VISUALIZATION AND ANIMATION FOR SITUATION AWARENESS IN BATTLEFIELD

A Thesis
Submitted to the Faculty
of
Purdue University
by
Young Jun Kim

In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

August 2000
To my parents, Kim and Ryu. To my grandparents.
ACKNOWLEDGMENTS

Looking back my Ph.D. life, I have been extremely lucky to be able to meet the right people at the right place at the right time. I have so many people to thank for, so please excuse me in case that I omit some of the names here.

First of all, I would like to thank my thesis advisor, Professor Christoph Hoffmann. Not only does he have been an excellent academic advisor from the day one, but he has been a caring mentor to me, not to mention his great sense of humor and down to earth personality. My thank also goes to my master thesis advisor, Professor Yeongil Shin, in Seoul National University. He has been so generous and helpful to guide my academic career from my early years at SNU.

I thank my Ph.D. committee members, Professor Zygmund Pizlo, Professor Anath Grama and Professor Mike Atallah, for their insightful comments on my dissertational work. Also I thank Professor Kihong Park to be a good advisor and kind senior.

I thank Cassian Durand, Minho Kyung, Hoi Chang, Minseok Kwon, Sungan Kim, Kujin Kim, KyungKoo Jun, and Carlos Gonzalez in Computer Science Department at Purdue for the share of their brilliance and wit. I especially thank Hoi to be such a good friend all the time.

I thank all the members of Korean Pan-Math group at Purdue, especially to
Yangsuk Ko, Hyungjun Kim, Yangseok Cho and their family for their invaluable life advice and kindness shown to me. I really embrace the moment of daily lunch meeting with them to talk about anything under the sun.

I thank all my friends in Korea to make myself feel like at home even while I was away. Especially I thank Jinkyu Byun, Seungwon Cho, Sewook Joo, Hongkeum Kim, and Changhoon Lee for their endless support and friendship. They have shown to me what a real friend is with their generosity, no matter how I bother them.

Finally I thank my parents, brother and sister in Busan, Korea. Without them, none of this work would be possible. I really appreciate their endless support, endurance and belief on me no matter how I feel bad or struggle for my life. They have been a source of my inspiration, and have shown to me what a great family I have.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>x</td>
</tr>
<tr>
<td>1  INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Problem Formulation</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1 Situation Awareness</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2 Battlefield Visualization</td>
<td>3</td>
</tr>
<tr>
<td>1.1.3 Moving Points in 2D</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Previous Work Review</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 Situation Awareness Application</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2 Battlefield Visualization Package</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Dissertation Overview</td>
<td>8</td>
</tr>
<tr>
<td>2  COMPUTATIONAL FRAMEWORK</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Various Approaches to Proximity Computation</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Voronoi Diagram and Delaunay triangulation</td>
<td>12</td>
</tr>
<tr>
<td>2.3 Delaunay Triangulation Based Approach</td>
<td>14</td>
</tr>
<tr>
<td>2.3.1 In-Circle Property</td>
<td>14</td>
</tr>
<tr>
<td>2.3.2 Maximizing Angle Property and Ray Shooting Monotonicity</td>
<td>16</td>
</tr>
<tr>
<td>2.3.3 Approximating a Complete Euclidean Graph</td>
<td>19</td>
</tr>
<tr>
<td>2.4 Dynamic Computational Geometry</td>
<td>19</td>
</tr>
<tr>
<td>2.4.1 Dynamic Computational Geometry for Discrete Motion</td>
<td>21</td>
</tr>
<tr>
<td>2.4.2 Dynamic Computational Geometry for Continuous Motion</td>
<td>22</td>
</tr>
<tr>
<td>2.5 Dynamic Update in Delaunay Triangulation</td>
<td>23</td>
</tr>
</tbody>
</table>
2.5.1 Successive Deletion and Insertion ............................................. 23
2.5.2 One-Time Update ............................................................... 24
2.5.3 Lazy Update ................................................................. 25
2.6 Kinetic Data Structures Approach ............................................. 27
  2.6.1 Basic Idea ................................................................. 27
  2.6.2 Davenport-Schinzel Sequence .......................................... 30
  2.6.3 Kinetizing Delaunay Triangulation .................................. 32
  2.6.4 Problems in KDS ......................................................... 32

3 COMPUTATIONAL TECHNIQUES FOR SITUATION AWARENESS .... 36
  3.1 Applying the Theory of Perception ....................................... 36
    3.1.1 Preattentive Processing ............................................... 37
    3.1.2 Gestalt Perceptual Processing ...................................... 38
  3.2 Density Computation ....................................................... 40
    3.2.1 Geometry Based Approach ........................................... 40
    3.2.2 Image Based Approach ................................................ 40
    3.2.3 Hybrid Approach ....................................................... 42
  3.3 Clustering ................................................................. 42
    3.3.1 Various Clustering ..................................................... 43
    3.3.2 Delaunay Based Approach ............................................ 44
  3.4 Lethality Assessment ....................................................... 46
    3.4.1 Probabilistic Model .................................................... 46
    3.4.2 Image Based Approach ................................................ 48

4 VISUALIZATION AND ANIMATION ISSUES ................................. 54
  4.1 Density Visualization ....................................................... 54
    4.1.1 Direct Shading .......................................................... 55
    4.1.2 Blobby Shading .......................................................... 55
  4.2 Lethality Assessment Visualization ..................................... 58
  4.3 Clustering Visualization ................................................... 58
    4.3.1 Boundary Detection in Human Visual System .................... 58
4.3.2 HSV Scheme ........................................ 58
4.4 Motion Scripting ........................................ 59
  4.4.1 Behavioral Animation ............................... 61
  4.4.2 Cognitive Modeling Approach ...................... 63
4.5 Terrain Visualization ................................... 65
4.6 Zooming Issues ....................................... 66
5 EXPERIMENT RESULT .................................... 70
  5.1 Implementation Workbench .............................. 70
  5.2 Dynamic Update on a Delaunay Triangulation ....... 71
  5.3 Various Computational Result ......................... 75
  5.4 Approximating Delaunay Triangulation ................ 76
6 SUMMARY .................................................. 79
  6.1 Conclusion ........................................... 79
  6.2 Future Work .......................................... 81
LIST OF REFERENCES ...................................... 84
VITA .......................................................... 89
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Dynamic Update on a Delaunay Triangulation</td>
<td>73</td>
</tr>
<tr>
<td>5.2</td>
<td>Simulation Performance Result</td>
<td>78</td>
</tr>
<tr>
<td>5.3</td>
<td>Experiment on Approximating Delaunay Triangulation</td>
<td>78</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 In-Circle Property of Delaunay triangulation</td>
<td>15</td>
</tr>
<tr>
<td>2.2 Fixed radius search on Delaunay triangulation using DFS</td>
<td>17</td>
</tr>
<tr>
<td>2.3 Ray Shooting Monotonicity of Delaunay triangulation</td>
<td>18</td>
</tr>
<tr>
<td>2.4 Delaunay triangulation approximates a complete Euclidean graph</td>
<td>20</td>
</tr>
<tr>
<td>2.5 Two Steps in Lazy update on Delaunay triangulation</td>
<td>26</td>
</tr>
<tr>
<td>2.6 Convex hull reestablishment when a point $p_1$ moves to $p_2$</td>
<td>29</td>
</tr>
<tr>
<td>2.7 Illustration of kinetizing Delaunay triangulation when a point $p_1$ moves to $p_3$ through $p_2$</td>
<td>33</td>
</tr>
<tr>
<td>3.1 Various Maps</td>
<td>39</td>
</tr>
<tr>
<td>3.2 Image based approach to the density computation</td>
<td>41</td>
</tr>
<tr>
<td>3.3 A typical example of clustering in the application</td>
<td>50</td>
</tr>
<tr>
<td>3.4 Threatening Region Model</td>
<td>51</td>
</tr>
<tr>
<td>3.5 One Step in Lethality Computation Using Image Based Approach</td>
<td>53</td>
</tr>
<tr>
<td>4.1 Comparison Of Density Visualization</td>
<td>56</td>
</tr>
<tr>
<td>4.2 Battlefield Visualization</td>
<td>60</td>
</tr>
<tr>
<td>4.3 A Behavioral Animation Example</td>
<td>62</td>
</tr>
<tr>
<td>4.4 Five Parameters in Boids</td>
<td>64</td>
</tr>
<tr>
<td>4.5 A Different Zooming Strategy</td>
<td>67</td>
</tr>
<tr>
<td>5.1 Update Time Comparison on a Delaunay Triangulation</td>
<td>74</td>
</tr>
</tbody>
</table>
ABSTRACT

Kim, Young Jun, Ph.D., Purdue University, August, 2000. Visualization and Animation for Situation Awareness in Battlefield. Major Professor: Christoph M. Hoffmann.

A battlefield is a complex multidimensional environment populated with a large number of various battle entities. The environment is constantly evolving and updated by a positional change of each entity. In order to win the battle, a commander must possess a clear understanding in a time-critical manner of who is where and what is happening; that is, a commander must build good Situation Awareness (SA). The task of building SA introduces a huge cognitive overhead due to the information complexity residing in the representation of battlefield. It is even more difficult to combine the understanding of individual aspects in the battlefield into a single comprehensive view. In this dissertation, we identify various visual abstractions of the battlefield that help a military commander to achieve situation awareness of the battlefield, and we provide techniques to efficiently compute them. The package also includes terrain visualization which has been traditionally favored in the battlefield visualization. Computationally, we explore the techniques from the point of view of dynamic computational geometry. In particular, Delaunay triangulation and its dynamic update have been extensively investigated. We also consider how to visualize
the result in both computationally and perceptually efficient ways. The principles of human visual perception play an important role in efficient delivery of the representation to the human visual system. Finally, using behavioral animation techniques, we simulate the battlefield which provides the inputs to the visual computation.
Suppose you are a top U.S. air defense commander boarding on the E-3 Sentry AWACS aircraft, in charge of making a strategic decision on the intrusion of enemy forces in a timely manner. You would be provided with tens of thousands of tactical data arriving from every corner of world every minute. Your job is to quickly grasp, and react to, changes in the evolving battle; you must possess good situation awareness (SA). This problem is known as command and control in the military community. However, understanding the huge positional data and combining it into a single comprehensive view of the battlefield can be extremely difficult, error prone and time consuming;[DIC+98, DJC+98]. Thus we need to reduce the information complexity.

We postulate that geometric queries on the positional data set and their appropriate visual presentation can reduce such a cognitive overhead. In this dissertation, we present various techniques from visualization and computational geometry to tackle those problems. The principles of human visual perception play an important role in this work too.

Most of work in this dissertation is developed in the context of the battlefield visualization package. However, our work is contrasted with previous ones in that we
successfully combine the battlefield visualization package with the perceptual consideration for a commander through situation awareness. Also, the main computational contribution from our work is that we provide various practical solutions to maintain the important geometric structure, Delaunay triangulation, in a dynamic setting, and illustrate their performance.

1.1 Problem Formulation

We start this section by explaining two important notions in our work, Situation Awareness and Battlefield Visualization. Then, we explain how we model the battlefield.

1.1.1 Situation Awareness

Situation awareness is the concept to describe the performance of a domain expert during the operation of complex systems, such as aircraft, vehicle, and chemical plants. The concept was first issued to discuss the critical difference between ordinary fighter pilots and ace pilots; [Suk97]. Due to the relative importance of the different aspects in situation awareness, there has been a broad range of definition as to the situation awareness.

From the point of view of human factors, Endsley [End88, Suk97] describes the situation awareness as follows;

“An expert’s perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future.”
From the military standpoint of view, Blanchard [Bla96] depicts battle space awareness as “knowing what is needed to win with the minimum number of casualties”. He identified situation awareness as one of seven core concepts in the battle space awareness, and referred to it as the situations of friendly and enemy forces.

1.1.2 Battlefield Visualization

Traditionally, and up until now, paper maps and acetate overlays are used predominantly in the military to visualize the situation. Only very recently, virtual reality technology and high computing power has started to take the place of acetate sheets and has proven to work well in a real battle scenario. Durbin and et al claim that such a battle space awareness system must present a comprehensive and timely view of the battle environment; [DIC+98]. The Army TRADOC pamphlet [UACAC95] further explains the term of battlefield visualization:

“The process whereby the commander develops a clear understanding of the current state with relation to the enemy and environment, envisions a desired end state which represents mission accomplishment, and then subsequently visualizes the sequence of activity that moves the commander’s force from its current state to the end state.”

