Directing Crowd Simulations Using Navigation Fields

Sachin Patil,Jur van den Berg,Sean Curtis,Ming Lin,Dinesh Manocha

Abstract—We present a novel approach to direct and control virtual crowds using *navigation fields*. Our method guides one or more agents towards desired goals based on *guidance fields*. The system allows the user to specify these fields by either sketching paths directly in the scene via an intuitive authoring interface or by importing motion flow fields extracted from crowd video footage. We propose a novel formulation to blend input guidance fields to create singularity-free, goal-directed navigation fields. Our method can be easily combined with most current local collision-avoidance methods and we use two such methods as examples to highlight the potential of our approach. We illustrate its performance on several simulation scenarios.

Index Terms—Multiagent systems, Animation, Virtual Reality

1 Introduction

Over the last few decades, advances in agent-based systems, cognitive modeling and AI have found widespread use in simulating autonomous agents and virtual crowds in computer games, training systems and animated feature films. In addition, multi-agent simulation systems are also used for studying human and social behaviors for architectural and urban design, training and emergency evacuation simulations.

Most existing agent-based systems assume that each agent is an independent decision making entity. Some of the methods also focus on group-level behaviors and complex rules for decision making. The problem with these approaches is that interactions of an agent with other agents or with the environment are often performed at a local level and can sometimes result in undesirable macroscopic behaviors. Due to the complex inter-agent interactions and multi-agent collision avoidance, it is often difficult to generate desired crowd movements or motion patterns that follow the local rules.

In this work, we address the problem of directing the flow of agents in a simulation and interactively controlling a simulation at runtime. Our approach is mainly designed for goal-directed multi-agent systems, where each agent has knowledge of the environment and a desired goal position at each step of the simulation. The goal position for each agent can be computed from a higher-level objective and can also dynamically change during the simulation.

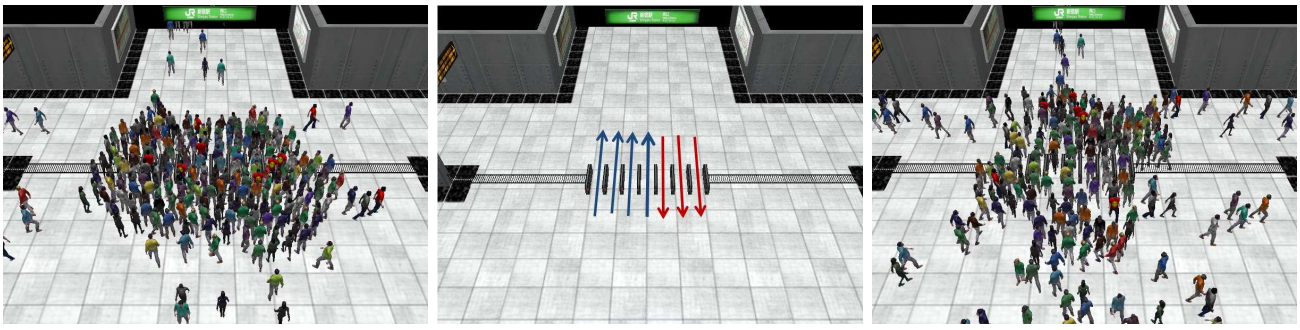
- *S. Patil, S. Curtis, M. Lin and D. Manocha are with the Department of Computer Science, University of North Carolina at Chapel Hill E-mail: {sachin,seanc,lin,dm}@cs.unc.edu*
- *J. van den Berg is with the Department of Electrical Engineering and Computer Science, University of California at Berkeley E-mail: berg@berkeley.edu*

Main Results: We present an interactive algorithm to direct and control crowd simulations. Our approach uses user-specified *guidance fields* to direct the agents in a simulation (Section 4). The user edits the simulation by specifying guidance fields, that are either drawn by the user or extracted from a video sequence. Based on these inputs, we compute a unified, goal-directed, smooth *navigation field* that avoids collisions with the obstacles in the environment. The guidance fields can be edited by a user to interactively control the trajectories of the agents in an ongoing simulation, while guaranteeing that their individual objectives are attained (Section 5). The microscopic behaviors, such as local collision avoidance, personal space and communication between individual agents, are governed by the underlying agent-based simulation algorithm (Section 6). Our approach is general and applicable to a variety of existing agent-based methods. We illustrate the usefulness of our approach in the context of several simulation scenarios (Section 7).

As compared to prior approaches, our algorithm offers the following benefits:

- Goal-directed navigation of multiple groups of *heterogeneous* agents using smooth navigation fields;
- Novel formulation for blending arbitrary user and procedural inputs to create singularity-free navigation fields for global planning;
- Ease and simplicity to be combined with any existing local collision avoidance schemes;
- Interactive editing scheme that provides real-time feedback to the user to direct and control crowd simulations.

The overall approach can be useful from both artistic and data-driven perspectives, as it allows the user to interactively model some macroscopic phenomena and group dynamics based on artistic input or actual crowd footage.



(a) (b) (c)

Fig. 1: Subway simulations: *(a) An agent-based system causes congestion at the turnstiles. (b) The user specifies guidance fields (shown in red and blue) for different groups of agents. (c) Our algorithm computes goal-directed navigation fields to guide the agents towards their original goals.*

2 Previous Work

Simulating virtual crowds has been extensively studied in several fields including computer graphics, robotics, traffic engineering and social sciences. We refer the readers to excellent surveys [1], [2].

Many existing approaches employ agent models, in which each autonomous agent perceives its own state and reacts to dynamic entities in its neighborhood. LaValle [3] provides an excellent overview of global navigation methods. Several methods have been proposed for local collision avoidance, including the popular rule-based flocking model [4], [5], the *social forces* model introduced by Helbing et al. [6], cellular automata models [7] or a velocity-obstacles based formulation [8], [9]. More sophisticated variants account for motion dynamics [10], sociological factors [11], psychological effects [12], [13], situation-guided control [14] and cognitive and behavioral models [15], [16], [17].

Macroscopic approaches directly attempt to govern the global behavior of crowds by computing velocity fields based on the environment description [18], [19], designing velocity fields manually [20], [21], or by applying the continuum theory for the flow of crowds to crowd simulation [22]. Recently, several researchers have presented data-driven methods for constructing group behavior models based on real crowd video footage [23], [24], [25], [26], [27], [28]. Extracting coherent flow information from video for dense crowds is considered a challenging problem and few solutions have been proposed [29].

Ulciny et al. [30] and Sung et al. [14] provide a user with the ability to author and control crowd simulations at run-time. Reynolds [5] and Jin et al. [21] choose user-specified velocity fields to steer agents in a simulation. Metoyer et al. [31] and Oshita et al. [32] suggest a reactive path following scheme based on forces to direct individual agent trajectories. Kwon et al. [33] and Takahashi et al. [34] propose an interactive editing scheme that allows the user to edit group locomotion and formation patterns to generate high fidelity motions for offline crowd simulations. Moreover, some commercial animation systems [35], [36] allow the user to *paint* simple, directional flow fields on the scene to direct the flow of agents in a crowd. These prior approaches do not provide rigorous guarantees (highlighted in Section 3.1) in terms of goal-directed navigation.

Prior work in the domain of computer graphics has also focused on the use of user-specified vector fields for directing texture synthesis [37], [38], [39] and fluid simulations [40]. These methods allow the user to design *arbitrary* vector fields over planar and polyhedral domains. Our problem of designing vector fields for agent navigation is different since the navigation fields have to be goal-directed, free from singularities such as local minima (except at the goals) and encode agent paths of least effort to the goals.

3 Definitions and Background

In this section, we give an overview of guidance and navigation fields and describe our overall approach for directing crowd simulations.

