Published in final edited form as:

Proc Pac Conf Comput Graph Appl. 2000 ; 2000: 243–448. doi:10.1109/PCCGA.2000.883947.

Turning Unorganized Points into Contours

Yong Zhou and Arthur W. Toga

UCLA Laboratory of Neuro Imaging

Abstract

There are a variety of research applications that require reconstruction of objects from unorganized points. In our implementation, we accomplish this task in two steps: first by connecting points to contours and then contours to objects. We focus on the first step in this paper. We present a Voxel-coding algorithm which assembles unorganized points into contours in a straightforward and efficient way. First the points are converted into a binary volumetric object using simple 3D Voxel-coding starting with voxels that include the sample points. Then contours are interpreted as centerlines of cross-sections. The centerlines are obtained by using a series of 2D Voxel-coding operations.

Input points are sampled from an unknown object satisfying certain sampling criteria. No additional details about the input data are needed. Output contours pass through or approximate sample points. The algorithm is tested with several data sets, showing its efficiency.

1 Introduction

Polygonal objects can be reconstructed from several types of data. Volume data, unorganized points, range data, and contour data are typical examples. A variety of data-dependent approaches have been proposed to achieve reconstruction, such as Marching Cubes [12] for volume data, zippering techniques [16] and the volumetric method [4] for range data, α shape [6], zero-set [9], and Voronoi-based methods [1] for unorganized points, voxel-based methods [10] and Voxel-coding techniques [21,22] for contours.

Interestingly, several of these data-dependent methods, essentially, reflect the conversions between different types of data. For instance, in order to employ the advantages of Marching Cubes, Hoppe *et al.* [9] and Curless *et al.* [4] convert unorganized points into volume data by calculating the directed distance of grid points to tangent planes at the closest points; Jones and Chen [10] convert contours into volume data by simple interpolation. For tiling complex volumetric objects, volume data is often converted into contours for the synthesis and control of generated triangular meshes.

However, for surface reconstruction from unorganized points, little work has been reported on the conversion from unorganized points into contours. This work is important because contour-based reconstruction methods are widely used in many different implementations. Particularly vexing problems such as multiple branching in contour-based methods can now be solved in a fully automatic, systematic, and straightforward way [2,21,22]. Once contours are available, these methods can be used for object reconstruction.

One of the most common methods for contour generation consists of deforming predetermined curves to fit a set of data points [11,13,18]. A satisfying approximation can be obtained if a deliberately designed equilibrium equation and a potential field are established and appropriate parameters are chosen, such as the deforming direction and moving step – all these strictly

depend on a prior knowledge of the distribution of the input points. Recently, Hemayed and Farag [8] generate contours from unorganized points using this method. But many of related issues are not discussed.

Two related approaches have been proposed to extract skeletal curves from unorganized points. Bittar *et al.* [3] first convert the point set into a volumetric object, then use a traditional skeletonization algorithm to extract skeletons from the voxelized objects. However, no theory is given to prove that the voxelized object is always closed so that the inside and outside voxels can be classified. In addition, great attention must be paid in connection of skeletal points. In comparison, Verroust and Lazarus [17] interpret the skeletal curves as line segments, successively connecting the center points of the shortest path levels relative to a given source point. The levels are generated based on the neighborhood and geodesic graphs.

There are also other varieties of contour extraction and curve-fitting methods proposed [5,7,15,20,21]. However, most of them have certain assumptions or requirements. For instance, mathematical functions are given[5], volume data are available[15], the data points must be arranged according to special features of range images[14], the endpoints of a curve must be known[7,20], or the operation must be guided by direct manual tracing.

The following related issues remain unsolved. A complicated object has several separate components intercepted by a slice, a component could be nested in another one. The unorganized points sampled from such an object within this slice should be divided into the same number of sets so that each set of points corresponds to a component. The issues include: (1) *how to efficiently group the sample points within a slice so that such a set of points can be fitted by contours*, (2) *how many contours are required to fit a point set*, and (3) *how to determine whether a closed or an open curve is appropriate*.