Thus, we can conclude that helping a commander to develop the situation awareness plays a crucial role on the battlefield visualization, and thereby it directly relates to a success in mission accomplishment.
1.1.3 Moving Points in 2D

We model the battlefield as a 3D virtual world populated with two opposing groups of entities which are moving endlessly on a 2D plane. Every entity is assumed to report its positional data at a discrete time interval or continuously. Each entity belongs to either the red or the blue group, and each group is further subdivided according to its internal formation rule. We assume that such an internal formation rule about the blue group is known. Although each entity moves like a point on a 2D plane, it has a vertical elevation determined by the underlying terrain model, but the usage of the elevation data is restricted to visualization in the terrain model. The possible range of elevation values is assumed to be insignificant compared to the range of horizontal values in the 2D position.

1.2 Previous Work Review

In this section, we review various applications using situation awareness and battlefield visualization packages.

1.2.1 Situation Awareness Application

Many applications from avionics and from the human factors community are deemed a success because of the use of situation awareness. The goals of such applications range from training [KHC90] to tactical engagements [RJJ+94]. They adopt multi-modalities to increase the situation awareness of a human subject, using head mount displays (HMD), data gloves, data suits, chamber technology, and a cave environment. Most of them are also incorporated into an immersive display technology
or virtual reality facility. Extensive references to situation awareness in the avionics applications can be found at [RC00].

TSAS (Tactile Situation Awareness System) is a system to utilize human’s touch to deliver situation awareness. It is designed to improve the performance of a human pilot in simulated rotorcraft under high load working conditions. The unique feature of the system is that it does not rely on visual or aural information for an efficient delivery of the situation awareness unlike other packages. Instead, a wearable suit equipped with a tactile device is provided as an intuitive human computer interface to the three dimensional space. The system helps a pilot to avoid a three dimensional disorientation during the maneuver of rotorcraft; [fHC00].

Sentinel is a software tool to provide an analysis capability of the battlefield to a military commander. It helps maintaining situation awareness; [SBS93]. The tool gives the user an indication of the importance of action within a “watchspace”, a designated area by the user. The importance indication is displayed on the configurable status board, and the fuzzy logic determines its result. The goal of Sentinel is very similar to our application in the sense that it tries to make the complex battlefield environment intuitive by providing situation awareness. However, it lacks the consideration of a human perceptual process when displaying the importance factor in the status board. Therefore there is a possible danger that it can lead to a perceptual overhead or delay as the delivered information becomes complex and increases in volume.

The research on tactical driving of an autonomous vehicle shows a novel appli-
carnation of situation awareness; [Suk97]. SAPIENT(Situation Awareness Planner Implementing Effective Navigation in Traffic) is developed as a part of the Automated Highway System(AHS) to allow an intelligent vehicle operating without a human control. The goal of SAPIENT is to simulate the situation awareness of a human cognitive process in traffic. That way, the system achieves an intelligent behavior of a vehicle.

1.2.2 Battlefield Visualization Package

The primary concern in battlefield visualization packages has been real time terrain visualization, simply because the terrain is an important visual reference in the battlefield. Generally the terrain is modeled as a Triangulated Irregular Network (TIN), and in practice its data size can be as huge as tens of millions of triangles. Therefore, many efforts have been made to reduce the substantial volume of triangle data while minimizing visual artifacts. The recent progress in mesh reduction technique combined with Level of Detail (LOD) control has played a major role in that work; [CVM+96, Hop96, Hop97, RB92, SZL92]. The alternative to such object based techniques is using image based rendering and volume rendering techniques; [MMI+98]. Unfortunately none of these efforts were made in the context of battlefield visualization.

The Naval Research Laboratory's Virtual Reality Responsive Workbench (VR-RWB) and the Dragon software system are well known work in battlefield visualization. With the extensive use of virtual reality technology, the dragon system was considered a critical break-through over previous battlefield visualization
system;[DJC+98]. However the system does not employ any technology to reduce the cognitive overhead of a commander, and it merely provides a “3D metaphor” of a real battle.

Plan View Display (PVD) from MÅK technologies provides a \textit{birds-eye-view} into a simulated battle by overlaying entities and information onto 2D views of tactical, strategic, and visual databases. The overlayed information includes tracking individual entities and groups of entities and displaying intervisibility of entities and points. Moreover, since PVD is compatible with both Distributed Interactive Simulation (DIS) and High Level Architecture (HLA) simulation protocols, it is highly interoperable with other products;[MAK]. However PVD lacks consideration of the human visual perception process, such as preattentiveness, thus the visual presentation in PVD could bring about an additional cognitive overhead.

The Army Research Laboratory’s Virtual Geographic Information System (VGIS) has the same objective as our application; Providing visual information to a commander to help building situation awareness [WWE+00]. The main functionality of the system is displaying concentration information of battlefield entities. The system uses a grid(raster) based approach to compute the concentration, and constructs isosurfaces of concentration by considering the concentration value as vertical data in three dimensional space. Depending on the viewpoint, concentration is shown by height or by color intensity. Thereby, the vertical data serves as a redundant encoding of the concentration. Another functionality in the system is temporally condensing potentially lengthy battle into a MPEG movie. Later, the movie can be played back
at a desired speed. By doing this, the commander can grasp strategic or tactical implication of lengthy battles that might be missed otherwise.

1.3 Dissertation Overview

In Chapter 2, we provide the computational framework used throughout this dissertation. The framework is essentially founded on the paradigm of dynamic computational geometry, particularly Delaunay triangulation. The proximity information embedded in Delaunay triangulation makes the Delaunay triangulation employed as a key technology in the framework. We review the useful properties of Delaunay triangulation and its dual, the Voronoi Diagram, and review the algorithms to construct them. Notably, the dynamic update of Delaunay triangulation is extensively studied, including successive deletion and insertion, one-time update, lazy update, and the Kinetic Data Structures (KDS) approach. The update techniques fully utilize temporal and spatial coherence depending on the motion constraints imposed by the applications.

In Chapter 3, we identify various visual tools (density, clustering, and lethality assessment) that help a military commander to achieve situation awareness of the battlefield, and provide geometric techniques to efficiently compute them. In order to gain the perceptual effectiveness from the tools, Gestalt perception and perattentive features in the human visual system are considered, leading them to serve as existential legitimacy of the tools. For the density computation, we provide a geometry based approach, an image based approach, and a hybrid approach. For the clustering, we need a suitable definition for our application. Based on the definition, we show an
efficient computational method based on Delaunay triangulation. Finally, we present a probabilistic model for the lethality assessment and an efficient way to compute it using hardware support where available.

In Chapter 4, we consider how to visualize the result of the visual tools discussed in Chapter 3 in both computationally and perceptually efficient ways. An efficient hardware-accelerated technique is discussed for visualizing each tool. The visual tools are integrated into a battlefield visualization package. The package also includes terrain visualization which has been traditionally favored in the battlefield visualization. Also, using behavioral animation techniques, we simulate the battlefield which provides the inputs to the visual computation.

In Chapter 5, we present the experimental measurement and the implementation result for the techniques explained in Chapters 2 - 4. In particular, we quantitatively judge each update technique for Delaunay triangulation discussed in Chapter 2. We also provide a reasonable explanation behind the empirical result. More than that, a more radical approach on updating the Delaunay triangulation is empirically studied, and the results are supplemented with a theoretical justification.

In Chapter 6, the summary of the dissertation is given and the future work is suggested.
2. COMPUTATIONAL FRAMEWORK

One of important goals in battlefield visualization is to convey an abstract view relevant to the battlefield to a military commander in a such way that the commander can understand the situation of the battle with a minimal cognitive effort. The example of such an abstract view includes concentration, grouping, or threatening levels of opposing forces. These abstractions can be derived from positional information or movement information reported periodically or known in advance. Hence, the problem of deriving the abstraction naturally boils down to a proximity problem such as nearest neighbor search or fixed radius query. Moreover, in order to efficiently compute the proximity, temporal and spatial coherence must be fully utilized in the course of the derivation.

Traditionally the proximity problem has been widely studied in the context of computational geometry, but usually in a static setting. However, our problem setting requires handling the constant change of a point in motion. This additional requirement makes it a dynamic computational geometry problem. The nature of acquiring the motion data also varies depending upon the application. It is possible to make four different combinations from discrete or continuous motion, and from predictable or unpredictable motion. However, since continuous and unpredictable
motion or discrete and predictable motion is inconceivable or practically non-existent, those combinations are not considered further.

2.1 Various Approaches to Proximity Computation

Many intriguing techniques have been developed to efficiently solve the proximity problem. The types of techniques are as diverse as the problem domains in which they can be applied.

For the orthogonal searching in a multi dimensional data set, spatial data structures such as quadtree, octree, Kd-tree, and range tree have been developed; [BKOS97]. The primary target application for these techniques is a range query in geographic databases. Using fractional cascading, the most efficient two dimensional range query is running in $O(\log n + k)$ with an $O(n \log n / \log \log n)$ storage requirement. Here $n$ is the number of items in database and $k$ is the number of items resulting from a query. A particular difficulty in spatial data structures is that it is costly to dynamize the data structures, especially for the case of deletion operations; [Sam90]. It is known that the deletion operation can have a poor performance in the worst case.

For the proximity computation on three dimensional models, techniques such as scheduling scheme, sorting-based sweep and prune, and spatial subdivision are developed; [LG98]. The target application for most of them is collision detection among geometric models in a virtual environment. Particularly, the sweep and prune technique in the I-COLLIDE system is highly adaptable to dynamic environments. The technique exploits spatial and temporal coherence from the motion of objects.

Recently, in order to deal with massive data sets, an algorithm using external
memory and dynamic prefetching has been proposed; [WLML98]. The main data structure in the algorithm is called *overlap graph* which partitioned the data set into objects that are likely to represent actual contacts at run time. The overlap graph is built by multi-level graph partitioning.

Despite all those efficient techniques, the proximity computation on a point set is most efficient with a Delaunay triangulation based approach. Particularly, the dynamization is much easier with the Delaunay triangulation based approach than other techniques.

2.2 Voronoi Diagram and Delaunay triangulation

It turns out that many important proximity problems can be reduced to a relatively easy computation on a Voronoi diagram or its dual, a Delaunay triangulation. For example, the closest pair, convex hull, Euclidean minimum spanning tree, and all nearest neighbor search problems, are all linearly transformable to some computation on Voronoi diagrams. As a matter of fact, this is a conceptually trivial result, since a Voronoi diagram embeds proximity information in its geometric structure. Consequently Delaunay triangulation based computation has been employed as the main computational framework of our work.

Voronoi Diagram or Delaunay triangulation is one of the most extensively studied geometric structures in computational geometry literature. The Voronoi Diagram of a 2D point set is defined as a convex cell partitioning of the plane such that each convex cell is composed of the set of points for which a single site in the point set is closest. Delaunay triangulation is the straight line dual of of the Voronoi Diagram.
Delaunay triangulation can be defined independently of the duality to the Voronoi Diagram. This will be looked into more in the following section.

Many algorithms to compute the Voronoi Diagram have been developed. Among them, the classical “Divide and Conquer” algorithm and the Plane Sweeping algorithm, better known as “Beach Line” algorithm, are most commonly known algorithms running in $O(n \log n)$ time. Even though Delaunay triangulation and Voronoi Diagram are linearly transformable from each other, many algorithms to compute the Delaunay triangulation have been also developed. There are two reasons for it. One is that Delaunay triangulation has a direct application on its own such as the finite element meshing, and generates a good mesh. The other reason is that an algorithm to compute a Delaunay triangulation is easier to devise due to the locality of Delaunay triangulation than the Voronoi Diagram counterpart. The locality in Delaunay triangulation means that a local Delaunay property in a Delaunay triangulation guarantees a global Delaunay property. Using locality, one can easily devise a straightforward algorithm to compute a Delaunay triangulation by consecutively flipping each triangulated faces that do not satisfy the local property, starting from an arbitrary triangulation. This algorithm is known as flipping algorithm and runs in $O(n^2)$ time in a worst case. The optimal algorithm such as the incremental Delaunay triangulation algorithm also capitalizes on the locality property of Delaunay triangulation. Incrementally the algorithm locates a face into which a new point should be inserted and flips faces as required consecutively. This is repeated until all the points have been inserted. Recently, a lot of work has been done on a randomized version
of the incremental algorithm in order to guarantee the optimal performance in the average case as well. Beside the simplicity, the incremental algorithm is particularly easy to make dynamic. Similarly as in the insertion case, one can perform the deletion of a point by removing the point and retriangulating the faces to which the deleted point was incident, followed by edge flipping.

