Ibero-American Symposium on Computer Graphics - SIACG (2006)
P. Brunet, N. Correia, and G. Baranoski (Editors)

Data-driven Tetrahedral Mesh Subdivision

L. Rodríguez[†] and I. Navazo and Á. Vinacua

Department of Software, Universitat Politècnica de Catalunya, Spain

Abstract
Given a tetrahedral mesh immersed in a voxel model, we present a method to refine the mesh to reduce the discrepancy between interpolated values based on either scheme at arbitrary locations. An advantage of the method presented is that it requires few subdivisions and all decisions are made locally at each tetrahedron. We discuss the algorithm's performance and applications.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

There are a number of applications (especially in medical imaging, and scientific visualization) that require support for alterations (deformations, cuts and time-evolution) of volume data.

The volume data are often sampled on a regular rectangular grid in 3D space, and stored in a voxel model. Each point on the grid has an associated (scalar or vector) value of a property, that we can think of as a function f whose domain contains the portion of space that we are modelling. To support further computations, or to afford topological flexibility, a simplicial-cells complex would be preferable, and indeed one often immerses into the voxel space such a network of tetrahedra for those purposes. These meshes may be a result of subdividing the voxels into tetrahedra, or may result from tetrahedralizations of a volume extracted from the model (for instance the volume bound by certain iso-surface).

When this kind of mixed models are used, one needs to address the difference in the way voxel models and tetrahedral meshes compute property values away from the sample points. In fact, in the case of voxels the most often used interpolation method to compute values inside a cell is a trilinear interpolation of the sample values known at the vertices of the cell. Analogously, for tetrahedral meshes the most often used method to compute a value inside a tetrahedron is the linear interpolation of the four vertices of that tetrahedron.

It is obvious that even if the initial values assigned to the vertices of the tetrahedra in the immersed mesh are computed from the values in the original volume data, these two different interpolation methods will yield different values of the property f at points interior to the tetrahedra.

In this paper we present an algorithm to adaptively subdivide a tetrahedral mesh immersed in a voxel model in order to reduce the discrepancy of these two approximations of the function f below a user-specified tolerance ε. Both the input and output tetrahedral meshes are conformal (i.e. a conforming mesh is one in which two tetrahedra $\mathcal{T}_i, \mathcal{T}_j, i \neq j$, of the mesh may only intersect at a vertex, along a complete edge or have a common triangular face). Our algorithm works by locally subdividing tetrahedra where the discrepancy exceeds the tolerance. This paper presents the following contributions:

- Analysis of the discrepancy between the two interpolations within a tetrahedron
- A scheme for locally subdividing tetrahedra with large discrepancies that ensures a conformal resulting mesh and a reduction of the discrepancy within the resulting tetrahedra
- The subdivision scheme aims at minimizing the number of tetrahedra needed to meet the requirement
- The subdivision is designed to yield good quality tetrahedra (not too skinny and elongated)

[†] This author have a grant from the Spanish Ministry of Science and Education
This project have been partially funded by the Spanish Ministry of Science and Education (TIN2004-08065-C02-01)
E-mail address: {lyudmila, isabel, alvar}@lsi.upc.edu

The rest of the paper starts by presenting an overview of previous work in the next section, In Section 3, we report on our analysis of the locations of the largest discrepancies and the subdivision patterns (see Section 3.2) used to achieve locality and convergence. Section 4 then discusses some aspects of the implementation of this algorithm, before moving on to the results on our test models, discussed in Section 5, and closing with some conclusions.

2. Previous Work

A subdivision scheme can be seen as a procedure to construct a collection of n different meshes $\mathscr{M} = \{\mathscr{M}_1 < \mathscr{M}_2 < \ldots < \mathscr{M}_n\}$, such that the mesh $\mathscr{M}_{i+1}$ is obtained from the previous one($\mathscr{M}_i$) by a local refinement. At each level, an element is refined if it exceeds a preset error criterion (appropriate for the problem) at that level.

In the context of finite elements, tetrahedral meshes are sometimes subdivided to improve their quality or suitability for the computation at hand. One of the first subdivision schemes that was proposed for two-dimensional triangle meshes, is the red/green method, by Bank and Sherman [BSW83]. In the *red* phase, triangles are subdivided into four similar triangles by splitting all three sides at their midpoint. The *green* phase is subsequently applied to all the neighboring elements that have not been subdivided, but which share exactly one edge with a subdivided triangle. These triangles are split in two joining the midpoint of the edge they share with a subdivided triangle with the opposite vertex. Triangles that have not been subdivided but that share more than one edge with a subdivided triangle are then subdivided using the *red* strategy, and a subsequent *green* pass is needed to blend them properly with their neighbors.

Another group of algorithms focuses on the edges, rather than the triangles. This is the case of the work of Rivara [Riv84, Riv89, RI96], in the two-dimensional case. They iteratively apply the longest edge bisection technique. In [Riv84] they use it in its pure form, where the longest edge of each triangle that needs to be subdivided is split at the midpoint, adding an edge to the opposite vertex. In [RI96] they use instead the 4-Triangles subdivision, where after subdividing a triangle by its longest edge, the midpoint of the longest edge is also joined with two additional edges with the midpoints of the other two edges, resulting in a subdivision into 4 triangles. Both schemes are supplemented by a pass to insure conformacy. The authors show numerical evidence that both strategies yield conformal triangulations of good quality.

These strategies have subsequently been extended to the three-dimensional case. Zhang [Zha95], Bey [Bey95], Liu and Joe [LJ96] have extended the strategy of Bank and Sherman [BSW83] to the 3D case. In a first step, they subdivide a tetrahedron in eight subtetrahedra. Four subtetrahedra similar to the one being subdivided result from cutting out the four corners of the tetrahedron, at the midpoint of the edges. The remaining four are obtained by subdividing the central octahedron that results from this corner-cutting. The way in which this octahedron is subdivided, and the schemes used to conformally blend the subdivided part of the mesh with its unsubdivided neighbors are the elements that differentiate the proposals listed above. More recently Greiner and Grosso [GG00] used a similar scheme, but the interior octahedron is subdivided into six octahedra and eight tetrahedra on demand.