3.1 Guidance and Navigation Fields

Our method relies on computing an underlying navigation field over the free space in an environment, which could be used for directing agents in a simulation. LaValle [3] provides an excellent overview of methods for computing navigation functions over discrete, as well as continuous spaces. Since trajectories of the agents are implicitly encoded in these fields, it is important that they exhibit the following key characteristics:

- In order to guarantee that the agents eventually reach their goals, the navigation field must be free from local minima, except for the presence of sinks at the specified goals.
- Agents should trace out paths of least effort (minimum cost) to their goals [41].
- These fields should be *almost* smooth [42] and should plan around static obstacles in the environment.

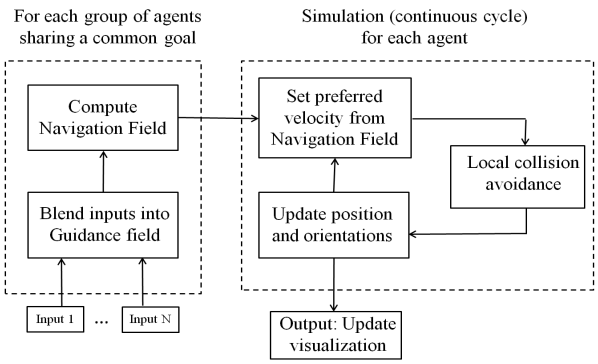


Fig. 2: A schematic overview of our method *For every distinct group of agents sharing a common objective, a goal-directed navigation field is interactively computed. This is used to drive individual agents at every simulation time-step. The agents run a continuous cycle of sensing and acting (as depicted by the right-hand box).*

We define a *navigation field* $\mathbf{N} : \mathbb{R}^2 \rightarrow \mathbb{S}^1, \mathbb{S}^1 = [0, 2\pi]/\sim$ (where $\sim$ is an equivalence relation with $0 \sim 2\pi$) as a vector field that exhibits the above properties. Note that the definition implies that the vectors of a navigation field are unit vectors.

Our work differs from previous methods in that we allow the user to arbitrarily create and edit such navigation fields, while still preserving these properties. To this end, we define a second vector field, called a *guidance field* $\mathbf{G} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, in which the user inputs are aggregated. We assume without loss of generality, that the vectors of a guidance field have a magnitude less than 1: $\|\mathbf{G}(\mathbf{x})\| < 1$ for all $\mathbf{x} \in \mathbb{R}^2$. A guidance field is in general not a goal-directed field and may contain local minima. We present an algorithm in Section 5.2 to transform a guidance field into a navigation field that is guaranteed to be free of local minima. Each time the user specifies additional input, it is blended with the existing guidance field, and interactively transformed into a new navigation field.

3.2 Overview

Our approach can be used with any underlying goal-directed agent-based simulation system. The goals may be dynamically generated at each time step based on decomposing a high-level objective. Given an environment description with static and dynamic obstacles and agents with specified goal positions, the task is to navigate each of these agents to its goal position without any collisions with the obstacles or other agents in the environment.

Our formulation is based on the two-level planning framework that is common to many agent-based systems [43], [2]. Figure 2 provides a schematic overview of our approach. At each simulation time-step, the preferred velocity of each agent is provided by the *global navigation* method (goal-directed navigation fields in our case). This is followed by a *local collision avoidance* phase which computes a *collision-free* velocity for each agent.

Our method computes navigation fields for each distinct group of agents based on the static description of the environment, specified goal positions, and the guidance field specified for that group of agents. These navigation fields are goal-directed, *almost* smooth with no local minima, and are used to guide the agents to their corresponding goals. The user can edit existing navigation fields either globally or on an individual basis and these edits are then composited with the existing guidance fields to compute a new navigation field. We note that the navigation fields are *not* recomputed in every time step of the simulation, but only when the user specifies additional input or when the goal position changes.

4 Specification of Guidance Fields

This section outlines the various methods that can be used to specify guidance fields using a sketch-based interface by the user or indirectly through motion fields extracted from an existing footage of real crowds. It should be noted that there can be other modes of input that might be used to specify guidance fields. The user might also choose to specify the guidance field procedurally or use a vector field editing method (e.g. [38] or [20]) to design them. It is also possible to specify guidance fields from multiple input sources, which can be blended together into one guidance field.

4.1 Sketch based Interface

We can use sketch based interface which allows the user to edit the underlying guidance field by "painting" vector fields onto the scene using freehand brush strokes (similar to the method proposed in [44]). The skeleton of the stroke is accepted as input using a mouse. We compute a region of influence of a user-specified width around the skeleton of the stroke. Depending on the application, the user can either choose to draw constant field strokes or the field could decay with the distance to the curve (see Figure 3). Parameters such as the width of the stroke and the rate of decay are specified by the user and vary depending on the application.

4.2 Input from Video Footage

Data-driven crowd simulation has received considerable attention in the recent years. Recently, [29] proposed a method to robustly detect global motion patterns from videos of crowded scenes (see Figure 5). Optical flow methods are used to compute flow vectors in each frame and then combined into a global motion field (see Figure 5(d)). Given the motion flow field, typical motion patterns are detected by clustering flow vectors using a

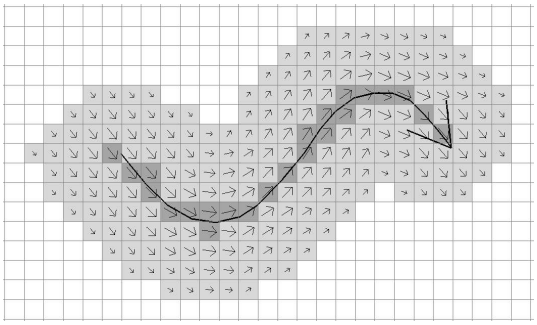


Fig. 3: Sketching interface for specifying gradients *A user drawn stroke is rasterized onto the regular grid and a region of influence of user-specified width is constructed around the skeleton of the stroke. The gradients along the sketched curve are propagated to grid cells within this region.*

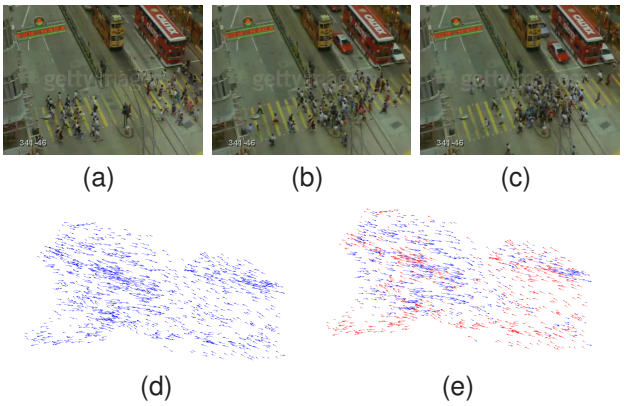


Fig. 5: *Motion patterns detected in a video of a crosswalk in Hong Kong. (a,b,c) Frames from the original video; (d) Global motion flow field; (e) Two distinct motion patterns detected by clustering (shown in blue and red).*

Fig. 4: Crosswalk Simulation: *(a) Sketch-based guidance fields to specify lane formation in the simulation; (b) Lane formation generated by goal-directed navigation fields.*

Fig. 6: Crosswalk Simulation: *(a) Video-based motion patterns from Figure 5(e) used as guidance fields; (b) Agent motion generated by the navigation fields.*

hierarchical clustering algorithm (see Figure 5(e)). These motion patterns are specified as flow fields on a discretized grid. We use the same grid used for defining the motion patterns as our simulation grid. These motion patterns can then be transformed (translated, rotated and scaled according to the simulated scene) and directly imported into our system as a guidance field, or blended with the existing guidance fields, to simulate the behavior observed in real-life crowds. The effectiveness of this approach is demonstrated in the results section.

