Formation Sketching: An Approach to Stylize Groups in Crowd Simulation

Qin Gu* Zhigang Deng†

Computer Graphics and Interactive Media Lab, Department of Computer Science
University of Houston

ABSTRACT

Most of existing crowd simulation algorithms focus on the moving trajectories of individual agents, while collective group formations are often roughly learned from video examples or manually specified via various hard constraints (*e.g.*, pre-defined keyframes of exact agent distributions). In this paper, we present an intuitive yet efficient approach to generate arbitrary and precise group formations by sketching formation boundaries. Our approach can automatically compute the desired position of each agent in the target formation and generate the agent correspondences between keyframes. When high-level group formations need to be formed on-the-fly in a dynamic environment such as "switching to the circle formation at about one hundred meters ahead", our algorithm will coordinate and compute appropriate actions for each agent by seamlessly fusing local formation dynamics and global group locomotion. Through a number of experiments, we demonstrate that our approach is efficient and adaptive to variations of group scales (*i.e.*, number of agents), group positions, and environment obstacles.

Index Terms: I.3.7 [Computer Graphics]: 3D Graphics and Realism—- Animation; I.3.6 [Computer Graphics]: Methodology and Techniques—- Interaction techniques

1 INTRODUCTION

In recent years large-scale crowd simulation has been increasingly used in various computer games, movies, virtual training, and education applications. In most crowd animation systems, each individual agent is capable of intelligently moving toward its destination through navigational path-finding algorithms [14; 18; 8], and avoiding collisions with other agents and obstacles through local behavior control models [23; 6; 22]. Hence, these approaches can often produce highly realistic simulation of path navigation, cognitive reaction, collision avoidance, and animation control.

However, up to date, relatively few research efforts have been focused on dynamically modeling the collective behavior of the entire crowd or multiple sub-groups in a crowd. In many scenarios, the collective behavior of a group plays a significant role to exhibit various macro- attributes of the crowd. From visual simulation point of view, the collective behavior of a crowd or group is usually exhibited as its general formations. For instance, when a large-scale battle is simulated in a computer game, the formations of two troops directly reflect the tactical commands of the controllers.

Currently, the popular means of forming a target group formation is to specify the desired position of each agent at a particular moment and then generate realistic transitions between the current position and the target/destination [32]. Nevertheless, this approach often requires users to manually specify many spatial correspondence constraints, which is a non-trivial, painstaking task. In particular, when the simulated crowd includes a large number of agents or the target formation is an arbitrary shape, it will be even more challenging.

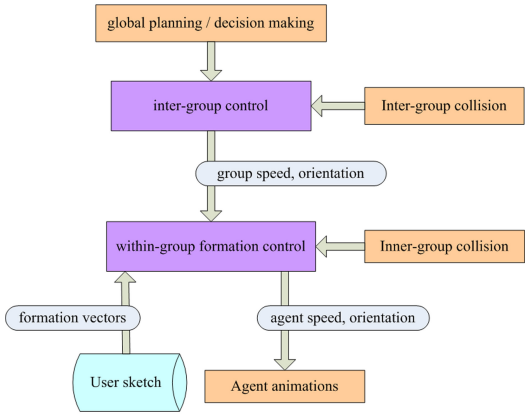

Figure 1: Schematic structure of our group formation synthesis scheme in a complete crowd simulation system

In this paper we propose an efficient framework to generate precise group formations in a crowd through an interactive sketching interface. Specifically, we first evaluate the formation boundary information from user sketches. The boundaries can be derived from multiple continuous curves or line strips depicting arbitrary convex or concave shapes, and the combination of using inclusive and exclusive boundaries gives us flexibility to create holes in a formation (Section 3). Then, we introduce a new concept called *formation coordinates* to find the desired position of each agent in the new formation while maintaining the adjacency naturalness among agents as much as possible during the local formation transition. When a group performs a global movement when forming a local formation, such as "*switching to the circle formation at about one hundred meters ahead*", our model will automatically coordinate and compute appropriate actions for each agent, by seamlessly fusing inter-group dynamics and within-group dynamics. As shown in Figure 1, from top to bottom, the final moving velocity of an agent (group member) is collectively determined by multiple factors including the target formation, local collision avoidance, and the navigational factor passed from the higher inter-group level.

The contributions of this work are summarized as follows:

(1) *A novel group formation sketching scheme*: Given a set of sketching curves, our method is capable of creating arbitrary formations (target agent distribution) with few parameter tuning. The new concept of formation coordinates enables our method to be adaptive to various number of agents, formation shapes and formation scales (radius) through formalizing a set of relative features.

(2) *A hierarchical scheme to fuse a local formation simulation and global group navigation*: We minimize the dimension of the used feature vectors solely for within-group formation control. Meanwhile, a higher level collision avoidance model and navigational path-planner are then used to guide the inter-group dynamics and interactions.

The remainder of this paper is organized as follows. Section 2 gives a brief review of related work. Section 3 describes the work

*e-mail: ericgu@cs.uh.edu
†e-mail: zdeng@cs.uh.edu

flow to generate target local formations from the user sketching. Section 4 illustrates our hierarchical model to control group formations when interacting with different environments or other groups. Experimental results and performances of our approach are reported in Section 5. Finally, limitations, future directions, and concluding remarks are presented in Section 6.

2 Related Work

Rule-based crowd models are flexible to simulate various autonomous crowd agents through a set of delicately-designed rules. Reynolds [23; 24] pioneered the concept of *Boids* that simulates flocks of birds and schools of fishes via several simple yet effective steering behavioral rules to keep the group cohesions, while avoiding collisions among group members. Later, as follow-ups, various rule-based approaches have been extended and systematically investigated by adding multi-level group rules [33], human psychological effects [27], perceptual knowledge of pedestrians [29], cognitive modeling [3], enhancing motion variety [4], behavioral actions selection [37], or egocentric affordance fields [9]. To handle more sophisticated environment constraints, a set of special agents, such proxy agents [36] and situation agents [28], were also proposed to receive classified rules.

