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 partitions 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).

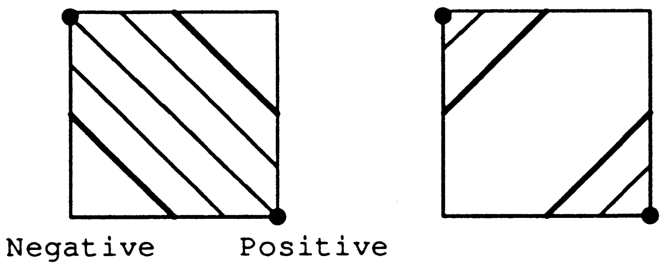

Figure 1: Ambiguous configuration. The hash-marked regions represent two conflicting polygonalizations of the underlying shape

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

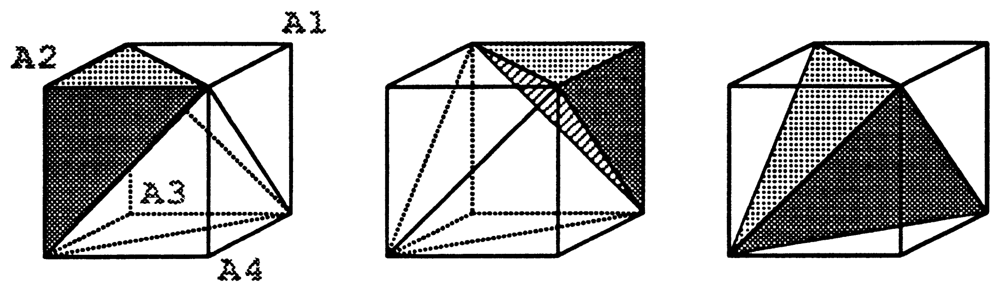

Figure 2: Decomposition of the cube into five tetrahedra. Four are right isoscele and thus isomorphic (only two are shown here on the left), the fifth one is regular and equilateral (right).

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

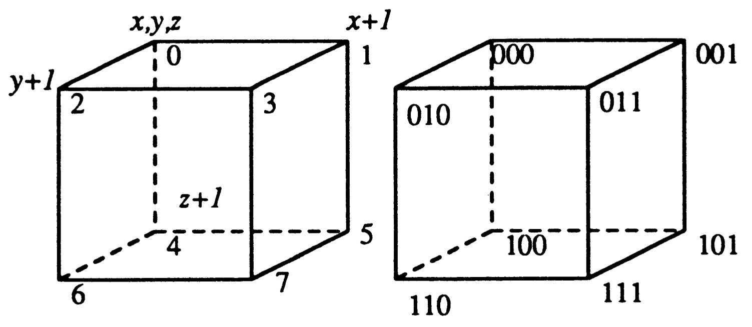

Figure 3: Binary transitions to traverse the 8-cell. Each vertex is labeled by a bit pattern corresponding to its coordinates in an (x, y, z) coordinate system, relative to a standard origin in the cell (upper left rear in the figure).

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

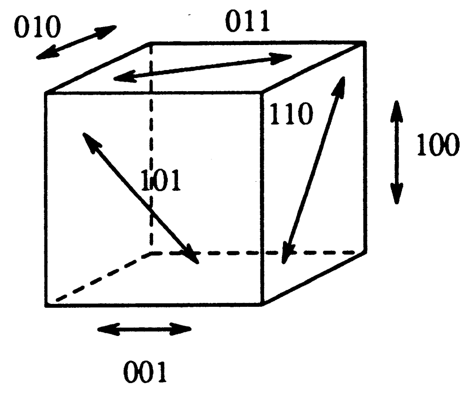

Figure 4: Some motion operators.

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

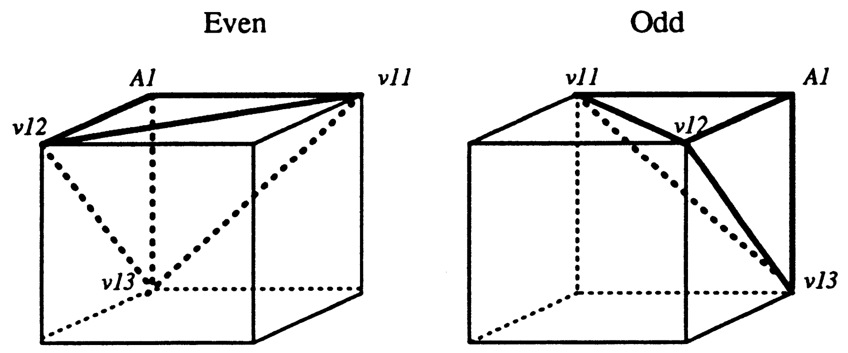

Figure 5: To determine tetrahedron number 1 within the cell, we select apex A_1 as vertex 000 in an even cell and as vertex 001 in an odd cell. We then obtain three other vertices from A_1 by applying the motion operators $\otimes 001$, $\otimes 010$ and $\otimes 100$.

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.

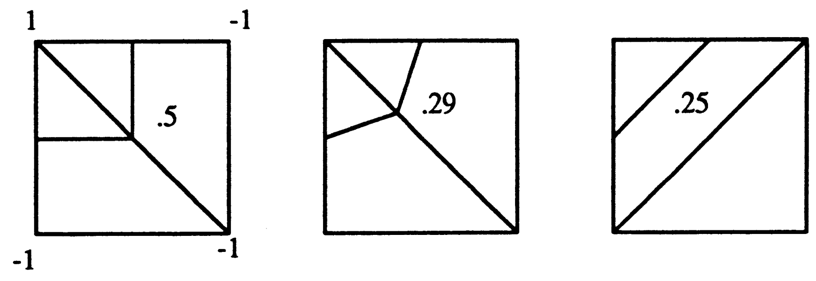

Figure 6: Using a linear interpolation along the diagonal edge results in a severe difference in position for the polygonal surface when the diagonal edge is swapped (left and right). Consequently, the polygonal surface represents a "spiky" aspect. Instead, we use a *bilinear* interpolant (middle) to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

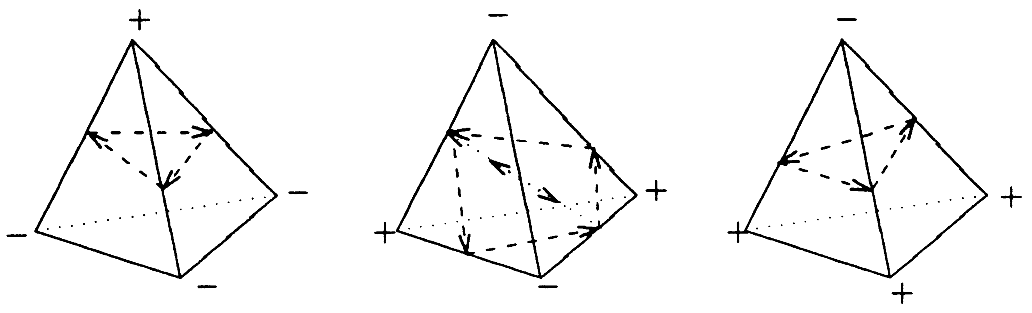

Figure 7: Defining one or two oriented triangles. Depending upon the number of vertices outside the surface (marked with the minus sign) we define three or four vertices of the surface approximation in the ordering specified in the text. the condition for having to reverse the ordering is $\det(\beta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

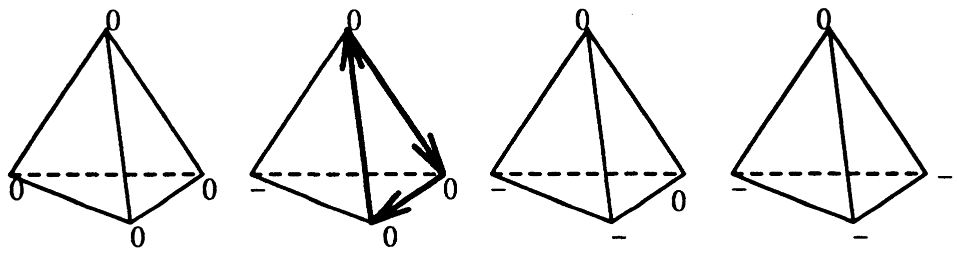

Figure 8: Special cases occurring when the largest voxel value is zero. We insert a polygon (namely, a triangle) when exactly one value is strictly negative, but if exactly two are negative, (and the other two are zero), or if exactly three are negative (and the other one is zero), no polygons are inserted.

triangles are contained in each tetrahedron and all meet at a single vertex, then conceivably 12 triangles could meet at a single location. In reality, the maximum number that can meet is nine.

