On Surface Reconstruction: A Priority Driven Approach

Xiaokun Li [a] Chia-Yung Han[b] William G. Wee[c]

[a]*DCM Research Resources LLC, Germantown, MD 20874, USA*
[b]*Department of Computer Science, University of Cincinnati, OH 45221, USA*
[c]*Department of Electrical & Computer Engineering, University of Cincinnati, OH 45221, USA*

Abstract

To reconstruct an object surface from a set of surface points, a fast, practical, and efficient priority driven algorithm is presented. The key idea of the method is to consider the shape changes of an object at the boundary of the mesh growing area and to create a priority queue to the advanced front of the mesh area according to the changes. The mesh growing process is then driven by the priority queue for efficient surface reconstruction. New and practical triangulation criteria are also developed to support the priority driven strategy and to construct a new triangle at each step of mesh growing in real time. The quality and correctness of the created triangles will be guaranteed by the triangulation criteria and topological operations. The algorithm can reconstruct an object surface from unorganized surface points in a fast and reliable manner. Moreover, it can successfully construct the surface of the objects with complex geometry or topology. The efficiency and robustness of the proposed algorithm is validated by extensive experiments.

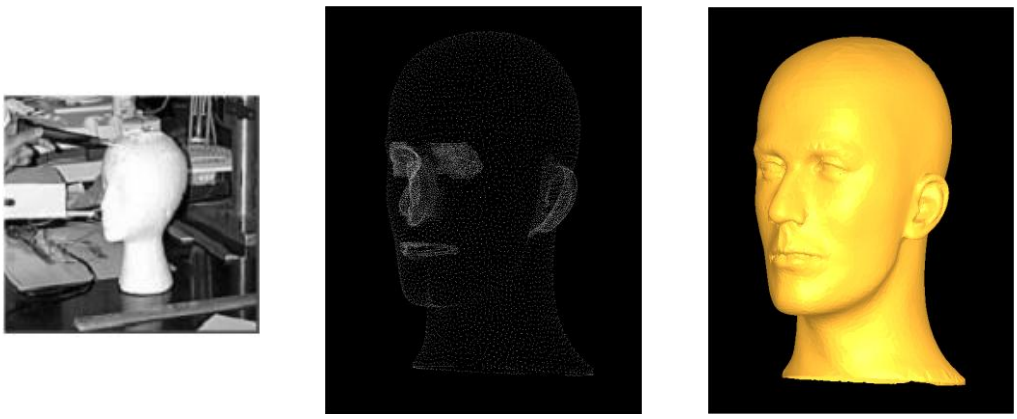
Keywords: Surface and solid modeling, Computational geometry, Surface reconstruction, Triangulation, Meshing

1. Introduction

Three dimensional (3D) object surface reconstruction and modeling plays a very important role in many fields of science and engineering including computer graphics, computer vision, medical imaging, virtual reality, CAD/CAM, and reverse engineering [1],[2]. For example, if people want to build a geometrical model to a 3D object for reproduction, the only way is to scan the object with a digital-data acquisition device and perform surface reconstruction to get its geometrical model. Since surface-based representation of a 3D object is crucial not only in data rendering, but also in 3D object analysis, modeling, and reproduction, many surface reconstruction algorithms have been proposed in recent years. In 3D object reconstruction and display, 3D surface points can be collected either by tactile methods such as Coordinate Measuring Machines (*CMM*), or via non-contact methods, such as magnetic field measurement machines and optical range scanners. After surface point acquisition, the next step of 3D object reconstruction will be data fusion (patch registration and integration) to translate the surface point sets captured at different view-angles into a common coordinate system, known as the World Coordinate System (*WCS*), and merge the overlapping points of any two neighboring data sets. The last step of the process is surface reconstruction (surface meshing/triangulation) and rendering. Fig. 1 shows the complete procedure of 3D object reconstruction. The surface reconstruction of an object discussed here can be stated as follows:

Given a set X *of surface points* x_i $(x_i \in X)$ *of a 3D object in* R^3 *space, a set of compact, connected, orientable two-dimensional manifolds are constructed to a surface S, so that the mapping function* $f : X \to Y$ *has* $S \approx S^{'}$, *where* $S^{'}$ is the real surface of the object.

The goal of object surface reconstruction is to generate a surface S that approximates the real surface S' of an object, whose CAD model is not available, does not exist, or cannot be obtained for whatever reason. The approximation should be as accurate as possible. A surface of an object without boundary is called *closed surface*, otherwise, called *bounded surface*. The efficiency and accuracy of the approximation are the major concerns in surface reconstruction.



(a) Acquired object **(b) Data fusion** **(c) Surface reconstruction and rendering**

Fig. 1. Example on 3D Object Reconstruction

2. Related Work

Since object surface reconstruction is very important in many industrial applications, huge research efforts have been made in recent years. In this section, a brief literature review of the existing surface reconstruction algorithms is given. According to their philosophy, these methods can be categorized roughly into three classes: the spatial subdivision scheme based, the surface distance function based, and the incremental mesh growing based.

The key idea of the spatial subdivision scheme can be briefly summarized as follows: First, it subdivides the bounding space defined by the input point set of an object into disjoint cells, and then reconstructs a surface from these cells. The methods in [4-12] are in the class of the spatial subdivision scheme. Among them, one typical method is called marching-cube algorithm proposed by Lorensen [4] in which a travel strategy was designed to construct an object surface from the selected cells by referring to a predefined lookup table. Another well-known method, called α-shapes, proposed by Edelsbrunner and Mucke [6] creates and represents an object surface through a finite set of some selected surface points for different levels of detail. The biggest problem of the α-shapes is that different α's are required to be defined at different places of an object. Guo, et al. [30] and Attali [7] improve the quality of the α-shape-based surface by applying visibility algorithms, and then normalize the generated meshes respectively. The Delaunay based "sculpting" method proposed by Boissonnat [8] tetrahedrizes the surface points and progressively removes the non-surface triangles of the tetrahedra according to the predefined geometric criteria. The methods developed in [5] and [9], have the similar principles, but use different criteria. Amenta et al. [10] propose a Voronoi filtering based method to create an interpolating shape from the sample points, which is called "crust". The method is based on 3D Voronoi diagrams and Delaunay triangulation to generate the object surface. The simplified and more efficient versions of the "crust" are proposed in [11] and [12]. The advantages of the spatial subdivision scheme based methods are their surface reconstruction can reach a global optimum and multi-resolution representation can be achieved very easily. Their main drawback is their reconstructed surface is not very accurate and their computational cost becomes expensive when people want to recover more details of the surface.

Surface distance function based methods first employ a least-square operation to generate an initial surface by estimating a local tangent plane for each input surface point. The signed distance of each surface point to the corresponded local tangent plane is then computed. These signed distances are used to build a signed distance function for all input surface points. The object surface is created by triangulating the zero-set of the signed distance function. The method proposed by Hoppe et al. [13] is a typical method of this category. Curless and Levoy [14] propose the similar principles, but they compute a tangent plane for each surface point, divide the 3D space into voxel grids, and build a volumetric function to these voxel grids. Radial Basis Functions, called RBFs, have become the favorite technique [33-36] in recent years. The recent developments in this category also include the level set based methods [19, 20], [31], or called surface deformation based methods, which construct an initial surface to the object, and then deform the initial surface by minimizing the energy of the level set. The deformation based methods are fast and robust against noise if the initial guess is not too far from the real shape. One problem of the methods in this category is that they are sensitive to the outlier noise due to the very reliance on the distance transform. However, some new methods [37-42] have been developed recently to process the noisy data sets with stratified reconstruction results. Another problem is that their processing speed is slow.

Incremental mesh growing (surface reconstruction) methods construct an object surface by using the surface-oriented properties of the input surface points. The methods start the meshing by selecting a single triangle as initial mesh area and grow the mesh area along the advanced front of the mesh area. All the surface points are the vertices of the constructed triangles and the mesh growing will be stopped when all surface points are processed. Unlike the methods in the classes of the subdivision-based and the surface distance-based, the incremental–based methods take every surface point as triangle vertex. Therefore, they will keep the details of the surface of the object and the reconstructed surface is more accurate. In recent years, a lot of effort has been made in this direction. Gopi, et al. [16] propose a method to triangulate the surface points with a localized Delaunay triangulation scheme by projecting local surface points onto a local 2-D plane and to perform 2-D Delaunay triangulation on the plane. Bernardini et al. [17] designed a straightforward triangulation method, called the ball-pivoting algorithm (BPA), to generate a surface from the input surface points by pivoting a virtual ball with a predefined radius along the advanced front of the mesh area and constructing a new triangle to each boundary edge in sequential order. Under some constrains to input data (point cloud), BPA guarantees to construct a correct surface. A graph search based approach is proposed by Mencl and Muller [15], which uses Euclidean Minimum Spanning Tree as an initial wireframe. Grossno and Angle [18] develop a Spiraling-Edge based triangulation method to create object surface. A surface reconstruction algorithm driven by an intrinsic property is proposed by Lin, et al. [49]. By combining local and global topological operations, Huang and Menq [48] propose a surface reconstruction and optimization algorithm. The advantage of the incremental mesh growing methods is their high accuracy for surface reconstruction. However, these methods are not resilient to noise and the processing speed of most of them is not fast enough.

There are also some approaches which combine the surface reconstruction methods from different categories to achieve better results, such as the method proposed by Kuo and Yau [43]. After surface reconstruction, to efficiently handle the large number of generated triangular meshes, surface simplification methods [23-25] have been developed to achieve an appropriate rendering frame rate on graphics hardware after surface reconstruction. Hoppe [26, 27], Kim and Lee [28], Pajarola and DeCoro [29] propose efficient methods on progressive mesh for representing different levels of detail. On the other hand, surface interpolation methods [44-46] have also been developed for high-fidelity rendering and surface modeling.

For the problems in surface reconstruction, the factors of processing speed, robustness, reliability, easiness on implementation, etc. are the major concerns of common users. In this paper, a new, fast, and practical surface reconstruction method, which is designed for dealing with these concerns and called priority driven based incremental surface reconstruction, is proposed and validated. Experimental results show that the proposed algorithm can deal with these problems efficiently while providing satisfied reconstruction results.

3. Algorithm Description

In this section, the proposed algorithm is described in detail through seven subsections. First, the overview of the algorithm is introduced in Section 3.1. Then, the data structure and terminology used in the algorithm are given in Section 3.2. The principle of the proposed priority driven strategy on mesh growing is described in Section 3.3. For better understanding, the details of the algorithm are described in the subsequent sections. The initialization strategy of the mesh growing is described in Section 3.4. In Section 3.5, a new triangulation method is developed to support the priority driven strategy. The topological operations used in meshing growing are presented in Section 3.6. At last, the advantages and limitations of using the algorithm are given in Section 3.7.