Another research line of crowd simulations is force-based model. It was originally derived from human social force study, first proposed by Helbing and Molnár [6]. Later, it was further applied and generalized to other simulation scenarios such as densely populated crowds [22] and simulation of pedestrian evolutions [13]. Several recent efforts have introduced the continuum theory into the crowd simulation domain [1; 34]. The core idea of these methods is to replace local collision avoidance with continuous flows throughout the simulation grids. Narain *et al.* [19] further introduce unilateral incompressibility constraints to successfully simulate extremely large crowds, *e.g.*, more than 100,000 agents in a crowd.

In recent years, significant efforts have been attracted to model sophisticated crowd behaviors using example-based (or called data-driven) approaches. For example, subtle agent features can be extracted from pre-recorded 2D video [16; 15; 17] and these features can be incorporated into crowd simulations. 3D motion capture data were also exploited to calibrate crowd simulations [21]. Furthermore, the motion graphs concept [10], widely known in single character animation synthesis, was also introduced to guide crowd agents [12; 31].

Group formation is a vital collective characteristic of many crowds. An off-line optimization method was proposed in [35] to fit the flocking agents to a pre-defined 3D geometry. Silveira *et al.* [30] introduce a "group map" concept to generate the boundary formation but without considering the inside agent distribution. Other approaches typically combine heuristic rules with explicit hard constraints to produce and control sophisticated group formations. For example, Takahashi *et al.* [11] proposed a shape manipulation technique to deform an existing group formation as a whole. Their recent work [32] developed a spectral-based group formation control scheme. However, in their approach, if a group of agents are forming a certain formation, the exact agent distributions at a number of keyframes need to be manually specified by users, which is non-trivial manual work if the target formation contains a large number of agents. By contrast, our approach only requires a few of intuitive user sketches to specify the general boundaries of a formation. The final agent distribution is automatically generated by an efficient online optimization pipeline.

Rodrigues *et al.* proposed a space colonization algorithm to simulate several steering behaviors [25; 26]. In these approaches, a target formation is represented as high-density spray markers to attract agents moving towards the target space. If the target space has been occupied, the agent will try to keep probing other available spaces in the formation. Compared with these approaches, our approach is more user-friendly and controllable to navigate all the group agents towards the desired target positions while maintaining their local neighborhoods as much as possible. On the other hand, Ju *et al.* [7] also successfully generated several collective crowd behaviors in their most recent work by blending existing crowd examples. Their approach is powerful in simulating rough collective features in a flexible crowd but not suitable for animators who require strict spatial formation matching. Our scheme, from this point of view, can tackle this limitation through precisely fitting formation boundaries.

3 Estimating Group Formation

Manually keyframing and specifying hundreds of agents to predefine a target crowd formation is a painstaking and labor-intensive task even for skilled animators. In this section, we describe how to automatically compute the agent distribution in the target group formation from arbitrary, user-sketched boundaries. In other words, the problem can be posed as: "*Given a particular number of agents, how can we reposition the agents in a way that the overall agent arrangement closely fits the sketch. Meanwhile, the general adjacency information around each agent should be retained as much as possible.*" The second constraint can be observed in most formation transitions in the real world. For example, during a transition from a square group formation to a circle one, an agent located in the top-left of the formation will prefer to keep this relative position within the group all the time despite the change of formation shapes and scales.

3.1 Sketching Interface

Different from other sketching scheme mostly focusing on crowd locomotion controls [20], our sketching interface provides users an interactive tool to freely sketch the target formation by drawing any closed curves or line strips on the simulation plane (left of Figure 2). Those curves are then considered as target formation boundaries to constrain the group agents. In order to generate holes within some special formations like a circle ring formation, we introduce a new exclusive boundary type opposing to the traditional inclusive boundary. Combining these two types of boundaries, our system can generate sophisticated formations such as English characters (Figure 3).

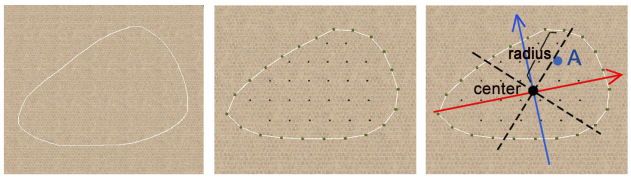

Figure 2: Computing formation coordinates from a user sketch: **Left** - sketch boundary, **Middle** - Initial formation template through filling boundaries, and **Right** - computing formation coordinate of position A through eigenvectors

3.2 Filling Formation

Given the formation boundaries and the original agent distribution, we construct its corresponding *formation template* as an over-sampled candidate space (i.e. template points more than agent numbers) for the target agent distribution. The template points represent possible candidate target positions for group agents (middle of Figure 2). A two-step sampling approach is applied to generate the template. First, we evaluate the desired target positions on the formation boundaries by evenly re-sampling all the sketch curves. The rest of the procedure becomes filling the region that is inside of the inclusive boundaries but outside of the exclusive boundaries, with discrete points. We extend the traditional Flood-Fill algorithm to tackle this problem. Specifically, our queue-based implementation discretely fills the curved boundary with user-specified sampling

units through a horizontal scan-line traversal order. See **Appendix A** for more details of this algorithm. For both steps in the filling procedure, the parameter selection of the sampling unit is critical to the final simulation result. A small sampling unit guarantees the over-sampled space for the group agents, while may break the balance between the agents at the formation boundaries and the agents inside the formation. Discussion of the optimal sampling unit selection is described in Section 3.6.

