keep the agent distribution consistent. If we need to simulate inconsistent crowd scenarios such as a turbulence transition (for example, when agents can't maintain their adjacency condition), we can intentionally set this feature to random values for different agents.

Given a point or agent A or B in the world space (see Figure 6), we can finally formulate the formation coordinate as a 5D feature vector (n_x, n_y, *dist*, θ_x, θ_y). Unlike the global formation orientation, the formation coordinate doesn't include the whole group's global movement because the global movement doesn't affect the group formation's shape and local distribution. We'll directly apply the formation coordinate to every agent during the higher-level simulation (we discuss this in more detail later).

Estimating the Target Distribution

We estimate the agent distribution in the target formation to find the correspondence between any agent in the initial formation and its appropriate candidate position in the formation template. Using formation coordinates, we design a correspondence construction algorithm based on two key heuristics (assumptions).

First, in the target formation, boundary agents should closely fit the boundary curves to clearly exhibit the user-specified formation shape. This is particularly important to user perception because, as we mentioned before, the contour shape is the most salient feature distinguishing one formation from another in crowd simulations.

Second, each nonboundary agent should keep its adjacency condition as much as possible. So, our algorithm first finds correspondences for the boundary agents and then finds correspondences for the nonboundary agents.

Finding correspondences for the boundary agents. To guarantee the exact formation shape, every boundary point in the formation template requires exactly one corresponding agent to fit in. So, this must be the first step of correspondence-finding. After converting the positions of all the agents in the initial distribution into formation coordinates, we subtract the formation orientation (θ_x, θ_y) from each agent's relative direction (n_x, n_y) to yield the relative agent direction. We store this direction along with the relative agent distance in a KD-tree data structure. For each point on the target formation template boundaries, we similarly convert its world coordinate to its formation coordinate. We can efficiently compute the agent corresponding to each boundary point by finding the nearest neighbor in the KD-tree (for example, agent B in Figure 6).

Finding correspondences for the nonboundary agents. Because the formation template is an oversampled candidate or point space, we shouldn't expect strict one-to-one correspondences between the inner template points and the remaining agents.

Performance generally is critical for agent-based crowd simulations because the system must update every agent individually at each time step.

Instead of finding the corresponding agent for each inner template point, we inversely identify the corresponding template point for each nonboundary agent that wasn't selected in the previous step. Similarly, we use each agent's formation coordinate to find the closest inner template point. We further transform that point to its world coordinate representation—that is, the agent's target position (see agent A in Figure 6).

Formation Relaxation and Customization

As we described earlier, the resultant nonboundary agent distribution might not be naturally distributed because the number of inner template points might be significantly larger than the number of nonboundary agents. In addition, the inner template points are orthogonally sampled, which looks less natural to humans.

To improve the naturalness of the resultant agent distribution or formation, we apply point-based relaxation optimization to the generated initial agent distribution (see Figure 7). This process uses the boundary points as relaxation constraints. In our experiments, a small number of iterations (for example, 3 to 5) usually produced acceptable results.

To perform the relaxation, Figure 7 uses a Gaussian function as the kernel of radial basis functions. If we used a KD-tree to store the agents' positions, the `updateDistribution()` function would require re-creation operations, costing $\Theta(n \log n)$. Performance generally is critical for agent-based crowd simulations because the system must update every agent individually at each time step. Our implementation reduces $n \log n$ to n by registering agents into a regular grid system at each

```
Input: bPoints, constraint boundary points.
Input: iPoints, points to be relaxed.
Input: η, relaxation iterations.
Input: β, Gaussian constant parameter (β > 0).
Input: speed, moving rate of relaxation in each iteration.
Input: w, relaxation weight.
Output: rPoints, outputted relaxed points inside the boundaries.

 1: rPoints = iPoints;
 2: for each η do
 3:   allPoints = rPoints + bPoints;
 4:   updateDistribution(allPoints);
 5:   for each rPoint in rPoints do
 6:     neighbors = closestPoints(rPoint);
 7:     for each neighbor in neighbors do
 8:       d = distance(rPoint, neighbor);
 9:       v = v + d × exp(-β × d²);
10:     end for
11:     rPoint = rPoint + v/numOfNeighbors × speed × w;
12:   end for
13: end for
14: return rPoints;
```

