The Wrapper Algorithm for Surface Extraction in Volumetric Data

Andre Guéziec
Courant Institute of Mathematical Sciences and
New York University Medical Center,
New York University
e-mail: gueziec@cs.nyu.edu

Robert Hummel
Courant Institute of Mathematical Sciences and
New York University
e-mail: hummel@cs.nyu.edu

Abstract

Beginning with digitized volumetric data, we wish to rapidly and efficiently extract and represent surfaces defined as isosurfaces in the interpolated data. The Marching Cubes algorithm is a standard approach to this problem. We instead perform a decomposition of each 8-cell associated with a voxel into five tetrahedra, as in the Payne-Toga algorithm. Following the ideas of Kalvin [KCHN91], Thirion *et al.* [TG93], and using essentially the same algorithm as Doi and Koide [KD91], we guarantee the resulting surface representation to be closed and oriented, defined by a valid triangulation of the surface of the body, which in turn is presented as a collection of tetrahedra, some of which are only partly filled. The surface is extracted as a collection of closed triangles, where each triangle is an oriented closed curve contained within a single tetrahedron. The entire surface is "wrapped" by the collection of triangles, using especially efficient data structures. The representation is similar to the homology theory that uses simplices embedded in a manifold to define a closed curve within each tetrahedron.

From the triangles that comprise the wrapping of the surface, we give methods to evaluate surface curvatures and principal directions at each vertex, whenever these

quantities are defined. We further present a fast method for rendering and approximating the surface. The triangles form a graph structure, which is very efficiently encoded, whose nodes are the triangles and whose edges are the common edges joining adjacent triangles. We can thus identify each surface using a connected component labelling algorithm applied to the graph.

This provides a highly parallelizable approach to boundary surface representation, providing an efficiently and compact surface representation. The wrapper algorithm has been used to extract surfaces of the cranium from CT-scans and cortical surfaces from MR-scans at full resolution.

Key words: *B-rep, boundary representation, Marching Cubes, tetrahedral decomposition, homology theory, surface curvature.*

1 Introduction

The Wrapper Algorithm is an efficient, simple, parallelizable method for creating a boundary representation (a "B-rep") of isosurfaces in volumetric data defined on a 3-D grid. The B-rep will consist entirely of planar triangular faces, linked in a graph structure to form a coherent, valid surface. Indeed, the B-rep will necessarily bound a polyhedral solid or multiple polyhedra. The principal advantage of the Wrapper Algorithm over Marching Cubes [LC87] are that (1) the algorithm is parallelizable, and (2) the algorithm provides a provably valid global representation of the voxel space, in all cases. The surface representation provided by the Wrapper Algorithm is relatively uniform, being composed of triangular faces, and we show that vertices in the representation have degree no higher than nine (i.e., no more than nine triangles ever meet at a single vertex). The Wrapper Algorithm uses a tetrahedral decomposition of the voxel space, and thus operates at a fine scale of detail, which provides the provable correctness of the representation, but also provides a voluminous data structure. In order to reduce the quantity of data in the representation, we have also developed simplification algorithms that operate on the B-rep provided by the Wrapper Algorithm. We display results obtained from the simplification process, but the details of the algorithms are described elsewhere.

The details of the Wrapper Algorithm coincide, to a large extent, with the algorithm of Doi and Koide, of IBM Japan, Ltd., described in [DK91]. Doi and Koide use the same tetrahedral decomposition, produce oriented cycles to represent triangular faces of the isosurface, and even use the same determinant test to establish the orientation of the cycles.

Since their work predates ours, a large part of our contribution is to explain the Doi/Koide algorithm, and to draw attention to its advantages. Our work also adds the following features. We provide a homology-theoretical basis for the Wrapper Algorithm, describing the representation in terms of boundary operators on singular chains of simplices. We also produce a B-rep in the form of a graph structure, and discuss the use of parallel connected component algorithms for graph analysis. We show that the orientation of cycles in the surface representation may be computed by a table look-up procedure, which has relatively few cases due to a particular organization of the construction. We define and use particularly efficient data structures for the implementation of the algorithm. We provide a bound on the degree of vertices in the representation produced by the algorithm. Our approximate rendering algorithm is new. Finally, our simplification methods (not described in detail here) are different and more aggressive than the methods described by Doi and Koide.

Apart from the Doi/Koide work, there is a large body of literature on methods of constructing B-reps of isosurfaces in an efficient and accurate fashion. The Wrapper Algorithm (and the equivalent Doi/Koide algorithm) provide a natural extension, and (we believe) improvement on other methods. While problems with the Marching Cubes algorithm have largely been addressed and fixed [NH91], the Wrapper Algorithm, with its appeal to homology theory for provable correctness, provides a much simpler and intuitive approach to surface representation. The Wrapper Algorithm also provides an orientation of the surface without additional cost. Although the Wrapper Algorithm operates at a fine scale and provides a more detailed representation, which can be disadvantageous, it provides a more uniform platform for representation simplification, and can be viewed as a decomposition of the Marching Cubes method (and other methods) to a finer scale so as to make all the special cases and difficult surface topologies within a single voxel break up into small collections of simple cases.

Starting with a 3-D image, i.e., a regular grid of volume elements representing the sampling of an intensity function $I(x, y, z)$, we examine the problem of constructing a polygonal approximation to the surfaces characterized by $I = I_0$, a constant value. By convention, we choose $I_0 = 0$ in the subsequent discussion. This problem is also known as *implicit surface tiling*, for $I(x, y, z) = 0$ is an implicit equation for a surface in three-space, or as *isosurface construction*, since an approximation to an iso-level surface is generated.

Cline and Lorenson [LC87] developed the "Marching Cubes" algorithm, which parti-

tions the space into cubical elements composed of eight neighboring voxel values. Inside each of these cells, a decision is made as to whether the surface intersects the cell, in which case polygons approximating this intersection are constructed. The original algorithm exploited a symmetry between situations involving positive and negative voxel values. It turns out that some of the polygonal representations so obtained are not valid, i.e., there are "holes" in the surface. The basic reason for this is that some configurations of positive and negative voxel values are ambiguous. For example, on a square face of a cube, when the diagonally opposite vertices are positive and the other two negative, two different polygonalizations are possible (see Fig. 1).

