Dynamic Proximity Calculations for Situation Awareness*

Young J. Kim
Department of Computer Science
University of North Carolina in Chapel Hill
youngkim@cs.unc.edu

Christoph M. Hoffmann
Department of Computer Sceicne
Purdue University
cmh@cs.purdue.edu

November 6, 2001

We apply dynamic proximity calculations (density and clustering) from *dynamic computational geometry* to a military application. The derived proximity information serves as an abstract view of a current situation in the battlefield that can help a military commander to achieve *situation awareness*. We employ Delaunay triangulation as a computational tool in our framework, and study its dynamic update in depth.

1 Introduction

For the past two decades or so, computational geometry (CG) has had a significant impact on algorithmic and theoretical development in geometric computing and its many applications. However, it has been argued until recently that the practical aspects of CG have been neglected considerably. The CG Impact Task Force Report [C+96] published in 1996 is a notable voice documenting unbalanced achievements during the development of CG. The report also recognizes limited success stories in some application areas; for instance, in computer graphics, in shape reconstruction, and in robotics.

*Work supported in part by ARO Contract 39136-MA and by NSF Grant CCR 99-02025

Nevertheless, even these areas still do not exploit fully the best and recent development of CG. Coding complexities and robustness issues are cited in the report as two main reasons why the full utilization of CG remains hampered in applications.

In this paper, we work to expand the domain of applied CG to a military application. Much prior work in the military domain as well as in some civilian applications indicates that providing *situation awareness* to a commander (or user) in a correct and timely fashion is key to success in accomplishing various tasks. These tasks include training [KHC90], tactical engagement [RJJ+94], maneuvering an aircraft [fHC00], battlefield analysis [SBS93], and more. We abstract a battle situation by means of proximity calculations, a well studied problem in CG. Density and clustering computation are typical examples of the proximity calculations that we relied on for the abstraction. This abstraction is eventually delivered to a battle commander in such a way that it helps the commander to increase the situation awareness on the battlefield.

Like other applications, our military application entails application-specific constraints. In particular, platforms engaged in battle are assumed to be constantly moving. This means that one needs to dynamically update the abstraction, in our case the proximity calculations, in order to correctly reflect a changing situation in the battlefield. In CG, this kind of study on the dynamic update of underlying geometric entities is known as *dynamic computational geometry*. It is generally understood that both in theory and in practice, "dynamization" of any geometric data structure or computation is much more difficult than its static counterpart. Especially in practice, constant running time factors of the dynamic algorithm and the coding complexity make time performance poor. However, running time performance is very critical for time demanding applications such as a military application. In order to meet such time-critical requirements, we explore various efficient techniques from dynamic CG. In particular, Delaunay triangulation and its dynamic update have been extensively investigated in this paper.

1.1 Main Results

The major results of this paper include the following.

- Inspired by Gestalt perceptual processing, we conceive density of platforms as abstraction of the concentration of forces in the battlefield. Also we provide an efficient algorithm to compute the density.

- We present a new clustering definition inspired by the military formation rules, and consider this as abstraction of the battle formation of adversary forces. We present an efficient linear time algorithm for this problem.

- Computationally our approach to proximity calculation requires fast algorithms and flexible data structures. We employ Delaunay triangulation as underlying computational framework, and explore various techniques, such as successive insertion and deletion, one-time update, and lazy update, to maintain the triangulation dynamically.

- We present a novel approach to approximate the Delaunay triangulation when one cannot always afford the computational cost of updating the triangulation. We report experiments that elucidate how effective approximation is for different motion scenarios.

1.2 Organization

The paper is organized as follows. We review some preliminary notions and theories relevant in our application and survey related previous work in section 2. Section 3 describes the definition and computational methods for density and clustering. Section 4 is mainly devoted to various update techniques for Delauany triangulation. Section 5 shows various experimental results of our computation. Finally we summarize our work and discuss future research directions in section 6.

2 Preliminaries

In this section, we explain about notions and theories of situation awareness, and also survey related prior work.

2.1 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 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 a military viewpoint, Blanchard [Bla96] depicts battle space awareness as "knowing what is needed to win with the minimum number of casualties". He identifies situation awareness as one of seven core concepts in the battle space awareness, and refers to it as the situations of friendly and enemy forces. Thus, we can conclude that helping a commander to develop situation awareness is an important factor to accomplish the mission.

2.2 Prior Work

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]. Most of them are also incorporated into an immersive display technology or a virtual reality facility. Extensive references to situation awareness in the avionics applications can be found in [RC00].

TSAS(Tactile Situation Awareness System) is a system to utilize human 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 system helps a pilot to avoid three dimensional disorientation during the maneuver of a 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 Naval Research Laboratory's Virtual Reality Responsive Workbench (VRRWB) 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 MÄK technologies provides a 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 concentrations by considering concentration values as vertical height in three dimensional space. Depending on the viewpoint, concentration is shown by height or by color intensity. Therefore, the height serves as a redundant encoding of concentration. Another functionality in the system is temporally condensing a potentially lengthy battle into an 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.

