The Visual Computer manuscript No.
(will be inserted by the editor)

Chi-Wan Lim · Tiow-Seng Tan

Surface Reconstruction by Layer Peeling

Abstract Given an input point cloud P in $\Re^3$, this paper proposes a novel algorithm 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 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. Our algorithm not only produces a topologically correct surface for well-sampled point sets, but also adapts well to handle under-sampled point sets. Furthermore, the computational cost of the algorithm increases almost linearly in the size of the point cloud. It thus scales well to deal with large input point sets.

Keywords Surface Reconstruction · Mesh Generation · Geometric Modeling · Sampling · Scattered data

1 Introduction

Surface reconstruction from unorganized point set is a difficult and ill-posed problem as no surface information is known and there is no unique solution. Existing work can be categorized into three broad types with noisy sampling having high sampling density at one end and undersampling at the other end. Most existing algorithms focus on optimal [4] and noisy sampling types [17] that require input data to be of good sampling density that satisfies ε-sampling criterion. Little discussion has been made regarding the output of these algorithms when this criterion is not fulfilled.

Indeed, the ε-sampling criterion may not be fulfilled in practice as objects are more likely to be regularly sampled than sampled at very high density. In another view, these algorithms frequently employed triangulation algorithms (or Voronoi diagrams) as an intermediate step. This results in non-linear computational time in general (though there are special cases of triangulation that runs in linear time [9]). It is thus interesting to investigate an almost linear time surface reconstruction algorithm to deal with large input point clouds.

This paper proposes a novel surface reconstruction algorithm that runs in time almost linear to the size of the input point cloud, and well suited to handle the case of under-sampled input. Very briefly, the algorithm employs 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 to form a surface for the input.

Section 2 reviews previous work in this area and identifies issues in the existing methods. Section 3 discusses the problems that exist in reconstructing under-sampled point sets. Section 4 describes our proposed layer peeling algorithm. Section 5 details the construction and merging of triangle fans. Section 6 provides an analysis of the algorithm with undersampling and with optimal sampling conditions. Section 7 details our experimental results, and Section 8 concludes the paper.

2 Related Work

Among many works in surface reconstruction, we discuss in the following a few important branches of approaches. Other branches of theoretical works or with algebraic patches and level set methods (such as [25,27,30,31]) are less relevant to our work here.

The use of k-nearest neighborhood is quite common among many methods. The earliest such work is developed by Hoppe et al. [21]. For a point p, the algorithm uses its k-nearest neighbors to compute a 3-by-3 covariance matrix to locally estimate the signed distance function and then to generate a mesh output. Other

School of Computing, National University of Singapore,
3 Science Drive 2, Singapore 117543
E-mail: {limchiwa | tants}@comp.nus.edu.sg
Webpage: *www.comp.nus.edu.sg/∼tants/layerPeeling.html*

Fig. 1 The layer peeling algorithm is able to reconstruct surfaces from point sets of complex topology.

works such as [1,2,6,22] have tried to fit implicit surfaces over the sets of k-nearest neighborhood. Works such as [12] have tried to similarly fit radial basis functions. In general, these algorithms face difficulties during the reconstruction when the k-nearest neighborhood contains points from disjoint portions of the surface.

Another branch of global based methods uses Voronoi diagrams and Delaunay triangulations [3–5,16]. Amenta et al. [5] used the concept of poles in Voronoi cells to help estimate the point normals which are in turn used to assist in surface reconstruction. The advantage of using Delaunay triangulation is that it provides a platform to prove the correctness of these algorithms on ε-sampled surfaces. However, when the ε-sampling criterion is not met, such as in practice that point clouds are just uniformly sampled, there is no guarantee on the output quality.

Bernardini et al. [11] used a ball-pivoting algorithm to detect and form triangles to construct a surface mesh. Another such advancing-front approach is the work proposed by Scheidegger et al. [29]. Their work focuses on meshing noisy point set surfaces, allowing users to select an error bound. Our work is similar to their techniques but improves upon them by taking into consideration of point clouds possibly under-sampled and forming more perceived layers (such as the point set shown in Figure 1).

Under-sampled point sets have only been briefly mentioned in a few works, such as [14]. Their work is, however, more focused towards the detection of undersampling, rather than adapting to the problem.

