Visual Computer manuscript No.
(will be inserted by the editor)

Convex Contouring of Volumetric Data

Tao Ju[1], **Scott Schaefer**[1], **Joe Warren**[1]

Department of Computer Science, Rice University

The date of receipt and acceptance will be inserted by the editor

Abstract In this paper we present a fast, table-driven isosurface extraction technique on volumetric data. Unlike Marching Cubes or other cell-based algorithms, the proposed polygonization generates convex negative space inside individual cells, enabling fast collision detection on the triangulated isosurface. In our implementation, we are able to perform over 2 million point classifications per second. The algorithm is driven by an automatically constructed look-up table that stores compact decision trees by sign configurations. The decision trees determine triangulations dynamically by values at cell corners. Using the same technique, we can perform fast, crack-free multi-resolution contouring on nested grids of volumetric data. The method can also be extended to extract isosurfaces on arbitrary convex, space-filling polyhedra.

Keyword: contour, polygonization, implicit modeling

1 Introduction

Recent advances in hardware technology of 3D scanning and sensoring has brought about the generation of large-scale volumetric data such as MRI scans, CT scans and geological images. A common approach to visualize these volume datasets is to represent the data as implicit functions and construct polygonal approximations of isosurfaces, i.e. locus of points with some given function value. This process is often referred to as *contouring*. The contoured surface partitions the whole volume into *negative space* (locus of points with lower function values) and *positive space* (locus of points with higher function values). In many interactive applications, such as computer gaming and real-world simulations, navigation is confined to the negative space, therefore fast operations for checking the side of the main subject (e.g. the navigator) with respect to the contoured surface becomes critical. These operations are often referred to as collision detections.

Traditionally, contouring algorithms consider the data volume in uniform cubical cells and polygonize isosurfaces in each non-empty cell, i.e., cells that are intersected by the isosurface. By doing so, the entire negative space is decomposed into sub-spaces within individual cells. Assuming that the negative space models free space, checking whether a point lies inside the free space can be greatly accelerated if the negative sub-space is *convex* within the enclosing cell. Note that this point-classification is the fundamental operation for computing collision detection with other convex objects. For example, an edge lies in the free space if for each cell it passes through, both endpoints of the line segment within that cell lie in the cell's negative space (figure 1 left). This observation is not true if the negative space within the cell is non-convex (figure 1 right).

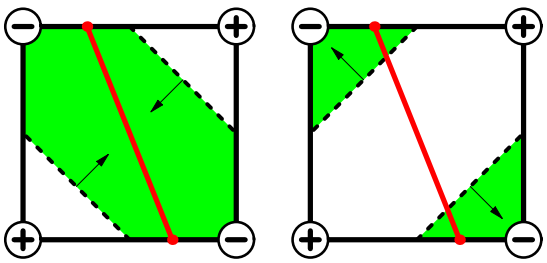


Fig. 1 Edge classification in a signed square with two different contours. The dashed lines indicate the contours, the gray areas represent negative space, and the dark gray line is the edge to be checked.

The most widely used cell-based algorithm is the Marching Cubes (MC) algorithm introduced by Lorensen and Cline [7]. MC generates triangles on isosurfaces within each cell based on a pre-computed table of positive/negative patterns (referred to as *sign configurations* hereafter)

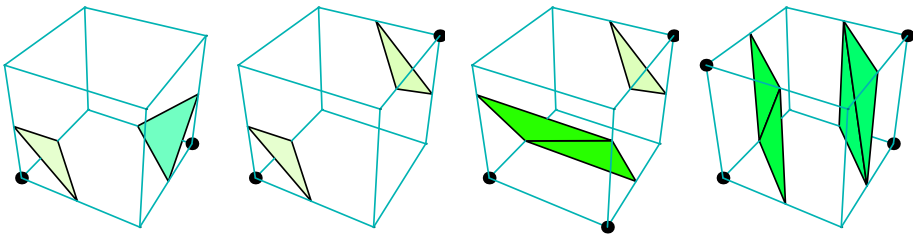


Fig. 2 Triangulations from the Marching Cubes' look-up table. Negative corners are colored black.

of cell corners. MC became popular for its fast polygonization and easy implementation due to its table-driven mechanism. However, for some sign configurations, MC does not produce polygonizations that are consistent in topology with neighboring cells, and thus result in surface discontinuities [4]. This drawback has sparkled extensive research on dis-ambiguation solutions [1], [4], [5], [10] and strategies to generate topologically correct polygonizations [9]. Although these solutions generate topologically consistent isosurfaces, the resulting negative space inside each cell is sometimes disconnected, and thus not convex.

In this paper, we propose a fast table-driven cell-based contouring method that generate topologically consistent contours, while preserving convexity of the negative space within each cell. The algorithm depends on a composite look-up table that associates each sign configuration with a compact decision tree for dynamic triangulation of isosurface patches. The reason for the use of a decision tree is that the actual triangulation depends not only on the signs, but also on the magnitude of corner values. The decision trees are pre-computed to ensure a minimal number of tests in determining each triangulation.

