An Evaluation of Implicit Surface Tilers

Paul Ning
Stanford University

Jules Bloomenthal
Xerox PARC

Abstract

In recent years, numerous techniques have been developed for the polygonization of implicit surfaces. This article reviews the principal algorithms and provides a framework for identifying their conceptual similarities as well as their practical differences. Particular attention is devoted to the much discussed problem of topological ambiguity, with solutions analyzed according to their consistency and correctness. Included in this evaluation are implementation suggestions for various application requirements.

1 Introduction

Implicit surface polygonization is a technique useful for visualizing implicit models and volumetric data. In this review, we evaluate the principal polygonization algorithms according to topological issues, implementation complexity, and polygon count. The algorithms we discuss are not new; we hope, however, that the organization of our presentation, which emphasizes practical considerations, will provide a helpful guide to users and implementors of these techniques.

A primary goal of our discussion is to clarify the differences among the existing techniques. We emphasize topological issues because they most clearly discriminate the algorithms. Our discussion also relates topology to the practical issue of implementation complexity. And we consider polygon count because of its significance for interactive applications.

Kalvin provides another review of polygonization algorithms [Kal92]; in our presentation we focus on practical concerns and offer a different interpretation of topological issues.

1.1 Implicit Surface Tilers

An implicit surface is defined as those points $\mathbf{p}_i = (x_i, y_i, z_i)$ such that $f(\mathbf{p}_i) = 0$, for some function f. Methods that construct a polygonal approximation to this surface are known as *implicit surface tilers*.[1] Several recently developed tilers construct the approximation by partitioning space into non-overlapping polyhedral cells within which the implicit surface is represented by one or more polygons. For each cell, the function f is evaluated (or sampled) at each of the cell vertices. The objective of an implicit surface tiler is to locate intersections of the implicit surface with the cell edges and to connect these intersections into polygons or triangles.

We consider those tilers that operate on fixed size cells (see [Blo88][HW90][Nin92][MS93] regarding adaptively sized partitionings) and confine our review to those tilers that partition space into cubes, although many of our observations apply to arbitrary cell shapes. Finally, we restrict our discussion to the usual case in which (possibly non-planar) polygons are converted to triangles.

1.2 Data Sources

The source of the function f is a practical matter that distinguishes two classes of tilers, those operating on *discrete data* and those operating on *continuous data*. Discrete data are available only at cell vertices; these data are usually in the form of experimental or computed volumetric datasets (such as in medical images or discrete grid simulations), where polygonization is a useful volume visualization technique. Continuous data may be obtained exactly at arbitrary spatial positions; these data are usually associated with implicitly defined geometric models (such as algebraic surfaces), where polygonization is a useful technique to obtain a convenient object representation.

A practical consequence of this distinction relates to the accuracy of the polygonal surface. For discrete data, the behavior of the function between samples must be approximated by an interpolant; for continuous data, however, no such interpolation is required. Thus, the location of edge/surface intersections must be approximated (*e.g.*, by linear interpolation) in the discrete case but may be determined with arbitrary resolution (*e.g.*, by binary subdivision) in the continuous case.

---

[1] They are also known as *isosurface generation* algorithms when applied to varying values of the function f. The set of points $\mathbf{p}_i$ such that $f(\mathbf{p}_i) = c$ is the isosurface for the *isocontour value* (or *isovalue*) c.

A second practical consequence relates to cell identification. In general, it is desirable to minimize the number of cells processed by considering only those that contain portions of the surface. The discrete data tiler can pre-sort its cells according to their minimum and maximum function values. Should an offset (*isovalue*) be applied to the function, those cells needing triangulation are quickly identified from the sort [WvG92]. In the continuous case, function evaluations are computationally expensive; thus obtaining all cells in a given volume for pre-sorting is impractical. Instead, continuous data tilers begin with some initial cell(s) and track the surface by selectively propagating new cells; this technique is known as *numerical continuation* [AG90].

1.3 Cubical Cell Polygonization

Most tiling methods developed to date partition space into cubical (or rectilinear) cells [WMW86][LC87][Blo88] [Bak89][Wal91]. The regularity of the cubical lattice is particularly appropriate for both discrete and continuous data. For discrete data, the regular grid is compatible with most scanning hardware used to gather samples. For continuous data, the regular grid permits a simple integer indexing scheme for cells so that the tiler will not propagate cells into previously examined locations. Unfortunately, however, the cubical cell is known to be susceptible to topological *ambiguity*. Without a careful polygonization of each cell, improperly closed surfaces (surfaces with 'holes') can result.

The tiling algorithms compute an edge/surface intersection for each edge that joins oppositely signed cell vertices. Intersections within each cell face are then connected into *face contours* (line segments) that are combined sequentially to form (possibly non-planar) polygons. This *polygon tracing* step is usually followed by conversion of the polygon into triangles.

