

Parallel Delaunay Triangulation in 3D and
Higher dimensions

Mantena V. Raju

May 14, 2003

Contents

1	Delaunay Triangulation in \mathbb{R}^d	1
1.1	Delaunay triangulation	1
1.1.1	Definition	2
1.1.2	Delaunay Triangulation and Voronoi diagrams	3
1.1.3	Delaunay Triangulation and Convex hulls	4
1.1.4	Optimality of Delaunay Triangulation in \mathbb{R}^d	6
1.2	Random Incremental Constructions	8
2	Constructing Delaunay triangulation in 3D	11
2.1	DeWall Algorithm	12
2.1.1	Construction of the simplex wall	14
2.2	InCoDe an incremental construction triangulator	18
2.3	Randomized incremental flip algorithm	20
3	Construction of Deluanay triangulation in \mathbb{R}^d	24
3.1	Flipping in \mathbb{R}^d	24
3.1.1	Regular triangulation	24
3.1.2	Lifting Regular triangulation	26
3.1.3	Definition of classification of flips	27
3.1.4	The Algorithm	27
3.2	Incremental construction using Delaunay Tree	29
3.2.1	Structure	30

3.2.2	Constructing the Delaunay triangulation	31
4	Parallel Delaunay triangulation in 3D	34
5	Results	37
5.1	Results	38
6	Conclusions	41

List of Figures

1.1	Delaunay triangulation in two dimensions.	2
1.2	Paraboloid of revolution and base hyperplane.	4
1.3	The Points of intersection of the paraboloid and hyperplane project onto a , b and c , which are on a circle on the base plane. Point below the hyperplane projects onto d inside the circle on the base plane.	5
1.4	The lower convex hull of the projected points and the Deluanay triangulation in a lower dimension.	6
2.1	Simplex wall is shaded in grey.	14
2.2	Triangulation of five points in general position in \mathbb{R}^3	21
3.1	Initialization of the delaunay tree.	29
3.2	Insertion of p into the Delaunay triangulation.	30
4.1	Partition of the triangulation space into four regions; the duplicated triangles are in grey.	35
5.1	Comparison of the run times of the algorithms.	38

List of Tables

5.1	Speed up of the parallel algorithm when compared to InCoDe . .	39
5.2	Duplication factor of the algorithm.	39
5.3	Number of simplices generated by the serial algorithms and the parallel algorithm for two and four processors.	39
5.4	Running times of the parallel algorithm on two processors in sec- onds.	40
5.5	Running times of the parallel algorithm on four processors in seconds.	40
5.6	Running times in seconds of the serial algorithms and the average time of the parallel algorithm on two and four processors	40

Abstract

Delaunay triangulation is a well known topic in Computational Geometry. The properties of Delaunay triangulation make it unique and interesting. It also has wide range of applications in mesh generation, volume visualization, molecular modeling etc. The aim of the thesis is to study the current 0 techniques for the construction of Delaunay triangulation of a point set in \mathbb{R}^3 and \mathbb{R}^d and study the possibility of parallelizing existing serial algorithms. The emphasis is on a practical parallel algorithm for triangulation in \mathbb{R}^d .

Chapter 1

Delaunay Triangulation in \mathbb{R}^d

This chapter defines Delaunay triangulation (DT), its relation to voronoi diagrams and convex hulls in higher dimensions. This chapter also describes the space complexity and properties of which make it unique among all the possible triangulations of a point set in \mathbb{R}^d . We also discuss the randomized incremental construction paradigm for geometric algorithms, which leads to simple and efficient algorithms for a wide range of geometric problems.

1.1 Delaunay triangulation

In this section we discuss Delaunay triangulation in \mathbb{R}^d , its relation to convex hulls and Voronoi diagrams, and also the optimality properties of Delaunay triangulation in d dimensions.

1.1.1 Definition

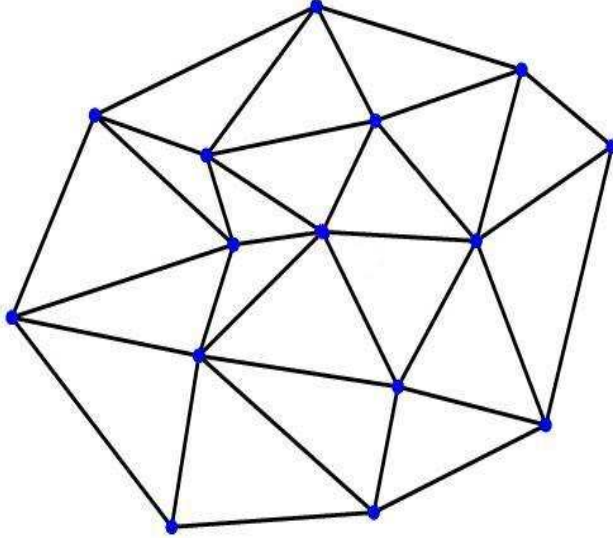


Figure 1.1: Delaunay triangulation in two dimensions.

Given a point set S in \mathbb{R}^d , a k -simplex, with $k \leq d$, is defined as a convex combination of $k + 1$ affinely independent points in S , called the vertices of the simplex (e.g. a triangle is a 2 -simplex and a tetrahedron is a 3 -simplex). An s -face of a simplex is the convex combination of a subset of $s + 1$ vertices of the simplex (i.e. a 2 -face is a triangular facet, 1 -face is an edge and 0 -face is a vertex).

A triangulation Σ of a point set S in \mathbb{R}^d is defined as the set of d -simplices such that

- A point s in \mathbb{R}^d is a vertex of a simplex σ in Σ iff $s \in S$.
- The intersection of two simplices in Σ is either empty or a common face.
- The set Σ is maximal: there does not exist any simplex σ that can be added to Σ without violating the previous rules.

A triangulation Σ is a *Delaunay triangulation* iff the hypersphere circumscribing each simplex does not contain any other point $s \in S$. The DT of a given point set S is unique if there does not exist in S $d + 2$ points lying on the same hypersphere. Such cases are known as degenerate cases. We will assume that the point set S is not degenerate when no $d + 2$ points in S lie on the same hypersphere. Such a configuration of the point set is known as general position. Delaunay triangulation for a set of points in 2-D is shown in Figure. 1.1.

1.1.2 Delaunay Triangulation and Voronoi diagrams

The Voronoi diagram of a set of sites partitions the Euclidean space \mathfrak{R}^d on the basis of the site closest to a particular point. Let S be a set of n point sites in \mathfrak{R}^d . For a nonempty subset R of S , the *Voronoi cell* of R , $V(R)$ is the set of all points of \mathfrak{R}^d equidistant from all sites in R and closer to every site in R than to any other site. The *Voronoi diagram* V is the collection of all nonempty Voronoi cells $V(R)$, for R a subset of S .

If $R \subset S$ and $V(R)$ be a nonempty Voronoi cell, then the *Delaunay Cell* $D(R)$ is the relative interior of the convex hull of R . The *Delaunay triangulation* is the collection of Delaunay cells $D(R)$, where R varies over subsets of S with $V(R)$ nonempty. If S is a set of n points in \mathfrak{R}^d with Voronoi diagram V and Delaunay triangulation D , then :

- V is a cell complex that partitions \mathfrak{R}^d .
- D is a triangulation of S .
- V and D are dual, i.e. for $R, R' \subseteq S$, $V(R)$ is a face of $V(R')$ iff $D(R')$ is a face of $D(R)$.
- If $R \subseteq S$, $V(R)$ is unbounded iff every site of R is on the boundary of the convex hull of S .

1.1.3 Delaunay Triangulation and Convex hulls

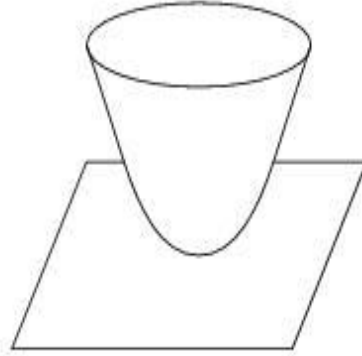


Figure 1.2: Paraboloid of revolution and base hyperplane.