4.3 Blending Multiple Inputs

Each source of input i gives a separate guidance field $\mathbf{G}_i$. In case there are multiple inputs, these guidance fields are *blended* together to form one composite guidance field $\mathbf{G}$ based on which a navigation field $\mathbf{N}$ is computed. Guidance fields are blended by weighted average of the vectors in the grid: $\mathbf{G}(X) = \sum_i w_i \mathbf{G}_i(X) \,/\, \sum_i w_i$, where w_i is the weight of input i. The blend weights are specified by the user during the editing process.

The user can edit the navigation fields at two levels. Global edits correspond to the specification of macroscopic phenomena and group behaviors such as formation of lanes and vortices (see Figure 10). Global edits apply to the guidance fields of all groups of agents in the simulation simultaneously. Edits at the individual level are limited to editing the individual guidance fields corresponding to each group of agents and they are used for higher level specification of agent trajectories (see Figure 9).

5 Navigation Fields

In this section, we describe our approach to blend different inputs and compute the navigation field from multiple guidance fields.

5.1 Discretization

Our choice of *vector fields* for global navigation is motivated by the fact that they span the entire free space available and can be edited to suit user specification. The processing and computation of guidance fields and navigation fields requires a discretization of free space in the environment. For simplicity, we use regular grids with a vector stored in each cell X of the grid. A regular grid also allows us to store information about the environment as a discrete cost function, where the cost of traversing each region depends on several factors such

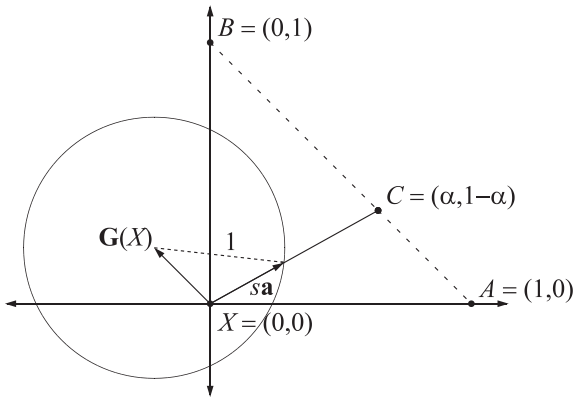


Fig. 7: Computing navigation fields *(a) Computing the path cost of cell X given the path costs at its two neighboring cells A and B and a guidance field vector $\mathbf{G}(X)$.*

as traversal time, safety risk or nature of the terrain. In our case, we label each cell as either 'free' or 'obstacle' and we assume that the free cells form a 4-connected component.

5.2 Computing Navigation Fields

A naïve combination of blended guidance fields with a goal-directed plan (generated by a global navigation method such as roadmaps, visibility graphs or potential fields) can result in ambiguities and cannot guarantee that agents would reach their respective goals. This is especially true when the guidance field specifies a flow that opposes the preferred direction of an agent's motion.

Computing a goal-directed navigation field $\mathbf{N}$ means that there should be a path from each free (non-obstacle) cell in the grid to (one of) the goal position(s). For this, we use a variant of Dijkstra's algorithm to propagate costs across the grid, starting with zero costs associated with the goal position(s). Storing the predecessor (i.e. back-pointer) for each grid cell during the propagation process yields a navigation field over the entire environment.

For the paths in the navigation field $\mathbf{N}$, we consider the guidance field $\mathbf{G}$ to be an extraneous factor that assists or hinders the intended progress towards the goal position(s). This means that the cost of traversing through a cell in the grid is *anisotropic*; it depends on the direction with which this cell is traversed with respect to the vector stored in $\mathbf{G}$.

The regular 4-connected grid can be considered as a graph, where each grid cell constitutes a node in the graph. Applying the traditional Dijkstra's algorithm to this grid results in "blocky" paths due to the limited set of permitted transitions (only axis-aligned transitions to neighboring grid cells), which gives undesirable behaviors. Therefore, our method is based on an approach suggested in the robotics literature [45] that uses an approximation to overcome this limitation and computes smooth paths. We extend this method to allow anisotropic cost functions.

The basic premise is that instead of choosing only from the set of neighbors at every grid cell, one can transition to any point in between the neighboring grid cells as well. To simplify the computation, we assume that the cost of any point between two neighboring cells is assumed to be a linear combination of the computed costs at the two cells.

For the sake of discussion, let us assume that X is the cell at which the path cost $T(X)$ needs to be evaluated and A and B are its north and east neighbors, respectively, that have already been *marked* (see Figure 7). Without loss of generality, we associate unit costs of traversal to the neighboring nodes A and B (in general, varying spatial costs of traversal can be associated with the neighboring nodes and this is specific to the simulation scenario). Instead of limiting possible transitions to only one of the neighboring cells, we assume that the optimal path taken from X meets the line segment $\overline{AB}$ at a point C lying between A and B (both inclusive). If the position $(\alpha, 1-\alpha)$ of C on $\overline{AB}$ is parameterized by α, where $\alpha \in [0,1]$, then the path cost associated with point C is assumed to be $\alpha T(A) + (1-\alpha)T(B)$. The *length* of the transition from X to C is given by $\|(\alpha, 1-\alpha)\|$. The speed s with which we can traverse $\overline{XC}$ depends on the guidance field vector $\mathbf{G}(X)$ at X. This scenario is similar to an airplane trying to move with unit speed in the presence of the wind blowing with velocity $\mathbf{G}(X)$. The speed s is given by the intersection of the line through $\overline{XC}$ and a unit circle centered at the point $\mathbf{G}(X)$ (see Figure 7):

$$\|s\mathbf{a} - \mathbf{G}(X)\| = 1, \tag{1}$$

where $\mathbf{a} = \frac{(\alpha,1-\alpha)}{\|(\alpha,1-\alpha)\|}$ is the direction of motion. As $\|\mathbf{G}(X)\| < 1$, this equation always has a positive solution, which is given by:

$$s(X,\mathbf{a}) = \mathbf{a}\cdot\mathbf{G}(X) + \sqrt{(\mathbf{a}\cdot\mathbf{G}(X))^2 - \mathbf{G}(X)\cdot\mathbf{G}(X) + 1}. \tag{2}$$

Hence, the path cost $T(X)$ at cell X is given as:

$$T(X) = \alpha T(A) + (1-\alpha)T(B) + \|(\alpha, 1-\alpha)\| \,/\, s(X,\mathbf{a}), \tag{3}$$

and the navigation field vector $\mathbf{N}(X)$ at cell X is:

$$\mathbf{N}(X) \;=\; \mathbf{a} \;=\; (\alpha, 1-\alpha) \,/\, \|(\alpha, 1-\alpha)\|. \tag{4}$$

We seek to compute the value for the parameter α that minimizes the path cost $T(X)$ at cell X. This is given by the solution to the equation $\frac{dT(X)}{d\alpha} = 0$. The path cost $T(X)$ is then computed by plugging in the value of α in Equation 3.

The path costs are propagated throughout the grid, starting at the goal locations, in a Dijkstra-like manner. At each instance of the computation, only two neighboring cells, for which the path costs have already been computed, are considered at a time for evaluating the

path cost at a given cell. Keeping track of the direction of the optimal path taken at each grid cell X yields a smooth navigation vector field over the entire free space in the environment.

We note that the anisotropic cost function as defined above is *non-monotonic*, i.e. it is possible that the new computed cost at the current cell is less than the cost of (at most) one of its neighboring cells. This may result in the cost at a cell being revised several times as the cell cost tries to converge to its true value. In our system, we allow the cost associated with a cell to be updated only up to a certain limit, after which we no longer process the cell. We observe that the convergence is fast and an upper limit of 5 produces a reasonably smooth navigation field in our benchmarks (see Figure 8(a)). In case $\mathbf{G}(X) = \mathbf{0}$ for all X, the navigation field gives the shortest paths to the goal (see Figure 8(b)).

