Intersection-free Contouring on An Octree Grid

Tao Ju
Washington University in St. Louis
One Brookings Drive
St. Louis, MO 63130, USA
taoju@cs.wustl.edu

Tushar Udeshi
Zyvex Corporation
1321 North Plano Road
Richardson, Texas 75081-2426, USA
tudeshi@zyvex.com

Abstract

A method for extracting intersection-free iso-surfaces from volumetric data with an octree structure is presented. Unlike contouring techniques designed for uniform grids (such as Marching Cubes), adaptive contouring methods (such as Dual Contouring) can and do often generate intersecting polygons. Our main contribution is a polygon generation algorithm that produces triangles enclosed in non-overlapping volumes, which guarantees an intersection-free mesh. Like other adaptive contouring methods, this new method generates crack-free and feature-preserving surfaces on both uniform and octree grids. We demonstrate the method on both scanned objects and industrial models.

1. Introduction

Contouring is the process of extracting iso-surfaces from volumetric data. Many applications, such as Finite Element Analysis, require their input surface to be a closed mesh with no intersecting polygons. Traditional contouring techniques, such as the Marching Cubes [13] method and its variations [4, 2, 15, 12], fulfill this requirement because they generate triangles enclosed in grid cells and, within each cell, the triangles are not intersecting. However, these methods are typically confined to a uniform grid structure and the resulting mesh size may become prohibitive for large volumes.

Dual Contouring, introduced by Ju et al. [11], is designed for extracting iso-surfaces with adaptive resolution. By extending the polygon generation scheme proposed in SurfaceNet [7] from a uniform grid onto an octree, Dual Contouring generates crack-free contours in an adaptive manner. In addition, DC is capable of reproducing sharp geometry features when Hermite data is available.

Unfortunately, unlike Marching Cubes, a polygon generated by Dual Contouring is not contained in a grid cell, and hence guaranteeing intersecting-free surfaces becomes difficult. In fact, we observed in our experiments that the surface produced by Dual Contouring is *rarely* intersection-free (an example is shown in Figure 1 (top)). Although subsequent work [19, 1, 17, 20, 14] has improved upon the topology representation in DC, the problem of geometric intersections has remained un-addressed.

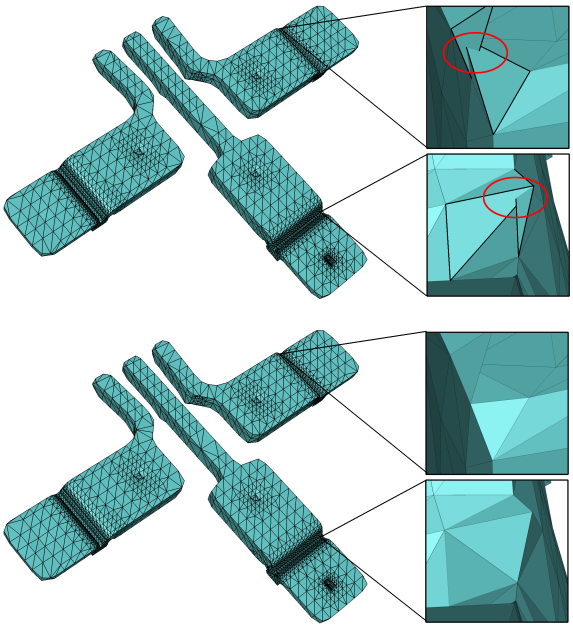


Figure 1. Adaptive contouring of the Actuator Pads model using Dual Contouring (top), which contains self-intersections (shown on right), and using intersection-free contouring (bottom).

Several other methods have also been proposed for adaptive contouring [9, 3]. These methods use a polygon generation technique similar to Dual Contouring, and hence they may also produce intersecting polygons. In addition, we

