Level Set Extraction from Gridded 2D and 3D Data

David Eberly
Magic Software, Inc.
http://www.magic-software.com

Created: August 16, 2002

1 Introduction

A standard isosurface extraction algorithm for a 3D image is the Marching Cubes algorithm [4]. The image is assumed to be defined on a regular lattice of size $N_0 \times N_1 \times N_2$ with integer points (x, y, z) where $0 \le x < N_0$, $0 \le y < N_1$, and $0 \le z < N_2$. The image values themselves are $F(x, y, z)$. An isosurface is of the form $F(x, y, z) = c$ for some specified level value c where x, y, and z are treated as continuous variables. A voxel in the image is a rectangular solid whose corners are eight neighboring lattice points (x_0, y_0, z_0), (x_0+1, y_0, z_0), (x_0, y_0+1, z_0), (x_0+1, y_0+1, z_0), (x_0, y_0, z_0+1), (x_0+1, y_0, z_0+1), (x_0, y_0+1, z_0+1), and (x_0+1, y_0+1, z_0+1). The Marching Cubes algorithm analyzes each voxel in the image and determines if the isosurface intersects it. If so, the algorithm produces a triangle mesh for the voxel that is intended to approximate that portion of the isosurface inside the voxel. By selecting a level value that cannot be an image value, for example by selecting a non–integer value when the image has only integer values, the voxel analysis requires determining the signs of $G(x, y, z) = F(x, y, z) - c$ at the eight corners, each sign positive or negative. If two adjacent corners have opposite signs, and if the image values are assumed to be linear along the edge connecting the corners, the isosurface $G(x, y, z) = 0$ must intersect the edge in a single point somewhere along the edge. The complexity of the surface of intersection is related to the sign changes that occur on all the edges of the voxel.

The heart of the Marching Cubes algorithm is that only a small number of sign combinations is possible, 2 signs at each of 8 corners for a total of 256 combinations. Each combination is analyzed to determine the nature of the isosurface of intersection; a triangle mesh is selected to represent that intersection. These meshes are stored in a table of size 256. The sign analysis for a voxel leads to an index into the table to select a triangle mesh representing the surface of intersection for that voxel. The strength of this algorithm is the speed in which the triangle meshes are generated for the entire isosurface, the performance due to the simplicity of the table lookups.

The Marching Cubes algorithm has two undesirable consequences. The first consequence is that for a typical 3D medical image and typical isosurface, the number of triangles in the mesh is on the order of a million. The generation of the mesh certainly requires only a small amount of time, but most rendering systems are slow to render millions of triangles per frame, even with graphics hardware acceleration. Of course the large number of triangles is a problem with any isosurface extraction algorithm that works at the voxel level. The second undesirable consequence is that the triangle mesh table can lead to topological inconsistencies in the mesh. Specifically, the two meshes generated at adjacent voxels might have triangles that should share edges, but do not, thereby producing holes in the final mesh. How these holes occur will be discussed later in this paper.

One approach that addresses the issue of the large number of triangles is to apply mesh reduction algorithms to the extracted surface [1, 2, 3, 7, 8]. The idea is to extract the triangles at the voxel level, build the triangle mesh using data structures that store the adjacency information for vertices, edges, and triangles, then attempt to reduce triangles according to some heuristic. The algorithm in [2] is based on the concept of an edge collapse where an edge is removed, the triangles sharing the edge are removed, and the remaining triangles affected by the removed triangles are modified to preserve the local topology. Although the reduced meshes are quality representations of the isosurface and can be quickly rendered, the reduction scheme is computationally expensive, thus offsetting the speed of an extraction algorithm such as Marching Cubes. Another approach is to apply methods that construct fewer triangles during the extraction itself. One of these algorithms is adaptive skeleton climbing [5], a multiresolution method that merges voxels into larger rectangular solid regions, each region supporting triangle extraction in the style of Marching Cubes.