5.3 Analysis

We highlight certain properties of the navigation field computed by our algorithm. Every grid cell whose cost T has been computed, has at least one neighbor with a lesser cost, except for the goal cells, from which the propagation is initialized with cost 0. So, there are no *local minima* in the navigation field, except for sinks at the goals. A direct consequence is that the navigation field always compute *goal-directed* paths. Given the fact that the free space is one connected component in the grid, the Dijkstra-algorithm will visit *all* the free cells and computes their navigation field vectors. Because we use an interpolation scheme to compute the path costs, the algorithm gives almost *smooth* paths. We note that grid cells next to sharp corners in obstacles may rely on only one neighboring cell to compute its cost, which can result in non-smoothness at these isolated points (see Figure 8(b)). Overall the navigation field computed by our algorithm has all the properties highlighted in Section 3.1.

The problem of navigation in the presence of an *anisotropic* guidance field belongs to a general class of anisotropic cost-optimal trajectory problems [46], characterized by the static Hamilton-Jacobi-Bellman equation:

$$\max_{\mathbf{a}\in\mathbb{S}^1}\{(-\nabla T(\mathbf{x})\cdot\mathbf{a})s(\mathbf{x},\mathbf{a})\} = 1, \tag{5}$$

with $s(\mathbf{x},\mathbf{a})$ as defined in Equation 2, and boundary conditions $T(\mathbf{x}) = 0$ for all goal locations $\mathbf{x}$. This equation is a necessary and sufficient condition for cost-optimal paths under an anisotropic speed function $s(\mathbf{x},\mathbf{a})$ that depends on position $\mathbf{x}$ as well as direction $\mathbf{a}$. In fact, our algorithm for computing navigation fields computes a discrete approximation of the solution of the given equation. It should also be noted that if the speed function s does *not* depend on the direction $\mathbf{a}$, then equation 5 reduces to the standard Eikonal equation [22]. We refer the reader to [46] for an elaborate analysis.

6 Local Collision Avoidance

Our method can be easily integrated into any multi-agent simulation system by replacing the global navigation framework with navigation field based path planning. At each simulation time-step, the cell containing the agent is determined. If $\mathbf{N}(X)$ be the navigation field at cell X and s_i^{pref} be the preferred speed of the agent a_i, then the preferred velocity is given by: $\mathbf{v}_i^{pref} = s_i^{pref}\mathbf{N}(X)$. In our implementation, we use a bilinear interpolation scheme to interpolate between the preferred direction vectors stored at the four closed neighboring cells instead of just a single cell, resulting in smoother paths to the goal for each agent.

This preferred velocity serves as an input to the local collision avoidance mechanism to compute a *collision-free* velocity for the agent. Our method can be easily combined with existing local collision avoidance methods. We outline two popular methods commonly used in multi-agent systems.

Social Force Model: The social forces model for pedestrian dynamics [6] is commonly used for collision avoidance in multi-agent systems. In this scheme, collision avoidance is achieved by means of forces acting on each agent. There is a repulsive social force component $\mathbf{F}^{soc}$ that prevents the agent from colliding with other agents and obstacles in the environment. In addition, there is also an attractive force $\mathbf{F}^{att}$ that guides the agent to the goal. This attractive force depends on the preferred velocity of the agent (given by the navigation field).

Let $\mathcal{A}$ denote the set of agents and $\mathcal{O}$ denote the set of obstacles in the simulation. The net force $\mathbf{F}(a_i)$ acting on an agent a_i at a given time-step is given by:

$$\mathbf{F}(a_i) = \mathbf{F}^{att}(a_i) + \sum_{a_j\in\mathcal{A}, j\neq i}\mathbf{F}_j^{soc}(a_i) + \sum_{o\in\mathcal{O}}\mathbf{F}_o^{soc}(a_i) \tag{6}$$

where,

$$\mathbf{F}^{att}(a_i) = m_i\frac{(s_i^{pref}\mathbf{N}(X) - \mathbf{v}_i)}{\tau} \tag{7}$$

Here, m_i denotes the mass of the agent a_i, $\mathbf{v}_i$ is the current velocity, $\mathbf{v}_i^{pref} = s_i^{pref}\mathbf{N}(X)$ is the preferred velocity and τ is the reaction time. The social forces acting on the agent $\mathbf{F}^{soc}(a_i)$ are computed as given in [6]. We refer the reader to the appendix and [6] for additional details.

Reciprocal Velocity Obstacles: Berg et al. [8], [9] propose a geometric framework for local collision avoidance. It takes as input the preferred velocity of the agent $\mathbf{v}_i^{pref}$ and returns an *optimal* collision-free velocity that minimizes the chosen *penalty* metric. The penalty of a candidate velocity $\mathbf{v}_i^{cand}$ depends on its deviation from the preferred velocity $\mathbf{v}_i^{pref} = s_i^{pref}\mathbf{N}(X)$ and the expected *time to collision* $tc(\mathbf{v}_i^{cand})$:

$$penalty(\mathbf{v}_i^{cand}) = w_i\frac{1}{tc(\mathbf{v}_i^{cand})} + ||\mathbf{v}_i^{pref} - \mathbf{v}_i^{cand}|| \tag{8}$$

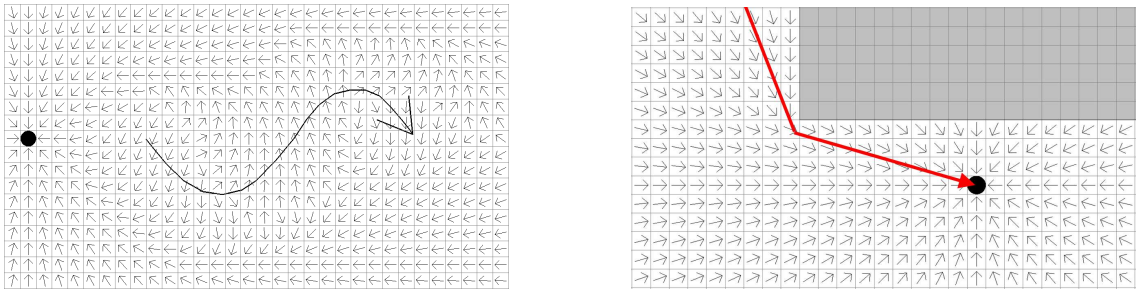


Fig. 8: Navigation fields *(a) A navigation field computed for the guidance field shown in Figure 3 (and indicated by the arrow here) with the goal (black dot) placed such that the guidance field points away from it. (b) If there is no guidance field, the navigation field gives the shortest paths to the goal (black dot). A shortest path is shown with a sharp turn at the obstacle (light gray) corner.*

for some user-defined weighting factor w_i. The expected time to collision $tc(\mathbf{v}_i^{cand})$ can easily be computed, given the definition of the Reciprocal Velocity Obstacles. A Monte-Carlo sampling over the set of admissible velocities is used to compute this velocity. We refer the reader to the appendix and [8], [9] for details. We used the publicly available RVO library [47] for all our experiments.

It is important to note that *none* of the collision avoidance methods suggested in literature can absolutely guarantee collision-free paths for all the agents in the simulation, but we found that the two methods outlined above perform well in practice, even in crowded scenarios and complex environments.

7 Results

In this section, we highlight the performance of our algorithm on different scenarios. In our current implementation, we use a simple 2D user interface for interactively specifying the guidance fields using a sketch-based interface. The user also has the option of importing flow fields (either extracted from video or designed procedurally) at the start of the simulation. The simulation results were recorded and visualized in real-time using the Horde3D graphics engine [48].

