ELSEVIER

Applied Numerical Mathematics 32 (2000) 401–418

APPLIED NUMERICAL MATHEMATICS

www.elsevier.nl/locate/apnum

A 3D refinement/derefinement algorithm for solving evolution problems ☆

Angel Plaza [a,*,1], Miguel A. Padrón [a,2,3], Graham F. Carey [b,4,5]

[a] *Department of Mathematics, University of Las Palmas de Gran Canaria, Spain*

[b] *Texas Institute for Computational and Applied Mathematics (TICAM), ASE/EM Department, University of Texas at Austin, Austin, TX, USA*

Abstract

In the present study, a novel three-dimensional refinement/derefinement algorithm for nested tetrahedral grids based on bisection is presented. The algorithm is based on an adaptive refinement scheme and on an inverse algorithm introduced by the authors. These algorithms work first on the *skeleton* of the 3D triangulation, the set of the triangular faces. Both schemes are fully automatic. The refinement algorithm can be applied to any initial tetrahedral mesh without any preprocessing. The non-degeneracy of the meshes obtained by this algorithm has been experimentally shown. Similarly, the derefinement scheme can be used to get a coarser mesh from a sequence of nested tetrahedral meshes obtained by successive application of the refinement algorithm. In this case, the algorithm presents a *self-improvement quality property*: the minimum solid angle after derefining is not less than the minimum solid angle of the refined input mesh. The refinement and derefinement schemes can be easily combined to deal with time dependent problems. These combinations depend only on a few parameters that are fixed into the input data by the user. Here we present a simulation test case for these kind of problems. The main features of these algorithms are summarized at the end.

Keywords: Mesh refinement; Derefinement; 3D bisection; Tetrahedra; Adaptivity

☆ Expanded version of a talk presented at the 15th IMACS World Congress on Scientific Computation, Modeling and Applied Mathematics (Berlin, August 24–29, 1997).

* Corresponding author. E-mail: aplaza@dma.ulpgc.es

1 Supported by grant number PR95-280 from the Dirección General de Investigación, Ciencia y Desarrollo. Spain, and by Proyecto PI1999/146 from Gobierno de Canarias, Spain.

2 Supported by the University of Las Palmas de Gran Canaria. Spain.

3 E-mail: mpadron@aries.dma.ulpgc.es

4 Supported by ARPA grant number DABT63-96-C-0061.

5 E-mail: carey@cfdlab.ae.utexas.edu

0168-9274/00/$20.00
PII: S0168-9274(99)00060-4

1. Introduction

Local refinement is critical to the efficient approximate solution of partial differential equations in many practical applications. Adaptivity of the mesh is particularly important in three-dimensional problems because the problem size and computational cost grow very rapidly as the mesh size is reduced. As it is well known, there are two main steps in local adaptive refinement [9]: the refinement of a subset of elements based on local error indicators [2,3,12], and the achievement of the conformity of the mesh [16,24]. The elements that offer the simplest choice in any dimension are the simplices: triangles in two dimensions, tetrahedra in three dimensions and their analog in even higher dimensions. Many different refinements and improvement techniques for two- and three-dimensional triangulations are now available. Even fractal concepts and iterated function systems have been used [8,20]. For a discussion of different techniques for local grid refinement see [9].

In 2D our refinement algorithm is equivalent to the 4T algorithm of Rivara [26,27,29]. Fig. 1 shows the three possible patterns that can appear by the application of the 4-T algorithm to a single triangle (a) and by the conformity process ((b) and (c)). Note that, if refined, the original triangle is bisected by its longest edge.

In three dimensions several techniques have been developed in the last five years for refining (and coarsening) tetrahedral meshes by means of bisection of tetrahedra. Algorithms based on the *simple* longest edge bisection have been developed by Rivara and Levin [31] and by Muthukrishnan et al. [18]. Since each tetrahedron is divided by its longest edge until the conformity is achieved, it is not known into how many tetrahedra each of the original tetrahedron will be subdivided. Mathematical proofs about the non-degeneracy of the grids created are needed, although experiments suggest this holds. The algorithms of Bänsch [4] and Liu and Joe [15,16] divide each tetrahedron into eight subtetrahedra by iterative edge bisection. Other similar approaches to the problem of refining a tetrahedral mesh can be found in [1,14, 17]. A recursive approach is proposed by Kossaczký [14]. This algorithm imposes certain restrictions and preprocessing in the initial mesh. The 3D algorithm is equivalent to that given in [4]. Recently, Mukherjee [1,17], has presented an algorithm equivalent to [4,15]. However, these algorithms are not applicable to any initial mesh and need some kind of pre-processing.

