Multiresolution Reconstruction of Pipe-Shaped Objects from Contours

Kyungha Min[1] and In-Kwon Lee[2,⋆]

[1] Center for Computer Graphics and Virtual Reality,
Ewha Womans Univ., Seoul, Korea
minkh@cs.rutgers.edu
[2] Dept. of Computer Science,
Yonsei Univ., Seoul, Korea
iklee@yonsei.ac.kr

Abstract. We reconstruct pipe-shaped objects from a set of contours, each of which is extracted from an image representing a slice sampled from 3D volume data. The contours are formed by connecting the intersection points between rays cast from a central pixel of an image slice and the boundary of the shape. The edges on the contours are classified into several types, which are exploited in triangulating the contours, thus eliminating most of the floating-point computation from the tiling. Initially, contours of lowest resolution are extracted to reconstruct a lowest-resolution object, which is refined by adding points to the contours.

1 Introduction

Reconstruction of a pipe-shaped object from contours is an important problem in many fields, such as medical imaging, computer-aided design, and reverse engineering. The reconstruction problem is composed of three subproblems [11]: the correspondence problem, the branch problem, and the tiling problem. In this paper, we concentrate on the tiling problem.

We propose a multiresolution approach that yields a polygonal surface from a set of contours sampled from 3D volume data. The proposed algorithm is outlined in Fig. **??**. Contours of the lowest resolution are extracted on image slices sampled from the 3D volume data, and an initial pipe-shaped object is reconstructed by tiling the contours. This object is brought to a higher resolution through refinement of the contours.

Our first key idea is a search algorithm for corresponding pairs of points or edges on the successive contours that does not require floating-point computations, except for a few degenerate cases. Note that the conventional tiling algorithms search the corresponding pairs by means of geometric tests that would

⋆ This work was supported (in part) by the Ministry of Information & Communications, Korea, under ther Information Technology Research Center (ITRC) Support Program at Ewha Womans Univ. and Yonsei Univ.

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3514, pp. 916–924, 2005.

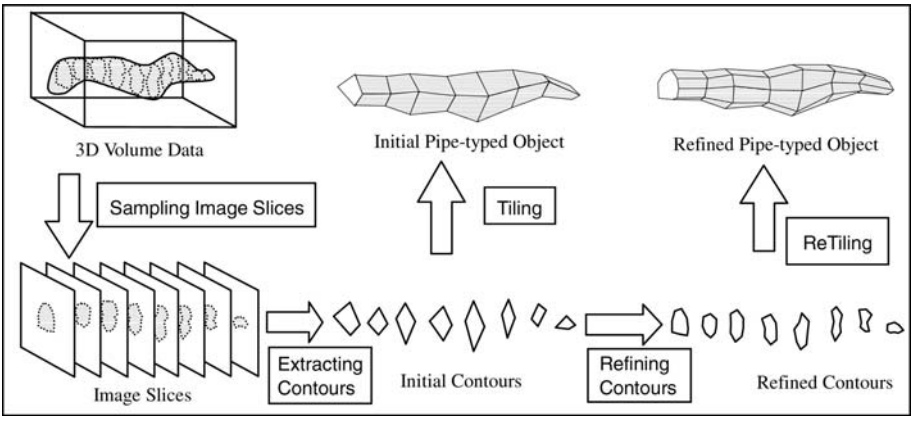

Fig. 1. Overview of the proposed algorithm

involve floating-point operations. The second key idea of this paper is the reconstruction of objects by a stepwise refinement. An object is refined by refinement of its constituent contours, which is achieved by casting additional rays, which increases the number of points on the contours in a stepwise fashion.

In Section 2, we review related works on the reconstruction of pipe-shaped objects from contours. We provide a scheme for extracting contours in Section 3, and a tiling algorithm in Section 4. In Section 5, we provide details of our implementation and results. Finally, in Section 6, we draw the conclusions and suggest future work.

2 Related Works

Keppel [9] originally proposed a scheme for reconstructing a surface from 2D contours. He built a toroidal graph between points on the contours and proposed a search using the graph that would generate polygons from the points. Many researchers improved Keppel's work by a divide and conquer approach [7], a greedy search algorithm [3], Delaunay triangulation [2], decomposing non-convex contours into convex sub-contours [5], tiling using ellipses [11], approximating the contours using discrete field functions [8], a constraint-based approach [1], a generalized Voronoi diagram [13], solving PDE [4], and a morphological dilation operators [10]. Some researchers have developed multiresolutional approaches for tiling surface from contours using wavelets [12], medial axis and simplification [15], and a combined scheme of wavelet and dynamic programming [6].

