Méthode basée sur la topologie digitale pour le filtrage topologique de surfaces reconstruites à partir d'un échantillon de points dans l'espace

par

Dong Lei Li

mémoire présenté au Département d'informatique en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, avril 2008

Your file Votre référence
ISBN: 978-0-494-42986-0
Our file Notre référence
ISBN: 978-0-494-42986-0

Le 22 avril 2008

le jury a accepté le mémoire de M. Donglei LI dans sa version finale.

Membres du jury

M. Djemel Ziou
Directeur
Département d'informatique

M. Madjid Allili
Codirecteur
- Bishop's University

M. Layachi Bentabet
Membre
- Bishop's University

M. Abdelhamid Benchakroun
Président-rapporteur
Département d'informatique

SUMMARY

In this work, 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 and Bern [2] 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 et al. [1] 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.

SOMMAIRE

Dans ce travail, nous utilisons des concepts de topologie digitale pour le filtrage topologique de surfaces reconstruites à partir d'un échantillon de points dans l'espace. Étant donné un échantillon S de points dans l'espace, nous utilisons l'algorithme de Amenta et Bern [2] basé sur les diagrammes de Voronoi pour reconstruire une surface linéaire par morceaux sous forme d'une grille triangulaire dont l'ensemble des sommets est S. Typiquement, les surfaces obtenues par le biais de cet algorithme exhibent des défauts de reconstruction qui se manifestent comme des petits trous et qu'on peut qualifier de bruit topologique. Nous proposons une méthode simple et efficace pour trouver et colmater ces trous indésirables qui fonctionne comme suit. Premièrement, nous plongeons la surface triangulée dans une représentation volumétrique. Par la suite, nous utilisons l'algorithme de fermeture de trous de Aktouf et al. [1] pour filtrer les trous selon leur grandeur et éliminer les plus petits qui sont moins plausibles et moins importants pour la topologie de la surface alors que les trous les plus larges représentent souvent des caractéristiques topologiques importantes de la surface. Nous présentons des résultats éxpérimentaux qui démontrent l'efficacité de cette méthode pour le traitement topologique automatique des surfaces dans l'espace.

ACKNOWLEDGEMENTS

I am indebted to my supervisor, Dr. Madjid Allili, for the guidance and support he has provided throughout the course of this work. His patience, despite many questions, is greatly appreciated. Without his help and encouragement, I would not have finished the thesis. Also, I would like to thank Djemel Ziou for accepting to be my co-supervisor.

I would like to thank several of my classmates and friends for their help and advice during the preparation of this thesis. Especially, David Corriveau who took lots of time to help me debug my experimental program and shared his experience in writing a thesis.

Also, I appreciate all the help my wife gave me. She took lots of time in checking errors in the thesis and supported me to finish the thesis by undertaking most of the home burden.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

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 work, 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 [14, 15, 16, 8]. 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.[11] 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 radius greater than α. More recently Amenta and Bern [2] 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. We will examine in details the algorithm of Amenta and Bern which does not require that surfaces are uniformly sampled. Indeed, the algorithm allows highly non-uniform sampling which is dense in detailed areas yet sparse in featureless ones.

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 [1] 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 [7] 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.

This thesis is organized as follows. In chapter 1 we review some important concepts in computational geometry that are relevant to surface reconstruction such as convex hulls, Delaunay triangulations, and Voronoi diagrams. Well known algorithms for the computation of convex hulls, Delaunay triangulations, and Voronoi diagrams are also presented. In chapter 2, we discuss in details the algorithm of Amenta & Bern for surface reconstruction. Chapter 3 is devoted to introduce some basic notions of digital topology which are used in the hole closing algorithm. These notions allow to define the concept of a hole and make the difference between holes, cavities, and concavities. At last, in chapter 4, we discuss in detail methods for surface disceretization, digital distance transformation in 3D, and the topological-based algorithm for filtering and closing holes in a reconstructed surface.

CHAPTER 1

Basic Concepts of Computational Geometry

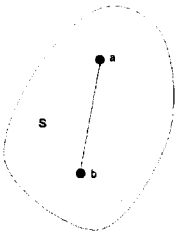
In this chapter, we introduce basic concepts of computational geometry which are used in surface reconstruction. We give definitions and examples of the convex hull, Voronoi diagram and Delaunay triangulation. We will also present some well known algorithms for their computation. For more details on these concepts, one can consult any classical book of computational geometry [9, 18, 22].

1.1 Convex Hulls

We first present the concept of convexity. Based on this concept, we will provide the definition of the convex hull. At last, several well known algorithms to compute the convex hull will be described.

Definition 1 *A subset S of the plane or the space is said to be* ***convex*** *if and only if for every pair of points in S, the line segment joining them lies also entirely within S.*

In Figure 1.1, the set S is convex, because any line segment AB with endpoints A and B in S is completely contained in the set S, whereas in Figure 1.2, AB is only partially contained in the set S. The definition of the convex hull is based on the notion of convexity.



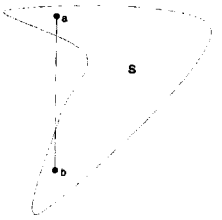


FIG. 1.1 – Example of a convex set. FIG. 1.2 – Example of a non convex set.

Definition 2 *The **Convex Hull** of a set S is the smallest convex set that contains S. More precisely, it is the intersection of all convex sets that contain S.*

Figure 1.3 shows a 2-dimensional convex hull of a set of points. The boundary of the convex hull consists of edges and vertices, i.e., *abcdef*. Figure 1.4 presents a 3-dimensional convex hull. In general, convex sets may have either straight or curved boundaries, be bounded or unbounded (e.g. an infinite cone is an unbounded convex set), and be topologically open or closed (a set is open when it contains no boundary points). In this work, we are only concerned with the convex hull of a finite set of points, which is necessarily a bounded, closed, convex polygon.

1.1.1 Algorithms for the Computation of Convex Hulls

Numerous algorithms for the computation of convex hulls have been devised. The most common algorithms are the Graham's scan, gift wrapping, divide and conquer, and the QuickHull.

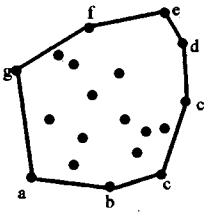


FIG. 1.3 – A 2D convex hull.

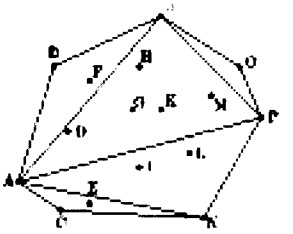


FIG. 1.4 – A 3D convex hull.

Graham's scan algorithm is based on an incremental construction. It proceeds by adding points one at a time, and the convex hull is updated with each new insertion. The algorithm stores the hull vertices in a stack S. Starting from the point P with minimum y coordinate, we scan every other point in increasing order of the angle they and P make with the x axis, and the vertex on the hull is pushed on the stack. If the consecutive triple of points consisting of the next point p, the top element p_t of S and the second element p_s of S make no left turn, then the point p_t will be replaced by p. Otherwise, we consider that p is a vertex on the hull and so we push it on the top of the stack.

In the *divide and conquer* algorithm, the point set is recursively partitioned into two sets, where one set consists of half the points with the lowest x coordinates and the other consists of half of the points with highest x coordinates. Until the partitioned point set is less or equal to 3, the two hulls will be merged into a common convex hull, by computing the upper and lower tangents for these hulls and discarding all the points lying between these two tangents.

The *Gift Wrap algorithm* starts from any point that is on the hull (the lowest point for example). Then we find the next edge on the hull in the counterclockwise order. Assuming the points p_k and p_{k-1} were the last two points added to the hull, we find the point q that maximizes the angle $\angle\, p_{k-1}p_kq$. We are done when we go back to the starting point.

The idea behind the *QuickHull algorithm* is to discard points that are not on the hull as

quickly as possible. Here is a brief presentation of the algorithm. It begins by finding four points on the hull, which we choose maximum and minimum respectively in the x and y coordinates. The points inside the quadrilateral Q defined by these four points will be discarded since they lie within the convex hull. For each edge ab of Q, we find the farthest point c that lies outside of the half-plane defined by the edge and draw a triangle abc. We discard the points inside abc and repeat the process for bc and ca. Then, we recurse until no more points lie outside of any triangle.

For a finite set of points in the space, the convex hull is a convex polyhedron. Its representation is not so simple as in the planar case. The pseudocode in the algorithm 4.3 gives the outline of the QuickHull algorithm in three dimensions.