2.3 Applying the Theory of Perception

Usually the battlefield is represented by large datasets with various attributes. These large datasets make 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 of 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, 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) and boundary detection (clustering) from given large positional dataset.

3 Proximity Calculations

One of the 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.

3.1 Modeling the Battlefield

We model the battlefield as a 3D virtual world populated with two opposing groups of entities which are moving 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 negligible compared to the range of horizontal values in the 2D position.

3.2 Density Computation

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 4.1. Once the Delaunay triangulation has been computed, density assignment can be done using Algorithm 3.1,

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.

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.

ALGORITHM 3.1: ComputeDensityUsingDT

3.2.1 Density Visualization

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 then render the *polygonal terrain* using Gouroud shading.

Another alternative to the visualization of density is rendering by *blobby shading*; [Bli82]. Simplifying, the density distribution is obtained by summing the contribution from each atom separately:

3.3 Clustering

A military commander would be interested in formation information of adversary forces in order to appropriately react to a potential attack from the enemy or to counter-attack the enemy. However, such formation information about the opposite side is not known in general. Therefore, the commander should infer the enemy formation from the available data, in our case positional data. For instance, a database file with useful attributes such as enemy position, date, and weapon type can be furnished by a spy satellite in a track file format. 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 Techniques

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; [GJ79, CCFM97]. Moreover, for Euclidean space, it is NP-hard to approximate, to within a factor 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 to minimize 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 $1 + O(\frac{1}{m})$ from the minimum clustering cost with high probability in $O(m^2 n)$ 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 appropriate case for military unit formation: Each entity in a military unit is moving while maintaining some distance from others, see Figure 1.

We can define this type of clustering formally as follows:

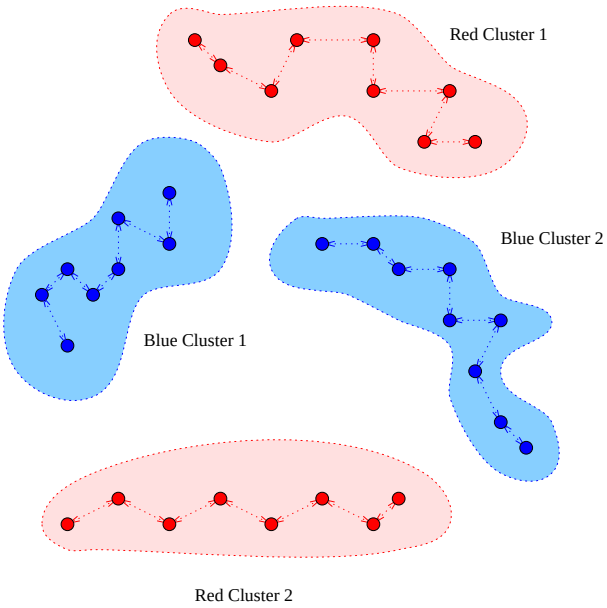


Figure 1: A typical example of clustering in the application
Each cluster has its own threshold value defining who is a member, namely a minimum distance between neighboring members. The dotted arrow between two points denotes such a minimum distance within a cluster.

DEFINITION 3.1 *Given n points in R^d and distance threshold r, find clusters $S_1, S_2, ..., S_k$ which satisfies the following*

1. $\forall x_i \in S_l, \exists x_j \in S_l$, *such that* $|x_i - x_j| \leq r$.
2. $\forall x_i \in S_l, \forall x_j \in S_m$ *where* $l \neq m$, *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 4.1. We can compute clusters as in Algorithm 3.2.

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 the clusters. This task involves two subtasks: Dynamic update of the Delaunay triangulation and of the connected components. The first subtask can be accomplished as explained in Section 4.2.1. The second subtask can be done by maintaining a spanning forest in a dynamic setting. This kind of problem is better known as a *dynamic*

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.