This technique can be extended without difficulty to multi-resolution contouring . In most multi-resolution approaches, direct application of the cell-based contouring algorithm to a grid of non-uniform cells can result in surface *cracks*, i.e. discontinuity between iso-surfaces generated from neighboring cells at different resolutions. Various multi-resolution frameworks have been proposed for contouring on non-uniform grids [3], [8], [11], [12], [14], yet they involve special crack-patching strategies. Bloomenthal [2] proposes an adaptive contouring method which requires run-time face tracing to maintain consistent contours for neighboring cells. We show that by constructing look-up tables for transition cells using the above algorithms, the same table-driven contouring method can be applied to non-uniform data to generate crack-free isosurfaces.

The remainder of this paper is organized as follows. After reviewing the table-driven Marching Cubes algrorithm, we present the convex contouring algorithm on uniform grids. Then we explain the automatic construction of look-up tables in more detail. Next we extend the proposed technique to non-uniform grids to produce crack-free contour surfaces. We conclude by discussing other possible extensions and applications.

2 Marching Cubes and Look-up Tables

Marching Cubes is a popular algorithm for extracting a polygonal contour from volumetric data sampled on a uniform 3D grid. For each cell on the grid, the edges intersected with the iso-surface are detected from signs at cell corners. The algorithm then forms triangles by connecting intersections on those edges (referred to as *edge intersections* hereafter). To speed up the process, it uses a look-up table that establishes triangulations for each sign configuration. Since there are 8 corners in a cell, the look-up table contains 256 entries, four of which are shown in figure 2 .

Using the look-up table, the Marching Cubes algorithm contours a cell in two steps:

1. Look up the triangulation in the table by the sign configuration,
2. For each triangle, compute the exact location of each vertex (edge intersection) from values at the cell corners by linear interpolation.

The Marching Cubes algorithm is fast because it uses table look-up to build polygonal contours. Unfortunately, the contoured surface generated by the original look-up table in [7] may contain holes, since the triangular contour within each cell is not always consistent with that of the neighboring cells (figure 3).

Although this problem was fixed in later work, it reveals another drawback of manually created tables. The entries in the Marching Cubes' look-up table are constructed by identifying 15 topologically distinct sign configurations and triangulating each case by hand. The lack of automation makes it susceptible to errors and difficult to adapt to other polyhedrons. As we shall see, the look-up tables used in convex contouring are constructed algorithmically based on the topology and geometry of given polyhedrons, and therefore minimizes possible errors.

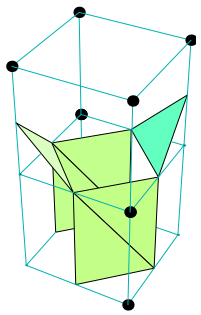


Fig. 3 Two adjacent cells contoured using the Marching Cubes algorithm that produce an inconsistent topology.

3 Convex Contouring Using Look-up Tables

The goal of convex contouring is to extract polygonal contours that enclose convex negative spaces within each cell. In this section, we will introduce the concept of convex contours and describe the proposed polygonization technique based on a pre-computed look-up table. Examples of convex contouring will be presented together with performance comparison with the Marching Cubes algorithm.

3.1 Convex contour

Assuming that the underlying implicit function is trilinear inside each cell, the *convex hull of the negative space* in a cell is the convex hull of all negative cell corners and edge intersections. The *convex contour* is the part of this convex hull that lies interior to the cell. In 2D, for example, the convex contour consists of interior line segments connecting edge intersections (i.e., dashed lines in figure 1 left). In 3D, the convex contour consists of interior triangles whose vertices are edge intersections (figure 4 left). Together with the triangles that fill the negative regions on cell faces (figure 4 right), they constitute the convex hull of the negative space inside the cell.

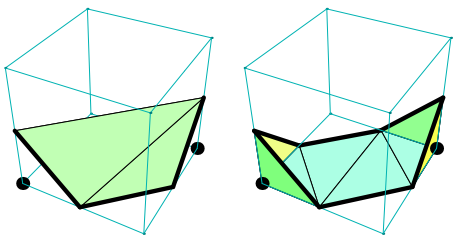


Fig. 4 Convex contour on the convex hull of the negative space.

Note that the convex contour in a cell is outlined by linear convex contours on the faces of the cell (highlighted in figure 4). Since the linear convex contour in a 2D square is uniquely determined given the signs at the 4 corners (allowing movement of the edge intersections along the edges), the polygonal convex contours from neighboring 3D cells always share the same linear contour on the common face. Hence convex contours always form topologically consistent isosurfaces.

In figure 5, we show the convex contours for the same cells from figure 2. In comparison, we observe that the convex contour always encloses a connected, convex negative space within the cell, whereas the contour from the Marching Cubes algorithm does not. Note that a convex contour is sometimes composed of multiple connected components, as shown in the rightmost cell of figure 5. Each of these connected piece of the contour is called a *patch*, which may contain multiple holes (such as the center left cell in figure 5). Each hole on the patch is surrounded by a ring of linear convex contours that can be detected by the signs at the corners. Hence a look-up table for patch boundaries can be constructed automatically for each sign configuration.

3.2 Triangulation and Decision trees

Unfortunately, although the boundary of the patches on the convex contour is unique for each sign configuration, the actual polygonization is not. In fact, different scalar values at cell corners, which determine the location of edge intersections, may modify the shape of the convex hull and result in different triangulation of the convex contour. As illustrated in figure 6, two cells that share the same sign configuration, but with different scalars at cell corners, result in different triangulated convex contours.

