The Dissertation Committee for Sunghee Choi
certifies that this is the approved version of the following dissertation:

Practical Delaunay triangulation algorithms for surface reconstruction and related problems

Committee:

Annamaria B. Amenta, Supervisor

Donald S. Fussell, Supervisor

Calvin Lin

C. Gregory Plaxton

Richard Hammersley

Practical Delaunay triangulation algorithms for surface reconstruction and related problems

by

Sunghee Choi, B.S., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2003

To my parents

Acknowledgments

Throughout my stay in Austin as a graduate student, I am truly blessed to have met so many wonderful people. I cannot express fully all my gratitude toward them.

First of all, I would like to thank my wonderful advisor, Nina Amenta. She has been amazingly patient and always encouraging and supportive to me throughout my years in PhD program. Her optimism and cheerful spirit kept me going forward. She has been the best advisor a graduate student can hope for. I am so fortunate to have met her and known her.

My special thanks go to my committee members, Don Fussell, Greg Plaxton, Calvin Lin, and Richard Hammersley for their valuable time, attention, and suggestions. I thank Jonathan Shewchuk for sharing his pyramid Delaunay triangulation codes and for helpful discussions and insights. I also thank Sylvain Pion and Monique Teillaud for their kind help and ideas about CGAL Delaunay program. I also want to express my special thanks to Herbert Edelsbrunner, Leo Guibas, Otfried Cheong, Chan-Su Shin, Hee-Kap Ahn, Hyeon-Suk Na and Sang-Min Park for making me feel welcome to computational geometry community.

I thank my fellow graduate students, Ravi Kolluri, Maria Jump, Thomas Wahl for their ideas and help in the research. I am also grateful for the friendship and kind help of Yoonsuck Choe and Ahran Lee who welcomed me to UTCS. I want to express my special thanks to Un Yong Nahm who is a great listener and friend.

I am indebted to many people in my church, especially the late Rev. Samuel

Kim, Rev. Duckjean Joe, Rev. Sung Bae Kim, Steve and Carol Lim. I thank them for sharing me the love of Jesus Christ and leading me to the truth. I am deeply grateful for the wonderful fellowship I had with my dear sisters in the Lord, Abigail, Dong-Ok, Seon-hi, Shinhye and Eun-Kyung. Without their prayers and support, it could have been a much more difficult and lonely journey.

I am sincerely grateful for my family, my parents and brother Ung-Young who have always loved and believed in me. Most of all, I owe everything to Jesus Christ, who gave Himself for me and is always with me.

SUNGHEE CHOI

The University of Texas at Austin
May 2003

Practical Delaunay triangulation algorithms for surface reconstruction and related problems

Publication No. _____

Sunghee Choi, Ph.D.
The University of Texas at Austin, 2003

Supervisors: Annamaria B. Amenta and Donald S. Fussell

The Delaunay triangulation is one of the fundamental problems in computational geometry, dual to the well-known Voronoi diagram. It has numerous applications in various disciplines such as computer graphics, computer vision, and robotics. This thesis deals with a number of interrelated questions arising in modeling shapes by computing Delaunay triangulations. We give algorithms for surface reconstruction, computing Delaunay triangulations given surfaces, and computing Delaunay triangulations in general.

Given a set of samples from a surface, the surface reconstruction problem is to construct a piecewise linear approximation of the original surface. We give two surface reconstruction algorithms with guarantees of both geometric and topological correctness using the 3D Delaunay triangulation of the input samples. The first algorithm selects a set of Delaunay triangles using a simple geometric test and extracts a manifold from it. It is the first surface reconstruction algorithm with topological guarantee of correctness. The second algorithm improves the robustness using the weighted Voronoi diagrams called power diagram.

When the surface on which the samples lie are known, we can use the connectivity of the surface to speed up the Delaunay triangulation. We give an $O(n \log^* n)$ expected time algorithm to compute the 3D Delaunay triangulation given the surface on which the samples lie, under some realistic assumptions. It improves upon $O(n \log n)$ expected time usual randomized incremental algorithm in this case. The algorithm can be useful for mesh generation and medial axis construction.

The randomized incremental algorithm for Delaunay triangulation is theoretically optimal in expected time but suffers from serious thrashing because of its random memory access pattern when the data structure gets too large to fit in memory. We propose a new insertion order called *biased randomized insertion order* (BRIO) which removes enough randomness to significantly improve performance, but leaves enough randomness so that the algorithm remains theoretically optimal. We show by experiments that the size of input data we can compute in a given machine increases dramatically using BRIO instead of the total random order.

Contents

Chapter 1

Introduction

The Delaunay triangulation is one of the fundamental problems in computational geometry, dual to the well-known Voronoi diagram. See Figure 1.1. They are ubiquitous structures in nature and science and have various applications in computer vision, computer graphics, mesh generation and many other fields. Fortune [43] gives a nice survey about the properties and the algorithms for the Delaunay triangulation and Voronoi diagram.

The Delaunay triangulation and Voronoi diagram are defined in any dimension, but in this dissertation, we study the three dimensional case, the Delaunay triangulation of points in $\mathbb{R}^3$, also called the Delaunay tetrahedralization. This particular case claimed our attention because of our work on the surface reconstruction problem.

Given a set of samples from the surface of an object, the goal of the surface reconstruction problem is to construct an approximation to the original surface. This problem has various practical applications in reverse engineering, computer graphics and animation, medical engineering and computer aided design. In Chapter 3, we present two algorithms using the 3D Delaunay triangulation.

The first algorithm selects a set of triangles from the 3D Delaunay triangula-

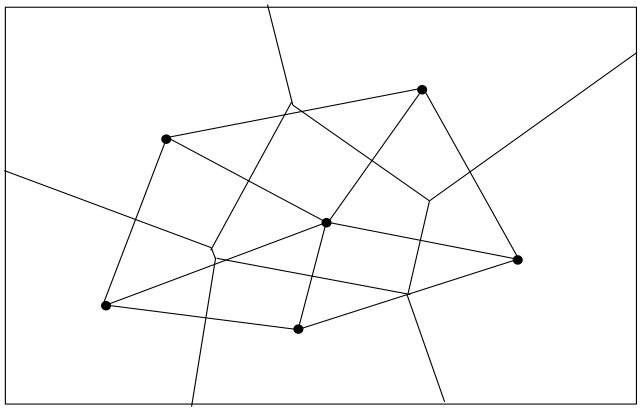

Figure 1.1: The Voronoi diagram and its dual Delaunay triangulation

tion using a simple geometric test. It is the first algorithm with both topological and geometric guarantees of correctness. From the set of triangles selected it extracts a triangulated manifold. Extracting a manifold is easy in theory but not trivial in practice. In case of sharp corners and boundaries, the algorithm is not particularly robust. The algorithm can fail to select some triangles and the output may have holes. The second algorithm fixes this problem by using the weighted Voronoi diagram called *power diagram*. It guarantees that the output is always a watertight boundary of a solid even with undersampling, and it does not require a manifold extraction step. It gives much more robust results in practice. With some heuristics, we can even correctly reconstruct sharp corners produce approximate offset surfaces and surfaces with boundaries.

Our surface reconstruction work revealed interesting connections to the medial axis. The medial axis is a compact representation of the shape of the surface and is a well-studied, important problem in computer vision and geometry. It has many applications in computer graphics, computer animation, mesh generation, computational biology, shape representation. Our second surface reconstruction algorithm also gives one of the best algorithms for the 3D medial axis construction. It has been used even when the surface is known, to construct the medial axis. Clearly in this

latter case, the surface should be useful for constructing the Delaunay triangulation.

The usual algorithm for the 3D Delaunay triangulation is the randomized incremental algorithm. It constructs the Delaunay triangulation by adding input points one by one in random order and updating the Delaunay triangulation after each insertion. The running time of the algorithm can be divided into two parts, the time required to find where each new point should be inserted into the Delaunay triangulation (*point location time*) and the time required to delete old tetrahedra and create new tetrahedra so as to actually perform the insertion (*update time*).

We use the connectivity of the surface to speed up the point location time of the 3D Delaunay triangulation computation. We present the algorithm in Chapter 4. Given some realistic assumptions about the input data, the algorithm takes an $O(n \log^* n)$ expected time improving upon the $O(n \log n)$ expected time usual randomized incremental algorithm in that case. It has applications in mesh generation and medial axis construction.

We turned our attention to the 3D Delaunay triangulation because it is the bottleneck of our surface reconstruction algorithms. We present the results from our experimental study for 3D Delaunay triangulation of samples from surfaces in Section 4.4. Though the worst case complexity of the 3D Delaunay triangulation is quadratic, our experiment shows that the size of the Delaunay triangulation is linear. In this case, the theoretical bottleneck of the algorithm is point location time. (This is the problem addressed by the speed-up presented in Chapter 4.) But, in practice, point location does not dominate the total running time. Instead, the main performance problem of the 3D Delaunay triangulation programs is thrashing.

The expected running time of the randomized incremental algorithm is optimal and it depends on the random permutation of insertion order. But, when the data structure gets too big to fit in memory, because of the inherent random access pattern the program thrashes very badly. This limits the size of the problem we can

compute on a given machine.

In Chapter 5, we propose a new insertion order called *biased randomized insertion order* (BRIO). We prove that using BRIO instead of random order, the algorithm still remains optimal both in the worst case and in the "realistic" case when the expected size of the Delaunay triangulation of r points is $O(r)$. Using a BRIO we could compute the Delaunay triangulation of much larger data sets than were possible with the completely randomized insertion order. Though we analyze BRIO in the context of the 3D Delaunay triangulation, the analysis holds for other randomized incremental construction such as convex hull, Delaunay triangulation in any dimension and the trapezoidation of line segments.

Chapter 2

Background

In this chapter, we introduce some of the important definitions and theorems that will be used in the thesis.

2.1 Voronoi diagram and Delaunay triangulation

In this section, we define the Voronoi diagram and its dual Delaunay triangulation. Though these definitions are given for general dimension d, in this thesis, we are only interested in the three dimensional case.

Let S be a finite set of points in R^d which we call the *samples* or *sites*. Let $dist(x, y)$ for two points x, y denote the Euclidean distance between them. For each site $p \in S$, the Voronoi cell V_p is defined as follows.

$$V_p = \{x \in R^d : dist(x, p) \leq dist(x, q), \forall q \in S\}$$

In other words, the Voronoi cell V_p is a set of points in R^d whose closest site is p. It follows from the definition that each Voronoi cell is a (possibly unbounded) convex polyhedron. The Voronoi diagram $Vor(S)$ of S is defined as a union of Voronoi cells

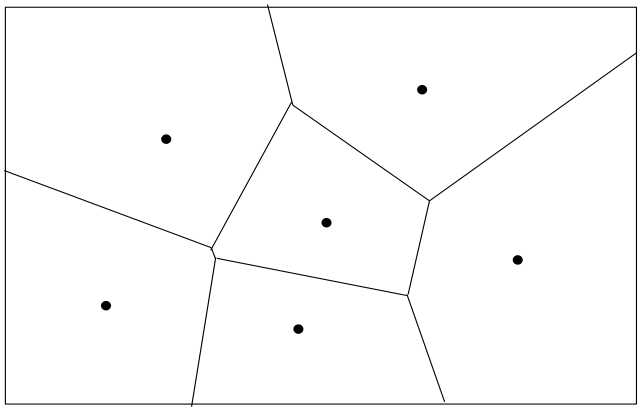

Figure 2.1: The Voronoi diagram of six sites

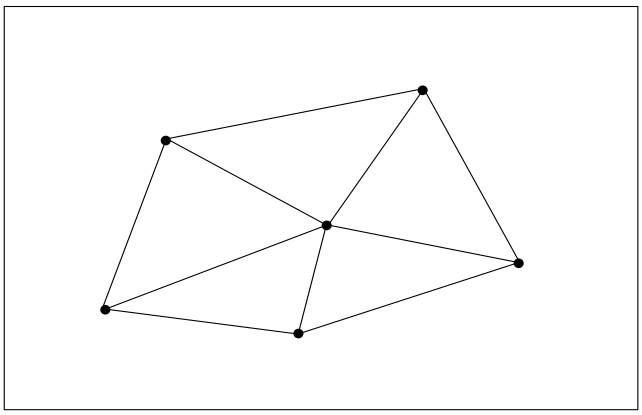

Figure 2.2: The Delaunay triangulation

V_p for $p \in S$:

$$Vor(S) = \bigcup_{p \in S} V_p$$

It is a decomposition of R^d into Voronoi cells.

The Delaunay triangulation is the dual of the Voronoi diagram. If two Voronoi cells V_s and V_t are adjacent in the Voronoi diagram then there is a dual Delaunay edge connecting s and t in the Delaunay triangulation. Given a Delaunay triangulation, one can compute the dual Voronoi diagram in linear time and vice versa.

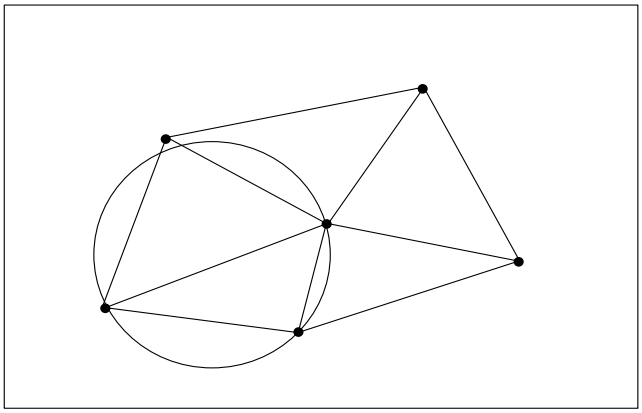

Figure 2.3: The Delaunay triangulation and an empty circumcircle of a Delaunay triangle

Another way to define the Delaunay triangulation is its *empty sphere property*. An *empty* circle (or sphere) is a circle (or sphere) which does not contain any site in its interior. The Delaunay triangulation $D(S)$ of S is a graph where S is its vertex set and there is a straight edge pq if there is an empty circle containing p and q in its boundary.

When $d = 3$, if no three points lie on a line and no four points lie on a circle, we say that the points are in general position. When S is in general position, the Delaunay triangulation of S is a tetrahedralization of the convex hull of S and each Delaunay tetrahedron $pqrs$ in $D(S)$ has an empty circumsphere (a sphere containing p, q, r, s on its boundary). The center of the empty circumsphere is the Voronoi vertex dual to the Delaunay tetrahedron $pqrs$. We assume for the rest of the thesis that S is in general position.

Among the important applications of the 3D Delaunay triangulation are surface reconstruction, medial axis approximation, and mesh generation. We will review some of the related works on surface reconstruction using the 3D Delaunay triangulation in Chapter 3. We define the medial axis in Section 2.5 and explain its relationship to the Delaunay triangulation and Voronoi diagram. We will not deal

with mesh generation in this thesis except to note here that Delaunay refinement is a popular way to generate high-quality tetrahedral meshes [65, 22].

2.2 Power diagram and regular triangulation

The power diagram and its dual regular triangulation are generalizations of the Voronoi diagram and Delaunay triangulation for the case of weighted points. We consider a (Euclidean) ball $B_{c,r}$ with center c and radius r as a weighted point c with weight r^2.

We define a new distance function called *power distance* $pow(B_1, B_2)$ between two weighted points (or balls) $B_1 = B_{c1,r1}$ and $B_2 = B_{c2,r2}$ as follows:

$$pow(B_1, B_2) = dist(c1, c2)^2 - r1^2 - r2^2$$

We consider an unweighted point as a ball with radius zero. So the power distance between a point x and a ball $B_{c,r}$ is :

$$pow(x, B) = dist(x, c)^2 - r^2$$

Using the power distance instead of the Euclidean distance, we define the power diagram as a weighted Voronoi diagram. The power diagram of a set W of balls is the union of power cells V_p for all $p \in W$. A power cell V_p of a ball p is defined as follows:

$$V_p = \{x \in R^d : pow(x, p) \leq pow(x, q), \forall q \in W\}$$

A nice thing about the power diagram is that each cell is a (possibly unbounded) polyhedron, like the Voronoi diagram. But, in the power diagram we can have empty power cells for some balls, unlike the Voronoi diagram which has a nonempty cell for every site.

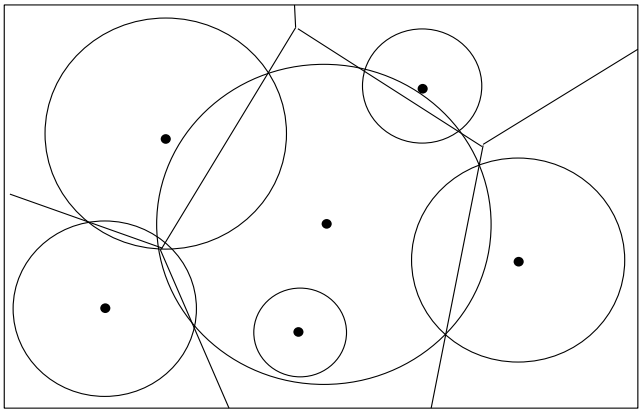

Figure 2.4: The power diagram of six balls

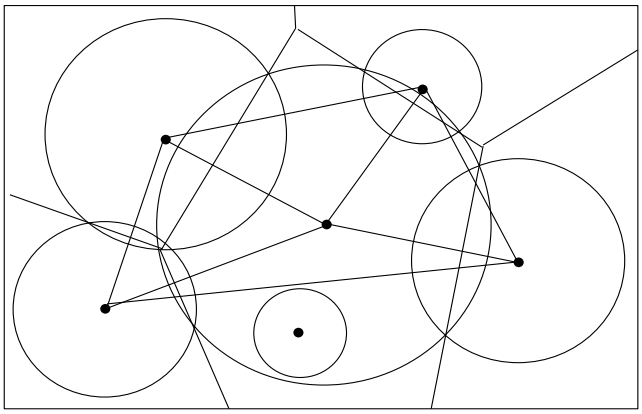

Figure 2.5: The power diagram and its regular triangulation

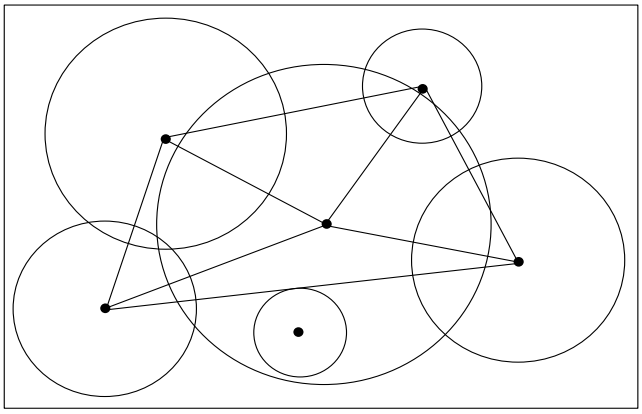

Figure 2.6: Regular triangulation

The dual of the power diagram is called the *regular triangulation*. It is obtained by connecting centers of balls with adjacent power cells with straight edges. Since some balls may have empty power cells, they may not appear in the regular triangulation.

2.3 Randomized incremental algorithm

The most widely used algorithm for the 3D Delaunay triangulation is the randomized incremental algorithm [24, 25, 26]. It constructs the Delaunay triangulation $D(S)$ by inserting samples of S in random order. When we insert a sample p into $D(R)$ for $R \subset S$, we first locate p in $D(R)$. Then, we remove all Delaunay tetrahedra in $D(R)$ whose circumsphere contains p and create new tetrahedra adjacent to p to construct $D(R \cup \{p\})$.

The running time of the algorithm can be divided into two parts - the *location time* which is the time to locate a new point to add in $D(R)$, and the *update time* which is the time to update the Delaunay triangulation.

There are many ways to locate a new point in the current Delaunay triangulation. One of the theoretically optimal methods is using the *history DAG* [26].

When we add a new point, instead of deleting Delaunay tetrahedra that are no longer Delaunay, we make them parents of the new Delaunay tetrahedra created by the insertion. To search for a new point, we follow a path of tetrahedra in the history DAG that all contain the new point.

2.4 Homeomorphism

We will show in Chapter 3 that the output of our surface reconstruction algorithm is topologically correct (i.e. topologically equivalent to the original surface.) So here we introduce a formal definition of topological equivalence called *homeomorphism*.

Let X and Y denote two topological spaces. A *homeomorphism*, $\mu : X \to Y$, is a continuous bijection whose inverse is also continuous. X and Y are *topologically equivalent* or *homeomorphic* if there is a homeomorphism between them. Intuitively, when two topological spaces are homeomorphic, we can continuously move either space into and onto the other space.

2.5 Medial axis transform

Let F be a bounded two-dimensional surface in $\mathbb{R}^3$. A ball is *empty* if it does not contain any point of F in the interior. A *medial ball* is a maximal empty ball; that is, it is not contained in any other empty ball. Then the *medial axis transform* of F is the set of medial balls, each represented by its center and radius. The set of the centers of the medial balls is called the *medial axis*. Equivalently, the medial axis of a surface F is defined as the closure of points that have more than one closest point on F.

The medial axis has various applications in mesh generation, animation, and computer vision. But, it is hard to compute the medial axis exactly. So there has been a lot of work on approximating the medial axis. A popular method is to

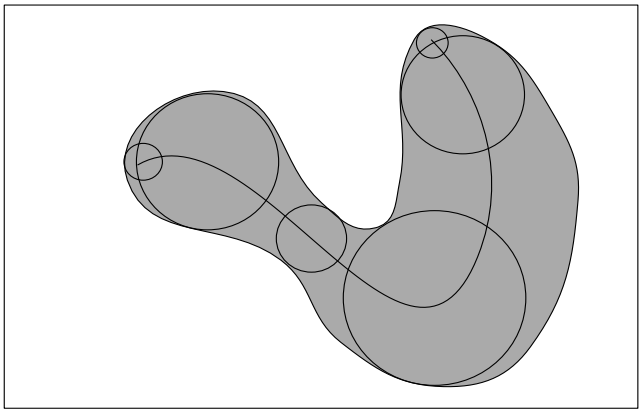

Figure 2.7: Medial axis transform

discretize the surface with samples and to use the Voronoi diagram of the samples as an approximation. But, even when the sampling density goes to infinity, it is not true that the 3D Voronoi diagram converges to the medial axis. In Section 2.8, we introduce the definition of the *pole* which is shown to approximate the medial axis.

2.6 Sampling condition

Here we define the sampling condition under which we can prove the correct reconstruction. This definition was proposed by Amenta et al [3, 2, 4]. Unlike *uniform sampling* where the sampling density is fixed throughout the surface, our sampling condition varies according to the surface complexity; we need more samples in the complex area and less samples in the simple, featureless area.

We first need the definition of *local feature size*(LFS) function : for a point $x \in \mathbb{R}^3$, $LFS(x)$ is the Euclidean distance from x to the nearest point on the medial axis of F. Note that in flat, featureless areas, the medial axis is far away from the surface, so LFS value is big, while in complicated areas where the curvature is big, the medial axis is close to the surface, so LFS value is small.

We now define our sampling requirement : A sample set $S \subset F$ is an r-sample

if the distance from any point $x \in F$ to the nearest sample point $s \in S$ is at most $r\ LFS(x)$. We will see later that S is dense enough for our purposes when $r \leq .08$. Note that at sharp corners, the medial axis meets the surface, so the definition of the r-sample implies that we need infinite number of samples at sharp corners.

The following lemma says that the LFS function is Lipschitz.

Lemma 1 (Amenta and Bern [2]) *For any two points p and q on W, $|LFS(p) - LFS(q)| \leq d(p,q)$.*

Observation 2 *If d(u,s)= O(r)LFS(u) then d(u,s)=O(r)LFS(s) as well, for $r < 1$.*

The following lemma is a Lipschitz condition on the surface normal with respect to LFS.