The edge bisection methods have also been extended into 3D. Rivara and Levin [RL92] extended first the pure bisection method by simply bisecting tetrahedra by splitting the longest edge and joining the split point with the opposite vertices of the adjacent faces. Liu and Joe [LJ95] later showed by numerical experiments that this may lead to the bisection of many tetrahedra, which not only increases the cost of the subdivision, but more importantly may severely impact the finite elements computation. Instead, they propose to map the tetrahedron onto a canonical one, and show how to use this to subdivide the tetrahedron into eight similar pieces, in a three steps process.

Plaza and Carey [PC96, PC00] have extended to three dimensions the 4-Triangle algorithm. They insert new vertices at the midpoint of each edge. Then they explore the neighbors to verify their conformity and they insert new vertices at the midpoint of the longest edge of each non conforming face and add a new vertex to the midpoint of the longest edge of the tetrahedra. From this points, it makes the subdivision of each face using the 4-Triangle algorithm, obtaining the skeleton. Once the set of faces have been consistently triangulated in this way, they complete the subdivision of the interior of the tetrahedra based on a set of 51 different precomputed patterns. More recently, they propose a new method denominated “8-Tetrahedra longest edge partition” [PR03, PPSF04].

In Computer Graphics applications, refinement algorithms used for multiresolution purposes are intended to allow the acceleration of the visualization and interaction processes. These algorithms use both subdivision and fusion techniques [DFLS02, ZCK97]. In Danovaro et al.’s work [DFLS02], unstructured meshes are refined with vertex split rule. On the other hand, Zhou et al. [ZCK97] propose a hierarchy of tetrahedra obtained by a recursive subdivision of the volume. Three subdivision rules and an error saturation strategy are defined for the multiresolution.

On the other hand, adaptive subdivision of triangle meshes for deformable models is presented by Ruprecht et al. [RNM95]. They apply adaptive subdivision for deformable models used in volumetric data matching or volumetric morphing. The subdivision is carried out when the distance between the edge midpoints in real space and the same points in the deformed space is greater than certain ε. They use this adaptive subdivision strategy in [RM98], to subdivide tetrahedral meshes. Their subdivision scheme is similar to the one we propose, although we solve differently the cases with additional degrees of freedom.

Yet another field of application where a need for these subdivisions arises is surgery simulation. Here one wants to simulate cuts into volumetric models based on tetrahedral meshes. Cuts are simulated by subdividing the tetrahedra intersected with the virtual scalpel. These techniques differ slightly from the previous methods because the subdivision points are given by the user interaction, and have to be duplicated in order to separate the mesh along the cutting line [GCMS01, BG00, FDA02, MK00].

Neither of the previous subdivision techniques takes into account in the subdivision process the volumetric information contained in the interior of the tetrahedra. In our application we focus on the extraction of a tetrahedral mesh from the volume data, and are therefore concerned with how well does the extracted mesh agree with the model in terms of the estimates of the property of interest in its interior points. Thus, we define the subdivision rules according to the interior information of the volumetric data. We are not aware of previous results in the literature that address this problem in these terms.

3. Our proposal

As we have previously introduced, we are interested in the case where we have a hybrid model, consisting on volume data in the form of a voxelization, and a tetrahedralization of a portion of the same volume, where the vertices of the tetrahedralization are in arbitrary positions within the volume. For instance, these tetrahedralization may come from the computation of an active-contour-like triangle mesh delimiting a portion of interest of the volume, followed by a subdivision of that volume compatible with the triangulation of the boundary. Notice that in this general setup a tetrahedron may span several voxels, or several tetrahedra may be completely contained inside a voxel.

In this section we present an algorithm to adaptively subdivide a tetrahedal mesh immerse in a voxel model in order to reduce the discrepancy between interpolated values computed using either scheme at arbitrary positions.

The next subsection discusses the nature of these discrepancies in a formal way. However, we have not reached a useful closed-form solution for the optimal way to subdivide tetrahedra, therefore we have resorted to experiments, which are discussed in Section 5. From these numerical experiments, we have seen that most of the time the point of maximum discrepancy happens near the midpoint of a face or edge. When the maximum occurs at the midpoint of a face, a similar value of the discrepancy appears near the midpoint of at least one of its edges. For this reason, and in the interest of speed, we chose to analyze only the edge midpoints of the given tetrahedron, splitting an edge at its midpoint if the discrepancy there exceeds a threshold. This has the interesting property of providing a completely local test.

3.1. On the computation of discrepancies within a tetrahedron

Let $\mathscr{M}$ be a tetrahedral mesh immersed in a voxel model $\mathscr{V}$. Let us further assume that we assign to each vertex $v \in \mathscr{M}$ a property value obtained by trilinear interpolation of the corners of the cell that contains v in the voxel model. A tetrahedron $\mathscr{T} \in \mathscr{M}$ is called a *good predictor* if for any point $P \in \mathscr{T}$ the discrepancy between the property value computed at P from the voxel model (by trilinear interpolation of the vertices of the cell that contains it) and from the tetrahedral mesh (by linear interpolation of the property values at the vertices of $\mathscr{T}$) is below a user-specified threshold ε.

That is, if P has coordinates (x,y,z) within its cell (i.e. $x,y,z \in [0,1]$), then we define

$$\begin{aligned} realValue(P) = \; & (1-x)(1-y)(1-z)I_{000} + (x)(y)(z)I_{111} + \\ & (1-x)(y)(1-z)I_{010} + (x)(1-y)(z)I_{101} + \\ & (1-x)(1-y)(z)I_{001} + (x)(y)(1-z)I_{110} + \\ & (x)(1-y)(1-z)I_{100} + (1-x)(y)(z)I_{011} \quad (1) \end{aligned}$$

where the I_{ijk}, denote the values at the corresponding corners of the cell.

