CHALMERS


# A dynamic approach to area mapping in a geographic information system

*Master of Science Thesis in Software Engineering and Technology*

ERIK JUTEMAR
MARTIN RUZICKA

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, September 2010

A dynamic approach to area mapping in a geographic information system

Erik Jutemar
Martin Ruzicka



Examiner: Björn von Sydow

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden September 2010

**Abstract**

This report introduces a dynamic approach to area mapping by investigating how a geographic information system can be utilized to create and manage areas. The main purpose of the report is to investigate how to create areas from user generated content and how to store and query the areas efficiently and correctly. The implemented system starts the processing of the input by applying the Autoclust algorithm in order to remove deviating data points. The actual area boundaries are generated using the Alpha shape algorithm. Both algorithms are dependent on the Delaunay graph, which is generated by a Delaunay triangulation algorithm. To dynamically create an area, an alpha value based on the standard deviation of the edge lengths of the Delaunay graph created from the Autoclust step is used. The areas are stored in an R-tree data structure using a MySQL database with spatial extension, allowing for efficient storage and fast queries. Alternative algorithms and data structures are discussed and compared in detail.

*Keywords:* Alpha shape, Autoclust, Delaunay triangulation, GIS, Point in polygon

**Sammanfattning**

Denna rapport introducerar ett dynamiskt tillvägagångssätt för areamappning genom att undersöka hur ett geografiskt informationssystem kan användas till att skapa och hantera areor. Rapportens huvudsakliga syfte är att undersöka hur areor kan skapas från användargenererad indata och hur dessa areor kan lagras och sökas igenom på ett korrekt sätt. Det implementerade systemet börjar med att processera indatan genom att applicera Autoclust-algoritmen för att avlägsna avvikande datapunkter. Den faktiska areagränsen genereras genom att använda Alpha shape-algoritmen. Båda algoritmerna är beroende av Delaunaygrafen som genereras genom en Delaunaytrianguleringsalgoritm. För att dynamiskt skapa en area används ett alfavärde baserat på kanternas standardavvikelse i Delaunaygrafen som skapats i Autocluststeget. Areorna lagras i en MySQL-databas med ett spatiellt tillägg, som utnyttjar en datastruktur baserad på R-träd. Detta tillåter effektiv lagring och snabba sökningar. Alternativa algoritmer och datastrukturer diskuteras och jämförs också i detalj.

## Acknowledgements

We would like to thank our examiner Björn von Sydow for all support during the process of this thesis. We also want to thank our technical supervisor Tomasz Matuszczyk for being available for questions and inquiries.

Madrid, September 2010

# Contents

# 1 Introduction

Geographic Information Systems (GIS) are systems focused on spatial database applications. GISs handle storing of geospatial information, spatial queries, and modification of spatial data. There are numerous GISs available, with different applications in several fields. The GISs discussed in this report focus on modeling geographic data, such as roads or areas. However, most GISs also include functionality for analyzing and transforming spatial data in order to extract additional information from it.

In this report, a suggestion on how to implement a GIS which dynamically creates areas and stores them is introduced. The input available is points ($XY$ coordinates), which are associated with an area. The association is defined by human users and the data is not guaranteed to be accurate and might contain errors. In order to fulfill the requirement of dynamic area creation, the errors have to be dealt with using nothing else than the relationships within the input data itself. The implemented GIS uses the Autoclust clustering algorithm to find an estimation of the area boundaries and to remove irregularities from the input. To create the area definitions, the Alpha shape algorithm is used. All areas are stored in a spatial database using MySQL with spatial extension and with this data structure, the areas can be queried quickly.

Together with the suggested solution, different alternative algorithms and data structures are introduced. The potential benefits, and drawbacks, of the alternatives are discussed in detail.

Existing GISs, and other geospatial related tools, do not solve this particular problem as a whole. There are tools which solve a subset of the problem, such as clustering or shape creation, but none which takes input points and creates areas dynamically. The following list introduces a selection of GISs and related software, such as code libraries, which are of interest to this work.

**OGC** - Open Geospatial Consortium [1], a consortium of 401 companies, government agencies and universities which are leading the development of standards for geospatial and location based services.

**ESRI** - ESRI [2] is one of the first and world leading companies within GIS. They are developing ArcGIS [3], one of the most used GIS application suites. The application suite provides capabilities such as geographic analysis, management and manipulation, viewers, servers and mobile products.

**PostGIS** - PostGIS [4] is a spatial extension to the PostgreSQL database and follows the simple features version of the OGC specification. It

provides features such as geometry types, operations on geospatial objects and efficient storage.

**GeoTools** - GeoTools [5] is an open source Java library which follows the OGC specifications. Implementations of e.g. Autoclust and the Delaunay triangulation can be found here.

**Oracle Spatial** - The spatial features of the Oracle database server. It provides a variety of applications, from a GIS and mapping management to wireless location services. [6]

**MySQL Spatial extension** - A subset implementation of the "SQL with Geometry Types" environment, proposed by the OGC. The spatial extension in MySQL is more limited than the one found in PostGIS. [7]

**CGAL** - A comprehensive C++ library with geometrical algorithms and data structures. Implementations of e.g. Delaunay triangulation and the Alpha shape algorithm can be found here. [8]

## 1.1 Purpose

The main purpose of this report is to investigate and present how to dynamically create areas from user defined data and how to store and query this newly created spatial data efficiently and correctly. The purpose can be split into the following parts:

- Investigate how to efficiently store and query spatial data, in the shape of areas.
- Dynamically, without other input than the user generated data, create areas efficiently and correctly.
- Suggest, based on theory, how to implement such a system and compare it to other similar solutions.

## 1.2 Method

Initially, a comprehensive literature study was performed. Currently available GISs were examined and evaluated together with algorithms for spatial area creation, storage, and usage. The algorithms were implemented and tested rigorously in order to be able to compare them and draw conclusions. Data structures for storing and querying spatial data were also studied, using existing tools such as the MySQL database management system. Finally, a complete system was implemented using some of the data structures and algorithms studied in the previous step.

## 1.3 Limitations

This report is limited to two-dimensional spatial data, including points, lines, and polygons - the prerequisites for representing areas. The only data format dealt with is vector based data. Other formats, such as rasters, are not considered at all.

The report focuses on the technical aspects of creating areas from user generated content. The report will not deal with principles on how to improve the quality of user generated data.

# 2 Theory

This chapter is outlined as follows: in section 2.1, a general description of a GIS system is presented, 2.2 describes different methods of storing spatial data, and section 2.3 investigates methods on how to manipulate spatial data using different algorithms and techniques.

## 2.1 Geographic information systems

A geographic information system is essentially a spatial database application with features such as data processing, analysis, and graphical user interfaces. GIS application ranges from different areas such as geography, biology, and chemistry[9] to name a few. Geographic data modeled by a GIS can be e.g. topologies, radiation levels, or train networks. The system might be used only to present real world data, such as roads, yet there is also the possibility of creating new spatial information from the acquired data. This report will focus on the latter, where a system implemented to handle inexact spatial data in order to create area definitions is presented.

### 2.1.1 Data modeling

In a vector based GIS, the most fundamental data types used to describe geographic data are points, line segments, and polygons. These data types are basically static representations in terms of $X$ and $Y$ coordinates and only model the spatial location of an object. A point is limited to a location that can be specified by a single pair of coordinates. However, this requires some sort of abstraction since it depends on what kind of level or resolution is needed for the point to represent a "real world" object. A line segment consists of two coordinate pairs that binds a line, although without a width attribute, abstraction is required depending on the level of resolution. The abstraction could be that a road can be seen as a line segment with a high level of resolution, but for a lower level a line segment might be insufficient to describe the road[10]. A line segment can also be referred to as an edge.

A polygon is defined as a two-dimensional shape whose boundaries are defined by a set of connected line segments, forming a simple closed curve. A polygon may contain holes and is usually classified as convex or concave. The definition of a convex polygon is that each internal angle of the polygon can not exceed 180 degrees or, equivalently, every line segment between the vertices of the polygon is inside or on the boundary[11]. A concave polygon is a polygon which is non-convex, with an interior angle of greater than 180 degrees. This report will only deal with simple polygons, i.e., polygons without non-adjacent edges that intersect each other. [10]

## 2.2 Storing spatial data

When building GIS applications, a spatial indexing method is needed for the database where geographic information is stored. The problem with traditional indexing methods is that they do not consider the spatial hierarchy of the data, which makes them unsuitable to use for large datasets of spatial data. The spatial indexing method is needed in order to handle the large and complex geographic data as well as operations such as distance between spatial objects or if spatial objects are found within a certain area. Spatial indexing improves on these issues with more efficient storage and faster search operations.