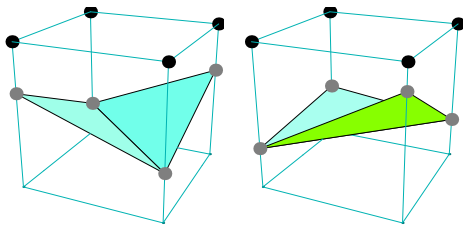


Fig. 6 Triangulation in cells with different scalar values at corners. Edge intersections are indicted by gray dots.

To determine the correct triangulation based on the location of edge intersections, one could apply a full-scale convex-hull algorithm to compute the convex hull of the negative space. However, such algorithms are too general for our purposes, since we want only the part of this convex hull that lies interior to the cell. Instead, we can take advantage of the fact that the topology of the boundary is known for each patch on the convex contour. In fact, we can construct a set S of all possible triangulations for each patch. For example, figure 6 shows the

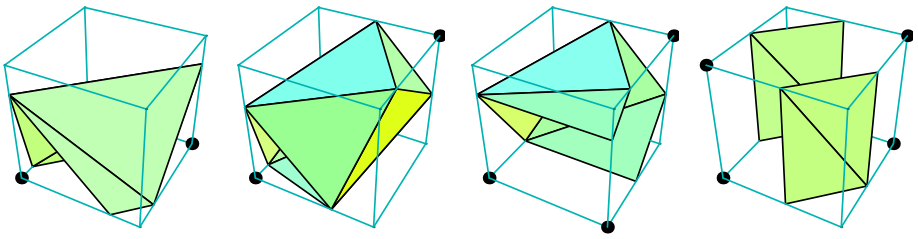


Fig. 5 Convex contours in cells with different sign configuration.

only two possible triangulations for the convex contour of that sign configuration, since the contour contains a single 4-sided patch. According to Euler's polygon division problem [6], there are $\frac{(2n-4)!}{(n-1)!(n-2)!}$ ways to triangulate a $n-sided$ patch into $n-2$ triangles. In fact, this size of S can be further reduced if we realize that the vertices of these triangles are restricted to fixed cell edges, hence some triangulations will never appear on the convex hull of negative space. We will discuss this process in detail when we describe how the look-up table is constructed.

Now we can think of the triangulation problem as the following: given the exact locations of the edge intersections, *choose* an appropriate triangulation from S so that the triangles satisfy the convex hull property (i.e., all edge intersections and negative corners lie on one side of the triangle). Hence we need a fast method to differentiate the correct triangulation from others by looking at the edge intersections. Assuming that triangles on the convex contour *face inside* the convex hull of the negative space, we can do this by the following *4-point test*: given four distinct edge intersections V_1, V_2, V_3, V_4, if V_4 lies on the front-facing side of the triangle (V_1, V_2, V_3), then the inverted triangle (V_1, V_3, V_2) does not belong to the convex contour (figure 7 left). Similarly, triangles (V_1, V_2, V_4), (V_2, V_3, V_4) and (V_3, V_1, V_4) do not satisfy the convex hull property. Otherwise, by symmetry, we claim that triangles (V_1, V_2, V_3), (V_1, V_4, V_2), (V_2, V_4, V_3) and (V_1, V_3, V_4) do not lie on the convex contour (figure 7 right). Note that if all vertices lie on the same plane, either choice can be made.

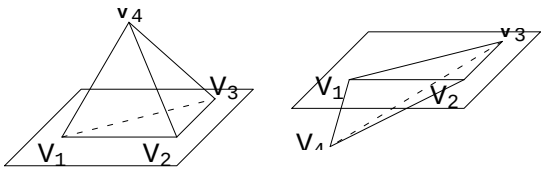


Fig. 7 Four-point test with vertices V_1, V_2, V_3, V_4.

Each 4-point test on the edge intersections rules out those from the set S of all possible triangulations that contain any of the four back-facing triangles. Since any two triangulations differ at least in the triangles shared by one of the boundary edge, the correct triangulation can be distinguished from every other triangulation in S through appropriate 4-point tests. For best performance, a decision tree can be built to distinguish each triangulation through a minimal set of tests. An example of such a decision tree is shown in figure 8 for a 5-sided patch. At each node, the remaining triangulations are shown and the 4-point test is represented by indices of the four vertices (the order is shown at the top left corner). If the fourth vertex lies on the front-facing side of the triangle formed by the first three vertices (in order), we take the left branch. Otherwise, we follow the right branch. The process stops at a leaf node, where a single triangulation is left.

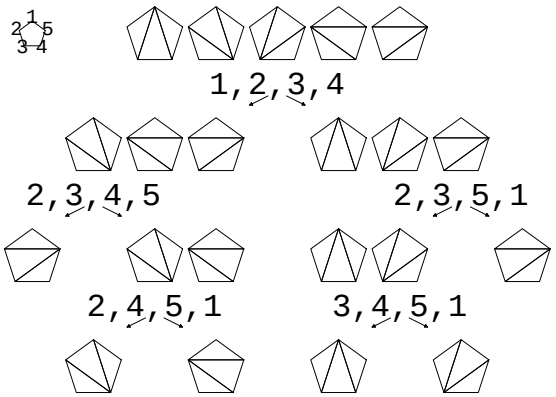


Fig. 8 A decision tree using 4-point test for 5-sided patches.

Since the construction of decision trees is based on the original set S, they can be pre-computed and optimized for every patch in each sign configuration. As we shall see, the maximum depth of all these decision trees is 5 and the average tree depth is 1.88. In other words, the correct triangulation of the convex contour in a cubic cell can be determined by performing a maximum of 5 point-face trials on the edge intersections, and on average no more than 2 trials.