Moreover, let b_i, $i = 0 \ldots 3$ be the barycentric coordinates of the point P with respect to its tetrahedron $\mathscr{T}$ (which satisfy $0 \le b_i \le 1 \forall i$ and $b_0 + b_1 + b_2 + b_3 = 1$), and let I_{vi} be the values assigned to the vertices of $\mathscr{T}$. Then we define

$$aproxValue(P) = b_0 I_{v0} + b_1 I_{v1} + b_2 I_{v2} + b_3 I_{v3} \qquad (2)$$

The condition that $\mathscr{T}$ is a good predictor can then be written as

$$\forall P \in \mathscr{T} : \frac{|realValue(P) - aproxValue(P)|}{normCoeff} \le \varepsilon \qquad (3)$$

where *normCoeff* is a normalization coefficient so that all values are in $[0,1]$ (and discrepancies measure relative error).

We shall also denote the discrepancy at a point P by $error(P) = |realValue - aproxValue|/normCoeff$. If $error > \varepsilon$ the tetrahedron is not a good predictor and must be subdivided. Choosing ε is relatively straightforward for the user, as it represents relative error. A value of 0.1, for example, indicates that errors below 10% are acceptable.

3.2. Subdivision scheme

In this subsection we present a scheme for subdividing bad predictor tetrahedra that minimizes the number of resulting tetrahedra. It is based exclusively on computing the discrepancy on the edge midpoints of a given tetrahedron, splitting an edge if the discrepancy exceeds a relative error.

We need to consider the different configurations of edges of a tetrahedron that need to be split. We would like to achieve a scheme that does not impose a subdivision of an edge that was not marked for splitting to begin with. This is both related to minimizing the number of resulting tetrahedra and to making the scheme local (an edge needs to be split based on an intrinsic property, and not on the configuration of its neighbors).

Let us first linger for a moment in the simpler two-dimensional case, where triangles are subdivided by breaking their edges. Since a triangle has three edges, there are $2^3 = 8$ different configurations. The extreme configurations, in which no edge is subdivided and all three edges are subdivided, only happen once each one. The cases where only one edge is subdivided and where two edges are subdivided occur three times each. The 8 configurations are thus reduced to 4 due to symmetries, shown in Figure 1. Notice that the quadrilateral region in case 2 can be triangulated in two different ways. Some authors solve the ambiguity by using the memory addresses of the different vertices. Instead, we choose to add the shortest of the two diagonals of the quadrilateral region to split it into two triangles. This yields better shaped tetrahedra (see [Riv84]).

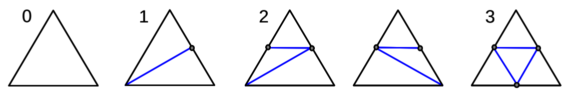


Figure 1: *Configurations for triangle subdivision*

We can therefore construct a Look Up Table with eight entries (indexed by the vertex classification), sorting the different configurations into one of the four cases depicted above. However, when we reach configuration number two, we need to also compute the shortest diagonal of the quadrilateral portion in order to decide the sub-case applicable. This yields a total of 11 possible triangulations.

Let us now consider the three-dimensional case. Our tetrahedron subdivision scheme is based on producing exactly these subdivisions on the faces of the tetrahedra it subdivides. Since neighboring tetrahedra share a face, and they both get subdivided in a way that is consistent with that face, the result is automatically conformal. Tetrahedra have 6 edges, so there are $2^6 = 64$ possible edge refinement patterns. Removing cases that differ by a symmetry or a rigid motion, the 64 cases are reduced to 11 different configurations, shown in Figure 2. As in the two-dimensional case, when splitting a face we get a quadrilateral region we compute the length of its both diagonals and then split the quadrilateral along the shortest diagonal. This yields better triangles, and therefore tetrahedra with better quality. If both diagonals have equal length, the one containing the vertex with smaller *id* on the mesh data structure is selected. This allows us to guarantee that these faces, when shared by two tetrahedra, are triangulated without ambiguity. Let us now examine these 11 configurations in detail.

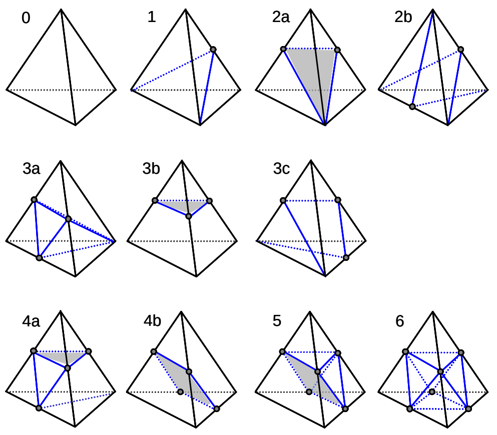


Figure 2: *Configurations for a tetrahedron subdivision*

Configuration 0 in Figure 2 corresponds to the trivial case where no edge needs refining, and the tetrahedron is not subdivided further. Almost as simple is the case of **Configuration 1**, where a single edge needs subdividing. Two sub-tetrahedra are obtained by joining the new midpoint with the opposite vertices.

There are two distinct cases where two different edges need to be subdivided:

Configuration 2a: The two edges belong to the same face. This case yields a subdivision into three tetrahedra as shown in Figure 3. The quadrilateral region is split according to the previously explained criteria (shortest diagonal).

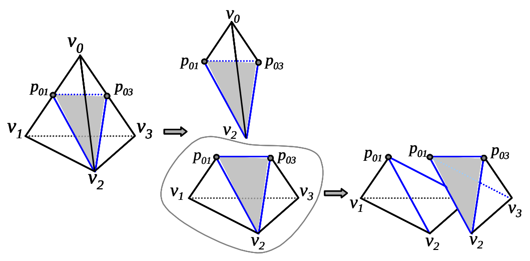


Figure 3: *Possible tetrahedralizations for Configuration 2a*

Configuration 2b: The two edges to divide are opposite edges of the tetrahedron. This case can be solved by applying the solution for configuration 1 twice in succession. In this simple way the tetrahedron is split in four sub-tetrahedra.