The issue of improving the timing of triangulation algorithm has been explored. Funke and Ramos [18] provided a theoretical improvement to the Cocone algorithm [4] by performing a local search for possible candidates for triangulation around a particular point. Our method differs from their work by using a convexity criterion to construct triangle fans and it runs in near linear time in our extensive experiments. Gopi et al. [20] used a projection plane to perform a lower dimension Delaunay triangulation within a local space. However, their projection plane is determined only by local eigenanalysis, which can be inaccurate for an under-sampled point set.

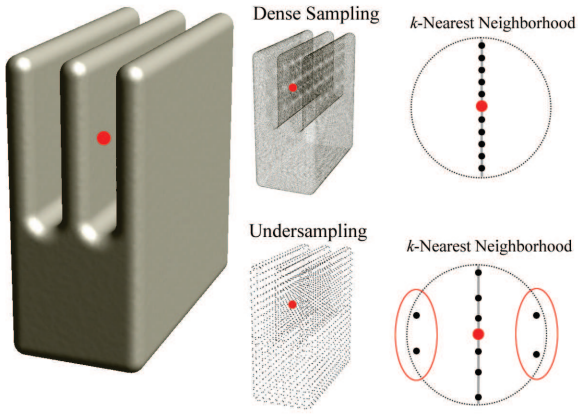


Fig. 2 The k-nearest neighbors for a point in an under-sampled point set can contain sampled points on the same surface, or sample points from other parts of the surfaces that are below or above (shown within red ovals).

3 Problems of Under-Sampled Points Sets

As discussed in the previous section, there are two main ways to perform surface reconstruction. The first one uses k-nearest neighbors for each point. In this case, if the k-nearest neighbors are all located around the local surface of the point, accurate reconstruction tends to occur. However, if some of the k-nearest neighbors are from different parts of the surface, errors during reconstruction are unavoidable. On the other hand, the second way is a more global approach that uses Delaunay triangulations. Still, the accuracy of a reconstruction (derived from the calculation of the poles) is inevitably affected by the same issue of nearby points coming from different parts of the surface [4,5].

In general, there are three possible types of sampled points in the region of interest around a single sampled point p; see Figure 2. These are sampled points that are on, above, or below the surface containing p. The main challenge for any surface reconstruction algorithm is to differentiate among them when constructing a local surface for p. This is particularly hard with under-sampled point sets as a k-nearest neighborhood generally contains more than one type of sampled points.

4 Layer Peeling Algorithm

Our objective is to construct a piecewise linear surface that fits the point cloud P. In Section 4.1, we explain the rationale behind our algorithm. Section 4.2 details one of the important procedure of the algorithm, and Section 4.3 describes the details of our algorithm.

1. Compute k-nearest neighbors of each point using the ANN software [8,26].
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.

Fig. 3 Layer Peeling Algorithm.

4.1 Algorithmic Rationale

We begin 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.*

As the manifold is closed, there exist no path that leads from the inside of the bounded volume to the outside (or vice versa) without passing through the surface. 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.*

In rendered images, any object closer to viewpoint occludes the other. Fact 2 thus follows. With these two facts in mind, 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. An example of a reconstructed surface from a point set with complex topology is shown in Figure 1. The outline of the algorithm is given in Figure 3.

4.2 Global Projection Test

A common operation needed in our algorithm is the global projection test (based on Fact 2). 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. To get around this, we determine when any point is within a certain proximity to a ray to mean also an intersection of the ray with the surface, i.e. such a ray fails the global projection test. To do this, we build an octree on the smallest bounding cube of the point set, and we say a ray intersect the surface of the point cloud when the ray passes through one or more leaf nodes of the octree containing input points.

Two notes are in order. First, in constructing this data structure, we need to decide when to stop subdividing a cube to designate 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. 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. Second, the global projection test is performed through marching from a leaf node to an adjacent one (using [28]), starting from the origin of the ray and along the ray.

4.3 Layer-by-Layer Peeling

The first layer starts with constructing a triangle fan for a point, 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 (as computed in the standard way from the covariance matrix defined on the k-nearest neighbors of p). However, we ignore points that have two out of three eigenvalues with abnormally low values. 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.

We construct a triangle fan at the chosen seed (Section 5). 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 prefer-

ence given to one with the smallest variance of dihedral angles where each is defined between a pair of triangles sharing an edge in the triangle fan. 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. 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.

