Technical Report 2008-004
A Digital Topology-based Method for the Topological Filtering of a Reconstructed Surface

D. Li[a], M. Allili[b]

[a]Université de Sherbrooke
Department of Computer Science
Sherbrooke, Qc J1K2R1, Canada

[b]Bishop's University
Department of Computer Science
Sherbrooke, Qc J1M 1Z7, Canada

ABSTRACT

In this paper, we use concepts from digital topology for the topological filtering of reconstructed surfaces. Given a finite set S of sample points in 3D space, we use the voronoi-based algorithm of Amenta & Bern[1] to reconstruct a piecewise-linear approximation surface in the form of a triangular mesh with vertex set equal to S. A typical surface obtained by means of this algorithm often contains small holes that can be considered as noise. We propose a method to remove the unwanted holes that works as follows. We first embed the triangulated surface in a volumetric representation. Then, we use the 3D-hole closing algorithm of Aktouf & al.[2] to filter the holes by their size and close the small holes that are in general irrelevant to the surface while the larger holes often represent topological features of the surface. We present some experimental results that show that this method allows to automatically and effectively search and suppress unwanted holes in a 3D surface.

Keywords: Surface reconstruction, Voronoi filtering, Volumetric representation, Topological filtering, Digital topology, Closing holes.

1. INTRODUCTION

Many applications in graphics and imaging such as the extraction of iso-surfaces from scalar functions defined on regular grids, object modeling, visualization, and 3D segmentation require reconstruction of surfaces from 3D sample points. As a result of noise and other artifacts, surfaces obtained by standard algorithms often suffer from topological irregularities and geometric noise. In this paper, we introduce an approach based on digital topology to remove the unnecessary nontrivial topology from reconstructed meshes.

Surface reconstruction has been widely studied in computer graphics, computer vision, and computational geometry. Previous work on surface reconstruction falls into two categories: the volumetric approaches and the sculpturing methods. The volumetric approaches are based on the determination of a distance function that can be seen as an implicit function, where the surface sought is given as the iso-surface with iso-value 0 of the distance function.[3–5] The sculpturing methods for surface reconstruction are based on the concepts of Voronoi diagrams and Delaunay triangulations. Among popular algorithms in the second category, there is an early algorithm due to Boissonnat,[6] which suggested to use Delaunay triangulations to form a structure that represents an approximation surface to a given point set. Edelsbrunner et al.[7] introduced the most famous computational geometry construction that associates a polyhedral shape called α-shape with an unorganized set of points. The α-shape is a subcomplex of the Delaunay triangulation obtained by removing the simplices with circumsphere

Further author information: (Send correspondence to M. Allili.)
M. Allili: E-mail: mallili@ubishops.ca, Telephone: 1 819 822 9600 2740.

radius greater than α. More recently Amenta and Bern[1] contributed a new algorithm for the reconstruction of surfaces from unorganized 3D sample points drawn from a smooth surface. In its implementation, the convex hull and Voronoi diagram of the sample points are first computed. Then, two Voronoi vertices called poles are associated to each sample point. At this point, the Delaunay triangulation is computed for the extended set of points containing the original sample points and the poles. Finally, by Voronoi filtering which eliminates triangles with circumspheres containing poles, and by normal filtering that eliminates any triangle forming a too large angle between its normal and the pole vector at a vertex of the triangle, a piecewise linear surface approximating the original smooth surface is obtained as a subset of the Delaunay triangulation.

Given a set of unorganized 3D sample points, we will make use of the algorithm of Amenta and Bern to reconstruct an approximation surface. Typically, the surfaces reconstructed by means of this algorithm often contain unwanted small holes that can be considered as noise. Identifying and closing these holes automatically is very useful in many applications. We use the three-dimensional holes closing algorithm described in[2] to process the reconstructed surface for topological correctness. In order to use this algorithm on our data, several preprocessing steps are necessary. We first embed the reconstructed surface in a volumetric representation consisting of voxels with a fixed resolution. The resolution can be refined if needed. Then, the distance transform[8] is computed to control the size of the holes that should be suppressed. More precisely, we find the minimum box that encloses the surface. Then, we decompose the box into a uniform grid of cubes or voxels. We scan all the voxels of the grid in the descending order of the distance transformation value and delete all the cubes according to the distance or the topological number. The remaining cubes in the grid are the ones in which the undesired holes have been closed. This construction presents a number of advantages. First, embedding the surface in a volumetric representation allows to automatically fill in the small holes in the surface and to focus on the more relevant ones. In addition, the more relevant topological features of a surface are more easily detected in a cubical structure than in a triangular mesh. Moreover, once the volumetric representation is filtered, one can use techniques such as marching cubes to extract a triangle mesh from the volumetric representation that exhibits the corrected topology.

The hole closing algorithm is based on some basic concepts of digital topology that allow to define the notion of a hole and make a difference between holes, cavities, and concavities. At last, we present some experimental results that show that this method allows to automatically and effectively search and suppress unwanted holes in a 3D surface.