3.3 Formation Coordinates

The formation template includes a reference space for the target agent distribution. The target formation can be created through selectively filling the template. In other words, we need to find the optimal correspondences between agents and template points. However, the original world coordinates of each group agent provides little information directly usable for our purpose. For example, it is hard to find a good corresponding position in the template for an agent having a world coordinate (10, 0, 10) due to the shape and scale variations of user sketches. A new metric therefore is needed to represent the agent positions containing collective group information. In this work, we propose *Formation Coordinate*, a normalized relative pattern representation partially similar to polar coordinate, to serve our purpose.

We first compute the weighted group center, p_c, by averaging all agent positions (assuming a total of n members in the group). Then, the following relative features are extracted and considered for each group member or each template point (Figure 2).

1. *Relative orientation to the formation center*, co_t (0 - 360 degrees): When forming a certain formation, an agent (*i.e.*, group member) only needs to know its own general orientation within the group with respect to the current group center and any pre-defined direction. However, if users want to continuously rotate the formation to align the inter-group velocity (Section4.2), our system needs to multiply the rotation matrix while computing the agent relative orientation.

2. *Relative normalized distance to the formation center*, cd_t (0 - 1): With the above co_t, the relative distance cd_t can together define the current relative position of an agent within the group. Furthermore, its normalized representation ensures it can be conveniently used for scalable runtime simulations (*e.g.*, varied target formation scales). The denominator of the normalization is the radius of the formation. Simply using a constant value for this radius, such as the furthermost agent distance will result in a poor estimation for non-circular formation shape. An ideal radius would be computed based on a set of distances from formation center to every boundary pixel. However, it is not efficient to compute such a large number of radiuses regardless of agent distribution. Instead, we retrieve eight most representative formation radiuses from the agent or template point distribution. For example, on the right of Figure 2, we first apply a singular value decomposition on the 2D distribution (z coordinates are temporarily ignored), and then use the two resulting eigenvectors as new orthogonal axes (red and blue lines) to evenly generate the eight radiuses by the vectors from the center to the eight intersection points on the boundary (dashed lines). When performing the normalization, we search for the closest radius as the normalization factor. In our experiments, this method is fast enough for our real-time application and provides acceptable results.

Given a point A in the world space (Figure 2), the formation coordinate can be finally formulated as a three dimensional feature vector (cd_t, co_{tx}, co_{ty}) where the co_{tx} and co_{ty} are two components of the normalized vector from the formation center to the point A to represent the relative agent orientation.

In our experiments, a total number of eight radiuses appeared to be a good threshold to balance the computational performance and the final formation quality. It is also noteworthy that our compact representation of group formation (*i.e.*, the above feature vectors) only contains relative information with respect to the group center, regardless of arbitrary positioning of the group. The general movement of the group (center) as a whole is dynamically determined in the higher inter-group level (refer to Section 4.2).

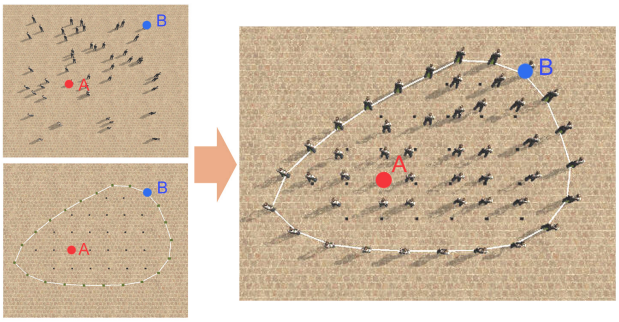

Figure 4: Generating the target formation based on the correspondences of formation coordinates: (Top-Left) Original agent distribution, (Bottom-Left) Finding corresponding positions in the formation template, and (Right) the final agent distribution after point-based relaxation.

3.4 Finding Correspondences

When transiting one formation to another, it is essential to find the correspondence between an agent's original position and its new position in the target formation. Typically, previous approaches manually specify the target position and its corresponding agent in the original formation via keyframing. By contrast, our method focuses on automatically finding the corresponding positions in the formation template, and then using the selected position to evaluate the final target position through a later relaxation process (Section 3.5).

Our correspondence finding scheme is based on the following two-fold assumption: (1) In the target formation, agents on the boundaries must closely fit the boundary curves to clearly demonstrate the desired formation shape sketched by users. This is crucial to user perception since profile is the most important feature to distinguish one formation from others. (2) Each agent prefers to keep its adjacency condition as much as possible. Similar to the principle of least effort used in [5], it is less likely that a group member would take more perceived efforts to go across the formation to reach a more distant position if a closer yet reachable one is available. In our approach, this assumption can be realized by searching for the closest formation coordinate.

Finding agents on the boundaries: In order to guarantee the exact boundary representation, every boundary point in the formation template needs to have one and only one corresponding agent to fit in. Hence this process must be the first step of the correspondence finding procedure. We first convert positions of all the agents in the original distribution into relative formation coordinates, as described in Section 3.3, and store the results in a KD-tree data structure. For each point on the template boundaries, its world coordinate is also converted to formation coordinate in a similar way. Consequently, the agent corresponding to every boundary point can be computed by finding the nearest neighbor in the KD-tree (point B at the bottom-left of Figure 4).