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. We then create a new octree (for the global projection test) 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). 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. For the subsequent layers (if needed), we flip the normals once every alternate layer.

5 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 work on surface reconstruction [23]. Our work differs in the criteria of a suitable triangle fan, and its use within a novel layer peeling approach to determine surface neighbors. Section 5.1 details the construction of triangle fans, and Section 5.2 merging of triangle fans. Section 5.3 describes the generation of a closed manifold. And, Section 5.4 discusses the approach to handle irregularly sampled point sets.

5.1 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. The vertices $p_0, p_1, \ldots, p_i$ form the set Q_p, which is the *surface neighbors* of p. By construction, there exists a vertex $q \in Q_p$ whose triangle fan T_q has already been constructed (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 $\mathbf{n}$ at p by normalizing $\sum_{j=0}^{i}(\mathbf{t}_j \cdot \angle\alpha_j)$.

With this approximation, we are ready to define the criteria of our triangle fan T_p:

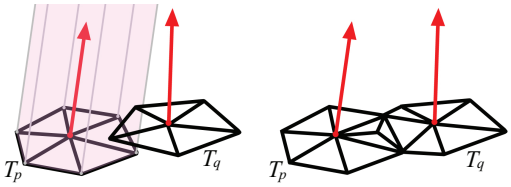
- **Local Convexity Criterion:** Each triangle $t_j \in T_p$ is such that no other point within the set $N_p - Q_p$ can be projected from above (based on normal direction and orientation of t_j) into t_j. This means t_j lies on the outermost layer of its neighborhood.
- **Normal Coherence Condition:** For all $t_j \in T_p$, we have $\mathbf{n} \cdot \mathbf{t}_j > 0$. This 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.

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 of T_p 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 dihedral angle (made with the previous triangle) closes to 180°. 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 constructed, if any, that passes the three criteria is then a candidate for triangle fan merging.

5.2 Triangle Fan Merging

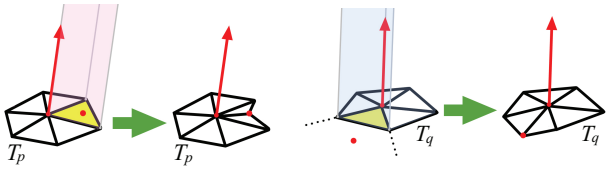
We approximate the local surface region around p using T_p, with the intention of forming a single piecewise linear surface covering over the entire point set. Starting with the first triangle fan that is created at the seed, the algorithm merges each successive new triangle fan into M. We describe the merging process in the next paragraph. Before that, we note that a triangle fan has the normal direction as given in Section 5.1, and the orientation by the global projection test. Also, each face of a triangle is considered to be two faces: the front face whose normal makes a positive dot product with the triangle fan's normal, and the back face otherwise.

We merge two triangle fans together based on a rule that is similar to the global projection test. For a triangle fan T_p, we project a ray from all its triangles' 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



(a) (b)

Fig. 4 Two triangle fans T_p and T_q are merged together. In (a), it shows that the ray projection from T_p hits the back face of T_q. In (b), both triangle fans are merged together.



(a) (b)

Fig. 5 A simple triangulation process is done by adding points (red dots) to the present triangulation. In (a), a point is added to the triangle which it is projected onto. In (b), a point can also be added to a triangle if it is projected within the range of that triangle (as shown within the wedge defined by the two dotted lines).

in T_p hits any back face of any triangle in T_q or vice versa, where $q \in N_p$. Figure 4 shows the case where the ray from T_p hits T_q, hence a merging process is required. Merging is formed by a simple triangulation process through the addition of points into a triangle fan as shown in Figure 5. 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. This is to maintain the normal coherence.

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 increase the minimum dihedral angle (i.e. preference on dihedral angle close to 180°) 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 180° [24].

5.3 Closed Manifold

Since T_p uses only points from N_p, it is likely that holes in M may exist after the layer peeling algorithm is completed. In order to produce a closed manifold, we use a simple hole filling algorithm as follows. Throughout the triangle fan construction and merging process, we maintain a list of boundary points. For each boundary point, there are two boundary edges incident to it to form an angle (outside of M). In the order of the priority with preferences to small angles, the algorithm repeatedly inserts into M a triangle formed by a boundary point and its two boundary edges. In the process, the list of boundary points thus changes and priorities are updated accordingly. In the event that self intersection occurs due to an insertion, the affected triangles are removed, creating new boundary points. This new list of boundary points is closed up in a similar fashion.