2. THE ALGORITHM OF SURFACE RECONSTRUCTION

Surface reconstruction is a very important and well known problem that is extensively studied in computer graphics, computer vision, and computational geometry. In many applications in the cited domains, objects are only known by the three coordinates of a set of points selected on their boundaries and often need to be rebuilt only from the knowledge of these points. Many people have made great contributions about this problem in the last decade. A very famous approach to this problem consists of using the computational geometry structures of Delaunay triangulations and Voronoi diagrams. It has been proved to be very successful and with guarantee homeomorphism between the reconstructed surface and the original one from which the sample points are drawn. The advantages and the importance of using Delaunay triangulations and Voronoi diagrams for surface reconstruction can be found in the paper of Jean-Daniel Boissonnat.[6] Boissonnat pointed out that using only a list of three coordinates of points set on the boundary of a given object leads to a poor representation of the object. A structure of a graph (whose vertices are the given points and edges join points that are related in some sense) on the set of points is needed in order to make explicit the proximity relationships between the points. Among the different possible structures, the simplest ones, that is those with the fewest number of edges and which allow to preserve the topology of the surface, are of crucial importance since they allow to simplify the problem and any other structure can be obtained from them. In the general case, the simplest structure is a simplicial polyhedron.[9] Boissonnat advocated the use of the Delaunay triangulation and Voronoi diagrams to associate a global structure to the points from which the simplest ones and other features of the surface can be obtained. Many other algorithms have been devised following Boissonnat's idea of using the Delaunay triangulation, however all of them require the use of uniform or organized sample of points.

In 1998, Amenta and Bern[1, 10] published the first algorithm for surface reconstruction from scattered sample of points that does not formally require a uniform or organized sample of points. The sampling can depend on the local feature size and can be highly nonuniform, dense in detailed areas yet sparse in featureless ones. In this section, we discuss the implementation of Amenta and Bern's algorithm with the aim of providing a complete visualization for several examples and outlining several difficulties and problems encountered, especially the issues related to the presence of unwanted holes in the surface. We start by defining some concepts and then we proceed with a brief description of the algorithm.

For an $(n-1)$-dimensional manifold in $\mathbb{R}^n$, we recall that the *medial axis* is the set of points with at least two closest points on the manifold.

The *local feature size* $LFS(p)$ at a point p on a surface F is the (Euclidean) distance to the nearest point of the medial axis of F.

A set S of points on the surface F is an *r-sample* of F if every point p on F is at a distance less or equal than $r.LFS(P)$ from a point of F.

The algorithm of Amenta et al. in $\mathbb{R}^3$ is an extension of their algorithm of reconstructing curves in the plane.[11] In $\mathbb{R}^2$, if S is the set of sample points of a curve and V consists of the vertices of the Voronoi Diagram of S. The reconstructed curve consists of the edges of the Delaunay triangulation of $S \cup V$ with both endpoints in S. In the extended version for surface reconstruction, not all the Voronoi vertices are used. These vertices may lie very close to the sample surface and cause the presence of holes in the reconstructed surface. The concept of *poles* is therefore introduced.

The *Poles* of a sample point s are the two farthest vertices of its Voronoi region one on the outside and the other on the inside of the surface that pass through the site of the region. The vector from the site to the pole is called the *pole vector*. The pole on the outside of the surface is labeled $p+$ and the opposite one is $p-$.

Unlike the algorithm of curve reconstruction, the surface reconstruction algorithm makes use only of the poles which are a subset of the Voronoi vertices. The following pseudocode summarizes the main steps of the reconstruction algorithm.

Input: A set S of sample points
Output: Filtered triangles (reconstructed surface) from the delaunay triangulation of S
1. Compute the convex hull $CH(S)$ of S;
2. Compute the Voronoi diagram $VD(S)$ of S;
3. For each sample point s, find its positive and negative poles (p_i+ and p_i-);
4. Compute the Delaunay triangulation of $S \cup P$, where P are all poles at finite distance;
5. ($Voronoifiltering$) Extract only the triangles of the Delaunay triangulation with all three vertices in S;
6. ($Normalfiltering$) Eliminate any triangle T for which the angle of the normal to the triangle and the pole vector to $p+$ at a vertex of T is too large.
7. ($Trimming$) Orient all the triangles and poles consistently. Discard the sharp edges.

Algorithm 1: Surface reconstruction Algorithm

Sampling. Clearly no algorithm can reconstruct any surface from any set of samples; we need some condition on the quality of S. The assumption is that the sample points set S comes from a compact surface F that has no boundary and it is an *r-sample* of F. This condition has the nice property that less detailed sections of the surface do not have to be sampled as densely as the ones with more features. The data sets that meet the above standard can be used to obtain an initial approximation (CRUST) to F. That is, the three-dimensional Delaunay triangulation of r-samples contain a piecewise-linear surface homeomorphic to F. It was proved in[1, 10] that any r-sample S of F with $r \leq 0.1$, the approximation surface includes all the Delaunay triangulation with all three vertices belonging to S. If $r \leq 0.06$, the approximation surface lies within a thick surface formed by placing a ball of radius $5rLFS(p)$ around each point $p \in F$.

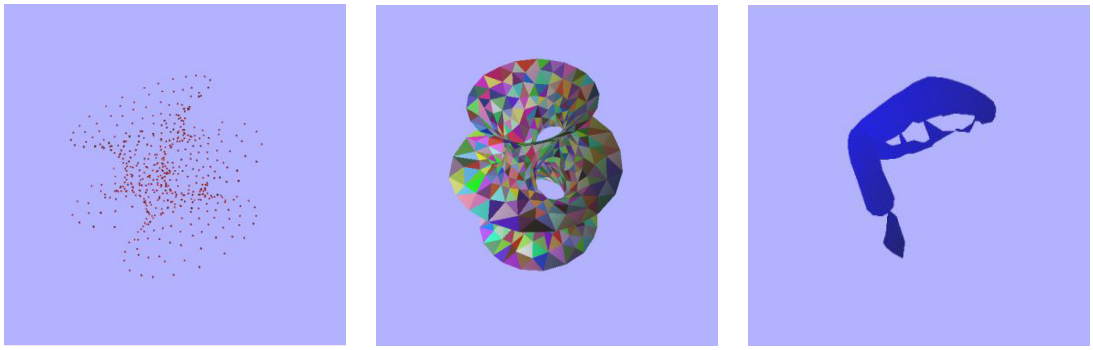
Voronoi and Normal Filtering. The first 4 steps of the algorithm compute the crust of the surface that is pointwise convergent to F as the sampling density increases. The algorithm may output some very thin crust triangles nearly perpendicular to the surface. A 3D Delaunay triangulation is formed of tetrahedrons, not