The DT of a set S of point sites in dimension d can be obtained by appropriately lifting the sites to $d + 1$ dimensions and then projecting the lower convex hull of the lifted sites back to d dimensions. A point x in \mathfrak{R}^d is represented as $(x_1, x_2, x_3, \dots, x_d)$, the dot product of two points x, y is defined as $x_1y_1 + x_2y_2 + \dots + x_dy_d$. The lifting map $\gamma : \mathfrak{R}^d \rightarrow \mathfrak{R}^{d+1}$ is defined as $\gamma(x) = (x, x \cdot x)$ i.e. a point $x (x_1, x_2, x_3, \dots, x_d)$ in \mathfrak{R}^d is mapped to $(x_1, x_2, x_3, \dots, x_d, x_1^2 + x_2^2 + \dots + x_d^2)$ in \mathfrak{R}^{d+1} . The image of γ is

$$\Gamma = \{(x, z) : x \in \mathfrak{R}^d, z = x_1^2 + x_2^2 + \dots + x_d^2\}.$$

Γ is a paraboloid of revolution about the vertical line $x_{d+1} = 0$ and \mathfrak{R}^d is the *base hyperplane*. The paraboloid and base hyperplane are shown in Figure 1.2.

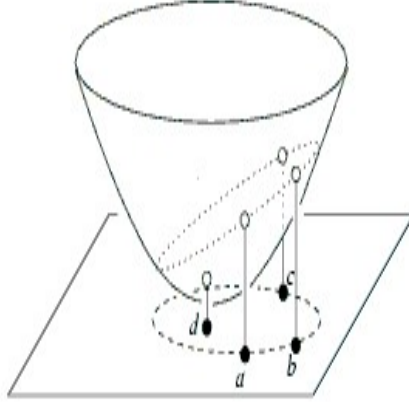


Figure 1.3: The Points of intersection of the paraboloid and hyperplane project onto \mathbf{a} , \mathbf{b} and \mathbf{c} , which are on a circle on the base plane. Point below the hyperplane projects onto \mathbf{d} inside the circle on the base plane.

A hyperplane P is *nonvertical* if it is not parallel to $x_{d+1} = 0$. If P is a nonvertical hyperplane, then any point in \mathfrak{R}^{d+1} is either above, on or below the hyperplane. If a nonvertical hyperplane P intersects Γ , then the projection $\Gamma \cap P$ in \mathfrak{R}^d is a hypersphere ρ . The portion of Γ above P projects outside the hypersphere ρ , the portion of Γ below P projects inside the hypersphere ρ [see Figure 1.3]. Suppose H is the convex hull of $\gamma(S)$, a face of H is a *lower face* if it has a nonvertical supporting plane P so that H lies above or on P . Equivalently, a lower face is visible to an observer positioned at negative infinity without looking through the interior of H .

Lemma 1.1 *Let H be the convex hull of $\gamma(S)$ and $D(S)$ the Delaunay triangulation of S , then $D(S)$ is the set of projections of lower faces of H onto the base hyperplane.*

Proof. Suppose F is a lower face of H , then F is the simplex of $\gamma(R)$ for some $R \subseteq S$. Since F is a lower face of H there exists a supporting plane P s.t. $\gamma(R) \subseteq P$ and $\gamma(S - R)$ is above the plane P . The projection of $\Gamma \cap P$ onto

the base hyperplane is a hypersphere ρ ; since $\gamma(S - R)$ is above P , $S - R$ is outside ρ . Hence the $\text{simplex}(R)$ is a Delaunay simplex, but $\text{simplex}(R)$ is the projection of face F onto the base hyperplane. See fig 1.4

Lemma 1.2 *Let H be the convex hull of n points in \mathfrak{R}^d . The number of k -faces and the number of incidences between k -faces and $(k+1)$ -faces of H is in $O(n^{\min\{\lfloor \frac{d}{2} \rfloor, k+1\}})$, for $0 \leq k \leq d-1$. These bounds are asymptotically tight.[11]*

From Lemma 1.1 and Lemma 1.2 it is clear that the combinatorial complexity of DT of n points in \mathfrak{R}^d is $O(n^{\lfloor \frac{d+1}{2} \rfloor})$.

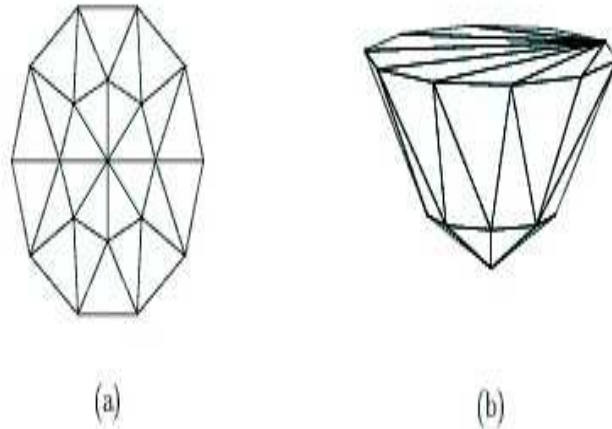


Figure 1.4: The lower convex hull of the projected points and the Delaunay triangulation in a lower dimension.

1.1.4 Optimality of Delaunay Triangulation in \mathfrak{R}^d

In this section, we discuss some of the known optimality properties of DT in \mathfrak{R}^d . In \mathfrak{R}^2 , DT is a wellstudied problem and many of its properties are known [3]. Of all the valid triangulations of a set of points in \mathfrak{R}^2 , the DT lexicographically maximizes the minimum angle, and also minimizes the maximum circumradii. If every triangle in the triangulation is non-obtuse then it is a DT.

In three and higher dimensions, very few results are known. In this section we describe some of the properties of DT in \mathbb{R}^2 that can be generalized to DT in \mathbb{R}^d . A simplex is said to be *self-centered* if the circumcenter of the simplex lies inside or on its boundary. In two dimensions, self-centered triangles are non-obtuse triangles. A *Min-Containment Sphere* of a simplex is the smallest sphere containing the simplex. We describe the following properties of DT in \mathbb{R}^d in detail.

- The maximum min-containment radius of DT of a point set in \mathbb{R}^d is less than the maximum min-containment radius of any other valid triangulation of the point set.
- If a valid triangulation of a point set consists of only self-centered simplices then it is the DT of the point set.

Let S_1, S_2, \dots, S_{d+1} be $d+1$ points that define a simplex T , consider the function $F(X)$ defined at every point X in the space :

$$\sum_{i=1}^{d+1} \lambda_i = 1 \text{ and } \sum_{i=1}^{d+1} \lambda_i S_i = X \quad (1)$$

$$F(X) = \sum_{i=1}^{d+1} \lambda_i (S_i - X)^2 = \sum_{i=1}^{d+1} \lambda_i S_i^2 - X^2 \quad (2)$$

The $(d+1)$ weights λ_i are uniquely determined by equations (1). The equation (2) defines the weighted distance to each of the vertices of the simplex. S_i^2 denotes the square of the norm of the point. For a point inside the simplex all λ_i are positive. This implies $F(X)$ is positive.

Lemma 1.3 *The $\text{Max}_X F(X)$ occurs at the circumcenter (X_C) of the simplex, the value of $F(X)$ at the point is equal to the circumradius (R^2) and the value of the function at any point X in space is given by $F(X) = R^2 - (X - X_C)^2$. (See[21] for proof.)*

Lemma 1.4 *The $\text{Max}_X F(X)$ constrained with $0 \leq \lambda_i \leq 1 \forall i$ occurs at the center of the min-containment sphere of the simplex and the maximum value is the square of its radius. (See [21] for proof.)*

Lemma 1.5 *Let $T_1 = [S_1, \dots, S_d, S_{d+1}]$ and $T_2 = [S_1, \dots, S_d, S_{d+2}]$ be two self-centered simplices that lie on opposite sides of common face $[S_1, \dots, S_d]$. Then the circumsphere of one does not contain the other. (See [21] for proof.)*

Let $S = \{S_1, \dots, S_n\}$ be a set of n points in \mathbb{R}^d . Consider the function $f(X)$ defined at every point X in the convex hull $H(S)$.

$$\lambda_i \geq 0 \quad \sum_{i=1}^{d+1} \lambda_i = 1 \quad \text{and} \quad \sum_{i=1}^{d+1} \lambda_i S_i = X \quad (4)$$

$$F(X, \lambda) = \sum_{i=1}^{d+1} \lambda_i (S_i - X)^2 = \sum_{i=1}^{d+1} \lambda_i S_i^2 - X^2 \quad (5)$$

$$f(X) = \text{Min}_\lambda F(X, \lambda) \quad (6)$$

Theorem 1.1 *At $\text{Min}_\lambda F(X, \lambda)$ for a fixed point X the only non-zero values of λ_i occur for the vertices of the Delaunay simplex containing the point X . (See [21] for proof.)*

Theorem 1.2 *The maximum min-containment radius of the Delaunay triangulation is less than the maximum min-containment radius of any other triangulation of the point set. (See [21] for proof.)*

Theorem 1.3 *If a valid triangulation of a point set consists of only self-centered triangles then it is the Delaunay triangulation of that point set. (See [21] for proof.)*

We will describe how to construct the Delaunay triangulation in \mathbb{R}^d in Chapter 3.