Recently Plaza and Carey [21,22] have presented a generalization of the 4-T Rivara algorithm to three dimensions. The algorithm has also been studied by Rivara and Plaza [32]. This algorithm works first on the triangular faces of the tetrahedra, the 2-skeleton of the 3D-triangulation, and then subdivides the interior of each tetrahedron in a *consistent* manner with the subdivision of the skeleton. This idea could be applied to develop similar algorithms in higher dimensions. The algorithm can be applied to any initial tetrahedral mesh without any preprocessing. This is an important feature with respect to other similar algorithms in the literature [14–16]. As in the refinement case, the coarsening algorithm is based on

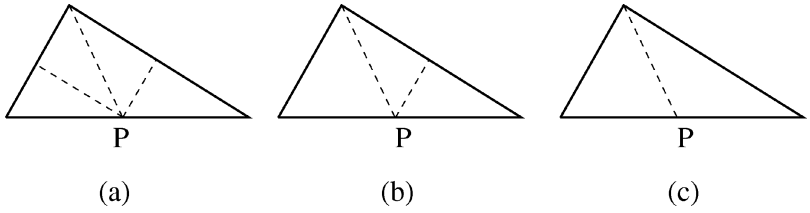


Fig. 1. 4-triangles-refinement patterns.

derefining the 2-skeleton assuring the conformity of the mesh, and then reconstructing the interior of the tetrahedra. For this second task the former 3D (local) refinement patterns can be used in each tetrahedron.

Here, for the first time in 3D, the main ideas for a suitable combination of refinement and coarsening will be presented. The paper is organized as follows. First, some definitions are provided. In the second section the refinement algorithm is summarized. Next, the 2D and 3D derefinement algorithms based on the skeleton are outlined. Finally, the scheme for evolution problems is shown. Some numerical examples are also provided for each algorithm, and the main properties of the algorithms are pointed out.

1.1. Some definitions

Let $V = \{X_0, X_1, \ldots, X_m\}$ be a set of $m+1$ points in $\mathbb{R}^n$ $(1 \leqslant m \leqslant n)$ such that $\{\overrightarrow{X_0 X_i}\colon\ 1 \leqslant i \leqslant m\}$ is a linearly independent set of vectors. Then the closed convex hull of V denoted by $S = \langle V\rangle = \langle X_0, X_1, \ldots, X_m\rangle$ is called an m-simplex in $\mathbb{R}^n$, while the points $X_0, \ldots, X_m$ are called *vertices of S*, and the number m is said to be the *dimension* of S.

Let Ω be a bounded set in $\mathbb{R}^n$ with non-empty interior, $\overset{\circ}{\Omega} \neq \emptyset$, and polygonal boundary $\partial\Omega$, and consider a partition of Ω into a set $\tau = \{t_1, \ldots, t_i\}$ of n-simplices, such that any adjacent simplex elements share an entire face or edge or a common vertex, i.e., there are no *non-conforming* nodes in τ. Then we can say that τ is a conforming simplex mesh or a conforming triangulation for Ω.

Let τ be an n-simplicial mesh. The set $\mathrm{skt}(\tau) = \{f\colon\ f$ is an $(n-1)$-face of some $t \in \tau\}$ will be called *the skeleton* or the $(n-1)$-skeleton of τ [6]. For instance, the skeleton of a triangulation in three dimensions is comprised of the faces of the tetrahedra, and in two dimensions the skeleton is the set of the edges of the triangles. It should be noted however, that the skeleton can be understood as a new triangulation: if τ is a 3-dimensional conforming triangulation in $\mathbb{R}^3$, $\mathrm{skt}(\tau)$ is a 2-dimensional triangulation embedded in $\mathbb{R}^3$. Furthermore, if τ is conforming, $\mathrm{skt}(\tau)$ is also conforming.

Two (conforming) triangulations τ and τ^* of the same bounded set Ω are said to be *nested*, and we write $\tau < \tau^*$ if the following condition holds: $\forall t \in \tau,\ \exists\, t_1, \ldots, t_p \in \tau^*$ such that $t = t_1 \cup \cdots \cup t_p$. We also say that τ is coarser than τ^* or that τ^* is finer than τ.