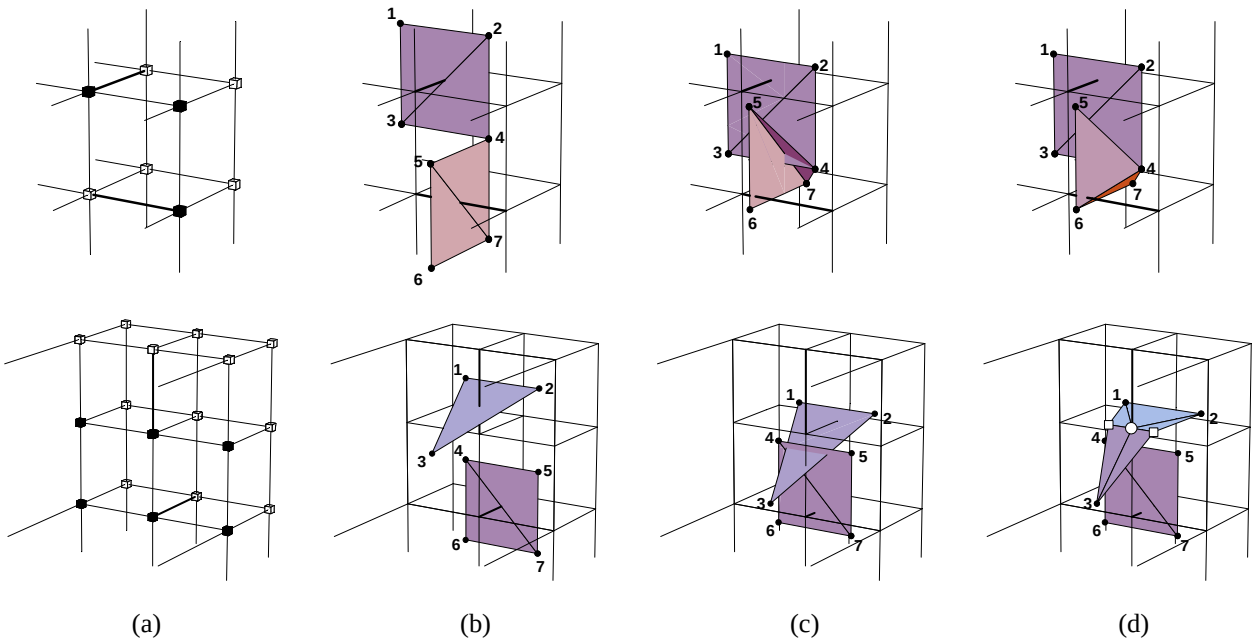


Figure 2. Resolving intersecting polygons on a uniform (top) and octree (bottom) grid. (a): Signs at grid points (black for positive and white for negative). (b,c): Non-intersecting (b) and intersecting (c) triangles generated by DC for the thickened grid edges in (a). (d): Our method resolves the self-intersections in (c) by changing the triangulation (top) and introducing additional vertices on two grid faces (empty squares) and a grid edge (empty circle).

note that one class of mesh-repair methods [3, 10] relies on volumetric scan-conversion and adaptive contouring to fix geometrically incorrect (e.g., self-intersecting) meshes. The effectiveness of these methods become questionable as the output of contouring may again contain new self-intersections.

1.1. Contributions

To date, we have not seen any study on the geometric correctness of adaptive contouring methods. In this paper, we analyze intersecting polygons created by Dual Contouring, and propose an intersection-free solution for adaptive contouring. Just as Marching Cubes generates triangles enclosed in grid cells, our method creates triangles enclosed in non-overlapping volumes, and hence avoids potential intersections. In particular, we make the following observations and contributions:

- We show that the polygon generation technique in Dual Contouring can, and often does, yield intersecting polygons on both uniform grids and octrees.
- We introduce a simple polygon generation technique that is guaranteed to produce intersection-free and crack-free meshes on an octree, but at the cost of an excessive number of output triangles.
- We improve upon the above technique by employing multiple polygon generation rules, which dramatically reduces the number of output triangles while still guaranteeing geometric correctness.

2. Primal and dual contouring

We start by investigating two competing contouring schemes. Given a uniform grid of signed scalar values, the Marching Cubes (MC) method [13] creates one vertex for each edge on the grid that contains a sign change, which are then connected by triangles that lie within cubic grid cells. Since the triangles within each cell are non-intersecting, and since all cells are disjoint, the surface extracted by MC is always intersection-free. Unfortunately, it is difficult to extend MC beyond a uniform grid, because cracks may occur between polygons extracted from grid cells of different sizes.

To extract crack-free iso-surfaces on adaptive grids, Dual Contouring (DC) [11] extends a polygon generation method that was first proposed in SurfaceNet [7] on a uniform grid.

Given a signed octree grid, DC creates one vertex for each grid cell that contains a sign change, and builds one polygon for each grid edge exhibiting a sign change, which connects the vertices of the cells sharing that edge [1]. The surface is guaranteed to be topologically closed as each polygon edge is shared by an even number of polygons.[2]

Note that a vertex (polygon) generated by DC is topologically dual to a polygon (vertex) generated by MC. Hence we refer to MC as a *primal method*, and call DC and other methods using polygon generation mechanisms similar to DC [7, 19, 1, 14, 20, 9, 3] *dual methods*.

Unlike MC, the polygons generated by dual methods are not constrained to lie within grid cells, and they may geometrically intersect. Figure 2 shows two examples of self-intersections resulted from applying DC on a uniform grid and on an octree.[3] Notice that the same sign configuration may yield either non-intersecting or intersecting polygons, depending on the geometric locations of the vertices within the grid cells.