Lemma 3 (Amenta and Bern [2]) *For any two points p and q on F with $d(p,q) \leq \rho \min\{LFS(p), LFS(q)\}$, for any $\rho < 1/3$, the angle between the normals to F at p and q is at most $\rho/(1-3\rho)$.*

2.7 Restricted Voronoi diagram and restricted Delaunay triangulation

Let S be a set of samples on a surface F. Consider the three-dimensional Voronoi diagram $V(S)$ of S, and its intersection with F. The Voronoi diagram $V(S)$ forms a partition of F into regions; this decomposition is the *restricted Voronoi diagram* $V(S,F)$ of S in F. Then the *restricted Delaunay triangulation* $D(S,F)$ of S in F is the set Delaunay triangles dual to the restricted Voronoi diagram. We call the triangles in the restricted Delaunay triangulation, *restricted Delaunay triangles*. Equivalently, a restricted Delaunay triangle is a Delaunay triangle of S dual to an edge of the Voronoi diagram of S intersecting F.

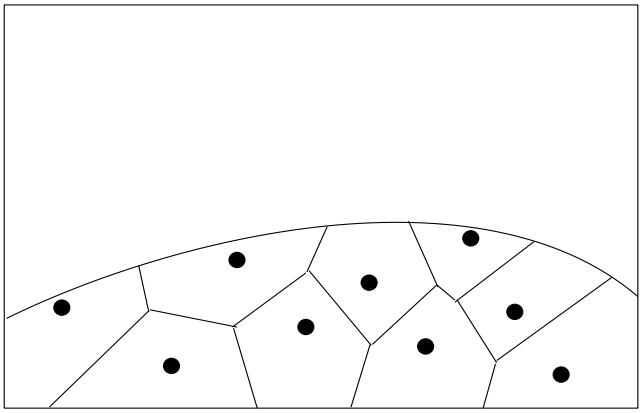

Figure 2.8: Restricted Voronoi diagram

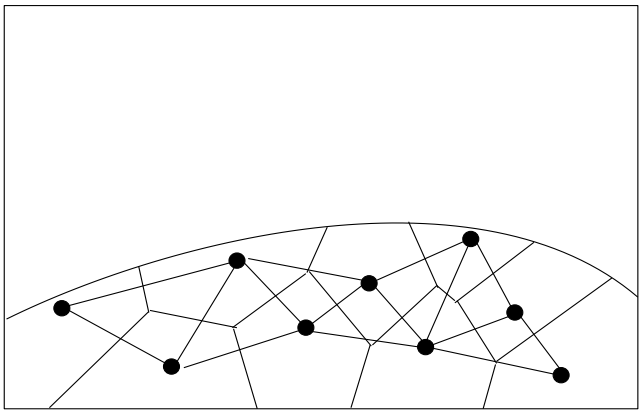

Figure 2.9: Restricted Voronoi diagram and Delaunay triangulation

Edelsbrunner and Shah [40] showed that the restricted Delaunay triangulation $D(S, F)$ is homeomorphic to the surface F if S has the following *closed-ball property* : the closure of each k-dimensional face, $1 \leq k \leq 3$, of the Voronoi diagram of S intersects F in either the empty set or in a closed $(k-1)$-dimensional topological ball.

Using the above theorem, Amenta and Bern showed that given a dense enough sample the restricted Delaunay triangulation is homeomorphic to the surface.

Theorem 4 (Amenta and Bern [2]) *If S is an r-sample of F for $r \leq .1$, then the restricted Delaunay triangulation $D(S, F)$ forms a polyhedron homeomorphic to F.*

2.8 Poles

The definition of the *pole* was introduced by Amenta et al [2, 4]. When S is a dense enough sample of a surface F, the Voronoi cell of every sample s is long and skinny and perpendicular to the surface F. This happens because in directions tangent to the surface the Voronoi cell is bounded by the proximity of other samples on the same local patch of surface. The following lemma makes this precisely.

Lemma 5 (Amenta and Bern [2]) *Let s be a sample point from an r-sample S. Let v be any point in V_s such that $d(v, s) \geq \rho LFS(s)$ for $\rho > \frac{r}{1-r}$. Let $\angle nv$ be the angle at s between the vector $\vec{v}$ from s to v and the surface normal $\vec{n}$ at s. Then $\angle nv \leq \arcsin \frac{r}{\rho(1-r)} + \arcsin \frac{r}{1-r}$.*

This lemma also shows that if we can find a point v in the Voronoi region which is sufficiently far away from s, $\vec{v}$ will be a good approximation of $\vec{n}$.

The *poles* of a sample $s \in S$ are the farthest vertices of its Voronoi cell in either side of the surface F.

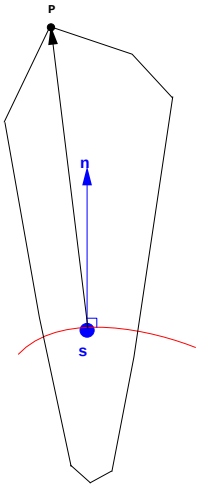

Figure 2.10: The pole of a sample s, the vector from s to its pole is a good approximation of the surface normal at s.

The distance to either pole of a sample s is at least $LFS(s)$ ([8], page 16). Let p be the pole of a sample s. Using Lemma 5, we can observe that the vector $\vec{p}$ from s to p is a good approximation of the surface normal $\vec{n}$ at s.

Lemma 6 (Amenta and Bern [2]) *Let $\angle np$ be the angle between $\vec{n}$ and $\vec{p}$. Since $d(s,p) \geq LFS(s)$, Then, $\angle np \leq 2 \arcsin \frac{r}{1-r}$.*

Chapter 3

Surface Reconstruction

In this chapter, we review our work on surface reconstruction.

Given a set of points S obtained from the surface F of an object, the surface reconstruction problem is to obtain a piecewise linear approximation of the original surface F (See Figure 3.1.) There are various ways to acquire the input point sets, such as laser range scanner, hand-held digitizers, 3D cameras, CT, MRI, etc. Continuing advancement of the acquiring devices has made this problem more important. Industrial applications include reverse engineering, product design and the construction of personalized medical appliances.

We have proposed two surface reconstruction algorithms with provable guarantees. The first algorithm called the *co-cone* algorithm simplifies the crust algorithm by Amenta et al. [2, 4] and the second - the *power crust* - gives more robust results in practice. The theoretical guarantees are of the following form : Given a good sampling defined with respect to the original surface, our reconstruction is topologically correct (homeomorphic to the original surface) and geometrically close to the original surface.

The bottleneck of these algorithms is the Delaunay triangulation computation. In Chapter 5, we propose a method to improve the performance of the

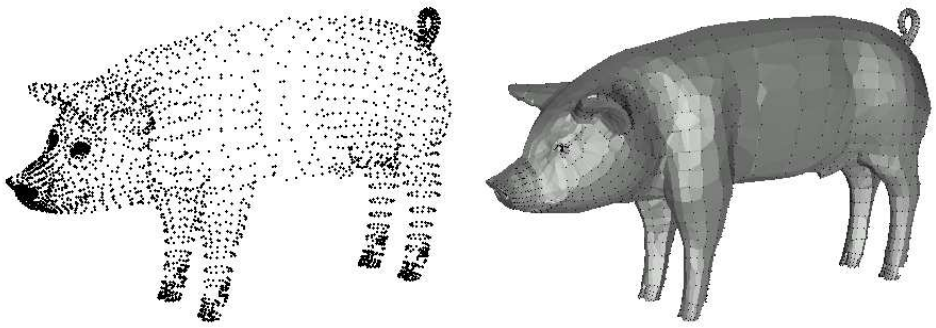

Figure 3.1: Surface reconstruction : Left is input samples, and right is the reconstructed surface using power crust with samples.

Delaunay triangulation computation.

3.1 Related work

The surface reconstruction problem has drawn much attention from researchers in computer graphics, solid modeling, computer vision and computational geometry. There are different variants of the problem. We can classify them into two groups - algorithms which only use the coordinates of the samples (unorganized points) and algorithms which use additional information. Some algorithms are specialized to a particular kind of input such as range data from laser range scanner which come with the surface normal, or data from computer vision technique. The reconstruction algorithms can also be categorized by the types of their output. The reconstructed surface may interpolate or approximate the samples. Some construct watertight models and others output surfaces with boundary. Some algorithms construct piecewise linear surfaces while others construct implicit functions which may be smooth or piecewise smooth. There are various and large amount of work in this problem. We only discuss some of the related work in this section.

Hoppe et al. [48] popularized the problem in computer graphics community.

The input is unorganized points. The algorithm constructs a signed distance function by estimating a tangent plane at each sample using the k nearest neighbors. Then a zero-set is extracted as a piecewise linear surface using a marching cubes algorithm [52].

Curless and Levoy [27] construct surfaces from range data. They use the additional information such as surface normal and reliability estimates available in the range data to construct a signed distance function. They introduced a hole-filling step. Their algorithm is remarkably efficient and works well in practice, and had been used in the Digital Michelangelo project [51] where they built huge models of the statues of Michelangelo. However, there are no theoretical guarantees that they output a "correct" reconstruction. Also, since they put a voxel grid and extract an iso-surface from it, they have a fixed output resolution over the whole model.

Zhao et al. [67] formulate the surface reconstruction problem using differential geometry and partial differential equations. They construct implicit surfaces using the level set method on fixed rectangular grids. The input data set may consist of points, curves, and/or surface patches. They handle noisy or highly non-uniform data sets easily but their algorithm is slow and the resolution is again dependent on the voxel grid.

Dinh et al. [35] generate smooth and seamless models by constructing a 3D implicit function from sparse, noisy, non-uniform, and low-resolution range data which come from computer vision techniques. The 3D implicit function is formulated as a sum of weighted radial basis functions. The surface is insensitive to noise because it can approximate, rather than interpolate, the data. But, they generate blurry and blobby models losing details.

In computational geometry, many surface reconstruction algorithms have been proposed which use the Delaunay triangulation of the samples. Boissonnat [15] proposed heuristics to reconstruct the surface by removing Delaunay tetrahedra one

by one from the Delaunay triangulation of samples. It only applies to objects with genus zero. Moreover, it has no theoretical guarantee, and in fact can fail on some reasonable inputs.

The α-shape by Edelsbrunner and Mücke [39] is also a subset of the Delaunay triangulation. Though α-shape works well for uniform samples, in case of non-uniform sampling, it is difficult to choose an appropriate parameter α to balance the hole-filling and the loss of detail.

The crust by Amenta et al. [2, 4] is the most closely related to our work. They introduced the non-uniform sampling condition called ϵ-sampling defined in Section 2.6, under which they prove that the reconstruction is geometrically close to the original surface. It is the first algorithm that had any theoretical guarantee of correctness. Many subsequent works including both of our algorithms used their sampling condition to prove the correctness of the reconstruction.

Boissonnat and Cazals [16] construct a smooth implicit surface by the natural neighbor interpolation using the Voronoi diagram given a set of samples with their normals. The surface interpolates all the sample points and the surface normal at a sample point corresponds to the normal the point is equipped with. The surface is represented as the zero-set of a signed pseudo-distance function. Computing the natural neighbor interpolation is time consuming.

Bernardini et al. [12] proposed the ball-pivoting algorithm for range data. The algorithm requires a uniform sampling and normals of sample points given in the range data. Since they avoid the Delaunay triangulation computation, they can process large data efficiently. They also give an out-of-core version of their algorithm.

There are several important advantages in using the Delaunay triangulation for surface reconstruction. First, since they do not require additional information except the coordinates of the samples, they can be applied to various kinds of in-

put data. Second, currently they are the only algorithms with some theoretical guarantees. Third, the resolution of reconstructed surfaces varies with the sampling density. Drawbacks are noise sensitivity and speed. Since the Delaunay triangulation based algorithm interpolates the samples, it does not handle noise very well. We find that the Delaunay triangulation computation is usually the bottleneck of these algorithms. By improving the performance of the Delaunay triangulation, these Delaunay triangulation based surface reconstruction algorithms can be more widely used. In Chapter 5, we propose one such method.

3.2 The co-cone algorithm

3.2.1 Introduction

Here we present our first surface reconstruction algorithm which we call the *co-cone* algorithm. This work simplifies the crust algorithm by Amenta et al. [2, 4], both in the computation and in the proof that the reconstructed surface is geometrically close to the original surface. In addition, it is the first algorithm that proved the topological correctness - the reconstruction is homeomorphic to the original surface. Here we give a brief overview of the algorithm. The details of the algorithm and the proofs are included in Appendix A.

3.2.2 Algorithm

The algorithm constructs a piecewise linear surface T from the set S of samples from a surface F. Like many previous algorithms, we select T as a subset of the Delaunay triangulation of S.

In Section 2.7, we saw that the restricted Delaunay triangulation $D(S, F)$ is in some sense the "correct" subset of the Delaunay triangulation to represent the surface F. But the restricted Delaunay triangulation is impossible to compute

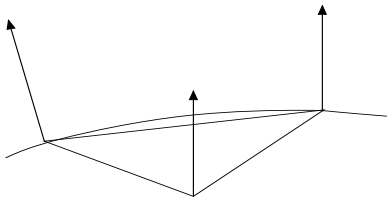

Figure 3.2: The candidate triangle normal agrees with surface normal.

without knowing the surface. Instead, we derive some properties of the restricted Delaunay triangulation that we can infer using the set of samples S and its Voronoi diagram.

We give a condition called the *co-cone test* for a Delaunay triangle to meet in order to be in the set of *candidate* triangles. This is simply done by considering the dual Voronoi edge and the poles of each vertices of a triangle. We then proved that given a dense enough sampling, the set of the candidate triangles includes the restricted Delaunay triangulation. Furthermore, the normal of every candidate triangle is similar to the normal at each of its vertices with respect to the original surface F (see Figure 3.2), and candidate triangles are small with respect to the distance to the medial axis. Using these facts, we could show that *any* manifold extracted from the set of candidate triangles would be homeomorphic to the original surface F. We give the proofs in Appendix A of this thesis.

We implemented the algorithm using the Delaunay triangulation function of Clarkson's `hull` program. The result is comparable to that of the crust algorithm although the algorithm runs faster. Our theoretical guarantee only applies under the assumption that the original surface is smooth closed manifold. For sharp corners and boundaries, the output may have gaps and holes. See Figure 3.3.

We did not implement the manifold extraction step which selects a piecewise-linear manifold from the set of the candidate triangles T. A theoretical algorithm for manifold extraction is given in the crust papers by Amenta et al [4, 2]. But, it does not work robustly in case of real data. In [5], a heuristic is given which works

Figure 3.3: A set of candidate triangles. There are some holes in undersampled areas such as ears and nose and the boundary around the neck.

quite well in practice.

In case of sharp corners and boundaries, the output generated by the co-cone algorithm may have holes. Dey and Giesen [31] detect undersampling using the shape of the Voronoi diagram. Undersampling usually takes place in sharp corners, boundaries, regions of high curvature where LFS value is small. Dey and Goswami [32] give a tight co-cone algorithm which fills small holes by stitching triangles to obtain water-tight models. Dey, Giesen, and Hudson [33] extend the co-cone algorithm to handle large data in the range of million points by avoiding the computation of the Delaunay triangulation of the entire input data. They divide the input data into manageable chunks using octree subdivision and apply the co-cone algorithm on each chunk. Then the reconstructed surfaces are matched together. Their software is publicly available.

3.3 Power crust

3.3.1 Introduction

We present another algorithm called the *power crust* for surface reconstruction. Its construction is based on approximating the medial axis transform (MAT) and then extracting a polygonal surface from the *power diagram*. It also outputs a piecewise linear approximation to the medial axis called the *power shape*.

One of the limitations of the co-cone algorithm is that if the sampling condition is not met - which is often the case in practice especially with sharp corners and boundaries - the output may contain holes. The power crust is guaranteed to be a watertight boundary surface of the three-dimensional solid on *any* input. No manifold extraction or clean-up postprocessing is required. Furthermore the power crust also has the same correctness guarantee as the co-cone algorithm : given a good sampling the power crust is homeomorphic and geometrically close to the original surface. The algorithm is also very simple and gives a much more robust results in practice. Our software to generate the power crust is publicly available.

The medial axis transform(MAT) is a representation of a shape with an infinite set of balls. We approximate MAT with a discrete set of balls called *polar balls*. Given a set of polar balls, we construct the power diagram and label each polar ball either inside or outside. The power crust is the faces of the power diagram separating the inside polar balls and the outside polar balls. By connecting adjacent poles with the same label we get the power shape - our approximation of the medial axis. A two-dimensional version of the algorithm is shown in Figure 3.4. Figure 3.5 shows the samples collected by a laser range scanner, its power crust and power shape. We give an overview of the power crust algorithm in this section; more details and theorems appear in Appendix B.

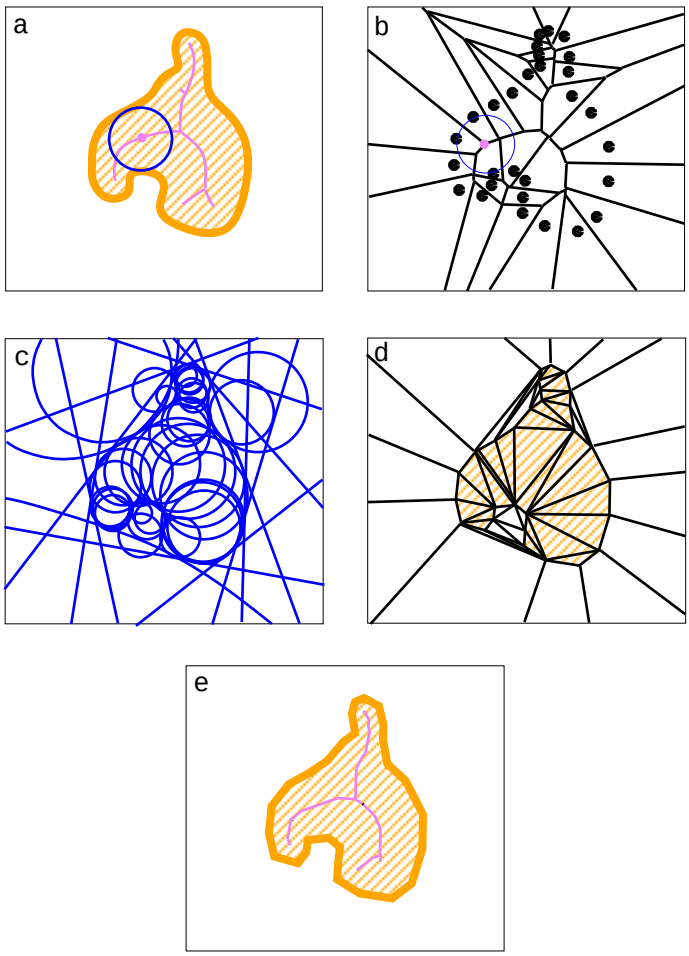

Figure 3.4: Two-dimensional example of power crust construction. a) An object with its medial axis; one maximal interior ball is shown. b) the Voronoi diagram of S, with the polar ball surrounding one pole shown. In 2D, we can select all Voronoi vertices as poles, but not in 3D. c) The inner and outer polar balls. Outer polar balls with centers at infinity degenerate to halfspaces on the convex hull. d) The power diagram cells of the poles, labeled inner and outer. e) The power crust and the power shape of its interior solid.

Figure 3.5: Samples from laser range scanner, the power crust, and the power shape

3.3.2 Algorithm

To approximate the MAT, we use the poles defined in Section 2.8. Each pole v is the center of a Voronoi ball, which we shall call its *polar ball*. The set of polar balls for all $v \in V$ gives our approximation of the medial axis transform: the MAT is an infinite set of balls, and the approximation is the similar finite set of polar balls.

The polar balls corresponding to poles inside of F are *inner polar balls*; *outer polar balls* are defined analogously. The union of the inner polar balls forms a good approximation of the object bounded by F, and similarly the union of outer polar balls forms a good approximation of the complement of the object (This is proved in Appendix B).

Now we consider the power diagram of the polar balls, which subdivides $\mathbb{R}^3$ into a set of cells. The *power crust* is the boundary between the power diagram cells belonging to inner poles and power diagram cells belonging to outer poles.

Since most points of the interior solid bounded by F are inside the union of

the inner polar balls, and outside of the union of outer polar balls, they belong to cells of the power diagram corresponding to inner poles. Similarly most points in the exterior solid belong to cells corresponding to outer poles.

A two-dimensional face of the power crust separates cells corresponding to an inner and an outer pole. The two polar balls should intersect shallowly, if at all, since the inner polar ball is mostly inside the object and the outer polar ball is mostly outside. So the power crust face lies near the boundaries of both unions of balls, and hence near the boundary F of the object. Figure 3.6 shows an example of the union of inner polar balls and the power crust approximating the original object. The power crust actually interpolates the input samples in S, which lie on the surface of the union of the inner, and of the outer, polar balls.

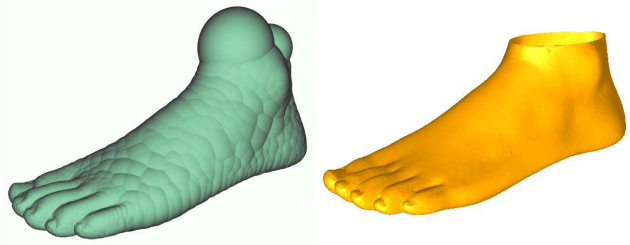

Figure 3.6: A set of inner polar balls and the resulting three-dimensional power crust. The opening at the top of the foot was detected because large inner polar balls protrude out of the model, and intentionally left as a hole; see Section 3.3.4.

The definition of the power crust implies a way to connect the poles to form a topologically correct approximation of the medial axis (homeomorphic to the medial axis of the original object) which we call the *power shape*. The vertices of the power shape are the poles themselves. Inner poles whose cells are adjacent in the power diagram are connected. The power shape is a subset of the regular triangulation dual to the power diagram. We summarize the basic algorithm in Figure 3.7, and elaborate on Step 4 in the next section.

1. Compute the Voronoi diagram of the sample points S.
2. For each sample point, compute its poles.
3. Compute the power diagram of poles.
4. Label each pole either inside or outside.
5. Output the power diagram faces separating the cells of inside and outside poles as the power crust.
6. Output the regular triangulation faces connecting inside poles as the power shape.

Figure 3.7: Power crust algorithm

3.3.3 Labeling algorithm

We label the poles as inner or outer by examining the power diagram. We define a natural graph on the power diagram cells: two cells are connected in the graph if they share a two-dimensional face. In addition, two cells are connected if they belong to the two poles of the same sample s. We traverse this graph, labeling poles `inner` or `outer` as we go. When S is well-sampled the simple algorithm below can be proved correct, using two facts. The first is that an inner polar ball and an outer polar ball can only intersect shallowly. The second is that one of the two poles of every sample is an inner pole and the other is an outer pole.

The naive traversal algorithm begins by labeling poles adjacent to points forming the bounding box Z as `outer` and then propagating labels as follows. For any pole p labeled `outer`, if it has an unlabeled neighbor q such that the polar balls of p and q intersect deeply, we give q label `outer` as well. And for each sample s for which p is a pole (there might be more than one), we give the other pole of s the label `inner`. We propagate the labels of inner poles similarly: deeply intersecting neighbors get labeled `inner`, and the opposite pole of the same sample gets labeled `outer`. The proof of termination and correctness of this algorithm is

given in Appendix B.

But because the sampling assumption is not met everywhere, a naive implementation of this graph-traversal algorithm could fail dramatically - once an error is made, it propagates. Instead, we choose which labels to propagate using the following greedy heuristic. We keep track of the "belief" that an unlabeled ball is inner or outer, based on the labels already assigned, and we label and propagate the labels of the poles for which we are most confident first.

Specifically, each ball keeps track of two values, *in* and *out*, which lie between 0 ("unknown") and 1 ("certain"). We start by giving all poles far away from the bounding box of the original samples an *out* value of 1 and an *in* value of zero, and initialize all other balls' *in*/*out* values to zero.

We put all the unlabeled poles in a priority queue, with the priority determined by the *in* and *out* values. If only one of the *in* or *out* values is non-zero, we use the non-zero value as the priority. If both *in* and *out* values are non-zero, it means that the pole is "confused"; we would like to label such poles as late in the process as possible, so we give them the priority $|in - out| - 1$, which is between zero and -1.

The algorithm is then to repeatedly remove the top element of the queue and label it *in* or *out*, whichever has the bigger value. We then propagate the newly assigned label to the *in* and *out* values of the remaining unlabeled poles, changing their priority in the queue.