1.2 Random Incremental Constructions

Randomized Incremental Construction (RIC) [9],[10] is a powerful paradigm for geometric algorithms. In this section we discuss general theorems that help in the expected-case analysis of RIC. Let S be a set with $|S| = n$ elements. Let $F(S)$ be a multiset whose elements are nonempty subsets of S , and let b be

the size of the largest element of $F(S)$. The elements of $F(S)$ are called *regions* or *ranges*. If all the regions have size b , then $F(S)$ is said to be *uniform*. For a region $F \in F(S)$ and an object x , if $x \in F$, we say F *relies on* x or x *supports* F . For $R \subseteq S$, define $F(R) = \{F \in F(S) \mid F \subseteq R\}$. A *conflict relation* $C \subseteq S \times F(S)$ is defined between objects and regions. For all $x \in S$ and $F \in F(S)$, if $(x, F) \in C$, then F does not rely on x .

For a subset $R \subseteq S$, $F_0(R)$ denotes the set $F \in F(R)$ having no $x \in R$ with $(x, F) \in C$; i.e., $F_0(R)$ is the set of regions over R which do not conflict with any object in R . In the static case of the incremental construction, $F_0(R)$ for some subset $R \subseteq S$ is already available, a random element $x \in S - R$ is chosen, and $F_0(R \cup \{x\})$ is constructed from $F_0(R)$.

In case of dynamic construction, the history of the intermediate construction is maintained. Let (x_1, \dots, x_j) be a sequence of distinct elements of S , and R_j the set (x_1, \dots, x_j) . Let $R_0 = \{\}$, the empty set. The *history* $H = H(x_1, \dots, x_r)$ for the insertion sequence (x_1, \dots, x_r) is defined as $H = \bigcup_{1 \leq i \leq r} F_0(R_i)$. Let \prod_S be the set of permutations of S . For $\pi = (x_1, \dots, x_r) \in \prod_S$, $H_r(\pi)$ or H_r denotes the history $H(x_1, \dots, x_r)$.

For subset $R \subseteq S$, $r = |R|$, and distinct objects $x, y \in R$, let

$$deg(x, R) = |\{F \in F_0(R); x \text{ supports } F\}|$$

$$pdeg(x, y, R) = |\{F \in F_0(R); x, y \text{ supports } F\}|$$

$$c(R) = \frac{1}{r} \sum_{x \in R} deg(x, R)$$

$$p(R) = \frac{1}{r(r-1)} \sum_{x,y \in R} pdeg(x,y,R)$$

We call $deg(x,R)$ the *degree* of x in R , $pdeg(x,y,R)$ the *degree* of the ordered pair (x,y) in R , $c(R)$ the *average degree* of a random object in R and $p(R)$ the *average pair degree* of a random pair of objects in R . For a random variable X , $E[X]$ denotes the expectation of X . For integer r , $1 \leq r \leq n$, let

$$c_r = E[c(R)] = \sum_{R \subseteq S, |R|=r} \frac{c(R)}{\binom{n}{r}}$$

and

$$p_r = E[p(R)] = \sum_{R \subseteq S, |R|=r} \frac{p(R)}{\binom{n}{r}}$$

be the expected average degree and pair degree for random $R_r \subset S$, and let

$$f_r = \sum_{R \subseteq S, |R|=r} \frac{|F_0(R)|}{\binom{n}{r}}$$

be the expected number of conflict free regions of $F(S)$ with respect to random R_r . We describe the average case analysis of randomized incremental constructions.

Lemma 1.6 *The expectations c_r , p_r and f_r satisfy $c_r \leq b \frac{f_r}{r}$ and for $r > 1$, $p_r \leq b(b-1) \frac{f_r}{r(r-1)}$ with equality if $F(S)$ is uniform.*

Theorem 1.4 *Let C_r be the expected size of history H_r . Then $C_r = \sum_{j \leq r} c_j$.*

Theorem 1.5 *The expected number of regions in H_{r-1} which are in conflict with x_r is $-c_r + \sum_{j \leq r} p_j$.*

Theorem 1.6 *The expected number of elements in the conflict history is $-C_n + \sum_j (n-j+1)p_j$.*

Chapter 2

Constructing Delaunay triangulation in 3D

In this chapter, we discuss various methods used for the construction of DT in 3D and discuss their performance on various datasets. The duality between DT and Voronoi diagrams is well known and therefore algorithms are given for the construction of DT from Voronoi diagrams. In this chapter we discuss the direct construction methods which are employed to construct DT.

Direct DT algorithms [2] can be classified as

- *local improvement* : starting with an arbitrary triangulation, these algorithms locally modify the faces of pairs of adjacent simplices according to the circum-sphere criterion.
- *online* : starting with a simplex and then inserting points one at a time. The simplex containing the current point is partitioned by inserting it as a new vertex. The circum-sphere criterion is tested on all the simplices adjacent to the new ones, recursively, and if necessary, their faces are flipped.
- *incremental construction* : the DT is constructed by successively building

simplices whose circum-hyperspheres contain no points.

- *higher dimensional embedding* : the algorithms transform the data set to \mathfrak{R}^{d+1} space, then compute the convex hull of the transformed points and project the lower convex hull into \mathfrak{R}^d .
- *divide-and-conquer* : this is based on the recursive partition and local triangulation, and then on a merging phase where the resulting triangulations are merged.

Next we will discuss the space complexity of DT and the time complexity to construct it in 3D. The lifting map of section 1.1 connects DT in \mathfrak{R}^3 to lower convex hulls in \mathfrak{R}^4 . According to the upperbound theorem of convex polytopes[19] the vertices, edges, triangles and tetrahedra can be bounded. If the number of points in the input dataset are n , the bounds are

1. The number of vertices = n
2. The number of edges $\leq \frac{1}{2}(n^2 - n)$.
3. The number of triangles $\leq \frac{1}{2}(n^2 - 3n)$.
4. The number of tetrahedra $\leq \frac{1}{2}(n^2 - 3n - 2)$.

The space complexity of DT is $O(n^2)$. The worst case lower bound for running time of *online* methods [13] is $O(n^2)$. In the section that follow we discuss some of the algorithms used to compute DT.

2.1 DeWall Algorithm

The DeWall [7] algorithm is a divide-and-conquer algorithm to compute DT in 3D. The construction of DT in 2D[10],[16] first splits the input data set P into two subsets P_1 and P_2 ; construct local triangulation \sum_1 and \sum_2 recursively and merge the partial results to build the solution \sum . DeWall approaches the problem in a different fashion. Instead of merging partial results, a more

complex dividing phase is applied which partitions the point set and builds, as first step, the merging triangulation. The algorithm is then recursively applied to triangulate the two subsets of the input dataset.

The splitting plane α divides the point set P into two subsets P_1 and P_2 , divides the triangulation Σ into Σ_1 , Σ_2 and Σ_α ; where Σ_α is the set of simplices intersecting the plane α . Σ_α is called the *simplex wall* and Σ_1 , Σ_2 are contained totally in the half spaces defined by α . The triangulation Σ can be divided into disjoint subsets Σ_1 , Σ_2 and Σ_α . The DeWall (Delaunay Wall) algorithm, consists of the following three steps:

- select the dividing plane α , split P into the two subsets P_1 and P_2 and construct Σ_α .
- starting from Σ , recursively apply DeWall on P_1 and P_2 to build Σ_1 and Σ_2 .
- return the union of Σ_α , Σ_1 and Σ_2 .

We will discuss the steps discussed above in more detail.

