Surface Reconstruction By Layer Peeling

LIM CHI WAN
(B.Eng (Hons), NUS)
(M.Sc, NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2007

Acknowledgement

The past four years of my life have been very fruitful and I have learnt a lot during this period of time, in both research and life. I owe special thanks to many people for their guidance, help, support and encouragement. My supervisor, A/P Tan Tiow Seng, has been especially helpful and patient with me. From him, I learnt a lot not only with regards to research, but also in terms of his attitude towards work, his attention to little details and his commitment towards students. When I have failed so many time during the process of my thesis, it was his encouragement and belief that eventually allowed me to find the breakthrough in my research. With that, I am grateful towards him for his guidance, kindness and constant complain of my work schedule.

During this stay in the graphics lab, I have made many invaluable friendships. Lab mates such as Ouyang Xin and Shi Xinwei have helped and taught me a lot regarding research through many discussion sessions. From them, I slowly understood that to do a proper research, a lot of time and effort have to be invested in performing a thorough background research. Guodong and Zhang Xia has been very good friends with me, and many days in the lab would have been boring if not without them. Geoffery and Junxiang have been very good tea break pals. And to my snooker pals, Jianliang, YongShan and Jeremy, for teaching me so much about the game.

Last but not least, to my parents who have been quietly supporting me during these three months of writing the thesis. To Mickay, who barks at me constantly whenever I have no food to offer. And to my three dogs, which I have neglected so much, hope they can find happiness in their doggie heaven.

Contents

Summary

Reconstructing surfaces from unorganized sample point sets is a fundamental problem in both geometry processing and computer graphics. Given an input point cloud P in $\mathbb{R}^3$, this thesis proposes a novel algorithm, termed as *Layer Peeling*, to identify surface neighbors of each point $p \in P$ respecting the underlying surface S, and then to construct a piecewise linear surface for P. The algorithm does not have any prior knowledge of the underlying surface or whether if there is such a surface in the first place, and outputs a closed triangle mesh that best represents the implied surface based on the input sample set.

The algorithm utilizes the simple k-nearest neighborhood in constructing local surfaces. It makes use of two concepts: a local convexity criterion to extract a set of surface neighbors for each point, and a global projection test to determine an order for the reconstruction. These two concepts extend upon the k-nearest neighborhood method which many other algorithms have based their reconstruction upon. By combining with the idea of visibility testing in the global projection test, the algorithm is able to filter away points in the k-nearest neighborhood point set of each point to better approximate the local surface.

Another advantage of this algorithm is that the computational cost of the algorithm increases almost linearly in the size of the point cloud. This is largely due to the localized nature of the algorithm, although some parts of the algo-

rithm are nevertheless non-local, their computational cost is insignificant when compared with the main algorithm. Most provable surface reconstruction algorithms make use of global methods such as Delaunay based triangulation, which can make reconstructing surfaces for large point sets to be very time-consuming and impractical.

In this thesis, a proof is given such that if the input sampled point set is sampled reasonably well from a smooth surface, our algorithm is able to produce a topologically correct surface. Furthermore, the algorithm adapts well for handling under-sampled point sets. One of the reasons for that is due to the nature of the algorithm itself, as it is able to peel away layers of the surface systematically and thus avoid constructing erroneous surfaces. In order to gauge the accuracy of the reconstruction, this thesis compares the output with a commonly used algorithm for surface reconstruction, *TightCocone*, for benchmarking purposes. The results obtained are favourable and showed that the layer peeling algorithm is suitable for under-sampled point sets.

List of Tables

List of Figures

Chapter 1

Introduction

In the field of computer graphics, a lot of research has been done in striving to generate lifelike photo-realistic images and physically realistic simulations. To achieve both aims, there is a need to represent real life objects in computers, which is known as object modeling. There are two main approaches to object modeling. In the CAD/CAM (Computer Aided Design and Manufacturing) industry, a computer representation of an object is built by using a combination of simple primitives and implicit surfaces which can be used to manufacture into physical models. Another approach to object modeling is by using image sensory equipments to obtain a computer representation of an existing physical object. This approach is generally reserved for real life objects which are difficult to be decomposed into simple shapes or surfaces, such as an art sculpture.

Technology like laser scanners have allowed us to accurately and directly capture the geometrical properties of physical objects. By using laser rays to strike the surface of the object and then calculating the time delay, a 3D point sampling of the object is precisely captured. A point set sampling of an object is a form of representation of the underlying object. It is simple to interpret without any complex data

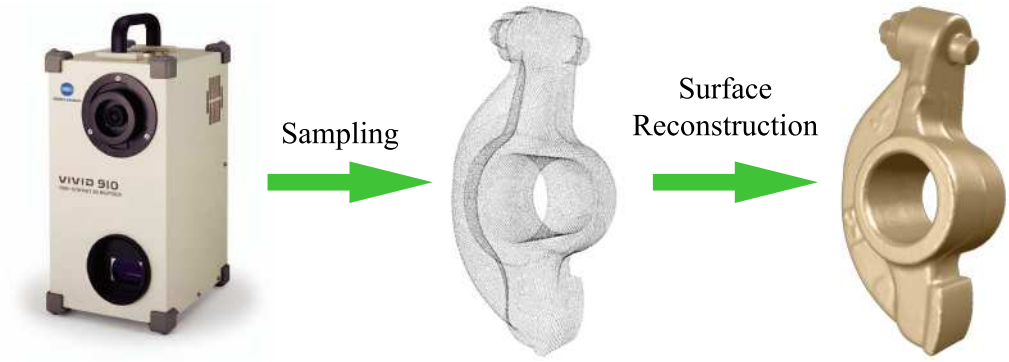


Figure 1.1: The surface reconstruction process.

structure, requires very little storage space and takes very little retrieval/reference time. However, using a point-based representation is not very useful towards producing photo-realistic computer images or physically realistic simulations. The generation of photo-realistic images from virtual objects requires that the underlying object representation is able to handle texture mapping, provide occlusion abilities and represent multiple level of details. For physical simulations, having a continuous surface makes computing deformation, collision detection, and the calculation of fractures easier. Thus, utilizing continuous surfaces, such as triangles or any other primitive types, is a more natural and appropriate form of representation for both visualization and physical simulations.

The problem of connecting the sample points appropriately to reconstruct the surface is commonly know as *surface reconstruction* in the computer graphics community. However, surface reconstruction from an unorganized point set is a difficult and ill-posed problem as no surface information is known and there is no unique solution. This problem has been the focus of research across many fields because of its wide applications. Such a reconstruction process usually involves a two-step process of first acquiring data points on the surface of the model and then reconstructing the surface from the data points, see Figure 1.1. In this thesis work, we

focus on the latter portion of this process and furthermore do not assume any prior knowledge of the original surface.

1.1 The Surface Reconstruction Problem

The first piece of work that addressed the problem of surface reconstruction from scattered data was probably by Boissonnat [20] in the mid-1980s. The surface reconstruction problem can be stated as follows: *Given a set of points P that are sampled from a surface S that is embedded in $\mathbb{R}^3$, construct a surface M such that the points in P lie on M, and M approximates S geometrically and is topologically equivalent.*

Some surface reconstruction algorithms follow another variation of the problem definition by allowing the reconstructed surface M to be "close" to the point set P, rather than directly passing through them.

The success of the surface reconstruction process depends largely on the sampling rate of the surface. A low sampling rate usually results in a poor reconstruction, whereas an overly high sampling rate usually contains noisy data samples which are difficult to handle. Most surface reconstruction algorithms depend on the sampling rate to guarantee a faithful reconstruction. Current works can be categorized into three broad types with noisy sampling having high sampling density at one end, under-sampling at the other end and lastly optimal sampling density. Most existing algorithms focus on optimal [11] and noisy sampling types [30, 44, 31] that require input data to be of good sampling density that satisfies the ε-sampling criterion (described in Section 2.2.2). However, very few works have addressed the issue on how to handle under-sampled point sets, or rather, to handle regions of under-sampling in a point set. The ability to handle under-sampled regions in a

point set is important because that is where most of the distortions and artifacts of the reconstruction usually happen at. In this thesis, we present a surface reconstruction solution that can handle this group of under-sampled point sets or regions.

There are many choices for the underlying representation of point sets, with triangular and polygonal meshes being the most commonly used. There are many practical reasons for using triangular meshes. Its data structure is not overly complex and there has been a substantial amount of research being done regarding the construction, manipulation, and visualization of triangular meshes. Other types of representation include implicit surfaces, level sets, algebraic patches, etc. Such representations allow the reconstructed surface to approximate the sample point sets and are less suitable for performing theoretical analysis on the quality of the output. In this dissertation work, we focus more on the former type of representation.

Other than the success of the reconstruction of the surface, the speed of the algorithm is also important. Most algorithms that produce triangular meshes as their output surfaces rely on Delaunay based techniques which usually have $O(n^2)$ time complexity. With the demand for higher quality object models, the number of sampled points has to increase drastically. Hence, having a linear time complexity surface reconstruction algorithm is very useful towards handling large point sets. This is one of the objectives of our work, to provide a robust surface reconstruction algorithm with output quality guarantees and yet able to achieve near-linear time complexity.

1.2 Contribution of this Thesis

This thesis proposes a novel surface reconstruction algorithm that is efficient and robust in handling under-sampled point sets. Through the understanding of why under-sampled points sets are difficult to deal with and their various characteristics, the layer peeling technique of this algorithm is able to deal with them from a fundamental approach. By itself, the layer peeling algorithm is also a reliable surface reconstruction algorithm. In this thesis, we prove that given an appropriately sampled point set, the reconstructed surface is topologically equivalent to its original underlying sampled surface.

The general outline of the algorithm involves employing a layer peeling approach to uncover a surface in a layer-by-layer manner without the use of triangulation techniques that are global in nature. At each layer, it strives to form triangle fans for some data points in order to determine their neighborhood points. These triangle fans are in turn merged together to form a surface for the input. By not employing global triangulation techniques, the layer peeling algorithm avoids the $O(n^2)$ worst case time complexity. This makes the layer peeling algorithm scale almost linearly with the size of the input point set, thus making it an attractive algorithm to be employed for large data point sets.

Here is a list of the contributions of the layer peeling algorithm towards the problem of surface reconstruction from unorganized point sets.

1. A simple and intuitive algorithm for surface reconstruction that computes in near linear timing and is comparable in output quality with other algorithms of a globalized nature.

2. A detailed proof which shows that given an appropriately sampled point set

input, the reconstructed surface formed by using the layer peeling algorithm is topologically equivalent to the original sampled surface.

3. By categorizing the points lying within the k-nearest neighborhood set of a typical point in an under-sampled region, a new perspective towards understanding their characteristics and how they can be handled is formed. This understanding is then used to develop the layer-by-layer extraction approach of the layer peeling algorithm which allows the effective reconstruction of objects that contain thin surfaces or objects that contain many different inner surfaces.

1.3 Outline of this Thesis

Section 2 reviews previous work in this area and identifies issues in existing methods. Section 3 discusses the problems that exist in reconstructing under-sampled point sets and introduces our proposed layer peeling algorithm. Section 4 describes the implementation details of the layer peeling algorithm, including the construction and merging of triangle fans. Section 5 provides an analysis of the algorithm with under-sampling and with optimal sampling conditions. Section 6 shows our experimental results, and Section 7 concludes the paper. Lastly, in Appendix A, a software implementation of the layer peeling algorithm is introduced.

Chapter 2

Background Research

There have been many works on the problem of surface reconstruction from unorganized point sets. Generally they can be classified into two broad types, namely local and global approaches. Local approaches tend to make use of the concept of k-nearest neighborhood for constructing local surfaces, whereas global approaches use algorithms such as Voronoi diagrams and Delaunay triangulations. One of the biggest advantages of using such global approaches is that the reconstruction process can be more systematically analyzed and hence easier for formulating the relationship between the input sample points with the final output. Local approaches however, tend to be computationally faster, which make the algorithms very practical. There have also been works on achieving some balance between the two major approaches by optimizing the global triangulation methods. In addition, there are also a few other newer approaches to the problem of surface reconstruction such as using geometric convection and template fitting. In this chapter, we introduce and discuss the various algorithms, concepts and proofs that are employed in surface reconstruction algorithms. A brief survey on surface reconstruction from scattered points can be found in [57]. For the rest of this

thesis, we refer to the set of input sample points as P, the original surface which the points are sampled as S, and the reconstructed surface as M.

Section 2.1 reviews the techniques and algorithms of surface reconstruction that are generally localized in nature. Section 2.2 introduces the concepts of Voronoi diagrams and Delaunay triangulations, and the various algorithms that make use of them. Section 2.3 describes the attempts that are made to optimize some of the global algorithms in order to bring them nearer to a linear timing. Finally in Section 2.4, we highlight some of the newer works in surface reconstruction.

2.1 Local Approaches

For the problem of surface reconstruction from unorganized point sets, the most natural method that comes to mind is using the local neighborhood of each point to form surfaces around them. The local neighborhood of a point consists of a set of points, excluding itself, that is the closest to it. We refer to the local neighborhood by the term, *k-nearest neighborhood*. The value k refers to the number of nearest neighborhood points which we determine to be in a local neighborhood. The local neighborhood estimate of a point refers to using its k-nearest neighborhood to make some inference on the surface region around the point. Algorithms of this nature fall into two general categories, discrete and continuous. Discrete algorithms usually produce a piecewise linear surface, formed by the sample points, while continuous algorithms tend to produce surfaces which is an approximation inferred from the sample points. Hence, the surfaces produced by continuous algorithms tend to be implicit surfaces which may or may not intersect with the sample points.

Before we present some of the local approaches used in surface reconstruction, an introduction to two important k-nearest neighborhood implementation details

is necessary. One is the efficient extraction of the set of k-nearest neighborhood for each point in the point set, and the other is the extraction of some intrinsic information useful to the reconstruction process from the k-nearest neighborhood.

2.1.1 k-Nearest Neighborhood Extraction

In order to extract the set of k-nearest neighbors for each point, a brute force approach using $O(n^2)$ time is always possible but impractical. An ideal solution is to preprocess the points in $O(n \log n)$ time, into a data structure requiring $O(n)$ space so that the queries can be answered in $O(\log n)$ time. In a 1-dimensional context, the points can be sorted in sequence and binary search can be used to answer queries. For the 3-dimensional case, a fast and efficient implementation of this approach are implemented by Arya et al. [16]. Their data structure is based on a hierarchical decomposition of space which they termed as a *balanced box-decomposition* (BDD) tree. This tree has $O(\log n)$ height, and subdivides space defined by axis-aligned hyper-rectangles (or cuboids in 3-dimensions) which have a bounded ratio between the longest and shortest side. Space is recursively subdivided into a collections of cells, each of which is either a 3-dimensional rectangle or the set-theoretic difference of two rectangles, one enclosed within the other. The tree has $O(n)$ nodes and can be built in $O(n \log n)$ time. Querying for the nearest neighbor of a point can be performed in $O(d \log n)$ time for d-dimensions. Their construction is a generalized one, which allows them to not only efficiently extract the nearest neighborhood set for each point in three dimensions, but also higher dimensions. Their implementation of the nearest neighbor extraction is employed in the algorithm proposed in this thesis.

A new piece of work on k-nearest neighborhood extraction algorithm is done

by Sankaranarayanan et al. [56]. They make use of the locality of successive points whose k-nearest neighbors are sought to significantly reduce the time needed for neighborhood extraction. Their implementation however, is more suitable for large data sets when it is not possible for the entire data set to be fitted into the memory.

2.1.2 Eigenanalysis and PCA

Principle component analysis (PCA) [59] is a very useful and appropriate tool for analyzing the k-nearest neighbors of each point. PCA decomposes a set of points in n dimensional space into n number of axes, known as eigenvectors. Each eigenvector is associated with an eigenvalue and the eigenvectors are usually sorted by the decreasing value of their eigenvalues. These eigenvectors are orthogonal to each other and their eigenvalues indicate the spread of the points in the corresponding directions. By using the coordinates of the k-nearest neighbors as input, we are able to use PCA to estimate the direction of spread of the neighbors. Therefore, if the sampling is adequate, meaning that the k-nearest neighbors should roughly be lying on a 2D plane, then we can estimate the point's normal by using the eigenvector with the smallest eigenvalue. The reason for that is because the spread of the k-nearest neighbors should be the least in the direction of the normal at that point. Therefore, the first two eigenvectors are likely to estimate the 2D plane which the point is lying on, while the last eigenvector gives an estimate of the normal at that point. In Figure 2.1, principle component analysis is performed on a sample point in 2D. In the left figure, the sample point and its neighboring points are shown. In the middle figure, the k-nearest neighbors are shown as lying within the dotted red circle. By using PCA on the k-nearest neighbors, we can extract two eigenvectors (in 3D, there are three eigenvectors) as shown in the right figure.

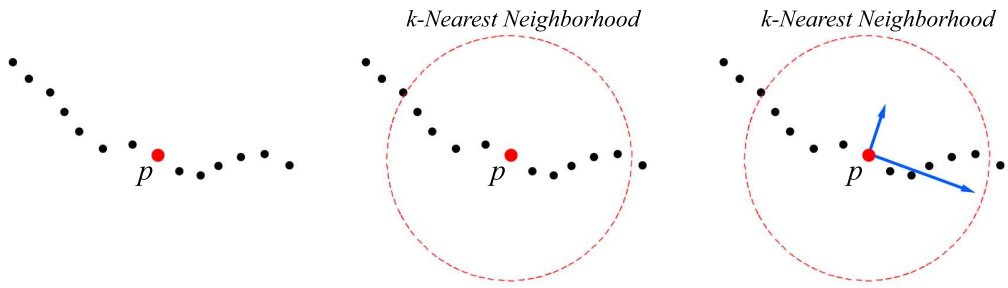


Figure 2.1: Using principle component analysis on the neighborhood of a point in 2D.

These two eigenvectors generally represent the direction of spread of the k-nearest neighborhood points, with their magnitude determined by their eigenvalues. The eigenvector with the least eigenvalue is usually taken to be a good candidate for an estimate of the initial normal vector .

Let N_p denote the set of k-nearest neighbors of a point p and o_p to denote the mean location of the points in N_p. To compute the eigenvectors and the eigenvalues of N_p, the covariance matrix of N_p is first formed. This is the symmetric 3×3 positive semi-definite matrix:

$$CovarianceMatrix = \sum_{\mathbf{x} \in N_p} (\mathbf{x} - o_p) \otimes (\mathbf{x} - o_p)$$

where $\otimes$ denotes the outer product vector operator. The outer product vector operator implies that if $\mathbf{a}$ and $\mathbf{b}$ have components a_i and b_j respectively, then the matrix $\mathbf{a} \otimes \mathbf{b}$ has $a_i b_j$ as its ij^{th} entry. For the covariance matrix, each $\mathbf{x} \in N_p$ is actually a vector that consists of the location of the point p in 3-dimensional space. Once the covariance matrix is obtained, the eigenvectors and eigenvalues can be extracted from the matrix easily [15].

2.1.3 Discrete Algorithms based on Local Estimate

This section delivers a brief overview of the common approaches to surface reconstruction that are based on local neighborhood estimate and are of a discrete nature. The outputs from such algorithms are usually a piecewise linear surface. The main advantage of discrete and local algorithms over most other types of algorithms is its computational speed.

Tangent Plane Estimation. One of the first attempt to tackle the problem of surface reconstruction is by Hoppe et al. [38]. In their work, they first employ PCA to construct a tangent plane for every point in the point set. In order to use the tangent plane as a surface estimate, it is necessary to maintain a consistent orientation between tangent planes of neighboring points. To achieve this aim, they construct a minimum spanning tree of the point set based on the absolute value of the dot product of normal vectors between neighboring points. Using this minimum spanning tree, a favorable propagation sequence is generated whereby the faces of the tangent planes are orientated based on their proximity to points that already have their orientation determined. Once this is achieved, they construct a signed distance function f by using the perpendicular distance to the tangent plane of the nearest point. The zero set $\mathcal{Z}(f)$ of the signed distance function is a piecewise linear surface, but it might contain discontinuities. Lastly, by using a contouring algorithm, they discretely sample the signed function f over a portion of a 3-dimensional grid near the data point set and construct a continuous piecewise linear approximation to the zero set.

This method generally produces a decent surface estimate when there are ample sampling. However, the main problem of this method is determining the correct

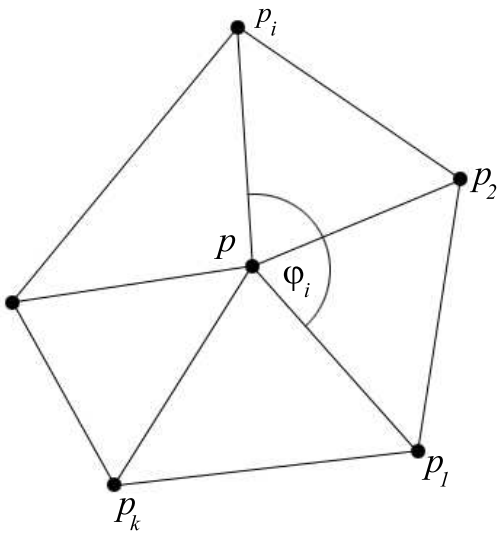


Figure 2.2: Using local neighborhood for constructing triangle fan.

orientation of the point normals. In regions of under-sampling, PCA becomes inaccurate and this leads to the formation of erroneous tangent planes. During the propagation of normal orientation, one single error can lead to defective and inconsistent surfaces being formed.

Triangle Fans. Another simple method related to surface reconstruction is using triangle fans, or collectively known as fan clouds. The idea of using a triangle fan to represent the local surface around a point is a very intuitive and computationally fast method. Linsen [48, 49] uses triangle fans as an alternative to triangle meshes, and as a choice of surface representation for points.

A fan cloud is a set of triangles that can be used to visualize and work with point clouds. It is made up of triangle fans formed at each sample point. According to Linsen [48, 49], to construct a triangle fan at a point p, one determines the set of k-nearest neighbors $N_p = \{p_1, ..., p_k\}$, computes the plane with the least sum of squared distances to $p, p_1, ..., p_k$, and projects all the points into such a plane.

After the projection, all the projected points are sorted into a sequence of angles $\varphi_i = \angle q_1 q q_i$, where q_i is the projection of p_i on the tangent plane; see Figure 2.2. In this order, the points p_i form a triangle fan of p. If the point density varies sharply around p, then the k-nearest neighborhood might not be able to form a triangle fan around p. Therefore, if $\bigtriangledown\varphi_i = \varphi_i - \varphi_{i-1} > 90°$, one replaces p_k by the $(k+1)^{st}$ neighbor and if necessary by further neighbors till the angle criterion $\bigtriangledown\varphi_i \leq 90°$ is met. However, at regions where points are sampled from surfaces with sharp corners or ridges, the best fitting plane may be normal to the surface and hence fulfilling the angle criterion might be problematic. To solve this problem, the fitting plane is rotated by 90° and the process is repeated. Once a triangle fan is constructed at each point, a triangular mesh can be constructed by using an advancing front algorithm by successively adding new triangles.