ALGORITHM 3.2: ComputeClustering

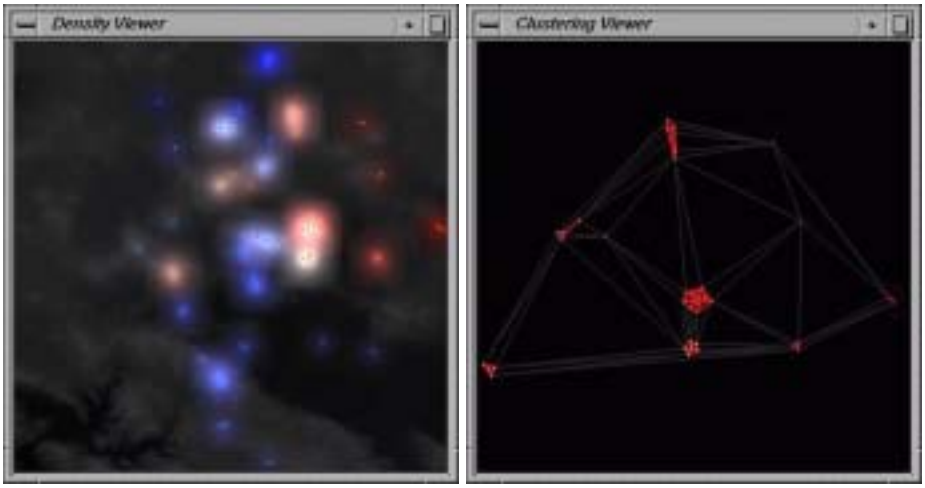
connectivity problem, since, if we have a spanning forest of a graph G, we can quickly answer whether two vertices in G are connected or not. There are three different known algorithms related to the dynamic connectivity problem; Fredrickson's topology tree [Fre97], sparsification [EGIN92], and Henzinger 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.

3.3.3 Clustering Visualization

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, 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



(a) Density and Clustering Visualization

(b) Underlying Delaunay Triangulation

Figure 2: Visualization Result

Figure (b) shows the visualization of the "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.

finally by different S we can further differentiate the boundary of opposing platforms, see Figure 2-(a).

4 Computational Tools

In this section, we explain our computational framework based on Delaunay triangulation to proximity calculation. We also address various issues related to updating a Delaunay triangulation dynamically.

4.1 Various Approaches to Proximity Computation

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

When it comes to dealing 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 partitions 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 with other techniques.

4.1.1 Dynamic Computational Geometry

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.

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. 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, one can have two possible scenarios of the movement; discrete and unpredictable motion or continuous and predictable motion[1].

This kind of study to investigate the dynamic update of a geometric structure is known as *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.

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.

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

[1]Without confusing the reader, from now on we refer discrete motion to discrete and unpredictable motion, and continuous motion to continuous and predictable motion. We also refer to a *kinetic* setting to describe the nature of continuous motion.

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 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.

4.2 Delaunay Triangulation Based Approach

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 4.1 *[DD90] Let S be a set of distinct points on a plane, δ a distance, 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 $o_1, o_2, ..., o_m$ 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. $|p, o_1| \leq \delta, |o_m, q| \leq \delta$, *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.

Besides the in-circle property, a Delaunay triangulation approximates the complete Euclidean graph within a ratio of less than 2.42 as the following theorem shows.

THEOREM 4.1 *[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 Delauany triangulation of S. Then,*

$$\frac{DT(p,q)}{d(p,q)} \leq 2.42 \tag{1}$$

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.

4.2.1 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. When we are not able to get a future position report of points and a 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 position 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.

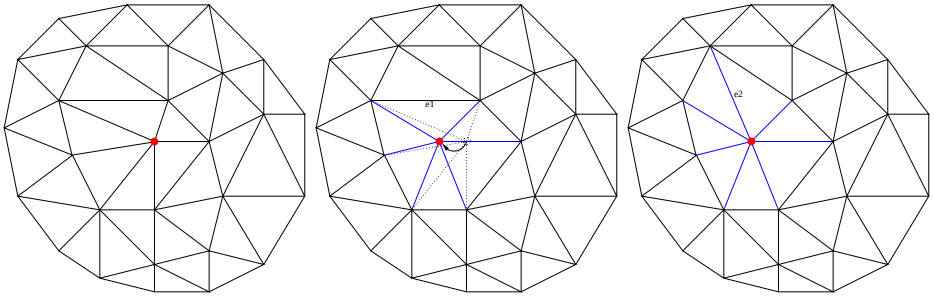
Successive Deletion and Insertion When a point has moved, we simply delete the point at the old position from the Delaunay triangulation, and re-insert it in the new position. The best insertion and deletion algorithms use randomization techniques, where insertion takes $O(\log n)$ expected time and 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].

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.

Lazy Update This approach also exploits spatial coherence. When points have moved only slightly or move while forming a group or flock, such as 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.*



