Real-time Path Planning for Virtual Agents in Dynamic Environments

Avneesh Sud Erik Andersen Sean Curtis Ming Lin Dinesh Manocha

{sud,andersen,seanc,lin,dm}@cs.unc.edu
Department of Computer Science, University of North Carolina at Chapel Hill

ABSTRACT

We present a novel approach for real-time path planning of multiple virtual agents in complex dynamic scenes. We introduce a new data structure, *Multi-agent Navigation Graph* (MaNG), which is constructed from the first- and second-order Voronoi diagrams. The MaNG is used to perform route planning and proximity computations for each agent in real time. We compute the MaNG using graphics hardware and present culling techniques to accelerate the computation. We also address undersampling issues for accurate computation. Our algorithm is used for real-time multi-agent planning in pursuit-evasion and crowd simulation scenarios consisting of hundreds of moving agents, each with a distinct goal.

Keywords: crowd simulation, Voronoi diagram, motion planning.

Index Terms: I.3.5 [Computing Methodologies]: Computational Geometry and Object Modeling—Geometric algorithms; I.3.7 [Computing Methodologies]: Three-Dimensional Graphics and Realism—Animation, virtual reality

1 INTRODUCTION

Crowds, ubiquitous in the real world from groups of humans to schools of fish, are vital features to model in a virtual environment. Realistic simulation of virtual crowds have diverse applications in architecture design, emergency evacuation, urban planning, personnel training, education and entertainment. Existing work in this area can be broadly classified into *agent-based methods* that focus more on individual behavior, or *crowd simulations* that aim to exhibit emergent phenomena of the groups.

In this paper, we address the problem of collision-free path computation for agents moving in a complex virtual environment. Since individuals constantly adjust their behavior according to dynamic factors (e.g. another approaching individual) in the environment, agent-based techniques that focus on modeling individual behaviors and intents offer many attractive benefits. They often result in more realistic and detailed simulations. One of the key challenges in a large-scale agent-based simulation is global path planning for each virtual agent. The path planning problem can become very challenging for real-time applications with a large group of moving virtual characters, as each character is a dynamic obstacle for other agents. Many prior techniques are either restricted to static environments or perform local collision avoidance computations. The latter can result in unnatural behavior or "getting stuck" in local minima. These problems tend to be more visible in a dynamically changing scene with multiple moving virtual agents.

Main Results: In this paper, we present a novel, real-time algorithm for path planning of multiple virtual agents in a dynamic environment. We introduce a new data structure called "multi-agent navigation graph" or MaNG and compute it efficiently using GPU-accelerated discrete Voronoi diagrams. Voronoi diagrams have been widely used for path planning computations in static environments [6, 20] and we extend these approaches to dynamic environments.

Voronoi diagrams encode the connectivity of the space and provide a path of maximal clearance for a robot from other obstacles. In order to use them for multiple moving agents in a dynamic scene, prior approaches compute the Voronoi diagram for each agent separately by treating the other agents and the environment as obstacles. This approach can become very costly as the number of virtual agents increases. Instead, we compute the *second order* Voronoi diagram of all the obstacles and agents, and show that the second order Voronoi diagram provides *pairwise* proximity information for all the agents simultaneously. Therefore, we combine the first and second order Voronoi graphs to compute the MaNG for global path planning of multiple virtual agents.

The MaNG computes paths of maximal clearance for a group of moving agents with different goals *simultaneously* and does not require a separate path planning data structure for each virtual agent. Furthermore, we compute a discrete approximation to this graph structure by using the rasterization hardware and propose an adaptive culling technique to accelerate the computation. We also address the undersampling issues that arise due to discretization. Some of our key results include:

1. A new global data structure, the "multi-agent navigation graph" (MaNG) for parallel computation of maximal clearance paths among multiple virtual agents moving independently;
2. Interactive global path planning and local collision avoidance for multiple virtual agents, each with possibly different goals, in a complex virtual environment;
3. A fast two-pass algorithm with adaptive culling techniques for computing a discrete MaNG using GPUs;
4. Resolving undersampling issues in discrete graph computation.

The resulting technique is scalable for global path planning of many dynamic agents in a complex virtual world, not necessarily moving in groups. Although our approach is specifically well suited for simulating multiple virtual agents with distinct intentions, it can also be used in conjunction with a crowd simulation. We have demonstrated our algorithm on two challenging scenarios: a pursuit-evasion game of many fruit pickers chased by farmers and crowd simulation. In both these environments, our algorithm is able to perform real-time global path planning and collision avoidance simultaneously for hundreds of virtual agents with distinct goals.

Organization: The rest of the paper is organized as follows. Section 2 reviews prior literature in related areas. In Section 3, we define our notation and give an overview of our approach. We introduce our data structure, MaNG, and show how it can be used for path planning of multiple agents in Section 4. Section 5 describes our efficient algorithm to compute the MaNG in real-time using GPUs. We describe the implementation and highlight two applications of our planning algorithm to complex virtual environments in Section 6, and analyze the algorithm performance in Section 7.

2 Related Work

In this section, we briefly survey related work on multi-agent simulation and Voronoi diagrams for path planning.

2.1 Multiple Agent Simulation

Agent-based methods, such as the seminal work of Reynolds [28], generate fast, simple local rules that can create visually plausible flocking behavior. Numerous extensions that account for social forces [7], psychological models [26], directional preferences [34], sociological factors [23], etc. have been proposed. Interesting techniques for collision avoidance have also been developed based on grid-based rules [22] and behavior models [37].

Most agent-based techniques perform local collision avoidance. However, global path planning techniques are needed to provide goal seeking capability. In practice, global planning algorithms typically use graph search techniques for each agent [2, 11, 19, 35]. Pettre et al. [27] proposed a graph structure that decomposes the space into multi-layered terrains to support fast graph search for multiple characters.

Most recently, a novel approach for crowd simulation based on continuum dynamics has been proposed by Treuille et al. [36]. This work computes a dynamic potential field that simultaneously integrates global navigation with local obstacle avoidance. The resulting system runs at interactive rates and demonstrate smooth traffic flows for three to four groups of large crowds that are moving with common goals. However this work does not address the case where each person's or agent's path needs to be computed separately. Multi-agent path planning has also been investigated extensively in robotics, mostly for performing collaborative tasks [3, 21, 25]. In addition, crowd simulation has also been heavily studied in other fields [14, 18, 29, 30]. We provide detailed comparisons with some closely related methods in Section 7.

2.2 Voronoi Diagrams and Path Planning

Voronoi diagram is a fundamental proximity data structure used in computational geometry and related areas [24]. Generalized Voronoi diagrams (GVD) of polygonal models have been widely used for motion planning [5, 20]. The boundaries of the generalized Voronoi diagram represent the connectivity of the space. Moreover, they are used to compute paths of maximal clearance between a robot and the obstacles based on potential field approaches [4, 16] or bias the sample generation for a randomized planner [10, 13, 39]. However, sampling-based methods are limited to static environments and the potential-field based planners have been used for 2D environments with very few robots or agents.