3.1. Overview

The algorithm input is a set of surface points and the output is a set of triangles. The main steps of the algorithm can be briefly described as follows: First, all input points are held in a hash table for both point indexing and the nearest neighbor searching. Then, a starting triangle is chosen at a ‘flat’ place of the object as the initial mesh area. The edges of the starting triangle constitute the initial advanced front of the mesh growing area. A priority driven strategy for mesh growing is then followed to sweep the advanced front ahead in an effective way. At each step of mesh growing, a new triangle constructed by a triangulation operation will be added into a triangle list. The mesh keeps growing along the object surface until all unmeshed surface points are processed. Four topological operations are defined and applied to support the mesh growing. The sketch of the proposed algorithm and some preliminary results can be found in [47]. The main contributions of the approach include a priority driven (“flat”-to-“sharp”) strategy, which forces the mesh growing process to propagate in an efficient and reliable fashion, four topological operations to guarantee the topological correctness of the generated triangles in different situations, and a real-time trian-

gulation method to process the data with non-uniform/varying sampling rate, complex geometry and/or topology. Compared with other conventional methods, such as the methods in [15-18], the proposed algorithm can reconstruct object surface in real time, has fewer limitations to input data, and works more robust to 3D objects with un-uniform sampling rate, closed and/or open (bounded) surface. It can even work well for the very challenging situation that two surfaces of an object are very close to each other.

3.2. Data Structure and Terminology

Efficiently finding a point and searching its neighborhood points in a 3D point set is an important task in object surface reconstruction. In the proposed algorithm, hash table based data structure proposed in [3] and [13] was adopted for fast neighbor search. To build a hash table, an axis-aligned bounding box is firstly set for the input data set. The points in the bounding box are then partitioned through a cubical grid whose size is decided by a pre-calculated dense value ρ of the surface points, and indexed into sets corresponding to the cube where they are located in. Then, a hash table is built for accessing these sets through the cube indices. For the nearest neighbor searching, the standard search strategy described in [21] is utilized. Compared with the most popular data structures, such as kd-tree [21] and bd-tree [22], it has been proven that the time complexity of finding a given point in a surface point set represented by hash table can be effectively reduced by a factor of $\log N$ (N is the size of input points). For the k-nearest-neighbor problem, the searching time of using hash table becomes $O(k)$, a constant time.

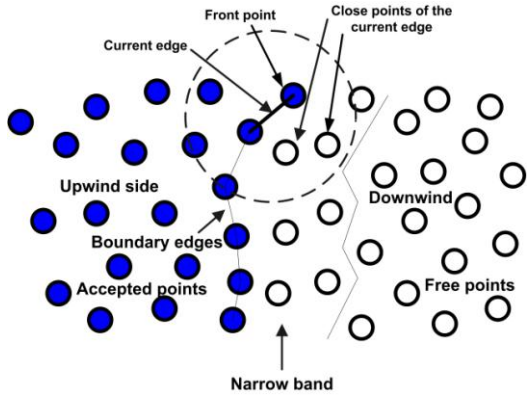


Fig. 2. Illustration of mesh growing

In any stage of the proposed algorithm, each surface point will be tagged as one of the four following states: *free*, *close*, *accepted*, and *refused*. The *free* points are the points that are not processed and included into the mesh (triangle) list. All input points are tagged as *free* at the initial stage. The *free* points close to the current front edge of the mesh are named *close* points. Points lay on the mesh areas are named *accepted* points. The noisy points (noise) of the input data are tagged as *refused* points. The boundary of a triangle is called *edge*. The edges on the advanced front are assigned as *front edges*. The *accepted* points on front edges are also named as *front* points. The triangle of the reconstructed surface is called *face*. The points chosen for constructing a new triangle are called *reference* points. These terms are illustrated in Fig. 2.

During the meshing, one invariant must be maintained to guarantee the topology correctness of the reconstructed surface. The invariant is any edge of the triangular surface can only be connected by two triangles or only one triangle when the edge is on the boundary of a bounded surface. In addition, three parameters need to be computed or assigned before the mesh growing process. These parameters are: the dense value (ρ) of input points, which is the average distance of a point to its nearest neighbor; the upper bound value ($\alpha_{\max}$) of anyone of the three angles of a new constructed triangle; the upper bound value ($\beta_{\max}$) of the angle between the normal of a new constructed triangle which is incident to the current front edge and the normal of the triangle on the other side of the current front edge. In our implementation, ρ is set as the average distance of some randomly selected points (100 points or up to 5% of the input points) to their nearest neighbor, $\alpha_{\max}$ is 120°, and $\beta_{\max}$ is 90°. ρ will be used to define the radius of the 3D search region in the newly developed triangulation operation. $\alpha_{\max}$ is used for triangle quality control. User can adjust the two parameters during meshing growing or assign different values to different input data sets, totally depending different quality requirements.

3.3. Mesh Growing Driven by Priority

One open issue in the incremental based methods is that it is difficult to guarantee the meshing correctness at the place where a sharp curvature change has happened. Actually, the issue also exists in the methods of the other two categories. However, it is always much easier and safer to generate correct meshes (triangles) at a 'flat' place. During mesh growing, if we can postpone the meshing operation when the advanced front reaches a 'sharp curvature change' place and force the mesh to propagate on 'flat' places first, one interesting thing will happen: In most cases, the meshing growing will gradually propagate from other directions to the other side of the 'sharp curvature change' place through the connected "flat" places as a reliable path, and naturally stitch or make it much easier and safer to connect the meshes (triangles) on both sides of the sharp curvature. One illustrative example in 2D space is given in Fig. 3. In the figure, each line can be thought as a triangle. When the mesh growing propagates from Point *c* to Point *d* as shown in Fig. 3 (a), according to the normal value variance of the triangle *bc* and its neighboring triangle *cd*, a potential sharp surface feature is identified near Point *c*. If the mesh growing can temporarily stop at Point *d* and propagates in another direction (from Point *b* to *a*), it will reach to Point *e* eventually as shown in Fig. 3 (b). Then, the mesh growing will connect Point *e* and Point *d* directly or restart the propagation from Point *d* to complete the meshing process as shown in Fig. 3 (c). Otherwise, a wrong mesh will most likely be constructed if we continue the mesh at Point *d* first as illustrated in Fig. 3(d). In 3D space, it is much easier to find such a "correct" path to construct surface (mesh) by this strategy because of the 3-dimensional freedom. Under the strategy of processing the 'flat' place (easy surface feature) first, the difficulty and possibility of causing a meshing error at the place with sharp curvature change (difficult surface feature) would be reduced significantly, especially incorporating with the new triangulation method described in Section 3.5 and four topological operations described in Section 3.6.

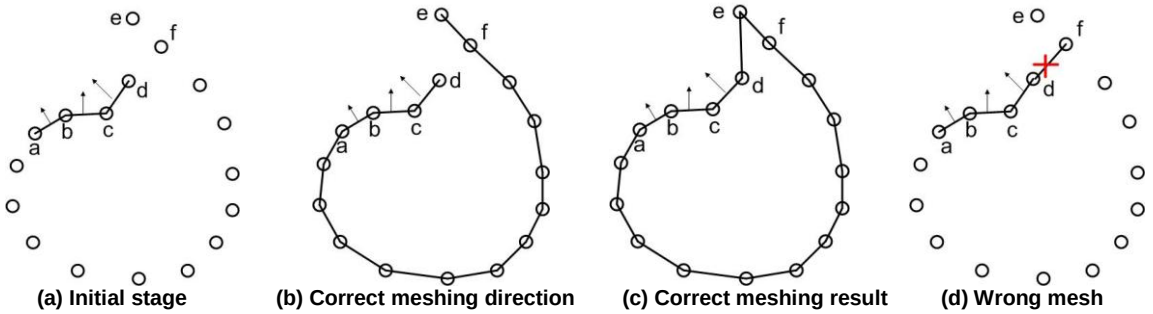


Fig. 3. Illustrative example in 2D space

To implement the strategy in an efficient and practical manner, a priority driven algorithm is proposed and developed. The main idea of this method is to create and maintain a priority queue during surface construction to guarantee the mesh growing always performs at 'flat' places first and then 'sharp change' places.

The major steps of the algorithm are described as follows:

STEP 1: All input points are tagged as *free*. Then, the initialization process, as described in Section 3.4, determines a starting triangle.

STEP 2: The three edges of the starting triangle are tagged as the front edges, whose cost T is set to zero, and pushed into a minimum priority queue (a sorted queue wherein all points are sorted by their cost T in an increased order), named *edge heap*.

STEP 3: Pop the front edge which has the smallest cost T from the edge heap and call it *current front edge*.

STEP 4: Find the *free* points close to the current front edge by using nearest neighbor search and change their state from *free* to *close*, identify the reference point from these *close* points (See Section 3.5 for details of these operations). If no reference point can be found, jump to **STEP 7** to continue the mesh growing.

STEP 5: Construct a new triangle with an appropriate topology operation (See Section 3.6 for the definition of topological operations) and register the new triangle to a *triangle list*. If there are new front edges created during the triangulation, compute their cost T by Eq. (1) and push them into the *edge heap*.

Calculating Cost T: Before adding any new front edge into the *edge heap*, its cost T needs to be calculated first for measuring the curvature change of the local surface. The definition of cost T is based on the fact that the normal deviation of a new triangle and its neighboring triangle (both are connected to the same front edge) reflects the shape change of the local surface area.

$$T = 1 - \frac{\vec{N}_a \bullet \vec{N}_b}{\left\|\vec{N}_a\right\|\left\|\vec{N}_b\right\|} \quad (1)$$

where $\vec{N}_a$ is the normal of the new triangle incident to the current front edge, $\vec{N}_b$ is the normal of the triangle connected to the other side of the current front edge. The change of Cost T is monotonic. When the difference of $\vec{N}_a$ and $\vec{N}_b$ increases, T will become larger too. Therefore, when $\vec{N}_a$ and $\vec{N}_b$ are in the same direction, which means the local surface is planar, T will be 0 (The minimum value of *T*); when $\vec{N}_a$ and $\vec{N}_b$ are in the completely opposite direction, which means the local surface has the biggest shape change (180 degree difference), T will be 2 (The maximum value of *T*).

STEP 6: Tag *accepted* to the reference point and assign *free* to the other *close* points.

STEP 7: Repeat the loop until the *edge heap* is empty.

STEP 8: In case the mesh growing stops somewhere before the entire data set is processed. A multiple-starting-seed strategy is designed here by restarting the initial process and mesh growing loop if there still have some unmeshed point patches in the data set.