To prove that the bound is actually nine, one must examine the geometry of the situation under the supposition that ten or more triangles meet at a point v on an edge. Since there must be at least five tetrahedra coincident at v, one quickly determines that the edge containing the vertex v must lie diagonally on a face between two 8-cells, and that six tetrahedra share the edge. Among the six tetrahedra, at least four must be of type II, and of these four, at least two must be adjacent (i.e., share a face). Each type II tetrahedron will contain a surface patch represented as a quadrilateral consisting of an ordered set of interpolation points (w_1, w_2, w_3, w_4), with one of the points being the common vertex v, and each quadrilateral is split into two triangles, such that all eight triangles must meet at v. However, diagonals are defined by choosing the pair $\{(w_1, w_2, w_4), (w_2, w_3, w_4)\}$ in an even 8-cell, and the alternative pair in an odd 8-cell. As a result, it turns out that no two diagonals in adjacent tetrahedra can meet at the same vertex. (The proof of this fact depends on the orientation of tetrahedra and the convention for the selection of the diagonals in odd and even cells.) Since at least two of the four tetrahedra must be adjacent, we have a contradiction.

2.5 Surface Data Structure

We have completed the description of the Wrapper Algorithm. However, the form of the data structures that are used to implement the algorithm can have a substantial influence on the execution speed.

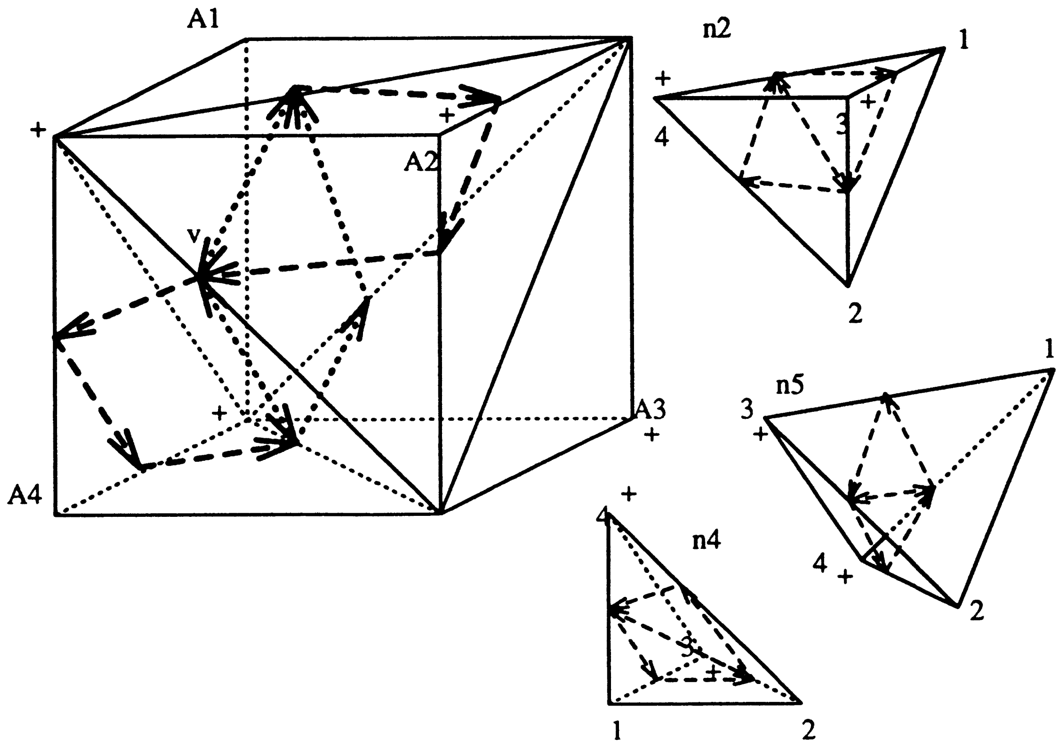

Figure 9: Highest degree of the triangulation. When six tetrahedra, each containing two surface triangles share a common surface vertex v, it is impossible that more that nine triangles meet a v. (Note that the $+$ sign at apex A_3 is not necessary for the discussion, but illustrates a case when an 8-cell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

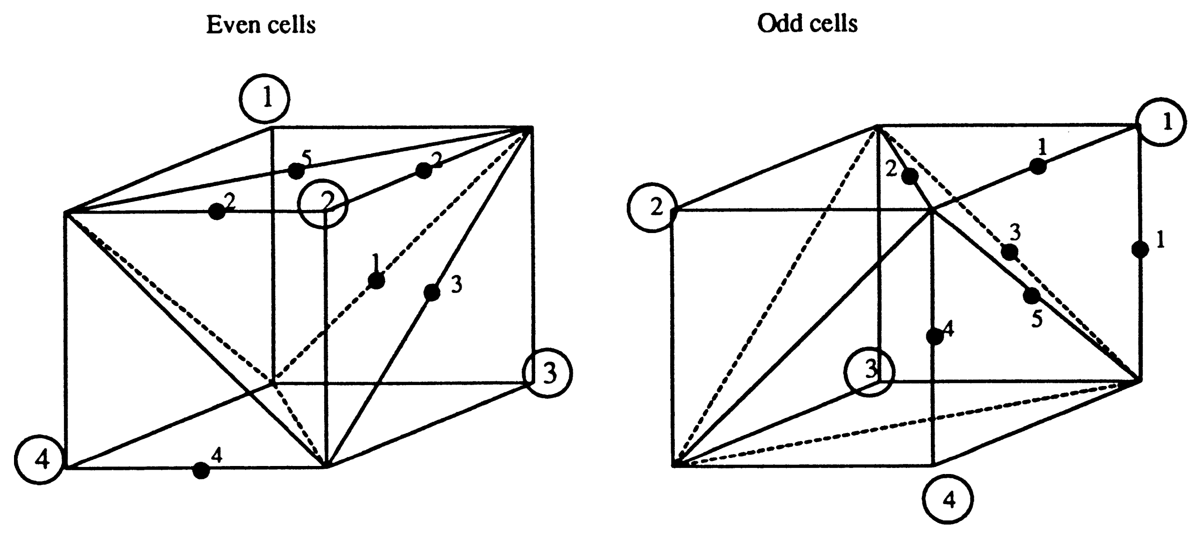

Figure 10: Static storage scheme. Each edge is allocated either to the adjoining even cell or odd cell, as indicated by the dots. Within each cell, the tetrahedra are numbered 1 to 5; the apices of the first four tetrahedra are shown by the circled numbers, and the fifth tetrahedron is the interior one,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

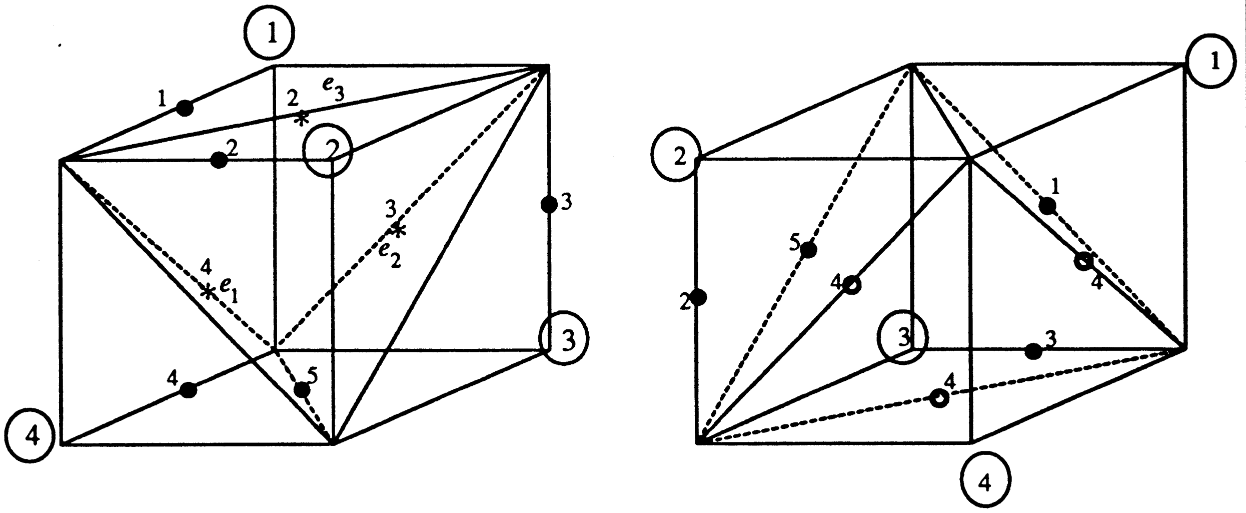

Figure 11: Dynamic storage scheme. Solid dots indicate deterministic allocation, as in Figure 10. For the three starred edges, the primary storage location is the labeled tetrahedron in the even cell, and the secondary storage location is given by the label on the open dot in the odd cell. Table 2 indicates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