2.3 Delaunay Triangulation Based Approach

2.3.1 In-Circle Property

Delaunay triangulation in 2D is defined as a triangulation of points where the circumcircle of each triangulated face does not contain any other points than those on the face.

Such in-circle property is a direct result from the fact that the Delaunay triangulation (or more precisely the Delaunay Diagram) is a dual of the Voronoi Diagram. The in-circle property makes it possible that many proximity questions in 2D are reduced to relatively easy computations on a Delaunay triangulation. For instance, the closest pair or Euclidean minimum spanning tree is a subgraph of Delaunay triangulation. It is not a coincidence that our computational framework heavily exploits the Delaunay triangulation, since most geometric queries arising in our application are essentially proximity questions. For example, neighborhood search within a given threshold \(d\) is a typical geometric query arising often in the application. The search computation is based on the following lemma on Delaunay triangulation.

**LEMMA 2.3.1** [DD90] Let \(S\) be a set of distinct points on a plane, \(\delta\) a distance,
Figure 2.1. In-Circle Property of Delaunay triangulation

In a Delaunay triangulation, the circumcircle of every triangle does not contain any other points except for those lying on the face.
and $D$ the Delaunay triangulation of $S$. If $|p, q| \leq \delta$ for $p, q \in S$, then either $<p, q>$ is an edge in $D$ or there exist distinct points $o_1, o_2, \ldots, o_m$ such that:

1. $<p, o_1>, <o_m, q>$ and $<o_i, o_{i+1}>$ are edges in $D$ for $1 \leq i < m$,

2. $|p, o_i| \leq \delta, |o_m, q| \leq \delta$, and $|o_i, o_{i+1}| \leq \delta$ for $1 \leq i < m$, and

3. $|p, o_i| \leq \delta, |o_i, q| \leq \delta$ for $1 \leq i < m$.

Depth first search (DFS) based on the above lemma suggests a simple strategy to compute the neighborhood in $O(k)$ time, where $k$ is the number of reported nodes.

Since the search does not consider any unnecessary further visit, the performance of search algorithm is strictly output sensitive. This is not the case for spatial division methods such as quad tree or octree. The neighborhood search can be extended to the query composed of an arbitrary polygon. Even though any triangulation can be used for this purpose, the Delaunay triangulation provides a nice performance analysis in this case.

### 2.3.2 Maximizing Angle Property and Ray Shooting Monotonicity

Besides the in-circle property, Delaunay triangulation has other useful properties such as maximizing angle property or ray shooting monotonicity, and approximating the complete Euclidean graph. The maximizing angle property is important for finite element methods, because it provides a good linear interpolant with a good mesh structure. We used such a linear interpolant for the visualization of density distribution as explained later on. Also the Delaunay triangulation provides a good TIN model, and enables a terrain model to be rendered smoothly.
Figure 2.2. Fixed radius search on Delaunay triangulation using DFS

When one needs to perform a fixed radius $r$ search starting from source $S$, a simple DFS traversal on a Delaunay triangulation solves the problem. The above figure illustrates such a traversal for the fixed radius search. In the figure, $a, b, c, d, e, f$ denotes the order of the traversal. The traversal does not explore unnecessary nodes, either whose distance from $S$ is farther than $r$ or whose edge length is longer than $r$. 
Consider the ordering of faces based on the intersection with any ray shooting from an arbitrary node $S$. Then this ordering is monotonic; that is, there is no cycle.
The ray shooting monotonicity property provides a foundation for the simple segment walking technique, which has been a practical choice for the face location on Delaunay triangulations [GS85, MN98]. Many other variants of this technique are conceivable particularly using the randomized approach, for example “jump and walk”.

2.3.3 Approximating a Complete Euclidean Graph

The complete Euclidean graph can be approximated by a Delaunay triangulation within the ratio of less than 2.42 as the following theorem shows.

**THEOREM 2.3.1** [KG89] Let \( p \) and \( q \) be a pair of points in a set \( S \) of \( N \) points in the plane. Let \( d(p, q) \) be the Euclidean distance between \( p \) and \( q \) and let \( DT(p, q) \) be the length of the shortest path from \( p \) to \( q \) in the Delaunay triangulation of \( S \). Then,

\[
\frac{DT(p, q)}{d(p, q)} \leq 2.42
\]  

(2.1)

The property suggests that we can approximate neighborhood of a node and know how it is distributed by following incident edges of the node in Delaunay triangulation. So we can have an quick approximation of the density or concentration of each point using above property. More than that, the triangulation can be also used to smoothly interpolate computed densities, and finally to visualize the density distribution. More details are given later in section 3.2.1.

2.4 Dynamic Computational Geometry

When we deal with a motion data set, its associated geometric data structure should be updated accordingly in order to reflect the positional changes in the set.
Figure 2.4. Delaunay triangulation approximates a complete Euclidean graph.

For any nodes $S$ and $D$ on a Delaunay triangulation, the length of the shortest path (blue line in the figure) from $S$ to $D$ in the Delaunay triangulation is no longer than 2.42 times the Euclidean distance (red line in the figure) between $S$ and $D$. 
This means, in our case, that we need to update the Delaunay triangulation in a
dynamic fashion. Furthermore, in order to get an efficient update, spatial or temporal
coherence inherent to the motion data set must be exploited on as well. Surely the
coherence that we can exploit heavily depends on the nature of movement. As we
have seen earlier in the beginning of this chapter, one can have two possible scenarios
of the movement; discrete and unpredictable motion or continuous and predictable
motion.\footnote{Without confusing the reader, from now on we refer discrete motion to the discrete and unpredictible motion, and continuous motion to the continuous and predictable motion. We also refer to a \textit{kinetic} setting to describe the nature of continuous motion.}

This kind of study to investigate the dynamic update of a geometric structure
is known as \textit{dynamic computational geometry}. The study of dynamic computational
geometry is further refined into two different categories in the computational geometry
literature. Interestingly, the refinement happens to coincide with the categorization
of the movement mentioned above.

\textbf{2.4.1 Dynamic Computational Geometry for Discrete Motion}

A well known concept of dynamic computational geometry is a field of study
where efficient insertion and deletion operation into a certain geometric structure is
investigated. In the context of our application, the geometric structure of interest
is Delaunay triangulation. This technique is well suited for the discrete motion of
points, whose location is reported periodically at discrete time samples. For example,
in the military application, positional information of each battle unit is reported to
Command Operations Center (COC) at periodic time intervals. In this case, we have
no choice but use the first methodology, since we do not have any other coherence from the movement that we can exploit than a local spatial coherence over a few time frames of the report. In particular, when future positional data is completely unpredictable, for instance when tracking an espionage unit of adversary forces, the usage of local spatial coherence is more restricted. Say, between previous time frame and current time frame of the report.

2.4.2 Dynamic Computational Geometry for Continuous Motion

The other concept of the dynamic computational geometry, introduced by Atallah [Ata85], investigates a combinatorial change of a “configuration function” related to a property of interest and its efficient update, when points move continuously. The motion is assumed to be known in advance as a functional form in time. For example, in the case of civil aviation, one can safely assume that a plane follows a predictable trajectory. In this case, one can compute a critical moment when the configuration function or its relevant sub-function changes. Hence, the overall update computation can be very efficient, since the computation is performed only when it is needed. In other words, the spatial and temporal coherences are fully utilized based on the critical moment computation. Recently this technique has been elaborated in Kinetic Data Structures (KDS) by Basch and et al; [BGH97, BGSZ97, BGZ97]. While Atallah’s seminal work on the dynamic computational geometry was focused on the theoretical aspect of the algorithm, KDS has been reviewed both theoretically and practically. Both works use Davenport and Schinzel sequences to derive a theoretical bound on the performance of the algorithm.
In the following two sections, we investigate each methodology in more detail and explain further why we have decided to adhere to the first concept of dynamic computational geometry, which, in our case, corresponds to a dynamic update in a Delaunay triangulation.

2.5 Dynamic Update in Delaunay Triangulation

Delaunay triangulation is a primary data structure in our application. As points (in our case, battle units) move, we must update the Delaunay triangulation too. In the case that we are not able to get a future positional report of points or such prediction is unreliable, we must use only a contemporary report. Basically, this is the case for the military application. We assume that we are provided with a positional report as a track file at discrete time intervals. At each interval, we update a Delaunay triangulation using only a current report and its prior report. Three approaches are considered for updating a Delaunay triangulation in this setting.

2.5.1 Successive Deletion and Insertion

When a point has moved, we simply delete the point at the old position from the Delaunay triangulation, and re-insert it in the new position. The best insertion and deletion algorithms use randomization techniques, where insertion takes $O(\log n)$ expected time and deletion takes $O(k \log k)$ expected time ($n$ is the number of nodes in the triangulation and $k$ is the number of incident edges of the deleted node). The algorithms require maintaining an extra search structure, such as the Delaunay tree, besides the triangulation; [DMT92, Dev98]. The search structure is used for locating the triangle into which the point should be re-inserted.
Library implementations, such as LEDA or CGAL, have chosen a more straightforward but less sophisticated implementation for this. Instead of maintaining an extra search structure, they use segment walking to locate points, update the triangulation locally, and finally perform consecutive Delaunay flipping to reestablish the Delaunay property; [MN98]. The simpler implementation takes $O(\sqrt{n} + k)$ and $O(k^2)$ expected time for each insertion and deletion operation respectively, where $n$ is the number of nodes and $k$ is the complexity of the face to which the new point belongs, i.e., the number of incident edges. When the segment walking is combined with bucketing, it provides a more effective, practical location method. For example, the location structure maintains a regular grid of known positions and the location query starts from the closest grid; [Sno97].

However when many points move or in a worst case when all points move, even complete recomputation from scratch easily beats successive deletion and insertion; [HKW+98].

2.5.2 One-Time Update

When a movement of points has spatial coherence, we can exploit this fact to reduce the computation needed for locating points. Suppose a point is moving slowly. We can use the last face from which the point was deleted as a starting position for the insertion search in segment walking. Since the majority of the insertion time is spent on locating the correct face, this simple modification cuts the total update time significantly. Furthermore, we can also reduce update time by performing Delaunay flipping at one time for all triangles generated by insertion and deletion.
2.5.3 Lazy Update

This approach also exploits spatial coherence. When points have moved only slightly or move while forming a group or flock, for instance formation movement in the military, the old Delaunay triangulation before points moved can be a good approximation even after they have moved. This observation suggests a slightly modified Delaunay flipping algorithm. The original flipping algorithm runs on $O(n^2)$ time in two steps as follows; 1) Construct arbitrary triangulation in $O(n \log n)$ and 2) perform Delaunay flipping in $O(n^2)$. However, instead of constructing the new triangulation from scratch as in the first step, we start with the old Delaunay triangulation. The old triangulation must be retriangulated only if necessary, but it would require few updates which should take much less than $O(n \log n)$ time. Furthermore the Delaunay flipping process also should take much less than $O(n^2)$ time, possibly $O(n)$ in some cases since most of edges are already Delaunayed.

Experimentally we have shown that even when all points move, the modified flipping algorithm beats the fastest Divide and Conquer algorithm by a factor of 2 and the original flipping algorithm by a factor of 4, see Table 5.1.