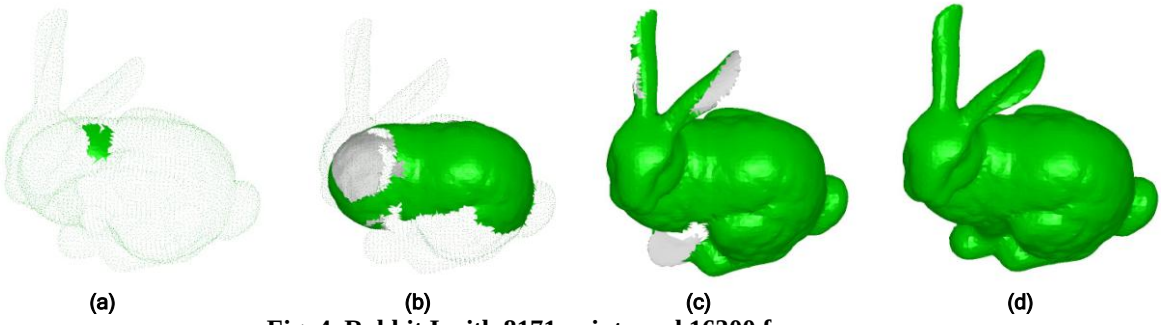


Fig. 4. Rabbit I with 8171 points and 16300 faces.

For better understanding, an illustration of mesh growing proposed in this paper is given in Fig. 3. A real example is given in Fig. 4 to illustrate the procedure of mesh growing. Fig. 4 (a) - (c) show three intermediate states of the priority driven based mesh growing process. Fig. 4 (d) is the final result.

3.4. Selection of the Starting Triangle

In the priority driven based meshing method, we let the flat portions of the object have higher priority to be triangulated than the places with sharp curvature changes. The proposed mesh growing method will start at a relative flat place. To do so, we randomly choose *m* (a constant value) small patches of surface points from the surface data. For each patch, several triangles are constructed. The new generated triangles do not need to obey the constraints discussed in Section 3.2, but their edge order must have the same direction (either clockwise or anticlockwise) to guarantee that their triangle normal has the same orientation (either outside or inside). For each patch, Eq. (2) to Eq. (5) are applied in sequential order to compute its normal variance. Based on the fact that the smaller the variance value the flatter the patch, the flattest patch among the *m* will be identified by Eq. (6) and selected as the place to start mesh growing.

$$\vec{N}_i = \vec{V}_{i1} \otimes \vec{V}_{i2} \quad (i \in n) \quad (2)$$

where $\vec{V}_{i1}$ and $\vec{V}_{i2}$ are two edges of the i[th] triangle of the current patch.

$$\vec{N}_i \leftarrow \frac{\vec{N}_i}{\left\|\vec{N}_i\right\|} \quad (i \in n) \quad (3)$$

$$\vec{N} = \frac{1}{n}\sum_{i=1}^{n} \vec{N}_i \quad (4)$$

$$Var = \frac{1}{n}\sum_{i=1}^{n}(\vec{N} - \vec{N}_i)(\vec{N} - \vec{N}_i)^T \quad (5)$$

$$S = \min\{S_j \mid j \in m\} \quad (6)$$

where n is the number of the selected triangles (in the range of 10 to 15 in our implementation), m is the number of the selected patches (in the range of 5 to 10 in our implementation), S_j is the j[th] patch and S is the flattest patch with the minimum variance.

After the flat-place selection, we still need to find a starting triangle in the selected patch to begin mesh growing. Since a small and local surface area, especially a flat surface, can be reasonably approximated as a planar surface, we can project the surface points of the selected patch onto a 2D plane and find three suitable points to construct the starting triangle in 2D space. To do so, we first fit a 2D plane to the selected fattest patch with Least-square, and then project all patch points onto the plane. We choose three points closed to each other and construct a triangle that no *free* point falls in the triangle and identify its corresponding triangle in 3D space and begin mesh growing.

3.5. Triangulation

Triangulation in the priority driven based surface reconstruction can be defined as a process of finding a suitable reference point for the current front edge and constructing a new triangle by linking the reference point and the endpoints of the current front edge with supporting of an appropriate topological operation. Most triangulation methods discussed in the previous section have strict requirements and/or assumptions to the input data, such as uniform sampling rate, closed surface, and/or noise-free sampling points. Another limitation of them is the processing speed is not fast enough for practical applications. In this section, a fast, easy-to-implement, practical, and efficient triangulation method is developed. The main idea of the proposed triangulation method is to define a 3D candidate region centered at the middle point of the current front edge and find all *free* and *front* points fallen in this region by searching the nearest neighboring points through the already-created hash table. Then, all unsuitable candidate points are removed to guarantee the mesh (triangle) quality and prevent any possible error which might occur during the new triangulation generation process. To fast and efficiently identify the reference point from the candidate set, priority driven strategy is used again by computing a priority cost for each candidate point and storing all the candidate points in a maximum priority queue. The details on how to compute the reference point p for the current front edge, e, whose endpoints are defined as a and b are described as follows:

Step 1: **Identify all candidate points near to *e***: Once a front edge, *e*, has been popped out from the *edge heap*, a search region for identifying the reference point for *e* is defined. The search region is a ball in 3D space and its center is the middle point of *e*. The radius *R* of the search region is a predefined value and equal to $k\rho$, where ρ is the dense value of the surface point set and k is a constant value (In our research, k was set as 4). The nearest neighbors of the middle point of *e* are searched in the 3D ball through the already-created hash table and all the found points are put into a candidate list. The distance of each candidate to the middle point of *e* is also computed and recorded. If the candidate list is empty, the edge *e* will be thought as the boundary of the surface and no meshing operation will be applied to the front edge. A new front edge will be popped up from the *edge heap* to repeat the triangulation procedure.

Step 2: **Remove all unsuitable points from the candidate list**: The removal is completed by checking each candidate point with the following two rules:

Assume t is the current trial point and use $\vec{ab}$ to present the current front edge *e*.

Rule 1: Compute the angle of Line $\vec{at}$ and Line $\vec{ab}$, and the angle of $\vec{bt}$ and $\vec{ba}$. If any angle of the two is larger than $\alpha_{\max}$, Point *t* will be removed from the candidate list. $\alpha_{\max}$ is used for triangle quality control. If any angle of a generated triangle is larger than 120°, the triangle will be abandoned.

Rule 2: Calculate the angle between the normal of the triangle Δatb and the normal $\vec{N}$ of the triangle also incident to the edge $\vec{ab}$. If the angle is larger than $\beta_{\max}$, Point *t* will be removed from the candidate list, which means if the angle between the normal of a new constructed triangle and the normal of the triangle on the other side of the current front edge is larger than $\beta_{\max}$, the new triangle might be a wrong triangle caused by a noisy point and needs to be abandoned.

Step 3: **Create a priority queue for reference point selection** : we first compute the cost (priority value) for each candidate point and create a priority queue accordingly. Then, we choose the reference point based on the priority queue. For each candidate point, its cost *C* is calculated by Eq. (7).

$$C = \cos(\beta(t)) + \cos(\gamma(t)) + (R - dist(t))/R \quad (7)$$

where C is the cost of the current trial point t. In Fig. 5, the dots represent the *close* points of the edge *e*, $\vec{N}$ is the normal of Δabc, and *o* is the middle point of *e*. To make the new triangulation method robust to noise, we consider one character of object surface, surface consistence (or called surface continuity), into the cost calculation by defining $\beta(t)$ in the equation. $\beta(t)$ is the angle between the normal of Δatb and the normal $\vec{N}$, reflecting the normal deviation of the new constructed triangle and the triangle connected to the other side of *e*. We use cosine function of $\beta(t)$ to support the character. For cosine function of $\beta(t)$, smaller the normal variance is and larger the value. By doing this, we can force the mesh to grow along the real surface of an object and make the triangulation insensitive to noise. In the equation, we also introduce cosine function of $\gamma(t)$ and $R - dist(t)$ for triangle quality control. $\gamma(t)$ is the angle of the line $\vec{ot}$ and the plane *od* which is vertically to the edge e and passing through the center of *e*. Cosine function of $\gamma(t)$ is used to make the candidate points close to the plane *od* have higher priority during triangulation. $dist(t)$ is the Euclidian distance of Point t to the edge e. $R - dist(t)$ (*R* is the radius of the search region) will enable the points close to *e* to have higher priority than the points far from *e*. $dist(t)$ and $R - dist(t)$ will make the point close to *e* and near the center line of e has the highest priority to be the reference point. *R* is used for normalize $R - dist(t)$. In some situations, un-equal weights for these three factors can achieve better reconstruction results. Therefore, Eq. (7) can be expressed as a weighted equation:

$$C = w_1 \times \cos(\beta(t)) + w_2 \times \cos(\gamma(t)) + w_3 \times (R - dist(t))/R \quad (8)$$

where w_1, w_2, and w_3 are the weight of each factor.

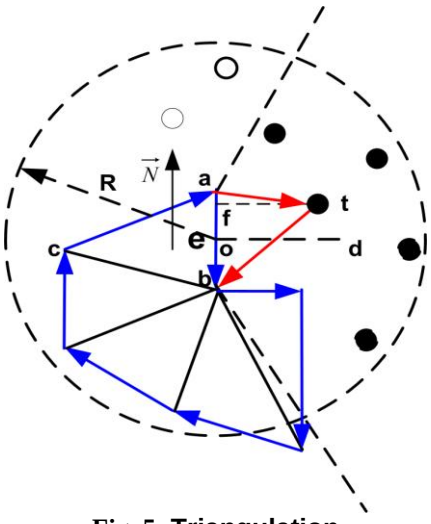


Fig. 5. Triangulation

Note that Fig. 5 is the top view of the local surface at the current front edge *e*, all *free* points and triangles in the figure are not necessarily located on a 2D plane. In the 2D illustration, white dots break Rule 1 while black dots satisfy *Rule 1* and *Rule 2*, and thus are classified as candidates for reference point selection. Among these candidates, Point *t* has the largest cost, and thus has the highest priority to be selected for reference point consideration.

Step 4: **Select the reference point for triangle construction and verifying its correctness**: Once we have completed the cost estimation for each candidate point, all candidate points are pushed into a maximum priority queue sorted by their cost in a decreasing order, so-called *point heap*. The candidate points will be popped out one by one for consideration. With the *point heap*, the points with larger cost will be considered with

higher priority. In most cases, the first popped point from the *point heap* will be the reference point, which means usually the rest points in the *point heap* will not be processed, and thus it will save huge computational time on triangulation via *point heap*.

To identify if the current trial point is suitable to be the reference point, we will link the current trial point and the endpoints of the current front edge to generate a new triangle. If the two edges of the new triangle have no interaction with the other front edges (e.g. $\vec{at}$ and $\vec{tb}$ as shown in Fig. 5), the point will thought as the reference point and the triangle will be a correct generation. If any new edge has intersection with the other front edges, the trial point will be discarded and a new candidate will be popped out from the *point heap*, and the verification process will keep repeating till the right reference point is found. In practice, it is very difficult to perform line-intersection check in 3D space. But, it is always easy to do this in 2D space. Fortunately, in surface reconstruction a local surface of an object can be approximated as a planar surface. Here, a 2D line-intersection check [32] is performed instead of having a 3D check.