In the case where three edges exhibit errors above the chosen threshold, three different configurations may arise:

Configuration 3a: The three edges needing subdivision belong to the same face. The face is divided in four triangles with new edges connecting the error points (see Figure 2). Each new triangle is joined with the opposite vertex and the tetrahedron is thus subdivided into four sub-tetrahedra.

Configuration 3b: The three edges have a common vertex. All three triangular faces sharing that common vertex have two edges with errors above the threshold value (See Figure 4). A sub-tetrahedron is formed by that vertex and the three midpoints of the converging edges. The remaining prism can be subdivided in two different ways, depending on the lengths of the diagonals of each of the three quadrilateral faces of the prism. The two possible resulting cases are shown in Figure 4 at right. The first one (the top right sub-figure) arises when two of the shorter diagonals converge at a splitting point (p_{03} in the figure). In that case the prism is subdivided into three tetrahedra (i.e. in the figure, the tetrahedra with vertices $(v_2, p_{01}, p_{02}, p_{03})$, $(v_1, v_2, p_{01}, p_{03})$, and (v_1, v_2, v_3, p_{03})). The second possibility occurs when no two of the shorter diagonals of the three quadrilaterals begin/finish at the same point. In this case, an untetrahedralizable region known as a Schönhardt prism is formed. This prism cannot be broken up into tetrahedra whose vertices are vertices of the prism unless the triangulation of one of the three quadrilateral facets is changed by doing an edge flip [RNV02]. We want our tetrahedron subdivision to depend only on local information, so we cannot afford this edge flip (which would go unnoticed to the neighbor tetrahedron). Instead, we add a point inside the prism to guarantee coherence of the subdivision. Then, each triangular face is joined with the inserted point. As a result, the prism is subdivided into eight tetrahedra.

Configuration 3c: Two of the three edges where the error exceeds the threshold are opposed. The third edge shares two different facets of the tetrahedron, one with each of these two opposed edges, as shown in Figure 5 at left. We break up these tetrahedra in five sub-tetrahedra as follows: first, consider the facets of the tetrahedron that have only one splitting point (v_0, v_1, v_2 and v_1, v_2, v_3 in the figure), and split these faces adding an edge from the splitting point to the opposite vertex. We thus obtain the first sub-tetrahedron, whose edges are these four points (v_1, v_2, p_{01} and p_{23} in the figure). What remains is the union of two pyramids with apices at these two splitting points (p_{01} and p_{23}), and with quadrangular bases (v_0, v_2, p_{23}, p_{03} and v_3, v_1, p_{01}, p_{03} respectively). These two pyramids share a triangular face whose vertices are the three splitting points as shown on the right of Figure 5. Each of these pyramids is broken up into two tetrahedra by splitting its base along the shortest diagonal.

There are two different cases where four edges need to be subdivided:

Configuration 4a: In this case three of the four edges with split points belong to the same face of the tetrahedron. The fourth one necessarily shares one vertex with that face. This configuration is depicted at the left side of Figure 6. We subdivide this configuration by forming two tetrahedra sharing the triangle p_{01}, p_{02}, p_{03} in the figure (i.e. the triangle formed by the split points on three converging edges). These tetrahedra are $(v_0, p_{01}, p_{02}, p_{03})$ and $(p_{01}, p_{02}, p_{03}, p_{12})$ in Figure 6. The remaining volume within the tetrahedron consists again of two pyramids with quadrangular bases separated by the triangle p_{12}, p_{03}, v_3. These pyramids are finally subdivided into two tetrahedra each one using the shortest diagonal criterion (see Figure 6 at right). Finally, six sub-tetrahedra are obtained.

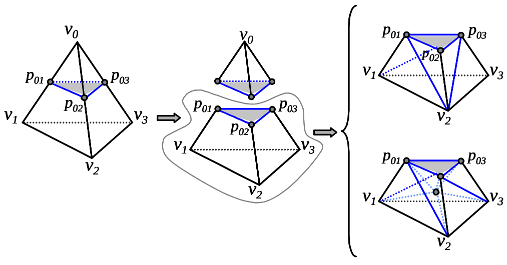


Figure 4: *Two possible tetrahedralization for Config. 3b*

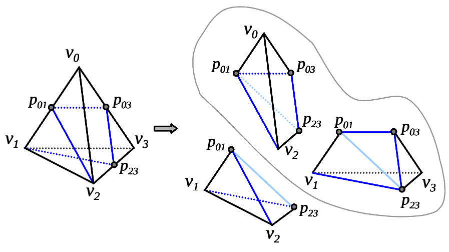


Figure 5: *Tetrahedralization of Configuration 3c*

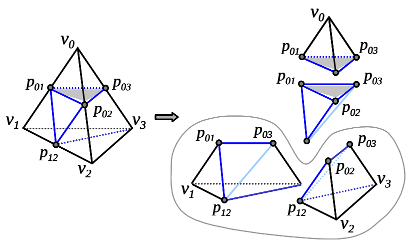


Figure 6: *Configuration 4a*

Configuration 4b: Each face of the tetrahedron has exactly two edges with split points. The split points are the vertices of a quadrangle that splits the tetrahedron into two similar prisms (see Figure 7).

To tetrahedrize each of the two prisms, we consider the shortest diagonal of each of the quadrangular faces contained in the boundary of the tetrahedron. For the first prism to be tetrahedrized, two cases are possible, depending on whether its two shorter diagonals of the exterior quadrangles have a point in common or not. If they do (Figure 7, top right) then we are still free to split the central quadrangle, so we look at the tetrahedralization of the second prism, in order to minimize the probability of producing Schönhardt prisms. If they do not (Figure 7, down right) then the interior quadrangle is split joining the endpoints of these diagonals. The second prism will inherit the choice of diagonal for the interior quadrangle. If it forms a Schönhardt prism, we have no degree of freedom left, and we must introduce an interior point to tetrahedrize it. The number of the resulting sub-tetrahedra is six or eleven when it is needed to insert an interior point.

