Communications in Information and Systems
Volume 17, Number 2, 85–113, 2017

Incremental reconstruction of water-tight surface from unorganized points

Rao Fu, Cheng Wen, Riqing Chen, Yifan Fu, and Jian Wu*

Most surface reconstruction algorithms require to input all sample points during a single process to construct a surface, but this requirement limits their applications in an incremental surface reconstruction under a consecutive point acquisition process. In this paper, we propose an incremental water-tight surface reconstruction algorithm, which allows the surface to be updated along with the incremental construction of the Delaunay triangulation and guarantee to output a water-tight surface. The core functioning structures, called regular umbrella-covered graph ($RUCG$) and constrained umbrella (CU), are defined to represent the dynamic reconstruction process. Experiments are carried out to demonstrate the robustness of the proposed algorithm to reconstruct surface from sparse, dense, little noisy point clouds and point clouds of discrete objects. Comparisons with other Delaunay-Voronoi based algorithms are presented to support the effectiveness and superiority of the proposed algorithm. The proposed algorithm is quite simple, fast and non-parametric, and its complexity is only related to the 3D Delaunay triangulation of the points.

1. Introduction

In recent years, 3D data-acquisition techniques, such as optics, ultrasound and electromagnetism, have thrived following the wide spread interest of reverse engineering, cartography, and medical imaging. Bernardimi et al. [1] summarize the typical four steps of 3D data-acquisition, including scanning, data registration, data integration and model conversion. In the field of computer graphics and computational geometry, surface reconstruction belongs to the data integration phase. The problem of surface reconstruction can be stated as follows: Given a finite point set $P \subset \mathbb{R}^3$, sampled from a surface

*Research supported in part by Knowledge Innovation Program of Basic Research Projects of Shenzhen under Grant No. JCYJ20160428182053361 and Guangdong Science and Technology Plan under Grant No. 2017B020210003.

$S \subset \mathbb{R}^3$, the aim of surface reconstruction is to recover a surface fitting the original one [2, 3].

Surface reconstruction is similar to signal reconstruction, and a theory resembling Nyquist sampling condition should be presented in the field of surface reconstruction. Amenta et al. [4, 5] first give a "provably correct" surface reconstruction algorithm based on the notion of ϵ-sample. However, ϵ-sample is a little restrictive, since a sample may be too sparse, too non-uniform or too noisy to capture the features of the original surface. Algorithms based on ϵ-sample such as [4, 6–8] fail to construct not "well-sampled" parts of the original surface. Dey [9] proposes the (ϵ, κ)-sample that takes noise into consideration, and gives a theoretical guarantee on his surface reconstruction algorithm. However, his algorithm will get into difficulty when the point set contains sparse sampling regions. Ohrhallinger et al. [10] present a heuristic algorithm to connect sparsely sampled points, but have trouble when encountering dense point sets. It seems that a universal sampling condition integrating with sparsity, density and noise is still challenging.

In this paper, we propose an incremental surface reconstruction algorithm to construct a genus zero water-tight surface in spite of different sampling conditions. Here, "water-tight" means that a surface bounds a solid. A water-tight surface is a 2-manifold without boundary. Especially in the discrete case, every triangle in the surface mesh has three neighboring triangles, and every edge in the surface mesh is incident to two triangles. Specific details about our algorithm is introduced in the following sections.

2. Related work

Surface reconstruction can be categorized into two major types. One is approximation and the other is interpolation [11, 12]. Approximation methods such as [13, 14] aim at finding an implicit function that best fits the input data, and use Marching Cubes (MC) [15] to extract the ultimate triangulated surface. Zhao et al. [16] describe another type of implicit surface reconstruction based on the level set method. In their opinion, the surface reconstruction problem can be solved by minimizing an energy function that measures the distance between the surface and points.

$$E(\mathbf{\Gamma}) = \left[\int_{\mathbf{\Gamma}} d^p(\mathbf{x})\mathrm{d}s\right]^{\frac{1}{p}}, 1 \leq p \leq \infty \tag{1}$$

where $\mathbf{\Gamma}$ is an arbitrary surface, ds is the surface area, and $d(\mathbf{x})$ is the distance from a point $\mathbf{x} \in \mathbf{\Gamma}$ to the closest point in the point set.

Interpolation methods build upon Delaunay-Voronoi concepts, and can directly output a surface mesh. These types of algorithms only require to input point clouds without any additional information such as normal vectors or scanner information, but are extremely useful in areas such CAD applications where sample points must be exactly on the reconstructed surface. The proposed algorithm in this paper falls into these type.

Delaunay triangulation and Voronoi diagram are usually utilized mutually in reconstruction. The pivotal point is that the medial axis can be approximated by the Voronoi diagram. Amenta et al. [5] propose the crust algorithm. They define poles as the two Voronoi vertices that are farthest from the cell's sample point on the two side of the surface and use them to filter triangles in the Delaunay triangulation. Later, they introduce the power crust algorithm [7], which assigns a weight valued by the radius of the corresponding polar ball to each pole to compute a power diagram, and use the power diagram to extract a surface. Amenta et al. [6] also propose another algorithm derived from the crust, the cocone algorithm, which selects surface triangles dual to the Voronoi edges intersected by the cocones. Dey et al. [17] develop the cocone algorithm into the tight cocone algorithm to obtain a water-tight surface, which functions as peeling off all tetrahedrons marked out. In a later study, they [18] propose a localized version of the cocone algorithm to reconstruct very large but of good quality point sets.

Only Delaunay complexes can be performed for reconstruction relying on certain topological selecting criteria. Boissonnat [19] develops a sculpture algorithm according to heuristic information, but sometimes the algorithm may quickly end up in local minima and miss interpolating many of the given points. Edelsbrunner and Mücke [20] propose the α-shapes algorithm, which requires the user to input a globally uniform parameter. It may fail in processing some details due to the globally uniform parameter. Later, Dey et al. [21] peel tetrahedrons from the α-complex greedily to construct a surface with provable guarantees. Adamy et al. [22, 23] introduce the notion of λ-interval, and choose umbrellas locally at the vertices from the Gabriel complex to minimize the maximum of the lower λ-interval bounds. But post-processing is inescapable for guaranteeing a water-tight surface. Chaine [24] unifies Zhao's convection model [16] to a geometric field, i.e., using Gabriel property to evolve an enclosing surface embedded in the 3D Delaunay triangulation is equal to minimizing the energy function (1). Ohrhallinger et al. [10] transfer the surface reconstruction problem to find the triangle mesh minimizing the sum of all triangles' longest edges, and they introduce the techniques of inflating and sculpturing to enhance the mesh quality. Peethambaran et al. [8] introduce a topological correct sculpture algorithm, whose output is a proximity graph to capture the proximity of sample points.

However, all aforementioned algorithms are designed to input all sample points during a single process to construct a surface. Thus, efficiency is reduced when encountering consecutive point acquisitions where a surface locally updates with inserting sample points dynamically. For example, a heart mapping procedure requires a subsequent reconstruction of an endocardial surface. Allgre et al. [25] extend Chanie's geometric convection approach [24] to a dynamic one, but a pseudo-surface initialization is still inevitable. Yu et al. [26] propose an incremental ray tracing algorithm to construct a 2-manifold surface from the image-based-modeling process. But the rays between cameras and sample points are prerequisite for estimating the triangular surface mesh.

In this paper, we propose an incremental surface reconstruction algorithm that selects surface triangles during updating Delaunay triangulation process, rather than after the triangulation is completed. The reconstructed surface is locally updated along with inserting new sample points one by one, and can be guaranteed as water-tight without any post-processing. Figure 1 presents how our algorithm updates the surface.