In this document I provide an extraction algorithm that has no ambiguities and preserves the topology of the isosurface itself when the image data within each voxel has a continuous representation using trilinear interpolation of the image values at the eight corners. The table lookup of Marching Cubes is replaced by constructing an edge mesh on the voxel faces. That mesh approximates the intersection of the isosurface with the faces. The mesh is then triangulated using an extension of an ear–clipping algorithm for planar polygons [6] to three dimensions. The triangulation does not introduce new points (called Steiner points in the computational geometry literature), something other researchers have tried in attempts to remove the topological ambiguities of Marching Cubes. The triangulation is fast and efficient, but it is also possible to avoid the run–time cost by having a table lookup. The table has 256 entries, just as in Marching Cubes, but each entry that has potential ambiguities stores multiple triangle meshes. Selection of the correct mesh in the table entry is based on a secondary index. The concepts are first discussed for 2D images to give the reader intuition on how the algorithms apply to 3D images.

2 Isocurve Extraction in 2D Images

A 2D image is assumed to be defined on a regular lattice of size $N_0 \times N_1$ with integer points (x, y) where $0 \le x < N_0$ and $0 \le y < N_1$. The image values themselves are $F(x, y)$. An isocurve is of the form $F(x, y) = c$ for some specified level value c where x and y are treated as continuous variables. A pixel in the image is a rectangle whose corners are four neighboring lattice points (x_0, y_0), $(x_0 + 1, y_0)$, $(x_0, y_0 + 1)$, and $(x_0 + 1, y_0 + 1)$. I choose $F(x, y)$ to be a bilinear interpolation of the four image values F_{00}, F_{10}, F_{01}, and F_{11} at the corners, respectively. The continuous representation of the image over the entire pixel is given below where $\delta_x = x - x_0$ and $\delta_y = y - y_0$:

$$F(x, y) = (1 - \delta_y)((1 - \delta_x)F_{00} + \delta_x F_{10}) + \delta_y((1 - \delta_x)F_{01} + \delta_x F_{11}). \tag{1}$$

The equation $F(x, y) = c$ is a quadratic equation in x and y when the xy term appears, a linear equation when xy does not. The isocurves for F when viewed as a function on all of the plane are either hyperbolas or lines. As in 3D, I make the simplifying assumption that the level value c is chosen not to be an image value. The isocurves can intersect interior edge points of a pixel, but cannot intersect the corner points. I also work with $G(x, y) = F(x, y) - c$ and its isocurves generated by $G(x, y) = 0$.

The specialization of Marching Cubes to 2D images is usually referred to as Marching Squares. The isocurve extraction for $G(x, y) = 0$ on a pixel is performed by analyzing the signs of G at the four corners. Since the signs can be only $+1$ or -1, there are 16 possible sign configurations. Figure 1 shows these.

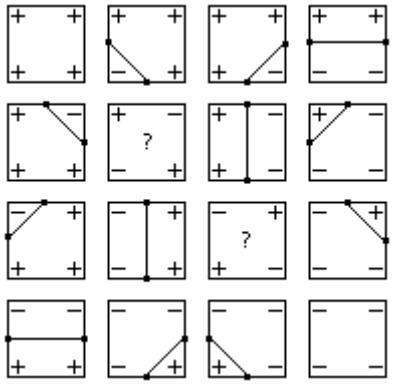

Figure 1. The sixteen possible sign configurations for a pixel.

In the case of sign changes on two edges, clearly we can connect the two edge points with a line segment. The actual isocurve is either a portion of a hyperbola or a line segment. In the first case, the segment connecting the two edge points is a reasonable approximation to the isocurve. In the second case, the segment is exactly the isocurve. Figure 1 shows the approximating segments in the unambiguous cases. However, two cases are ambiguous and are labeled with question marks. The possible resolutions are shown in Figure 2.

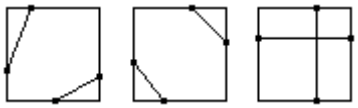

Figure 2. The three possible resolutions for the ambiguous pixel case.

The question is how to select which of the three possibilities in Figure 2 to lead to a mesh that is topologically consistent with the isocurves. The answer is based on an analysis of the quadratic equation $G(x, y) = 0$ to actually determine what the isocurves look like. For simplicity, we may consider the problem when $0 \le x \le 1$ and $0 \le y \le 1$ and where the pixel–image values are $(0, 0, G_{00})$, $(1, 0, G_{10})$, $(0, 1, G_{01})$, and $(1, 1, G_{11})$. The equation is of the form