The foregoing steps make two assumptions regarding the behavior of the implicit surface, although the actual surface can, in general, have arbitrary complexity within the cell. First, exactly one intersection exists on an edge connecting oppositely signed cell vertices and no intersections exist otherwise. Second, the intersection of the surface with a face is topologically equivalent to a line segment (or segments) joining the edge intersections. These assumptions, which simplify implementation and are common to all of the methods we discuss, are reasonable for high sampling resolutions (in other words, the functional variation must be small with respect to cell size). Examples that violate these assumptions include a surface with multiple zero crossings along an edge, and a surface that produces a closed contour in the interior of a face.

1.4 Overview

In the next section we examine the fundamental elements of topological ambiguity and review several proposed methods for disambiguation in terms of *consistency* and *correctness*. We then present several criteria useful in evaluating implicit surface tilers, including complexity of implementation and triangle count of the surface. Finally, we offer suggestions for the most appropriate techniques under different sets of application requirements.

2 Topological Ambiguity

Figure 1 illustrates an ambiguity occuring on the face of a cube; given the diagonal arrangement of vertex polarities, it is unclear whether the edge/surface intersections should be connected as shown on the left or on the right.[2] A basic assumption is that one of these is topologically equivalent to the actual surface.

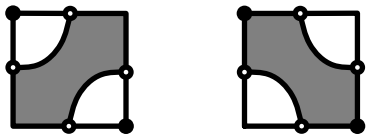


Figure 1: An ambiguous face.

In three dimensions, such an ambiguity occurs whenever a cube face includes two sets of diagonally opposed like-polarity vertices. Unwanted artifacts of the ambiguity are visible when neighboring cells make inconsistent connectivity decisions for their common face, as shown in Figure 2. As noted in [Duu88], these topological inconsistencies are manifested as holes in the surface, which can be quite objectionable in computer imagery.

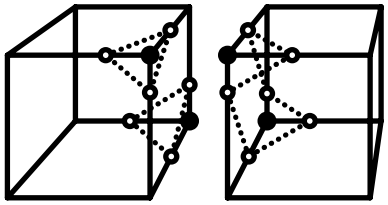


Figure 2: Ambiguous cube configuration
(inconsistent polygonization by neighboring cells).

2.1 Consistency vs. Correctness

Clearly, topological consistency is paramount because it is needed to prevent the appearance of easily detected artifacts. Beyond topological consistency, it is also desirable to achieve topological correctness, *i.e.* faithfulness to the geometry of the function. For discrete data, correct topological decisions must match the behavior of some assumed interpolant and, for continuous data, the decisions must correspond to the true functional variation. Thus, correctness implies that multiple, data-dependent topologies be allowed for each configuration of cell vertex polarities.

---

[2] In this and subsequent figures, we display positive cell vertices as solid black dots and surface vertices as open circles; negative cell vertices are not highlighted.

It may be argued that, in some instances, correctness is not a significant issue provided consistency is attained. Indeed, the difference between a correct and incorrect disambiguation decision affects only a small region on the order of the cell size (Figure 3). Provided that topological decisions are consistent, the resulting surfaces are well-behaved; only if the ambiguous region is magnified, or if it corresponds to an interesting portion of the surface, does correctness become significant.

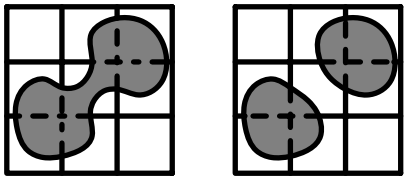


Figure 3: Two conflicting polygonizations (2-D).

In the following section, we discuss three classes of disambiguation techniques: *cell decomposition*, *preferred polarity*, and *topology inference*. These are reviewed with particular emphasis on their consistency and correctness properties.

2.2 Disambiguation Strategies

2.2.1 Cell Decomposition

Cell decomposition methods attempt to resolve ambiguity by partitioning the cube into unambiguous sub-cells. New edges are introduced and, consequently, more triangles may be created.

A tetrahedron has the property that its negative vertices may always be separated from its positive vertices by a single plane; thus an unambiguous polygonization of a tetrahedron always exists. By decomposing the cube into tetrahedral sub-cells and polygonizing the sub-cells, ambiguous cube types are resolved [Blo88][PT90] [HW90][Nie91]. As long as neighboring cubes are decomposed so that they share common tetrahedral faces at their boundaries, a consistent polygonization will result. Correctness, however, is ignored because, as discussed in Section 4, the orientation of the decomposition imposes an arbitrary choice on the direction of the surface contour.

Another decomposition method is recursive subdivision of the cube into eight smaller cubes [WvG90]. Unfortunately, the number of recursions needed to remove ambiguities may be arbitrarily large. For example, the face in Figure 4 (left), upon subdivision, can yield either an unambiguous result (middle) or an ambiguous result (right), which can perpetuate the ambiguity indefinitely. This only postpones the disambiguation decision until some maximum recursion level is reached (although beyond a certain level any decision is visually inconsequential).