3. Preliminaries

Let P denote the point set in the three dimensional Euclidean space $\mathbb{R}^3$, ∂ denote the boundary, and $Del(P)$ denote the Delaunay triangulation of P.

Definition 1 (Simplicial complex). A simplicial complex K [8, 27] is a finite set of simplices satisfying the following two conditions:

(a): Any face of any simplex in K is also included in K.

(b): For any two simplices, $\sigma, \tau \subset K$, the intersection $\sigma \cap \tau$ is either empty or a face of both σ and τ.

A simplicial k-complex K is a simplicial complex where the largest dimension of any simplex is equal to k.

Definition 2 (Regular simplicial 2-complex). A regular simplicial 2-complex $\Re K_2$ [8] is a simplicial 2-complex K_2 satisfying the following two restrictions:

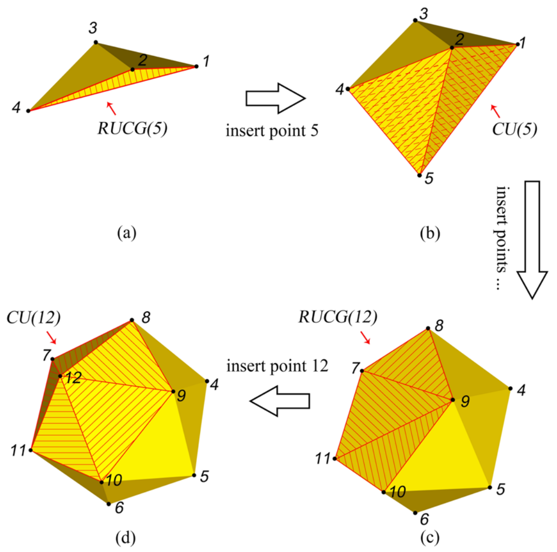


Figure 1: Point-by-point reconstruction of an icosahedron. The red line parts represent the regular umbrella-covered graph ($RUCG$) and the constrained umbrella (CU). (a) The initial four points constitute a tetrahedron. (b) When inserting point 5, we substitute the $CU(5)$ for the $RUCG(5)$. (c) Reconstruction of 11 points, and the insertion of some points is neglected. (d) The ultimate result is acquired by substituting the $CU(12)$ for the $RUCG(12)$.

(a): All points in K_2 are connected in pairs.

(b): There are no bridges, junction points and dangling edges in K_2.

Definition 3 (Umbrella). For a vertex v, an umbrella U_v [17] is the union of the surface triangles incident to v, which is homeomorphic to a disk.

An umbrella is a closed triangle fan [10], which is also a $\Re K_2$.

Definition 4 (Circle). A circle [28] is a cyclic sequence of edges such as AB, BC, CD, DE, EA, where m successive edges are incident to m distinct common vertices.

$\partial \Re K_2$ and ∂U_v are circles.

Figure 2 gives an illumination of simplicial 2-complex K_2, regular simplicial 2-complex $\Re K_2$, umbrella and circle.

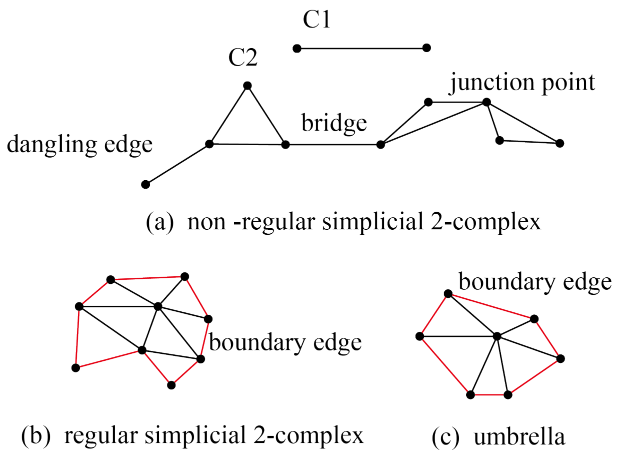


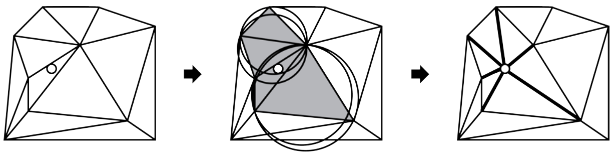
Figure 2: Illumination of the difference between the non-regular simplicial 2-complex K_2 (consisting of two components C1&C2), the regular simplicial 2-complex $\Re K_2$ and the umbrella. The boundaries of the regular simplicial 2-complex and the umbrella are marked with red. As we can see, both $\partial \Re K_2$ and ∂U_v are a circle

Definition 5 (Delaunay triangulation). For a point set P, a Delaunay triangulation (*Del* for short) is a subdivision of the space into simplices such that the circum-hypersphere of any simplex in the triangulation contains no other point in P (Empty-ball property).

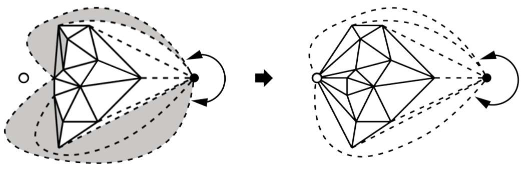
Integration of Bowyer-Watson's incremental algorithm [29, 30] with infinite point [27, 31] can help to avoid special care of the points outside the convex hull. Simplex incident to infinite point is named as ghost otherwise as solid. In this paper, we use $\hat{f}$ to denote an oriented facet (with a normal), use $mirror(\hat{f})$ to denote the one that shares common vertices but has an opposite normal regarding $\hat{f}$, and use $Incident(\hat{f})$ to denote the incident cell of $\hat{f}$. Robust geometric predicates [32] and the remembering stochastic walk [33, 34] for point location are implemented in operation of computing Delaunay triangulation. Figure 3 gives a brief introduction of the process.

Definition 6 (Gabriel simplex). A simplex is called Gabriel [35] if it meets Gabriel property, which means the smallest circumscribing ball of the simplex is empty.

The Gabriel graph (more generally the least squares adjacency graph) is a subgraph of the Delaunay triangulation excluding the obtuse triangles [28]. The edges both belonging to the Delaunay triangulation and the Gabriel



(a) Insert a point inside the triangulation



(b) Insert a point outside the triangulation

Figure 3: Illumination of inserting a point [27]. The white dot is a new vertex to be inserted, and the black dot is the infinite point. The shaded triangle is the conflict cell, and the unshaded triangle is the Delaunay cell. All shaded triangles make up the Delaunay hole (DH).

graph must intersect the Voronoi diagram. Gabriel graph of the sample points provides a good surface description [36]. Chaine [24] has proved that using Gabriel property to filter triangles in the Delaunay triangulation is equal to minimizing the energy function (1). See [24] for more details.

4. Surface reconstruction

Figure 4 provides a step-by-step description of inserting a new sample point for reconstruction. Initialization step is to choose four affinely independent points (no three points are collinear and no four are coplanar) to initialize a solid tetrahedron (labeled as internal) and four ghost tetrahedrons (labeled as external), and choose the four solid oriented triangles to be the initial surface mesh. Then inserting another new sample point follows the sequences of the remaining steps of Figure 4.

4.1. Searching for the regular umbrella-covered graph

Definition 7 (Regular umbrella-covered graph). A regular umbrella-covered graph ($RUCG$) is a graph satisfying the following three conditions:

(a): $RUCG$ is a $\Re K_2$.