Figure 7. Point-based formation relaxation. In our experiments, a small number of iterations (for example, 3 to 5) usually produced acceptable results.

update. The `closestPoints()` function simply collects the agents registered in surrounding grids with $\Theta(1)$ cost.

The weight parameter, *w*, determines the relaxation effectiveness on each agent. By default, we apply a uniform weight to all the agents, which means every agent has the same pushing or pulling effect on its neighbors. This results in an even agent distribution in the target formation. However, some applications might prefer an unbalanced agent distribution to achieve certain special effects such as local clustering. So, users can lower the relaxation weights for those agents at the clustering center so that they'll have a smaller pushing effect on their neighbors.

Agent Motion Trajectory Control

Our two-level agent motion trajectory control comprises *local formation transition control* and *global formation trajectory control*. The first level enables all the agents to reach their proper target positions with different moving styles while forming the target formation. The other level guides the agent group as a whole to follow arbitrary user-sketched paths.

Local Formation Transition Control

Local formation transition is the transition from one formation to another without considering the whole group's general locomotion. Many factors can affect it. The most intuitive way to control this transition would be to simply compute a linear interpolation from an agent's initial position to its estimated target position. However, this process can't reflect user control or realistic collision avoidance among agents. So, we further consider two factors that influence agent movements: local force-based[3,4] collision avoidance and sketch-based local trajectory control.

We employ force-based collision avoidance because it can handle very-high-density crowds. Each agent is pushed away by the tangential forces from its neighbor agents and environmental obstacles in the area of interest (that is, the human visible range). For extremely high-density crowds, the tangential forces alone might appear insufficient to drive agents apart. If the simulation system detects a collision between two agents, it applies extra repulsion force to immediately stop the agents to prevent unrealistic penetrations.

Without user interactions, each agent would go straight to the target position with minor transition adjustments on the way to avoid local collisions with other agents. However, many scenarios expect agents to follow a more sophisticated path during the local formation transition, such as a splitting and merging transition. Similarly to the formation specification interface, our local formation transition control lets users draw freehand curves to specify one or multiple moving paths as general guides (see Figure 8).

Unlike many previous approaches, we construct a second virtual local grid field (instead of regular world grids) to evaluate a flow vector to guide transitions. This is because the formation transi-

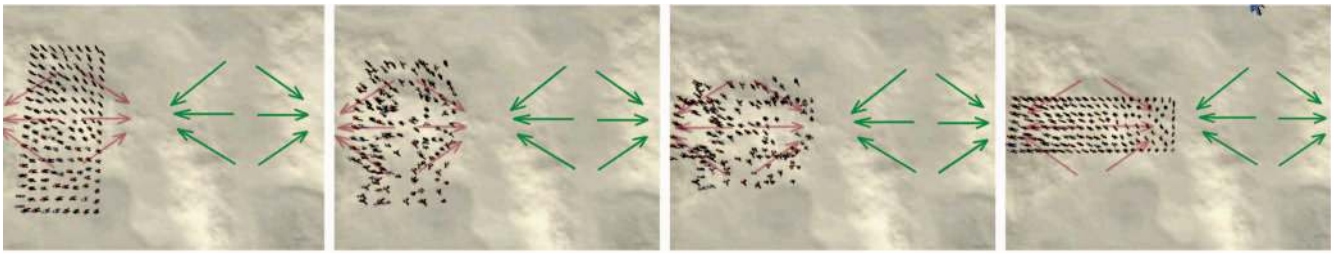


Figure 8. Local formation transition control. Users can specify relative local transition trajectories (the green arrows) anywhere in the scenario. Our approach can automatically snap the user specifications to align the actual formation location overlapping with the group center (see the red arrows).

tions are local behaviors despite group positions. So, even if users draw curves farther from the absolute group location, our algorithm will still consider these actions as local guidance to each agent. In other words, our algorithm will internally snap the center of user curves to the center of the virtual grid field overlapping with the transition area (see Figure 8).