2.1.1 Construction of the simplex wall

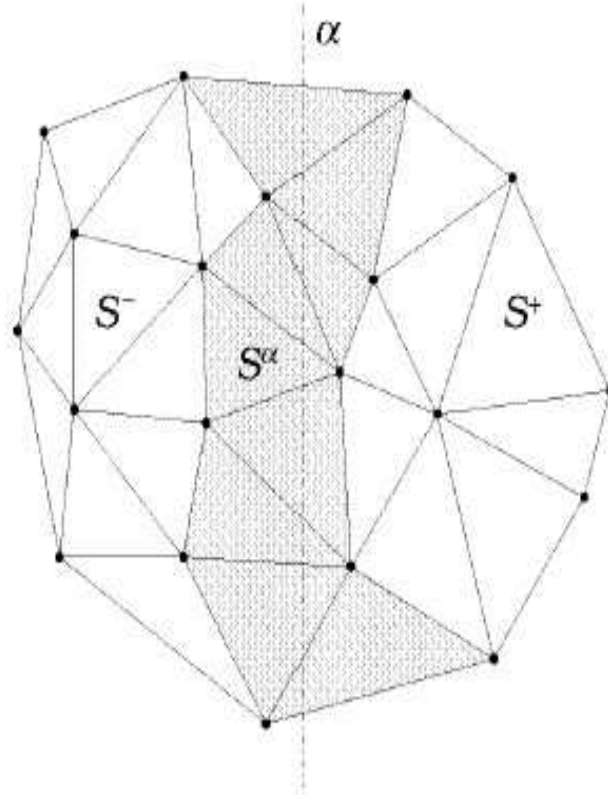


Figure 2.1: Simplex wall is shaded in grey.

The simplex wall is built using an *incremental construction* approach. A starting simplex is built and then other simplices of Σ are built by adding a new simplex at each step and without modifying the current triangulation. If we consider a triangle face f , which does not lie on the $ConvexHull(P)$, it is shared by exactly two simplices belonging to Σ .

The algorithm starts by constructing the initial tetrahedron σ_i intersecting the plane α . Each triangle face of σ_i is processed to build the tetrahedron adjacent to them (if it exists, i.e. the face does not belong to the convex hull of P) and added to the current list of tetrahedra in Σ . All of the new triangle faces of

the new tetrahedron built are used to update a structure called **Active Face List (AFL)**. Update of the AFL is as follows: if a new triangle face is already contained in the AFL, then it is removed from the AFL; otherwise, it is inserted in AFL because its adjacent tetrahedron has not yet been built. This process continues iteratively (extract a triangle face f from AFL, build the tetrahedron σ adjacent to the triangle face f , update the AFL with the new triangle faces of σ and then again extract another face from AFL) until the AFL is empty (see Figure 2.1).

The generalized algorithm described above, needs to specialize for the construction of the simplex wall, construction of the first tetrahedron, construction of a tetrahedron from a triangle face extracted from AFL. The algorithm is specified in pseudocode below.

```

Function DeWall (P : pointset, AFL : face_list) : tetrahedra_list;

    var f : face; AFL $\alpha$ , AFL $_1$ , AFL $_2$ : face_list;

        t : tetrahedra;  $\Sigma$  : tetrahedra_list;  $\alpha$  : splitting_plane

    begin

        AFL $\alpha$ , AFL $_1$ , AFL $_2$  := emptylist;

        Pointset_Partition (P,  $\alpha$ , P $_1$ , P $_2$ );

        /* Simplex Wall Construction */

        if AFL =  $\phi$  then

            t := MakeFirstTetra (P);

            Insert (t,  $\Sigma$ ); AFL := faces (t);

        for each f  $\in$  AFL do

            if IsIntersected (f,  $\alpha$ ) then Insert (f, AFL $\alpha$ )

            if Vertices (f)  $\subseteq$  P $_1$  then Insert (f, AFL $_1$ )

```

```

    if Vertices ( $f$ )  $\subseteq$   $P_2$  then Insert ( $f$ ,  $AFL_2$ )
while notempty ( $AFL_\alpha$ ) do
     $f :=$  Extract ( $AFL_\alpha$ );
     $t :=$  MakeTetra ( $f$ ,  $P$ );
    if  $t \neq null$  then
        Insert ( $t$ ,  $\Sigma$ );
        for each  $f$ :  $f \in \text{faces}(t)$  AND  $f \neq f$  do
            if IsIntersected ( $f$ ,  $\alpha$ ) then Update ( $f$ ,  $AFL_\alpha$ )
            if Vertices ( $f$ )  $\subseteq$   $P_1$  then Update ( $f$ ,  $AFL_1$ )
            if Vertices ( $f$ )  $\subseteq$   $P_2$  then Update ( $f$ ,  $AFL_2$ )
/* Recursive Triangulation */
    if  $AFL_1 \neq \phi$  then  $\Sigma := \Sigma \in DeWall(P_1, AFL_1)$ 
    if  $AFL_2 \neq \phi$  then  $\Sigma := \Sigma \in DeWall(P_2, AFL_2)$ 
    DeWall :=  $\Sigma$ ;
end;

```

Construction of the first tetrahedron

First, the function *MakeFirstSimplex* produces a Delaunay tetrahedron intersecting the plane α , in order to start from this tetrahedron the incremental construction of the simplex wall Σ_α . *MakeFirstSimplex* selects the point $p_1 \in P$ nearest to the plane α . Next it selects a second point p_2 such that p_2 is the nearest point to p_1 on the otherside of α . Then it searches the point p_3 such that the circumcircle around 1-face (p_1, p_2) and the point p_3 has the minimum radius; (p_1, p_2, p_3) is therefore a 2-face of Σ . This process continues until the first tetrahedron is built.

Construction of the generic tetrahedron

Given a face f , the function *MakeSimplex* builds the adjacent simplex by applying the DT definition. For each point $p \in P$, *MakeSimplex* computes the radius of the sphere which circumscribes p and the face f . We choose the point p which minimizes this radius to build the tetrahedron adjacent to f .

MakeSimplex selects a point $p \in P$ which minimizes the function dd (Delaunay distance):

$$dd(f, p) = \begin{cases} r & \text{if } c \in \text{Halfspace}(f, p) \\ -r & \text{if } c \notin \text{Halfspace}(f, p) \end{cases}$$

with r and c the radius and the center of the circumsphere around f and p ; given the plane in which f lies, $\text{Halfspace}(f, p)$ returns the halfspace which contains the new tetrahedra.

Construction of tetrahedra in \sum_α alone

A slight modification is required to the incremental approach to construct \sum_α . Instead of using a single list of active faces (AFL), the algorithm uses three disjoint lists containing:

- AFL_α : the triangle faces intersecting the plane α .
- AFL_1 : the triangle faces with all vertices in P_1 .
- AFL_2 : the triangle faces with all vertices in P_2 .

For each tetrahedron σ , the algorithm inserts its triangle faces in the suitable face list. It then extracts faces from the AFL_α alone; this ensures that each simplex built is part of the simplex wall \sum_α . The simplex wall construction process terminates when AFL_α is empty. This process returns both \sum_α and the pair of active face lists AFL_1 and AFL_2 . DeWall is then recursively applied to the pairs (P_1, AFL_1) and (P_2, AFL_2) .

DeWall Complexity and Performance Efficiency

The DeWall algorithm is not optimal in asymptotic time complexity nor in practical efficiency. An analysis of the algorithm shows that the main inefficiency is in the *MakeSimplex* method. For each tetrahedra constructed from the adjacent triangle face every point is checked for the *dd* metric. This search entails performing an $O(n)$ test for each simplex built. This implies that the running time of the algorithm is $O(n^3)$.

The space requirements are bounded by the space complexity of :

1. the pointset P
2. the active face list $O(n^2)$ (given in the previous section).
3. the outcoming triangulation, (like the incremental construction algorithms, however, DeWall can return each tetrahedron as soon as it is built).

The space complexity of the algorithm is $O(n^2)$.

The main bottlenecks for the algorithm are the operations of AFL and searching for the *dd-minimum* point in the *MakeSimplex* method. Hash tables can be used to improve the performance of the operations on AFL. To improve the performance of *MakeSimplex*, uniform grid technique[1] can be used to speedup the location of *dd-minimum* point. The point set is distributed in the uniform grid and the cells closer to the face f can be checked for the *dd-minimum* point instead of checking all the points.

2.2 InCoDe an incremental construction triangulator

In this section, we discuss the **InCoDe**(Incremental Construction of Delaunay triangulation)[6]. Some of the construction steps in the algorithm are similar to

the DeWall algorithm described in the previous section. The algorithm starts from a tetrahedron σ in Σ and it incrementally builds Σ by adding a new tetrahedron in each step.

A triangle face f , which does not lie on the $ConvexHull(P)$ is shared by exactly two tetrahedra belonging to Σ . In each step, the tetrahedron lying on the face of one of the previously computed tetrahedra is built and added to the current triangulation Σ . All the new faces of the tetrahedron are used to update an **Active Face List (AFL)**. The AFL is updated as follows: if the new face f is already in the AFL, then the face is removed from the AFL; otherwise f is inserted into AFL, because only one of the two tetrahedra sharing the face has not yet been built.