In order to efficiently query for spatial objects, it is common to approximate spatial objects by simpler representations. The simple representations can then be used to index the more complex objects, allowing for quick queries at the cost of accuracy. A simple yet effective way to enclose two-dimensional spatial objects, such as polygons, is to use minimum bounding rectangles (MBR). An MBR only needs two points to describe its spatial information, thence requiring little storage space. MBRs approximate regular convex or concave polygons fairly well, yet they do not efficiently approximate diagonal objects such as lines (e.g. a road). [12]

A typical record for a line can be represented in a database by storing its two defining points. This makes it easy to perform queries on line segments but it is very inefficient for proximity queries since there is no inherent hierarchy which can map the relationship between the spatial data. A common way to solve this issue is to use an index which sorts the spatial occupancy of the object. [13]

### 2.2.1 Data structures

In order to improve the indexing performance in geographic databases, a good data structure is needed for the storage. The two most common data structures used in GIS databases are Quadtrees and R-trees, or other data structures using these two as a foundation. Both Quadtrees and R-trees adapt concepts from the B-tree in different aspects[14, 15]. The B-tree is a generalization of the binary tree and keeps data sorted, allows search, insertion and deletion in logarithmic time, sequential access to the data and is optimized for reading large data blocks.

### 2.2.2 Quadtree

The Quadtree data structure is a tree structure used to partition two-dimensional space. The data structure is recursively divided into four quadrants of equal size. Each cell has exactly four children or none. In the case when a cell has no children, it is considered to be a leaf. The structure of the Quadtree can be seen in Figure 1. The first level contains the quadrants $Q1$, $Q2$, $Q3$ and $Q4$. The first quadrant contains sub-levels and is divided into four new quadrants. Each quadrant in a Quadtree has a determined size, and therefore the bounds of the Quadtree itself is of a specific size.

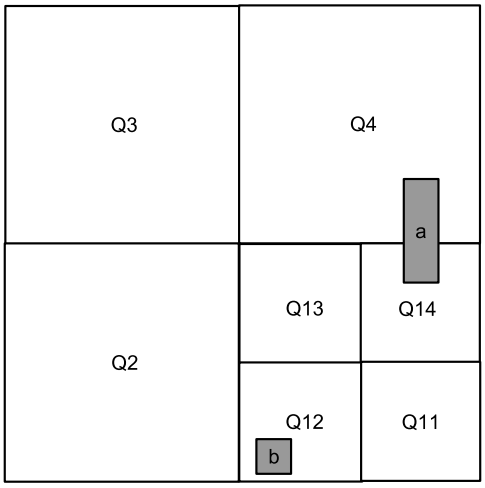



**Figure 1:** Visualization of the Quadtree structure. The first level has four quadrants. Quadrant $Q1$ contains both the data items $a$ and $b$ and is split into four new quadrants.

Quadtrees can be implemented using different composition schemes, where the space is divided in to adaptive cells, of which different types are better suited for storing certain data types. Examples of composition schemes are[13]:

**Region subdivision** - The space of the Quadtree is subdivided until a homogeneous region is found. As long as a quadrant contains more than one data item, the space is divided. In Figure 1, the items $a$ and $b$ can both be seen in the first quadrant. Therefore, the quadrant is divided so that both items are contained within one cell each, $Q12$ and $Q14$, respectively.

**Point subdivision** - When a point is inserted in a Quadtree with point subdivision, it is matched to a quadrant in which it can be contained. If the quadrant already contains a point, it is split into four new quadrants using the existing point as the center for the split. The point to

be inserted is then placed in one of the new quadrants, in correct relation to the center point. This is suitable for location-based queries, e.g. "where is location $x$?". The drawback is its inoptimality for examining every block, e.g. "is this the block I am looking for?".

**Edge subdivision** - Edge subdivision is used when storing lines or curves. When subdividing edges, each vertex in the edge is located in its own cell in the Quadtree.

The insertion operation in Quadtrees works similar to that of B-trees. At each node a comparison is made and a sub-tree is chosen until a leaf node is reached. A problem when deleting nodes from a Quadtree is to decide what to do with sub-trees to that node. According to *Finkel et al*[15], the merge operation does not seem to have a better solution than re-inserting the sub-trees.

Two ways of using the data structure with spatial querying is (i), to determine if a given point is located within the data structure, and if so, where. In this case, a similar algorithm to the insertion algorithm can be utilized. The other way of querying the structure, is (ii) to find all points within a given an area (circle, rectangle, polygon, etc.). One solution to this problem could be, while looking at the MBR of the input search area, to recursively traverse the Quadtree and find intersections between the input search area and the quadrants at the current tree level.

### 2.2.3 R-tree

The R-tree is a hierarchical data structure for spatial storage and retrieval. It is based on the B-tree data structure and is well suited for geographic information systems.

The properties of the R-tree which makes it suitable for GISs are: (i) allowance for efficient dynamic updates and modifications of the tree; (ii) handling of large datasets; (iii) efficient range searching and (iv) there is no need for pre-defined area coverage. It is also well suited for spatial data with gaps or incomplete coverage. [14]

Each node in an R-tree has a rectangular region. The rectangular region of all children to a node must fit within its parent's rectangular region. When searching, the rectangle of the search region is compared recursively with the rectangles of the tree nodes. Since the R-tree uses rectangles rather than complex shapes, the search is simple and efficient. This is illustrated as in Figure 2, where the spatial objects $a$ and $b$ are entirely enclosed by region $R3$, $c$ and $d$ by the region $R4$. These two regions are successively enclosed by region $R1$. A search for the object $a$ would result in a comparison of the

search subject with the regions, and recursively traverse from region $R1$ to $R3$ in which $a$ is found.

A leaf node in the R-tree contains index entries for the actual data items. An index entry is described by a pointer to the actual data item along with a minimum bounding rectangle (MBR) which entirely encloses the data item. Non-leaf nodes contain index entries which point to the sub level indices and an MBR which entirely encloses the child data items.

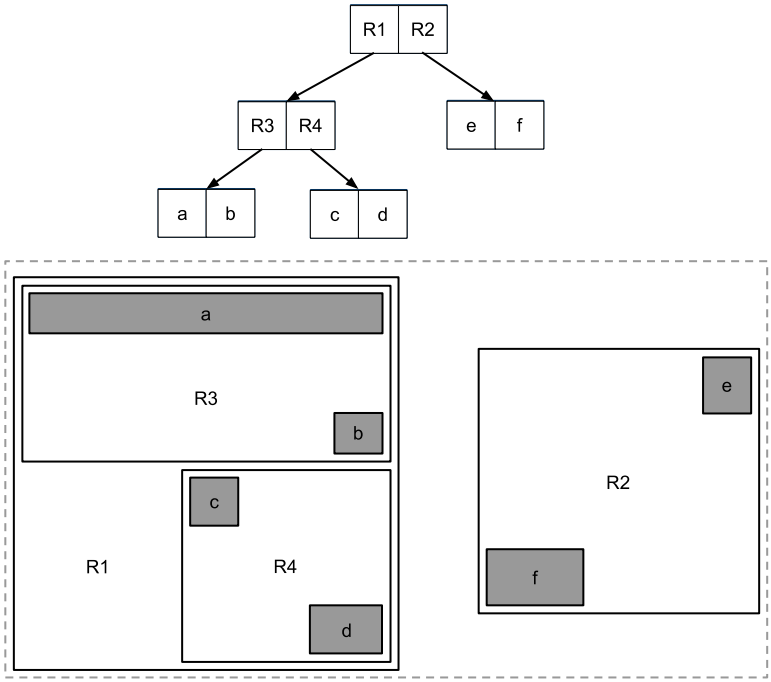



**Figure 2:** The hierarchy of a R-tree, showing how the nodes are located in relation to its MBRs.

The R-tree order is defined as the minimum number of nodes in a non-leaf that is not the root, $b$, and the maximum number of entries allowed in a non-leaf node, $B$, which is equal to or larger than $2b$. With these definitions a fill criteria can be defined: each node must have between $b$ and $B$ nodes unless it is the root node to keep the tree balanced when inserting uniform data. The root node may only contain two entries.