We demonstrate some of the benefits of our approach in different scenarios. The accompanying video illustrates several situations in which our approach was successfully used to direct simulations.

Four Blocks: This scenario comprises of four groups of 25 agents each in each corner of the environment that need to move to the opposite corner. In the middle, there are four square-shaped static obstacles that form narrow passages. Existing agent-based systems cause congestion in the center. The user directs the simulation (either on a individual group level as in Fig. 9 or on a global scale as in Fig. 10) to resolve this congestion. Depending on the user input, the user can get different crowd flows through the blocks.

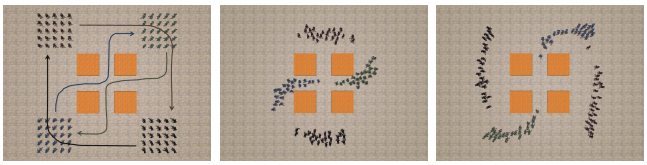


Fig. 9: Group level editing: *(a) Individual guidance fields for each of the four groups of agents; (b, c) Intermediate positions of the agents using the computed navigation field.*

Crossing: This simulation shows four streams of agents meeting at a crossing. The user sketches alternative trajectories to generate interesting behaviors at the crossing and allowing for more streamlined flow (as illustrated in the accompanying video).

Subway: Fig. 1 shows the simulation of a crowd of 435 agents entering/exiting a subway station. The initial positions and goals of the agents are known. However, opposing flows of agents at the turnstiles leads to congestion and undesirable behavior. Local collision avoidance methods are unable to resolve this congestion. We successfully demonstrate how our interface can be used to specify guidance fields (both global and individual as in Fig. 1(b)) to direct agent flows and can generate natural looking behavior.

Crosswalk: Fig. 4 shows pedestrians crossing at a regular crosswalk. Lane-formation is an emergent behavior that is commonly observed in such situations [22]. We use the motion patterns extracted from a video (see Fig. 5) to replicate the motion and behavior of the agents in our simulation. The flow fields extracted from the video footage are static and typically noisy. In such cases, one could also augment it with user-provided guidance fields to get the desired behavior. Our approach is able to direct the flow of agents and generate multiple lane formation. The user can have precise control over the lane formation for streamlined flow. The width of the brush stroke can also be used to control the dispersion

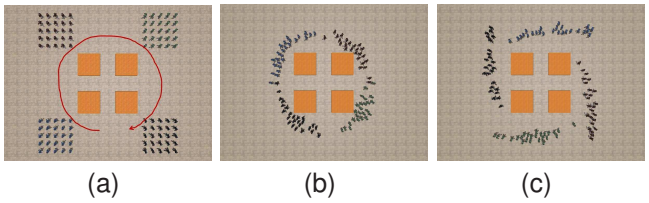


Fig. 10: Global guidance fields: *(a) A global guidance field applied to all the agents; (b, c) Intermediate positions of the agents in the simulation using the computed navigation field.*

of agents in a group.

7.1 Performance

Table 1 summarizes the performance of our system on four demo scenarios (as measured on a single Intel Xeon 2.66GHz processor). The third column indicates the average time to compute the navigation field (per user edit) and the fourth column indicates the average time taken per simulation frame. The results indicate that the navigation fields can be computed in real-time, even for complex environments such as the one shown in the subway simulation scenario. The runtime complexity of computing the navigation field is O(mnlogmn), where $m \times n$ is the grid dimension (similar to Dijkstra's algorithm). This complexity is only dependent on the grid resolution and is independent of factors such as the complexity of arrangement of obstacles in the environment, the number of agents participating in the simulation or the crowd density.

The agent-based simulation methods used (Helbing [6] and RVO [8], [9]) exhibit *almost* linear runtime behavior in the number of agents, per simulation time-step. This can be attributed to the fact that interaction with only a fixed, constant number of *nearby* agents is considered for computing the collision-free velocity for an agent at each simulation time-step. We refer the reader to [6], [9] for additional details.

Scene	#Agents	Grid dimensions	Local collision avoidance	Average NF time (msec)	Average sim. time (msec/frame)
Four Blocks	100	100x100	RVO	5.0	2.0
Crossing	640	100x100	Helbing	5.0	1.1
Crosswalk	145	225x100	Helbing	13.0	0.4
Subway	435	200x200	RVO	22.0	5.5

TABLE 1: *Performance of our approach on the four demo scenarios. The results indicate our approach is capable of computing navigation fields in real-time and incurs very little overhead in terms of simulation time (measured in milliseconds).*

8 Comparison and Analysis

We compare some features of our approach with prior literature and also highlight some limitations.

8.1 Comparison

Many prior works have addressed the problem of simulating virtual crowds, either using vector fields or user-driven control. We present a novel unifying framework to seamlessly composite (anisotropic) guidance fields from multiple sources to compute navigation fields for each group of agents (or each agent). This allows for more fine-grained control over the simulations in comparison to previous approaches which use a single global vector field for directing crowd flow. Our method also allows the user to edit the simulations at interactive rates.

Reynolds [5] extended the flocking model for agents from [4] to incorporate simple steering behaviors for individual agents. One of the proposed features allows the user to draw flow fields that can be used to steer agents. This method is primarily designed for flocking simulations and does not address the broader issue of goal-directed navigation or singularities in all kind of navigation fields.

Chenney [20] proposed an approach to design divergence-free velocity fields on a planar domain for simulating fluid-like phenomena and crowds. The divergence-free property serves the primary purpose of collision avoidance among agents, but is not applicable to goal-directed navigation. The field is also constructed on the basis of a limited number of template flow tiles that are stitched together and the approach is not interactive. Jin et al. [21] use a scattered RBF interpolation scheme to compute a global vector field over the the entire domain for guiding agents. This alone is not sufficient in order to prevent any occurrences of singularities in the navigation field. Our approach guarantees that there are no singularities in the navigation vector field (except for local minima at the goal positions).

The continuum crowd formulation [22] solves the Eikonal equation at every time-step of the simulation, per group of agents. This can become expensive for a simulation with a large number of groups. In contrast, our approach only recomputes the navigation vector field corresponding to a user-edit or when a new guidance field is specified. Since our method uses a per-agent collision avoidance scheme, our approach can be easily combined with other agent-based methods, suited for simulating heterogeneous crowds. Our method is also capable of compositing *both* isotropic and anisotropic costs as compared to the continuum crowd approach. Recently, Paravisi et al. [49] extended this work by learning the parameters involved from individual trajectories of agents extracted from video footage. This approach has been demonstrated on video footage of sparse scenes, whereas our approach is not limited by the density of agents in the scene.

Kwon et al. [33] presented an interactive editing scheme

for a user to edit group locomotion based on mesh deformation. This approach is primarily designed for generating high fidelity motions for offline simulations of crowds, and has a different goal as compared to our algorithm. Our approach can be considered to be complementary to the work of Takahashi et al. [34] on controlling macroscopic arrangements of agents in a simulation. Other algorithms [30], [14] allow the user to specify agent attributes, actions, emotion states and certain group behaviors but these methods do not address the issue of navigation.

We extend the approach suggested by Ferguson et al. [45] to incorporate anisotropic costs (using the framework outlined in [46]) to compute cost-optimal agent trajectories in the presence of user-specified guidance fields. To the best of our knowledge, this is the first attempt at robustly integrating user input (in the form of user-drawn strokes, flow fields extracted from video or procedurally defined guidance fields) with a multi-agent system to provide a unifying framework to direct crowd simulations. We provide rigorous guarantees in terms of goal-directed navigation of agents. Our approach is general and applicable to a variety of existing local collision avoidance methods.