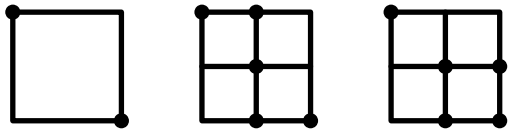


Figure 4: Recursive subdivision in two dimensions.

2.2.2 Preferred Polarity

The direction of contours in ambiguous faces may be chosen based on the polarity of the four face vertices; *i.e.*, one may elect to always separate the positives (and join the negatives) or always separate the negatives (and join the positives). This is effectively the result of algorithms in [Blo88][Bak89][WJ90].

In [Blo88][WJ90] a polygon tracing technique is employed that connects one edge intersection to the next by turning clockwise along a face's boundary, with an initial direction towards the positive end of that edge. It can be shown that this rule always separates the positive vertices in an ambiguous face. In [Bak89] positive region connectivity is defined in a stricter sense than negative region connectivity so that, once again, positive vertices in an ambiguous face are always separated.

All of these methods make the arbitrary choice of positive vertex separation. Neighboring cells will produce the same contours in their common face, maintaining topological consistency. Since the choice is arbitrary (indeed, one may select negative vertex separation everywhere instead) no attempt is made to achieve topological correctness.

2.2.3 Topology Inference

Rather than decompose the cell or select a preferred polarity, some methods attempt to infer the correct topology of an ambiguous face from the data values, and then construct the appropriate contours for the face. The three principal inference schemes are *face center resampling*, *bilinear contours*, and *gradient heuristics*. All of these methods make consistent inferences across adjoining cells and, therefore, no holes result. Correctness is more difficult to achieve and depends on the particular inference scheme.

The simplest scheme is to resample at the center of the face and join those vertices that have the same polarity as the new sample. For continuous data, this requires an extra evaluation of the function; for discrete data, this involves an interpolation (usually the facial average [WMW86][WvG90] [Wal91]). This is not, however, guaranteed to make the correct decision, as a simple example demonstrates (Figure 5). Although the value at the face center indicates that the positive vertices should be joined, they are, in fact, topologically separated.

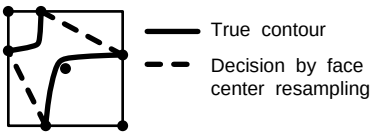


Figure 5. Face center resampling (example of failure)

The bilinear contours method [Nin91][Nat91][NH91] assumes a bilinear interpolant in each face of the cube. The face contours are hyperbolic and their directions are easily obtained from the face samples. Since the interpolation employs only the four face vertices, consistency is insured across neighboring cells. The decision is correct if bilinear interpolation is an accurate estimate of the actual function's behavior between samples. For continuous data, correctness is highly data-dependent. For discrete data, where an interpolant must be assumed in any case, the bilinear method is arguably a good choice, and agrees within the face with the trilinear kernel commonly employed for volumetric reconstruction (the trilinear kernel also is used in [Nat91] to examine the surface internal to the cube).

Finally, gradient heuristics have also been developed to infer topology [WvG90]. This technique incorporate vertex gradients as well as vertex samples in order to fit a quadratic estimate of the function across the ambiguous face. This general quadratic may produce hyperbolic contours as in the bilinear approach, but tries to match the gradient data as well. Consistency is again guaranteed because the same disambiguation decision is made by adjoining cells. Correctness is usually achieved if the underlying function is quadratic, but is not assured in general.

2.3 Implementation Complexity

There are numerous aspects to implementation complexity. We do not consider output format (in terms of polygons only or points/polygons) or data storage (in terms of hashing versus directly referenced) because these issues are independent of the particular tiling technique. Rather, we focus on the method of polygon generation, which is related to the previously discussed issues of consistency and correctness.

As mentioned, each edge of a cell has at most one intersection with the surface; these intersections are connected into polygons either by manual analysis of the polarity configurations [LC87] or by some algorithm as described above. For either approach, the polygon connectivity may be precomputed and stored in a *connectivity table* for fast access during data processing [LC87]. For the cube, the table consists of polygon specifications for each of 256 cube types (a *cube type* is defined by the polarities of the cube's eight corners).

As noted in [Bak89], the connectivity table in [LC87] is susceptible to topological inconsistency due to the symmetry operations used in its construction. Using preferred polarity [Blo88][Bak89][WJ90], however, a table can be constructed that guarantees consistency.

Topology inference methods [WMW86][WvG90][NH91] [Nat91][Nin91][Wal91] attempt to achieve correctness in addition to consistency; to accommodate the data-dependent topological decisions, however, the connectivity table must allow multiple entries per cube type.

In summary, consistency may be achieved with a *single-entry* cubical table (a unique topology for each cube type)[3], whereas correctness requires a *multi-entry* cubical table (multiple topologies for each cube type). The higher complexity of multi-entry tables suggests that correctness should be a goal only when this level of accuracy is required by the application.