The retriangulation process is done by successive deletion and insertion into the old triangulation. The deletion process exploits the “star-shaped\footnote{When every vertex in a face $f$ is visible from a vertex $v$ in $f$, $f$ is called “star-shaped”} property” of the triangulated face where the node to be deleted is incident. The star-shaped face implies that we can always find a convex quadrilateral formed by two adjacent triangles from the face. By flipping the diagonal of such a quadrilateral, we can reduce the
Figure 2.5. Two Steps in Lazy update on Delaunay triangulation

When spatial coherence exists, the retriangulation step merely reduces to be a triangulation check most of time, see Figure (b). Furthermore, not many edges need flipping; in Figure (b) and (c), only edge $e_1$ must be flipped into $e_2$. 
degree of the node until it is lowered to three, and finally we can delete the node. The $POINT SET$ class in LEDA also exploits this property for the deletion of a point from the Delaunay triangulation; [MN98]. The insertion process is accomplished by performing the similar location operation as done in the one-time update: Starting from the deleted face, we walk toward the face to which a new point belongs. After correctly locating the face, we retriangulate it.

Since we are interested in knowing which other points are nearby, for each point, we do not have to maintain the precise Delaunay property for every movement of points as long as the triangulation is valid and the adjacency in it is a good approximation of what is nearby. Therefore we can use the retriangulation without flipping as an approximation of the new Delaunay triangulation. See Table 5.1 to know how closely the retriangulation can approximate the new Delaunay triangulation.

Lazy update technique is summarized in Algorithm 2.5.1.

2.6 Kinetic Data Structures Approach

2.6.1 Basic Idea

In a kinetic setting where future positions of all entities can be predicted as a continuous functional form, $f(t)$, an efficient algorithm known as KDS (Kinetic Data Structures) has been proposed. When devising an application of KDS, a configuration function is defined that quantifies properties and states of the configuration relevant to the application. The configuration function is represented implicitly as a set of certificates, assertions whose correctness implies the validity of the configuration function. A certificate is always an assertion related to a small number of geometric
ALGORITHM 2.5.1
UpdateDelaunayTriangulation($P_1$, $P_2$, $DT$)

**Input** A set $P_1$ of $n$ points in the plane and its Delaunay triangulation $DT$, and a set $P_2$ after the points have moved.

**Output** Return an updated $DT$.

1. for each point $p_1$ in $P_1$ and its future position $p_2$ in $P_2$, do
2.  
3.  
4.  
5.  

SimpleRetriangulate($p_1$, $p_2$, $T$)

**Input** A point $p_1$ in a triangulation $T$ and its future position $p_2$ after $p_1$ has moved.

**Output** If replacing $p_1$ with $p_2$ from $T$ forms a valid triangulation, modify $T$ accordingly and return true. Otherwise, do not replace and return false.

1. Let $f$ be the face of $T$ to which $p_1$ belongs, and let $E$ be the set of boundary edges of $f$.
2. for every non convex hull edge $e$ in $E$,
3.  
4.  
5.  
6.  
7.  
8.  


If $e_1$ and $e_2$ form a left turn (Case 1), a new hull edge *new hull* is created. Else if $e_1$ (upper hull) and its predecessor or $e_2$ (lower hull) and its successor form a left turn (Case 2), new hull edges *upper hull1*, ..., *lower hull1*, ... should be created consecutively until the upper hulls and lower hulls form a right turn with their predecessors and successors respectively.
entities. An event refers to changing a certificate, whether by expiration or because of a change in flight plan, f(t). Such events are computed using f(t) and pushed into a event queue (priority queue). Then, one can think of the KDS as a variant of a plane sweeping algorithm, where the sweeping proceeds in the time direction rather than along a principal spatial axis. At each time event, updating occurs in the KDS by popping out the event if necessary.

2.6.2 Davenport-Schinzel Sequence

Davenport-Schinzel (DS) sequence is a basic tool in many geometric applications for deriving bounds on the number of changes or events in a dynamic setting; [Ata85, BGH97, SA95]. The sequence has been used to analyze the combinatorial complexity of the lower envelope of univariate functions. The sequence is formally defined as follows:

**DEFINITION 2.6.1** [SA95] Let n, s be positive integers. A sequence U =< u₁, ..., uₘ > of integers is an DS(n, s) Davenport-Schinzel sequence, if it satisfies the following conditions:

1. 1 ≤ uᵢ ≤ n for each i.

2. For each i < m we have uᵢ ≠ uᵢ₊₁.

3. There do not exist s + 2 indices 1 ≤ i₁ < i₂ < ... < iₛ₊₂ ≤ m such that uᵢ₁ = uᵢ₃ = uᵢ₅ = ... = a, uᵢ₂ = uᵢ₄ = uᵢ₆ = ... = b, and a ≠ b.
We denote the possible maximum length of Davenport-Schinzel sequence, $DS(n, s)$ by $\lambda(n, s)$. In other words,

$$\lambda(n, s) = \max\{|U| \mid U \text{ is a } DS(n, s) \text{ sequence}\} \quad (2.2)$$

For example, $\lambda(n, 2)$ is the maximum length that a sequence of integers $\{u_1, \ldots, u_n\}$ can have without containing any pair of equal adjacent elements, such as $u_i u_i$, and without containing any $u_i u_j u_i (i \neq j)$ as a subsequence. The following lemma suggests the applicability of DS sequences to dynamic computational geometry and KDS;

**LEMMA 2.6.1 [Ata85]** Let $f_1, \ldots, f_n$ be real valued functions of time, each of which is continuous. If no two distinct functions $f_i$ and $f_j$ intersect more than $s$ times, then $h(t) = \min_{1 \leq i \leq n}\{f_i(t)\}$ is made up of no more than $\lambda(n, s)$ pieces, and this bound is best possible.

Surprisingly $\lambda(n, s)$ is almost linear with respect to $n$. Using lemma 2.6.1, we can estimate a number of discrete events at which a specific configuration function changes. Consider an example:

Assume there are $n$ straight lines in a plane, none of them parallel to the $y$-axis. Ask how many times can there be a change in the lower envelope of $n$ straight lines? In other words, as we move on from minus infinity to plus infinity on the $x$-axis, the lower envelope will change at specific $x$-values. How many times can there be such a change?
Clearly the answer to this simple question is \( n - 1 \), which is \( \lambda(n, 1) \). Instead of using \( n \) straight lines, if we use \( n^2 \) quadratic curves, the problem precisely corresponds to keeping tracking of the number of possible changes in the closest pair of \( n \) points\(^3\), which is \( \lambda(n^2, 2) = O(n^2) \). In this case, the point is assumed to move in a linear motion on the plane. As in the above example, Basch and et al uses the DS sequence to derive specific complexity bounds in KDS assuming the trajectories are linear.

### 2.6.3 Kinetizing Delaunay Triangulation

When the configuration function is a Delaunay triangulation, its certificate is particularly easy to devise. The certificate directly corresponds to the in-circle function in time of every four points forming a convex quadrilateral. We associate each edge with such a certificate. This type of certificates is called a “self-certifying” structure; [BGH97]. In this case, kinetization is immediate. Whenever the certificate of an edge has expired, we update the triangulation by flipping the edge and reschedule affected other edges.

This idea has been recently extended to higher dimensions; [AGMR98].

### 2.6.4 Problems in KDS

Even though KDS can be extremely efficient, we have discovered two major problems in our application. First, update events may be generated so often and accumulated in an event queue that even computation from scratch at that particular moment is faster than processing the update events in the queue one by one. This comes from the fact that a current KDS algorithm for the Delaunay triangulation does

\(^3\)This analysis is a worst case. The expected changes in the closest pair is known to be \( \Theta(n) \).
Figure 2.7. Illustration of kinetizing Delaunay triangulation when a point $p_1$ moves to $p_3$ through $p_2$

In Figure (a), a certificate $c_1$ is associated with an edge $e_1$. In Figure (b), when a point $p_1$ reaches a point $p_2$ while moving toward $p_3$, the certificate $c_1$ expires and flipping occurs; edges $e_1$ and $e_2$ are flipped. A certificate $c_2$ is, then, associated with $e_2$. Finally the point $p_1$ arrives at the point $p_3$ without any more certificate expires.
not satisfy the local criterion, meaning that the number of events that depend on a single entity is not polylogarithmic in the number of moving entities involved. Our application demands frequent interaction among entities, and it requires for all the interacting entities an update of their trajectory plans. This eventually causes many new rescheduling events due to the non-local property of the KDS applied to Delaunay triangulation. Recently, a new simulation technique to improve the performance of KDS is developed. The technique is based on interval arithmetic to reduce an unnecessary computation to calculate an expiration time that will not happen. Even though this speedup can not totally eliminate the possibility of “queue thrashing” in the dynamic maintenance of Delaunay triangulation, it is a still good technique and significantly improves the overall performance of KDS; [GK99]. Another possible reason for queue thrashing is that Delaunay triangulation inherently undergoes more changes compared to other configuration functions such as the closest pair or convex hull$^4$.

Second, in many applications we are rarely able to predict future positions as a functional form as must be assumed in KDS. Especially in our battle field application, the position of each entity in a battle unit is reported at discrete time intervals. In this case, we cannot predict future positions by a simple extrapolation technique. However when the past history of movement provides a good prediction of the future

---

$^4$Basch and et al showed that if the position and linear velocity of $n$ points are drawn independently at random from the uniform distribution on the square, their Voronoi diagram undergoes $\Theta(n^{2})$ combinatorial changes in expectation, whereas their convex hull undergoes $\Theta(\log^{2} n)$ combinatorial changes and their closest pair undergoes $\Theta(n)$ combinatorial changes; [BDIZ97]. In a worst case, the tightest upper bound on the number of combinatorial changes in Delaunay triangulation is known to be roughly cubic.
movement, we can formulate the possible trajectory of the movement and use a similar
technique like KDS. This kind of technique has been developed in the physically-based
simulation and discrete-event simulation communities. A statistical analysis on the
past movement pattern plays a key role in the formulation of trajectory; [Pop].
3. COMPUTATIONAL TECHNIQUES FOR SITUATION AWARENESS

In this section, we suggest three visual tools - *density, clustering* and *lethality assessment* - to enhance the situation awareness in the battlefield, and investigate how to compute them efficiently. In the following section, we explain how to visualize them.

We start this section by investigating the underlying theory on perceptual psychology to justify why we compute those visual tools and how we should present them visually.

3.1 Applying the Theory of Perception

Usually the battlefield is represented by large datasets with various attributes. These large datasets makes it difficult for a user, in our case a military commander, to assess the situation in the battlefield in a timely manner. Moreover it is more troublesome when it comes to dealing with the multi attribute or multi dimensional datasets, since the users must spend more time to build up a single comprehensive view to the battle.

We postulate that an appropriate visual interpretation of the battlefield helps a military commander to build up a comprehensive view of the battlefield effortlessly and rapidly, and as a result to make a strategic decision accurately. In order to
accomplish this objective, we address two important discoveries from the domain of perceptual psychology such as preattentive features and gestalt perception in human perceptual processing. Taking advantage of the low level human visual systems, these perceptual techniques allow users to perform exploratory tasks on large multi dimensional datasets rapidly, accurately, and effortlessly. Such tasks include identifying concentration(density), boundary detection(clustering), and threatening level( lethality assessment) from given large positional dataset.

3.1.1 Preattentive Processing

Psychophysicists have identified some limited set of features that the human visual system detects very quickly without the need for search. These features are called preattentive, and such features include color(hue), intensity, texture, length, and width. It is known that by using these features one can perform various exploratory tasks independent of the total number of elements involved in the tasks, and these tasks can be performed in a single glance taking in less than 200 milliseconds. It has also been shown experimentally that the preattentive features can work not only in a static setting but also in a dynamic setting where datasets are constantly changing. Furthermore, some preattentive features have no interferences with each other, and they seem to be prioritized by the human visual system. For instance, form, hue, and intensity are prioritized features; [Hea99].

In our application, such hierarchy of features are used for density and clustering visualization later in section 4.3.2. We use the HSV color scheme to exploit the feature hierarchy. Since we have only a limited number of preattentive features which
do not interfere with each other, we must limit the number of features coexisting in a single view. We resolve this problem by distributing some of features into different views (windows) while minimizing the number of views.