3. A hybrid approach

We first introduce a simple, hybrid method that combines primal and dual contouring to generate both crack-free and intersection-free iso-surfaces on octree grids. Recall that MC [13] generates intersection-free surfaces because the triangles are contained within disjoint volumes (i.e., cubic grid cells), and within each volume, the triangles do not intersect. In the same spirit, our hybrid approach generates triangles contained in non-overlapping volumes, called *envelopes*, and within each envelope, triangles do not intersect.

As described in the previous section, a primal contouring method creates vertices on grid edges, whereas a dual method creates vertices within grid cells. In the hybrid method, we associate one *edge vertex*, *face vertex*, or *cell vertex* on each grid edge, face or cell that contains a sign change (the placement of these vertices are discussed in next section). To create the polygons, we generate a triangle fan for each grid edge e that contains a sign change, where each triangle connects the edge vertex v_e on e, the face vertex v_f on a face f sharing e, and the cell vertex v_c in a cell c sharing f, as shown Figure 3 (top). Since a grid edge may be shared by 3 or 4 cells on an octree, a fan of 6 or 8 triangles will be generated.

Note that each triangle fan generated by this hybrid method is a tessellation of a polygon produced by the DC method. Like DC, this hybrid approach generates a crack-free surface. However, unlike DC, adding extra vertices on edges and faces provides us with additional geometric freedom to place triangles so that they do not intersect:

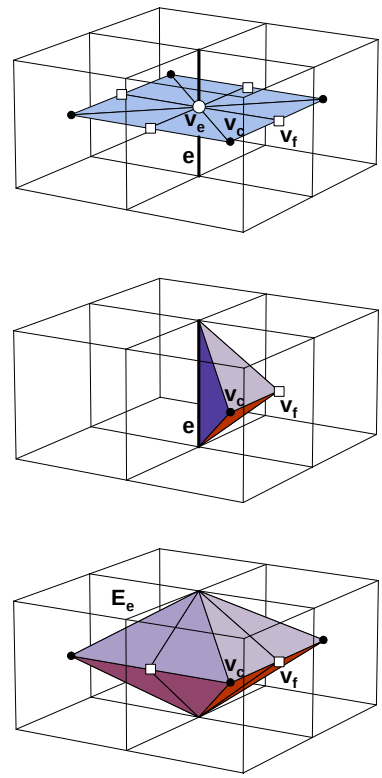


Figure 3. Triangle fan (top) generated by the hybrid method at a grid edge e, a single tetrahedron (middle) and the union of all tetrahedra that form the envelope E_e (bottom). Cell vertices, face vertices and edge vertices are drawn as filled circles, empty squares and empty circles.

Proposition 1 *Triangles generated by the hybrid method do not intersect if each cell vertex, face vertex and edge vertex lies interior to the corresponding cell, face or edge.*

Proof: We define the *envelope* E_e at a grid edge e as the union of (6 or 8) tetrahedra, each formed by edge e, vertex v_f on a face f sharing e, and vertex v_c in a cell c sharing f. In Figure 3, we show one such tetrahedron in the middle and the union of all tetrahedra at the bottom. In particular, E_e is called a *valid* envelope if each face vertex v_f and cell vertex v_c lies interior to the face f or cell c.

We first observe that valid envelopes E_e of different edges e are disjoint. This is because different envelopes do

[1] In our discussion, a grid edge (face) on an octree refers to a minimal edge (face) that does not contain smaller edges (faces).

[2] DC may produce non-manifold surfaces, which has been studied in several other works [1, 20, 14].

[3] Every quadrilateral generated by DC is split into two triangles along an arbitrary diagonal.

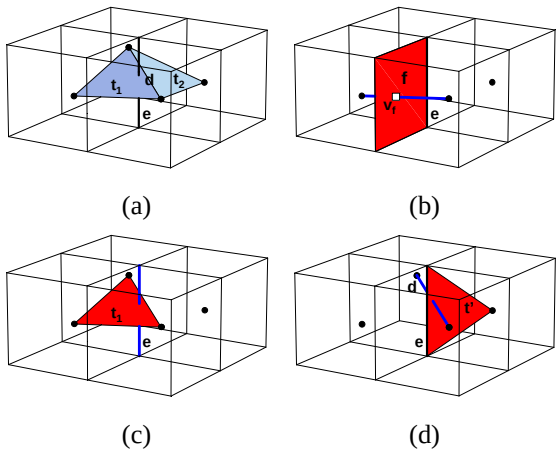


Figure 4. A trivial triangulation (a), and the polygon-edge intersections for the first (b) and second (c,d) condition in Proposition 1 to determine if the triangulation is contained in the envelope.