3 Extraction of Contours

Extracting Contours. Our reconstruction process starts with the extraction of a contour from an image slice. Fig. 2 illustrates the three steps of the ex-

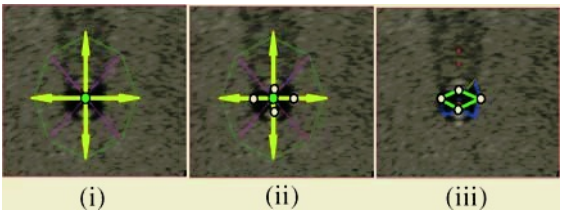

Fig. 2. Extraction of a contour from an image slice

traction: (i) Casting rays from a central pixel (left column), (ii) Computing intersection points between the rays and the boundaries of the object (middle column), and (iii) Connecting the intersection points to build a contour (right column). Depending on the relationship between the ray and the boundary of the object, we can classify an intersection point into two types: either IN-OUT (ray passing out of the object) or OUT-IN (ray coming in the object). In order to connect the intersection points, we apply the well-known chain code algorithm [14] to traverse the boundary pixels in counterclockwise order. The algorithm is slightly modified to traverse the border edges of the boundary pixels.

Classifying Edges. We classify edges of the contours into the following categories: Forward (F), Backward (B), Down (D), and Up (U). A D edge is further classified into Down and In (D_I) and Down and Out (D_O), while a U edge is into Up and In (U_I), and Up and Out (U_O). The classification is based on the following three factors:

i. **The relation between the ray and the endpoints.** An F edge is an edge with a start point is determined by the x-th ray and end point by the $(x+1)$-th ray, A B edge has its start point determined by the $(x+1)$-th ray and its endpoint by the x-th ray. Note that the start and end points of the U and D edges lie on the same ray.
ii. **The direction of the edge and of the ray.** We distinguish a U edge from a D edge based on the direction of the edges. The direction of U edge is identical to the ray, while the direction of D edge is reverse to the ray.

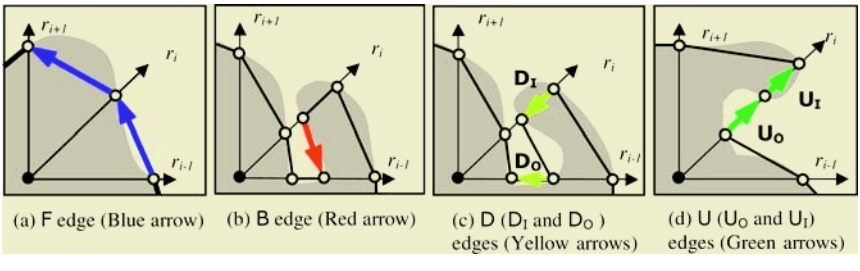

Fig. 3. Types of edges

iii. The types of the endpoints of an edge. $\mathsf{D_I}$ and $\mathsf{U_I}$ edges connect IN-OUT and OUT-IN endpoints, while $\mathsf{D_O}$ and $\mathsf{U_O}$ edges connect OUT-IN and IN-OUT endpoints.

Fig. 3 illustrates the types of the edges on a contour.

4 Tiling Algorithm

4.1 Building a Contour Tree

Description of a Contour Tree. A contour tree represents a contour based on the types of the edges in that contour. It is composed of three types of nodes.

Root node. Initially, the root node of a contour tree has n_0 child nodes, where n_0 denotes the number of initial rays.
Level-1 node. A level-1 node is a child of a root node. The x-th level-1 node contains an F edge determined by the x-th ray and the $(x+1)$-th ray.
Edge node. An edge node is a node that contains one of the edge types.

Note that the leaf nodes and level-1 nodes contain the information about the edges of the contour.

Building Rules. A contour is unambiguous if n rays create exactly n edges of type F. If n rays determine more F edges, then the contour is said to be ambiguous. The rules for building a contour tree for an unambiguous contour are listed as follows:

[Building rule for level-1 nodes]

[1] An x-th level-1 node stores the F edge determined by the x-th ray and the $(x+1)$-th ray.
[2] A D edge node is attached to the right child of a level-1 node and a U edge node to the left child.

[Building rule for leaf nodes]

[3] For all non-F edges, we build edge nodes that contain the geometry of the edge. These edge nodes are the leaf nodes of a contour tree.