3 Evaluation of Tilers

We have reviewed the primary cubical cell polygonization algorithms and classified them according to disambiguation strategy. In this section, we discuss several important practical criteria for comparing the techniques and we offer suggestions for different sets of application requirements. In subsequent sections, we provide more detail on the selected schemes.

3.1 Evaluation Criteria

As minimum requirements for recommended techniques we insist on *topological consistency*, *efficient run-time execution*, and *automatic vertex connectivity*. Topological consistency insures that contours in ambiguous faces are agreed upon by adjoining cubes, thus avoiding holes. For good run-time speed, we restrict our attention to algorithms where the vertex connectivity is precomputed and stored in a table for fast access (efficiency is less of a consideration for continuous functions whose evaluation time can easily dominate the polygonization process); a table implementation is compatible with all of the techniques discussed. We include automatic connectivity design (*i.e.* algorithmic construction of the connectivity table) as a desirable feature because manual case entry may be tedious and prone to error.

With these minimum requirements as a basis, we consider three additional issues: *implementation complexity*, *triangle count*, and *degree of disambiguation*. Implementation complexity is gauged by the size of the connectivity table. Triangle count becomes an important issue for interactive applications. Degree of disambiguation indicates whether topological correctness is sought in addition to topological consistency.

Other than topological considerations, we do not undertake a comprehensive comparison of the *visual quality* of surfaces that result from the different tiling methods. Rather, we focus on those objective issues that most clearly

---

[3] Tetrahedral decomposition requires a single-entry table describing the 16 tetrahedral types.

discriminate existing algorithms. Visual quality is difficult to define and measure, and a useful comparison requires careful consideration of many factors (such as renderer implementation, camera position, shading model, and intended application) that are independent of the tiler.

3.2 Overview of Techniques

For the simplest implementation of topologically consistent surfaces, we find *tetrahedral decomposition* to be the most attractive. In applications where triangle count is an important issue, we recommend a *single-entry cubical table* method that also satisfies consistency but produces considerably fewer triangles. Finally, if consistency is insufficient and correctness is also desired, the solution is an automatically generated *multi-entry cubical table*.

A comparison of the triangle counts for these three methods is shown in Figures 6 and 7; details of the methods are given in later sections. Also included are statistics for the widely used 'marching cubes' algorithm [LC87], which generates the fewest triangles but does not produce topologically consistent surfaces.

Figure 6 displays the number of triangles generated for each of the 256 cube types. In the lower portion of the figure, the relative frequencies of cube types for two real datasets are shown. Figure 7 displays the average number of triangles generated when the distribution of cube types is uniform, as well as for the two real datasets. Note that the distribution of cube types in 'Air jet' and 'Hand' is far from uniform; the types are concentrated in those indices that produce lower numbers of triangles. This skews the average number of triangles towards lower numbers as compared to the uniform distribution. Rendered surfaces from the example datasets are shown in Figures 8 and 9.

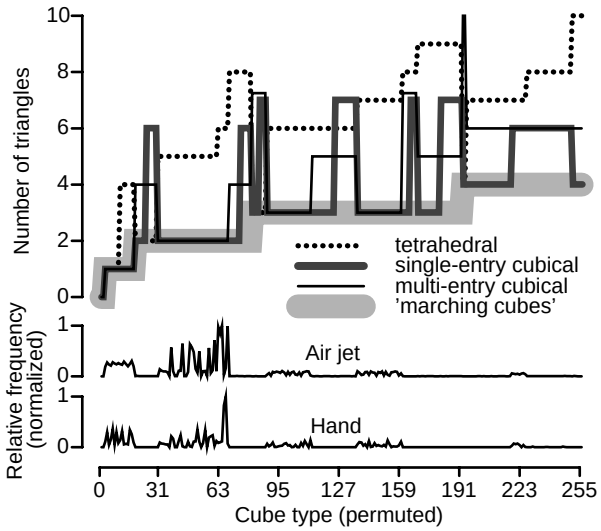


Figure 6. Triangle count and relative frequency vs. cube type
(the 8-bit indices have been permuted so that higher triangle counts tend to appear at larger indices; cells that do not contain the surface are not considered)

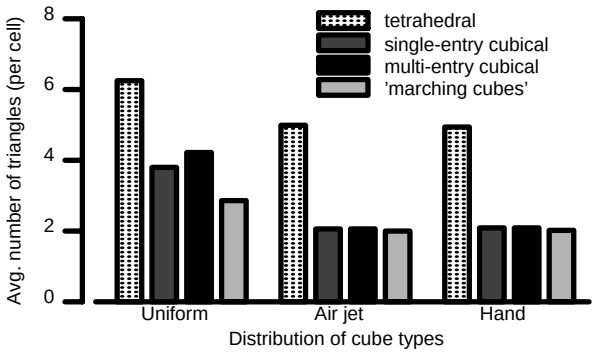


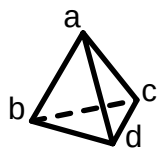
Figure 7. Avg. triangle count vs. distribution of cube types
(averages taken over cells that contain the surface)

Figure 8: Air jet.