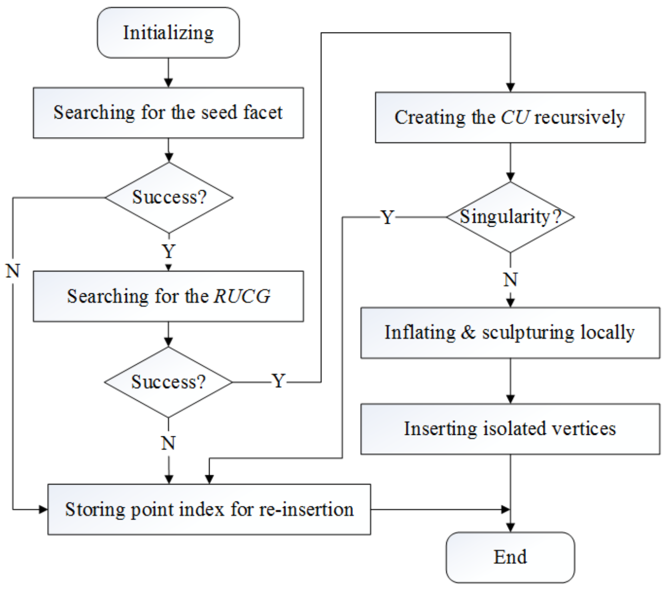

Initializing
Searching for the seed facet
Success?
Y
N
Searching for the RUCG
Success?
Y
N
Creating the CU recursively
Singularity?
Y
N
Inflating & sculpturing locally
Inserting isolated vertices
Storing point index for re-insertion
End

Figure 4: A description of inserting one sample point. There may be exceptions in the steps of "searching for the seed facet", "searching for the $RUCG$" and "creating the CU recursively". When encountering the exception, we temporarily store the point index and re-insert it again at the end of the algorithm.

(b): $RUCG$ is the union of triangles in the surface.

(c): Each triangle in the $RUCG$ locates in the interior of the Delaunay hole, or it locates on the boundary of the Delaunay hole and does not satisfy the Gabriel property.

In the three dimensional space, each triangle is incident to two Delaunay cells. Here, "the interior of the Delaunay hole" means that the triangle's two incident cells are both conflict cells, while "the boundary of the Delaunay hole" means that only one cell incident to the triangle is a conflict cell and the other is not.

Lemma 8. *$\partial(RUCG)$ is a circle.*

Proof. Since $RUCG$ is a $\Re K_2$, $\partial \Re K_2$ is a circle. As a result, $\partial(RUCG)$ is a circle. $\square$

$RUCG$ starts from an initial seed facet, and is grown by adding surface triangles one by one. But the seed facet should meet condition 3 of $RUCG$. In other words, the seed facet is a subset of $RUCG$.

Let S_k denote the current reconstructed surface, and p_{k+1} denote the $(k+1)th$ point needed to be inserted. The seed facet is estimated by the surface triangle which is nearest to p_{k+1} among others. First, locate the cell c containing p_{k+1} using the remembering stochastic walk; second, choose the knn (we choose $knn = 5$ based on experience) nearest points to p_{k+1} among the vertices of c and the vertices of its four neighboring cells; finally search the umbrellas in the surface incident to knn nearest points, and find the nearest facet $fnear$ to p_{k+1}. Besides, we can easily predicate whether p_{k+1} is inside or outside the surface mesh depending on whether c is labeled as internal or external.

The technique of inflating and sculpturing will be introduced thoroughly in Section 4.3. Here we only summarize the inflating and sculpturing pre-requirements used in algorithm "searching for the seed facet". See Figure 5 for illumination.

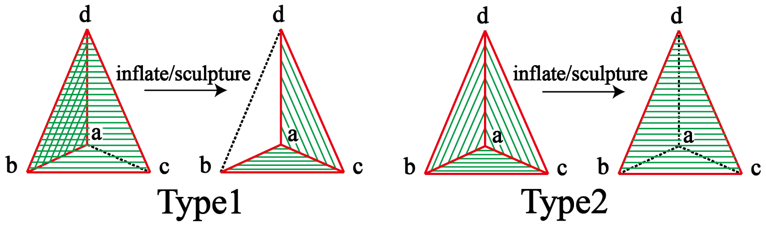


Figure 5: Inflating is to make an external tetrahedron become internal, while sculpturing is to make an internal tetrahedron become external. Meanwhile both inflating and sculpturing need to convert surface triangles to non-surface triangles and convert non-surface triangles to surface triangles. As presented, surface triangles are shaded, while non-surface triangles are not. Type1: conversion of two surface facets to the other two surface facets. Type2: conversion of three surface facets to the other one surface facet.

Pre-requirement of inflating used in seed facet searching *(Rule1)*:

1) The tetrahedron only has two facets in the surface and ac is not incident to any surface facet.

Pre-requirements of sculpturing used in seed facet searching *(Rule2)*:

1) The tetrahedron only has two facets in the surface and ac is not incident to any surface facet;

2) $ac < bd$.

However, our algorithm needs to use a list, named as *CandidatePoints*, to store the temporarily abandoned points. Here, "abandoned points" means that p_{k+1} cannot be inserted into S_k. But p_{k+1} may be inserted after other points have been inserted. As a result, we re-insert points stored in *CandidatePoints* when all other points have been inserted.

If "searching for the seed facet" returns false, this means we cannot find a seed facet when inserting p_{k+1}. Thus, we put p_{k+1} in *CandidatePoints*.

```
Algorithm 1: searching for the seed facet
Input: the nearest oriented surface facet $\widehat{fnear}$, $p_{k+1}$
Output: seed facet
1  $c\_fnear := Incident(\widehat{fnear})$ ;
2  $c\_mirror := Incident(mirror(\widehat{fnear}))$ ;
3  if $c\_fnear$ is conflict or $c\_mirror$ is conflict then
4      return seed facet $\widehat{fnear}$;
5  else
6      push umbrellas incident to $knn$ nearest points into a list $L$;
7  for each surface facet $\widehat{tmpf}$ in $L$ do
8      $c\_tmpf := Incident(\widehat{tmpf})$;
9      if $c\_tmpf$ is conflict then
10         return seed facet $\widehat{tmpf}$;
11     else if $p_{k+1}$ is inside and $c\_tmpf$ meets $Rule2$ then
12         sculpture $c\_tmpf$;
13         push new created facets into $L$;
14     else if $p_{k+1}$ is outside and $c\_tmpf$ meets $Rule1$ then
15         inflate $c\_tmpf$;
16         push new created facets into $L$;
17 return false;
```

Staring from the seed facet, we use a greedy algorithm to generate a pseudo *RUCG*, which may not be regular at all (Figure 6). Then, we utilize Lemma 8 to test the regularity of the pseudo *RUCG*. If regularity test fails, we still push p_{k+1} into *CandidatePoints*.

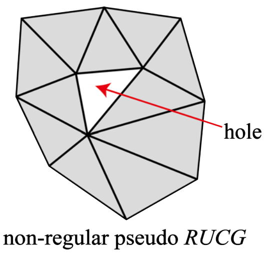


Figure 6: Examples of a non-regular pseudo $RUCG$. The shaded triangles are parts of the pseudo $RUCG$, while the white triangle is a hole in the pseudo $RUCG$.

4.2. Creating the constrained umbrella recursively

The main purpose of this step is to simultaneously update a 3D Delaunay triangulation and compute the constrained umbrella (CU) in a depth-first manner. It labels each newly created tetrahedron as internal or external respectively, and marks the boundary of all newly created internal tetrahedrons as the updated part of the reconstruction.

Let CU_v (CU for short) denote the constrained umbrella incident to vertex v.

Definition 9 (Constrained umbrella). Given a set of edges B, a constrained umbrella CU_v is an umbrella satisfying the following condition:

For an edge e, if $\forall e \in \partial(CU_v)$, we have $e \in B$. We say that the boundary of U_v is constrained by B, namely, CU_v.