Finding agents within the boundaries: Next, we need to find the corresponding target positions for the rest of the agents. Un-

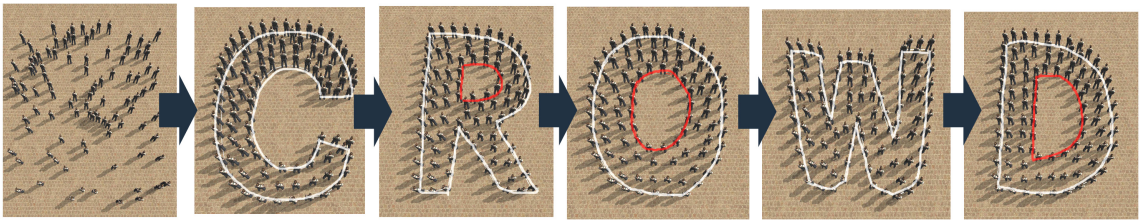

Figure 3: Generating arbitrary formations from random agent distributions (Left-most) through inclusion sketch (white boundaries) and exclusion sketch (red boundaries)

like the boundary fitting described above, we cannot find one-to-one correspondence between each inner template point and the rest agent, since the formation template is an over-sampled space. Instead of finding corresponding agent based on each template point, in this step, we inversely evaluate the corresponding template point for each of the non-boundary agents that are not selected in the previous step. Similarly, the formation coordinate of each agent is used to find the closest neighbor in the template. If the closest neighbor has been occupied, we iteratively search the next closest neighbor. The resultant formation coordinate of the template point is then converted back to its world coordinate as the agent target position (point A in the bottom-left of Figure 4).

3.5 Relaxation

Until this step, we have already generated the exact profile and general inner-distribution of the target formation. However, the agent distribution inside the formation boundaries is not evenly distributed due to the fact that the number of evenly distributed template points is larger than the number of non-boundary agents. To deliver the final formation result, a few relaxation steps are needed to optimize the agent distribution. Unlike the element-based relaxation used in many geometric deformation algorithms, the virtual connections among group members typically should not be strictly fixed during the simulation, especially in a transition between two significantly different formations. To this end, we perform a point-level relaxation on the non-boundary agents using boundary points as relaxation constraints. For each agent, the impacts of its surrounding neighbors are evaluated via a Radial Basis Function with Gaussian kernel (detailed in **Algorithm 1**). The parameter *speed* is a constant to all the agents to control the relaxation progress while the relaxation weight w varies among agents to indirectly control the "evenness" of agent distributions. Larger weights will make an agent faster to be relaxed and tend to push other agents to form sparser areas. Although we only pre-script the agent weights in our experiments, they can be assigned in multiple ways such as a brush sketch interface.

Note that, in the above Algorithm 1, the function *updateDistribution()* requires a re-creation operation if the KD-tree data structure is used, which costs $\Theta(n \log n)$. Performance is a critical issue for an agent-based crowd simulation since it has to update every agent individually at each time interval. In our implementation, the $n \log n$ complexity is reduced to n by registering agents into a discretized grid at each update. The function *closestPoints()* therefore simply collects the agents registered in the surrounding cells with the complexity of $\Theta(1)$.

3.6 Sampling Refinement

The selection of a proper sampling unit is critical to the success of the proposed formation generation pipeline (described from Section 3.2 to Section 3.5). If the sampling unit is too large, it may risk the under-sampling of the formation template so that some "redundant" agents cannot find their corresponding positions in the target formation. On the other hand, if the sampling unit is too small, it may lead to a bad balance between the number of boundary agents