not share common tetrahedra, and each tetrahedron of E_e formed by e, a face vertex v_e and a cell vertex v_c (e.g., Figure 3 (middle)) is contained within the cell c and is disjoint from other tetrahedra in the same cell. Next, when the edge vertex v_e lies interior to the edge e, the triangles generated by the hybrid method around e are contained in separate tetrahedra of the envelope E_e. Since valid envelopes are disjoint, the triangulated surface is intersection-free. □

4. Intersection-free contouring

The hybrid method creates a geometrically correct surface, unfortunately at the cost of a significant increase in mesh size. In particular, at a grid edge where DC would produce 1 or 2 triangles, the hybrid method creates 6 or 8 triangles. Next we present our full algorithm, which generates far fewer triangles than the hybrid method yet still ensures an intersection-free output. The idea is to replace a triangle fan created by the hybrid method with a smaller number of triangles which are still contained in a valid envelope.

4.1. Trivial triangulation

Given a grid edge e containing a sign change, we define a *trivial triangulation* T_e as the one triangle (if e is shared by 3 cells) or two triangles (if e is shared by 4 cells) that connect cell vertices between cells sharing e. A trivial triangulation with two triangles is an arbitrary triangulation of a quadrilateral generated by DC, as shown in Figure 4 (a).

Ideally, we would like to use the trivial triangulation in place of a fan of 6 or 8 triangles (e.g., Figure 3 (top)) whenever possible. In particular, we observe that if T_e is contained inside some valid envelope E_e, replacing the triangle fan generated in the hybrid method with the trivial triangulation T_e will not introduce intersecting polygons. This is because valid envelopes are disjoint (as shown in the proof of Proposition 1). To determine if a valid envelope exists for T_e, we develop the following tests (see proof in Appendix A):

Proposition 2 *Let e be an edge on an octree grid containing a sign change. Trivial triangulation T_e is contained in a valid envelope if and only if:*

1. *Each edge of T_e that connects vertices of cells sharing a common grid face f must intersect f (Figure 4 (b)).*
2. *If T_e is a triangle, e must intersect T_e. If T_e consists of two triangles t_1, t_2 sharing a diagonal d (Figure 4 (a)), then e must intersect t_1 (or t_2) (Figure 4 (c)),* and *d must intersect the triangle t' formed by e and the vertex of T_e not contained in t_1 (or t_2) (Figure 4 (d)).*

Intuitively, the two tests examine the envelope E_e constructed by placing the face vertex v_f on each face f sharing e at the intersection between f and a side of T_e (Figure 4 (b)). While the first condition checks if E_e is valid, the second condition further verifies if the triangulation T_e is completely contained within E_e. Note that performing the above tests on an octree edge involves no more than 6 polygon-edge intersections (4 for the first condition, and 2 for the second), which can be computed efficiently using the Separating Axes Theorem [8]. In addition, since the first condition is always satisfied when the vertices of an edge of T_e lie in cells at the same level on the octree, the number of intersections tests can be further reduced in such situations.

4.2. The algorithm

To create a crack-free and intersection-free iso-surface, we apply different polygon generation rules when contouring an octree grid. At each grid edge e exhibiting a sign change, we perform the two tests in Proposition 2 and proceed by the following rules, as illustrated in Figure 5:

Rule 1: If both conditions in Proposition 2 hold, we output the trivial triangulation T_e (Figure 5 (a)). Note that when e is shared by four cells, there are two possible triangulations made by swapping the diagonal. Either triangulation that satisfies Proposition 2 is used.

Rule 2: If condition (1) holds but condition (2) fails, we generate a fan of triangles from the edge vertex v_e to cell vertices v_c in all cells c sharing e (Figure 5 (b)).

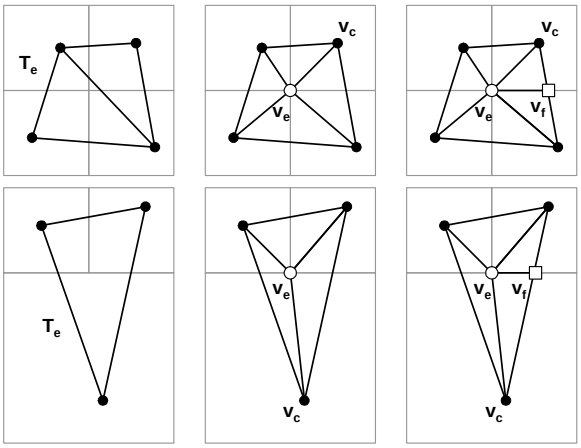


Figure 5. A top-down view of the three polygon generation rules for a grid edge shared by four cells (top) and three cells (bottom). Cell vertices, face vertices and edge vertices are drawn as filled circles, empty squares and empty circles.