Note that if over some initial mesh τ successive refinements by bisection are performed, a sequence of nested triangulations are obtained. In a sequence of nested triangulations, the respective sets of nodes are also nested. This fact enables us to talk about *proper* nodes and *inherited* nodes as follows: let $T = \{\tau_1 < \tau_2 < \cdots < \tau_k\}$ be a sequence of nested grids, where τ_1 represents the initial mesh, and let τ_j be any triangulation of T. One node $N \in \tau_j$ is called a *proper node* of τ_j or a *j-new node* if N does not belong to any previous mesh. Otherwise, N is said to be an *inherited node* in τ_j. The edges, faces, and elements may be named similarly [23,28].

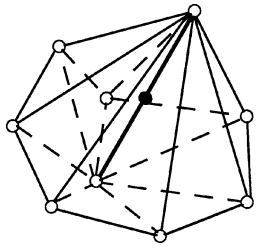


Fig. 2. Hull of an edge relative to the tessellation.

Since we use refinement by bisection of the elements, all the new nodes will appear at the midpoints of edges of previous levels of mesh. We call the *surrounding edge* of a node N the edge in which N is at the midpoint. Note that for each proper node $N \in \tau_j$, its surrounding edge e belongs to the previous mesh level, $e \in \tau_{j-1}$. Furthermore, for each edge e we call the *hull of* e, denoted by $h(e)$, the set of elements sharing edge e. Note that this set is not convex in general. In Fig. 2 the surrounding edge of a node (in black) and the hull of the edge are drawn. If the edge has a node at its midpoint, as in the figure, the set of the *extreme nodes* of its hull (open circles in the figure) will be called the *molecule* for the black node of the figure. Observe that this definition can be extended to any dimension.

2. The refinement procedure: Algorithm I

2.1. The refinement algorithm in 2D

Our approach for development of a 3D refinement algorithm is based on applying 2D refinement ideas to the skeleton of the tetrahedral grid. For a better understanding of the 3D refinement algorithm we first show how the 2D version works. Fig. 3(a) represents an initial triangulation in which triangle t has to be subdivided. The first step of the algorithm is the subdivision of the edges of t, Fig. 3(b). Then the adjacent triangle t^* is checked to make it conforming. The process ends when no further edge in any adjacent triangle is divided for conformity.

Let us recall the way in which the conformity of the mesh is assured by the 2D refinement algorithm. Let τ be the initial triangulation, let t be a triangle to be subdivided, and let L be the list of triangles to be subdivided.

/* Input variables: t, triangle to be refined, and τ, 2D triangular mesh
Output variables: L, list of triangles to be refined
Internal variables: t^*, triangle; $l(t)$, $l(t^*)$ longest-edges of triangles t and t^*, respectively */
$L = \{\emptyset\}$
Add t to the list L: $L = L \cup t$
For each edge e of t, **do**
Subdivide e
/* let t^* be the neighboring triangle of t by the edge e,
and let $l(t^*)$ be the longest-edge of t^* */
While $l(t) \neq l(t^*)$, **do**
Add t^* to the list L: $L = L \cup t^*$
Subdivide edge $l(t^*)$
$l(t) := l(t^*)$; $t := t^*$
/* let t^* be the neighboring triangle of t by the edge $l(t)$
and $l(t^*)$ be the longest-edge of t^* */
End While
End For

The way in which conformity in 2D is ensured is shown in Fig. 4. The arrows indicate the process of checking the conformity in the adjacent triangle through each edge.