$$G(x, y) = a_{00} + a_{10}x + a_{01}y + a_{11}xy \tag{2}$$

where $a_{00} = G_{00}$, $a_{10} = G_{10} - G_{00}$, $a_{01} = G_{01} - G_{00}$, and $a_{11} = G_{00} - G_{10} - G_{01} + G_{11}$. Of course the interesting case is when all four edges have sign changes. We may consider the case $G_{00} < 0$, $G_{10} > 0$, $G_{01} > 0$, and $G_{11} < 0$. The opposite signs case has a similar analysis. Notice that $a_{00} < 0$, $a_{10} > 0$, $a_{01} > 0$, and $a_{11} < 0$. The four edge points where $G(x, y) = 0$ are $(0, \bar{y}_0)$, $(1, \bar{y}_1)$, $(\bar{x}_0, 0)$, and $(\bar{x}_1, 0)$. The linear interpolation will show that

$$\bar{x}_0 = \frac{-G_{00}}{G_{10} - G_{00}}, \quad \bar{x}_1 = \frac{-G_{01}}{G_{11} - G_{01}}, \quad \bar{y}_0 = \frac{-G_{00}}{G_{01} - G_{00}}, \quad \bar{y}_1 = \frac{-G_{10}}{G_{11} - G_{10}}. \tag{3}$$

Since $a_{11} \neq 0$, the product $a_{11}G(x, y)$ is not formally zero and can be factored as

$$a_{11}G(x, y) = (a_{00}a_{11} - a_{01}a_{10}) + (a_{01} + a_{11}x)(a_{10} + a_{11}y). \tag{4}$$

Moreover, some algebra will show that $a_{00}a_{11} - a_{01}a_{10} = G_{00}G_{11} - G_{01}G_{10}$. Define $\Delta = G_{00}G_{11} - G_{01}G_{10}$. I consider the two cases when Δ is zero or nonzero.

If $\Delta = 0$, then $a_{11}G(x,y) = (a_{01} + a_{11}x)(a_{10} + a_{11}y)$. The isocurves $G = 0$ occur when $x = -a_{01}/a_{11}$ and $y = -a_{10}/a_{11}$. The isocurves in the entire plane consist of the vertical and horizontal lines defined by the two equations. Thus, the right–most pixel in Figure 2 shows the isocurve structure within the pixel. In this case the line segments forming the plus sign are exactly the isocurves. The center of the plus sign was not found by edge intersections, but is added to the vertex–edge data structure for storing the edge mesh representing the total isocurve for the image. That is, when a plus sign configuration is encountered, we add the four edge intersections and plus–sign center as vertices to the mesh and we add the four segments to the mesh edges that connect the edge intersections with the center point.

If $\Delta \neq 0$, then the isocurves of $G = 0$ are hyperbolas with asymptotes $x = -a_{01}/a_{11}$ and $y = -a_{10}/a_{11}$. The two possible graphs are shown in Figure 3.

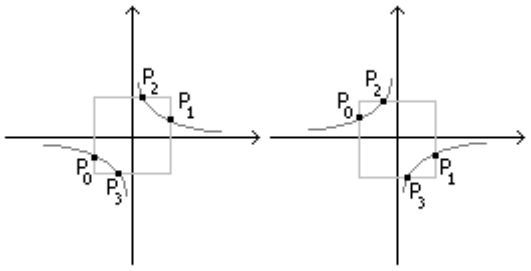

Figure 3. Two possible configurations for hyperbolic isocurves with pixels superimposed. The four edge intersections are P_0, P_1, P_2, and P_3 as marked.

To distinguish which configuration is correct for the given pixel, observe that a pair of edge points is on the same hyperbolic component whenever the signs of the expression $a_{01} + a_{11}x$ are the same at those points. This test follows from the observation that points (x, y) on the vertical asymptote satisfy $a_{01} + a_{11}x = 0$. Points to the right of the vertical asymptote satisfy $a_{01} + a_{11}x > 0$ for the left image in Figure 3 and $a_{01} + a_{11}x < 0$ for the right image in Figure 3. Points to the left of the vertical asymptote have opposite sign, $a_{01} + a_{11}x < 0$ for the left image and $a_{01} + a_{11}x > 0$ for the right image. Let $\sigma(P)$ denote the sign of $a_{01} + a_{11}x$ for point $P = (x, y)$. Some simple computations produce

$$\sigma(P_0) = \text{Sign}(a_{01}) = \text{Sign}(G_{01} - G_{00}) = -\text{Sign}(G_{00}) \tag{5}$$

and

$$\sigma(P_1) = \text{Sign}(a_{01} + a_{11}) = \text{Sign}(G_{11} - G_{10} = \text{Sign}(G_{00}) \tag{6}$$