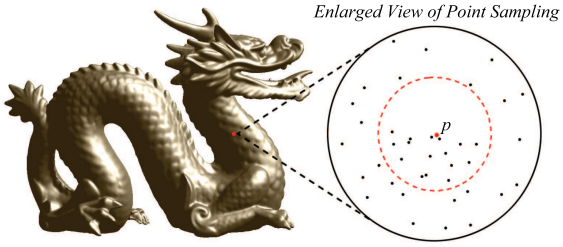


Fig. 6 The irregular sampling of points around a point p in the Dragon point set is shown. The red dashed circle indicates the bounds of the k-nearest neighbors of p.

5.4 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 6. The first problem occurs when some neighbors (relative to the other k-nearest neighbors) are too close to p, thereby forming non-uniformly sized triangles within T_p that causes (projection) problems in the merging process. 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.

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. Those surviving points at the end of the decimation process then have their k-nearest neighbors recalculated, 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 triangles for use in the triangle fan at p.

6 Analysis

In this section, we provide an analysis of our layer peeling algorithm. Section 6.1 provides an explanation that the proposed layer peeling algorithm can handle undersampled point sets well. Furthermore, Section 6.2 shows

that under optimal-sampling condition, the layer peeling algorithm is also a provable surface reconstruction algorithm. Section 6.3 discusses the computational time of our algorithm.

6.1 Under-Sampled Point Sets

By the local convexity criterion used in the triangle fan construction, we avoid very effectively the problem of p forming a triangle fan with points in N_p lying below the surface containing p. As for the other problem of points lying above p but in N_p, we explain in the next paragraph that the layer peeling process can resolve it effectively too.

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 (Section 6.2), 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 the local surface around p.

6.2 Optimal-Sampled Point Sets

The medial axis of the surface S is the closure of the set of points in $\Re^3$ 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. 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 a point in P at distance at most $\varepsilon f(p)$. For the purpose of our proof, we require a stricter sampling condition, known as an (ε, δ)-sampling [15]. 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 $\frac{\varepsilon}{2} \leq \delta < \varepsilon < 1$.

For the remaining part of this section, we assume the point set P to be an (ε, δ)-sample. A method to obtain an (ε, δ)-sample from an ε-sample is provided in [18]. The Delaunay triangulation of P restricted to S is the dual complex of the restricted Voronoi diagram of P. The restricted Voronoi diagram is the collection of all restricted Voronoi cells, and the restricted Voronoi cell of a sample point $p \in P$ is the intersection of the Voronoi cell of p with S. With these, we next provide the analysis of our proposed algorithm.

We require two lemmas from [3,19]. 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.

Lemma 1 [19] *For $p, q \in P$, if pq is an edge of the restricted Delaunay triangulation, then*

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

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

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

Corollary 1 *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 38.1° 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 $\varepsilon = 0.1$. □

In the following, 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. For an (ε, δ)-sample, [7] provides a formula to calculate the value of k, such that N_p contains all points within the local region around p. The next theorem shows that the layer peeling algorithm does not prematurely terminate before a manifold is constructed. For the proof, it is sufficient to show the existence of a seed to construct a triangle fan, though our algorithm usually utilizes points from the boundary of M for the purpose.

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

Refer to Figure 7. For point p, the local region around p (shown in dashed red circle) is bounded by two balls of radius $f(p)$. Now consider one of the ball B. We tilt the ball in any arbitrary direction while pivoting at point p until a point q is hit. Similar to [11], we now pivot the ball on the edge pq. By rotating the ball on the edge pq, ball B comes into contact with another point r (not shown in the 2D Figure 7), forming a triangle pqr. Since the surface S is ε-sampled, therefore a ball of radius $\varepsilon f(p)$ cannot penetrate S. Thus the maximum radius of the

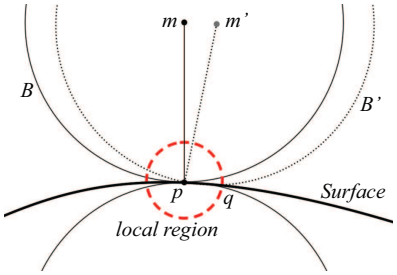


Fig. 7 A medial ball centered at m is pivoted at point p. The maximum deviation of the line pm is pm'.

circumcircle of pqr can be at most $\varepsilon f(p)$. The maximum tilt of ball B happens when its surface intersects the other medial ball's surface 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 7. 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. 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. □

In the way we derive subsets of P (in Step 3 of Figure 3), Theorem 1 also holds for each subset. The next lemma proves that the intersection of S with the local region around any particular point p is a topological disk.