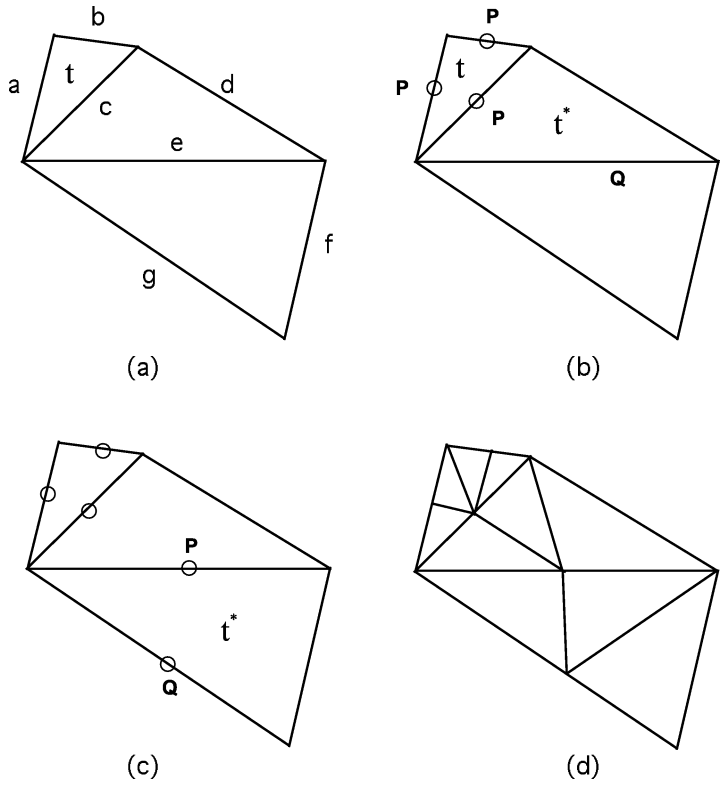


Fig. 3. 2D refinement algorithm.

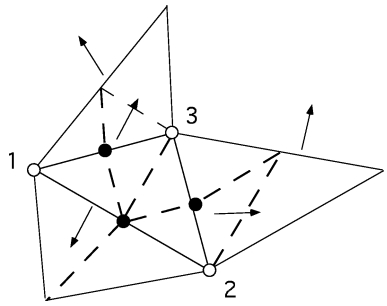
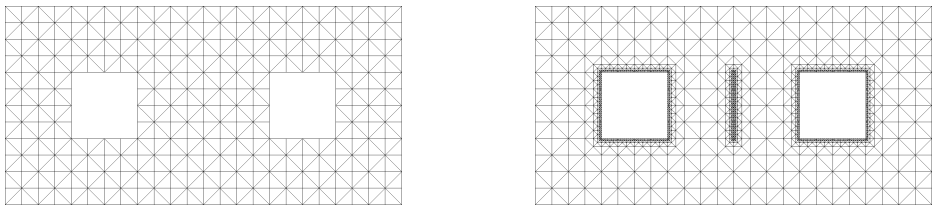


Fig. 4. Local conformity tests when refining.

In order to show the behavior of the refinement algorithm in 2D, we present in the Fig. 5 an example of refinement in a strong non-convex domain.

2.2. Outline of the 3D refinement algorithm

The 3D algorithm as proposed by Plaza and Carey [25] is:
Let τ be a three-dimensional tetrahedral grid and t_0 be a tetrahedron in the grid to be refined.