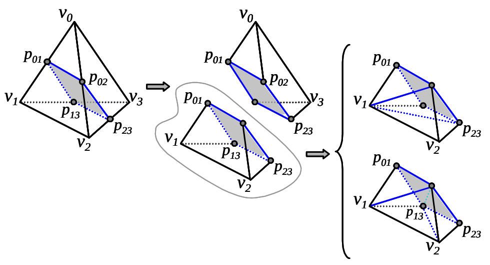


Figure 7: *Configuration 4b*

Configuration 5: This is the case where there are five split points on the edges of the tetrahedron (see Figure 8). The figure shows how the tetrahedron is naturally split into a prism, a pyramid with quadrangular base, and two tetrahedra. The prism is tetrahedrized like the first prism in case 4b, determining the diagonal to use in the interior quadrangle shaded in grey, which determines the splitting of the pyramid and hence the whole tetrahedralization. Seven sub-tetrahedra are obtained.

Configuration 6: All six edges of the tetrahedron contain a split point. Joining the split points on the edges that converge to each vertex of the tetrahedron, we obtain four small corner tetrahedra, and a central octahedron. The octahedron is tetrahedrized inserting the shortest of the three internal diagonals that join split points on opposed edges, and thus dividing it into four more tetrahedra (see Figure 9). The tetrahedon is split in eight sub-tetrahedra.

The Table 1 summarizes the subdivision process. The column labeled "Case" lists the configuration number. The second column shows the number of sub-tetrahedra into which the standard tetrahedron being considered is split. In cases 3b and 4b, where a Shönhardt prism may arise, include in parenthesis the number of tetrahedra needed in that case. The third column indicates the number of entries corresponding to each base case (taking symmetries and rigid motions into account), and adds up to the 64 edge possible refinement patterns. The last column, finally, indicates the number of different tetrahedralizations (NDT) that can arise in each base case because of the different possible subdivisions of the quadrangular facets. Notice that the 11 subdivision patterns yield thus 269 different tetrahedralizations (the sum of the products of columns NE and NDT).

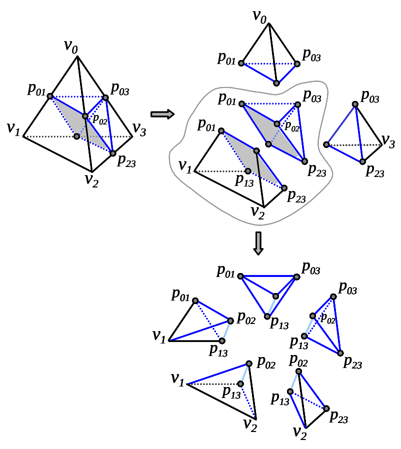


Figure 8: *Tetrahedralization of Configuration 5*

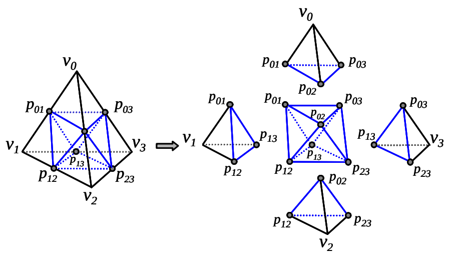


Figure 9: *Tetrahedralization of Configuration 6*

Case	$\|\mathcal{T}\|$	NE	NDT
0	1	1	1
1	2	6	1
2a	3	12	2
2b	4	3	1
3a	4	4	1
3b	4 (9)	4	8
3c	5	12	4
4a	6	12	4
4b	6 (11)	3	22
5	7	6	6
6	8	1	1

Table 1: *Summary of the subdivision process*

An analysis of the different cases just discussed will convince the reader that when a triangular facet is shared by two tetrahedra, the splitting on both neighbors will be consistent at the facet: every facet with just one splitting point will have been broken up in two triangles by joining that splitting point with the vertex opposed to it; every facet with two splitting points will have been broken up into three triangles, by adding an edge joining the two splitting points, plus the shortest of the two diagonals of the remaining quadrangle; and every facet with a splitting point on each edge will have been broken up into four triangles by adding the three edges defined by the three splitting points. Therefore this subdivision scheme allows for completely local decisions and automatically produces conformal tetrahedralizations. Note that the discrepancies along edges will coincide regardless of which tetrahedron is used to compute them, so indeed all neighboring tetrahedra see the same configuration and therefore consistent decisions are made on all neighbors.

4. Implementation details

Our algorithm proceeds as follows. Initially, we compute values for each vertex of the given tetrahedralization from the voxel data, using trilinear interpolation. Then the error at the midpoint of each edge is computed and compared with the threshold. For each tetrahedron, the collection of edges exceeding the threshold determines an index into a lookup table of 64 entries. The lookup table store five different values. The first value represents the subdivision case. The remaining four values are the order in which the vertices must be sorted so as to match the standard configuration for that case. This maps each possible rotation or symmetry onto a canonical position so the algorithm does not worry about these transformations. A pseudo-code description of the algorithm is shown in Algorithm 1.

Figure 10 shows an example of the program at work with a tetrahedron with configuration 3b. The example shows how the code computes the resulting subdivision without computing the transformation from the given setting to the standard configuration. Instead, intrinsic properties are used to find the vertices of the new tetrahedra, greatly simplifying the code.

5. Experimental Results

We have performed two different kinds of experiments. The first one was addressed at studying the distribution of the discrepancy inside tetrahedra, and the second at measuring the performance of our subdivision algorithm on real testcases.

5.1. Discrepancy analysis

To determine the behaviour of the discrepancy within a tetrahedron, we computed the value of the left-hand side of inequality (3) at regularly spaced points (those with barycentric coordinates of the form $\left(\frac{i}{n}, \frac{j}{n}, \frac{n-i-j}{n}\right), i, j \in \{0, \dots, n\}$).