Similar to the previous tangent plane algorithm, the triangle fan approach suffers when there is insufficient sampling since it too depends on the calculation of the tangent plane using PCA. Furthermore, the formation of triangle fans based on just angle difference is prone to errors.

Ball Pivoting. The ball pivoting algorithm [19] by Bernardini et al. is a method that is similar to our approach. It is closely related to alpha-shapes [32], which is an effective tool for computing the shape of a point set. In their approach, they assume that the point set sampling P is dense enough such that a ρ-ball (a ball of radius ρ) cannot pass through the surface without touching sample points. With this assumption, they start the algorithm by placing a ρ-ball in contact with three sample points. By keeping the ρ-ball in contact with any two points, they pivot the ball until it touches another point. Each triplets of points that the ball comes into contact forms a new triangle. By "walking" the ball throughout the entire

point set, a piecewise linear surface can be formed.

Bernardini et al. [19] stated that the ball pivoting algorithm has provable reconstruction guarantees under some sampling assumptions. Their two assumptions are such that for the smooth manifold S, the sampling P satisfies the following properties.

1. The intersection of any ball of radius ρ with the manifold is a topological disk.

2. Any ball of radius ρ centered on the manifold contains at least one sample point in its interior.

These two properties ensure that the sampling is dense enough for a ρ-sized ball to "walk" on the surface without passing through it. However, under less than ideal sampling conditions as stated above, the ball pivoting algorithm might face problems when there are ambiguous cases while "walking" the ball. To resolve those situations, they assume the presence of point normals that come with range data sets. These point normals give an indication of the orientation of the surface, which allow the ball pivoting process to progress correctly.

2.1.4 Implicit Surface Algorithms based on Local Estimate

In the context of implicit surfaces, the surface reconstruction problem can be stated as follows: *Given n distinct points on a surface S in* $\mathbb{R}^3$*, find a surface S' that is a reasonable approximation to S.* To achieve this, the main idea behind most implicit surface interpolation techniques consist of building a function $y = f(x)$ whose zero level set $f(x) = 0$ approximates or interpolates the surface S. Usually $y = f(x)$ is constructed as a composition or weighted sum of simple primitives. A direct fit is

normally not possible and the reconstructed implicit surface S' is an estimation of the original surface S where the sample points might not be lying upon.

MLS Surfaces. One of the more well-established method that uses implicit surfaces is the MLS (*Moving Least Squares*) algorithm. The MLS approach is first introduced by Levin [46] and subsequently developed further in [47, 3, 4]. Basically, it introduces a projection procedure such that any point in $\mathbb{R}^3$ can be projected onto an estimated surface, based on its nearby set of neighborhood set of points. The projection procedure consists of two steps. The first is a plane estimation stage, which is similar to PCA. However, they attach weights to the neighborhood points, with priority given to closer points. The priority is determined by a θ function, where θ is a smooth, radial, monotone decreasing function. With the 2D tangent plane estimated, the second step consists of computing a local bivariate polynomial approximation to fit the neighborhood points. The projection of the point onto the estimated surface is then defined by the bivariate polynomial value at the origin. Levin proves that the surface defined as the points that project onto themselves is a two-dimensional manifold [47]. Furthermore, a general analysis of moving least squares [46] leads to the conjecture that the resulting surface is infinitely smooth as long as $\theta \in C^\infty$. However, this traditional projection approach is computationally expensive because of the non-linear optimization problem when performing the bivariate polynomial approximation. In a piece of work by Amenta and Kil [13], they analyze the stability of the projection operator for points that are not sufficiently close to the MLS surface. They notice that Levin's projection function produces output points on the MLS surface that are very near, but not actually on, to the original surface.

Adamson and Alexa [1, 2] propose a simpler projection technique for the defini-

tion of an implicit surface from point cloud data. They iteratively project a point x onto a plane defined by a weighted average of neighboring points:

$$a(x) = \frac{\sum_j \theta(\| x - p_j \|) p_j}{\sum_j \theta(\| x - p_j \|)}$$

and the normal value $n(x)$. They assume that the normals n_j for the sample points are given. Thus, the value of $n(x)$ can be computed by averaging the input point normals:

$$n(x) = \frac{\sum_j \theta(\| x - p_j \|) n_j}{\| \sum_j \theta(\| x - p_j \|) n_j \|}$$

If however the input point normals are not available, the value of $n(x)$ is determined by using PCA on the nearby neighborhood points. Further improvement to the MLS algorithm includes the work by Fleishman et al. [33] where they make use of the forward search method to detect outlier points which can affect the reconstructed surface. The forward search method iteratively refines the surface by adding one sample at a time and generally adds the good sample points first before adding outlier points. In this way, their approach is able to handle sharp features well.

Radial Basis Functions. Radial Basis Functions (RBF) are used commonly in implicit surface reconstruction algorithms. Carl et al. [23] and Turk and O'Brien [37] use globally supported radial basis functions to fit data points by solving a large dense linear system. They use a composite function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ defined as a linear combination of simple primitives:

$$f(p) = \sum_{i=1}^{m} \alpha_i \phi(\| p - c_i \|)$$

where α_i and c_i are unknown weights and $\phi : \mathbb{R}^+ \rightarrow \mathbb{R}$ is the radial basis function. Some common radial basis functions include $\phi(r) = r$, $\phi(r) = r^3$ or $\phi(r) = r^2 \log(r)$. Surface reconstruction using these radial basis functions gives a smooth implicit interpolating surface, since the zero level set of the function have the same continuity properties. To interpolate a surface through the sample point, a function is constructed such that it evaluates to zero at the surface, and non-zero at off-surface points. The function values of sample points naturally evaluate to zero, since they are lying on the surfaces. Off-surface points are generated by adding new sample points, usually along the normals of the original set of sample points. Hence, let $F = f_i$ be a set of n values of the function f at some scattered distinct points $P = p_i \in \mathbb{R}^3$:

$$f(p_j) = \sum_{i=1}^{m} \alpha_i \phi(\| x - c_i \|) = f_j$$

for $\forall i = j \ldots n$.

However, the computation process of solving the linear system of equations is very involved and time-consuming. To improve on the timing of computing the fitting function, Morse et al. [51] reduce the computational cost of the linear system by using locally supported radial basis functions. They segment the entire point set into regions (or patches) which they fit radial basis function onto. Developing this idea further is Ohtake et al. [53], they propose the Multi-level Partition of Unity Implicits method which is a hybrid between algebraic patches and radial basis

functions. They use an adaptive octree based segmentation method to subdivide the region of space which is occupied by the input point set. Inside each cell, they choose different fitting functions based on the property of the sample points located within it. Each patch within a cell is then subsequently blended together. Xie et al. [61] use a similar approach by employing the use of the modified Shepard's blending method for blending multiple patches together. In a recent work by Samozino et al. [55], they use a set of points located on the medial axis as centers of the RBF instead of the input point set. In this way, fewer centers of RBF are used and this leads to faster computations.

Iso-Surface Extraction. A newer approach of using implicit functions for surface reconstruction is by calculating the characteristic function of the solid model which the input point set data is sampled from and then using iso-surfacing techniques to extract the boundary of the solid model. In the work by Kazhdan [42], he makes use of the Stokes' Theorem for computing the characteristic function. Specifically, by only using the location and normal information of the input point set data, the Fourier coefficients of the characteristic function can be computed. Henceforth, the characteristic function can be calculated by using inverse Fourier transform. Similar to this idea is the work by Kazhdan et al. [43], where a Poisson formulation is adopted instead.

In both [42, 43], they require the knowledge of oriented normals for the input point sets. In the work by Hornung [39], an iso-surface can be extracted without having normal information. By using a voxel grid, their method reconstructs the surface from a volumetric unsigned distance function. This unsigned distance function represents the probability that the surface passes through a given voxel. The closed surface can then be extracted via graph-cut based energy minimization.

They can efficiently process highly non-uniformly sampled input data with large gaps, without loosing fine details in densely sampled regions.

2.2 Global Approaches

The Voronoi diagram of a set of points is the subdivision of the whole space into convex cells where each cell is associated with exactly one point. The dual of the Voronoi diagram is the Delaunay triangulation, which is a cell complex that subdivides the convex hull of the points. Using Delaunay triangulation as a tool for surface reconstruction is especially suitable, since surface reconstruction boils down to establishing neighborhood connections between nearby sample points. In the previous Section 2.1, we introduced some of the local approaches to surface reconstruction. One of the disadvantages of using local approaches is that it is difficult to perform any theoretical analysis on them, although they might be practical algorithms. Global algorithms, on the other hand, are suitable for such analysis. For a general survey on Delaunay based surface reconstruction methods, see [24]. In this section, we first introduce some of the basic concepts of Voronoi diagrams, Delaunay triangulations and the sampling definitions. Then we introduce some of the commonly used surface reconstruction algorithms involving these structures.

2.2.1 General Concepts

The *Voronoi diagram* $V(P)$ of P is a cell decomposition of $\mathbb{R}^3$ to convex polytopes. Every Voronoi cell corresponds to exactly one sample point and contains all points of $\mathbb{R}^3$ that do not have a shorter distance to any other sample points, i.e. the Voronoi cell corresponding to $p \in P$ is given as follows:

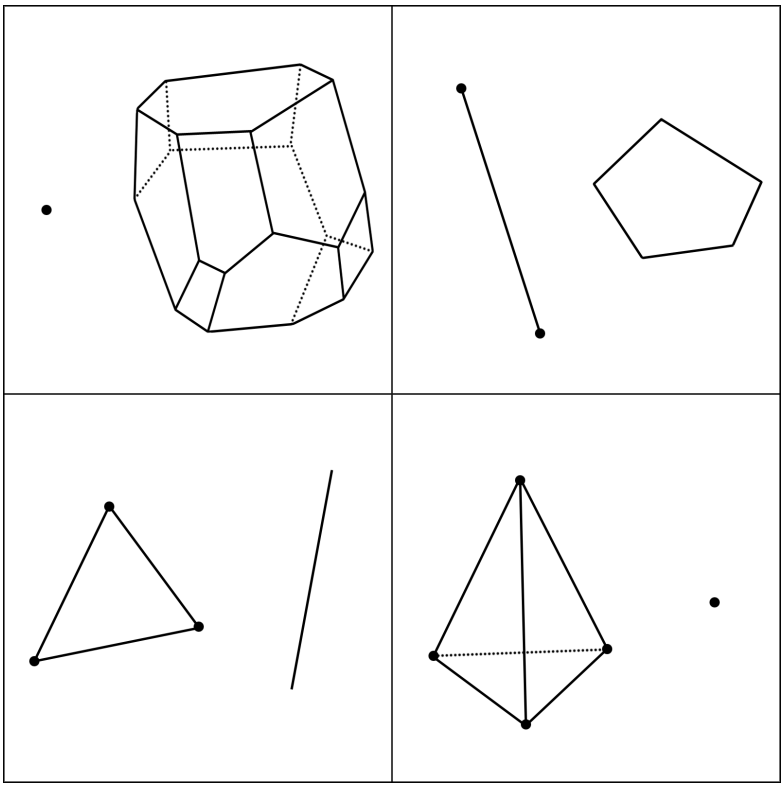


Figure 2.3: The dual relationship between Delaunay simplices and Voronoi Cells. Clockwise from top: a Delaunay point with a Voronoi polyhedron, a Delaunay edge with a Voronoi face, a Delaunay face with a Voronoi edge, and a Delaunay tetrahedron with a Voronoi vertex.

$$V_p = \{x \in \mathbb{R}^3 : \forall q \in P, \|x - p\| \leq \|x - q\|\}$$

Facets shared by two Voronoi cells are called Voronoi facets, edges shared by three Voronoi cells are called Voronoi edges and points shared by four Voronoi cells are called Voronoi vertices. The term Voronoi object can denote either a Voronoi cell, facet, edge or vertex. The Voronoi diagram is the collection of all Voronoi objects.

The *Delaunay triangulation* $D(P)$ of P is the dual of the Voronoi Diagram, in the following sense (see Figure 2.3). Whenever a collection $V_1, ..., V_k$ of Voronoi

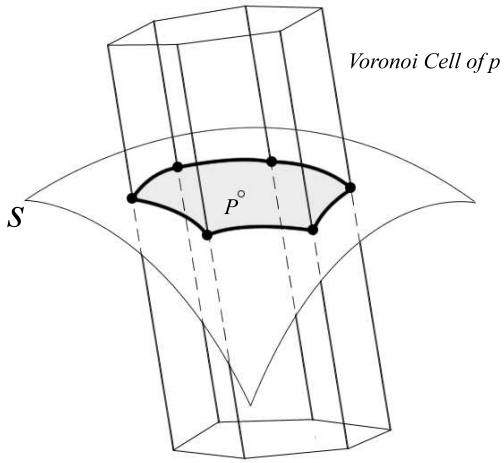


Figure 2.4: The restricted Voronoi cell of a point p, shown in shaded region. It is formed by the intersection of the Voronoi cell of p with the surface S.

cells corresponding to points $p_1, ..., p_k$ have a non-empty intersection, the simplex whose vertices are $p_1, ..., p_k$ belong to the Delaunay triangulation. It is a simplicial complex that decomposes the convex hull of the points in P. That is, the convex hull of four points in P defines a Delaunay cell (tetrahedron) if the common intersection of the corresponding Voronoi cells is non empty. Analogously, the convex hull of three or two points defines a Delaunay face or Delaunay edge, respectively, if the intersection of their corresponding Voronoi cells is non empty. Every point in P is a Delaunay vertex. The term Delaunay simplex can denote either a Delaunay cell, face, edge or vertex.

We call the *restricted Voronoi diagram* of P restricted to S as the intersection of every Voronoi object in $V(P)$ with S. In Figure 2.4, an illustration depicting the intersection of a Voronoi cell with the surface to form a restricted Voronoi cell is shown. The restricted Voronoi diagram is then denoted as $V_S(P)$. The dual of $V_S(P)$ is the *restricted Delaunay triangulation* of P restricted to S, denoted as $D_S(P)$ (see Figure 2.5).

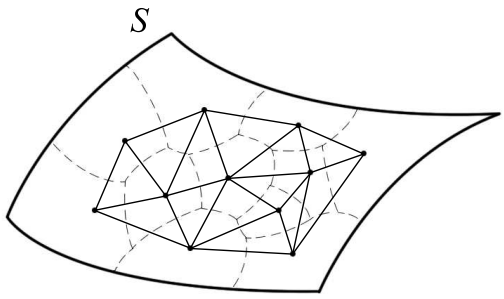


Figure 2.5: The restricted Delaunay triangulation of a partial sampling on a surface S. The dashed lines are the restricted Voronoi polygons and the solid lines are the restricted Delaunay triangulation.

2.2.2 Epsilon Delta Sampling

The *medial axis* of the surface S is the closure of the set of points in $\mathbb{R}^3$ that has two or more closest points in S. The *local feature size*, $f(p)$, at a point p on S is the least distance of p to the medial axis. Figure 2.6 shows the medial axis of a surface. The dotted line connecting the point p to the medial axis is the local feature size of p. The medial balls at p are defined as the balls that touch S tangentially at p and have their centers on the medial axis. A point cloud P is called an *ε-sample* of S (where $0 < \varepsilon < 1$), if every point $p \in S$ has another point in P at a distance of at most $\varepsilon f(p)$. Thus, for an ε-sample of S:

$$\forall p \in P : \exists q \in P, \|p - q\| \leq \varepsilon f(p)$$

An ε-sample generally gives an estimation on the minimum number of samples that is required to sample a surface. For global algorithms, this condition is necessary in order to place some guarantee on the accuracy of the output surface. Generally, the more samples that the point set has, the more accurate the computed surface will be. However, this is not strictly the case for local algorithms, as local algorithms depend a lot on the uniform spread of the k-nearest neighbors

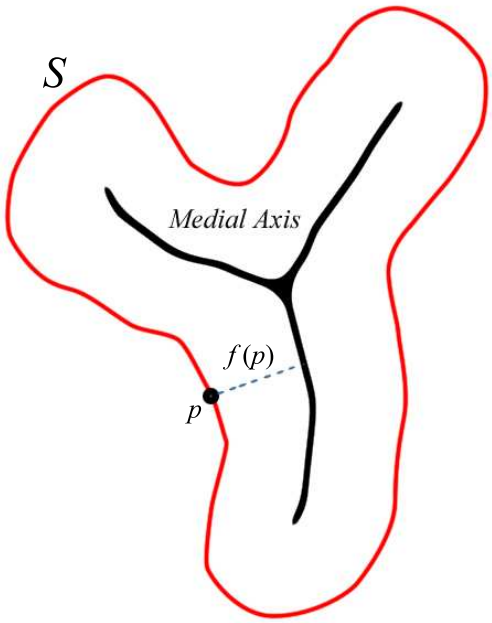


Figure 2.6: Medial Axis Diagram.

around the local region. Thus, having a highly but irregularly sampled point set can actually work against a localized algorithm. Thus, another stricter sampling condition, known as an (ε, δ)-sampling [28] is required.

An ε-sample of S is called an (ε, δ)-sample if it satisfies an additional condition:

$$\forall p, q \in P : \|p - q\| \geq \delta f(p)$$

for $\frac{\varepsilon}{2} \leq \delta < \varepsilon < 1$.

Essentially, an (ε, δ)-sample implies that any two sample points have to be sufficiently apart from each other, yet the whole point set must be sufficiently well sampled. A method to obtain an (ε, δ)-sample from an ε-sample in linear time is provided in [34].

2.2.3 Algorithms based on Global Triangulations

Global triangulation methods such as Delaunay triangulations form triangles where their vertices tend to be points which are close together. The reason is that Delaunay triangulation is the dual complex of the Voronoi diagram, which divides the total space according to which point that it is closest to. Hence, such a construction naturally has its application in surface reconstruction, since surface reconstruction too requires the formation of surfaces among points that are close together. However, using Delaunay triangulation in 3-dimensions forms a tetrahedralization, and thus most algorithms need to deal with the problem of removing triangles from the tetrahedralization in order to form a 2-dimensional manifold.

Crust and Power Crust. One of the earliest works which aims to provide a provable surface reconstruction algorithm is [10, 8]. It is the first piece of work that provides a constraint on the input point set in order to guarantee that the reconstructed surface is homeomorphic to the original surface that the points are sampled from. Their work is an extension from curve reconstruction in 2D [9]. For each sample point, they consider its Voronoi cell and choose two farthest Voronoi vertices as the *"poles"* of the sample point. The two poles ideally should be located on both sides of the surface. Since the surface is unknown, the first pole (also known as the positive pole) is selected as the farthest Voronoi vertex, while the other (negative pole) is the farthest Voronoi vertex in the opposite half-space. After the poles are found, a Delaunay triangulation of the sampled point set combined with the poles are computed. By removing all the triangles in the Delaunay triangulation except those triangles whose three vertices are all sample points, an initial manifold surface is obtained. They then proceed to filter away triangles

whose normals form a large angle with any of the poles of its vertices. Finally, a vertex lying on the convex hull of the point set is selected and its orientation is determined by setting the pole facing away from the point set to be a positive pole. Using this vertex, a breath first search is applied to orientate all the poles and triangle to produce a piecewise linear surface, termed as the crust. In their paper [8], they claim that if the sampling is an ε-sample with $\varepsilon \leq 0.06$, the computed crust is homeomorphic to the original sampled surface.

Developing the idea is the work on *Power Crust* [12]. In this paper, Amenta et al. notice that the poles of the sample points are actually a good approximation to the medial axis of the original surface. The poles can be divided into two sets, one that exists inside the surface manifold and the other outside. Using the two sets of poles to construct polar balls, the intersection between them can be used to roughly approximate the surface. Therefore, they first construct a Voronoi diagram using the sample points, and use the poles of each point to further construct a power diagram to produce a piecewise-linear surface approximation of the surface. It should be noted that the piecewise-linear surface that is produced does not always go through all the original sample points.

Generally, the main criticism against these two approaches is that it take too long for practical computation. In both algorithms, they require two calculations of the Voronoi diagram or Delaunay triangulation.

Cocone And Tight Cocone. Cocone [11] is the first paper that presents a proof that its reconstructed surface is homeomorphic to the original surface if their sampling requirement is fulfilled. Their proof is based on the following observation. Let T be a set of triangles spanning the sampling points satisfying the following three conditions :

1. T contains all triangles whose dual Voronoi edges intersect S.

2. Each triangle in T is small, that is, their circum-circle has a small radius compared to the local feature size.

3. All triangles in T are flat, that is, their normals make small angles with the normals to the surface at their vertices.

Then, if T fulfills the three conditions, then any piecewise linear manifold extracted from T that spans all its vertices must be homeomorphic to S. With that, they define the term Cocone, see Figure 2.7. The definition of *Cocone* C_p of a point p is as follows:

$$C_p = \{y \in V_p : \angle((y-p), \mathbf{n}) \geq \frac{3\pi}{8}\}$$

In other words, C_p is the complement of a double cone (clipped within V_p) centered at p with an open angle $\frac{3\pi}{8}$ around the axis aligned with $\mathbf{n}$, the vector from p to the positive pole of p.

To obtain such a triangle set T, Amenta et al. used a similar method to the crust algorithm. A pole is obtained for each point as the farthest vertex in its Voronoi cell. Based on this pole, they defined the Cocone. Voronoi edges which make an intersection with the Cocone of each of its 3 corresponding sample points are placed in a set E. The dual Delaunay triangle set of all the Voronoi edges in E is then formed. Similar to the crust algorithm, triangles which make sharp angles with their adjacent triangles are filtered away. Finally, a depth-first walk over the adjacency graph of the remaining triangles extracts the final reconstructed piecewise linear surface. Similarly, they prove that their algorithm computes a

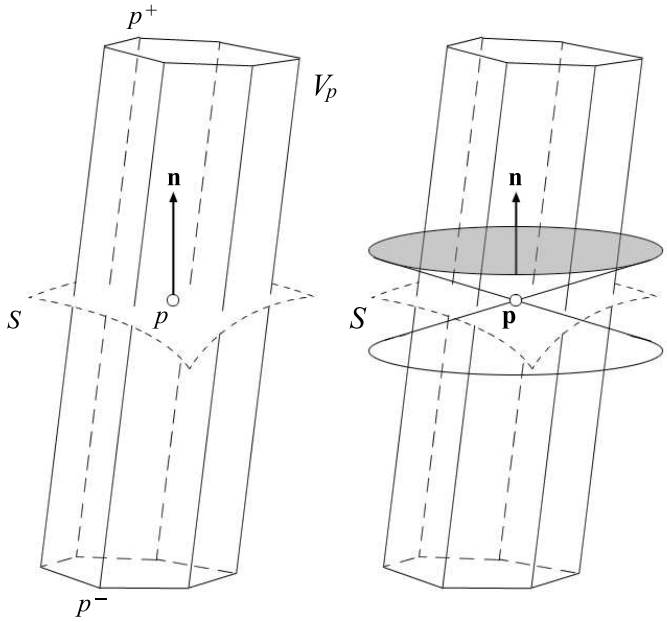


Figure 2.7: Left: A Voronoi cell of a point p intersecting with the surface S. The two poles p^+ and p^- is shown. Right: An illustration of the Cocone shown as two inverted cones.

piecewise linear surface homeomorphic to a surface for which P is an ε-sample with $\varepsilon \leq 0.06$

A provable surface reconstruction algorithm tells us about the quality of the output when its sampling condition is met. However, in most practical situations the sampling requirement is seldom fulfilled. Hence, for practical reasons, it is important to take extra steps when handling areas of under-sampling in order to produce reasonable outputs. The Cocone algorithm faces exactly such problems around areas of under-sampling. The reason for this is because of the fact that at areas of under-sampling, the Voronoi cell of a point is no longer of an elongated shape. Thus, the poles of the point differ greatly from the actual normal, resulting in erroneous triangles being placed in set E.