All the virtual grids around the curves are filled with a flow vector. We set the flow vector's direction to the tangential vector of the closest point on the trajectory curve, and the flow vector's magnitude is inversely proportional to the distance between the grid and that point. If multiple curves exist, we blend the grid's final flow vector from all the influencing trajectory points with an experimental threshold, D, as the upper-limit distance for qualified influencing trajectories.

Finally, we compute each agent's local velocity:

$$V_{\text{local}} = w_1 V_f + w_2 \sum_{i=1}^{n-1} f_i + w_3 \sum_{j=1}^{9} s_j \, .$$

The first term is the velocity driven purely by target formation. The second term is the summed velocity driven by the combination of tangential forces from the agent's neighboring group members and obstacles.[4] The third term is the summed velocity driven by averaging flow vectors in the current and eight adjacent virtual grids (that is, $\{s_j\}_{j=1}^{9}$) derived from user sketches. The weights balance the expected speed of formation generation (w_1), the expected accuracy of collision avoidance (w_2), and the expected influence of user interference (w_3). Our experiments provided visually plausible results with $w_1 = 0.2$, $w_2 = 0.5$, and $w_3 = 0.3$.

Global Formation Trajectory Control

So far, we've assumed that all the agents in a group move and change formations relative to a fixed group center. However, many crowd simulation scenarios expect a group of agents to form a certain formation while moving as a whole to other locations. Furthermore, users can guide this global movement.

To achieve such effects, we introduce three factors to the group level:

$$V_{\text{global}} = w_4 V_{\text{gn}} + w_5 V_{\text{gc}} + w_6 V_{\text{gs}} \, .$$

V_{gn} is the global navigation vector heading to the target formation's location. V_{gc} is the velocity driven by global collision avoidance between different groups. V_{gs} is the user-guided velocity computed from our sketching interface (see Figure 9). Our simulations used $w_4 = 0.4$, $w_5 = 0.4$, and $w_6 = 0.2$.

Instead of using virtual grids, this form of control directly applies the user sketches of global group trajectories on the regular world grids to compute flow vectors. This is intuitive because the global trajectories are sensitive to the current group location. If a drawing curve is too far from the current group position, it won't affect the group's trajectory.

Although this global curve is more visually intuitive, it might introduce an undesired situation. If a user sketch starts close to the initial group location but heads away from the target formation location along the sketched path, the group will most likely stick at the end of the sketched curve. This is because the magnitude of V_{gs} is inversely proportional to the distance between the current group location and the closest point on the sketched curve.

To tackle this issue, we introduce a valve parameter α to V_{gs}:

$$V_{\text{gs}} = \alpha \times \sum_{j=1}^{9} s_j, 0 < \alpha < 1 \, .$$

When a group moves toward the end of a sketched curve, the valve value will keep decreasing so that V_{gn} will gradually dominate the group's path to guarantee the group reaches the target position.

At the group level, each group in the scene is considered a single entity. Any rule- or force-based model can serve as the collision controller, just as in a single-agent simulation.

Finally, we combine the obtained global velocity with the local velocity to deliver the final agent velocity.