triangles. However, the paper[1] mentions triangles. Actually, the triangles used are the faces of each tetrahedron. For each tetrahedron returned by the Delaunay triangulation, one must extract the triangles in which all three vertices belong to the sample points set S. An imporatnt result in[10] states that the vectors $n+ = sp+$ and $n- = sp-$ from a sample point to its poles are guaranteed to be nearly orthogonal to the surface at s. We can eliminate any triangles whose normals having big differences from $n+$ or $n-$. When normal filtering is used, the normals of the output triangles approach the surface normals as the sampling density increase[10] and the remaining set of triangles still contain a subset forming a piecewise-linear surface homemorphic to F. Normal filtering can be dangerous, however, at boundaries and sharp edges. The directions of $n+$ and $n-$ are not nearly normal to all nearby tangent planes, and desirable triangles might be deleted.[10] After the normal filtering, the extracted Delaunay triangles are roughly parallel to F.

Topological issues. The normal filtering can only guarantee that the direction of the approximate surface F' is the same as of real surface F. But in some cases, extra triangles enclosing small bubbles and pockets, may exist, since some flat sliver with four faces pass through the Voronoi and Normal filtering. To ensure the topological equivalence, filtering by trimming is needed. Before trimming, we have to orient triangles. To do this, we can choose a sample point $p \in S$ that is on the Convex hull $CH(S)$ of S. Suppose the normal pointing out of the surface is positive. We can use the direction of $pp+$ as the positive normal of F, since the Voronoi Diagram cell which includes a site on the $CH(S$ is an unbounded cell and face outside. Then, we can orient any triangle T that is incident to p by checking the angle between $pp+$ and the T's face normal. If the angle is greater than $\pi/2$, we flip T's face normal. We make the rest of vertices' poles agree with p's setting. This can be recursively repeated until all the poles and triangles are oriented. At last, the surface's orientation is in principle consistent. Normally, small bubbles and pockets are always connected to the *sharp edge*, that is, an edge which has a dihedral angle greater than $3\pi/2$ between a successive pair of incident triangles in the cyclic order around the edge with consistent orientation.[10] In the step 7 of the algorithm, we can breadth-first search on triangles to extract triangles by removing triangles with sharp edges. This makes each edge bounds at least two triangles.

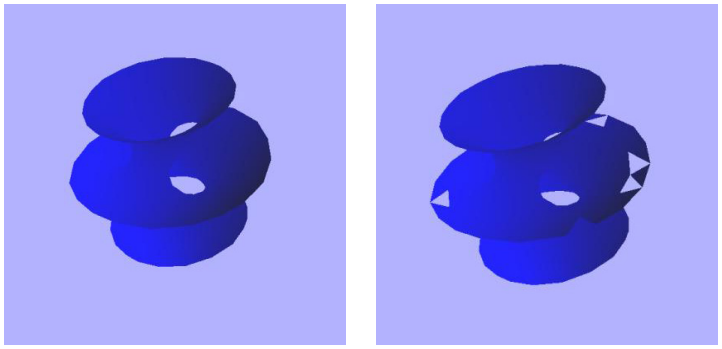
Implementation. We implemented steps 1-6 of the CRUST algorithm. Microsoft MFC and openGL were used to do the reconstruction and visualization. We used Barber's Qhull library[12] to compute the Convex Hull, the Voronoi Diagram and the Delaunay triangulation. The routine that computes the convex hull outputs a list of vertices on the convex hull and the normals to triangles on the convex hull. The Voronoi diagram program provides a list of Voronoi regions and the vertices in each region. Delaunay triangulation gives all the tetrahedrons for 3D. The library gives almost all the data needed about the computational geometry structures introduced in Chapter 2. In step 1, we marked the sample points (sites) that are on the convex hull and calculated the average normal of the triangles incident to them. In step 3, we computed poles for each vertex. In calculating the pole $p+$ for the unbounded Voronoi regions, we just added the unit normal of the vertex's normal. Since these poles will not be added to the set of vertices used to compute the Delaunay triangulation, this does not affect the triangulation. They are only used in the normal filtering. In step 5, we extracted the triangles in which all three vertices are in the original sample points set S. In step 6, since we do not know if the normal of a triangle is oriented correctly or not, the angle of the triangle's face normal and the pole vector chosen is actually the smaller one of the two angles from the positive and negative normal. We used random color to show each of the triangle of the crust. Our implementation of this algorithm runs in time $O(n^2)$ in the worst case. Qhull runs in $O(n^2)$ time. So computing the convex hull, Voronoi diagrams and Delaunay triangulation (at most $3n$ points) takes $O(n^2)$ time. In Voronoi filtering, triangles of each tetrahedron are processed. There are $O(n^2)$ triangles, so it takes time $O(n^2)$. The same analysis holds for normal filtering as well.

Results. We used several data sets at our disposal. There is an interface information for each data set. So, we can first visualize the surface with this information. Then, we discard the interface information and use the CRUST algorithm to reconstruct the surface. The data sets are not necessarily r-samples of the surfaces. So the guarantee of correctness does not work. Some of the results are not satisfactory at all. In the reconstruction of the hyper sheet model, the Amenta's algorithm works very well by comparing two results. One of the reasons is that the sample points have relatively good density and nearly uniform. In Figure 1 (a), we used the given interface information to reconstruct the surface. In Figure 1 (b), we used only the sample points information and Delaunay triangulation to do the reconstruction. Except for the hypersheet, all reconstructed surfaces using the CRUST algorithm are full of holes. For example, in Figure 1 (c), the result of the reconstruction of a golf



(a) (b) (c)

Figure 1. Examples of cell complexes. (a) Hyper sheet sample points with relatively good density and nearly uniform. (b) Reconstructed hyper sheet surface using the algorithm. (c) Some holes exist on the reconstructed surface of the golf club.