Our ultimate goal is to reconstruct objects from unorganized points in two steps; first by connecting points to contours and then contours to surfaces. We focus on the first step in this paper. We present a Voxel-coding algorithm which turns unorganized points into contours accurately and efficiently. The input points are sampled from an unknown object, satisfying definitions given by Hoppe *et al.* [9]. Our algorithm solves the above three problems. The algorithm includes two basic operations: 3D Voxel-coding and 2D Voxel-coding. First, the 3D operation is responsible for converting the input points into binary volumetric objects. Then contours are interpreted as centerlines of cross-sections. Several 2D operations are responsible for obtaining these centerlines. The first 2D Voxel-coding extracts initial contours, the second maps all contour points into the closest sample points, and the third inserts points between adjacent contour points.

There are two original methods proposed: efficient region clipping and robust contour extraction from arbitrarily complicated binary images.

The rest of the paper is organized as follows. After introducing Voxel-coding techniques in Section 2, we first discuss the conversion from unorganized points to volumetric objects in Section 3, then contour generation in Section 4, including initial contour extraction (Section 4.1) and final contour generation (Section 4.2). Example applications and results are given in Section 5. Finally, conclusions are presented in Section 6.

2 Related Voxel-Coding Techniques

2.1 Preliminaries

Voxel-coding plays a critical role in our contour extraction. It works within 3D volumetric objects and 2D slices. We follow the concepts used in [20]. For a voxel p , an adjacent voxel

q is called an *F-neighbor*, *E-neighbor* or *V-neighbor* of p if they share a face, an edge or a vertex, respectively. In this case, they are also called *F-connected*, *E-connected* or *V-connected*.

Within a slice, that two pixels are *V-connected* or *V-neighbors* (*E-connected* or *E-neighbor*) means they share a vertex (a grid edge). Two pixels are *connected* if they are V-neighbors or E-neighbors. A *2D path* is a sequence of pixels if adjacent pixels are connected and every other pair of pixels are disconnected. A *2D region* is a non-empty set of pixels that if, for any two pixels within it, there is a path within the set connecting them; otherwise it is disconnected. Usually, regions divide their slice into three parts: inside, on, and outside. Inside pixel are those which do not belong to, but are surrounded by, such a region. A *hole* is defined as a set of E-connected inside pixels (note not V-connected). A pixel in a region is called an *exterior or interior boundary pixel* if it is E-adjacent to outside or inside pixels. The exterior (interior) boundary of a region is the set of all exterior (interior) boundary pixels.

Voxel-coding is a recursive voxel or pixel propagation and coding operation within a volumetric object or a region R , starting with a seed set S (S is a subset of R) and using a specific coding metric M until constraint conditions are met. The concept of Voxel-coding was described for 3D in [20].

A pixel propagation using the “ $n_e - n_v$ ” metric (n_e and n_v are integers and $n_e < n_v$) can be described as a recursive process: First, all the pixels in R are initialized with a value of infinity, then the propagation starts with all the seed points in S assigned a value of zero. All their E-neighbors are assigned a value of n_e and V-neighbors a value of n_v . Iteratively, suppose pixels with a value of n are processed—All their E-neighbors and V-neighbors are assigned values of $n + n_e$ and $n + n_v$, respectively, if these values are smaller than their current values. This process continues until the constraint conditions are met (see [20] for more details).

In some applications, we need to test whether a region R is E-connected or to visit only E-connected parts of R . In this case, the propagation can be constrained so that only E-neighbors are visited and assigned values rather than both E-neighbor and V-neighbor in each iteration. A 2D Voxel-coding with only E-neighbors visited in each iteration is referred to as *E-coding*. For distinction, the original 2D Voxel-coding is referred to as *EV-coding*. Thus, if R is not E-connected but V-connected, the E-coding cannot retrieve all the pixels in R . However, the EV-coding can.