Now $\sigma(P_2) = \text{Sign}(a_{01} + a_{11}\bar{x}_0)$. Some algebra will show that the argument of the right–hand side is

$$a_{01} + a_{11}\bar{x}_0 = \frac{G_{01}G_{10} - G_{00}G_{11}}{G_{10} - G_{00}}. \tag{7}$$

Therefore,

$$\sigma(P_2) = \text{Sign}(G_{01}G_{10} - G_{00}G_{11})\text{Sign}(G_{10} - G_{00}) = -\text{Sign}(\Delta)\text{Sign}(G_{00}). \tag{8}$$

Similarly, $\sigma(P_3) = \text{Sign}(a_{01} + a_{11}\bar{x}_1)$ where

$$a_{01} + a_{11}\bar{x}_1 = \frac{G_{01}G_{10} - G_{00}G_{11}}{G_{11} - G_{01}}. \tag{9}$$

Therefore,

$$\sigma(P_3) = \text{Sign}(G_{01}G_{10} - G_{00}G_{11})\text{Sign}(G_{10} - G_{00}) = \text{Sign}(\Delta)\text{Sign}(G_{00}). \tag{10}$$

Each of the four signs is computed and the points are grouped into two pairs, each pair having the same sign. Equivalently, we may analyze the signs of $\text{Sign}(G_{00})\sigma(P_i)$ and pair the points accordingly. In this formulation, the modified signs are

$$\begin{aligned} \text{Sign}(G_{00})\sigma(P_0) &= -1 \\ \text{Sign}(G_{00})\sigma(P_1) &= +1 \\ \text{Sign}(G_{00})\sigma(P_2) &= -\text{Sign}(\Delta) \\ \text{Sign}(G_{00})\sigma(P_3) &= +\text{Sign}(\Delta) \end{aligned} \tag{11}$$

Clearly P_0 and P_1 can never be paired just as P_2 and P_3 can never be paired. This should be clear geometrically from Figure 3. We pair (P_0, P_2) and (P_1, P_3) when $\Delta > 0$ or (P_0, P_3) and (P_1, P_2) when $\Delta < 0$.

The following table summarizes all the possible vertex–edge configurations based on analysis of the bilinear function for the pixel. The signs at the four pixels are written from left to right and correspond to the signs of G_{00}, G_{10}, G_{01}, and G_{11} in that order. The sign of Δ is only relevant in the ambiguous cases, so nothing is listed in this column in the unambiguous cases. The names P_0, P_1, P_2, and P_3 always refer to edge points on the edges $x = 0$, $x = 1$, $y = 0$, and $y = 1$, respectively. The center point, if any is labeled C.

signs				sign of Δ	edges
+	+	+	+		
+	+	+	-		$\langle P_0, P_3 \rangle$
+	+	-	+		$\langle P_1, P_3 \rangle$
+	+	-	-		$\langle P_0, P_1 \rangle$
+	-	+	+		$\langle P_1, P_2 \rangle$
				+	$\langle P_0, P_2 \rangle$, $\langle P_1, P_3 \rangle$
+	-	+	-	-	$\langle P_0, P_3 \rangle$, $\langle P_1, P_2 \rangle$
				0	$\langle P_0, C \rangle$, $\langle P_1, C \rangle$, $\langle P_2, C \rangle$, $\langle P_3, C \rangle$
+	-	-	+		$\langle P_2, P_3 \rangle$
+	-	-	-		$\langle P_0, P_2 \rangle$
-	+	+	+		$\langle P_0, P_2 \rangle$
-	+	+	-		$\langle P_2, P_3 \rangle$
				0	$\langle P_0, C \rangle$, $\langle P_1, C \rangle$, $\langle P_2, C \rangle$, $\langle P_3, C \rangle$
-	+	-	+	-	$\langle P_0, P_3 \rangle$, $\langle P_1, P_2 \rangle$
				+	$\langle P_0, P_2 \rangle$, $\langle P_1, P_3 \rangle$
-	+	-	-		$\langle P_1, P_2 \rangle$
-	-	+	+		$\langle P_0, P_1 \rangle$
-	-	+	-		$\langle P_1, P_3 \rangle$
-	-	-	+		$\langle P_0, P_3 \rangle$
-	-	-	-		

Table 1. The vertex–edge configurations for a pixel.