If an insertion causes the number of entries in a leaf node to have more than $B$ entries, the entries in the leaf node must be split into two new leaf nodes. This split can propagate up the tree and if necessary the root must be split and the tree height will increase by 1.

To improve the performance of the R-tree, the node split algorithm can be optimized. The goal is to minimize the area coverage of the nodes after

the split, which improves pruning during search as well as minimizes the intersection area between nodes and reduces collision. Variants of the R-tree (see section 2.2.4) focus on improving the node split algorithm.

When deleting nodes in an R-tree a problem occurs. There may be multiple paths down the tree to the item to be deleted. While descending in the tree, it is impossible to know if the MBR of the node currently observed truly contains the data item to be deleted. Therefore the tree must be traversed to its leaves, and potential alternative paths must be remembered in order to backtrack when a leaf proves not to be the item to be deleted. Once the data item has been deleted, the tree may propagate MBR changes to the root through the traversed path. [16]

The R-tree is more suitable for nearest neighborhood searches and requires less space than the Quadtree. To search for all records contained by a given area, the tree can be recursively traversed by checking if the nodes at the current level intersecting the given area and, if so, calling the search function again, otherwise return the entries which intersects with the given area. A drawback when searching in a R-tree is that many nodes need to be examined since rectangles can overlap, even though the data item searched for only is contained in one leaf. [14]

### 2.2.4 Variants of Quad/R-trees

There are numerous variants of the Quadtree and R-tree with different optimizations or adaptations to special cases of usage. This section mentions some of them with a brief description.

**Octree** - The Octree builds on the same principles as the Quadtree but uses eight children at each node instead of four. This data structure is primarily used when storing three-dimensional spatial objects. [13]

**R$^*$-tree** - The R$^*$-tree is adapted to minimize the overlap of MBRs, thus minimizing the node coverage. The overlap minimization comes with a new insertion algorithm which requires re-insertion. The re-insertion step is necessary since the MBR coverage produced by the split algorithm in the original R-tree can be further optimized. The R$^*$-tree also supports storing points as well as areas. [17]

**R$^+$-tree** - The R$^+$-tree focuses on making sure that internal nodes do not overlap, and on minimizing the paths to the nodes. This is done by allowing a split of regions MBRs. Due to the reduction of overlaps, the tree becomes larger and more complex than a normal R-tree since it contains duplicate MBRs. [18]

## 2.3 Managing spatial data

As described in the previous section, storing spatial data is a complicated task. This, however, is only a first step as there is a need for transforming the data into something useful. If the GIS would be seen as a black box, the users of that system could ask queries to retrieve the data but also to extract new information such as which points are in the proximity and where a point is located. There are numerous possible queries for such a system and the following subsections will focus on how to query and transform spatial data with respect to areas and polygons.

### 2.3.1 Point in polygon

The point in polygon (PIP) problem is one of the most elementary operations in computational geometry and it is defined as: given an arbitrary point $Q$ and a polygon $P$ with $N$ vertices, determine whether point $Q$ lies within the boundaries of $P$. The problem itself is relatively elementary, yet for polygons with many vertices and different shapes, an efficient algorithm is required for finding a solution quickly. The problem can also be referred to as the point inclusion test or point inclusion query. [19]

Applying a PIP-algorithm to all objects in the data structure in order to test for inclusion is not feasible for larger datasets. Thence, the PIP-algorithms are useful when performing spatial queries which return the minimum bounding rectangle of spatial objects (as described in section 2.2). Given the MBRs of possible spatial objects that may encapsulate the point, one can now perform a PIP-algorithm on a much smaller subset of spatial objects, compared to examining all stored spatial objects for point inclusion.

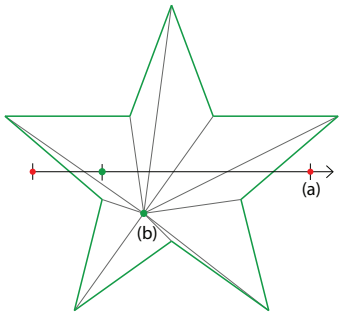



**Figure 3:** Two PIP-algorithms on a star shaped polygon. (a) Ray intersection, (b) Sum of angles.

The field of PIP-algorithms has been well studied and there are several proposed solutions for the problem, with different complexities and efficiencies. Two of the most fundamental algorithms are the sum of angles method and the ray intersection method, shown in Figure 3. There are also several kinds of PIP-algorithms that divide the polygon into smaller parts, thus minimizing the search space. One way to achieve this is to use the cell-based grid method [20].

**Sum of angles** If the sum of all angles from each vertex of $P$ to a point $Q$ is 360 degrees, the point $Q$ is inside the polygon, Figure 3 (b). The angles are computed using the point $Q$ and each edge's vertices in sequence, namely $p_i q p_{i+1}$. This algorithm works for both convex and concave polygons and has a time complexity of $O(N)$. However, the calculations required for determining the angles are inefficient and often time consuming due to the floating point operations required in real implementations. This will also affect the correctness of the algorithm since an implementation on a standard computer architecture may yield different results if it is run several times with the same input. [20]

**Ray intersection** The ray intersection method casts a ray from the query point (in any direction) and counts the number of intersections with the edges of the polygon. If the number of intersections is an odd number $Q$ is inside $P$, otherwise $Q$ is outside. The most common approach for the ray used in the algorithm is to use a ray parallel to the one of the coordinate axis'. Intersections are then calculated for each edge, giving the algorithm a time complexity of $O(N)$. The algorithm is applicable for both convex and concave polygons. The working principle of the ray intersection can be seen in Figure 3 (a) where three points are shown togeather with three rays lying on the same vertical line, two points with an even number of intersections and one with an odd number. [20]

The implementation of a ray intersection algorithm is straightforward in its simplest form. One selects a ray, preferrably parallel to one of the coordinate axis', and examine possible intersections for each edge of the polygon. However, there are special cases that must be considered. The chosen ray may coincide with a vertex (edge node) of the polygon which will result in both edges being counted as intersecting with the ray. The solution to this problem is to only count the second edge if it is below the ray. [21]

**Cell-based** The cell-based containment algorithm is a method to solve the point in polygon problem by dividing the occupied space of the polygon into cells before performing point inclusion tests on the newly created cells of the polygon. The algorithm relies heavily on a rather time consuming pre-processing step where the polygon is divided into a grid of cells where each cell is assigned different colors. Once the grid structure has been created, repeated queries can be performed in $O(1)$ or $O(\sqrt{n})$ time depending on where the query point is located in the grid. [22]

The pre-processing consists of three steps. The first step is to construct a grid of cells of equal size. The size of the cells determines how much time is needed for the whole pre-processing step, a smaller size requires more time

but the inclusion test becomes quicker. The second step identifies the cells containing polygon edges, and these cells are marked as being gray. *Zalik* and *Kolingerova*[22] suggest using a code-based algorithm to find the gray cells. The third, and last, step marks cells inside of the polygon black and outer cells white. The process starts with iterating through the outer cells of the grid and mark them as white until a gray cell is found. This does not mark all outer cells as white since a cell might be encapsulated by gray cells. The solution is to proceed with a ray intersection inclusion test for an arbitrary point in each of the unmarked cells to mark the cell as either white or black. The pre-processing step runs in $O(n^2)$ time. [22]

To determine whether a point resides within the polygon, there are two possibilities: the point can reside in a cell classified as black or white, then the lookup will take $O(1)$ time. The second case is if the cell is marked as gray. Here, the closest edge in the cell is selected to determine whether the point is inside or outside the polygon. By examining the scalar product of the vector for the closest edge and the vector for the point to the first point in the chosen edge, a signed value is acquired . If the sign is 0, the point is located on the border of the polygon. If the sign is greater than 0, the point is inside the polygon and if the sign is less than 0, the point is outside. This is done in $O(\sqrt{n})$ time.

The cell based PIP-algorithm works for both convex and concave polygons and is suitable for polygons with a large number of vertices. [22]

### 2.3.2 Transforming spatial data

In a GIS it is commonly not applicable to use a raw set of spatial data without any further manipulation, making the structuring and modeling of the spatial data a vital part of the analytical capabilities of a GIS[23]. Transforming the data in order to extract relevant information from the system is therefore an essential operation. The following section describes methods on how to transform sets of points into other entities which can further be manipulated so that additional value can be added to the data.