Rule 3: If condition (1) fails, we generate a fan of triangles from the edge vertex v_e to cell vertices v_c in all cells c sharing e as well as face vertices v_f on only those grid faces f where condition (1) fails (Figure 5 (c)).

When the cell vertices, face vertices and edge vertices are placed interior to their corresponding cells, faces and edges, each polygon generation rule above results in triangles contained in a valid envelope, and hence the resulting triangulated surface contains no intersecting polygons. The surface is also crack-free as the introduction of a face vertex on a grid face is consistent when processing every grid edge on that face. This polygon generation process can be efficiently implemented as recursive traversals on the octree using the procedures described in [11], which visits each edge on the grid with running time linear to the size of the octree.

Figure 2 (d) compares the result of intersection-free contouring with that of DC on both uniform grid and octree. Note that self-intersections at the top and bottom of 2 (c) are resolved in the new contouring method respectively by polygon generation **Rule 1** (i.e., by flipping the diagonal of the trivial triangulation) and **Rule 3**.

4.3. Geometry creation

We follow the DC method [11] in computing the cell vertices. Specifically, given a signed uniform grid with Hermite data attached to edges of the grid [10], the location of each cell vertex is computed by minimizing a Quadratic Energy Function [6] defined by the Hermite data on the edges of the cell. The minimization allows representation of sharp features such as edges and corners. The uniform grid can be adaptively simplified into an octree by merging cell vertices. Note that cell vertices can also be created from distance maps as in SurfaceNet [7] or from scalar fields [16].

Given cell vertices on an octree, the remaining face vertices and edge vertices are placed to approximate the geometry represented by the cell vertices while maintaining a smooth appearance.[4] To locate an edge vertex v_e, we interpolate the locations of the cell vertices of cells sharing e. This is done by computing 2D barycentric coordinates for the cell vertices projected onto a plane orthogonal to e. Specifically, assume that the grid edge e has end points at $\{x, y, z^-\}$ and $\{x, y, z^+\}$, and let each cell vertex around e have coordinate $\{x_i, y_i, z_i\}$, the z component of v_e is computed as:

$$z = \sum_i w_i z_i$$

where w_i are affine weights satisfying $\sum_i w_i = 1$. To better approximate the underlying geometry, we further require that when $\{x_i, y_i, z_i\}$ forms a triangle or a planar quadrilateral, $\{x, y, z\}$ should lie on the same plane. Therefore w_i must also satisfy the *linear precision* property:

$$x = \sum_i w_i x_i, \text{ and } y = \sum_i w_i y_i \qquad (1)$$

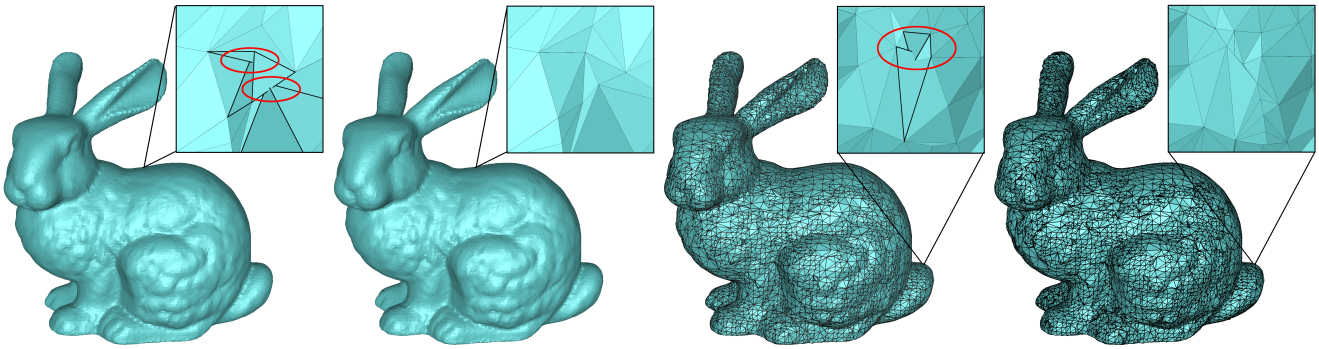
There are a wealth of techniques for computing weights of this form on a 2D plane. We choose the Mean Value weights [5] due to its stability and positivity even for non-convex shapes (which are common for the projected polygon $\{x_i, y_i\}$). To ensure that v_e lies on the edge e, we further clamp z to be within $[z^-, z^+]$.

To locate a face vertex v_f, we take the intersection of the supporting plane of the grid face f with the line connecting the two cell vertices on each side of f. If the intersection point lies outside f, we place the face vertex v_f at the centroid of the edge vertices of all grid edges on f (observe from polygon generation **Rule 3** that an edge vertex is created for each grid edge shared by a face f containing a face vertex).