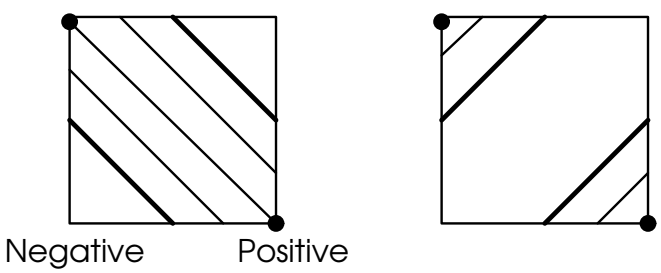


Figure 1: Ambiguous configuration. The hash-marked regions represent two conflicting polygonalizations of the underlying shape

Koi, Doi, and Kajioka [KDK86] propose the decomposition of the 8-cells into five tetrahedra. They enumerate all possible cases, depending on the sign of the different tetrahedra vertices, and create either zero, one, or two triangles inside each tetrahedron with a table look-up procedure. They applied this technique to the visualization of complex molecules.

The same decomposition was made famous by Payne and Toga [PT90], applied to the representation of brain structures. An advantage of the tetrahedra-based method over a cubical-cell based method is that with standard interpolation schemes, there are no ambiguous cases for the polygonalization of the surface. Hall and Warren [HW90] use an adaptive subdivision of the volume into tetrahedral cells, in order to approximate implicit surfaces defined with a continuous function $f(x, y, z) = 0$. They concentrate on the problem created by triangles with a high aspect ratio, and propose a solution consisting in relocating some of the surface vertices.

Several methods have been proposed to cope with ambiguous configurations, without creating holes. Wallin [Wal91] suggests using an interpolated voxel value at the center of the face, in order to choose one polygonalization or the other, as in the case of Figure 1. The polygons that are so defined can have up to 12 edges within each cell, and are thus not trivially triangulated. As pointed out in the recent survey by Ning and Bloomenthal [NB93], some triangulations might create invalid edges in the cube, and it is advisable to add the centroid of polygons that have more than five edges to the set of vertices.

Kalvin [Kal91] developed another method for disambiguation by selecting a preferred polarity. By using 6-connectivity within the positive-valued volume and 26-connectivity in the exterior, he observes that positive vertices cannot then be connected along diagonals. Thus in Figure 1, Kalvin will always choose the second polygonalization. He proves that the surfaces that are produced are valid, and he builds a "winged edge" data structure [Bau74] to represent and manipulate the surfaces, building them up sequentially, by applying a succession of surface construction operators. The method developed by Kalvin is related to the "Weaving Wall" algorithm of Harlyn Baker [Bak89], which is used to analyze image sequences.

Thirion [TG93] defines cycles in each voxel, similarly to Wallin, and shows that the cycles can be oriented. Further, he introduces oriented segments inside the cycles and a new method, "Marching Lines," to track characteristic lines on isosurfaces. The observation by Monga *et al.* [MBF92], that it is possible to compute surface curvatures from the discrete differentiation of voxel values, plays a central role in the work of Thirion. Marching Lines has been applied to the problem of registering 3-D medical images [AGTG93].

2 Description of the Wrapper Algorithm

2.1 Tetrahedral decomposition

We regard voxels as being values defined at points of a rectangular lattice, and eight adjacent voxels are taken to form the eight vertices of an 8-cell, or cube. We use the tetrahedral decomposition made famous by Payne and Toga [PT90] (see Figure 2).

For any given cube, two such tetrahedral decompositions are possible, one which is mirror symmetric with respect to the y-z plane to the other (see Figure 12). In order to be consistent between neighboring 8-cells, i.e., in order that faces and edges of tetrahedra

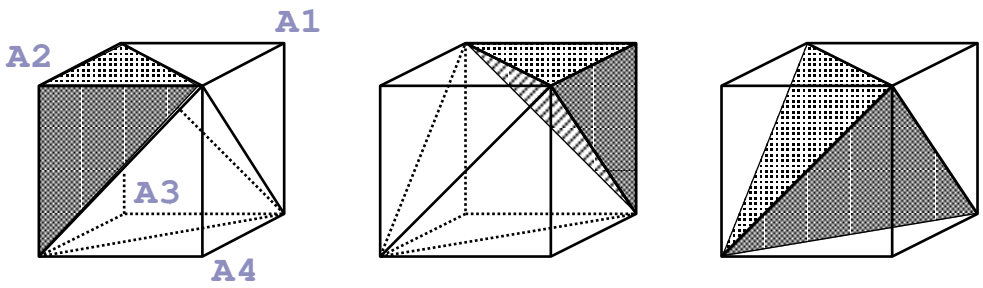


Figure 2: Decomposition of the cube into five tetrahedra. Four are right isoscele and thus isomorphic (only two are shown here on the left), the fifth one is regular and equilateral (right).

in one cell match faces and edges of tetrahedra in the neighboring cells, we must alternate between the two decompositions from cell to cell, in a 3-D checkerboard fashion.

We next describe the use of binary operations to perform the decomposition easily. Each of the vertices of the 8-cell is numbered from 0 to 7, as in Figure 3. Moving along an edge oriented along the x, y, or z axis is performed by inverting one of the three bits of the representation. The three possible 1-bit inversions will be denoted by $\otimes 001$, $\otimes 010$, and $\otimes 100$ respectively (see Figure 4).

Each 8-cell has a coordinate location (x, y, z) for its local origin. We use x, y, and z to measure the row, column, and height in the array of cells. In order to identify each tetrahedron within a cell, we perform the following steps:

1. If $x + y + z$ is even, we call the cell an *even cell*, and we say that the parity of the cell is even. If the sum is odd, the cell's parity is odd. To determine tetrahedron number 1 within the cell, we select apex A_1 as vertex 000 in an even cell, and as vertex 001 in an odd cell. We then obtain three other vertices from A_1 by applying the motion operators $\otimes 001$, $\otimes 010$, and $\otimes 100$, resulting in v_{11}, v_{12}, and v_{13} respectively. Tetrahedron number 1 is spanned by $(A_1, v_{11}, v_{12}, v_{13})$, which we view as an ordered tuple of vertices (see Figure 5).