(a) Before Update (b) Step 1: Retriangulation (c) Step 2: Flipping

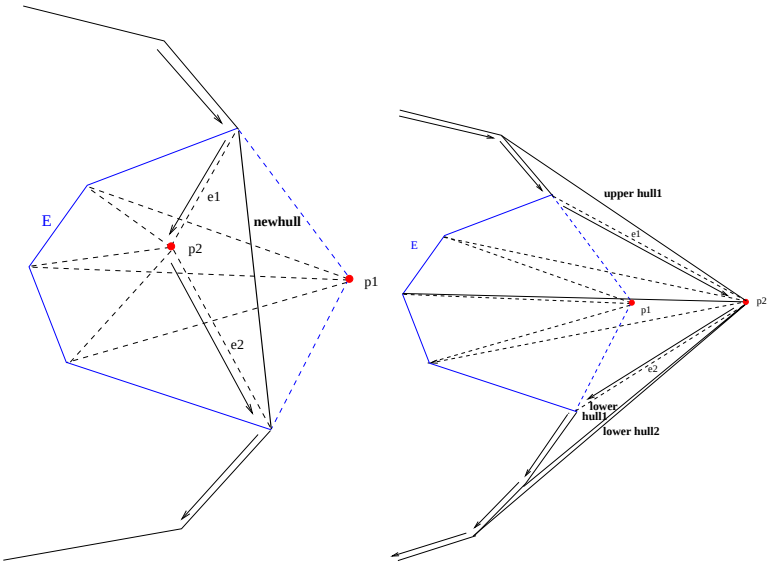
Figure 3: Two Steps in Lazy update on Delaunay triangulation
In case of a spatial coherence, most of time the retriangulation step merely reduces to be a triangulation check, see Figure (b). Furthermore, many edges are already flipped; In Figure (b) and (c), only edge $e1$ is flipped into $e2$.

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 1.

The retriangulation process is done by successive deletion and insertion into the old triangulation. The deletion process exploits the "star-shaped[2] 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 degree of the node until it is lowered to three, and finally we can delete the node. The *POINT SET* class

[2]When every vertex in a face f is visible from a vertex v in f, f is called "star-shaped".



(a) Case 1: Shrinking Hull (b) Case 2: Expanding Hull

Figure 4: Convex hull reestablishment when a point $p1$ moves to $p2$
If $e1$ and $e2$ forms a left turn(Case 1), a new hull edge *newhull* is created. Else if $e1$(upper hull) and its predecessor or $e2$(lower hull) and its successor forms a left turn(Case 2), new hull edges $upperhull1, ..., lowerhull1, ...$ should be created consecutively until the upper hulls and lower hulls form a right turn with their predecessors and successors respectively.

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 1 to know how closely the retriangulation can approximate the new Delaunay triangulation.

Lazy update technique is summarized in Algorithm 4.1.

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. *if* SimpleRetriangulate(p_1, p_2, DT) is *false*
3. Let f_1 be the face where p_1 is located in DT.
   Using *star-shaped property*, delete p_1 from f_1.
4. Walk from f_1 toward a face f_2 to which p_2 belongs in DT, and retriangulate f_2 by connecting p_2 with the three vertices of f_2.
5. Perform *Delaunay flipping* for all edges in DT. Omit this step if an approximation of DT was requested.

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. *if* p_2 is contained in the left half space of e, *then* continue.
4. *else* return $false$.
5. *if* there are convex hull edges in E, say e_1 and e_2 in a clockwise order,
6. *if* e_1 and e_2 forms a left turn, *then* return *true*.
7. *elseif* p_2 is in one of the cases in Figure 4, *then* appropriately reestablish the convex hull property as shown.
8. *else* return *false*.

ALGORITHM 4.1: UpdateDelaunayTriangulation

4.3 Kinetic Data Structures Approach

4.3.1 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 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.

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 complexity bounds in KDS assuming the trajectories are linear.

4.3.2 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].

4.3.3 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 the KDS algorithm for the

Delaunay triangulation does 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; [BGH97, AGMR98]. Our application demands frequent interaction among entities, and it requires for all the interacting entities an update of their trajectory plans. Eventually this 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 has been developed. The improvement is based on interval arithmetic to reduce the 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 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[3].

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 movement, we can formulate a plausible trajectory of the movement and use a similar technique to KDS.

4.4 Image Based Approach

When fast graphics hardware supporting, e.g. OpenGL is available which is common in modern graphics workstation, one can exploit the hardware to perform various proximity calculations.

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 proximity function from each grid position or pixel to neighboring pixels depending on a target proximity function. This local proximity function defines an area of influence around each pixel position containing at least one entity. For example, for the density compu-

[3]Basch 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 $\Theta(n^{\frac{3}{2}})$ 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 the worst case, the tightest upper bound on the number of combinatorial changes in Delaunay triangulation is known to be roughly cubic.