In the first step, we choose four points $S(4) = \{p_1, p_2, p_3, p_4\}$ in S that do not lie in a common plane to construct a convex hull $CH(S(4))$ (i.e.,a tetrahedron). Choose any two points, say p_1, p_2. Then, we find a third point p_3 that is not on the same line in the direction p_1p_2. We scan the rest of the points to find a point p_4 that is not coplanar with p_1, p_2, p_3. If all the points share a common plane, the planar convex hull algorithm will be used. Otherwise, we will enter into the second step. To continue the algorithm, we check whether or not the remaining points are visible from the facets of the intermediate convex hull (the tetrahedron). In other words, they are outside of the current convex hull or in front of the visible facets. Otherwise, if the facet is invisible, this means the point is behind the facet. We can make use of the facet's normal vector to determine which side of the facet is the front and which is the back. To find the facets of $CH(S(i))$ that are visible to the point p, instead of checking every facet, which is time consuming, we can use a conflict graph. The algorithm can be run in $O(n \log n)$ time with this graph. A conflict graph is a bipartite graph. Let the undirected graph $G = V(P, F)$ contain a vertex $p_i \in P$ for each point that has not yet been added to the current convex hull $CH(S(i))$, and facets $f_k \in F$ for all facets of $CH(S(i))$. The linkages between points and

Input : A set $S = \{s_1, s_2, ..., s_n\}$ with n distinct points in 3D.
Output : The convex hull $CH(S)$

IntCV : Intermediate Convex Hull(tetrahedron)
Assign 4 points in S to p_1, p_2, p_3, p_4 so that they are not in a common plane;
Assign the rest of the points in S to $p_5, p_6, ..., p_n$;
Initialize IntCV with the points $\{p_1, p_2, p_3, p_4\}$ and their facets set $\{f_1, f_2, f_3, f_4\}$;
Initialize a conflict graph G (adding points $\{p_5, p_6, ..., p_n\}$ and 4 facets of initial 4 points into G and making a link with all visible pairs (p_i, f_j), where i = 5,..n, j = 1,2,3,4 (see Figure 1.5);
for(int i =5; $i \leq n$; i++)
    Insert p_i to IntCV;
    If ($F_{conflict}(P_i) \neq$ Null)
    p_i is outside of IntCV
      Then Delete all facets in $F_{conflict}(P_i)$ from IntCV;
          Find Horizon Edges list E;
          For all edges $e \in E$
            creating a triangular facet f_e by connecting two endpoints of e and p_i;
            If f_e is coplanar with its neighbor facet f' along e;
              Then Merge f_e and f';
              Change nothing since $P_{conflict}(f_e) = P_{conflict}(f')$
              Else
                Establish conflicts for f_e
                Create a node for f_e in G using the facets f'_e, f''_e that share the edge e in the old convex hull;
                For All the points P' that conflict f'_e or f''_e
                  If f_e is visible from P'
                    Add (P',f_e) to G;
            Delete the node corresponding to p_i and the nodes corresponding to the facets in $F_{conflict}(P_i$ from G, together with their incident arcs;
Return IntCV

Algorithm 1: Algorithm for Computing convex hulls

facets represent visibility and conflicts. If a point and a facet are connected, the facet is visible from the point. Conversely, the point is in conflict with this facet, because the point and the facet can not coexist at the same time in the convex hull. Whenever a new point is added to the current convex hull, the conflict graph is automatically updated.

Initially, a conflict graph G for $CH(S(4))$ can be set by just walking through the list of points in P to find the visible facets in $CH(S(4))$. The graph (Figure 1.5) shows an initial conflict graph.

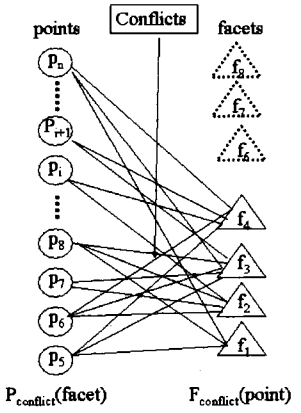


FIG. 1.5 – Conflict Graph G. The triangles with dashed lines are the facets that will be added later.

To maintain the conflict graph G, we need to replace the visible facets by the new convex hull facets. Suppose we add a new point p_i to the current convex hull $CH(S(i-1)), i > 4$. The nodes of all the visible facets of p_i will be deleted by checking the conflict facet list of p_i in G, and then replaced with new facets by connecting p_i with all horizon edges. If any of these new facets is coplanar with the facet in $CH(S(i-1))$, they will be merged with the old node only. Otherwise, a new facet with a new node will be added. At last, we check the remaining points $p_{i+1}, ..., p_n$ and newly added facets to add connections of the pairs which are visible. Similar to the QuikHull algorithm in the planar case, an optimized order is used to add a new point for a facet of the intermediate convex hull, which is the farthest point in the outside points set P of the current convex hull. The running time of the algorithm is $O(n^2)$. But normally it will not be worse than $O(n \log n)$ in practice.

1.2 Voronoi Diagrams and Delaunay Triangulations

1.2.1 Duality

Generally speaking, a duality transform translates concepts, theorems or mathematical structures into other concepts, theorems or structures, in a one-to-one fashion. Duality is an involution operation; that is, if the dual of A is B, then the dual of B is A. As involutions sometimes have fixed points, the dual of A is sometimes A itself. The fundamental structures of computational geometry, i.e., the Voronoi diagram and the Delaunay triangulation, which will be introduced later, have this duality relationship. In a plane, a Voronoi diagram of a point set can be obtained from the Delaunay triangulation of this point set by drawing perpendicular bisectors on the edges joining pairs of points.

1.2.2 Voronoi Diagrams

The Voronoi diagram of a set of points (usually called *sites*) is a partition of the plane or the space into regions. Each region corresponds to one and unique site, and all the points in one region are closer to the corresponding site than to any other site. It is a fundamental structure of computational geometry. A typical example is the "point location" problem. Suppose we want to set up regions for the post offices in a city. Each region has one post office. It would be nice to make every house in a region have the closest post office to be the one in the region rather than the others. All we need to do is figure out how to partition the city into regions as a Voronoi diagram. To establish the Voronoi diagram for the post offices, we can proceed as follows. Let two sites p_1 and p_2 represent two post offices in a city (see Figure 1.6). Here the distance between two sites is the Euclidean distance. We draw the perpendicular bisector of the line $\overline{p_1p_2}$ to split the plane into two half-planes $P(p_1)$ and $P(p_2)$. It is not hard to notice that the distance of any point q in

$P(p_2)$ to p_2 is less than the distance to p_1. Similar notice can be made in the half-plane $P(p_1)$.

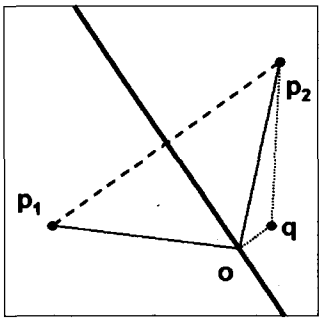


FIG. 1.6 – A Simple Voronoi Diagram with 2 Sites

The Voronoi region (cell) associated to a site p is the intersection of all the half planes (hyperplanes) containing p and formed by the perpendicular bisectors of the lines associated with all pairs of sites. In Figure 1.7, we see clearly that every Voronoi diagram edge is the perpendicular bisector of a pair of two sites. A Voronoi diagram region (cell) can be a polygon or an unbounded region. In this work, there will be no unbounded regions since a city has a boundary. In general, a Voronoi diagram cell can be a polyhedron or an unbounded region. A formal definition is as follows.

Definition 3 *Consider n distinct sample points (or sites) set* $S = \{p_1, p_2, \ldots, p_n\}$ *in the Euclidean space* $\mathbb{R}^d$. *The Voronoi diagram of S splits the Euclidean space* $\mathbb{R}^d$ *into regions under the following conditions :*

- *Each region has a unique site ;*
- *The nearest site (or site with shortest distance) to any point in a Voronoi region is the site of that region.*

Figure 1.7 shows a 2D Voronoi diagram where p is a site, q is a point in the Voronoi region associated to the site p, v is a vertex of the Voronoi Diagram and b is an unbounded Voronoi cell. Similarly in 3 dimensions, instead of polygons, a Voronoi cell is a convex polyhedron for bounded regions.

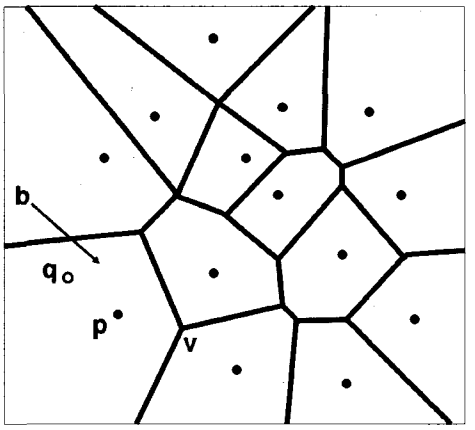


FIG. 1.7 – A 2D Voronoi Diagram

Voronoi diagrams satisfy the following important properties.

Theorem 4 *A Voronoi region is unbounded if and only if the site of that region is in the convex hull of the set S.*

Theorem 5 *The largest empty circle (containing no sites) of an edge of a Voronoi Diagram (circumcircle) contains only two sites on its boundary.*

Each point on the edge of a Voronoi diagram is equidistant from its two nearest sites p_i and p_j. You will never have a point (site) that is in the interior of the circle (Figure 1.8) according to the above theorem.

Similarly, for the Voronoi vertex, we have the following theorem.

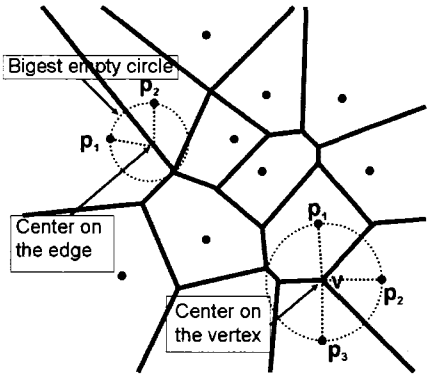


FIG. 1.8 – Circumecircle in a Voronoi Diagram

Theorem 6 *The largest empty circle of a Voronoi vertex has at least 3 sites on its boundary.*

Here the largest empty circle refers to the largest circle without any site inside. In Figure 1.8, the vertex at which three Voronoi cells $VC(p_1), VC(p_2)$ and $VC(p_3)$ intersect is equidistant from all the three sites p_1, p_2 and p_3. So it is the center of the circle passing through these three sites. No other site is inside this circle.

Theorem 7 *Let n denote the number of sites, the Voronoi diagram is a partition of the plane (if we consider all the unbounded cells as faces) with exactly n faces. The number of Voronoi vertices is at most $2n - 5$ and the number of edges is at most $3n - 6$.*

1.2.3 Delaunay Triangulations

Briefly, the Delaunay triangulation (DT) is a popular technique for generating well-shaped triangulations for given sets of points. It can be defined as follows.

Definition 8 *Consider a set of n distinct planar points. Its Delaunay triangulation is a*

special and unique triangulation where the circumcircle of every triangle contain no other points from the set.