2. Tetrahedron number 2 uses the second apex A_2, obtained from A_1 by the 2-bit inversion $\otimes 011$. Three vertices v_{21}, v_{22}, v_{23} are obtained from A_2 by the one-bit

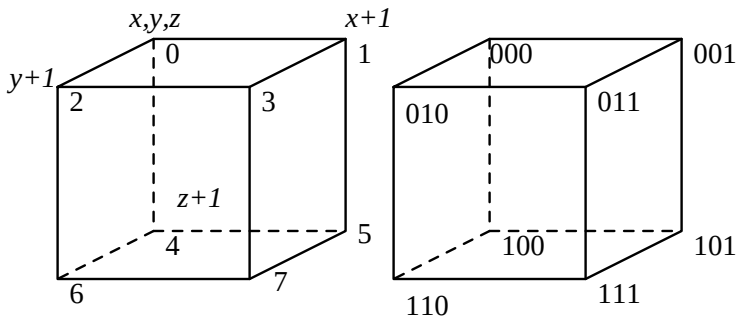


Figure 3: Binary transitions to traverse the 8-cell. Each vertex is labeled by a bit pattern corresponding to its coordinates in an (x, y, z) coordinate system, relative to a standard origin in the cell (upper left rear in the figure).

transitions $\otimes 001$, $\otimes 010$, and $\otimes 100$.

3. Next, A_3 is obtained by applying $\otimes 101$ to A_1. The span of $(A_3, v_{31}, v_{32}, v_{33})$ defines tetrahedron number 3, where the v_{31}, v_{32}, v_{33} once again come from A_3 after applying $\otimes 001$, $\otimes 010$, $\otimes 100$.

4. By applying $\otimes 110$ to A_1, we define A_4, and the corresponding tetrahedron 4 is defined analogously to tetrahedra 1, 2, and 3.

5. The fifth tetrahedron is defined by shifting the four apices (A_1, A_2, A_3, A_4) by $\otimes 001$, resulting in $(A_1 \otimes 001,\ A_2 \otimes 001,\ A_3 \otimes 001,\ A_4 \otimes 001)$, which spans tetrahedron number 5.

2.2 Intersection of surfaces with tetrahedra

The next step consists in determining whether a portion of the isosurface will intersect a given tetrahedron (v_1, v_2, v_3, v_4). The intensity values (I_1, I_2, I_3, I_4) corresponding to the four vertices are retrieved from the 3-D data. (In the case of CT data from X-ray scans, the vertex values are called Hounsfield numbers.) Supposing $I < I_0$ at a vertex, we will assume that the vertex lies outside the body bounded by the isosurface. If $I \geq I_0$, then the vertex lies inside the body. If the vertices of a tetrahedron are of mixed sign relative

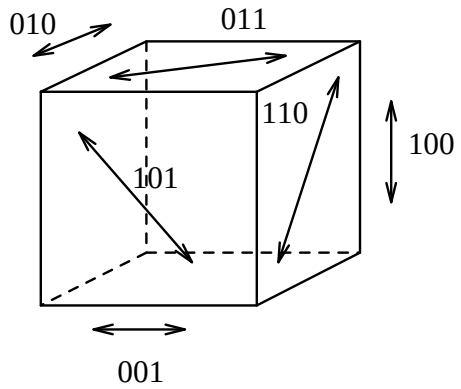


Figure 4: Some motion operators.

to I_0, then the isosurface will intersect the tetrahedron. For convenience, we assume that $I_0 = 0$. Our first task is to determine the points of intersection of $I = 0$ along the edges of the tetrahedron. The location of these points depends on the interpolation function that is used.

For each edge that exhibits an intensity sign change, we will create a vertex of the polygonal approximation to the isosurface $I = 0$. The exact position of the vertex is determined by the zero-crossing of a function interpolating intensity values along the edge. Our experience shows that using linear interpolation along each edge yields a poor result, with an excessive spikiness of the surface, as illustrated in Figure 6.

We instead choose a bilinear function that interpolates the four intensity values at the corner of each face in the original volumetric grid. This allows us to evaluate values on the faces of the 8-cells; we will never need to explicitly evaluate values internal to the 8-cells. The bilinear interpolation reduces to linear interpolation along the edges of an 8-cell, but results in a quadratic interpolation along diagonal edges on an 8-cell face. Specifically, if (a, b, c, d) denote the four intensity values on the face of an 8-cell, then the intensity value along the (a, d) diagonal edge is given by:

$$I(u) = (a + d - b - c)u^2 + (-2a + b + c)u + a.$$

Here, u varies from 0 to 1 linearly along the diagonal. Provided $ad < 0$, I will have exactly one zero in the range $0 < u < 1$, which can easily be determined from the quadratic formula (the other zero will fall outside this range).

Having established the interpolation function along tetrahedral boundaries, we can thus

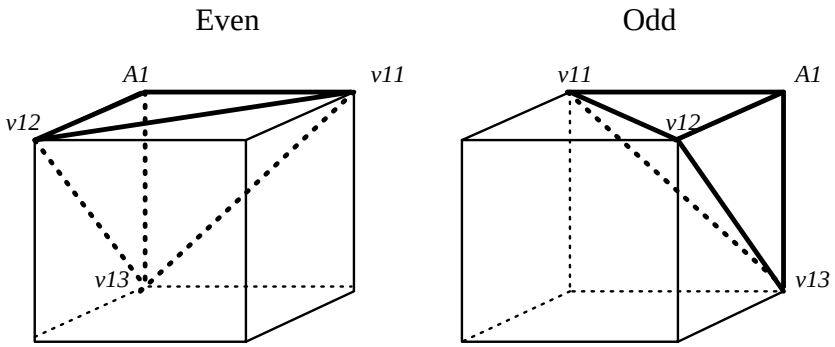


Figure 5: To determine tetrahedron number 1 within the cell, we select apex A_1 as vertex 000 in an even cell and as vertex 001 in an odd cell. We then obtain three other vertices from A_1 by applying the motion operators $\otimes 001$, $\otimes 010$ and $\otimes 100$.

compute the locations of the intersections of the isosurface with those edges. The exact shape of the isosurface within the tetrahedron will depend on the volumetric interpolation function. We will, however, not be concerned with this interpolation function and will instead approximate the surface using either a planar triangle or a pair of triangles, both lying inside the tetrahedron.