3 Isosurface Extraction in 3D Images

A 3D image is assumed to be defined on a regular lattice of size $N_0 \times N_1 \times N_2$ with integer points (x, y, z) where $0 \le x < N_0$, $0 \le y < N_1$, and $0 \le z < N_2$. The image values themselves are $F(x, y, z)$. An isosurface is of the form $F(x, y, z) = c$ for some specified level value c where x, y, and z are treated as continuous variables. A voxel in the image is a rectangular solid whose corners are eight neighboring lattice points (x_0, y_0, z_0), $(x_0 + 1, y_0, z_0)$, $(x_0, y_0 + 1, z_0)$, $(x_0 + 1, y_0 + 1, z_0)$, $(x_0, y_0, z_0 + 1)$, $(x_0 + 1, y_0, z_0 + 1)$, $(x_0, y_0 + 1, z_0 + 1)$, and $(x_0 + 1, y_0 + 1, z_0 + 1)$. I choose $F(x, y, z)$ to be a trilinear interpolation of the eight image values are F_{000}, F_{100}, F_{010}, F_{110}, F_{001}, F_{101}, F_{011}, and F_{111} at the corners, respectively. The continuous representation of

the image over the entire voxel is given below where $\delta_x = x - x_0$, $\delta_y = y - y_0$, and $\delta_z = z - z_0$:

$$\begin{aligned} F(x,y,z) &= (1-\delta_z)((1-\delta_y)((1-\delta_x)F_{000} + \delta_x F_{100}) + \delta_y((1-\delta_x)F_{010} + \delta_x F_{110})))+ \\ &\quad \delta_z((1-\delta_y)((1-\delta_x)F_{001} + \delta_x F_{101}) + \delta_y((1-\delta_x)F_{011} + \delta_x F_{111}))) \end{aligned} \tag{12}$$

The equation $F(x,y,z) = c$ is a cubic equation in x, y, and z when the xyz term appears, a quadratic equation when xyz does not, and a linear equation when none of xyz, xy, xz, or yz occur. I make the simplifying assumption that the level value c is chosen not to be an image value. The isosurfaces can intersect interior edge points of any of the 12 edges of a voxel, but cannot intersect the corner points. I also work with $G(x,y,z) = F(x,y,z) - c$ and its isosurfaces generated by $G(x,y,z) = 0$.

3.1 Table–Based Mesh Selection

As I mentioned in the introduction, the Marching Cubes algorithm is based on the fact that each corner has an image value that is either positive or negative, leading to 256 possible sign configurations. The corner sign values are used to construct an index into a precomputed table of 256 triangle meshes. I discussed the analogy of this in 2D and showed the ambiguities that arise in two sign configurations. In 2D, rather than having a precomputed table of 16 edge meshes, we needed a secondary index to select one of three edge meshes that can occur in each of the two ambiguious cases. Thus, we have a total of 20 edge meshes to select from. The same ambiguities arise in 3D. In fact, the ambiguities have a more serious consequence: the triangle mesh generated by adjacent voxels can have topological inconsistencies. In particular, when two voxels share a face that is ambiguous in the 2D sense, the table lookup can produce triangle meshes that do not properly share edges on the common face. Figure 4 illustrates this.

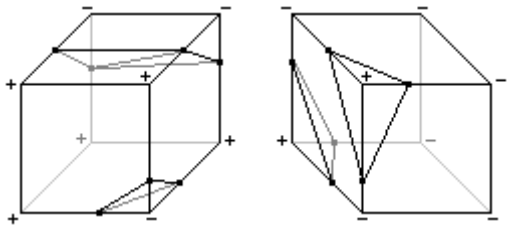

Figure 4. Topological inconsistencies introduced in two voxels sharing an ambiguous face.

The voxel on the right had its edge points on the ambiguous face paired differently than the voxel on the left. This leads to a triangle mesh where a pair of triangles occurs, one triangle from each voxel, but the triangles touch at a single edge point rather than sharing an entire edge. To remedy this, all we need to do is make sure that the pairing of edge points on ambiguous faces occur according to the scheme I constructed for 2D. Interpolating each face bilinearly is consistent with the trilinear interpolation assumed for the entire voxel.