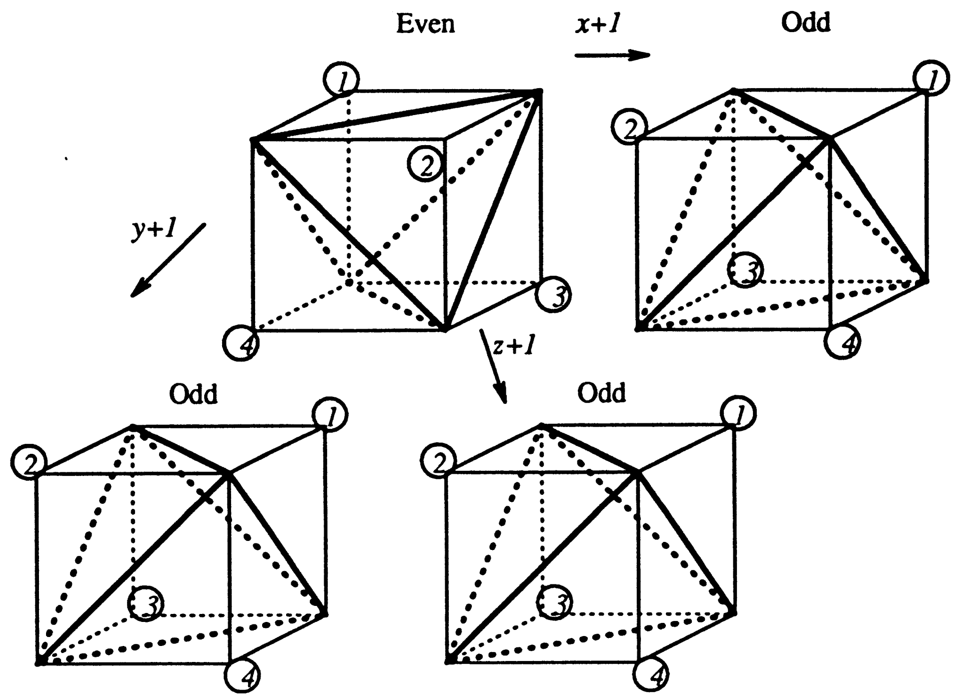

Figure 12: The correspondence between codes of neighboring tetrahedra. An "even 8-cell" (top left), where A_1 is vertex 000, is adjacent to six "odd 8-cells", where A_1 is vertex 001. The four neighbors of each of the tetrahedra will be found among these seven 8-cells.

are of interest, since tetrahedra of type I and II with zero values at the positive vertices are special cases, as discussed in Section 2.3. Recall that each type III tetrahedron has a single negative vertex (which is outside the volume), separated by a triangular patch from the tetrahedral face containing the three positive vertices, which we call the *outface*. By redefining these positive vertices to be zero, the triangular patch becomes the entire face of the type III tetrahedron.

In order to perform a rapid rendering of this approximate surface structure, we can use the following algorithm. We first locate all type III tetrahedra in the original volume, and using the tetrahedron index consisting of the (x, y, z) coordinates of the origin of the 8-cell as well as the tetrahedron number k $(1 \leq k \leq 5)$, we determine the number of the vertex opposite the outface $(1 \leq v \leq 4)$. This vertex can either be computed from the tetrahedron index and the bit code representing the vertex signs, or it can be stored in type III tetrahedra concurrent with the Wrapper Algorithm. Then, using the tetrahedron number k and the vertex v, it is easy to generate the coordinates of an oriented cycle giving the vertices of the outface. For rendering the face, a better result is obtained if the face is assumed to have an orientation (i.e., surface normal) different than its actual direction. The phantom orientation can be taken from the orientation of the triangular surface patch in the type III tetrahedron before the perturbation, or by smooth interpolation of an orientation at each vertex obtained by averaging the surface normals of the adjoining surface elements, or by using a direction obtained from a local evaluation of the gradient of the voxel data. Figure 13 shows a result using the last method.

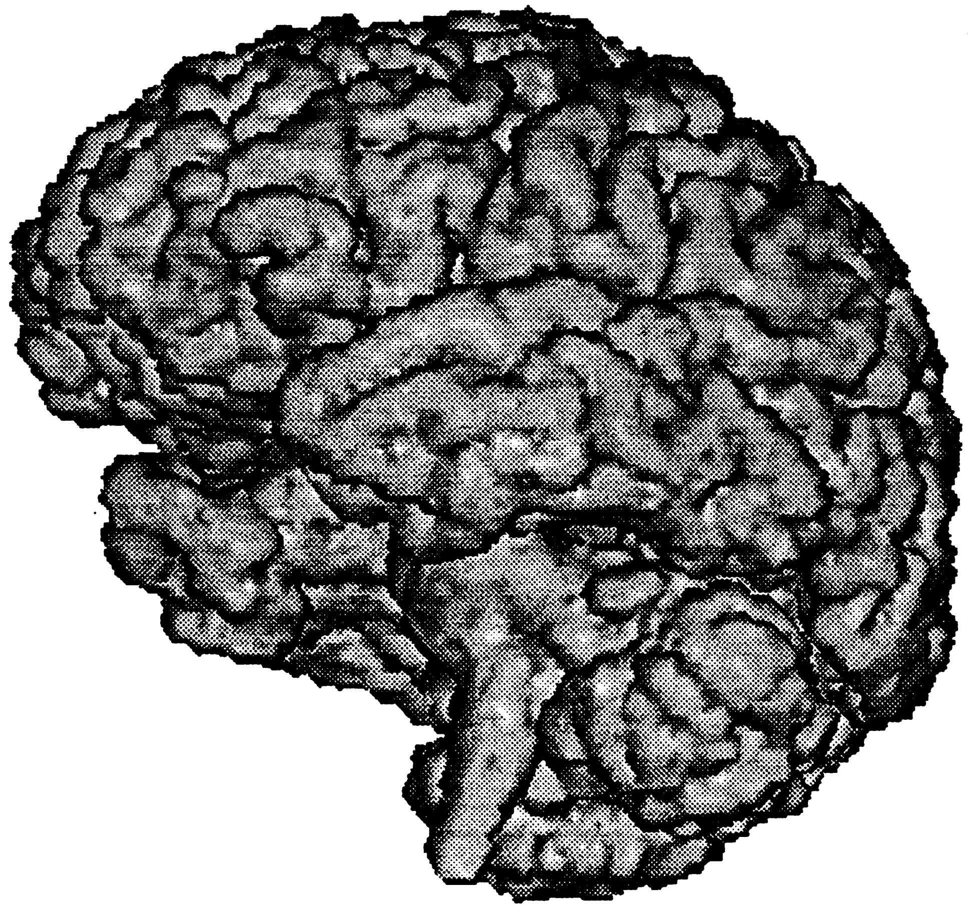

Figure 13: Approximate rendering of the cortical surface of the brain for a normal individual (104K type III tetrahedra), extracted from a 256 by 256 by 130 MR scan (courtesy of H. Rusinek, Department of Radiology, NYU School of Medicine). The surface normals are colinear with the gradient of voxel data. Prior to rendering, the segmentation of the volume uses techniques described in [MAB93].

3.3 Surface curvatures

Monga *et al.* [MBF] provide a method for computing principal curvature directions and curvature values of an isosurface directly from voxel data and spatial derivative of the voxel data (see also [Gué93]). Specifically, at a point (x_0, y_0, z_0), the normal vector $\mathbf{N}$ of the surface S defined by $I(x,y,z) \equiv I(x_0,y_0,z_0)$ lies in the same direction as the gradient of voxel data $\nabla I = (I_x, I_y, I_z)$: $\mathbf{N} = \nabla I/||\nabla I||$.