(a) (b)

Figure 2. (a) Hyper sheet sample points with 180 degree angle of normal filtering. (b) Reconstructed hyper sheet surface using the algorithm. (c) Hyper sheet sample points with 172 degree angle of normal filtering.

club has obvious holes on the surface. Although the sample point sets are not necessarily r-samples, some of them are quite dense, yet the results are not very satisfactory. As for the importance of normal filtering, the reconstructed hyper sheet without normal filtering or with 180 degree looks perfect (see Figures 2 (a) and 2 (b)). But if we set an angle less than 180 degrees to do the normal filtering, some good triangles will be eliminated at the same time as some pockets. This creates several holes in the surface.

Discussion. From the test results, we believe that the sampling condition is crucial to the reconstruction result. If the set of sample points is dense and almost uniform then the CRUST algorithm will give a very good result in general. However, when the sample point is not dense then it is hard to test the r-sample condition and then the reconstruction is very sensitive and the filtering may lead to several holes in the surface. The normal filtering is the most sensitive to the density of the sampling, since in the CRUST algorithm, the pole vector is used to represent the local surface normal which may differ completely from the real normal to the surface.

3. TOPOLOGICAL FILTERING OF RECONSTRUCTED SURFACES

In this section, we develop a method based on digital topology for the topological filtering of a reconstructed surface. We first find a volumetric representation of the surface and use it to track unwanted holes in the structure using the hole closing algorithm in.[2] We refer the reader to the following works[13–15] for the basic notions of digital topology used in this section. First, embedding the surface in a volumetric representation automatically fills in small holes in the surface. So, one can discard studying those holes and focus on the more relevant ones. In addition, the more relevant topological features of a surface are more easily detected, and their sizes are more easily estimated, and processed in a cubical structure than in a triangular mesh. Once the volumetric

representation is filtered and the topological noise is eliminated, one can use techniques such as marching cubes to extract a triangle mesh from the volumetric representation that exhibits the corrected topology.

Volumetric Representation of a Surface. The method used to embed a triangular surface into a discrete volumetric cubical volume follows the following steps.

1. Put the smallest cube that encloses the entire surface into the collection L;
2. Check the cube size

   if it is less or equal to the given size, stop and save the result;

   else continue.
3. Divide the cube into 4 smaller equal sized cubes;
4. Check the intersection of these cubes and the surface triangles:

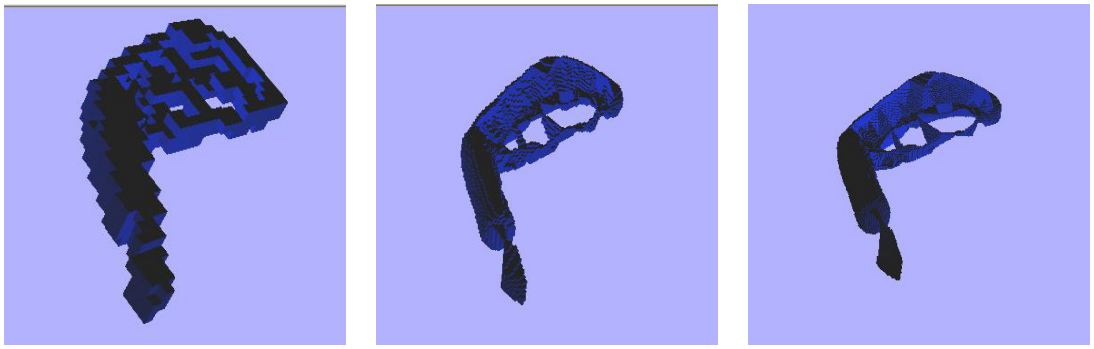
   If there is an intersection, save the smaller cubes into L;

   If there is no intersection, delete them.
5. Choose the largest cube from L and loop back to 2.

The key point of this method is to determine the existence of an intersection of the triangle and a voxel. Fortunately, in Graphics Gems III, Douglas Voorhies[16] gives an algorithm that detects whether a given triangle intersects the axially-aligned cube with edges of length 1 centered at the origin. Voorhies's approach uses the general approach which proceeds from cheap trivial accept and reject tests through more expensive edge and face intersection tests. Let C be a unit cube centered at the origin. The intersection algorithm is as follows.

1. Check whether the origin of C is contained in the solid obtained by dragging a unit cube centered at one vertex of the triangle to the other vertex.
2. If no intersection exists in the first step, check whether any of the four diagonals of C intersects the interior of the triangle.

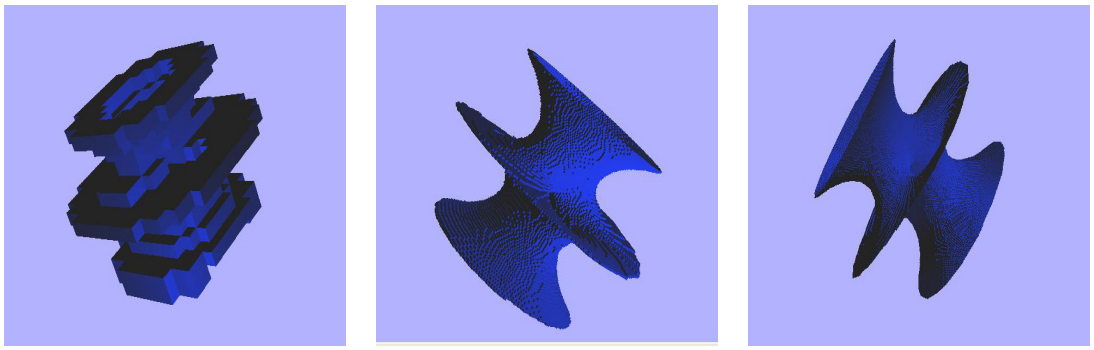
Figures 3 and 4 show the volumetric representations of the surfaces of a golf club and a hypersheet at different resolutions, where the resolution is controlled by the size of any edge of the uniform cubes that make up the cubic grids in which the surfaces are embedded.