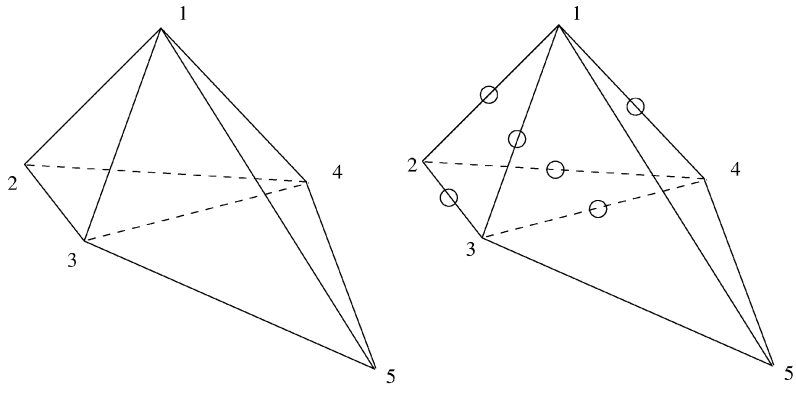


(a) Initial mesh (b) Final mesh

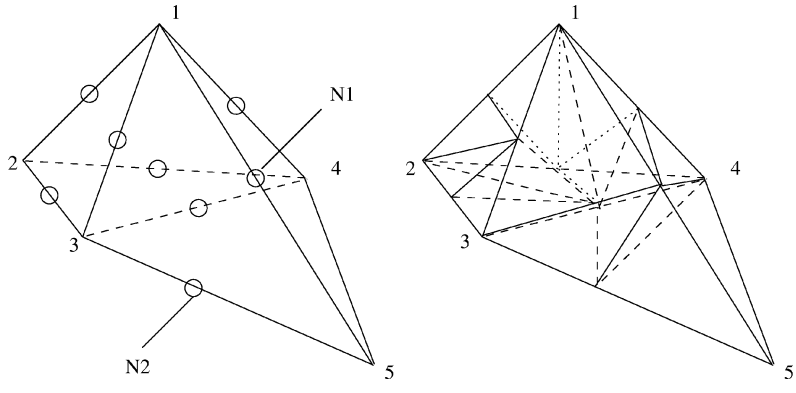
Fig. 5. Local refinement in a non-convex domain.

/* Input variables: t_0, tetrahedron to be refined, and τ, 3D triangular mesh
Output variables: new mesh τ
Internal variables: L, set of nodes; N_e, P, $P_{e^\star}$, nodes; e, e_P, $e^\star$, edges; f, face; t, tetrahedron */
/* *1. Edge subdivision* */
$L = \{\emptyset\}$
For each edge $e \in t_0$ **do**
Bisect e producing *new-node* N_e
$L = L \cup N_e$
End For
/* *2. The conformity is ensured* */
While $L \neq \emptyset$ **do**
/* Let P be a node from L with surrounding edge e_P */
For each tetrahedron $t \in h(e_P)$ **do**
For each non-conforming face f of t **do**
Bisect the longest-edge $e^\star$ of f producing *new-node* $P_{e^\star}$ at the midpoint of $e^\star$
$L = L \cup P_{e^\star}$
End For
End For
$L = L - P$
End While
/* *3. The subdivision of the skeleton is performed* */
For each triangular face $f \in \mathrm{skt}(\tau)$ to be subdivided **do**
Subdivide f
End For
/* *4. The subdivision of the tetrahedra is performed* */
For each tetrahedron $t \in \tau$ to be subdivided **do**
Subdivide t
End For
End.

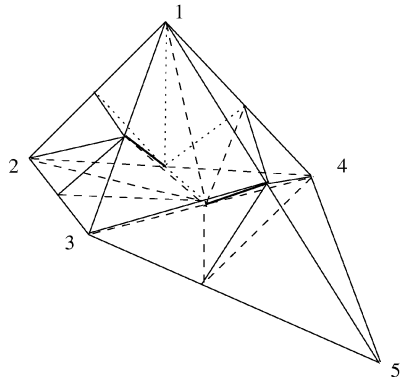
As an example, Fig. 6 shows the application of the algorithm to a very simple initial mesh comprised of only 2 tetrahedra (Fig. 6(a)). We assume that tetrahedron $\langle 1, 2, 3, 4\rangle$ is to be refined based on some



(a) Initial mesh (b) Step 1: Edge subdivision



(c) Step 2: Conformity (d) Step 3: Skeleton division



(e) Final mesh

Fig. 6. Example of application of the 3D algorithm. The initial mesh is given in (a) and the edges of one tetrahedron are bisected in (b). New edges in the neighboring tetrahedron are bisected in (c) and the surface triangles (skeleton) are refined in (d). The interior of each tetrahedron is defined in (e) to yield the final mesh.

error indicator. The edges of that tetrahedron are subdivided in Fig. 6(b) and the edges of the adjacent tetrahedron are divided to satisfy conformity (see Fig. 6(c), in which nodes $N1$ and $N2$ are introduced). Fig. 6(d) presents the division of the skeleton and Fig. 6(e) the final mesh in which the new tetrahedra have been defined.

In the final stage, step 4 of the algorithm, only the local subdivision of each 3-simplex of the original mesh must be made. An iterative method based on bisection by the edges as in [16] could be used.

Some of the properties of the previous algorithm are summarized at the end of the paper. Next we underline the linear complexity in the number of nodes $\mathrm{O}(N)$.

2.3. *Efficiency of the algorithm*

First step, *edge subdivision*, has a complexity of $\mathrm{O}(1)$.

Assuring the conformity, step 2, has a complexity of $\mathrm{O}(N_a)$ since it is accomplished locally, for each node in the set L as the procedure below shows. N_a is the number of added nodes at refinement. Note that this task can be performed locally by means of vectors *surrounding edge* $\mathrm{srr}(N)$, and *supporting face* $\mathrm{spp}(e)$.