3.6. Topological Operations

At each step of mesh growing, once the reference point *p* of the current front edge, named e, has been selected, a new triangle will be constructed and added into a triangle list. A topological operation is needed to finalize the new triangle construction and perform appropriate actions to maintain the advanced front (the *edge heap*). According to the type (*free* or *front*) of the reference point and its situation to the advanced front, four topological operations (*join*, *fill*, *link*, and *glue*), which can happen at each step of mesh growing, are carefully defined. In [17], only *join* and *glue* were defined for surface reconstruction. To deal with more complex situations, two new operations (*fill* and *link*) are defined here. The four topological operations illustrated in Fig. 6 are described as follows.

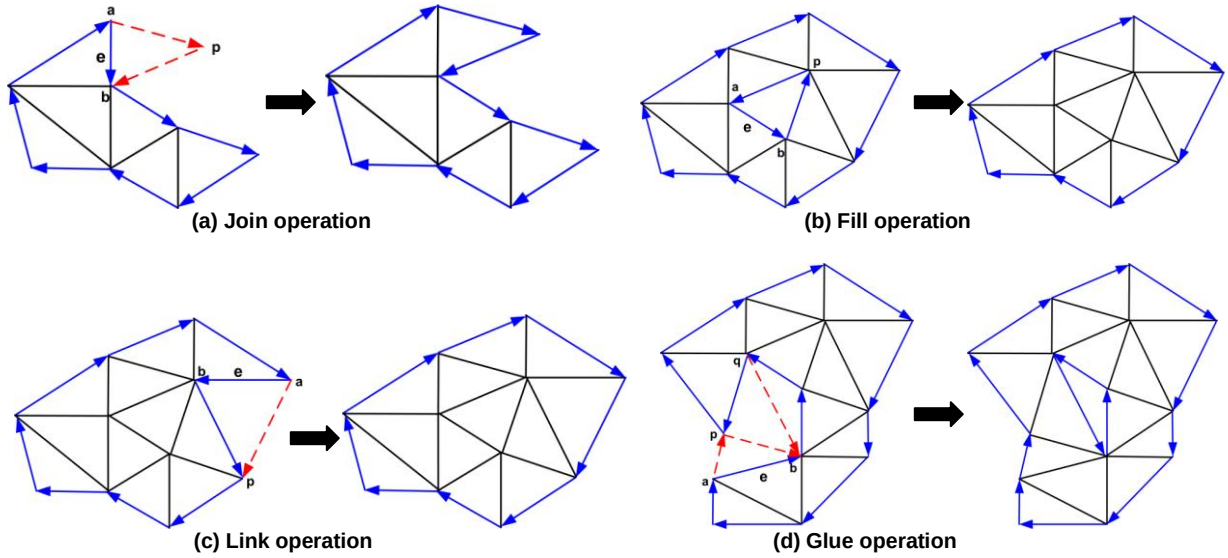


Fig. 6. Four topological operations

Let B present the set of *front* points and C present the set of *close* points. Given a triangle list used for registering the new created triangle and the *edge heap* created for the purpose of priority driven, we have the following regulation for the four topological operations.

● ***Join:*** If the reference point p is in *C*, we generate a new triangle, Δapb, and register the new triangle to the triangle list, insert the edge $e_{(ap)}$ and the edge $e_{(pb)}$ into the *edge heap*, and tag Point *p* as *accepted*.

● ***Fill:*** If the reference point p is in *B* and the two endpoints of the current front edge, $e_{(ab)}$, are connected to Point *p*, we create a new triangle, Δapb, and register the new triangle to the triangle list. Then, we remove the edges, $e_{(ap)}$ and $e_{(bp)}$, of Δapb from the *edge heap*.

● ***Link:*** If the reference Point p is in *B* and only one endpoint of the current front edge, $e_{(ab)}$, is connected with Point *p*, we link Point *a* and Point *p*, place the edge $e_{(ap)}$ into the *edge heap*, construct a new triangle, Δpba, and register the new triangle to the triangle list. Then, we remove the edge $e_{(bp)}$ from the *edge heap*.

● ***Glue:*** If the reference point p is in *B* and no any endpoint of the current front edge, $e_{(ab)}$, is connected with Point *p*, we link Point *a* and Point *p*, and link Point *p* and Point b to construct a new triangle, Δapb, and register the new triangle to the triangle list. In this situation, four edges (two edges are in, and two edges are out) are incident to *p*, which will cause difficulties for future mesh growing at this place because two advanced fronts are intersected at the point *p* and thus bring confusion to effectively maintaining the advanced front. To avoid this confusion, only one input and one output front edge are permitted for a *front* point during mesh growing. Therefore, to solve this confliction (four edges are incident to Point *p*), another new triangle, Δpqb, is constructed and registered to maintain this character. Two new edges, $e_{(ap)}$ and $e_{(qb)}$, are added into the *edge heap*, and the edge, $e_{(qp)}$, is removed from the *edge heap*.

During mesh growing, the advanced front of mesh area is always closed, but the number of the advanced front might be not limited to one and not a constant number during the whole process. The front might be divided into two or more independent fronts after one *glue* operation or might be eliminated after one *fill* operation as demonstrated in Fig. 6(d) and 5(b) respectively. With the directional front, we can keep the normal consistence of the surface and easily distinguish the outside and inside of the surface. According to the number of the advanced fronts after mesh growing, we can get the number and the position of the holes on the constructed surface. Based on the information of the holes, a water-tightness surface can be obtained by applying a hole-filling algorithm to these holes.

3.7. Algorithm Summary

For more clear description, the following pseudocodes summarize the implementation of the entire process of the proposed algorithm wherein the triangle list is named as L, the edge heap is named as H, and the *free* point set is named as F.

Surface Reconstruction Algorithm:

1: $H \leftarrow$ Selection_of_starting_triangle(F)
2: While ($H \neq empty$) do
3: $(e,T(e)) \leftarrow$ Pop a front edge from H
/* e is the current front edge and $T(e)$ is the cost of e */
4: $p \leftarrow$ Triangulation(e, F)
/* p is the reference point */
5: switch(p) do
/* classify p by the definition of the topological operations */
6: case 1: Join_operation(p, H, L) and triangle registration
7: case 2: Fill_operation(p, H, L) and triangle registration
8: case 3: Link_operation(p, H, L) and triangle registration
9: case 4: Glue_operation(p, H, L) and triangle registration
10: others cases: can't find reference point and continue the process
11: end_switch
12: end_while
13: If (F still contains point patches), recalculate the dense value and jump to **Line 1**
/*A multiple-starting-seed strategy for surface reconstruction */
14: The isolated *free* points left in F (noisy points) are tagged as *refused*
/* The noisy points are collected at the data acquisition stage */

Compared with the other methods, including the methods in the categories of the spatial subdivision based and the surface distance function based, the proposed method has some salient advances on the aspects of processing

speed, robustness, and implementation. Like others, it also has its own limitations on surface reconstruction. The major advantages and disadvantages of the proposed surface reconstruction algorithm are discussed as follows:

3.7.1. Advantages

A good surface reconstruction algorithm/system is always hoped to work robustly on the surface data (point cloud) in all kinds of situations. The ideal system should have fewer assumptions/limitations on input data and can recover surface details as many as possible. Some other important factors, such as processing speed, requirements/constrains on input data and topological complexity of the object, and robustness to noise, have to be taken into account for performance evaluation. Compared with the other surface construction methods, the proposed algorithm has the following advantages.

● **Work well on non-uniform sampled data:** In practice, due to the shape variance and surface smoothness of an object, it almost impossible to keep a uniform sampling rate for an object, especially the object with complicated surface. Therefore, if a surface reconstruction algorithm/system can work well on non-uniform sampled data is a very important consideration when people evaluate its performance. Unlike the other incremental methods, such as the method proposed by Gopi, et al. [16] and BPA proposed by Bernardini, et al. [17], which requires a uniform sampling rate or several fixed sampling rates to input data, the proposed approach can work well on non-uniform sampled data. In the proposed algorithm, all *free* points within the 3D search region to the current front edge will be screened for triangulation, not only limited to the points with a fixed distance to the current front edge, such as in [16] and [17]. In case the dense of a local surface becomes very sparse wherein the dense value is larger than the predefined radius $k\rho$, the proposed algorithm will automatically recalculate the dense value for this local surface region and restart the triangulation process to this area. Or, K can be set to a larger value to deal with this situation, but the trade-off thing is it would increase the computational cost of triangulation.

● **Object with complicated topology and/or geometry:** In the proposed algorithm, we consider the consistency of the normal of the new triangle and its neighboring triangles already created, which is reflected by $\beta(t)$ in Eq. (7) and Eq.(8). The triangle with similar normal to its neighboring triangles has a higher priority to be considered as a correct triangle. Under this consideration, the proposed mesh growing prefers to grow along the local surface and refuses the attraction of noisy points and the surface points of other regions of the object. Thus, the new triangulation method can work correctly and efficiently to the surfaces with complicated geometry and /or topology as demonstrated in Section 4.

● **Object with sharp-varied surfaces:** It is always difficult to guarantee meshing correctness of the sharp-varied surface of an object. However, since the priority driven based method will process all 'flat' places first before processing the sharp-varied surfaces, most difficult areas will be correctly reconstructed via the "flat"-to-"sharp" strategy together with the newly developed triangulation method and topological operations. One example is given in Fig. 10.

● **Robust to noise:** Without any post-processing (e.g. filtering), it is almost impossible to avoid the case that the input point set might contain some random noisy points which lie out of the object surface. The noise might be caused by surface smoothness, projection rays, reflection surfaces, and other factors during surface data acquisition. In the proposed method, due to the large distance and/or the big normal change of the noisy points to the current front edge, the noisy points will either not be considered at all or has a very low probability to be considered for triangulation as these noisy points will be located at the bottom of the priority queue during mesh growing even when they are considered for triangulation. After surface reconstruction, all the noisy points will be left and tagged as *refused* in the proposed algorithm. This advantage is already proven by many data sets with a variety of noise. Some experimental results of the data sets with heavy noise are shown in Section 4.

● **Fast processing speed:** The important feature of the proposed surface reconstruction algorithm is it is pretty simple and easy to be implemented. Because of the low computational cost of the proposed algorithm, triangulation can be done in real time at each step of mesh growing. More details of the discussion can be found in Section 4.6.

Another important advantage of using the proposed incremental mesh growing method can be very easy to be implemented and executed parallel by a multi-threading method. Since the proposed algorithm maintains the inde-

pendency of the advanced front of a mesh area and support multiple advanced fronts for mesh growing, it can be executed by multiple threads without any big modification to the method. Compared with its single-thread-execution mode, the only changes for its multi-thread-execution mode include the sharing of triangle list, input data set and the status of each surface point between multiple threads. As a result of using multi-threads, the processing speed of object surface reconstruction will be improved significantly. Some experimental results on processing speed can be found in Section 4.