A disadvantage of using the GVD is the practical complexity of computing it efficiently and robustly. Hence, several approaches have been proposed to compute an approximation of the GVD. Vleugels and Overmars[38] use adaptive spatial subdivision. Choset and Burdick [5] define a related structure called *hierarchical generalized Voronoi graph* which is computed using continuation methods. Wilmarth et al. [39] compute points on the GVD without explicitly computing a representation of the entire set. Another set of approaches compute a discrete Voronoi diagram along a uniform grid using graphics hardware [15, 33, 8].

3 Background and Notation

In this section we introduce the notation used in the paper, give a background on Voronoi diagram based motion planning, and present an overview of our approach.

3.1 Notation

A geometric primitive or an object (in 3-dimensions) is called a *site*. In this work, a site refers to a point, an open edge, an open triangle, or a connected polygonal object, and we restrict ourselves to 2D environments. An entity for which a path needs to be computed is called an *agent* (or a robot). All obstacles and agents are represented as sites. The center of mass of a site p_i is denoted as $\pi(p_i)$.

Given a site p_i, the scalar distance function $d(\mathbf{q}, p_i)$ denotes the distance from the point $\mathbf{q} \in \mathbb{R}^n$ to the closest point on p_i. Given a set of sites P in domain D, and a subset T of P, with $|\mathsf{T}| = k$, the *k-th order Voronoi region* is the set of points closer to a site in T than to any other site:

$$\mathrm{Vor}^k(\mathsf{T}|\mathsf{P}) = \{\mathbf{q} \in \mathsf{D} \mid d(\mathbf{q}, p_i) \leq d(\mathbf{q}, p_j) \; \forall \; p_i \in \mathsf{T}, p_j \in \mathsf{P} \setminus \mathsf{T}\}.$$

The *k-th order Voronoi diagram* is a partition of of the domain D into the k-th order Voronoi regions:

$$\mathrm{VD}^k(\mathsf{P}) = \bigcup_{p_i \in \mathsf{P}} \mathrm{Vor}^k(\mathsf{T}, \mathsf{P}) \; , \; |\mathsf{T}| = k.$$

The standard Voronoi diagram is the same as the 1st order Voronoi diagram $\mathrm{VD}^1(\mathsf{P})$. We are specifically interested in the 1st and 2ndorder Voronoi diagrams, denoted as $\mathrm{VD}^1(\mathsf{P})$ and $\mathrm{VD}^2(\mathsf{P})$, respectively. A 1storder Voronoi region $\mathrm{Vor}^1(p_i|\mathsf{P})$ contains points closest to site p_i, and the 2ndorder Voronoi region $\mathrm{Vor}^2(\{p_i, p_j\}|\mathsf{P})$ contains points closest to two sites p_i and p_j. For ease of notation, we drop the superscript for the 1st order Voronoi diagram $\mathrm{VD}(\mathsf{P})$. The complement of a sub-domain X is denoted as X^c and given by $\mathsf{D} \setminus \mathsf{X}$.

The set of closest k-tuples of sites to a point is called the k-th order *governor set*. For a point $\mathbf{q} \in \mathsf{D}$, let the set of closest k-tuple of sites be $\mathsf{U} = \{\mathsf{T}_0, \ldots, \mathsf{T}_m\}, |\mathsf{T}_i| = k$, i.e. $\mathbf{q} \in \mathrm{Vor}^k(\mathsf{T}_i|\mathsf{P})$. Then the k-th order governor set of $\mathbf{q}$ is denoted as $\mathrm{Gov}^k(\mathbf{q}|\mathsf{P}) = \mathsf{U}$. The 1st order governor set is the set of closest sites, while the 2nd order set of a point is the set of closest pairs of sites.

In 2D, the boundaries of Voronoi regions consist of Voronoi edges which are subsets of the bisector between two sites, and Voronoi vertices which are equidistant from three or more sites. The arrangement of all Voronoi edges and vertices in the k-th order Voronoi diagram is called the k-th order *Voronoi graph*, denoted $\mathrm{VG}^k(\mathsf{P})$. Formally, $\mathrm{VG}^k(\mathsf{P}) = (\mathsf{V}, \mathsf{E})$, where,

$$\begin{aligned}\mathsf{V} =& \{\mathbf{v} \in \mathsf{D} \mid |\mathrm{Gov}^k(\mathbf{v}|\mathsf{P})| \geq 3\} \\ \mathsf{E} =& \{e \mid e = (\mathbf{v_1}, \mathbf{v_2}), \mathbf{v_1} \in \mathsf{V}, \mathbf{v_2} \in \mathsf{V}, \exists \text{ connected curve } c, \; s.t. \\ & \quad c = \mathrm{Vor}^k(p_i|\mathsf{P}) \cap \mathrm{Vor}^k(p_j|\mathsf{P}), \mathbf{v_1} \in c, \mathbf{v_2} \in c\}\end{aligned}$$

The k-th order Voronoi diagram is closely related to the *k-th nearest neighbor diagram*. The k-th nearest neighbor diagram is the partition of D into k-th nearest neighbor regions. The k-th nearest neighbor region of site p_i is the set of points for which p_i is the k-th nearest site. Similarly, the arrangement of the vertices and edges in the k-th nearest neighbor diagram is called the k-th nearest neighbor graph, denoted $\mathrm{NG}^k(\mathsf{P})$. Examples of the 1st and 2nd order Voronoi diagrams, Voronoi graphs, and nearest neighbor diagrams are shown in Figure C.2. The 1st nearest neighbor diagram is identical to the 1st order Voronoi diagram. Further properties of higher order Voronoi diagrams are presented in [9, 24].

3.2 Motion Planning Using Voronoi Diagrams

Voronoi diagrams have been used in motion planning in many ways, including roadmap computation, sample generation, or combined with potential field methods. The set of sites P is the set of obstacles, and the Voronoi diagram of the workspace $\mathrm{VD}(\mathsf{P})$ is computed. The Voronoi graph $\mathrm{VG}(\mathsf{P})$ captures the connectivity of the workspace and provides paths of maximal clearance between the obstacles. The Voronoi vertices closest to the robot and goal are classified as source and destination and the minimum weight path is then computed.

For complex 3D environments, an approximate Voronoi diagram is computed. The computation of discrete Voronoi diagrams and discrete Voronoi graphs can be accelerated using GPUs and has been used for motion planning in dynamic 2D [17] and 3D environments [33]. The Voronoi vertex closest to the agent is set as an intermediate goal and the Voronoi diagram is recomputed as the obstacles move.