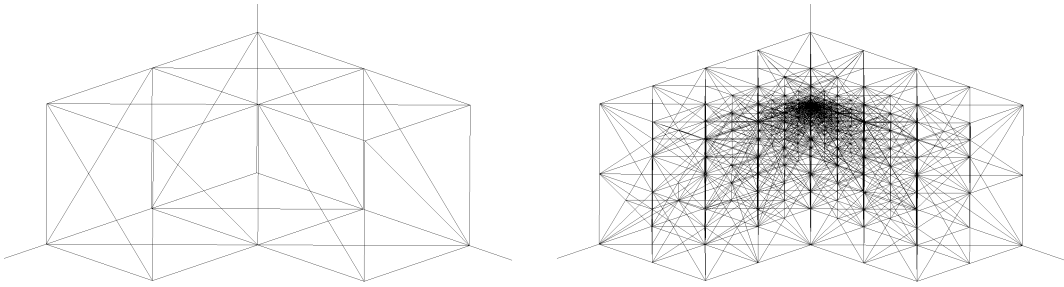
Procedure *Find-hull*

/* Input variables: N, node
Output variables: $h(e)$, list of tetrahedra
Internal variables: e, edge; f, f^*, faces; t_1, t_2, t^*, tetrahedra */
$e = \mathrm{srr}(N)$
$h(e) = \{\emptyset\}$
$f = \mathrm{spp}(e)$
/* Let t_1 and t_2 be the two tetrahedra sharing face f */
While $t_1 \neq t_2$ **do**
$h(e) = h(e) \cup t_1$
/* Let f^* be the face in t_1 sharing edge e, with $f^* \neq f$
and t^* the neighbor tetrahedron of t_1 through f^* */
$t_1 = t^*$
$f = f^*$
End While
$h(e) = h(e) \cup t_1$
End.

In the previous procedure, for simplicity, it has been supposed that the edge e is an internal edge to the domain Ω. The cases in which edge e is *external*, that is e belongs to the boundary of Ω, or the supporting face of e, $f = \mathrm{spp}(e)$, is in the boundary of Ω, can be studied in an analogous way.

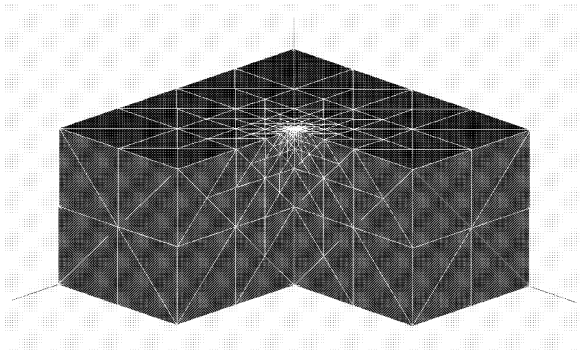
Following the 3D algorithm, steps 3 and 4 present a complexity of $\mathrm{O}(N_f)$ and $\mathrm{O}(N_t)$ respectively, where N_f is the number of involved faces, and N_t the number of involved tetrahedra. Hence the complexity of the algorithm can be estimated by $\mathrm{O}(N_a) + \mathrm{O}(N_f) + \mathrm{O}(N_t)$.

In the case of an initial triangulation in which the number of faces and the number of tetrahedra are of the same order as the number of nodes, linear complexity in the number of nodes follows. Note, that this



(a) Initial mesh

(b) After 13 refinement iterations



(c) Trace on the surface

Fig. 7. Example of a very local refinement.

is the general case for the meshes used in finite element applications (see, for example, [7, p. 75; 10]). The relation between the numbers of topological entities in real meshes is studied in detail in [5]. When the global refinement proceeds n times, both the number of vertices and the number of tetrahedra, tend to be of the same order. This same order of magnitude is implied by the result: $\lim_{n\to\infty}(4T_n/N_n) = 24$ (see [32]), where T_n is the number of tetrahedra after n global refinements, and N_n is the number of nodes after n global refinements. Numerical experiments also indicate that in practice the algorithm performs like a linear complexity algorithm [21].