In our method, we generate CU_v, which shares common boundary edges concerned with $RUCG$, to substitute $RUCG$ in the surface. If two adjacent tetrahedrons have different labels (one is internal and the other is external), the common facet shared by the two tetrahedrons must be a surface facet. If two adjacent tetrahedrons have the same label (both are either internal or external), the common facet cannot be a surface facet. As a result, if a newly generated tetrahedron is adjacent to a same labeled tetrahedron and the common facet is a surface facet, the surface facet must be deleted. On the contrary, if two tetrahedrons have different labels and the common facet has at least an edge $e \in \partial(RUCG)$, a new surface facet f will be created, i.e. , $f \in CU_v$.

Algorithm 2: searching for the *RUCG*

```
Input: seed facet $\widehat{seed}$, $p_{k+1}$, Delaunay hole $DH$;
Output: $RUCG$, $\partial(RUCG)$;
1  mark $\widehat{seed}$ visited;
2  push $\widehat{seed}$ into a list $L$;
3  while $L$ is not empty do
4      $l$ := pop $L$;
5      for $i = 0 : 2$ do
6          $\widehat{tmpf}$ := the $ith$ neighbor facets of $l$ in the surface;
7          if $\widehat{tmpf}$ has not been visited then
8              if $\widehat{tmpf}$ locates in the interior of $DH$ then
9                  mark $\widehat{tmpf}$ visited;
10                 push $\widehat{tmpf}$ into $L$;
11             else if $\widehat{tmpf}$ locates on $\partial(DH)$ then
12                 if $\widehat{tmpf}$ is against Gabriel property then
13                     mark $\widehat{tmpf}$ visited;
14                     push $\widehat{tmpf}$ into $L$;
15 //regularity test;
16 if $\exists \hat{f} \in DH$ has not been marked then
17     return false;
18 $B$ := the boundary of all marked facets;
19 if $B$ is not a circle then
20     return false;
21 else
22     $\partial(RUCG) = B$;
23     return all marked facets ($RUCG$);
```

Let c denote a Delaunay cell, and (c, i) denote an oriented facet $\hat{f}$, where $c = Incident(\hat{f})$ and $i (i = 0, 1, 2, 3)$ is the relative index of the vertex opposite to $\hat{f}$. Let $neighbor(c, i)$ denote the ith neighboring cell of c, and $\overrightarrow{(u, v)}$ denote an oriented edge, where u and v are vertices respectively.

In a single recursive step, only one tetrahedron will be created. As shown in Figure 7, *cnew* is the newly created tetrahedron adjacent to *c_li*, and

$cnew$ has a facet $(cnew, li)$ located on the boundary of the Delaunay hole. c is a conflict cell, while c_li is not. The recursive method starts from an initial facet, which means that the initial facet must have at least one edge $e \in \partial(RUCG)$, and label the first created tetrahdron as internal. Then the process searches around an oriented edge (denoted as $\overrightarrow{(vj1, vj2)}$ in the pseudocode, and $ii, vj1, vj2$ and li are in line with the right-hand rule in cell c) to find another pair composed of a conflict cell c' and a non-conflict cell c_li'. Nevertheless, if the oriented edge is a subset of the boundary of the $RUCG$, the label given for next created cell should be changed.

However, when inserting a point and implementing the recursive creation process, $cnew$ and c_li may have different labels. We assume that $cenw$ is internal and c_li is external. If we generate a new surface facet $(cnew, li)$, then the surface facet is opposite to the new sample point v and $(cnew, li) \notin CU_v$. Thus, a water-tight surface cannot be guaranteed. We name this phenomenon as umbrella singularity, and still push the point into $CandidatePoints$.

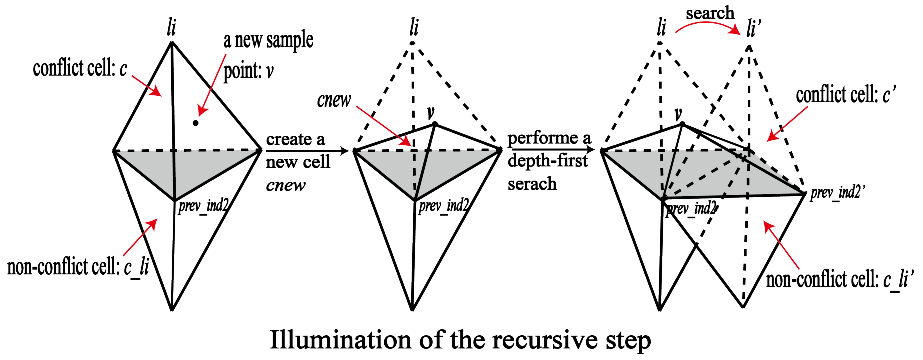


Figure 7: An illumination of creating a CU. During a single recursive step of creating a CU, (c, li) that is shaded in the figure denotes the boundary facet of the Delaunay hole. Moreover, if c_li and $cnew$ have different labels and the common facet between c and c_li is not a surface facet, umbrella singularity occurs.

The algorithm "recursive_creation_CU" is shown in Algorithm 3. The variable "*prev_ind2*" is used to prevent creating redundant tetrahedrons at the same location. Furthermore, all newly created surface facets make up the constrained umbrella.

Theorem 10. *When inserting a new sample point p, substituting the $CU(p)$ for the $RUCG(p)$ can always guarantee a water-tight surface mesh.*

Algorithm 3: recusive_creation_$CU(c, li, label, change, prev_ind2)$

```
Initialization: initial facet $(c, li)$, $RUCG$, $\partial(RUCG)$,
                $label := internal$, $change := true$, $prev\_ind2 := -1$;
1  create a new tetrahedron $cnew$, and give $cnew$ a label;
2  if $change$ is true and $cnew$ is internal then
3      //$(cnew, prev\_ind2) \in CU$;
4      create a surface facet $(cnew, prev\_ind2)$;
5  $c\_li := neighbor(c, li)$;
6  if $cnew$ and $c\_li$ have the same label then
7      delete surface facet; //delete $RUCG$;
8  else if umbrella singularity then
9      recover all states before inserting $p_{k+1}$;
10     exit recursive process;
11 mark $(c, li)$ processed; //$(c, li) \in \partial DH$
12 for $ii = 0 : 3$ do
13     if $ii == prev\_ind2$ or $ii == li$ then
14         continue;
15     $\overrightarrow{(vj1, vj2)}$ := the oriented edge opposite to $\overrightarrow{(ii, li)}$ in cell $c$;
16     search cells around $\overrightarrow{(vj1, vj2)}$ to find cell $c'$ (conflict) and $c\_li'$
         (not conflict) satisfying $c\_li' == neighbor(c', li')$;
17     $prev\_ind2'$ := the vertex index in $c'$ ($prev\_ind2'$, $vj2$, $vj1$ and $li'$
         are line with right-hand rule);
18     $label' := label$; $change' := false$;
19     if $\overrightarrow{(vj1, vj2)} \in \partial(RUCG)$ then
20         $label' := !label$; $change'$ := true;
21         if $cnew$ is internal then
22             create surface facet $(cnew, ii)$; //$(cnew, ii) \in CU$
23     if $(c', li')$ has not been processed then
24         recusive_creation_$CU(c', li', label', change', prev\_ind2')$;
25 return;
```

Proof. We use mathematical induction to prove this theorem. Let S_k denote the current reconstructed surface, p_k denote the kth point that has been inserted, and p_{k+1} denote the $(k+1)th$ point needed to be inserted.

In the beginning, we choose four affinely independent points to initialize a tetrahedron surface mesh, and the initial tetrahedron surface mesh is water-tight.