However, these approaches are inefficient for computing the path of multiple agents in a dynamic environment. For an agent p_i, the remaining agents need to be considered as obstacles, i.e. the set of obstacles is $\mathsf{P} \setminus \{p_i\}$. Hence to compute the path for agent p_i, the modified Voronoi graph $\mathrm{VG}(\mathsf{P} \setminus \{p_i\})$ needs to be computed. Thus the cost of computing the path for all agents is $O(nc)$, where n is the number of agents and $O(c)$ is the cost of computing each modified Voronoi graph $\mathrm{VG}(\mathsf{P} \setminus p_i)$ for $1 \leq i \leq n$.

3.3 Overview

Our approach for motion planning of multiple agents uses the 1st and the 2nd order Voronoi diagrams to compute a global navigation data structure, the MaNG. The MaNG graph is the union of the 1st and the 2nd order Voronoi graphs and is formally presented in Section 4. We treat each agent as a site (in addition to other obstacles in the environment) and the MaNG is computed. The MaNG can be computed in time $O(c)$, and provides a path of maximal clearance for each agent. In addition, we compute the proximity information from the second order Voronoi diagram [31] and apply it within a potential-field based simulator [16].

4 Multiple Agent Planning using Hybrid Voronoi Graph

In this section we introduce the multi-agent navigation graph and demonstrate its application to multiple agents planning.

4.1 Multi-agent Navigation Graph (MaNG)

For multi agent planning, each moving agent represents a dynamic obstacle for the remaining agents. Hence, our goal is to compute a global navigation data structure that provides the clearance information for each agent. In particular, for each agent we want to compute a graph that provides maximal clearance to the obstacles and remaining agents.

We partition the set of sites P into two subsets - the set of obstacles P_o and the set of agents P_a. The multi-agent navigation graph (MaNG), denoted $\mathrm{MG}(\mathsf{P})$, is a union of the first order Voronoi graph $\mathrm{VG}^1(\mathsf{P})$ and a subset of the second order Voronoi graph $\mathrm{VG}^2(\mathsf{P})$ contained inside the 1st order Voronoi region of each agent. Formally,

$$\begin{aligned}
&\mathrm{MG}(\mathsf{P}) = (\mathsf{V}, \mathsf{E}), \text{ where} \\
&\quad \mathsf{V} = \{v \mid v \in \mathsf{V}^1 \cup (\mathsf{V}^2 \cap \mathrm{Vor}(p_i|\mathsf{P})) \forall p_i \in \mathsf{P}_a\}, \\
&\quad \mathsf{E} = \{e \mid e \in \mathsf{E}^1 \cup (\mathsf{E}^2 \cap \mathrm{Vor}(p_i|\mathsf{P})) \forall p_i \in \mathsf{P}_a\}, \\
&\mathrm{VG}^1(\mathsf{P}) = (\mathsf{V}^1, \mathsf{E}^1) \text{ and } \mathrm{VG}^2(\mathsf{P}) = (\mathsf{V}^2, \mathsf{E}^2).
\end{aligned}$$

The $\mathrm{MG}(\mathsf{P})$ consists of vertices and edges from the 1st and the 2nd order Voronoi graphs $\mathrm{VG}^1(\mathsf{P})$ and $\mathrm{VG}^2(\mathsf{P})$. Some vertices in $\mathrm{MG}(\mathsf{P})$ are common to both $\mathrm{VG}^1(\mathsf{P})$ and $\mathrm{VG}^2(\mathsf{P})$, however $\mathrm{VG}^1(\mathsf{P})$ and $\mathrm{VG}^2(\mathsf{P})$ do not share any edge [24].

We assign a coloring to each edge and vertice in $\mathrm{MG}(\mathsf{P})$ based on its membership in $\mathrm{VG}^1(\mathsf{P})$ or $\mathrm{VG}^2(\mathsf{P})$. Edges from $\mathrm{VG}^1(\mathsf{P})$ are colored **red** and edges from $\mathrm{VG}^2(\mathsf{P})$ are colored **black**. Further, vertices in $\mathrm{VG}^1(\mathsf{P})$ are colored red, and vertices in $\mathrm{VG}^2(\mathsf{P}) \setminus \mathrm{VG}^1(\mathsf{P})$ are colored black. Finally, each edge in the MaNG is assigned weight based on the cost of traveling that segment. Details on weight computation are presented in Section 5. A 2D example of the MaNG for some point agents and obstacles is shown in Figure C.2.

$\mathrm{MG}(\mathsf{P})$ is closely related to the 2nd nearest neighbor graph $\mathrm{NG}^2(\mathsf{P})$. In particular, we state the following result about their relation. Detailed proofs are provided in a technical report.

Lemma 1. *Given a set of sites P, the 2nd nearest neighbor graph $\mathrm{NG}^2(\mathsf{P})$ and the MaNG $\mathrm{MG}(\mathsf{P})$:*

- $\mathrm{MG}(\mathsf{P}) \subseteq \mathrm{NG}^2(\mathsf{P})$,
- *Given an edge $e \in \mathrm{NG}^2(\mathsf{P})$ incident on two 2nd nearest neighbor regions of sites p_i and p_j. For any point $\mathbf{x} \in e$: If $d(\mathbf{x}, p_i) = d(\mathbf{x}, p_j) = d(\mathbf{x}, \mathsf{P}) \Rightarrow e \in \mathrm{VG}^1(\mathsf{P})$. If $d(\mathbf{x}, p_j) = d(\mathbf{x}, p_i) > d(\mathbf{x}, \mathsf{P}) \Rightarrow e \in \mathrm{VG}^2(\mathsf{P})$.*

As a consequence of lemma 1, both the 1st and 2nd order Voronoi graphs can be extracted from the 2nd nearest neighbor diagram. We use this result to efficiently compute the MaNG from the 2nd nearest neighbor diagram in Section 5.

4.2 Multiple Agent Planning

In this section, we present our approach for efficient path planning of multiple agents using the MaNG. The path planning problem for each agent is defined as follows: we are given an agent $p_i \in \mathsf{P}_a$, its current position in the workspace given by its center of mass $\pi(p_i)$, and a goal position of the center of mass $\mathbf{g_i}$. We wish to compute a path for p_i from $\pi(p_i)$ to $\mathbf{g_i}$, which is maximally clear and collision free to the remaining sites $\mathsf{P} \setminus \{p_i\}$. Such a path can be computed using the Voronoi graph $\mathrm{VG}^1(\mathsf{P} \setminus \{p_i\})$. We state a result on the equivalence of the paths computed using the 1st order Voronoi graphs and the MaNG.

Lemma 2. *Given an agent p_i, and the Voronoi graphs $\mathrm{VG}^1(\mathsf{P} \setminus \{p_i\})$, $\mathrm{MG}(\mathsf{P})$:*

1. $\mathrm{VG}^1(\mathsf{P} \setminus \{p_i\}) \subseteq \mathrm{MG}(\mathsf{P})$
2. $\mathrm{VG}^1(\mathsf{P} \setminus \{p_i\}) \cap \mathrm{Vor}(p_i|\mathsf{P}) = \mathrm{MG}(\mathsf{P}) \cap \mathrm{Vor}(p_i|\mathsf{P})$.