Consider a tetrahedron (v_1, v_2, v_3, v_4) which is defined by the ordered tuple of vertices spanning the volume. The order is as determined by one of the five steps from Section 2.1. The corresponding intensity values are given by a four-tuple (I_1, I_2, I_3, I_4). As noted before, if all four values have the same sign, then the surface does not intersect the tetrahedron. If the signs are mixed, however, we then have three major cases. Among I_1, I_2, I_3, I_4, there are either one, two, or three positive values. These cases are illustrated in Figure 7. For the Cases I and III, three of the values have the same sign. For Case II, two vertices are positive and two are negative. In this case, the surface will intersect all four faces, and we have a quadrilateral. By choosing arbitrarily a diagonal of the quadrilateral, we obtain two triangles within the tetrahedron. Combining all cases, we have a patch of the surface represented as either one or two triangles.

More importantly, the triangles can be oriented. An orientation for a triangle is given by a cycle, or an ordering of the edges, such that viewed from the outside, the triangle is traversed in a counterclockwise direction. We next explain how to determine the proper ordering in an efficient manner.

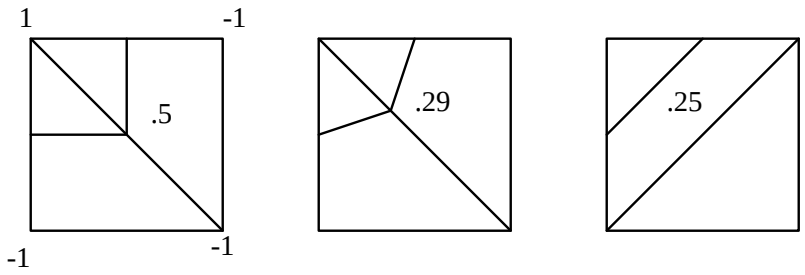


Figure 6: Using a linear interpolation along the diagonal edge results in a severe difference in position for the polygonal surface when the diagonal edge is swapped (left and right). Consequently, the polygonal surface represents a "spiky" aspect. Instead, we use a *bilinear* interpolant (middle) to reduce this difference.

2.2.1 Cases I and III

For Cases I and III, exactly one value among (I_1, I_2, I_3, I_4) has sign opposite to the others. Using this value, we compute intersections along the edges connecting its vertex to the other three vertices, preserving order among (v_1, v_2, v_3, v_4). So, for example, if I_3 has the different sign, we compute intersection along (v_3, v_1), then (v_3, v_2), and then (v_3, v_4). These three intersections determine an ordering which is either the correct direction, or opposite to the correct direction. One way to determine if the orientation is correct is to check it with the following simple procedure.

First, we reorder the vertices (v_1, v_2, v_3, v_4) to obtain $(\alpha, \beta, \gamma, \delta)$ such that the vertices with negative values precede the vertices with positive values, without otherwise disturbing relative order. Viewing the vertices as coordinates in three-space, we can then compute the determinant

$$\det(\beta - \alpha, \gamma - \beta, \delta - \gamma).$$

If this determinant is negative, then the ordering of the triangle vertices is correct; otherwise, the ordering must be reversed. (Doi and Koide use this same determinant procedure.)

To see that this procedure works, consider Case I, where exactly one vertex (name δ) will have a positive value. The order of the cycle defining the triangle is, by definition, the intersection points along (δ, α), then (δ, β), and finally (δ, γ), which provides a cycle whose orientation, relative to δ, is the same as the orientation of (α, β, γ). The orientation of this cycle, pointing outwards, is given by the vector cross-product $(\beta - \alpha) \times (\gamma - \beta)$, which

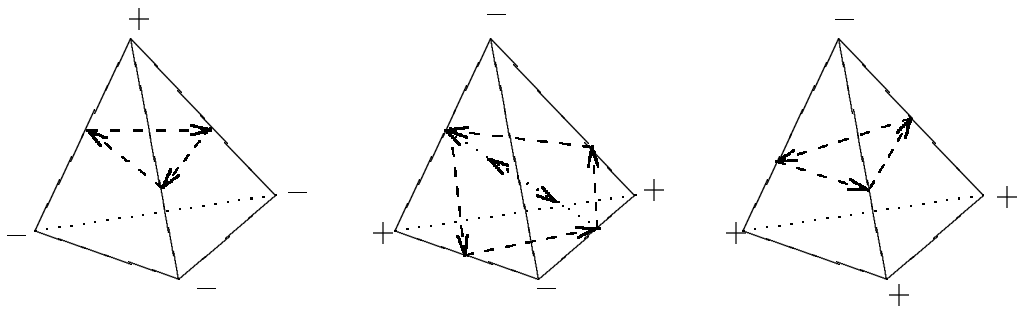


Figure 7: Defining one or two oriented triangles. Depending upon the number of vertices outside the surface (marked with the minus sign) we define three or four vertices of the surface approximation in the ordering specified in the text. the condition for having to reverse the ordering is $\det(\beta-\alpha, \gamma-\beta, \delta-\gamma) > 0$. Note that this value is six times the *volume*, positive or negative, of the tetrahedron.

points away from the interior point δ according to whether the dot product with $\delta - \gamma$ is positive or negative. The dot product is precisely the determinant $\det(\beta-\alpha, \gamma-\beta, \delta-\gamma)$, and since the correct ordering gives an orientation pointing to the exterior, a negative determinant indicates a correct ordering, whereas a positive determinant indicates that the ordering must be reversed. Equivalently, the frame at node δ given by the triple of vectors $(\alpha-\delta, \beta-\delta, \gamma-\delta)$ should form a right-hand coordinate frame if α, β, γ are oriented clockwise with respect to the exterior node δ. This means that the ordering is correct if and only if $\det(\alpha-\delta, \beta-\delta, \gamma-\delta)$ is positive, which can be seen to be equivalent (through determinant manipulations) to requiring that $\det(\beta-\alpha, \gamma-\beta, \delta-\gamma)$ is negative.