Algorithm 1 Point-based Formation Relaxation

Input: *bPoints*, constraint boundary points.
Input: *iPoints*, points to be relaxed.
Input: η, relaxation iterations.
Input: β, Gassian constant parameter ($\beta > 0$).
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;
```

and non-boundary agents (an example is shown in the right of Figure 5). Since we force the exact boundary fitness, there might be not enough agents to fill the area within the boundaries. In this case, the non-boundary agents will appear to be sparsely distributed in the target formation even with the above relaxation process. To this end, we propose an automatic way to compute the near-optimal sampling unit, described as follows.

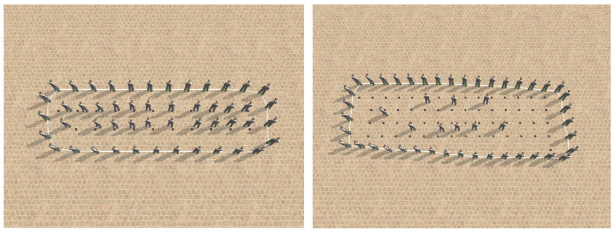

Figure 5: Refining the sampling unit: (Left) Our automatic sampling optimization, (Right) a too small sampling rate specified by users.

First, we follow the formation filling process (described in Section 3.2) using any small constant sampling unit, such as 100 as opposed to the world grid unit as 1000. Once the initial formation template is generated, we compute the near-optimal sampling unit as shown in Equation 1. We compute the number of points on the initial boundary b' and the square root of the number of points inside the initial boundary f'. Since the ratio between these two numbers is approximately a constant for a formation in spite of different scales, we can estimate the desired number of boundary agents b by

solving the quadratic Equation 2 so as to infer the sampling unit by dividing the total length of the user-sketched boundaries. Note if the sampling unit is shorter than the radius of each agent when the skeching boundary is extremely small, the agents will be pushed away from each other by the local repulsion force so as to break the sketching boundary.

$$samplingUnit = \frac{\rho \times p}{b} \tag{1}$$

$$b^2 + (2 \times (n + \frac{\sqrt{f'}}{b'})) \times b + n^2 = 0 \tag{2}$$

In Equations 1 - 2, ρ denotes the length of the user-sketched curves, n denotes the number of agents, and parameter p is a percentage value (0-1). Due to the possible offset errors and curve direction variations during re-sampling, p ranging from 0.8 to 0.9 would provide plausible results based on our experiments. The left of Figure 5 shows the re-generated target formation when the new sampling unit is used.

4 Generating Formations in Crowd Simulations

Now we describe how to simulate the transition to form the estimated target formation described above. In particular, our runtime crowd formation generation algorithm can handle target formations with any scale (independent of the original formation scale) due to the use of relative formation coordinates. Also, a force-based crowd behavior model is employed to handle fast local collision avoidance.

4.1 Formation Adjustment

The force that drives an agent from its original position to its estimated target position can simply be the direction vector between the two positions. However, this force only considers the group formation factor. Driven solely by the estimated formation distribution, each agent will perform the movement directly to the desired target. Typically, this works reasonably well for sparse groups. However, in a dense group, such pure formation-driven strategy cannot fully avoid agent collisions. As such, a local collision model is needed to refine within-group collision avoidance.

In this work, we employ a force-based model [22] for the above collision avoidance task due to its capability of handling very high density crowds. Therefore, the final within-group velocity is adjusted as follows.

$$V_{within} = w_1 V_f + w_2 \sum_{i=1}^{n-1} f_i \tag{3}$$

Here the second term is the summed velocity driven by the combination of collision avoidance forces and repulsion force from neighboring group members and obstacles [22], and the weights balance the expected speed of formation generation (w_1) and the expected accuracy of collision avoidance (w_2).

4.2 Hierarchical Group Control

In the previous sections, we assume all the agents in a group only move and change formations relative to a fixed group center. However, in many crowd simulations, a group of agents are often required to form or maintain a certain group formation while they are moving to other locations. Explicitly encoding such global movements in the obtained group formation feature vectors will inevitably increase computational costs. On the other hand, directly solving the absolute velocity of each agent through traditional heuristic crowd models will easily break the collective group formation in a dynamic environment (top of Figure 6).

In this work, we introduce a novel two-level hierarchical group control scheme, that is, breaking the full group dynamics to *within-group dynamics* and *inter-group dynamics* so that the final absolute velocity of an agent can be computed as follows.

$$V = (w_1 V_f + w_2 V_c) + (w_3 V_{gn} + w_4 V_{gc}) \tag{4}$$

Here V_f and V_c denote the first local formation velocity term and second local collision avoidance term as shown in Equation 3, V_{gn} denotes the inter-group navigational velocity, and V_{gc} denotes the inter-group collision avoidance. It should be noted that, in Equation 4, the weights, w_3 and w_4, balance the expected accuracy of way-point following of the group and the expected accuracy of the group's collisions with other groups or environments, respectively.

At the inter-group level, each group in the scene is considered as a single entity. Any collision avoidance and path planner model can be applied on the group entity as a whole just as the single agent simulation. Then, the obtained inter-group velocity is passed to the within-group dynamics as the local navigational velocity. Thus, Equation 4 can also be equivalently formulated as follows.

$$V = V_{within} + V_{inter} = w_1 V_f + w_2 V_c + w_{ln} V_{ln} \tag{5}$$

Here V_{ln} is the local navigational velocity passed from the combined inter-group velocities. It is noteworthy that, in our actual implementation, we use the above Equation 4, not Equation 5. The latter is mainly used to reveal the hierarchical composition.

The main advantages of our hierarchical velocity representation are:

- Simplifying the formation dynamics to a partially static process,
- Maintaining the collective feature of the group since all the group members share the same local navigator,