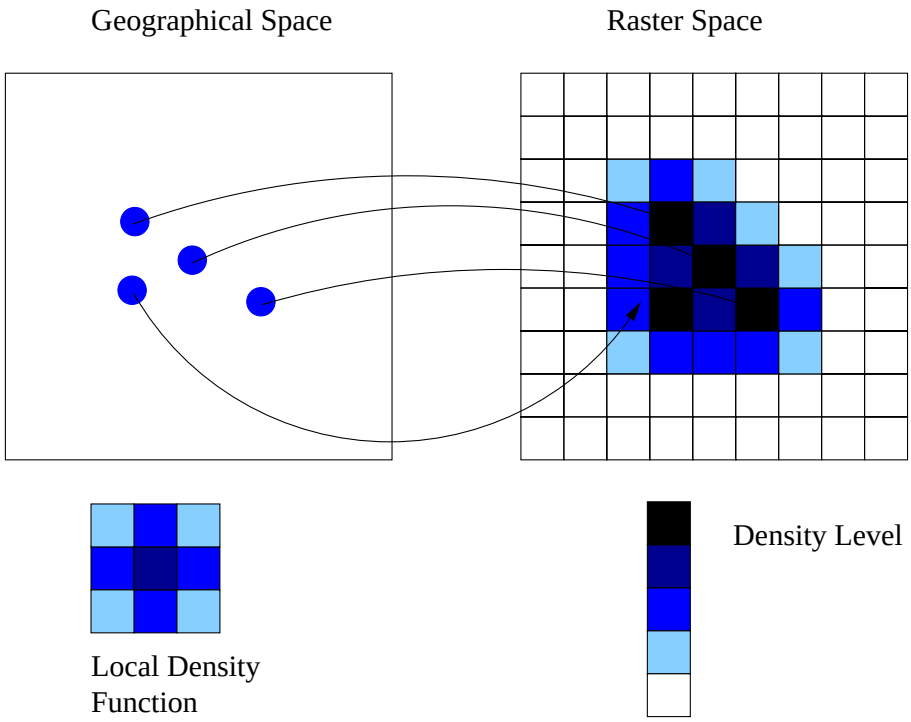


Figure 5: Image based approach to the density computation
Based on the mapping from geographical space to raster space, the density is accumulated by superimposing the local density function.

tation, we construct a local proximity function as in Figure 5.

For each entity, we sequentially compute the appropriate pixel position and then cumulatively applying the local proximity 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. When graphics hardware is available, this accumulation process can be accelerated by the use of fast alpha blending capability in the hardware.

The image based approach is attractive because it is very easy to implement. Also, 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. Moreover, when one wants to render the result of the proximity computations, the rendering process corresponds to a simple dumping operation from the raster buffer to the

ComputeDensityUsingAlpha(P)

Input A set P of n points in the plane.

Output A buffer B containing density assignment of P.

1. Define the local density function either in an explicit or implicit form.
2. *Polygonalize* it.
3. Set the source and destination alpha to one in the alpha blending function[4]; i.e. $DestinationColor = SourceColor \cdot 1.0 + TargetColor \cdot 1.0$.
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.

ALGORITHM 4.2: ComputeDensityUsingAlpha

frame buffer residing in a graphics display.

4.5 Hybrid Approach

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

One can take advantage of graphics hardware to construct a highly flexible local proximity function. The technique is similar to Manocha 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.

The main idea is that we approximate the proximity function by polygonalizing it within a limit of user defined error, and render it in the depth buffer (e.g. Z buffer) on the fly. Once we have a footprint of this local proximity function, we compute the overall proximity by iteratively blending it together as in the image based approach. In case of the density computation, Algorithm 4.2 summarizes the process.

5 EXPERIMENT RESULT

5.1 Implementation Workbench

The experiment was carried out on 32 SGI 250 MHZ IP27 processors with 16G memory[5]. 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.

5.2 Dynamic Update on a Delaunay Triangulation

In section 4.2.1, 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 Delauany triangulation. By measuring how many flipping operations are needed after retriangulation, we can understand empirically how closely a retriangulation approximates an updated Delaunay triangulation. The experimental setting

[5]All the experiments except for the one performed in section 5.4 were taken on this 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 selection from two different speed groups, a slow motion group (maximum speed is 5 mph) and a fast motion group (maximum speed is 45 mph)[6] Then, each point keeps moving by following the Boid animation rules.

The experimental results are given in Table 1 and Figure 6. 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(DC) 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 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[7]. 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 maintain a Delaunay triangulation, and it is two times faster than a pure recomputation. However, when the number of points is more than 2000, one might as well recompute a new Delaunay triangulation from scratch. We offer a possible explanation for this. The retriangulation process capitalizes on a segment walk for face location, and a

[6] We took the example of M1 Abrams tank whose maximum speed is 45 mph.

[7] Note that, in worst case, the number of flipping operations needed to obtain a Delaunay triangulation from an arbitrary triangulation is $O(n^2)$.

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. At 2000 of points the $O(\sqrt{n})$ factor has become dominant in the computation. According to Basch et al. [BDIZ97], in a similar probabilistic setting like ours, a Voronoi diagram undergoes $\Theta(n^{\frac{3}{2}})$ combinatorial changes. It seems that the 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 approximation 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 density and clustering computation are given in Table 2. We used the hybrid approach to compute density distribution. The raster window size used in density computation was 400×400. Particularly in the case of density computation, 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. This is possible because each local influence region is 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 4.2.1. 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. 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. Nevertheless, 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 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 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.