Figure 9: Hand.

4 Tetrahedral Decomposition

A disambiguation strategy that enjoys the advantages of cubical partitioning and maintains topological consistency is the decomposition of the cube into tetrahedra. Because a tetrahedron's negative vertices can be separated from its positive vertices by a single plane, no ambiguities are possible when the tetrahedron is polygonized.[4] For any tetrahedron, the polygon(s) may be generated at run-time by algorithm [Blo88] or determined from a precomputed table of 16 tetrahedral types, which are given in Figure 10. None of the cases is ambiguous.



positive vertices	surface vertex order (indexed by edge)
case {}	{}
case {a}	{ab, ad, ac}
case {b}	{ab, bc, bd}
case {c}	{ac, cd, bc}
case {d}	{bd, cd, ad}
case {ab}	{ad, ac, bc, bd}
case {ac}	{ab, ad, cd, bc}
case {ad}	{ab, bd, cd, ac}
case {bc}	{ab, ac, cd, bd}
case {bd}	{ad, ab, bc, cd}
case {cd}	{ad, bd, bc, ac}
case {abc}	{bd, ad, cd}
case {acd}	{ab, bd, bc}
case {abd}	{cd, ac, bc}
case {bcd}	{ab, ac, ad}
case {abcd}	{}

Figure 10: Tetrahedral polygonization.

Without requiring additional cell vertices, the cube may be decomposed into five [PT90][HW90][Nie91] or six [Nie91] tetrahedra, as shown in Figure 11. These decompositions introduce diagonals on the cube faces, thus determining the resulting face contours. Consider the two faces in Figure 12; although their polarity configurations are the same, the orientation of the diagonal affects the connectivity of the surface vertices. Because this orientation is arbitrarily determined by the decomposition, topological correctness is not provided.

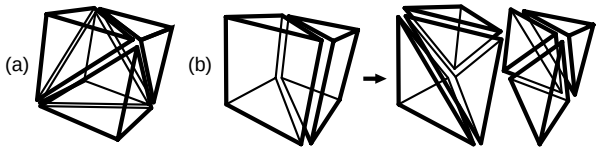


Figure 11: Decomposition into (a) five and (b) six tetrahedra.

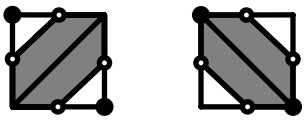


Figure 12: Effect of diagonal orientation on face contours.

In order to maintain topological consistency, the orientation of the five-tetrahedral decomposition must alternate between face-adjacent cubes (Figure 13). This insures that the diagonal introduced on a cube face agrees with that of its neighbor.

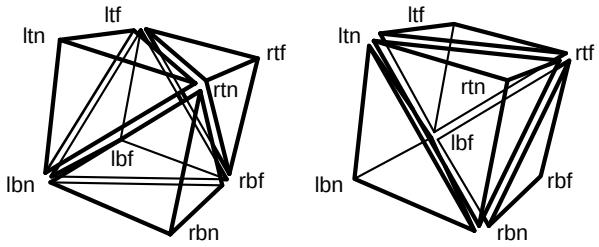


Figure 13: Alternate orientations of five tetrahedra.
(*l*, *r*, *b*, *t*, *n*, and *f* indicate left, right, bottom, top. near, and far, respectively)

The introduction of diagonal edges may produce a higher resolution surface by increasing the number of facets in the surface approximation (see Figure 14). As plotted in Figures 6 and 7, tetrahedral decomposition yields over twice as many triangles as cubical techniques (the tetrahedral data for these figures are based upon the five-decomposition).

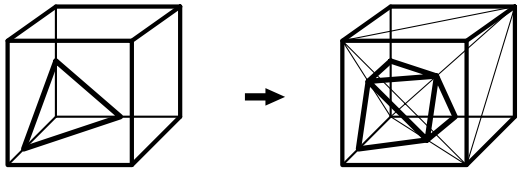


Figure 14: Additional facets produced by five-tetrahedral decomposition.

5 Single-Entry Cubical Table

Tetrahedral decomposition is simple to implement, but for applications in which reduced triangle count is more important than increased surface resolution, cubical table methods are recommended. In this section we discuss a topologically consistent, single-entry cubical table based on preferred polarity disambiguation [Blo88][Bak89][WJ90].

The idea is to impose arbitrary and consistent face decisions while still polygonizing based on the original cubical intersections. Unlike tetrahedral decomposition, new vertices along the face diagonals are not introduced. As discussed previously, a simple way to do this is to always separate the positive vertices. Thus, for each of the 256 cube types a unique polygonization may be precomputed and stored as a single entry.[5]

---

[4] For this reason, the tetrahedron is commonly used as the partitioning cell in simplicial continuation methods [AG90].