2.2 Properties

Shortest Path Extraction—One of appealing features of Voxel-coding is the Shortest Path Extraction called the *SPE* procedure (see [20] for details). In this paper, only the 2D *SPE* procedure is applied. On the basis of a coding field, starting with a point, the *SPE* procedure recursively extracts the adjacent point with the smallest code. This process stops when a seed point is found. The extracted path is taken approximately as the shortest path within the coding region which connects the starting point to its closest point in the seed set. It should be noted that, in choosing the next point, if there is more than one candidate, the E-neighbors have a higher priority. The extracted paths satisfy the definition of a path. Furthermore, different paths have the following properties.

Theorem 1—Let R be a connected set of pixels and S be its non-empty subset. EV-coding is implemented in R with S as seed set using the “ $n_e - n_v$ ” metric. Suppose that C_1 and C_2 are the two shortest paths, extracted by the *SPE* procedure starting from any two points in R and ending with points in S . Then C_1 and C_2 cannot cross each other, and if they have a common point, then the remaining points in both paths must be the same.

Proof: Let p_i and p_{i+1} be any two adjacent pixels in C_1 . If C_2 crosses C_1 between p_i and p_{i+1} , p_i and p_{i+1} must be V-connected, and there must be a point q_j which is in C_2 and E-adjacent to both p_i and p_{i+1} . We prove that the next point of q_j in C_2 cannot be another E-neighbor q of both p_i and p_{i+1} . According to the SPE procedure, two E-neighbors of p_{i+1} must have higher code values than p_{i+1} , otherwise, p_i 's next point is not p_{i+1} , but its E-neighbors. Thus q_i 's next point cannot be q but could be p_{i+1} since p_{i+1} is adjacent to q_j and has lower code value than q . Therefore, two paths can never cross each other. If they have a common point, the same criteria apply for the remaining point extraction, thus they must be the same.

Note that the above proof does not depend on a specific metric – this is desirable for the selection of more complicated coding metrics for more accurate path extraction.

Region Boundary Tracing—Generating curves that pass through the exterior boundary of a 2D region can be difficult. Often arbitrary regions possess very complex boundaries. For instance, they can include (a) very small branches, (b) tiny holes consisting of only one inside pixel, or (c) collapsed portions in the middle. A robust method must be able to handle all these cases. The following results will be the foundation for our contour extraction.

Theorem 2—*Let E be the exterior boundary of a region R , and suppose E (taken as another region) contains n holes ($n \geq 1$). Then there are n closed curves within E such that*

1. Any curve can be represented as a closed path which surrounds a hole,
2. Any two curves can contact but cannot cross each other, and
3. An interior boundary pixel of E must belong to one of these curves.

Proof: Given a hole H , let C be the set of the pixels of E which are E-connected to H . We first prove that a pixel p in C is connected to only two pixels in C . Obviously, p has at least two connected points in C , since C constructs a chain of pixels surrounding H . If p has three or more connected points in C , there must be one of them which is not E-connected to H , since H is E-connected. This conflicts with the definition of C . Then we prove that C forms a path. Starting from any point in C and along one direction, the next point is taken as a V-neighbor or E-neighbor in C . Since the next point is unique, this process can proceed until the last point is traced to close the path. According to the definition of holes, holes are separated by pixels in E , thus any two curves can contact but cannot cross each other. Furthermore, by construction, an interior boundary pixel of E must belong to one of these curves.

Fig. 1(1) shows such a region R without holes. Its exterior boundary E is shown in Fig. 1(2). E is another region containing three holes. The interior boundary of E (Fig. 1(3)) removes the pixels in E which are not E-adjacent to pixels of holes, and keeps E 's configurations, and constructs three subsets. Each subset forms a sequence of pixels surrounding a hole, marked by a closed curve in Fig. 1(4).

According to Theorem 2, each contour extracted from R corresponds to a hole of E . A hole can be retrieved by E-coding with its pixels marked with the index of the hole so that the pixels in E which are E-adjacent to the same hole construct a closed curve. Theorem 2 indicates that small branches of an exterior boundary can be removed by considering its interior boundary. Tiny holes and collapsed situations can be recognized by checking the contents of holes.