Lemma 2 provides an approach for extracting the Voronoi graph $\mathrm{VG}^1(\mathsf{P} \setminus \{p_i\})$, for each agent p_i, from $\mathrm{MG}(\mathsf{P})$. The complete algorithm for computing a path for an agent p_i is given in Algorithm 1, and an example path is shown in Figure C.4. The function *LocatePoint*($\mathbf{g_i}$) returns the 1st order Voronoi region which contains $\mathbf{g_i}$. The source and goal positions are connected to vertices in the MaNG using green edges. *ShortestPath*(p_i,$\mathbf{g_i}$,$\mathrm{MG}(\mathsf{P})$) computes

the minimum weight path from $\pi(p_i)$ to $\mathbf{g_i}$ following only the green and red edges in MG(P). This is equivalent to computing the shortest path by following the 2nd order Voronoi graph inside the 1st order Voronoi region of agent p_i, and the 1st order Voronoi graph everywhere else (see Figure C.4). The first vertex along this path is chosen as an intermediate goal for agent p_i.

Input: Agent p_i, Goal position $\mathbf{g_i}$, Set of sites P, MaNG MG(P)
Output: Path S_i from current position to goal position

$k \leftarrow$ LocatePoint($\mathbf{g_i}$)
if $k = i$ **then**
  $S_i \leftarrow edge(\pi(p_i), \mathbf{g_i})$
  return
Compute $V_i \leftarrow$ set of black vertices in Vor(p_i|P)
Compute $E_i \leftarrow$ set of black edges in Vor(p_i|P)
if $V_i \neq \emptyset$ *and* $\pi(p_i) \notin V_i$ **then**
  Augment MG(P) with green edges
  $e_j = (\pi(p_i), v_j) \forall v_j \in V_i$
  Assign weight to $e_j, w(e_j) \leftarrow d(\pi(p_i), v_j)$
else
  foreach *edge* $e_j \in E_i$ **do**
    Compute $v_j \leftarrow$ closest point on e_j to $\pi(p_i)$
    Augment MG(P) with green edge $e_j = (\pi(p_i), v_j)$
    Assign weight to $e_j, w(e_j) \leftarrow d(\pi(p_i), v_j)$
  end
Compute $V_k \leftarrow$ set of red vertices in Vor(p_k|P)
Augment MG(P) with green edges $e_j = (\mathbf{g_i}, v_j) \forall v_j \in V_k$
Assign weight to $e_j, w(e_j) \leftarrow d(\mathbf{g_i}, v_j)$
Add green labels to each edge $e_j \in E_i$
$S_i \leftarrow$ ShortestPath (p_i,$\mathbf{g_i}$,MG(P))
Remove green labels from each edge $e_j \in E_i$
Remove all green edges from MG(P)

Algorithm 1: ComputePath(p_i, $\mathbf{g_i}$, P, MG(P)): Computes a path for agent p_i to goal $\mathbf{g_i}$ given the set of sites P and the MaNG MG(P)

5 MANG COMPUTATION

In this section we present our approach for efficiently computing the MaNG, which is based on the 1st and the 2nd order Voronoi diagrams. However, exact computation of generalized Voronoi diagrams of polygonal models is non-trivial. Rather, we compute the discrete Voronoi diagram along a uniform grid using graphics hardware [15]. The 2nd nearest neighbor diagram is computed using a second pass with depth peeling, as presented in [9]. We compute the generalized 2nd nearest diagram of higher order sites (lines, polygons) by rendering the generalized distance function for each site [32]. We compute the 1st order Voronoi diagram in the first pass, and compute the 2nd nearest neighbor diagram in the second pass. Finally we extract the 1st and the 2nd order Voronoi graphs from the 2nd nearest neighbor diagram and compute the MaNG.

5.1 Culling Techniques

The distance field is computed by evaluating the distance function to each site at each pixel, and this computation is efficiently performed using the rasterization capabilities of the GPU. However, for a large number of sites, this leads to redundant computation for each pixel, and the computation becomes fill bound. Hence, we use culling techniques to compute conservative bounds on the 1st and the 2nd order Voronoi regions. The distance function to each site is computed on the pixels that are contained within its conservative Voronoi region.

Our goal is to efficiently derive a tight upper bound on the 1st and the 2nd order Voronoi regions for each site. We compute these bounds by determining the closest site (and closest 2 sites) along each principle direction ($+X, -X, +Y, -Y$). We compute the bounds using a quadtree, which subdivides the domain. Each node in the quadtree contains the number of sites contained in the subtree rooted at the node. Using this quadtree we can efficiently determine the set of nearest neighbors for each site.

The quadtree is constructed as follows. Each leaf nodes contains the number of sites contained within the node. Let δ be the size of a leaf node. Each intermediate node contains the number of sites contained in each of its 4 children. Let the function $E(l)$ compute the closest non-empty leaf node to the right of node l in the quadtree. Similarly, $W(l)$ and $N(l)$, $S(l)$, respectively, return closest leaf nodes to the left, top and bottom of node l. To compute the bound along $+X$ for the 1st order Voronoi region of a site p_i, we first identify the leaf node l_i that contains the centroid of the site $\pi(p_i)$. Next we compute the closest leaf nodes $E(l_i)$, $W(l_i)$, $N(l_i)$ and $S(l_i)$. Finally we compute the bisectors of the pairs of nodes $E(l_i), S(l_i)$ and $W(l_i), S(l_i)$. Then the bound on the first order Voronoi region along X axis is given by the intersection of these two bisectors. Similarly, the bounds along Y-axis are computed, and the first order Voronoi region of site p_i is bounded by a quad covering these bounds. In addition, for a leaf node l_i, we store the locations of its closest neighbors $E(l_i)$, $W(l_i)$, $N(l_i)$ and $S(l_i)$.

We compute the bounds on all the 2nd order Voronoi regions of site p_i in the second pass as follows. Along $+X$ axis, we check the number of sites stored in the closest node $E(l_i)$. If the number of sites in node $E(l_i)$ is 2 or more, then the bound along $+X$ is $\Delta X^+ = d(l_i, E(l_i)) + 2\delta$. If number of sites in node $E(l_i)$ is less than 2, then we lookup the node $E(E(l_i))$ (this has been computed in the 1st pass), and the bound along $+X$ is $\Delta X^+ = d(l^i, E(E(l_i))) + 2\delta$. Similarly we compute bounds along $-X, +Y$ and $-Y$ axes and compute the distance function of site p_i in a quad that covers these bounds.