An example of an Non-Delaunay triangulation is given in Figure 1.9. The triangles $\triangle dab$ and $\triangle bac$ constitute the triangulation of the sites a, b, c, d. They do not meet the condition to be a Delaunay triangulation, since the site d is inside of the circumcircle (circle 1) of the triangle $\triangle abc$.

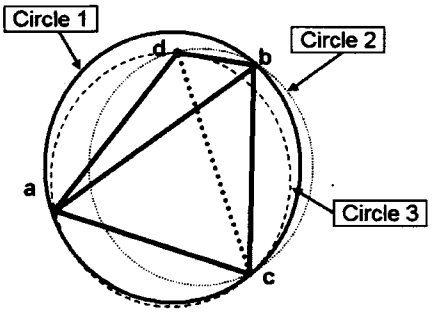


FIG. 1.9 – In 2D, the circumcircle of the triangle *abc* encircles the site *d*. So *abc* is not a triangle of a Delaunay triangulation.

There are multiple characteristics that make Delaunay triangulation the best triangulation. For instance, the Delaunay triangulation maximizes the minimum angle among all the triangulations of a point set.

Theorem 9 *Among all triangulations of a given planar point set, the Delaunay triangulation is angle-optimal.*

Let $A(T) = (\partial_1, \partial_2, ..., \partial_{3k})$ be the vector of angles in the triangulation T in increasing order with ∂_1 being the smallest angle. $A(T)$ is larger than $A(T')$ if and only if there exists an i such that $\partial_j = \partial'_j$ for all $j < i$ and $\partial_i > \partial'_i$. A triangulation T will be better than

T' if $A(T)) > A(T')$. The best triangulation is the triangulation that is **angle optimal**, i.e. the one that has the largest angle vector or maximum minimum angle.

In order to increase the angle-vector, we can flip the edges of a triangulation. In Figure 1.9, if we replace ab with the edge dc, the two triangles which consist of the quadrilateral $abcd$ incident with the edge, we obtain a larger angle-vector. The circumcircles of the pair of triangles, i.e., circles 2 and 3, become empty circles.

An edge is called legal when the circumcircles of its incident triangles do not contain the opposite node of the other triangle. The edge ab in Figure 1.10 is an illegal edge and the edge dc in Figure 1.11 is a legal edge. Delaunay triangulations and Delaunay edges are intimately related : a Delaunay triangulation has only legal edges. Illegal edges of a triangulation can always be flipped : the two triangles incident with the edge always form a convex quadrilateral, and the diagonal may be switched. The flipped diagonal is always Delaunay, and the minimum angle of the pair of triangles is always increased, thus improving mesh quality. Flipping illegal edges only affects a single part of the mesh, so it can be seen as local improvement strategy. The Delaunay triangulation can be constructed by starting with an arbitrary triangulation and flipping illegal edges until none are left. The final result maximizes the minimum element angle. This is called the maxmin-angle property.

These properties are presented in the following theorems.

Theorem 10 *For a given planar point set S , a triangulation T of S is legal if and only if T is a Delaunay triangulation of S.*

Theorem 11 *For a given planar point set S, the circumcircle of three vertices of the same face of the Delaunay triangulation of S contains no point of S in its interior.*

Theorem 12 *For a given planar point set S, any circle passing through the two points*

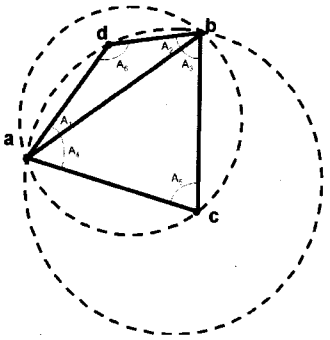


FIG. 1.10 – An illegal edge has a triangle whose circumcircle contains the opposite vertex of a neighboring triangle

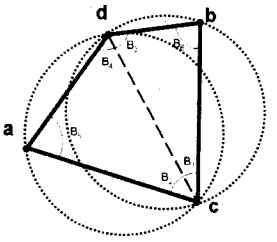


FIG. 1.11 – By reconnecting and flipping the edge dc, the new circumcircles only contain the vertices it circumscribes

of an edge has no point of S inside.

Theorem 13 *The Delaunay triangulation is the geometric dual of the Voronoi diagram.*

In Figure 1.2.3, the dashed line represents the Delaunay triangulation and the bold line represents edges of the Voronoi diagram. Every edge of the Voronoi diagram is a perpendicular bisector of an edge of the Delaunay triangulation.

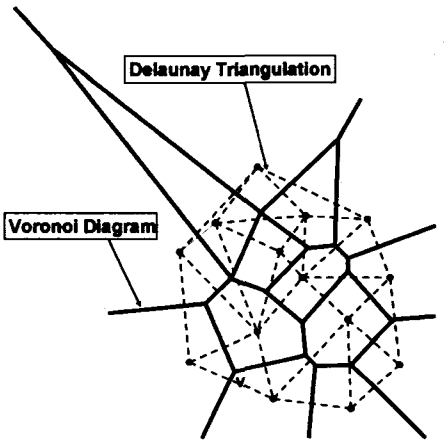


FIG. 1.12 – In 2D, the dashed line graph is a Delaunay triangulation. The bold line graph is a Voronoi diagram

1.2.4 Computing the Voronoi Diagram and the Delaunay triangulation

Since the Delaunay triangulation of a point set S is the dual of the Voronoi diagram, we can start to compute either one of these two structures, the other can be easily derived in $O(n)$ time. There exists several well-known algorithms addressing the computation of a planar Voronoi diagram and Delaunay triangulation such as divide and conquer, randomized incremental, Fortune's plane sweep, and reduction to the convex hull.

Divide and Conquer algorithm : Here is an outline of the algorithm. It begins by sorting the points by their x-coordinate. The remainder of the algorithm is shown in the section below.

1. Equally split the sorted point set into two subsets S_1 and S_2 ;
2. Compute the Voronoi diagram of two subsets recursively (Note that since each diagram alone covers the entire plane, these diagrams overlap).
3. Merge the two diagrams into a single diagram, by computing the contour and discarding the portions in the different sides of the subsets (e.g. For S_1, delete the portion that overlap S_2 and vice versa).

The algorithm works for both Voronoi diagram and Delaunay triangulation. It is hard to be used widely because the planar implementation is complicated and difficult to be generalized to higher dimensions.

Randomized Incremental algorithm : It works for both Voronoi diagram and Delaunay Triangulation. For Delaunay triangulation, this algorithm adds sites one by one randomly, updating the Delaunay triangulation after each addition. The update consists of discovering all Delaunay faces whose circumcircle contain the new site. These faces are deleted and the empty region is partitioned into new faces, each of which has the new site as a vertex. An efficient algorithm requires a good data structure for finding the

faces to be deleted. The running time is determined by the total number of face updates, which depends upon site insertion order. This algorithm can be generalized to higher dimensions as with the randomized algorithm for convex hulls.

Fortune's Plane Sweep algorithm : It works for Voronoi diagrams. It computes a deformed Voronoi diagram by plane sweep in $O(nlogn)$ time from which the true diagram can be extracted easily.

Reduction to convex hulls algorithm : It works for Delaunay triangulation. Computing a Delaunay triangulation of n points in dimension d can be reduced to computing a convex hull of n points in dimension $d + 1$. It is then possible to use our favorite convex hull algorithm to do the desired computation.

CHAPTER 2

Algorithms for 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 most general case, the simplest structure is a simplicial polyhedron. In 3D, it is defined as a triangulation satisfying the following conditions [13] :

1. Any two triangles are disjoint, or share a vertex, or have a common edge, i.e, share two vertices;
2. The triangles are connected;
3. For any vertex P, the edges opposite to P in the triangles form a simple polygon with P as a vertex.

Two ways can be used to reduce the number of all possible combinations of the triangulations. The first way is a *surface-based approach.* It is a kind of local procedure. Any point P and neighbors on the surface S can be projected onto the tangent plane H of S that passes through P with orthogonal angle. In certain regions, the projection of the triangulation of S and the triangulation of its projection on H are diffeomorphic. The following proposition reported in [6] guarantees that the projection is a diffeomorphism in a certain region around P.

Proposition 14 *Let S be a smooth surface in three dimensional space whose principal radii of curvature exceed R at every point. Let p denote the orthogonal projection onto a tangent plane H of S at P. Then there exists an open set U of S such that p is a diffeomorphism from U onto any disk lying in H whose center is P and whose radius is smaller than R.*

This means that $p(U)$ cannot fold over itself and therefore any triangulation with straight edges in H will correspond to a triangulation with straight edges on S. If one wants to

control the correspondence in such a way that areas and lengths are not greatly changed, one would need to avoid the use of too thin triangles. This can be guaranteed by the use of Delaunay triangulations. Furthermore, the triangles in the Delaunay triangulation entirely meet the first and second conditions of the definition of the simplest structure. The computation of the Delaunay triangulation is very efficient. Many available algorithms can compute the Delaunay triangulation in $O(n \log n)$. Many other algorithms have been devised following Boissonnat's idea of using the Delaunay triangulation [10, 11], however all of them require the use of uniform or organized samples of points.

In 1998, Amenta and Bern [2] 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. Given a "good sample" from a smooth surface, the output of the algorithm is guaranteed to be topologically correct and convergent to the original surface as the sampling density increases [4]. They also provide a theoretical proof of the correctness of the algorithm. The algorithm is based on the concepts of Voronoi diagram and Delaunay triangulation in the three dimensional space. More precisely, it uses only a subset of the Voronoi vertices to extract a subset of Delaunay triangles from the Delaunay triangulation of the sample points. The algorithm introduced the concept of *poles* and the Delaunay triangles with circumspheres empty of poles give a piecewise-linear surface pointwise convergent to the real surface [2] from which the sample points are drawn. The poles also enable further filtering to ensure the convergence of the surface and the approximation triangles. We call these Delaunay triangles the CRUST of the sample points.