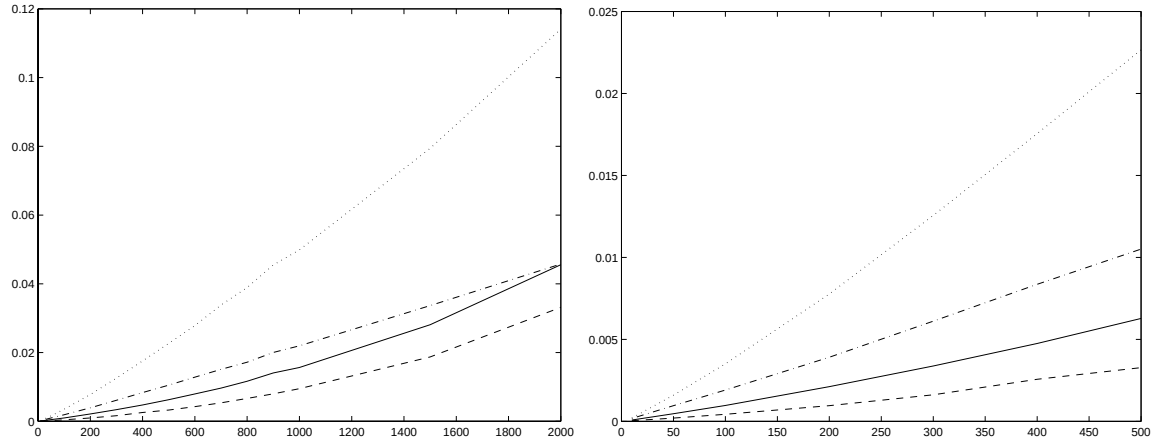
NODES	EDGES	DC	FLIP	LAZY	RETRI	FLIPS
10	44	0.00018	0.00021	0.00008	0.00004	0
50	276	0.00096	0.00160	0.00047	0.00020	0
100	570	0.00191	0.00351	0.00097	0.00043	0.03
200	1160	0.00391	0.00777	0.00212	0.00096	0.03
300	1760	0.00611	0.01256	0.00338	0.00162	0.15
400	2368	0.00836	0.01755	0.00475	0.00257	0.13
500	2961	0.01051	0.02266	0.00628	0.00328	0.36
600	3567	0.01284	0.02783	0.00794	0.00426	0.35
700	4160	0.01511	0.03382	0.00965	0.00537	0.62
800	4754	0.01717	0.03891	0.01165	0.00665	1.44
900	5350	0.02003	0.04546	0.01405	0.00799	1.31
1000	5957	0.02196	0.04988	0.01565	0.00949	2.23
1500	8953	0.03367	0.07942	0.02808	0.01873	4.26
2000	11953	0.04575	0.11408	0.04556	0.03315	9.13

a. When the maximum speed is 5 *mph*.

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

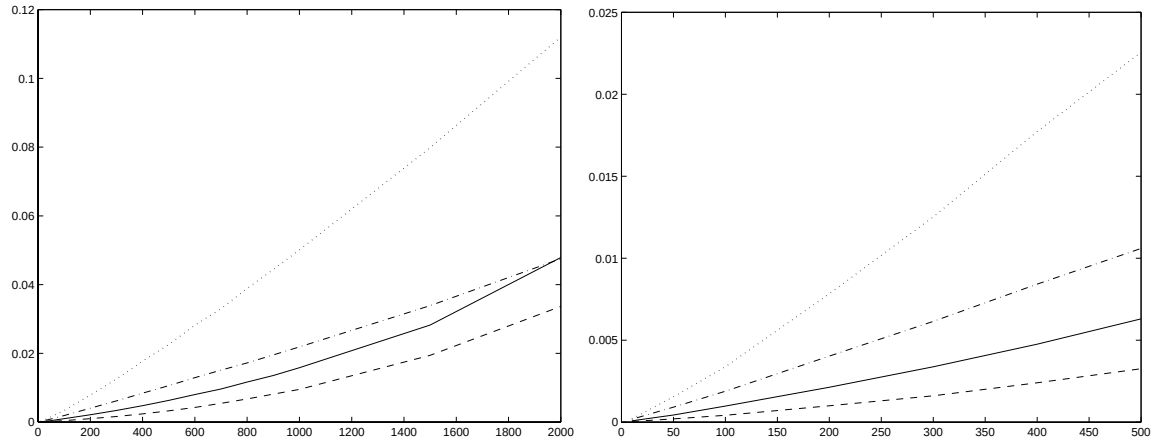
b. When the maximum speed is 45 *mph*.