(a) (b) (c)

Figure 3. Volumetric representations of a surface of a golf club at different resolutions (size of grid cube) (a) 0.05. (b) 0.01. (c) 0.005.

Digital distance Transformation. The distance between a point p and a given object is the minimum among all the distances from p to all the points of the object. In other words, it is the distance from p to the nearest points belonging to the object. Since the computational cost of the exact Euclidean distance transform is relatively high, the digital distance transform is used to approximate it in 3 dimensional digitized space. To avoid computing the distances between one point to every point on the object every time, a point and a distance



(a) (b) (c)

Figure 4. Volumetric representations of a surface of a hypersheet at different resolutions (a) 0.1. (b) 0.01. (c) 0.005.

mapping can be pre-computed and maintained. The distance of a given point p can be retrieved from the mapping quickly. the process of building this mapping is called the digital distance transformation. Formally, the *distance transformation* (often called DT) is the process that computes the distance map from every point outside a specific object to the object in a binary image. More precisely, DT can be expressed in a mathematical way. Suppose an object lies in a $M \times M \times M$ cubic grid B, where M is a positive integer. X and $\overline{X}$ are represent the foreground and background of the object respectively. $DT(i, j, k)$ denotes the digital distance from a given voxel to the object, where $i,\ j,\ k$ represent distances in the different axes directions and they are in increasing order $0 \leq k \leq j \leq i \leq M$. The distance transformation can be expressed as follows[8]:

$$DT(i,j,k) = min\{D(max\{|i-p|, |j-q|, |k-r|\}, mid\{|i-p|, |j-q|, |k-r|\}, min\{|i-p|, |j-q|, |k-r|\}), p, q, r \in \overline{X}\}$$

where max, mid and min represent the maximum, the middle and minimum of the three values respectively. D is the distance between two pixels through a minimal path between them. Since we are in a digital image, the computation of the distance is quite different from the Euclidean distance. We use a discrete distance to approximate the Euclidean distance. We introduce now some DT computation methods. The basic idea[8] of a digital DT is to approximate the global Euclidean distance computation with repeated propagation of local distances within a small neighborhood range. Here, the *local distance* refers to the distance between neighboring pixels or voxels in a small volume region (normally with size of 3 or 5). This approach is motivated by the easiness of the computation. This idea was first presented by Rosenfeld and Pfaltz in about 1966.[17]

The simplest algorithm is the *Path Generated Distance Transforms (PGDT)*. In the algorithm, the number of steps in the minimum path is used as the distance. It counts as one step from one point to its neighborhood. The neighborhood is defined using n-connectedness. Here distances of the points in the object are always set to be zero. Starting from the boundary of the object, the distances of the points (P_1) which are immediately n-connected to the object are 1. The points that have no connection to the object but are n-connected relationship to P_1 have distances 2, which is increased by one from the distance to P_1. Similarly, this can be repeated until all the points out of the object are labeled with distances. It should be noticed that when one point is labeled with different distances, we always keep the smallest distance. This method is utilized by many DT procedures and it is easy to compute sequentially. In the computation, only a small fixed neighborhood region need to be considered and the number of these neighbors is at most 8 in 2D and at most 26 in 3D. But the disadvantage of this method is that the difference existing in the distances of different kinds of neighbors is ignored. To improve that, some optimizations emerged.

Optimal distance computations. The development of a digital DT was made through several steps. At first, simply local distances accumulation were used. Then weighted local distances were also taken into account. More recently, several papers mentioned the optimal distance derived in the context of minimizing the maximum error and the unbiased mean square error. Integer approximations for the local distances are developed for neighborhood sizes of three and five. Minimization is performed over circles and spheres to preserve the symmetries of the neighborhoods.

Weighted distance transformation. In $PGDT$, the Euclidean Distances of a face, an edge and a point connected to the object are respectively 1, $\sqrt{2}$ and $\sqrt[3]{3}$, but a fixed local distance is counted. In the optimal distance computation of DT, different weights are added to various types of connections. The computation with weighted distance transformation is called WDT. In a $3 \times 3 \times 3$ voxel neighbors, the local distances of a face, an edge and a point can be set to be $a,\ b$ and c respectively. Only these three types of straight paths can possibly exist in the volume grids. After investigating all the ways these three straight paths can be approximated by paths using other steps at the conditions for the straight path to be shortest. We have some regularity criteria for $3 \times 3 \times 3$ distance transforms (see table). Suppose the computed distance starts from the origin. We choose the shortest paths and assume that the local distances have the properties: $a < b < 2a$ and $b < c < 3b/2$. These inequalities define an area regularity in parameter space. The maximum difference from the weighted Euclidean distance in an $M \times M \times M$ image was computed and minimized by optimizing local step lengths. In order to get a unique expression for WDTs, we use 2 cases to simplify the problem. Case I: $a + c \leq 2b$ and Case II: $a + c > 2b$. Borgefors[8] generated the weighted distance between the origin and (x, y, z) as follows.