Lemma 3 *Consider a point $p \in P$ with $\mathbf{n}$ 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 $\mathbf{n}$ 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 38.1° by Corollary 1. 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. 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 [4].

First, we show that the restricted Delaunay triangulation within the local region of p can be projected to a 2D plane. By Lemma 3, 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 those dual 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 sliver, for example a splinter or spike sliver, as classified in [13], 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 [10]. Such a triangulation does not have badly shaped triangles as the local region to be constructed is known to obey Lemma 3. With this, we can now project the triangulation around the local region of p to a 2D plane.

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. 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. This completes our series of operations and the proof. □

6.3 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 structures consist of spatial tree

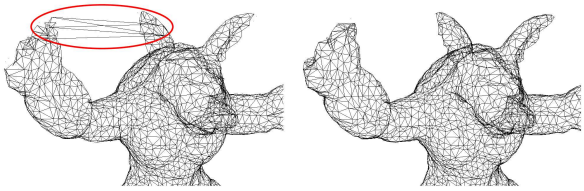
decomposition approaches. Both require $O(n \log n)$ time to construct, and $O(\log n)$ time to process for each point where n is the number of input points. For the former, we only construct it once at the start of the algorithm and the actual timing taken by this process is insignificant when compared with that by 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 the next section.

7 Experimental Results

We have implemented our algorithm on a Pentium IV 3.0GHz, 4GB DDR2 RAM and nVidia GeForce 6600 with 256MB DDR3 video memory. For purposes of comparison, we downloaded the commonly used TightCocone software [16] to run on the same machine as a benchmarking algorithm. For our implementation, we take 16 to be the value of k. Although the upper bound stated in [7] is 32, we found that for our experiments 16 is sufficient. For a comprehensive comparison, we use nine real point sets (indicated in Figure 12) available from *www.cs.princeton.edu/gfx/proj/sugcon/models/* and *www.cyberware.com*, and one artificially created point set as shown in Figure 2. The point sets have sizes ranging from 35,947 (Bunny) to 183,408 (Lion). In each case, we run the algorithms on the original point sets, and then progressively run on smaller samples of the original data to assess the robustness of the algorithms in the presence of undersampling. Smaller samples are obtained through uniformly undersampling the original point sets using Geomagic Studio software. Furthermore, we run the algorithm on another three large and irregularly sampled point sets from *graphics.stanford.edu/data/3Dscanrep/*.

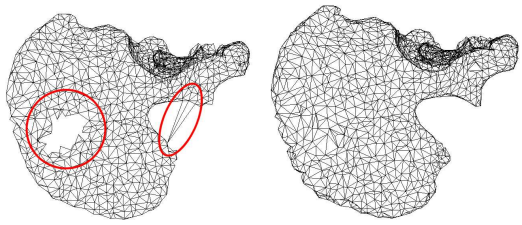
Visual Quality. Figure 8 to Figure 10 highlight the differences in some of the outputs of our algorithm as compared to that of TightCocone; see accompanying video for more examples. Our algorithm generally respects the local features of the point cloud, and handles thin regions well. It usually does 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.

Normal. In many cases, comparing output meshes from TightCocone and our algorithm do not provide any insight to the quality of the results as both outputs use different sets of edges and triangles but yet looked identical visually in most parts. As such, we turn to comparing normals extracted by both algorithms. To do this as presented in Table 1 for our nine sets of point data, we fix the ground truth of the normals as one that is computed by TightCocone on the original point set. (This is the reason that the entries in the column for the original point sets in Table 1 for TightCocone are 0.) We also include in Table 1 the results of our implementation of the normal computation with simple k-nearest neighbor-



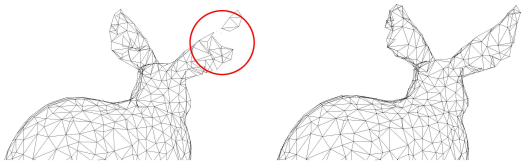
(a) (b)

Fig. 8 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)

Fig. 9 Meshing results of the Hip Bone point data (1964 points). Result (a) is produced by the TightCocone where various deficiencies in the meshing results are highlighted. Result (b) is produced by our layer peeling algorithm.