Table 1: Dynamic Update on a Delaunay Triangulation
NODES and EDGES denote the number of nodes and edges in a Delaunay triangulation (DT). DC, FLIP and LAZY respectively denote the time for updating a DT using *Divide and Conquer*, *Flipping* and *Lazy Update* algorithms. RETRI denotes retriangulation time in Lazy Update, and FLIPS denotes the number of flipping operations performed after retriangulation.



(a) Maximum speed is 5 *mph*

(b) Maximum speed is 5 *mph*



(c) Maximum speed is 45 *mph*

(d) Maximum speed is 45 *mph*

Figure 6: Update Time Comparison on a Delaunay Triangulation
In each plot, the dotted line, the dot-dashed line, the solid line and the dashed line respectively denote *Flipping*, *Divide and Conquer*, *Lazy Update* and retriangulation time.

NUM	DEN	CLUSTER
100	0.0500	0.0060
200	0.0860	0.0123
300	0.1218	0.0190
400	0.1576	0.0256

Table 2: Simulation Performance Result
NUM denotes the number of platforms involved in the simulation. DEN denotes the time(sec) to compute density distribution. CLUSTER denotes the time(sec) to compute clustering.

A	B	C	D
5	6.10	0.04	0.3153
10	6.1414	0.4738	0.7243
15	6.3	1.38	1.7278
20	6.46	2.63	2.4604

Table 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 the threat distance of 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 Reported Work

We have presented proximity calculations (density and clustering) to enhance situation awareness a military commander would have of the battlefield. This work extends the functionality of battlefield visualization packages which have been traditionally devoted to terrain visualization. By capitalizing on perceptual psychology aspects in the low level human visual system, we rendered the result of proximity calculations with cognitive efficiency.

Computationally, our approach to the proximity calculations needs fast geometry algorithms. In particular, dynamic updates are needed since platforms in the application are assumed to change their position frequently. We have extensively explored various solutions from dynamic computational geometry to effectively solve the problem, namely a Delaunay triangulation based approach, an image based approach, a hybrid approach, and a KDS approach. Each approach has its own strengths depending on restrictions 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.

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 dynamic updates, especially when there exist sufficient coherence in the movement. This idea is extended by exploiting what happens when no update is performed within some period of time. We showed experimentally that two unconventional approaches can be acceptable for those who cannot afford more expensive computation all the time.

6.2 Future Work

Most of the techniques in this paper have been developed in two dimension. The two dimensional view ultimately helps understand a three dimensional combat view. In some situations, a 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 and battles take place in three dimension. This should offer more opportunities to exploit preattentive rendering and computing density critical attributes.

Expanding into three dimensional space, spatial Delaunay triangulations[8] would be needed. Most of the techniques explored throughout the paper can be directly applied to the computation in a three dimensional Delaunay triangulation, since the techniques are independent of the ambient dimension. Unfortunately, the construction of Delaunay triangulation in three dimensions becomes more difficult. It is known that the optimal computation time of Delaunay triangulation in three dimension is $O(n^2)$ time using a randomized incremental approach; [Müc95]. A particular problem in three dimensional Delaunay triangulation is that it is very difficult to devise a dynamic algorithm, especially for a deletion operation[9]. It is mainly because the locality property in three dimensional Delaunay triangulation is intrinsically more complicated than the two dimensional counterpart; 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 4.2.1 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.

References

[ACI97] D. Alberts, G. Cattaneo, and G. F. Italiano. An Empirical Study of Dynamic Graph Algorithm. *ACM J. of Experimental Algorithmics*, 2, 1997.

[AGMR98] G. Albers, Leonidas J. Guibas, Joseph S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points. *Internat. J. Comput. Geom. Appl.*, 8:365–380, 1998.

[Ata85] M. Atallah. Some dynamic computational geometry problems. *Comput. Math. Appl.*, 11(12):1171–1181, 1985.

[BDIZ97] J. Basch, H. Devarajan, P. Indyk, and L. Zhang. Probabilistic analysis for combinatorial functions of moving points. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 442–444, 1997.

[8] For the better readability, we use "triangulation" instead of "tetrahedrization".

[9] At this moment, we are not aware of any reported work on the deletion operation from a three dimensional Delaunay triangulation.

[BGH97] J. Basch, Leonidas J. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 747–756, 1997.

[BGSZ97] J. Basch, L. J. Guibas, C. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 388–390, 1997.

[BGZ97] J. Basch, L. J. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 344–351, 1997.

[BKOS97] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry - Algorithms and Applications*, chapter 5. Springer, 1997.

[Bla96] H. Blanchard. Dominant battlespace awareness. Posted on the C4I-Pro Archive, February 1996.

[Bli82] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[C+96] Bernard Chazelle et al. Application challenges to computational geometry: CG impact task force report. Technical Report TR-521-96, Princeton Univ., April 1996.

[CCFM97] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proc. 29th Annu. ACM Sympos. Theory Comput.*, pages 626–635, 1997.