3.1.2 Gestalt Perceptual Processing

Early in the 20th-century, gestalt psychologists observed that when elements were gathered into a figure, the figure took on a perceptual salience that exceeded the sum of its parts. Then, two decades ago, it was demonstrated that people extract the global aspects of a scene before smaller (local) details are perceived; [Nav77, Hof80]. Taking advantage of this pre-attentive, global processing we can make the density distribution of troops within a geographical area instantly apparent, see Figure 3.1.

The user can thus determine troop concentrations with less visual interrogation and cognitive effort because the differential saturation is immediately apparent. From this, we postulate that density is a very important tool to enhance situation awareness; [EWW98, HKW+98].

It is previously known, and we also found out experimentally that some preattentive feature such as blinking can highly distract the gestalt perceptual process. Therefore, one should be very careful about putting various preattentive features together, and ensure that they do not interfere with each other while performing the preattentive processing.
Figure 3.1. Various Maps

In map (a) where an individual battle unit is displayed as a point, it takes some time to recognize the whole troop boundary and concentration. In a shaded map (b), even though the troop boundary becomes easy to recognize, it is still hard to see the troop concentration. In density map (c) and (d), the troop concentration becomes apparent. But a correct understanding of the troop concentration depends on a correct display.
3.2 Density Computation

3.2.1 Geometry Based Approach

By density of a point, we mean a measure of how many other points are close by. Both number and distance of the nearby points are considered. In the Delaunay triangulation, nearness is approximated by adjacency, See Theorem 2.3.1. Once the Delaunay triangulation has been computed, density assignment can be done as in Algorithm 3.2.1,

After all vertices have been processed, the average density is obtained as the density value divided by the number of incident edges.

The density computation requires $O(n)$ steps for the edge quantization and averaging, and $O(n \log n)$ steps for Delaunay triangulation construction.

3.2.2 Image Based Approach

In the image based approach, we assume the existence of a mapping from geographic coordinates to pixel positions displayed on a digitized map. Using the mapping, we define a local density function from each grid position or pixel to neighboring pixels. This local density function defines an area of influence around each pixel position containing at least one entity. For each entity, we sequentially compute the appropriate pixel position and then cumulatively applying the local density function to each neighboring pixel in the area of influence around that pixel. After all entities are processed, the result is a pixel array containing the global cumulative results of the repeated application of the local density functions; [HKW+98].

The image based approach is attractive because it is very easy to implement. Also,
**ALGORITHM 3.2.1**

ComputeDensityUsingDT\((DT)\)

**Input** A Delaunay triangulation \(DT\) of \(n\) points in the plane.

**Output** Density assignment on each vertex in \(DT\).

1. Set each vertex density to zero.

2. For each edge incident to vertex \(p\), compute the edge length and classify into one of a small number of length ranges.

3. Increase the density of \(p\) by an integer derived from the length range.

---

**Geographical Space**

**Raster Space**

![Diagram showing geographical space and raster space with local density function and density level](image)

Figure 3.2. Image based approach to the density computation

Based on the mapping from geographical space to raster space, the density is accumulated by superimposing the local density function.
because only a constant number of pixels are processed for each report, spatial and
temporal processing complexity can be made linear with respect to the number of
entity position reports.

3.2.3 Hybrid Approach

A particular difficulty in the image based approach is how to construct the local
density function in a uniform and automatic fashion. One possible solution is to
precompute a table of all the possible local density functions and retrieve it if nec-
essary. Clearly this requires a lot of extra storage and can not be easily customized
and extended by an end user.

When fast graphics hardware supporting, e.g., OpenGL is available, which is com-
mon in a modern graphics workstation, one can exploit the hardware to construct a
highly flexible local density function. The technique is similar to Manocha and et al’s
idea to compute Voronoi diagram [ICK+99], but applied from a different perspective.
We use the fast rendering pipeline and alpha blending function available in OpenGL
as in Algorithm 3.2.2.

We can further accelerate the density computation by storing the result of step 4
into a template buffer and reusing it later instead of re-rendering it.

This technique is applied also to lethality assessment, see Section 3.4.2.

3.3 Clustering

A military commander would be interested in formation information of adversary
forces in order to appropriately react to possible attack from the enemy or to counter-
attack the enemy. However, such formation information about the opposite side is not
known in general. In this situation, the commander should infer the enemy formation from the available data, usually positional data. For instance, a database file with useful attributes such as enemy position, date, and weapon type can be furnished via a spy satellite in a track file form. One possible way to infer the enemy formation from the track file is to use proximity of enemy entities to each other, since entities in the same unit tend to move together. In computational geometry, this kind of problem is considered a clustering problem.

3.3.1 Various Clustering

Depending on the application, the clustering problem has different objective functions. In general, the clustering problem is known to be NP-hard regardless of the objective function. Moreover, for Euclidean space, it is NP-hard to approximate, to within factors close to two, in higher than one dimension. Therefore, most clustering algorithms work only on a fixed number $k$ of clusters.

In $k$-center clustering or pairwise clustering, the objective function is minimizing the radius or diameter of each partitioned cluster. The doubling algorithm is a typical example of such a $k$-center clustering algorithm. Here, the objective function is to minimize the maximum cluster diameter. Motivated by an information processing application, it is an incremental clustering algorithm that does dynamic insertion of an item at a time. The algorithm runs in $O(k \log k)$ time per update. The time is spent mainly on maintaining a complete graph of centers of induced clusters. Its performance ratio to optimal clustering is 8 in any metric space; [CCFM97].

In variance-based clustering, the objective function is to minimize the sum of
squared errors in each cluster. The *Voronoi diagram based approach* is one such algorithm; [IKI94]. The main idea is that an optimum clustering which minimizes such an objective function is a Voronoi partition, i.e., the ordinary Euclidean Voronoi diagram for some $k$ points. Initially, the algorithm finds two linearly separable clusters using a randomized approach, and then recursively applies the same technique to each partition of clusters until $k$ clusters have been found. The algorithm, sampling $m$ points from total $n$ points, finds a 2-clustering whose clustering cost is within a factor of $1 + O\left(\frac{1}{m}\right)$ from the minimum clustering cost with high probability in $O(m^2n)$ time. However, due to exhaustive linearly separability checking, the algorithm is not suitable for a real time application.

### 3.3.2 Delaunay Based Approach

One can also think of a different definition of clustering. Suppose points belonging to the same cluster should move maintaining at most a given maximum distance from some of their neighbors. In this case, the outline of clustering can be any arbitrary shape, for example a *sickle* shape. This is a particularly important case for the military unit formation: Each entity in a military unit is moving while maintaining some distance from others, see Figure 3.3. We can define this type of clustering formally as follows:

**DEFINITION 3.3.1** *Given $n$ points in $\mathbb{R}^d$ and distance threshold $r$, find clusters $S_1, S_2, ..., S_k$ which satisfies the following*

1. $\forall x_i \in S_l, \exists x_j \in S_l, \text{ such that } |x_i - x_j| \leq r$. 
2. \( \forall x_i \in S_l, \forall x_j \in S_m \text{ where } l \neq m, \text{ we have } |x_i - x_j| > r. \)

Fortunately, this computation is not NP hard, and can be solved easily especially in 2D based on the lemma 2.3.1. We can compute clusters as in Algorithm 3.3.1.

Once a Delaunay triangulation has been computed, the computation requires \( O(n) \) running time for edge cutting plus extracting connected components by a Depth First Search (DFS) on the triangulation.

As points move, we must update clustering. This task involves two subtasks; Dynamic update of Delaunay triangulation and connected component. The first subtask can be accomplished as explained in Section 2.5. The second subtask boils down to maintaining a spanning forest in a dynamic setting. This kind of problem is better known as a dynamic connectivity problem, since, if we have a spanning forest of a graph \( G \), we can quickly answer whether two vertices in \( G \) are connected or not. There are three different known algorithms related to the dynamic connectivity problem; Fredrickson’s topology tree [Fre97], sparsification [EGIN92], and Henzinger and King’s randomized algorithm [HK95]. The Henzinger and King algorithm achieves an expected amortized update time of \( O(\log^3 n) \) for a sequence of at least \( m_0 \) updates, and experimentally it was shown to be the fastest algorithm for random inputs [ACI97]. However, in the dynamic setting where all points move at discrete time intervals, we might as well recompute the connected component from scratch.
3.4 Lethality Assessment

Consider tank warfare. Every battle unit, a tank, has an associated threatening region which also has a preferred direction, the direction into which the on-board cannon of the tank is positioned. Typically it is the direction in which the tank moves. The commander must assess overall lethality distribution of all battle entities involved in the warfare, in order to appropriately avoid a “danger zone” or even safely retreat friendly forces from the combat zone if necessary.

In light of the computation, a lethality assessment is similar to the density computation discussed in Section 3.2 in that the concept of threatening region merely replaces the local density function in the density computation.

3.4.1 Probabilistic Model

We model the threatening region with a probability density function that has exponential decay in two dimensions, see Figure 3.4.

Here the probability measures the likelihood of being killed at a certain position in the plane. Note that the probability outside the preferred direction range is zero. We model the killing probability by the penetration ratio of steel armor shot by a projectile. Ideally, the penetration is fully governed by the kinetic energy $E$ of the projectile [Oku98].

Consider one dimensional trajectory of a projectile fired by a cannon, where air resistance is the only force exerted on the projectile after shooting. The other parameters such as gravity, firing angle and wind are ignored. Since the projectile moves fast, the air resistance increases proportionally to the square of velocity (Newtonian
drag) $F = -kv^2$, where $k$ is the coefficient of air resistance [Wil00]. Thus we get the following equation of the motion for the projectile,

$$m \frac{d^2x}{dt^2} = -k \left( \frac{dx}{dt} \right)^2$$  \hspace{1cm} (3.1)$$

where $m$ denotes mass of the projectile and $k$ denotes the air resistance coefficient.

By solving the nonlinear differential equation 3.1, we get the following equations for the velocity $v$ and the displacement $D$ of the projectile,

$$D = x = \frac{m}{k} \log \left( \frac{v_0kt + m}{m} \right)$$  \hspace{1cm} (3.2)$$

$$v = \frac{dx}{dt} = \frac{mv_0}{v_0kt + m}$$  \hspace{1cm} (3.3)$$

where $v_0$ denotes the nozzle velocity of the projectile.

From equation 3.2 and 3.3, we get the following relationship between the kinetic energy $E$ and the displacement $D$,

$$E = \frac{1}{2}mv^2$$

$$= \frac{1}{2}m \left\{ v_0 \exp \left( -\frac{kD}{m} \right) \right\}^2$$

$$= \frac{1}{2}mv_0^2 \exp \left( -\frac{2kD}{m} \right)$$

$$= E_0 \exp \left( -\frac{2k}{m}D \right)$$  \hspace{1cm} (3.4)$$

Here, we can see the exponential decay of the penetration (kinetic energy) as a function of the displacement.

Once all threatening regions have been defined, we compute the overall lethality using the the additive rule for two independent random variables; i.e. $p_{\text{composite}} = p_1 + p_2 - p_1p_2$. 


3.4.2 Image Based Approach

The computational method used for the lethality assessment is similar to the hybrid approach in section 3.2.3, except for using a different blending formula. The main idea is that we interpret the destination color \((D_c)\) as probability \(p_1\) with alpha value \((A_2)\) \(1 - p_2\), and the source color \((S_c)\) as probability \(p_2\) with value alpha \((A_1)\) \(1\), since

\[
D_c = S_c A_1 + D_c A_2
\]

\[
= p_2 \cdot 1 + p_1 (1 - p_2)
\]

\[
= p_1 + p_2 - p_1 p_2
\]  

(3.5)

Initially \(p_1\) is zero, and iteratively accumulates the composite probability by blending the probability \(p_2\) of each threatening region. So the whole process is given in Algorithm 3.4.1.

Unlike the density computation, we cannot reuse the result of step 4 since the threatening region is constantly changing.
ALGORITHM 3.2.2

\text{ComputeDensityUsingAlpha}(P)

\textbf{Input} A set \( P \) of \( n \) points in the plane.

\textbf{Output} A buffer \( B \) containing density assignment of \( P \).