In this chapter, 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.

2.1 The CRUST 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.

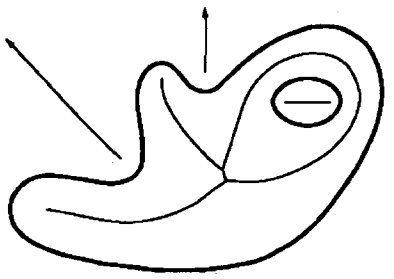


FIG. 2.1 – An example of medial axis in 2D.

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 S.

The algorithm of Amenta et al. in $\mathbb{R}^3$ is an extension of their algorithm of reconstructing curves in the plane [3]. 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.

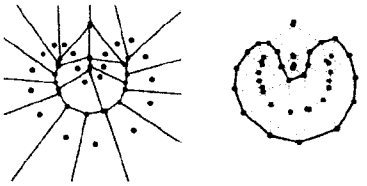


FIG. 2.2 – Reconstructed curve using Delaunay triangulation and Voronoi diagram in R^2.

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 passes 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. (*Voronoi filtering*) :
   Extract only the triangles of the Delaunay
   triangulation with all three vertices in S ;
6. (*Normal filtering*) :
   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 2: Surface reconstruction Algorithm

2.1.1 Calculation of the Poles

The notion of pole was introduced earlier. The reason that the algorithm uses only poles (a subset of Voronoi vertices) and the original sample points set S to compute the Delaunay triangulation can be explained as follows. The algorithm is derived from Amenta's 2D algorithm that uses all Voronoi vertices of the cell to reconstruct a curve correctly. But a problem occurs in the 3D case if we use $S \cup V$. In Figure 2.3, $(p_1, p_2, p_3, p_4, p_5, p_6, p_7)$ are sites while $(p_8, p_9, p_{10}, p_{11}, p_{12})$ are Voronoi vertices of the Voronoi cell. After performing the Voronoi filtering in the algorithm, $(p_1, p_2, p_3, p_4, p_5, p_6, p_7)$ will be kept in the approximation of the surface. However, when a Voronoi vertex is very close to the surface, an unexpected hole will be created in the reconstructed surface. For instance, in Figure 2.4), everything looks the same as in Figure 2.3, except there is a Voronoi vertex p_{13} which is very close to the surface F. The extra dash line segments will be added in the Delaunay triangulation compared to the previous one. When we do the Voronoi filtering, the triangles $\triangle p_1 p_2 p_{13}$, $\triangle p_2 p_5 p_{13}$, $\triangle p_5 p_4 p_{13}$ and $\triangle p_4 p_1 p_{13}$ will be deleted. Because they are not triangles with three vertices all belonging to the sample points S. This results in a hole symbolized by the polygon $(p_1, p_2, p_3, p_5, p_4)$ in the the reconstructed surface.

The method to compute the poles of a Voronoi cell is outlined in the following algorithm 2.1.1. Let (p_i+ and p_i-) be the poles of a site $s_i, i \in n$ in $\mathbb{R}^3$. We denote by $Averg(\overrightarrow{n_i})$ the average normal of all the triangles' normals incident to s_i on the convex hull $CH(S)$. $V(s_i)$ are the vertices of the Voronoi cell associated to the site s_i.

2.1.2 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

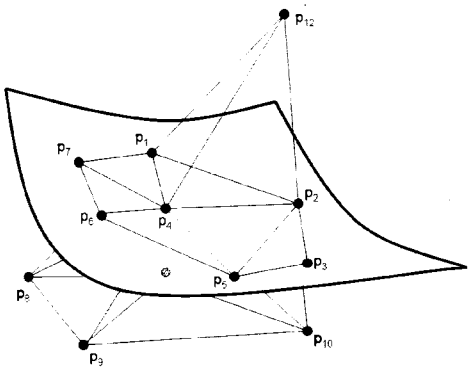


FIG. 2.3 – Without any Voronoi vertices in a cell that is very near the surface.

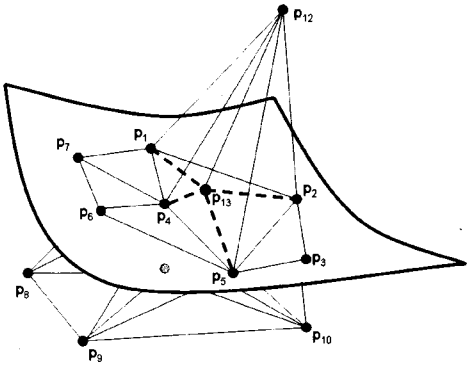


FIG. 2.4 – One of the Voronoi vertices P_{13} is very close to the surface. Using all the Voronoi vertices and the original sample points will create a hole in the reconstructed surface.

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 [4, 2] 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 $p \in F$.

2.1.3 Voronoi and Normal Filtering

The first four 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

If (s_i is on $CH(S)$) {
    Compute $Averg(\overrightarrow{n_i})$;
    p_i+ is at "infinite distance" outside of $CH(S)$ with the direction of $\overrightarrow{s_i p_i+}$
    the same as $Averg(\overrightarrow{n_i})$
}
If (s is in the interior of $CH(S)$) {
    Choose the furthest vertex among $V(s_i)$ of the Voronoi cell as p_i+;
}
Choose the furthest vertex as p_i- in $V(s_i)$ with $\angle p_i s_i p \leq \pi/2$;

Algorithm 3: Algorithm for finding the poles.

formed of tetrahedrons, not triangles. However, the paper [2] 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 [4] 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 increases [4] 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 [4]. After the normal filtering, the extracted Delaunay triangles are roughly parallel to F.

2.1.4 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 [4]. 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.

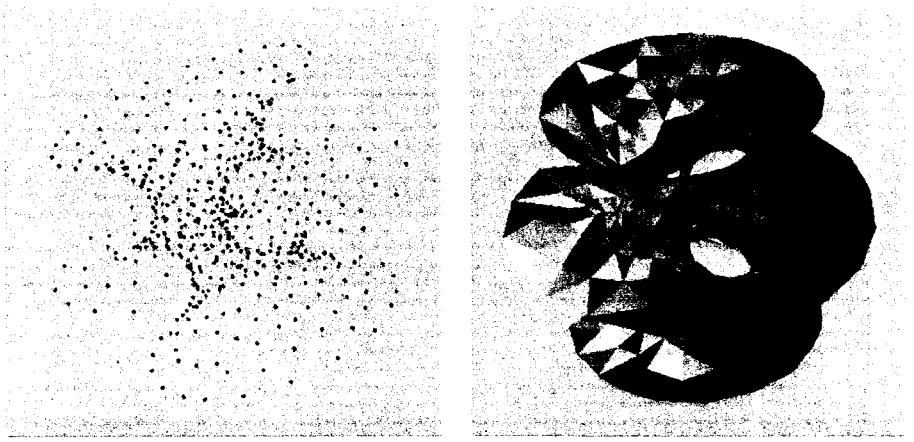
2.1.5 Implementation

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 [5] 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.

2.1.6 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 2.5 (a), we used the given interface information to reconstruct the surface. In Figure 2.5 (b), we used only the sample points information and Delaunay triangulation



(a) (b)

FIG. 2.5 – Reconstruction of hyper sheet. (a) Hyper sheet sample points with relatively good density and nearly uniform. (b) Reconstructed hyper sheet surface using the algorithm.

(a) (b)

FIG. 2.6 – Normal filtering of a hyper sheet. (a) Hyper sheet sample points with 180 degree angle of normal filtering. (b) Hyper sheet sample points with 172 degree angle of normal filtering.

(a) (b)

FIG. 2.7 – Reconstruction of a golf club. (a) Golf club sample points. (b) Some holes exist on the reconstructed surface of the golf club.

to do the reconstruction. Except for the hypersheet, all reconstructed surfaces using the CRUST algorithm are full of holes. For example, in Figure 2.7 (b), the result of the reconstruction of a golf 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 2.6 (a) and 2.6 (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.

2.1.7 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.

CHAPTER 3

Basic Concepts in Digital Topology

The surfaces reconstructed by means of many algorithms such as the one of Amenta and Bern [2] often exhibit unwanted small holes that can be considered as topological noise. This noise complicates subsequent operations such as parametrization or smoothing of the surface. Thus, identifying and closing these holes automatically is a significant endeavor to improve the quality of the surface and the quality of its display. Reconstructed surfaces lead to meshes containing vast amounts of data, therefore the manual removal of unwanted artifacts is a tedious and time-consuming task. Taking advantage of the features of digital topology can make the detection processing easier. The reconstructed surfaces are first embedded into a cubical volumetric representation, then the holes in the surface are tracked by using an algorithm based on concepts from digital topology that will be discussed in the next chapter.

This chapter is devoted to an overview of the important concepts in digital topology. We start with reviewing some basic concepts of topology, and then extend into digital topology and its basic properties.

3.1 Topology

Topology is the branch of Mathematics that studies the properties of objects which are preserved through deformations, twisting and stretching, without regard to the size and absolute position [19, 20]. According to the definition, tearing is prohibited since it would generate discontinuities. New holes cannot be created in an object. Practically, topology can be simply defined as the study of holes and gaps in geometric objects. For topology, two objects with different number of holes cannot be homeomorphic, because neither of them can be continuously deformed into the other. Figure 3.1 presents objects that are topologically equivalent.

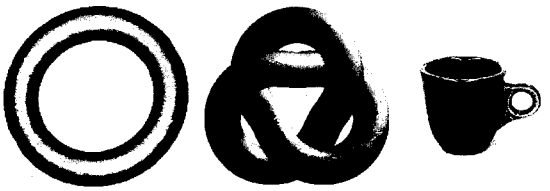


FIG. 3.1 – A torus, knotted torus and cup are homeomorphic.