The algorithm starts by constructing a first valid Delaunay tetrahedron σ_i over the pointset P . Then, Σ is initialized with σ_i and all the triangle faces of σ_i are inserted into AFL. Now extract a face f from AFL, build the tetrahedron σ adjacent to f , insert the triangle faces of σ into AFL, extract a new face f' from AFL and build the tetrahedron adjacent to f' . Repeat this until AFL is empty. The algorithm is specified in pseudo code below.

```

Function InCoDe (P : pointset) : tetrahedra_list;

    var f : face; AFL : face_list;

        t : tetrahedra;  $\Sigma$  : tetrahedra_list;

    begin

        AFL := emptylist;

        t := MakeFirstTetra (P);

        Insert (t,  $\Sigma$ );

        AFL := faces (t);

        while notempty (AFL) do

```

```

f := Extract (AFL);
t := MakeTetra (f, P);
if t ≠ null then
    Insert (t,  $\Sigma$ );
    for each f' f' ∈ faces (t) AND f' ≠ f do
        Update (f', AFL);
InCoDe :=  $\Sigma$ ;
end;

```

MakeFirstTetra Function

MakeFirstTetra selects a point $p_1 \in P$ randomly, then selects a second point p_2 such that p_2 is the nearest point to p_1 . Then it searches the point p_3 such that the circumcircle around 1-face (p_1, p_2) and the point p_3 has the minimum radius; (p_1, p_2, p_3) is therefore a 2-face of Σ . This process continues until the first tetrahedron is built. The function *MakeSimplex* is the same as described in the previous section. The operations on AFL are handled in the same fashion described in the previous section.

The space complexity of the algorithm is $O(n^2)$. The runtime complexity is $O(n^3)$. The speed up techniques for the algorithm are the same as the once described in the previous section.

2.3 Randomized incremental flip algorithm

The basic idea of the incremental flip[19] algorithm is the following. Let P be a set of n points in \mathfrak{R}^3 . Let $4 < i \leq n$ and assume the Delaunay triangulation of the first $i - 1$ points in P is already constructed; call it \mathfrak{S}_{i-1} . Add the i -th point $p_i \in P$ to the triangulation, and restore the Delaunay hood by flipping; this results in \mathfrak{S}_i . Repeat this process until $i = n$. We describe some definitions required to describe the algorithm.

Local Delaunayhood

A triangle $\sigma_T = f_{ijk}$, formed by $T = \{p_i, p_j, p_k\}$ is called *locally Delaunay* if either

- $\sigma_T \in \text{conv}(P)$, or
- σ_T is the shared facet of two tetrahedra σ_{T_1} and σ_{T_2} , with $T_1 = T \cup \{p_u\}$ and $T_2 = T \cup \{p_v\}$, and p_v lies outside the circumsphere of σ_{T_1} .

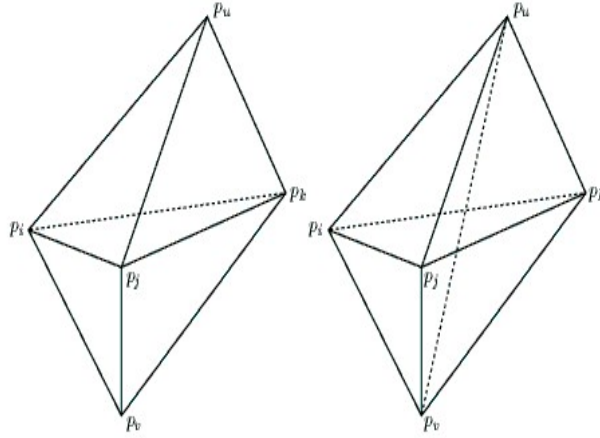


Figure 2.2: Triangulation of five points in general position in \mathfrak{R}^3 .

Flips in \mathfrak{R}^3

Consider a set of five points P_1 in general position. If there exists a point $p \in P_1$ such that $\text{conv}(P_1) = \text{conv}(P_1 - \{p\})$, then P_1 has only one triangulation with four tetrahedra, all incident to p . The triangulation is the Delaunay triangulation of P_1 . If all the points lie on the boundary of $\text{conv}(P_1)$ then there are two possible triangulations, one with two tetrahedra, and one of them with three tetrahedra. One of them has to be the Delaunay triangulation. See figure 2.2.

We consider the lifted pointset P_U of P_1 . The projection of the lower boundary of convex hull of P_U is the closest-point Delaunay triangulation of P_1 , the projection of the upper boundary of convex hull of P_U is the farthest-point Delaunay triangulation of P_1 . The two triangulation of P_1 correspond to one of the triangulations described above. A *flip* is the operation that replaces the triangulation that corresponds to the upper boundary by the one that corresponds to the lower boundary.

The algorithm

The algorithm is described in pseudo-code below. Let (p_1, p_2, \dots, p_n) be a random sequence of points of P , and let Σ denote the evolving triangulation. Initially Σ consists of the tetrahedron of the first four points and its faces. The main loop incrementally adds points p_i to Σ .

procedure `delaunay`(P, n)

 let (p_1, p_2, \dots, p_n) be a random sequence of points of P ;

 initially, Σ consists of the tetrahedron of the first four points
and its faces;

for $i \leftarrow 5, \dots, n$ **do**

assert $\Sigma = \mathfrak{S}_{i-1}$;

$a \leftarrow \text{search}(p_i)$;

$\text{collect}(p_i, a, a', \ell)$;

$\text{connect}(p_i, a', \ell)$;

$\text{flip}(p_i, \ell)$;

end for;

assert $\Sigma = \mathfrak{S}$;

end

The triangulation Σ is stored using a triangle-edge[19],[15]. Function **search**(p_i) implements the point location step returning a triangle facet $a \in \Sigma$ visible from p_i . If p_i lies outside Σ , then a will lie on the boundary of Σ , otherwise a is a triangle facet of a tetrahedron which contains p_i . The function **collect**(p_i, a, a', ℓ); collects all the visible faces of Σ visible from p_i and stored in list ℓ . The function **connect**(p_i, a', ℓ); adds point p_i to Σ by connecting the facets of ℓ to p_i . The function **flip**(p_i, ℓ) restores the Delaunayhood of Σ . In the next chapter we discuss the construction of Delaunay triangulation in \mathfrak{R}^d .

Chapter 3

Construction of Delaunay triangulation in \mathbb{R}^d

The current design paradigms for the construction of the Delaunay triangulation in \mathbb{R}^d constitute flipping, random incremental construction, and from higher dimensional convex hulls. In this chapter we describe flipping and random incremental construction methods for constructing Delaunay triangulation.

3.1 Flipping in \mathbb{R}^d

In this section we describe the flipping algorithm [13] for a generalized triangulation called *regular triangulation*. Delaunay triangulation is a special case of regular triangulation. Some definitions required to describe regular triangulation are stated below.

3.1.1 Regular triangulation

We now describe a dual of regular triangulation known as Power diagrams.

Power distance and power diagrams.

Let S be a finite set of points in \mathfrak{R}^d , and assign a real valued weight w_p to each point $p \in S$. For each p , define $\Pi_p : \mathfrak{R}^d \rightarrow \mathfrak{R}$ so that

$$\Pi_p(x) = |xp|^2 - w_p$$

where $|xp|$ is the Euclidean distance between points x and p . $\Pi_p(x)$ is called the *power distance* of x from p . For points $p, q \in S$, the locus of points $x \in \mathfrak{R}^d$ with $\Pi_p(x) = \Pi_q(x)$ is the hyperplane

$$N_{p,q} : 2 \sum_{i=1}^d x_i(q_i - p_i) + \sum_{i=1}^d (p_i^2 - q_i^2) - w_p + w_q = 0$$

Let $H_{p,q}$ denote the halfspace of points $x \in \mathfrak{R}^d$ for which $\Pi_p(x) \leq \Pi_q(x)$. For each $p \in S$, define its *power cell* as

$$P(p) = \bigcap_{q \in S - \{p\}} H_{p,q}$$

The collection of power cells for all $p \in S$ and their faces defines the cell complex $P(S)$ known as the *power diagram* of S .

Orthogonality and general position of Regular triangulation

We assume that the weighted points of S are in general position. General position in the context of regular triangulation, means that for every $d + 1$ weighted points in S , there is a unique unweighted point $x \in \mathfrak{R}^d$ with the same power distance from all the $d + 1$ points, and for every $d + 2$ weighted points of S , there is no such point. Two weighted points p, q are said to be *orthogonal* if

$$|pq|^2 = w_p + w_q$$

A subset T of $d + 1$ weighted points of S defines a unique d -simplex Δ . There is a unique weighted point z that is orthogonal to all points $p \in T$. We call z the *orthogonal center* of Δ . If the weights of all $p \in T$ are zero then z is the circumcenter of Δ .