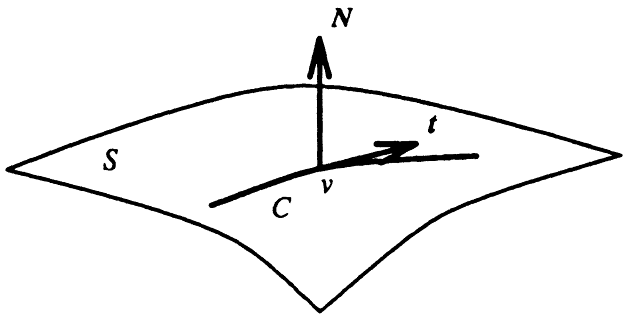

Figure 14: Normal N of the isosurface at vertex v.

In order to evaluate the surface curvatures, we consider a curve C, with tangent vector $\mathbf{t}$, along a normal section of S (see Figure 16). The normal of C is also $\mathbf{N} = (n_x, n_y, n_z)$. The differentiation of $\nabla I \cdot \mathbf{t} = \mathbf{0}$ yields

$$\mathbf{t}^t H \mathbf{t} + \nabla I \cdot k\mathbf{N} = 0$$

where k denotes the normal curvature of C which is also by definition the surface curvature in direction $\mathbf{t}$, and H is the three by three Hessian:

$$H = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{pmatrix}.$$

Thus $k = -\mathbf{t}^t H \mathbf{t}/||\nabla I||$. Accordingly, the principal curvatures are the negatives of the eigenvalues of $P^T R^T H R P$ divided by the magnitude of the gradient $||\nabla I||$, where P is the projection of (x,y,z) onto (x,y) and is a three by two matrix:

$$P = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix},$$

and R is a rotation matrix composed from $R = R_1 R_2$, with

$$R_1 = \frac{1}{\sqrt{1-n_z^2}} \begin{pmatrix} n_x & -n_y & 0 \\ n_y & n_x & 0 \\ 0 & 0 & 1 \end{pmatrix}, R_2 = \begin{pmatrix} n_z & 0 & \sqrt{1-n_z^2} \\ 0 & 1 & 0 \\ -\sqrt{1-n_z^2} & 0 & n_z \end{pmatrix},$$

and R_1 is the identity if $n_z = \pm 1$. All vertices of surface patches occur along edges of 8-cells, and derivatives of the image data may be computed and interpolated at these locations by suitable methods. Solving the eigenproblem at the interpolated position of each surface vertex, we may attach curvatures (and also principal curvature directions) to each vertex stored in the surface representation. Interpolation of values into interior points of surface patches can be based on weighted sums of values at triangle vertices.

3.4 Surface simplification

The Wrapper Algorithm, based on a tetrahedral decomposition, results in significantly more triangles than the Marching Cubes method. If we count the number of vertices created by both methods, and assuming that all vertices land on the interior of edge segments, we find that the Wrapper Algorithm results in between two and three times more vertices. Experimental results for this ratio are presented in Table 4 for a variety of surface types.

In addition, the representation of the surface data has an impact on the volume of data. Alternative representations include the Baumgart winged-edge data structure [Bau74], and face or edge lists at each vertex. In most of our implementations, we choose to use a heap of vertex coordinates, with edge lists for each triangle represented through pointers into the collection of vertices. The representation that stores vertex information local to the tetrahedra, as presented in Section 2.3, is especially compact.

In order to reduce the number of triangles, and to eliminate badly shaped triangles, we have developed a simplification algorithm that we outline below. Details will be given in a subsequent report.

The simplification algorithm operates serially, successively visiting edges and testing them for removal. This approach is contrary to the methods in [SZL92,Tur92], but related to the method in [RR94]. A parallelization of the process is possible, but edges that are concurrently considered for deletion must be sufficiently distant in the graph data structure.

Figure 15 shows the configuration for an edge deletion test, where (v_1, v_2) denotes the edge that is under test. If the edge is deleted, then v_1 and v_2 are modified to terminate at v, which we call a simplified vertex.

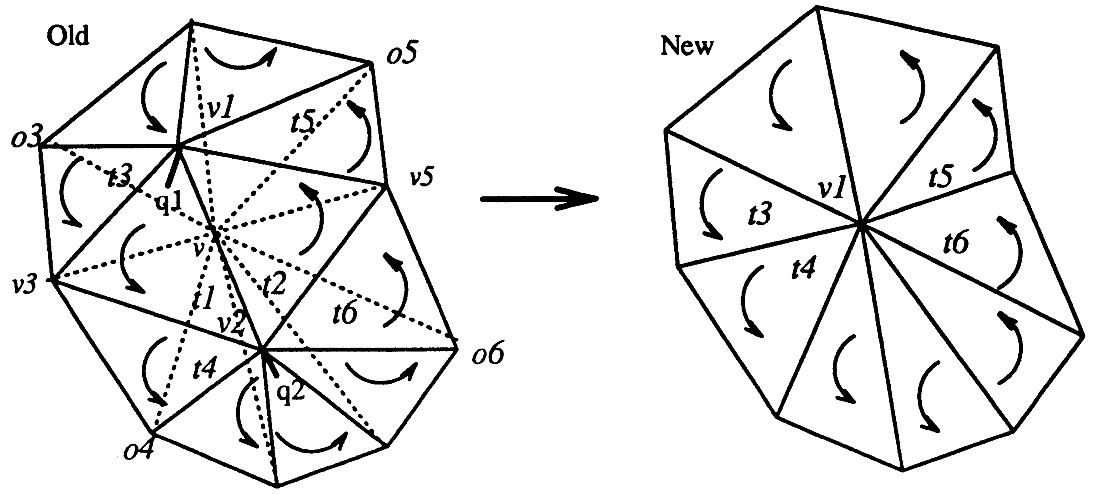

Figure 15: Removing the edge shared by triangles t_1 and t_2.

The criterion for deleting an edge can include an assortment of tests. If v_3 and v_5 denote vertices of the two triangles that contain the edge (v_1, v_2) as in Figure 15, then we demand that the distance between v_1 and v_2 be less than the distance from v_3 to v_5. Also, we might require that the former distance be sufficiently small. The Euclidean distances q_1 and q_2 between v_1 and v_2 and the new triangulated surfaces are computed. These distances are obtained by projecting v_1 and v_2 onto the closest triangle or edge of the surface, and should be small in order to pass the deletion test. The smallest angle in the new configuration of triangles is also computed and compared to the smallest angle in the old configuration. Again, we want that the new configuration improves on the old.

We maintain three positive real values d, e, and p at each simplified vertex v (see Figure 16). The value e is an upper bound to the closest distance between v and the set of triangles in the original polygonal surface, and d represents an upper bound to the distance between a triangle t that meets v and the vertices of the original surface patch (see Fig. 16). If an immediate neighbor w of v has been simplified more recently, then the value d at w takes precedence over d at v. Finally, we record in p the maximum value of q_1 and q_2, as defined above. The value of p is used in updating d and e during the simplification process, and simplification terminates, when all values of d or e exceed a threshold.

Use of this simplification algorithm, which is incorporated into the visualizations of Figures 18, 19, 20 and 21, typically provides 10 to 1 compression ratios in the surface representation without noticeable degradation in surface fidelity. The thresholds may change dynamically, and the simplification can terminate when we obtain a given compression ratio, or when we reach at each vertex a maximum error bound.

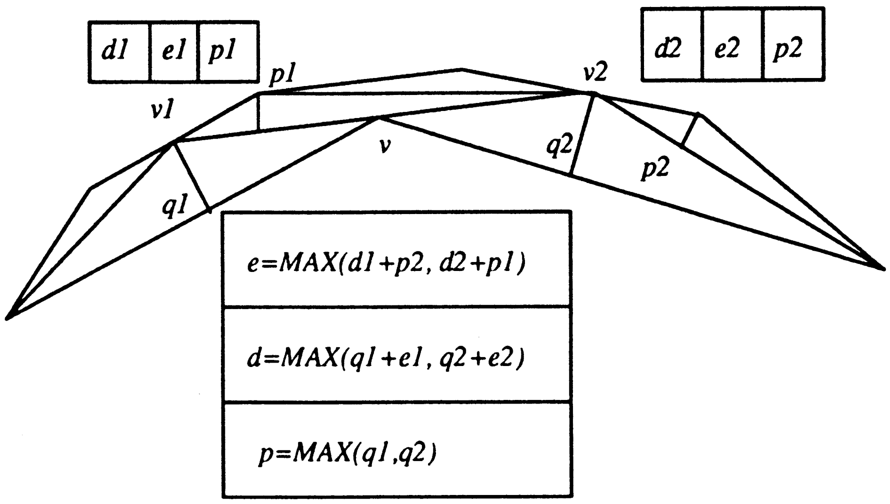

Figure 16: We associate to each simplified vertex v a maximum bound e for the distance between v and the original surface. Then d is the maximum distance between the triangles meeting at v and the original surface vertices. The value p reflects the latest simplification error when collapsing v_1 and v_2 into v.