To compute the bounds for a higher order site (a line segment or a convex polygon), we store the position of the centroid of the site in the quadtree. We compute the distance bounds for the centroid using the quadtree, and add the distance between the centroid and a vertex to compute the distance bounds for the site.

5.2 Undersampling Errors

Computation of the Voronoi graph on a uniform grid may result in undersampling errors, which may lead to the Voronoi regions to become disconnected [33],and the computed discrete Voronoi graph may have many small disjoint components [16]. As a result, for complex environments with a large number of sites, the combinatorial complexity of the MaNG becomes very high.

We address the issue of undersampling for motion planning, by reducing the combinatorial complexity of the MaNG without changing its connectivity. We reduce the complexity by appropriately modifying the MaNG near undersampled areas. We rely on the fact that when two Voronoi edges are arbitrarily close, then the agent might follow either edge, as long as the path connectivity does not change. Such edges can be removed from the MaNG provided their removal does not change the connectivity of the MaNG.

We present the details of our algorithm for reducing the complexity of MaNG. We treat an edge with an adjacent edge less than one pixel away as a candidate for removal. Such edges are exactly those edges that bound a discrete Voronoi region of width 1 pixel. Thus the test for eliminating such edges is equivalent to removing certain pixels from a discrete Voronoi region, which does not change the

connectivity of the Voronoi graph. Hence our test for removal of a pixel from a discrete Voronoi region relies on a local 3×3 stencil around a pixel. Let p_a be the governor of a pixel (i, j), and the set α denote the governor set of the 4 adjacent pixels $(i-1, j), (i+1, j), (i, j-1), (i, j+1)$. Then the pixel (i, j) can be removed if either of the following conditions holds (see Figure C.5):

1. $p_a \notin \alpha$. Then site p_a has an isolated discrete Voronoi region at pixel (i, j), with 4 Voronoi edges surrounding it. Removal of this Voronoi region does not change the path connectivity in the stencil.

2. $p_a \in \alpha$ and p_a occurs in α exactly once. Then the pixel (i, j) represents an end point of a discrete Voronoi region of site p_a and its removal does not change the path connectivity in the stencil.

After a pixel (i, j) satisfies the criteria for removal, we assign it to another discrete Voronoi region to maintain the connectivity of Voronoi edges. The pixel is assigned to a site in $\alpha \setminus \{p_a\}$ with the minimum distance to pixel (i, j). The distance of a site in α to pixel (i, j) can be efficiently computed by relying on the fact that distance vectors are bi-linearly interpolated [32]. Thus distance computation involves a vector summation with a basis vector and vector norm computation.

The operation performed at each pixel is a read followed by a conditional write, and the output of one pixel may affect the connectivity of adjacent pixels. Thus an efficient parallel algorithm is not feasible, and we perform a sequential scan of the discrete Voronoi diagram to update the Voronoi graph.

5.3 Graph Construction

We now present our algorithm to compute the MaNG. We compute the 1st order Voronoi diagram $\mathrm{VD}^1(\mathsf{P})$ and the 2nd nearest neighbor diagram on the GPU, and refine the connectivity information based on the algorithm described in Section 5.2. We then perform sequential tracing of vertices and edges to compute the 2nd nearest neighbor graph [15].

We use the result presented in Lemma 1 to classify the edges in the 2nd nearest neighbor graph, $\mathrm{NG}^2(\mathsf{P})$. An edge is classified as belonging to the 1st order Voronoi graph if the distance to closest site for all pixels on the edge is identical in $\mathrm{VG}^1(\mathsf{P})$ and $\mathrm{NG}^2(\mathsf{P})$. Due to pixel resolution errors, we treat two distance values as identical if they are within one pixel width of each other. Each edge is assigned a weight proportional to its length and inversely proportional to the minimal clearance along the edge. An edge belonging to $\mathrm{VG}^1(\mathsf{P})$ is labeled red, and the remaining edges are labeled black. A vertex is labeled red if it has at-least one red edge incident on it, otherwise it is labeled black. These colors are used by Algorithm 1 to search for an optimal path.

6 Implementation and Results

In this section we describe the implementation of our multi agent planning algorithm and highlight its application to various multi-agent simulations.

6.1 Implementation

We have implemented our algorithm on a PC running Windows XP operating system with an AMD Opteron 280 CPU, 2GB memory and an NVIDIA 7900 GPU. We used OpenGL as the graphics API and Cg language for implementing the fragment programs. The discrete Voronoi diagram and distance field are computed at 32-bit floating point precision using floating point buffers. The Voronoi diagram is stored in the red channel, and the distance field in the depth buffer. We use stencil tests to disable 2nd order Voronoi diagram computation in the 1st order Voronoi regions of the obstacles. In the first pass, the stencil mask is set for all pixels in the 1st order Voronoi regions of the agents. In the second pass, distance functions are evaluated at pixels with stencil mask set. This optimization speeds up both discrete Voronoi diagram computation and MaNG construction. We perform readback of the discrete Voronoi diagrams and construct the MaNG on the CPU. The optimal path is computed using an A^* search with Euclidean distance metric to guide the search.

We use a complete quadtree for Voronoi region culling described in Section 5.1. The depth of the quadtree is set such that one leaf node corresponds to a block of 32×32 pixels. We need to determine if a node contains up to 2 sites - hence the number of sites per node is encoded in 1 byte. By using a complete quadtree, the node addresses can be efficiently computed using bit shifts, avoiding pointer addressing.

6.2 Demos

We describe two multi-agent simulations, demonstrating the effectiveness of the MaNG for real-time path planning. The first simulation involves a coverage problem, while the second one is of a crowd simulation.

Fruit Stealing Game: The first simulation is of fruit stealing in a dense orchard (see Figure C.1). There are several agents (thieves) which attempt to steal the fruit on the trees. The environment also contains some old farmers who chase the thieves. As the thieves move through the orchard, they steal fruit in close proximity. The goal is for each thief to move towards denser regions of fruit while avoiding the farmers, the trees and other thieves. The thieves, farmers and trees are treated as cylindrical sites. The trees are fixed obstacles, farmers are dynamic obstacles and the thieves are the agents. A coarse density map is used to track the density of fruit remaining in the orchard. Trees with desirable fruit are assigned higher density. The agents are initially spread near the boundary of the orchard, and the goal position is set to a distant high density region. The goal position for each agent is also dynamically updated as the density of the current goal drops below a certain threshold.

The global path of each agent is computed using the approach presented in Algorithm 1. We compute the proximity to nearest site for each agent from the 2nd nearest neighbor diagram, which is used in a potential planner for local planning. Finally, we also use the 2nd order Voronoi diagram to compute the closest agent (thief) for each farmer. This is set as the goal for each farmer and the farmer moves directly towards this agent. The farmers do not use the MaNG for path planning, however they use the potential and repulsive forces to stay clear of other farmers and trees. A thief is eliminated if caught by a farmer. Hence it is desirable for each thief to compute shortest paths of maximal clearance from the farmers (dynamic obstacles) and other thieves (agents) in order to collect the most fruit.