[Building rule for edge nodes]

[4] If a $\mathsf{D_I}$ ($\mathsf{D_O}$) edge and a $\mathsf{D_O}$ ($\mathsf{D_I}$) edge are incident, we build a new D ($\mathsf{D_R}$) internal node and assign the nodes as the child nodes of the new node.
[5] If a $\mathsf{U_O}$ ($\mathsf{U_I}$) edge and a $\mathsf{U_I}$ ($\mathsf{U_O}$) edge are incident, we build a new U ($\mathsf{U_R}$) internal node and assign the nodes as the child nodes of the new node.
[6] If two D (U) internal nodes are incident, we build a new D (U) internal node and assign both of the D (U)-typed nodes as the child node of the new node.
[7] If a D internal node and a U internal node are incident, we build a new D (or U) internal node and assign both of the nodes as child nodes of the new internal node.

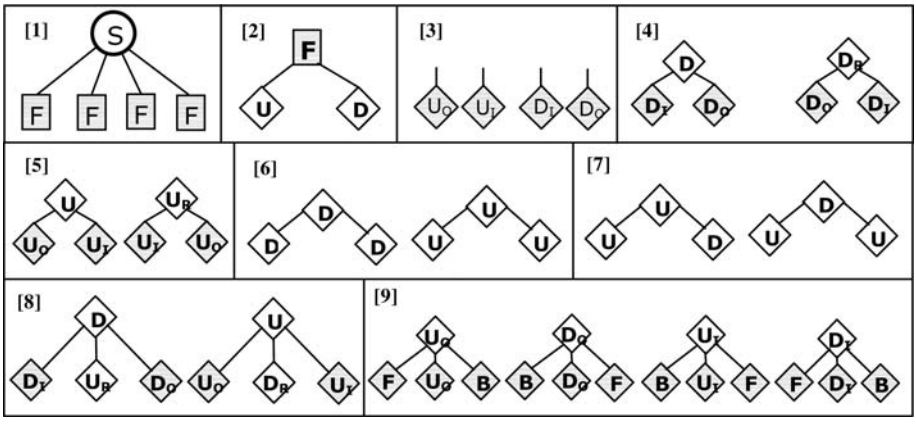

Fig. 4. Examples of the building rules

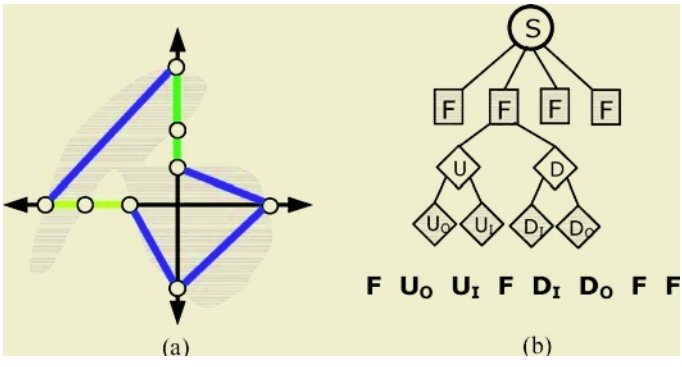

Fig. 5. An example of an unambiguous contour tree: (a) illustrates an example contour and its edges, and (b) illustrates the corresponding contour tree. The string in (b) denotes the edges represented its types in the counterclockwise order

[8] If D_I (U_O) and U_R (D_R) and D_O (U_I) internal nodes are incident, then we build a new D (U) internal node and assign three internal nodes are the child nodes of the new node.

[9] For D_I (D_O, U_I, or U_O) edge, which is encapsulated by F edge and B edge, we build a new internal node of the identical type and assign the three internal nodes that contain the three edges as child nodes.

Fig. 4 illustrates the examples of the building rules [1] - [9]. Fig. 5 illustrates an example of a contour tree for an unambiguous contour.

An ambiguous contour can correspond to more than two contour trees, each of which is defined from a contour. In this case, we choose a contour tree based on the tree of the neighboring contour. In case that a contour is ambiguous, more than two contour trees can be derived from a contour. Among the contour trees, we select one tree based on the contour tree of a neighboring contour. The

building rule [**1**] is rewritten for the level-1 node. Other building rules for an unambiguous case can be applied for the ambiguous case.

[**1'**] There are t edges of type F determined by x-th ray and $(x+1)$-th ray, where $t > 1$. We choose one of them and store it in the x-th level-1 node. The F edge to will be stored in the level-1 node should be the one that is geometrically closest to the F edge stored in the x-th level-1 node of the corresponding contour tree.