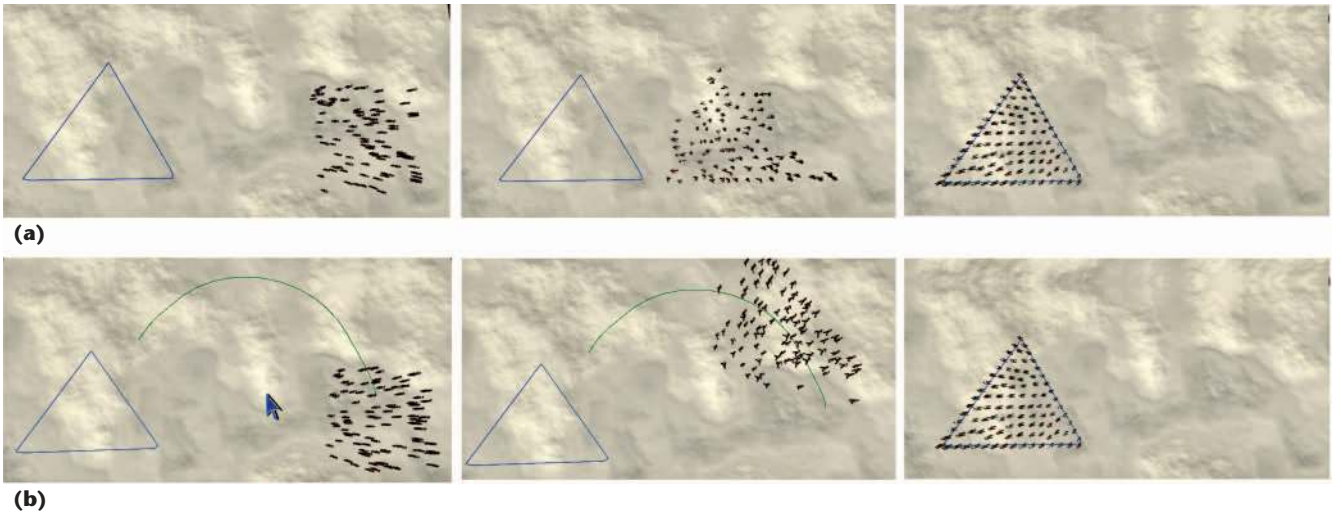


Figure 9. A group of agents reaching a target formation (a) without and (b) with explicit transition control. Without explicit control, each agent takes the optimal path to reach its target position. With explicit control, the user-sketched curves guide the whole group to take a specific route while also changing the formation.

Table 1. Runtime performance for a HiDAC (High-Density Autonomous Crowds) simulation model with and without our approach.

Crowd size (no. of agents)	Random crowd (fps)	HiDAC (fps)	HiDAC with our approach (fps)	Overhead (%)
100	202.3	162.0	158.9	2.2
200	156.2	113.1	109.2	3.3
300	96.8	54.1	51.2	4.3
400	52.3	35.4	32.0	5.9
500	38.5	29.4	27.1	7.1

Figure 9 compares simulations with and without global formation trajectory control. The initial and target formations are the same in both simulations. Our trajectory control lets us fully control the intermediate transitions without changing the final formation state.

Our trajectory control has two main advantages:

- Splitting it into the local and global levels gives users flexible, intuitive control over the transition paths.
- Maintaining the group's collective feature during the formation transition allows all the group members to share the same global navigator.

Although our approach computes collision avoidance twice for each agent and each group, the number of groups in a simulation is typically much smaller than the number of the agents. So, the additional global collision avoidance won't noticeably affect the overall runtime performance.

Performance and Results

We demonstrated several applications using our approach. A force-based HiDAC (High-Density Autonomous Crowds) simulation model[4] handled local and global collision avoidance. We used a regular PC with a 3.0-GHz CPU, 4 Gbytes of memory, and an Nvidia GFX 260 GPU. Using 3D human models (700 to 800 polygons) driven by high-quality motion capture primitives with 30 joints (62 degrees of freedom), we simulated a crowd of up to 500 agents at 30 fps.

To determine the computational overhead, we compared the average frame rates with and without our approach. Table 1 shows that our approach adds negligible overhead to the original crowd simulation system. For animation results, see the video at www.youtube.com/uhcgim.