A ball can be continuously deformed into a teddy bear, but it cannot be continuously deformed into a donut since a hole in the ball would have to be created. Similarly, a donut is equivalent to a cup since we can deform the hole of the donut in the hole in the handle of the cup. Topology is only interested in the number of holes, but not in their spatial location. Hence, it must be complemented with geometric tools to find the positions of the holes. Among important concepts in topology is the one of a *non-separating loop*, i.e., whenever there exists a continuous loop that cannot be homotopically deformed into a point within the manifold itself [21] (see Figure 3.2). In that case, the loop identifies a hole in a the volume or the surface. A *non-separating loop* is the fundamental concept used for the development of the hole closing algorithm.

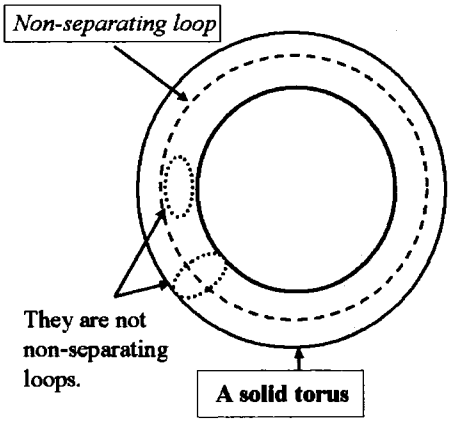


FIG. 3.2 – A non-separating loop in a solid torus.

3.2 Basic Concepts in Digital Topology

Digital topology can be defined as the study of the topological properties of image arrays [17]. The concepts and theories of digital topology provide a sound mathematical basis for image processing operations. Unlike classical topology which is based on set theory operations, digital topology is defined through the concepts of adjacency and connectivity. In this section, we discuss mainly the notions of path, hole, cavity and we introduce several very important notions such as the ones of a topological number, simple point, and isthmus, which are crucial in the implementation of the hole closing algorithm discussed in the next chapter.

3.2.1 Adjacency Relations and Connectivity

We confine our attention to two- and three-dimensional binary image arrays. By convention, the two-dimensional elements that form our images are called *pixels* and the three-dimensional elements are called *voxels*. The digital topology concepts apply on a grid

composed of a set of pixels or voxels. They are related to each other by connectivity relationships. Pixels or voxels can be connected in various ways, which can be defined as different adjacencies. In two-dimensions, there are two types of connectivities : 4-, 8-connected. For instance, two pixels are 4-connected if they are horizontally or vertically adjacent. Two pixels are 8-connected if they are 4-connected or diagonally adjacent (see Figure 3.3).

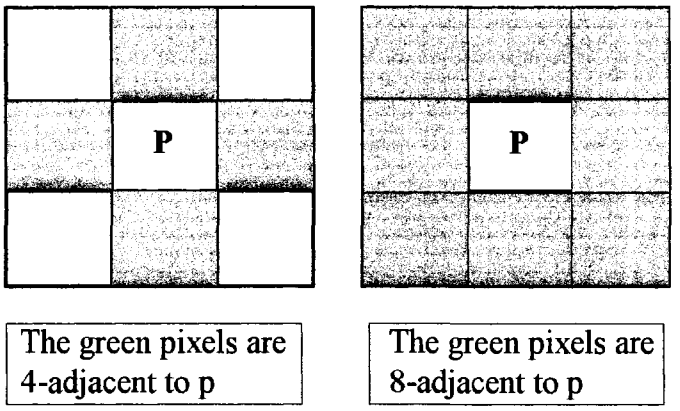


FIG. 3.3 – 4 and 8-adjacent relationships in 2D

Every pixel is associated with integer coordinates. Suppose that our image support is $W=\mathbb{Z}^2$. Each pixel p ($p \in W$) has coordinates (x_1, x_2) with $x_i \in \mathbb{Z}, i = 1, 2$. Any pixel p' ($p' \in W$) with coordinates (x'_1, x'_2), where $x'_i \in \mathbb{Z}, i = 1, 2$ is defined as 4-connected to p if it belongs to $N_4(p)$ and 8-connected to p if it belongs to $N_8(p)$, where

$N_4(p) = \{p' \in W, Max(|x_1 - x'_1| + |x_2 - x'_2|) \leq 1\}$; and
$N_8(p) = \{p' \in W, Max(|x_1 - x'_1|, |x_2 - x'_2|) \leq 1\}$.

In three-dimensions, there are at least three types of connectivities corresponding to the

6-, 18- and 26-adjacency relations (See Figure 3.4). If two voxels are connected by a face, they are 6-adjacent. If two voxels are connected by a face or an edge, they are 18-adjacent. If two voxels are connected by a vertex, an edge or a face, then they are 26-adjacent.

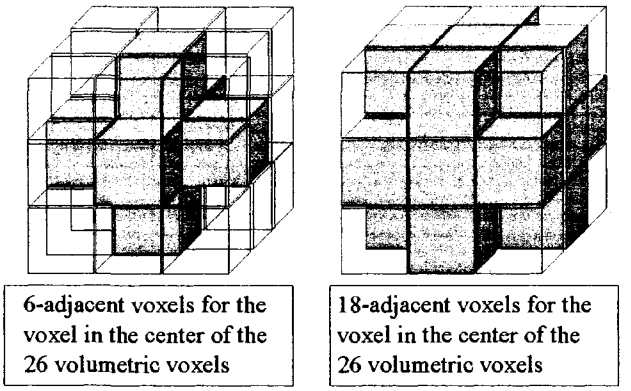


FIG. 3.4 – 6 and 18-adjacent relationship in 3D

Similarly, every voxel is associated with integer coordinates. Suppose we have $W{=}\mathbb{Z}^3$. Each voxel v ($v \in W$) has coordinates (x_1, x_2, x_3) and a neighboring voxel with coordinates (x'_1, x'_2, x'_3), where $x'_i \in \mathbb{Z}, i = 1, 2, 3$ is 6-,18- or 26-adjacent to v if it belongs to to the sets $N_6(v)$, $N_{18}(v)$, or $N_{26}(v)$ respectively, where $N_6(v) = \{v, v' \in W, Max(|x_1 - x'_1| + |x_2 - x'_2| + |x_3 - x'_3|) \leq 1\}$;

$N_{18}(v) = \{v, v' \in W, Max(|x_1 - x'_1| + |x_2 - x'_2| + |x_3 - x'_3|) \leq 2$
$\bigcap\ Max(|x_1 - x'_1|, |x_2 - x'_2|, |x_3 - x'_3|) \leq 1\}$; and

$N_{26}(v) = \{v, v' \in W, Max(|x_1 - x'_1|, |x_2 - x'_2|, |x_3 - x'_3|) \leq 1\}$.

We denote by $N_i^*(p), i = 4, 8$ for 2D or $i = 6, 18, 26$ for 3D the set of i-adjacent pixels or voxels to p excluding p itself. In the sequel, we denote by F the foreground of our image, and by B its background. Then, $W = F \cup B$. Sometimes B is denoted as $\overline{F}$. In order to

avoid topological ambiguities, different connectivities have to be chosen in the foreground and the background of the image. For instance, if the 6-adjacency relation is used for the foreground, then 6-adjacency should not be used in the background . Instead, we can use 18- or 26-adjacencies. Here, we use an example(Figure 3.5) in 2D to explain the concept. Suppose both the foreground (black pixels) and the background (white pixels) sets are explored using the 4-adjacency relation.

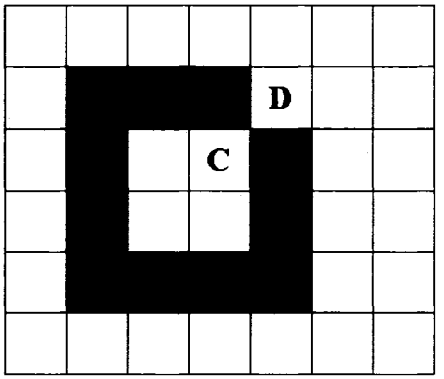


FIG. 3.5 – Different adjacencies should be used for the foreground and the background.

The foreground pixels do not form a closed curve since A and B are not 4-adjacent, yet the background is divided into two regions (connected components) containing the disconnected points C and D respectively. Using the same connectivity for the foreground and the background leads to several paradoxes such as violating the Jordan curve theorem and problems of defining boundaries of objects. One of the solutions is to use dual connectivities, for instance, 4-adjacency for the foreground and 8-adjacency to study connectivity in the background. Then, the union of the subsets containing C and D become an 8-connected region, which is conform to our intuition and to the axioms of

classical topology. We call the 4 and 8-adjacencies a compatible pair. Similarly, the following pairs of connectivity in the foreground and background are compatible in 2D : $\{4,8\}$ and $\{8,4\}$. For the 3D case, the pairs $\{6,26\}$, $\{6,18\}$, $\{18,6\}$ and $\{26,6\}$ are also compatible pairs.

3.2.2 Path, Hole, and Cavity

Suppose we have an object X in an image grid. For any two points (pixels or voxels) P_1 and P_2 in X, a n-**path** from P_1 to P_2 [1] is a sequence of points starting from P_1 to P_2 in which each point (except the last one) is n-adjacent to the next point. Here, n can be 4 or 8 in 2D and 6,18 or 26 in 3D. The simplest path is the one that includes only P_1 *and* P_2 whenever they are n-adjacent. P_1 and P_2 are called n-connected if they are connected by an n-path. Normally, there are multiple n-paths between P_1 and P_2 in X. Sometimes, one path can be obtained from an other path by simply changing one point or multiple points in the sequence. This is called a deformation. For an n-path, the deformation is an n-deformation. In special case where P_1 and P_2 are the same point, i.e., only one point actually exists and the path includes more than one point, we say, the path forms a closed path. With the introduction of n-paths and n-deformations, it is easy to define the notions of hole and cavity.