Crowd Simulation: We simulate a crowd of people moving in an urban environment with dynamic obstacles (Figure C.3). We simulate only the individual behavior and not the group behavior. The set of sites consists of buildings, cars and humans. The humans enter the scene from one of the buildings and exit through another building or the sidewalks. Each human is an individual agent with an independent goal. The cars are dynamic obstacles, while the buildings, benches, fountains are static obstacles. Similar to fruit picking, the proximity information for local planning is computed using the 2nd order Voronoi diagram. The total force applied on each agent is a sum of an attractive force to move it towards the intermediate goal computed by the MaNG, and the repulsive forces

from the nearest neighbors. For goals in close proximity, only the local potential field planner is used, disregarding the MaNG.

6.3 Results

We now highlight the performance of our algorithm in complex dynamic environments. Our approach can perform real-time path planning for each agent in environments up to 200 agents with different destinations, at the rates of 5 to 20 fps. The discrete Voronoi diagrams are computed on grid of resolution $1K \times 1K$ pixels. The fruit stealing simulation has 64 trees with a varying number of thieves and farmers. The crowd simulation has 15 static obstacles and between 2 and 5 moving cars, with a varying number of humans. The performance of our approach, with a timing breakup is presented in Table 1.

Demo	Agents	Graph		Time(ms)			
		$\|\mathsf{V}\|$	$\|\mathsf{E}\|$	DVD	MaNG	Plan	Total
Crowd	10	206	1051	7	20	0.23	52
Crowd	25	330	1949	9	22	0.8	57
Crowd	50	560	3500	10	36	2.0	73
Crowd	100	946	7058	15	65	5.6	110
Crowd	200	1927	14669	20	150	18	213
Fruit	10	565	2282	8	25	1.0	59
Fruit	100	1378	6099	15	70	20	130

Table 1: Performance of multi-agent path planning algorithm (average over all frames): $|\mathsf{V}|$ and $|\mathsf{E}|$ denote number of vertices and edges in the MaNG. DVD = Time to compute the 2^{nd} order discrete Voronoi diagram on the GPU, and removing undersampled regions. MaNG = Time to extract the MaNG from the discrete Voronoi diagram. Plan = time for path planning for all agents. Time for readback of discrete Voronoi diagram and depth buffers at $1K \times 1K$ resolution = 25ms.

7 Analysis and Comparison

In this section, we analyze the performance of our algorithm. We highlight its computational complexity and compare it with other approaches for multi-agent path planning.

7.1 Analysis

Let the number of sites be n, and the size of the grid used to compute the discrete Voronoi diagrams be $m \times m$. We assume the number of agents $|\mathsf{P}_a| = O(n)$. We now present the time complexity of each stage in our algorithm.

The cost of computing the 1^{st} and 2^{nd} order discrete Voronoi diagrams is as follows. The size of the quadtree is $O((\frac{m}{32})^2)$, and depth $= O(\log m)$. Then the cost of computing the bounds for each site (see Section 5.1) is $O(\log m)$. The cost of rasterizing the distance function for a site p_i is $O(r|\mathrm{Vor}^k(p_i|\mathsf{P})|)$, where $|\mathrm{Vor}^k(p_i|\mathsf{P})|$ is the number of pixels in the Voronoi region of p_i and r depends on the tightness of the computed Voronoi region bounds, $1 < r < O(n)$. Typically, we have observed $r = O(1)$. Then the cost of computing the Voronoi diagram is $O(n \log m + \Sigma_{i=1}^{n}(r|\mathrm{Vor}^k(p_i|\mathsf{P})|)) = O(rm^2 + n \log m)$.

The cost of reading back the framebuffers is $O(m^2)$. The cost of extracting the MaNG is $O(|\mathsf{E}|)$, where $|\mathsf{E}|$ is number of edges in MaNG. From lemma 2, the number of edges in MaNG, $|\mathsf{E}| \leq |\mathsf{E}^1| + |\mathsf{E}^2|$, where $|\mathsf{E}^k|$ is number of edges is $\mathrm{VD}^k(\mathsf{P})$, and $|\mathsf{E}^k| = O(kn)$ [9]. Thus cost of extracting the MaNG is $O(n)$. The cost of path planning using A^* is typically polynomial in $O(|\mathsf{E}| + |\mathsf{V}|)$. Therefore cost of computing all paths is $O(n(|\mathsf{E}| + |\mathsf{V}|)) = O(n^2)$. In practice, as shown by Table 1 the associated constant with path planning is much smaller and the bottleneck is the discrete Voronoi diagram computation and graph construction.

7.2 Comparisons

Next we provide qualitative comparisons of our approach with prior methods for multi-agent planning.

Comparison with 1^{st} order Voronoi diagram: Our approach provides a global solution for path planning of each agent using the MaNG. The MaNG computes a roadmap of maximal clearance collision free paths for each agent in $O(1)$ passes, as compared to $O(n)$ passes for computing $O(n)$ Voronoi roadmaps. In particular, using the 2^{nd} order Voronoi graph for path planning guarantees that the position selected as the first intermediate goal along the computed path is unique. This prevents adjacent agents from moving towards the same intermediate goal and getting stuck in a local minimum of the potential function. An example is presented in Figure C.6. In this example, adjacent agents select the same intermediate goal from 1^{st} order Voronoi diagram, whereas the intermediate goals from the 2^{nd} order Voronoi diagram are unique. In addition, the path computed has maximal clearance. More specifically, vertices on the Voronoi diagram are used to compute the area of maximum coverage for a new site [1]. Hence by following the vertices on the MaNG, our planning approach ensures a maximum coverage region for each agent.

The closest related work by Pettre et al. [27] computes an initial roadmap of a static environment using Voronoi diagrams, and constructs a set of homotopic paths for a group of agents. This work implicitly groups agents by their origins and goals. Furthermore, local collision avoidance is not guaranteed. In contrast our algorithm is able to handle dynamic environments as the roadmap is updated in real-time, and the use of 2^{nd} order Voronoi diagrams provides pairwise proximity information which is used to guarantee collision avoidance.

The work on continuum crowds [36] computes a dynamic potential field and updates the position of each agent by moving along the gradient of the potential function. The potential field is computed for a small number of groups of agents moving with common goals. However, due to the use of a potential function the agents may get stuck in a local minimum. In contrast, our approach allows for an independent goal for each agent.

In comparison to agent based methods, our MaNG based path planning algorithm provides global paths, and may be combined with rule-based techniques to simulate more complex and realistic agent behavior.