3 Conversion from Unorganized Points into Volumetric Objects

Suppose that we are given a set X_{pt} of points sampled from an unknown object O_{bj} , satisfying ρ -dense criterion [9] that any sphere with radius ρ and center in O_{bj} contains at least one sample point in X_{pt} . We are concerned with the problem: how can a set of points in X_{pt} sliced by a

cross-section of O_{bj} be converted into a contour model so that O_{bj} can be reconstructed from these contours? Our method is to convert X_{pt} into a volumetric object first.

3.1 3D Conversion

After X_{pt} is input, its bounding box is divided uniformly in three axis directions into a volume with a voxel size of β so that each voxel contains n sample points. In addition to the distribution of points, the volume dimension strictly depends on the size of β . Further, β depends on n . Ideally, the voxel size β is taken as the maximum which guarantees n to be a value of one. We first have β then calculate n , n is evaluated as the value of the total number of sample points divided by the number of voxels which contain sample points. If the value of n is larger (smaller) than its threshold, β decreases (increases) accordingly. In the present experiment, n is chosen as a floating point value ranging from 1.0 to 1.5.

Once the volume is determined, voxels containing sample points are labeled as object voxels and assigned a value of one; otherwise, they are background voxels assigned a value of zero. Initially, the object voxels are disconnected from each other.

The purpose of conversion is to turn the smallest possible number of background voxels, which are close to object voxels, into object voxels so that the whole object O_{bj} can be included in these object voxels. To this end, 3D Voxel-coding is applied with all the object voxels as the seed set, the visited background voxels are marked as object voxels. The voxels closest to object voxels are visited first, and the closer to object voxels the background voxels are, the more likely they will be visited.

Theorem 3—Given a set X_{pt} of points sampled from an object O_{bj} satisfying ρ -dense criteria and its associated volume R with a voxel size of β . Then, there is a number $n_{ly} = \sqrt{3}\rho/\beta$ so that any point in O_{bj} must be contained in an object voxel after n_{ly} layers of voxel propagation starting from voxels containing points in X_{pt} and using a 3D Voxel-coding.

Here a layer of voxel propagation means one recursive iteration in which only F-neighbors of the last layer of voxels are visited.

Proof: According to the ρ -dense criteria, any sphere S_{ph} with radius ρ and center x in O_{bj} contains at least one point p in X_{pt} . If point x can be reached in n_{ly} layers of voxel propagation, the above result holds. In fact, the value of n_{ly} is a conservative estimation. We consider the worst case, i.e., suppose p is on the boundary of S_{ph} . A 2D explanation is illustrated in Fig. 2. It shows a portion of voxel-coding from seed point p and related sphere centered at x . We analyze how many layers of propagation are required before x can be reached. The figure indicates that every two consecutive coding layers make the visited region extend at least a length of $\sqrt{2}\beta$. In other words, each coding layer propagates a length of $\beta/\sqrt{2}$ on average. Thus, to propagate a length of ρ far away along the slowest direction (i.e., the grid diagonal direction), it needs $\sqrt{2}\rho/\beta$ layers of propagation. On the basis of a similar derivation in 3D, we have $n_{ly} = \sqrt{3}\rho/\beta$.

Theorem 3 provides a full condition for turning sample points into a volumetric object which approximates O_{bj} after n_{ly} layers of voxel propagation and labeling. Fig. 3 shows an example, and Fig. 4(a1) shows a slice after 3D Voxel-coding.

In applications, if ρ is unknown, an interactive operation by the user is required for the choice of n_{ly} .

Let us consider the results within a slice after the above 3D Voxel-coding. Usually, a slice consists of several separate regions. Each region is a set of connected object voxels with the same z-value, i.e. the set of non-zero pixels. Each region corresponds to a connected component of the object within the slice. Thus the 3D coding operation gets all sampling points which originally belong to the same component into the same region.

3.2 2D Region Clipping

Since the parameter n_{ly} in Theorem 3 is a conservative estimation, after n_{ly} layers of isotropic voxel propagation, the originally disconnected regions could be merged to form one region. For instance, the region in Fig. 4(a1) consists of three obviously separate parts from the original object. If these parts are not separated, it is difficult to automatically determine how many contours are required to fit this region. This situation happens in the neighborhood of branching points, where a slice usually intersects two very close but disconnected branching regions.