Local Formation Control

In these experiments, agents were driven by only their local formation velocities and local collision avoidance. When each simulation started, we computed the group center of the initial agent positions and fixed the target group formation center at the same position.

These experiments used three sets of examples. Figure 10 shows a group of 100 autonomous agents forming the letter A, as specified by our brush-painting interface. In a separate experiment, our texture map interface used a real picture of the

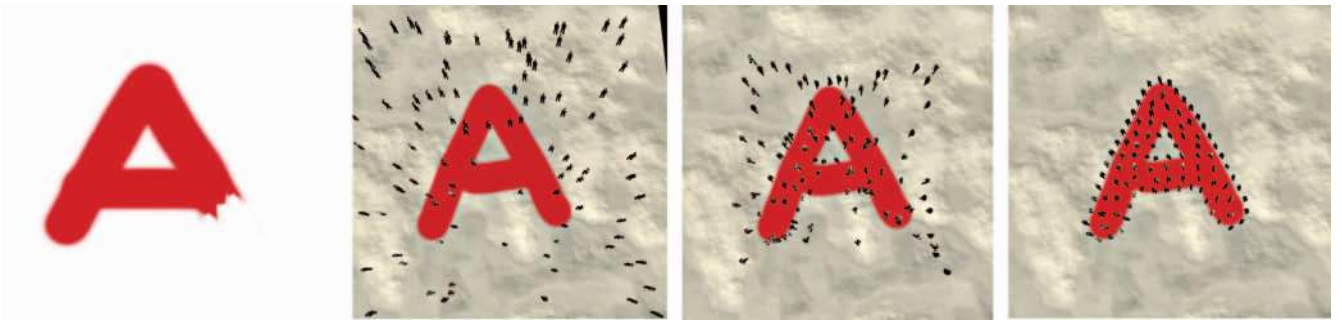


Figure 10. Using the brush-painting interface. (a) The user input. (b) The initial state. (c) The simulation. (d) The results. The group comprised 100 autonomous agents.