### 2.3.3 Convex hull

One of the most common techniques for defining the outer boundaries of a point set is to use a convex hull algorithm. The convex hull of a finite point set $S$, or $conv(S)$, is a set of all convex combinations of the points in $S$. We will not try to formalize this further, but will only say that the $conv(S)$ is the convex combination of all points, resulting in a convex polygon, see

Figure 4. Convex hull has many applications in computational geometry, e.g. surface approximation or collision detection. [11, p. 63-96]

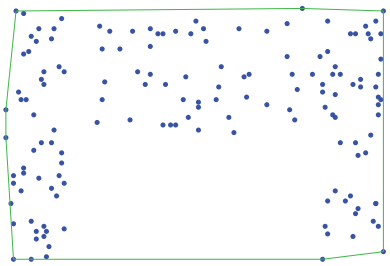


**Figure 4:** Convex hull of a point set, produced by the QuickHull algorithm.

Algorithms for defining the convex hull have been studied thoroughly and some common algorithms used are QuickHull, Graham's scan, or Chan's algorithm. All of them accomplish a convex hull with different time complexities for different point sets. We will not further study convex hull algorithms, but only mention them since they are building blocks for many other applications in computational geometry. [11, p. 63-96]

### 2.3.4 Delaunay triangulation

In order to extract valuable information from a set of points so that new spatial information can be generated, the Delaunay triangulation ($DT$) is a fundamental step. It is used for many different applications, such as terrain modeling[24], fingerprint identification[25], computation of skeletons[26], and finding efficient routing in ad hoc wireless networks[27], to name a few. In a GIS however, the $DT$ is often used to analyze the inner relationships of a point set.

The Delaunay triangulation in two dimensions $DT(S)$ of a set of points $S$ is a set of triangles which are defined as[28]:

- The point $p$ is a vertex of a triangle in $DT(S)$ if and only if $p$ belongs to $S$.
- The intersection of two triangles in $DT(S)$ is either empty or it is a shared edge or a shared vertex.
- The triangles in $DT(S)$ cover the area of the convex hull of $S$.
- The circumcircle of any triangle in $DT(S)$ does not contain any other point in $S$.

Figure 5 shows a Delaunay triangulation for an example point set. From the figure, it is apparent that the definition of a Delaunay triangulation is

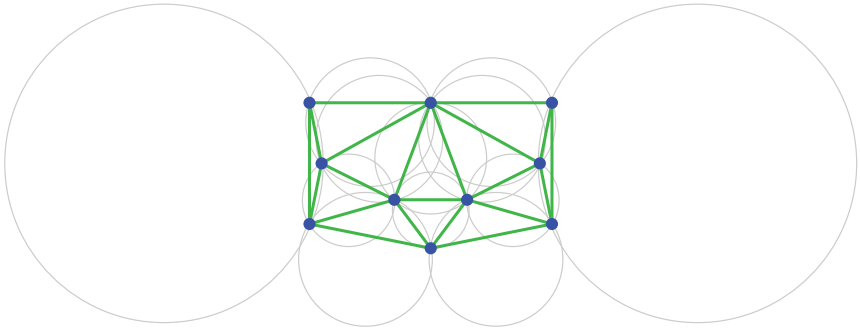


**Figure 5:** The Delaunay triangulation of a small point set, showing the triangles, points and circumcircles for each triangle.

fulfilled, i.e. all points are vertices of triangles, the triangles do not intersect (only share edges), the convex hull is covered and the circumcircles of each triangle do not contain any other points than the vertices of the triangle.

There are several algorithms for constructing the Delaunay triangulation. The sequential ones can be divided into five categories: divide-and-conquer, sweep-line, incremental, growing triangles in a manner similar to convex hull algorithms using gift wrapping, and lifting the point set into three dimensions using the convex hull to calculate the triangulation. The algorithms mentioned have different performance depending on the properties of the point set, e.g. if the set is uniformly distributed or not. [29]

Incremental algorithms for constructing a Delaunay triangulation, which perhaps are the simplest implementation wise, add points one at a time and update the graph accordingly after each insert. The insertion process can be divided into two steps: locate and update. [29]

The locate step of the insertion process finds the triangle containing the new point by starting at a random (or latest added) triangle in the diagram and then walking across the diagram in the direction of the new point until the correct triangle is found. In order to find the exact triangle, every edge of the triangle must be compared to the point to see on which side of the triangle it lies. This is typically done by a counter clockwise orientation in the triangle. If the point is not on the correct side for an edge, while it is still in the correct direction, the other triangle sharing that edge is investigated. [29]

The update step flips all edges, which were invalidated by the newly inserted triangle, so that each triangle fulfills the properties of the Delaunay triangulation. The worst case time complexity for an algorithm implemented with locate and update is $O(n^2)$ since it is possible that a point set requires $O(k)$ flips for each newly added $k^{\text{th}}$ point.

However, it has been shown that inserting in random order makes the number of expected edge flips linear. [28]

The Delaunay triangulation of a point set also has several applications in graph theory, such as solving nearest neighbor problems or finding the largest empty circle. [11, p. 161-9]

### 2.3.5 Spatial clustering

Acquired spatial data is often not ready for use as is and sometimes it is necessary to process the data in order to extract additional value from it. A GIS needs to detect patterns and classify data automatically since this process is impossible to do manually for large datasets. Classifying spatial data into clusters is a common approach in GISs for this purpose. Clustering of data points in its essential form is to separate elements, without prior labeling, into different groups. This means that the outcome of a clustering process will be a classification of the input, allowing for a more comprehensive analysis on the spatial data. [30]

There are several spatial clustering methods which can roughly be divided into five categories: partition methods, hierarchical methods, density-based methods, graph-based methods, and learning-network methods. Each method deals with the datasets differently. Many of these algorithms, especially the ones based on the traditional method $k$-means clustering [31, p. 278], suffer from the drawback that they rely on pre-defined values for the algorithm to work. These values can not be used on all datasets and need to be set differently depending on the correlation of the data. [32]

Dependencies on pre-defined input values are expensive. The values will most likely not work for all datasets which makes the clustering process a trial and error process, and for large sets of input this can be infeasible. A better approach is to let the data speak for itself by only relying on relations within the input set. However, in geographic clustering there is still a perceived correctness in the eye of the beholder and there is no way of implementing a clustering that correctly defines clusters for all purposes and interpretations. [23]

The clustering problem can be defined as follows: Given a set of points $P = p_1, p_2, ..., p_n$ in some study region $R$, find smaller homogeneous groups according to spatial proximity [23]. This is a slightly diffuse definition, since measuring the correctness of a clustering algorithm usually involves measuring similarities or dissimilarities in the point set which leads to the correctness being dependent on the properties of the dataset. This report will not try to define this further, but only state that the clustering algorithm should divide the dataset into smaller homogeneous groups.

**Autoclust** Autoclust is a clustering algorithm which belongs to the family of graph-based clustering algorithms [23]. Autoclust makes use of the graph composed by the Delaunay triangulation of the point set, with each point considered as a node in the graph, to identify statistical features of the point set. By only using the Delaunay triangulation for acquiring statistical features there is no need for predefined or user-defined input, making Autoclust generic in the sense that it produces quality output from different kinds of data and removes human-generated bias.

*Estivill-Castro* and *Lee* argue that the standard deviation is the most valuable measure for describing dispersion in Delaunay triangulations. The border points of clusters have a greater spread of edge lengths than that of intra-cluster points since the border points have both intra-connected edges and inter-connected edges (edges with end points in different clusters), and in order to find these edges the standard deviation is used. [23]

The working principle of the Autoclust algorithm is divided into three phases where each phase manipulates the graph acquired from the Delaunay triangulation by continually correcting the edges.