Here we propose a method, called *region clipper*, to split these regions. The region clipper follows the 3D coding conversion. According to the spatial coherence of an object, as a slice moves along the z-direction far away from the branching points, the intersected branches become explicitly separated even after 3D voxel-coding. The basic idea of clipping is to project an adjacent slice π_a onto the current slice π_c and to check whether a region R_c in π_c is covered by at least two disconnected regions in π_a , such as R_{a1} and R_{a2} . If this is the case, R_c is a candidate to be split by clipping off a strip of its pixels which do not include sample points using R_{a1} or R_{a2} as a clipper.

In order to detect such a strip of pixels, we perform 2D EV-coding within π_c by taking the projection of R_{a1} as a seed set and propagating towards the outside of R_{a1} . After a limited number of layers of propagation, we check if there are at least n_{ly} (see Theorem 3) consecutive layers of visited pixels in R_c without including any sample points. If there are, R_c is split by turning these layers of pixels into zero-pixels. Otherwise, the operation with R_{a1} fails, R_{a2} is chosen as a new clipper and another test is performed. This is a recursive process, that one region is split means that its corresponding regions in adjacent slices also should be tested accordingly.

Once clipping is completed, each separate region is tested to determine whether it contains sample points. If not, the region is discarded. This process can remove tiny holes and small branches. Since a clipper is taken as a region in an adjacent slice which has a similar shape as the clipped region, region clipping works fairly well with complicated regions. Fig. 4(b1) shows the clipping result.

After clipping, all the sample points within the same region construct a set ready for contour extraction. Thus we solved the first problem: how to group the sample points.

4 Contour Generation

The purpose of this section is to extract contours from slices of the volumetric object generated above. All the following operations are implemented in 2D slices.

We face challenges which are different from those addressed in [5,7,11,15,18]. First, the sample point set X_{pt} is ρ -dense, but the point set within a slice does not strictly meet ρ -dense criteria (see Fig. 6); second, it is unknown how many contours can be generated, whether these points construct closed or open contours, and how the endpoints of a contour can be determined if it is open.

4.1 Initial Contours

As we mentioned before, each slice consists of non-zero regions (see Fig. 4(a1) where zero-pixels are in black and non-zero pixels are in white or grey). For a closed object, most regions contain holes. A region could be nested in another region. The basic idea of initial contour extraction is to recognize all the regions, and for each region, Theorem 2 is applied to extract contours.

First we recognize all the regions. Algorithmically, starting with the boundary of the slice, E-coding is performed over zero-pixels. The result is all exterior boundary pixels of regions are reached. These pixels can be divided into different sets, each set corresponds to a region, i.e., constructs its exterior boundary. The division can be made by another pixel propagation with connected pixels into the same set. Consequently, the exterior boundary of a region is obtained.

Then for each region, Theorem 2 is applied for contour extraction. This can be accomplished by first retrieving pixels of holes and then searching the exterior boundary pixels of the region which are E-adjacent to a hole.

The above process cannot reach a region which is completely enclosed in other regions. Therefore, for each obtained region, a similar process is performed starting with its interior boundary, this is a recursive process which continues until a non-nesting region is met.

According to Theorem 2, the number of contours extracted from a region is equal to the number of holes which its exterior boundary includes. The same exterior boundary pixel could occur in different contours. Thus extracted contours could contact or coincide in the middle, but could not cross each. In extremely complicated objects, generated contours within a region construct a graph.

Here, we have described a robust and complete contour extraction algorithm from a binary image. Meanwhile, we solved the second problem: how many contours are required to fit a connected region. By using coding strategy, our method avoids the multiple path tracing in traditional counterpart.

Figs. 4(a3) and 4(b3) demonstrate the contour extracted. Although the initial contours pass through exterior boundaries of all regions, not sample points, the significance lies in the establishment of connectivity for the unorganized points. On the basis of this connectivity, we can easily make them pass through sample points.