In 2D, there is no distinction between a hole and a cavity. But in 3D, things are more complex. Sometimes a hole is different from a cavity. The existence of a hole in X is characterized by the following definition. In X, if there is a closed path that cannot be deformed to one point through one or a sequence of n-deformations, a hole should exist[1]. In Figure 3.6, the torus has two holes and one cavity. Clearly, the circles C1 and C2 cannot be deformed to a point within the hollow torus. These circles are associated to two holes in the hollow torus.

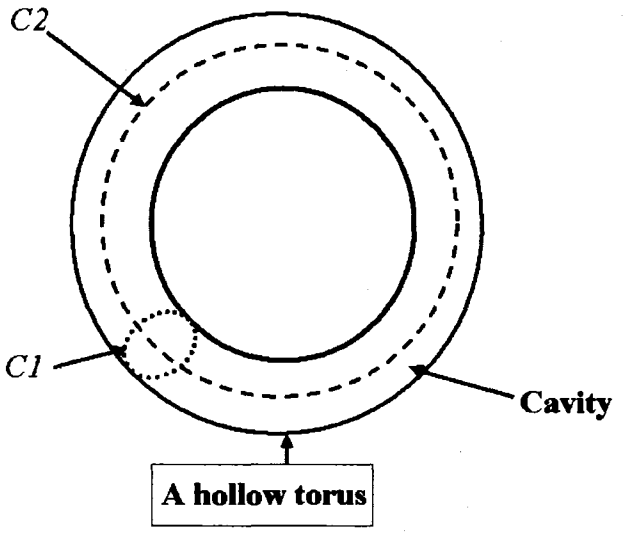


FIG. 3.6 – In the hollow torus, the circles C1 and C2 can not be deformed to a point by an n-deformation. They are associated to holes. The hollow is a cavity, since points in the hollow (background) have no connection to the outside background.

More formally, suppose h is a subclass of the background $\overline{X}$. If h has no path to any other subset of $\overline{X}$, we say that h is a *cavity*. In Figure 3.6, the hollow in the torus is a cavity. But if the torus is solid, then there will be no cavity in it. For the hollow sphere, there is a cavity, but no hole in it. A tube with two open ends has a hole but no cavity.

3.2.3 Topological Number (TN)

In order to give a definition of the topological number, several concepts need to be introduced first [1]. Let $W = \mathbb{Z}^3$, $X \subset W$, and $p \in W$. The *geodesic n-neighborhood of p inside X of order m* is the set $N_n^m(p, X)$ which is defined recursively by the following formula :

$$N_n^1(p, X) = N_n^*(p, X) \bigcap X \text{ and } N_n^m(p, X) = \bigcup \{N_n(q) \bigcap N_{26}^*(p) \bigcap X, q \in N_n^{m-1}(p, X)\}.$$

Here, $N_{26}^{*}(p) \bigcap X$ represents all the voxels that are 26-adjacent to p excluding p. The meaning of order m can be observed in the following figures. Suppose that p and its neighboring voxels all belong to X and $n = 6$.

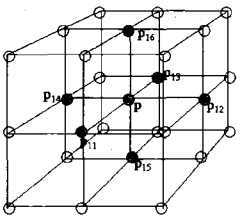


FIG. 3.7 – The geodesic 6-neighborhood of p inside X of order 1 or $N_6^1(p) \bigcap X$ is the solid green circles.

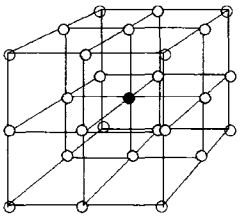


FIG. 3.8 – The geodesic 6-neighborhood of p inside X of order 2 or $N_6^2(p) \bigcap X$ is the solid blue circles.

Figure 3.7 depicts $N_6^1(p, X) = N_6^*(p, X) \bigcap X$ for order 1. The geodesic 6-neighborhood of p inside X of order 1 are the elements of the set $\{p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\}$, which is equivalent to the voxels 6-adjacent to p. Figure 3.8 shows the result of $N_6^2(p, X) = \bigcup\{N_6(q) \bigcap N_{26}^*(p) \bigcap X, q \in N_n^1(p, X)\}$, that is, the geodesic 6-neighborhood of p inside X of order 2. Based on the above order 1's result, $N_6^2(p, X)$ can be expressed more clearly as follows.

$$N_6^2(p, X) = \{N_6(p_{11}) \bigcap N_{26}^*(p) \bigcap X\} \bigcup \{N_6(p_{12}) \bigcap N_{26}^*(p) \bigcap X\}$$
$$\bigcup \{N_6(p_{13}) \bigcap N_{26}^*(p) \bigcap X\} \bigcup \{N_6(p_{14}) \bigcap N_{26}^*(p) \bigcap X\}$$
$$\bigcup \{N_6(p_{15}) \bigcap N_{26}^*(p) \bigcap X\} \bigcup \{N_6(p_{16}) \bigcap N_{26}^*(p) \bigcap X\}.$$

In order 2, the extra process is to find the 6-adjacent neighborhood for every elements of order 1 in the $N_6^*(p, X) \bigcap X$. The geodesic 6-neighborhood of p inside X of order 2 is equivalent to the 18-adjacent neighbors of p. Similarly, the geodesic 6-neighborhood of p inside X of order 3 is $N_6^3(p, X) = \bigcup\{N_6(q) \bigcap N_{26}^*(p) \bigcap X,\ q \in N_n^2(p, X)\}$. The order 3 represents finding the 6-adjacent neighbors of the set $N_6^2(p, X)$ except p, which is

equivalent to the 26-adjacent neighbors of p.

Based on the concept above, the definition of the *geodesic neighborhoods* $G_{p,X}$ [1] can be presented as follows. Let X be a subset of E and p be a point of E. The *geodesic neighborhoods* $G_n(p, X)$ can be defined as.

$G_6(p, X) = N_6^2(p, X)$;

$G_{6+}(p, X) = N_6^3(p, X)$;

$G_{18}(p, X) = N_{18}^2(p, X)$;

$G_{26}(p, X) = N_{26}^1(p, X)$;

An *n-connected component of* X is a non-empty set such that any two points are n-connected. Normally, there exist multiply n-connected components in X. We can denote the set of all n-connected components as $C_n[X]$. $C_n^p[X]$ is used to denote the elements in $C_n[X]$ that have an n-path to p . Clearly, the elements in $C_n[X]$ and $C_n^p[X]$ are sets. These sets are composed of point(s). With the concepts above, the **topological number** can be defined as follows.

Definition 15 $TN_n(p, X) = \#C_n[G_n(p, X)]$, *where sign* $\#G_n$ *represents the number of n-connected components of* $G_n(p, X)$.

For example, we consider the grid in Figure 3.9. Let X represent the green voxels. According to the definition of $G_6(p, X)$, it is not difficult to find m_{21} and m_{10} to be in the set $N_6^1(p, X)$. On the other hand, t_{10}, b_{10},t_{21} and b_{21} can be found in the set $N_6^2(p, X)$. In the set $N_6^2(p, X)$, one 6-connected block consists of b_{21},m_{21} and t_{21}. Another consists of b_{10},m_{10} and t_{10}. There is no 6-path between these two components. So, $TN_6(p, X) = \#C_6[G_6(p, X)] = 2$. Note that even though t_{20} and b_{20} link those two components in X, they are still two separate 6-connected components. Because we count only in the set $N_6^2(p, X)$, t_{20} and b_{20} should not be taken into consideration.

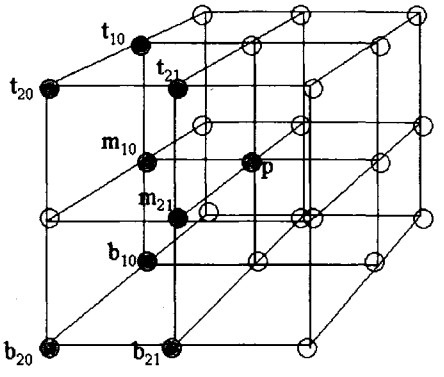


FIG. 3.9 – Let X consist of the green voxels. The topological number of p in X is 2.

3.2.4 Simple Point and Isthmus

The concepts of simple point and isthmus are topological characteristics of a point based upon the topological number. They are extensively used in topological filtering. The definition of a simple point is as follows .

Definition 16 *Let X be an object such that $X \subset W$ and $p \in X$. If the removal of p does not change the topology of the object, then p is called an* ***n-simple point (of X)****.*

An isthmus point can be defined as follows.

Definition 17 *Let $X \subset W$ and $p \in X$. If p is not an interior point and the removal of p will change the topology of X locally, then p is called an* ***isthmus*** *point. If the removal of p makes X disconnected locally, p is a* ***1D isthmus****. If the removal of p makes $\overline{X}$ connected, p is a* ***2D isthmus****.*

In figure 3.10, p_1 is a 1D isthmus and p_2 is a 2D isthmus.

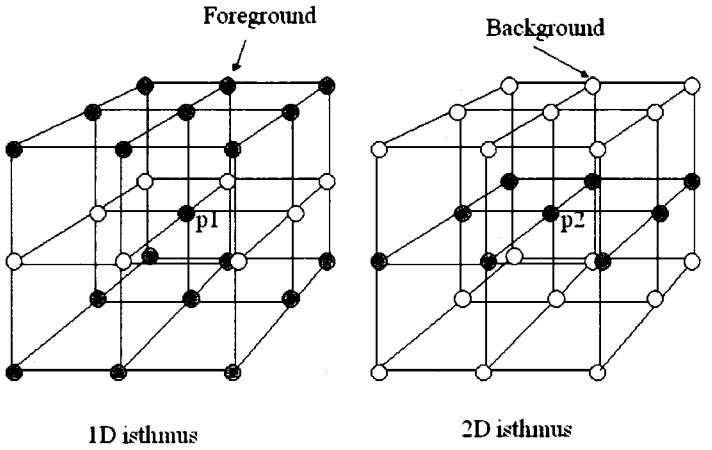


FIG. 3.10 – In 1D isthmus, $p1$ links 2 sets of foreground points, whereas in 2D isthmus, $p2$ connects 2 sets of background points

The topological types and the topological number of a point are closely related. The topological type of a point can be identified by the topological number of the point. Because the topological number is easy to check, it is significant to establish the relationship between them. In the following theorem [1], some rules are introduced to determine the characterization of a given point by using the topological number.