Local and global regularity

$\Pi_z(p) = w_p$ for all $p \in T$. We call Δ *globally regular* if for all $q \in S - T$, $\Pi_z(q) > w_q$. The regular d -simplices, together with their faces, define a simplicial cell complex known as the *regular triangulation* of S , denoted by $R(S)$. $R(S)$ and $P(S)$ are dual to each other. If the weights of all points in S are zero, then $P(S) = V(S)$ and $R(S) = D(S)$.

Consider an arbitrary triangulation Σ of S . Let $\Delta = \Delta_U$ be a $(d - 1)$ -simplex, $\Delta' = \Delta_{U \cup \{a\}}$ and $\Delta'' = \Delta_{U \cup \{b\}}$ be the two incident d -simplices Δ . Let z' be the orthogonal center of Δ' . Then Δ is *locally regular* in Σ if $w_b < \Pi_{z'}(b)$.

3.1.2 Lifting Regular triangulation

This section describes the relation between regular triangulations in \mathfrak{R}^d and convex hulls in \mathfrak{R}^{d+1} . For a point $p = (p_1, p_2, \dots, p_d) \in \mathfrak{R}^d$ with weight w_p , define its lifted point

$$p^+ = (p_1, p_2, \dots, p_d, p_{d+1}) \in \mathfrak{R}^{d+1}$$

where $p_{d+1} = \sum_{i=1}^d p_i^2 - w_p$. For each set $S \subseteq \mathfrak{R}^d$, define $S^+ = \{p^+ | p \in S\}$.

Lemma 3.1 *Let S be a finite set of weighted points in \mathfrak{R}^d . The vertical projection of the lower facets of $\text{conv}(S^+)$ into \mathfrak{R}^d gives the d -simplices of $R(S)$.*

3.1.3 Definition of classification of flips

Consider a set T of $d+2$ points in \mathbb{R}^d . There are exactly two ways to triangulate T . The two ways correspond to the two sides (lower and upper) of the $(d+1)$ -simplex that is the convex hull of the corresponding lifted points in \mathbb{R}^{d+1} . A flip is the operation which replaces one triangulation of T with each other. Given $d+2$ weighted points in \mathbb{R}^d , one of the two triangulations is the regular triangulation of the points and the other is not regular. In the construction of $R(S)$, flips are applied in this directional sense, replacing a non-regular triangulation of $d+2$ points by the regular one.

Flippability

Let $\Delta = \Delta_U$ be a $(d-1)$ -simplex of an arbitrary triangulation Σ of S , and let $\Delta' = \Delta_{U \cup \{a\}}$ and $\Delta'' = \Delta_{U \cup \{b\}}$ be the two incident d -simplices. The *induced subcomplex* of $T = U \cup \{a, b\}$ consists of all simplices in T spanned by points in T . We call T (and Δ) *flippable* if $\text{conv}(T)$ is the underlying space of the induced subcomplex of T .

3.1.4 The Algorithm

The algorithm constructs the regular triangulation of a given set $S = \{p_1, p_2, \dots, p_n\}$ of weighted points incrementally, that is, points are added one at a time. An artificial simplex $\Delta_{S_0} = \text{conv}(S_0)$ with $S_0 = \{p_{-d}, \dots, p_0\}$, so that S is inside Δ_{S_0} . The $d+1$ artificial points can be conveniently chosen at infinity. For example, set $w_p = \theta$, and

$$p_{ij} = \begin{cases} 0 & \text{if } -i > j \\ +\infty & \text{if } -i = j \\ -\infty & \text{if } -i < j \end{cases}$$

where p_{ij} denotes the j -th coordinate of p_i , for $-d \leq i \leq 0$. The number ∞ is the place holder for a large enough number.

Global algorithm

Define $S_i = \{p_{-d}, p_{-d+1}, \dots, p_i\}$, given $R(S_{i-1})$ and let Δ be the d -simplex containing p_i . If Δ is still regular then $R(S_i) = R(S_{i-1})$. Otherwise flip $T \cup \{p_i\}$. This is a flip of the type "1 to $d + 1$ " [13]. Continue flipping locally non-regular $(d - 1)$ -simplices until none remain. The resulting triangulation is $R(S_i)$. A $(d - 1)$ -simplex Δ_U of a triangulation belongs to the *link* of the vertex p_i and if $\Delta_{U \cup \{a\}}$ is a d -simplex of the triangulation. The $(d - 1)$ -simplices of the link p_i are called *link facets*. The algorithm is described below.

Construct $R(S_0) = \Delta_{S_0}$;

for $i \leftarrow 1, \dots, n$ **do**

locate the d -simplex Δ_T in $R(S_{i-1})$ that contains p_i

if $R(T \cup \{p_i\}) \neq \Delta_T$ **then**

flip $T \cup \{p_i\}$;

while there exist locally non-regular link facets **do**

find a locally non-regular link facet Δ that is flippable;

flip Δ

endwhile

endif

endfor

Point location is performed by storing the history of flips that are performed. This is done by maintaining the collection of discarded d -simplices in a directed acyclic graph called the *history dag* [13]. A randomized analysis of the incremental algorithm results in a total expected time of $O(n \log n + n^{\lceil \frac{d}{2} \rceil})$.

3.2 Incremental construction using Delaunay Tree

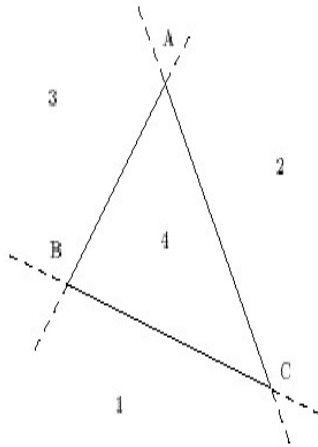


Figure 3.1: Initialization of the delaunay tree.

The Delaunay tree [23] is a hierarchical data structure which is defined from Delaunay triangulation. It allows an online or semi-dynamic construction of the Delaunay triangulation of a finite set of n points in any dimension. Each point is inserted into the triangulation as it is incrementally built. The point to be inserted is located in the current triangulation by the maintaining the *history* of construction, the successive versions of triangulations, linked together in the tree.

3.2.1 Structure

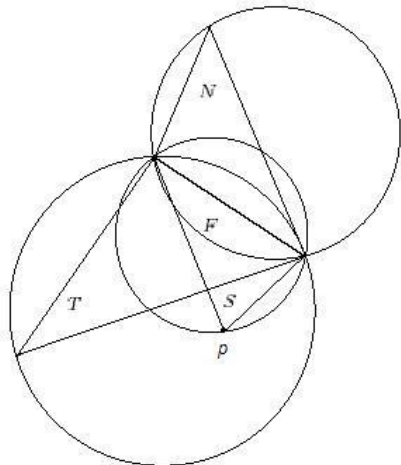


Figure 3.2: Insertion of p into the Delaunay triangulation.

The nodes of the tree are associated with the successive simplices of the triangulation. The word *simplex* denotes the geometric object as well as the corresponding node. For the initialization step $d + 1$ points are chosen. They generate one finite simplex and $d + 1$ *infinite* ones (see Figure 3.1). A halfspace is also called a simplex, and will be considered as having d finite vertices and one vertex at infinity. These $d + 2$ simplices will be the sons of the root of the tree. Then the other points are inserted one after the other building the triangulation.

A site p is in conflict with a simplex if p lies in the interior of its circumscribing hypersphere. After the insertion of site p , the simplices in conflict with p disappear from the triangulation, but remain in the Delaunay tree. They are called *dead* and p is their *killer*. Some of those simplices have a facet on the boundary $F(p)$ of the region $R(p)$ formed by the simplices in conflict with p . Let T be one of the simplices in conflict with p that has a facet F belonging to $F(p)$. We construct a new simplex S , created by p , having vertex p and facet F . Let N be the simplex sharing facet F with T before the insertion of

p . Because the triangulation is a Delaunay one, we have the following property (see Figure 3.2).

Property 3.1 The circumscribing hypersphere of S is included in the union of two balls circumscribing T and N .