After inserting p_k, we assume S_k is water-tight, thus every edge in S_k is incident to two surface triangles. When inserting p_{k+1}, $RUCG(p_{k+1})$ is first computed (Algorithm 1 and Algorithm 2). According to Definition 7 and Lemma 8, $RUCG(p_{k+1})$ is a 2-manifold with boundary. According to the Definition 9, $CU(p_{k+1})$ is also a 2-manifold with boundary. Moreover, $CU(p_{k+1})$ and $RUCG(p_{k+1})$ share the same boundary. When we substitute the $CU(p_{k+1})$ for the $RUCG(p_{k+1})$ to obtain S_{k+1} (Algorithm 3), every edge in S_{k+1} is still incident to two surface triangles because $CU(p_{k+1})$ and $RUCG(p_{k+1})$ share the same boundary. On the other hand, the surface triangles incident to p_{k+1} form a closed fan because $CU(p_{k+1})$ is an umbrella. Thus, S_{k+1} is still water-tight.

Since both the initialization step and inserting points step have been performed, by mathematical induction, we can guarantee theorem 10 is true. $\square$

4.3. Inflating and sculpturing locally

Our recursive method can guarantee to extract a water-tight mesh directly. However, substituting the CU for the $RUCG$ may lead to surface facets violating Gabriel property. In order to let the surface mesh smoother, we develop the technique of inflating and sculpturing to filter non-Gabriel triangles locally. According to [37], six types of tetrahedrons can be added to a 2-manifold and the result is still a manifold. Similar to [37], our method only focuses on two types to guarantee a water-tight surface, as previously presented in Figure 5. However, when Type1 (Figure 5) is implemented in a water-tight surface, the edge (ac) opposite to the common edge (bd) shared by two surface facets should not be incident to any surface facet. Otherwise, a singular edge may occur.

From the perspective of energy function (1), substituting the CU for the $RUCG$ may fall into a local minimum because there would be existing facets of CU against the Gabriel property. Ohrhallinger et al. [10] use the method of minimizing edge length to overcome this difficulty, but this method fails when encountering dense input point clouds. But we can convert Ohrhallinger's method to a local scheme that chooses the minimal edge length between two opposing edges. As a result, if we sculpture a tetrahedron, not only do we need to avoid a singular edge, but also we want the new common edge (ac) to be shorter than the old one (bd) to avoid falling into a local minimum.

Now, we summarize the pre-requirements of inflating and sculpturing for every newly generated tetrahedron.

Pre-requirements of inflating a newly created tetrahedron ($Rule3$):

1) The tetrahedron is not a Gabriel simplex;
2) The tetrahedron only has two facets in the surface and ac is not incident to any surface facet.

Pre-requirements of sculpturing a new tetrahedron ($Rule4$):

1) The tetrahedron is not a Gabriel simplex;
2) The tetrahedron only has two facets in the surface and ac is not incident to any surface facet;
3) $ac < bd$.

4.4. Inserting isolated vertices

Isolated vertices are those vertices that are not incident to any surface facet. Isolated vertices occur when $RUCG$ contains umbrellas. The example in Figure 8 shows that two isolated vertices exist after the $RUCG$ is substituted by the CU.

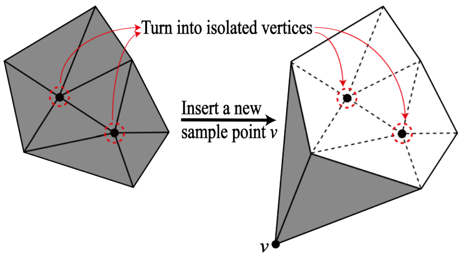


Figure 8: The black dots in the dotted red circle represent the isolated vertices. Isolated vertices occur when $RUCG$ contains umbrellas.

Inserting isolated vertices is the same as inserting sample points, but isolated vertices have been in the structure of the 3D Delaunay triangulation. As a result, there will be no new tetrahedrons created when inserting isolated vertices. We push isolated vertices into a list $isolVerList$, and re-insert them after a new sample point has been processed. For each isolated vertex $isol$, the process is summarized as follows:

1) Find knn nearest points of $isol$.
2) Search for the umbrellas incident to knn nearest points and find the nearest oriented surface facet $\widehat{fnear}$ to $isol$.
3) Search for the seed facet $\widehat{fseed}$ via $\widehat{fnear}$.
4) Search for the $RUCG$ and confirm the $\partial(RUCG)$.
5) According to the $\partial(RUCG)$, create surface facets incident to $isol$ in the 3D Delaunay triangulation, and all these facets make up the CU.

However, inserting a vertex v (whether it is a sample point or an isolated vertex) may cause new isolated vertices, depending on whether the $RUCG(v)$ contains umbrellas. As a result, processing $isolVerList$ may fall into an endless loop. For example, inserting an isolated vertex $isol1$ causes an isolated vertex $isol2$, and then inserting $isol2$ causes $isol1$. As a consequence, we should limit the iterative times to avoid an endless loop. Inserting isolated vertices after a new sample point insertion consumes much time. However, for extremely dense point sets, there is no need to process isolated vertices, and thus a similar result can be acquired using much less time.

Figure 9 shows a comparison of two reconstructions of a sparse point set: one neglects isolated vertices insertion, while the other does not. Note that isolated vertices stored in $isolVerList$ have been in the structure of the 3D Delaunay triangulation. However, the points stored in $CandidatePoints$ are not in the structure of the Delaunay triangulation.

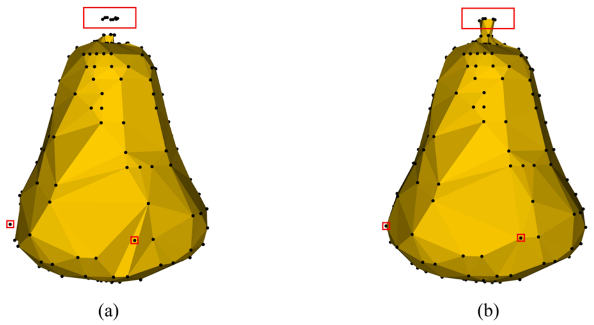


Figure 9: Reconstruction of a Pear (235 points): (a) Isolated vertices insertion is ignored. (b) Isolated vertices insertion is taken into consideration.

4.5. An overview of the proposed algorithm

Now, we summarize our algorithm (Algorithm 4).

Algorithm 4: The whole algorithm

```
1  Initialize 4 affinely independent points;
2  for the remaining point $p$ in $P$ do
3      Search for the seed facet, and if the process returns false, push $p$
         into the $CandidatePoints$; //Algorithm 1
4      Search for the $RUCG(p)$, and if the process returns false, push
         $p$ into the $CandidatePoints$; //Algorithm 2
5      Create the $CU$ recursively, and if singularity occurs, push $p$ into
         the $CandidatePoints$; //Algorithm 3
6      for every new created surface facet $\widehat{f}$ do
7          $ctemp := Incident(\widehat{f})$;
8          if $ctemp$ obeys $Rules3$ then
9              inflate $ctemp$;
10         if $ctemp$ obeys $Rules4$ then
11             sculpture $ctemp$;
12         //This can be ignored for large point sets;
13         Insert points stored in $isolVerList$;
14 Re-insert points stored in $CandidatePoints$;
```

Re-inserting points stored in *CandidatePoints* is to insert these points again when all other points have been processed, and the permutation of the insertion order should be randomized for the benefits of acceleration. However, some points cannot be inserted in any case, and therefore, an upper bound of iterative times needs to be set to prevent an endless loop.

5. Results and discussion

Our algorithm initializes the solid tetrahedron as the initial surface mesh, and updating the surface mesh by substituting the constrained umbrella for the regular umbrella-covered graph does not change the genus. As a result, our algorithm is designed to reconstruct this type of surface that is genus zero and has no boundary. In practical applications, sample conditions and noises have a great impact on the quality of the reconstructed surface, as well as the smoothness. In this section, we give some experiments on different sample conditions, and compare the reconstruction results with some well-known Delaunay-Voronoi algorithms. The proposed algorithm is implemented on an Intel 3.70GHz i3 CPU and 16GB RAM PC, and Visual Studio 2012 is used to compile the source code in the release model. Other algorithms prepared for comparison are implemented in the same environment.