2.2.2 Case II

In this case, exactly two vertices are negative, and two are positive. We reorder the vertices (v_1, v_2, v_3, v_4) as above, to give vertices $(\alpha, \beta, \gamma, \delta)$, where the values associated with α and β are negative, and the values associated with γ and δ are positive. We compute the vertices of the quadrilateral as follows. We first find the zero along (α, γ). Next, we find the zero along (α, δ), then (β, δ), and finally (β, γ). This sequence of four points establishes a cycle, which is either the correct ordering, or the incorrect ordering, which can be easily

checked.

It turns out that in this case, the verification procedure is exactly the same as in the previous case. That is, viewing the vertices as vectors in $\Re^3$, we compute $\det(\beta - \alpha, \gamma - \beta, \delta - \gamma)$. The ordering of the interpolation points defined above is correct if the determinant is negative; if the determinant is positive, then the order should be reversed.

If the resulting ordering of the interpolation points is given by (w_1, w_2, w_3, w_4), then there are two possible triangulation: One is given by the following pair of ordered triangles $\{(w_1, w_2, w_4), (w_2, w_3, w_4)\}$, and the other is given by the pair $\{(w_1, w_2, w_3), (w_3, w_4, w_1)\}$. The triangulations may be chosen arbitrarily. By always choosing the first one in an even 8-cell, and the second one in an odd 8-cell, we constrain the maximum degree at vertices to be nine.

The justification of the verification procedure is once again done by examining carefully Figure 7, and using the fact that the vertices α and β have negative values. Since α, β, γ, and δ are the coordinate locations of 8-cell vertices, and can be represented relative to the local origin, all matrix entries in the determinant calculation are either 0, 1, or -1.

2.2.3 A look-up procedure to replace the determinant test

The sign of the determinant value $\det(\beta - \alpha, \gamma - \beta, \delta - \gamma)$ that specifies whether the orientation of a given cycle is correct can be obtained by a table look-up procedure, as opposed to a computation. This new procedure greatly speeds up the orientation determination.

Let us first suppose that for a given tetrahedron (v_1, v_2, v_3, v_4) in an even 8-cell, no re-ordering is required to obtain $(\alpha, \beta, \gamma, \delta)$. It then turns out, based on the construction of the ordered tuple representing the tetrahedron, that $\det(\beta - \alpha, \gamma - \beta, \delta - \gamma)$ will be positive, regardless of whether (v_1, v_2, v_3, v_4) is a tetrahedron number 1, 2, 3, 4, or 5, as long as we are inside an even 8-cell (For an odd cell, the result is always negative.) Next, we suppose that some re-ordering is required in determining $(\alpha, \beta, \gamma, \delta)$. Recall that vertices with negative values are listed first, without changing other relative orderings. Suppose, for example, that v_2 and v_1 must be exchanged to obtain $(\alpha, \beta, \gamma, \delta)$. The result of this permutation is that the sign of the determinant is reversed. If instead v_3 must be brought to the front, then two transpositions are required, and the sign of the determinant stays the same.

In general, the sign of the determinant $\det(\beta - \alpha, \gamma - \beta, \delta - \gamma)$ is determined by the number of transpositions required to permute the negative vertices among (v_1, v_2, v_3, v_4) to

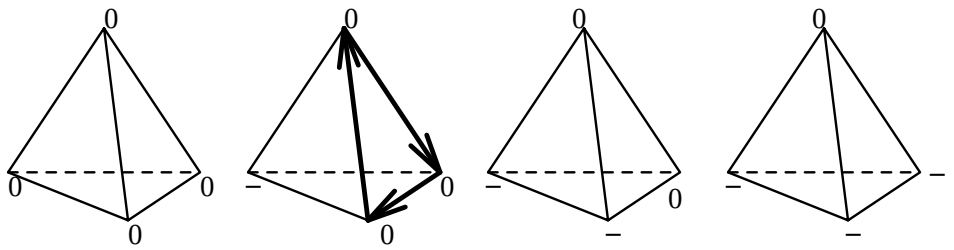
the beginning of the list. The number of transpositions required can be pre-tabulated, as follows. Representing a negative vertex in (v_1, v_2, v_3, v_4) by a 0, and a positive vertex by a 1, we obtain a 4-bit code for the tetrahedron, yielding a number between 0 and 15. The resulting determinant will be positive or negative according to whether an even or odd number of transpositions are required, as tabulated in Table 1. Note that it is no longer required to physically re-order the vertices—only the bit code is needed. Accordingly, if the table yields a positive entry for the determinant, the oriented cycle representing the surface patch should be reversed; a negative entry indicates that the orientation is correct. The signs are opposite for an odd cell.

2.3 Special Cases

We next consider certain special cases, when $I = 0$ at some or all of the positive vertices. (Recall that $I \geq 0$ corresponds to a positive vertex.) Note that when intensity values are integer values as in medical imaging, the property $I \neq I_0$ can always be guaranteed trivially by taking a non-integer value for I_0. However, when $I = I_0 = 0$ is possible, a single vertex $I = 0$ within a tetrahedron is handled exactly as any positive vertex. Indeed, we treat a vertex with $I = 0$ exactly as any other positive vertex except for two special cases: (i) If there is one vertex with $I = 0$, and the other three are strictly negative, then we do not create a triangle, even though we are in a Case I situation; (ii) Also, if there are two vertices with $I = 0$, and the other two vertices are strictly negative, then we do not create polygons, even though we have a Case II situation. In these two cases, the polygons are degenerate. On the other hand, if three values among (I_1, I_2, I_3, I_4) are zero and the other is negative, then we create one triangle, as shown in Figure 8, exactly as a Case III situation would predict. Also, contrary to [KDK86], when $I_1 = I_2 = I_3 = I_4 = 0$, we do not create any polygons, instead viewing this situation as four positive vertices lying inside the body.