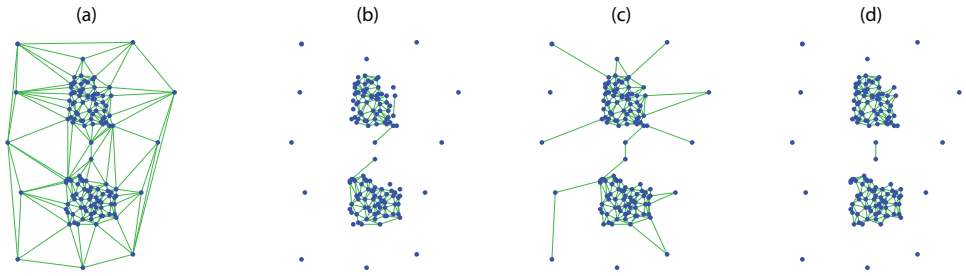



**Figure 6:** Autoclust flow: (a) The Delaunay triangulation, (b) After phase 1, (c) After phase 2, (d) Clustering complete with three clusters

**Phase 1** The first phase involves identifying edges into three categories; *short edges*, *long edges*, and *other edges*. This classification is done for each neighborhood of each node in the graph with the goal of eliminating inter-cluster edges, thus achieving a rough boundary estimation of the clusters. All edges classified as long edges are removed with the justification that these edges are merely connecting two clusters (or single nodes far away). All edges classified as short edges are also removed - this is to remove "bridges" between clusters. A bridge can be nodes connected with short edges, located between two high density clusters as can be seen in Figure 6 (b), where a short edge has been removed in the centre between the two high density clusters. The short edges which are not considered as bridges will be re-added at a later stage.

**Phase 2** The goal of phase two is to restore the short edges which has endpoints in one cluster. Other edges which are creating paths that might result in bridges are deleted. However, as can be seen in Figure 6 (c), a point with another edge but also a short edge to a cluster will keep the other edge.

**Phase 3** In phase three, the immediate neighborhood for each node is extended slightly. The local neighborhood is extended to two or less connected edges and for each of these neighborhoods a reclassification is made to the edges. All long edges in the new neighborhood are removed, resulting in the final result of the algorithm, shown in Figure 6 (d).

**Triclust** Triclust is a clustering algorithm which focuses on statistical features derived from Delaunay triangulation, and is thereby a graph-based algorithm. The Delaunay triangulation is good for measuring proximity because all points are connected in a graph and relationships between points and statistical data can easily be extracted from it. [32]

Triclust has the ability to handle non-uniform inner cluster density cases, such as variations of data density distributions between and inside clusters. Triclust focuses on analyzing the actual data points instead of the edges, as in Autoclust. The data points are classified into different categories based on statistical features extracted from the neighbouring points.

The statistical features are calculated from the edge lengths in the neighbor sub-graph, from the Delaunay triangulation. The actual measurements of the edge lengths are; (i) mean of the length, (ii) standard deviation of the length divided by the mean, and (iii) the positive part of the derivative of the mean.

The main goal of the Triclust algorithm is to classify all data points into two categories; inner cluster data points, and boundary data points. From the statistical features a criteria function is defined for the classification and a final feature value can be calculated for each data point. The $k$-means algorithm is applied on the final feature values and is used for threshold detecting. With the threshold value, inner cluster data points can be distinguished from boundary data points.

The statistical features that are used in the criteria function have different characteristics depending on the data distribution. When the data density is uniform and similar, the mean is sufficient for finding inner cluster data points and boundary points. Non-uniform clusters are better detected with the standard deviation divided by mean approach. It can find boundary data points when both uniform and non-uniform clusters exist. However, if the data distribution inside the clusters is irregular, it works less well. The

benefit of the derivative of the mean approach is that it is better used to detect clusters with irregular inner distributions, although it is less suited to detect boundary data points.

The complexity of the Triclust algorithm is $O(n \log n)$ if the Delaunay triangulation algorithm has complexity of $O(n \log n)$ or less. Finding neighbouring data points and calculating edge lengths are linear in $n$ as well as updating the cluster neighborhoods. [32]

### 2.3.6 Concave hull

Given a set of points $S$ in $\mathbb{R}^2$, it is sometimes desirable to extract the boundary shape of $S$. However, the notion of a "shape" has no formal meaning due to its ambiguity and that the shape's correctness lies in the eye of the beholder. The Alpha shape algorithms offers one interpretation of a shape derived from a set of points.

**Alpha shape** The Alpha shape algorithm is a formal way of trying to define the shape of $S$ with a certain degree of detail, as suggested by *Edelsbrunner*[33]. The level of detail is defined by an alpha value which is the radius of a circle used to determine which points that defines the Alpha shape, also called the "alpha ball". Figure 7 (a) shows how the shape can look like when using a low value. The algorithm can not define a boundary shape and the result will be a few connected edges or the original point set if alpha is zero. However, a large value of alpha would lead to the convex hull of the point set, as can be seen in Figure 7 (c) where a value of 100 results in a shape approaching the convex hull. As the value for meets somewhere in the middle, the shape will gradually follow the outline of the points, Figure 7 (b).

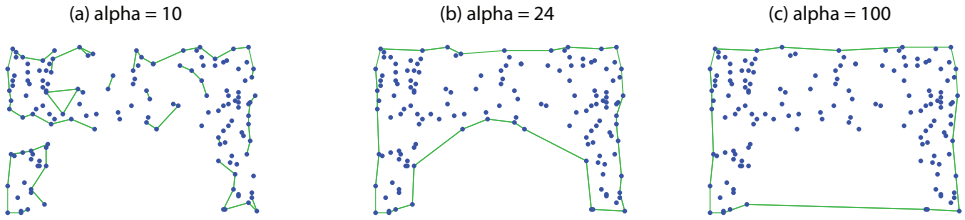



**Figure 7:** Alpha shape examples: (a) low alpha value, (b) medium alpha value, (c) a large alpha value.

The algorithm used to calculate the Alpha shape in two dimensions is described below, for a more detailed explanation and for higher dimensions and proof, see *Edelsbrunner's* paper[33]. The algorithm uses the Delaunay triangulation of $S$ as a starting point. This is because it is known that the

boundary Alpha shape is contained in it. The goal of the algorithm is to return two lists of edges, one with the boundary edges and one with all internal edges. For each edge generated from the Delaunay triangulation, two interval values, $a$ and $b$, are found by examining the circumcircle of the edge. If it is empty, $a$ is set to the radius of the circumcircle and if it is not, $a$ is set to the minimum of the circumcircle radius of the triangle of the edge and the neighboring triangle's circumcircle radius. To find $b$, the edge is examined to see if it lies on the convex hull and in that case sets $b$ to $\infty$. If it is not on the convex hull, the maximum of the circumcircles radius' of the edge's triangle and its neighbor is chosen. The intervals $a$ and $b$ are saved to the boundary list and the interval $b$ is saved to the internal list. When all edges have been dealt with, the alpha check can be performed by seeing if the alpha value is in the interval defined in $B$ and $I$ for each edge, if it is, the edge is on the boundary or in the internal edges of the shape.

# 3 Results

We successfully managed to implement a GIS which creates areas from input points of different quality and stores the results in a spatial data structure. The stored areas are then subject to different queries supported by the data structure. Figure 8 shows the data flow of the implemented GIS, where the *Data manager* and *Mapping API* are the most relevant.

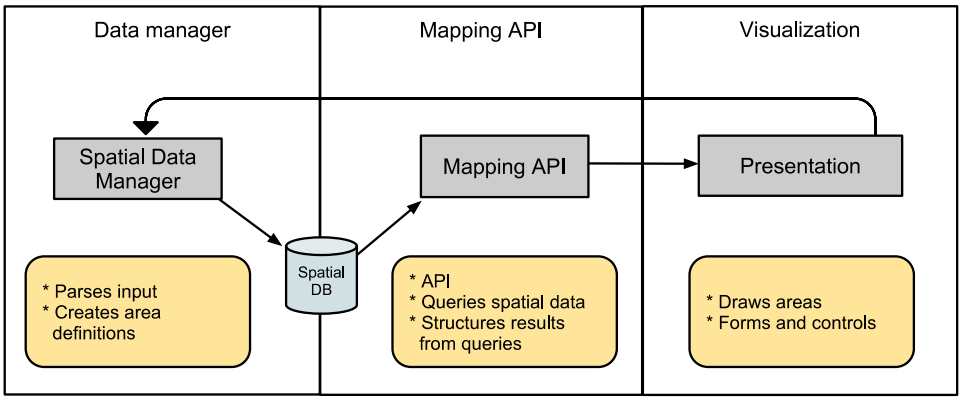



**Figure 8:** The GIS data flow divided into three different parts.

The data manager parses the input and creates the area definitions, which are stored in an R-tree data structure. We used the MySQL database with spatial extension. The areas may consist of several polygons, although they must consist of at least one. The mapping API defines the available queries for the system and structures the data properly. The visualization interface is only a simple tool to visualize the spatial information and to trigger the queries in the API. All implementations were done using the PHP programming language.