3.3 The look-up table

The look-up table contains 256 entries, one for each sign configuration of a cubic cell. In each entry, the look-up table stores the decision trees computed for each patch

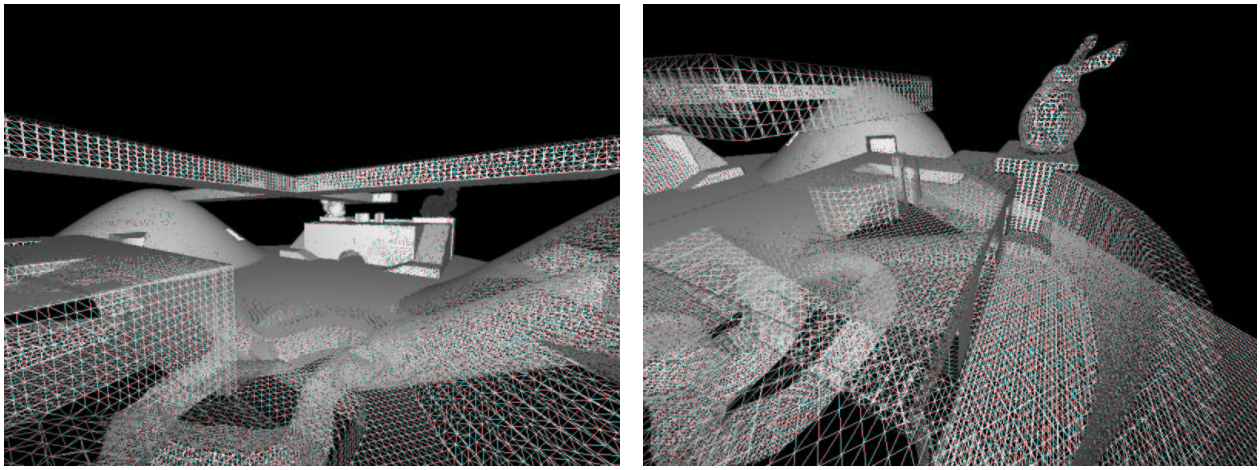


Fig. 9 Two screen shots of a real-time navigation application using a convexly contoured terrain.

by traversing the nodes in the decision trees in pre-order. Each non-leaf node contains a 4-point test, and each leaf node stores a triangulation. The edge intersections in 4-point tests and triangulations are represented by indices of the edges on which they lie. Two example entries in this table are shown in table 1, with their corresponding sign configurations and edge indexing drawn on the right.

3.4 Contouring by table look-ups

In comparison with the Marching Cubes algorithm, convex contouring extracts the polygonal contour in a cell in two steps:

1. Look up the decision trees (one for each patch) in the table by the signs at the cell corners,
2. For each decision tree, perform 4-point tests on specified edge intersections until arriving at a single triangulation.

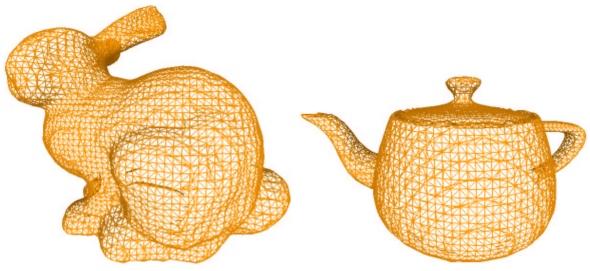


Fig. 10 Convex contouring on volumetric data.

In figure 10, two sets of volumetric data generated by scan-conversion of polygonal models are contoured using the new method. Observe that the edges on the surface are manifold and the contour is crack-less. Since the negative space inside each non-empty cell is convex, collision detection can be localized into cells and therefore become independent of the grid size. Figure 9 shows two screen shots of a real-time navigation program in which the movement of the viewer is confined within the negative space. The terrain is an iso-surface constructed using convex contouring on a 256 cubic grid. On a consumer level PC machine, we achieved over 2 million point classifications per second.

As we mentioned before, the average number of tests used to determine triangulation for each patch is less than 2. Hence we can perform convex contouring on volumetric data with speed comparable to the Marching Cubes algorithm. In table 2, the performance of convex contouring is compared with that of the Marching Cubes for contouring the terrain in figure 9 on different grid sizes. We also compared the average number of triangles generated in each cell in both methods. Notice that convex contouring generates on average only about 2% more triangles than the Marching Cubes algorithm. These extra triangles are needed to preserve the convexity of the negative space.

Grid Size	Marching Cubes	Convex Contouring
128^3	125 ms	141 ms
256^3	781 ms	907 ms
512^3	2109 ms	2437 ms
Avg. Triangles	3.181	3.257

Table 2 Comparison of total contouring time and average number of triangles per cell in convex contouring and the Marching Cubes.