1. Define the local density function either in an explicit or implicit form.
2. \textit{Polygonalize} it.
3. Set the source and destination alpha to one in the alpha blending function\textsuperscript{1}; i.e. \( \text{DestinationColor} = \text{SourceColor} \cdot 1.0 + \text{TargetColor} \cdot 1.0 \).
4. For each entity \( p \) in \( P \), render the polygonalized density function using the above alpha blending formula.
5. The resulting density distribution is stored in a frame buffer.
Figure 3.3. A typical example of clustering in the application

Each cluster has its own threshold value to be its member, which is a minimum distance between members. The dotted arrow between any of two points denotes such a minimum distance within a cluster.
ALGORITHM 3.3.1
ComputeClustering\((P, r)\)

**Input** A set \( P \) of \( n \) points in the plane, and minimum distance to neighbors in a same cluster, \( r \).

**Output** Return clustered sets \( S_1, S_2, ..., S_k \) of \( P \).

1. Compute a Delaunay triangulation of given points.
2. Cut edges whose length is more than the given threshold, \( r \).
3. Compute connected components, and output each component as a different cluster \( S_i \).

Figure 3.4. Threatening Region Model
ALGORITHM 3.4.1
\texttt{AssessLethalityUsingAlpha}(P)

\textbf{Input} A set \( P \) of \( n \) points in the plane.

\textbf{Output} A buffer \( B \) containing lethality assessment of \( P \).

1. Define the threatening region either in an explicit or implicit form.

2. \textit{Polygonalize} it.

3. Set the alpha blending formula as in equation 3.5.

4. For each threatening region, render its polygonalized function using the above alpha blending formula, See the illustration in Figure 3.5.

5. The resulting lethality assessment is stored in a frame buffer.
1) Render a Lethality Region on Sc.

2) Copy the region on Alpha2, but set it as a complement.

3) Blend all.

Source Color(Sc) = p2

Source Alpha(A1) = 1

Dest Alpha(A2) = 1-p2

Dest Color(Dc) = p1

Figure 3.5. One Step in Lethality Computation Using Image Based Approach
4. VISUALIZATION AND ANIMATION ISSUES

In this section, we address the problem of how to render the concepts that we discussed in previous section. In order to assure situation awareness, the visualization should be delivered to a commander both perceptually meaningfully and computationally efficiently. In addition, the delivery must fully utilize the preattentiveness of the human visual system in order to minimize the cognitive overhead of users.

We also address the animation issue for moving entities. We simulate the military formation movement over terrain using behavioral animation technique. Finally, we briefly mention the terrain visualization techniques employed in the animation.

4.1 Density Visualization

Depending on the way we compute the density, we can render the density distribution using different approaches. If the density has been approximated by Delaunay approach, the underlying Delaunay triangulation provides a quick rendering method utilizing hardware supported acceleration. If more intuitive visual presentation is needed, Blobby shading meets the demand. Since both the image based approach and the hybrid approach maintain a raster, the raster can be rendered immediately by redirecting the raster into a “frame buffer”. These methods are to be looked into in more detail.
4.1.1 Direct Shading

Once a density has been assigned to each point, we have reduced the problem to height interpolation and terrain visualization. A simple way to do this is to use the piecewise linear interpolant induced by the Delaunay triangulation. Considering the density value of each vertex as an intensity, we render the polygonal terrain using Gouroupr shading. This is especially easy with OpenGL. Note that good OpenGL implementations exploit available hardware for this, and the approach is suitable for real-time. This method, however, cannot render density outside the triangulation. Moreover, the rendered image looks polygonal especially when there are only a few points.

4.1.2 Blobby Shading

Another alternative to the visualization of density is rendering by blobby shading. Blobby shading was motivated by molecule visualization, where the electron in an atom is represented as a density function of the spatial location; [Bli82]. Simplifying, the density distribution is obtained by summing the contribution from each atom separately:

\[ D(x, y, z) = \sum_i b_i \exp(-a_i r_i) \]  \hfill (4.1)

where \( a_i \) is the radius parameter of atom \( i \), \( b_i \) is a “blobbiness” parameter for atom \( i \), and \( r_i \) is the distance from the point \((x, y, z)\) to the center of atom \( i \).

We apply blobby shading by considering the density value of each point as a radius...
(a) Direct shading (with Delaunay triangulation)

(b) Blobby Shading

Figure 4.1. Comparison Of Density Visualization
parameter of each atom. Using the equation 4.1, for every pixel in a screen window, we compute its blobby sum from all points based on the distance between the pixel and each point.

The brute-force implementation of this approach is much more expensive than the triangle-based direct shading, but there are many possibilities to increase efficiency. One possibility is hierarchical decomposition used in N-body algorithms [BN97]. Moreover, the computation is inherently parallel. Improvements are also possible by applying interpolation techniques at the raster level. Finally, when the density is computed by the image based approach or hybrid approach using the equation 4.1 as local density function, its visualization is immediately available; Just dump the resulting frame buffer onto screen. Thus when the primary goal of density computation is only its visualization, the hybrid approach is a most flexible and efficient approach. The computation and visualization time of the hybrid approach is significantly faster than the brute-force implementation of blobby shading.

Blobby shading provides a highly intuitive presentation of the density distribution. Moreover, it shows pleasing boundaries and does not degrade in quality for a small number of points.

When density has been computed either by image based approach or by hybrid approach, the computation result is stored in a raster buffer. By construction, the raster directly corresponds to the approximating result of Blobby shading. Therefore the rendering process corresponds to a simple dumping operation from the raster buffer to the frame buffer residing in a graphics display. However, when the raster size
is not sufficiently big compared to the frame buffer size, the result of the computation is not only inaccurate, but there could be visual artifacts similar to aliasing as well.

4.2 Lethality Assessment Visualization

Since the result of lethality assessment computation is readily available at the frame buffer, we can directly dump it onto the screen to visualize it, see Figure 4.2-(c). Generally in a graphics work station, the dumping process is very fast because it is facilitated by Direct Memory Access (DMA).

4.3 Clustering Visualization

4.3.1 Boundary Detection in Human Visual System

Researchers from both the cognitive psychology and scientific visualization domains have found out various preattentive features to assist in the boundary detection in human visual system, and such features include hue, form(shape) and intensity; [Hea99]. Callaghan reported that the visual system seems to prioritize the features in order of importance. In his feature hierarchy, the intensity is more important than hue to the low-level visual system during boundary identification, and the hue is more important than form; [Cal84, Cal89, Hea99]. Moreover, it turns out that the feature hierarchy still applies to a dynamic environment where a real time sequence of frames is displayed at ten frames per second. We exploit this fact for clustering visualization.

4.3.2 HSV Scheme

The clustering visualization is overlayed on top of density visualization. Using the Hue, Saturation, Value (HSV) color scheme, we prioritize the clustering visualization
as follows;

- Assign different Hues \( H \), \( H = \frac{2}{3} \) (blue) for friendly platforms and \( H = 0 \) (red) for hostile platforms.

- Assign different Saturations \( S \) to different clusters.

- Value \( (V, \text{pixel intensity}) \) at position \( p \), is determined by its density level.

Thus, by different \( V \) we can identify the boundary of platforms, and by different \( H \) we can differentiate between the opposing platforms, and finally by different \( S \) we can further differentiate the boundary of opposing platforms, see Figure 4.2-(a).

### 4.4 Motion Scripting

Battle simulation serves as a test bed of the visual tools that we have discussed so far. A particular problem here is how to achieve realism of the movement of entities engaged in the battle simulation. Possibly, scripting the motion of each entity by hand would be a most accurate way to simulate real battle. However, clearly scripting by hand is a time consuming job and it is very difficult to put interacting behavior among entities. Furthermore motion scripted data is hard to get and even classified.

In computer graphics, various animation techniques have been studied to tackle the autonomous movement problem. The basic philosophy underlying such techniques is to input the least amount of knowledge about the motion details into the “animation engine” and let the engine decide what is going to happen next in detail. By doing so, we can achieve fully automated and unpredictable motion as in real battles.
Figure 4.2. Battlefield Visualization

Figure (b) shows the visualization of “edge cutting” technique discussed in Section 3.3.2. The dark edge represents a disconnected edge in the Delaunay triangulation. Here the clustering computation is performed only for enemy (Red) forces.
In this section, we discuss two animation techniques; a \textit{behavioral animation} and a \textit{cognitive modeling} approach.

\subsection{Behavioral Animation}

Behavioral animation has been developed to simulate natural flocking behavior of animals. The Boids of Craig Reynolds’ is a good example of behavioral animation; [Rey87]. It uses a weighted linear combination of three parameters:

1. Separation: the tendency to avoid colliding with local flock mates.

2. Alignment: the tendency towards the average heading of local flock mates.

3. Cohesion: the tendency to move toward the average position of local flock mates.

It turns out that this simple model shows very nice flocking behavior, and has been adopted in various applications demanding realistic flocking behavior, for example in the movie, Lion King, where a group of bulls rush into a valley. With various combinations of weight values on the parameters, we can achieve different flocking behavior, from chaotic wandering to a very disciplined movement.

But still human flocking behavior including the military movement is a lot more intelligent than that of animals, especially when an interaction among flocks is required. However, one can achieve enough realism of human flocking behavior, in our case soldiers or tanks, by introducing two additional parameters:

4. Path Following: the tendency to follow a initially given path.

5. Hostility: the tendency to attack or retreat from a nearby enemy.
Figure 4.3. A Behavioral Animation Example

Friendly forces (Blue tanks) are under attack from enemy forces (Red tanks)
Path following is essential to simulate strategic military movement. The hostility factor can be implemented in various ways. We make an attack on a nearby enemy when our entity has more friendly entities than hostile ones within some given range $r$. Otherwise the entity retreats. By assigning different weights to all five parameters, we can make a battle scene totally brutal or almost peaceful, see Figure 4.4.

A straightforward implementation of the Boids requires $O(n^2)$ running time, mainly spent on neighborhood queries for each Boid. Once again, the Delaunay triangulation, followed by a DFS on it, can reduce the time to $O(n + k)$, where $k$ is the number of query reports$^1$; [DD90]. Moreover, the lazy Delaunay triangulation update technique discussed in Section 2.5 can be fully utilized here, because the motion coherently preserves proximity.

4.4.2 Cognitive Modeling Approach

The cognitive modeling (CM) approach is useful for more sophisticated motion requirement such as performing a certain task. This approach capitalizes on a rule-based AI language called cognitive modeling language, CML. By specifying “sketch plan” using CML and through reasoning, we can produce a motion sequence of entities satisfying the specification; [FTT99]. Thanks to the interval arithmetic technique, implementing the CM approach becomes practically plausible, and recently the approach has been applied to some applications such as advanced character animation and automated cinematography. Even though CM approach is more suitable for sim-

---

$^1k$ depends on the radius of the neighborhood. Typically, the radius choice and the structure of the flock are such that there is a constant number of neighbor for each member of the flock; i.e., $k$ is $O(n)$. 

Figure 4.4. Five Parameters in Boids

From the numbers, 1 to 5, each illustrates Separation, Alignment, Cohesion, Path following, and Hostility parameter for the blue unit in the middle. The arrow associated with each number represents the direction of acceleration.
ulating a complex environment such as battle, the scalability issue of the approach is still in doubt. The scalability is particularly crucial for a military simulation application, since the application easily demands that a great number of platforms should be involved.

4.5 Terrain Visualization

Traditionally most work in battle visualization applications has been devoted to an efficient way of visualizing terrain. Research-wise, the terrain visualization does not play a key role in our work. But we do provide a reasonably efficient way to render the terrain, simply because terrain serves as a geographical reference for the battlefield, hence is an important tool to enhance situation awareness. In order to do that, we use a static mesh reduction technique, vertex decimation, and LOD control as follows;

1. Initially, terrain elevation data is provided as a two dimensional array.

2. Triangulate the array.

3. Offline, apply iteratively vertex decimation to the triangulation and store some intermediate results until the total number of nodes in the triangulation is sufficiently small.

4. In real time rendering, the distance between a user’s viewpoint and terrain decides which LOD of triangulation will be chosen.
4.6 Zooming Issues

The human visual system experiences a processing delay when the attention changes from a local detail to a global view. Many interactive visualization applications try to solve this problem by providing a zoom facility to users. However, without carefully considering the low level human visual system, such an effort could induce unnecessary perceptual overhead or could hinder a correct understanding of the visual information.