## 3.1 Area creation

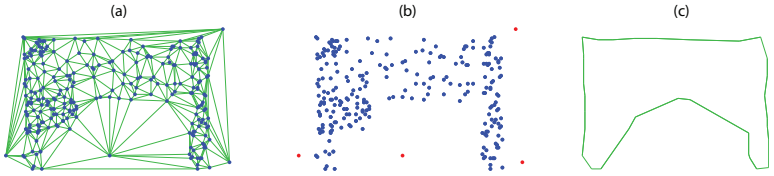



**Figure 9:** Algorithm flow. (a) The Delaunay triangulation of the input points. (b) Autclust's removal of points. (c) The final shape generated by the Alpha shape algorithm.

The area creation process starts by applying the Autoclust algorithm to the input points in order to create the clusters, using the Delaunay triangulation,

as shown in Figure 9 (a). This will also remove outlying points which are not supposed to be included in the area definition, shown in Figure 9 (b). The Alpha shape algorithm is then applied to each cluster, in order to define the boundaries of the area. In Figure 9 (c) this results in one area since there is only one cluster.

### 3.1.1 Execution times

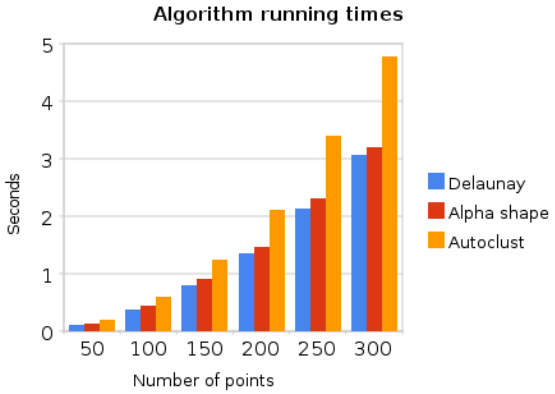
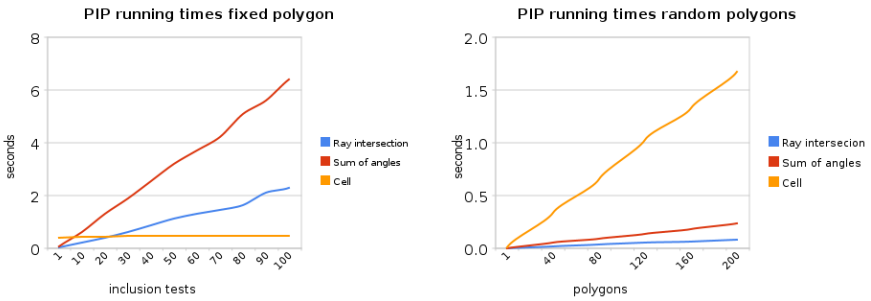



**Figure 10:** Execution times of the algorithms; Delaunay triangulation, Autoclust and Alpha shape

Figure 10 shows the execution times of the algorithms involved in the area creation. The input points for each test was randomly generated with a uniform distribution.

## 3.2 Point in polygon




**(a)** Running times using one polygon and variable inclusion tests.

**(b)** Running times using random polygons and one inclusion test per polygon.

**Figure 11:** Running times with different settings for the PIP-algorithms.

Figure 11a shows running times of the PIP-algorithms using a fixed polygon with approximately 1250 vertices. Points, both inside and outside of the polygon, were randomly generated for the inclusion tests. Figure 11b shows running times when random polygons were generated together with one random point each to test for inclusion.

# 4 Discussion

This chapter will discuss the implemented GIS and how it relates to the previous findings from the theory section. Problems with the algorithms and our solution are discussed in detail, with the additional goal of suggesting future work that could improve the solution.

## 4.1 GIS

GISs such as ArchGIS allow for storing, analysis, and manipulation of various kinds of geographic data. These applications and other GIS related tools partially solve our problem, and a combination of different systems with adaptations could possibly create a system similar to ours. In our case we wanted a consistent system that solved the problem of area definition from user generated data.

We chose to use MySQL as a database for our GIS based on its availability and compliance with other systems. Also, the spatial extensions comes bundled with the database distribution, which means that no extra configuration is needed in the installation process. It would be interesting to investigate how MySQL with its spatial extensions differs from PostGIS. PostGIS has a different implementation of the data structure for storage, as well as more functionality for spatial operations.

The CGAL library and GeoTools packages provide a comprehensive set of algorithms and data structures for geometry- and spatial-related calculations. These are however made more as a general set of tools to work with geometry- or spatial-related problems and would need a lot of adaptation in order to solve the problem presented in this report. Also, GeoTools does not provide any functionality to create Alpha shapes, and the CGAL library does not provide any clustering algorithms.

It is not possible to retrieve points within an area with our implementation. The main focus has been about defining areas from a set of points. This is however a trivial extension to our current system. The points used when defining areas can be stored in a separate structure, mapped to its corresponding area. With this addition, it is possible to ask queries, such as what original input data is located within a specific area.

## 4.2 Storing spatial data

When storing spatial data in a GIS, the underlying data structure must be able to handle spatial data in an efficient way when it comes to insertion,

searching, storage space etc. For our system, the most important property of the data structure is the ability to perform quick searches. Looking at available GISs and their storage data structure, it usually comes down to two choices, the R-tree or the Quadtree.

R-trees have the advantage over Quadtrees that there is no need for predefined boundaries of the data structure where the spatial data may be located. The R-tree has also been shown to be a much faster data structure than the Quadtree when working with spatial data when datasets are large, which usually is the case. In small datasets the Quadtree usually performs faster, but the need for this study was a data structure which would be able to store and search large amounts of data. Range searches and nearest neighborhood queries are also faster in R-Trees, which further implies its usability in our application. The R-tree data structure is used by many leading database vendors, such as MySQL, PostgreSQL and Oracle. [34, 35, 14]

## 4.3 Managing spatial data

The transformation of spatial data is fundamental in a GIS. In our case, the input data is of varying quality and since the purpose is to create areas there was a need for several different algorithms in order to succeed. The managing of the data is divided into two parts. First, the problem of actually using areas and how to as quickly as possible be able to query the data structure and get the correct response back. One part of the solution for this problem is the data structure, which handles MBR queries very fast, and the other part is to find the correct answer from the MBRs.

Second, the problem of actually defining the areas must be solved. Several algorithms and methods were studied and eventually a combination became the solution, together with some modifications in order to suit the needs of our GIS.

The solution to the problem of how to dynamically, without other input than the user generated data, create areas efficiently and correctly is discussed below, together with a discussion of the point in polygon algorithms.

### 4.3.1 Point in polygon

The requirement of a PIP-algorithm in our GIS is that it has to be able to return an answer quickly for dynamically defined polygons. This means that a pre-processing step of the algorithm can not take precedence of the total running time and that the algorithm can return correct results for convex and concave polygons. All of the three implemented algorithms were able

to give correct results for both convex and concave polygons. However, the running times differ a lot using different settings.

Of the two PIP-algorithms that do not require any pre-processing presented in section 2.3.1, the sum of angles algorithm is significantly slower due to its reliance on float operations, as can be seen in Figure 11b. The ray intersection algorithm however, with its straightforward implementation, does better and returns an answer much quicker than the sum of angles algorithm.

The cell-based algorithm performs very well when the results from the preprocessing step can be used for each inclusion test without having to be recalculated. Compared to the other two algorithms, the running times for the cell-based grid algorithm barely differs from 10 inclusion tests to 100 inclusion tests. The difference can be seen in Figure 11a. When it comes to inclusion tests on several different polygons, the preprocessing step of the cell-based algorithm takes too much time for it to be faster than the other algorithms. The additional time taken can be seen in Figure 11b.