Algorithm 1 General algorithm

```
for all $tetrahedron\,\mathcal{T} \in \mathcal{M}$ do
  $errorEdges \leftarrow \emptyset$
  for all $i$ such that $0 \le i < 6$ do {each edge of the tetrahedron}
    $realValue = CalculateRealValue(splitpoint_i)$ {with eq. 1}
    $aproxValue = CalculateAproxValue(splitpoint_i)$ {with eq. 2}
    $error = |realValue - aproxValue|/normCoeff$
    if $error > \varepsilon$ then
      $errorEdges \leftarrow errorEdges \cup edge_i$
    end if
  end for
  if $errorEdges.size() > 0$ then
    $lookUpEntry = LookUpCode(errorEdges)$
    $SubdivisionProcess(lookUpEntry)$
  end if
end for
```

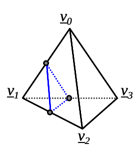
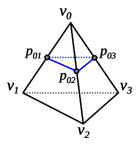
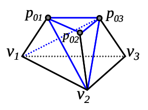
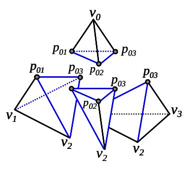
011001 → 25 	lookUpEntry = 25 case = *lookUpTable*[25][0] order[ ] = *lookUpTable*[25] $= [\underline{v}_1, \underline{v}_2, \underline{v}_0, \underline{v}_3]$ $= [v_0, v_1, v_2, v_3]$
	$EraseTetra(idTetra)$ $AddTetra(v_0, p_{01}, p_{02}, p_{03})$
	$a = ShortDiag(v_1, p_{02}, v_2, p_{01}) \equiv (v_2, p_{01})$ $b = ShortDiag(v_2, p_{03}, v_3, p_{02}) \equiv (v_2, p_{03})$ $c = ShortDiag(v_3, p_{01}, v_1, p_{03}) \equiv (v_1, p_{03})$ $x = (a \cap b) \cup (a \cap c) \cup (b \cap c) \equiv (v_2, p_{03})$ $y = (a \cup b \cup c) - x \equiv (v_1, p_{01})$ $AddTetra(p_{01}, p_{02}, p_{03}, v_2)$ $AddTetra(p_{03}, p_{01}, v_1, v_2)$ $AddTetra(p_{03}, v_1, v_2, v_3)$
	*Final subdivision: 4 tetrahedra*

Figure 10: *Example of a subdivision process for case 3b*

First we tested regular subdivisions of voxels —into five tetrahedra each— to measure the part of the discrepancy due to the difference in interpolation schemes. We estimated the error in each tetrahedron by sampling 165 equally spaced points ($n = 8$ in the formula above).

In this experiment, 93% of the time the maximum discrepancy happened at the midpoint of one of the edges. In the remaining cases the maximum occurred at a face of the tetrahedron (i.e. one of the barycentric coordinates was zero). In these cases, however, at least one of the edges of the face exhibiting the maximum discrepancy had a discrepancy of the same order.

To assess the contribution to the discrepancy coming from the higher sampling rate of the voxels, we run further tests with coarse tetrahedralizations of volume models. We set a threshold for the relative error such that discrepancies below that threshold were ignored. The results obviously presented more variability. However, for a threshold of 0.001, we still found that in over 73% of the cases (again computed taking $n = 8$), the maximum discrepancy occured large at the midpoint of some edge or in a point near to this one. Increasing the threshold to 0.05 increased the number of hits to 85%.

Based on these results we concluded that a more precise estimation of the position of maximum discrepancy would probably not yield substantial performance benefits, and decided to proceed with a straightforward test based on comparing the errors at the midpoints of the tetrahedra to decide wich edges to split. This choice seems to be confirmed by experiments on medical datasets that are the subject of the next subsection.

5.2. Performance for medical data

We have tested our algorithm on several medical datasets. The heart models were obtained with SPECT and the liver was obtained with MRI. The MRI model has been undersampled to a low resolution. Table 2 shows the resolutions and voxel sizes for three of these models.

For the heart data we also had reconstructions of the inner and outer surfaces of the ventricle, and proceeded to build a tetrahedralization between them, as presented in [RNV02]. For the liver model, a tetrahedralization of the voxels was performed instead.

We tested all models with different relative errors. Due to space limitations we can not report here overall tests in detail, but the interested reader can see a complete description in report [RNV06]. Table 3 summarizes the results obtained using a threshold of 0.1 (i.e. a relative error of 10%). The table shows the number of tetrahedra of each configuration found for the first three iterations of our algorithm, and for each of the example models. Notice that the first row corresponds to case zero, where no further subdivision is required. The last row indicates the number of Shönhardt prisms found in each iteration.

It can be noted that after the second subdivision, most of the tetrahedra that require further subdivision correspond to the cases in which the subdivision process produces fewer tetrahedra (refer to the discussion in Section 3.2). Nonetheless the discrepancy quickly decreases below the specified threshold, as shown in Table 4. In this table INT designate the number of tetrahedra at the beginning of each iteration, and FNT the number of tetrahedra after one step of subdivision. NTE shows the number of tetrahedra that exceed the threshold and must be subdivided. Finally, “max. err” and “av. err” columns show the maximum relative discrepancy in the model at the start of the iteration and the average of the relative error. Also notice that for the chosen thresholds (see report [RNV06]), it is reasonable that the average errors to stop decreasing as shown.