In [27], they attempt to detect areas where under-sampling is occurring by

testing for two properties in each Voronoi cell. The first property is that the largest possible ball centered at a sample point that is enclosed by its Voronoi cell must be less than the distance to its pole by a factor of ρ, a parameter used in their algorithm. The second property is that the angle between the estimated normals of itself and its neighboring Cocone neighbors is less than 0.14 radians. The Cocone neighbors are defined as the set $CN_p = \{q \in P : C_p \cap V_q \neq \emptyset\}$. Once a point fails either of the two properties, it is marked as being located in an area of under-sampling. Thus, by identifying which are the points that are located in areas of under-sampling, the Cocone algorithm can choose to ignore constructing triangles that contain vertices from those points.

Having a surface reconstruction algorithm producing holes on the surface is not desirable to many applications. For most applications, such as CAD designs, they require that the output surface be water-tight, i.e. a surface that bounds a solid. A piece of work by Dey and Goswami [29] expands and develops the Cocone algorithm further by producing a water-tight output, even when there are regions of under-sampling in the point set. The main idea in their work, *TightCocone*, is to label the Delaunay tetrahedra computed from the input sample as "in" or "out" according to an initial approximation of the surface and then peeling off all the tetrahedra that are marked as "out", leaving the rest as the "in" tetrahedra. The tetrahedra are classified based on the set of surface triangles at each point, depending on which side of the surface triangles that they lie on. The boundary of the remaining "in" tetrahedra is output as the water-tight surface. The initial approximation is obtained through the Cocone algorithm which is possibly having holes and other artifacts. In a way, the TightCocone approach can be seen as a "sculpting" process. Another piece of work that is similar in this aspect is by

Attene and Spagnuolo [17].

2.3 Linear Time Triangulation Algorithm

Global triangulation algorithms are usually based on algorithms such as Delaunay triangulations and Voronoi diagrams. These approaches are very algorithmic and thus more open to theoretical analysis, which is a big advantage. However, Delaunay triangulation algorithms have a worst-case computational timing of $O(n^2)$, which is undesirable for large point sets. Furthermore, in constructing a piecewise linear surface, a point is only likely to form triangles with nearby points. Hence, it seems that redundant operations are wasted to construct a global structure when a local structure seems to suffice.

Funke and Ramos Algorithm. Funke and Ramos [34] observe that the Cocone algorithm uses 3D Delaunay triangulation which can result in a worst case timing of quadratic size. Furthermore, a point is only likely to form triangles with nearby points. Hence, the Cocone algorithm can be optimized further to produce a near-linear timing. The most time consuming operation in the Cocone algorithm is the generation of the Voronoi diagram, which is a global data structure. They deduce an initial estimate of the Voronoi cell of each point p, by using a well-separated-pair decomposition technique [22]. Using this initial estimate, they decimate the point set to get a sampling similar to (ε, δ)-sample (described in Section 2.2.2). Once such a point set is obtained, they use an incremental approach to calculate the Voronoi cell for each point which can then be used in the Cocone algorithm. Hence, they are able to avoid creating the Voronoi diagram as a global structure and thus avoid incurring a worst case quadratic time complexity. The time complexity for their

approach is $O(n \log n)$. However, it should be noted in perspective that although their time complexity is near linear, some of the approximations that are used can be too time consuming, making the algorithm unsuitable for practical usage.

Gopi Algorithm. Gopi et al. [36] similarly see the potential in reducing the computational complexity of Delaunay triangulation for surface reconstruction. Their reconstruction process is based on the advancing front techniques. They used a normal estimation method, similar to Hoppe et al. [38], to create local surfaces at each point. The local surface that is constructed at each point, is based on a 3D Delaunay triangulation that is projected onto a 2D plane. By using such a projection, they are able to effectively speed up the calculation for the Delanuay triangulation computation. And similarly with [38], the tangent plane estimation process is unreliable at regions of undersampling and reconstruction tends to fail.

2.4 Meshing Fitting Algorithm

Another different approach to the problem of surface reconstruction is by the gradual deformation of a coarse representation and approximation of the surface towards a more detailed mesh representation of the input point set. Such an approach can either start from a simple template model which resides within the surface manifold and gradually growing it to fit the input point set, or from the convex hull of the input point set and then gradually sculpting it.

Geometric Convection Algorithm. The geometric convection algorithm [25] is based on the convection model introduced by Zhao et al. [62]. They solve the surface reconstruction problem by first computing a closed surface that minimizes

a global distance function to the input point set. Such a closed surface is usually the convex hull of the input point set. The approximated closed surface is then gradually shrunk to fit more closely to the point set. Each facet of the surface (usually Delaunay in nature) must be oriented consistently towards the interior of the shape that fulfills an oriented Gabriel property. This means that each oriented facet must have half of its minimum enclosing sphere, which is directed towards the interior of the closed manifold, empty of any point in its interior. This approach is then further developed by Allègre et al. [5, 6] whereby a framework is developed to adapt the geometric convection algorithm for large point sets.

Template Fitting Algorithm. In contrast to the geometric convection algorithm where a coarse mesh is gradually shrunk to fit the point set, another approach is to gradually grow a small coarse mesh model to fit the input set. Such an algorithm is termed as template fitting algorithm. In the work by Sharf et al. [58], they gradually grow a deformable mesh model inside the point cloud and evolve it in incremental steps to finally fit the input point set. In the process of deformation, mesh optimization operators are applied to maintain a high mesh quality. It guarantees water-tightness and allows simple tracking of topological events. When the model is sufficiently close to the input point set, a moving least squares projection [3] is applied to map it to the final deformed mesh model. Other type of template fitting algorithms include works by Kraevoy and Sheffer [45] and Stoll et al. [60]. Their works comprise of having a template model which is generally similar to the surface which the input point set is sampled from. This template model is then used to guide the reconstruction process. However, these methods require user interaction to establish reference points between the template model and the input point set.

2.5 Overview of Existing Approaches

The following figure shows an overview of the characteristics of the existing approaches that are used in surface reconstruction.

Algorithm	Require Normals	Points on Surface	Non Linear Time	Piecewise Linear	Handle Noisy Data
Tangent Plane		Yes		Yes	
Triangle Fans		Yes		Yes	
Ball Pivoting		Yes		Yes	
MLS	Yes				Yes
RBF	Yes				Yes
Iso-Surface	Yes				Yes
Power Crust			Yes	Yes	
TightCocone		Yes	Yes	Yes	
Linear Time Triangulation		Yes		Yes	
Geometric Convection	Yes	Yes	Yes	Yes	
Template Fitting	Yes		Yes		Yes

Figure 2.8: Summary of existing methods in surface reconstruction.

Referring to Figure 2.8, "require normals" states whether the algorithm requires normal information from the input data set. "Points on Surface" refers to whether the input points lie on the reconstructed surface. "Non linear time" states whether the algorithm scales non-linearly with the size of the data set. The last

two categories of “piecewise linear” and “handle noisy data” refers to whether the output is a triangulated surface and whether the algorithm is capable of applying it to noisy data sets. It should be noted however, that none of the methods that are introduced has openly tackled the issue of undersampling in the input point set. Hence, in our piece of work we attempt to address the problem of undersampling directly.

Chapter 3

The Layer Peeling Algorithm

This chapter develops an effective surface construction algorithm which focuses on under-sampled point data sets. As to be illustrated in chapter 5, this algorithm produces a piecewise linear surface that is homeomorphic to the original surface that the point set is sampled from, provided that the sampling fulfills certain criterion. We term this algorithm the *Layer Peeling Algorithm*. Our layer peeling algorithm is derived fundamentally from two areas. The first area is from the rendering perspective and the second is from the segregation of the different categories of points that exist in a k-nearest neighbor set within an under-sampled region. By utilizing both concepts, the algorithm is able to filter away undesirable points in the neighborhood, using a combination of both local and global methods.

Section 3.1 discusses some of the problems that face current surface reconstruction algorithms when dealing with under-sampled point sets. Section 3.2 illustrates the different categories of points that exist in an under-sampled k-nearest neighborhood. Section 3.3 introduces the fundamental principles that the layer peeling algorithm is based on. Finally in Section 3.4, an outline of the layer peeling algorithm is given.

3.1 Problems of Under-Sampled Points Sets

As introduced in Section 2, there are mainly two ways to perform surface reconstruction. The first way uses k-nearest neighborhood to form local surfaces such as tangent planes, triangle fans, or implicit surfaces for each point. The local surfaces that are formed are then merged together to form a single 2D manifold. The other method uses global triangulation methods such as Voronoi diagrams or Delaunay triangulations. By working on the simplicial complexes that these two methods provide, a piecewise linear surface can be extracted.

In most cases where there are adequate sampling, the surface reconstruction process tend to be smooth and error free. However, in regions where under-sampling occur, it is possible to see distortions, artifacts, and holes appearing in these reconstructions. The problem that most algorithms face is the fact that the importance value given to each neighborhood point when constructing local surfaces is independent of their distance value, which can lead to inaccurate reconstruction. Variations include imposing a radial [23] or Gaussian function [3, 4] on the distance value to further lessen the impact of erroneous neighboring points. For local surface estimates, the accuracy of the extracted normal vector for each point usually determine how much the reconstructed surface resembles the original sampled surface.

As shown in Figure 3.1, there are usually two reasons why using the k-nearest neighborhood tend to make erroneous normals estimation in under-sampled point sets. On the left picture, two points p and q are very near to each other. Using a k-nearest neighborhood approach, the point p is located within q's neighborhood and vice versa. As such, it is difficult to correctly orientate the normals at both p and q in the correct direction. On the right picture, a k-nearest neighborhood of a

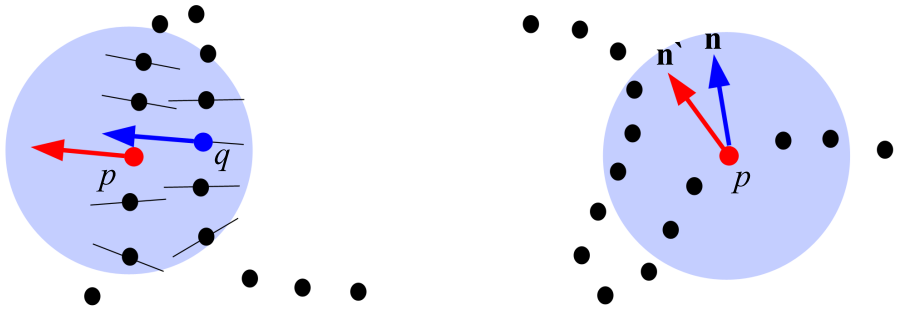


Figure 3.1: Left: Two points p and q are too close to each other and their normals orientations are difficult to resolve. Right: The PCA performed in an under-sampled neighborhood of a point p does not provide an accurate estimation.

point p has included some other points which do not lie geodesically close to p. A PCA performed on the k-neighborhood of p gives a slightly different estimation of the normal $\mathbf{n}'$ than the correct normal $\mathbf{n}$ at p. Using Hoppe et al. [38] algorithm as an example to run on the Screwdriver point set model in Figure 3.2, the effects of under sampling can be seen clearly. The tip of the screwdriver point model is sharp and the two opposite surfaces are close to each other. The two images show the direction of the normal vectors at each point sample near the tip of the screwdriver point model. The left image shows the normal estimation result when a simple k-nearest neighborhood approach in Hoppe el at. algorithm is employed. The right image shows a better normal estimation when the correct neighborhood of each point is used. By correct neighborhood, it is taken to mean points which lie geodesically close, rather than geometrically close.

For most triangulation algorithms such as Crust or Cocone [8, 11], they too require an initial normal vector estimate in order to start their surface construction process. Their estimation is based on the concept of poles, as described in Section 2.2.3. The choice of using poles to estimate the normal values comes from the observation that the shape of the Voronoi cells of sampled points from a surface

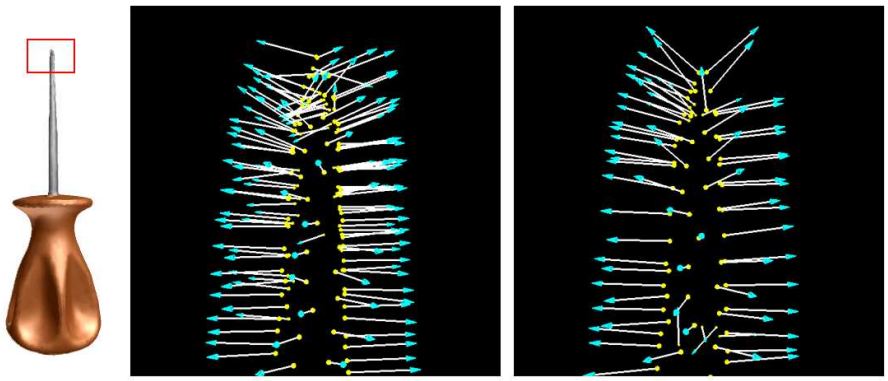


Figure 3.2: Normal estimation on the Screwdriver point set model. Two different neighborhood is used. The result on the left is done by using k-nearest neighborhood, while on the right the result is obtained by using the correct set of neighbors.

is usually elongated, and most likely along the direction of their normal vectors. Hence, the choice of using the furthest Voronoi vertex as an estimate of the normal vector is typically a good choice. However, in regions of under-sampling, such a property does not always hold true and can therefore lead to erroneous results.

3.2 Different Types of Neighborhood Points

The assumption that most local approach algorithms depend on is that all the points in the k-nearest neighborhood set are geodesically nearby. In other words, it means that the surface which all the k-nearest points are lying on forms a connected component. If this assumption is fulfilled, most local approaches tend to work correctly and produce a reasonable surface that approximate the original manifold well. However, when under-sampling occurs, such an assumption may not always be true. In those situations, the k-nearest neighborhood is likely to contain points from other regions of the original surface.

In general, there are three possible types of sampled points in the region of

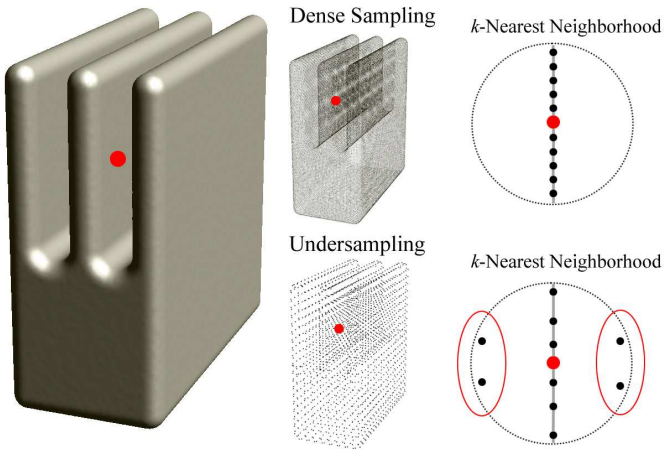


Figure 3.3: The k-nearest neighborhood set of a point in both dense sampling and under-sampling conditions.

interest around a single sampled point p; see Figure 3.3. Taking the points lying on the original surface as a divider, it is possible that other sampled points are lying above and below in the k-nearest neighborhood set. With no additional information that is available on the point set or the sampled surface, it is difficult to differentiate among the three types of points. This is the main challenge for any surface reconstruction algorithm. For under-sampled point sets, most of the k-nearest neighborhood sets contain all three types of sampled points.

3.3 Algorithmic Rationale

In our layer peeling algorithm, the central idea is determining how to differentiate between the three types of sampled points in a typical k-nearest neighborhood set. In order to explain how the layer peeling manages to resolve that, we begin first with two simple observations about closed-manifold in general and their influences on our algorithm.

Fact 1 *For any closed-manifold surface in 3D that is watertight and bounds a volume, a ray intersecting the surface is always alternating between front-facing (i.e., from outside the bounded volume to the inside) and back-facing intersection.*

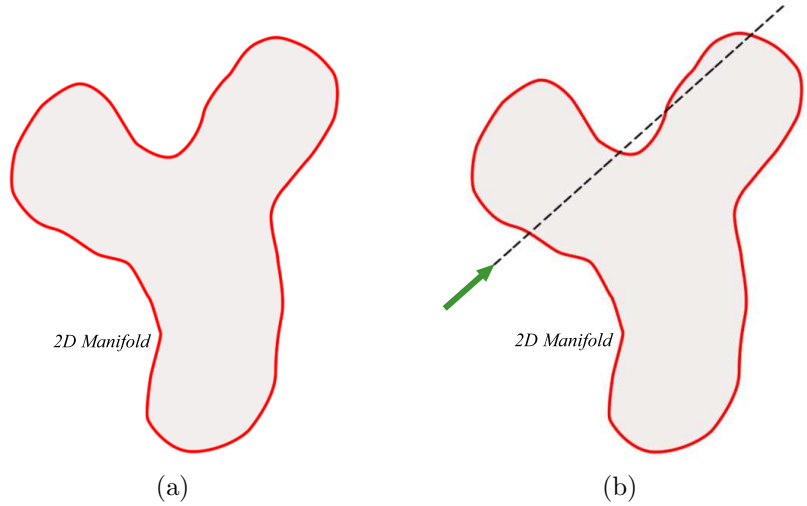
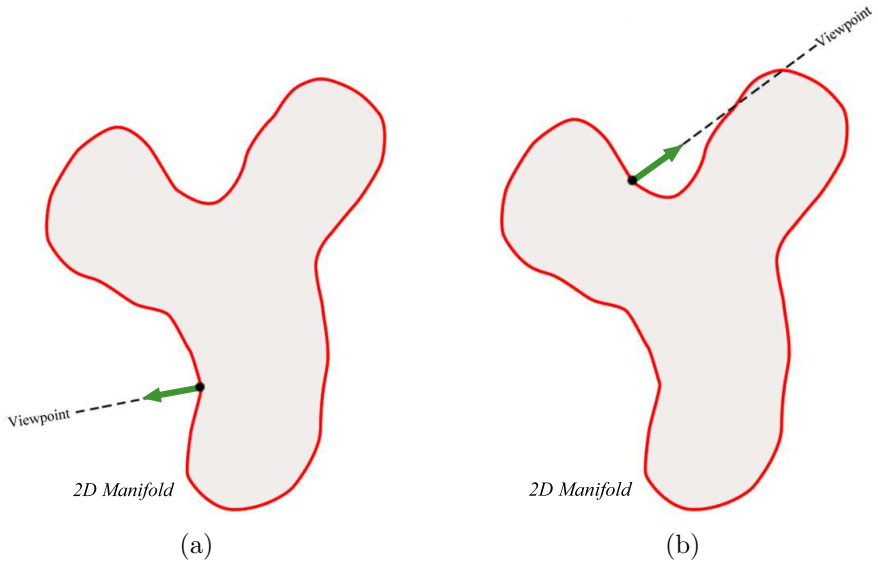


Figure 3.4: Intersection between a closed manifold and a ray in 2D.

For a closed manifold, there exists no path that leads from the inside of the bounded volume to the outside (or vice versa) without passing through the surface. In Figure 3.4(a), a typical manifold in 2D is shown. For any ray coming from infinity which intersects the manifold, it first strikes the front face of the manifold. Thereafter, the ray will be inside the manifold and the next intersection of the ray with the manifold be with a back facing one. This process, as shown in Figure 3.4(b), can potentially happen a few times for a single ray. Generally speaking, each intersection brings the ray from outside into the inside of the bounded volume, and another intersection is needed to bring the ray out of the bounded volume.

Fact 2 *Consider a rendering of a point set using splats or small disc at each point. For a viewpoint aligned along the normal of a point, the point itself is visible if,*

and only if, no splats rendered at the other points intersect with the normal ray from the point.



(a) (b)

Figure 3.5: Visible line of sight for a point on the manifold.

In rendered images, any object closer to viewpoint occludes the other. Fact 2 thus follows. Refer to Figure 3.5(a), the black point is visible if the viewpoint is placed along the normal at that point (shown as the green arrow). Conversely for Figure 3.5(b), it is obvious that the point is occluded and thus not visible from the viewpoint. These two facts allow us to infer a few things about manifold in general. Firstly, for points lying on the convex hull of the point set, it is trivial to use Fact 2 to claim that they can always be visible if the viewpoint is placed along the direction of their normal vectors. The term visible meant that no other points should be lying between the viewpoint and the local surface at the point. In other words, for such a point lying on the convex hull, there are only two types of points in its k-nearest neighborhood set, since no point will be lying above its local surface. Secondly, once we determine the local surface for a point, by removing and

keeping the viewpoint at the normal vector, we are able to see some back facing surface inside the manifold (Fact 1). Similarly using Fact 2, for a point lying on the back facing surface inside the manifold, its k-nearest neighborhood set consists of two types of points, since no point is lying below its local surface.

Constructing the local surface for a point with only two types of points in its k-nearest neighborhood is generally much easier, since we only have to take the convex hull from one particular direction as its local surface. Hence, the approach is to process or propagate the surface construction from the outermost surface (or layer), since those are the points that are having only two types of points in their k-nearest neighborhood set. After we construct the surfaces for these points, we remove them and expose another layer which is back facing. Points lying on this newly exposed layer are also likely to contain only two types of points in their k-nearest neighborhood set. By repeating this process, we can perform surface reconstruction on the point set. This is the rationale of our layer peeling algorithm.