4 The Analogy with Homology Theory

The basic idea behind the wrapper algorithm is quite simple, but has a sophisticated mathematical basis. The motivation is derived from singular theory in the mathematical field of homology theory, which in turn is part of algebraic topology. For the wrapper algorithm, it suffices to consider only q-simplices in singular theory for q equal to 1, 2, and 3.

Simply put, the idea of the wrapper algorithm is to represent the boundary of a region by a collection of oriented cycles, where each cycle defines a triangle lying in space forming a boundary element of the surface. Each triangle has three neighbors, where each neighbor is itself an oriented cycle representing a triangle. The orientations are such that any given edge in the representation is traversed twice, once in each cycle associated with the adjacent faces, and the directions are opposite for the two cycles. The upshot of the representation is that one has a graph structure representing an oriented solid, whose nodes are (oriented) triangles, and whose edges are pairs of adjacent triangles sharing a common edge.

The representation is very much akin to the Baumgart winged-edge data structure, but is specialized for triangular faces, and is augmented by the orientation information on each boundary face. Further, the wrapper algorithm makes use of a standard tetrahedral decomposition of the array of rectangular cells, and incorporates certain algorithmic efficiencies in the determination of boundary cycle orientations.

For the purpose of defining the connection to singular homology theory, and following Greenberg [Gre67], we define the simplex Δ_q to be the subset of $\Re^q$ obtained from convex combinations of (0, ..., 0), (1,0,...,0), and (0,1,0,...,0), etc. Thus Δ_1 is the line from 0 to 1, Δ_2 is the triangle (including the interior) spanned by (0,0), (1,0), and (0,1), and Δ_3 is a tetrahedron spanning the origin and unit vectors in $\Re^3$.

A singular q-simplex in $\Re^3$, for q equal to 0, 1, 2, or 3, is a continuous map from Δ_q into $\Re^3$. Roughly, a singular 0-simplex is a point, a singular 1-simplex is a line segment, a singular 2-simplex is a face, and a singular 3-simplex is a volume. When the continuous maps are affine linear, then the singular simplices are called affine, and the maps are completely defined by the images of the vertices. For singular affine simplices, all the faces and edges are flat.

The border of a singular q-simplex can be viewed as composed of singular $(q-1)$-simplices. That is, a singular 3-simplex is bordered by four singular 2-simplices, and a singular 2-simplex is bordered by three singular 1-simplices, etc. To systematize this observation, it suffices to show that the k-th border of the q-simplex $(0 \leq k \leq q)$ can be seen as an affine singular $(q-1)$ simplex. To specify such an affine singular $(q-1)$ simplex, we simply define the image of the vertices of Δ_{q-1}, (0,...,0), (1,0,...,0), etc., to be the the points in $\Re^q$ (0,...,0,0), (1,0,...,0,0), etc., where the k-th vector (with the (0,...0,0)-vector counting as the zeroth) is left out. We call this affine singular $(q-1)$-simplex Δ_q^k, since it represents the k-th face of Δ_q (Figure 17).

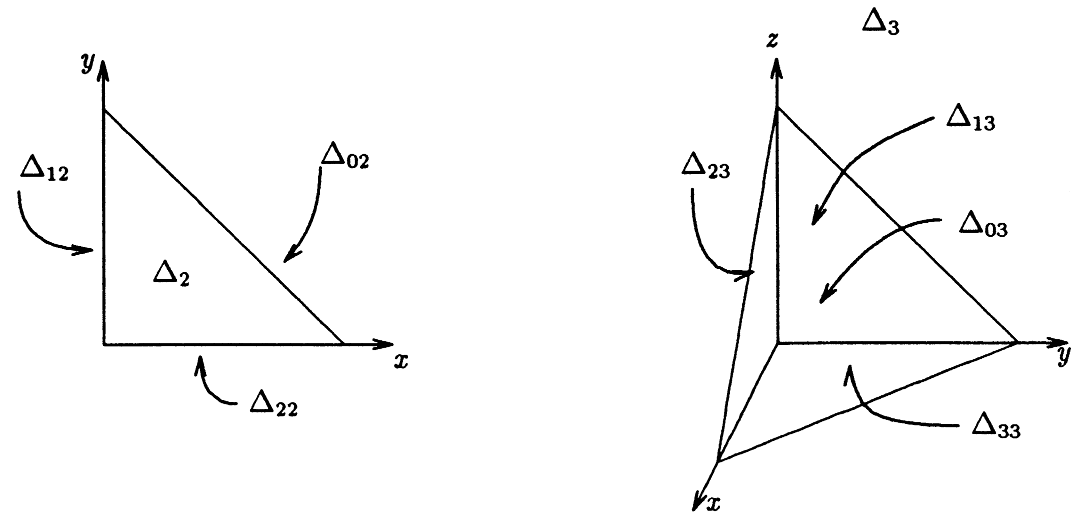

Figure 17: To specify such an affine singular $(q-1)$ simplex, we simply define the image of the vertices of Δ_{q-1}, (0,...,0), (1,0,...,0), etc., to be the the points in $\Re^q$ (0,...,0,0), (1,0,...,0,0), etc., where the k-th vector (with the (0,...0,0)-vector counting as the zeroth) is left out. We thus define the k-th face of Δ_q (here q =2,3).

A singular q-chain is a formal weighted sum (or difference) of singular q-simplices. Thus if $\sigma_1, ..., \sigma_n$ are n singular 3-simplices, their sum $\sigma_1 + \sigma_2 + ...\sigma_n$ is a singular 3-chain that represents nothing more than a "sum" of simplices. The singular simplices are allowed to have integer coefficients, either positive or negative. This is useful for defining the boundary operator. We define the boundary of Δ_q as $\sum(-1)^k \Delta_q^k$. Thus the boundary of the singular 3-simplex Δ_3 (which we can view as the identity map on Δ_3) is given by $\partial\Delta_3 = \Delta_3^0 - \Delta_3^1 + \Delta_3^2 - \Delta_3^3$. Similarly, $\partial\Delta_2 = \Delta_2^0 - \Delta_2^1 + \Delta_3^2$. Using the boundary operator defined for simplices, the boundary operator can be defined analogously for a singular q-simplex, and further extended in a linear fashion to singular q-chains.

A fundamental fact in singular theory is that the boundary operator applied to the boundary of any q-chain is zero, in the sense that all the coefficients of the resulting singular $q-2$ simplices cancel. Normally, in homology theory, we are only concerned with boundaries of q-chains, and two $(q-1)$-chains are considered equivalent if their difference is a cycle (i.e., the boundary operator applied to the difference results in the empty chain).

For the wrapper algorithm, however, we use the boundary operator to compute the boundary of a 3-chain composed of maps from the 3-simplex to tetrahedra in the volume. Specifically, we begin with a function $f(x, y, z)$ defined over a volume, which we model using data at voxel vertices and interpolate using tri-linear interpolation within the cells. We consider the shape $X = \{(x, y, z) | f(x, y, z) > 0\}$, using the assumption that f is trilinear within each cell. Using the tetrahedral decomposition of $\Re^3$, we consider a single tetrahedron T in the collection. We have:

Proposition 1 *The intersection $T \cap X$ has at most one connected component.*

The proof of this proposition depends critically on both the decomposition method and the interpolation scheme, and is part of the motivation for chosing the tetrahedral decomposition and the trilinear interpolation method. Once established, however, it assures us that the topology within any given tetrahedron is quite simple, and can be modeled as a singular 3-simplex. That is, for every tetrahedron T that meets the space X, either T is entirely contained in X, or the X portion in T is a single volume element, occupying a portion of T. But more is true:

Proposition 2 *Further, the boundary portion of X within T is either empty or a single "face," (i.e., a single component) which either meets three faces of T or all four faces of T.*

For the sake of brevity, we omit proofs of these and subsequent propositions, but instead comment on the significance.