### 4.3.2 Delaunay triangulation

The Delaunay triangulation algorithm is used as a fundamental building block of the area creation process since both the Autoclust and Alpha shape algorithms rely on it. Autoclust uses it by examining the statistical features of the nodes' neighborhoods in the graph that the Delaunay triangulation produces. This gives Autoclust the possibility of dynamically creating clusters without relying on predefined values. The Alpha shape algorithm makes use of the fact that it is known that the boundary edges of an Alpha shape are actually a subset of the edges in the Delaunay graph.

The incremental Delaunay triangulation algorithm used is relatively simple to implement. However, the incremental algorithms suffer from the drawback that the worst case complexity is very poor. This could be improved and with great benefits, due to the fundamental role of the area creation process, by using one of the more efficient algorithms such as the Dwyer algorithm[29] which is a divide and conquer algorithm running in $O(n \log \log n)$ expected time. Autoclust's and Alpha shape's heavy dependency on the Delaunay triangulation can be seen in Figure 10, where the Delaunay triangulation step stands for the majority of the time taken for the algorithms. Although this would speed up the area creation process significantly, the time for creating the areas is not an issue compared to the importance of speed of the data structure queries and PIP-queries. The Dwyer algorithm can be seen as a future work improvement to this system of dynamic area creation.

### 4.3.3 Autoclust

We use Autoclust to accomplish two things. The first is removal of points that are too far away by running Autoclust repeatedly until the resulting point set has a certain density. This is necessary due to the low quality of our input data and the incorrect points must be omitted from the calculations in order to achieve clusters that actually represent the area. The second is to define the "inner shape" of an area by assigning the points to different clusters in order to take into account obstacles in the terrain, such as lakes or rivers.

**Using Autoclust as a density function** When running the Autoclust algorithm we continually check if there is an edge that is too long, with a certain factor. Identifying this edge is done by comparing the length to the median length of all the edges. If the length is, with a certain factor (in our case 20), longer than the median length, the clusters will lack precision. In order to find the clusters with the most density, the algorithm is run again until the longest edge is closer to the mean. This method will utilize the algorithm both as a density function and as a clustering method. However, there is still the possibility of having several clusters near each other. On the other hand, this will remove the possibility of having two clusters far away from each other since there will be edges that are too long between them (imagine Figure 6 with greater distances). This behavior is, in our case, desired due to the fact that we are only interested in approximating areas and as an area might be divided by a river for example, it is highly unlikely that an area has two separate polygons which are far away from each other.

The factor of 20 times the median length has been defined by empirically examining the results of the clustering process, visually validating the result. The factor works well for the purpose of creating areas, however the actual value is probably not applicable for other implementations even though the method of using Autoclust as both a clustering method and density function is.

The actual clusters returned from Autoclust after the density step will in reality almost always only be one cluster since the last time the algorithm is run it will be on a set of points that has a high density. However, there might be cases where the result will be more than one cluster. Obstacles such as rivers for example can still divide an area into two clusters.

Autoclust fulfills the purpose of removing points which do not fit the area definition well. The implementation is straightforward and its dependency on the Delaunay triangulation, with its statistical features, makes it easy to adapt for different purposes. Since Autoclust in itself does not require

predefined variables to produce the output, there is an inherited stability of the algorithm and running the algorithm twice with the same input will generate the same results. Even though our implementation uses predefined values (the density factor), this is more of a consequence of the business requirements of the implemented GIS.

Autoclust does not work as well as for example Triclust when determining bridges between clusters. Autoclust will, in most cases, connect clusters with bridges into one cluster while Triclust would separate the two clusters. For the area approximation, it is not desirable to always disconnect the clusters since the bridge could imply that there is a connection and that there are sample points missing around the bridge. Examples of how the Triclust algorithm handles bridges compared to Autoclust is described by Liu et al[32].

The decision to opt for Autoclust instead of Triclust was due to an implementation issue of the Triclust algorithm. There seemed to be an error in the algorithm description made by the authors. The authors were contacted about the issue but none of them replied to our questions, making it impossible for us to implement the algorithm properly. Thus, the comparison of the two clustering algorithms is only theoretical. To improve the clustering process, implementing Triclust and comparing the results from the two clustering algorithms using real data would be adept.

### 4.3.4 Alpha shape

Alpha shapes, as described in section 2.3.6, rely on a predefined value which defines the level of detail for the shape. This makes the Alpha shape algorithm static in the sense that the result has to be manually evaluated, visually, and possibly recalculated several times with different alpha values to produce the desired shape.

*Edelsbrunner*[33] also recognized this problem and suggests different methods of weighting each point so that the alpha value varies depending on the weight. The weighted Alpha shape algorithm produces shapes which follow the outline of a point set very closely. For the purpose of generating vaguely defined areas that are approximations of perceived definitions by humans, the algorithm could actually give a result which is too detailed. However, this study does not consider how to interpret users and user generated content and will not try to analyze this in further detail.

To solve the problem of generating an alpha value that approximates the shape of a point set, without using a weighted Alpha shape algorithm, we used the standard deviation of the edge lengths from the Delaunay graph.

As mentioned in section 2.3.5, Autoclust uses the standard deviation to measure dispersion and divide the point set into clusters. By also using the standard deviation for calculating the alpha value, we can create shapes that approximate the point set well in most cases. Figure 12 shows an example distribution of edge lengths from the Delaunay graph, generated by Autoclust from a typical point set in the application.

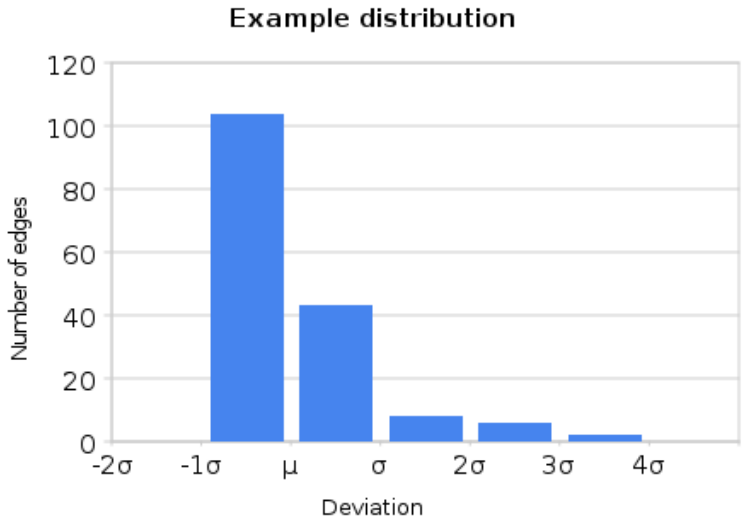



**Figure 12:** Example distribution of edge lengths of a Delaunay graph generated from a set of clustered points from Autoclust.

In the example, the majority of the edges are of short or longer lengths from the mean length by one standard deviation. This means that since most edges have lengths closer to the mean value, edges with a length greater than $\mu + 2\sigma$ are quite few. By using this as the Alpha shape value, we can create a shape dynamically. This implies that the "alpha ball" will have a radius which is greater than most edges' lengths and thereby generate a slightly more detailed area definition than what the convex hull would. This can be seen in Figure 13 where the Alpha shape is defined by the green boundary edges and the convex hull edges are brown. The alpha value (to the right) is relatively large and if a smaller value is to be used the Alpha shape will be more detailed, particularly in the inner regions of the point set which the convex hull ignores completely. However, since the Alpha shape in our case is an approximation of an area that is not strictly defined as correct, a more detailed shape could be either a worse, or a better representation of the area. This is impossible to say for certain, and therefore a mean is used, ranging from a small alpha value and detailed shape to the convex hull shape with less detail.

If the point set is not evenly distributed, the alpha value will become so large

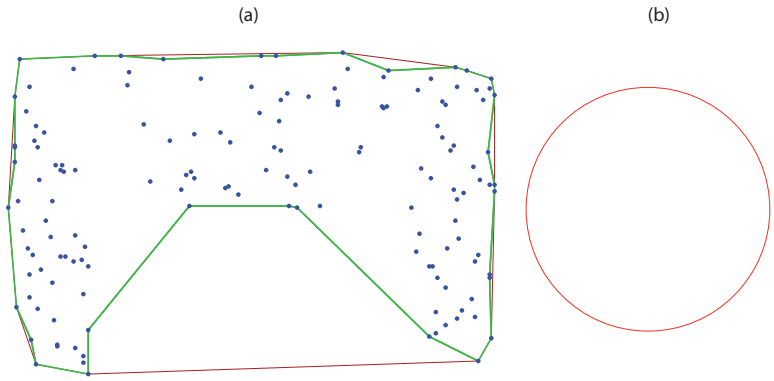



**Figure 13:** Dynamic alpha calculation: (a) The Alpha shape acquired using the alpha value based on the mean and standard deviation, (b) The "alpha ball" with alpha as radius.

that the Alpha shape will approach the convex hull. Again, we can avoid this behavior since the Alpha shape calculations are always performed on a clustered point set that has a high density.