- And reducing the computational costs by replacing the path-planning of individuals with the global group path-planning.

Note that our hierarchical scheme does compute collision avoidance twice. However, compared with the number of agents, the number of groups in the scene is typically much smaller so that the additional global collision avoidance will not have noticeable impact on the overall runtime performance. In our experiments, we found the two-level collision avoidance can better keep the formations when avoiding obstacles, due to the fact that instead of easily scattering agents caused by local collision avoidance, the inter-group dynamics tend to guide all the agents to avoid an obstacle as a whole if possible (bottom of Figure 6) from the group perspective.

In the follow-up result section, we will demonstrate several selected applications of our hierarchical group control scheme. In addition, four important weights (w_1, w_2, w_3, and w_4) are used in the above Equations 3 and 4. In this work, these weights are empirically determined (shown in Table 1).

Scenario	w_1	w_2	w_3	w_4
Arbitrary formations (Fig. 3)	0.6	0.4	n/a	n/a
Single group dynamics (Fig. 7)	0.4	0.2	0.3	0.1
Single character and group (top of Fig. 8)	0.3	0.2	0.4	0.1
Three groups (bottom of Fig. 8)	0.5	0.2	0.2	0.1

Table 1: Weight distributions (refer to Equations 3 and 4) used for all the simulations shown in this paper

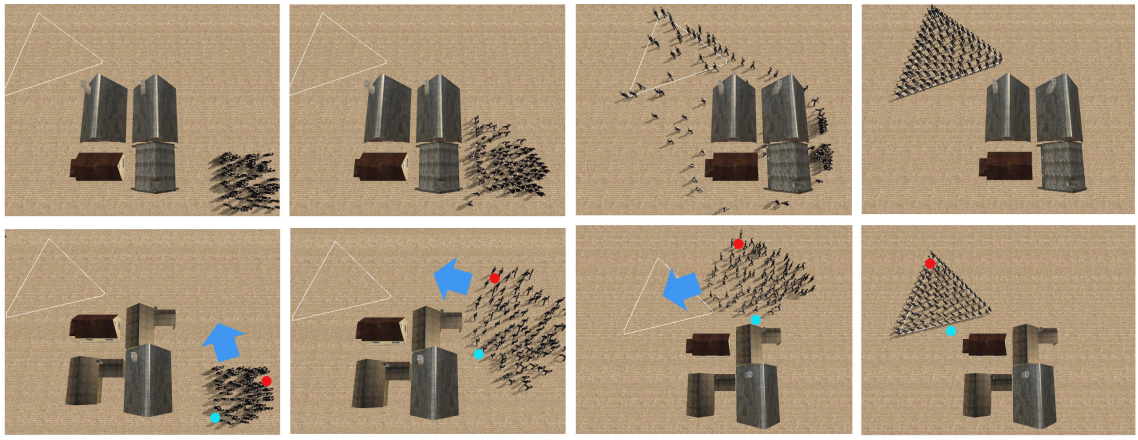

Figure 6: A random formation (Left column) forms a triangle formation (Right column) without (Top) and with (Bottom) our hierarchical group control. The different transition progresses (Left Middle column - Right Middle column) show that our hierarchical scheme can better preserve the collective behavior of a group when encountering obstacles. The bottom group bypasses the building obstacles as a whole when changing the formation gradually. Note the agents further away from the obstacles (red dot) automatically choose a longer path with larger orbitting radius to save spaces for closer agents (blue dot) in the formation.

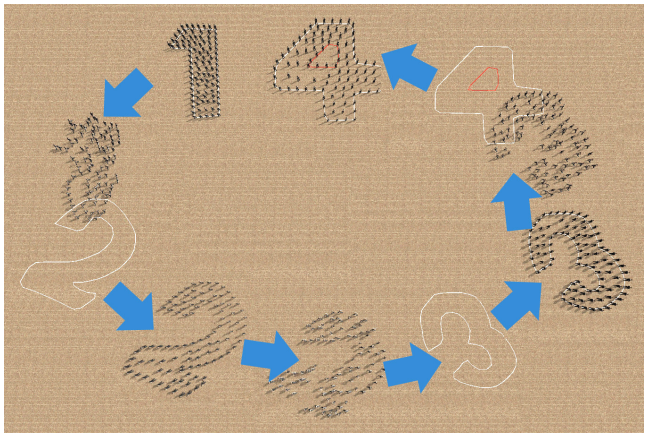

Figure 7: Single group transitions with four formations and four navigational way-points: by combining the within-group and inter-group dynamics, our model is spatially translatable. This figure shows a group with 100 agents heading to 4 navigational way-points (center of sketchings) sequentially driven by the inter-group dynamics, while forming and maintaining the specified 4 types of formations. When the global velocity changes its directions dramatically (4 corners of the figure), the within-group dynamics still attempt to maintain the formation consistent with the correct relative orientations.

5 Results and Runtime Performances

In this section, we demonstrate several applications of our formation and hierarchical group control approach. A force-based, HiDAC crowd simulation model [22] was used to handle collision avoidance for both single agent and group entity to avoid inter-agent and inter-group collisions. A regular PC with 2.4 GHz CPU, 2GB memory and Nvidia GF260 was used in our experiments. For each crowd agent, articulated 3D human models (700 to 800 polygons) and high-quality motion capture primitives with 30 joints (62 DOF) were used to drive his/her walking animations. The detailed animation results can be found in the enclosed demo video.

5.1 Static Formations

We designed a set of crowd simulation experiments to evaluate the within-group dynamics, that is, the agents in a single group were driven by their relative formation velocities and local collision avoidance. At the beginning of the simulation, we computed the group center of the initial agent positions and fixed the target group formation center at the same position. Two sets of examples were used in this test. Figure 3 shows a group of 100 autonomous agents forming several English word formations through the freeform formation sketches that are composed of inclusive and exclusive boundary curves. The results show that our within-group formation features can effectively capture main characteristics of the formation sketches. In the actual implementation, there is a trade-off between movement smoothness and formation accuracy. For instance, when an agent evaluates his/her appropriate velocity heading to his/her target position (destination), the position may have been temporarily taken by other agents. This situation will result in unsmooth movements of the agent; the agent may continuously attempt to reach the exact position in the target formation while pushed away by the collision avoidance forces. Therefore, we specify a “termination threshold" for the movements of each agent. At each update, if the difference between the target (ideal) position and the current position is smaller than the termination threshold, we stop the formation process and the agent becomes stable at the current location. In our experiments, we experimentally set the orientation vector and distance thresholds to (0.1, 0.1) and 1/20 of the formation radius, respectively.

5.2 Single Group Dynamic Interactions

In this experiment, we assume the entire crowd is a single group. It moves and interacts with different environments while forming or maintaining a certain group formation. This type of group formation and interaction is often observed in various military operations and performance shows. During the runtime, we let users dynamically sketch the formation boundaries at any location with specified target group orientations, and employed the widely-used A* pathfinding algorithm [2] as the global navigator for the group entity. However, we did not use global path planner for each agent because our local formation control will guarantee each agent move along the group. Figure 7 shows a group with 100 agents heading to 4 navigational way-points sequentially driven by the inter-group dynamics. Given a group destination and an expected target formation type, our approach automatically generated accurate group positions and orientations from one way-point to another, and successfully simulated different group formations and their inter-formation transitions. As shown in the enclosed demo video, the agents managed to successfully maintain the group formation while moving toward the specified destinations.

5.3 Multiple Group Interactions