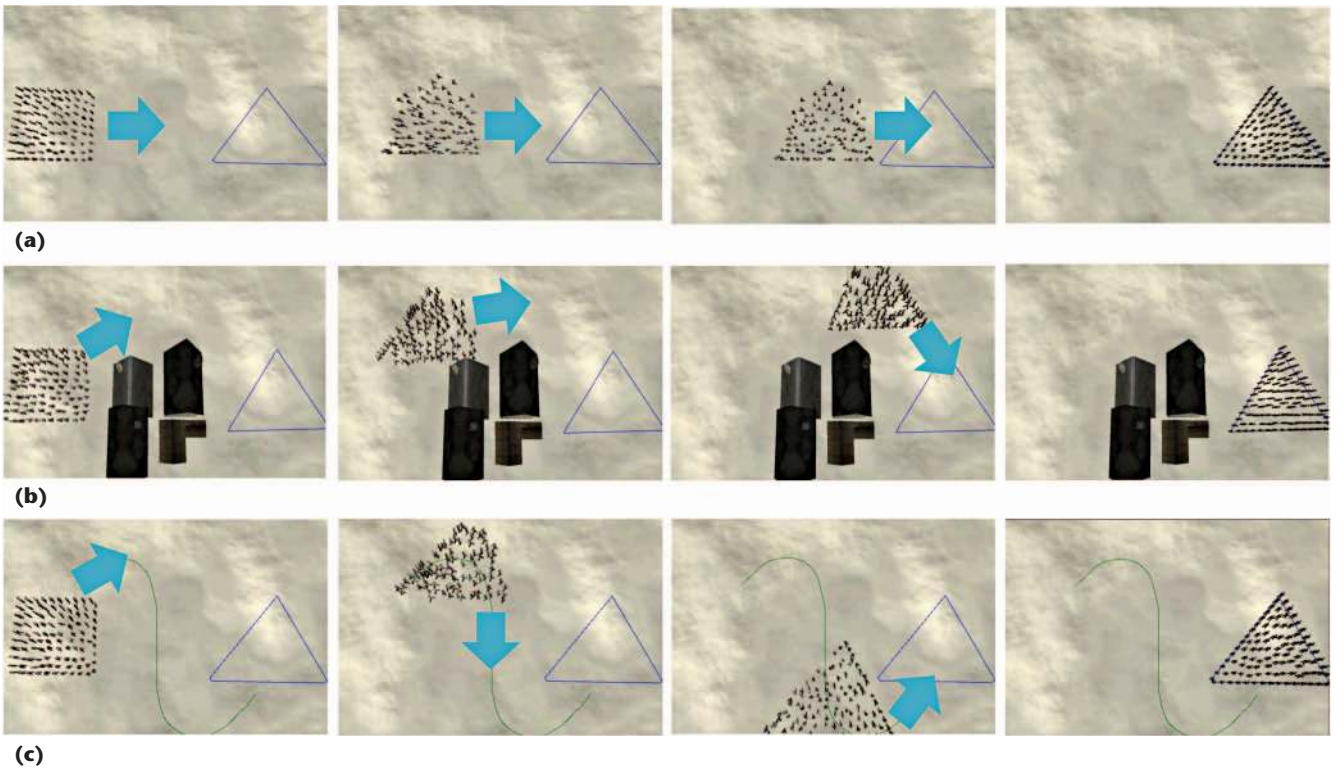


Figure 11. A square formation transforms to a triangle formation, using (a) unconstrained formation transitions, (b) formation transitions with obstacle constraints, and (c) formation transitions with trajectory controls. Our two-level formation control effectively preserved the group's collective behavior as it navigated through waypoints and encountered obstacles.

2008 Beijing Olympic logo as the input. Our system automatically generated an accurate formation that exactly fits the shape in the picture. Finally, using our boundary-sketching interface, we could procedurally create arbitrary formations with or without holes (see Figure 3). The resultant formations closely matched the inputted boundaries, which we could update during the simulation. However, because our approach doesn't currently support control of the agents' speed, empty areas in the crowd could occur during the formation transition.

Global Formation Control

In these experiments, we assumed the entire crowd was a single group and combined local and global formation control. First, we let the group move and interact with simple environments while forming and maintaining a specified formation. This type of group formation and interaction often occurs in military operations on flat terrain. We used our approach to generate key formations at different waypoints; the widely used A* pathfinding algorithm served as the global navigator.

Figure 11 shows a group of 150 agents heading to several navigational waypoints, sequentially driven by the global navigation vector while changing its formation according to user specifications. The group bypassed building obstacles (see Figure 11b) or followed a sketched control curve

Related Work in Crowd Simulation and Group Formations

As the first step of any crowd simulation, researchers have explored navigational pathfinding algorithms to handle complex dynamic environments. On the other hand, increasingly more researchers are focusing on middle-level perception models to simulate local dynamics. Among those models, force-based models[1] and their extensions[2] apply repulsion and tangential social forces to drive interaction between individual agents or subgroups. Following Craig Reynolds' seminal research,[3] researchers have also extensively studied rule-based simulation (for example, adding multilevel group rules and cognitive planning[4]) to achieve highly realistic human behavior. For a comprehensive review of crowd simulation techniques, see *Crowd Simulation*.[5]

Group formation and transition are vital characteristics of many crowds. Chris Wojtan and his colleagues used standard adjoint calculations with gradient-based optimization to control a flocking simulation while offering certain controllability (that is, keyframing control).[6] However, keyframing control in a large-scale, dynamic crowd simulation isn't always efficient. Renato Silveira and his colleagues introduced a group map structure to generate boundary group formations through user sketching, but without considering agent distribution.[7]

Taesoo Kwon and his colleagues' approach edited group motion as a whole while maintaining neighborhood formation and agents' moving trajectories in the original group motion data as much as possible.[8] The editing operations, such as pinning or dragging individuals, employed a graph structure in which vertices represented agents' positions at specific frames and edges encoded neighborhood formations and moving trajectories. This approach focused on minimizing the distortion of relative arrangements among adjacent agents. In contrast, our approach (see the main article) focuses on a more macroscopic perspective of keeping or changing the crowd formation's overall shape.