- Case I:
  $D = z(c - b) + y(b - a) + xa$
  The difference between the computed distance and the true Euclidean distance is as follows: $D'_{difference} = z(c - b) + y(b - a) + xa - \sqrt{x^2 + y^2 + z^2}$
  We can minimize $D_{difference}$ and get optimal value of $a,\ b$ and c as follows:
  $a'_{optimal} \approx 1$
  $b'_{optimal} \approx 1.31402$
  $c'_{optimal} \approx 1.62803$
  with $D'_{max_difference} \approx 0.10402$
- Case II:
  The difference between the computed distance and the true Euclidean distance is given by.
  $D"_{difference1} = (z + y)(c - b) + x(2b - c) - \sqrt{x^2 + y^2 + z^2}$ if $x \leq y + z$
  $D"_{difference2} = (z + y)(b - a) + xa - \sqrt{x^2 + y^2 + z^2}$ if $x \geq y + z$
  We can minimize $D_{difference}$ and get optimal value of $a,\ b$ and c as follows:
  $a"_{optimal} \approx 1$
  $b"_{optimal} \approx 1.31178$
  $c"_{optimal1} \approx 1.62960$
  $c"_{optimal2} \approx 1.80621$
  with $D'_{max_difference} \approx 0.10245$
  The result of case II is slightly better than case I for $a = 1$.

Integer Approximation. In the above section, we get optimal local distances. However, they are in real values. It is not preferable to compute DT as real number. This can be solved by integer approximation. A simple way to do that is by scaling a real to an integer. The real optimized local distances are multiplied by a scale factor at first, then rounded to integers. Borgefors[8] did some approximation tests. The results are listed in the following table.

For instance, let three kinds of local distances of $3 \times 3 \times 3$ be $< a, b, c >$. We can scale them to $< 1, b/a, c/a >$. But b/a and c/a are not necessarily close to integers and rounding may result in losing precision. We can scale them with a scale factor F. Note that the maximum differences are not predictable in case I and case II by using different scale factors according to the test results. In addition, When the scale factor is greater than 2, the maximum difference does not decrease obviously. In case II, sometimes the local distance can vary within a relative big range but the maximum difference stays in a small range. Here, the option of <3,4,5> is a good choice for later use.

Algorithm of computing distance transforms. After the optimal local distances in the $3 \times 3 \times 3$ neighborhood size are determined, it is not difficult to compute the distance of every voxel point out of the given object. The algorithm can is as follows.

Type	a	b	c	Maximum difference	Scale
PGDT	1	-	-	1.26795	-
PGDT	1	1	-	0.41421	-
PGDT	1	1	1	0.73205	-
Case I	1	1.31402	1.62803	0.10402	-
Case II	1	1.31178	1.62960	0.10245	-
Case I/II	2	3	4	0.29289	2
Case I/II	3	4	5	0.11808	3
Case I	8	11	13	0.10732	8
Case II	13	17	22 to 23	0.10652	13
Case II	16	21	27 to 28	0.10296	16

Table 1. Integer $3 \times 3 \times 3$ Distance Transformations.

1. Suppose $M \times M \times M$ image, where $3 \leq M$. Set the value of the distance of each voxel in the object in the image to be *zero* and others to be *Infinite*;
2. Give a label *unvisited* to every voxel;
3. For i from 1 to M
   For j from 1 to M
   For k from 1 to M
   $DT(i, j, k)$ = minimum DT of n **visited** neighbors plus local distance and label it has visited.
4. Set every voxel to be **unvisited**.
5. For i from M to 1
   For j from M to 1
   For k from M to 1
   $DT(i, j, k)$ = minimum DT of n **visited** neighbors plus local distance and label it has visited.

It is not hard to notice that two loops are needed to find the digital distance transformation. Since the distance of any point on the object is set to zero and outside of the object is set to infinity initially, the points that are next to the object point can get distance by adding local distances. If we start from a background grid point, the value of the distance will not be updated until a point on the object is met. So the first scan can only update the immediate background voxels of the object. The second scan will get the values of background voxels that the first scan has not given.

Algorithm of hole closing. Based on the preprocessing above, a continuous object is discretized, i.e. the Delaunay triangulation representation of the object is embedded in a volumetric and cubical representation that is formed of a grid of voxels. Then through the computation of distance transformation, a label is assigned to every voxel, where the voxels on the Delaunay Triangulation are assigned the distances of zero. At last, we are ready to implement the algorithm of hole closing. Simply speaking, we keep a list L and a list H containing all voxels' coordinates and indexed by their DT. Then, we check the voxels in L in the descending order of the index and in the first in first out order for the index with the same values.

In the first step, the voxels with DT greater than ϵ are deleted. We obtain a thick layer of the surface with all the holes and gaps closed. In order to close the specific holes, which are the holes with their radii less than ϵ, on the surface, the topological number, which is crucial criterion in the algorithm is used to identify voxels to be eliminated. When a voxel is not on the object and its topological number equals one, it will be deleted. In other words, these voxels are on the virtual boundary of the temporary foreground and background. Normally, the voxels will be deleted until the object is reached. For the holes whose radii are greater than *epsilon*, the voxels in these holes will be eliminated. Because the DT of voxels in the center of the holes are greater than the ϵ, these voxels will be eliminated from H. The rest of the voxels in the hole and not on the object will also be deleted since their topological numbers are 1, which mean they are on the boundary. It is not difficult to observe that the voxels on the object will never be picked up from the list L since we use the restriction $DisIndex \geq 3$ as a condition, i.e., theoretically, the DT of the voxel should not be zero. The crucial and tricky problem is to stop