[5] An interesting observation is that a single-entry *cubical* table can also be derived from the topological decisions imposed by the *tetrahedral* decomposition. For the five-tetrahedral decomposition, two such tables would be necessary, however, to accommodate the alternating orientations of the tetrahedra (Figure 13).

The construction of the connectivity table is algorithmic but, for faster operation, actual triangle generation employs only lookup. Face contours are joined as in [WMW86], forming *n*-sided polygons. These are, in general, non-planar and must be triangulated for display. From the Euler relation, one can show that an *n*-sided polygon may be decomposed into *n*-2 triangles. Naive application of such a triangulation, however, may result in a new edge in a face of the cube, thus violating the contour decision already made for that face (see Figure 15a). In these instances, the introduction of a centroid allows a valid triangulation into *n* triangles (see Figure 15b). The centroid is guaranteed to lie in the interior of the cube so no invalid edges are created. Note that the introduction of an additional vertex creates two more triangles but should still produce fewer triangles than tetrahedral decomposition, which adds as many as seven new points.

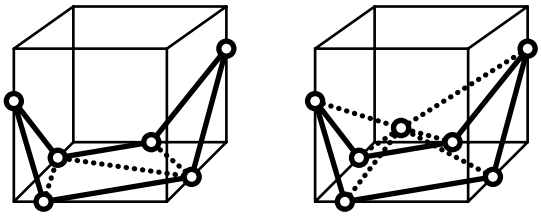


Figure 15:
(a) Triangulation that violates face contour decision.
(b) Use of centroid for valid triangulation.

The cost of two additional triangles is more significant for small *n*, so it is desirable to avoid using a centroid except for large *n*. As is shown in the Appendix, for $n=4$ and $n=5$ one can always safely decompose the *n*-sided polygon into *n*-2 triangles with a fixed rule and, for $n \geq 6$ a centroid is required; this is the strategy that we suggest for the single-entry cubical table. It is interesting to compare this algorithmic method to the manual case analysis of 'marching cubes,' which always converts *n*-sided polygons to *n*-2 triangles. From Figure 7 we see that, if the distribution of cube types is uniform, the automated centroid method creates 33% more triangles than manual case analysis; for the example datasets, however, the two methods produce nearly the same number of triangles. For all distributions, the single-entry cubical table produces considerably fewer triangles than tetrahedral decomposition.

The technique of tracing polygons from face contours has been used by numerous authors [WMW86][NH91][Wal91]. There are slight differences, however, in their triangulation techniques. Wyvill [WMW86] employs centroids for $n \geq 4$. Nielson [NH91] manually designs triangulations with *n*-2 triangles, where possible, and *n* triangles otherwise; instead of introducing a centroid, a linear interpolation between intersections of hyperbolic asymptotes is used. Wallin [Wal91] implements a hybrid technique that uses existing vertices until an inconsistency requires the computation of a centroid for the remaining vertices.

6 Multi-Entry Cubical Table

If topological correctness is required, data-dependent decisions must replace arbitrary topological decisions. Thus, the connectivity table must allow each ambiguous face to be disambiguated with either of the two face contour orientations, with the decision made at run-time. As shown in Figure 16, all six faces of the cube may be simultaneously ambiguous. Therefore, as many as 64 different connectivity topologies must be included for each arrangement of cube polarities. This may be implemented as a table with multiple entries per cube type [WvG90][NH91].

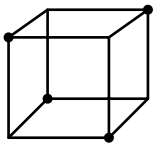


Figure 16: Cube with six ambiguous faces.

The construction of the table is similar to that of the single-entry table; polygons are traced and triangulated as before, but repeated for each combination of face contour decisions. During processing of each cube, the table is accessed according to the cube's vertex polarities; for ambiguous cube types, topological decisions from the ambiguous faces select which of the multiple entries apply. Face disambiguation methods that are arbitrary (tetrahedral decomposition, preferred polarity) are not appropriate since they do not consider correctness. Instead, one of the topology inference schemes (face center resampling, bilinear contours, gradient heuristics) should be used.

The multi-entry nature of the table implies that more than one triangle count may be associated with each cube type. In Figures 6 and 7, we assume that all topologies for any cube type are equally likely and use their average triangle count for the 8-bit index. It is evident that the multi-entry table generates slightly higher triangle counts than single-entry or 'marching cubes' when the distribution of cube types is uniform, but yields nearly an identical number for the example datasets.

7 Summary and Conclusions

We have reviewed the principal implicit surface polygonization methods and evaluated their merits according to topological disambiguation, implementation complexity, and triangle count (summarized in Figure 17). We have devoted special attention to the ambiguity problem and analyzed proposed solutions in the context of consistency and correctness.