8.2 Limitations

Our approach has some limitations. We make some assumptions about the underlying multi-agent simulation system, such as global goal-directed navigation and computation of preferred velocity for each agent. Moreover, local collision avoidance is also performed by the underlying algorithm and governs the behavior of all agents. Currently, we use a simple 2D mouse-based interface and it can be hard to select a particular agent or a group of agents in a dense scenario and specify guiding trajectories for such groups. We use a regular grid to partition the free space and it can have high memory cost and runtime overhead in the case of heterogeneous crowds. Finally, it may not be possible to generate all kinds of macroscopic behaviors, social interactions or crowd motion patterns using our approach. There might be situations where a combination of a simple agent-based model and our method may still not result in the desired behavior and more sophisticated behavior models might be required.

9 Conclusion and Future Work

We presented an intuitive approach to direct simulation of virtual crowds using goal-directed navigation functions. We have successfully demonstrated the approach for a wide variety of simulation to generate different macroscopic behaviors and natural looking motion patterns, resolve congestion and perform goal-directed navigation. Overall, this approach offers a simple, yet powerful method to direct or control crowd simulations.

There are several possible avenues for future work in this area. Higher-level behavioral specification in terms of guidance fields is essential for authoring realistic crowd simulations. Our current framework could be augmented with behaviors such as following, queueing or social interactions to allow for a more diverse range of permitted behaviors. We would like to combine our approach with other multi-agent simulation algorithms that use cognitive modeling techniques. We may also wish to include other information in the cost formulation such as density of agents (e.g. continuum formulation). Our method could be used in combination with a better local collision avoidance schemes [28], [27], [50] and character animation schemes [33], [2] to generate more realistic behaviors. The flow fields extracted from crowd footage are currently static but dynamic flow fields can be incorporated into our approach as well and yield potentially better results.

Acknowledgments

This research was supported in part by ARO contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134, 0429583, and 0404088, DARPA/RDECOM contract N61339-04-C-0043, Intel, Carolina Development, and Disney. We thank Min Hu, Saad Ali and Prof. Mubarak Shah ([29]) for making the video and motion flow field data for the Hong Kong pedestrian crossing available to us.

References

[1] D. Thalmann, C. O'Sullivan, P. Ciechomski, and S. Dobbyn, *Populating Virtual Environments with Crowds*. Eurographics Tutorial Notes, 2006.

[2] N. Pelechano, J. M. Allbeck, and N. I. Badler, *Virtual Crowds: Methods, Simulation and Control*. Morgan and Claypool Publishers, 2008.

[3] S. M. LaValle, *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/), 2006.

[4] C. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, pp. 25–34, 1987.

[5] ——, "Steering behaviors for autonomous characters," *Game Developers Conference*, 1999.

[6] D. Helbing, I. Farkas, and T. Vicsek, "Simulating dynamical features of escape panic," *cond-mat/0009448*, Sep. 2000, nature 407, 487-490 (2000). [Online]. Available: http://arxiv.org/abs/cond-mat/0009448

[7] A. Kirchner and A. Schadschneider, "Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics," *Physica A*, vol. 312, no. 1-2, pp. 260–276, September 2002.

[8] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for realtime multi-agent navigation," *Proc. of IEEE Conference on Robotics and Automation*, pp. 1928–1935, 2008.

[9] J. van den Berg, S. Patil, J. Sewall, D. Manocha, and M. Lin, "Interactive navigation of multiple agents in crowded environments," in *Proc. Symposium on Interactive 3D Graphics and Games*, 2008, pp. 139–147.

[10] D. C. Brogan and J. K. Hodgins, "Group behaviors for systems with significant dynamics," *Autonomous Robots*, vol. 4, pp. 137–153, 1997.

[11] S. R. Musse and D. Thalmann, "A model of human crowd behavior: Group inter-relationship and collision detection analysis," *Computer Animation and Simulation*, pp. 39–51, 1997.

[12] T. Sakuma, T. Mukai, and S. Kuriyama, "Psychological model for animating crowded pedestrians: virtual humans and social agents," in *Computer Animation Virtual Worlds*, vol. 16, 2005, pp. 343–351.

[13] N. Pelechano, J. M. Allbeck, and N. I. Badler, "Controlling individual agents in high-density crowd simulation," *Proc. of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pp. 99–108, 2007.

[14] M. Sung, M. Gleicher, and S. Chenney, "Scalable behaviors for crowd simulation," *Computer Graphics Forum*, vol. 23, no. 3 (Sept), pp. 519–528, 2004.

[15] W. Shao and D. Terzopoulos, "Autonomous pedestrians," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005, pp. 19–28.

[16] Q. Yu and D. Terzopoulos, "A decision network framework for the behavioral animation of virtual humans," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2007, pp. 119–128.

[17] S. Paris, J. Pettre, and S. Donikian, "Pedestrian reactive navigation for crowd simulation:a predictive approach," in *Computer Graphics Forum : Eurographics*, 2007, pp. 665–674.

[18] B. Yersin, J. Maim, P. Ciechomski, S. Schertenleib, and D. Thalmann, "Steering a virtual crowd based on a semantically augmented navigation graph," in *VCROWDS 2005*, 2005.

[19] J. Pettré, H. Grillon, and D. Thalmann, "Crowds of moving objects: navigation planning and simulation," in *SIGGRAPH '08: ACM SIGGRAPH classes*, 2008, pp. 1–7.

[20] S. Chenney, "Flow tiles," in *Proc. ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2004, pp. 233–242.

[21] X. Jin, J. Xu, C. C. L. Wang, S. Huang, and J. Zhang, "Interactive control of large crowd navigation in virtual environment using vector field," in *IEEE Computer Graphics and Application*, vol. 28, no. 6, 2008, pp. 37–46.

[22] A. Treuille, S. Cooper, and Z. Popovic, "Continuum crowds," *Proc. of ACM SIGGRAPH*, pp. 1160 – 1168, 2006.

[23] H. Lee, M. Choi, Q. Hong, and J. Lee, "Group behavior from video: a data-driven approach to crowd simulation," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2007, pp. 109–118.

[24] A. Lerner, Y. Chrysanthou, and D. Lischinski, "Crowds by example," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 26, no. 3, pp. 655–664, 2007.

[25] S. R. Musse, C. R. Jung, J. C. S. Jacques, and A. Braun, "Using computer vision to simulate the motion of virtual agents," in *Computer Animation Virtual Worlds*, 2007, pp. 83–93.

[26] N. Courty and T. Corpetti, "Crowd motion capture," in *Computer Animation Virtual Worlds*, vol. 18, 2007, pp. 361–370.

[27] J. Pettré, J. Ondrej, A. Olivier, A. Cretual, and S. Donikian, "Experiment-based modeling, simulation and validation of interactions between virtual walkers," in *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2009, pp. 189–198.

[28] A. Lerner, E. Fitusi, Y. Chrysanthou, and D. Cohen-Or, "Fitting behaviors to pedestrian simulations," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2009, pp. 199–208.

[29] M. Hu, S. Ali, and M. Shah, "Learning motion patterns in crowded scenes using motion flow field," in *IEEE International Conference on Pattern Recognition (ICPR)*, 2008, pp. 1–5.

[30] B. Ulicny, O. Ciechomski, and D. Thalmann, "Crowdbrush: Interactive authoring of real-time crowd scenes," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004, pp. 243–252.

[31] R. A. Metoyer and J. Hodgins, "Reactive pedestrian path following from examples," in *Proc. Computer Animation and Social Agents*, 2003, pp. 149–156.

[32] M. Oshita and Y. Ogiwara, "Sketch-based interface for crowd animation," in *Proc. of 10th Smart Graphics*, 2009, pp. 253–262.