Shigeo Takahashi and his colleagues combined heuristic rules with explicit hard constraints to produce and control sophisticated group formations.[9] However, users had to manually specify exact agent distributions, which was time-consuming and labor intensive if the crowd contained many agents. Unlike their approach, ours needs only a few types of user input to specify high-level formations and automatically generates an appropriate agent distribution.

Sachin Patil and his colleagues proposed a comprehensive framework to guide moving agents with navigation fields.[10] Their approach showed convincing simulation results with flexible inputs of either user sketches or 2D video. The major distinction between their approach and ours is that we focus more on collective formation control than on guiding each agent individually.

Our preliminary research involved an intuitive sketching interface to generate arbitrary group formations in a crowd.[11] There are two main differences between that research and the research we report in the main article. First, our previous research only sketched boundary curves to specify formations, whereas this new research supports three types of user input (brush painting, texture maps, and boundary sketches). Flexible combination of these inputs can produce a richer variety of freestyle group formations.

Second, in our previous research, all the agents found their unconstrained optimal paths, so no user control existed. In contrast, our new research provides users with two-level agent motion transition control to intuitively guide simulations. First, *local formation transition control* treats the entire group as a local distribution with a static formation center. Users can specify different transition preferences for individual agents when the agents head to their target positions. Second, *global formation trajectory control* lets users specify arbitrary moving paths as high-level guidance constraints when navigating the group to its target position.

References

1. D. Helbing and P. Molnar, "Social Force Model for Pedestrian Dynamics," *Physical Rev. E*, vol. 51, no. 5, 1995, pp. 4282–4286.
2. N. Pelechano, J.M. Allbeck, and N.I. Badler, "Controlling Individual Agents in High-Density Crowd Simulation," *Proc. 2007 ACM Siggraph/Eurographics Symp. Computer Animation* (SCA 07), Eurographics Assoc., 2007, pp. 99–108.
3. C.W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Proc. Siggraph*, ACM, 1987, pp. 25–34.
4. Q. Yu and D. Terzopoulos, "A Decision Network Framework for the Behavioral Animation of Virtual Humans," *Proc. 2007 ACM Siggraph/Eurographics Symp. Computer Animation* (SCA 07), Eurographics Assoc., 2007, pp. 119–128.
5. D. Thalmann and S.R. Musse, *Crowd Simulation*, Springer, 2007.
6. C. Wojtan, P.J. Mucha, and G. Turk, "Keyframe Control of Complex Particle Systems Using the Adjoint Method," *Proc. 2006 ACM Siggraph/Eurographics Symp. Computer Animation* (SCA 06), Eurographics Assoc., 2006, pp. 15–23.
7. R. Silveira, E. Prestes, and L.P. Nedel, "Managing Coherent Groups," *Computer Animation and Virtual Worlds*, vol. 19, nos. 3–4, 2008, pp. 295–305.
8. T. Kwon et al., "Group Motion Editing," *ACM Trans. Graphics*, vol. 27, no. 3, 2008, article 80.
9. S. Takahashi et al., "Spectral-Based Group Formation Control," *Computer Graphics Forum*, vol. 28, no. 2, 2009, pp. 639–648.
10. S. Patil et al., "Directing Crowd Simulations Using Navigation Fields," *IEEE Trans. Visualization and Computer Graphics*, vol. 17, no. 2, 2011, pp. 244–254.
11. Q. Gu and Z. Deng, "Formation Sketching: An Approach to Stylize Groups in Crowd Simulation," *Proc. Graphics Interface 2011* (GI 11), Canadian Human-Computer Communications Soc., 2011, pp. 1–8.