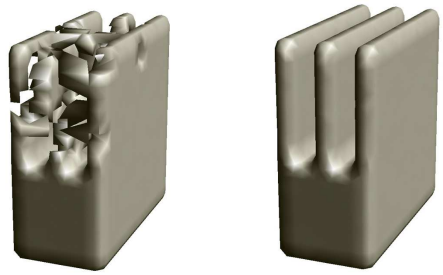


(a) (b)

Fig. 10 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 algorithm result is shown in (b).

Algorithm		Fraction of Original Point Set					
		1	1/2	1/4	1/8	1/16	1/32
Hoppe [21]	Ave	2.808	4.053	6.092	9.151	16.628	33.181
	Min	1.602	2.712	4.158	5.642	7.274	8.802
	Max	7.761	7.171	9.776	12.798	33.123	118.234
TightCocone [16]	Ave	0	1.194	3.218	4.99	7.603	10.5
	Min	0	1.151	1.794	2.758	4.835	6.367
	Max	0	4.319	5.801	9.563	13.044	15.674
Layer Peeling	Ave	0.53	2.021	3.213	4.733	7.109	9.925
	Min	0.121	1.248	1.921	2.93	4.909	6.42
	Max	2.744	4.449	5.949	8.184	11.357	15.777

Table 1 Average, minimum and maximum difference (in degree) of normals computed by the different methods for the nine models.



(a) (b)

Fig. 11 Meshing results of the artificially created undersampled 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).

hood method [21]. The result produced by [21] indicates that by only using a simple eigenanalysis of k-nearest neighborhood is often not desirable, especially for highly undersampled point set. Furthermore, it highlights the fact that our layer peeling algorithm does in fact improve upon simple k-nearest neighborhood algorithm. In addition, we observe that our algorithm matches very well with TightCocone in terms of computed normals for both the original point sets as well as those undersampled point sets.

To test the three algorithms on a point set which have known normals, we use the data set from the object shown in Figure 2. The result is tabulated in Table 2 and shown in Figure 11. The proximity between the opposite surfaces gets relatively smaller as the point set is being gradually undersampled. When the point set is being reduced to $\frac{1}{32}$ of its original size, [21] breaks down while [16] suffers from some distortion as shown in Figure 11(a). On the other hand, Figure 11(b) shows that our algorithm is still able to maintain relatively good output.

Running Time. Our algorithm runs in general faster than TightCocone. However, when the size of the point set model is small, the converse is true. This is mainly due to the overhead incurred for the calculation of the k-nearest neighbors and the construction of the octree for the global projection test. As the point set grows larger in size, the advantage of using the layer peeling algorithm becomes evident. We also notice the average time taken per point in our case across different sizes of each point set is roughly a constant; see Figure 12. This means our algorithm runs in almost linear time, as it is largely a local algorithm. This compares favorably to TightCocone which runs in non-linear time as observed in our experiment. Our algorithm thus scales well to large point clouds. This can be very important as large point clouds will be commonly used to produce high quality models.

Irregularly Sampled Point Sets. Table 3 shows the result of our algorithm on three large and irregularly sampled point sets: Buddha, Dragon, and Lucy. These