3.4 Algorithm Outline

In this section, an outline of the layer peeling algorithm is given. Most of the implementation details such as the global projection test and the triangle fan construction are omitted and are explained further in Section 4. Using the two facts as stated in the previous section, we approach the problem of reconstructing surfaces from point sets as follows. We start the reconstruction process from points that lie on the outermost layer (i.e. points that lie on the convex hull of the point set). By using Fact 2, we can extract the local surface around those points. Once a layer is found, we can use Fact 1 to recursively extract the remaining layers. The

1. Compute k-nearest neighbors of each point using the ANN software [16, 52].
2. Perform eigenanalysis for each point so as to select a seed to start the layer peeling process to construct the surface mesh M.
3. Divide points that are not yet part of M into subsets where two points are in the same subset when one is a k-nearest neighbor of the other.
4. For each subset, repeatedly construct a triangle fan at a boundary point (based on Fact 2 applied to within each subset) to merge it into M. Note that the orientation of a triangle fan is flipped during the even iterations of this step (as stated by Fact 1).
5. Each point in an isolated group of three or less points (that cannot possibly form a volume) is merged to its nearest triangle in M.
6. Step 3 to Step 5 create a layer of the point set; we now repeat from Step 3 to Step 5 until no more triangle fans (i.e., another layer) can be constructed.

Figure 3.6: The outline of the Layer Peeling Algorithm.

outline of the algorithm is given in Figure 3.6.

Refer to Figure 3.7 for a 2D description of the layer peeling process. The algorithm first starts with some preprocessing on the point set, Figure 3.7(a). Using the software from [16, 52], the k-nearest neighbors of each point are computed. For each set of k-nearest neighbors, a PCA is performed and triplets of eigenvectors and eigenvalues are extracted. An octree spatial partitioning structure is created on the point set which is used for the global projection test. The first layer begins with finding a suitable point to start the reconstruction, calling it a *seed*. We sort all points in increasing order of their eigenvalue ratios to select a seed. We define *eigenvalue ratio* e_p for each point p as the ratio of its smallest eigenvalue to the sum of all its three eigenvalues. However, we ignore points whose two out of three eigenvalues are having very low values, which indicate that their k-nearest neighborhood forms a straight line instead of a planar region. To determine

whether a point p can be a seed, we use the ray $\mathbf{r}$ which is the third (smallest) eigenvector associated with p, and check whether it passes the global projection test. If it does, p qualifies as a seed and $\mathbf{r}$ is assigned as its normal. Otherwise we repeat the test for $-\mathbf{r}$ to determine whether p can still be a seed with $-\mathbf{r}$ as its normal.

Once the seed is found, we construct a triangle fan at the chosen seed, Figure 3.7(b). This triangle fan becomes the initial mesh M for us to iteratively select another point which is lying on the boundary of M to form a triangle fan to merge into M. We term *boundary points* as points in M whose triangle fans have yet to be constructed. There are generally many boundary points and thus many possible triangle fans to consider for merging into M. As such, we prioritize all triangle fans using a heap with preference given to one with the smallest variance of dihedral angles. Each dihedral angle in a triangle fan is defined between a pair of triangles sharing an edge. A triangle fan can only be added to the heap if it passes the global projection test with its normal as the test ray. Each time a triangle fan is merged to M, the boundary of M changes with new points, and new triangle fans on these points are constructed for consideration to merge into M, Figure 3.7(c). The construction of this layer ends when no triangle fans can be constructed for the boundary points of M and at the same time no new seeds can be found, Figure 3.7(d).

The algorithm then moves on to the next layer of peeling by subdividing the input points not included in previous layers into subsets where two points are in the same subset when one is a k-nearest neighbor of the other. Breaking the point set down into smaller subsets is essentially a divide and conquer approach. Note that in Figure 3.7(e), there is only one such subset. Since each subset is spatially

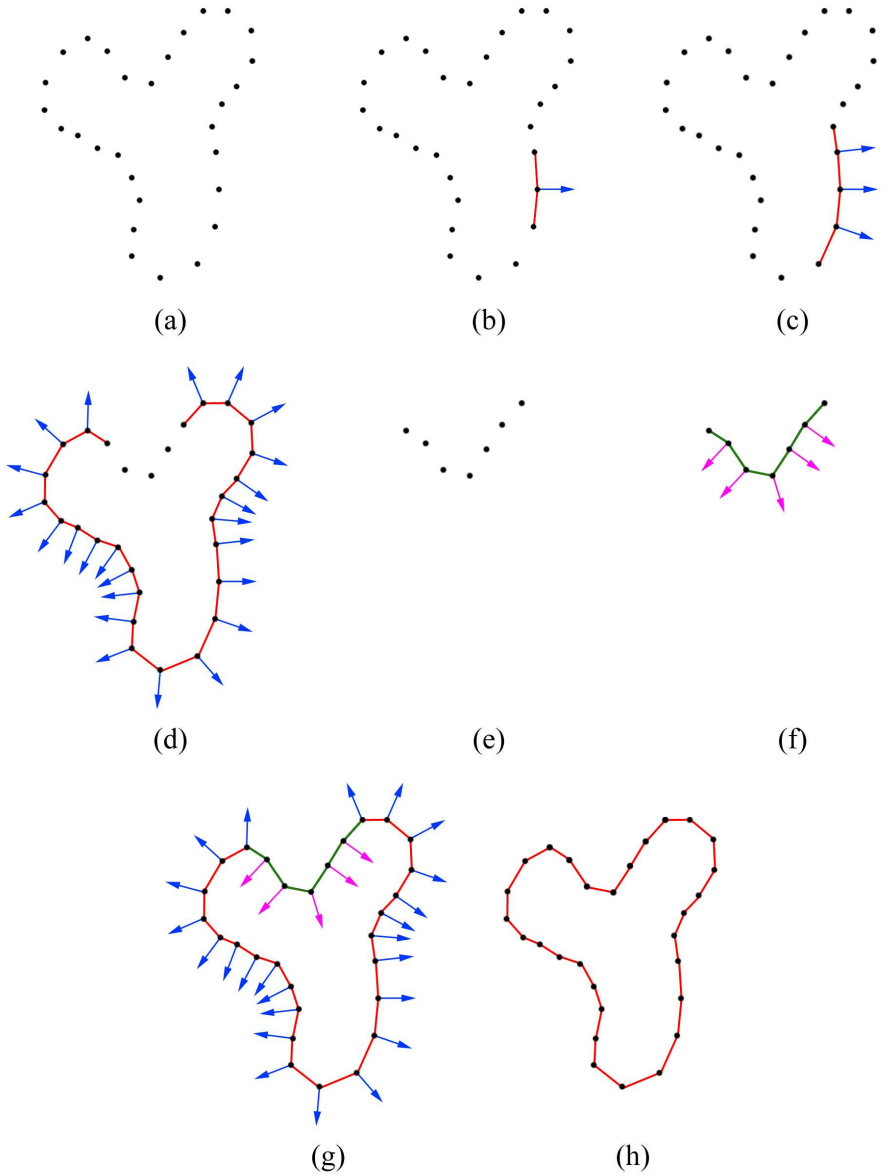


Figure 3.7: The various stages of the Layer Peeling process.

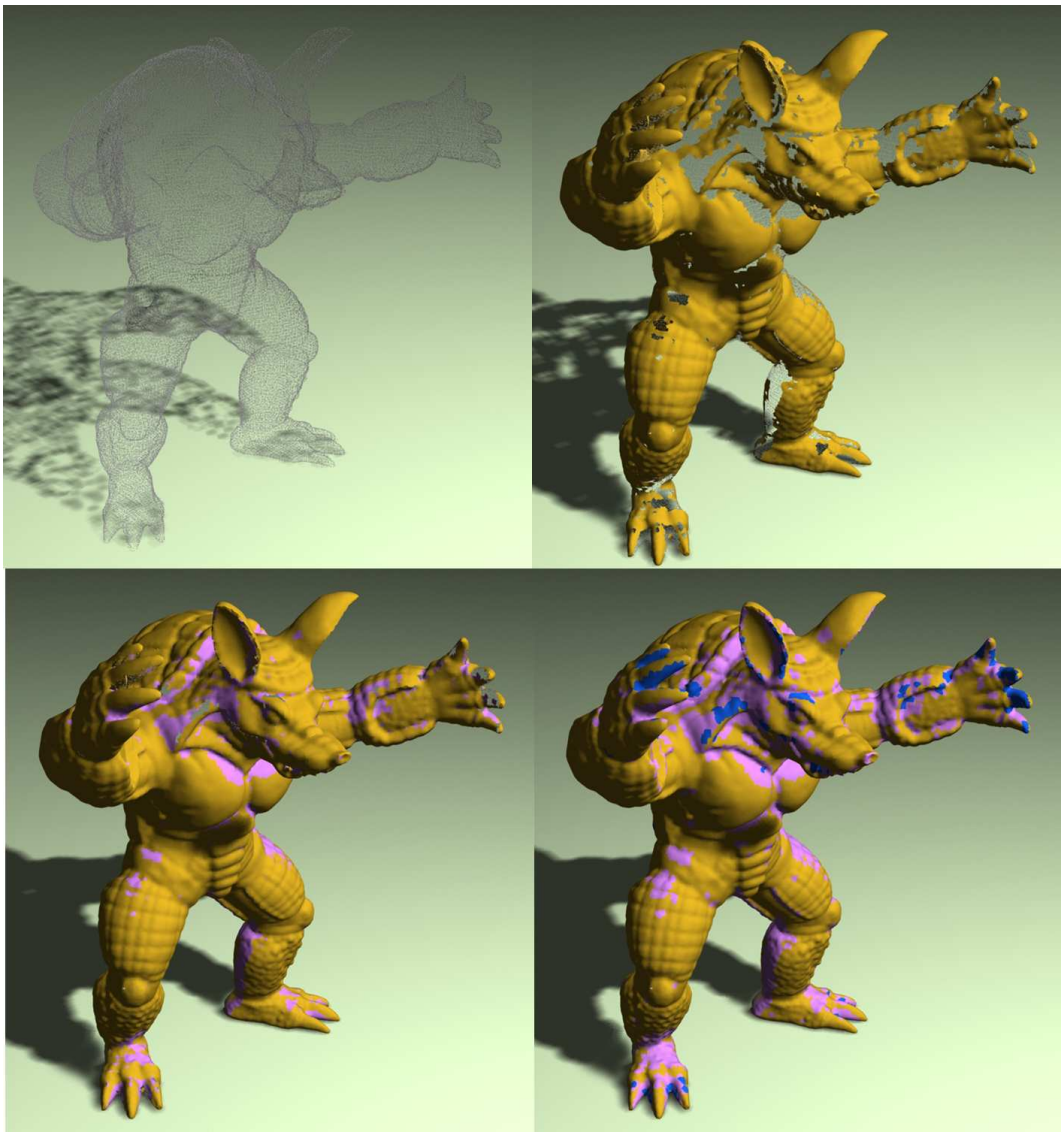


Figure 3.8: The various stages of the layer peeling algorithm (from left to right, top to bottom) on the Armadillo point set. The different colors of the reconstructed surface represent the different layers created by the layer peeling algorithm.

apart from each other, thereby making the computation of the global projection test much faster. We then create a new octree structure for each subset to extract its next layer with respect to the reverse side of the surface (i.e., the orientations of normals are now inverted), Figure 3.7(f). We continue the extraction process from all the boundary points again, but with a reversed orientation (Fact 1). Once this is completed, all the normals that are found in the process are flipped (negated) back, Figure 3.7(g). For the subsequent layers (if needed), we flip the normals once every alternate layer. The final reconstruction result is shown in Figure 3.7(h). An actual rendering of the layer peeling process performed on the Armadillo point set is shown in Figure 3.8.

This layer peeling approach to surface reconstruction can be used on objects with very complex topology also. An example of a reconstructed surface from such a point set with complex topology is shown in Figure 3.9. The various stages of the reconstruction can be seen with the formation of surfaces from the outermost regions of the Heptoroid point set and moving towards the inner regions.

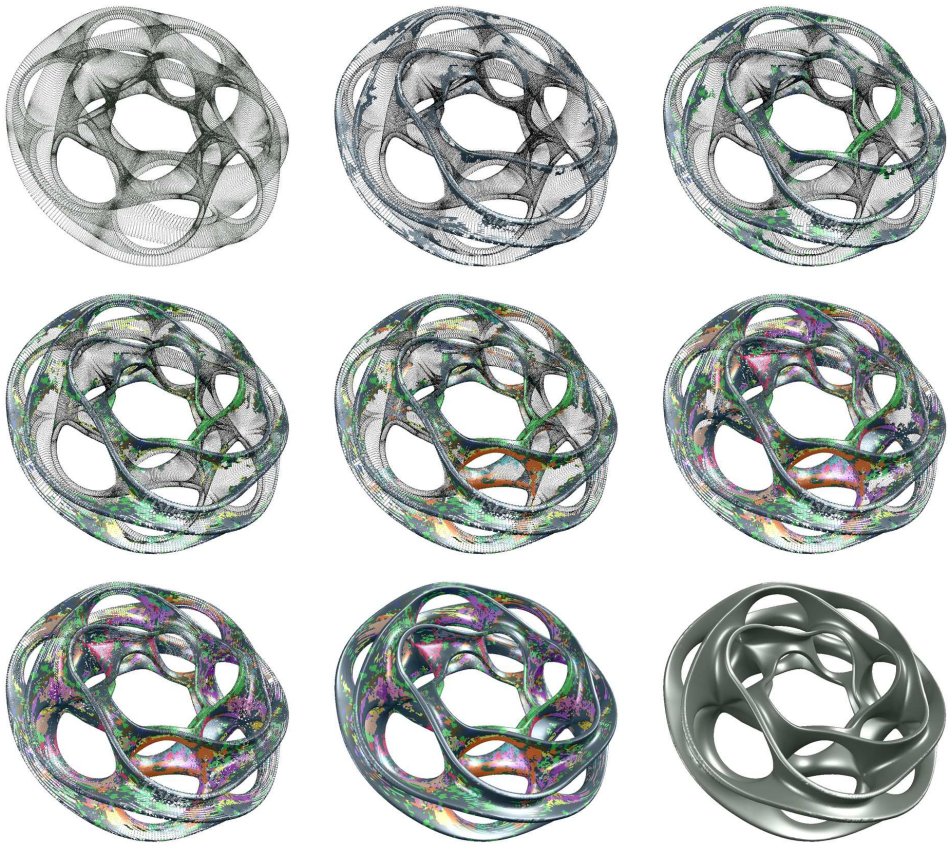


Figure 3.9: The various stages (from left to right, top to bottom) of surface reconstruction using the Layer Peeling algorithm on the Heptoroid point set.

Chapter 4

Implementation

In this chapter, we describe the implementation details of the layer peeling algorithm. There are two main processes to the layer peeling algorithm, the first is the global projection testing and the second is constructing a local surface for a point. For the global projection test, we use an octree data structure to test for an intersection between a ray and the point set. To construct local surfaces, we make use of a simple geometric structure known as triangle fans.

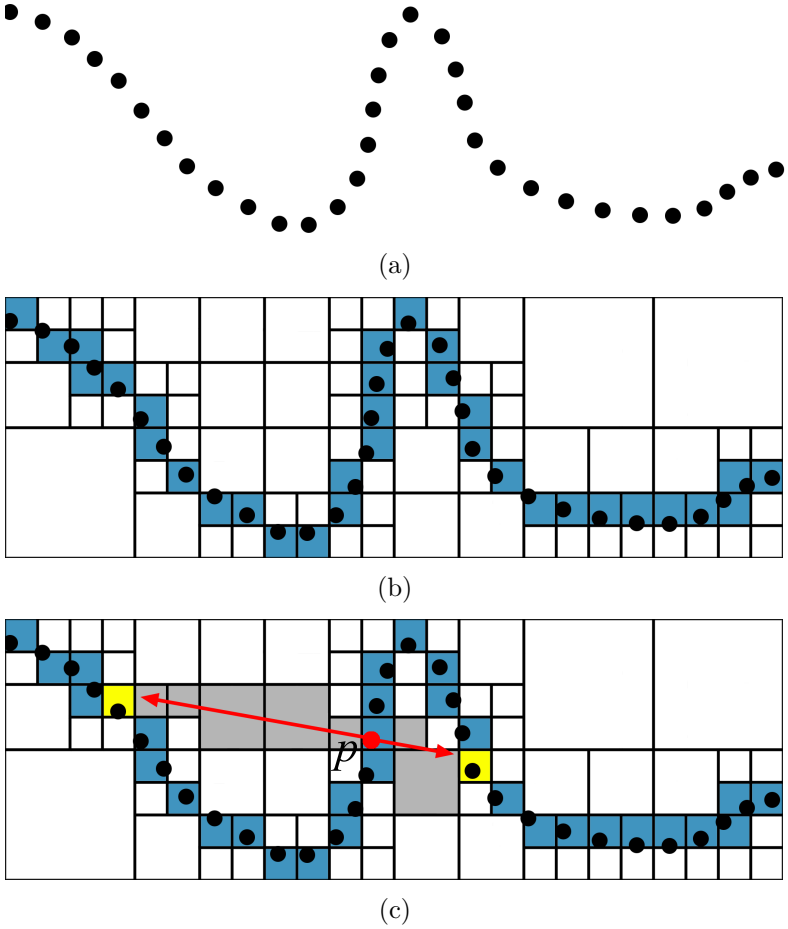
4.1 Global Projection Test

A common operation needed in our algorithm is the global projection test (refer to Fact 2 in Section 3.3). It tests for intersection of a ray with the surface of the point cloud, while on the other hand the surface has yet to be constructed. Since we have no pre-knowledge of the original sampled surface, we determine when any point is within a certain proximity to a ray to indicate that the ray has intersected with the surface. In our case, an intersection with the surface (or close proximity to any point) is deemed to have failed the global projection test. To efficiently

perform this global projection test, we build an octree on the smallest bounding cube of the point set, and we say a ray intersects the surface of the point cloud when the ray passes through one or more leaf nodes of the octree containing input sample points. In Figure 4.1, an illustration of the global projection test in 2D is given. In Figure 4.1(a), a small subset portion of the point set is shown and Figure 4.1(b) shows the construction of the octree in 2D. The occupied leaf nodes in the octree are shown as shaded. In Figure 4.1(c), a point p is tested for the global projection test. The path traversed by the ray is shown as colored in grey. The point p failed the test as it hits an occupied leaf node in both directions of the ray.

The choice of using an octree for our purposes is an appropriate one, as opposed to a regular grid structure. Since surface reconstruction deals with point set sampled from a surface manifold, most of the spaces within the smallest bounding cube of the point cloud are usually empty. In the construction of the octree, two notes are in order. First, we need to decide when to stop subdividing a cube and designating it as a leaf node. Our input point sets can possibly be regularly or irregularly sampled. For the former, we can fix the length of the leaf node, i.e. recursively dividing the cube into smaller cubes whenever there are points in it until the cube's length reaches a pre-determined limit. However, this approach does not work for the latter. Therefore, to handle both cases, we first define the *estimated sampling distance* of a point to be the distance from itself to its k^{th} nearest neighbor. Then, we only subdivide a cube when the estimated sampling distances of all the points in the cube is shorter than half the length of the cube. This is to efficiently and appropriately size the leaf node, depending on the sampling density around its vicinity. Second, the global projection test is performed

through marching from a leaf node to an adjacent one (using [54]), starting from the origin of the ray and along the ray.



(a) (b) (c)

Figure 4.1: The global projection test in 2D.

4.2 Triangle Fan

The triangle fan of a point $p \in P$ is a convenient notion for approximating a small region of the surface around p. We use it to support the extraction of local surfaces. We note that there are also similar notions of triangle fans in previous works on surface reconstruction [49]. Our work differs in the criteria of a suitable triangle fan,

and its use within a novel layer peeling approach to determine surface neighbors. This section details the construction (Section 4.2.1) and merging (Section 4.2.2) of triangle fans, and finally the generation of a closed manifold (Section 4.2.3).

4.2.1 Triangle Fan Construction

Let N_p denote the set of k-nearest neighbors of p. A *triangle fan* T_p of p is formed by a ring of triangles $t_0, t_1, \ldots, t_i$ where $i < k$. These triangles are formed using points $p_0, p_1, \ldots, p_i$ where $p_0, \ldots, p_i \in N_p$. For $0 \leq j < i$, t_j uses vertices p_j, p and p_{j+1}, and t_i uses vertices p_i, p and $p_0 = p_{i+1}$. The vertices $p_0, p_1, \ldots, p_i$ form the set Q_p, which we term as the *surface neighbors* of p. Since we always construct new triangle fans on points that lie in the current boundary point set, there always exist a vertex $q \in Q_p$ which has already constructed its triangle fan Q_q (unless p is a seed). Using q, we can determine the facing of each triangle in the triangle fan of p, and subsequently the approximated normal at p. Let $\angle\alpha_j$ denote the angle at the vertex p in triangle t_j, and $\mathbf{t}_j$ the normal of triangle t_j. We approximate the normal at p by:

$$\mathbf{n} = \sum_{j=0}^{i} (\mathbf{t}_j \cdot \angle\alpha_j)$$

With the definition of the triangle fan, we are ready to define the characteristics of our triangle fan. We require that T_p satisfies the following criteria:

- **Local Convexity Criterion:** Each triangle t_j is such that no other point within the set $N_p - Q_p$ can be projected from above (based on normal direction and orientation of t_j) into t_j. This means t_j lies on the outermost layer of its neighborhood.

- **Normal Coherence Condition:** For all $t_j \in T_p$, we have $\mathbf{n} \cdot \mathbf{t}_j > 0$. This

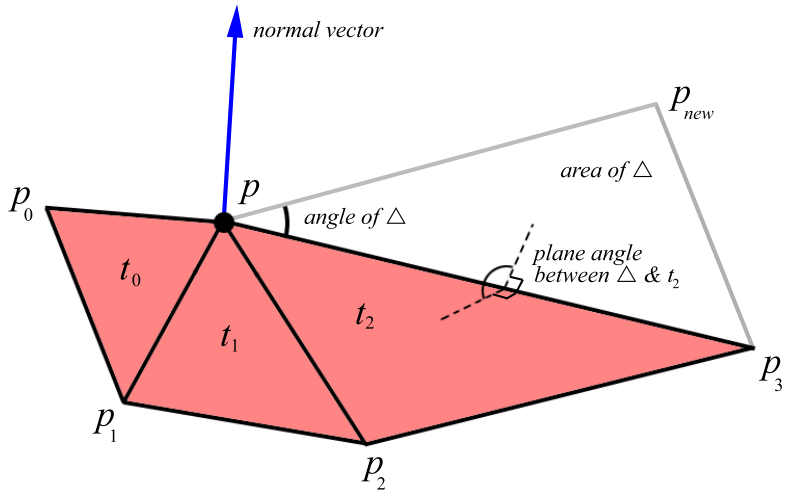


Figure 4.2: Construction of the triangle fan.

is because we want a triangle fan to represent a local surface that is similar to a topological disk.

- **Global Projection Test:** A ray from p (in the direction of $\mathbf{n}$) passes the global projection test. This is in the spirit of processing the input point set from outer layer towards inner ones.

The local convexity condition allows the triangle fan to exclude points that may not be lying on its nearby surface. Furthermore, together with the normal coherence condition, it allows the triangulation within a nearby region of a point to be projectable onto a 2D plane (more details is given in Section 5.2.1). The last condition, the global projection test, governs the sequence of the propagation of the mesh construction. It generally guides the layer peeling algorithm to progress from the outer layer of the point set towards the inner layers. It is easier to construct triangle fans for points that lie on the outermost later of the point sets, since they only have two types of neighborhood points and one can be easily filtered away by the local convexity condition.