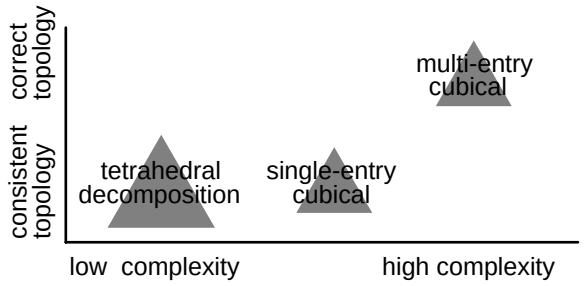


Figure 17: Evaluation summary
(size of shaded triangles indicates relative triangle count)

We have argued that consistency suffices for many purposes, and is achievable with single-entry tetrahedral or cubical tables. For those situations requiring correct disambiguation, run-time computation of topological decisions is needed to index a multi-entry table. For discrete datasets, correctness is only as accurate as the assumed interpolant and, for continuous datasets, correctness is not generally guaranteed.

For each of these techniques, polygon connectivity can be algorithmically generated and stored in a table for run-time access; for the cubical table methods, manual case analysis is avoided with only a slight increase in triangle count.

Appendix

Naive triangulations of polygon traces may create line segments in cube faces that are *invalid* face contours, *i.e.*, that do not agree with the topological decision made for that face (see Figure 15a). In this appendix we demonstrate that a valid, fixed triangulation of an n-sided polygon is possible for n=4 and n=5, but not for n=6. This motivates the use of original vertices only for $n < 6$ (yielding n-2 triangles) and the introduction of a centroid for $n \geq 6$ (yielding n triangles).

Polygons are constructed by tracing contour segments along the faces of the cube. There are only three basic face contour topologies (Figure A-1), and the tracing algorithm never traces consecutively two segments from the same face (a trace may cross the same face later, however).

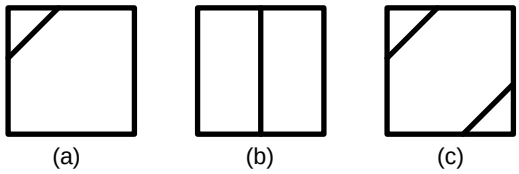


Figure A-1: Face contour topologies

We claim that any 4- or 5-sided polygon may be 'safely' triangulated as shown in Figure A-2 and argue by contradiction. Suppose that such a triangulation is invalid. This means that one of the internal segments, say x, lies completely in a face but is not a valid face contour. Then, since a and b are face contours and also lie in a face, a-b-x defines the cyclical intersection of three different faces, and therefore must be positioned in a corner of the cube (Figure A-3). But if x is not a valid face contour, the face that contains x has two surface intersections that are not joined, so its topology must be that of Figure A-1c. The resulting 4-sided partial polygon requires at least two more segments to close it, implying that $n \geq 6$ and contradicting the assumption that n=4 or 5.

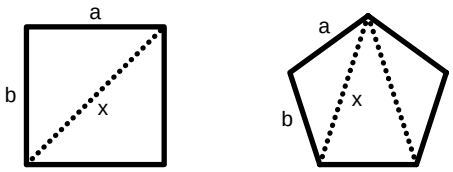


Figure A-2: Triangulation of 4- and 5-sided polygons

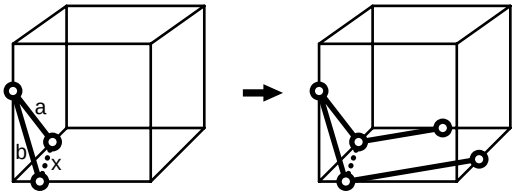


Figure A-3: Implication of 4- and 5-sided violation

To see why a fixed triangulation for n=6 is not always valid, consider the three candidate triangulations in Figure A-4. For each, an example is given that shows how an invalid face contour may be created. Note that the hexagon may be triangulated properly by candidates b or c if a variable assignment of vertices is allowed (Figure A-5). We forego this option, however, in favor of implementation simplicity by employing centroids for all 6-sided polygons (and $n > 6$).