The newly created simplex S will be called *son* of T and *stepson* of N through facet F . These edges will never be modified during the construction. When it is killed, a simplex receives from 0 (if it has no facet on $F(p)$) to $d + 1$ sons. As long as it is not dead it can receive any number of stepsons. When a new point p' in conflict with S and not with N , S will be killed and its son S' having vertex p' and facet F will be another stepson of N . Thus a node has at most one son and one list of stepsons through each facet, i.e. 0 to $d + 1$ sons and 0 to $d + 1$ stepsons.

This hierarchical structure is called a *Delaunay tree*, but it is more exactly a rooted direct acyclic graph. A simplex of the current triangulation is not dead, and so corresponds to a node having no son, but possibly stepsons.

3.2.2 Constructing the Delaunay triangulation

Let p be a site to be introduced in the triangulation. Two steps are performed, first we locate p in order to find the set $R(p)$ of all the simplices in conflict with p and then we create the new simplices.

Location

If p is in conflict with a simplex T , then p is in conflict with the father or stepfather of T (or both of them). So all the simplices which are killed by p can be found by recursively exploring the Delaunay tree. Procedure location is described below.

Procedure location(p, T)

```

if  $T$  has not been visited yet and  $p$  is in conflict with  $T$  then
    for each stepson  $S$  of  $T$  location( $p,S$ );
    for each son  $S$  of  $T$  location( $p,S$ );
    if  $T$  is not dead then
        mark  $T$  killed by  $p$ ;
        add  $T$  to the list  $R(p)$  of the killed simplices.

```

Creating new simplices

Go through the list $R(p)$ of the killed simplices. A facet F of a simplex T is on $F(p)$ if the simplex N neighbor of T through F is not killed. In this case, we create the simplex S with vertex p and facet F , and the two edges between S , its father T and its stepfather N . Moreover N and S are neighbors through F . The procedure is described below.

```

Procedure creation( $p$ );
    for each simplex  $T$  killed by  $p$ 
        for each neighbour  $N$  of  $T$  through a facet  $F$ 
            if  $p$  is not in conflict with  $N$  then
                create the simplex  $S$  having vertex  $p$  and facet  $F$ 
                replace the adjacency relation between  $N$  and  $T$ 
                by the adjacency relation between  $N$  and  $S$ 
                create the relations:  $S$  son of  $T$  and  $S$  stepson of  $N$ 
                through facet  $F$ ;
            create the adjacency relationships between the new simplices.

```

The analysis of the algorithm leads to the following result.

The Delaunay triangulation of n points in d -dimensions can be computed online with $O(n^{\lfloor \frac{d+1}{2} \rfloor})$ expected space in dimension $d \geq 2$.

The expected update time for an insertion is $O(\log n)$ if $d = 2$ and $O(n^{\lfloor \frac{d+1}{2} \rfloor - 1})$ if $d \geq 3$.

Chapter 4

Parallel Delaunay triangulation in 3D

In this chapter, we discuss the parallelization of Delaunay triangulation on m independent asynchronous processors. An earlier paper[6] describes parallelizing the divide-and-conquer(D&C) DeWall algorithm described in the previous chapter. A usual D&C algorithm requires two synchronization points: the first at the partitioning of input data set before process splitting, and the second at the merge time, when split processes terminate. In case of the DeWall algorithm, merging is not required, and the subprocesses can then run asynchronously. Though this algorithm has a simpler implementation, the problem with this algorithm is limited scalability and a complex splitting phase. We describe a more scalable algorithm to compute the Delaunay triangulation in \mathfrak{R}^3 .

A spatial decomposition approach

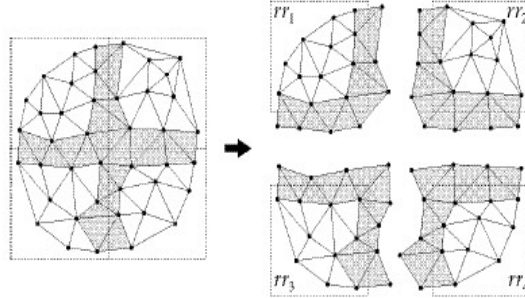


Figure 4.1: Partition of the triangulation space into four regions; the duplicated triangles are in grey.

The incremental Delaunay triangulator can be parallelized on m independent asynchronous processors by means of spatial subdivision. The spatial subdivision is not of the input dataset P , but of the triangulation to be computed. The bounding box of the pointset P is divided into m rectangular regions, called rr_i for $1 \leq i < m$. The same InCoDe process is replicated on each processor, together with the full pointset P because the triangulation contained in rr_i is also dependent on points outside rr_i . A snapshot of the algorithm in two dimensions can be seen in fig 4.1. The pseudocode specification of the parallel InCoDe is described below.

Function ParaInCoDe (P :pointset, rr_i :rect_reg) : tetrahedra_list;

var f : face; AFL : face_list;

t : tetrahedra; Σ : tetrahedra_list;

begin

 AFL := emptylist;

t := MakeFirstTetra (P, rr_i);

 Insert (t, Σ);

 AFL := faces (t);

```

while notempty (AFL) do

  f := Extract (AFL);

  t := MakeTetra (f, P);

  if t  $\neq$  null then

    if ULFVertex(t)  $\in$  rri then Insert (t,  $\Sigma$ );

    for each f': f'  $\in$  faces (t) AND f'  $\neq$  f AND f'  $\cap$  rri  $\neq$   $\emptyset$ 
  do

    Update (f', AFL);

  InCoDe :=  $\Sigma$ ;

end;

```

The main difference from the sequential version is the management of the AFL list. For each new tetrahedron, we only insert into AFL the faces completely or partially contained in the local region rr_i . Each processing element computes the tetrahedra within rr_i and those shared with adjacent regions. The tetrahedra shared among adjacent regions are duplicated into each region. This can be avoided with a vertex containment test. The tetrahedron is included if the apex of the tetrahedron is contained in the local rr_i region. We compare the results of the serial algorithms and the parallel algorithm in the next chapter.

Chapter 5

Results

In this chapter we compare the results of the serial algorithms described in chapter two and parallel algorithm in chapter four. We compare the results on eight datasets. The parallel algorithm has been implemented using MPI and C. The point datasets are uniform point distributions. We compare the parallel algorithm with

- *Detri* a randomized incremental algorithm implemented for alpha shapes for E. Mucke.
- *DeWall* and *InCoDe* algorithm implemented by P. Cignoni

The results of the parallel algorithm are collected for two and four processors. The results of the algorithms have been collected on a nine-processor Ultra Sparc cluster; the clock speed of the processors in the cluster is 168MHz. The results of the serial algorithms along with the average running time of the parallel algorithm are also collected. *Detri* outperforms *InCoDe* and *DeWall* on most of the test cases. The parallel algorithm on two processors does not outperform *Detri* but definitely fares better than *DeWall* and *InCoDe*. The parallel algorithm on four processors outperforms all the serial algorithms (see Tables 5.3, 5.4 and 5.5). The comparison of the runtimes is shown in the graph shown in Figure 5.1.

5.1 Results

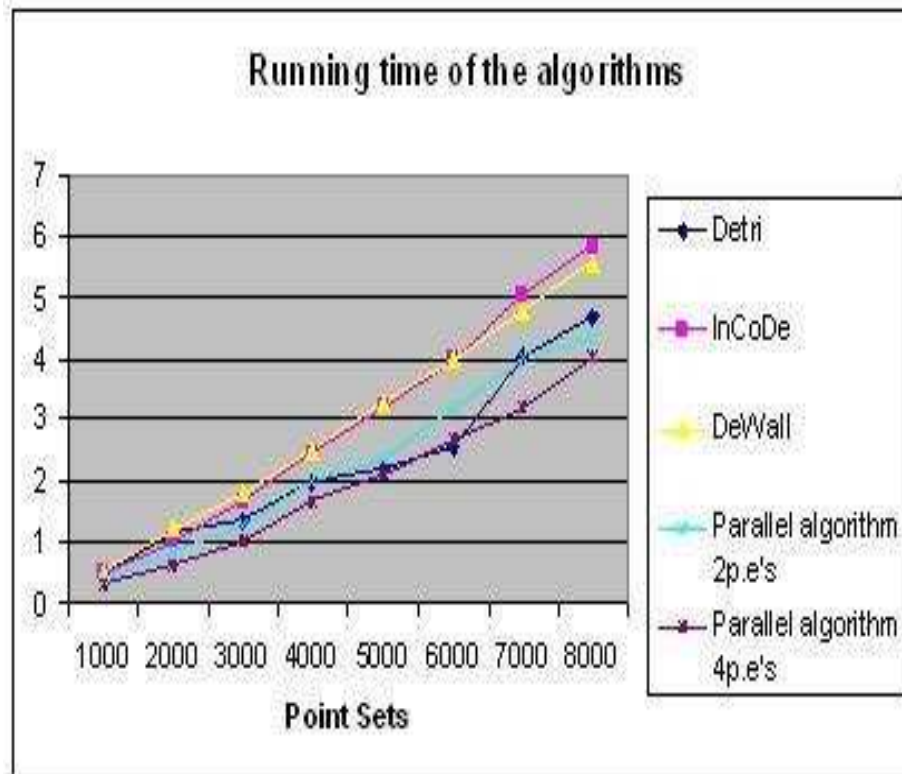


Figure 5.1: Comparison of the run times of the algorithms.