We use the local geometry to weight the effect of a newly labeled pole on its neighbors. For a sample s of which p is the pole, let β denote the angle formed by p,s and the other pole q of s, so that we have $\pi/2 \leq \beta \leq \pi$. If the surface is sampled densely enough, the vectors to the two poles should point in nearly opposite directions. So the denser the sampling, the larger β should be. So the bigger β is, the more "likely" is it that q should get the opposite label from p. We

Label_Poles() {
    For all poles p,
        initialize $in(p) = out(p) = 0$.
        insert p in the queue.
    For each pole p adjacent to points of Z,
        $out(p) = 1$.
        Update_Priority(p)
    while (queue is not empty) {
        Remove the top element p of the priority queue
        If $in(p) > out(p)$, $label(p) = in$, $tmp(p) = in(p)$
        Otherwise, $label(p) = out$, $tmp(p) = out(p)$
        For each sample s of which p is the pole,
            let q be the other pole of s,
            $opp(label(p))(q) = max(tmp(p) * w_{pq}, opp(label(p))(q))$
    /* $opp(in) = out$, $opp(out) = in$, $w_{pq} = -\cos(\angle psq)$*/
            Update_Priority(q)
        For each deeply intersecting neighboring poles q,
            $(label(p))(q) = max(tmp(p) * w_{pq}, (label(p))(q))$
    /* $w_{pq} = -\cos(\alpha)$, α is angle between balls p and q*/
            Update_Priority(q)
    }
}

Update_Priority(pole p) {
    If $in(p) > 0$ and $out(p) > 0$, $pri(p) = |in(p) - out(p)| - 1$.
    Otherwise, $pri(p) = max(in(p), out(p))$.
}

Figure 3.8: The labeling algorithm we implemented. This is a special case of the naive labeling algorithm, which is provably correct when S and F meet the sampling assumptions.

use $0 \leq -\cos(\beta) \leq 1$ as the weight of the connection between p and q.

Since two balls with different labels should intersect shallowly, the deeper the intersection, the more "likely" q will have the same label as p. Let α denote the angle of intersection between two balls. We set the weight of the connection between p and q to $0 \leq -\cos(\alpha) \leq 1$.

We summarize the labeling algorithm with the pseudo-code in Figure 3.8. Note that once a label is assigned, it is never changed. An algorithm which toggles labels to find a locally optimal labeling might be better, but we have not found it necessary.

3.3.4 Outputs

We tested the power crust using both well-known models and data we collected using a Cyberware M15 (tabletop) scanner. Almost all of the inputs immediately produced perfect surface reconstructions, requiring no tweaking.

In general, power crust faces are not triangles, and although the power crust interpolates the input samples, not all input samples are power crust vertices and not all power crust vertices are input samples. Though, power crusts have more faces than comparable triangulated surfaces, the shapes of the faces seem to conform to the shape of the object more nicely than polygonal models which are constrained to have triangular faces with the samples as vertices.

With some heuristics, we can even correctly reconstruct the sharp corners which was not possible with the previous Delaunay triangulation based algorithms. Discarding both poles of the badly-shaped Voronoi cells allows the power crust faces formed in nearby well-sampled regions to extend into the region of uncertainty and meet at a sharp corner. Figure 3.10 shows an example. Notice that a sharp edge can be reconstructed nicely even though there are no sample points on the edge itself.

Sharp cornered models of mechanical parts can be produced from fairly sparse

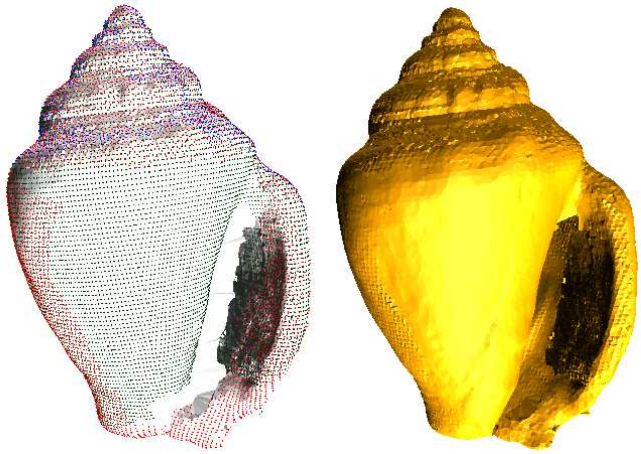

Figure 3.9: A reconstruction from four different scans, 37,073 samples total. The area in the interior, where there are no samples, is detected and intentionally left as a hole. A silhouette-based hole filling algorithm would have trouble filling this hole properly, but our method does not (it is filled in the solid model under the samples on the left).

samples, such as the Renault steering knuckle data shown in Figure 3.11, which was reconstructed from only 6002 points.

An ϵ-*offset surface* from F is a surface F', formed by the points x such that the distance from x to the nearest point on F is exactly ϵ. In terms of the MAT, the inside offset surface is formed by adding ϵ to the radius of every ball in the exterior MAT, and subtracting ϵ from every ball in the interior MAT; the outer offset surface can be defined analogously.

Computing an exact offset surface is difficult, in part because it can differ topologically from F. When F is represented by an approximate MAT, we can construct an approximate inside offset surface of P by increasing the radius of every outer polar ball by ϵ and decreasing the radius of every inner polar ball by ϵ, and then computing the power crust as usual. Since the power crust is always the water-tight boundary of a solid, the output cannot suffer from cracks or self-intersections. Figure 3.13 shows an example.

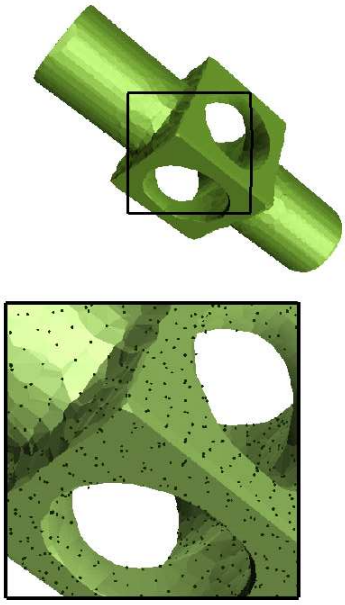

Figure 3.10: Sharp corners with no nearby features can be reconstructed without any samples on the edge itself. The corner between the cylinder and the top of the cube is not resolved as well because the polar balls on both sides of the surface are not very large compared to the sampling density.

Holes in the data seem to be filled in an appropriate and predictable way, for example the hard-to-scan spaces between the fingers, and the end of the wrist, in Figure 3.5. A 'watertight' surface representation is desirable in some contexts, for instance as input to a layered manufacturing system or for CSG. But we would also like the flexibility to produce surfaces which are not closed manifolds. In general, we would like to be able to fill in small holes in the data, but not cover over large ones: for instance, we might want to leave a hole at the bottom of the hand in Figure 3.5, while still filling in the hard-to-scan gaps between the fingers. Big holes in the data need not lie on the convex hull or even on any silhouette of the object; for instance the hole inside the shell in Figure 3.9 where the scanner could not reach the visible surface.

On well-sampled regions of the surface, inner and outer polar balls cannot

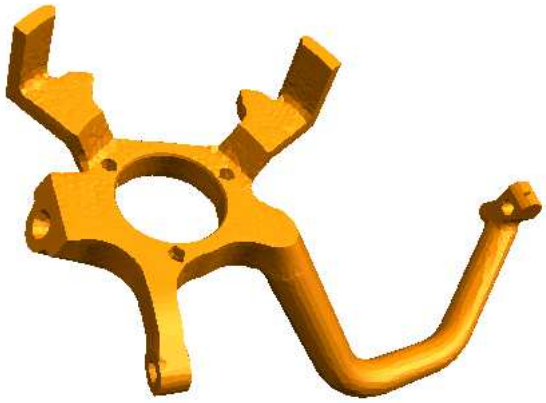

Figure 3.11: All the holes, and even the small notch at the end of the arm, are correct. This data was an example of something beyond the capabilities of the algorithm of [2]

intersect deeply. At a hole, inner polar balls can bulge out of the object, as in Figure 3.6, and outer polar balls can bulge into the interior. A power crust face formed by a deeply intersecting pair of polar balls, one inner and one outer, fills in the hole in the surface. When the intersecting pair of balls is large, we can choose to omit the face from the power crust. Examples are the sea shell in Figure 3.9 and the foot in Figure 3.6.

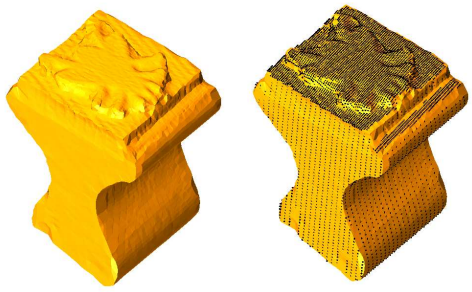

Figure 3.12: A rubber stamp model, and the imperfect laser range data (9755 points) used to produce it. Six scans were combined, with the top sampled at a higher resolution than the sides and bottom. Note the sharp corners on the sides, and the fill-in where scans fail to overlap.

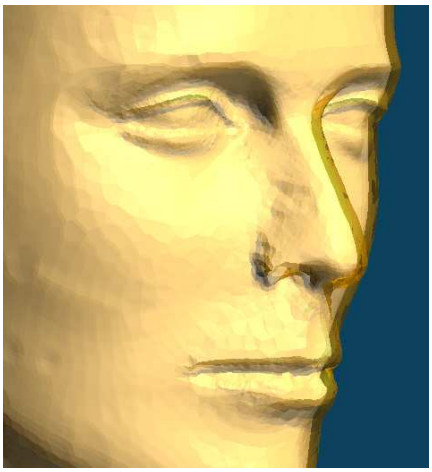

Figure 3.13: An approximate inner offset surface. The transparent surface is the original. Like the power crust, the offset surface is always the watertight boundary of a solid.

Chapter 4

The Delaunay triangulation of surface data

The power crust algorithm in Section 3.3 produces an approximate medial axis of the surface called the power shape. Even when the surface is known, the power crust algorithm has been used to obtain only the approximate medial axis. We notice that if the surface is given, we can use the additional information of the surface connectivity to speed up the Delaunay triangulation. Then the Delaunay triangulation can be used for medial axis approximation and mesh generation of surface.

In this chapter, we consider the problem of computing the 3D Delaunay triangulation given the surface mesh. It can be thought of as an inverse problem to the surface reconstruction algorithms which select surfaces from the 3D Delaunay triangulation of samples.

We solve this problem in two steps. First, we present an algorithm which constructs the Delaunay tetrahedralization of S given a bounded degree spanning subgraph T of F. It accelerates the incremental Delaunay triangulation construction by exploiting the connectivity of the points on the surface. If the expected size of

the Delaunay triangulation is linear, we prove that our algorithm runs in $O(n \log^* n)$ expected time, speeding up the standard randomized incremental Delaunay triangulation algorithm, which is $O(n \log n)$ expected time in this case.

Second, we discuss how to find a bounded degree spanning subgraph T from surface mesh F and give a linear time algorithm which takes any triangulated surface with genus g and obtains a spanning subgraph with maximum degree at most $12g$ for $g > 0$ or three for $g = 0$.

We also report the results from our experiments with several Delaunay triangulation programs on surface data. We find that the size of the Delaunay triangulation grows linearly during incremental construction, and the main performance problem of these programs in solving large data is thrashing due to random ordering of input data. We address this problem in Chapter 5.

4.1 Introduction

Given a surface mesh F with a vertex set S and consisting of Delaunay triangles, we consider the problem of computing the Delaunay tetrahedralization of S. The problem has important applications including mesh generation and medial axis construction.

For arbitrary point sets in 3D, the randomized incremental algorithm has been the most popular solution for the 3D Delaunay triangulation. It incrementally constructs the Delaunay triangulation by adding points one by one in random order. The cost of insertion is dependent on the cost of locating the new point to be added in the Delaunay triangulation of the already inserted points. The standard randomized incremental algorithm which uses an optimal point location scheme (e.g., the history DAG [26]) is theoretically optimal in expected time: it takes $O(n \log n)$ expected time if the expected size of the Delaunay triangulation of n points is $O(n)$, and $O(n^{1+k})$ if the expected size is $O(n^{1+k})$ for $0 < k \leq 1$. Thus, for a Delaunay

triangulation of size $O(n^2)$ the randomized incremental algorithm is already optimal; the interesting case is the one in which the size is $O(n)$.

Though the complexity of the 3D Delaunay triangulation can be quadratic for the worst case, it appears to be linear, in practice. Some theoretical results support this. Dwyer showed that if points are generated uniformly at random from the unit ball the expected size of the Delaunay triangulation is linear [37]. Recently, many researchers have studied the complexity of the 3D Delaunay triangulation for the special case of the points on a surface. Golin and Na showed that if the points are drawn from a 2-dimensional Poisson distribution with rate n from the surface of a fixed *convex* polytope the probabilistic complexity of the Voronoi diagram is $O(n)$ [45]. Erickson showed, in contrast, that there exist smooth surfaces that have a uniform sample whose Delaunay triangulation has quadratic size [41]. But, he also proved that for any *fixed* smooth surface, the Delaunay triangulation of any uniform ϵ-sample has complexity $O(n^{3/2})$ as n goes to infinity [42]. Finally, in [10] Attali and Boissonnat proved a linear bound on the complexity of Delaunay triangulation of a "well-sampled" polyhedral surface.

We assume for the theoretical analysis of our algorithm that the expected size of the three-dimensional Delaunay triangulation is linear. In Section 4.4, we show the results of our experiments with samples from surfaces which suggest that this assumption is usually correct.

We accelerate the point location by exploiting the connectivity of the boundary surface on which the input points lie. This idea is inspired by Seidel's randomized incremental trapezoidation algorithm of polygons [63]. By periodically tracing the boundary of the polygon and locating its segments in advance, he obtains an expected $O(n \log^* n)$ time algorithm for trapezoidation of simple polygons ($\log^* n$ is the largest integer l so that $\log^{(l)} n \geq 1$). Devillers applied the idea to the problem of 2D Delaunay triangulation given the Euclidean minimum spanning tree [28]. Our

algorithm similarly traces the surface to speed up the location of unadded points later.

The improved running time of our algorithm is only proved in the case where the surface mesh is composed of Delaunay triangles. This case is important since many surface reconstruction algorithms produce surfaces that are subsets of the Delaunay triangulation. Though many such algorithms require the 3D Delaunay triangulation as a preprocessing step, there are some that avoid it. Attali and Lachaud [11] give a modified marching cubes algorithm [52] to construct an iso-surface all of whose triangles satisfy the Delaunay constraint in linear time. This is important because there are very efficient surface reconstruction algorithms by Hoppe et al. [48] and Curless and Levoy [27]. They extract a zero-set of a signed distance function defined on voxel grids as the reconstructed surface. By using Attali and Lachaud's method we can make those surfaces consist of Delaunay triangles.

The algorithms by Bernardini et al. [12] or by Funke and Ramos [44] construct surfaces consisting of Delaunay triangles in near-linear time without computing the 3D Delaunay triangulation of all samples. Also there are other ways to obtain Delaunay triangulated surfaces without the 3D Delaunay triangulation. Cheng et al. [19] triangulate skin surfaces using Delaunay triangles. Constructing the 3D Delaunay triangulation given any of these surfaces can be useful, for instance, in generating a 3D mesh or medial axis.

We present and analyze an algorithm which constructs the Delaunay tetrahedralization given a bounded degree spanning subgraph T consisting of Delaunay edges in Section 4.2. We show that the algorithm runs in $O(n \log^* n)$ expected time, if the expected size of the Delaunay triangulation is linear. In section 4.3, we discuss how to obtain a bounded degree spanning subgraph T from a surface F consisting of Delaunay triangles. We give a linear time algorithm to find a spanning subgraph with maximum degree at most $12g$ for $g > 0$ or three for $g = 0$ in any triangulated

surface with genus g.

4.2 Algorithm

For a point set P, we denote the Delaunay triangulation of P as $D(P)$. For the analysis, we restrict the boundary surface F to be a subset of the Delaunay triangulation $D(S)$ of S and assume that we are given a spanning subgraph T of F with bounded degree d. In section 4.3, we discuss how to obtain such a spanning subgraph T given F. We assume that the points of S are in general position. Let $s_1, s_2, ..., s_n$ be a random ordering of the vertices of S and let $S_i = \{s_1, ..., s_i\}$ for $0 \leq i \leq n$.

To efficiently locate s_i in $D(S_{i-1})$ to construct $D(S_i)$, randomized incremental Delaunay triangulation algorithms maintain a location query structure $Q(S_{i-1})$. For our analysis, we use the theoretically optimal history DAG by Clarkson et al. [26] which uses all the intermediate Delaunay tetrahedra created to construct our point location query structure. A newly created Delaunay tetrahedron becomes a child of the Delaunay tetrahedra that are destroyed by it. To locate s_i in $D(S_{i-1})$, using the history DAG, we start at the root and traverse all the intermediate Delaunay tetrahedra whose circumspheres contain s_i until we find the one in $D(S_{i-1})$ that contains s_i. After locating s_i in $D(S_{i-1})$ using $Q(S_{i-1})$, we update both the Delaunay triangulation $D(S_{i-1})$ and the query structure $Q(S_{i-1})$ to produce $D(S_i)$ and $Q(S_i)$.

In the worst case when the size of $D(S_i)$ is $O(i^2)$, locating s_i in $D(S_{i-1})$ using $Q(S_{i-1})$ takes $O(i)$ expected time and updating $D(S_i)$ and $Q(S_i)$ takes $O(i)$ expected time. Thus the 3D Delaunay triangulation of n points takes $O(n^2)$ expected time. But, if the expected size of $D(S_i)$ is $O(i)$, the location takes $O(\log i)$ expected time and the update takes $O(1)$ expected time. For the remainder of the analysis, we assume that the expected size of $D(S_i)$ is $O(i)$, for all i.

Tracing DT() {

1. Let $h = 1$.
2. For $1 \leq i \leq n$ do
   (a) If $i = N_h$, trace T through $D(S_{N_h})$ to determine $d_{N_h}(s_k)$ for all $k > N_h$. Let $h = h + 1$.
   (b) Locate s_i in $D(S_{i-1})$ using $Q(S_{i-1})$ starting from $d_{N_{h-1}}(s_i)$.
   (c) Insert s_i and update $D(S_i)$ and $Q(S_i)$.

}

Figure 4.1: Tracing Delaunay Triangulation (TracingDT) algorithm

Our algorithm, given in Figure 4.1, is inspired by Seidel's trapezoidation algorithm for simple polygons [63]. Let $N_h = \lceil n/\log^{(h)} n \rceil$. We denote $d_i(v)$ of a point v to be the Delaunay tetrahedron of $D(S_i)$ which contains v. Periodically, we execute a tracing step, in which we traverse the spanning subgraph T and locate all the uninserted points in the intermediate Delaunay triangulation in advance. Later, when we actually add a new point, we start from the Delaunay tetrahedron located in the last tracing step instead of at the root of the history DAG.

4.2.1 Analysis

We first analyze the cost of each tracing step 2a. Since $N_{\log^* n+1} > n$, step 2a is done only $\log^* n$ times. Each tracing step N_h for $1 \leq h \leq \log^* n$ is done by traversing the bounded degree spanning subgraph T and recording $d_{N_h}(s_k)$ for all $k > N_h$. Thus the cost is determined by the number of intersections between T and $D(S_{N_h})$. Here we need the restriction that T is a subset of $D(S)$. To count the expected number of intersections, we make some definitions and prove the following lemma about the intersection of a triangle of $D(R)$ for $R \subseteq S$ and a Delaunay edge of $D(S)$.

A sphere is *empty* of S if its interior contains no point $s \in S$. The *region* of Delaunay triangle abc is the union of the two empty circumspheres of the two

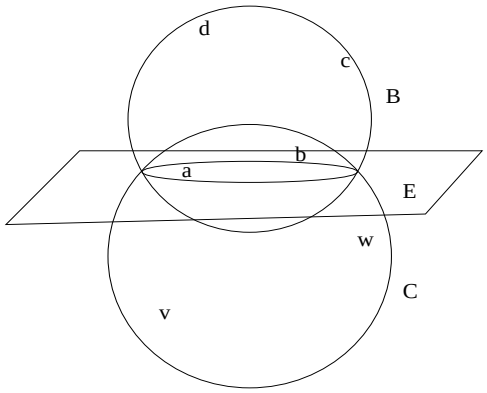

Figure 4.2: Proof of Lemma 7

Delaunay tetrahedra containing *abc*.

Lemma 7 *Let abc be a triangle in $D(R)$ for any $R \subseteq S$. If edge vw in $D(S)$ intersects abc, at least one of $\{v, w\}$ lies in the region determined by abc in $D(R)$.*

Proof: Suppose for the purpose of contradiction that the edge vw intersects abc but both v and w lie outside the region determined by abc in R. Let $abcd$, $abce$ be the two Delaunay tetrahedra adjacent to abc in $D(R)$. Consider $K = \{a, b, c, d, e, v, w\}$ Then, the tetrahedra $abcd$, $abce$ and the edge vw are all in $D(K)$ because they are in $D(R)$ and $K \subset R$. Let B be the circumsphere of $abcd$, and C be the circumsphere of a Delaunay tetrahedron $vwxy$ (x, y may be any pair of $\{a, b, c, d, e\}$) that contains vw in $D(K)$. Let E be the plane supporting the circle of the intersection of C and B. Both B and C are empty of K, because $abcd$ and $vwxy$ are in $D(K)$. So, $\{a, b, c, d\}$ and $\{v, w, x, y\}$ lie on the opposite sides of E as in Figure 4.2. Thus, vw cannot intersect abc. This is a contradiction. Therefore at least one of $\{v, w\}$ lies in the region determined by abc.

□

Lemma 7 may also be phrased and proved using the standard lifting trans-

formation which maps the Delaunay triangulation in $\mathbb{R}^3$ to a lower convex hull in R^4.

Lemma 8 *The expected number of intersections of $D(S_i)$ and T is $O(n)$.*

Proof: By Lemma 7, if an edge vw in T intersects a triangle t in $D(S_i)$, at least one of $\{v, w\}$ lies in the region determined by t in $D(S_i)$. So we count each edge of T adjacent to v as something that might intersect the triangle t, so each vertex v in the region of a triangle t can contribute to the number of intersections at most the degree $d(v)$ of v in T.

Let I denote the number of intersections of $D(S_i)$ and T. We bound I by summing over each triangle t of $D(S_i)$ the degrees $d(v)$ of vertices v of T that lie in the region of t. This is equivalent to summing over each of the vertices v of T the degree $d(v)$ of v times the number of regions containing v, which is bounded by the maximum degree d times the number of regions containing v. Since each Delaunay tetrahedron contains four Delaunay triangles, a vertex in a Delaunay circumsphere is counted as belonging to the four regions of the Delaunay triangles. Thus :

$$\begin{aligned} I &< \sum_{\substack{\textit{triangles} \\ t \in D(S_i)}} \Big(\sum_{\substack{\textit{vertices } v \in T \\ \textit{in region of } t}} d(v) \Big) \\ &= \sum_{\substack{\textit{vertices} \\ v \in T}} ((\#\text{ of regions containing } v) * d(v)) \\ &\leq d \times \sum_{\substack{\textit{vertices} \\ v \in T}} (\#\text{ of regions containing } v) \\ &= 4d \times \sum_{\substack{\textit{vertices} \\ v \in T}} (\#\text{ of Del. circumspheres containing } v) \end{aligned}$$

We obtain the expected number of the Delaunay circumspheres of $D(S_i)$ that contain v using backwards analysis. If we insert vertex v in $D(S_i)$, the number of the Delaunay circumspheres of $D(S_i)$ which contains v is the number of Delaunay tetrahedra in $D(S_i)$ destroyed by the insertion of v. Since we are assuming that

the expected size of $D(S_i)$ is $O(i)$, the expected number of the Delaunay tetrahedra destroyed by the insertion of v is $O(1)$. Hence, the expected number of intersections is $4nd \times O(1) = O(n)$.