As stated previously, the correctness of the Alpha shape lies in the eye of the beholder. However, it is still possible to verify that the edges returned by the Alpha shape algorithm form a connected shape in such a way that the shape does not intersect with itself or have holes. We use a linked list structure to verify that all the edges are connected and that each node only occurs twice in exactly two edges.

The Alpha shape algorithm depends greatly on the Delaunay triangulation, as can be seen in Figure 10 where the Delaunay triangulation stand for the majority of the computation time required for the Alpha shape algorithm. The time taken for the Alpha shape algorithm could possibly be greatly reduced if the Delaunay triangulation algorithm were to be improved.

## 4.4 Future work

An interesting aspect to the area creation process could be to implement the weighted Alpha shape algorithm[36] instead of the original one with our addition. The created areas would have a better level of detail, yet it would require that the input data to be more complete since the weighted Alpha shape would not approximate the areas, but rather create a more exact definition from the input points.

The input data is provided by users and when evaluating the created areas visually, input points which were not included could imply that the credibility of that user is low. A system where the users are ranked based on the value

of their contribution and incorporate the rank into the algorithms in order to exclude unwanted points, not only using Autoclust, could also be interesting.

As discussed, the Delaunay triangulation is fundamental for the algorithms used and in order to improve the overall performance of our GIS, investigating an algorithm with a lower time complexity can be highly beneficial. A future work could involve implementing Dwyer's algorithm[29] to see how it can affect the speed.

Another improvement to our system would be to use a variant of the R-tree data structure for storage. In our system, the most important requirement is the ability to perform fast search queries. By using the $R^*$-tree or $R^+$-tree data structure which optimizes the MBRs, search query time could be minimized. Effects such as slower insertion time or more required space for storage were of less importance for this specific project.

Exchanging the clustering algorithm to another one could also be an improvement. A clustering algorithm that can find the most dense clusters and at the same time cope with obstacles would be desirable. Triclust might be a good alternative and at least a comparison with Autoclust for this particular problem could be interesting.

## References


[1] OGC, "Open geospatial consortium." http://www.opengeospatial.org/ogc, August 2010.

[2] ESRI, "Environmental systems research institute." http://www.esri.com/about-esri/about/facts.html, August 2010.

[3] ArchGIS, "Archgis - suite of gis software products by ersi." http://www.esri.com/software/arcgis/index.html, August 2010.

[4] PostGIS, "Spatial extension for the postgresql database." http://postgis.refractions.net/, August 2010.

[5] GeoTools, "Open source java code library for manipulation of geospatial data." http://geotools.org/about.html, August 2010.

[6] Oracle, "Spatial support for the oracle database." http://www.oracle.com/technetwork/database/options/spatial/index.html, August 2010.

[7] MySQL, "Spatial support for the mysql database." http://dev.mysql.com/doc/refman/5.0/en/gis-introduction.html, August 2010.

[8] CGAL, "Computer geometry algorithms library." http://www.cgal.org/, August 2010.

[9] W. Zhang and T. Beaubouef, "Geographic information systems: real world applications for computer science," *SIGCSE Bull.*, vol. 40, no. 2, pp. 124–127, 2008.

[10] P. Burrough and R. McDonnell, "Principles of geographical information systems," *Oxford Univ. Press*, 1998.

[11] J. O'Rourke, *Computational Geometry in C*, vol. 2. Cambridge University Press, 1997.

[12] D. Papadias, T. Sellis, Y. Theodoridis, and M. J. Egenhofer, "Topological relations in the world of minimum bounding rectangles: a study with r-trees," in *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 92–103, ACM, 1995.

[13] H. Samet, "Sorting in space: multidimensional, spatial, and metric data structures for computer graphics applications," in *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, (New York, NY, USA), pp. 1–106, ACM, 2008.

[14] M. Sardadi, M. Rahim, Z. Jupri, and D. bin Daman, "Choosing r-tree or quadtree spatial data indexing in one oracle spatial database system to make faster showing geographical map in mobile geographical information system technology," pp. 326 – 334, 2009.

[15] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, pp. 1–9, 1974.

[16] J. Sheng and O. Sheng, "R-trees for large geographic information systems in a multi-user environment," (Los Alamitos, CA, USA), pp. 10 – 17, 1990.

[17] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: an efficient robust access method for points and rectangles," vol. 19, (USA), pp. 322 – 31, 1990/06/.

[18] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The r+-tree: A dynamic index for multi-dimensional objects," pp. 507–518, 1987.

[19] W. Wang, J. Li, and E. Wu, "2d point-in-polygon test by classifying edges into layers," *Comput. Graph. (UK)*, vol. 29, no. 3, pp. 427 – 39, 2005/06/.

[20] C.-W. Huang and T.-Y. Shih, "On the complexity of point-in-polygon algorithms," *Comput. Geosci. (UK)*, vol. 23, no. 1, pp. 109 – 18, 1997/02/.

[21] G. E. Taylor, "Point in polygon test. survey review," vol. 32, pp. 479–84, 1994.

[22] B. Zalik and I. Kolingerova, "A cell-based point-in-polygon algorithm suitable for large sets of points," *Comput. Geosci. (UK)*, vol. 27, no. 10, pp. 1135 – 45, 2001/12/.

[23] V. Estivill-Castro and I. Lee, "Argument free clustering for large spatial point-data sets via boundary extraction from delaunay diagram," *Computers, Environment and Urban Systems*, vol. 26, no. 4, pp. 315 – 34, 2002/07/04.

[24] S.-S. Kim and J.-H. Park, "Space-efficient terrain rendering using constrained delaunay triangulation," vol. vol.4, (Piscataway, NJ, USA), pp. 2441 – 3, 2002.

[25] G. Bebis, T. Deaconu, and M. Georgiopoulos, "Fingerprint identification using delaunay triangulation," (Los Alamitos, CA, USA), pp. 452 – 9, 1999.

[26] J. Reddy and G. Turkiyyah, "Computation of 3d skeletons using a generalized delaunay triangulation technique," *Computer Aided Design*, vol. 27, no. 9, pp. 677 – 94, Sept. 1995.

[27] D. Satyanarayana and S. Rao, "Constrained delaunay triangulation for ad hoc networks," *Journal of Computer Systems, Networks, and Communications*, vol. 2008, 2008.

[28] J. Kohout and I. Kolingerová, "Parallel delaunay triangulation based on circum-circle criterion," in *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, (New York, NY, USA), pp. 73–81, ACM, 2003.

[29] P. Su and R. Drysdale, "A comparison of sequential delaunay triangulation algorithms," vol. 7, (Netherlands), pp. 361 – 85, 1997/04.

[30] V. Estivill-Castro and I. Lee, "Clustering with obstacles for geographical data mining," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 59, no. 1-2, pp. 21 – 34, 2004. Advanced Techniques for Analysis of Geo-spatial Data.

[31] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.

[32] D. Liu, G. Nosovskiy, and O. Sourina, "Effective clustering and boundary detection algorithm based on delaunay triangulation," *Pattern Recognition Letters*, vol. 29, no. 9, pp. 1261 – 73, 2008/07/01.

[33] H. Edelsbrunner and E. Mucke, "Three-dimensional alpha shapes," *ACM Transactions on Graphics*, vol. 13, no. 1, pp. 43 – 72, 1994/01/.

[34] MySQL, "Gis and spatial extensions with mysql." http://dev.mysql.com/tech-resources/articles/4.1/gis-with-mysql.html, August 2010.

[35] PostGIS, "Using postgis: Data management and queries." http://postgis.refractions.net/docs/ch04.html, August 2010.

[36] H. Edelsbrunner, "Weighted alpha shapes," vol. UIUCDCS-R-92-1760, pp. 19 –, 1992/07/.