7.3 Limitations

There are some limitations of our work. We compute the MaNG in the workspace, hence the approach does not scale well for agents with many degrees of freedom (e.g. snakes). We use an A^* graph search algorithm, which may not be optimal. Finally, we compute an optimal path for each frame, however there is no guarantee on coherence of paths across frames, or on convergence over a period of time. In fact, the optimal paths across two time steps may not be coherent, potentially resulting in noisy motions.

8 Conclusions and Future Work

We have presented a novel approach for real-time path planning of multiple virtual agent, based on a new data structure - the Multi-agent Navigation Graph (MaNG). The MaNG is used to simultaneously compute the paths of maximal clearance for a set of moving agents with independent goals. The MaNG is constructed dynamically using discrete Voronoi diagrams. We also presented culling techniques for accelerating the discrete Voronoi diagram computation and addressed undersampling issues due to discretization. We have demonstrated the application of our approach to real time simulation involving a large number of independent agents, each with an individual goal.

There are several avenues for future work. One relevant avenue is to constrain the graph search to compute temporally coherent paths which are guaranteed to converge to the final goal. We would like to exploit coherence in graph search when many agents have similar goals and initial positions. Efficient parallel algorithms for simplifying the discrete Voronoi graphs and computing the MaNG would be useful for accelerating the computation. Finally, we would like to extend our approach to handle agents with high degrees of freedom.

References

[1] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991.

[2] O. B. Bayazit, J.-M. Lien, and N. M. Amato. Better group behaviors in complex environments with global roadmaps. *Int. Conf. on the Sim. and Syn. of Living Sys. (Alife)*, 2002.

[3] M. Bennewitz and W. Burgard. Finding solvable priority schemes for decoupled path planning techniquesfor teams of mobile robots. *Proceedings of the 9th Int. Symposium on Intelligent Robotic Systems (SIRS)*, 2001.

[4] J. Champagne and W. Tang. Real-time simulation of crowds using voronoi diagrams. *EG UK Theory and Practice of Computer Graphics*, 2005.

[5] H. Choset and J. Burdick. Sensor based motion planning: The hierarchical generalized Voronoi graph. In *Algorithms for Robot Motion and Manipulation*, pages 47–61. A K Peters, 1996.

[6] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[7] O. C. Cordeiro, A. Braun, C. B. Silveria, S. R. Musse, and G. G. Cavalheiro. Concurrency on social forces simulation model. *First International Workshop on Crowd Simulation*, 2005.

[8] M. Denny. Solving geometric optimization problems using graphics hardware. In *Proc. of Eurographics*, 2003.

[9] I. Fischer and C. Gotsman. Fast approximation of high order Voronoi diagrams and distance transforms on the GPU. Technical report CS TR-07-05, Harvard University, 2005.

[10] M. Foskey, M. Garber, M. Lin, and D. Manocha. A voronoi-based hybrid planner. *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001.

[11] J. Funge, X. TU, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *Proc. of ACM SIGGRAPH*, 1999.

[12] P. Glardon, R. Boulic, and D. Thalmann. Dynamic obstacle clearing for real-time character animation. *Computer Graphics International*, 2005.

[13] L. Guibas, C. Holleman, and L. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proc. of IROS*, 1999.

[14] D. Helbing, L. Buzna, and T. Werner. Self-organized pedestrian crowd dynamics and design solutions. *Traffic Forum 12*, 2003.

[15] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. *Proceedings of ACM SIGGRAPH 1999*, pages 277–286, 1999.

[16] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Interactive motion planning using hardware accelerated computation of generalized voronoi diagrams. *IEEE Conference on Robotics and Automation*, pages pp. 2931–2937, 2000.

[17] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 145–148, 2001.

[18] A. Kamphuis and M. Overmars. Finding paths for coherent groups using clearance. *Proc. of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2004.

[19] 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(3 (Sept)), 2004.

[20] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[21] T.-T. Li and H.-C. Chou. Motion planning for a crowd of robots. *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2003.

[22] C. Loscos, D. Marchal, and A. Meyer. Intuitive crowd behaviour in dense urban environments using local laws. *Theory and Practice of Computer Graphics (TPCG'03)*, 2003.

[23] S. R. MUSSE and D. Thalmann. A model of human crowd behavior: Group inter-relationship and collision detection analysis. *Computer Animation and Simulation*, 1997.

[24] A. Okabe, B. Boots, and K. Sugihara. *Spatial tessellations: concepts and applications of Voronoi diagrams*. Wiley & Sons, 1992. ISBN 0 471 93430 5.

[25] L. E. PARKER. Designing control laws for cooperative agent teams. *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1993.

[26] N. Pelechano, K. O'Brien, B. Silverman, and N. Badler. Crowd simulation incorporating agent psychological models, roles and communication. *First International Workshop on Crowd Simulation*, 2005.

[27] J. Pettre, J.-P. Laumond, and D. Thalmann. A navigation graph for real-time crowd animation on multilayered and uneven terrain. *First International Workshop on Crowd Simulation*, 2005.

[28] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 25–34, July 1987.

[29] M. SOFTWARE. http://www.massivesoftware.com, 2006.

[30] G. Still. *Crowd Dynamics*. PhD thesis, University of Warwik, UK, 2000. Ph.D. Thesis.

[31] A. Sud, N. Govindaraju, R. Gayle, I. Kabul, and D. Manocha. Fast proximity computation among deformable models using discrete voronoi diagrams. *ACM Trans. Graph. (Proc ACM SIGGRAPH)*, 25(3):1144–1153, 2006.

[32] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha. Interactive 3d distance field computation using linear factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games*, pages 117–124, 2006.

[33] A. Sud, M. A. Otaduy, and D. Manocha. DiFi: Fast 3D distance field computation using graphics hardware. *Computer Graphics Forum (Proc. Eurographics)*, 23(3):557–566, 2004.

[34] M. Sung, M. Gleicher, and S. Chenney. Scalable behaviors for crowd simulation. *Computer Graphics Forum*, 23(3 (Sept)), 2004.

[35] M. Sung, L. KOVAR, and M. Gleicher. Fast and accurate goal-directed motion synthesis for crowds. *Proc. of SCA 2005*, pages 291–300, 2005.

[36] A. Treuille, S. Cooper, and Z. Popovic. Continuum crowds. *Proc. of ACM SIGGRAPH*, 2006.

[37] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In A. Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 43–50. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[38] J. Vleugels and M. H. Overmars. Approximating Voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry and Applications*, 8:201–222, 1998.

[39] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. *IEEE Conference on Robotics and Automation*, pages 1024–1031, 1999.