In the 2D setting, I mentioned that the precomputed table of edge meshes have a primary and a secondary index. The primary index takes on 16 values, each value representing a sign configuration for the corners of the pixel. The secondary index is 0 for the nonambiguous cases; that is, if the primary index corresponds to a nonambiguous case, the entry in the table stores a single edge mesh. Assuming the edge meshes in the table entry are stored as an array with zero–based indexing, the secondary index of 0 will always locate the correct (and only) mesh. For the ambiguous case, the secondary index takes on 3 values that represent

whether the quantity Δ I defined earlier is zero, positive, or negative. The table entries for the ambiguous cases have arrays of three edge meshes.

A similar construction can be applied in 3D. However, the table construction can be a bit tedious. An ambiguity for a voxel occurs whenever one or more of its faces is an ambiguous case in 2D. Suppose that exactly one face is ambiguous (for example, Figure 4). Marching Cubes has a single triangle mesh to approximate the isosurface of the voxel. However, the ambiguous face has one of three possible interpretations, so the table entry for this case really needs an array of three triangle meshes. As in 2D, a secondary index can be used to select the correct mesh. Now suppose that exactly two faces are ambiguous. Each face can be resolved in one of three ways, thus leading to nine possible triangle meshes for the table entry of the given primary index. Worst case, of course, is that all six faces are ambiguous, requiring a secondary index that takes on $3^6 = 729$ values. Consequently, the tables will be quite large, but still constructible.

3.2 Ear–Clipping Based Mesh Construction

An alternative to the table lookup is to generate the triangle mesh for each voxel at run time. The concept is quite simple. The edge meshes of a voxel are generated for each face of the voxel. A vertex–edge data structure is used to store the isosurface points on the edges of the voxel and to keep track of which points are paired by an edge. The assumptions that the image is trilinearly interpolated on the voxel and that the level values are not image values guarantees that isosurface points on the voxel edges share exactly two mesh edges. If a plus–sign configuration occurs on the face of a voxel, then the center point of that configuration is added as a vertex of the mesh. That point shares four edges. Thus, a vertex shares either two or four edges. The triangle generation amounts to finding a vertex sharing two edges, locating its two adjacent vertices, adding the triangle formed by those three vertices to a list, then removing the original vertex and the two edges it shares. If necessary, an edge is added to connect the remaining adjacent vertices. This process is repeated until no more vertices exist that share exactly two edges.

I illustrate with an example. Figure 5 shows a voxel and the edge mesh generated by analyzing the six faces using the 2D algorithm.

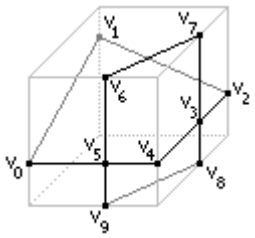

Figure 5. A voxel and its extracted edge mesh.

Vertices V_3 and V_5 are centers of plus–sign configurations and share four edges each. The other vertices share two edges each. Vertex V_0 shares two edges. The adjacent vertices are V_1 and V_5. The triangle $\langle V_5, V_0, V_1 \rangle$ is added to a list. V_0 and its edges to the adjacent vertices are removed from the edge mesh. A new edge is added to connect V_5 and V_1. The image in row 1, column 1 of Figure 6 shows the resulting edge mesh.

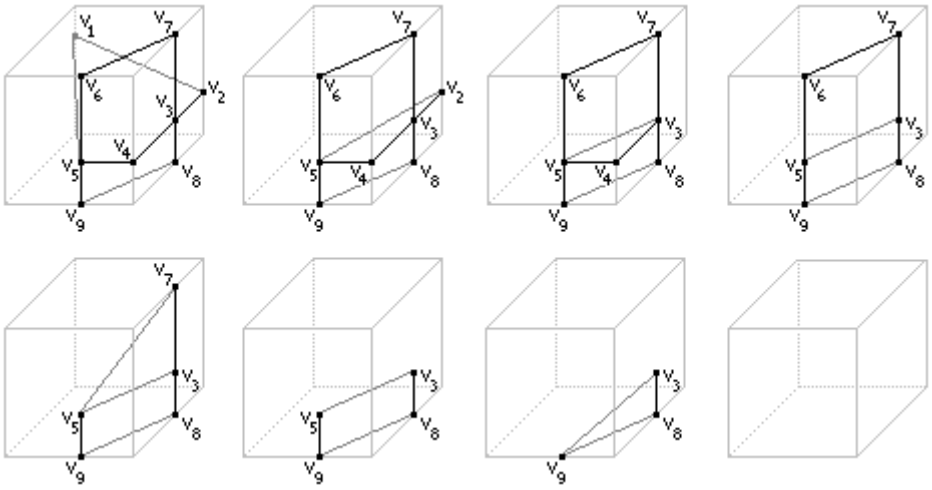

Figure 6. Triangle removal in the edge mesh of Figure 5. The order of the images is top row to bottom row, left to right in each row.