Theorem 18 *Let $X \subset W$ and $p \in X$. We denote $TN = TN_n(p, X)$ and $\overline{TN} = TN_{\overline{n}}(p, \overline{X})$, where $(n, \overline{n})$ is a compatible pair such as $(6,26), (6,18), (18,6)$ or $(26,6)$. Depending on the different possible values of TN and $\overline{TN}$, the point p can be identified as follows :*

1. *If $TN = 0$, p is an isolated (by background) point;*
2. *If $\overline{TN} = 0$, p is an interior (of foreground)point;*

3. *If* $\overline{TN} \neq 0$, *p is a point on the border;*

   If $TN = 1, \overline{TN} = 1$, *p is a simple point;*

   If $TN = 2, \overline{TN} = 1$, *p is a simple 1D isthmus point;*

4. *If* $TN = 1, \overline{TN} = 2$, *p is a simple 2D isthmus point;*
5. *If* $TN \geq 2$, *p is a 1D isthmus point;*
6. *If* $\overline{TN} \geq 2$, *p is a 2D isthmus point;*

3.3 Topological Hull

In the following, the definition of the topological hull will be given first. Then, several theorems will be introduced to establish an easier and more efficient way to calculate the topological hull. At last, the notion of ψ topological hull, which contributes to obtain the optimization of topological hulls of the object, will be presented.

Definition 19 *Suppose X and T are finite subsets of W and* $X \subset T$ *. Then, T is called a topological hull of X if it meets the following conditions :*

1. *T has no hole and cavity;*
2. *For all* $p \in T \backslash X$, $T \backslash \{p\}$ *has a hole or cavity.*

In Figure 3.11, (b) and (d) are topological hulls of (a) and (c) respectively.

It is hard to use the definition directly to compute the topological hull. The theorems in the following section provide an easier way to extract the topological hull. The following theorem gives a local characterization of topological hull sets, which is proved in [1].

Theorem 20 *Let X and Y be finite subsets of W and* $X \subset Y$*. Suppose that Y has no cavities and no holes, then Y is a topological hull of X if and only if,* $\forall p \in Y \backslash X$, *p is an interior point or a 2D isthmus of Y, i.e.,* $T_{\overline{n}}(p, \overline{Y}) \neq 1$.

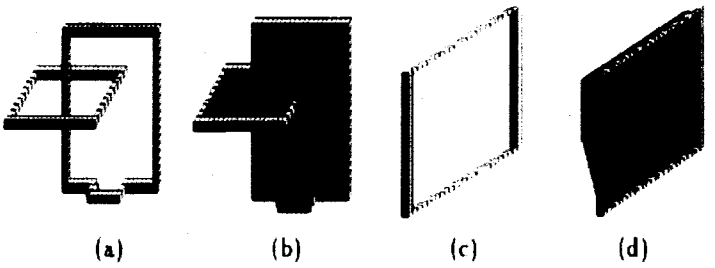


FIG. 3.11 – The subsets in (b) and (d) are respectively the topological hulls of the subsets in (a) and (c)

According to the theorem, building a topological hull can be simplified as follows.

1. Finding a bounding box P of X such that P has no cavities and holes.
2. Deleting the points of P which are not interior or 2D isthmus or points belonging to X.

In addition, in order to formalize the process to check each point in $Y \backslash X$, we can take advantage of the following binary relation.

Definition 21 *(Binary relation) : A binary relation from the set S_1 to the set S_2 is a set of ordered pairs <U, V> where U is an element of S_1 and V is an element of S_2. When an ordered pair <U, V> is in a relation $\mathfrak{R}$, we write $U\mathfrak{R}V$, or $< U, V >\in \mathfrak{R}$. It means that the element U is related to the element V in the relation $\mathfrak{R}$.*

A recursive binary relation can be denoted as $V\mathfrak{R}^m U$, where m is a positive integer. We can express it as follows. $V\mathfrak{R}^{\mathrm{m}}U = V\mathfrak{R}M$, where $\mathrm{M} = V\mathfrak{R}^{\mathrm{m}-1}U, m \geq 2$. When $\exists m$ such that $V' = V\mathfrak{R}^{\mathrm{m}}U$ and $V\mathfrak{R}V' = \emptyset$, we define $V\mathfrak{R}^{\infty}U$. We now define a binary relation on the set $S(X)$.

Definition 22 *Let $X \subseteq W$. We define the binary relation $\Re$ on $S(X)$: $\forall U, V \in S(X), V \in \{V \backslash V \Re U\}$ if $\exists p \in U \backslash X$ such that $V = U \backslash p$ and $TH_{\overline{n}}(p, \overline{U}) = 1$.*

Suppose we have a finite subset B of W and $X, TH(X) \subset B$. The element p of $B \backslash X$ such that $TH_{\overline{n}}(p, \overline{B}) = 1$ will be deleted when being processed with this binary relation. The rest of B are elements with properties of $TH_{\overline{n}}(p, \overline{U}) = 0$ or $TH_{\overline{n}}(p, \overline{U}) \geq 2$. From the previous theorem, We know the rest of B is the topological hull of X.

The above can be summarized as a corollary.

Corollaire 23 *Let X and B be finite subsets of W such that $X \subset B$ and such that B has no cavities and no holes : for each subset Y of W, if $Y = \{Y \backslash Y \Re^{\infty} B\}$. Y is the topological hull of X ($TH(X)$).*

The concepts above provide a theoretical basis to extract the topological hull practically. But the topological hull of a specific set X is not unique. Further steps are needed to find the optimal one among multiple topological hulls of X. The specific topological hull is called ψ topological hull.

3.3.1 The ψ Topological Hull

In Figure 3.12, we can see two instances of topological hulls of a hole in a plane. A layer that covers the hole with various shapes and positions represents different instances of topological hull. In order to close the hole in an appropriate way, we can stretch the layer to the edge of the hole to get the result in Figure 3.12 (b), which is an optimal hull that covers the hole. Alternatively, we can also proceed by finding the points that have minimal distance to the object and compare them to the ones on the layer in the other position. The optimal layer is called ψ topological hull. The definition of the ψ topological hull [1] is as follows.

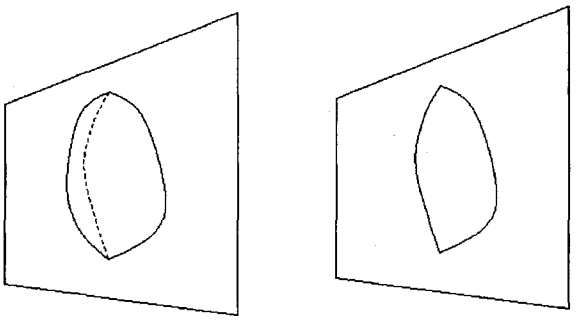


FIG. 3.12 – The picture in (a) is any topological hull, whereas the picture in (b) is a ψ topological hull.

Definition 24 *Let $X \subset W$ and let ψ be a function from $\mathbb{Z}^3$ to $\mathbb{N}$. We define the binary relation $\mathcal{Q}$ on $S(X)$: $\forall U, V \in S(X), V \in \{V \backslash V \mathcal{Q} U\}$ if $\exists p \in U \backslash X$ with $TH_{\overline{n}}(p, \overline{U}) = 1$ such that : $V = U \backslash p$ and $\psi(p) = max\{\psi(y), \exists y \in U \backslash X$ with $TH_{\overline{n}}(p, \overline{U}) = 1\}$. Let $B \in S(X)$ be a set with no holes and no cavities. If $Y \in \{Y \mathfrak{R}^{\infty} B\}$, then Y is called a ψ topological hull of X.*

In order to compute the ψ topological hull, the process should be strictly controlled under the distance of the points. The concept of *Distance transformation* is introduced to compute the minimal distance of each point to the object. The method will be described in detail in the next chapter.

CHAPTER 4

Topological Filtering of Reconstructed Surfaces

In this chapter, 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 [1]. First, embedding the surface in a volumetric representation automatically fills in all 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.

In the following sections, several preprocessing steps will be introduced. First, the method used to embed a mesh surface into a discrete volumetric representation is described, then

the calculation and optimization of the digital distance transformation are explained and discussed. The algorithm of hole closing for volumetric surface will be presented and explained in the third section. At the end, we present some experimental results.

4.1 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 collection L;
2. Check the cube size and delete it from L :

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

   else delete the cube from L then continue.
3. Divide the cube into 4 smaller equal size cubes;
4. Check the intersection of these cubes and triangles :

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

   If there is no intersection, ignore it.
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 between a given triangle of the surface and a voxel of the grid. Fortunately, in Graphics Gems III, Douglas Voorhies [25] gives an algorithm that detects whether a given triangle intersects the axially-aligned unit cube centered at the origin. Voorhies's method uses the general approach which proceeds from some trivial cheap acceptance and rejection tests, through more expensive edge and face intersection tests. Finally, he checks whether the interior of the triangle is penetrated by the cube.