2.4. A 3D example of refinement

We present one numerical example to show the behavior of our refinement algorithm. Fig. 7(a) shows an initial coarse grid that is locally refined to obtain the final grid in Fig. 7(b), which contains 615 nodes and 2704 tetrahedra. Fig. 7(c) shows the trace of the refinement on the surface of the domain. Here a punctual singularity is supposed in the top vertex of the reentrant edge. Note that a very local refined area can be achieved by the algorithm. Since the refinement algorithm works first on the skeleton of

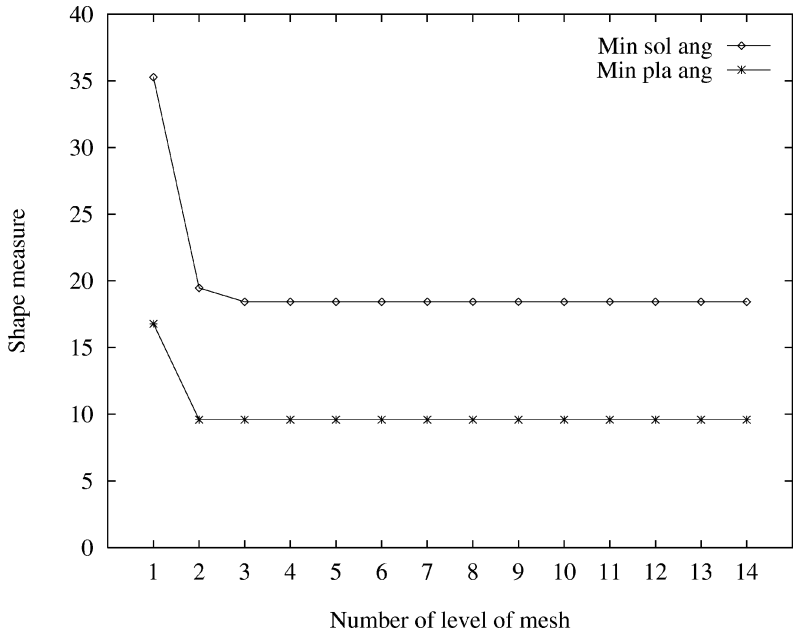


Fig. 8. Evolution of the shape measure when refinement proceeds.

the triangulation, and then on the interior of each involved tetrahedron, the refinement algorithm can be applied to any given tetrahedral mesh, without any restriction, even to strong non-convex regions. Non-convexity is not an issue for the algorithm.

To show the evolution of the shape measure at refining in the previous example, we have calculate the minimum planar angle in the mesh (see Fig. 8). Furthermore, we have used, for each vertex P, following [31], the measure Φ_P associated with the solid angle at P:

$$\Phi_P = \sin^{-1}\left(1 - \cos^2\alpha_P - \cos^2\beta_P - \cos^2\gamma_P + 2\cos\alpha_P\cos\beta_P\cos\gamma_P\right)^{1/2},$$

where $\alpha_P, \beta_P, \gamma_P$ are the associated corner angles, to define $\Phi_t = \min\{\Phi_P\colon\ P \in t\}$ as a shape measure for each tetrahedron t. The minimum of these numbers over all the tetrahedra of each level of mesh is also depicted in the Fig. 8. It should be noted here that, as in all the examples of refinement, after a few refinements in which the shape measure for the mesh goes down, it seems to attain a fixed value greater than zero. Experiments show a similar behavior of our refinement algorithm to other algorithms like that of Rivara and Levin [31] or Liu and Joe [15,16]. See [32] for a comparative study. A general study on mesh quality and optimization has recently been presented in [33].

3. The derefinement procedure: Algorithm II

For coarsening a refined mesh we may construct an inverse algorithm based on the previous refinement scheme. This new algorithm is the inverse of the refinement one. However, it is more difficult since now all levels of mesh are involved.

One item we have to take into account during derefining is the *genealogy* of the edges, faces or elements. That is, we need to know what the *children-edges* and *parent-edge* for each edge are, and similarly for faces and elements.

The derefinement condition is the condition that a node N must be satisfied if it is to be removed. Here we have used the relative (and at other times the absolute) difference between the numerical solution in a particular node and the average of the numerical solution at the extreme nodes of its surrounding edge. However, many strategies can be devised for the derefinement condition. More complicated formulas involving approximate second derivatives of the solution can also be applied. Derefinement conditions can be used over the tetrahedra instead of between solution at nodes. For example, the same error indicator used at refining the mesh could be used at the derefinement stage.