3.7.2. Limitations

The proposed triangulation operation can run in real time, but the trade-off is it is not a Delaunay triangulation operation, which means the operation cannot guarantee every triangle is a Delaunay triangle. Therefore, the triangle quality is not as good as that of using Delaunay-based methods. Actually, even for the Delaunay-based triangulation methods, it is impossible to construct a Delaunay triangle at every where of a surface. Since triangle quality and topology correctness of the proposed surface reconstruction method is guaranteed by triangulation criteria and topological operations, most generated triangles are Delaunay triangle or approximated Delaunay triangle. But, some sliver-liked triangles are also created, especially for some non-uniform sampled data sets. To achieve high-quality triangulation results, triangle optimization algorithms need to be called to re-triangulate some triangles after surface reconstruction with our proposed algorithm. Another limitation of the proposed method is its limited ability on multi-resolution representation as the proposed method will use every input point as triangle vertex. To support multi-resolution representation, the proposed algorithm has to be coupled with surface simplification methods as described in the literature.

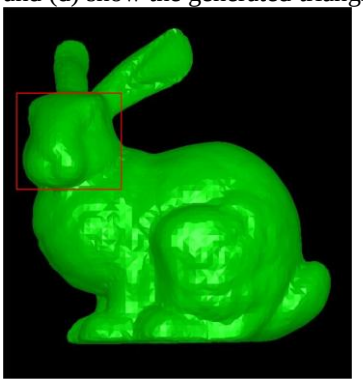
During triangulation, if a local region is very noisy or in a feature region, the proposed projection-based operation for reference selection might not work correctly all the time. Also, if all the surface of an object is very bumpy, it might be difficult to find a flat region to start mesh growing automatically. In this situation, a manual selection is needed to choose a flat region. In addition, the parameter settings of the proposed triangulation method affect triangle quality and processing speed. Although the default settings work well for all testing data sets so far, better triangle quality and less processing time can be achieved by tuning parameter settings for each input data set individually.

4. Experimental Results

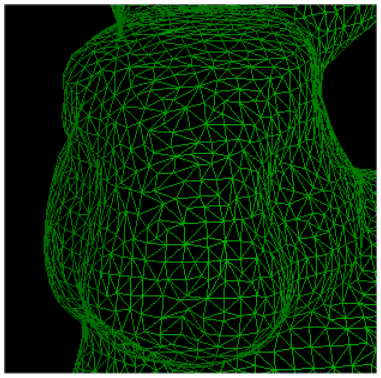
The algorithm was implemented by C++ with OpenGL, and ran in MS Windows XP environment. Extensive experiments were performed to validate the proposed algorithm. All tests were run on a PC with 2.6GHz P4 processor and 512MB RAM. The in-core memory consumption during the executing time is in the range of 50MB to 260 MB depended on the size of input data. For performance evaluation, the proposed algorithm was tested for 3D objects with different sampling rates (resolutions), complicated geometries and topologies, open and close surface, non-uniform sampled, and noisy data sets. Its processing time was recorded and compared with three selected methods. The investigation on parallel implementation was also performed and summarized in this section.

4.1. Testing results of an object with different sampling rates/grids

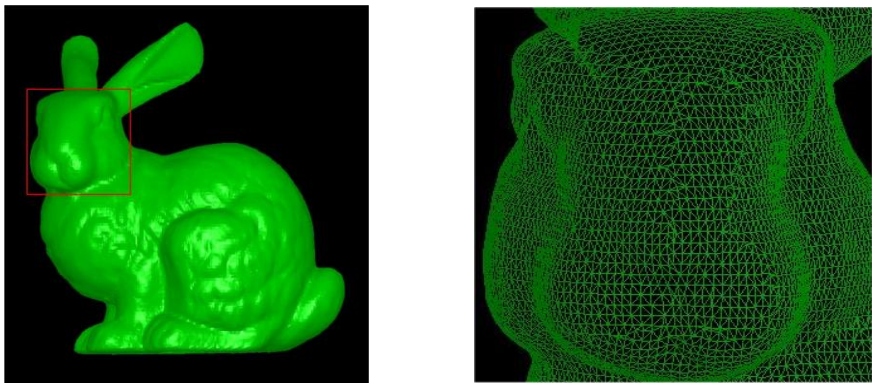
The proposed method can work well on an object with different resolutions. In Fig. 7, the proposed algorithm was employed on an object (rabbit) with different sampling rates (resolutions). Fig. 7 (a) and (c) show the constructed surfaces. Fig. 7 (b) and (d) show the generated triangles on surface.



(a) Rabbit I



(b) Part of Rabbit I

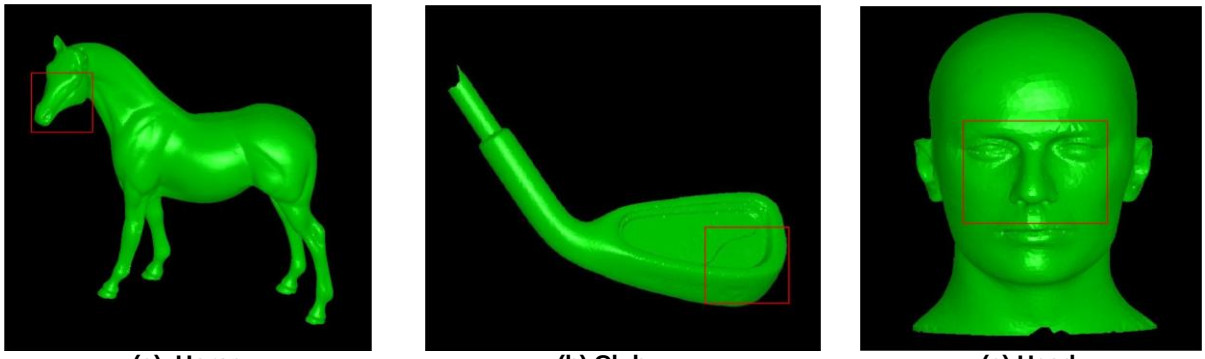


(c) Rabbit II (d) Part of Rabbit II

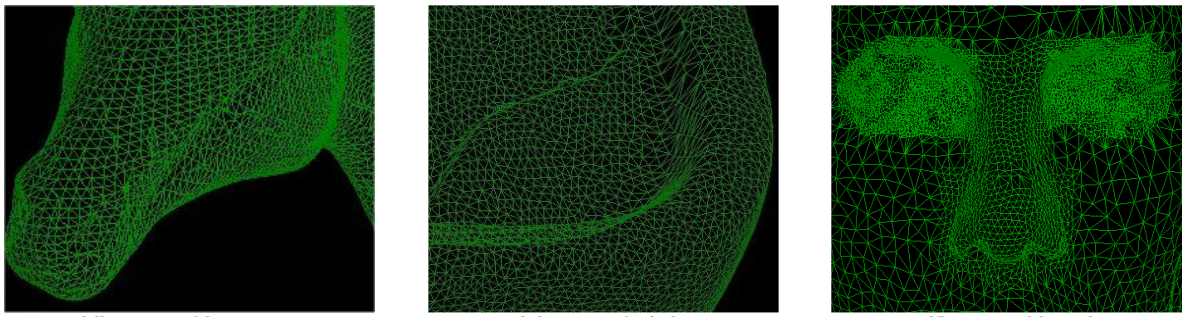
Fig. 7. Rabbit with different sampling rates

4.2. Testing results of the objects with different geometries and/or topologies

Fig. 8(a)-(c) and Fig. 9 demonstrate the robustness of the proposed algorithm in handling the various cases of open/close surface, and unorganized point sets (non uniform sampled data). Fig. 8(d)-(f) show the constructed triangles on surface to observe the quality of the generated triangles. Fig. 10 illustrates the proposed algorithm works fine on the object with complicated topology and unorganized point set. The real challenge here is the distance of two layers is very close to each other, and even less than the dense value ρ . But, with the proposed algorithm, the surface of the object still can be reconstructed correctly.

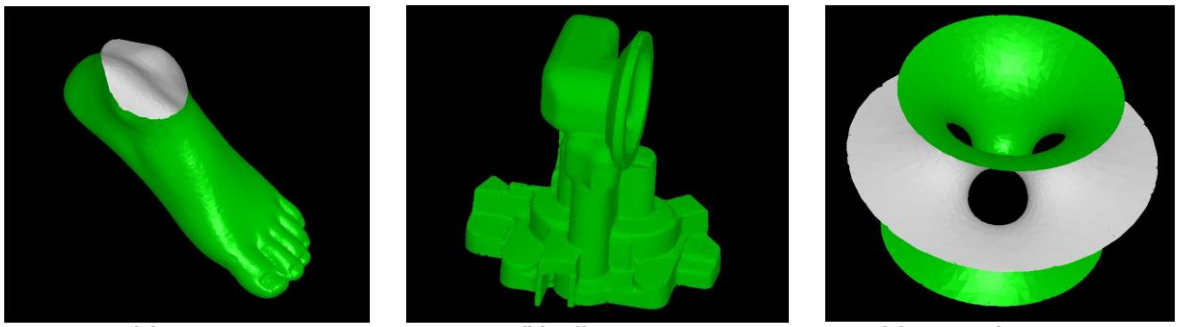


(a) Horse (b) Club (c) Head



(d) Part of horse (e) Part of club (f) Part of head

Fig. 8. Horse, Club, and Head



(a) Foot (b) Oilpmp (c) Hypersheet

Fig. 9. Foot, Oilpmp, and Hypersheet

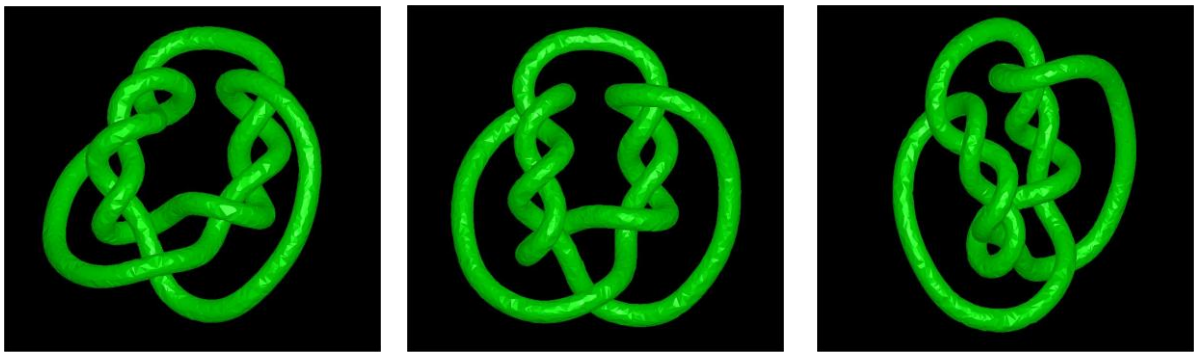
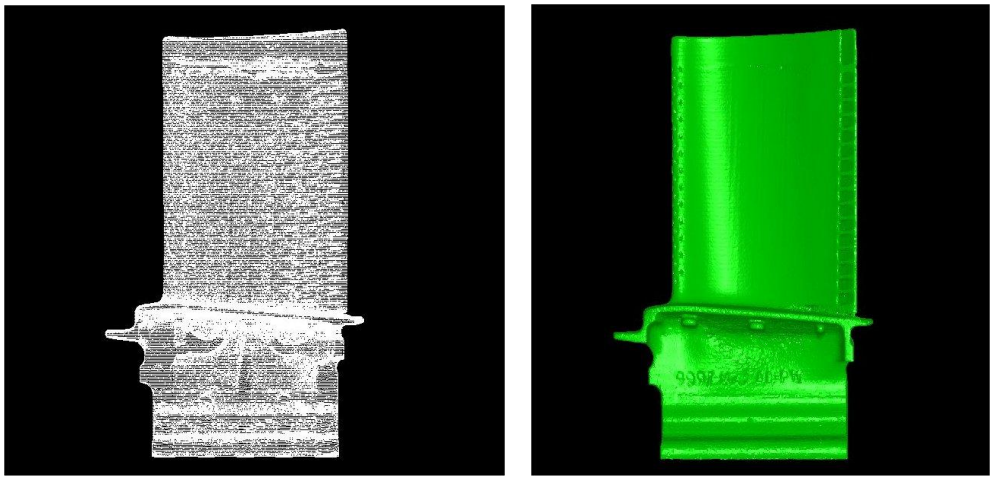