□

Theorem 9 *Tracing T in $D(S_i)$ takes $O(n)$ expected time.*

Proof: Tracing time is determined by the number of intersections of $D(S_i)$ with T. By Lemma 8, the expected number of the intersections of $D(S_i)$ and T is $O(n)$.

□

So each tracing takes $O(n)$ expected time and since we trace $\log^* n$ times, all the tracing steps take $O(n \log^* n)$ expected time. Then the rest of the analysis is to determine the cost of locating and inserting a new point given the information from the tracing. This is basically the same as the proofs for the corresponding theorems of Seidel's trapezoidation algorithm [63] and Devillers' 2D Delaunay triangulation algorithm [28]. Nevertheless, we include the following lemma and theorem for completeness.

Lemma 10 *If $d_k(s_i)$ is known, the expected cost of locating s_i in $D(S_{i-1})$ is $O(\log(i/k))$.*

Proof: The cost of locating a point s_i in $D(S_{i-1})$ given $d_k(s_i)$ is proportional to the number of Delaunay tetrahedra in $D(S_j)$ for $k < j < i$ whose circumsphere contains s_i, which is $O(\log i) - O(\log k) = O(\log(i/k))$.

□

Theorem 11 *The algorithm Tracing DT runs in $O(n \log^* n)$ expected time.*

Proof: By Theorem 9, step 2a takes $O(n)$ expected time for fixed h and is executed $O(\log^* n)$ times, thus the total cost of for step 2a is $O(n \log^* n)$ expected time.

For $N_{h-1} < i \leq N_h$, locating s_i takes $O(\log(i/N_{h-1}))$ expected time by Lemma 10. Since $i \leq n$, $O(\log(i/N_{h-1}) = O(\log(n/N_{h-1}) = O(\log^{(h)} n)$. For fixed h, step 2b is executed at most N_h times. Thus locating s_i for all $N_{h-1} < i \leq N_h$ takes $O(\log^{(h)} n) \times N_h = O(n)$ expected time. Since $h \leq \log^* n$, locating s_i for all $i \leq N_{\log^* n}$ takes $\log^* n \times O(n)$ expected time. For $N_{\log^* n} < i \leq n$, since $N_{\log^* n} \geq n/e$, locating s_i takes $O(\log(i/N_{\log^* n})) = O(1)$. Thus, the total cost of step 2b is $O(n \log^* n)$ expected time.

By the linear size assumption of the Delaunay triangulation, step 2c takes $O(1)$ expected time for each i, so the total cost is $O(n)$ expected time.

□

4.3 Input to the algorithm

Even if the surface F is a subset of the Delaunay triangulation, it can have an arbitrarily large maximum degree (e.g., a point can have an arbitrary number of nearest neighbors). The analysis in the previous section is based on the assumption that we are given a spanning graph T of surface mesh F which is a subset of the Delaunay triangulation $D(S)$ and whose maximum degree d is bounded. Here we discuss how to obtain such input spanning graph T from F.

4.3.1 Special cases

One such spanning graph is the Euclidean minimum spanning tree of S. The Euclidean minimum spanning tree $EMST(S)$ of a point set S is the spanning tree of S that minimizes the sum of the lengths of the edges in the tree. Since it has bounded

degree 6 in 2D and 12 in 3D [61], and is a subset of its Delaunay triangulation, the proofs of the previous section hold if we are given $EMST(S)$. Though $EMST(S)$ is usually constructed by computing $D(S)$ and running a minimum spanning tree algorithm on the edges of $D(S)$, it can also be computed without Delaunay triangulation in $O(n^{4/3}\log^{4/3} n)$ expected time. using the algorithm by Agarwal et al [1]. It is not of course guaranteed that a given surface triangulation F contains $EMST(S)$, but if F contains $EMST(S)$, we can find $EMST(S)$ from F in $O(n)$ expected time [49].

Some surface meshes already have bounded degree and we can just use any spanning tree of F as T. For instance, the iso-surfaces constructed from voxel data using the marching cubes algorithm [52] have a bounded degree. Though in general, the output of the marching cubes algorithm is not guaranteed to be Delaunay triangles, Attali and Lachaud's [11] modified marching cubes algorithm constructs iso-surfaces composed of Delaunay triangles.

Lemma 12 *Let F be the output of the marching cubes algorithm on a rectangular voxel grid. Then the degree of vertices in F is bounded.*

Proof: By definition, the vertices of F are always on the voxel edges. A voxel edge is adjacent to at most 4 voxels. So for each vertex of F, the iso-surface edges adjacent to the vertex always lie in the 4 adjacent voxels. Each vertex lies in a loop in a voxel. Each loop is a $n-gon$ in $\mathbb{R}^3$ where $3 \leq n \leq 7$ so after triangulation the maximum degree of a vertex inside a loop is at most 6. Hence the maximum degree of a vertex of the surface is $6 \times 4 = 24$.

□

Similarly, in case of the *locally uniform* samples defined by Funke and Ramos [44], it is shown that the surface Delaunay triangulation has a bounded degree. They

bsg(G,C) {

1. If $|V(G)| = 3$, output $H = G$, and we're done.
2. If C is a triangle, we choose a cycle vertex v with degree greater than two as z.
   Otherwise, we choose z from a list of cycle vertices with no chord.
3. Let $z_1, z_2, ..., z_m$ be the neighbors of z in the clockwise order such that $z_1, z_m \in V(C)$.
   Let $G' = G - z$ and C' be the outer cycle of G'.
   Update chord information for new cycle C'.
   $H' = bsg(G', C')$
   If $m > 2$, output $H = H' - z_1z_2 - z_mz_{m-1} + z_1z + z_2z + z_mz$.
   If $m = 2$, $H = H' - z_1z_2 + z_1z + z_2z$.

}

Figure 4.3: Bounded degree spanning subgraph(bsg) algorithm: G is the triangulation homeomorphic to a disk and C is its boundary cycle.

also give a decimation algorithm which decimates dense samples into locally uniform samples in $O(n \log n)$ time.

4.3.2 Bounded degree spanning subgraph algorithm

If the surface F is composed of Delaunay triangles but not guaranteed to have bounded degree, we need to find a bounded degree spanning tree T from F. We give a linear time algorithm to construct a spanning subgraph with maximum degree at most three if $g = 0$ and $12g$ otherwise from any triangulation of a surface with genus g.

If the surface has genus greater than zero, Thomassen [66] showed that any triangulation of orientable surface S_g with a fixed genus g contains a spanning tree of maximum degree at most $5 \times 2^g - 2$. We obtain an improved bound $12g$ on the maximum degree using the algorithm by Lazarus et al [50]. They find in $O(gn)$ time a set of $2g$ cycles on triangulated surfaces such that cutting the surface along them

yields a topological disk called a polygonal schema. So we can convert triangulated surfaces with genus greater than zero into a topological disk in linear time. Since a vertex can appear in a cycle only once and there are $2g$ cycles, after cutting the surface along $2g$ cycles, a vertex may appear at most $4g$ times. Below we give a linear time algorithm to construct a spanning subgraph with maximum degree three given a triangulation homeomorphic to a disk. By identifying the vertices belonging to the cycles, we can obtain a spanning subgraph with maximum degree at most $4g \times 3 = 12g$.

Our algorithm for finding a spanning subgraph in a triangulation homeomorphic to a disk is described in Figure 4.3. It is inspired by the proof of Theorem 3.1 in [66] but gives a different, constructive, and somewhat simpler proof for the existence of a spanning subgraph with maximum degree three. We denote the set of the vertices of a graph G as $V(G)$. A *chord* of a vertex v in a cycle is a non-cycle edge between v and a vertex in the cycle. The algorithm is recursive. Each call $bsg(G, C)$ for the surface triangulation G and its outer cycle C outputs the spanning subgraph H with maximum degree at most three containing C. Initially, we start from the surface triangulation G. If $|V(G)| = 3$, G is the spanning subgraph H and we're done. Otherwise, we choose a cycle vertex z to remove from G to construct G' and its outer cycle C', and call $bsg(G', C')$ to obtain a spanning subgraph H' of G'. Then, we construct H from H' as in Figure 4.4; that is, we remove z_1z_2 and z_mz_{m-1} and add z_1z, z_2z and z_mz where $z_1, ..., z_m$ are the neighbors of z in a clockwise order.

Lemma 13 *Given a triangulation G of S_0, we let any triangle of G be our initial cycle C. The algorithm $bsg(G, C)$ outputs a spanning connected subgraph of H with the maximum degree at most three.*

Proof: First, to guarantee the termination of the algorithm, we need to show that during every recursive call to $bsg(G', C')$ we can find a vertex $z \in V(C')$ to remove.

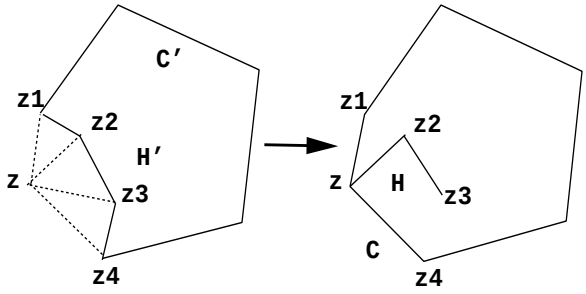

Figure 4.4: Constructing H from H'

If C' is a triangle, there is a vertex in $V(C')$ that has degree greater than two, because otherwise $|V(C')| = 3$ and we're done. So assume that C' has n vertices (with $n > 3$). Since any triangulation of an n-gon has only $n - 3$ diagonals, there is a vertex with no chord. Therefore, there is always a vertex $v \in V(C')$ to remove during every recursive call $bsg(G', C')$.

Second, we show that for each recursive call to $bsg(G', C')$, the outer cycle C' is homeomorphic to a circle. Whenever we remove a vertex z from C, its neighbors $z_1, .., z_m$ as defined in Figure 4.3 become the new vertices in the cycle. Since z has no chord, $z_2, ...z_{m-1}$ are not in C. C' is homeomorphic to a circle since it is obtained from C by removing zz_1, zz_m and adding $z_1z_2, z_2z_3, ..., z_{m-1}z_m$.

Finally, we prove that H is a spanning connected subgraph of F with maximum degree at most three. We first show that every recursive subcall to $bsg(G, C)$ returns a spanning subgraph H of G with maximum degree at most three such that $C' \subset H'$. We prove it by induction on $|V(G)|$. If $|V(G)| = 3$, $H = G$. So assume that $|V(G)| \geq 4$. Let z be any vertex with no chord and apply the induction hypothesis on $G - z$ to obtain H'. If the degree of vertex z is greater than two ($m > 2$, in Figure 4.3), $z_2, ..., z_{m-1} \in V(C)$. Then, H is obtained from H' by deleting the edges z_1z_2, z_mz_{m-1} and adding the edges z_1z, z_2z and z_mz. If the degree of vertex z is two ($m = 2$), then H is obtained from H' by deleting the edge z_1z_2 and adding the edges z_1z and z_2z. Either case, the degree of z in H is at most three, and the

degrees of the rest of the vertices are the same as or less than in H'. Therefore, H is a spanning subgraph of G with maximum degree at most three.
□

Lemma 14 *The algorithm $bsg(G, C)$ terminates in linear time.*

Proof: For the analysis of the time complexity of the algorithm, we assume that for every vertex we are given a list of its neighbors in clockwise order. We maintain the cycle C as a doubly linked list and each vertex has markers for whether it is a cycle vertex and whether it has a chord. We also maintain a *no-chord list* - a list of cycle vertices that have no chord. At every step, we just choose the first vertex in no-chord list as z.

Removing a vertex z from G and modifying C to be the new outer cycle C' of $G' = G - z$ takes time proportional to the degree of z.

Also we need to update chord records for the new cycle C'. For each new cycle vertex $v \in V(C') \setminus V(C)$, we look at all its neighbors and if a neighbor w of v is a cycle vertex, we mark v as a chord vertex, and if w was in the no-chord list, we remove it from no-chord list. If v does not have a chord, we add v to the no-chord list. This takes $\sum_{v \in V(C') \setminus V(C)} d(v)$.

Since every vertex becomes a vertex in C just once and is removed just once, the total execution time is proportional to $\sum_{v \in V(G)} d(v) = O(2|E(G)|) = O(|V(G)|)$.
□

4.4 Experiment

In this section, we present results from our experiments with Delaunay triangulation programs on sets of points which lie on or near two-dimensional surfaces in $\mathbb{R}^3$. Our

surface reconstruction algorithms in Chapter 3 use the 3D Delaunay triangulation of samples as a preprocessing step, and we find that the Delaunay triangulation is the bottleneck. The motivation of the experiment is to figure out the main problem of the 3D Delaunay triangulation computation in this case to improve the performance.

The Delaunay triangulation based surface reconstruction algorithm is considered slow because of the quadratic worst-case complexity of the 3D Delaunay triangulation. Our first goal is to verify that for the case of samples from surfaces, the Delaunay triangulation is almost always linear, in practice. The second goal is to identify the bottlenecks of the Delaunay triangulation computation.

4.4.1 Experiment Set-up

We use three robust Delaunay triangulation programs. Clarkson's `hull` is an older program for general dimension convex hull and Delaunay triangulation. Delaunay function of Shewchuk's `pyramid` and `Delaunay hierarchy` function of the `CGAL` library [17], represent a new standard of quality.

The three programs all use the randomized incremental algorithm but different point location scheme. If we assume that the Delaunay triangulation is always of linear size, then point location is the only operation that is not $O(1)$ expected time per insertion. `Hull` uses a theoretically optimal $O(\lg n)$ history DAG [26], which is memory intensive. `Pyramid` uses a simple $O(n^{1/4})$ *jump-and-walk* strategy [59]. `CGAL Delaunay hierarchy` uses a few levels of intermediate Delaunay triangulations [29] as a search structure, something like a skip list, known to be optimal.

We use data from laser range scanner and samples from iso-surfaces. The `dragon` data (1.7 million points) from the Stanford 3D Scanning Repository. It comes from a Cyberware range scanner; it contains noise and is very non-uniform. `MTD` (185,000 points) is from a protein electron density iso-surface, selected via

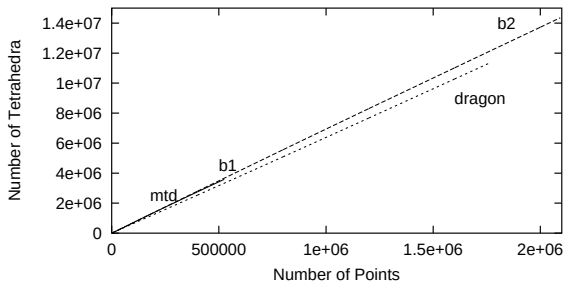

Figure 4.5: The number of Delaunay tetrahedra per number of input points added.

marching cubes. `B-1` (525,000 points) and `B-2` (2 million points) were obtained by applying butterfly subdivision to such an iso-surface, once and twice, respectively, to get a smooth and dense point set. Timing does not include file I/O and all experiments are done in Linux on an Intel Pentium III (864 MHz) with 511M RAM.

4.4.2 Results

All of our datasets produced linear-sized Delaunay triangulations and the size of all intermediate Delaunay triangulations was linear. The number of Delaunay tetrahedra created/destroyed per insertion averaged about 27/20 and the average ratio of the number of Delaunay tetrahedra to the number of input points is about 6-7. See Figure 4.5.

The shape of the overall performance profile of all the programs was similar: near-linear running time until memory is exceeded, at which point thrashing occurs. See Figure 4.6. It took about 400 seconds for `CGAL Delaunay hierarchy` and about 350 seconds for `pyramid` to compute the Delaunay triangulation for a million points.

Contrary to our expectation that the point location dominates the running

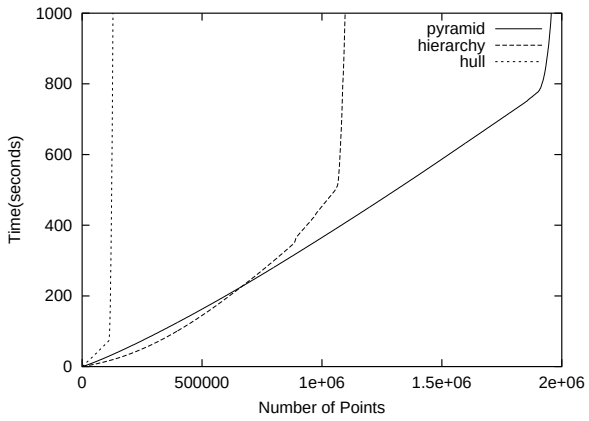

Figure 4.6: `Hull`, `CGAL Delaunay hierarchy`, and `pyramid` begin thrashing around 120,000, 1,000,000, and 1,800,000 points, respectively. `Hull` timing for `MTD` data, `CGAL Delaunay hierarchy/pyramid` for `B-2` data.

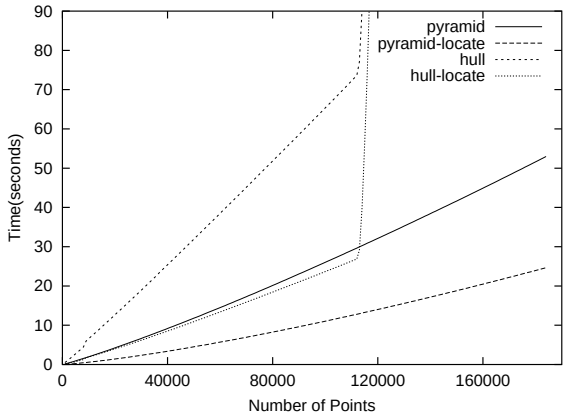

Figure 4.7: Total point location time vs. the total time for `hull` and `pyramid` on `MTD` data

time, it forms a significant, but not overwhelming fraction of the overall running time. See Figure 4.7.

4.5 Discussion

Our algorithm speeds up the location of a new point in randomized incremental Delaunay triangulation algorithms. But, the actual insertion cost of the new point and update of the Delaunay triangulation is not affected. Contrary to our expectation that the location time dominates the whole running time, our experiments in Section 4.4 show that the location time and update time are pretty well balanced for several real Delaunay triangulation programs. Hence, this algorithm is not expected to improve the performance of the Delaunay triangulation programs by more than a factor of two.

Chapter 5

Incremental construction con BRIO

In Section 4.4, we have seen that the main performance problem of the randomized incremental algorithm for computing the Delaunay triangulation of large data is thrashing due to the inherently bad random memory access pattern. The random ordering is required to guarantee the optimality of the algorithm. Here we define a new insertion order which we call the *biased randomized incremental order* (BRIO) that removes enough randomness to significantly improve performance in practice, but leaves enough randomness so that the algorithms remain theoretically optimal.

5.1 Introduction

The randomized incremental algorithm is the most popular choice for 3D Delaunay triangulation of a set of input points, because it is worst-case optimal and (compared to the alternatives) easy to implement robustly. But as we have seen in the experiments in Section 4.4, it performs badly when the input data gets too big. Randomized incremental algorithms access the geometric data structures randomly,

and random access to large data structures works very poorly with modern memory hierarchies.

Virtual memory systems cache recently used data in memory, on the assumption of *locality of reference*, that is, that recently used data is likely to be used again soon. Randomized incremental programs violate this assumption, and soon after the data structure exceeds the size of physical memory, thrashing occurs and the program grinds (audibly!) to a halt [21]. This limits the size of the Delaunay triangulations that can be computed in practice.

A simple fix is to insert points in an order which improves the locality of reference, while preserving enough randomness to retain the optimality of the algorithm. In Section 5.3 we present such a scheme, which we call a *biased randomized insertion order* (BRIO). We prove in Sections 5.5, 5.6, 5.7 that this order is no worse than a completely randomized insertion order in terms of asymptotic complexity: it gives an optimal algorithm for Delaunay triangulation in the worst case, and also under less pessimistic but more realistic assumptions about the output complexity. Our evidence that it indeed reduces or eliminates thrashing is experimental: in Section 5.8 we use a BRIO with `hull`, `CGAL Delaunay hierarchy` function, and `pyramid`, and can solve much larger problems than were possible with a completely randomized insertion order.

5.2 Related work

The development of randomized incremental algorithms and their analysis was a major project of computational geometry in the late eighties and early nineties, as described in textbooks [13, 57] and surveys [58, 62]. We touch on a few relevant highlights. A classic paper by Clarkson and Shor [25] showed that the randomized incremental paradigm could be applied to many problems, and gave a general analysis. Mulmuley [54, 55] and Clarkson, Mehlhorn and Seidel [26], among oth-

ers, extended this theory. Seidel [64], harking back to an idea in an early paper of Chew [20], popularized a simplifying idea called *backwards analysis* which became the standard tool for analyzing randomized incremental algorithms. It requires that after inserting r points, every inserted point has the same probability to be the last one in the random permutation of the set of r points. This is not true with a BRIO, so we cannot see how to apply backwards analysis. Instead we build on results from the earlier work, in particular the bounds on $(\leq k)$-sets from Clarkson and Shor and from Mulmuley.

The traditional approach to thrashing is to develop explicit out-of-core algorithms, usually using divide-and-conquer. Though there are out-of-core algorithms for 2D Delaunay triangulation [9, 47, 68], extending it to 3D in a practical way seems to be challenging.

Clarkson and Shor's random sampling paradigm [25] provides a way to split geometric problems into subproblems each guaranteed to be small. This gives a simple and near-optimal divide-and-conquer algorithm. But each input point is assigned to multiple subproblems so that the total size of the subproblems are much bigger than the original input size. Moreover, it requires numerical predicates of a high degree, which are expensive to compute reliably.

We can avoid this problem by computing the part to be merged first and dividing the subproblems subsequently, thus eliminating the overlap of the output. This scheme has been called the "marriage-before-conquest" paradigm. Edelsbrunner and Shi [38] gave a divide-and-conquer algorithm for 3D convex hull which *first* solves the part where the subproblems merge using projection and 2D convex hull before computing each subproblem. Blelloch et al. [14] applied Edelsbrunner and Shi's idea to the problem of 2D Delaunay triangulation to give a projection-based parallel algorithm which does $O(n \log n)$ work and has $O(\log^3 n)$ depth (parallel time). However there are several challenges in extending their algorithm to compute the 3D

Delaunay triangulation. To compute the boundary where the subproblems merge, we need the 3D convex hull computation on the input points which would itself requires an out-of-core or distributed implementation. A second problem is that in the base cases they call a special subroutine which does not generalize directly to the 3D case.

Chan et al. [18] gave an output-sensitive algorithm based on a divide-and-conquer approach similar to Edelsbrunner and Shi's. Though the algorithm is asymptotically near-optimal, the implementation is not straightforward and does not seem to be practical at all.

Cignoni et al. [23] proposed a parallel 3D Delaunay triangulation algorithm. They split the input points using a splitting plane and construct the Delaunay tetrahedra intersected by the plane first, and then recurse on each subproblems. But there is no theoretical guarantees for their algorithm.

In conclusion, though there are several theoretical works on a divide-and-conquer or parallel Delaunay triangulation, there seems to be no satisfactory solution for practical 3D Delaunay triangulation of large data. Instead, we stick to the randomized incremental paradigm but define an insertion order that, heuristically, helps the memory hierarchy work effectively.

Devillers and Guigue [30] considered a different kind of partially randomized insertion order, for handling constructions for which the data is provided sequentially rather than all at once. Arriving data can be stored and reshuffled (randomly) in a buffer of limited size before it has to be inserted into the data structure. They showed that the expected running time degrades as a smaller shuffling buffer is used and the randomness is more limited. For an analysis similar to ours to work, it seems important to have at least a random sample of all of the points available at the beginning of the construction.

5.3 Insertion order

We define a *biased randomized insertion order* for a set P of n input objects, which we shall call points. The points are inserted in *rounds*, from round 0 through round $\lg n$. For simplicity, we assume that n is a power of 2.

To allocate points to rounds, we choose each point independently with probability $1/2$ to be inserted in the final round. We choose each of the remaining points independently with probability $1/2$ to be inserted in the next-to-last round, and so on. When we get to round zero, we choose any remaining points with probability one. Thus the probability that a point is chosen in round $i > 0$ is $2^{i-1}/n$, and the remaining probability $1/n$ goes to the event that the point is chosen in round zero. As far as our proof of asymptotic optimality is concerned, the points can be inserted in an arbitrary order within each round.