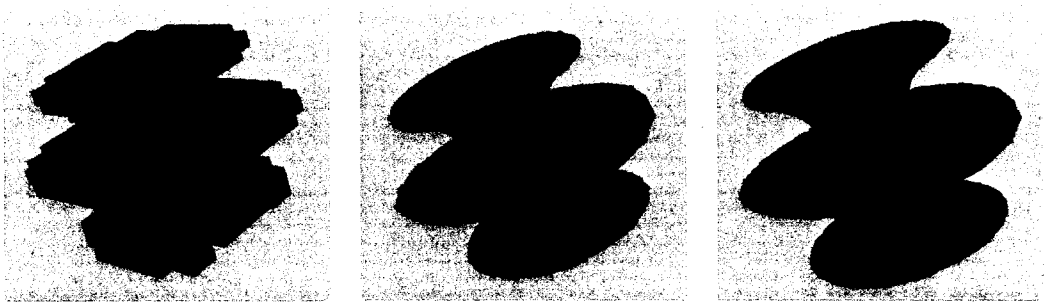
Let cube C be a unit cube centered at the origin and T a triangle of the mesh. The intersection algorithm is as follows.

1. Check if any of the vertices of T are inside C. If so return true (trivial accept).
2. If not (trivial reject, all vertices are outside the cube), test every edge of T against the 6 faces of C. If there is an intersection, return true.
3. If not, check any of the four diagonals of C intersects T. If there is an intersection, return true.
4. If not, return false (no intersection of T and C).

Figures 4.1 and 4.2 show the volumetric representations of the reconstructed surfaces of a golf club and a hyper sheet 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)

FIG. 4.1 – 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



(a) (b) (c)

FIG. 4.2 – Volumetric representations of a surface of a hyper sheet at different resolutions (a) 0.1. (b) 0.01. (c) 0.005.

4.2 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, 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 distance 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.*

4.2.1 Definition of the Digital Distance Transformation and Notations

Definition 25 *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 the 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 $\overline{X}$ represent the foreground and the background of the image respectively. $DT(i,j,k)$ denotes the digital distance from a given voxel to the object, where i,j,k represent distances along axes directions and they are in increasing order with $0 \leq k \leq j \leq i \leq M$.

The distance transformation can be expressed as follows [7] : $DT(i,j,k) = min\{D(max\{|i-p|,|j-q|,|k-r|\}, mid\{|i-p|,|j-q|,|k-r|\}, min\{|i-p|,|j-q|,|k-r|\}), p,q,r \in \overline{X}\}$, 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. In the following section, we introduce some computation methods for the digital distance transformation.

4.2.2 Computation Methods

The basic idea [7] 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 [24].

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 noted 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.

4.2.3 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.

4.2.4 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 as shown in the table below.

Path	Steps	Inequality
a	b	$2a < 2b$
a	c	$2a < 2c$
a	b c	$a < b + c$
b	a	$b < 2a$
b	c	$2b < 2c$
b	a c	$b < c + a$
c	a	$c < 3a$
c	b	$2c < 3b$
c	a b	$c < a + b$

TAB. 4.1 – Regularity Criteria for $3 \times 3 \times 3$ Distance Transformations.

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 [7] 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 $a = 1$.

4.2.5 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 [7] did some approximation tests. The results are listed in the following table.

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

TAB. 4.2 – Integer $3 \times 3 \times 3$ Distance Transformations.

For instance, let three kinds of local distances of $3\times3\times3$ 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.

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 outside of the given object. The algorithm can is as follows.

1. Suppose $M \times M \times M$ image, where $3 \leq M$. Initialize the value of the distance of each voxel on the object in the image to be *zero* and others to be *Infinite*;
2. Label every voxel as *unvisited*;
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 as 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 as 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 the distances of points outside of the object are set to infinity initially, the distances of the points that are next to the object are computed 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 the background voxels that the first scan has not given.

4.3 The Hole Closing Algorithm

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 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 means they are on the boundary. It is not difficult to observe that the voxels in 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 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 4.3. 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.

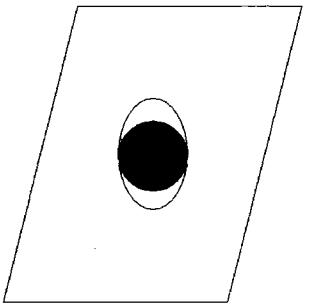
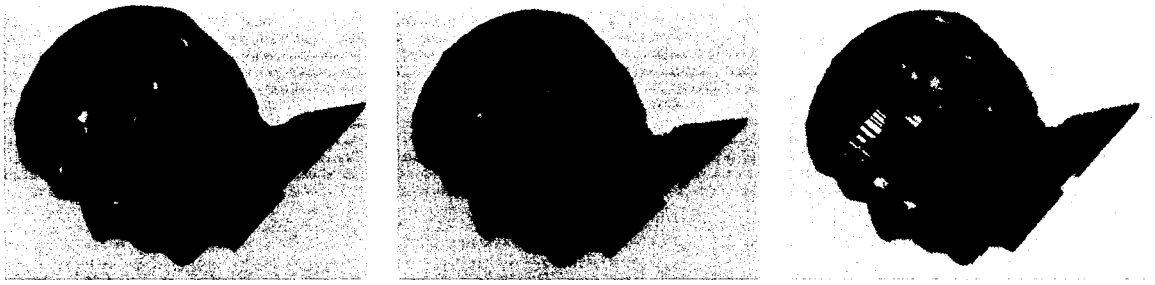


FIG. 4.3 – Suppose that only one layer is left and the distance d of the voxel O in the center of the hole is the furthest one to the object surface. Then, the voxel is a 2D isthmus.

4.4 Results

The algorithms of surface reconstruction, volumetric representation, and 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 4.4, 4.5 and 4.6. Figure 4.4(a) shows the reconstructed surface of a head using the algorithm in the previous section. The surface exhibits several unwanted holes. Figure 4.4(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 0.03 as shown in Figure 4.4(c). Figure 4.5(b) shows the volumetric representation of the surface in Figure 4.5(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. The remaining holes are closed by the hole closing algorithm since the hole diameter cannot exceed 0.03 as shown in Figure 4.5(c). Figure 4.6(c) shows that the hole closing algorithm closed all the holes in the golf club when the

hole diameter can exceed 0.03.



(a) (b) (c)

FIG. 4.4 – (a) The reconstructed surface of a head. (b) The volumetric representation. (c) Hole closing with hole diameter less or equal than 0.03.

(a) (b) (c)

FIG. 4.5 – (a) The reconstructed surface of a golf club (b) The volumetric representation. (c) Hole closing with hole diameter less or equal than 0.03.

(a) (b) (c)

FIG. 4.6 – (a) Volumetric representation of the surface of a golf club (different view). (b) Hole closing with hole size less than or equal to 0.03 (different view). (c) Hole closing with hole diameter less or equal than 0.06.

Input : ϵ, M;
Output : Topological hull of the object in the image.

Set all voxels 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 the grid voxels;

Define a list array $L[MAXDIST]$ with the size of maximum distance of DT;

Assign the coordinates of each voxel to 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 a in the $M \times M \times M$;
                    2. It is not in the 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 4: Algorithm of Hole Closing In 3D Surface

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 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.

BIBLIOGRAPHY

[1] Z. Aktouf, G. Bertrand, and L. Perroton. A three-dimensional holes closing algorithm. *Pattern Recongnition Letters*, 23 :523–531, 2002.

[2] N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. *Proceedings of the 14th ACM Symposium on Computation Geometry*, pages 39– 48, 1998.

[3] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton : combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60/2 :2 :125–135, 1998.

[4] 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)*, pages 415–421, 1998.

[5] C. Barber, D. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *The ACM Transactions on Mathematical Software (TOMS)*, 22 :469 – 483, 1996.

[6] JD. Boissonnat. Geometric structures for threedimensional shape reconstruction. *ACM Trans. Graphics*, 3 :266–286, 1984.

[7] G. Borgefors. On digital distance transforms in three dimensions. *Computer vision and image understanding*, 64 :368–376, 1996.

[8] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *Proc. SIGGRAPH´ 96*, pages 303–312, 1996.

[9] M. de Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry : Algorithms and Applications.* Prentice-Hall, 1997.

[10] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Trans. on Information Theory*, 29 :551–559, 1983.

[11] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. Three dimensional alpha shapes. *ACM Trans. Graphics*, 13 :43–72, 1994.

[12] A. W. Fitzgibbon, M. Pilu, and R. B. Fisher. Direct Least-Squares Fitting of Ellipses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5) :476–480.

[13] P.J. Giblin. *Graphs, Surface and Homology.* Chapman and Hall, London,England, 1977.

[14] H. Hoppe. Surface reconstruction from unorganized points. *Computer Science and Engineering, U. of Washington*, 1994.

[15] H. Hoppe and al. Surface reconstruction from unorganized points. *Proc. SIGGRAPH´ 92*, pages 71–78, 1992.

[16] H. Hoppe and al. Piecewise smooth surface reconstruction. *Proc. SIGGRAPH´ 94*, pages 19–26, 1994.

[17] T.Y. Kong and A. Rosenfeld. Digital topology : Introduction and survey. *Computer vision. Graphics and image proceeing*, 48 :357–393, 1989.

[18] Michael J. Laszlo. *Computational Geometry and Computer Graphics in C++.* Springer, 1996.

[19] W. Massey. *Algebraic Topology : An Introduction.* Brace World, Inc., 1967.

[20] J. Munkres. *Topology.* Prentice Hall., 2000.

[21] Noname. http ://people.csail.mit.edu//*fsegonne*//research//*topology*//topology.html.

[22] P. Preparata and Michael Ian Shamos. *Computational Geometry : An Introduction.* Springer-Verlag, 1985.

[23] R. J. Prokop and A. P. Reeves. A Survey of Moment-Based Techniques for Unoccluded Object Representation and Recognition. *CVGIP : Graphical Models and Image Processing*, 54(5) :438–460.

[24] A. Rosenfeld and J. Pfaltz. Swquential operations in digital picture processing. *J. Assoc. Comput.*, 13 :471–494, 1966.

[25] D. Voorhies. Triangle-cube intersection. *Graphics Gems III*, pages 236 – 239, 1992.