[DD90] M. Dickerson and R. Drysdale. Fixed-radius near neighbors search algorithms for points and segments. *Information Processing Letters*, 35:269–273, August 1990.

[Dev98] O. Devillers. On deletion in Delaunay triangulation. Research Report 3451, INRIA, 1998.

[DJC+98] J. Durbin, S. Julier, B. Colbert, J. Crowe, B. Doyle, R. King, T. King, C. Scannell, Z. Wartell, and T. Welsh. Making Information Overload work: The dragon software system on a Virtual Reality Responsive Workbench. *Proceedings of the 1998 SPIE AeroSense Conference*, 3393:96–107, April 1998.

[DMT92] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Comput. Geom. Theory Appl.*, 2(2):55–80, 1992.

[EGIN92] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification: A technique for speeding up dynamic graph algorithms. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 60–69, 1992.

[End88] M. Endsley. Design and evaluation for situation awareness enhancement. In *In Proceedings of Human Factors Society 32nd Annual Meeting*, volume 1, 1988.

[fHC00] Institute for Human and Machine Cognition. Tactile situation awareness system. http://www.coginst.uwf.edu/tsas/main.html, Feb 2000.

[Fre97] G. Frederickson. A data structure for dynamically maintaining rooted trees. *J. Algorithms*, 24:37–65, 1997.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[GK99] L. Guibas and M. Karavelas. Interval methods for kinetic simulations. In *Proceedings of the fifteenth annual symposium on Computational Geometry*, June 1999.

[HK95] M. R. Henzinger and V. King. Randomized dynamic graph algorithm with polylogarithmic time per operation. In *Proc. 27th Sympos. on Theory of Computing*, pages 519–527, 1995.

[HKW+98] C. Hoffmann, Y. Kim, R. Winkler, J. Walrath, and P. Emmerman. Visualization for situation awareness. In *Workshop on New Paradigms on Information Visualization and Manipulation*, 1998.

[ICK+99] K. Hoff III, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of SIGGRAPH '99*, Aug 1999.

[IKI94] M. Inaba, N. Katoh, and H. Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k-

clustering. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 332–339, 1994.

[KG89] J. Keil and C. Gutwin. The Delaunay triangulation closely approximates the complete Euclidean graph. In *Proc. 1st Workshop Algorithms Data Struct.*, volume 382 of *Lecture Notes Comput. Sci.*, pages 47–56. Springer-Verlag, 1989.

[KHC90] S. Kass, D. Hershler, and M. Companion. Are they shooting at me: An approach to training situation awareness. In *Proceedings of Human Factors Society 34th Annual Meeting*, 1990.

[LG98] M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proceedings of IMA Conference on Mathematics of Surfaces*, 1998.

[MAK] MAK Technologies, 185 Alewife Brook Parkway Cambridge, MA 02138 USA. *MÄK Plan View Display User's Guide*, 1.2 edition.

[MN98] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing.* Cambridge University Press, New York, 1998.

[Müc95] E. P. Mücke. A robust implementation for three-dimensional delaunay triangulations. In *Proceedings of the 1st International Computational Geometry Software Workshop*, pages 70–73, 1995. To appear in the *International Journal of Computational Geometry & Applications.*

[RC00] OTT Resource Coordinator. References and readings related to situation awarenes. http://www.ott.navy.mil/, April 2000. OTTAdmin@navair.navy.mil.

[RJJ+94] P. Rosenbloom, W. Johnson, R. Jones, F. Koss, J. Laird, J. Lehman, R. Rubinoff, K. Schwamb, and M. Tambe. Intelligent automated agents for tactical air simulation: A progress report. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, 1994.

[SA95] M. Sharir and P. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*, chapter 8. Cambridge University Press, 1995.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures*, chapter 2. Addison Wesley, 1990.

[SBS93] M. Stytz, E. Block, and B. Soltz. Providing situation awareness assistance to users of large-scale, dynamic, complex environments. *Presence*, 2(4), 1993.

[Sno97] J. Snoeyink. *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.

[Suk97] R. Sukthankar. *Situation Awareness for Tactical Driving*. PhD thesis, Carnegie Mellon University, Jan. 1997.

[Tei99] Monique Teillaud. Three dimensional triangulations in CGAL. In *15th European Workshop on Computational Geometry*, 1999.

[WLML98] A. Wilson, E. Larsen, D. Manocha, and M. Lin. IMMPACT: A system for interactive proxmity queries on massive models. Technical report, UNC Computer Science TR98-031, 1998.

[WWE+00] J. Walrath, R. Winkler, P. Emmerman, C. Hoffmann, and Y. Kim. Visualization technique for improved situation awareness. In *Society for Imaging Science and Technology (IS&T)International Society for Optical Engineering (SPIE) conference on Human Vision and Electronic Imaging V*, Jan 2000.