In many applications such as computer games, a user need to control the main character to interact with autonomous groups in the virtual world. For example, an infantry squad needs to follow a user-controlled character heading toward a target location. Traditional AI typically pre-scripts the position of each agent relative to the main character's position. This simple and heuristic solution is fast at the expense of the loss of animation and variation realism. Our method, on the other hand, is much more flexible while maintaining the desired group formation as similar as possible (top of Figure 8). In this simulation, the global navigational way-points are constrained by the main character's positions and updated with the user's actions.

Another typical multiple group interaction scenario is formation combinations in large-scale performance shows, as shown in the bottom of Figure 8. In this simulation, we also used an existing force-based model [22] to handle inter-group and within-group collision avoidance. When three group formations were interleaved with each other, our hierarchical group control scheme (Section 4.2) can automatically ensure the formation maintenance of all the involved groups. For the animation results, please refer to the enclosed demo video.

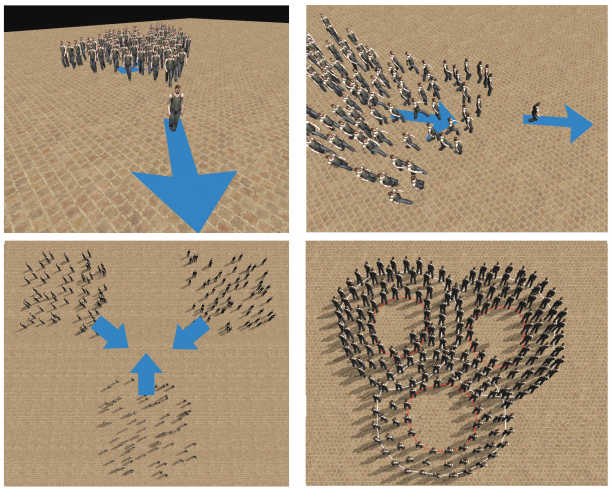

Figure 8: **Top**: In this simulation, users directly control the main character walking ahead of others like a squad leader. The other autonomous agents will act like followers to follow the leader as close as possible. At the same time, they need to keep their square formation. Since the main character is much more flexible with respect to rotation in the real-world. We add an angular rotation constraint to the squad group to prevent it from rotating too fast so that the squad appears certain "lag" to the main character's (the leader's) action. **Bottom**: It shows three groups of agents forming a combined formation together in a big performance show. Each group starts with a random agent distribution, and the three groups are relatively distant. Then, they simultaneously transit to a combined interleaved formation. With the inter-group navigational velocity and within-group formation velocity, most agents are capable of reaching the plausible positions without scattering via local collision avoidance.

5.4 Runtime Performances

Runtime performance has always been one of the most critical issues for an agent-based crowd simulation system since the system need to update every agent individually at each update step. We made the update interval of group formation control larger than the update frequency of high-level simulation and staggered the update of different agents. This can efficiently utilize the computational resource while maintaining perceptually believable formation transitions.

With our hierarchical group control scheme, we are able to simulate a crowd up to 400 agents with 30 FPS (frames per second) before the rendering bottleneck, which is higher than most pure example-based crowd simulation methods. To quantify the overhead of our approach, we compared the average FPS rates with and without our group formation control scheme combined with the heuristic collision avoidance model [22]. Table 2 shows that our group formation control scheme only added a small amount of overhead on top of the higher-level crowd simulation system.

crowd size	original FPS	new FPS	overhead
100 agents	140.1	135.5	3.8%
200 agents	100.5	97.2	4.0%
300 agents	55.2	53.4	4.1%
400 agents	34.6	33.3	3.7%

Table 2: Runtime performance statistics: the overhead is computed by comparing the unlimited FPS rates between the original HiDAC model and the HiDAC model with our group formation control approach.

6 Discussion and Conclusions

The central objective of this work is to provide a flexible sketching-based framework to generate arbitrary group formations in a crowd. Unlike example-based approaches, our method can freely create the precise crowd representations specified by users, instead of limited by the number of training examples. The result of our formation estimation algorithm can be incorporated into most of agent-based crowd simulation pipelines. We also propose a two-level hierarchical scheme to embed our group formation control into dynamic environments as well as simulate realistic multi-group interactions.

Most heuristic crowd simulation methods typically focus on the action of each agent based on his/her local information such as adjacent neighbors in the crowd. However, many real-world crowd scenarios such as battles between two sides and football games are required to consider the global information of each group and interactions between different groups. Also, distinguished from the previous work by Takahashi et al. [32] that is focused on generation of the transitions between key-frames via various hard constraints, our approach is designed to automatically generate scalable and adaptive group formations. Thus, using our approach, users only need to specify several intuitive features such as formation sketches, group locations, and group environments to generate large-scale interacting crowds with arbitrary yet controllable formations.

Certain limitations still exist in the current work. First, our approach set the agent distribution in the target formations to be nearly even by default, since even distribution is desired in many formation scenarios such as the grand ceremony shows. However, if a user want to specify partially denser or sparser areas intentionally, manual scripting of the relaxation weights in the **Algorithm 1** is required in the current system. We plan to develop a more interactive sketching interface, such as a brush tool, to tackle this issue. Second, agents in our approach directly move towards the target positions when computing within-group velocities. In many real world scenarios, group members tend to show sophisticated behaviors while approaching the target, such as following a curve path. We plan to take this issue into our future work to further increase the simulation realism. Also, the current work does not consider the exact time constraint for every agent as most of group editing tools (e.g., the work of [11]). We plan to explore a time-dependent velocity to tackle this issue in the future.

Acknowledgements