2.4 A bound on the degree of a vertex

The *degree* of a vertex is the number of triangles that share that vertex. High degrees are associated with long and thin triangles, i.e., triangles with high aspect ratios. We must assume that $I(x, y, z) \neq 0$ at all vertices. Then all intersections must occur along the interior of edge segments. At most six tetrahedra can share a single edge, and if two


0
0
0

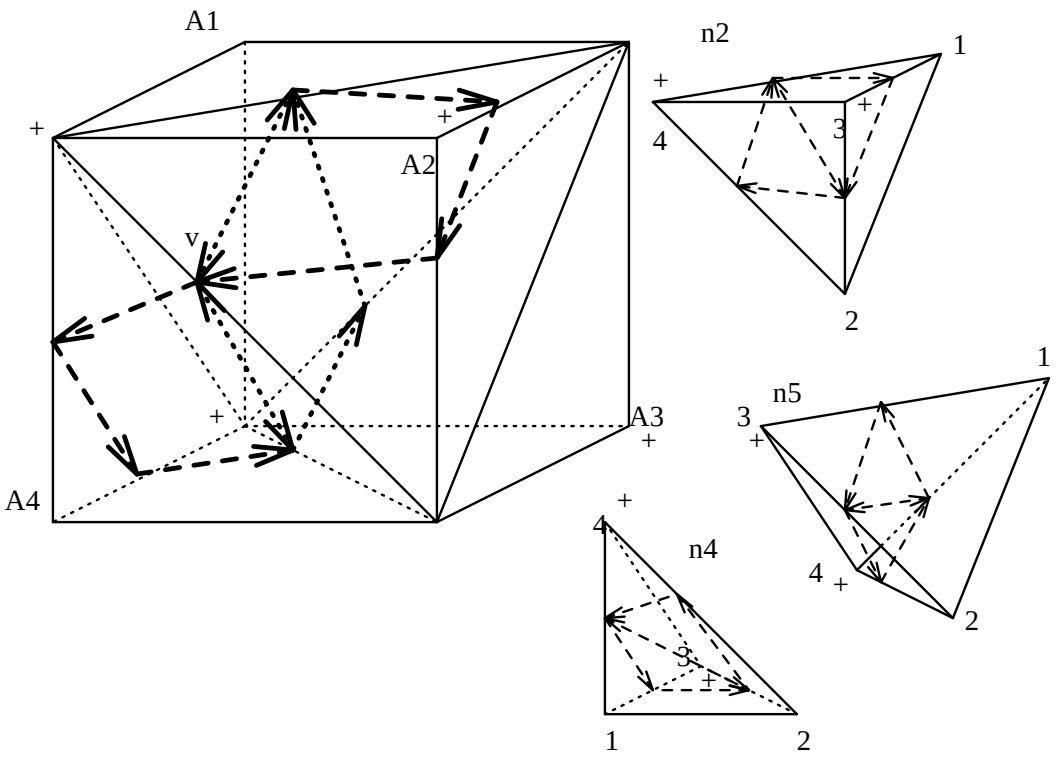


Figure 9: Highest degree of the triangulation. When six tetrahedra, each containing two surface triangles share a common surface vertex v, it is impossible that more that nine triangles meet a v. (Note that the + sign at apex A_3 is not necessary for the discussion, but illustrates a case when an 8-cell contains ten triangles. It is easily seen that four such cases are possible.)

2.5.1 The surface graph

The surface representation consists of a collection of "surface patches," and each patch can be represented as a single triangle or a pair of triangles. Further, each patch has either three or four neighbors. We can represent the collection of patches (the nodes of the abstract graph structure representing the surface) as a set of tetrahedra indices, which we denote by $t = (x, y, z, k)$. The (x, y, z) coordinates indicate the location of the 8-cell containing the tetrahedron, and $1 \leq k \leq 5$ indicates the tetrahedron number, as defined in Section 2.1. The neighbors of the patches can be encoded by a simple code c with five possibilities. Either there are four neighbors, in which case every neighbor of the

tetrahedron contains a contiguous patch, or there are three neighbors, and one of the four faces of the tetrahedron is omitted from the list of neighbors. We use $c = 0$ in the first case, and $c = 1, 2, 3$, or 4 to indicate which face is omitted in the latter case. The c-th face of a tetrahedron number k is the face opposite vertex $v_{k,c}$, as defined in Section 2.1. Accordingly, the abstract graph structure of the patches can be encoded very compactly, as a set of tetrahedra and their codes $\{(t_i, c_i)\}$.

Each triangle is represented by a sequence of three vertices. However, each vertex will be shared by more than one triangle (a maximum of ten triangles when the vertex is interior to an edge). Each triangle should use a pointer structure to encode the locations of the vertices, so as to avoid multiply encoding the positions of the vertices. One method is to store three pointers with each triangle, pointing into a heap of vertices. In this representation, we have a set of vertices $\{v_j\}$, where each v_j is a location in three-space, and each tetrahedron t_i is supplemented with either three indices $(j_{i,1}, j_{i,2}, j_{i,3})$, indicating a triangle in t_i, or in case $c_i = 0$, a list of four indices $(j_{i,1}, j_{i,2}, j_{i,3}, j_{i,4})$.

2.5.2 Distributed vertex allocation

For parallel computing and improved cache performance, it is desirable to store the vertex information local to the tetrahedra. In this section, we show how to replace the heap of vertices with an allocation of vertices to individual tetrahedra that are used in the surface representation.