Our procedure could be viewed alternatively as a sort of randomized divide-and-conquer procedure. We first select half of the points randomly and recursively build the data structure for them. Then we insert the remaining half of the points incrementally.

This restricted requirement of randomness allows us to bias the insertion order to favor locality. The approach we take here is to organize the points into *blocks* which respect locality within three-dimensional space; in our experiments we use the cells of an octree or a *kd*-tree (e.g., [13, Chapter 5]). Within each round, we group the points by block, and we order the blocks within a round to favor locality in $\mathbb{R}^3$ by taking them in depth-first order. Within each block we order the points randomly.

The intuition is that in the early rounds the insertions tend to be sprinkled nearly randomly across all the data, producing a nicely balanced data structure, while in the later rounds they are clustered within blocks, accessing local regions of the data structure mostly independently.

Our algorithm makes an effort to choose an ordering that respects locality in $\mathbb{R}^3$. This only indirectly attacks the question of locality in the layout of the data structure in virtual memory, which is what we really need to address. The hope is that out algorithm would also lead to a stronger coherence between closeness in space and in memory than the purely randomized approach. However, this depends on the implementation of the storage management scheme and may be hard to predict.

5.4 Analysis Setup

We analyze the use of a BRIO in the context of the incremental construction of a three-dimensional Delaunay triangulation. First, we review the analysis of the usual randomized incremental algorithm for three-dimensional Delaunay triangulation [25, 26] (see also [13, 57]). The running time can be divided into two parts, the time required to find where each new point should be inserted into the Delaunay triangulation (*location time*) and the time required to delete old tetrahedra and create new tetrahedra so as to actually perform the insertion (*update time*). Point location can be done in various ways; the theoretically optimal methods have been shown to take $\Theta(X(n))$ time, where $X(n)$ is a quantity known as the *total conflict size*.

The total update time is $\Theta(C(n))$, where $C(n)$ the total number of tetrahedra which appear over the course of the construction.

In the worst case, the size of a Delaunay triangulation of n points in $\mathbb{R}^3$ is $O(n^2)$, and it turns out this is also the bound on the total conflict size and hence the running time. But in practice the size of the Delaunay triangulation is often $O(n)$.

The "realistic" case. We define a "realistic" instance to be a point set P for which the Delaunay triangulation of a random subset of r points has expected size $O(r)$, for every r. (This definition is with respect to some fixed constant in the O-notation.) Experiments with data from various sources [21] corroborate the claim that practical instances have this property. In this realistic case, there is a better bound of $O(n \log n)$ on the total conflict size and the running time.

More generally, we can consider a probability distribution of "practical" instances P, and the $O(r)$ size requirement should hold for a random (uniform) r-subset of a random problem instance (according to the given distribution). Our results about expected running time are then average-case results (in addition to the expectation with respect to the random choices of the algorithm).

We show that our new algorithm remains optimal using a biased randomized insertion order both in the worst case and in the "realistic" case.

The general framework. We will use some terminology of Mulmuley. The four vertices of a tetrahedron are known as its *triggers*, and the other points of P contained in its circumsphere, are called its *stoppers*. Every choice of four points of P as triggers determines a possible tetrahedron, but in a particular run of the randomized incremental construction not every possible tetrahedron *appears* as part of one of the intermediate triangulations, or in the final triangulation. A tetrahedron appears in some Delaunay triangulation if and only if all of its triggers are selected for insertion before any of its stoppers. The probability that a tetrahedron appears during the construction thus depends in part on its number s of stoppers; if $s = 0$, for instance, the tetrahedron belongs to the final Delaunay triangulation and the probability that it appears is one.

The structure of the analysis follows the scheme in the early papers [25, 55, 56]. Let p_s be the probability that a tetrahedron with s stoppers appears in some triangulation of the construction, and let k_s be the number of tetrahedra

with s stoppers for point set P. Then we can write the expected total number of tetrahedra that appear as

$$E[C(n)] = \sum_{s=0}^{n} k_s p_s$$

The total conflict size is the sum, over all tetrahedra τ which ever appear in the construction, of the number of stoppers of τ. Thus, the expected total conflict size is

$$E[X(n)] = \sum_{s=0}^{n} k_s p_s s.$$

Now all we need are upper bounds on k_s and p_s.

5.5 One Tetrahedron

Let us consider a tetrahedron τ with s stoppers. If s is zero, τ has to appear in the Delaunay triangulation. If s is non-zero *and* τ appears in some intermediate triangulation, inevitably in some later insertion one of τ's stoppers will be chosen and τ will be *popped* (it will no longer be part of the triangulation). In other words, the probability that τ appears is the same as the probability that τ is eventually popped.

We bound the probability p_s that τ appears by considering each round i of the BRIO, building on the following.

Observation 15 *The probability that a point $x \in P$ is chosen in round i is $2^{i-1}/n$ for $i \geq 1$ and $1/n$ for $i = 0$, independently of x. Different points are assigned to rounds independently.*

□

Lemma 16 *A tetrahedron τ with s stoppers appears with probability $p_s = O(1/s^4)$.*

Proof: If τ is popped in round i, it must be case that all triggers of τ were chosen in or before round i and that the first stopper is chosen in round i. So the probability that τ appears is at most

$$p_s = \sum_{i=0}^{\lg n} P[\text{ first stopper in round } i\ ] \cdot P[\text{ all triggers in round } i \text{ or earlier }] \qquad (*)$$

Let a_i be the probability that a particular point has been chosen at the beginning of round i, so that

$$a_0 = 0,\ a_1 = 1/n,\ a_2 = 2/n,\ a_3 = 4/n,\ \ldots,\ a_{\lg n+1} = 1,$$

and in general $a_i = 2^{i-1}/n$. Then we have

$$a_{i+1} = 2a_i \qquad (**)$$

for $i > 0$. The probability that all of the triggers are chosen in or before round i is a_{i+1}^4 since the events are independent.

The probability that the first stopper is selected in round i is denoted by $G_i = (1 - a_i)^s - (1 - a_{i+1})^s$. By $(*)$, the probability that τ appears is

$$p_s = \sum_{i=0}^{\lg n} G_i \cdot a_{i+1}^4.$$

We split off the first term, and for the remaining terms we use $(**)$.

$$p_s \leq 1/n^4 + \sum_{i=1}^{\lg n} G_i \cdot 16a_i^4$$

To bound the latter sum, note that $G_i \cdot a_i^4$ can be interpreted as the probability of another event: the first stopper is chosen in round i, and at the beginning of the i-th round, all of the triggers have been chosen. Thus $\sum_{i\geq 1} G_i \cdot a_i^4$ is the probability that the round in which all of the triggers are chosen is smaller than the first round where any stopper is chosen. This is the probability that among $s + 4$ independent

identically distributed random variables (namely the rounds of the 4 triggers and the s stoppers), the first 4 are (strictly) smaller than the others. By symmetry, we get

$$\sum_i G_i \cdot a_i^4 \leq \frac{1}{\binom{s+4}{4}}.$$

(This is the same argument that applies to the "usual" (fully) randomized insertion order.) This gives us

$$p_s \leq 1/n^4 + \frac{16}{\binom{s+4}{4}} = O(1/s^4).$$

□

5.6 Counting Tetrahedra

Now we need to bound k_s, the number of tetrahedra with s stoppers. Let K_s be the number of tetrahedra with *at most* s stoppers. Clarkson and Shor gave an upper bound on $(\leq k)$-sets that implies that K_s is at most $O(n^2 s^2)$ in the worst case, and $O(ns^3)$ in the "realistic" case. Their proof holds as $n/s \to \infty$. The bound was proved for all $1 < s \leq n$ in excruciating generality by Mulmuley [56]. In this section we give proofs using Mulmuley's basic idea (see also [25, Section 3]) but with much simpler arithmetic; this is possible because we deal only with the special cases of K_s that we need.

Consider the following thought experiment. From the set P of n points, we select each point with probability $1/s$ to form a random sample R. Let $r = |R|$ be the random variable for the size of R. Let T_R be the random variable for the number of tetrahedra in the Delaunay triangulation of R.

Lemma 17 *For every point set P, we have*

$$K_s = O(s^4 E[T_R]).$$

In the "realistic" case, we assume that $E[T_R] = O(r)$ so that by the linearity of expectation $E[T_R] = O(E[r]) = O(n/s)$, and $E[K_s] = O(ns^3)$.

In the quadratic worst case, we have $T_R = O(|R|^2)$, and hence $E[T_R] = E[O(r^2)] = O(E[r^2])$.

$$\begin{aligned} E[r^2] &= \mathrm{Var}[r] + E^2[r] \\ &= n\left(\frac{1}{s}\right)\left(1 - \frac{1}{s}\right) + (n/s)^2 = O((n/s)^2) \end{aligned}$$

So, $K_s \leq O(s^4(n/s)^2) = O(n^2 s^2)$.

For completeness, we indicate the easy proof of the lemma. We assume without loss of generality that $s \geq 2$. Let $\tilde{p_j}$ denote the probability that a tetrahedron with j stoppers appears in the Delaunay triangulation of R. For $j \leq s$,

$$\tilde{p_j} = \left(\frac{1}{s}\right)^4 \left(1 - \frac{1}{s}\right)^j \geq \left(\frac{1}{s}\right)^4 \left(1 - \frac{1}{s}\right)^s = \Theta\left(\frac{1}{s^4}\right)$$

and therefore $\tilde{p_j} = \Theta(1/s^4)$. We can express $E[T_R]$ in another way:

$$E[T_R] = \sum_{j=0}^{n} \tilde{p_j} k_j \geq \Theta\left(\frac{1}{s^4}\right) \sum_{j=0}^{s} k_j = \Theta\left(\frac{1}{s^4}\right) K_s$$

From this, the lemma follows.

□

5.7 Running time

Theorem 18 *With incremental construction using a BRIO, the expected total number of tetrahedra that are created during the construction of the Delaunay triangulation of n points in three dimensions is $O(n^2)$ in the worst case and $O(n)$ in the*

realistic case. The expected total size of the conflict graph (and hence the expected running time) is $O(n^2)$ in the worst case and $O(n \log n)$ in the realistic case.

Proof: Recall that our expression for the expected total number of tetrahedra is

$$E[C(n)] = \sum_{s=0}^{n} k_s p_s$$

and that $p_s = O(1/s^4)$, and note that $p_0 = 1$. We choose a constant c such that $p_s \leq c/s^4$, for $s \geq 1$. So

$$\begin{aligned} E[C(n)] &\leq K_0 + \sum_{s=1}^{n} (K_s - K_{s-1}) \frac{c}{s^4} \\ &= (1-c)K_0 + c \cdot \sum_{s=1}^{n-1} K_s \left(\frac{1}{s^4} - \frac{1}{(s+1)^4} \right) + \frac{K_n}{n^4} \\ &= O(K_0) + \sum_{s=0}^{n-1} O\left(\frac{K_s}{s^5} \right) + O\left(\frac{K_n}{n^4} \right) \end{aligned}$$

In the "realistic" case, $K_s = O(ns^3)$ so $E[C(n)] = O(n)$. In the worst case $K_s = O(n^2 s^2)$ and we find that $E[C(n)] = O(n^2)$.

We can use a similar argument to bound the total conflict size and hence the point location time.

$$E[X(n)] = \sum_{s=1}^{n} k_s \frac{c}{s^4} \cdot s = O(K_0) + \sum_{s=1}^{n-1} O\left(\frac{K_s}{s^4} \right) + O\left(\frac{K_n}{n^3} \right)$$

This gives $E[X(n)] = O(n \log n)$ in the "realistic" case and $E[X(n)] = O(n^2)$ in the worst case.

□

5.8 Experiments

We used three Delaunay triangulation programs, Clarkson's `hull`, Delaunay hierarchy function of CGAL, and Shewchuk's `pyramid`, to test the effect of our biased

randomized insertion order. These programs differ in their point location schemes and memory management strategies. Nonetheless, with all the programs, using the BRIO produced a near-linear performance profile, allowing us to handle much larger inputs than we could with a completely randomized insertion order.

While we of course had hoped that the effect of increased locality in $\mathbb{R}^3$ on the performance would be beneficial, it is not easy to predict. A fundamental problem with trying to optimize memory access by increasing the locality of insertions in $\mathbb{R}^3$ is that a point might well share Delaunay edges with other points that are quite far away. In particular, for sets of input points which lie on or close to surfaces (important to applications such as surface reconstruction and mesh generation) every point usually has at least one edge to some vertex on a different "opposite" part of the surface. Our experiments use this kind of input data.

Moreover, since the Delaunay triangulation is represented by a pointer structure, there is no guarantee that adjacent tetrahedra in the triangulation are stored together in virtual memory; this is implementation dependent. All three of the programs do their own memory management for the tetrahedra, to avoid making too many calls to the memory allocation procedure of the operating system. Records for tetrahedra are stored in a list in virtual memory. In the basic randomized incremental construction implemented by `hull`, tetrahedra are never deleted, but in `pyramid` and `CGAL` they are. Records are freed as tetrahedra are destroyed and reused as new tetrahedra are created, so that a tetrahedron might end up being stored with others created much earlier, further reducing locality.[1]

The idea of using a BRIO is to avoid having to explicitly manipulate disk access and letting the (hopefully very efficient) virtual memory system do the work. In all of our experiments we used a large (4 GB) virtual memory, which on our systems required some reconfiguration. This is important for duplicating our results;

[1] We thank Jonathan Shewchuk for pointing out this issue.

the programs fail if they run out of virtual memory.

Data sets

We used two data sets in the experiments, both of them are on or near a 2D surface in $\mathbb{R}^3$.

To make the B1 data set (525,296 points), we extracted a molecular electron density iso-surface using marching cubes and then made it larger by applying one level of butterfly subdivision. This gives a nicely distributed point set which lies on a smooth surface.

The happy buddha data set (2,643,633 points) is taken from the Stanford 3D scanning repository[2]. We use the raw scanner data as an example of typical input to a surface reconstruction computation. The data is noisy and unevenly distributed near the surface of the object.

We concentrated on experimenting with different insertion orders and different programs rather than many data sets; it would be interesting to try similar experiments with other data, especially data from different kinds of distributions.

Hull

To test exactly the situation considered in our analysis, we used a BRIO with the standard randomized incremental construction as implemented in `hull`. This program uses a theoretically optimal point location data structure, the history DAG, so that the expected point location time is proportional to the total conflict size. Also, since no tetrahedra are ever deleted, the layout of tetrahedra in virtual memory should correspond well to the order in which they are created. The history DAG takes a lot of memory, however, so `hull` thrashes relatively early.

[2]Stanford 3D scanning repository,
`http://www-graphics.stanford.edu/data/3Dscanrep`

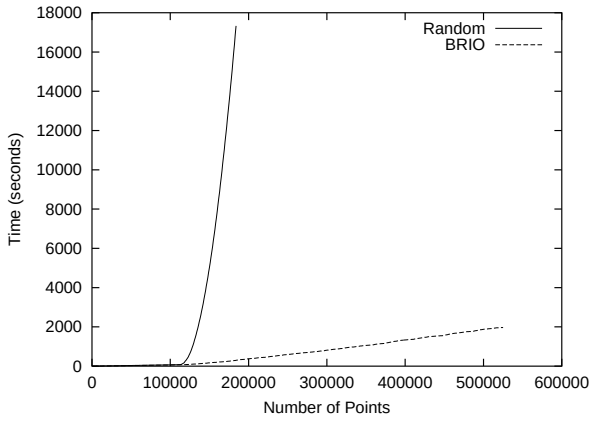

Figure 5.1: The running time of `hull` on B1 data using a completely randomized insertion order and a BRIO (512 MB RAM).

We ran `hull` on a Linux machine with an Intel Pentium III (864 MHz) and 512 MB RAM, using the smaller B1 data.

In Figure 5.1, the running time for `hull` using the completely random insertion order and the BRIO are shown; the random insertion order led to thrashing before the triangulation could be completed. Though the slope for the biased randomized insertion order becomes steeper around 120K points—just where the serious thrashing began for the random order—the BRIO maintains a roughly constant slope and shows a near-linear running time.

CGAL hierarchy

The Delaunay hierarchy function in the `CGAL` library also implements an optimal algorithm, due to Devillers [29], using a data structure for point location which requires much less memory than the history DAG. Deviller's analysis depends on the randomized insertion order to bound the expected point location time, but does not explicitly relate the running time to the total conflict size.

To exaggerate the memory behavior of the program, we ran it on a very small PC: a Sun UltraSPARC 360 MHz CPU with just 128M of physical memory. With

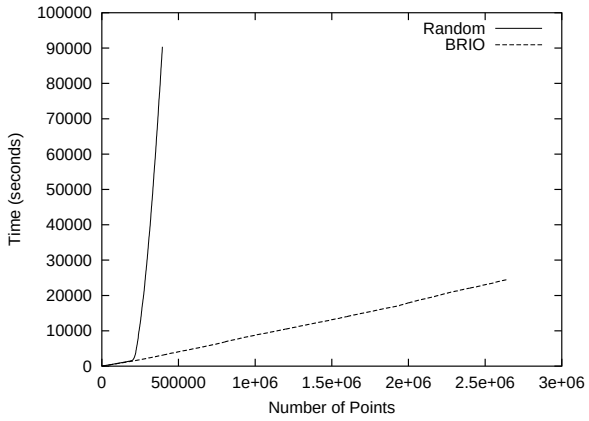

Figure 5.2: The running time of `CGAL` using a completely randomized insertion order and a BRIO (128 MB RAM).

so little memory, the `CGAL` Delaunay hierarchy program begins to thrash at about 250K points when using a completely randomized insertion order. Using a BRIO (Figure 5.2) we can in contrast compute the Delaunay triangulation of the happy buddha data set containing more than 2.5 million points.

Pyramid

Shewchuk's `pyramid` is designed to be very memory efficient. It uses a theoretically non-optimal point location scheme but needs no additional storage beyond the Delaunay triangulation itself. Our analysis shows that when using a BRIO the total number of tetrahedra created is asymptotically optimal, but other than that the relationship to the theory is tenuous in the case of this program. The fact, however, that we can compute huge 3D Delaunay triangulations, very quickly, using very little physical memory, is exciting.

We again use the small PC and the happy buddha data. We were able to complete the Delaunay triangulation using the BRIO, which was not possible with the completely randomized insertion order. But we found that as the size of the data structure grew, `pyramid`'s point location strategy slowed down significantly

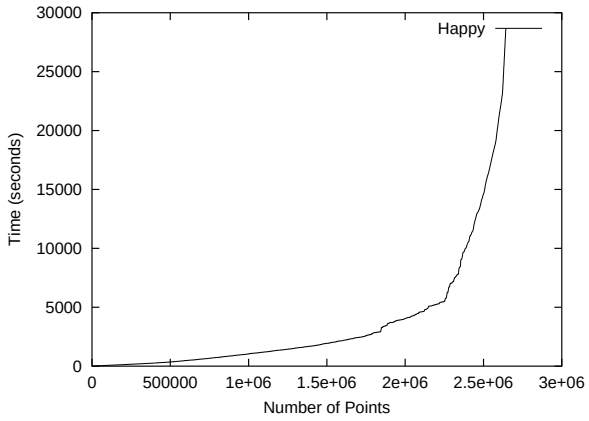

Figure 5.3: The running time of `pyramid` on the happy buddha data using BRIO with `pyramid`'s original point location scheme (128 MB RAM).

(Figure 5.3).

The point location strategy used in `pyramid` is known as jump-and-walk. At the insertion of the ith point p, it selects $O(i^{1/4})$ already-inserted points at random, and finds the closest of these to p. It then selects either this point, *or the last point inserted*, whichever is closer to p, and begins "walking" in the Delaunay triangulation on a straight line from there to find the place at which to insert p. With the BRIO, the last point inserted was almost always the closest to p, and the expensive search for a closer point was generally wasted.

Eliminating the $O(i^{1/4})$ search and always starting from the last point inserted gave us an essentially linear running time (Figure 5.4). To see how far we could push the results, we duplicated and translated the buddha data, making inputs that were the union of two and of four buddhas. We found that we could complete the Delaunay triangulation of four buddhas, over 10 million points (Figure 5.5). This represents an increase by a factor of 20 in the size of the Delaunay triangulations computable on this machine with this program.

Using this small machine is useful for studying memory performance, but it makes Delaunay triangulation look slower than it really is. On the machine we used

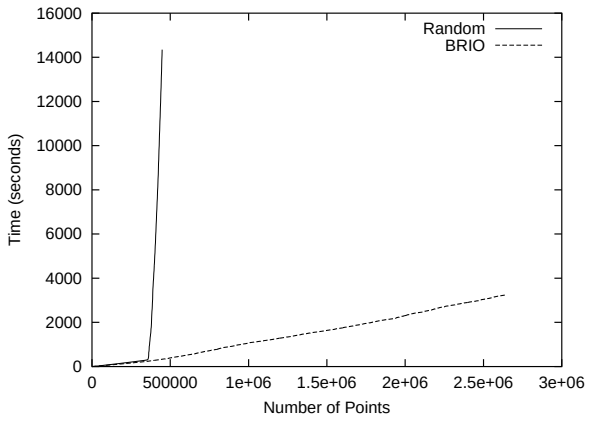

Figure 5.4: The running time of `pyramid` with the simplified point location strategy, on the happy buddha data set using a BRIO and a completely randomized insertion order for comparison (128 MB RAM).

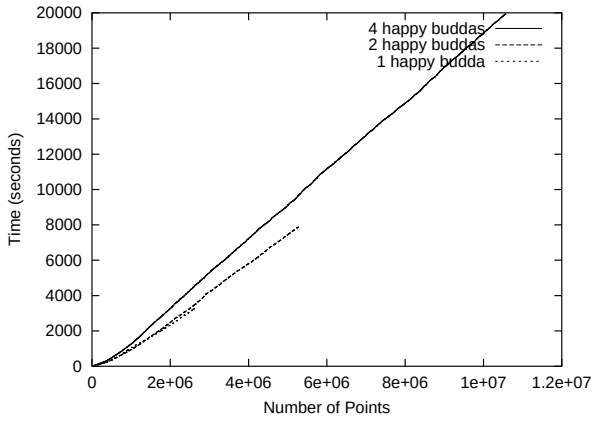

Figure 5.5: The running time of `pyramid` on 1,2 and 4 happy buddha data sets (128 MB RAM).

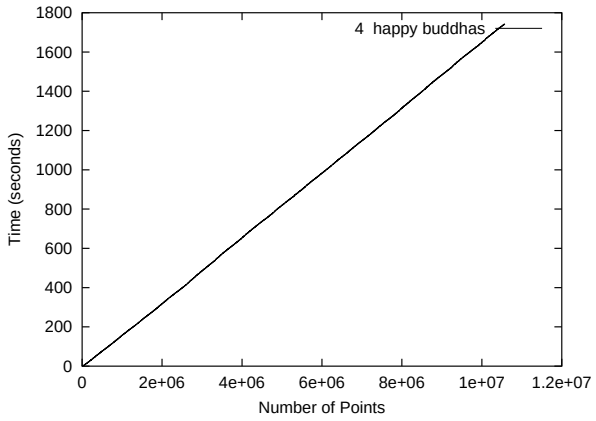

Figure 5.6: The running time of `pyramid` on a more typical workstation (864 MHz, 512 MB RAM). We computed the Delaunay triangulation of 10 million points (four translates of the happy buddha scan data) in about half an hour, using a BRIO with the simplified point location scheme.

for the experiments with `hull`, we can compute the Delaunay triangulation of 10 million points in about half an hour (Figure 5.6).

Of course, with this point location strategy there are no theoretical bounds on the expected running time besides the pessimistic worst case of $O(i)$, for a total running time of $O(n^2)$. However, for reasonable point sets, and for good insertion orders within each round, it seems reasonable to expect good performance. For uniformly distributed point sets, a simple algorithm with linear expected running time has been proposed by Dwyer [37]. It is conceivable that our approach might lead to an alternate algorithm for uniformly distributed point sets which degrades gracefully as the uniform distribution assumption is violated.