Fig. 10. Knot with 10,000 points and 19,317 faces

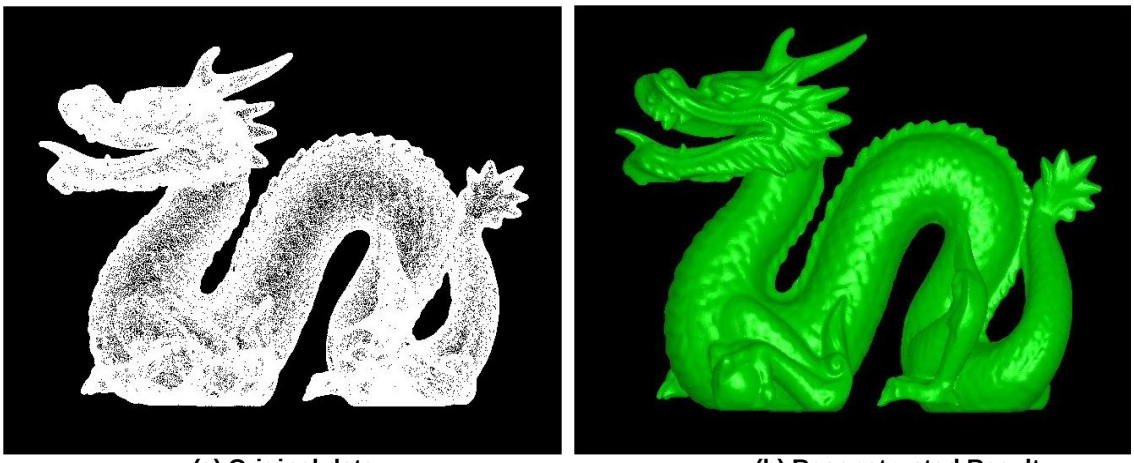
4.3. Testing results of the complicated objects with dense sampling rate

To test the efficiency of the proposed algorithm on large data sets, some complicated objects with dense sampling rate were chosen for surface reconstruction. Fig. 11, Fig. 12, and Fig. 13 show the proposed algorithm can work well on large and dense data sets with complicated geometry and/or topology.



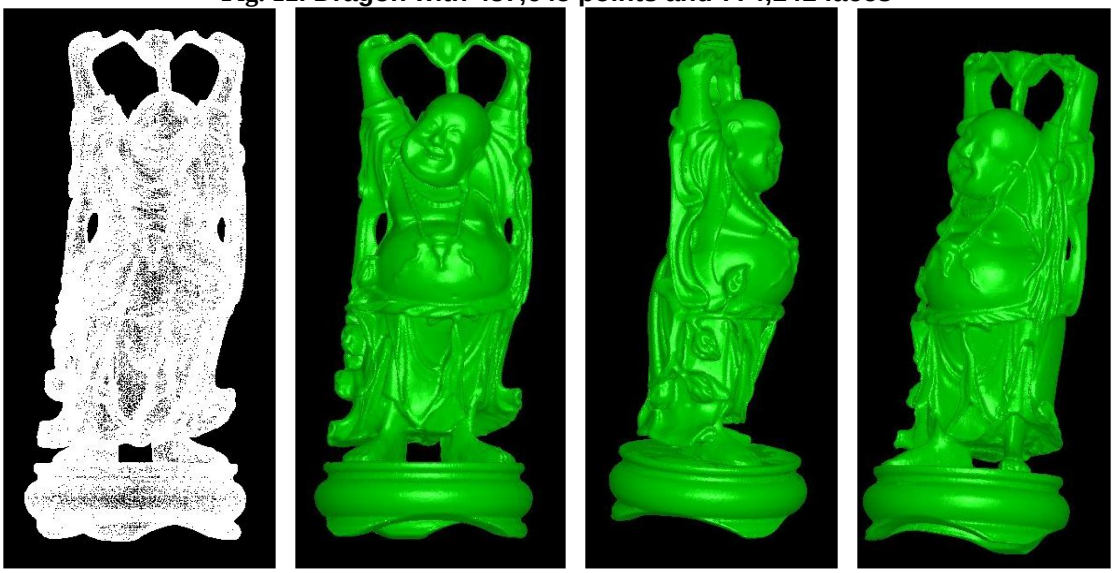
(a) Original data (b) Reconstructed Result

Fig. 11 Fan-blade with 441,477 points and 866,137 faces



(a) Original data (b) Reconstructed Result

Fig. 12. Dragon with 437,645 points and 774,141 faces



(a) Original data (b) Reconstructed Result: Face view (c) Side view I (d) Side view II

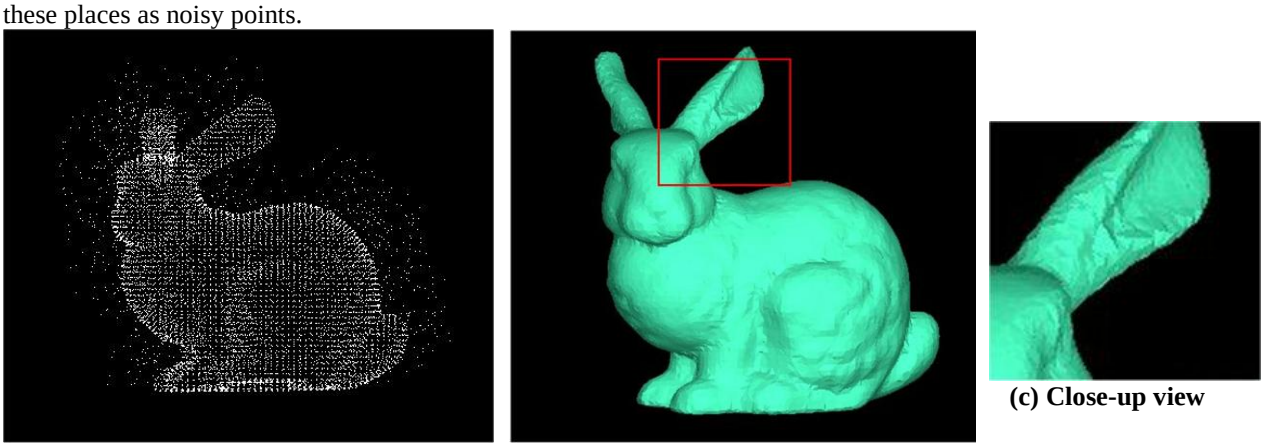
Fig. 13. Buddha with 543,652 points and 981,595 faces

4.4. Testing results of the objects with noise

A number of outliers, expressed as a percentage of the number of input samples, were added in order to produce noisy data sets. In our performance investigation, noisy points in the range of 20% to 40% of the total number of an input data set were added in. Since the proposed approach is insensitive to noise disruption, object surface can be reconstructed successfully for the noisy data sets. In the investigation, all testing data sets with noise were correctly reconstructed through the proposed algorithm without changing the default parameter-settings. Two test results are selected to show in Fig. 14 and Fig. 16. In Fig. 14, near two-thousand noisy points were added into the original input, the proposed algorithm reconstructed the surface accurately and resisted noise disturbances without any parameter tuning. In Fig. 16, a foot's surface was reconstructed successfully even with a higher noise percentage (totally four-thousand noisy points were added in). If we increase the weight of $\beta(t)$ in the proposed triangulation operation to enhance its resistance to noise, more accurate results can be obtained.

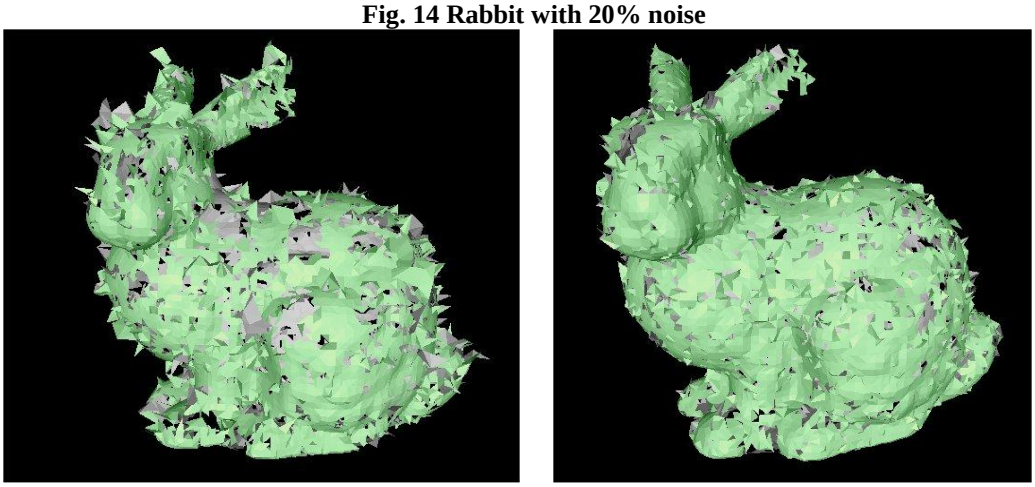
To compare the performance of the proposed method with other conventional methods, three well-known surface reconstruction algorithms were selected. We implemented the BPA algorithm proposed by Bernardini, et al. [17] and the GKS method proposed by Gopi, Krishnan, and Silva [16], and used the binary codes of the Cocone algorithm devised by Prof. Dey [11, 12] to repeat the tests described in this subsection. Since Cocone is sensitive to noise, the tests to the noisy data sets shown in Fig. 14 (a) and Fig. 16 (a) are all failed. GKS and BPA can work on the data set shown in Fig. 14 (a) and their reconstruction results are shown in Fig. 15. But, they both failed to con-

struct a surface for the data sets with 40% noise shown in Fig. 16 (a). Note that the holes on the surfaces shown in Fig. 15 are either caused by the failure of surface reconstruction at these places or misjudged the surface points in these places as noisy points.