Pizlo and et al presented a new model of visual size processing based on an algorithm called “exponential pyramids”, [Pizlo95]. In this hierarchical model, the hierarchy represents multiple levels of spatial resolution. The bottom layer in the hierarchy represents a level of full resolution, and the higher layers represent increasingly more coarse level of resolution than those on lower layers. Each cell in the hierarchy receives the visual information from a given portion of the “retina”, depending on the layer in which the cell is located.

This model explains particularly well the logarithmic relationship between time and size in size perception. By conducting a psychophysical experiment on the visual size processing of a human subject, they concluded that mental representations of objects can be compared if these representations involve the same relative resolution, otherwise the resolution of one representation must be transformed to the other representation until the relative representation becomes nearly the same. This causes the processing delay of a human subject in the size processing.

We provide the zooming interface into our application based on the above size
Figure 4.5. A Different Zooming Strategy

In Figure (a), the relative resolution with respect to the window size is variable. In Figure (b), the relative resolution is fixed.
processing model. The model suggests that we should maintain multiple hierarchies of the same view while keeping the relative resolution of each hierarchy same. By doing so, we are able to facilitate a rapid switch from the local view to the global view to a particular battle situation. In Figure 4.6, we illustrate two different ways of keeping the multi-resolution. Both approaches maintain the multi-resolution of a particular view. The first approach (Figure 4.6-a) keeps the window size the same regardless of the window resolution. The second approach (Figure 4.6-b) shrinks the window size proportionally to the window resolution. Therefore only the second approach keeps the relative resolution same, and we adopt it in our application.

In the context of our applications, the rapid switch from a local view to a global view is especially important for the lethality visualization, because a user (commander) needs to frequently switch his/her attention from a locally dangerous zone to the view of the entire battle in order to appropriately avoid a possible disaster.

In the lethality visualization, there are two ways to implement the multi-resolution model. One is to use the texture mapping. This is an easiest and fastest way to build up the multi-resolution model where the texture mapping hardware is supported. In this case, the implementation resembles the mipmap construction in the texture map. Once the full resolution view is rendered on the highest level of window, we progressively apply the texture mapping to a lower level of window. This reduction process is essentially a bilinear interpolation between the windows of different sizes.

The other way to implement the multi-resolution model is to construct a multi-resolution lethality model in the modeling stage. Since the lethality model has been
approximated by a polygonal model (Figure 3.4), we only need to subdivide the polygonal model more finely in order to get the finer resolution of model. At rendering time, depending on the resolution of each window to be rendered on, an appropriate resolution of the polygonal model is selected.

When incorporating the multi resolution view into our application, there could arise a user disorientation. When a user sees the same view with a different resolution, he or she could be confused about what portion he or she is looking at with respect to each different window, and therefore needs to do a search. This extra search work could cause yet another processing delay in the human visual system. One way to overcome this difficulty is to put a navigational frame in each window in order to indicate which portion is being investigated currently.
5. EXPERIMENT RESULT

5.1 Implementation Workbench

The experiment was carried out on 32 SGI 250 MHZ IP27 processors with 16G memory\(^1\). An Infinite Reality 2E graphics board was used for fast polygon rendering for visualization and geometry computation. No parallel computation capability was used during the computation. Most components of the program were coded in C++ and compiled by MIPSpro C++ compiler(CC).

In order to provide a convenient 3D interaction(GUI) to an end user and to easily manipulate 3D geometry objects including the Boid objects and the terrain object, we used the Open Inventor graphics library.

Last but not least, LEDA(Library of Efficient Data structures and Algorithms) version 3.8 was extensively used for various basic geometric data structures. In particular, the POINT SET data structure was modified to implement the Lazy update technique in Delaunay triangulation. LEDA was also used for the comparison of performances between Lazy update and two different Delaunay triangulation computation methods, namely the Divide and Conquer and Flipping algorithms.

\(^1\)All the experiments except for the one performed in section 5.4 was taken on this setting.
5.2 Dynamic Update on a Delaunay Triangulation

In section 2.5, we discussed various techniques to update a Delaunay triangulation when points move. As reported in [HKW+98], when all points move, a full recomputation from scratch beats successive and deletion technique, thus we did not consider a successive insertion and deletion technique in our experiments. We employed both Divide and Conquer and Flipping algorithms for the full recomputation method, and compared their performances to that of a Lazy update technique. The performance of the flipping algorithm has been measured also in order to see how much improvement the Lazy update technique can achieve. Note that the Lazy update technique is based on the flipping algorithm. We measured also the computation time of the retriangulation process in the Lazy update technique, since a retriangulation alone can give an approximation of an updated Delaunay triangulation. By measuring how many flipping operations are needed after retriangulation, we can understand how closely a retriangulation approximates an updated Delaunay triangulation. The experimental setting for the comparison is as follows;

Consider a square, 10 miles wide in each direction. Points are uniformly distributed within the square and assigned a randomly chosen speed from two groups of uniform distributions. We vary the number of points from 50 to 2000, and assign a speed range selecting from the two different speed groups, the slow motion group(maximum speed is 5 mph) and fast motion group(maximum speed is 45 mph)\(^2\) Then, each point keeps moving

\(^2\)We took the example of M1 Abrams tank whose maximum speed is 45 mph.
by following the Boid animation rules.

The experimental results are given in Table 5.1 and Figure 5.1. We have observed the following facts from the experiment:

1. With up to 400 points, Lazy update (LAZY) cuts the recomputation time of the Divide and Conquer (DD) approximately by half and of the Flipping algorithm (FLIP) by $\frac{1}{4}$. Update time is almost linear. As the number of points increases to more than 400 points, the improvement diminishes. When the number of points reaches 2000, there is almost no improvement.

2. Up to 800 points, the retriangulation time (RETRI) accounts for the almost half of the Lazy update time. In other words, Delaunay flipping time also takes half of the update time. Retriangulation time grows linearly. As the number of points increases, retriangulation time takes more than Delaunay flipping time in the process of Lazy update.

3. In Lazy update, the number of flipping operations (FLIP) needed after retriangulation is very small and much less than a worst case estimate would suggest. This becomes more apparent when points move slowly at 5 mph.

Observation 1 tells us that when the number of points is less than 400, Lazy update is a good technique to update a Delaunay triangulation, and it is two times faster than a pure recomputation. However, when the number of points is more than

\footnote{Note that, in worst case, the number of flipping operations needed to obtain a Delaunay triangulation from an arbitrary triangulation is $O(n^2)$.}
<table>
<thead>
<tr>
<th>NODES</th>
<th>EDGES</th>
<th>DD</th>
<th>FLIP</th>
<th>LAZY</th>
<th>RETRI</th>
<th>FLIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>44</td>
<td>0.00018</td>
<td>0.00021</td>
<td>0.00008</td>
<td>0.00004</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>276</td>
<td>0.00096</td>
<td>0.00160</td>
<td>0.00047</td>
<td>0.00020</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>570</td>
<td>0.00191</td>
<td>0.00351</td>
<td>0.00097</td>
<td>0.00043</td>
<td>0.03</td>
</tr>
<tr>
<td>200</td>
<td>1160</td>
<td>0.00391</td>
<td>0.00777</td>
<td>0.00212</td>
<td>0.00096</td>
<td>0.03</td>
</tr>
<tr>
<td>300</td>
<td>1760</td>
<td>0.00611</td>
<td>0.01256</td>
<td>0.00338</td>
<td>0.00162</td>
<td>0.15</td>
</tr>
<tr>
<td>400</td>
<td>2368</td>
<td>0.00836</td>
<td>0.01755</td>
<td>0.00475</td>
<td>0.00257</td>
<td>0.13</td>
</tr>
<tr>
<td>500</td>
<td>2961</td>
<td>0.01051</td>
<td>0.02266</td>
<td>0.00628</td>
<td>0.00328</td>
<td>0.36</td>
</tr>
<tr>
<td>600</td>
<td>3567</td>
<td>0.01284</td>
<td>0.02783</td>
<td>0.00794</td>
<td>0.00426</td>
<td>0.35</td>
</tr>
<tr>
<td>700</td>
<td>4160</td>
<td>0.01511</td>
<td>0.03382</td>
<td>0.00965</td>
<td>0.00537</td>
<td>0.62</td>
</tr>
<tr>
<td>800</td>
<td>4754</td>
<td>0.01717</td>
<td>0.03891</td>
<td>0.01165</td>
<td>0.00665</td>
<td>1.44</td>
</tr>
<tr>
<td>900</td>
<td>5350</td>
<td>0.02003</td>
<td>0.04546</td>
<td>0.01405</td>
<td>0.00799</td>
<td>1.31</td>
</tr>
<tr>
<td>1000</td>
<td>5957</td>
<td>0.02196</td>
<td>0.04988</td>
<td>0.01565</td>
<td>0.00949</td>
<td>2.23</td>
</tr>
<tr>
<td>1500</td>
<td>8953</td>
<td>0.03367</td>
<td>0.07942</td>
<td>0.02808</td>
<td>0.01873</td>
<td>4.26</td>
</tr>
<tr>
<td>2000</td>
<td>11953</td>
<td>0.04573</td>
<td>0.11408</td>
<td>0.04556</td>
<td>0.03315</td>
<td>9.13</td>
</tr>
</tbody>
</table>

a. When the maximum speed is 5 mph.

<table>
<thead>
<tr>
<th>NODES</th>
<th>EDGES</th>
<th>DD</th>
<th>FLIP</th>
<th>LAZY</th>
<th>RETRI</th>
<th>FLIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>42</td>
<td>0.00018</td>
<td>0.00020</td>
<td>0.00008</td>
<td>0.00004</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>272</td>
<td>0.00092</td>
<td>0.00155</td>
<td>0.00044</td>
<td>0.00020</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>568</td>
<td>0.00189</td>
<td>0.00338</td>
<td>0.00098</td>
<td>0.00042</td>
<td>0.02</td>
</tr>
<tr>
<td>200</td>
<td>1166</td>
<td>0.00403</td>
<td>0.00783</td>
<td>0.00213</td>
<td>0.00100</td>
<td>0.11</td>
</tr>
<tr>
<td>300</td>
<td>1760</td>
<td>0.00615</td>
<td>0.01252</td>
<td>0.00338</td>
<td>0.00161</td>
<td>0.24</td>
</tr>
<tr>
<td>400</td>
<td>2353</td>
<td>0.00842</td>
<td>0.01773</td>
<td>0.00476</td>
<td>0.00240</td>
<td>0.7</td>
</tr>
<tr>
<td>500</td>
<td>2964</td>
<td>0.01060</td>
<td>0.02256</td>
<td>0.00630</td>
<td>0.00326</td>
<td>1.43</td>
</tr>
<tr>
<td>600</td>
<td>3561</td>
<td>0.01291</td>
<td>0.02817</td>
<td>0.00801</td>
<td>0.00427</td>
<td>2.23</td>
</tr>
<tr>
<td>700</td>
<td>4160</td>
<td>0.01511</td>
<td>0.03298</td>
<td>0.00964</td>
<td>0.00546</td>
<td>3.3</td>
</tr>
<tr>
<td>800</td>
<td>4764</td>
<td>0.01722</td>
<td>0.03887</td>
<td>0.01164</td>
<td>0.00673</td>
<td>4.3</td>
</tr>
<tr>
<td>900</td>
<td>5356</td>
<td>0.01953</td>
<td>0.04439</td>
<td>0.01356</td>
<td>0.00811</td>
<td>5.45</td>
</tr>
<tr>
<td>1000</td>
<td>5961</td>
<td>0.02188</td>
<td>0.05009</td>
<td>0.01582</td>
<td>0.00951</td>
<td>8.35</td>
</tr>
<tr>
<td>1500</td>
<td>8956</td>
<td>0.03389</td>
<td>0.07983</td>
<td>0.02823</td>
<td>0.01943</td>
<td>26.28</td>
</tr>
<tr>
<td>2000</td>
<td>11960</td>
<td>0.04753</td>
<td>0.11205</td>
<td>0.04788</td>
<td>0.03366</td>
<td>48.26</td>
</tr>
</tbody>
</table>

b. When the maximum speed is 45 mph.