4.2 Final Contours

Several methods can be employed to obtain the final contours. One is skeletonization [19,20], another is the de-formable model [11]. Alternatively, we implement the following steps on a contour consecutively:

Step 1 maps contour points into their closest sample points. First, EV-coding is implemented within the region starting from all sample points using the '3-4' metric. Then, each contour point leads to the shortest path connecting the point to the closest sample point, called its target. This path is extracted by the *SPE* procedure. Thus the points are replaced by their target while the connectivity of initial contours is maintained. According to Theorem 1, the curve after replacement cannot self-intersect in the middle. Usually several adjacent contour points could be mapped to the same sample point; in this case, only one point is retained while other coincident ones are removed. Also, non-adjacent contour points could be mapped to the same point, the folded line segments are removed too. Fig. 4(a4) shows the results after mapping on (a3).

For a dense data set, such as the dragon (see results in Section 5), the intervals between adjacent contour points in some slices are short enough to represent the final contours. However, a ρ -dense data set does not imply a strict ρ -dense sampling within each slice. Consequently, the intervals between adjacent contour points could be relatively larger than ρ , so the following steps are required.

Step 2 extracts the shortest paths connecting adjacent contour points with an interval exceeding a threshold. A direct linear interpolation is undesirable, since it is impossible to guarantee that inserted points lie within the non-zero region. This requirement is essential for avoiding intersections between the contours. Our approach is to extract the path using the *SPE* procedure, and take it as the intermediate contour segment in between. Not only does each inserted point lie on the same connected non-zero region, but also adjacent points share at least a grid corner.

Step 3 moves the inserted path to the medial axes. An ideal contour should pass through sample points or medial axes if sample points are not available. Inserted points may not be centered relative to the non-zero region boundary. In this step, each non-sample point x is directly replaced by its medial point. The medial point of x is defined as a point in the same region and with a minimum non-zero value along the direction normal to the curve, and its distance to x does not exceed ρ . Here the value means one relative to the EV-coding within the slice generated in Step 1. The replacement is recursive, since after the operation, the interval could be increased again. The same operation is recursively performed on the increased intervals. This case usually occurs where the object O_{bj} extends deeply inside. Fig. 4(b4) shows the final contour.

According to the above analysis, final contours do not change the topological connectivity of the initial contours, but a contour could (1) collapse into a point, (2) collapse into a sequence of lines, or (3) contact its other side in the middle.

Case 1 happens when a region includes few sample points; Case 2 indicates an open contour is detected; Case 3 usually indicates that the region, from which its initial contour is extracted, originally consisted of two separate regions but had not been clipped into two parts successfully (see Fig. 4(a4)), or there was an undersampling within the region. In both cases, the postprocessing is implemented by checking generated contour points, resulting in a separation of the contours at the contact points.

5 Applications and Results

Three data sets are used to test our algorithm. The algorithm is written in C, and run on a SGI Power Onyx – R10000 CPU. All the color pictures are presented as follows. The sample points are represented in white voxel form while contours are ordered counterclockwise and colored as in Fig. 4(b4). There are five parameters associated with each point set: the number of points, volume dimension, voxel size β (relative to input point coordinates), number of coding layers for generating corresponding binary objects, time (in seconds) for generating the corresponding contours including point input, 3D Voxel-coding, preprocessing for region clipping, initial contour extraction, and final contour generation. For each data set used, the sampling parameter ρ is not available, so the number of coding layers was chosen by experience. Thus ρ can be derived according to Result 4. Table 1 provides the details.

Fist

Fig. 5 shows the combination of sample points and generated contours. Contours are extracted from all the slices. The set of points is not strictly ρ -dense, particularly in the joints between fingernails and the palm where points are undersampled. Four layers of voxel propagation make the whole fist filled except the joints.

Fig. 6 shows the details in the neighborhood of these joints after one more layer of propagation is added. For clarity, only contours in every other slice are shown. The increase of coding layers makes the regions of the same cross-section extend towards each other, resulting in the merging of these regions, such as in the upper part of the fist. Preprocessing is applied to separate them.