4.2 Tiling by Traversing Contour Trees

The tiling step generates triangles between contours by searching corresponding edges and vertices on the contours. We determine corresponding pairs on the contours by traversing their contour trees simultaneously. The rules for traversal are described as follows:

1. At the root nodes
   From the root nodes, the level-1 nodes at the same position on each tree are visited simultaneously.
2. At level-1 nodes
   Child nodes of the same type are traversed simultaneously.
3. At the internal edge nodes, we have three different cases (See Fig. 6):
   Case 1. Both of the nodes have an identical set of child nodes. All the child nodes of the same type on each tree are traversed simultaneously.
   Case 2. Both of the nodes have at least one child node. The same-typed child nodes on each of the tree are traversed simultaneously. If more than two child nodes are of the same type with a child node in the corresponding tree, we choose one of them arbitrarily.
   Case 3. The nodes have no children of the same type. The child node whose type is identical to the parent node is traversed, while the edge node without a child of the same type is not traversed.
4. At the leaf edge nodes, the edges stored in the nodes become corresponding edges to each other, and the basis of tile pairs.

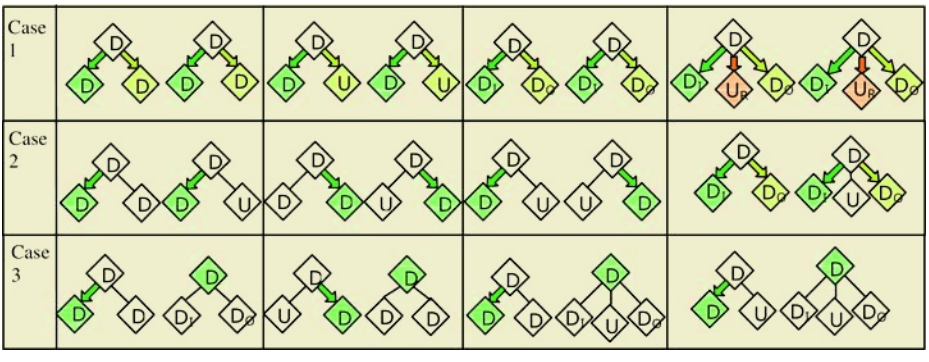

Fig. 6. Traversing rules for D-typed edge node. Note that the child nodes of the identical colors are traversed simultaneously

4.3 Refinement

We refine a contour by casting new rays between the existing rays. The new intersection points sampled on the boundary of the object are inserted into the existing contour. The contour tree follows the refinement of its contour as follows:

Refinement of a level-1 node. Suppose a new ray is cast between the x-th ray and the $(x+1)$-th ray. The x-th level-1 node is divided into two level-1 nodes, and new F edges determined by the new ray are stored in the new level-1 nodes.

Refinement of an edge node. All the existing edges interrupted by new intersection points are removed. Consequently, the nodes that store these edges are removed from the contour tree recursively. To add the new edges to the contour, we apply the building rules described in Section 3 to build a contour tree, and attach the resulting fragment as a child of a level-1 node.

5 Implementation and Results

We implemented our algorithm on an 800 MHz Pentium III CPU PC with 256 MB of RAMs. For testing, we used a phantom object and an artery at the neck. Both of the objects are 3-D volume with 256 X 256 X 512 resolutions. To capture the volume from the source object, we exploited 3-D ultrasonic scanner developed for medical purpose. Fig. 7 (a) illustrates four levels of detail for the phantom object, and Fig. 7 (b) illustrates the artery at four levels of details. For comparison, we also implemented the well-known toroidal search algorithm [2, 3, 7]. The resulting computation time, number of triangles are itemized in Table 1.