Table 5.1
Dynamic Update on a Delaunay Triangulation

NODES and EDGES denotes the number of nodes and edges in a Delaunay triangulation (DT). DD, FLIP and LAZY respectively denotes the time for updating a DT using Divide and Conquer, Flipping and Lazy update algorithm. RETRI denotes retriangulation time in Lazy update, and FLIPS denotes the number of flipping performed after retriangulation.
(a) Maximum speed is 5 \textit{mph} \hspace{1cm} (b) Maximum speed is 5 \textit{mph}

(c) Maximum speed is 45 \textit{mph} \hspace{1cm} (d) Maximum speed is 45 \textit{mph}

Figure 5.1. Update Time Comparison on a Delaunay Triangulation

In each plot, each line, from top to bottom, respectively denotes \textit{Flipping}, \textit{Divide and Conquer}, \textit{Lazy update} and \textit{retriangulation time}.
2000, one might as well recompute a new Delaunay triangulation from scratch. A possible explanation for this is as follows. The retriangulation process capitalizes on a segment walk for face location, and a segment walk takes $O(\sqrt{n})$ time. Observation 2 says that retriangulation becomes a dominant factor in Lazy update as the number of points grows. Probably, 2000 is the number of points where the $O(\sqrt{n})$ factor becomes dominant in the computation. According to Basch and et al [BDIZ97], in a similar probabilistic setting like ours, a Voronoi diagram undergoes $\Theta(n^{3/2})$ combinatorial changes. It seems that such combinatorial changes introduce more walks in the retriangulation as $n$ increases.

Observation 3 suggests that a retriangulation without flipping from an old Delaunay triangulation is a very good approximator for a new Delaunay triangulation, especially when the points move slowly. So, when the number of points is less than 400, we can update a Delaunay triangulation four times faster than doing a pure recomputation. At up to 1000 points, we can still update a Delaunay triangulation at least two times faster.

5.3 Various Computational Result

The performance results of a typical simulation is given in Table 5.2. Each Boid character, in this simulation a tank, is composed of 360 polygons, and the terrain data is composed of 80 $K$ polygons. The raster window size used in density computation and lethality assessment was $400 \times 400$. Particularly in the case of density computation, since each local influence region is identical, after one of the local influence regions has been rendered once onto an offscreen buffer, its raster copy is used for the
remaining regions instead of re-rendering them each time. However, this cannot be applied to lethality assessment because the threatening regions are not identical.

5.4 Approximating Delaunay Triangulation

In some applications, an approximate Delaunay triangulation is sufficient. One way to approximate the Delaunay triangulation is the lazy update as explained in section 2.5. However, in Army applications it is often the case that a fraction of units does not move. Hence, we can keep the same triangulation for a number of steps. This has motivated our investigation of keeping the same triangulation for a number of steps. That is, we pretend that even though some of the points may have moved appreciably, the old triangulation is still a valid Delaunay triangulation.

In this experiment we use a different motion scenario than the one used in the other experiments, that used Boids movement. The reason is that we initiated this experiment in order to compare the performance of KDS with that of approximating Delaunay triangulation. In the case of KDS, the trajectory must be represented as a continuous functional form. Nonetheless, there is probably no characteristic motion scenario in military applications, that is, there is probably no such thing as an average set motion sequences in a combat situation. In view of this situation, we assume that all points move at random, on random trajectories, with an average but random speed. The motion scenario in the experiment is as follows:

A set of 100 points moves inside a 2D rectangle of size 1.33 by 1, at speeds that are on average 0.05 units/sec. The maximum speed is 0.1, and
minimum speed is zero. At random intervals, a randomly chosen point alters its course. When a point reaches the boundary of the rectangle, it alters course by reflection so as to remain within the rectangle.

We keep track of all pairs that are within some distance threshold from each other, calling such a pair threshold pair. The threshold distance is 0.14, and it is assumed to be a threat distance for hostile platforms. We compute such a threshold pair both on a correct Delaunay triangulation and on an approximate Delaunay triangulation. The approximate triangulation is constructed by keeping the old triangulation. Then, we ask how many the deductions, based on the assumed triangulation, differ from those that would be made from the true Delaunay triangulation. The measurement was taken on an SGI R10000 with 192MB main memory, and Table 5.3 gives some insight into the above question.

The mean time for constructing the Delaunay triangulation from scratch has been 0.0077 sec using the LEDA library. Table 5.3 shows that in this experiment the effect of using the incorrect triangulation is small for 5 time steps. In some cases, keeping the old triangulation for 10 steps may be acceptable, but after that the triangulation should be updated. Therefore, holding over the triangulation for up to 5 time steps may incur an acceptable safety risk for those who cannot afford to update the triangulation at every time step.
<table>
<thead>
<tr>
<th>NUM</th>
<th>POLY</th>
<th>FULLREND</th>
<th>BOXREND</th>
<th>DEN</th>
<th>LETHAL</th>
<th>CLUSTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>106</td>
<td>12.4</td>
<td>31.0</td>
<td>0.0500</td>
<td>0.0264</td>
<td>0.0060</td>
</tr>
<tr>
<td>200</td>
<td>132</td>
<td>7.4</td>
<td>15.2</td>
<td>0.0860</td>
<td>0.0847</td>
<td>0.0123</td>
</tr>
<tr>
<td>300</td>
<td>158</td>
<td>5.1</td>
<td>10.0</td>
<td>0.1218</td>
<td>0.1436</td>
<td>0.0190</td>
</tr>
<tr>
<td>400</td>
<td>184</td>
<td>4.0</td>
<td>7.3</td>
<td>0.1576</td>
<td>0.2014</td>
<td>0.0256</td>
</tr>
</tbody>
</table>

Table 5.2
Simulation Performance Result

NUM denotes the number of Boid platforms involved in the simulation. POLY denotes the number ($\times 10^3$) of polygons to be rendered in each frame. FULLREND denotes the time (frames/sec) to render a full scene including Boids and terrain in the simulation. BOXREND denotes the time (frames/sec) to render a scene in a bounding box mode. DEN denotes the time (sec) to compute density distribution. LETHAL denotes the time (sec) to assess lethality. CLUSTER denotes the time (sec) to compute clustering.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6.10</td>
<td>0.04</td>
<td>0.3153</td>
</tr>
<tr>
<td>10</td>
<td>6.1414</td>
<td>0.4738</td>
<td>0.7243</td>
</tr>
<tr>
<td>15</td>
<td>6.3</td>
<td>1.38</td>
<td>1.7278</td>
</tr>
<tr>
<td>20</td>
<td>6.46</td>
<td>2.63</td>
<td>2.4604</td>
</tr>
</tbody>
</table>

Table 5.3
Experiment on Approximating Delaunay Triangulation

The A-column shows the number of time steps for which the triangulation is kept the same topologically. The B-column indicates the number of pairs, deduced from the deformed triangulation, that are within threat distance, 0.14. The C-column indicates the mean number of errors between the threshold pairs of the kept triangulation and those of the correct Delaunay triangulation. The D-column shows the standard deviation.
6. SUMMARY

6.1 Conclusion

We have presented three visual tools (density, clustering, and lethality assessment) to enhance the situation awareness of battlefield for a military commander. The unique feature of this work is to extend the functionality of the battlefield visualization package, which has been traditionally devoted to terrain visualization, to providing the visual abstractions of a battle.

By capitalizing on the perceptual psychology aspect in the low level human visual system, we effectively incorporated the visual tools into the battlefield visualization package. Gestalt perception and preattentive features of the human visual system were the key aspects considered in the development of the tools. The theory of gestalt perception helps understanding the global situation of battlefield. The preattentive features such as intensity, hue, saturation further assist fast search and quick boundary detection used by the low level human visual system.

Computationally, our visual tools need efficient geometry algorithms. Primarily, the geometry computation is related to a proximity problem. Moreover, the computation needs a dynamic update since platforms in the application are assumed to change their position at all times. Therefore, the main computational contribution
from this dissertation is that we have presented various solutions from dynamic computational geometry to effectively solve the problem, namely Delaunay triangulation based approach, image based approach, hybrid approach, and KDS approach.

Each approach has its own strength depending on the restriction either given by the nature of problem itself or given by the computing platform, However, we have found that the Delaunay triangulation based approach is the most flexible approach among them. Coincidentally we also explored various techniques to dynamically update the Delaunay triangulation, namely successive insertion and deletion, one-time update, and lazy update. We have experimentally shown that the lazy update technique is a most efficient way to handle the dynamic update, especially when there exist sufficient coherence in the movement. This idea is extended further, rather radically, to the point where no update is performed within some period of time. We also showed experimentally that the approach is reasonably acceptable for those who cannot afford the expensive computation all the time, while taking the risk.

Once the visual tools have been computed, they should be visualized, which is the main goal of the battlefield visualization. Exploiting the aforementioned perceptual psychology aspects, we rendered the visual tools in a computationally efficient and perceptually meaningful way. The gestalt perception rule played an important role in density rendering, and the preattentive features and their hierarchy in the human visual system were exploited both in clustering and in density rendering.

We developed a simulation model for autonomous battle units using Boids animation. It served to check the validity of the dynamic geometric computation and
visual tools computation. We made a slight modification to the original Boids to simulate a realistic battle scenario. Finally, a simple terrain model with a level of detail was incorporated into our application. The different levels of detail were built by progressive vertex decimation.

6.2 Future Work

Most of the techniques and visual tools in this thesis has been developed in two dimension. The two dimensional view ultimately assists to understand a three dimensional combat view in the application. In some situations, such two dimensional approach is sufficient or even more suitable than the three dimension counterpart. Nevertheless, the extension to three dimensions is inevitable, simply because the human visual system perceives the world in three dimension. This should offer more opportunities to exploit in preattentive rendering and in density critical attributes.

Expanding into three dimensional space, spatial Delaunay triangulations\(^1\) would be needed. Most of the techniques explored throughout the thesis can be directly applied to the computation in a three dimensional Delaunay triangulation, since the techniques are independent of the ambient dimension. Unfortunately, however, the construction of Delaunay triangulation in three dimension becomes more difficult. It is known that an optimal computation time of Delaunay triangulation in three dimension takes \(O(n^2)\) using a randomized incremental approach; [Müc95]. A particular problem arising in a three dimensional Delaunay triangulation is that it is very

\(^1\)For the better readability, we use “triangulation” instead of “tetrahedron”.
difficult to devise a dynamic algorithm, especially for a deletion operation \textsuperscript{2}. It is mainly because the locality property in three dimensional Delaunay triangulation is more complicated \textit{per se} than the two dimensional counterpart; the locality is still valid in three dimensional Delaunay triangulation, but its usage must be carefully “choreographed”; [Tei99]. Accordingly, update techniques for the two dimensional Delaunay triangulation discussed in section 2.5 cannot be easily extended to three dimension. An efficient deletion operation must be investigated first before designing an efficient update of a three dimensional Delaunay triangulation.

From the perceptual psychology point of view, a three dimensional extension to our application means that we need to consider three dimensional preattentive features. Some recent works on oceanographic and atmospheric visualization has successfully incorporated such three dimensional preattentive features as height and texture, in what they call “pexel(perceptual texture element)”, in their three dimensional visualization application; [Hea99]. However, there could arise another question as to whether or not three dimensional features might interfere with existing two dimensional features and so limit comprehending all features present.

Although we dealt with the issues of autonomous unit movement and terrain visualization in our application, they both leave much room for independent research. The autonomous agent (unit) generation is particularly interesting to the military community, and some important work has been done as a part of Modular Semi-Automated Forces (ModSAF). Also the video game industry has come up with many

\textsuperscript{2}At this moment, we are not aware of any reported work on the deletion operation from a three dimensional Delaunay triangulation.
practically feasible solutions using techniques similar to those explained in 4.4.
LIST OF REFERENCES
LIST OF REFERENCES


VITA
VITA

Young Jun Kim was born and raised in Busan, Korea. He earned his B.S. and M.S. in computer science and statistics department at Seoul National University. He earned his Ph.D. in computer science department at Purdue University. His research interest includes computer graphics, visualization, computational geometry, computer aided geometric modeling, and electronic entertainment.