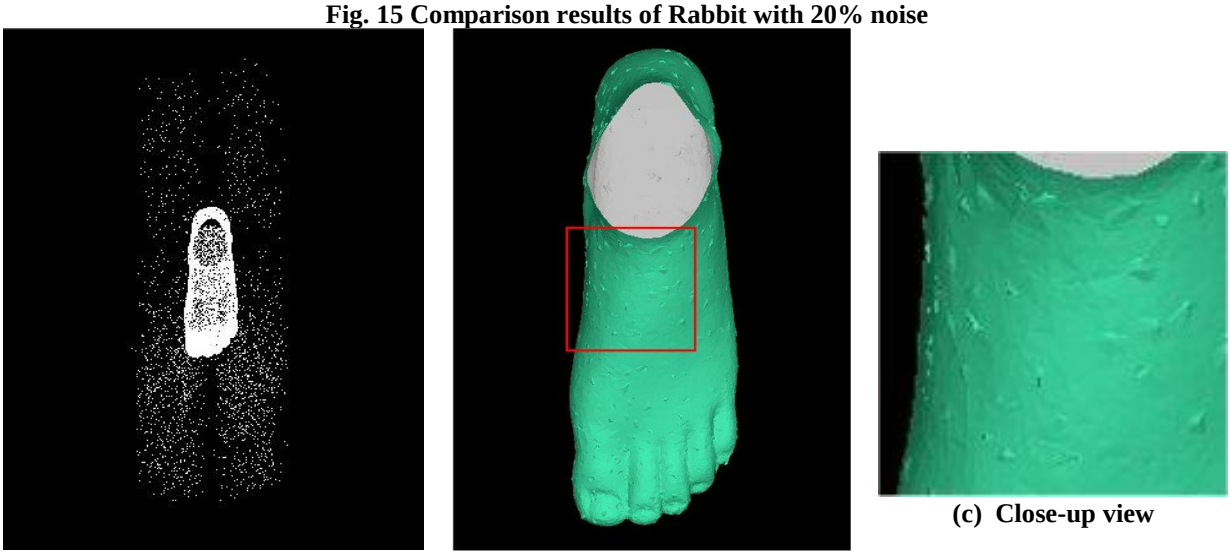
(a) Original data **(b) Reconstructed result** **(c) Close-up view**

Fig. 14 Rabbit with 20% noise



(a) Reconstructed result by GKS **(b) Reconstructed result by BPA**

Fig. 15 Comparison results of Rabbit with 20% noise



(a) Original data **(b) Reconstructed result** **(c) Close-up view**

Fig. 16 Foot with 40% noise

From the tests given in Section 4.1 to 4.4, we can draw the conclusion that the proposed algorithm can work efficiently on the objects with different sampling rates, geometries, topologies including open/close surface, and unorganized point sets (non uniform sampled data), and also work well on noisy data sets. Note that the experiments discussed in this paper use additional outliers. If all surface points are shifted a random amount, preprocessing (e.g. Moving Least Square filter) is needed to smooth out such noise before performing surface reconstruction.

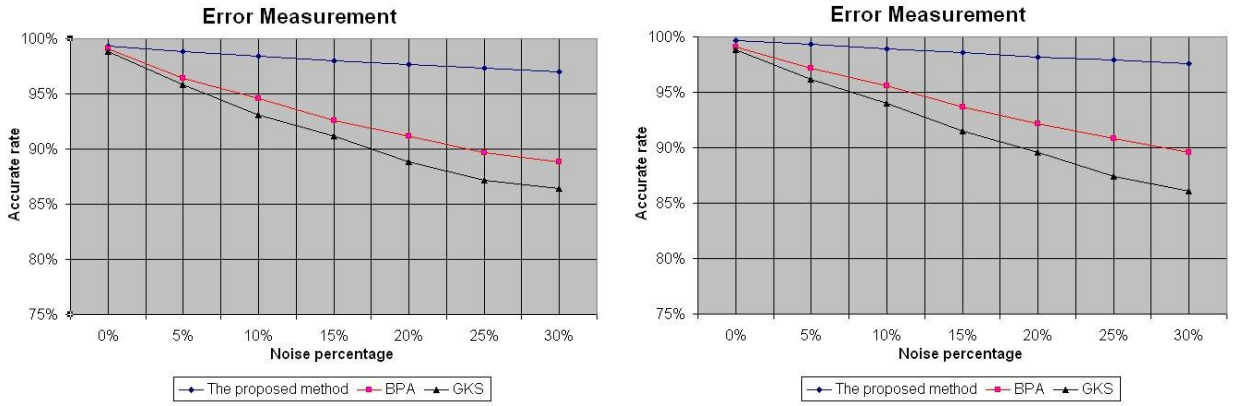
4.5. Error measurement

To investigate the topological difference between the reconstructed mesh and original object surface, we first down-sampled the data set of Rabbit and Foot and used these down-sampled data sets for surface reconstruction. Then, we measured the difference between the reconstructed surface and the original data set (the data before down-sampling). More specifically, we selected the surface points, not included in the down-sampled data set, from the original data set, and computed the volume of the tetrahedrons constructed by the surface points and their corresponding mesh of the reconstructed surface. We summed all these volumes to perform error analysis. To make the error measurement more clear, an accurate rate was computed to specifically evaluate the performance of reconstruction accuracy of a surface reconstruction algorithm. The accurate rate is defined as:

$$\delta = (1 - \frac{\sum_{i=1}^{n} v_i}{V_{object}}) \times 100\% \qquad (9)$$

where δ denotes the accurate rate, V_{object} represents the estimated volume of the object, v_i is the volume of a tetrahedron, and n is the number of the surface points not used for surface construction.

During error analysis, we first performed the error measurement tests to the down-sampled data sets (1/3 of the original data size in our study) without any noise. The results show all algorithms (the proposed algorithm, GKS, and BPA) have 99% accurate rate on surface reconstruction. Then, we added noise to the down-sampled data sets with different noise levels (5%, 10%, 15%, 20%, 25%, and 30%), performed surface reconstruction with the proposed algorithm, GKS, and BPA respectively, and calculated the accurate rate of the reconstructed results for error measurement. The measurement results are shown in Fig. 17. From the figure, we can see that the proposed algorithm has the best performance. Note that Cocone is not used for error measurement due to its sensitivity to noise.



(a) Error measurement for Rabbit II **(b) Error measurement for Foot**

Fig. 17 Error measurement

4.6. Triangulation speed

Besides its efficiency and practice, another significant feature of the proposed algorithm is its fast processing speed. Since the proposed algorithm is a one-pass operation for surface construction, its computational complexity for triangulation operation is $O(n)$, where n is the number of input points. Because we apply priority queue during reference selection, the complexity of identifying a reference point for a front edge is $O(\log m)$, where m is the average number of the candidate points selected for reference selection. Therefore, the cost of the entire process of finding candidate point during triangulation is $O(n \log m)$. For BPA [17], its complexity for triangulation operation is

$O(kn)$, where k is the loop number of applying BPA for non uniform sampled data. Its overall complexity on finding reference for triangulation is $O(kmn)$, where m is the average number of the points on the pivoting ball. The complexity of GKS [16] for triangulation operation is $O(kn)$, where k is the loop number for non uniform sampled data. Although the overall complexity of GKS on finding reference for triangulation is $O(kmn)$ close to the complexity of BPA, GKS spends longer time on finding Delaunay neighbors for each triangulation than our proposed algorithm and BPA. The entire complexity of Cocone [11, 12] for triangulation operation is $O(n\log n)$. The processing time of Cocone on 3-D Voronoi diagrams or Delaunay tetrahedrizations for every triangulation operation is much longer than that of the other methods. Among the four methods, the proposed algorithm is the simplest and fastest one. Furthermore, several strategies are also incorporated into our proposed algorithm to reduce its computational cost further. For example, hash table is used to have the searching time of finding k-nearest-neighbor as a constant time, $O(k)$. We also pre-build a lookup table in program to store the cosine values we often use.

For both small and medium surface data sets, the triangulation speed of using the proposed algorithm is in the range of 39K to 52K triangles per second (a real-time processing speed) as shown in Table 1.Due to the high cost of floating computation and huge memory occupation for the large-size data sets, such as fan-blade, dragon, and Buddha, triangulation rate decreases from 45K to 30K triangles per second (still an acceptable processing speed). In addition, Table 1 shows that the triangulation speed is also affected by the geometric and topological complexity of an object. For comparison, BPA, GKS, and Cocone were applied to repeat the twelve tests listed in Table 1. The comparable results are listed in Table 2. From the table, we can see that the proposed algorithm is faster than BPA, and much faster than Cocone and GKS. The experimental results obtained from the multi-thread implementation are listed in Table 3. The experimental results in Table 3 show that the processing speed can be improved significantly by parallel implementation. In these tests, processing time is reduced by 30% averagely with a two-thread implementation, and 60% averagely with a four-thread implementation.

Table 1: Run-time performance of the priority driven algorithm including I/O times on a PC with 2.6GHz processor

Model	Input (point)	Output (face)	Time (sec)	Rate (k Δ /sec)
Rabbit I	8,171	16,300	0.33	49.4
Rabbit II	35,947	71,669	1.47	48.8
Horse	48,485	96,507	2.07	46.6
Club	16,864	33,617	0.64	52.5
Head	12,772	24,405	0.55	44.4
Foot	10,010	19,750	0.46	42.9
Oilpmp	30,937	61,617	1.31	47.4
Hypersheet	6,752	11,709	0.30	39.0
Knot	10,000	19,317	0.47	41.1
Fan-blade	441,477	866,137	31.35	27.6
Dragon	437,645	774,141	26.88	28.8
Buddha	543,652	981,595	43.82	22.4

Table 2: Performance comparison with other methods (Cocone, GKS, and BPA algorithm)

Model	Priority Driven (The proposed method)		Cocone		GKS		BPA	
	Time (sec)	Rate (k Δ /sec)	Time (sec)	Rate (k Δ /sec)	Time (sec)	Rate (k Δ /sec)	Time (sec)	Rate (k Δ /sec)
Rabbit I	**0.33**	**49.4**	0.96	16.77	5.72	2.84	0.46	35.4
Rabbit II	**1.47**	**48.8**	2.94	24.22	27.01	2.65	1.88	41.0
Horse	**2.07**	**46.6**	4.58	20.94	41.62	2.31	2.54	37.8
Club	**0.64**	**52.5**	1.27	26.20	12.71	2.62	0.86	38.4
Head	**0.55**	**44.4**	1.12	21.44	9.11	2.66	0.97	24.7
Foot	**0.46**	**42.9**	0.86	22.83	7.23	2.73	0.63	30.2
Oilpmp	**1.31**	**47.4**	3.05	20.2	31.3	1.97	1.69	36.1
Hypersheet	**0.30**	**39.0**	0.62	18.32	4.92	2.39	0.43	25.6
Knot	**0.47**	**41.1**	0.86	22.26	9.89	1.98	0.74	25.7
Fan-blade	**31.35**	**27.6**	74.86	11.57	787.40	1.10	37.72	23.0
Dragon	**26.88**	**28.8**	59.23	13.02	664.27	1.16	31.73	24.4
Buddha	**43.82**	**22.4**	93.30	10.51	913.21	1.09	52.89	18.6