(see Figure 11c) while gradually changing its formation. Agents farther from the obstacles or curve points automatically chose a longer path with a larger orbiting radius to save space for the agents that were closer to the obstacles.

Our results demonstrate our approach's flexibility and effectiveness. Most agent-based crowd simulation approaches focus on each agent's individual behaviors on the basis of its local information such as adjacent neighbors in the crowd. However, many real-world crowd scenarios such as battles and football games must consider each group's global information and interactions between groups. Also, unlike Shigeo Takahashi and his colleagues' research,[1] which focused on generating the transition between keyframes through various hard constraints, our approach automatically generates scalable, adaptive group formations. (For more on Takahashi and his colleagues' research and other related research, see the sidebar.) So, with our approach, users need only specify several intuitive high-level features such as formation sketches and trajectory sketches to generate large-scale interacting crowds with freestyle group formations.

In the actual implementation, a trade-off exists between movement smoothness and formation accuracy. For instance, when an agent evaluates its appropriate velocity heading to the target position, other agents might already have taken that position. This situation will result in the agent's unsmooth movement; the agent might continually try to reach the exact position in the target formation by going back and forth in a small area. In this situation, we let the agent probe the next available position by searching for the second-closest neighbor in the KD-tree.

We found that two-level collision avoidance can better keep the formations while avoiding obstacles. The intergroup dynamics tend to guide all the agents, from a group perspective, to avoid an obstacle as a whole if possible (see Figure 11c), instead of easily scattering the agents owing to local collision avoidance (see Figure 11a).

The current global trajectory control treats a group as a single entity by assuming that the group formations' shape is generally isotropic, such as a square or circle. So, global collision avoidance ignores the shape effect for anisotropic group formations. Thus, a group's agents still depend highly on local collision avoidance. We plan to work on this issue in the future.

Finally, our approach currently doesn't let users put timing constraints on formation transitions. Our next step will be to introduce this feature—for example, by incorporating agent velocity control to dynamically adjust agents' speed.

Acknowledgments

US National Science Foundation award IIS-0914965, Texas Norman Hackerman Advanced Research Program project 003652-0058-2007, and research gifts from Google and Nokia partly supported this research. Any opinions, findings, and conclusions or recommendations expressed in this article are the authors' and don't necessarily reflect the agencies' views.

References

1. S. Takahashi et al., "Spectral-Based Group Formation Control," *Computer Graphics Forum*, vol. 28, no. 2, 2009, pp. 639–648.
2. B. Ulicny, P.d.H. Ciechomski, and D. Thalmann, "Crowdbrush: Interactive Authoring of Real-Time Crowd Scenes," *Proc. 2004 ACM Siggraph/Eurographics Symp. Computer Animation* (SCA 04), Eurographics Assoc., 2004, pp. 243–252.
3. D. Helbing and P. Molnar, "Social Force Model for Pedestrian Dynamics," *Physical Rev. E*, vol. 51, no. 5, 1995, pp. 4282–4286.
4. N. Pelechano, J.M. Allbeck, and N.I. Badler, "Controlling Individual Agents in High-Density Crowd Simulation," *Proc. 2007 ACM Siggraph/Eurographics Symp. Computer Animation* (SCA 07), Eurographics Assoc., 2007, pp. 99–108.

Qin Gu *is a PhD candidate in the University of Houston's Department of Computer Science. His research interests include computer graphics and character and crowd animation. Gu received an MS in computer science from the University of Houston. Contact him at ericgu@cs.uh.edu.*

Zhigang Deng *is an associate professor of computer science at the University of Houston. His research interests include computer graphics, computer animation, and human-computer interaction. Deng received a PhD in computer science from the University of Southern California. He's a member of IEEE and ACM. Contact him at zdeng@cs.uh.edu or zdeng4@uh.edu.*