For this purpose, and for other purposes as well (as in Table 1), it is useful to replace the neighbor code c_i with a bit-code b_i representing the signs of the voxel data at the vertices of tetrahedron t_i. This bit-code requires four bits: $b_i = (b_{i,1}, b_{i,2}, b_{i,3}, b_{i,4})$, and each bit $b_{i,m}$ equals 1 if the value at the m-th vertex of tetrahedron t_i has greater value than I_0, and equals zero otherwise. Although the bitcode b_i requires four bits while c_i only requires three bits, the extra information will be useful, and a simple table can be used to recover c_i from b_i.

Armed with the tetrahedron index t_i and the bitcode b_i, it is easy to determine the edges of the tetrahedron that will contain vertices of the surface triangulation. Indeed, the tetrahedron index is irrelevant, except that the oriented list can be produced by incorporating the test of Section 2.2.3, which may be precomputed for each case, and which depends, as it turns out, only on the parity of the 8-cell containing t_i (i.e., on whether $x_i + y_i + z_i$ is even or odd) and on the bit code b_i.

The result is that for every tetrahedron t_i in the surface representation, we can produce an ordered list of three or four tetrahedral edges, each of which will contain exactly one vertex of the surface triangulation. We now describe how the information about those vertices may be stored so that (1) each surface vertex is represented in only one location, and (2) the information about a vertex along a tetrahedral edge is allocated to a tetrahedron that contains that edge.

In fact, we describe two such allocation schemes. The first, the *static allocation scheme*, stores either zero, one, or two surface vertices in any given triangle. The scheme is illustrated by Figure 10. First, edges are allocated to either the even 8-cells or the odd 8-cells, as indicated by edges with dots in Figure 10. Note that in a checkerboard concatenation of even and odd 8-cells, all edges of all tetrahedra are allocated. Next, the six edges in the even 8-cells are distributed among the five tetrahedra of the 8-cell, and the six edges of the odd 8-cells are similarly distributed among the five corresponding tetrahedra. In each case, one tetrahedron has responsibility for two edges. Also note that each edge is allocated to an adjoining tetrahedron. Using this scheme, every edge of every tetrahedron has a fixed storage location in an adjoining tetrahedron in order to store the location of a vertex that might occur on that edge. Indeed, the value that is stored can be a simple real value, giving a relative distance along the edge, in a predetermined direction. Then, given a tetrahedron t_i and the bitcode b_i, we can use the parity of the cell and the bitcode in order to obtain a list of the storage locations of the values representing the vertices of the surface patch, in order. Each element in the list can be represented as a *differential tetrahedral number*, which is a code $(\Delta x, \Delta y, \Delta z, k')$ giving the relative cell location, and the actual tetrahedron number in the cell. In the case of the tetrahedron containing two storage locations, an extra bit is required to point to one of the two values.

It is perhaps surprising that we can distribute the six edges in each cell among the five tetrahedra in a dynamic allocation scheme in such a way that no tetrahedron ever contains more than one vertex. In this scheme, we use six different edges in each 8-cell, as shown in Figure 11. The key to this scheme is that the allocation of edges depends on the bitcode of the tetrahedron number 4 in the odd cell. The retrieval algorithm works as follows. In the figure, all solid dots indicate that the vertex information for that edge is unambiguously located in the indicated tetrahedron, within the corresponding 8-cell. For the starred nodes, vertex information for the edge may be stored in the indicated tetrahedron in the even 8-cell, which is where an access is first checked. The tetrahedron information contains

not only a vertex value (a real value giving a distance along the edge), but also a couple of bits indicating which edge is actually stored there. If the starred edge discovers that its information is not located in its primary storage tetrahedron, it then accesses its secondary storage location. The secondary storage for edges are shown with open circles, and they are always contained in the odd 8-cell. The open circles are associated, one-for-one, with the starred edges. Note that these dynamic edges have their primary storage tetrahedra in the even cell, and a secondary storage location in an adjoining tetrahedron in the odd 8-cell.

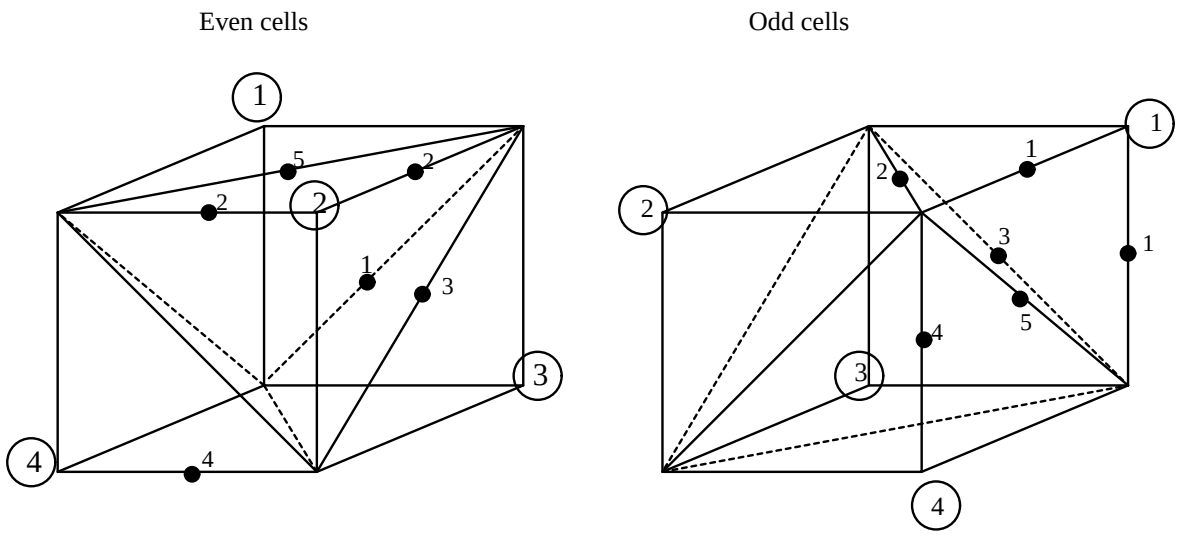


Figure 10: Static storage scheme. Each edge is allocated either to the adjoining even cell or odd cell, as indicated by the dots. Within each cell, the tetrahedra are numbered 1 to 5; the apices of the first four tetrahedra are shown by the circled numbers, and the fifth tetrahedron is the interior one, in both cells. The labels on the dots give the tetrahedron number in the corresponding cell to which the edge is allocated.