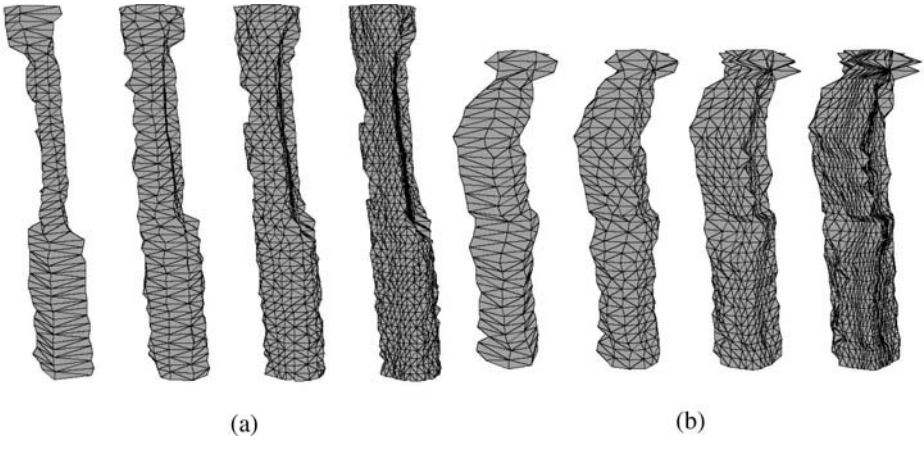

Fig. 7. Reconstruction of pipe-shaped objects in four resolutions (from left to right)

Table 1. Comparison of the proposed scheme to the conventional scheme

Resolution	Applied Algorithm	Time (sec)	No. of Triangles	Size of data (Kbytes)
1	Toroidal search	0.340	444	6.384
	Proposed	0.241	412	2.684
2	Toroidal search	0.761	1,061	15.126
	Proposed	0.550	867	6.054
3	Toroidal search	1.442	2,424	34.032
	Proposed	0.992	1,754	12.652
4	Toroidal search	2.904	4,791	136.178
	Proposed	2.062	3,392	50.354

6 Conclusion and Future Work

We have presented a new multiresolution scheme to reconstruct a pipe-shaped object from contours, which are extracted from a set of image slices by connecting the intersection points between the rays cast from a central pixel and the boundary of the shape. The edges on the contour are classified into several types, and this classification is exploited in searching for correspondence between adjacent contours. Efficiency is improved by eliminating most of the floating-point computations from the tiling process. A pipe-shaped object at the lowest resolution is refined in a stepwise way by refinement of the contours, which is achieved by casting new rays on the image slices.

We aim to develop a reconstruction scheme that addresses the branching problem based on the approach reported in this paper. We also intend to apply our scheme in areas such as medical imaging.

References

1. Bajaj, C. L., Coyle, E. J., and Lin, K. N., "Arbitrary Topology shape reconstruction from planar cross sections,", GMIP, 58(6), pp. 524-543, 1996.
2. Boissonnat, J. D., "Shape reconstruction from planar cross sections," CVGIP, 44, pp. 1-29, 1988.
3. Christiansen, H. N. and Sederberg, T. W., "Conversion of complex contour line definitions into polygonal element mosaics," SIGGRAPH 78, pp. 187-192, 1978.
4. Cong, G. and Parvin, B., "An algebraic solution to surface recovery from cross-sectional contours," GMIP, 61(4), pp. 222-243, 1999.
5. Ekoule, A. B., Peyrin, F. C., and Odet, C., L., "A triangulation algorithm from arbitrary shaped multiple planar contours," ACM ToG, 10(2), pp. 182-199, 1991.
6. Fix, J. D. and Ladner, R. E., "Multiresolution based refinement to accelerate surface reconstruction from polygons," Computational Geometry, 13(1), pp. 49-64, 1999.
7. Fuchs, H., Kedem, Z. M., and Uselton, S. P., "Optimal surface reconstruction from planar contours," Communications of ACM, 20(10), pp. 693-702, 1977.
8. Jones, M. W. and Chen, M., "A new approach to the construction of surfaces from contour data," Computer Graphics Forum, 13(3), pp. 75-84, 1994.

9. Keppel, E., "Approximating complex surfaces by triangulation of contour lines," IBM Journal of Res. Dev. 19, pp. 2-11, 1975.
10. Marsan, A. L. and Dutta, D., "Computational techniques for automatically tiling and skinning branched objects," Computers & Graphics, 23(1), pp. 111-126, 1999.
11. Meyers, D., Skinner, S., and Sloan, K., "Surfaces from contours," ACM ToG, 11(3), pp. 228-258, 1992.
12. Meyers, D., "Multiresolution tiling," Computer Graphics Forum, 13(5), pp. 325-340, 1994.
13. Oliva, J. M., Perrin, M., and Coquillart, S., "3D reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoi diagram," Computer Graphics Forum, 15(2), pp. 397-408, 1996.
14. Pitas, I., Digital Image Processing Algorithms, Prentice Hall, 1993.
15. Schilling, A. and Klein, R., "Fast generation of multiresolution surfaces from contours," Eurographics Workshop on Visualization, pp. 35-46, 1998.