The *dihedral angle* ϕ_{AB} between two triangles, A and B, sharing an edge is the angle between their two normal unit vector $\mathbf{n}_A$ and $\mathbf{n}_B$. A dihedral angle can be signed; the dihedral angle ϕ_{AB} is defined as the angle through which triangle A must be rotated (about their common edge) to align it with triangle B. Thus, $\phi_{AB} = -\phi_{BA}$. In Figure 4.2, the dihedral angle made with the newly added triangle can be calculated by taking the difference between the plane angle with 180°. Dihedral angles are used both during the construction of the triangle fan and during the selection of the next triangle fan to be used for merging into the mesh M. One of the criteria of selecting the next triangle fan to be used for merging is by calculating the variances of all the dihedral angles in the triangle fan:

$$var(\phi_{T_P}) = \frac{1}{i+1}\sum_{j=0}^{i}(\phi_j - \overline{\phi})^2$$

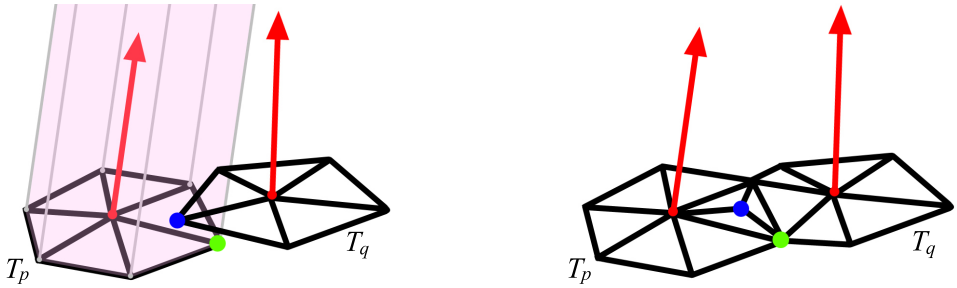
where ϕ_j represent the dihedral angle between t_j and t_{j+1} if $j < i$, and between t_i and t_0 if $j = i$.

In the construction of a triangle fan for point p, we do not seek to construct a unique or optimum triangle fan that best represents the local surface around p. For our purposes, any triangle fan selecting only points from N_p and fulfilling the above three criteria is sufficient. As stated earlier, we start the construction from point q. Using q, we employ a greedy algorithm to search for the next triangle (selecting another point from N_p) by giving each triangle a priority value with preference to smaller area and small dihedral angle (made with the previous triangle). (Note that we only select a triangle if it passes the local convexity criterion.) If no suitable triangle can be found, the algorithm backtracks and searches for the triangle with the next highest priority value. The construction terminates when a triangle fan is formed, or when it backtracks to point q. The triangle fan thus constructed, if

any, that passes the three criteria is then a candidate for triangle fan merging as described in the next subsection.

4.2.2 Triangle Fan Merging

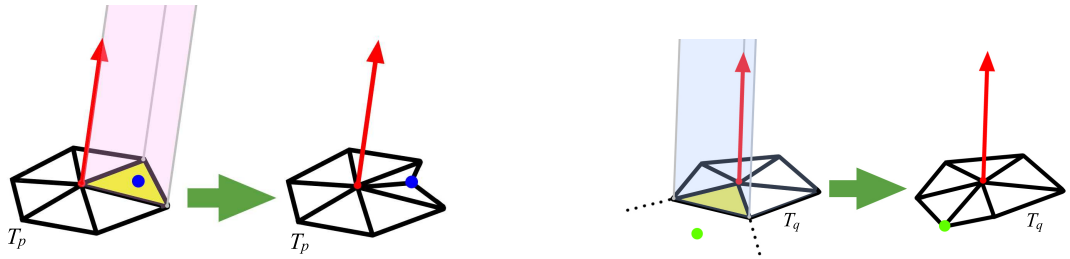
We approximate the local surface region around p using T_p, with the intention of forming a single piecewise linear surface covering over the entire point set. Starting with the first triangle fan that is created at the seed, the algorithm merges each successive new triangle fan into M. We describe the merging process in the next paragraph. Before that, we note that a triangle fan has the normal direction as given in Section 4.2.1, and the orientation by the global projection test (depending on whether the layer requires flipping). Also, a triangle is considered to have two faces: the front face whose normal makes a positive dot product with the triangle fan's normal, and the back face otherwise.



(a) (b)

Figure 4.3: Merging of two triangle fans.

We merge two triangle fans together based on a rule that is similar to the global projection test. Consider two triangle fans, T_p and T_q. Note that one particular vertex of T_p is colored as green, and one particular vertex of T_q is colored as blue as shown in Figure 4.3. For the triangle fan T_p, we project a ray from all its triangles'



(a) (b)

Figure 4.4: Merging process of a single point to a triangle fan.

front faces along $\mathbf{n}$. For any two triangle fans T_p and T_q, we merge them together if either the ray from any triangle in T_p hits any back face of any triangle in T_q or vice versa, where $q \in N_p$. Figure 4.3(a) shows the case where the ray from T_p hits T_q around the blue vertex area, hence a merging process is required. The two triangle fans T_p and T_q are then merged together as shown in Figure 4.3(b). Note that the merging process as described in this section is for general case between any two triangle fans. Merging is formed by a simple triangulation process through the addition of points into a triangle fan as shown in Figure 4.4. When a new point is added into a triangle fan T_p, it is added into the triangle that it is projected onto, along the direction of the normal of T_p, shown as the addition of the blue vertex in Figure 4.4(a). However, new points do not always have to be projected onto the triangle fan itself in order to be merged, a point can also be added to a triangle if it is projected within the range of that triangle. This situation is shown in Figure 4.4(b), where the green vertex from T_p is projected within the range of a triangle in T_q (as shown within the wedge defined by the two dotted lines) and then merged into T_q. The rationale for merging new points into triangle fans with such an approach is essentially to maintain the normal coherence property of triangle fans after merging.

When the above simple triangulation is performed, we next seek to optimize

the resulting mesh to fit it more closely to the original surface. We achieve this by performing edge flips in 3D to minimize the absolute value of the dihedral angles in the mesh. Our rationale is stemmed from the fact that as sampling density increases, every edge in the restricted Delaunay triangulation tends to have a dihedral angle close to 0° [50].

4.2.3 Closed Manifold

Since for each point p, T_p uses only points from N_p, thus it is likely that holes in M may exist after the layer peeling algorithm is completed. This is generally a consequence of utilizing the k-nearest neighborhood method, since no large triangles are able to form over patches of under-sampled regions. Hence for a practical solution to produce a closed manifold, we use a simple hole filling algorithm. Throughout the triangle fan construction and merging process, we maintain a list of boundary points. For each point in this list, there are two boundary edges that are incident to it and a priority value is attached to the point based on the angle formed by the two edges. Small angles are given high priority values. The algorithm then proceeds to insert a triangle into the mesh M which is formed by the two boundary edges incident to the point with the highest priority value. The list of boundary points thus changes and priorities are updated accordingly. This process repeats until no boundary point exists. In the event that self intersection occurs due to the insertion of a new triangle, the affected triangles are removed from M and in the process creating new boundary points. This new list of boundary points is closed up in a similar fashion.

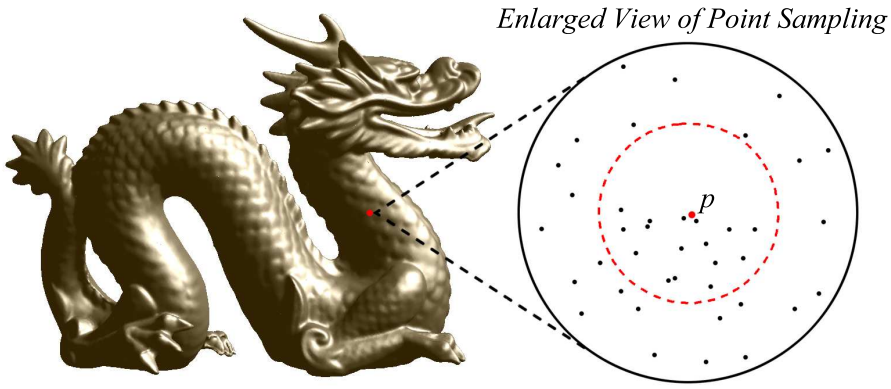


Figure 4.5: The irregular sampling of points around a point p in the Dragon point set is shown.

4.3 Handling Irregularly Sampled Point Sets

The success of the construction of a triangle fan T_p for p relies on the uniform distribution of N_p. For irregularly sampled point sets, two problems can exist; refer to Figure 4.5. The first problem occurs when some neighbors (relative to the other k-nearest neighbors) are too close to p. Having neighboring points that are too close present several issues during the construction of its triangle fan, such as forming non-uniformly sized triangles which can lead to projection problems during the merging process. More importantly, points that are too close together provide very little new information about the sampled surface since it implies that both points are sampled around the same position. The second problem is due to an uneven sampling around a point, resulting in the situation that its k-nearest neighbors are all located on one side of the point. Constructing triangle fans for such points result in awkwardly shaped triangles.

To handle the first problem, we run a decimation process after the k-nearest neighbors are calculated for each input point. In this process, we scan through each point in some order (such as the input order) to remove its neighbors that

are within $\frac{1}{10}$ of its estimated sampling distance, which is the distance to their k^{th} nearest neighbor. Those surviving points at the end of the decimation process then have their k-nearest neighbors re-calculated, and used to form the mesh M with the layer peeling algorithm. Thereafter, those points previously removed are merged into their nearest triangles in M. For the second problem, for each point p, we augment its k-nearest neighborhood to include points which have p in their k-nearest neighborhood. This provides more choices for the construction of the triangles fan at p.

Chapter 5

Analysis

In this chapter, we provide an analysis of our layer peeling algorithm. There are three parts in this chapter. Section 5.1 provides an intuitive explanation on why the proposed layer peeling algorithm can handle under-sampled point sets well. Section 5.2 gives a detailed proof regarding the termination of the algorithm and the correctness of the reconstruction. Similar to other algorithms in provable surface reconstruction, the proof only states the correctness of the reconstruction when optimum sampling is present. In other words, under what conditions in which the algorithm is guaranteed to reconstruct correctly. The proof, however, does not cover under what conditions the reconstruction is likely to fail. Lastly, Section 5.3 discusses the computational time complexity of our algorithm.

5.1 Under-Sampled Point Sets

As stated in Section 3.1, the main problem facing under-sampled point set is that the k-nearest neighborhood of a point usually contains points which does not lies on its nearby surface. In general, there are three types of points that can exist in

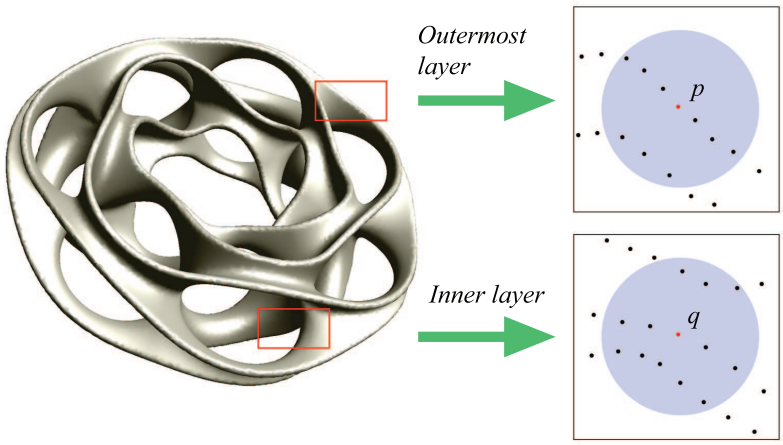


Figure 5.1: The k-nearest neighborhood of a point p lying on the outermost layer and a point q lying in the inner layer of the Heptoroid point set.

a typical k-nearest neighborhood, points that lies on top and below the surface, and points lying on the nearby surface itself, as shown in Figure 5.1 (bottom right) image. The layer peeling algorithm is directed towards the identification and removal of points that lie above and below the nearby surface when creating the local surface.

From the construction of the triangle fan itself, the local convexity criterion allowed the algorithm to avoid effectively the problem of differentiating between points in N_p lying below the surface and on the surface containing p when forming a triangle fan for p. However, this approach of using the local convexity criterion to form triangle fans to exclude point from "under" the surface can only work if there are no points lying "above" the local region. Note that the terms "above" and "under" are used loosely in the context, since due to the alternative flipping nature of the layer peeling algorithm. Thus, the two terms can be taken to mean either side of the surface. The solution to this problem comes from the propagation sequence (the order of constructing new triangle fans) of the layer peeling algorithm.

The algorithm starts with a seed to construct a triangle fan. This seed is

taken to be a point lying on the convex hull of the point set. Based on the local convexity criterion and the global projection test, the triangle fan at the seed is free of problems with points above and below the surface containing p. Figure 5.1 (top right image) shows a point p which lies on the convex hull of the point set. It only has points that either lie on its nearby surface, or below it. Next, for each triangle fan constructed, we test whether it passes the global projection test before adding it to the heap for selection during the merging process. In this way, the layer peeling algorithm can be visualized to be progressing from the outer portion of the point set, and then slowing moving inwards. As alluded by Theorem 1 (below), at any instance of the algorithm, there exists a point with no triangle fan constructed yet and is free of points lying above (or below, depending on the current iteration) its local surface. By always choosing such a point as the next candidate to construct and merge its triangle fan, we can avoid the problem of points within N_p that lie above and below the local surface around p.

5.2 Optimal-Sampled Point Sets

For this section, we assume the point set P to be an (ε, δ)-sample. As stated in Section 2.2.2, a method to obtain an (ε, δ)-sample from an ε-sample in almost linear time is provided in [34]. We require two lemmas from [8, 35]. The first lemma bounds the maximum length of an edge in a restricted Delaunay triangulation. The second lemma bounds the angle of the normals between two points that are sufficiently close. Note that the function f refers to the distance to the medial axis.

Lemma 1 [35] *For $p, q \in P$, if pq is an edge of the restricted Delaunay triangula-*

tion, then

$$\|p - q\| \leq \frac{2\varepsilon}{1-\varepsilon} \min\{f(p), f(q)\}.$$

Lemma 2 [8] *For any two points p and q on S with $\| p-q \| \leq \rho \min\{f(p), f(q)\}$, for any $\rho < 1/3$, the angle between the normal to S at p and at q is at most $\rho/(1-3\rho)$.*

Based on the above lemmas, we have the following corollary:

Corollary 3 *For $p, q \in P$, if pq is an edge of the restricted Delaunay triangulation, then the angle between the normal at p and at q is at most 40° for $\varepsilon \leq 0.1$.*

Proof. Combining Lemma 1 and Lemma 2, we let ρ be $\frac{2\varepsilon}{1-\varepsilon}$ to obtain $\frac{\rho}{1-3\rho} = \frac{2\varepsilon/(1-\varepsilon)}{1-6\varepsilon/(1-\varepsilon)} = \frac{2\varepsilon}{1-7\varepsilon}$. The maximum angle difference of 38.1° is achieved with $\varepsilon = 0.1$. □

For the following proofs, we define the *local region* around a point p as the space where all points within that region is at most a distance of $\frac{2\varepsilon}{1-\varepsilon}$ away from p. Hence, the value of k should be such that the k-nearest neighborhood encompass all the samples points within the local region. By doing so, no potential edge connections that are present within a restricted Delaunay triangulation will be lost when taking the k-nearest neighborhood local approach. For an (ε, δ)-sample, [14] provides a formula to calculate the value of k, such that N_p contains all the neighboring sample points within the local region around p. Hence, the value of δ directly affects the value of k that is required. Generally, the larger the value of δ, the lower the value of k. In their paper [14], Andersson et al. use a packing theory to calculate the maximum number of neighboring points that is bound to contain all its possible restricted Delaunay neighbors.

5.2.1 Theorem on Complete Reconstruction

The first theorem shows that the layer peeling algorithm does not prematurely terminate before a manifold is constructed. Essentially, it implies that the algorithm can always find another point to create a new triangle fan. For the proof, it is sufficient to show the existence of a new seed to construct a triangle fan each time. In order to so do, the algorithm has to be able to locate a new seed where the three criteria of triangle fan construction can be satisfied within its k-nearest neighborhood. In this proof we simplify the propagation sequence by using a new point each time, rather than in the actual implementation where the algorithm usually utilizes points from the boundary of M for the purpose. Note that in the way we derive subsets of P (in Step 3 of Figure 3.6), the following Theorem 1 also holds for each subset.

Theorem 1 *At any instance during the execution of the layer peeling algorithm on a point set P, it always exists a point p to construct a triangle fan T_p to become a part of M.*

Proof. We pick $p \in P'$ to be a vertex of the convex hull of $P' \subseteq P$ where each point in P' has no triangle fan constructed yet. Next, we construct a triangle fan T_p for p. Clearly, p with T_p passes the global projection test, since p is lying on the convex hull of P'. We next show that T_p satisfies the local convexity criterion and the normal coherence condition.

Refer to Figure 5.2. For point p, the local region around p (shown in dashed red circle) is bounded by two medial balls on each side of the surface, both of radius $f(p)$. Now consider one of the balls B. We tilt the ball in any one random direction while pivoting at point p until a point q is hit. Similar to [19], we now pivot the ball on the edge pq. By rotating the ball on the edge pq, ball B comes into contact with

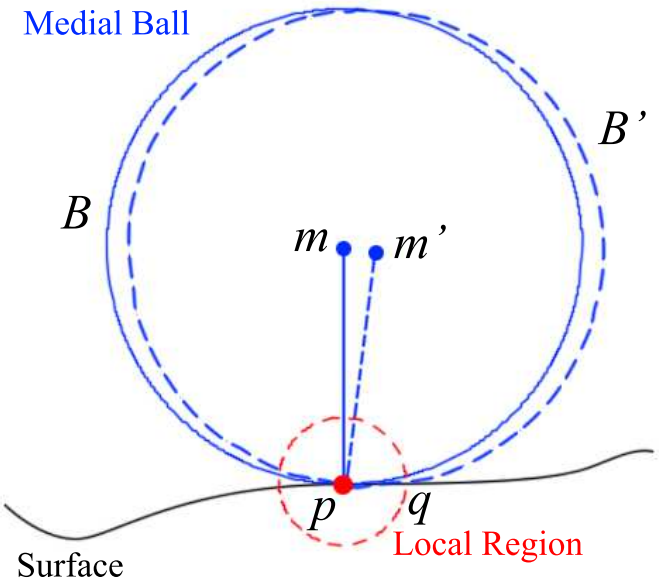


Figure 5.2: A medial ball centered at m is pivoted at point p. The maximum deviation of the line pm is pm'.

another point r (not shown in the 2D Figure 5.2), forming a triangle pqr. Since the surface S is ε-sampled, a ball of radius $\varepsilon f(p)$ cannot penetrate S. Thus the maximum radius of the circum-circle of pqr can be at most $\varepsilon f(p)$. The maximum tilt of ball B happens when its surface intersects with the tilted medial ball B' to form a circle of radius at most $\varepsilon f(p)$. (In the case when ε is 0.1, the maximum tilt is only 12° by a simple calculation.) The maximum tilt of ball B is shown as B' in Figure 5.2. Furthermore, we note that p, q, and r exist in N_p as q and r are of at most $2\varepsilon f(p)$ distance away from p, since $2\varepsilon f(p) < \frac{2\varepsilon}{1-\varepsilon} f(p)$. (Note that all sample points within $\frac{2\varepsilon}{1-\varepsilon} f(p)$ are contained within the k-nearest neighborhood.) To extract the full T_p, we continue to pivot the ball B on the edge pr and rotate away from q to extract the next triangle. We continue in this fashion until T_p is formed.

To prove that T_p obeys the local convexity criterion, we consider each triangle of T_p in turn. For each triangle, ball B is able to pivot on its three vertices. Since ball B is empty of points, the local convexity rule is easily seen to obey.

To show that normal coherence is obeyed by T_p, we consider the line pm, where m is the center of ball B. During the extraction of T_p, the line pm' traverses within a cone-like space. After T_p is formed, $\mathbf{n}$ lies within this cone-like space. Since the tilt of pm' never exceeds 90° (recall the maximum tilt for $\varepsilon = 0.1$ is only 12°), normal coherence condition is obeyed. □

5.2.2 Theorem on Correctness of Reconstruction

The following lemma proves that the intersection of S with the local region around any particular point p is a topological disk. By proving that the local region is a topological disk, it provides a basis where the linear piecewise surface produced by the layer peeling algorithm can be compared against the restricted Delaunay triangulation.

Lemma 4 *Consider a point $p \in P$ with $\boldsymbol{n}$ being the normal to S at p, and a region $S' \subseteq S$ where S' is the intersection of S with the local region around p. Then there exists an injective function to map S' to a 2D plane with a normal of $\boldsymbol{n}$ for $\varepsilon \leq 0.1$.*

Proof. For any $q \in S'$, we know that the maximum angle difference between the normals to S at p and q is 40° by Corollary 3. Consider a line along the direction of $\mathbf{n}$. It can intersect S' at most once, since for intersection to occur twice, the normal at some part of S' needs to be at least more than 90° away from $\mathbf{n}$. Thus we can define the function μ as a linear projection from S' using $\mathbf{n}$ as the projection normal. Such a linear projection allows S' to be projected onto a 2D plane with

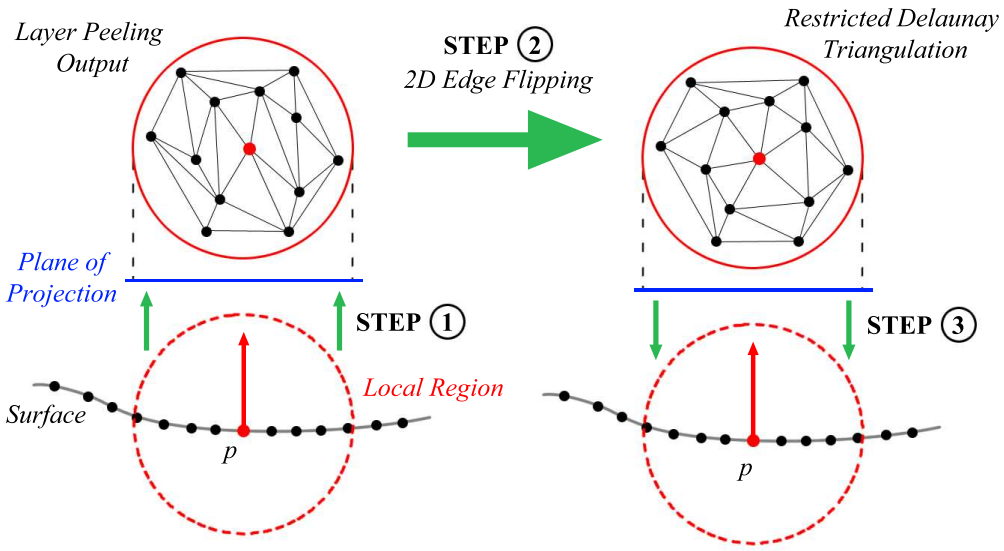


Figure 5.3: The sequence of transformation from the triangulation produced by the layer peeling algorithm to a restricted Delaunay triangulation.

a normal of $\mathbf{n}$. It can be easily seen that μ is an injective function, since no two points within S' can be projected to a single point. □

From here, we can now begin to show how the output from our layer peeling algorithm is homeomorphic to the original surface where the point set is obtained.