	Resolution	voxel size (in mm.)
heart1	64*x*64*x*24	2.87*x*2.87*x*5.74
heart2	32*x*32*x*11	10.776*x*10.776*x*10.776
liver	16*x*16*x*16	24.5624*x*24.5624*x*3.19998

Table 2: *Medical data sets used*

	INT	FNT	NTE	max. err	av. err
It 1	1009	3176	728	0.47172	0.17986
heart1 It 2	3176	4496	723	0.25994	0.07201
It 3	4496	4554	44	0.18381	0.04910
It 4	4554	4566	10	0.13882	0.04872
It 1	988	3263	744	0.53082	0.19967
heart2 It 2	3263	4043	519	0.25771	0.07193
It 3	4043	4116	53	0.23537	0.06140
It 4	4116	4162	30	0.18470	0.06066
It 1	10955	20482	3648	0.54084	0.10795
liver It 2	20482	25552	3333	0.44306	0.06128
It 3	25552	26613	813	0.23997	0.05033
It 4	26613	26836	171	0.16655	0.04844

Table 4: *Evolution of discrepancies at each iteration (Threshold = 10%)*

In order to visualize the meaning of these discrepancies, Figures 11 and 12 show perfusion levels at two slices of a heart model. In each case, subfigure (a) displays the original perfusion levels stored in a voxel model, and subfigures (b) the linear interpolation with the initial tetrahedral mesh. In subfigures (c) we show the result after four iterations of our algorithm with a threshold of 10%, and in (d) the result after four iterations for a threshold of 5%. Notice that the subfigures (d) are in both cases very similar to the original (in (a)), showing the quick convergence of the algorithm. The extreme right subfigures show the discrepancies between (a) and (d), scaled by a factor of 20 in Figure 11 and by a factor of 15 in Figure 12.

	Initial quality	thrshld. 10%	thrshld. 5%
heart1	0.565231	0.578719	0.574900
heart2	0.499718	0.508499	0.507415
liver	0.333292	0.313060	0.309224

Table 5: *Quality of the meshes*

Case	heart1			heart2			liver		
	Iter. 1	Iter. 2	Iter. 3	Iter. 1	Iter. 2	Iter. 3	Iter. 1	Iter.2	Iter. 3
0	281	2453	4452	244	2744	3990	7307	17149	24739
1	110	348	31	117	327	33	984	1793	566
2a	130	212	12	145	148	20	638	1354	246
2b	2	9	1	3	0	0	40	157	1
3a	1	0	0	3	0	0	15	9	0
3b	325	114	0	280	28	0	1355	11	0
3c	36	21	0	29	12	0	120	7	0
4a	13	0	0	15	0	0	89	1	0
4b	97	19	0	143	4	0	331	1	0
5	14	0	0	9	0	0	75	0	0
6	0	0	0	0	0	0	1	0	0
SP	7	0	0	10	1	0	0	0	0

Table 3: *Distribution of tetrahedra among the different configurations at each iteration. Threshold=10%*

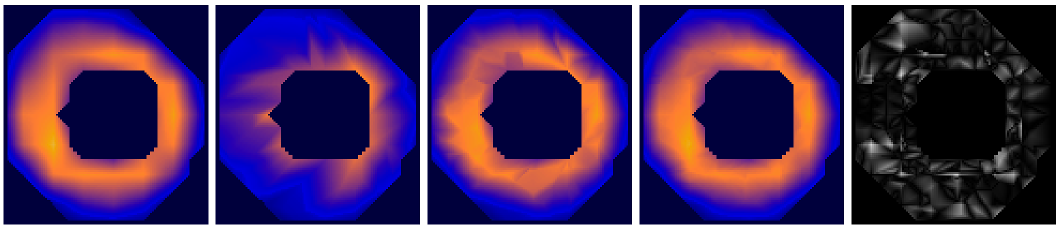


Figure 11: *Perfusion levels of a heart's slice. (a) Original, (b) Original tetrahedral mesh, (c) Threshold=10%, (d) Threshold=5%, (e) Scaled discrepancies by a factor of 20*

The time needed to process a model varies linearly with the number of tetrahedra and the number of tetrahedra with error. At each iteration seeing if a tetrahedron exceeds the threshold costs approximately 0.14ms in an Intel Pentium 4 at 1.70GHz and each tetrahedron that we subdivide costs approximately 1ms.

Finally, we conducted an experiment to measure the quality of the resulting tetrahedra. We measure the quality of the tetrahedra in a standard way, using the *mean ratio*, defined by Liu and Joe [LJ94], that has the advantage of being invariant under traslation, rotation and uniform scaling, and is efficient to compute. For a tetrahedron $\mathscr{T}$, its quality is defined as $\eta = 12(3v)^{2/3}/\sum_{0\leq i<j\leq 3} l_{ij}^2$, where v is the volume of $\mathscr{T}$ and l_{ij}^2 are the lengths of their edges. This measure is always a number between 0 and 1, where 1 corresponds to a regular tetrahedron.

Our last Table 5, shows the resulting qualities for the two threshold values used in our experiments. Notice that in all cases the final qualities are similar (within 8%) to the initial qualities. This is all that can be expected of an algorithm designed to minimize subdivision.

6. Conclusions

A method of adaptive subdivision has been developed to refine a tetrahedral mesh immersed in a voxel model (and which inherits from the voxel model the property values stored at its vertices) until the discrepancy of the volume property of interest computed from the tetrahedral mesh and from the original volume data is below a user-specified threshold.

The subdivision process is local, and requires no propagation from one cell to another. As such, each iteration is computed in a single sweep of the model, and requires no backtracking. Also for this reason, it does not require any additional information about the connectivity of the tetrahedral mesh to be stored.

If the initial mesh is conformal, the algorithm produces a conformal mesh as a result. Although the algorithm is local, its behavior on a facet of a tetrahedron depends solely on the information within that facet, and not the rest of the tetrahedron, which ensures that neighboring tetrahedra will be handled in a consistent way automatically.

The algorithm has shown a fast convergence to the desired discrepancy level in all models tested.