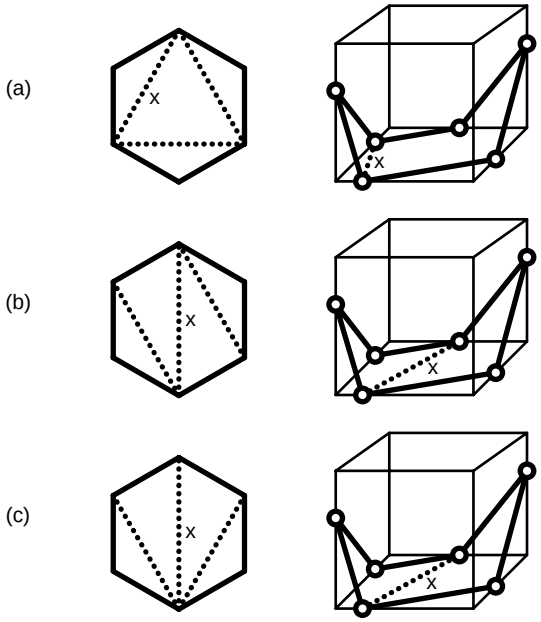


Figure A-4: Candidate triangulations of 6-sided polygon

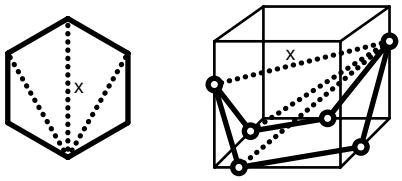


Figure A-5: A valid triangulation of 6-sided polygon

Acknowledgments

Paul Ning gratefully acknowledges funding by NASA under grant NCA 2-579, including support from the NASA Ames Numerical Aerodynamics Simulation Program and the NASA Ames Fluid Dynamics Division, and by NSF under grant ECS 8815815.

We thank Ram Subbarao and Brian Cantwell for providing the air jet data.

References

[AG90] Allgower and Georg, ‘‘Numerical Continuation Methods, an Introduction,’’ Springer-Verlag, 1990.

[Bak89] H.H. Baker, ‘‘Building Surfaces of Evolution: The Weaving Wall,’’ International Journal of Computer Vision, vol. 3, pp. 51-71, 1989.

[Blo88] J. Bloomenthal, ‘‘Polygonization of Implicit Surfaces,’’ Computer Aided Geometric Design, vol. 5 (1988), pp. 341-355.

[Duu88] M.J. Duurst, ‘‘Additional Reference to Marching Cubes,’’ Computer Graphics, vol. 22, no. 2, pp. 72-73, April 1988.

[HW90] M. Hall and J. Warren, ‘‘Adaptive Polygonalization of Implicitly Defined Surfaces,’’ IEEE Computer Graphics and Applications, 10(6), Nov. 1990, pp. 33-42.

[Kal92] A.D. Kalvin, ‘‘A Survey of Algorithms for Constructing Surfaces from 3D Volume Data,’’ IBM Research Report RC 17600 (#77606), January 1992.

[LC87] W.E. Lorensen and H.E. Cline, ‘‘Marching Cubes: A High Resolution 3-D Surface Construction Algorithm,’’ Computer Graphics, vol. 21, pp. 163-169, July 1987.

[MS93] H. Muller and M. Stark, ‘‘Adaptive Generation of Surfaces in Volume Data,’’ The Visual Computer, vol. 9 (1993), pp. 182-199.

[Nat91] B.K. Natarajan, ‘‘On Generating Topologically Correct Isosurfaces from Uniform Samples,’’ Hewlett-Packard Laboratories Technical Report HPL-91-76, June 1991.

[Nie91] G.M. Nielson, T.A. Foley, B. Hamann, and D. Lane, ‘‘Visualizing and Modeling Scattered Multivariate Data,’’ IEEE Computer Graphics and Applications, vol. 11, no. 3, pp. 47-55, May 1991.

[NH91] G.M. Nielson and B. Hamann, ‘‘The Asymptotic Decider : Resolving the Ambiguity in Marching Cubes,’’ Proceedings of IEEE Visualization 91, pp. 83-91, October 1991.

[Nin91] P. Ning and L. Hesselink, ‘‘Adaptive Isosurface Generation in a Distortion-Rate Framework,’’ Proceedings of SPIE, vol. 1459, pp. 11-21, February 1991.

[Nin92] P. Ning and L. Hesselink, ‘‘Octree Pruning for Variable-Resolution Isosurfaces,’’ Proceedings of Computer Graphics International, Tokyo, Japan, June, 1992.

[PT90] B.A. Payne and A.W. Toga, ‘‘Surface Mapping Brain Function on 3D Models,’’ IEEE Computer Graphics and Applications, 10(5), pp. 33-41, September 1990.

[Wal91] A. Wallin, ‘‘Constructing Isosurfaces from CT Data,’’ IEEE Computer Graphics and Applications, 11(6), Nov. 1991, pp. 28-33.

[WvG90] J. Wilhelms and A. Van Gelder, ‘‘Topological Considerations in Isosurface Generation,’’ Computer Graphics, vol. 24, no. 5, pp. 79-86, November 1990.

[WvG92] J. Wilhelms and A. Van Gelder, ‘‘Octrees for Faster Isosurface Generation,’’ Transactions on Graphics, vol. 11, no. 3, pp. 201-227, July 1992.

[WJ90] B. Wyvill and D. Jevans, ‘‘Table Driven Polygonization,’’ in SIGGRAPH Course Notes (Modeling and Animating with Implicit Surfaces), 1990, pp. 7.1-7.6.

[WMW86] G. Wyvill, C. McPheeters, and B. Wyvill, ‘‘Data Structures for Soft Objects,’’ The Visual Computer, vol. 2 (1986), pp. 227-234.