Uniform and dense point clouds are relatively easy to reconstruct a desirable surface mesh. Figure 10 presents some reconstruction results of the proposed algorithm from uniform point sets in different sizes. The algorithm is able to reconstruct very fine details of the models, for example, the furs of the Bunny, the wrinkles of Caesar and the curled hair of Bimba.

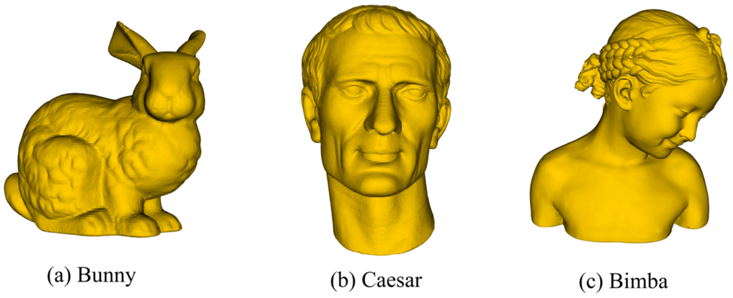


Figure 10: (a) Bunny (35,947 points, 4.149s), (b) Caesar (387,900 points, 67.626s), and (c) Bimba (502,693 points, 77.913s).

Figures 9 and 11 present the robustness of our algorithm on the non-uniform point sets, and compare the results with Peethambaran's algorithm [8]. The point cloud of the Pear (Figure 9) is highly non-uniformed. The presented point cloud in Figure 11 has a higher point density in the ears, nose and eye regions, and the distribution is also non-uniform. Due to the non-uniformness, it is likely for most algorithms to create undesired surface facets that may hardly match the original one. As we can see, uncorrected surface facets are generated in Peethambaran's algorithm, while the output of our algorithm yields a better result. Furthermore, our algorithm uses 0.234s to reconstruct this non-uniform Mannequin, while Peethambaran's algorithm uses 2.062s to reconstruct an undesired Mannequin.

Noise that is inevitable in practical use is another challenge for surface reconstruction. Delaunay-Voronoi based surface reconstruction algorithms are sensitive to noise. However, the robust cocone [9] proposed by Dey is a prior Delaunay-Voronoi based algorithm in the presence of noise. Figure 12 shows the test of our algorithm and the robust cocone on little noisy point sets. Although the output of the robust cocone is smoother, it looses many

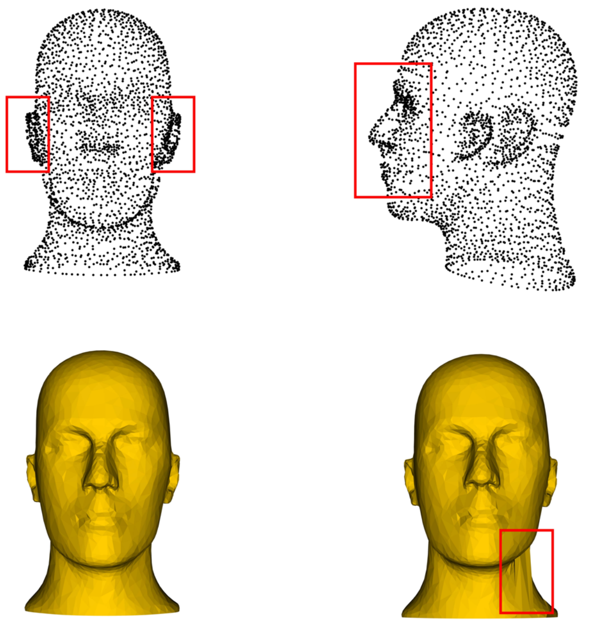
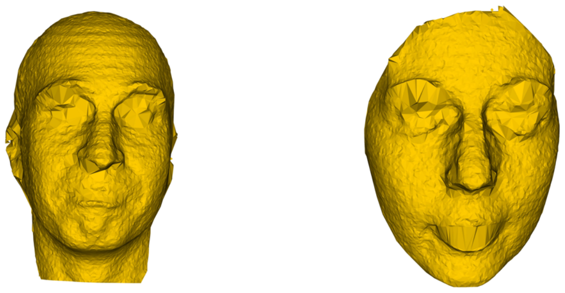


Figure 11: Reconstruction of a non-uniform Mannequin (2,527 points). The above is the point set, and the below is the comparison.

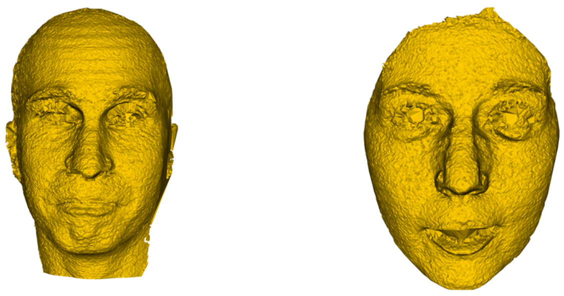
details, such as the eyes and the mouth, due to dropping many points. On the contrary, though our algorithm also drops some points, it preserves the fine features well.

Most Delaunay-Voronoi based surface reconstruction algorithms need to compute a Delaunay triangulation or a Voronoi diagram first (some need both of them), and then extract a surface graph. Unlike them, our algorithm simultaneously updates a spatial Delaunay triangulation and outputs a surface graph. Thus, the time complexity is mainly dominated by computing the Delaunay triangulation. As a result, it is faster than most algorithms. Except the running time of the male face in Figure 12 (dropping too many sample points of this model reduces the running time of the robust cocone), our algorithm is faster than others.

For off-line reconstruction, we strongly recommend inserting points randomly. Randomization not only helps to support fast point location, but also



(a) Robust cocone



(b) Ours

Figure 12: Reconstruction of two human faces with little noise (left: 24,859 points, right: 16,599 points). (a) Robust cocone (left: 10.156s, right: 6.225s), and (b) Ours (left: 10.343s, right: 3.869s).

can optimize the running time of the Bowyer-Watson's algorithm. If we exclude the point location step, the time complexity of the three-dimensional Bowyer-Watson algorithm is $O(n^2)$, where n is the size of the point set. In fact, most global Delaunay-Voronoi based surface reconstruction algorithms use the quick hull algorithm [38] to compute a Delaunay triangulation. The quick hull algorithm is a variation of the randomized incremental algorithms for computing a Delaunay triangulation. In our program, we only use the array to store geometric and topological information, such as the vertices of each Delaunay cell, the vertices of each surface facet, and the adjacent relationships. Thus, a quick access to this information can be acquired in a short time. We do not use the conflict graph [38] to store the information related to the uninserted points, which may be unreliable for online reconstruction.