5. Results

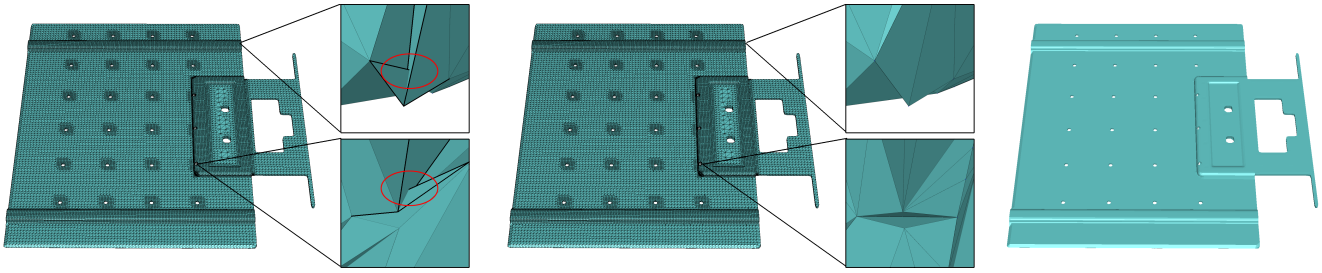
Figure 6 (a,c) demonstrate self-intersections resulted from DC on a uniform and an adaptive grid. In comparison, our method generates intersection-free surfaces (b,d) on both grids. Note that the self-intersections shown in the

[4] Although locations of these vertices may be obtained directly from the original uniform volume from which the octree is created, in general we do not make assumption about the availability of such volume.



(a) (b) (c) (d)

Figure 6. Contouring a bunny on a uniform grid (a,b) and an octree grid (c,d). Dual Contouring creates undesirable intersections in both cases (a,c), whereas our method generates intersection-free surfaces (b,d).



(a) (b) (c)

Figure 7. Adaptive contouring of a simulated MEMS mirror model at octree depth 30 using Dual Contouring (a) and our method (b,c).

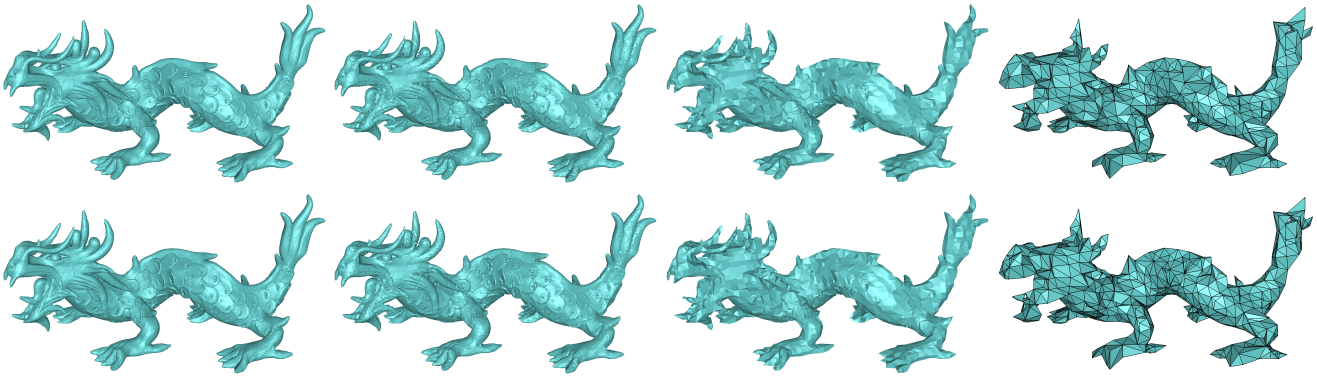
close-up views are resolved either by choosing an appropriate trivial triangulations (as in (b)) or by generating a triangle fan (as in (d)).

We next apply our method to extract iso-surfaces for octree models of Micro Electro Mechanical (MEMS) devices generated using the MEMulator^{TM} software [18]. These models (Figure 1 and 7) are generated by geometric modeling a MEMS fabrication process using 2D layout masks as input. A water-tight, intersection-free surface mesh needs to be generated from the octree so that Finite Element Analysis can be performed. The octree in each example has depth 30 and is created from a scalar volume in a bottom-up fashion [16]. Observe from the figures that our new method produces a geometrically correct mesh while preserving surface details.

Figure 8 compares our method with DC on large scanned models. The Dragon model was converted to Hermite data at octree depth 10 using the PolyMender software [10], and the surfaces shown are extracted after adaptive simplification with increasing error thresholds. Although difficult to see, Dual Contouring (top) generates 4583, 401, 74, and 30 self-intersections respectively on each model. On the other hand, the surfaces generated by our new method (bottom) contain no intersecting polygons.