Using these propositions, we proceed to represent X by a singular 3-chain $\sigma = \sum \sigma_j$. The main idea is that the boundary $\partial\sigma$ will be a 2-chain that represents ∂X. The 3-chain is constructed in such a way that all interior boundaries in $\partial\sigma$ cancel, and thus all singular simplices with nonzero coefficients in $\partial\sigma$ have ranges that cover ∂X. This is done as follows.

Each tetrahedron T_i is either disjoint from X, contained in X, or intersects X but is not contained in X. In the first case, no singular 3-simplex is required. In the second case, we include in the sum for σ an affine singular 3-simplex σ_i mapping Δ_3 to T_i. The orientation of σ_i matters, and depends on the identity of the tetrahedron T_i. In the third case, where ∂X meets T_i, we use:

Proposition 3 *There exists a polyhedral region Y_i in T_i that is homotopically equivalent to $X \cap T_i$ such that if E is the union of (1-D) edges of the tetrahedron T_i, then $E \cap Y_i = E \cap X$. Further, Y_i is either the range of a single affine singular 3-simplex σ_i or equals the range of a singular 3-chain containing three affine singular 3-simplicial terms $\sigma_i = \sigma_{i,1} + \sigma_{i,2} + \sigma_{i,3}$ such that the range of $\partial\sigma_i$ is ∂Y_i.*

Accordingly, the region X, using the tetrahedral decomposition, is represented by a singular 3-chain $\sum \sigma_i$, where each σ_i is either a singular 3-simplex mapping onto the tetrahedron T_i, or a singular 3-simplex mapping onto a subregion Y_i, or the sum of three singular 3-simplices whose sum maps onto a subregion Y_i. We may consider the range of σ to be a region X' that provides a polyhedral approximation of X. By construction, X' and X are homotopically equivalent, and thus have the same topology and homotopy. We then have:

Proposition 4 *The affine singular 3-chain σ may be constructed so that the range of the singular 2-simplices in $\partial\sigma$ cover $\partial X'$.*

That is, proposition 4 tells us that if the affine singular 3-simplices that are used to construct σ are chosen correctly, all of the internal faces cancel when computing the boundary. The wrapper algorithm uses Proposition 4 to construct a representation of $\partial X'$, computing the singular 2-simplices that remain in $\partial\sigma$.

Proposition 4 tells us even more. The fact that $\partial\sigma$ is a 2-cycle that lies entirely in $\partial X'$ means that σ establishes an orientation on X'. An orientation in homology theory can be precisely defined in terms of elements of relative homology modules, but it suffices to exhibit a q-chain σ such that, as in Proposition 4, $\partial\sigma$ lies in $\partial X'$. (An additional requirement is that σ must be a generator of a particular homology module, which is guaranteed in our case, because the construction of σ involves singular 3-simplices all of whose coefficients are 1 or -1.) Moreover, it is then true that $\partial\sigma$ establishes an orientation on $\partial X'$.

In our case, the orientation on $\partial X'$ can be understood simply. Since X' is a polyhedron in $\Re^3$, $\partial X'$ is a 2-manifold, and an orientation on X' is established if for every $x \in X'$, there is a cycle defined around x such that the direction of the loops "match up" for neighboring x. For the interior of the faces represented in $\partial\sigma$, the orientation for the corresponding face of X' is established in its entirety by the singular 2-simplex that covers the face. That is, suppose that τ is a singular 2-simplex in $\partial\sigma$, which covers a triangular face of X'. Then $\partial\tau$

is a singular 1-chain, containing three singular 1-simplices, which are simply oriented lines circling the border of the triangular face. The direction of the cycle gives the orientation of the face. To define an orientation of the entire 2-manifold $\partial X'$, it is only necessary to observe that the orientations of the faces "match up." That is, if we have two neighboring triangular faces on $\partial X'$, then the orientations of the two must be compatible, in the sense that if the two cycles about the triangles are combined, a single cycle about the pair of triangles results which maintains the same sense as the original cycles. This is assured, however, because $\partial\partial\sigma = 0$, which means that if an edge e is traversed in one direction along a face, then it is traversed in the opposite direction (the coefficient of the singular 1-simplex is -1) on the adjacent face. Accordingly, if the edge is removed, then the points along the edge belong to a quadrilateral (a conjoined pair of faces), and the orientation of the points along the edge is defined by the cycles about the quadrilateral.

5 Experimental Results

We applied the Wrapper Algorithm to the representation of the cranium in CT scans at full resolution. The Figures 18 and 19 represent a normal individual, from of the CMNH Hamman-Todd collection. Figure 21 represents a pathological case, an individual with Crouzon's syndrome, from the collection of the Vienna Museum of Natural History.

We extracted the cortical surface of a normal individual from an MR scan (courtesy of H. Rusinek, Department of Radiology, NYU School of Medicine). The result is in Fig. 20. We thank Gregoire Malandain [MAB93] for applying his segmentation algorithms to the MR-scan. Rendering in these images is done using an orthographic projection of the polyhedral structure. We employ Gouraud shading, using surface normals in the direction of the gradient of voxel data. The color display (Fig. 21) uses the curvature computation of section 3.3. The maximum principal curvature is color-coded, such that high curvature regions appear in red and low curvature regions appear in blue. Green and yellow areas indicate medium curvature values. Derivatives of voxel data are estimated by convolving image data with B-spline functions [Gué93].

We gratefully acknowledge Drs. Court Cutting and David Dean for helpful discussions with the authors. We thank Bruce Latimer, Director, Laboratory of Physical Anthropology, Cleveland Museum of Natural History, for access to the Hamman-Todd morgue collection.

References

[AGTG93] N. Ayache, A. Guéziec, J.P. Thirion, and A. Gourdon. Evaluating 3D registration of CT-scan images using crest lines. In *Mathematical Methods in Medical Imaging II*, volume 2035-06, pages 60–71, San Diego, California, July 14–15 1993. SPIE, The International Society for Optical Engineering.

[Bak89] H. Harlyn Baker. Building surfaces of evolution: The weaving wall. *The International Journal of Computer Vision*, 3:51–71, 1989.

[Bau74] B.G. Baumgart. *Geometric Modeling for Computer Vision.* PhD thesis, Stanford University, 1974.

[CLR89] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* Mac Graw Hill, 1989.

[DK91] Akio Doi and Akio Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Transactions*, E74(1):214–224, January 1991.

[Gre67] Marvin J. Greenberg. *Lectures on Algebraic Topology.* W.A. Benjamen, Reading, Massachussets, 1967. Mathematics Lecture Notes Series.

[Gué93] André Guéziec. Large deformable splines, crest lines and matching. In *Fourth International Conference on Computer Vision (ICCV 93)*, Berlin, May 1993. IEEE.

[Hum86] Robert A. Hummel. *Connected Components Labelling in Image Processing with MIMD Architectures*, chapter Seven, pages 101–127. Intermediate-Level Image Processing. Academic Press, 1986.

[HW90] Mark Hall and Joe Warren. Adaptive polygonalization of implicitly defined surfaces. *IEEE Computer Graphics & Applications*, 10(6):33–42, November 1990.

[Kal91] Alan Kalvin. *Segmentation and Surface-Based Modeling of Objects in Three-Dimensional Biomedical Images.* PhD thesis, New York University, June 1991.

[Kar92] Dan B. Karron. The "SpiderWeb" algorithm for surface construction in noisy volume data. In R.A. Robb, editor, *Visualization in Biomedical Computing' 92*, volume 1808, pages 462–476, 1992. SPIE, Society of Photo-Optical Instrumentation Engineers.

[KCHN91] Alan Kalvin, Court Cutting, Betsy Haddad, and Marylin Noz. Constructing topologically connected surfaces for the comprehensive analysis of 3D medical structures. In *Medical Imaging V: Image Processing*, volume 1445, pages 247–258, Bellingham, Washington, October 1991. SPIE, The International Society for Optical Engineering.

[KDK86] Akio Koide, Akio Doi, and Koichi Kajioka. Polyhedral approximation approach to molecular orbital graphics. *J. Molecular Graphics*, 4:149–160, 1986.

[KT93] Alan D. Kalvin and Russel H. Taylor. Superfaces: Polyhedral approximation with bounded error. Technical report, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1993.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Siggraph*, volume 21-4, pages 163–169, Anaheim, California, July 1987.

[MAB93] Gregoire Malandain, Nicholas Ayache, and Gilles Bertrand. Topological segmentation of discrete surfaces. *International Journal of Computer Vision*, 10(2):183–197, 1993.

[MBF92] Olivier Monga, Serge Benayoun, and Olivier Faugeras. Using third order derivatives to extract ridge lines in 3D images. In *Conference on Vision and Pattern Recognition*, Urbana Champain, Illinois, June 1992. IEEE.