Pure locality

The locality of the BRIO certainly seems to improve the memory performance. To see if the randomness in the BRIO provides any practical benefit, we compared the three programs using a BRIO and an order which visits each octree cell in turn and

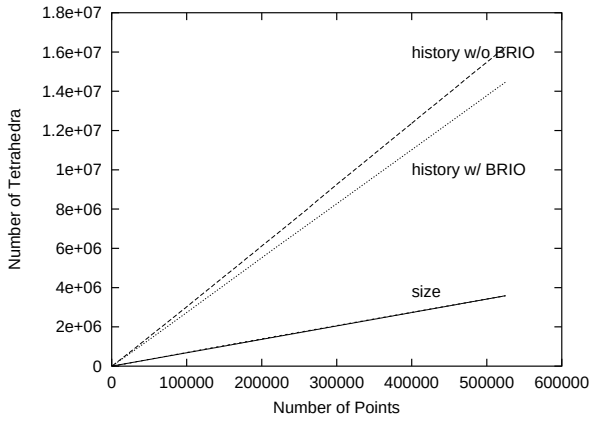

Figure 5.7: Total number of tetrahedra in the history DAG, for `hull` using the B1 data set. The bottom curve is the number of tetrahedra in the actual Delaunay triangulation.

inserts all of its points in random order. We found that both orders worked well, but which is better varied.

Figures 5.7, 5.8 show an example using `hull`. The BRIO creates a slightly smaller history DAG. Before physical memory is exceeded, the BRIO makes the program run faster, but once it begins paging the better locality of the octree order dominates.

Running the larger data set with `CGAL`, we found that the BRIO gave a slightly better running time (Figure 5.9); here again possibly the point location data structure is better with the BRIO.

Using pyramid, on the other hand, we found that the purely local order was better (Figure 5.10).

It is hard to draw conclusions from these examples, other than to note that factors like the specific memory hierarchy, layout in virtual memory, number of tetrahedra created and destroyed, balance in the point location data structure, etc., interact in complicated ways.

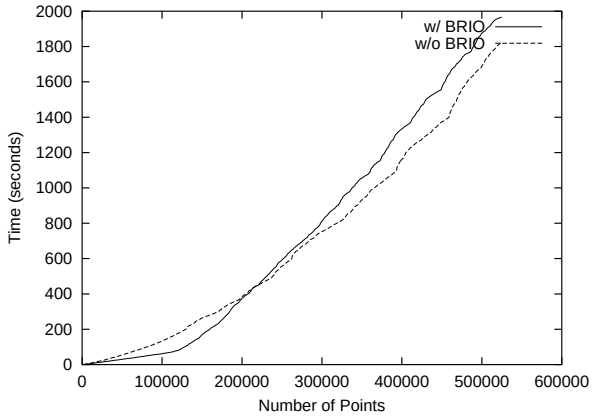

Figure 5.8: The octree insertion order with `hull` runs slightly faster on the B1 data, although the BRIO creates fewer total tetrahedra (512 MB RAM).

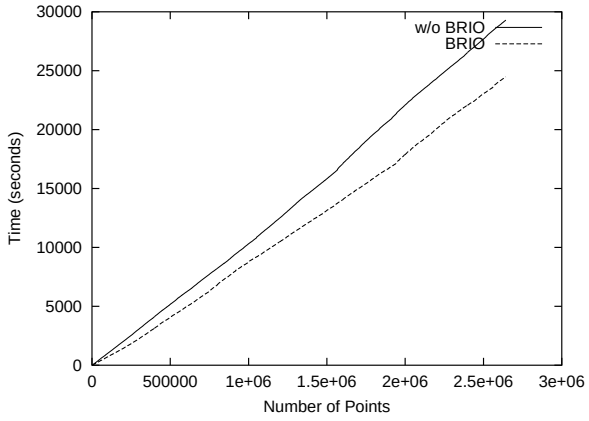

Figure 5.9: Using a BRIO gives a slightly better running time for the happy budda data using `CGAL` (128 MB RAM).

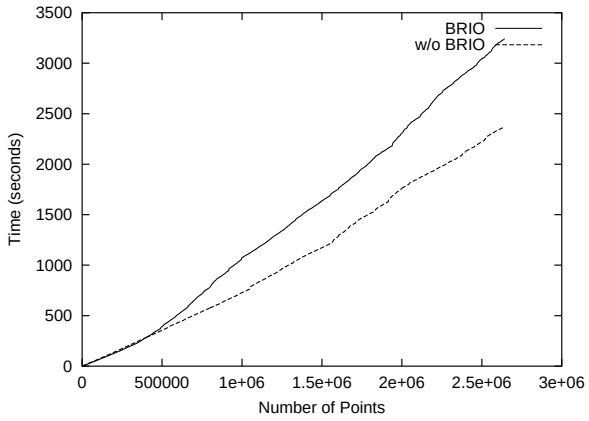

Figure 5.10: The purely local octree order runs faster than the BRIO for the happy budda data with `pyramid`, and the simplified point location scheme (128 MB RAM).

Computing a BRIO

Using a BRIO to improve the performance of Delaunay triangulation programs would not be sensible if computing the BRIO itself was time consuming compared with the time required to shuffle points for the randomized incremental construction. Fortunately, BRIOs can be computed very efficiently[3].

We tried both *kd*-trees and octrees for decomposing the input point sets into blocks for creating BRIOs. The *kd*-tree construction is appealing since it has logarithmic depth, and we can guarantee that every block has roughly the same number of points. However, neither of these properties are actually required, and constructing a *kd*-tree requires sorting while constructing an octree does not. Interestingly, the standard UNIX sort we used thrashed on our smallest (128 MB) machine. Since the performance of the two kinds of BRIOs were essentially the same, we prefer the octree.

For all the BRIOs, we used an upper limit of 2000 points per block. Our experience has been that block size should be a few thousand, but the exact number

[3]We thank Yong Kil (UC Davis) for his contribution to this section, including his implementation of the octree BRIO computation and his ideas for optimizing it.

does not seem to make much difference.

After computing the octree, the points are randomly shuffled within each block. In round i, we visit each block in turn. Within each block, we visit each uninserted point and insert it with the appropriate probability for the round. The overhead for visiting each point in each round is small compared to the time required to compute the octree.

The entire computation is extremely fast. We can compute the BRIO on 10 million points in about 2 minutes, including the computation of the octree (on a 1.7 GHz Pentium IV with 512 MB RAM).

Chapter 6

Discussion and future work

In this thesis, we have presented surface reconstruction algorithms and new methods to speed up the 3D Delaunay triangulation. We discuss here some possible directions for future research.

Among the challenges of the Delaunay triangulation based surface reconstruction algorithms are the sharp corners and noise. Though we address the problems in the power crust algorithm with heuristics, it would be nice to have some kinds of theoretical guarantees for these difficult cases. Though there are some work done for correctly reconstructing sharp corners in 2D curve reconstruction [34], but it seems much more challenging in 3D.

Another interesting topic is efficient computation of the approximate medial axis. The power shape is shown to be a good approximation of the medial axis, but it requires the power diagram computation. It would be nice to have a theoretically correct medial axis approximation without the expensive power diagram computation.

The analysis of the BRIO given in Chapter 5 also applies to other similar randomized incremental constructions such as the optimal construction of the trapezoidation of a set of non-intersecting segments in the plane [25, 54, 55], and

the similar construction for intersecting segments. The analysis must be modified for situations in which the objects have different numbers of triggers. As long as the number of triggers is bounded by some constant, we can simply analyze the objects separately for each possible number of triggers.

We would like to apply BRIOs to other elated randomized incremental algorithms which use *tracing*, such as Seidel's practical $O(n \lg^* n)$ algorithm for trapezoidation of a simple polygon [63]. BRIOs might also work with the LP-type (also known as GLP, "generalized linear programs") randomized incremental algorithms, which optimize an objective function over a set of input constraints. We would also like to explore other ways to implement BRIOs so that we can give some kinds of theoretical analysis about their effects in thrashing.

Appendix A

Proofs for the co-cone algorithm

In this section, we present the theoretical analysis of the co-cone algorithm[1].

For completeness, we review the manifold extraction step of the crust algorithm [2]. First, we delete all triangles incident to *sharp edges*. An edge is called *sharp* if the angle between any two consecutive triangles around the edge is more than $3\pi/2$. An edge with a single incident triangle is also sharp. Next, we extract the outer boundary N of the set of triangles by a depth-first walk along the outer boundary of each of its connected components.

Here we prove the following main theorem of the co-cone algorithm.

Theorem 19 *Let S be an r-sample for a smooth surface F, with $r \leq 0.08$. The co-cone algorithm computes a piecewise-linear 2-manifold N homeomorphic to F, such that any point on N is at most $\frac{1.3rLFS(x)}{1-r}$ from some point $x \in F$.*

First we prove that the following three conditions for the set T of candidate triangles.

I. Restricted Delaunay condition. The set T includes all restricted Delaunay

[1]Most of the result here is reprinted from [6] with permission from the publisher.

triangles.

II. SMALL TRIANGLE CONDITION. The circumcircle of each triangle $t \in T$ is small, that is, its radius is $O(r)LFS(s)$, where s is a vertex of t.

III. FLAT TRIANGLE CONDITION. The normal to each $t \in T$ makes a small, $O(r)$, angle with the surface normal at the vertex p, where p is the vertex with the largest interior angle in t.

Since the restricted Delaunay triangulation is homeomorphic to the original surface F given a dense enough sampling (Theorem 4), the first condition ensures that the set of candidate triangles contains a manifold homeomorphic to the original surface F. Then, we use the second and the third conditions to show that any manifold extracted from T is homeomorphic to F.

A.1 Restricted Delaunay condition

Here we show that the set of candidate triangles includes all restricted Delaunay triangles.

We begin with a technical observation, which says that the line segment connecting two points close together on F must be nearly parallel to the surface.

Observation 20 *A line segment connecting two points $x, x' \in F$, such that the distance $|x, x'| \leq crLFS(x)$, with $c \leq \sqrt{2}$, makes an acute angle with the surface normal n_x at x of at least $\pi/2 - sin\frac{cr}{2}$.*

This follows from the fact that x' must lie outside the two tangent balls of radius $LFS(x)$ at x.

Let $\vec{y}$ denote any ray from p to a point $y \in V_p$.

Lemma 21 *Let y be any point in the restricted Voronoi cell $V_{p,F}$. The acute angle between n_p and $\vec{y}$ is larger than $\pi/2 - r$, for $r < 0.1$.*

Proof: The distance $|yp| \leq rLFS(y)$, since $y \in V_{p,F}$ and S is an r-sample. By the Lipschitz condition $LFS(y) \leq LFS(p) + |py|$ giving $LFS(y) \leq \frac{LFS(p)}{1-r}$, and hence $|py| \leq rLFS(y) \leq \frac{r}{1-r}LFS(p)$. We can therefore apply Observation 20.
□

We can now prove that T satisfies Condition I.

Theorem 22 *All restricted Delaunay triangles are in T, for $r \leq 0.1$ and $\theta \leq \pi/8$.*

Proof: Let e be the dual edge of a restricted Delaunay triangle. Consider the point $y = e \cap F$. We have $y \in V_{p,S}$ for each of the three points $p \in S$ determining e. For each such p, the acute angle between n_p and $\vec{y}$ is larger than $\pi/2 - r$ by Lemma 21. Therefore $\angle \vec{y}\vec{v} \in [\pi/2 - r - \alpha, \pi/2 + r + \alpha]$, where α is the acute angle between the vector to the pole $\vec{v}$ and n_p. Plugging in the upper bound on α from Lemma 6 we find that $\alpha + r < \pi/8$, so $\angle \vec{y}\vec{v} \in I$.
□

A.2 Small triangle condition

We now show that T meets Condition II. Looking at the contrapositive of Lemma 5 and plugging in a value of $\rho = \frac{1.3r}{1-r}$ we obtain the following.

Corollary 23 *Let x be any point in V_p so that the acute angle between $\vec{x}$ and n_p is at least $\pi/2 - \theta - 2\sin^{-1}\frac{r}{1-r}$. Then $|px| < \frac{1.3r}{1-r}LFS(p)$, for $\theta = \pi/8$ and $r \leq 0.08$.*

Proof: If the acute angle between $\vec{x}$ and n_p is at least $\alpha = \arcsin \frac{r}{\rho(1-r)} + \arcsin \frac{r}{1-r}$, then $|px| < \rho LFS(p)$ according to Lemma 5. With $\rho = \frac{1.3r}{1-r}$ we have

$$\alpha \quad = \quad \arcsin \frac{1}{1.3} + \arcsin \frac{r}{1-r}$$

which is less than $\pi/2 - \theta - 2 \arcsin \frac{r}{1-r}$ for $\theta = \pi/8$ and $r \leq 0.08$.

□

Lemma 24 *Let p be a vertex of any triangle $t \in T$. The radius of the smallest Delaunay ball of t is at most $\frac{1.3r}{1-r} LFS(p)$ for $r \leq 0.08$ and $\theta = \pi/8$.*

Proof: Let e be the dual edge of t and p any vertex of t. By our choice of e, there is a point $x \in e$ so that $\vec{x}$ makes an angle in the range $I = [\pi/2 - \theta, \pi/2 + \theta]$ with $\vec{v}$. Taking into account the angle between $\vec{v}$ and n_p we conclude that this ray makes an acute angle more than $\pi/2 - \theta - 2 \arcsin \frac{r}{1-r}$ with n_p. From Corollary 23, $|px| < \frac{1.3r}{1-r} LFS(p)$.

□

Theorem 25 *Let r denote the radius of the circumcircle of any triangle $t \in T$. Then, for each vertex p of t, $r \leq \frac{1.3r}{1-r} LFS(p)$ for $r \leq 0.08$ and $\theta = \pi/8$.*

Proof: The radius of the smallest Delaunay ball, bounded in Lemma 24, is an upper bound on the radius of the circumcircle of t, which is centered at the intersection of the line containing e with the plane containing t.

□

A.3 Flat triangle condition

Here we show that T meets Condition III.

Theorem 26 *The normal to any triangle $t \in T$ makes an acute angle of no more than $\alpha + \arcsin(\frac{2}{\sqrt{3}} \sin 2\alpha)$ with n_p where p is the vertex at the largest interior angle of t, and $\alpha \leq arcsin\frac{1.3r}{1-r}$, $r \leq 0.08$.*

Proof: Consider the medial balls M_1 and M_2 touching F at p with the centers on the medial axis. Let D be the ball with the circumcircle of t as a diameter; refer to Figure A.1. The radius r of D is equal to the radius of the circumcircle of t. Denote the circles of intersection of D with M_1 and M_2 as C_1 and C_2 respectively. The normal to F at p passes through m, the center of M_1. This normal makes an angle less than α with the normals to the planes of C_1 and C_2, where

$$\begin{aligned} \alpha &\leq \arcsin r/|pm| \\ &\leq \arcsin \frac{1.3r}{1-r} \end{aligned}$$

since $|pm| \geq LFS(p)$ by the definition of f and $r \leq \frac{1.3r}{1-r} LFS(p)$ by Theorem 25. This angle bound also applies to the plane of C_2, which implies that the planes of C_1 and C_2 make a wedge, say W, with an acute dihedral angle no more than 2α.

The vertices other two vertices q, s of t cannot lie inside M_1 or M_2. This implies that t lies completely in the wedge W. Consider a cone at p inside the wedge W formed by the three planes; π_t, the plane of t, π_1, the plane of C_1 and π_2, the plane of C_2. A unit sphere centered around p intersects the cone in a spherical triangle uvw, where u, v and w are the points of intersections of the lines $\pi_1 \cap \pi_2$, $\pi_t \cap \pi_1$ and $\pi_t \cap \pi_2$ respectively with the unit sphere. See the picture on right in Figure A.1. Without the loss of generality, assume that the angle $\angle uvw \leq \angle uwv$. We have the following facts. The arc length of wv, denoted $|wv|$, is at least $\pi/3$ since p subtends the largest angle in t and t lies completely in the wedge W. The spherical angle $\angle vuw$ is less than or equal to 2α.

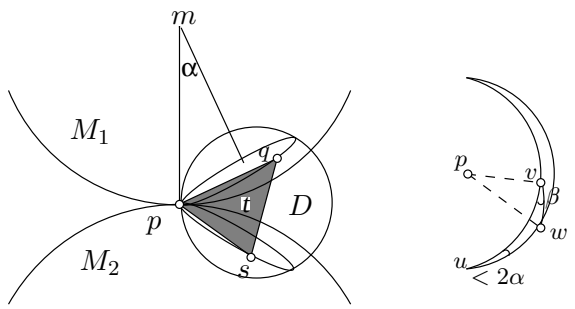

Figure A.1: Normal to a small triangle and the normal to S at the vertex with the largest face angle.

We are interested in the spherical angle $\beta = \angle uvw$ which is also the acute dihedral angle between the planes of t and C_1. By standard *sine laws* in spherical geometry, we have $\sin\beta = \sin|uw|\frac{\sin\angle vuw}{\sin|wv|} \leq \sin|uw|\frac{\sin 2\alpha}{\sin|wv|}$. If $\pi/3 \leq |wv| \leq 2\pi/3$, we have $\beta \leq \arcsin\frac{2}{\sqrt{3}}\sin 2\alpha$. For the range $2\pi/3 < |wv| < \pi$, we use the fact that $|uw| + |wv| \leq \pi$ since $\angle vuw \leq 2\alpha < \pi/2$ for sufficiently small r. So, in this case $\frac{\sin|uw|}{\sin|wv|} < 1$. Thus, $\beta \leq \arcsin\frac{2}{\sqrt{3}}\sin 2\alpha$.

The normals to t and F at p make an acute angle at most $\alpha + \beta$ proving the theorem.

□

The upper bound on the angle between the normal to t and n_p provided by this theorem is 23°; and the angle is $O(r)$.

A.4 Geometric consequences

Triangle interiors

Conditions II and III relate properties of each triangle $t \in T$ to the value of $LFS(\cdot)$ and the surface normal direction, respectively, at its vertices. But since the triangles are small, we can use the Lipschitz properties to show that similar properties hold at any point q in the interior of t. To define these properties, we map q to the nearest surface point. Let $\mu : \mathbb{R}^3 \rightarrow F$ map each point $q \in \mathbb{R}^3$ to the closest point of F. The restriction of μ to T is a well-defined function $\mu : T \rightarrow F$, since if some point q had more than one closest point on the surface, q would be a point of the medial axis; but by Lemma 27 every point $q \in T$ is within $\frac{1.3r}{1-r} LFS(p)$ of a triangle vertex $p \in F$.

Lemma 27 *Let q be any point on a triangle $t \in T$. The distance between q and the point $x = \mu(t)$ is at most $0.165 LFS(x)$, for $r \leq 0.08$.*

Proof: The circumcircle of t is small; the distance from q to the vertex p of t with largest angle is at most $2\delta\ LFS(p)$, with $\delta = \frac{1.3r}{1-r} \leq .12$, by Theorem 25. Since there is a sample, namely, a vertex of t within $\delta LFS(p)$ from q, we have $|qx| \leq \delta LFS(p)$. We are interested in expressing this as a function of $LFS(x)$, so we need an upper bound on $|px|$.

The triangle vertex p has to lie outside the tangent ball at x, while, since x is the nearest surface point to q, q must lie on the segment between x and the center of this tangent ball. For any fixed $|pq|$, these facts imply that $|px|$ is maximized when the angle pqx is a right angle. Thus, $|px| \leq \sqrt{5}\delta LFS(p) \leq 0.27 LFS(p)$ for $r \leq 0.08$. This implies that $LFS(p) \leq 1.37 LFS(x)$ by Lipschitz property of $LFS(\cdot)$, giving $|qx| \leq 0.165 LFS(x)$.

□

With a little more work, we can also show that the triangle normal agrees with the surface normal at the surface point closest to t.

Lemma 28 *Let q be a point on triangle $t \in T$, and let n_x be the surface normal at $x = \mu(q)$. The acute angle between n_x and the normal to t is at most 42° for $r \leq 0.08$. Also, the acute angle between n_x and the surface normal n_p at the vertex p of t with largest angle is at most 19°.*

Proof: Applying Lemma 3, and taking $\rho = 0.165$, shows that the angle between n_x and n_p is less than 19°. The angle between the triangle normal of t and n_p is less than 23° for $r \leq 0.08$ (Theorem 26). Thus, the triangle normal and n_x make an angle of at most 42°.

□

Sharp edges

The manifold extraction step selects a piecewise-linear manifold from T. It begins by recursively removing any triangle in T adjacent to a *sharp* edge; recall that a sharp edge is one for which the angle between two adjacent triangles, in the circular order around the edge, is greater than $3\pi/2$. Let T' be the remaining set of triangles. The following lemma shows that none of the restricted Delaunay triangles are removed, so that T' is guaranteed to contain a piecewise-linear manifold homeomorphic to F.

Lemma 29 *No restricted Delaunay triangle has a sharp edge, for $r \leq 0.08$.*

Proof: Let t and t' be adjacent triangles in the restricted Delaunay triangulation, let e be their shared edge, and let $p \in e$ be any of their shared vertices. Since t and t' belong to the restricted Delaunay triangulation, they have circumspheres B and B', respectively, centered at points v, v' of F.

The boundaries of the circumspheres B and B' intersect in a circle C contained in a plane H, with $e \subset H$. H separates t and t', since the third vertex of each triangle must lie on the boundary of its circumsphere, and $B \subseteq B'$ on one side of H, while on the other $B' \subseteq B$. The line through v, v' is perpendicular to H, and the distance $|vv'| \leq 2r/(1-r)LFS(v)$ (using the sampling condition). So segment v, v' forms an angle of at least $\pi/2 - sin\frac{r}{1-r}$ with n_v (Observation 20). This normal differs, in turn, from n_p by at most $\frac{r}{1-4r}$ (Lemma 3), so H is nearly parallel to n_p, at an angle of at most 7°. The normals of both t and t' differ from the surface normal at p by at most 23° (Theorem 26).

Thus we have t on one side of H, t' on the other, and the smaller angle between H and either triangle is at least 60°. Hence the smaller angle between t and t' is at least 120°, and e is not sharp.

□

A.5 Homeomorphism

In this section, we will show a homeomorphism between F and any piecewise-linear surface made up of candidate triangles from T with two additional properties. The piecewise-linear manifold N selected by the manifold extraction step of our algorithm does in fact have these properties, thus completing the proof of Theorem 19.

Additional properties

A pair of triangles $t_1, t_2 \in N$ are *adjacent* if they share at least one common vertex v. Since the normals to all triangles sharing v differ from the surface normal at v by at most 42° (Lemma 28), and that normal in turn differs from the vector to the pole at v by less than 5° (Lemma 6), we can orient the triangles sharing v, arbitrarily

but consistently calling the normal facing the pole the *inside* normal and the normal facing away from the pole the *outside* normal. Let α be the angle between the two inside normals of t_1, t_2. We define the angle at which the two triangles meet at v to be $\pi - \alpha$.

PROPERTY I: Every two adjacent triangles in N meet at their common vertex at an angle of greater than $\pi/2$.

Requiring this property excludes manifolds which contain sharp folds and, for instance, flat tunnels. Since the triangles of T are all nearly perpendicular to the surface normals at their vertices, and the manifold extraction step eliminates triangles adjacent to sharp edges, N has this property.

PROPERTY II: Every sample in S is a vertex of N.

Lemma 29 ensures that T' contains the restricted Delaunay triangulation, which contains a triangle adjacent to every sample in S. Lemma 31, below, ensures that at least one triangle must be selected for each sample by the manifold extraction step. This implies that N has the second property as well.

Homeomorphism proof

We define the homeomorphism explicitly, using the function $\mu : N \rightarrow F$, as defined above. The function μ defines a homeomorphism between N and F if it is continuous, one-to-one and onto. Our approach will be first to show that μ is well-behaved on the samples themselves, and then show that this behavior continues in the interior of each triangle of N.