This work is supported in part by NSF IIS-0914965, Texas NHARP 003652-0058-2007, and research grants from Google and Nokia.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the agencies.

REFERENCES

[1] S. Chenney. Flow tiles. In *SCA '04*, pages 233–242, 2004.

[2] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *the Journal of ACM*, 32(3):505–536, 1985.

[3] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *Proc. of SIGGRAPH '99*, pages 29–38, 1999.

[4] Q. Gu and Z. Deng. Context-aware motion diversification for crowd simulation. *IEEE Computer Graphics and Applications*, March (online first) 2010.

[5] S. Guy, J. Chhugani, S. Curtis, P. Dubey, M. Lin, and D. Manocha. Pledestrians: A least-effort approach to crowd simulation. In *SCA'10*.

[6] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282+, 1995.

[7] E. Ju, M. G. Choi, M. Park, J. Lee, K. H. Lee, and S. Takahashi. Morphable crowds. *ACM Transaction on Graphics*, 29:140:1–140:10, December 2010.

[8] A. Kamphuis and M. H. Overmars. Finding paths for coherent groups using clearance. In *SCA '04*, pages 19–28, 2004.

[9] M. Kapadia, S. Singh, W. Hewlett, and P. Faloutsos. Egocentric affordance fields in pedestrian steering. In *I3D'09*, pages 215–223, 2009.

[10] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. *ACM Transaction on Graphics*, 21:473–482, 2002.

[11] T. Kwon, K. H. Lee, J. Lee, and S. Takahashi. Group motion editing. In *Proc. of ACM SIGGRAPH '08*, volume 27, page 80, 2008.

[12] Y.-C. Lai, S. Chenney, and S. Fan. Group motion graphs. In *SCA '05*, pages 281–290, 2005.

[13] T. I. Lakoba, D. J. Kaup, and N. M. Finkelstein. Modifications of the helbing-molnar-farkas-vicsek social force model for pedestrian evolution. *Simulation*, 81(5):339–352, 2005.

[14] F. Lamarche and S. Donikian. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, 23:509–518, 2004.

[15] K. H. Lee, M. G. Choi, Q. Hong, and J. Lee. Group behavior from video: a data-driven approach to crowd simulation. In *SCA '07*, pages 109–118, 2007.

[16] A. Lerner, Y. Chrysanthou, and D. Lischinski. Crowds by example. *Computer Graphics Forum*, 26:655–664, 2007.

[17] A. Lerner, E. Fitusi, Y. Chrysanthou, and D. Cohen-Or. Fitting behaviors to pedestrian simulations. In *SCA '09*, pages 199–208, 2009.

[18] R. A. Metoyer and J. K. Hodgins. Reactive pedestrian path following from examples. In *CASA'03*, page 149, 2003.

[19] R. Narain, A. Golas, S. Curtis, and M. C. Lin. Aggregate dynamics for dense crowd simulation. *ACM Transaction on Graphics*, 28(5):1–8, 2009.

[20] M. Oshita and Y. Ogiwara. Sketch-based interface for crowd animation. In *Proceedings of the 10th International Symposium on Smart Graphics*, pages 253–262, 2009.

[21] S. Paris, J. Pettre, and S. Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. *CGF*, 26:665–674, 2007.

[22] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling individual agents in high-density crowd simulation. In *SCA '07*, pages 99–108, 2007.

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

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

[25] R. A. Rodrigues, A. de Lima Bicho, M. Paravisi, C. R. Jung, L. P. Magalhães, and S. R. Musse. Tree paths: A new model for steering behaviors. In *Proceedings of the 9th International Conference on Intelligent Virtual Agents*, IVA '09, pages 358–371, 2009.

[26] R. A. Rodrigues, A. de Lima Bicho, M. Paravisi, C. R. Jung, L. P. Magalhaes, and S. R. Musse. An interactive model for steering behaviors of groups of characters. *Applied Artificial Intelligence*, 24:594–616, July 2010.

[27] T. Sakuma, T. Mukai, and S. Kuriyama. Psychological model for animating crowded pedestrians. *Computer Animation and Virtual Worlds*, 16(3-4):343–351, 2005.

[28] M. Schuerman, S. Singh, M. Kapadia, and P. Faloutsos. Situation agents: agent-based externalized steering logic. *The Journal of Computer Animation and Virtual Worlds*, 21(3‐4):267–276, 2010.

[29] W. Shao and D. Terzopoulos. Autonomous pedestrians. In *SCA '05*, pages 19–28, 2005.

[30] R. Silveira, E. Prestes, and L. P. Nedel. Managing coherent groups. *Comput. Animat. Virtual Worlds*, 19(3-4):295–305, 2008.

[31] M. Sung, L. Kovar, and M. Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *SCA '05*, pages 291–300, 2005.

[32] S. Takahashi, K. Yoshida, T. Kwon, K. H. H. Lee, J. Lee, and S. Y. Y. Shin. Spectral-based group formation control. *Computer Graphics Forum*, 28:639–648, 2009.

[33] D. Thalmann, S. R. Musse, and M. Kallmann. Virtual humans' behaviour: Individuals, groups, and crowds. In *International Conference on Digital Media Futures*, volume 1, page 122, 1999.

[34] A. Treuille, S. Cooper, and Z. Popovic. Continuum crowds. In *Prof. of ACM SIGGRAPH '06*, pages 1160–1168, 2006.

[35] J. Xu, X. Jin, Y. Yu, T. Shen, and M. Zhou. Shape-constrained flock animation. *Comput. Animat. Virtual Worlds*, 19(3-4):319–330, 2008.

[36] H. Yeh, S. Curtis, S. Patil, J. van den Berg, D. Manocha, and M. Lin. Composite agents. In *SCA '08*, 2008.

[37] Q. Yu and D. Terzopoulos. A decision network framework for the behavioral animation of virtual humans. In *SCA '07*, pages 119–128, 2007.

Appendix A The pseudo-code Algorithm 2 is based on the traditional Flood-Fill algorithm that was originally designed for a continuous space (e.g. pixel by pixel). In our work, when checking the connected neighbors of each sampling point inside the formation template, we use a function *PointInBound*() to check if a point is inside the polygon composed from the boundary points. There are a number of ways to evaluate this problem. We choose the angle sum solution due to its robustness in 3D space. Furthermore, to avoid sampling points too close to the boundaries, the *PointInBound*() function actually checks four points with constant offsets (top, bottom, left, right) to the current checkpoint.

Algorithm 2 Discrete Flood-fill Algorithm

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 PointInBound(left) && notChecked(left) do
7:          fPoints.push(left);
8:          top = left + unitY;
9:          if PointInBound(top) && notChecked(top) then
10:             q.push(top);
11:         end if
12:         bottom = left - unitY;
13:         if PointInBound(bottom) && notChecked(bottom) then
14:             q.push(bottom);
15:         end if
16:         left = checkPoint - unitX;
17:     end while
18:     perform the same steps for its neighbors at the right
19:     queue.pop(checkPoint);
20: end while
21: return fPoints;
```