Input: ϵ, M ;
Output: Topological hull of the object in the image.
Set all voxels as unvisited;
Set value of DT of each voxel on the background to be "infinite" and value of DT of each voxel on the object to be 0;
Compute DT of the grid voxels;
Define a list array $L[MAXDIST]$ with the size of maximum distance of DT;
Assign the coordinates of each voxel to the list array $L[DisIndex]$ with the index equals the value of DT of the voxel. Give a true sign to $E[i, j, k]$ representing the existence of voxel $V(i, j, k)$ on the list array $L[DisIndex]$;
Set topological hull list H;
While ($DisIndex < 3$) // (3 is the minimum local distance) {
    From $L[DisIndex]$ with DisIndex from maximum value of DT (i.e.,MAXDIST), pick up each voxel $V[ii, jj, kk]$ in $L[DisIndex]$ in the order of the beginning to the end and set $E[ii, jj, kk]$ to be false.
        While ($L[DisIndex]$ is not empty) {
            Check the value of the voxel, i.e., $V[ii, jj, kk]$.value
            If ($V[ii, jj, kk]$.value greater than ϵ or topological number of $V[ii, jj, kk] == 1$) {
                Delete $V[ii, jj, kk]$ from H ;
                Add 26 neighbors which meet the following conditions.
                    1. It is in the $M \times M \times M$;
                    2. It is not in L;
                    3. It is in H;
                    4. The DT of the neighbor is greater than 0;
                Insert qualified voxel into $L[Index]$ and set its E to be true;
                // Index equals the DT of the voxel
                    If (DisIndex < Index)
                      DisIndex = Index ;
            }
        }
}
Return The rest of the voxels in H, i.e., the topological hull of the object.

Algorithm 2: Algorithm of Hole Closing In 3D Surface

deleting voxels in the holes whose radii are less than ϵ. This problem can be solved by checking the topological numbers of the voxels in the holes. This can be observed in details in Figure 5. Since the center voxel in the hole has the furthest distance, it should be selected from the list L right away. Because the hole has only one layer, taking away the voxel O results in the connection of the backgrounds on the two sides of the surface object locally. It is classified as a 2D isthmus and will be kept in the list H. Similarly, O's neighbors are also isthmuses and kept. So the single layer voxels that cover the hole will be kept in the list H.

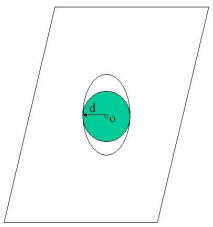
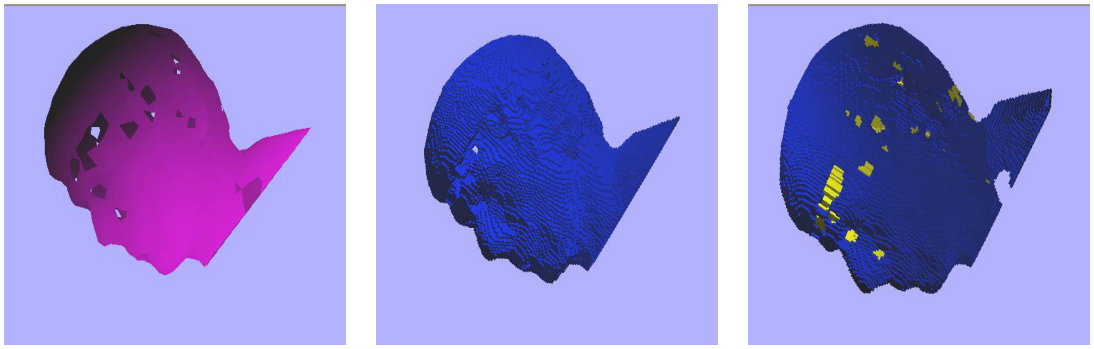


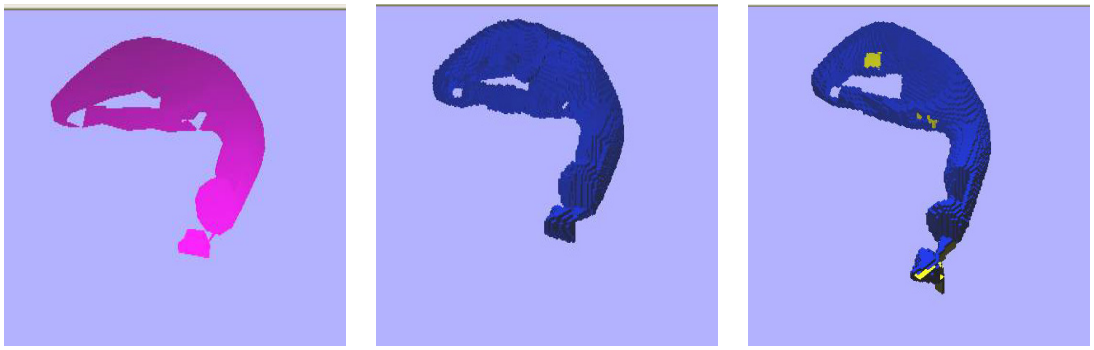
Figure 5. Suppose only one layer left and the distance d of the voxel O in the center of the hole is the furthest one to the object surface, the voxel is a 2D isthmus.

Results. The algorithms of surface reconstruction, the volumetric representation, and the hole closing are applied on two sample data sets representing the surface of a head and the surface of a golf club. The results are shown in Figures 6, 7, and 8. Figure 6(a) shows the reconstructed surface of a head using the algorithm in the previous section. The surface exhibits several unwanted holes. Figure 6(b) shows the volumetric representation of the surface in (a) in a cubic grid of resolution 0.01. Several small holes have been already closed in the representation, but several other holes remain which are closed by the hole closing algorithm when the hole diameter cannot exceed 0.1 as shown Figure 6(c). Figure 7(b) shows the volumetric representation of the surface in Figure 7(a) in a cubic grid of resolution 0.01. Several small holes have been closed and there are new holes that are created by this process. Some of the remaining holes remain are not even closed by the hole closing algorithm when the hole diameter cannot exceed 0.1 as shown Figure 7(c). Figure 8(c) shows that the hole closing algorithm closed all the holes in the golf club when the hole diameter can exceed 0.1.