The point set is split into regions such that each region has roughly equal number of points. The processing time on individual processing elements does not differ by much shows that for uniform datasets the load on the processors is pretty well balanced. The drawback of the parallel algorithm is that it recomputes some of the tetrahedra that fall across regions. The duplication factor is also calculated to get an estimate on the number of tetrahedra recomputed. The duplication factor calculated in table 5.2 seems to increase as the numbers of processors on which the parallel algorithm is running. The speedup of the

parallel algorithm with respect to InCoDe algorithm is reported in table 5.1

Speed up	1000	2000	3000	4000	5000	6000	7000	8000
Parallel algorithm on 2 p.e.'s	1.25	1.21	1.30	1.2	1.36	1.24	1.15	1.35
Parallel algorithm on 4 p.e.'s	1.63	1.68	1.63	1.45	1.52	1.48	1.56	1.45

Table 5.1: Speed up of the parallel algorithm when compared to InCoDe

Duplication factor	1000	2000	3000	4000	5000	6000	7000	8000
Parallel algorithm on 2 p.e.'s	5.6%	6.5%	6.3%	3.5%	4.5%	5.39%	5.6%	5.27%
Parallel algorithm on 4 p.e.'s	8.17%	6.85%	7.18%	8.06%	8.2%	7.68%	8.12%	6.65%

Table 5.2: Duplication factor of the algorithm.

Simplex Count	1000	2000	3000	4000	5000	6000	7000	8000
Serial algorithm	6140	12654	19139	25287	32370	39130	45677	52305
Parallel algorithm on 2 p.e.'s	6486	13477	20346	26192	33839	41243	48235	55214
Parallel algorithm on 4 p.e.'s	6642	13521	20515	27326	35029	42137	49386	55784

Table 5.3: Number of simplices generated by the serial algorithms and the parallel algorithm for two and four processors.

Two Processors	1000	2000	3000	4000	5000	6000	7000	8000
Processing element 1	0.371	0.8523	1.4620	2.2070	2.3612	3.1563	4.2314	4.413
Processing element 2	0.397	0.8381	1.0918	1.8430	2.3438	3.2163	3.8446	4.241

Table 5.4: Running times of the parallel algorithm on two processors in seconds.

Four Processors	1000	2000	3000	4000	5000	6000	7000	8000
Processing element 1	0.3142	0.8237	0.9835	1.5291	1.9834	2.6945	3.0946	4.1045
Processing element 2	0.2984	0.8909	1.0368	1.7384	2.0938	2.7187	3.3256	4.0580
Processing element 3	0.3011	0.8354	1.1352	1.5942	2.1219	2.6026	3.1890	3.9256
Processing element 4	0.2675	0.8592	1.2117	1.8423	2.2129	2.7014	3.2460	3.9346

Table 5.5: Running times of the parallel algorithm on four processors in seconds.

Algorithm	1000	2000	3000	4000	5000	6000	7000	8000
Detri	0.48	1.2	1.34	1.98	2.17	2.53	4.03	4.69
InCoDe	0.48	1.02	1.68	2.43	3.19	3.96	5.02	5.82
DeWall	0.54	1.22	1.81	2.50	3.28	3.98	4.76	5.56
Parallel algorithm on 2 p.e's	0.384	0.8452	1.2769	2.025	2.3525	3.1858	4.038	4.327
Parallel algorithm on 4 p.e's	0.2953	0.608	1.0172	1.676	2.1030	2.6973	3.2138	4.008

Table 5.6: Running times in seconds of the serial algorithms and the average time of the parallel algorithm on two and four processors

Chapter 6

Conclusions

The parallel solution based on spatial division approach does not require communication or synchronization among processing elements. For nonuniform datasets, we can apply an adaptive partitioning approach defining regions with equidistribution of the pointset for better load distribution among the processing elements. The general position assumption can be discarded by using simulation of simplicity. The parallel algorithm is static, i.e., no new points can be added to the triangulation. Incremental algorithms cannot be parallelized due to high amount of interprocess communication and unequal load balancing. The current serial algorithms to construct the Deluanay triangulation in d dimensions are incremental in nature, so current algorithms cannot be parallelized to acheive good performance.

The statistics gathered for the algorithms described in the thesis have shown that the randomized algorithm performs better in general than the incremental and the divide-and-conquer algorithm for the triangulation. It would be interesting to build a generalized framework to parallelize randomized incremental construction because many geometric problems in d dimensions have simple and elegant randomized algorithms to solve them.

Bibliography

- [1] V. Akman, W. R. Franklin, M. Kankanhalli and C. Narayanaswami. Geometric computing and uniform grid technique. *Computer Aided Design*, 21(7):410-420, September 1989.
- [2] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Survey*, 23(3):345-405, September 1991
- [3] M. D. Berg, M. V. Kreveld, M. Overmars, O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Germany, 2000.
- [4] N. Carrerio and D. Gelenter. How to write a parallel program: A guide to the Perplexed. *ACM Computing Surveys*, 21(3):323-358, September 1989
- [5] P. Cignoni, D. Laforenza, C. Montani, R. Perego, R. Scopigno. Evaluation of Parallelization strategies for an Incremental Delaunay Triangulation in \mathbb{R}^3 , *Concurrency: Practice and Experience*, J. Wiley and Sons, Vol.7(1),61-80, 1995.
- [6] P. Cignoni, C. Montani, R. Perego, and R. Scopigno. Parallel 3D Delaunay triangulation. *Computer Graphics Forum*, 12(3):129-142, 1993.
- [7] P. Cignoni, C. Montani, and R. Scopigno. DeWall: A fast Divide & Conquer Delaunay triangulation algorithm in \mathbb{R}^d . *Computer-Aided Design*, Elsevier Science, vol. 30(5):333-341, April 1998.
- [8] K. L. Clarkson, K. Melhorn and R. Siedel. Four results on randomized incremental constructions. *Proceedings of the 9th Symposium on Theoretical*

aspects of Computer Science, vol. 577, *Lecture notes in Computer Science*, 463-474, Springer-Verlag, 1992.

- [9] K. L. Clarkson, P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387-421, 1989.
- [10] R. A. Dwyer, A faster divide-and-conquer algorithm for constructing Delaunay triangulations, *Algorithmica* 2(2):137-151, 1987.
- [11] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, Germany, 1987.
- [12] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*, Cambridge Univ. Press, England, 2001.
- [13] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations, *Proceedings of the 8th Annual ACM symposium on Computational Geometry*, 43-52, 1992.
- [14] S. Fortune. Voronoi diagrams and Delaunay triangulations, *Computers and Euclidian Geometry*, 1992.
- [15] L. Guibas, D. Knuth and M. Sharir. Randomized incremental construction of Voronoi and Delaunay diagrams, *Proc. 17th Intl. Colloquium on Automata, Languages and Programming*, 414-431, 1990.
- [16] L. J. Guibas and J. Stolfi. Primitives for manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM transactions on Graphics* 4 74-123, 1985.
- [17] R. Motwani and P. Raghavan. *Randomized Algorithms*, Cambridge University Press, New York, 1995.
- [18] K. Mulmuley. *Computational Geometry An Introduction through Randomized algorithms*, Prentice Hall, New Jersey, 1994.

- [19] E. P. Mucke. Alpha Shapes. Doctoral thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1993
- [20] F. Preparata and M. Shamos. *Computational Geometry - An Introduction*, Springer-Verlag, New York, 1985.
- [21] V. T. Rajan, Optimality of the Delaunay triangulation in \mathbb{R}^d , *Proc. of the Seventh Annual Symp. on Comp. Geom*, 357-363, 1991.
- [22] R. Seidel. Backward analysis of randomized geometric algorithms. Report TR-92-014, ICSI, University of California, Berkeley, 1992.
- [23] M. Teillud. *Towards dynamic randomized algorithms in computational geometry*, Lecture Notes in Computer Science, Springer-Verlag, Germany, 1993.