Table 3: Performance of applying multi-threads

Model	Single thread		Two threads		Four threads	
	Time (sec)	Rate (k Δ /sec)	Time (sec)	Rate (k Δ /sec)	Time (sec)	Rate (k Δ /sec)
Rabbit I	0.33	49.4	0.23	69.58	0.14	117.62
Rabbit II	1.47	48.8	1.06	67.78	0.63	113.49
Horse	2.07	46.6	1.43	67.54	0.85	113.66
Club	0.64	52.5	0.44	75.00	0.26	128.05
Head	0.55	44.4	0.39	60.82	0.23	105.71
Foot	0.46	42.9	0.34	60.09	0.18	110.00
Oilpmp	1.31	47.4	0.89	70.75	0.51	121.54
Hypersheet	0.30	39.0	0.21	56.52	0.12	97.50
Knot	0.47	41.1	0.33	58.71	0.19	100.24
Fan-blade	31.35	27.6	19.44	44.52	10.03	86.25
Dragon	26.88	28.8	16.93	45.71	8.87	87.27
Buddha	43.82	22.4	26.73	36.72	13.15	74.67

5. Conclusions

In this paper, a priority driven based algorithm for surface reconstruction is presented. As we develop a priority driven strategy to force the mesh to grow in a reliable manner, and design efficient topological operations and triangulation criteria for each step of mesh growing, the proposed algorithm can work efficiently for various data sets with different sampling rates, complicated geometries and/or topologies. The experimental results demonstrate the efficiency of the proposed algorithm. Future work includes continuous improvements of overall performance and the possibility of high-speed processing with modern, programmable GPU hardware.

6. Acknowledgements

The authors would like to thank the computer graphics lab of Stanford University and the graphics and imaging lab of the University of Washington for providing 3D surface data sets.

References

[1] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, second edition, PWS, 1999.

[2] H. Edelsbrunner, *Geometry and Topology for Grid Generation*, Cambridge University Press, 2001.

[3] G. Turk and M. Levoy, "Zippered Polygon Meshes from Range Images," *ACM SIGGRAPH*, pp. 311-318, 1994.

[4] W. E. Lorensen, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *ACM Trans. Computer Graphics*, Vol. 21(4), pp. 163-169, July, 1987.

[5] C. Bajaj, F. Bernardini, and G. Xu, "Reconstructing Surfaces and Functions on Surfaces from Unorganized 3D Data," *Journal of Algorithmica*, Vol. (19), pp. 243-261, 1997.

[6] H. Edelsbrunner and E. Mucke, "Three Dimensional Alpha Shapes," *ACM Transaction on Graphics*, Vol. 13(1), pp. 43-72, 1994.

[7] D. Attali, "r-regular Shape Reconstruction from Unorganized Points," *ACM Symposium on Computational Geometry*, pp. 248-253, 1997.

[8] J. D. Boissonnat, "Geometric Structures for Three-dimensional Shape Representation," *ACM Transaction on Graphics*, Vol. 3(4), pp. 266-286, 1984.

[9] R. C. Veltamp, "Boundaries through Scattered Points of Unknown Density," *Journal of Graphical Models and Image Proceeding*, Vol. 57(6), pages 441-452, 1995.

[10] N. Amenta, M. Bern, and M. Kamvysselis, "A New Voronoi-based Surface Reconstruction Algorithm," *ACM SIGGRAPH*, pp. 415-421, 1998.

[11]T.K. Dey, J. Giesen, N. Leekha, and R. Wenger, "Detecting boundaries for surface reconstruction using co-cones," *Int'l Journal of Computer Graphics and CAD/CAM*, vol. 16, pp. 151-159, 2001.

[12] N. Amenta, S. Choi, T. K. Dey, and N. Leekha, "A simple algorithm for homemorphic surface reconstruction," *International Journal of Comput. Geom. & Applications*, Vol. 12, pp. 125-141, 2002.

[13] H. Hoppe, T. Derose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface Reconstruction from Unorganized Point Clouds," *ACM SIGGRAPH*, pp. 71-78, 1992.

[14] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," *ACM SIGGRAPH*, pp. 303-312, 1996.

[15] R. Mencl and H. Muller, "Interpolation and Approximation of Surfaces from Three-dimensional Scattered Data Points," *State of the Art Reports, Eurographics*, pp. 51-67, 1998.

[16] M. Gopi, S. Krishnan, and C. Silva, "Surface Reconstruction using Lower Dimensional Localized Delaunay Triangulation," *Eurographics*, Vol. 19(3), pp. 467-468, 2000.

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

[18] P. Crossno and E. Angel, "Isosurface Extraction using Particle Systems," *IEEE Proc. Visualization*, pp. 495-498, 1997.

[19] J. A. Sethian, *Level Set Methods and Fast Marching Methods*, 2nd edition, Cambridge University Press, 1999.

[20] S. J. Osher and R. P. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer Verlag, Nov., 2002.

[21] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software*, Vol. 3(3), pp. 209-226, 1977.

[22] S. Arya, D. M. Mount, N. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching in fixed dimensions," *Proceeding of 5th ACM-SIAM symposium Discrete Algorithms*, pp. 573-582, 1994.

[23] P. Cignoni, C. Montani, and R. Scopigno, "A comparison of mesh simplification algorithms," *Computers & Graphics*, 22(1), pp. 37-54, 1998.

[24] P. S. Heckbert and M. Garland, "Survey of polygonal surface simplification algorithms," *SIGGRPAH* 97, Course Notes 25, 1997.

[25] M. Shafae and R. Pajarola, "DStrips: dynamic triangle strips for real-time mash simplification and rendering," *Proceeding of Pacific Graphics 2003, IEEE*, pp.271-280, 2003.

[26] H. Hoppe, "Progressive meshes," *Proceeding SIGGRAPH 96, ACM SIGGRAPH*, pages 99-108, 1996.

[27] H. Hoppe, "View-dependent refinement of progressive meshes," *Proceeding SIGGRAPH 97, ACM SIGGRAPH*, pp.189-198, 1997.

[28] J. Kim and S. Lee, "Truly selective refinement of progressive meshes," *Proceeding of Graphics Interface 2001*, pp.101-110, 2001.

[29] R. Pajarola, C. DeCoro, "Efficient Implementation of Real-Time View-Dependent Multiresolution Meshing," *IEEE Trans. on Visualization and Computer Graphics*, Vol. 10, No. 3, pp. 353-368, 2004.

[30] B. Guo, J. Menon, and B. Willette, "Surface Reconstruction Using Alpha Shapes," *Computer Graphics Forum*, Vol. 16, No. 4, pp.177-190, 1997.

[31] H.K.Zhao, S. Osher, and R. Fedkiw, "Fast Surface Reconstruction Using the Level Set Method," Proceedings of *IEEE Workshop on Variational and Level Set Methods in Computer Vision (VLSM 2001)*, 2001.

[32] J. O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1998.

[33] R. Chaine, "A geometric convection approach of 3-D reconstruction," *In Proc. Eurographics Symposium on Geometry Processing*, pp.218–229, 2003.

[34] H. Q. Dinh, G. Turk, and G. Slabaugh, "Reconstructing surfaces by volumetric regularization using radial basis functions," *IEEE Trans. Pattern Anal. Machine Intell*., pp.1358–1371, 2002.
[35] N. Kojekine, V. Savchenko, and I. Hagiwara, "Surface reconstruction based on compactly supported radial basis functions," *In Geometric modeling: techniques, applications, systems and tools*, pp.218–231. Kluwer Academic Publishers, 2004.
[36] B. S. Morse, T. S. Yoo, P. Rheingans, D. T. Chen, and K. R. Subramanian, "Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions," *In Shape Modeling International*, pp. 89–98, 2001.
[37] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien, "Spectral surface reconstruction from noisy point clouds," *In Symposium on Geometry Processing*, pp.11–21, ACM Press, July 2004.
[38] A. C. Jalba and J. B. T. Roerdink, "Efficient surface reconstruction using generalized coulomb potentials," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 13, No. 6, pp. 1512-1517, 2007.
[39] W. Saleem, O. Schall, G. Patane, A. Belyaev, and H. Seidel, "On stochastic methods for surface reconstruction," *The Visual Computer: International Journal of Computer Graphics*, Vol. 23, No. 6, pp. 381-395, 2007.
[40] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," *Eurographics Symposium on Geometry Processing*, 2006.
[41] M. B. Vieira, P.P. Martins Jr., A.A. Araujo, M. Cord, and S. Philipp-Foliguet, "Smooth surface reconstruction using tensor field as structuring elements," *Computer Graphics Forum*, Vol. 23, No. 4, pp. 813-823, 2004.
[42] M. Yoon, Y. Lee, S. Lee, I. Ivrissimtzis, and H.-P Seidel, "Surface and normal ensembles for surface reconstruction," *Computer-Aided Design*, 39, pp. 408-420, 2007.
[43] C.-C Kuo and H.-T Yau, "A new combinatorial approach to surface reconstruction with sharp features," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 1, pp. 73-82, 2006.
[44] X. Yang, "Surface interpolation of meshes by geometric subdivision," *Computer Aided-Design*, Vol. 37, pp. 497-508, 2005.
[45] S. Hahmann and GP. Bonneau, "Polynomial surface interpolating arbitrary triangulations," *IEEE Transactions on Visualization and Computer Graphics*, 9(1), pp. 99-109, 2003.
[46] N. Dyn, D. Levin, and D. Liu "Interpolatory convexity-preserving subdivision schemes for curves and surfaces," *Computer-Aided Design*, Vol. 24(4), pp.211-216, 1992.
[47] X. Li, C. Han, and W. G. Wee, "Surface Reconstruction of 3D Objects," *Int. Conf. on Computer Vision and Graphics (ICCVG2004)*, pp. 642-647, 2004.
[48] J. Huang and CH. Menq, "Combinatorial manifold mesh reconstruction and optimization from unorganized points with arbitrary topology," *Computer-Aided Design*, Vol.34, pp. 149-65, 2002.
[49] H. Lin, C. Tai, and G. Wang, "A Mesh Reconstruction Algorithm Driven by Intrinsic Property of Point Cloud," *Computer-Aided Design*, Vol. 36, pp. 1-9, 2004.