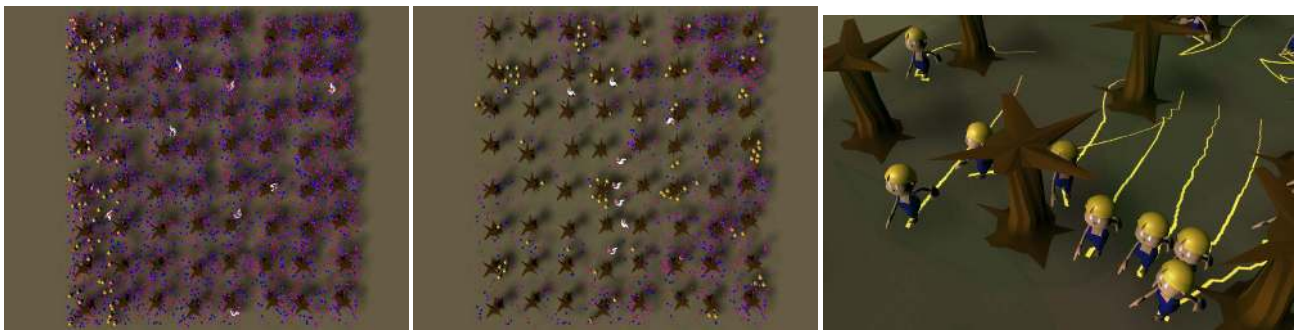

Figure C.1: Fruit stealing simulation: A simulation of 96 *fruit pickers (with yellow hair) in an orchard with* 64*K fruit (dark blue and purple) on* 64 *trees (brown trunks) and* 4 *farmers (in white shirts). Each agent maintains an independent goal.* Left*: Initial top view of the orchard.* Middle*: Top view during the middle of simulation with many fruit collected.* Right*: Perspective view of path traces of the agents in fruit stealing simulation. The yellow curves trace the position of each agent over a range of time steps. The trace demonstrates lane formation as the agents move around obstacles.*

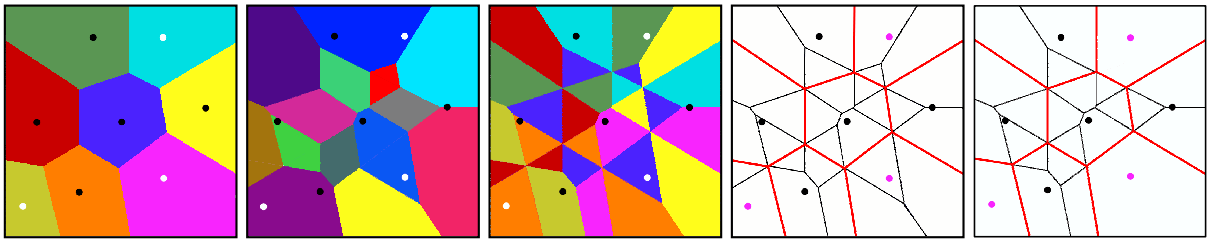

Figure C.2: Voronoi Diagrams and Voronoi Graphs: 8 *point sites consisting of* 3 *obstacles (shown in white) and* 5 *agents (shown in black). (a)* 1^{st} *order Voronoi diagram (b)* 2^{nd} *order Voronoi diagram of the* 8 *sites. Each region is closer to one of a pair of sites than to any other site (c)* 2^{nd} *nearest neighbor diagram. Each region has the same site as the second closest site. (d)* 2^{nd} *nearest neighbor graph. Red edges denote edges from* 1^{st} *order Voronoi graph, black edges are edges from* 2^{nd} *order Voronoi graph (e) the Multi-agent Navigation Graph (MaNG) for the* 5 *agents, which is a subset of the* 2^{nd} *nearest neighbor graph.*

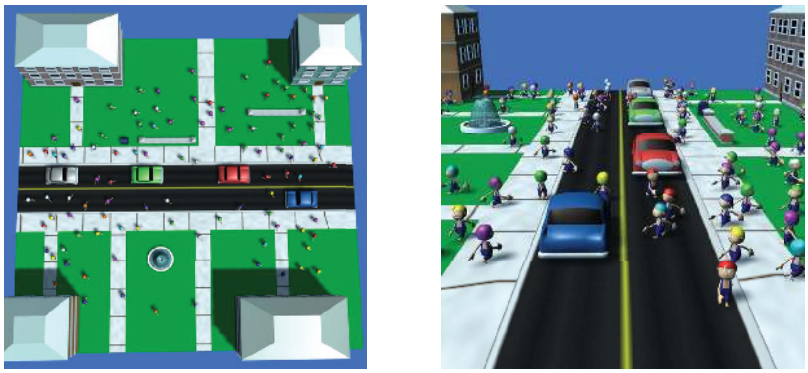

Figure C.3: Crowd Simulation: Two scenes of a crowd simulation with agents moving between buildings and the sidewalks. The cars represent dynamic obstacles. Our MaNG based algorithm can perform path planning on 200 *agents, each with distinct goals, at* 5 *frames per second.*

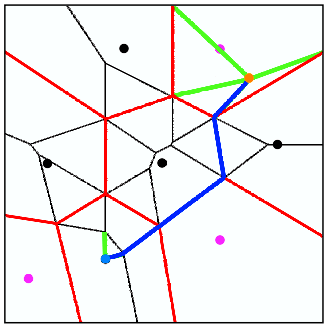

Figure C.4: Multi-Agent Path Planning using the MaNG. The MaNG is augmented with green edges connecting the start position (blue dot) to the goal position (orange dot). The computed shortest path for one agent is shown with blue edges.

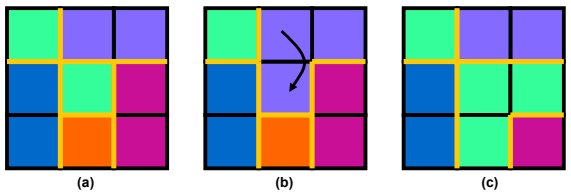

Figure C.5: Discrete Voronoi region shrinking for under-sampling errors: A 3×3 *pixel neighborhood of a discrete Voronoi diagram. The discrete MaNG is shown in thick orange lines. (a) The green discrete Voronoi region is disconnected. (b) The center pixel may be assigned to an adjacent Voronoi region reducing complexity of the MaNG, without changing its connectivity (c) Reassigning the center pixel will change connectivity of the MaNG.*

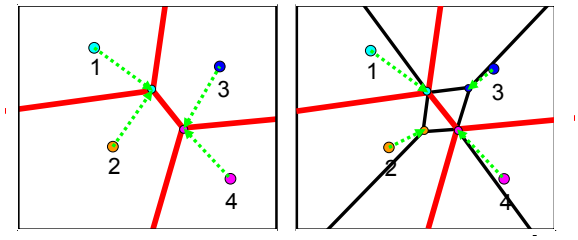

Figure C.6: Comparison of 1^{st} *order Voronoi graph and MaNG:* 4 *agents, with goals in opposite corners.* Left: *Intermediate goals computed from* 1^{st} *order Voronoi graph. Pairs of agents move towards same goal.* Right: *Intermediate goals from MaNG. Each agent has a unique intermediate goal.*