Theorem 2 *The piecewise linear surface constructed by our layer peeling algorithm is homeomorphic to the surface S for an (ε, δ)-sampled point set P where $\varepsilon \leq 0.1$.*

Proof. We aim to prove that, through a series of local operations, we are able to transform the piecewise linear surface constructed by our algorithm to the Delaunay triangulation of the input point set P restricted to S. The theorem thus follows as a Delaunay triangulation of P restricted to S with $\varepsilon \leq 0.1$ is homeomorphic to the original surface S as proved in [11].

Refer to Figure 5.3. First, we show that the restricted Delaunay triangulation within the local region of p can be projected to a 2D plane. By Lemma 4, the local region around p can be projected into a 2D plane smoothly. Since those restricted Voronoi cells are on the surface within the local region of p, they can also be projected similarly. Thus, it follows that the dual edges of the restricted Voronoi edges, which are the restricted Delaunay edges, can be projected as well.

Next, we consider the piecewise linear surface produced by our algorithm. It cannot be projected straightforwardly to a 2D plane as in the restricted Delaunay triangulation case. This is because, with a small chance, the merging of a triangle fan at p to the mesh M can produce a triangle incident to p whose normal can be almost orthogonal to $\mathbf{n}$, where $\mathbf{n}$ is the normal to S at p. Such a triangulation occurs because of badly shaped slivers, for example splinters or spike slivers, as classified in [26], where edge flipping may not be able to remove. Nevertheless, we can transform the triangulation around the local region of p to one that minimizes the maximum slope via the edge insertion technique [18]. Such a triangulation does not have badly shaped triangles (slivers) as the local region to be reconstructed is known to obey Lemma 4. Note that the edge insertion technique requires that the initial triangulation is projectable onto a 2D plane and the local convexity rule (enforced at the triangle fan at p) ensures that such a projection is always possible. The need for such an operation stems from the fact that although each triangle fan by itself is projectable onto a 2D plane, multiple triangles fans in the same local region might not be able to projected onto a common 2D plane (due to slivers).

In the extreme case where a projection is not possible due to the occurence of multiple slivers, the triangulation edges which cross when projected can always be removed and re-triangulated. This is always possible because the local connectivity

of the vertices are known and the sampling is ε-sampled and follows Lemma 4. With this, we can now project the triangulation around the local region of p to a 2D plane (Step 1 in Figure 5.3). Note that the set of vertices used in this projection is the same as those for the case of the Delaunay triangulation, since both sets are located within the local region of p.

With both the restricted Delaunay triangulation and our triangulation around the local region of p projected to a 2D plane, we can use edge flip operations in 2D to transform from one to the other (Step 2 of Figure 5.3). This is because in 2D for a fixed set of points, any triangulation is transformable to another one through a series of edge flips. Thus, we can transform our piecewise linear surface to the restricted Delaunay triangulation in 2D. Finally, the 2D restricted Delaunay triangulation is transformed back to 3D (Step 3 of Figure 5.3). This completes our series of operations and the proof. □

5.3 Computational Time

In general, our algorithm is mostly local. However, there are two portions of the algorithm with non-linear time complexity. The first is the computation of the k-nearest neighbors while the other is the global projection test. For both cases, the data structure consists of a spatial tree decomposition approach. Both require $O(n \log n)$ time to construct, and $O(\log n)$ time to process for each point where n is the number of input points. For the former, we only construct it once at the start of the algorithm and the actual timing taken by this process is quite insignificant when compared with the rest of the algorithm. For the latter case, the construction time is similarly insignificant, but the global projection test can be expensive as each point may perform the test many times during its triangle

fan construction. However, we note that for each subsequent layer, the size of the octree gets progressively smaller as the point set is split into subsets. Hence, the influence of the non-linear time complexity portions of the algorithm is not so evident as shown in our experimental results reported in Section 6.

Chapter 6

Experimental Results

In chapter 5, we provided a proof of the correctness of the layer peeling algorithm under optimum sampling conditions. However, given such optimum sampling conditions, most algorithms generally produce reasonable output and thus making it difficult to make any comparison between different methods. Many algorithms were tested and compared against the layer peeling algorithm, mostly triangulation based algorithms rather than those based on implicit surfaces. Implicit surface based algorithms generally require solving equations which is based on a number of parameters. Furthermore, the resulting surface may or may not go through the original set of input points, hence making comparisons difficult. Among those triangulation based algorithms that were tested, Tightcocone produces the best results. The other softwares that were tested include the Geomagic Studio software (*www.geomagic.com*) and PowerCrust [12]. In this chapter, we compare the layer peeling algorithm against the TightCocone algorithm under varying degrees of under-sampling conditions. Lastly, we discuss some of the weakness and limitations of the layer peeling algorithm.

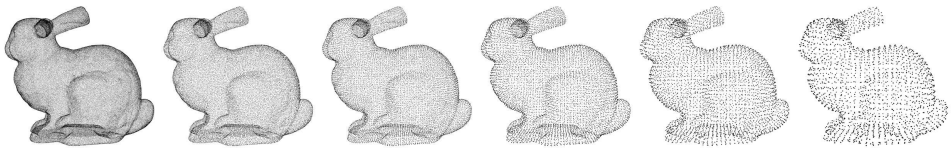


Figure 6.1: The Bunny point set is progressively down-sampled for our experiments. The leftmost figure is the original point set, while the rightmost figure is $\frac{1}{32}$ of the original.

6.1 Experimental Settings and Details

The experiments are conducted on a Pentium IV 3.0GHz, 2GB DDR2 RAM and nVidia GeForce 6600 with 256MB DDR3 video memory. In the layer peeling algorithm, we take 16 to be the value of k. Although the upper bound stated in [14] is 32, we found that for our experiments 16 is sufficient. For a comprehensive comparison, we use 23 real point sets available from the web :

- *http://www.cs.princeton.edu/gfx/proj/sugcon/models/*
- *http://www.cyberware.com/*
- *http://www-static.cc.gatech.edu/projects/large_models/*
- *graphics.stanford.edu/data/3Dscanrep/*

and one artificially created point set (E-shaped point model). The reconstructed surfaces of all the point models are shown in Figure 6.2 and Figure 6.3. The point sets have sizes ranging from 35947 points (Bunny) to 543652 points (Buddha). The size of the point sets are shown in Table 6.1. For each of the point models, we progressively sub-sampled them to get a new point set model that is half of its original size. This process is repeated 5 times to get new point sets that is $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$ and $\frac{1}{32}$ of its original size, as shown in Figure 6.1. The purpose of using smaller sub-sampled point models is to assess the robustness of the algorithms

under the presence of under-sampling, and to see the behavior of both the output quality and the amount of time taken for reconstruction. These smaller samples are obtained by uniformly under-sampling the original point sets using Geomagic Studio software. Each set of under-sampled point set, together with the original point model, are tested with the algorithms against three criteria of *Visual Quality*, *Normal Vectors*, and *Running Time*.

Point Set	Point Size	Point Set	Point Size	Point Set	Point Size
Armadillo	172974	Goddess	137406	Male	303380
Buddha	543652	Hand	327323	Man	148138
Bunny	35947	Hipbone	139205	Maxplanck	49132
Club	209779	Horse	48484	RockerArm	40177
Cow	46433	Igea	134345	Santa	75781
Dinosaur	56194	Isis	187644	Ship	388165
Dragon	437645	Lion	183408	Teeth	116604
Egyptian	51095	Lucy	262909	E-Shaped	60483

Table 6.1: The sizes of the point sets that are used in the experiments.

Visual Quality. The most obvious mode of comparison for the quality of the outputs from the various algorithms is visual comparison. For most of the original point sets, they are usually sufficiently sampled enough to produce a good reconstruction from the algorithms. By using the original as a benchmark, the gradual deterioration of the reconstruction can be observed as the point sets get more under-sampled. Such deterioration usually includes unusually long and thin triangles linking from one part of the reconstructed manifold to another part which is of quite a distance away. Other defects include the formation of holes due to thin surfaces or in some situations, a failure to properly reconstruct the surface. Hence, by visually observing the quality of the reconstruction as more and more

Figure 6.2: The first set of point set models. (From left to right, top to bottom) Armadillo, Buddha, Bunny, Club, Cow, Dinosaur, Dragon, Egyptian, Goddess, Hand, Hipbone, Horse.

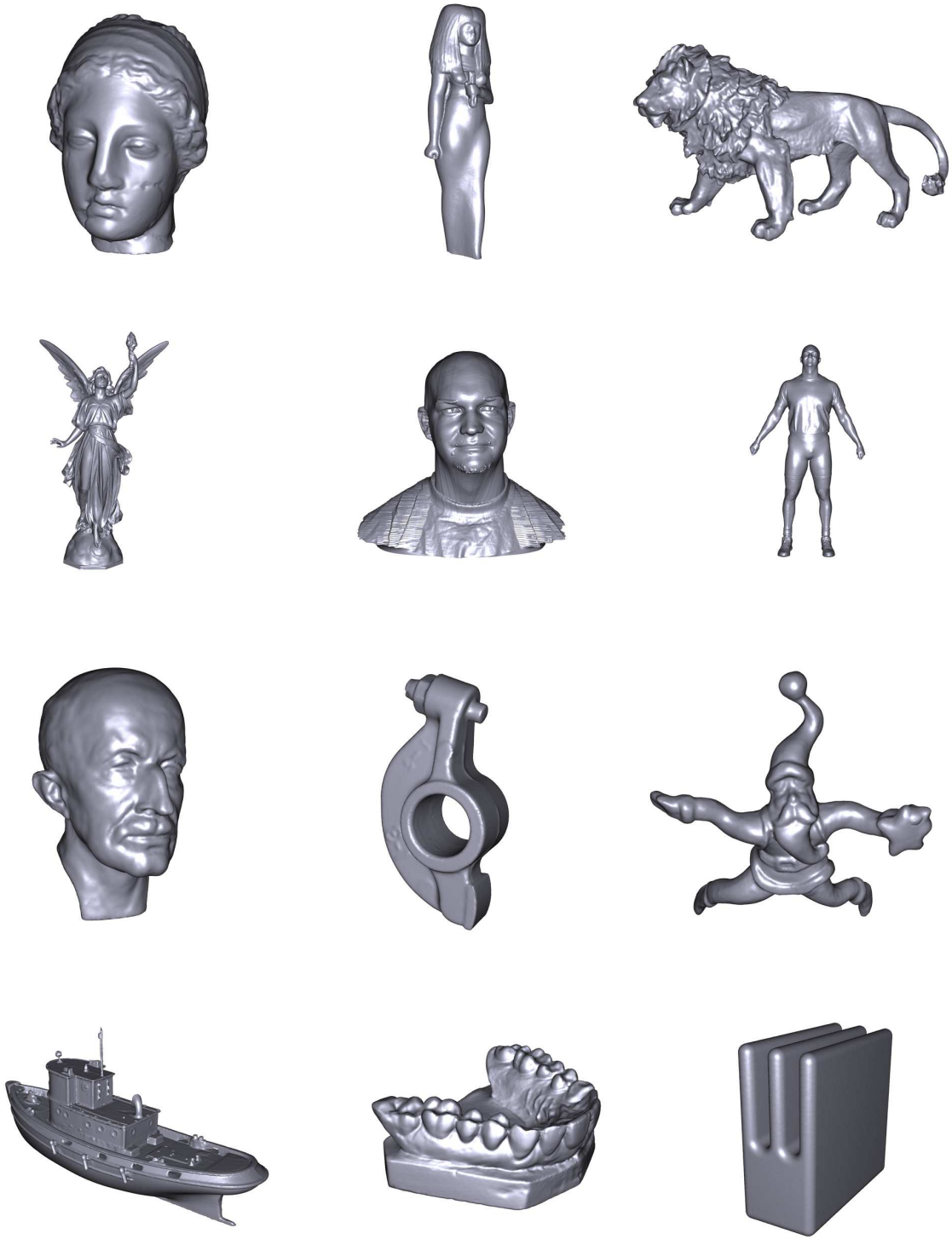


Figure 6.3: The second set of point set models. (From left to right, top to bottom) Igea, Isis, Lion, Lucy, Male, Man, Maxplanck, RockerArm, Santa, Ship, Teeth, E-Shaped object.

under-sampling occurs, a comparison can be made regarding the robustness of the algorithm against under-sampling.

Normal. Visual observation allows us to see the obvious differences that exist in two different meshes of the same point set. However, it does not provide any insight into the quality of the results as both outputs use different sets of edges and triangles but yet looked visually identical in most parts. Hence, we need a similarity metric index, for comparing how faithful the reconstruction is to the original when under-sampling takes place. Furthermore, by using such a metric, we are able to compare the quality of the output as compared to the output produced by current surface reconstruction approaches. However, the input point sample does not provide an actual normal value at each sample point. Hence, we require a standard set of normal values for each point set which the output of the algorithms can be compared against.

To achieve this aim, we use the TightCocone method as the benchmark algorithm. We take the original point sample set and run the TightCocone software on it to obtain a piecewise linear surface. Using this output surface as the benchmark, we can compare the result of the reconstruction of the under-sampled point sets against it. It should be noted that there is no correspondence of points between the original point set and the under-sampled point set. Thus, to compare the normal values between the two point sets, we first scale both point sets to have the same bounding box dimensions. Then a point in the under-sampled point set is mapped to the nearest point in the original point set and compared against its normal value. The similarity index is calculated based on the average difference (in degrees) of the normal values for all the points in the under-sampled point set and their corresponding nearest point in the original point set.

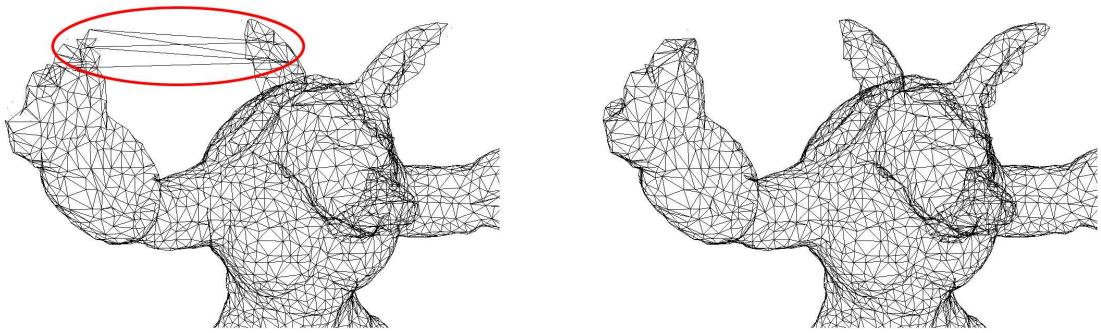
Running Time. The time taken per point for each point set model is taken and tabulated across the various under-sampled models. This allows us to see the variation of the size of the point set model against the time taken per point. A linear time complexity algorithm should generally register the same amount of time taken per point, regardless of the size of the point model. A non-linear time complexity algorithm, on the other hand, should have a longer time taken for each point as the size of the point model increases. By noting the shape of the plotted graph of time taken vs point set size, we are able to deduce whether an algorithm is suitable for scalability with larger point sets.

6.2 Visual Quality

Figures 6.4 to 6.11 highlight the differences in some of the outputs of our algorithm as compared to that of TightCocone. Our algorithm generally respects the local feature of the point clouds, and handles thin regions well. It usually does not generate erroneous triangles that span across unrelated parts of the surface. These show that our algorithm can produce meshes that match well with human perceptions of the point clouds. Through the examples shown from Figures 6.4 to 6.11, we can be seen that the layer peeling algorithm does in fact produce meshes that are more accurate than those produced by TightCocone to the original.

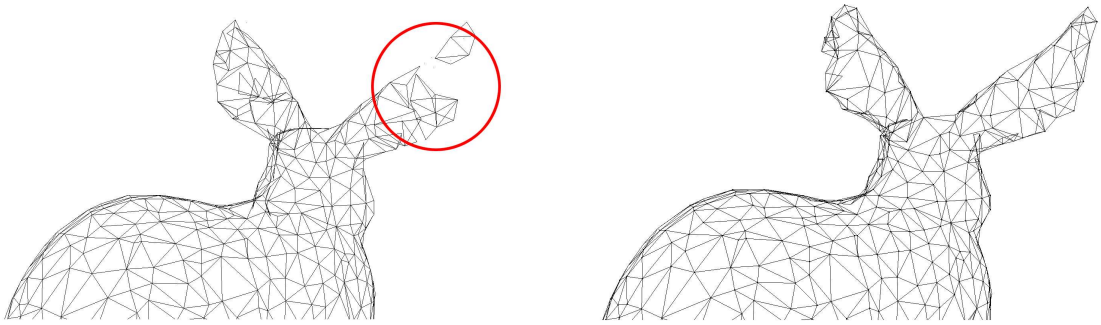
6.3 Normal Values

Each of the 23 point sets uses the TightCocone software to generate a "benchmark" surface manifold. This "benchmark" surface manifold is then compared against the surface generated by both TightCocone and the Layer Peeling algorithm for the



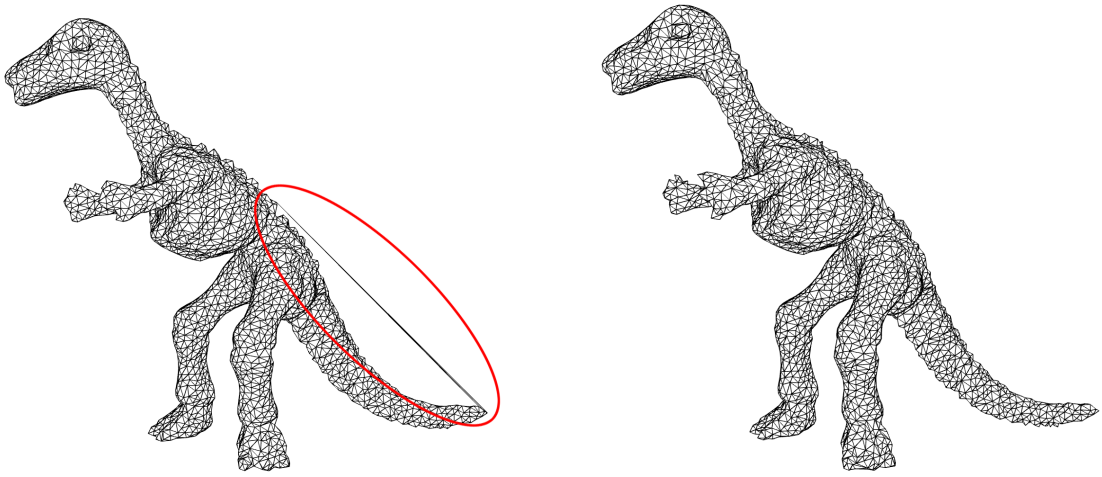
(a) (b)

Figure 6.4: Meshing results of the Armadillo point data (5787 points). (a) is produced by TightCocone where abnormal triangles are formed between the ear and the hand area. (b) is produced by our layer peeling algorithm.



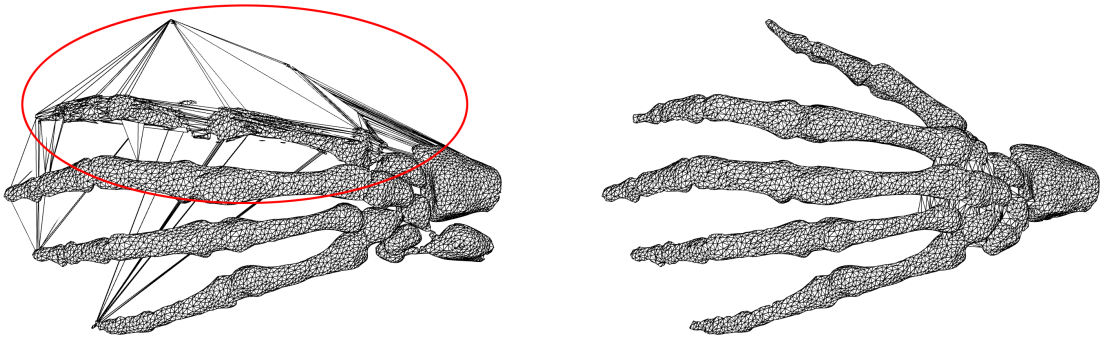
(a) (b)

Figure 6.5: Meshing results of the Bunny point data (1220 points). The result of TightCocone is shown in (a) where the ear of the Bunny is shown to be disconnected. Our layer peeling result is shown in (b).



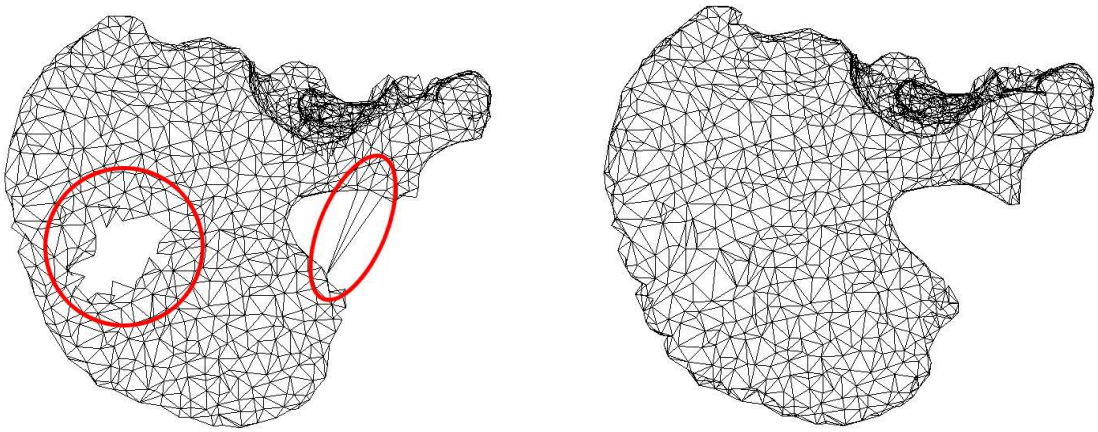
(a) (b)

Figure 6.6: Meshing results of the Dinosaur point data (3973 points). It can be clearly seen that the result of TightCocone in (a) shows the formation of triangles between the back and the tail of the Dinosaur. Our layer peeling result is shown in (b).