Table 1 reports the performance of the algorithm on various examples and compares it with DC. All tests are run on a P4 3.4GHz computer with 2GB memory. Our method typically takes 2 to 3 times longer than Dual Contouring, with the extra time spent on performing triangle-edge intersection tests and computing locations of new vertices.

Note from Table 1 that when the octree is mostly "restricted" (i.e., the octree levels of two adjacent cells differ at most by 1), such as examples in Figure 1, 7 and 6 (a), our method results in little increase in total triangles because the two conditions in Proposition 2 are more likely to hold. Since such restricted octrees are desirable in FEA-type applications to provide graded meshes without wildly varying element size, our method provides a low-



(a) (b) (c) (d)

Figure 8. Adaptive simplification of a complex model. DC (top) results in 4583, 401, 74 and 30 self-intersections respectively on each surface. The surfaces generated by our method (bottom) are intersection-free.

overhead, intersection-free iso-surfacing solution for these applications. When applying our method to adaptively simplified scanned models, where the octree can be highly non-restricted, a larger increase in the polygon count is observed. Nevertheless, such increase is typically less than 50% even in the most complex examples.

	Time (DC)	Triangles (DC)	Intersections (DC)	Time (IC)	Triangles (IC)
Fig 1	0.015	6644	16	0.016	6918
Fig 6 (b)	0.016	16194	58	0.047	22594
Fig 6 (a)	0.062	91720	438	0.109	91728
Fig 7	0.047	80990	109	0.11	81850
Fig 8 (d)	<1ms	1712	30	<1ms	2353
Fig 8 (c)	0.016	11128	74	0.031	15255
Fig 8 (b)	0.063	72366	401	0.203	103724
Fig 8 (a)	0.547	659948	4583	2.64	1025807

Table 1. Polygon generation time in seconds of Dual Contouring (DC) and our intersection-free contouring (IC) (timing excludes I/O and octree simplification).

6. Discussion

In this paper we present a new method for extracting crack-free and intersection-free surfaces on adaptive grids. Computing geometrically correct iso-surfaces is an important yet under-addressed area. As future work, we plan to incorporate recent developments in adaptive contouring [19, 1, 20] to improve topological flexibility of the current algorithm, e.g., to allow multiple vertices in a cell, on a face or on an edge and to generate a topologically manifold surface. Moreover, we shall also investigate tighter conditions than those in Theorem 1 to further reduce the triangle overhead of the current method. For example, an interesting question that we are still seeking an answer for is whether applying polygon generation **Rule 1** (i.e., always using a trivial triangulation) alone is sufficient to create an intersection-free surface on an uniform grid.

7. Acknowledgement

We would like to thank Zyvex Corporation for the permission to use the Mirror and Actuator Pads models, and thanks to the Stanford 3D Scanning Repository for the Bunny and Dragon models.

References

[1] K. Ashida and N. I. Badler. Feature preserving manifold mesh from an octree. In *Symposium on Solid Modeling and Applications*, pages 292–297, 2003.

[2] H. Baker. Building surfaces of evolution: The weaving wall. *IJCV*, 3:51–71, 1989.

[3] S. Bischoff, D. Pavic, and L. Kobbelt. Automatic restoration of polygon models. *ACM Trans. Graph.*, 24(4):1332–1352, 2005.

[4] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5(4):341–356, 1988.

[5] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.

[6] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 209–216, Los Angeles, California, August 1997. ACM SIGGRAPH / Addison Wesley.

[7] S. F. F. Gibson. Using distance maps for accurate surface reconstruction in sampled volumes. In *1998 Volume Visualization Symposium*, pages 23–30. IEEE, October 1998.