It is possible to store the information in such a way that all surface vertex data is either located in the primary or secondary storage location, and that there are never conflicts in the secondary storage locations. There are only three dynamic edges in any given even 8-cell, and edges in the odd cells are all unambiguous. Let us denote the three edges in the even 8-cell as e_1, e_2, and e_3 as shown in Figure 11. (Edge e_1 is the diagonal edge on the negative x-face of the even cell, e_2 is the edge on the negative y-face, and e_3 is on the

negative z face.) Consider first e_1. Vertex information along this edge is either stored in the primary location (tetrahedron 4 in the even 8-cell), or the secondary location (tetrahedron 4 in the adjoining odd cell). The determination depends on the bitcode b of the tetrahedron 4 in the odd cell (i.e., the tetrahedron that is the secondary storage location). Similarly, edge e_2 and edge e_3 are allocated to their primary or secondary storage location according to the bitcode of the adjoining tetrahedron forming the secondary storage location. Table 2 shows whether the primary storage location (with entry value 1) or the secondary storage location (table entry 2) should be used, for edges e_1, e_2, and e_3, as a function of the 16 possible bitcodes for the secondary tetrahedron.

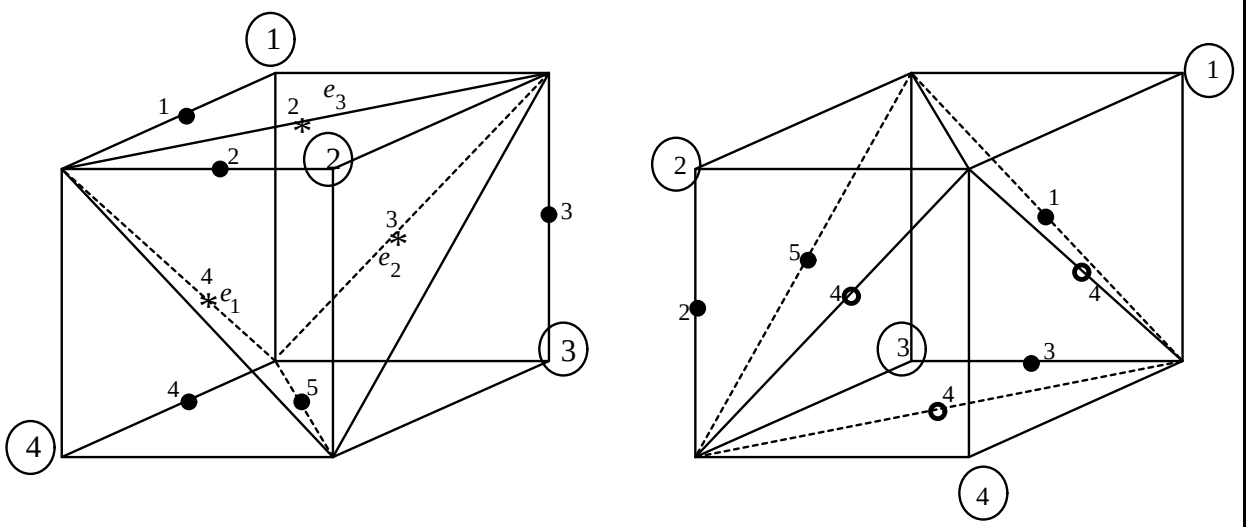


Figure 11: Dynamic storage scheme. Solid dots indicate deterministic allocation, as in Figure 10. For the three starred edges, the primary storage location is the labeled tetrahedron in the even cell, and the secondary storage location is given by the label on the open dot in the odd cell. Table 2 indicates when primary and secondary storage locations are used.

3 Operations on the Wrapper output

In this section, we consider a number of operations that may be applied to the data structures produced by the wrapper algorithm. These operations are particularly efficient due to the representation of the surface as produced by the Wrapper algorithm.

3.1 Connected Components

Connected components of the boundary surface are connected closed surfaces. For example, a sphere or torus will have one surface component, whereas a 3-D annulus will have two components. Similarly, two distinct solid spheres will result in two components. More than counting components, a connected component analysis of the surface is important for analyzing the topology or measuring volumes and surface areas.

The output of the wrapper algorithm is a graph data structure. The nodes are represented by tetrahedra indices (x, y, z, k), and each node contains a code that can be converted, through the use of a simple table, to a list of differential indices to neighboring tetrahedra that contain contiguous surface patches. The table depends only on whether the 8-cell is even or odd, the tetrahedron type $1 \leq k \leq 5$, and the code c, $0 \leq c \leq 4$, indicating which face, if any, is omitted among the list of neighbors (Recall that the code c can be determined from a table look-up from the bitcode b for the signs of the vertices of the tetrahedron.) Indeed, we may begin with a list (see Table 3 and Figure 12) of the four neighboring tetrahedra for each type of tetrahedron.

Nearly all connected components algorithms will begin by choosing a single pointer at each node. By ordering the tetrahedra lexicographically, we may choose the maximum neighbor for each surface patch. These pointers may be pretabulated for each case, depending on the parity of the 8-cell, the tetrahedron number, and the omitted neighbor code. The table has 50 entries.

A standard connected components algorithm is based on the Union/Find disjoint set operators (see [CLR89]). A more interesting alternative is to use the $O(\log n)$ iterative parallel connected components algorithm of Shiloach/Vishkin, or a MIMD version of this algorithm [Hum86].

3.2 Approximate Rendering

Suppose that we perturb the voxel values at the vertices of tetrahedra of types I, II, and III by setting each positive vertex value to zero. This perturbation only affects tetrahedra that contain surface patches. Due to the change of values near the surface, the polygonal surface structure will be perturbed. However, no surface patch will move outside the boundaries of its enclosing tetrahedron. The advantage of this modification of voxel values is that all surface patches will now lie on tetrahedra faces, and only tetrahedra of type III