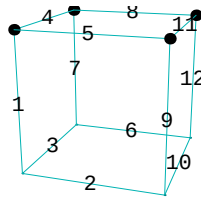
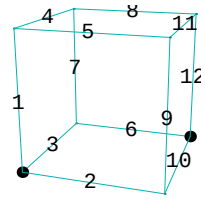
Index	Table Entry	Cell Configuration
171	{1, 9, 12, 7} → *4-point test* {{1, 9, 7}, {9, 12, 7}} → *Triangulation* {{1, 9, 12}, {1, 12, 7}}	
125	{1, 3, 6, 12} → *4-point test in parent node* {1, 12, 10, 2} → *4-point test in left child* {{1, 3, 12}, {1, 12, 2}, {3, 6, 12}, {12, 10, 2}} {{1, 3, 12}, {1, 12, 10}, {1, 10, 2}, {3, 6, 12}} {1, 12, 10, 2} → *4-point test in right child* {{1, 3, 6}, {1, 6, 12}, {1, 12, 2}, {12, 10, 2}} {{1, 3, 6}, {1, 6, 12}, {1, 12, 10}, {1, 10, 2}}	

Table 1 Two example entries in the look-up table. Each entry is a list of triangulations (each stored as a list of triangles) and 4-point tests (each stored by the indices of the vertices) traversed from the decision tree in pre-roder.

4 Automatic Construction of Look-up Tables

In this section, we will review the table construction process in more detail. For a given sign configuration, we first detect the patch boundaries as a group of rings of linear contours. Then, for each patch, we construct the set of all possible triangulations that could take place on the convex hull of the negative space. Finally, an optimal decision tree is built for every patch detected on the contour. The look-up table can be found on the web at http://www.cs.rice.edu/ jutao/research/contour_tables/.

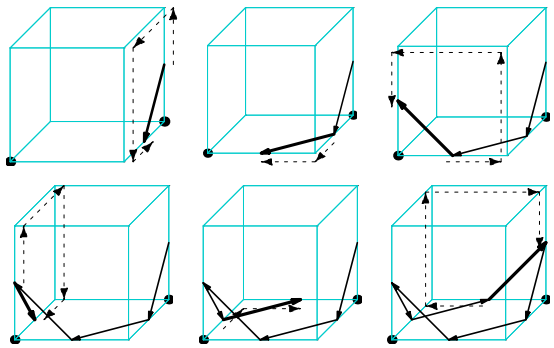


Fig. 11 Face tracing of a single ring. Solid arrows represent linear contours already built, and dashed arrows indicate the tracing route.

4.1 Detection of patch boundaries

A patch on the convex contour in a cubic cell is bounded by linear convex contours on cell faces. These linear contours form single or multiple rings that surround the "holes" of the patch. Since the linear convex contours are unique on each cell face for a given sign configuration, a single ring can be constructed using the following tracing strategy: starting from a cell edge that exhibits a sign change (where an edge intersection is expected) and facing the positive end, look for the next edge with a sign change by turning counter-clockwise on the face boundary. Repeat the search process until it returns to the starting edge, when a closed ring is formed (see figure 11).

Notice that face-tracing produces *oriented* linear convex contours on each cell face. Each linear contour on the cell face is directed so that the negative region on that face lies to its left when looking from outside. Therefore the triangles on the convex contour of the cell that share these linear contours will face towards the negative space. The orientation of the rings are important for determining the set of possible triangulations that share the same patch boundary.

Similar tracing techniques have been described by numerous authors in [2], [9], [13], in which convexity of the negative region on each cell face is preserved. These algorithms construct a single ring for each patch boundary. However, the problem remains on how to group multiple rings to form the boundary of a multiple-genus patch (such as the center left cell in figure 5). Although such patches arise in only 4 cases among 256 sign configurations, they may appear much more often in other non-cubic cells (such as transition cells in multi-resolution grids, see Section 6). We need to be able to identify these cases automatically from the sign configuration and group the rings appropriately.

By definition, a patch is a continuous piece on the convex hull of negative space. Hence it also projects onto a continuous piece of regions on the cell faces. These regions are positive areas that surround positive cor-

ners connected by cell edges. Therefore the boundary of each patch on the convex contour isolates a group of positive corners that are inter-connected by cell edges. The previous face-tracing algorithm could be modified so that rings constructed around a same *edge-connected component of positive corners* are grouped to form the boundary of a single patch. This rule is demonstrated in figure 12. In the top left cell, two rings of linear contours form the boundary of two patches, due to the presence of two isolated positive corners. In the top right cell, where there is only one edge-connected component of positive corners, the two rings are grouped to form the boundary of a single cylinder-like patch. The connectivity of the positive corners in these two cells are illustrated at the bottom of figure 12. In contrast, face-tracing without ring-grouping would give the same result in both cells, thus violating the convex hull property in the second cell.

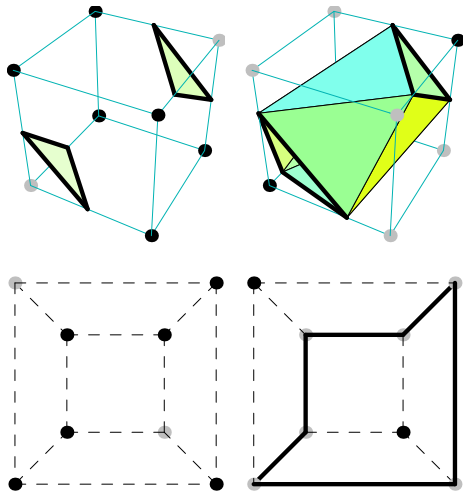


Fig. 12 Top: Grouping of rings of linear contours (highlighted) to form boundaries of patches. Positive corners are colored gray. Bottom: Connectivity graph of positive corners

4.2 Pre-triangulation of Convex Contour

The set of possible triangulations for a patch with an oriented boundary topology can be enumerated by recursive algorithms. However, this often results in redundant triangulations that never appear on the convex contour. For example, the genus-2 patch in the center left cell in figure 5 can be triangulated in only one way on the convex hull of the negative space, regardless of the magnitudes of scalar values at the corners. In contrast, brute-force enumeration would return 21 possible triangulations for an arbitrary genus-2 patch with two triangular holes. The key observation is that, the patches on the convex contour are not arbitrary patches, their vertices (edge intersections) are restricted to fixed edges on the cell. These spacial restrictions limit the number of possible triangulations that could occur on the convex contour. For fast polygonization at run-time, we hope to *pre-triangulate* each patch as much as possible during table construction, based only on the sign configuration.