[NB93] Paul Ning and Jules Bloomenthal. An evaluation of implicit surface tilers. *IEEE Computer Graphics & Applications*, 13(6):33–4, November 1993.

[NH91] G. Nielson and B. Hamann. The asymptotic decider: Resolving the ambiguity in marching cubes. In *Conference on Visualization*, pages 83–91. IEEE, 1991.

[PT90] B.A. Payne and A.W. Toga. Surface mapping brain function on 3-D models. *IEEE Computer Graphics & Applications*, 10(5):33–41, September 1990.

[RR94] Rémi Ronfard and Jarek Rossignac. Simplifying a triangular mesh with multiple planar constraints. Technical report, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1994.

[Sch93] Larry L. Schumaker. Triangulations in CAGD. *IEEE Computer Graphics & Applications*, 13(1):47–52, January 1993.

[SZL92] Willian Schroeder, Jonathan Zarge, and William E. Lorensen. Decimation of triangular meshes. In *ACM Siggraph*, volume 26-2, pages 65–70, July 1992.

[TG93] J. P. Thirion and A. Gourdon. The marching line algorithm: New results and proofs. Technical Report 1881, INRIA, Sophia Antipolis, France, 1993.

[TR93] Gabriel Taubin and Rémi Ronfard. Implicit simplicial models I: Adaptive curve reconstruction. Technical report, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1993.

[Tur92] Greg Turk. Retiling polygonal surfaces. In *ACM Siggraph*, volume 26-2, page 55064, July 1992.

[Wal91] Ake Wallin. Constructing isosurfaces from CT data. *IEEE Computer Graphics & Applications*, 11(6):28–33, November 1991.

[WMW86] G Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *Visual Computer*, 2:227–234, 1986.

Bit code	Index	Sign of det() in an even cell
0000	0	+
0001	1	+
0010	2	−
0011	3	+
0100	4	+
0101	5	−
0110	6	+
0111	7	+
1000	8	−
1001	9	+
1010	10	−
1011	11	−
1100	12	+
1101	13	+
1110	14	−
1111	15	+

Table 1: Tabulating the number of transpositions in order to map from the bit code representing the signs of the values at the tetrahedra vertices to the sign of the determinant that determines whether the constructed cycle has the correct orientation (negative determinant) or must be reversed (positive determinant). For an odd cell, the entries are opposite those of an even cell.

Bit code	e_1	e_2	e_3
0000			
0001	1	2	
0010	2		1
0011		2	1
0100		1	2
0101	1		2
0110	2	1	
0111			
1000			
1001	2	1	
1010	1		2
1011		1	2
1100		2	1
1101	2		1
1110	1	2	
1111			

Table 2: Storage locations for edges e_1, e_2, and e_3, as a function of the bit code on the tetrahedron containing the secondary storage location. A "1" entry indicates that the primary storage location should be used for the edge, whereas a "2" denotes the secondary storage. A blank indicates that the edge does not have a zero-crossing with the corresponding bit code.

Tetrahedron		Neighbors $(\Delta x, \Delta y, \ \Delta y, k)$			
Even or Odd	Number	1	2	3	4
Even	1	$(-1,0,0,1)$	$(0,-1,0,2)$	$(0,0,-1,3)$	$(0,0,0,5)$
Even	2	$(1,0,0,2)$	$(0,1,0,2)$	$(0,0,-1,4)$	$(0,0,0,5)$
Even	3	$(1,0,0,3)$	$(0,-1,0,4)$	$(0,0,1,1)$	$(0,0,0,5)$
Even	4	$(-1,0,0,4)$	$(0,1,0,3)$	$(0,0,1,2)$	$(0,0,0,5)$
Even	5	$(0,0,0,1)$	$(0,0,0,2)$	$(0,0,0,3)$	$(0,0,0,4)$
Odd	1	$(1,0,0,1)$	$(0,-1,0,2)$	$(0,0,-1,3)$	$(0,0,0,5)$
Odd	2	$(-1,0,0,2)$	$(0,1,0,2)$	$(0,0,-1,4)$	$(0,0,0,5)$
Odd	3	$(-1,0,0,3)$	$(0,-1,0,4)$	$(0,0,1,1)$	$(0,0,0,5)$
Odd	4	$(1,0,0,4)$	$(0,1,0,3)$	$(0,0,1,2)$	$(0,0,0,5)$
Odd	5	$(0,0,0,1)$	$(0,0,0,2)$	$(0,0,0,3)$	$(0,0,0,4)$

Table 3: List of four neighboring tetrahedra for each type of tetrahedron.

Algorithm	Ellipsoid	Sphere		I=rand()		Cranium		Brain
Wrapper	1720	3102	6200	9228	186703	37594	65752	65668
Marching Cubes	808	1328	2648	4432	93976	15784	27500	30004
Ratio	2.12	2.33	2.34	2.08	2.09	2.38	2.39	2.18

Table 4: Experimental result for the ratio between number of vertices created by the marching cubes and the Wrapper. The ratio is relatively invariant to the sampling rate in the volume.

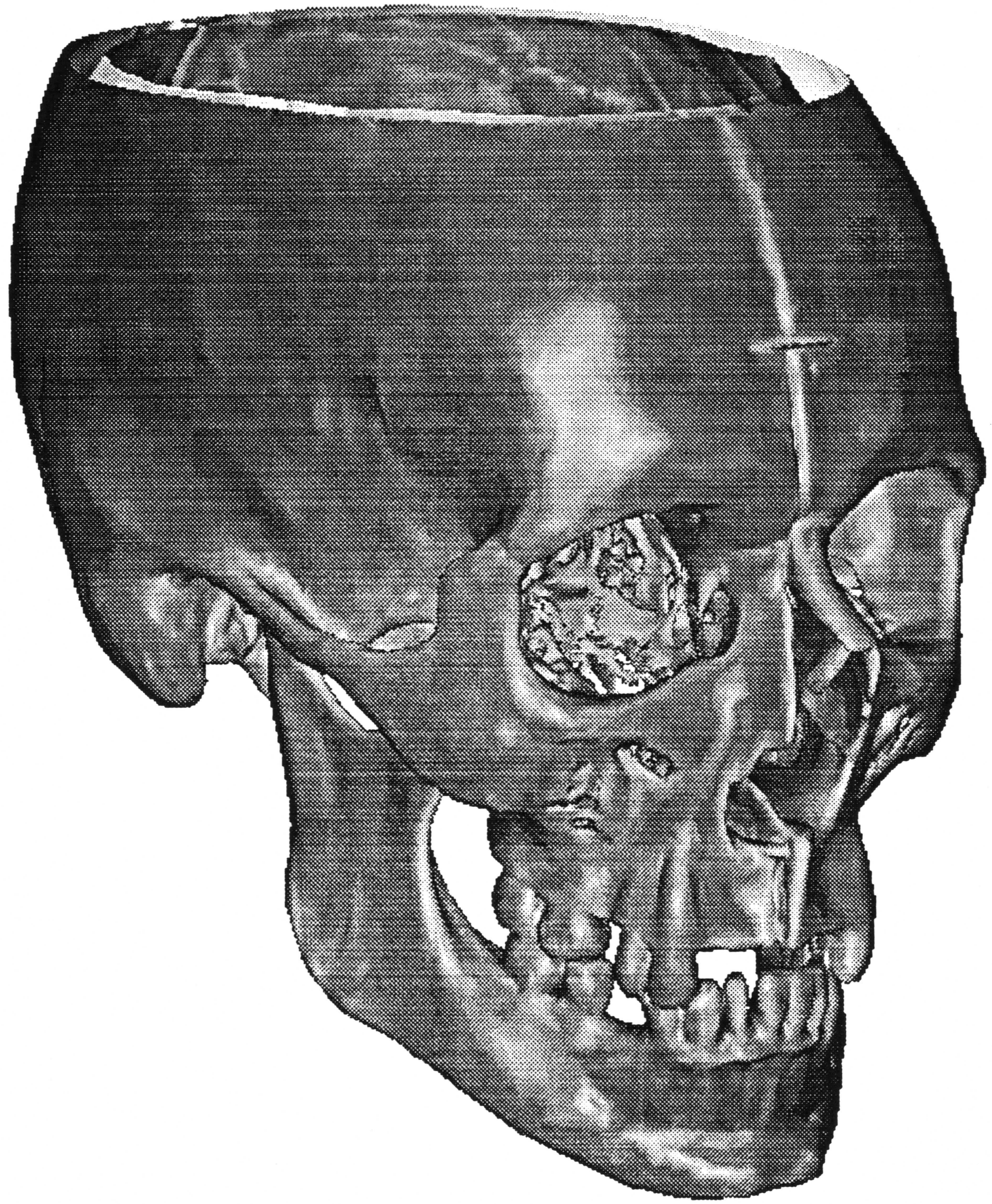

Figure 18: Model of cranium CMNH # 1331, comprising 66K vertices and 129K triangles, extracted from an original 512 by 512 by 150 CT scan. The total number of triangles before simplification was 3.450K, which corresponds to a compression ratio of 27. The maximum error bound is 0.5 millimeter.