Man's head

Fig. 7 shows the fitting of points sampled from a man's head, and contours extracted from each slice. The data is densely sampled around the ears, eyes, nose and mouth. The original points are sampled with the top of the head towards positive z-axis, face down; thus contours are colored starting at the front of the head.

Dragon

This is a densely sampled data set, only three layers of voxel propagation are required to convert the points into a volumetric object. Fig. 8 shows the reconstruction from partial consecutive slices where contours are extracted in every other slice. The current viewpoint is chosen in the direction of the positive z-axis, and the intervals between z-slices are enlarged. The figure indicates that extracted contours only pass through or approximate sample points close to the exterior boundary of the object. Fig. 9 gives a complete view of all generated contours.

6 Conclusions and Future Work

We propose a robust Voxel-coding algorithm for turning unorganized points into contours efficiently and systematically.

Contours are generated in two basic operations: turning unorganized points into 3D volumetric objects and extracting contours from the volumetric objects. In both operations, Voxel-coding plays a critical role. After the initial volume is created with object voxels containing sample points, 3D Voxel-coding is performed starting from sample points, resulting in neighboring voxels becoming object voxels. This allows the unknown surface to be contained in a generated volumetric object. A series of 2D Voxel-coding techniques are used for initial contour extraction, mapping to sampling points, and contour point insertion in sparse sampling intervals. The generated contours can be directly used as an input of contour-based systems for surface reconstruction.

Two useful methods are proposed: efficient region clipping and robust contour extraction from arbitrarily complicated binary images. Both theoretical proofs and experimental results are given, showing the efficiency of our algorithm.

Currently we are integrating our algorithm introduced in this paper with a tiling algorithm to establish an integrated 3D system for surface reconstruction from unorganized points, contour data, and general volumetric objects. Meanwhile, we are also investigating extension of our algorithm for application to sparsely sampled data.

Acknowledgments

We thank Paul M. Thompson for reading the draft, Ping Fu (Raindrop Geometric) for providing the fist data, Hughes Hoppe (Microsoft) for the head data, and the Stanford Data Repository for the dragon data. This work has been supported by the National Science Foundation DBI 9601356 and the National Institutes of Health NCRR RR13642 and NS38753.