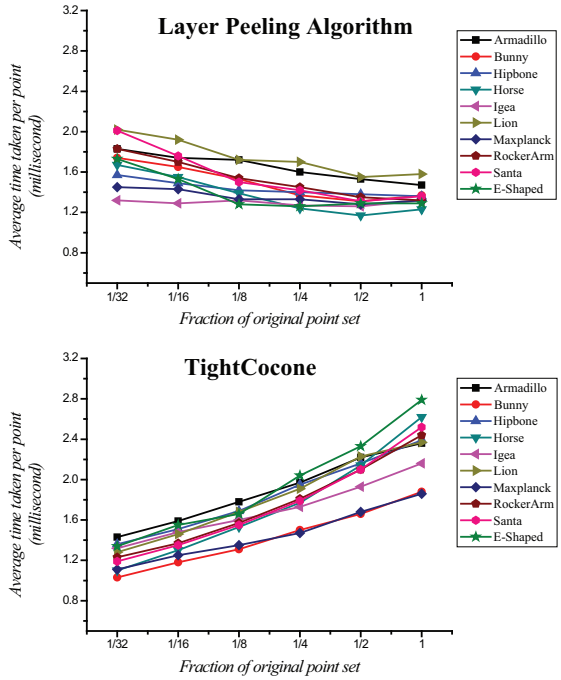


Fig. 12 The average time taken to process a point.

Algorithm	Fraction of Original Point Set					
	1	1/2	1/4	1/8	1/16	1/32
Hoppe [21]	1.608	1.83	2.386	3.954	5.515	93.929
TightCocone [16]	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 2 Average difference (in degree) of normals computed by the different methods for the E-shaped object in Figure 2.

Point Set	Point Size	Time (Layer Peeling)	Time (TightCocone)	Computed Normals Difference
Buddha	543652	1.505	3.882	0.013
Dragon	437626	1.297	4.154	0.107
Lucy	262909	1.94	2.675	0.283

Table 3 Results of the three irregularly sampled models. The time taken per point is in millisecond while the average difference in computed normals is in degree.

three point sets are much larger in size than the nine regularly sampled point sets. Not only does the running time of our layer peeling algorithm remains in the same order as that for the nine regularly sampled point sets (as shown in Figure 12), it is also much faster than the TightCocone algorithm. Furthermore, the similarity in the calculated normal values indicates the accuracy of our reconstruction on irregularly sampled point sets.

8 Conclusion

We present a novel approach for constructing surfaces from point clouds through a triangle fan construction. Under optimal sampling conditions, we prove that our algorithm is able to produce a homeomorphic surface to

the sampled surface. Our algorithm adapts well to undersampled point sets with the use of the convexity criterion and the global projection test. Furthermore, the largely local nature of the algorithm allows the computational cost of the reconstruction process to scale almost linearly with the size of the input. Potential future work includes the extension of our layer peeling algorithm to handle noisy point sets.

Acknowledgements This research is supported by the National University of Singapore under grant R-252-000-216-112.