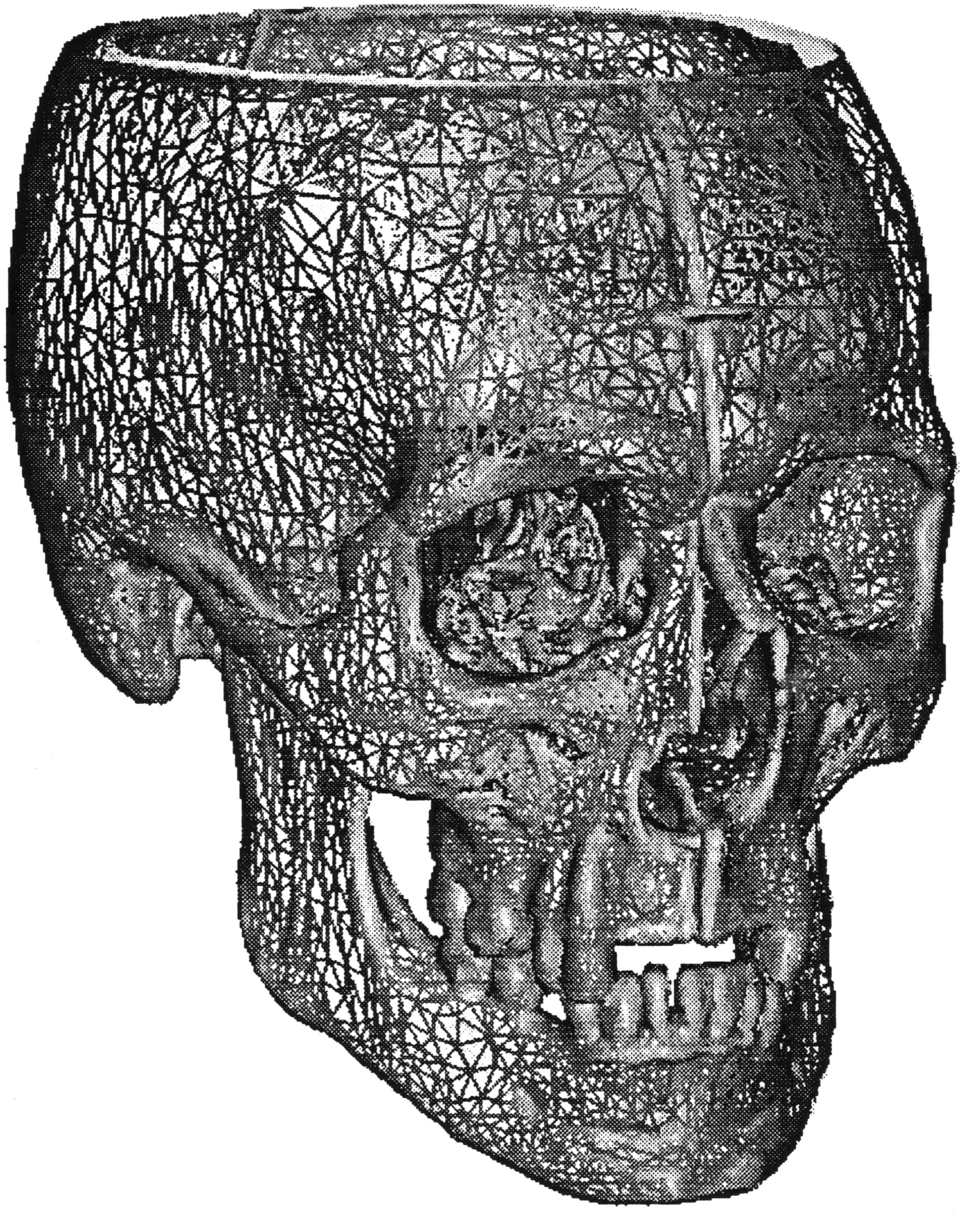

Figure 19: Visualization of the triangular mesh for the model of Fig. 18.

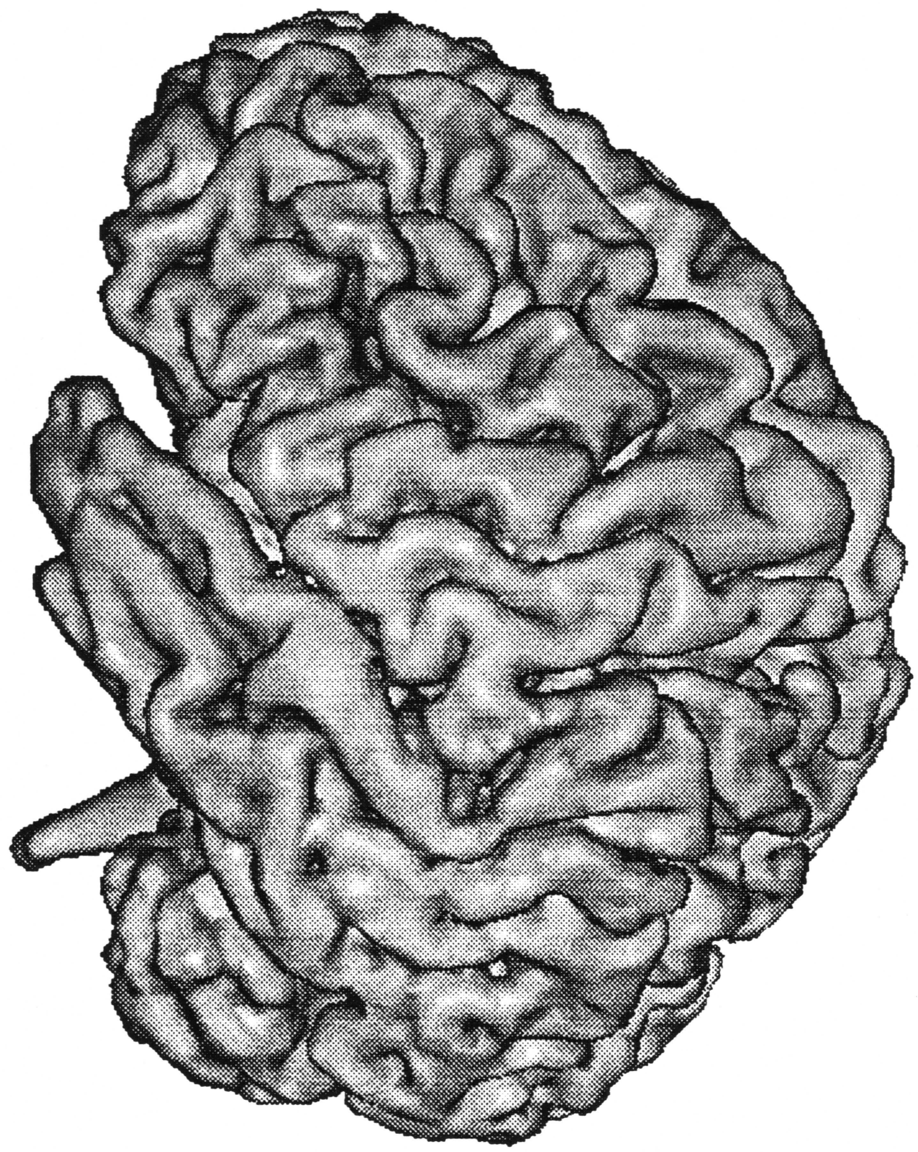

Figure 20: Cortical surface of the brain for a normal individual (31K vertices and 62K triangles), extracted from a 256 by 256 by 130 MR scan. Before simplification, the original number of triangles was 370K. Some important anatomical structures stand out, such as the Inter-Hemispheral Fissure, the Temporal Lobe, the Rolandic Fissure and the Middle Temporal Sulcus.