By the convex hull property, the half-space on the front-facing side of a triangle on the convex contour must contain (or partially contain) every other cell edge that exhibits a sign change. Note that the three vertices of the triangle can move only along three fixed cell edges, this half-space is always contained in the union of the half-spaces formed when the three vertices are at the ends of their cell edges. Hence we have a way to identify triangles that will not appear on the convex contour: given three cell edges (C_1, C_2), (C_3, C_4) *and* (C_5, C_6) (C_i are cell corners), construct border triangles, i.e., triangles formed by one end of each edge in order (such as (C_1, C_3, C_5)). If there is an edge on the cell exhibiting a sign change that lies *completely* to the back of all non-degenerate border triangles, any triangle whose vertices belong to these three edges (in order) will not lie on the convex contour. This idea is illustrated in figure 13. The highlighted cell edge on the left exhibits a sign change, and lies to the back of all the border triangles constructed from the three dashed edges (E_1, E_2, E_3). Hence the dashed triangle formed by intersections on those edges never appears on the convex contour. In contrast, the highlighted cell edge on the right lies partially to the front of at least one of the border triangles formed by the edges (E_1, E_2, E_3), hence the dashed triangle may exist in the triangulation of the convex contour.

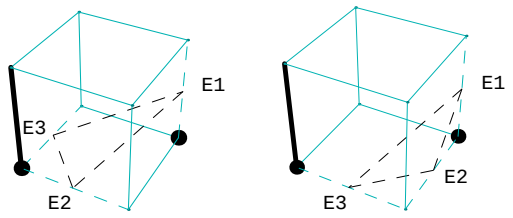


Fig. 13 Identifying triangles that do not lie on the convex contour.

By eliminating triangulations that contain these ineligible triangles, we can trim down the space of possible triangulations. For example, the number of remaining triangulations for the first three cells in figure 5 are respectively 4, 1 and 4.

4.3 Construction of decision trees

Even after pre-triangulation, some patches still have many potential triangulations on the convex hull of the negative space. To speed up the polygonization at run-time,

we can pre-compute a set of 4-point tests on the vertices of the patch (edge intersections) to distinguish between the remaining triangulations. These tests can be organized in a decision tree structure, introduced in the last section. Different sets of tests result in differently shaped decision trees. To obtain optimal performance, we implement a search algorithm that looks for the optimal set of tests that produces a decision tree with the smallest depth. Since each of the two outcomes of a single test eliminates a non-intersecting subset of the remaining triangulations, the minimal depth of the corresponding decision tree is lower bounded by $\log_2 N$, where N is the total number of triangulations. For example, the depth of an optimized decision tree for a patch of length 4, 5 and 6 are 1, 3 and 5 respectively. Further computation reveals that the maximum length of a patch (which can not be pre-triangulated) in a cubic cell is 6; hence any patch can be triangulated within 5 point-face trials on the fly by walking down the pre-computed decision tree. On average, however, it only takes 1.88 tests to determine the triangulation of a patch, due to infrequent occurrence of large patches and the reduced triangulation space as a result of pre-triangulation.

5 Multi-resolution Convex Contouring

In volume visualization, the number of polygons generated by uniform contouring easily exceeds the capacity of modern hardware. For real-time applications, it is often advantageous to display the geometry at different levels of detail depending on the distance from the viewer. This technique has the advantage that it speeds up the rendering process without sacrificing much visual accuracy. By using a view-dependent approach, the grid is contoured at different resolutions depending on the distance from the navigator. In particular, we can create a series of nested bounding boxes centered at the viewer, with the grid resolution decreasing by a factor of 2.

A 2D example of this multi-resolution framework is shown in figure 14 left, in which a circle is contoured using cell grid at two different resolutions. When the coarse cells at the top meet the fine cells at the bottom, the contour in the coarse cells need to be consistent with the contour from the neighboring fine cells on the common edges. We call these coarse cells transition cells, which are adjacent to cells at a finer resolution. A 2D transition cell thus has five corners and five edges, as shown in the middle of figure 14. By connecting edge intersections on each cell edge, the transition cells can be contoured in a way similar to a regular 4-corner cell, yielding consistent contours with the adjacent fine cells. Some of the contoured example are shown in figure 14 right. Notice that the convexity of the negative region is still preserved in each transition cell.

In 3D, a transition cell between two resolutions is either adjacent to two fine cells on an edge, or adjacent to four fine cells on a face (see figure 15 left). These two types of cells can be regarded as convex polyhedrons with 9 corners (figure 15 center) and 13 corners (figure 15 right) respectively.

Since the previous discussion on regular cells applies to any convex polyhedron, we can also build look-up tables and perform convex contouring on these transition cells. In this way, crack-free surfaces can be contoured on nested grids within the same framework as uniform contouring.