Lemma 30 *The restriction of μ to N is a continuous function $\mu : N \rightarrow F$.*

Proof: By Theorem 25 every point $q \in N$ is within $\frac{1.3r}{1-r} LFS(p)$ of a triangle vertex $p \in F$. Function μ is continuous except at the medial axis of F, so that since N is continuous and avoids the medial axis, μ is continuous on N.

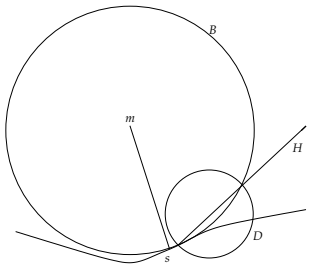

Figure A.2: Proof of Lemma 31.

□

Lemma 31 *Let p be a sample and let m be the center of a medial ball M tangent to the surface at p. No candidate triangle intersects the interior of the segment pm.*

Proof: In order to intersect segment pm, a candidate triangle t would have to intersect M, and so would the smallest Delaunay ball D of t. Let H be the plane of the circle where the boundaries of M and D intersect. We show that H separates the interior of pm and t.

On one side of H, M is contained in D, and on the other, D is contained in M. Since the vertices of t lie on F and hence not in the interior of M, t has to lie in the open halfspace, call it H^+, in which D is outside M. Since D is Delaunay, p cannot lie in the interior of D; but since p lies on the boundary of M, it therefore cannot lie in H^+. We claim that $m \notin H^+$ either. (see Figure A.2.) Since $m \in M$, if it lay in H^+ then m would have to be contained in D. Since m is a point of the medial axis, this would mean that the radius of D would be at least $1/2\ LFS(p')$ for any vertex p' of t, contradicting, by Lemma 24, the assertion that t is a candidate triangle. Therefore p, m and hence the segment pm cannot lie in H^+, and H separates t and pm.

□

Since any point q such that $\mu(q) = p$ lies on such an open segment pm, we have the following.

Corollary 32 *The function μ is one-to-one from N to every sample p.*

In what follows, we will show that μ is indeed one-to-one on all of N.

Our proof proceeds in three short steps. We show that μ induces a homeomorphism on each triangle, then on each pair of adjacent triangles, and finally on N as a whole.

Lemma 33 *Let U be a region contained within one triangle $t \in N$ or in adjacent triangles of N. The function μ defines a homeomorphism between U and $\mu(U) \subset F$.*

Proof: We know that μ is well-defined and continuous on U, so it only remains to show that it is one-to-one. First, we prove that if U is in one triangle t, μ is one-to-one. For a point $q \in t$, the vector $\vec{n_q}$ from $\mu(q)$ to q is perpendicular to the surface at $\mu(q)$; since F is smooth the direction of $\vec{n_q}$ is unique and well defined. If there was some $y \in t$ with $\mu(y) = \mu(q)$, then q, $\mu(q)$ and y would all be collinear and t itself would have to contain the line segment between q and y, contradicting Lemma 28, which says that the normal of t is nearly parallel to $\vec{n_q}$.

Now, we consider the case in which U is contained in more than one triangle. Let q and y be two points in U such that $\mu(q) = \mu(y) = x$, and let v be a common vertex of the triangles that contain U. Since μ is one-to-one in one triangle, q and y must lie in the two distinct triangles t_q and t_y. Let n_x be the surface normal at x. The line l through x with direction n_x pierces the patch U at least twice; if y and q are not adjacent intersections along l, redefine q so that this is true ($\mu(q) = x$ for any intersection q of l with U). Now consider the orientation of the patch U according to the direction to the pole at v. Either l passes from inside to outside and back to inside when crossing y and q, or from outside to inside and back to outside.

The acute angles between the triangle normals of t_q, t_y and n_x are at most 42° (Lemma 28), that is, the triangles are stabbed nearly perpendicularly by n_x. But since the orientation of U is opposite at the two intersections, the angle between the two *oriented* triangle normals is at least $\pi - 84^\circ$, meaning that t_q and t_y must meet at v at an acute angle of at most 84°. This would contradict PROPERTY I, which is that t_q and t_y meet at v at an obtuse angle. Hence there are no two points in y, q with $\mu(q) = \mu(y)$.

□

We finish the theorem using a theorem from topology.

Theorem 34 *The mapping μ defines a homeomorphism from the triangulation N to the surface F for $r \leq 0.08$.*

Proof: Let $F' \subset F$ be $\mu(N)$. We first show that (N, μ) is a *covering space* of F'. Informally, (N, μ) is a covering space for F' if function μ maps N smoothly onto F', with no folds or other singularities; see Massey [53], Chapter 5. Showing that (N, μ) is a covering space is weaker than showing that μ defines a homeomorphism, since, for instance, it does not preclude several connected components of N mapping onto the same component of F', or more interesting behavior, such as a torus wrapping twice around another torus to form a *double covering*.

Formally, the (N, μ) is a covering space of F' if, for every $x \in S'$, there is a path-connected *elementary neighborhood* V_x around x such that each path-connected component of $\mu^{-1}(V_x)$ is mapped homeomorphically onto V_x by μ.

To construct such an elementary neighborhood, note that the set of points $|\mu^{-1}(x)|$ corresponding to a point $x \in S'$ is non-zero and finite, since μ is one-to-one on each triangle of N and there are only a finite number of triangles. For each point $q \in \mu^{-1}(x)$, we choose an open neighborhood U_q of around q, homeomorphic to a disk and small enough so that U_q is contained only in triangles that contain q.

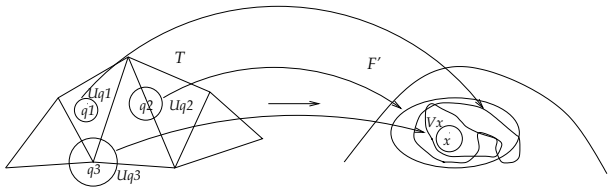

Figure A.3: Proof of Theorem 34.

We claim that μ maps each U_q homeomorphically onto $\mu(U_q)$. This is because it is continuous, it is onto $\mu(U_q)$ by definition, and, since any two points x and y in U_q are in adjacent triangles, it is one-to-one by Lemma 33.

Let $U'(x) = \cap_{q \in \mu^{-1}(x)} \mu(U_q)$, the intersection of the maps of each of the U_q. $U'(x)$ is the intersection of a finite number of open neighborhoods, each containing x, so we can find an open disk V_x around x. V_x is path connected, and each component of $\mu^{-1}(V_x)$ is a subset of some U_q and hence is mapped homeomorphically onto V_x by μ. Thus (N, μ) is a covering space for F'.

We now show that μ defines a homeomorphism between N and F'. Since N is onto F' by definition, we need only show that μ is one-to-one. Consider one connected component G of F'. A theorem of algebraic topology (see eg. Massey [53], Chapter 5 Lemma 3.4) says that when (N, μ) is a covering space of F', the sets $\mu^{-1}(x)$ for all $x \in G$ have the same cardinality. We now use Corollary 32, that μ is one-to-one at every sample. Since each connected component of F contains some samples, it must be the case that μ is everywhere one-to-one, and N and F' are homeomorphic.

Finally, we show that $F' = F$. Since N is closed and compact, F' must be as well. So F' cannot include part of a connected component of F, and hence F' must consist of a subset of the connected components of F. Since every connected component of F contains a sample p (actually many samples), and $\mu(p) = p$, all components of F belong to F', $F' = F$, and N and F are homeomorphic.

□

Appendix B

Proofs for the power crust algorithm

Here we give the correctness proofs for the power crust algorithm that the power crust is a good approximation of the original surface F, we need some theorems about the union of polar balls first. [1]

B.1 Unions of polar balls

Let $\mathcal{P}$ be the set of poles. The surface F divides the set of poles into the set $\mathcal{P}_I$ of *inside poles* and the set $\mathcal{P}_O$ of *outside poles*. The corresponding sets of polar balls are $\mathcal{B}_I$ and $\mathcal{B}_O$.

Let $\mathcal{U}_I = \bigcup \mathcal{B}_I$ be the union of Voronoi balls centered at inside poles, and $mathcal U_O = \bigcup \mathcal{B}_O$ be the union of Voronoi balls centered at outside poles. Let $U_I = \delta \mathcal{U}_I$ and $U_O = \delta \mathcal{U}_O$ be the boundaries of these unions. We will now show that, under the sampling assumption, first, U_I and U_O are both close to F, second, their surface normals agree with those of F, and third, each of them is homeomorphic to

[1] Most of the content here is reprinted from [8] with permission from Elsevier.

F.

Shallow intersections

The main idea in the proof is that inside and outside balls cannot intersect each other deeply. We say this in three different ways in the lemmata below. We measure the depth of the intersection by the angle α at which the balls intersect, as in Figure B.1.

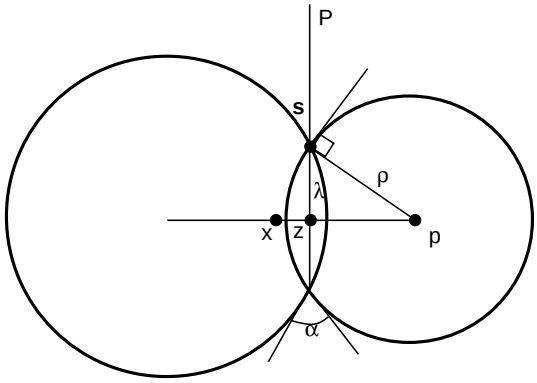

Figure B.1: An inside and outside ball can intersect only at a small angle α.

Figure B.2 illustrates the following observation.

Observation 35 *Let B_I and B_O be two intersecting balls, and let x be a point on the segment connecting them. Any ball centered at x and containing point outside of bothB_I and B_O also completely contains $B_I \cap B_O$.*

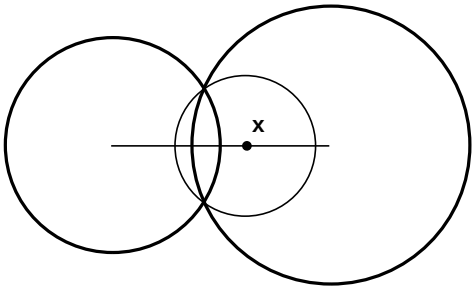

Figure B.2:

The first version of the lemma deals with the special case in which the two polar balls are the inner and outer polar balls of the same sample s, for which we can get the best bound.

Lemma 36 *The two polar balls of a sample s intersect at an angle of $O(r)LFS(s)/\rho$, where ρ is the radius of the smaller polar ball.*

Proof: Without loss of generality let the inner polar ball $B_{p,\rho}$ be smaller than the outer polar ball B_O. The line segment between p and the center of B_O intersects the surface in at least one point x. Since $B_{p,\rho}$ and B_O cannot contain samples, s is the nearest sample to x (Observation 35) and $d(x,s) \leq rLFS(x)$.

Let z be the center of the circle C in which the boundaries of B_I and B_O intersect, and let λ be the radius of C, as in Figure B.1. We have $\lambda \leq d(x,s)$, and so, using Observation 2, $\lambda \leq O(r)LFS(s)$. The angle between P and the tangent plane to $B_{p,\rho}$ at s is the same as $\angle zps = \arcsin(O(r)LFS(s)/\rho)$. Since $LFS(s) \leq \rho$, for small enough r this is $O(r)LFS(s)/\rho$. The angle between P and the tangent plane to B_O is no greater, so $\alpha = O(r)LFS(s)/\rho$.

□

Now we show that in the general case, any pair consisting of an inner and an outer polar ball must intersect shallowly. We need the following corollary of Theorem 5.

Corollary 37 *Every polar ball contains a point of the medial axis, when $r < 1/3$.*

For convenience, let $r' = r/(1-r) = O(r)$.

Lemma 38 *Let B_I be an inside polar ball and B_O be an outside polar ball. B_I and B_O intersect at an angle of at most $2\arcsin 3r = O(r)$.*

Proof: Consider the line segment connecting c_I and c_O, the centers of B_I and B_O. Since c_I and c_O lie on opposite sides of F, this segment crosses F in at least one point x.

Let $B_{c,\rho}$ be the smaller of the two balls B_I and B_O. If $x \in B_{c,\rho}$, we have $LFS(x) \leq 2\rho$. Since, the polar ball , $B_{c,\rho}$ contains a point of the medial axis (Corollary 37).

Otherwise x is in the larger of the two balls, but not in the smaller, as in Figure B.1. Let c be the center of the smaller ball, and again define z and λ as in Figure B.1. By Corollary 37, we have $LFS(x) \leq d(x,c) + \rho = d(x,z) + d(z,c) + \rho$. But the distance from x to the nearest sample is at least

$$\sqrt{\lambda^2 + d^2(x,z)} = \sqrt{\rho^2 - d^2(z,c) + d^2(x,z)}$$

So the r-sampling requirement means that

$$\sqrt{\rho^2 - d^2(z,c) + d^2(x,z)} \leq r[\rho + d(x,z) + d(z,c)]$$

Since $d(z,c) \leq \rho$, we can simplify to

$$d(x,z) \leq 2r'\rho$$

which, for $r \leq 1/3$, means that x is very close to $B_{c,\rho}$, and $LFS(x) \leq 3\rho$.

Since the distance from x to the nearest sample is at least λ and at most $3r\rho$, we know that $\lambda \leq 3r\rho$. The angle between the plane P containing C and a tangent plane on $B_{c,\rho}$ at C is thus at most $\arcsin 3r$, the angle between P and the tangent plane of the larger ball is smaller, and the two balls meet at an angle of at most $2 \arcsin 3r$.

□

The third lemma shows that a similar fact holds when one of the balls is a medial, rather than a polar, ball.

Lemma 39 *Let B_p be an inside (outside) polar ball and let B_m be an outside (inside) medial ball. The angle at which B_p and B_m intersect is at most $2 \arcsin 2r = O(r)$.*

Proof: Again we consider the line segment connecting p and m, the centers of B_p and B_m, which crosses W in at least one point x, which is in B_p but not B_m (since the interior of any medial ball is empty of points of the surface).

We have $LFS(x) \leq 2\rho_p$, since B_p contains a point of the medial axis. When $2\rho_p \leq \rho_m$, we use this bound to show that the balls intersect at an angle of at most $2 \arcsin 2r$, as in the proof of Lemma 38.

Otherwise, since m itself is a point of the medial axis, we have $LFS(x) \leq d(x,m) = d(x,z) + d(z,m)$. Again, the distance from x to the nearest sample is at least

$$a = \sqrt{\lambda^2 + d^2(x,z)} = \sqrt{\rho_m^2 - d^2(z,m) + d^2(x,z)}$$

So the r-sampling requirement means that

$$\sqrt{\rho_m^2 - d^2(z,m) + d^2(x,z)} \leq r[d(x,z) + d(z,m)]$$

Since $d(z,c) \leq \rho_m$, we can simplify to

$$(1-r)d(x,z) \leq r\rho_m$$

which, for $r \leq 1/2$, means that $LFS(x) \leq 2\rho_m$. We use this bound to show that the angle between the two balls is most $2 \arcsin 2r$, again as in Lemma 38.

□

Proximity

We now turn to the proof that the union boundaries U_I and U_O approximate F. We can immediately infer from Lemma 38 that the surface F cannot penetrate too far into the interior of either union, as a function of the radii of the balls forming the unions. We extend this to a stronger bound in terms of LFS, which could be much smaller than the radius of either medial ball at a surface point x.

Lemma 40 *Let u be a point in the Voronoi cell of s but not in the interior of either polar balls at s. The distance from u to s is $O(r)LFS(s)$.*

Proof: We assume without loss of generality that $LFS(s) = 1$. Let p_1 be the pole farther from s. If $\angle usp_1 \leq \pi/2$, we let $p = p_1$, otherwise we consider $p = p_2$, the pole nearer to s. We let $B_{p,\rho}$ be the polar ball centered at p. In either case $d(u, s) \leq \rho$, because of the way in which the poles were chosen. Let θ be the angle between vectors $\vec{su}$ and $\vec{sp}$. Since u is outside the polar ball, $d(s, u) \geq 2\rho \cos\theta$.

Since $d(s, u) \leq \rho$, we have $\theta \geq \frac{\pi}{3} > 3 \arcsin r'$. Let $\vec{n}$ represent the normal at s. We find $\angle \vec{n}\ \vec{sp} < 2 \arcsin r'$ by Lemma 5. So $\angle \vec{n}\ \vec{su} > \pi/3 - 2 \arcsin r' > \arcsin r'$. It follows that, for any point u in the Voronoi cell of s,

$$d(u, s) \leq \frac{r'}{(sin(\theta - 3 \arcsin r'))}$$

Since $\theta \geq \frac{\pi}{3}$, the angle, $(\theta - 3 \arcsin r') \geq \frac{\pi}{6}$. Thus $d(u, s) \leq 2r'$. Since we assumed $LFS(s) = 1$, the lemma follows.

□

Corollary 41 *Any point u which does not lie in the interior of either $\mathcal{U}_I$ or $\mathcal{U}_O$ is within distance $O(r)LFS(s)$ of its closest sample s.*

It remains to bound the distance from any point on the boundary of one union and in the interior of the other to the surface.

Lemma 42 *For a point u contained in both $\mathcal{U}_I$ and $\mathcal{U}_O$, the distance to the closest sample s is $O(r)LFS(s)$.*

Proof: Point u is contained in an inner ball B_I and an outer ball B_O. The line joining the centers of B_O and B_I intersects the surface at some point x. Let s_x be the closest sample to x and let s be the closest sample to u; see Figure B.1. A ball

centered at x, and with radius $d(x, s_x)$, must also contain u (Observation 35). This and the r-sampling condition give a bound on $d(x, u)$.

$$d(x, u) \leq d(x, s_x) = O(r)LFS(x)$$

Hence

$$d(u, s) \leq d(u, s_x) \leq d(u, x) + d(x, s_x) = O(r)LFS(x)$$

By Observation 2, $d(u, s) = O(r)LFS(s)$.

□

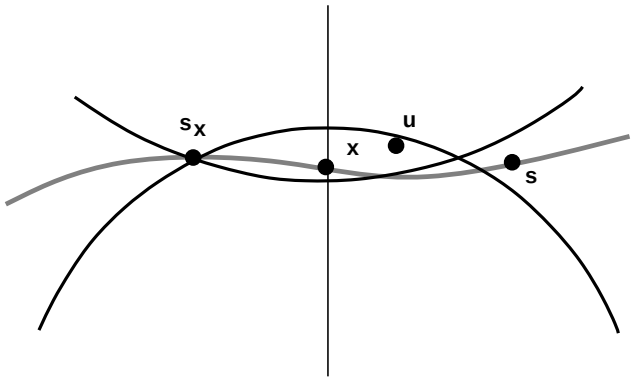

Figure B.3: The point u is closer to x than s_x, which is outside both the polar balls.

We use the two lemmata above to show that the two union boundaries $U_I = \delta\mathcal{U}_I$ and $U_O = \delta\mathcal{U}_O$ have to be close to the surface.

Theorem 43 *The distance from a point $u \in U_I$ or $u \in U_O$ to its closest point on the surface $x \in F$ is $O(r)LFS(x)$.*

Proof: Let s be the closest sample to u. Assume without loss of generality that u is on the boundary U_I. The either $u \in \mathcal{U}_I$ and $u \in \mathcal{U}_O$, so that $d(u, s) = O(r)LFS(s)$ by Lemma 42, or u is in the interior of neither $\mathcal{U}_I$ or $\mathcal{U}_O$, so that $d(u, s) = O(r)LFS(s)$ by Corollary 41. The point x is at least as close to u as s is, and hence $d(x, u) =$

$O(r)LFS(s)$ and $d(x,s) = O(r)LFS(s)$. The result follows from Observation 2.

□

Lemmata 40 and 42 imply that most of $\mathcal{Q}$ lies in either the union of inner balls or the union of outer balls, while only points in a small part - the *tubular region* within $O(r)$ of the distance to the medial axis - might lie in both, or neither, as in Figure B.4.

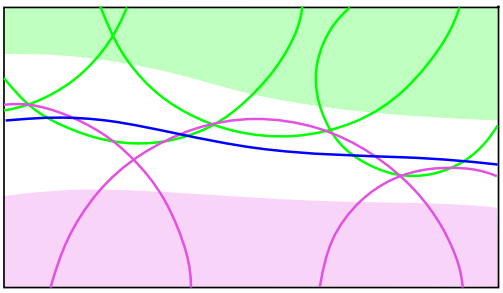

Figure B.4: The boundaries of the unions of balls $\mathcal{U}_I$ and $\mathcal{U}_O$ must lie close to the surface F. Specifically, the boundaries are contained in the *tubular region*, defined as the set u of points such that the distance from u to the closest point $x \in F$ is at most $O(r)$ times the distance from x to the medial axis.

Normals

Now we show that the normals on the boundaries U_I and U_O are also close to the normals of nearby points of F, approaching the correct normal as $O(\sqrt{r})$ as $r \to 0$.

Observation 44 *Let $B = B_{c,\rho}$ be a polar ball, at distance at most k from a point $x \in F$. Then $\rho \geq \frac{LFS(x)-k}{2}$.*

This follows because B is a polar ball, so it contains a point of the medial axis, by Corollary 37, while the nearest point of the medial axis to x is at distance $LFS(x)$.

Lemma 45 *Let u be a point such that the distance to the nearest surface point $x \in F$ is at most $O(r)LFS(x)$. Let $B_{c,\rho}$ be a polar ball containing u. Then the angle, in radians, between the surface normal at x and the vector $\vec{uc}$ is $O(\sqrt{r})$.*

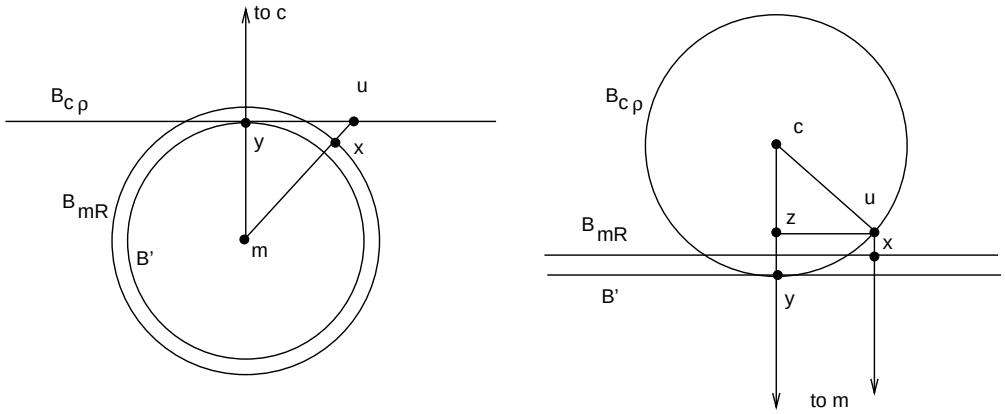

Figure B.5: Since B cannot intersect B_M very deeply, and $d(u, x)$ has to be small, the indicated angle cannot be too large.

Proof: Let $B_{m,R}$ be the medial ball at x on the opposite side of the surface from c. Since x is the nearest surface point to u, the vector x, u is normal to the surface at x. So we can write the angle we are interested in as $\alpha = \angle ucm + \angle umc$. We begin by bounding $\angle umc$. Without loss of generality, assume $LFS(x) = 1$.

Since $B_{c,\rho}$ and $B_{m,R}$ cannot intersect at x at an angle greater than $2 \arcsin 2r$ (Lemma 39), the thickness of the lune in which they intersect is at most a factor of $O(r^2)$ times the smaller of the two radii. Let B' be the ball centered at m and touching this lune, as in Figure B.5.

Angle $\beta = \angle umc$ will depend on the ratio of the two radii R and ρ. The worst case is on the left in Figure B.5. Since β decreases as u moves towards the center c, we assume u is on the boundary of $B_{c,\rho}$. For any fixed ρ, increasing R makes β smaller, so we assume $R = 1$, its minimum value since $B_{m,R}$ is a medial ball at x. For any fixed R, increasing ρ makes β larger, so we assume that B is infinitely large. Since $B_{m,R}$ is the smaller ball, the radius of B' is $R(1-O(r^2))$. Let y be the point at which segment c, m intersects B'. We get $\angle umc = \sqrt{d^2(m, y) - d^2(m, u)} = O(\sqrt{r})$.