V_1 shares two edges. The adjacent vertices are V_5 and V_2. The triangle $\langle V_5, V_1, V_2\rangle$ is added to the list. V_1 and its edges to the adjacent vertices are removed from the edge mesh. A new edge is added to connect V_5 and V_2. The image in row 1, column 2 of Figure 6 shows the resulting edge mesh.

V_2 shares two edges. The adjacent vertices are V_5 and V_3. The triangle $\langle V_5, V_2, V_3\rangle$ is added to the list. V_2 and its edges to the adjacent vertices are removed from the edge mesh. A new edge is added to connect V_5 and V_3. The image in row 1, column 3 of Figure 6 shows the resulting edge mesh.

V_4 shares two edges. The adjacent vertices are V_5 and V_3. The triangle $\langle V_5, V_4, V_3\rangle$ is added to the list. V_4 and its edges to the adjacent vertices are removed from the edge mesh. An edge already exists between V_5 and V_3, so a new one does not have to be added. The image in row 1, column 4 of Figure 6 shows the resulting edge mesh.

V_6 shares two edges. The adjacent vertices are V_5 and V_7. The triangle $\langle V_5, V_6, V_7\rangle$ is added to the list. V_6 and its edges to the adjacent vertices are removed from the edge mesh. A new edge is added to connect V_5 and V_7. The image in row 2, column 1 of Figure 6 shows the resulting edge mesh.

V_7 shares two edges. The adjacent vertices are V_5 and V_3. The triangle $\langle V_5, V_7, V_3\rangle$ is added to the list. V_7 and its edges to the adjacent vertices are removed from the edge mesh. An edge already exists between V_5 and V_3, so a new one does not have to be added. The image in row 2, column 2, Figure 6 shows the resulting edge mesh.

V_5 shares two edges. The adjacent vertices are V_3 and V_9. The triangle $\langle V_3, V_5, V_9\rangle$ is added to the list. V_5 and its edges to the adjacent vertices are removed from the edge mesh. A new edge is added to connect V_3 and V_9. The image in row 2, column 3 of Figure 6 shows the resulting edge mesh.

Finally, V_3 shares two edges. The adjacent vertices are V_8 and V_9. The triangle $\langle V_8, V_3, V_9\rangle$ is added to the list. V_3 and its edges to the adjacent vertices are removed from the edge mesh. No more vertices exist, so the triangulation is finished. The image in row 2, column 4 of Figure 6 shows the voxel with all vertices and edges removed.

References

[1] Michael J. Dehaemer and Michael J. Zyda, *Simplification of objects rendered by polygonal approximations*, Computer & Graphics, vol. 15, no. 2, pp. 175–184, 1991.

[2] Michael Garland and Paul Heckbert, *Surface simplification using quadric error metrics*, Proceedings of SIGGRAPH 1997, pp. 209–216.

[3] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle, *Mesh optimization*, Proceedings of SIGGRAPH 1993, pp. 19–26.

[4] William E. Lorensen and Harvey Cline, *Marching cubes: A high resolution 3D surface construction algorithm*, Proceedings of SIGGRAPH 1987, pp. 163–169.

[5] Tim Poston, Tien–Tsin Wong, and Pheng–Ann Heng, *Multiresolution isosurface extraction with adaptive skeleton climbing*, Eurographics 98, vol. 17, no. 3, 1998.

[6] Joseph O'Rourke, Computational Geometry in C, 2nd edition, Cambridge University Press, Cambridge, England, 1998

[7] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen, *Decimation of triangle meshes*, Proceedings of SIGGRAPH 1992, pp. 65–70.

[8] Greg Turk, *Re–tiling of polygonal surfaces*, Proceedings of SIGGRAPH 1992, pp. 55–64.