5.1 Convex contours in transition cells

As in a regular cell, the convex contour inside a transition cell is outlined by the linear contours on the cell faces. Since the linear convex contour is unique on each face for a given sign configuration, the boundary of patches on the convex contour can be pre-computed using the proposed face-tracing algorithm. At the top of figure 16, the oriented boundaries of patches in different transition cells are detected and drawn as dashed arrows. Since convex contours from neighboring cells in a multi-resolution grid always share the same linear contour on the common face as their boundaries, topological consistency is preserved everywhere on the contoured surface.

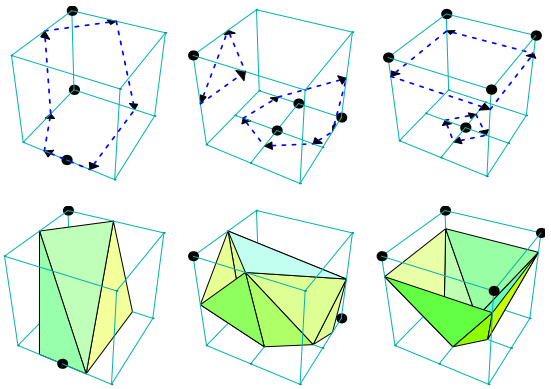


Fig. 16 Patch boundaries (top) and triangulation (bottom) in three transition cells.

By pre-computing optimal decision trees for each patch, the triangulation can be determined by applying successive 4-point tests on the edge intersections. At the bottom of figure 16, patches detected from the cells on the top are triangulated on the convex hull of the negative space. However, due to the presence of four co-planar faces on a 13-corner cell, this convex hull could degenerate onto a plane (figure 17 left). To determine the correct triangulation of the convex contour on the degenerate convex hull, we introduce an outward perturbation to

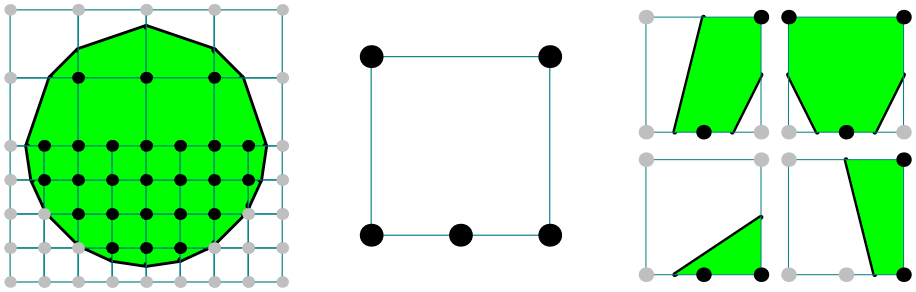


Fig. 14 A circle contoured on a 2D multi-resolution grid (left), consisting of regular cells and transition cells (middle). Example contours in a transition cell are shown on the right. Positive corners are colored gray, and negative corners are colored black.

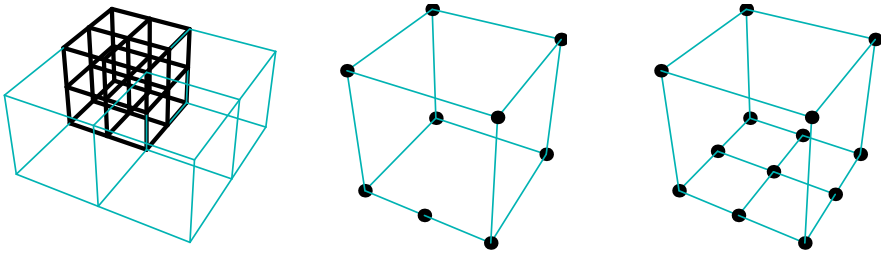


Fig. 15 Transition cells on nested grids: 9-corner cell (in the middle) with 6 faces and 13-corner cell (on the right) with 9 faces. Their positions in the nested grids are illustrated on the left.

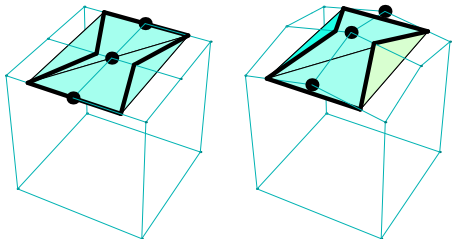


Fig. 17 Perturbing cell corners on co-planar faces. The patch boundary is outlined by highlighted lines.

the cell corners of the co-planar faces so that they no longer lie on the same plane (figure 17 right). Since the perturbations are small and perpendicular to the plane that contains those faces, the triangles on the degenerate convex hull still forms a convex hull after perturbation. Therefore the 4-point tests on edge intersections that all lie on that plane will instead be applied to the new edge intersections computed from the perturbed cell corners. Since the final triangulation determined by these tests take place on the unperturbed geometry, the patches are still bounded by the linear contours on cell faces and topological consistency is preserved.