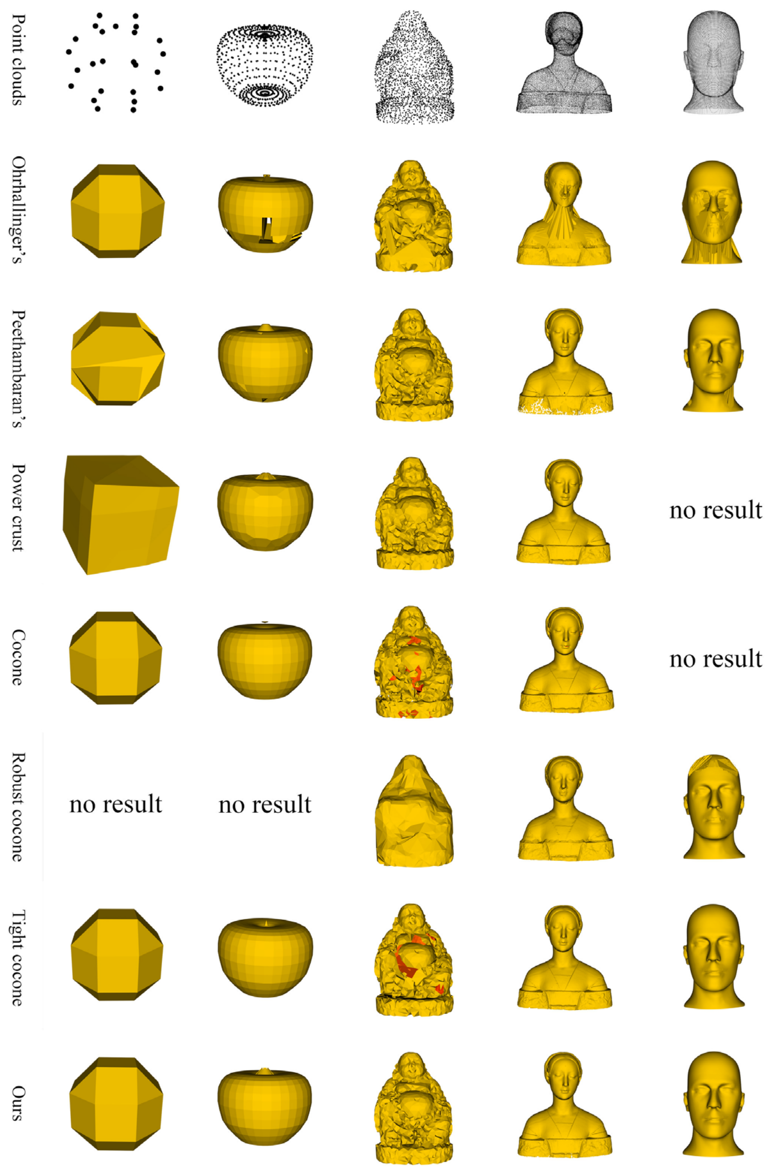


Figure 13: Reconstruction of different models. From the left to the right are Rhombicuboctahedron, Apple, Buddha, Laurna and Manequin (dense and uniform) respectively. Hole regions in the surface are marked with red.

Another advantage of our algorithm is that no parameters are needed, and it can reconstruct a surface from only points. Some algorithms, such as poisson surface reconstruction [14] and ball pivoting [1], require the user to input parameters, which have a strong effect on the ultimate results. Especially reconstructing point clouds lack of normals challenges the task of normal estimation when the input data have imperfections [39].

Though our algorithm is heuristic, experiments show that it can be adapted to point sets of discrete objects, highly uniform point sets, sparse unorganized point sets and dense point sets. Figure 13 and table 1 testify the superiority of our algorithm over others. It can achieve a balance between different requirements such as fine feature preservation, robustness to little noise, and less reconstruction time.

As presented in Figure 13 and table 1, Ohrhallinger's algorithm [10] works well for sparse point sets, but performs poorly for dense point sets. Especially when reconstructing large point sets, their algorithm produces many undesired surface facets. Furthermore, the running time of their algorithm is a little longer than others. Peethambaran's algorithm [8] fails to reconstruct the Rhombicuboctahedron, which is a discrete object. When reconstructing the Laurana, the pedestal has been separated into many pieces. The power crust [7] also fails to reconstruct the Rhombicuboctahedron. On the other hand, the output of the power crust is not a triangular mesh and introduces extra points. Thus, some vertices of the output surface mesh are not the sample points at all. Especially when reconstructing the Apple, the texture of the output surface mesh is not as good as others. Moreover, there is no result output when reconstructing the dense and uniform Mannequin, due to an adjacent error. The cocone [6], the robust cocone [9] and the tight cocone [17] fail to reconstruct sparse sampled points. For reconstruction of the Buddha, both the cocone and the tight cocone leave holes on the surface mesh, while the robust cocone fails to interpolate many points.

Though our algorithm may also produce some unwanted surface facets, it can achieve a balance in those requirements. The advantages of our algorithm are summarized as follows:

1) It is non-parametric and only takes points as input;

2) It can reconstruct discrete objects and be robust to little noise;

3) It can reconstruct smooth objects from both sparse and dense points;

4) It simultaneously outputs a triangular water-tight surface mesh with a spatial Delaunay triangulation;

Table 1: Running time of different algorithms

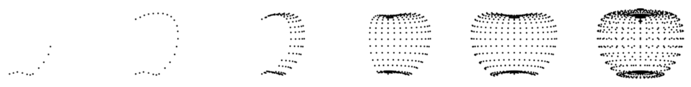
Methods	Running Time (s)		
	Buddha	Laurana	Manequin
Ohrhallinger's	3.4	48.94	183.8
Peethambaran's	2.047	2.748	4.184
Power Crust	0.446	7.222	/
Cocone	0.78	13.31	/
Robust cocone	0.687	14.7	38.673
Tight cocone	0.733	11.398	26.457
Ours	0.261	6.708	8.239

5) Its complexity is only related to the 3D Delaunay triangulation of the points, and thus it can reconstruct a surface graph in a shorter time.

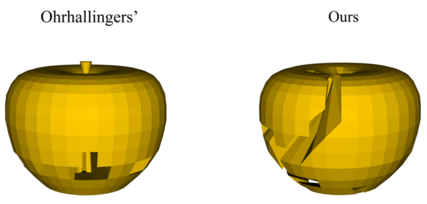
6. Conclusion and future work

In this paper, an incremental and dynamic surface reconstruction algorithm is proposed to output a water-tight surface graph that can be guaranteed. The main novelty of this algorithm is that the surface graph updates with a spatial Delaunay triangulation of the points processed up to now. Another contribution of our work is to implement the technique of local inflating and sculpturing to smooth the reconstructed surface and take isolated vertices into consideration to avoid dropping too many points. We have done some experiments on different sampling conditions to prove the robustness of our algorithm. Meanwhile the running time of our algorithm is faster than others.

However, our algorithm still has some drawbacks. Most Delaunay-Voronoi based algorithms fail when encountering degenerate cases. Figure 14 illustrates an example of inserting points in a badly arranged order. The drawback can be avoided by randomizing the insertion order. Guibas et al. [40] give a randomized incremental algorithm of constructing Delaunay triangulation. They give provable guarantees for randomization. Though we do not give any provable guarantees, randomization not only accelerates the process of surface reconstruction, but also enhances the reconstruction results. However, a global randomization is a little restrictive for incremental algorithms. In recent years, a biased increment construction of the Delaunay triangulation [41, 42] has been studied by researchers. Derived from this idea, we recommend using a buffer to store a biased randomized insertion order to avoid degeneracy, and we wish to combine a biased randomized insertion order with other theoretically proved techniques to overcome the degenerate cases in the future work.



(a) The permutation of the inseration order



(b) Reconstruction results

Figure 14: Reconstruction of Apple in a badly arranged point insertion order: (a) the permutation of the point insertion order; (b) reconstruction results.

Acknowledgement

All models are rendered by VTK [43]. Furthermore, the models are available online, including Standford 3D scanning repository, VRML models, and Aim@shape. Especially, we appreciate that Dey and Jiju Peethambaran offer their compiled programs.

References

[1] 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** (200), no. 4, 349–359.

[2] M. Berger, J. A. Levine, L. G. Nonato, G. Taubin, and C. T. Silva, *A benchmark for surface reconstruction*, Acm Transactions on Graphics **32** (2013), no. 2, 20.

[3] H. W. Lin, C. L. Tai, and G. J. Wang, *A mesh reconstruction algorithm driven by an intrinsic property of a point cloud*, Computer-Aided Design **36** (2004), no. 1, 1–9.