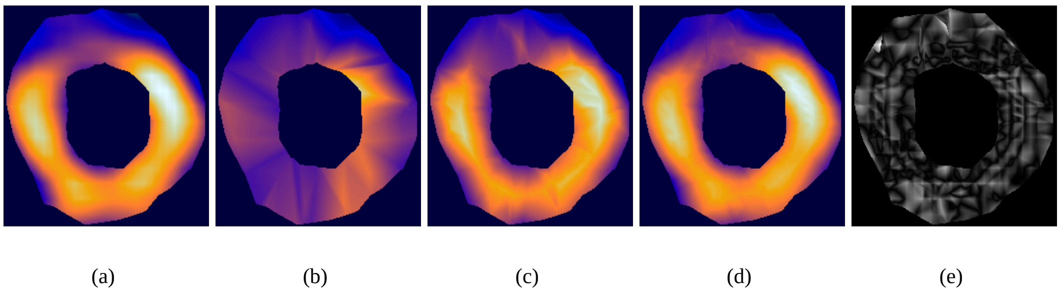


Figure 12: *Perfusion levels of a heart's slice. (a) Original, (b) Original tetrahedral mesh, (c) Threshold=10%, (d) Threshold=5%, (e) Scaled discrepancies by a factor of 15*

References

[Bey95] BEY J.: Tetrahedral grid refinement. *Computing 55*, 4 (1995), 355–378. 2

[BG00] BIELSER D., GROSS M.: Interactive Simulation of Surgical Cuts. In *Proc. of Pacific Graphics* (2000), IEEE CS Press, pp. 116–125. 3

[BSW83] BANK R., SHERMAN A., WEISER A.: Refinement algorithm and data structures for regular local mesh refinement. *Scientific Computing 44* (1983), 3–17. 2

[DFLS02] DANOVARO E., FLORIANI L. D., LEE M., SAMET H.: Multiresolution tetrahedral meshes: an analysis and a comparison. In *Proc. Intern. Conference on Shape Modeling* (2002), IEEE CS Press, pp. 83–91. 2

[FDA02] FOREST C., DELINGETTE H., AYACHE N.: Cutting simulation of manifold volumetric meshes. In *MICCAI* (2002), vol. 2489 of *LNCS*, Springer-Verlag, pp. 235–244. 3

[GCMS01] GANOVELLI F., CIGNONI P., MONTANI C., SCOPIGNO R.: Enabling cuts on multiresolution representation. *The Visual Computer 17*, 5 (2001), 274–286. 3

[GG00] GREINER G., GROSSO R.: Hierarchical tetrahedral-octaedral subdivision for volume visualization. *The Visual Computer 16*, 6 (2000), 357–369. 2

[LJ94] LIU A., JOE B.: On the shape of tetrahedra from bisection. *Mathematics of Computation 63*, 207 (1994), 141–154. 9

[LJ95] LIU A., JOE B.: Quality local refinement of tetrahedral meshes based on bisection. *SIAM Journal on Scientific Computing 16*, 6 (1995), 1269–1291. 2

[LJ96] LIU A., JOE B.: Quality local refinement of tetrahedral meshes based on 8-subtetrahedron subdivision. *Mathematics of Computation 65*, 215 (1996), 1183–1200. 2

[MK00] MOR A., KANADE T.: Modifying soft tissue models: Progressive cutting with minimal new element creation. In *MICCAI* (2000), vol. 1935 of *LNCS*, Springer-Verlag, pp. 598–607. 3

[PC96] PLAZA A., CAREY G.: About local refinement of tetrahedral grids based on bisection. In *Proc. Intern. Meshing Roundtable* (1996), Sandia Corporation, pp. 123–136. 2

[PC00] PLAZA A., CAREY G.: Local refinement of simplicial grids based on skeleton. *Applied Numerical Mathematics 32* (2000), 195–218. 2

[PPSF04] PLAZA A., PADRÓN M., SUÁREZ J., FALCÓN S.: The 8-tetrahedra longest-edge partition of right-type tetrahedra. *Finite Elements in Analysis and Design 41* (2004), 253–265. 2

[PR03] PLAZA A., RIVARA M. C.: Mesh refinement based on the 8-tetrahedra longest-edge partition. In *Proc. Intern. Meshing Roundtable* (2003), Sandia Corporation, pp. 67–78. 2

[RI96] RIVARA M., IRIBARREN G.: The 4-Triangles longest-side partition of triangles and linear refinement algorithms. *Mathematics of Computation 65*, 216 (1996), 1485–1502. 2

[Riv84] RIVARA M.: Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis 21*, 3 (1984), 604–613. 2, 4

[Riv89] RIVARA M.: Selective refinement/derefinement algorithms for sequences of nested triangulations. *Intern. Journal for Numerical Methods in Engineering 28* (1989), 2889–2906. 2

[RL92] RIVARA M., LEVIN C.: A 3d refinement algorithm suitable for adaptive and multigrid techniques. *Communications in Applied Numerical Methods 8* (1992), 281–290. 2

[RM98] RUPRECHT D., MÜLLER H.: A scheme for edge-based adaptive tetrahedron subdivision. In *Visualization and Mathematics*. Springer Verlag, 1998, pp. 61–70. 2

[RNM95] RUPRECHT D., NAGEL R., MÜLLER H.: Spatial free form deformation with scattered data interpolation methods. *Computers & Graphics 19*, 1 (1995), 63–71. 2

[RNV02] RODRÍGUEZ L., NAVAZO I., VINACUA Á.: A tetrahedral model to represent the left ventricle volume of the heart. In *Proc. Vision, Modeling and Visualization* (2002), IOS Press, pp. 249–256. 5, 8

[RNV06] RODRÍGUEZ L., NAVAZO I., VINACUA Á.: *Data-driven Tetrahedral Mesh Subdivision*. Research Report LSI-06-19-R, Universitat Politècnica de Catalunya, 2006. 8

[ZCK97] ZHOU Y., CHEN B., KAUFMAN A.: Multiresolution tetrahedral framework for visualizing regular volume data. In *Proc. IEEE Visualization conference* (1997), pp. 135–142. 2

[Zha95] ZHANG S.: Successive subdivisions of tetrahedra and multigrid methods on tetrahedral meshes. *Houston Journal of Mathematics 21*, 3 (1995), 541–556. 2