We use a similar argument to bound $\gamma = \angle ucm$. Again we can assume that u is on the boundary of $B_{c,\rho}$. For any fixed ρ, increasing R increases γ, and for any

fixed R, increasing ρ decreases γ, so in contrast to the previous situation, we let ρ take on its minimum value of $(1 - d(u,x))/2 = \theta(1)$ (Observation 44), and let R become infinitely large. The worst case is shown on the right in Figure B.5, giving $\angle ucm = \sqrt{d^2(c,u) - d^2(c,z)} = O(\sqrt{r})$.

□

Theorem 46 *Let u be a point on U_I or U_O, and let $x \in F$ be the closest surface point to u. The difference between the normal n_u (where it is defined) to the union boundary at u and the surface normal n_x at x is $O(\sqrt{r})$.*

Proof: Point u is contained in the tubular region, and the distance $d(u,x) = O(r)LFS(x)$ (Theorem 43). If n_u is defined, then u is contained in the surface of exactly one ball and n_u is the vector pointing towards the ball center, so we can apply Lemma 45.

□

Homeomorphism

We use these geometric theorems to show that the surface of either U_I or U_O is homeomorphic to the actual surface F. We'll do this using a natural map from U to F.

Definition: Let $\mu : \mathbb{R}^3 \to F$ map each point $q \in \mathbb{R}^3$ to the closest point of F.

Lemma 47 *Let U be either U_I or U_O. The restriction of μ to U defines a homeomorphism from U to F.*

Proof: We consider U_I; the argument for U_O is identical. Since U_I and F are both compact, it suffices to show that μ defines a continuous, one-to-one and onto function. The discontinuities of μ are the points of the medial axis. From Theorem 43, every point of U_I is within distance $O(r)LFS(x)$ from some point $x \in F$, whereas every point of the medial axis is at least $LFS(x)$ from the nearest point $x \in F$. Thus μ is continuous on U_I.

Now we show that μ is one-to-one. For any $u' \in U_I$, let $x = \mu(u')$ and let $n(x)$ be the normal to F at x. Orient the line $l(x)$ through x with direction $n(x)$ according to the orientation of F at x. Any point on U_I such that $\mu(u) = x$ must lie on $l(x)$; let u be the outer-most such point.

Let $B_{c,\rho}$ be the ball in U_I with u on its boundary. Let α be the angle between $\vec{uc}$ and the surface normal $n(x)$. By Theorem 46, $\alpha = O(\sqrt{r})$. Meanwhile $\rho = \Omega(LFS(x))$, by Observation 44.

Point u' is at most $O(rLFS(x))$ from u, while $l(x)$ lies in the interior of $B_{c,\rho}$ for distance at least $2\rho\cos\alpha = O(LFS(x))$. Since u' must be on $l(x)$ but outside of $B_{c,\rho}$, and u is the outermost such point, it must be the case that $u = u'$.

Finally, we need to establish that $\mu(U)$ is onto F. Since μ maps U, a closed and bounded surface, continuously onto F, $\mu(U)$ must consist of some subset of the closed, bounded connected components of F. But since every connected component of F contains samples of F, and $\mu(s) = s$ for $s \in S$, $\mu(U)$ must consist of all the connected components of F.

□

B.2 The power crust

It seems natural that since $\mathcal{U}_I$ and $\mathcal{U}_O$ are accurate representations of $\mathcal{F}$ and its complement, that the power crust that they induce is also an accurate representation of F. We establish this formally in this section.

Proximity

The fact that the power crust is close to F is actually immediate from our results so far. Since any point on a face separating an inside from an outside cell is contained in either both of their Voronoi balls or in no Voronoi ball at all, Theorem 43 implies the following.

Corollary 48 *Any point u on a face of the power crust lies within $O(r)LFS(x)$ of some point $x \in F$.*

Notice that although a point u on the power crust might be nearest to inner (outer) polar ball B, in Euclidean distance, it might belong to the power cell of some other inner (outer) ball B' which is nowhere near B. Our proof that the power crust is homeomorphic to the original surface hinges on showing that B and B' cannot, in fact, be too far apart.

Observation 49 *Let p be a point in the tubular neighborhood, and let s be the sample nearest p. Then $d(p, s) = O(r)LFS(s)$.*

Let $x \in F$ be the closest point on the surface to p. The Observation above follows since the distance $d(p, s)$ is at most distance $d(p, x) + d(x, s')$, where s' is the sample nearest x, using Observation 2.

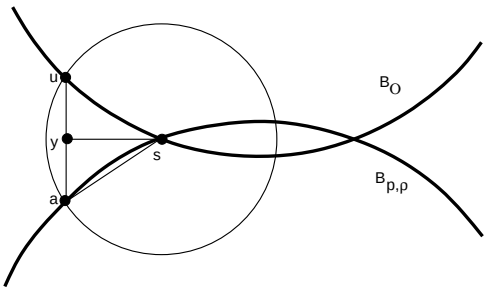

Figure B.6:

Lemma 50 *Let u be a point in the tubular neighborhood outside of any polar ball, let $x \in F$ be the nearest surface point to u, and let s be the closest sample to x. Let $B_{p,\rho}$ be the smaller of the two polar balls at s. Then $d(u, B_{p,\rho}) = O(r^2)LFS^2(s)/\rho$.*

Proof: Since u is in the tubular neighborhood, $d(u,x) = O(r)LFS(x)$, and $d(x,s) \leq rLFS(x)$. So by Observation 49, $d(u,s) = O(r)LFS(s)$, that is, u is contained in a ball of radius $O(r)LFS(s)$ centered at s, as in Figure B.6. The distance from u to $B_{p,\rho}$ will be maximized when a) the two polar balls intersect in as large an angle as possible (which is $O(r)LFS(s)/\rho$, by Lemma 36) and b) the radius of B_O is as small as possible (which is ρ).

From Figure B.6, we have $d(u, B_{p,\rho}) \leq d(u,a)$. The length of the chord sa is $O(r)LFS(s)$, so the angle between the chord and the tangent plane to $B_{p,\rho}$ at s is $\arcsin[O(r)LFS(s)/2\rho] = O(r)LFS(s)/\rho$. So the total angle $\angle ysa = O(r)LFS(s)/\rho$ as well.

This gives $d(u,a) = O(r)LFS(s)\sin[O(r)LFS(s)/\rho]$, and hence $d(u, B_{p,\rho}) = O(r^2)LFS^2(s)/\rho$, for small enough r.

Lemma 51 *Let u be a point in the tubular neighborhood, and let p be the inner (outer) pole at minimum power distance to u, with polar ball $B_{p,\rho}$. Let $x \in F$ be the nearest surface point to u and let s be the nearest sample to x. Let $B_{c,\mu}$ be the smaller of the two polar balls at s. If $u \notin B_{p,\rho}$, then $d(u, B_{p,\rho}) = O(r)LFS(s)$, for small enough r.*

Proof: If u is inside $B_{c,\mu}$, then it is inside $B_{p,\rho}$, and the lemma is trivial. Otherwise, we claim that the radius λ of the ball B_u centered at u and orthogonal to $B_{c,\mu}$ is at most $O(r)LFS(s)$, for small enough r. Since this ball must also intersect $B_{p,\rho}$, the Lemma follows.

To establish the claim, assume without loss of generality that $LFS(s) = 1$, so that $LFS(x) = 1$ (Observation 2). By Lemma 50, $d(u, B_{p,\rho} \leq k(r^2/\mu)$, for some

constant k. We have,

$$\lambda = \sqrt{\mu^2 + 2(kr^2/\mu)\mu + k^2r^4/\mu^2 - \mu^2}$$
$$= \sqrt{2kr^2 + k^2r^4/\mu^2} = O(r)$$

□

Homeomorphism

From Lemma 45 and Lemma 51 we get the following observation, which we need to establish the homeomorphism between the power crust and F.

Observation 52 *Let u be a point in the tubular neighborhood, and let p be the inner (outer) pole at minimum power distance to u, with polar ball B_p. Let $x \in F$ be the surface point closest to u with surface normal n_x. The vector from $u\vec{,}c$ forms an angle of at most $\alpha = \pi/6$ with n_x, for small enough r.*

The set of points in the tubular neighborhood closest to a point $x \in F$ forms a line segment g, perpendicular to the surface at x. Note that when we take a point u in the tubular neighborhood to its nearest point $x \in F$, it travels along the segment g corresponding to x.

Lemma 53 *The segment g normal to the surface at a point $x \in F$ and passing through the tubular neighborhood intersects the power crust exactly once.*

Proof: Consider the function $f_I(u)$ which returns the minimum power distance to any pole $p \in P_I$. The restriction of f_I to the segment g is a piecewise quadratic function. We claim that this function is monotonically decreasing as u goes from the outer end of g to the inner end, since, by Observation 52, the direction in which f_I is decreasing is always within $\pi/6$ of g, and any angle less than $\pi/2$ would suffice.

Similarly, the function f_O is monotonically increasing on g. So f_I and f_O are equal at exactly one point, at a face of the power diagram separating the cells of an inside and an outside pole.

□

Theorem 54 *There is a space homeomorphism taking the power crust to F.*

Proof: Let Y be the power crust. We define a deformation of all of the domain $\mathcal{Q}$ which takes Y into F, and hence the interior of Y into the interior of F and the exterior of Y into the exterior of F. Specifically, we define a continuous parameterized map $f_t : \mathcal{Q} \to \mathcal{Q}$, for $t \in [0, 1]$, such that at any time t, f_t is a continuous, one-to-one and onto map, and such that at time $t = 0$, $f_0(Y) = Y$, and at time $t = 1$, $f_1(Y) = F$.

The power crust is strictly contained in the tubular neighborhood around F (Lemma 48). Outside of the tubular neighborhood, we define f_t to be the identity, at every time t. By Lemma 53, the segment g normal to F at a point $x \in F$ and passing through the tubular neighborhood intersects the power crust exactly once, in a point $y \in Y$. By the definition of the tubular neighborhood, g intersects F only in x. Let g_i and g_o be the inner and outer endpoints of g. We define $f_t(y) = tx + (1-t)y$, and we let f_t linearly map the segments g_i, y to $g_i, f_t(y)$ and y, g_o to $f_t(y), g_o$.

□

B.3 Theoretical algorithm

Here we present the theoretical algorithm and its correctness proof. The labeling algorithm in Section 3.3 is a special case of the theoretical algorithm. We know

that the polar balls of an inner and an outer pole can only intersect shallowly. In addition, we use the following lemma to label poles correctly.

Lemma 55 *Two inside (resp. outside) polar balls inducing a face within the tubular neighborhood meet at an angle of at least $\pi/2$, for small enough r.*

Proof: Let p be a point any point on the face inside the tubular neighborhood, and let c_1, c_2 be the centers of the two inside (resp. outside) polar balls inducing the face. The angle between the surface normal $n(x)$ at the point $x \in F$ closest to p and either pc_1 or pc_2 is at most $\pi/6$, so $\angle c_1 p c_2$ is at most $\pi/3$.
□

This leads to the following algorithm to label each pole as either outside (O') or inside (I').

Input: An r-sample S from a closed, bounded smooth surface F.
Output: The power crust of S.

Step 1: Construct the Delaunay triangulation of S, find the Voronoi vertices, and select two poles for each sample. Let $\mathcal{B}_P$ be the set of polar balls.

Step 2: Construct the power diagram $Pow(\mathcal{B}_P)$.

Step 3: Select a sample on the convex hull of S.
Label its infinite outer pole with O' and the opposite inner pole I'.
Insert both poles in a queue.

Step 4: While the queue is non-empty:

Remove a labeled pole p from the queue, and examine
each unlabeled neighbor q of p in $Pow(\mathcal{B}_P)$.
If the Voronoi ball surrounding q intersects the Voronoi ball of p
at an angle of more than $\pi/4$:
Give q the same label as p and insert it in the queue.
For each sample s such that q is a pole of s,
if the pole q' opposite q at s is unlabeled:
Give q' the opposite label from
q and insert q' into the queue.

Step 5: Output the faces of $Pow(\mathcal{B}_P)$ separating the cells of one pole labeled I' and one pole labeled O' as the power crust.

To prove that this algorithm is correct, we need to show that the sets I and O, corresponding to the inside an outside of F, are identical to the sets I' and O'.

Lemma 56 *No pole in I receives label O' and no pole in O receives label I'.*

Proof: Let q be the first mislabeled pole, and let p be the pole from whose label that of q was determined. Either p and q should have opposite labels but they meet at an angle of more than $\pi/2$, or p and q should have the same label but they are opposite poles of the same sample s. The first case is impossible by Lemma 38, and the second is impossible because the two poles of any sample always should have opposite labels.

□

Lemma 57 *Every pole receives a label.*

Proof: We consider a pole $p \in I$, Every ball in I has at least one point on the power crust, since each sample s such that p is a pole of s appears on the power crust.

By Lemma 55 we know that every power crust edge is contained in two balls which intersect deeply (the meet at an angle of at most $\pi/2$). Therefore if any pole q in the same connected component of $\mathcal{U}_I$ receives label I', then p will eventually as well.

Each connected component of either I or O eventually gets at least one labeled pole. Assume not; consider some component that remains unlabled; we claim that there must be a sample on this component. If this is true, we are done, because a label will be propagated across this sample.

The claim must be true; otherwise, consider any point x on the boundary of that component. The line segment connecting x to its nearest sample s must cross the medial axis, so that the distance $d(x, s) \geq LFS(x)$, a contradiction.

□

Bibliography

[1] P. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete and Computational Geometry*, 6(5):407–422, 1991.

[2] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, **22**, (1999), 481–504.

[3] N. Amenta, M. Bern and D. Eppstein. The crust and the β-skeleton: combinatorial curve reconstruction. *Graphical Models and Image Processing*, **60** (1998), 125-135.

[4] N. Amenta, M. Bern and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. *SIGGRAPH 98*, (1998), 415-421.

[5] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm to homeomorphic surface reconstruction. *Proceedings of the 16th ACM Symposium on Computational Geometry*, pages 213–222, June, 2000.

[6] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm to homeomorphic surface reconstruction. *International Journal of Computational Geometry and Applications*, 12:(1-2), 125-141, 2002.

[7] N. Amenta, S. Choi, and R. Kolluri. The power crust. *Proceedings of the sixth*

ACM Symposium on Solid Modeling and Applications, pages 249–260, June, 2001.

[8] N. Amenta, S. Choi, and R. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19(2-3):129–153, 2001.

[9] L. Arge. External memory data structures, in J. Abello, P. M. Pardalos, and M. G. C. Resende, eds., *Handbook of Massive Data Sets*, Kluwer Academic Publishers, 2001

[10] D. Attali and J.-D. Boissonnat. A linear bound on the complexity of the Delaunay triangulation of points on polyhedral surfaces. *Proceeding of the ACM Symposium on Solid Modeling*, 2002.

[11] D. Attali and J.-O. Lachaud. Constructing iso-surfaces satisfying the delaunay constraint; application to the skeleton computation. *Proc. 10th International Conference on Image Analysis and Processing (ICIAP'99)*, pages 382–387. IEEE, 1999.

[12] F. Bernardini, J. Mittleman, H. Rusahmeir, C. Silva and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Vision and Computer Graphics*. Also IBM Tech. Report RC21463(96842).

[13] M.de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications* Springer-Verlag, Berlin, 1997.

[14] G. Blelloch, J. Hardwick, G. Miller, and D. Talmor. Design and Implementation of a Practical Parallel Delaunay Algorithm, *Algorithmica*, 24(3/4), 1999.

[15] J-D. Boissonnat. Geometric structures for three-dimensional shape reconstruction. *ACM Transactions on Graphics* 3 (1984) 266–286.

[16] J-D. Boissonnat and F. Cazals. Natural coordinates of points on a surface. *Proceedings of the 16th Annual ACM Symposium on Computational Geometry* pp. 223–232, (2000).

[17] The CGAL website. `www.cgal.org`

[18] T. M. Chan, J. Snoeyink, and C.-K. Yap. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional voronoi diagrams, *Discrete and Computational Geometry*, 18:433-454, 1997.

[19] H.-L. Cheng, T. K. Dey, H. Edelsbrunner, and J. Sullivan. Dynamic skin triangulation. *Proceeding of the 12th Annu. ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[20] L. P. Chew. Building voronoi diagrams for convex polygons in linear expected time. CS Tech Report TR90-147, Dartmouth College, 1986.

[21] S. Choi and N. Amenta. Delaunay triangulation programs on surface data, *Proceeding of the 13th ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[22] N. Chrisochoides, D. Nave, and P. Chew. Delaunay refinement for restricted polyhedral domains, *Proceedings of the 18th. ACM Symposium on Computational Geometry*, pp 135–144, June, 2002.

[23] P. Cignoni, C. Montani, R. Perego, and R. Scopigno. Parallel 3D Delaunay triangulation, *Computer Graphics Forum*, 12(3):129–142, 1993.

[24] K. L. Clarkson. New applications of random sampling in computational geometry, *Discrete and Computational Geometry* 2:195-222, 1987.

[25] K. L. Clarkson, and P.W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry* 4:387-421, 1989.

[26] K. L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions, *Computational Geometry: Theory and Applications*, 3(4):185-212, 1993.

[27] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *SIGGRAPH 96*, (1996), 303-312.

[28] O. Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *International Journal of Computational Geometry and Applications*, 2(1):621–635, 1992.

[29] O. Devillers. The Delaunay hierarchy. *Internatiional Journal of Foundation of Computer Science*, **13** (2002), 163–180.

[30] O. Devillers and P. Guigue. The shuffling buffer, *International Journal of Computational Geometry and its Applications*, 11(5): 555-572, 2001.

[31] T. Dey and J. Giesen. Detecting undersampling in surface reconstruction. *Proceeding of the 17th ACM Symposium on Computational Geometry*, (2001), 257–263.

[32] T. Dey and S. Goswani. Tight Cocone: A watertight surface reconstruction, *Manuscript*, 2002.

[33] T. Dey, J. Giesen and J. Hudson. Delaunay based shape reconstruction from large data. *IEEE Symposium in Parallel and Large Data Visualization and Graphics*, (2001), 19–27.

[34] T. Dey and R. Wenger. Reconstructing curves with sharp corners. *Computational Geometry: Theory and Applications*, 19, (2001), 89–99.

[35] H. Q. Dinh, G.Turk, and G. Slabaugh. Reconstructing surfaces by volumet-

ric regularization using radial basis functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10): 1358-1371, October, 2002.

[36] R. A. Dwyer. Average-case analysis of algorithms for convex hulls and Voronoi diagrams. Ph.D. thesis, Computer Science Dept., Carnegie Mellon Uni., Pittsburgh, PA, 1988. Report CMU-CS-88-132.

[37] R. A. Dwyer. Higher-dimensional voronoi diagrams in linear expected time. *Discrete and Computational Geometry*, 6(4):343–367, 1991.

[38] H. Edelsbrunner and W. Shi. An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem. *SIAM Journal of Computing*, 20(2):259–269, 1991.

[39] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, **13**, (1994), 43–72.

[40] H. Edelsbrunner and N. Shah. Triangulating topological spaces. *Proceedings of the 10th ACM Symposium on Computational Geometry*, (1994), 285–292.

[41] J. Erickson. Nice point sets can have nasty delaunay triangulations. *Proceedings of the 17th ACM Symposium on Computational Geometry*, 2001.

[42] J. Erickson. Dense point sets have sparse delaunay triangulations. *Proceedings of the 13th. ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[43] S. Fortune. Voronoi diagrams and Delaunay triangulations. *Computing in Euclidean Geometry.* D.-Z. Di amd F. Hwang (eds.), World Scientific Publ., 1992, 193–223.

[44] S. Funke and E. Ramos. Smooth-surface reconstruction in near-linear time. *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[45] M. Golin and H. Na. On the average complexity of 3d-voronoi diagrams of random points on convex polytopes. *Proceedings of the 12th Canadian Conference on Computational Geometry*, pages 189–196, 1999.

[46] M. J. Golin and H. S. Na. On the average complexity of 3D-Voronoi diagrams of random points on convex polytopes. Tech. report HKUST-TCSC-2001-08, Hong Kong University of Science and Technology, 2001.

[47] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S.Vitter. External-Memory Computational Geometry. *Proceedings of the 34th Foundations of Computer Science (FOCS)* (1993), pp. 714–723.

[48] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH 92*, (1992), 71–78.

[49] D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, 1995.

[50] F. Lazarus, M. Pocciola, G. Vegter, and A. Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. *Proceedings of the 17th ACM Symposium on Computational Geometry*, 2001.

[51] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade and D. Fulk. The Digital Michelangelo project: 3D scanning of large statues. *SIGGRAPH 2000*, pages 131–144.

[52] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):163–169, 1987.

[53] W. S. Massey. *Algebraic Topology: An Introduction*, Springer-Verlag, Graduate texts in Mathematics 56, 1967.

[54] K. Mulmuley. A Fast Planar Partition Algorithm, I, *Journal of Symbolic Computation*, 10:253-280, 1990

[55] K. Mulmuley. A Fast Planar Partition Algorithm, II. *Journal of the ACM*, 38(1):74-103, 1991

[56] K. Mulmuley. On levels in arrangements and Voronoi diagrams, *Discrete and Computational Geometry*, 6:307-338, 1991

[57] K. Mulmuley. Computational Geometry: An Introduction Through Randomized Algorithms. Prentice Hall, New York, 1993.

[58] K. Mulmuley and O. Schwarzkopf. Randomized Algorithms, Chapter 34 in J. E. Goodman and J. O'Rourke, eds., *Handbook of Discrete and Computational Geometry*, CRC Press, 1997.

[59] E. P. Mücke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations, *Proceeding of the 12th. Symposium on Computational Geometry*, 1996.

[60] G. Narasimhan, M. Zachariasen, and J. Zhu. Experiments with computing geometric minimum spanning trees. *Proceedings of ALENEX'00*, pages 183–196, 2000.

[61] G. Robins and J. S. Salowe. On the maximum degree of minimum spanning trees. *Proceedings of the 10th ACM Symposium on Computational Geometry*, pages 250–258, Stony Brook, New York, 1994.

[62] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete and Computational Geometry* 6:423-434, 1991.

[63] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons, *Computational Geometry: Theory and Applications*, 1:51-64, 1991.

[64] R. Seidel. Backwards analysis of randomized geometric algorithms. In J. Pach, ed., *New Trends in Discrete and Computational Geometry*, pp 37-68, Springer-Verlag, Berlin, 1993.

[65] J. Shewchuk. Delaunay refinement mesh generation, Ph.D. thesis, Technical Report CMU-CS-97-137, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 18 May 1997.

[66] C. Thomassen. Trees in triangulations. *Journal of Combinatorial Theory*, Series B:56–62, 1994.

[67] H.-K. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. *Proceedings of IEEE Workshop on Variational and Level Set Method in Computer Vision (VLSM 2001)*, July 2001.

[68] B. Zhu. Further computational geometry in secondary memory. In *5th Annual International Symposium on Algorithms and Computation (ISAAC)* vol. 834 of *Lecture Notes Computer Science* Springer-Verlag, 1994, pp. 514–522.

Vita

Sunghee Choi was born in Seoul, Korea on February 8, 1973, the daughter of Sun-Jung Choi and Hae-Sang Jung. After graduating from Eun-Kwang Girls' High School, Seoul, Korea, in 1991, she entered the Seoul National University where she received her Bachelor of Science degree in Computer Engineering in 1995.

In September 1995, she entered the graduate school of the University of Texas at Austin where she received her Master of Science degree in Computer Science in May 1997. She worked as an intern at the Schlumberger during the summer of 1999.

Ms. Choi's research interests are in computational geometry, computer graphics, and visualization. She has published 6 peer-reviewed articles in internationally renowned conferences and journals.

Permanent Address: 183 Hyundai Villat 101-101
Tokok-Dong, Kangnam-Gu, Seoul, Korea

This dissertation was typeset with LaTeX 2_ε[2] by the author.

[2] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay and James A. Bednar.