5.2 Look-up tables

The look-up tables are constructed in the same way as for uniform cells, except for the fact that the transition cells have more complicated topology that results in larger patches and more possible triangulations. Each entry in the look-up table corresponds to a sign configuration at the cell corners. For the 9-corner cell, the table consists of $2^9 = 512$ entries, and the look-up table for the 13-corner cell consists of $2^{13} = 8192$ entries. Each entry stores compact decision trees for each patch in the same format as described in the uniform case. The resulting decision trees have maximum depth of 5 in a 9-corner cell, and 14 in a 13-corner cell. On average, however, it only takes 1.92 tests to determine the triangulation of a patch in a 9-corner cell, and 4.76 tests in a 13-corner cell. The look-up tables can also be found on the web at http://www.cs.rice.edu/ jutao/research/contour_tables/.

5.3 Multi-resolution contouring

Convex contouring of a cell on a nested grid proceeds in exactly the same way as on a uniform grid. The only noticeable differences are the different look-up tables to use for different type of cells, and appropriate perturbations to be applied when the four points in the test lie on the co-planar faces on the 13-corner cell. Therefore contouring on uniform and multi-resolution grids can be implemented within the same framework. The pre-computed look-up tables guarantee the consistency of the surface, therefore no crack-filling is ever needed. In our implementation (see figure 18), users navigate in space and the grid resolution changes in real time depending on the viewer's position. The table driven patch detection

and triangulation provides fast crack-less contouring of both regular cells and transition cells.

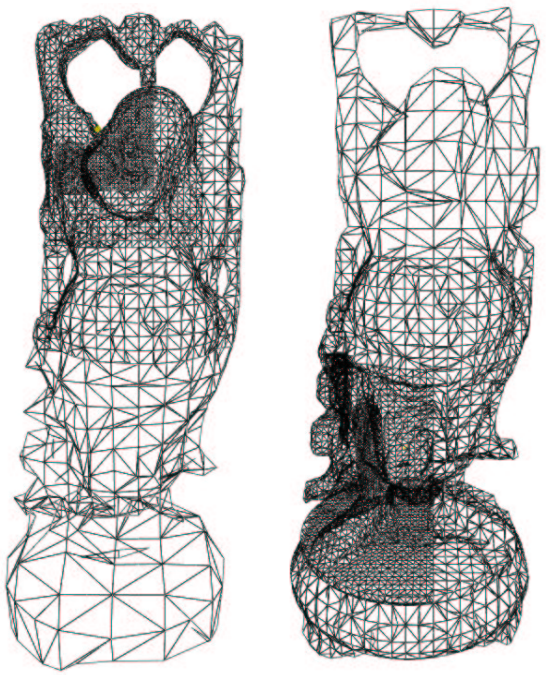


Fig. 18 Multi-resolution contouring on the Happy Buddha with different viewer positions.

6 Conclusion

In this paper, we presented a table-driven contouring method that produces topologically consistent surfaces with a local convexity property. The polygonal contour inside each cell encloses a convex negative space, which enables fast collision detection on the contoured surface. The construction of contours is greatly accelerated via table look-up of triangulation from the signs and magnitudes of corner values. We also showed how this method could be used to perform crack-free contouring on multi-resolution grids. By constructing look-up tables for transition cells, convex contouring of both uniform and multi-resolution grids can be implemented within a single unified framework. In fact, convex contours are also defined for arbitrary convex polyhedrons, such as square pyramids and tetrahedrons. Hence the same technique proposed here can be used to construct look-up tables for more complex shapes. The look-up tables for uniform cells and transition cells in multi-resolution grids are also available on the web.

References

1. H.H. Baker (1989). Building Surfaces of Evolution: The Weaving Wall. International Journal of Computer Vision, 3:51-71.
2. J. Bloomenthal (1988). Polygonization of Implicit Surfaces. Computer Aided Geometric Design, 5:341-355.
3. I.Boada and I.Navazo (2001). Multiresolution Isosurface Fitting using an Octree based Surface Hierachy. Research Report IIiA 01-02-RR, Institut Inform'atica i Aplicacions, University of Girona.
4. M.J. Duurst (1988). Additional Reference to Marching Cubes. Computer Graphics, 22(2):72-73.
5. A. Van Gelder and J. Wilhelms (1994). Topological Considerations in Isosurface Generation. ACM Transactions on Graphics, 13(4):337-375.
6. R. K. Guy (1958). Dissecting a Polygon Into Triangles. Bulletin Malayan Math. Society, 5:57-60.
7. W. Lorensen and H. Cline (1987). Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Computer Graphics (SIGGRAPH '87), 21(4):163-169
8. H. Muller and M. Stark (1993). Adaptive Generation of Surfaces in Volume Data. The Visual Computer, 9(4):182-199.
9. G.M. Nielson and B. Hamann (1991). The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes. Proceedings of IEEE Visualization 91, 83-91.
10. P.Ning and J. Bloomenthal (1993). An Evaluation of Implicit Surface Tiles. IEEE Computer Graphics & Applications, 13(6):33-41.
11. T. Poston, T.T. Wong and P.A. Heng (1998). Multiresolution Isosurface Extraction with Adaptive Skeleton Climbing. Computer Graphics Forum, 17(3):137-148.
12. R. Shekhar, E. Fayyad, R. Yagel, and J.F. Cornhil (1996). Octree-based Decimation of Marching Cubes Surfaces. IEEE Visualization '96:335-344.
13. A. Wallin (1991). Constructing Isosurfaces from CT Data. IEEE Computer Graphics and Applications, 11(6):28-33.
14. R. Westermann, L. Kobbelt, and T. Ertl (1999). Real–Time Exploration of Regular Volume Data by Adaptive Reconstruction of Isosurfaces. The Visual Computer, 15:100-111.