[8] S. Gottschalk. Separating axis theorem. *Technical Report TR96-024, Dept. of Computer Science, UNC Chapel Hill*, 1996.
[9] A. Greß and R. Klein. Efficient representation and extraction of 2-manifold isosurfaces using kd-trees. *Graphical Models*, 66(6):370–397, 2004.
[10] T. Ju. Robust repair of polygonal models. *ACM Trans. Graph.*, 23(3):888–895, 2004.
[11] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Transactions on Graphics*, 21(3):339–346, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).
[12] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature-sensitive surface extraction from volume data. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 57–66. ACM Press / ACM SIGGRAPH, August 2001.
[13] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, volume 21, pages 163–169, Anaheim, California, July 1987.
[14] G. M. Nielson. Dual marching cubes. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 489–496, Washington, DC, USA, 2004. IEEE Computer Society.
[15] G. M. Nielson and B. Hamann. The asymptotic decider: Resolving the ambiguity in marching cubes. In *IEEE Visualization*, pages 83–93, 1991.
[16] E. Parker and T. Udeshi. Exploiting self-similarity in geometry for voxel based solid modeling. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 157–166, New York, NY, USA, 2003. ACM Press.
[17] S. Schaefer and J. Warren. Dual marching cubes: Primal contouring of dual grids. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference on (PG'04)*, pages 70–76, Washington, DC, USA, 2004. IEEE Computer Society.
[18] T. Udeshi and E. Parker. MEMulator^{TM}, a fast and accurate geometric modeling, visualization and mesh generation tool for 3D MEMS design and simulation. *Technical Proceedings of the 2003 Nanotechnology Conference and Trade Show*, pages 480–484, 2003.
[19] G. Varadhan, S. Krishnan, Y. Kim, and D. Manocha. Feature-sensitive subdivision and iso-surface reconstruction. In *IEEE Visualization 2003*, pages 99–106. IEEE, 2003.
[20] N. Zhang, W. Hong, and A. Kaufman. Dual contouring with topology-preserving simplification using enhanced cell representation. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 505–512, Washington, DC, USA, 2004. IEEE Computer Society.

A. Proof of proposition 2

Proof: Recall that an envelope E_e involves the grid edge e, the cell vertices in cells sharing e, and face vertices on faces sharing e. When condition 1 holds, we can construct a valid envelope by placing the face vertex v_f on each face f at the intersection between f and an edge of T_e connecting cell vertices in face-adjacent cells (see Figure 4 (b)). If condition 1 fails at some edge of T_e, no placement of face vertices will result in a valid envelope that contains that edge of T_e.

We next show that, when condition (1) holds and E_e is constructed as above, T_e is completely contained in E_e if and only if condition (2) is satisfied. One point that we will be relying on is that E_e now becomes the union of the tetrahedra each formed by e and one edge of T_e connecting cell vertices in face-adjacent cells.

If T_e is a triangle, and assuming that T_e intersects e at v_e, we can split T_e into three triangles using v_e, each of which lies in the tetrahedra formed by e and an edge of T_e. Therefore T_e lies in E_e, which is the union of these tetrahedra. To show necessity, we assume T_e is contained in E_e. Since each edge of T_e passes through a grid face that shares e, T_e intersects with the supporting line of e at some point v_e. Note that v_e must lie on e, because the extension of e is not contained in the envelope E_e.

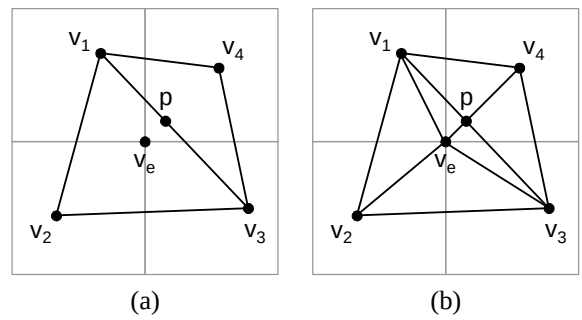


Figure 9. Notations for proving Proposition 2.

If T_e consists of two triangles t_1, t_2 that form a quadrilateral, let the triangles have vertices $t_1 = \{v_1, v_2, v_3\}$ and $t_2 = \{v_1, v_3, v_4\}$. Without loss of generality, assume that t_1 intersects e at v_e and that diagonal $d = \{v_1, v_3\}$ intersects the triangle t' formed by e and v_4 at p, as shown in Figure 9 (a). We can use v_e, p to split the two triangles into 6 smaller triangles, as shown in Figure 9 (b). We observe that $\{v_1, v_2, v_e\}$ lies in the tetrahedron formed by e and $\{v_1, v_2\}$, $\{v_2, v_3, v_e\}$ lies in the tetrahedron formed by e and $\{v_2, v_3\}$, $\{v_3, p, v_e\}$ and $\{v_3, v_4, p\}$ lie in the tetrahedron formed by e and $\{v_3, v_4\}$, and $\{v_1, v_e, p\}$ and $\{v_1, p, v_4\}$ lie in the tetrahedron formed by e and $\{v_4, v_1\}$. Therefore, the original triangles t_1, t_2 are contained in the envelope E_e, which is union of the four tetrahedra. To show necessity, we assume that t_1, t_2 are contained in E_e. Following a similar argument as above, either t_1 or t_2 must intersect with the supporting line of e at some point v_e, and that v_e must lie on e. Assuming that v_e lies on t_1, the diagonal d must intersect the supporting plane of the triangle t' at some point p, which lies in the cell containing v_4. Note that p must lie on t', as the extension of t' on its supporting plane within the cell containing v_4 is not contained in the envelope E_e. □