(a) (b) (c)

Figure 6. (a) The reconstructed surface of a head. (b) The volumetric representation. (c) Hole closing with hole diameter less or equal than 0.1.

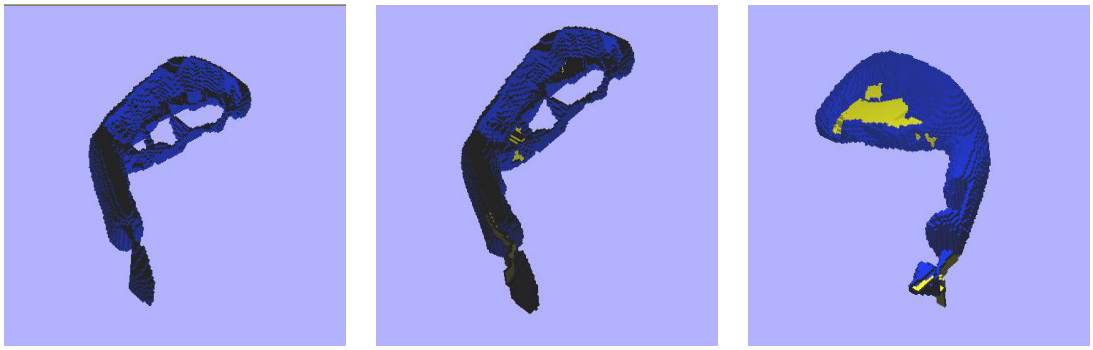


(a) (b) (c)

Figure 7. (a) The reconstructed surface of a golf club (b) The volumetric representation. (c) Hole closing with hole diameter less or equal than 0.1.

4. CONCLUSION

The main contribution of this work is to provide a novel algorithm for the topological filtering of a reconstructed surface that presents numerous advantages because it operates on a volumetric and cubical structure that contains the reconstructed mesh. First, embedding the surface in a volumetric representation automatically fills in small holes in the surface. So, one can discard studying those holes and focus on the more relevant ones. In addition, the more relevant topological features of a surface are more easily detected, and their sizes are more easily estimated, and processed in a cubical structure than in a triangular mesh. The use of a volumetric structure in which the



(a) (b) (c)

Figure 8. (a) Volumetric representation of the surface of a golf club (different view). (b) Hole closing with hole size less than or equal to 0.1 (different view). (c) Hole closing with hole diameter less or equal than 0.5.

reconstructed surface is embedded represents a way to work with a smoothed version of the reconstructed surface and a more refined structure with higher resolution can be obtained when needed. Hence, the thickness of that volumetric representation represents a degree of smoothing of the reconstructed surface. Once the volumetric representation is filtered and the topological noise is eliminated, one can use techniques such as marching cubes to extract a triangle mesh from the volumetric representation that exhibits the corrected topology.

In a work in progress, we are attempting to force the extracted triangle mesh to agree with the original reconstructed surface in the regions where topology is not corrected.

REFERENCES

1. N. Amenta and M. Bern, "Surface reconstruction by voronoi filtering," *Proceedings of the 14th ACM Symposium on Computation Geometry* , pp. 39– 48, 1998.
2. Z. Aktouf, G. Bertrand, and L. Perroton, "A three-dimensional holes closing algorithm," *Pattern Recongnition Letters* **23**, pp. 523–531, 2002.
3. H. Hoppe and al., "Surface reconstruction from unorganized points," *Proc. SIGGRAPH´ 92* , pp. 71–78, 1992.
4. H. Hoppe and al., "Piecewise smooth surface reconstruction," *Proc. SIGGRAPH´ 94* , pp. 19–26, 1994.
5. B. Curless and M. Levoy, "A volumetric method for building complex models from range images," *Proc. SIGGRAPH´ 96* , pp. 303–312, 1996.
6. J. Boissonnat, "Geometric structures for threeŋdimensional shape reconstruction," *ACM Trans. Graphics* **3**, pp. 266–286, 1984.
7. H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel, "Three dimensional alpha shapes," *ACM Trans. Graphics* **13**, pp. 43–72, 1994.
8. G. Borgefors, "On digital distance transforms in three dimensions," *Computer vision and image understanding* **64**, pp. 368–376, 1996.
9. P. Giblin, *Graphs, Surface and Homology*, Chapman and Hall, London,England, 1977.
10. N. Amenta, M. Bern, and M. Kamvysselis, "A new voronoi-based surface reconstruction algorithm," *Proceedings of the 25th annual conference on Computer graphics and interactive techniques (SIGGRAPH 98)* , pp. 415–421, 1998.
11. N. Amenta, M. Bern, and D. Eppstein, "The crust and the β-skeleton: combinatorial curve reconstruction," *Graphical Models and Image Processing* **60/2:2**, pp. 125–135, 1998.
12. C. Barber, D. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *The ACM Transactions on Mathematical Software (TOMS)* **22**, pp. 469 – 483, 1996.
13. T. Kong and A. Rosenfeld, "Digital topology: Introduction and survey," *Computer vision. Graphics and image proceeing* **48**, pp. 357–393, 1989.

14. A. Rosenfeld and A. Kak, *Digital Picture Processing*, Academic Press, New York, 1982.
15. G. Bertrand, "Topological numbers and geodesic neighborhoods in cubic grids," *Patten Recognition Letters* **15**, pp. 1003–1011, 1994.
16. D. Voorhies, "Triangle-cube intersection," *Graphics Gems III* , pp. 236 – 239, 1992.
17. A. Rosenfeld and J. Pfaltz, "Swquential operations in digital picture processing," *J. Assoc. Comput.* **13**, pp. 471–494, 1966.