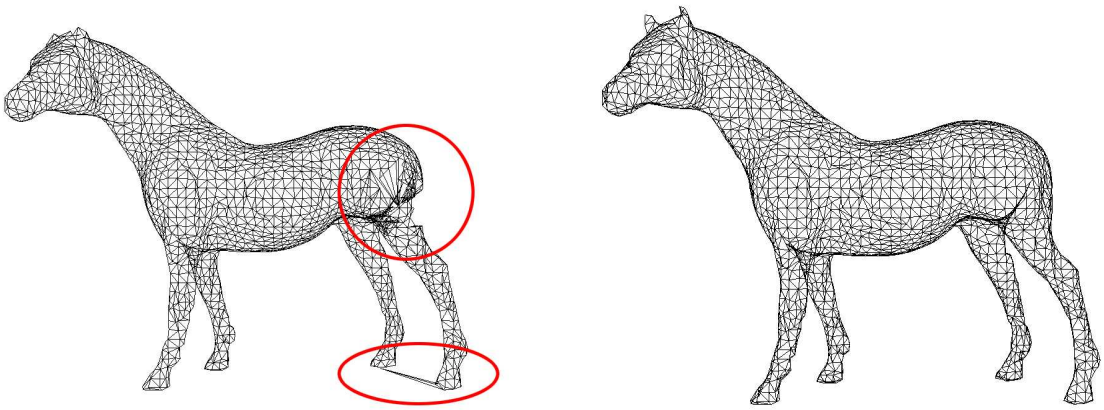
(a) (b)

Figure 6.7: Meshing results of the Hand point data (10597 points). The result shown in (a) by TightCocone clearly shows a failure of the reconstruction. The corresponding result by our layer peeling result is shown in (b).



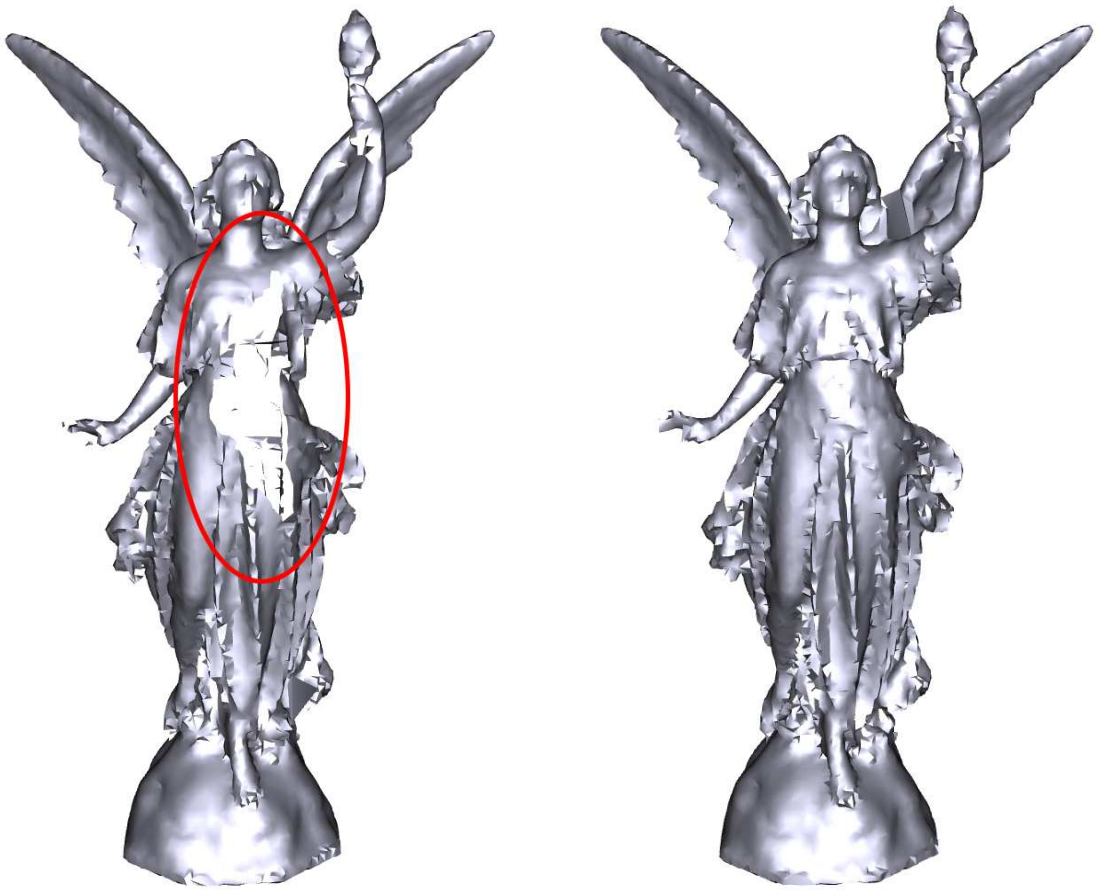
(a) (b)

Figure 6.8: Meshing results of the Hipbone point data (1964 points). Result (a) is produced by TightCocone where various deficiencies in the meshing result are highlighted. Result (b) is produced by our layer peeling algorithm.



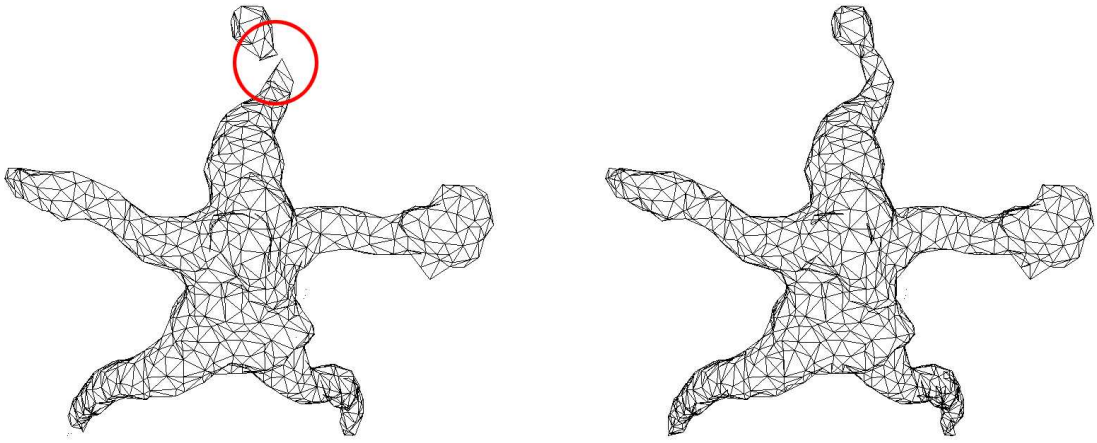
(a) (b)

Figure 6.9: Meshing results of the Horse point data (3042 points). In the result produced by TightCocone shown in (a), the hind area of the Horse contains some artifacts. The result of the layer peeling algorithm is shown in (b).



(a) (b)

Figure 6.10: Meshing results of the Lucy point data (16132 points). The rendered result of the mesh produced by TightCocone is shown in (a) and the chest area of the Lucy shows some surfaces having wrong orientations. The result of the layer peeling algorithm is shown in (b).



(a) (b)

Figure 6.11: Meshing results of the Santa point data (1098 points). (a) is produced by TightCocone and it shows some disconnections at the hat area of the Santa. The result of the layer peeling algorithm is shown in (b).

successive under-sampled point models. Note that the last point set (E-shaped object) is artificially created and hence normals are known. However, in order to obtain a better indication of the usefulness of the layer peeling algorithm, we use a simplified version of Hoppe et al. [38] algorithm to provide another point of reference. In the simplified algorithm, we uses PCA to run through the k-nearest neighborhood of each point to get an initial estimate of the normal values. Thereafter, we obtain a consistent orientation of the normal values by using the propagation sequence as stated in [38]. This simplified algorithm gives us an indication of the estimated normal values that most local algorithms generally obtain when no filtering of the k-nearest neighborhood is done. Hence, using this simplified algorithm as a point of reference, we are able to view the improvements and benefits of the layer peeling algorithm.

In Table 6.2, we see the tabulated result of the average normal difference across the various levels of under-sampling for the 23 point sets by the three methods.

Algorithm	Fraction of original point set (23 Point sets) 1	1/2	1/4	1/8	1/16	1/32
Hoppe	2.391	4.198	6.208	8.961	14.201	32.572
TightCocone	0	4.391	5.242	7.132	8.848	12.386
Layer Peeling	1.719	3.865	5.326	6.874	8.724	10.976

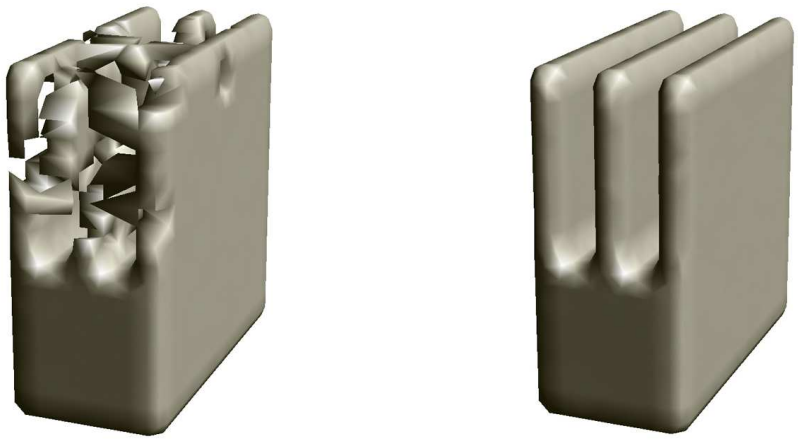
Table 6.2: The average difference (in degree) of normals computed by different methods for the 23 point set models.

Note that the tabulated result for the average normal different for the original point set by the TightCocone method is zero (since it is comparing against itself). In each of the column of Table 6.2 we can observe a similar trend. A significant improvement of the average normal difference can be seen from the layer peeling method over the simplified algorithm of Hoppe et al. This provides evidence that the layer peeling approach of filtering the k-nearest neighborhood resulted in an improvement of the accuracy of the normal values. Furthermore, the result of average normal values over the successive under-sampled point sets are very similar to the result produced by the TightCocone method.

Algorithm	Fraction of original point set (E-shaped point set) 1	1/2	1/4	1/8	1/16	1/32
Hoppe	1.608	1.83	2.386	3.954	5.515	93.929
TightCocone	1.482	1.589	1.724	2.292	3.089	13.158
Layer Peeling	1.488	1.59	1.726	2.295	3.099	5.02

Table 6.3: The average difference (in degree) of normals computed by different methods for the E-shaped point set.

Other than the 23 point sets that were tested, we also test the algorithms on the E-Shaped object as shown in Figure 6.12. By using an artificially created point set, a set of perfect normal values set can be made available and compared against,



(a) (b)

Figure 6.12: Meshing result of the artificially created under-sampled E-shaped point set model (1728 points). The result of TightCocone is as shown in (a) having huge distortion, while our result is as shown in (b).

without any error bias in the data set. The result is shown in Table 6.3. Similar to the previous average result computed by the 23 point sets, the result once again showed that the layer peeling algorithm improves on the simplified algorithm and is very close to the result of TightCocone.

	Fraction of original point set					
	1	1/2	1/4	1/8	1/16	1/32
Average Difference	1.719	0.843	0.242	0.795	1.224	1.787

Table 6.4: The average difference (in degree) of normals between the outputs produced by TightCocone and the Layer Peeling algorithm.

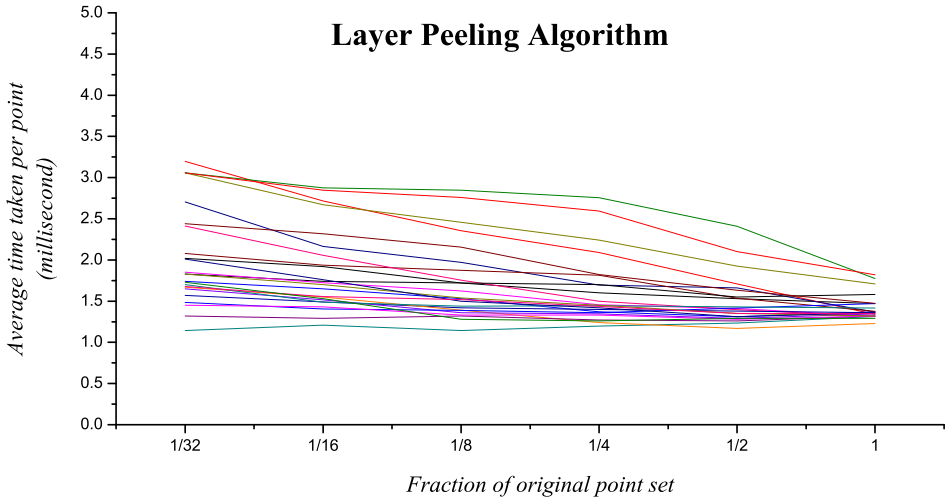
By comparing the output of the normal value between corresponding outputs by both the layer peeling and TightCocone method, we obtain the result listed in Table 6.4. It can be seen from the result that the output of layer peeling is very similar to the one produced by TightCocone across all the 24 point sets and their

corresponding under-sampled point sets. This gives confidence to the fact that the layer peeling algorithm does in fact produce correct output of high quality.

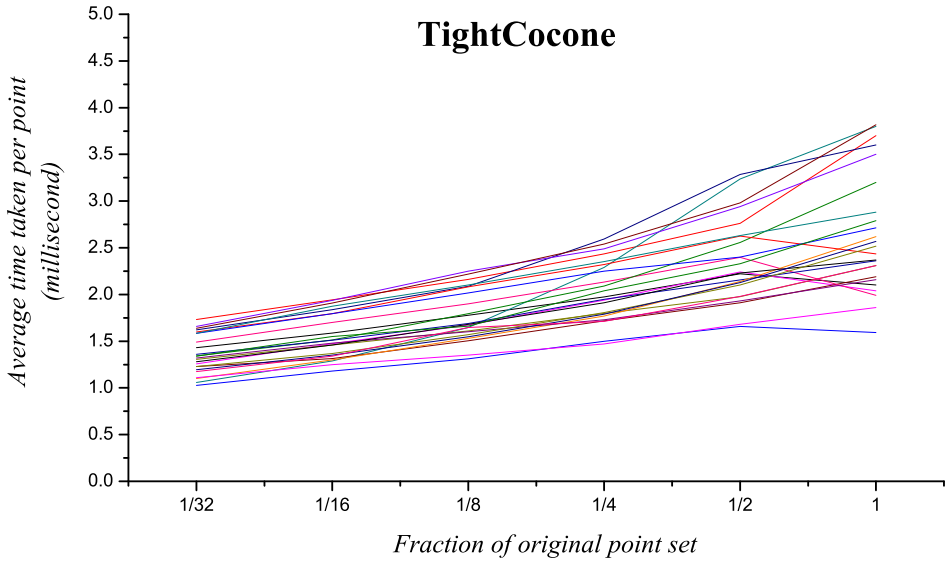
6.4 Running Time

The time taken for each of the 24 point sets to compute their surfaces are recorded for both methods. In Figure 6.13(a), we see the graph plot of each point set versus the time taken per point when using the layer peeling algorithm. Each line in the graph represents the time taken per point for one point set throughout varying degrees of under-sampling. In Figure 6.13(b), the time taken for the TightCocone algorithm is shown. Through the observation of both graphs, a few points are noted. First, it can be observed that almost all the line plots for each model follow a distinct pattern. For the case of the layer peeling algorithm, there is a steady decrease in time taken per point as point sampling gets more adequately sampled. For the case of the TightCocone algorithm however, there is a steady increase in time taken per point. This result for the TightCocone is expected as it employs a Delaunay triangulation algorithm that is known to take $O(n^2)$ time in the worst case.

Note that Figure 6.13 shows the time taken per point for various degrees of under-sampling of each point set. Hence, a $\frac{1}{32}$ sampling of the Buddha point set has more sample points than the $\frac{1}{4}$ sampling of the Bunny point set. To compare the actual timing based on size of point set, Figure 6.14(a) shows the scattered graph plot of time taken per point vs size of point set for the layer peeling algorithm. The corresponding graph for the TightCocone algorithm is shown in Figure 6.14(b). The general traits of both algorithms can be seen clearly in both graphs. The time taken per point for the layer peeling algorithm stabilizes as point set sizes

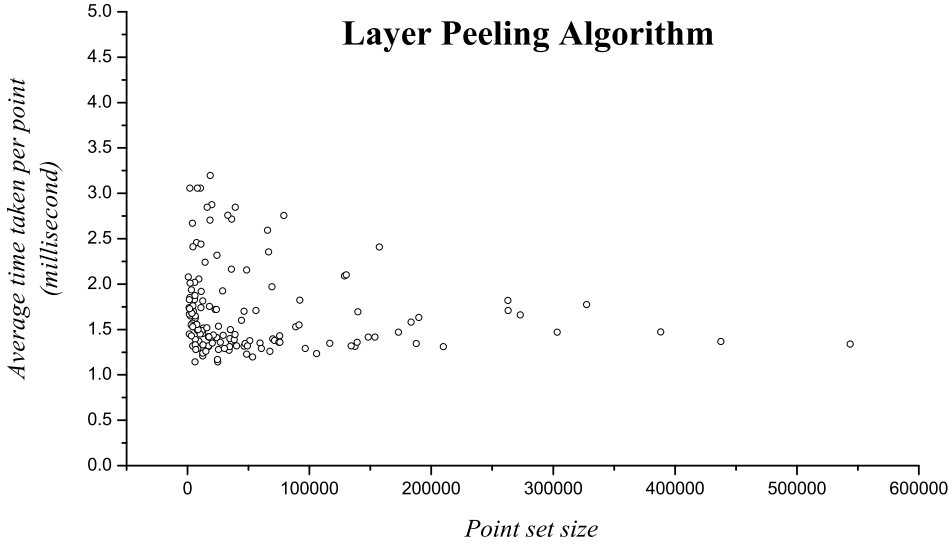


(a)

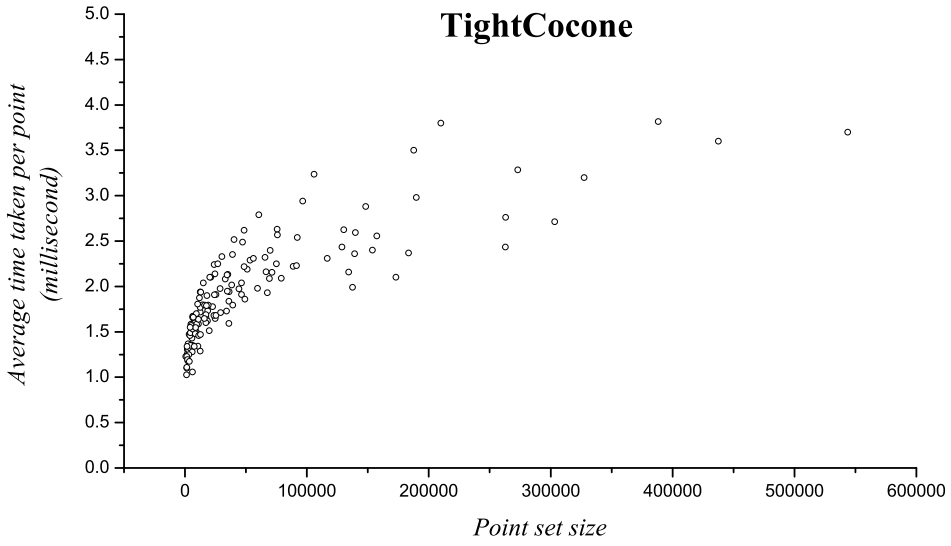


(b)

Figure 6.13: The average time taken to process a point for all the 24 point set models based on their successively down-sampled point sets.



(a)



(b)

Figure 6.14: The average time taken to process a point for all the 24 point set models based on size of point sets.

grows larger, while in the case for the TightCocone version, it can clearly be seen increasing steadily.

Taking the average of time taken per point over all the 24 point sets, the improvement in time taken is tabulated in Table 6.5. As can be seen from the table, when point sets are small the layer peeling algorithm generally takes up to around 49% more time than TightCocone. However, as the point size increases, the situation is reversed. The average time improvement for the original point set is shown to be around 46% for all the 24 point sets.

	Fraction of original point set					
	1	1/2	1/4	1/8	1/16	1/32
Time Improvement	46.39%	36.39%	20.19%	2.38%	-19.70%	-49.08%

Table 6.5: The time improvement (in percentage) of the layer peeling algorithm over the TightCocone algorithm.

6.5 Experiment Reviews

It is not surprising that the layer peeling method generally takes a shorter amount of time than the TightCocone method to perform surface reconstruction, since it is mainly a local algorithm. The noteworthy point is that the time taken for all the different point sets with different number of genus have the same behavior when the point sets are successively under-sampled. However, it remains to see if this behavior can be maintained when the point sets reach a few tens of millions of points. Currently, memory constrains do not allow the testing of such large point sets as the layer peeling algorithm does not have an out-of-core implementation. In the layer peeling algorithm, the neighborhood calculation and the global projection

test are non linear time complexity processes. However, their influences on the time taken for the point sets are not yet felt up to point set sizes of 500,000. Further experiments can be conducted to determine the timing for super large point sets.

In conclusion, the results of the experiment are as follows:

- In most cases where point sets are heavily under-sampled, some distortions and artifacts can be observed for the surfaces generated by TightCocone, while the outputs by the layer peeling algorithm remain reasonable.
- The outputs from both our algorithm and TightCocone, based on normal value similarity, are very alike, differing by at most an average of a few degrees for each point.
- The time taken by the layer peeling algorithm is much faster than the TightCocone as the size of the point set gets larger. Furthermore, the trend of the time plot indicates that the layer peeling algorithm is more suitable for large point sets.

6.6 Weaknesses and Limitations

As shown in the results of the experiments, the layer peeling algorithm is able to reconstruct the surfaces for all the point sets and their under-sampled point sets reasonably well. However, one of the reason for that is because all the points sets are not noisy. The layer peeling algorithm is dependent on the global projection method and the local convexity criterion for correctly constructing a triangle fan for each point. In the presence of noise however, most of the points are likely to encounter difficulty during the construction of their triangle fans, leading to further global projection problems in subsequent layers.

Chapter 7

Conclusion

This thesis aims to develop an efficient and robust surface reconstruction algorithm that is especially suitable for under-sampled point sets. The reconstructed surfaces from the tested point models using our algorithm have shown good results, and outperform standard algorithms that are used commonly. In addition, the layer peeling algorithm has shown to be useful towards under-sampled point sets. During the development of the layer peeling algorithm, many ideas were experimented and explored. The major contributions of this thesis are:

- The characterizing of the k-nearest neighborhood into three different types of neighborhood points. By doing so, the layer peeling algorithm can use both the propagation sequence of the surface mesh construction and the local convexity condition to effectively remove neighborhood points not on the same nearby surface.