[33] T. Kwon, K. H. Lee, J. Lee, and S. Takahashi, "Group motion editing," in *Proceedings of ACM SIGGRAPH*, 2008, pp. 1–8.

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

[35] L. Prasso, J. Buhler, and J. Gibbs, "The PDI crowd system for ANTZ," in *ACM SIGGRAPH '98 Conference abstracts and applications*, 1998, p. 313.

[36] MASSIVE, "http://www.massivesoftware.com," 2006.

[37] J. J. van Wijk, "Image based flow visualization," in *SIGGRAPH 2002: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002, pp. 745–754.

[38] E. Zhang, K. Mischaikow, and G. Turk, "Vector field design on surfaces," *ACM Transactions on Graphics*, vol. 25, no. 4, pp. 1294–1326, 2006.

[39] M. Fisher, P. Schroder, M. Desbrun, and H. Hoppe, "Design of tangent vector fields," *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, vol. 26, no. 3, 2007.

[40] J. Stam, "Flows on surfaces of arbitrary topology," in *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 2003, pp. 724–731.

[41] G. Still, "Crowd dynamics," Ph.D. dissertation, University of Warwick, UK, 2000.

[42] G. Arechavaleta, J.-P. Laumond, H. Hicheur, and A. Berthoz, "An optimality principle governing human walking," *IEEE Trans. on Robotics*, vol. 24, no. 1, pp. 5–14, 2008.

[43] O. B. Bayazit, J. Lien, and N. M. Amato, "Roadmap-based flocking for complex environments," in *Proc. Pacific Conference on Computer Graphics and Applications*, 2002, pp. 104–113.

[44] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang, "Interactive procedural street modeling," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 1–10, 2008.

[45] D. Ferguson and A. Stentz, "Field D*: An interpolation-based path planner and replanner," *Springer Tracts in Advanced Robotics*, vol. 28, pp. 239–253, 2007.

[46] J. Sethian and A. Vladimirsky, "Ordered upwind methods for static hamilton-jacobi equations: theory and algorithms," *SIAM Journal of Numerical Analysis*, vol. 41, no. 1, pp. 325–363, 2003.

[47] RVOLibrary, "RVO Library: Reciprocal velocity obstacles for real-time multi-agent simulation, http://gamma.cs.unc.edu/RVO/Library/index.html," 2008.

[48] Horde3D, "http://www.horde3d.org," 2007.

[49] M. Paravisi, A. Werhli, J. J. Junior, R. Rodrigues, C. R. Jung, and S. R. Musse, "Continuum crowds with local control," in *Computer Graphics International*, 2008, pp. 108–115.

[50] S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey, "Clearpath: highly parallel collision avoidance for multi-agent simulation," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2009, pp. 177–187.

APPENDIX

LOCAL COLLISION AVOIDANCE

9.1 Social Forces Model

The social forces model for pedestrian dynamics [6] is commonly used for collision avoidance in multi-agent systems. In this scheme, collision avoidance is achieved by means of forces acting on each agent. There is a repulsive social force component $\mathbf{F}^{soc}$ that prevents the agent from colliding with other agents and obstacles in the environment. In addition, there is also an attractive force $\mathbf{F}^{att}$ that guides the agent to the goal. This attractive force is based on the preferred velocity of the agent (which is provided by the navigation field).

Let $\mathcal{A}$ denote the set of agents and $\mathcal{O}$ denote the set of obstacles in the simulation. The net force $\mathbf{F}(a_i)$ acting on an agent a_i at a given time-step is given by:

$$\mathbf{F}(a_i) = \mathbf{F}^{att}(a_i) + \sum_{a_j \in \mathcal{A}, j \neq i} \mathbf{F}_j^{soc}(a_i) + \sum_{o \in \mathcal{O}} \mathbf{F}_o^{soc}(a_i) \quad (9)$$

where,

$$\mathbf{F}^{att}(a_i) = m_i \frac{(\mathbf{v}_i^{pref} - \mathbf{v}_i)}{\tau} \quad (10)$$

$$\mathbf{F}_j^{soc}(a_i) = \alpha e^{\frac{(r_i + r_j - ||\mathbf{p}_i - \mathbf{p}_j||)}{\beta}} \mathbf{n}_{ij} \quad (11)$$

$$\mathbf{F}_o^{soc}(a_i) = A e^{\frac{(r_i - d_{io})}{B}} \mathbf{n}_{io} \quad (12)$$

Here, m_i denotes the mass of the agent, r_i is the radius, $\mathbf{p}_i$ is the current position, $\mathbf{v}_i$ is the current velocity, $\mathbf{v}_i^{pref}$ is the preferred velocity (given by the navigation field), τ is the reaction time, α is a social scaling constant, β is the agent's personal space drop-off constant, $\mathbf{n}_{ij}$ is the normal direction between a_i and a_j, A is an obstacle scaling constant, B is the obstacle distance drop-off constant, and $\mathbf{n}_{io}$ is the vector from a_i to the nearest point on obstacle o. For efficient computation of these forces, we compute forces to agents and obstacles within a predetermined radius R_i. We use a numerical integration scheme to compute the agent's velocity after applying the forces.

Based on these forces and reasoning about other agents, additional agent behaviors can also be incorporated into this framework. For example, aggressiveness or urgency can be described by increased scaling constants for the attractive forces towards the goal and repulsive forces. Moreover, behavior such as following and queuing can be simulated by adding attractive forces that are generated by the appropriate agents.

In order to resolve contacts with a colliding agent or obstacle, additional forces are applied to each agent in collision. A pushing force, $\mathbf{F}^{push}$, acts to force a separation between agents and a frictional force, $\mathbf{F}^{fric}$ simulates the act of slowing down due to a collision. Unlike the associated repulsive force, these forces are only applied when a collision occurs. For agent a_i and an intersecting agent a_j , the following forces are added to Equation 9:

$$\mathbf{F}_j^{push}(a_i) = \kappa(r_i + r_j - ||\mathbf{p}_i - \mathbf{p}_j||)\mathbf{n}_{ij} \quad (13)$$

$$\mathbf{F}_j^{fric}(a_i) = \lambda||\mathbf{F}_j^{push}(a_i)||\mathbf{t}_{ij} \quad (14)$$

where κ is a pushing spring constant, λ is a sliding friction constant, and $\mathbf{t}_{ij}$ is the tangent vector to $\mathbf{n}_{ij}$. Contacts with obstacles are resolved in a similar fashion. We used the parameter values given in [6] for all our experiments and we refer the reader to the paper for additional details.

9.2 Reciprocal Velocity Obstacles

Berg et al. [8], [9] proposed a geometric framework for multi-agent collision avoidance using *Reciprocal Velocity Obstacles*. Each agent in the simulation, a_i has position $\mathbf{p}_i$, velocity $\mathbf{v}_i$, and geometric shape g_i associated with it. At each time-step of the simulation, the local collision avoidance scheme based on the RVO algorithm takes into account the position and velocity of agents in a fixed neighborhood of a_i to compute a new *collision-free* velocity and direction of motion in the following manner.