References

1. Amenta N, Bern M, Kamvysselis M. A New Voronoi-Based Surface Reconstruction Algorithm. SIGGRAPH'98 1998:415–421.

2. Bajaj CJ, Coyle EJ, Lin K. Arbitrary Topology Shape Reconstruction from Planar Cross Sections. *Graphical Models and Imaging Processing* 1996;58(6):524–543.
3. Bittar E, Tsingos N, Gascuel MP. Automatic Reconstruction of Unstructured 3D Data: Combining a Medial Axis and Implicit Surfaces. *Computer Graphics Forum (EUROGRAPHICS'95)* 1995;14(3)
4. Curless B, Levoy M. A Volumetric Method for Building Complex Models from Range Images. *SIGGRAPH'96* 1996:303–312.
5. Dobkin DP, Levy SVF, Thurston WP, Wilks AR. Contour Tracing by Piecewise Linear Approximations. *ACM Transactions on Graphics* 1990;9(4):389–423.
6. Edelsbrunner H, Mucke EP. Three-dimensional Alpha Shapes. *ACM Transactions on Graphics* 1994;13:43–72.1994
7. Fang L, Gossard DC. Multidimensional Curve Fitting to Unorganized Data Points by Nonlinear Minimization. *Computer-Aided Design* 1995;27(1):48–58.
8. Hemayed EE, Farag AA. Slicing, Fitting and Linking (SFL): A Modular Triangulation Approach. *Proc of SPIE, Conf 3640*. Jan;1999
9. Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W. Surface reconstruction from unorganized points. *SIGGRAPH'93* 1993:19–26.
10. Jones MW, Chen M. A New Approach to the Construction of Surface from Contour Data. *Computer Graphics Forum (EUROGRAPHICS'94)* 1994;13(3):c-75–c-84.
11. Kass M, Witkin A, Terzopoulos D. Snakes: Active Contour Models. *International Journal of Computer Vision* 1988;1(4):321–331.
12. Lorensen WE, Cline HE. Marching Cubes: A High Resolution 3D Surface Reconstruction Algorithm. *Computer Graphics (SIGGRAPH'87)* 1987;21(4):163–169.
13. Miller JV, Breen DE, Lorensen WE, O'Bara RM, Wozny MJ. Geometrically Deformed Models: A Method for Extracting Closed Geometric Models from Volume Data. *SIGGRAPH'91* 1991:217–226.
14. Sapidis NS, Besl PJ. Direct Construction of Polynomial Surfaces from Dense Range Images through Region Growing. *ACT Transactions on Graphics* 1995;14(2):171–200.
15. Sutcliffe, DC. Contouring over Rectangular And Skewed Rectangular Grids – An Introduction. *SIGGRAPH'88, Course 18: Algorithms for Data Representation Graphics; Atlanta, Georgia. August, 1988; p. 69-92.*
16. Turk G, Levoy M. Zippered Polygon Meshes from Range Images. *SIGGRAPH'94* 1994:311–318.
17. Verroust, A.; Lazarus, F. Extracting Skeletal Curves from 3D Scattered Data. *Shape Modeling International'99; Aiza Wakamatsu. March 1–4, 1999;*
18. Xu, C.; Prince, J. Snakes, Shapes, and Gradient Vector Flow. *IEEE Transactions on Image Processing; March 1998. p. 359-369.*
19. Zhou Y, Toga AW. Extracting and Representing the Cortical Sulci. *IEEE Computer Graphics and Applications May;1999* 19(3):49–55.
20. Zhou Y, Toga AW. Efficient Skeletonization of Volumetric Objects. *IEEE Transactions on Visualization and Computer Graphics* 1999;5(3):196–209.
21. Zhou, Y.; Toga, AW. Voxel-Coding for Tiling Complex Volumetric Objects. *Pacific Graphics'2000; Hong Kong. Oct. 2000;*
22. Zhou, Y.; Toga, AW. Voxel-Coding for Surface Reconstruction from Contours. 1999. Submitted, the abstract is available at <http://www.loni.ucla.edu/yzhou/contour.html>

(1) a region R (2) exterior boundary E of R (3) interior boundary of E

(4) boundary tracing

Figure 1.

A region and curves passing through its exterior boundary.

Figure 2.

Each two layers of voxel propagation make the visited region extend one additional pixel along the grid diagonal direction in 2D.

Figure 3.
Conversion from sample points to volumetric objects.

Figure 4.

Each row lists a series of operations on the initial slices (a1) and (b1). (b1) is the result after a clipping operation is implemented on (a1). First, the exterior boundaries (in red in (a2) and (b2)) are extracted. The sample points are represented in white grid, the inside regions in blue. Then initial contours (in yellow in (a3) and (b3)) are extracted from the exterior boundaries. Finally the final contours are generated and colored starting from its lowest point counterclockwise in blue, green, red and bright blue, respectively in (a4) and (b4). (a4) is the result after only contour point mapping while (b4) is the result after mapping, point insertion. The sample points in (a4) and (b4) are represented as points rather than grids as in other figures.

Figure 5.
18734 points are turned into 254 contours in 83 seconds.

Figure 6.
Zoom-on of the joints of fingernails and palm in Fig. 5.

Figure 7.
12772 points are turned into 221 contours in 67 seconds.

Figure 8.

A combination of points and a group of corresponding contours in between slices 51 through 95 along z-axis direction. Sample points are represented in white, all the contours are closed curves, ordered counterclockwise, and colored as in Fig. 4(b4).

Figure 9.
100250 points are turned into 536 contours in 134 seconds.

Table 1

Test data and related parameters

	points	Volume	voxel size	Layers	time (secs)
fist (Fig. 5)	18,734	197*193*257	0.5	5	83
head (Fig. 7)	12,772	142*190*224	1.29	6	67
dragon(Fig. 9)	100,250	414*293*187	0.0005	3	134