- An implementation of the layer peeling algorithm that is able to correctly reconstruct surfaces which have thin surfaces and are under-sampled. In the experiments, point set models are reduced to a fraction of their original

point size. Even when reduced to $\frac{1}{32}$ of their original size, the layer peeling algorithm is able to correctly reconstruct them without suffering from visible distortion.

- The layer peeling algorithm is mainly a localized one. Although certain portions of the algorithm has non-linear time complexity, such as the global projection test, their effects on the overall total time taken are still negligible. As illustrated in the experiments performed for the layer peeling algorithm, the increment in the time taken per point is very little even when the size of the point set is doubled. The potential for using the layer peeling algorithm on large point sets is very favorable.

- A proof is given such that with a sufficient sampling, the layer peeling algorithm is able to generate a surface that is homeomorphic to the original surface which the sample points are sampled from. Such a proof ensures that the layer peeling algorithm is not just a specialized algorithm dealing with under-sampled point sets, but also useful for general surface reconstruction purposes.

The layer peeling algorithm was implemented and has shown to be a reliable and robust algorithm for surface reconstruction. Many interesting future researches and improvements can be made with regards to the layer peeling algorithm. The first improvement that is possible is having a better implementation for the hole filling algorithm. The current hole filling algorithm joins the boundary vertices directly, thereby producing a "flat enclosure" for the boundary points even when the surrounding surfaces are curved in shape. From an aesthetic viewpoint, the current implementation is not pleasing to the eye when the hole is large. The second

improvement that can be explored is the removal of points that are very close together (as described in Section 4.3). In Funke's algorithm [34], they estimate an initial normal for each point and then proceed to estimate the size of the restricted Voronoi cell. With that estimation, they begin to remove points from the point set. Although their implementation might be erroneous in certain situations (since the initial normal estimate is inaccurate), it is a heuristic that is useful in a highly non-uniform point set. More research can be devoted towards this area.

Some of the future works that is based on the layer peeling algorithm is extending it for out-of-core surface reconstruction, improving the speed of visualization by performing visibility determination and adapting the algorithm to noisy input point sets. As one of the purposes of the layer peeling algorithm is to make reconstructing surfaces for large point sets viable, more work can be done optimizing the layer peeling algorithm when it comes to memory management and parallel processing. With the current personal computers or workstations having dual or quad cores CPUs, we should fully utilize all the available computing power. In some of the newer works in out-of-core surface reconstruction [21, 40, 7], they make use of incremental local refinements to a coarse representation of the final reconstructed surface, making it suitable for out-of-core implementation. The layer peeling algorithm is in general a local algorithm and we hope to develop the algorithm further for adaptation to large point sets.

In another area, a new piece of work done by Katz et al. [41] uses the "Hidden" Point Removal operator to determine the visible points as viewed from a given viewpoint. This determination is performed without reconstructing the surface or estimating normals. We found this work to be similar to our approach of the global projection method, which performs the same task. Further research can be done

to develop the global projection method into an algorithm that is suitable for large scale visualization.

Finally, the last area which the layer peeling algorithm can be further adapted towards is the handling of noisy point sets. Handling noisy point sets are currently been researched upon in many other algorithms [30, 31]. The layer peeling algorithm similarly can be developed into that direction, thereby making it a more all-rounded algorithm that is suitable for all types of input point sets.

Bibliography

[1] A. Adamsom and M. Alexa. Approximating and intersecting surfaces from points. In *Proceedings of Symposium on Geometry Processing*, pages 245–254, 2003.

[2] M. Alexa and A. Adamsom. On normals and projection operators for surfaces defined by point sets. In *Proceedings of 1st Symposium on Point Based Graphics*, pages 150–155, 2004.

[3] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Defining point set surfaces. In *Proceedings of IEEE Visualization*, pages 21–28, 2001.

[4] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. In *Proceedings of IEEE Transactions on Visualization and Computer Graphics*, volume 9, pages 3–15, 2003.

[5] R. Allègre, R. Chaine, and S. Akkouche. Convection-driven dynamic surface reconstruction. In *Shape Modeling International*, pages 33–42, 2005.

[6] R. Allègre, R. Chaine, and S. Akkouche. A dynamic surface reconstruction framework for large unstructured point sets. In *Proceedings of Symposium on Point-Based Graphics*, pages 17–26, 2006.

[7] R. Allègre, R. Chaine, and S. Akkouche. A streaming algorithm for surface reconstruction. In *Proceedings of Symposium on Geometry Processing*, pages 79–88, 2007.

[8] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. In *Proceedings of 14th Annual Symposium on Computational Geometry*, pages 39–48, 1998.

[9] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: Combinatorial curve reconstruction. *Graphical models and image processing*, 60(2):125–135, 1998.

[10] N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *Proceedings of ACM SIGGRAPH*, pages 415–421, 1998.

[11] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of ACM Symposium on Computational Geometry*, pages 213–222, 2000.

[12] N. Amenta, S. Choi, and R. Kolluri. The power crust. In *Proceedings of 6th ACM Symposium on Solid Modelling*, pages 249–260, 2001.

[13] N. Amenta and Y. J. Kil. Point-set surfaces. In *Proceedings of ACM SIGGRAPH*, pages 264–270, 2004.

[14] M. Andersson, J. Giesen, M. Pauly, and B. Speckmann. Bounds on the k-nearest neighborhood for locally uniformly sampled surfaces. In *Proceedings of 1st Symposium on Point Based Graphics*, pages 167–171, 2004.

[15] H. Anton. *Elementary Linear Algebra*. John Wiley & Sons, 2000.

[16] S. Arya, D. M. Mount, N. S. Natanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest searching in fixed dimension. *Journal of the ACM*, 45(6):891–923, 1998.

[17] M. Attene and M. Spagnuolo. Automatic surface reconstruction from point sets in space. 19(1):457–465, 2000.

[18] M. Bern, H. Edelsbrunner, D. Eppstein, S. Mitchell, and T. S. Tan. Edge insertion for optimal triangulations. *Discrete & Computational Geometry*, 10(1):47–65, 1993.

[19] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[20] J. D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.

[21] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe. Multilevel streaming for out-of-core surface reconstruction. In *Proceedings of Symposium on Geometry Processing*, pages 69–78, 2007.

[22] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest neighbors and n-body potential fields. *Journal of the ACM*, 42(1):67–90, 1995.

[23] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of ACM SIGGRAPH*, pages 67–76, 2001.

[24] F. Cazals and J. Giesen. Delaunay triangulation based surface reconstruction: Ideas and algorithms. 2006. Technical Report.

[25] R. Chaine. A geometric convection approach of 3-D reconstruction. In *Proceedings of Symposium on Geometry Processing*, pages 218–229, 2003.

[26] S. W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S. H. Teng. Sliver exudation. *Journal of the ACM*, 47:883–904, 2000.

[27] T. K. Dey and J. Giesen. Detecting undersampling in surface reconstruction. In *Proceedings of 17th ACM Symposium of Computational Geometry*, pages 257–263, 2001.

[28] T. K. Dey, J. Giesen, S. Goswami, and W. Zhao. Shape dimension and approximation from samples. *Discrete and Computational Geometry*, 29:419–434, 2003.

[29] T. K. Dey and S. Goswami. Tight cocone: A water-tight surface reconstructor. *Journal of Computing and Information Science in Engineering*, 3:302–307, 2003.

[30] T. K. Dey and S. Goswami. Provable surface reconstruction from noisy samples. In *Proceedings of 20th ACM Symposium of Computational Geometry*, pages 330–339, 2004.

[31] T. K. Dey and J. Sun. An adaptive MLS surface for reconstruction with guarantees. In *Proceedings of Symposium on Geometry Processing*, pages 43–52, 2005.

[32] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[33] S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics*, 24(3):544–552, 2005.

[34] S. Funke and E. A. Ramos. Smooth-surface reconstruction in near-linear time. In *Proceedings of Symposium on Discrete Algorithms*, pages 781–790, 2002.

[35] J. Giesen and U. Wagner. Shape dimension and intrinsic metric from samples of manifolds with high co-dimension. In *Proceedings of 19th Annual Symposium on Computational Geometry*, pages 329–337, 2003.

[36] M. Gopi, S. Krishnan, and C. T. Silva. Surface reconstruction based on lower dimensional localized Delaunay triangulation. *Computer Graphics Forum (Eurographics)*, 19(3):C467–C478, 2000.

[37] T. Greg and J. O'Brien. Variational implicit surfaces. In *Tech Report GIT-GVU-99-15, Georgia Institute of Technology*, 1999.

[38] H. Hoppe, T. DeRose, and T. Duchamp. Surface reconstruction from unorganized points. In *Proceedings of ACM SIGGRAPH*, pages 71–78, 1992.

[39] A. Hornung and L. Kobbelt. Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information. In *Proceedings of Symposium on Geometry Processing*, pages 41–50, 2006.

[40] M. Isenburg, Y. Liu, J. Shewchuk, and J. Snoeyink. Streaming computation of Delaunay triangulations. *ACM Transactions on Graphics*, 25(3):1049–1056, 2006.

[41] S. Katz, A. Tal, and R. Basri. Direct visibility of point sets. *ACM Transactions on Graphics*, 26(3):24(1)–24(11), 2007.

[42] M. Kazhdan. Reconstruction of solid models from oriented point sets. In *Proceedings of Symposium on Geometry Processing*, pages 183–192, 2005.

[43] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of Symposium on Geometry Processing*, pages 61–70, 2006.

[44] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien. Spectral surface reconstruction from noisy point clouds. In *Proceedings of Eurographics*, pages 11–21, 2004.

[45] V. Kraevoy and A. Sheffer. Template-based mesh completion. In *Proceedings of Symposium on Geometry Processing*, pages 13–22, 2005.

[46] D. Levin. The approximation power of moving least-squares. *Mathematics of Computation*, 67(224):1517–1531, 1998.

[47] D. Levin. Mesh-independent surface interpolation. *Geometric Modelling for Scientific Visualization*, pages 37–49, 2003.

[48] L. Linsen. Point cloud representation. In *Technical Report, Faculty of Computer Science, University of Karlsruhe*, 2001.

[49] L. Linsen and H. Prautzsch. Fan clouds - an alternative to meshes. In *Proceedings of Dagstuhl Seminar 02151 on Theoretical Foundations of Computer Vision - Geometry, Morphology and Computational Imaging*, 2003.

[50] B. Mederos, L. Velho, and L. Henrique de Figueiredo. Smooth surface reconstruction from noisy clouds. In *Proceedings of Eurographics Symposium on Geometry Processing*, pages 53–62, 2005.

[51] B.S. Morse, T.S. Yoo, P. Rheingans, D.T. Chen, and K.R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly

supported radial basis functions. In *Shape Modeling International*, pages 89–98, 2001.

[52] D. M. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching, 2005. *http://www.cs.umd.edu/∼mount/ANN/.*

[53] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H. P. Seidel. Multi-level partition of unity implicits. In *Proceedings of ACM SIGGRAPH*, pages 463–470, 2003.

[54] J. Revelles, C. Urena, and M. Lastra. An efficient parametric algorithm for octree traversal. In *Proceedings of* 8^{th} *International Conference in Central Europe on Computer Graphics, Visualization and Interactive Media*, pages 212–219, 2000.

[55] M. Samozino, M. Alexa, P. Alliez, and M. Yvinec. Reconstruction with Voronoi centered radial basis functions. In *Proceedings of Symposium on Geometry Processing*, pages 51–60, 2006.

[56] J. Sankaranarayanan, H. Samet, and A. Varshney. A fast k-neighborhood algorithm for large point clouds. In *Proceedings of Symposium on Point Based Graphics*, pages 75–84, 2006.

[57] O. Schall and M. Samozino. Surface from scattered points: A brief survey of recent developments. In *Workshop towards Semantic Virtual Environments*, 2005. *http://cgal.inria.fr/Publications/2005/SS05.*

[58] A. Sharf, T. Lewiner, A. Shamir, L. Kobbelt, and D. Cohen-Or. Competing fronts for coarse-to-fine surface reconstruction. *Computer Graphics Forum (Eurographics)*, 25(3):389–398, 2006.

[59] J. Shlen. A tutorial on principle component analysis. 2005. *http://www.cs.cmu.edu/∼elaw/papers/pca.pdf*.

[60] C. Stoll, Z. Karni, C. Rössl, H. Yamauchi, and H. P. Seidel. Template deformation for point cloud fitting. In *Proceedings of Symposium on Point-Based Graphics*, pages 27–35, 2006.

[61] H. Xie, K. T. McDonnell, and H. Qin. Surface reconstruction of noisy and defective data sets. In *Proceedings of IEEE Visualization*, pages 259–266, 2004.

[62] H. K. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *Proceedings of IEEE Workshop on Variational and Level Set Methods*, pages 194–202, 2001.

Appendix A

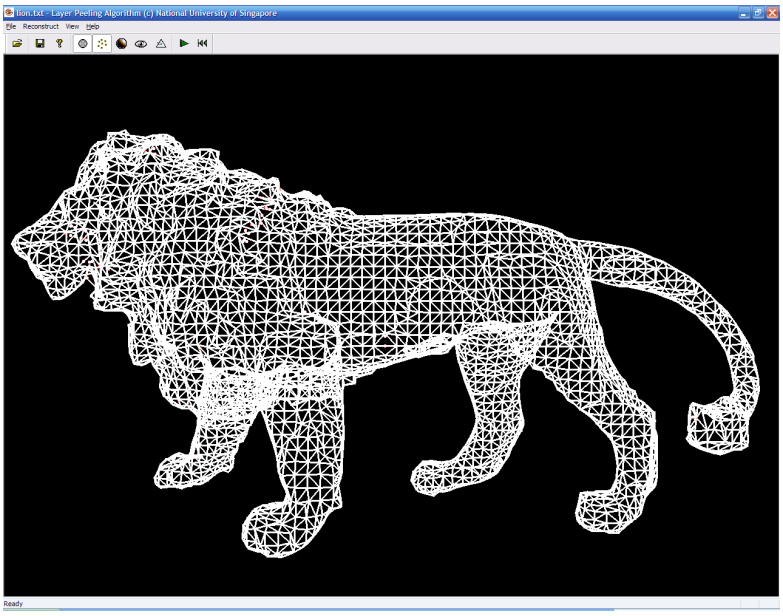
The Layer Peeling Software

The layer peeling software is an application which implements the algorithm described in this thesis. There are two versions of it, one having a graphical user interface (GUI) (Figure A.1(a)), while the other is console-based (Figure A.1(b)). They are built on the Windows platform in C++ language and are efficient, robust and user friendly. In terms of the result of the reconstruction, both versions give the same output. The GUI version provides the user with visualization capabilities, while the console version computes the surface reconstruction faster. The input is a set of 3D coordinate points without any additional information. In this appendix, we describe the settings, operations, and visualization options of the layer peeling software.

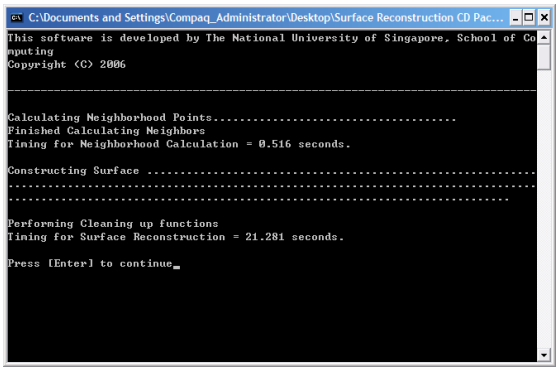
A.1 Software Setting

There are four parameter settings for both versions. These four parameters are as follows:

Single Manifold The algorithm might produce more than one connected compo-



(a)



(b)

Figure A.1: A screenshot of the layer peeling software with GUI is shown in (a), while the console version is shown in (b).

nent in the output. By selecting "yes", the algorithm searches for the largest connected component, and removes all the smaller ones.

Point Removal Some point sets have points having coordinates which are either identical or very close to each other, making the construction of the triangle fan to be extremely problematic. Based on a local estimate of the sampling density (distance to the 16th nearest neighbor), the algorithm removes points which are within a selected percentage of the distance.

Number of Layers The user can choose to construct only a stated number of layers.

Sharp Faces Removes faces which are particularly sharp.

Of the few settings that are available to the layer peeling software, the point removal function has the largest impact on the layer peeling algorithm. The layer peeling algorithm is heavily dependent on the successful creation of triangle fans. A highly irregular point sample tends to cause triangle fans to be very unbalanced in both shape and size, making the merging operations to be problematic. For such a point set, setting a high value for the point removal option will usually solve the problem. From another viewpoint, the higher the point removal value, the more points the software removes from the main surface reconstruction algorithm (though those points are added back once the reconstruction is finished). By removing more points the layer peeling algorithm takes less time to compute the surface reconstruction. Thus, for a point set that is over sampled, this method allows the user to compute the surface in a much shorter amount of time. Note that the quality of the reconstruction tends to be affected slightly if the point removal setting is set too high.

A.2 Software Operation

Once the input set is loaded into the program, the software automatically calculates the k-nearest neighborhood set of each point using the ANN software package [16] that is packaged along with the layer peeling software. Along with that, the eigenvectors and eigenvalues of all the points are also calculated. Points which are deemed too close to each other are also culled away based on the settings of the software. These points that are removed are then added back into the final reconstructed surface.

For the GUI version, the user can use standard mouse operations to view the input mode model. These operation includes:

Left Button Rotates the point model.

Middle Button Zooms in/out.

Right Button Pans the viewing plane.

In addition, there is a toolbar icon button which allows the user to modify the left mouse button rotation functionality. The user can choose to set the rotation functionality from a world view rotation (rotating the object) to a first person perspective rotation (rotating the viewpoint).

After selecting the various settings, the user can then start the reconstruction process. A progress bar is shown to notify the user of the progress of the reconstruction, including the current layer that the software is working on and the number of points that have been processed so far. Once the surface reconstruction is done, the triangulated piecewise linear surface is rendered on the viewing pane. The user can also bring up the *statistics* panel (as shown in Figure A.2) to see the time taken for the reconstruction.

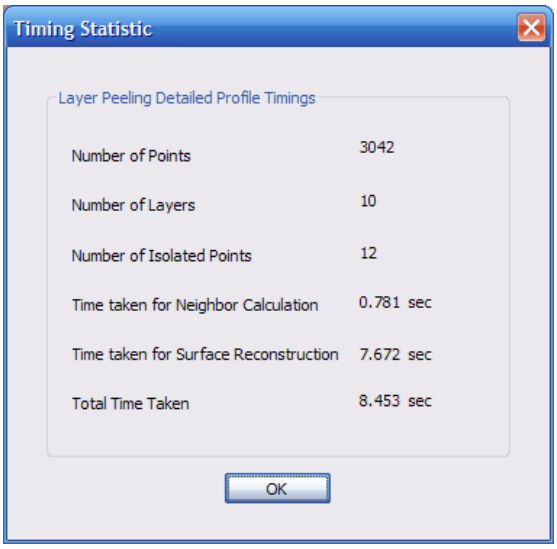


Figure A.2: The timing result of the reconstruction.

A.3 Visualization Tools

There are various visualization settings that the user can choose to view the reconstructed model.

Wireframe View Display the wireframe model.

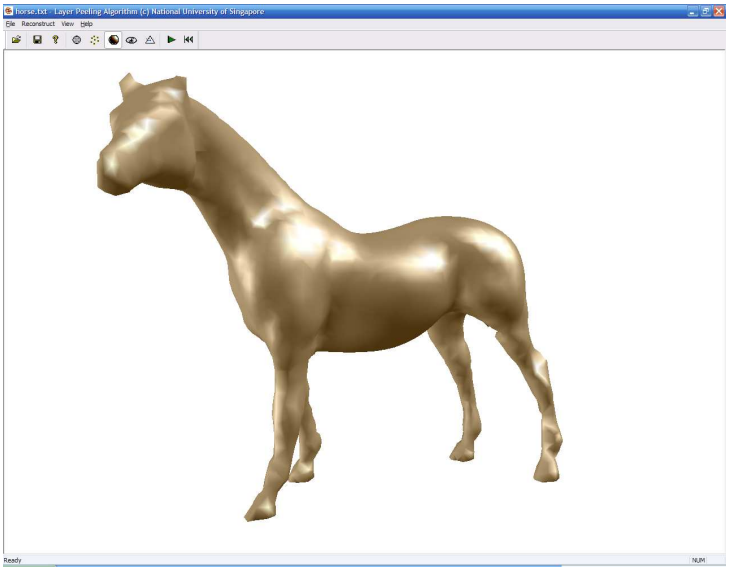
Backface Culling Hide/Show triangles faces that are facing away from the viewpoint.

Natural Rendering View the reconstructed model using Gouraud shading model.

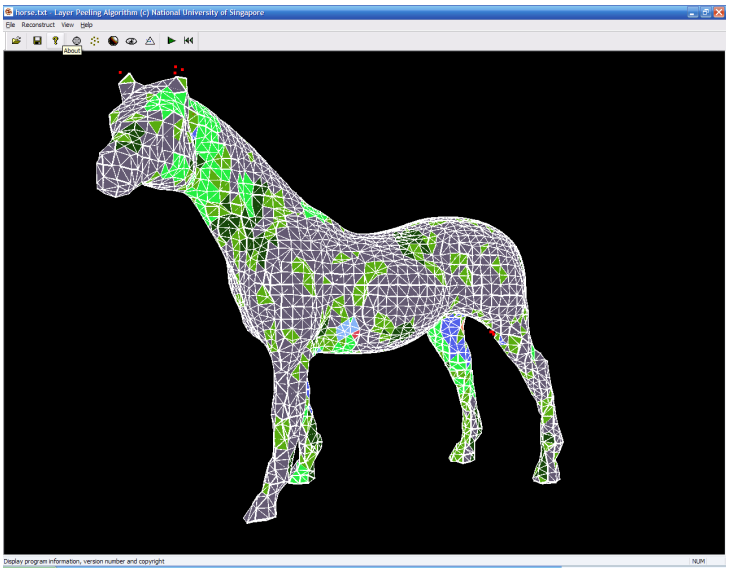
Reverse Faces Show the reverse faces.

Colored Layers Using different colors to represent the different layers that are constructed by the layer peeling algorithm.

Screenshots of the reconstructed surface in two different rendering modes are shown in Figure A.3. In addition to the different types of rendering modes, the



(a)



(b)

Figure A.3: Two different visualization types of the reconstructed surface. In (a), the reconstructed model is shown as shaded with the Gouraud shading model, while in (b) the surface is colored based on the layer which it was reconstructed in.

software has an animation sequence tool which shows the triangle fan propagation sequence. By viewing the animation sequence, the user can get a better picture of how the layer peeling algorithm progresses from the outermost layer to the inner layers.

A.4 Software Download

The layer peeling software can be download at the following URL :

- *www.comp.nus.edu.sg/∼tants/layerPeeling.html*