[4] N. Amenta and M. Bern, *Surface reconstruction by Voronoi filtering*, Discrete and Computational Geometry **22** (1999), no. 4, 481–504.

[5] N. Amenta, M. Bern, and D. Eppstein, *The Crust and the β-Skeleton: Combinatorial curve reconstruction*, Graphical Models and Image Processing **60** (1998), no. 2, 125–135.

[6] N. Amenta, S. Choi, T. K. Dey, and N. Leekha, *A simple algorithm for homeomorphic surface reconstruction*, International Journal of Computational Geometry and Applications **12** (2000), 213–222.

[7] N. Amenta, S. Choi, and R. K. Kolluri, *The power crust, unions of balls, and the medial axis transform*, Computational Geometry **19** (2000), no. 2-3, 127–153.

[8] J. Peethambaran and R. Muthuganapathy, *Reconstruction of water-tight surfaces through Delaunay sculpting*, Computer-Aided Design **58** (2015), 62–72.

[9] T. K. Dey and S. Goswami, *Provable surface reconstruction from noisy samples*, Computational Geometry **58** (2006), 124–141.

[10] S. Ohrhallinger, S. Mudur, and M. Wimmer, *SMI 2013: Minimizing edge length to connect sparsely sampled unstructured point sets*, Computers and Graphics **37** (2013), no. 6, 645–658.

[11] D. Boltcheva and B. Lévy, *Surface reconstruction by computing restricted Voronoi cells in parallel ☆*, Computer-Aided Design (2017).

[12] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva, *A survey of surface reconstruction from point clouds*, Computer Graphics Forum **36** (2017).

[13] H. Hoppe, T. Derose, T. Duchamp, J. Mcdonald, and W. Stuetzle, *Surface reconstruction from unorganized points*, ACM, 1992.

[14] M. Kazhdan, M. Bolitho, and H. Hoppe, *Poisson surface reconstruction*, Eurographics Symposium on Geometry Processing (2006), 61–70.

[15] W. E. Lorensen and H. E. Cline, *Marching cubes: a high resolution 3D surface construction algorithm*, ACM siggraph computer graphics (1987), 163–169.

[16] H. K. Zhao, S. Osher, and R. Fedkiw, *Fast surface reconstruction using the level set method*, Variational and level set methods in computer vision (2001), Proceedings. IEEE Workshop on, 194–201, 2001.

[17] T. K. Dey and S. Goswami, *Tight cocone: a water-tight surface reconstructor*, Proceedings of the eighth ACM symposium on Solid modeling and applications, ACM (2003), 127–134.

[18] T. K. Dey R. Dyer, and L. Wang, *Localized Cocone surface reconstruction*, Computers and Graphics **35** (2011), no. 3, 483–491.

[19] J. D. Boissonnat, *Geometric structures for three-dimensional shape representation*, ACM Transactions on Graphics **3** (1984), no. 4, 266–286.

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

[21] T. K. Dey, K. Li, E. A. Ramos, and W. Rephael, *Isotopic reconstruction of surfaces with boundaries*, Symposium on Geometry Processing (2009), 1371–1382.

[22] U. Adamy, J. Giesen, and M. John, *Surface reconstruction using umbrella filters*, Computational Geometry Theory **21** (2002), 63–86.

[23] U. Adamy, J. Giesen, and M. John, *New techniques for topologically correct surface reconstruction*, Conference on Visualization, IEEE Computer Society Press, 373–380, 2000.

[24] R. Chaine, *A geometric convection approach of 3-D reconstruction*, Eurographics Symposium on Geometry Processing (2003), 218–229.

[25] R. Allgre, R. Chaine, and S. Akkouche, *Convection-driven dynamic surface reconstruction*, International Conference on Shape Modeling and Applications (2005), 33–42.

[26] S. Yu and M. Lhuillier, *Incremental reconstruction of manifold surface from sparse visual mapping*, International Conference on 3d Imaging (2012), 293–300.

[27] S. W. Cheng, T. K. Dey, and J. Shewchuk, *Delaunay mesh generation*, Computational Mathematics and Mathematical Physics **50** (2013), no. 1, 38–53.

[28] D. W. Matula and R. R. Sokal, *Properties of gabriel graphs relevant to geographic variation research and the clustering of points in the plane*, Geographical Analysis **12** (1980), no. 3, 205–222.

[29] A. Bowyer, *Computing Dirichlet tessellations*, Computer Journal **24** (1981), no. 2, 162–166.

[30] D. F. Watson, *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*, Computer Journal, **24** (1981), no. 2, 167–172.

[31] R. Chen and C. Gotsman, *Localizing the Delaunay triangulation and its parallel implementation*, Ninth International Symposium on Voronoi Diagrams in Science and Engineering (2012), 24–31.

[32] J. R. Shewchuk, *Robust adaptive floating-point geometric predicates*, Scg Proceedings of the Twelfth Annual Symposium on Computational Geometry (1996), 141–150.

[33] CGAL, *Computational Geometry Algorithms Library.*

[34] O. Devillers, S. Pion, and M. Teillaud, *Walking in a triangulation* (2001), 106–114.

[35] F. Cazals and J. Giesen, *Delaunay triangulation based surface reconstruction*, 2006.

[36] J. Giesen and M. John, *Surface reconstruction based on a dynamical system* †, Computer Graphics Forum **21** (2002), 363–371.

[37] V. Litvinov and M. Lhuillier, *Incremental solid modeling from sparse and omnidirectional structure-from-motion data*, British Machine Vision Conference (2013), 61.1–61.11.

[38] C. B. Barber and D. P. Dobkin, *The quickhull algorithm for convex hulls*, ACM Transactions on Mathematical Software **22** (1998), no. 4, 469–483.

[39] S. Xiong, J. Zhang, J. Zheng, J. Cai, and L. Liu, *Robust surface reconstruction via dictionary learning*, ACM Transactions on Graphics **33** (2014), no. 6, 201.

[40] L. J. Guibas, D. E. Knuth, and M. Sharir, *Randomized incremental construction of Delaunay and Voronoi diagrams*, Algorithmica **7** (1992), no. 1, 381–413.

[41] N. Amenta, S. Choi, and G. Rote, *Incremental constructions con BRIO*, Nineteenth Symposium on Computational Geometry (2003), 211–219.

[42] O. Devillers, *Delaunay triangulation and randomized constructions*, Springer New York, 2014.

[43] Kitware, *The VTK user's guide*, 2010.

Research Center of Biomedical Engineering
Graduate School at Shenzhen, Tsinghua University
Shenzhen, 518055, China
Department of Biomedical Engineering, Tsinghua University
Beijing, 100084, China
E-mail address: fur15@mails.tsinghua.edu.cn

Research Center of Biomedical Engineering
Graduate School at Shenzhen, Tsinghua University
Shenzhen, 518055, China
Department of Biomedical Engineering, Tsinghua University
Beijing, 100084, China
E-mail address: wc984145602@126.com

Research Center of Biomedical Engineering
Graduate School at Shenzhen, Tsinghua University
Shenzhen, 518055, China
Department of Biomedical Engineering, Tsinghua University
Beijing, 100084, China
E-mail address: chenrq12@mails.tsinghua.edu.cn

Research Center of Biomedical Engineering
Graduate School at Shenzhen, Tsinghua University
Shenzhen, 518055, China
Department of Biomedical Engineering, Tsinghua University
Beijing, 100084, China
E-mail address: fuyf14@mails.tsinghua.edu.cn

Research Center of Biomedical Engineering
Graduate School at Shenzhen, Tsinghua University
Shenzhen, 518055, China
E-mail address: wuj@sz.tsinghua.edu.cn