References

1. Alexa, M., Adamsom, A.: On normals and projection operators for surfaces defined by point sets. In: Proceedings of 1st Symposium on Point Based Graphics, pp. 150–155 (2004)
2. Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C.T.: Point set surfaces. In: Proceedings of IEEE Visualization, pp. 21–28 (2001)
3. Amenta, N., Bern, M.: Surface reconstruction by Voronoi filtering. In: Proceedings of 14th Annual Symposium on Computational Geometry, pp. 39–48 (1998)
4. Amenta, N., Choi, S., Dey, T.K., Leekha, N.: A simple algorithm for homeomorphic surface reconstruction. In: Proceedings of 16th Anuual Symposium on Computational Geometry, pp. 213–222 (2000)
5. Amenta, N., Choi, S., Kolluri, R.: The power crust. In: Proceedings of 6th ACM Symposium on Solid Modelling, pp. 249–260 (2001)
6. Amenta, N., Kil, Y.J.: Point-set surfaces. In: Proceedings of ACM SIGGRAPH, pp. 264–270 (2004)
7. Andersson, M., Giesen, J., Pauly, M., Speckmann, B.: Bounds on the k-nearest neighborhood for locally uniformly sampled surfaces. In: Proceedings of 1st Symposium on Point Based Graphics, pp. 167–171 (2004)
8. Arya, S., Mount, D.M., Natanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest searching in fixed dimension. Journal of the ACM **45**(6), 891–923 (1998)
9. Attali, D., Boissonnat, J.: A linear bound on the complexity of the Delaunay triangulation of points on polyhedral surfaces. In: Proceedings of 14th Annual Symposium on Computational Geometry, pp. 39–48 (1998)
10. Bern, M., Edelsbrunner, H., Eppstein, D., Mitchell, S., Tan, T.S.: Edge insertion for optimal triangulations. Discrete & Computational Geometry **10**(1), 47–65 (1993)
11. Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., Taubin, G.: The ball-pivoting algorithm for surface reconstruction. IEEE Transactions on Visualization and Computer Graphics **5**(4), 349–359 (1999)
12. Carr, J.C., Beatson, R.K., Cherrie, J.B., Mitchell, T.J., Fright, W.R., McCallum, B.C., Evans, T.R.: Reconstruction and representation of 3D objects with radial basis functions. In: Proceedings of ACM SIGGRAPH, pp. 67–76 (2001)
13. Cheng, S.W., Dey, T.K., Edelsbrunner, H., Facello, M.A., Teng, S.H.: Sliver exudation. In: Proceedings of 15th Annual Symposium of Computational Geometry, pp. 1–13 (1999)
14. Dey, T.K., Giesen, J.: Detecting undersampling in surface reconstruction. In: Proceedings of 17th Annual Symposium of Computational Geometry, pp. 257–263 (2001)
15. Dey, T.K., Giesen, J., Goswami, S., Zhao, W.: Shape dimension and approximation from samples. Discrete & Computational Geometry **29**, 419–434 (2003)
16. Dey, T.K., Goswami, S.: Tight cocone: A water-tight surface reconstructor. Journal of Computing and Information Science in Engineering **3**, 302–307 (2003)
17. Dey, T.K., Goswami, S.: Provable surface reconstruction from noisy samples. In: Proceedings of 20th Annual Symposium of Computational Geometry, pp. 330–339 (2004)
18. Funke, S., Ramos, E.A.: Smooth-surface reconstruction in near-linear time. In: Proceedings of Symposium on Discrete Algorithms, pp. 781–790 (2002)
19. Giesen, J., Wagner, U.: Shape dimension and intrinsic metric from samples of manifolds with high co-dimension. In: Proceedings of 19th Annual Symposium on Computational Geometry, pp. 329–337 (2003)
20. Gopi, M., Krishnan, S., Silva, C.T.: Surface reconstruction based on lower dimensional localized Delaunay triangulation. Computer Graphics Forum (Eurographics) **19**(3), C467–C478 (2000)
21. Hoppe, H., DeRose, T., Duchamp, T.: Surface reconstruction from unorganized points. In: Proceedings of ACM SIGGRAPH, pp. 71–78 (1992)
22. Levin, D.: Mesh-independent surface interpolation. In: Geometric Modelling for Scientific Visualization, pp. 37–49 (2003)
23. Linsen, L., Prautzsch, H.: Fan clouds - an alternative to meshes. In: Proceedings of Dagstuhl Seminar 02151 on Theoretical Foundations of Computer Vision - Geometry, Morphology and Computational Imaging (2003)
24. Mederos, B., Velho, L., de Figueiredo, L.H.: Smooth surface reconstruction from noisy clouds. In: Proceedings of Eurographics Symposium on Geometry Processing, pp. 53–62 (2005)
25. Mitra, N.J., Nguyen, A.: Estimating surface normals in noisy point cloud data. In: Proceedings of 19th Annual Symposium on Computational Geometry, pp. 322–328 (2003)
26. Mount, D.M., Arya, S.: ANN: A library for approximate nearest neighbor searching (2005). http://www.cs.umd.edu/∼mount/ANN/
27. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., Seidel, H.P.: Multi-level partition of unity implicits. In: Proceedings of ACM SIGGRAPH, pp. 463–470 (2003)
28. Revelles, J., Urena, C., Lastra, M.: An efficient parametric algorithm for octree traversal. In: Proceedings of WSCG, pp. 212–219 (2000)
29. Scheidegger, C.E., Fleishman, S., Silva, C.T.: Triangulating point set surfaces with bounded error. In: Proceedings of Eurographics Symposium on Geometry Processing, pp. 63–72 (2005)
30. Xie, H., McDonnell, K.T., Qin, H.: Surface reconstruction of noisy and defective data sets. In: Proceedings of IEEE Visualization, pp. 259–266 (2004)
31. Zhao, H.K., Osher, S., Fedkiw, R.: Fast surface reconstruction using the level set method. In: Proceedings of IEEE Workshop on Variational and Level Set Methods, pp. 194–202 (2001)

C. W. Lim is a PhD student at the School of Computing, National University of Singapore. His interests in computer graphics include point based graphics and surface reconstruction.

T. S. Tan is an Associate Professor at the School of Computing, National University of Singapore. His research interests are in interactive computer graphics and computational geometry.