For the sake of illustration, consider two agents, A and B. Let $\mathbf{p}_A$ and $\mathbf{p}_B$ be their current positions and $\mathbf{v}_A$ and $\mathbf{v}_B$ be the current velocities, respectively. Let $\lambda(\mathbf{p}, \mathbf{v})$ define the ray shot from point $\mathbf{p}$ in the direction along $\mathbf{v}$ (i.e. $\lambda(\mathbf{p}, \mathbf{v}) = \mathbf{p} + t\mathbf{v}$). Moreover, $g_A \oplus g_B$ denotes the *Minkowski* sum of two geometric primitives g_A and g_B, i.e. $g_A \oplus g_B = \{\mathbf{x}_A + \mathbf{x}_B | \mathbf{x}_A \in g_A, \mathbf{x}_B \in g_B\}$. Let $-g_A$ denote the shape g_A reflected in its reference point, i.e. $-g_A = \{-\mathbf{x}_A | \mathbf{x}_A \in g_A\}$. The reciprocal velocity obstacle RVO_B^A that agent B induces on agent A is defined as follows:

$$RVO_B^A = \{\mathbf{v}'_A | \lambda(\mathbf{p}_A, 2\mathbf{v}'_A - \mathbf{v}_A - \mathbf{v}_B) \cap g_B \oplus -g_A \neq \emptyset\}. \quad (15)$$

RVO_B^A represents the set of all velocities of agent$A$ that would lead to a potential collision with agent B at some point ahead in time. If agent A chooses a new velocity outside RVO_B^A and agent B chooses a new velocity outside RVO_A^B, the agents are *guaranteed* to have chosen a collision-free and oscillation-free trajectory in this case (see Figure 11).

In the context of multi-agent navigation, the RVO formulation is applied as follows to each agent independently. Among the set of admissible velocities for each agent, the RVO algorithm selects the one with the minimum *penalty*. The penalty of a candidate velocity $\mathbf{v}_i^{cand}$ depends on its deviation from the preferred velocity $\mathbf{v}_i^{pref}$ and the expected *time to collision* $tc(\mathbf{v}_i^{cand})$:

$$penalty(\mathbf{v}_i^{cand}) = w_i \frac{1}{tc(\mathbf{v}_i^{cand})} + ||\mathbf{v}_i^{pref} - \mathbf{v}_i^{cand}|| \quad (16)$$

for some user-defined weighting factor w_i, where w_i can vary among the agents to reflect differences in personal preferences such as aggressiveness and shyness.

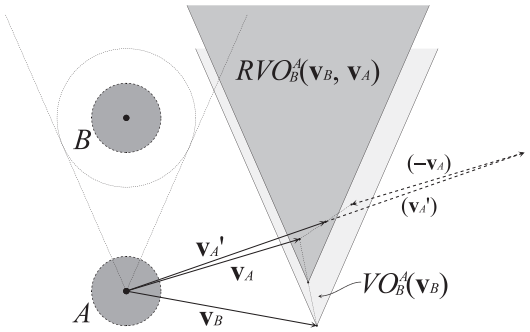


Fig. 11: The Reciprocal Velocity Obstacle $RVO^A_B(\mathbf{v}_B, \mathbf{v}_A)$ *of agent* B *to agent* A*. It is used for local collision avoidance. We compute this locally for each agent during each simulation cycle.*

The expected time to collision $tc(\mathbf{v}_i^{cand})$ can be easily computed, given the definition of the Reciprocal Velocity Obstacles (based on other agents and obstacles in the environment).

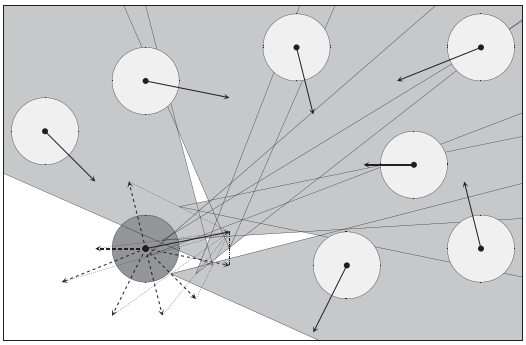


Fig. 12: *The combined reciprocal velocity obstacle for the agent (dark) is the union of the individual reciprocal velocity obstacles of the other agents.*

Overall, given a set of agents $\mathcal{A} = \{a_1, a_2, \ldots a_n\}$, the function RVO$(a_i, \mathcal{A} - a_i)$ returns the optimal collision-free velocity for agent a_i for the next simulation cycle, as shown in figure 12. It should be noted that computing the union of all the Reciprocal Velocity Obstacles $\{RVO^{a_i}_{a_1}, RVO^{a_i}_{a_2} \ldots RVO^{a_i}_{a_n}\}$ is not feasible computationally and we use a Monte-Carlo sampling scheme in velocity space to compute the desired velocity.For details on this function, we refer the readers to [8], [9]. We used the publicly available RVO library [47] for all our experiments.

Sachin Patil received his B.Tech degree in Computer Science and Engineering from the Indian Institute of Technology, Bombay, in 2006. He is currently a Ph.D. candidate in the Department of Computer Science at the University of North Carolina at Chapel Hill. His research interests include motion and path planning in virtual environments, and medical robotics.

Jur van den Berg received his M.S. degree from the University of Groningen, and his Ph.D. degree from Utrecht Univerisity, The Netherlands, in 2003 and 2007, respectively. From 2007 to 2009 he was a postdoctoral researcher at the University of North Carolina at Chapel Hill. Currently he is a postdoctoral researcher at the University of California at Berkeley. His research interests include motion and path planning, navigation of virtual characters, and medical robotics.

Sean Curtis received a BA in German from Brigham Young University, a BS in Computer Science from the University of Utah, and is currently a Ph.D. candidate in the Department of Computer Science at the University of North Carolina at Chapel Hill. His current research interests include character animation, crowd simulation and collision detection. He also has over 10 years of experience working in all facets of the digital animation and entertainment industry.

Ming C. Lin is currently the Beverly W. Long Distinguished Professor of Computer Science at the University of North Carolina at Chapel Hill. Her research interests include physically-based modeling, haptics, robotics, real-time 3D graphics for virtual environments, geometric computing, and distributed interactive simulation. She has (co-)authored more than 190 refereed scientific publications, co-edited/authored three books, including "Applied Computation Geometry" by Springer-Verlag, "High-Fidelity Haptic Rendering" by Morgan-Claypool, and "Haptic Rendering: Foundations, Algorithms and Applications" by A.K. Peters.

She has received several honors and awards, including the NSF Young Faculty Career Award in 1995, Honda Research Initiation Award in 1997, UNC/IBM Junior Faculty Development Award in 1999, UNC Hettleman Award for Scholarly Achievements in 2002, Carolina Women's Center Faculty Scholar in 2008, Carolina's WOWS Scholar 2009–2011, and 6 best paper awards. She has served as a program committee member for over 90 leading conferences on virtual reality, computer graphics, robotics, haptics and computational geometry and co-chaired over 20 international conferences and workshops. She is the Associate Editor-in-Chief of IEEE Transactions on Visualization and Computer Graphics (TVCG). She also has served as an associate editor and guest editor of over 15 journals and magazines.

Dinesh Manocha is currently a Phi Delta Theta/Mason Distinguished Professor of Computer Science at the University of North Carolina at Chapel Hill. He was selected an Alfred P. Sloan Research Fellow, received NSF Career Award in 1995 and Office of Naval Research Young Investigator Award in 1996, Honda Research Initiation Award in 1997, and Hettleman Prize for scholarly achievement at UNC Chapel Hill in 1998. He has received more than 13 best paper awards at leading conferences.

His research interests include geometric computing, interactive computer graphics, physics-based simulation and robotics. He has published more than 270 papers in these areas. Some of the software systems developed by his group on collision and geometric computations, interactive rendering, and GPU-based algorithms have been widely downloaded and used by leading commercial vendors. He has served as a program committee member or program chair for more than 75 leading conferences and also served as a guest editor or member of editorial board of ten leading journals. He has supervised 40 MS and Ph.D. students and has served as a PI or Co-PI on more than 55 grants. His research has been sponsored by AMD/ATI, ARO, DARPA, Disney, DOE, Honda, Intel, Microsoft, NSF, NVIDIA, ONR, RDECOM and Sloan Foundation.