To each topological element of the mesh (node, edge, face or tetrahedron) we assign an integer number. This number gives us the level in which the corresponding element is proper. Besides we use the sign of these numbers to control the derefinement procedure. That is, if the number is positive, the corresponding topological element must remain. On the other hand, if the number is negative it implies that this element must be removed. The vector in which for each type of topological elements these numbers are kept is called, *derefinement vector*.

Finally, a proper node N will be called an *eligible* proper node for derefining if N can be removed from the mesh while attending to the conformity condition. This means, that a particular node N is eligible if N can be removed from the mesh and the mesh remains conforming, and N is *not-eligible* if its removal makes the mesh non-conforming. Hence, the conformity of each level is assured by maintaining some nodes that, otherwise, given the derefinement condition, might have been removed.

3.1. The derefinement algorithm in 2D

Let $T = \{\tau_1 < \tau_2 < \cdots < \tau_k\}$ be a sequence of nested two-dimensional grids, where τ_1 represents the initial mesh and τ_k the finest mesh in the sequence. Derefining the sequence means the generation of a new sequence: $T^m = \{\tau_1 < \tau_2' < \cdots < \tau_m'\}$, where $m \leqslant k$. An outline of this algorithm has been proposed in [24] in the form:

```
/* Input variables: Refined sequence of meshes T = {τ1 < τ2 < ··· < τk}
   Output variables: New derefined sequence T^m = {τ1 < τ2' < ··· < τm'}
   Internal variables: N, P, nodes; c, edge; t, triangle; derefinement indicators */
/* Loop in levels of T */
For j = k to 2, do
   For each eligible proper node N ∈ τj, do
      /* 1. The derefinement condition is evaluated and the nodes and edges are marked */
      If node N must remain, then
         /* 2. Conformity is assured locally */
         /* let c be the surrounding edge of N */
         For each neighboring element t of c, do
            If t is non-conforming then
               Change the derefinement indicator for the node P of its longest
               edge, and the nodes of the molecule of P must remain
            End If
```

End For
End If
End For
/* *3. Redefinition of the mesh* */
If some proper node of τ_j remains **then**
3.a. New nodal connections are defined, and genealogy vectors are modified
In other cases
3.b. The current level j is deleted in the data structure
End If
4. The changes are inherited by the following meshes
/* A new sequence of nested meshes is obtained */
End For.

Note that the conformity is ensured locally by modifying the neighboring elements of the surrounding edge of each proper node. An example of the application of the derefinement to a sequence of five levels of mesh is shown in Fig. 9. In the figure, the first column on the left represents the input sequence $T = \{\tau_1 < \tau_2 < \cdots < \tau_5\}$ of the algorithm. The next three columns show the result of the process as it progresses derefining the input from the finest level (τ_5) to the third one (τ_3). This is highlighted surrounding the level of mesh by a dashed line. In the figure, black nodes mean that they have to remain in the mesh for the sake of conformity, while white nodes are *eligible proper* nodes, that is, *proper* nodes in this level of mesh available for derefinement.

3.2. The derefinement condition

The derefinement condition we are using in 3D is the same as that which has been used in 2D [11]. A proper node may be removed if the absolute difference between the values in this node of the numerical solution an its corresponding interpolated function is less than a sufficiently small parameter $\varepsilon > 0$. That is, if u_h is the numerical solution for a given mesh and u_h^i is the interpolated function of u_h in the derefined mesh, we will get

$$\left\|u_h - u_h^i\right\|_\infty = \sup_x \left|u_h(x) - u_h^i(x)\right| < \varepsilon.$$

This error indicator at derefining does not allow us to control the discretization error; in an adaptive algorithm this control is usually performed by an error indicator in the refinement process. The subjacent idea is that when a time-step integration scheme is used, a good approximation of the solution when time is $t_{n+1} = t_n + \Delta t_n$ is the previous solution when time is t_n. So the described error indicator at derefining can be considered optimal in the sense that a given solution is approximated with a minimum number of nodes after derefining. If $\delta > 0$ is a given tolerance for the error in the maximum norm, a practical criterion to choose ε would be to take a value sufficiently smaller than δ, for example $\varepsilon \approx 0.1\delta$. Similarly, one may choose for ε a small fraction of $\|u\|_\infty$ or, another characteristic value according to the problem or to the range of the expected solution.

However, it should be noted that the algorithms are independent from the refinement or derefinement criteria involved. So the same error indicator used at refining could be used as a derefinement error indicator as well.

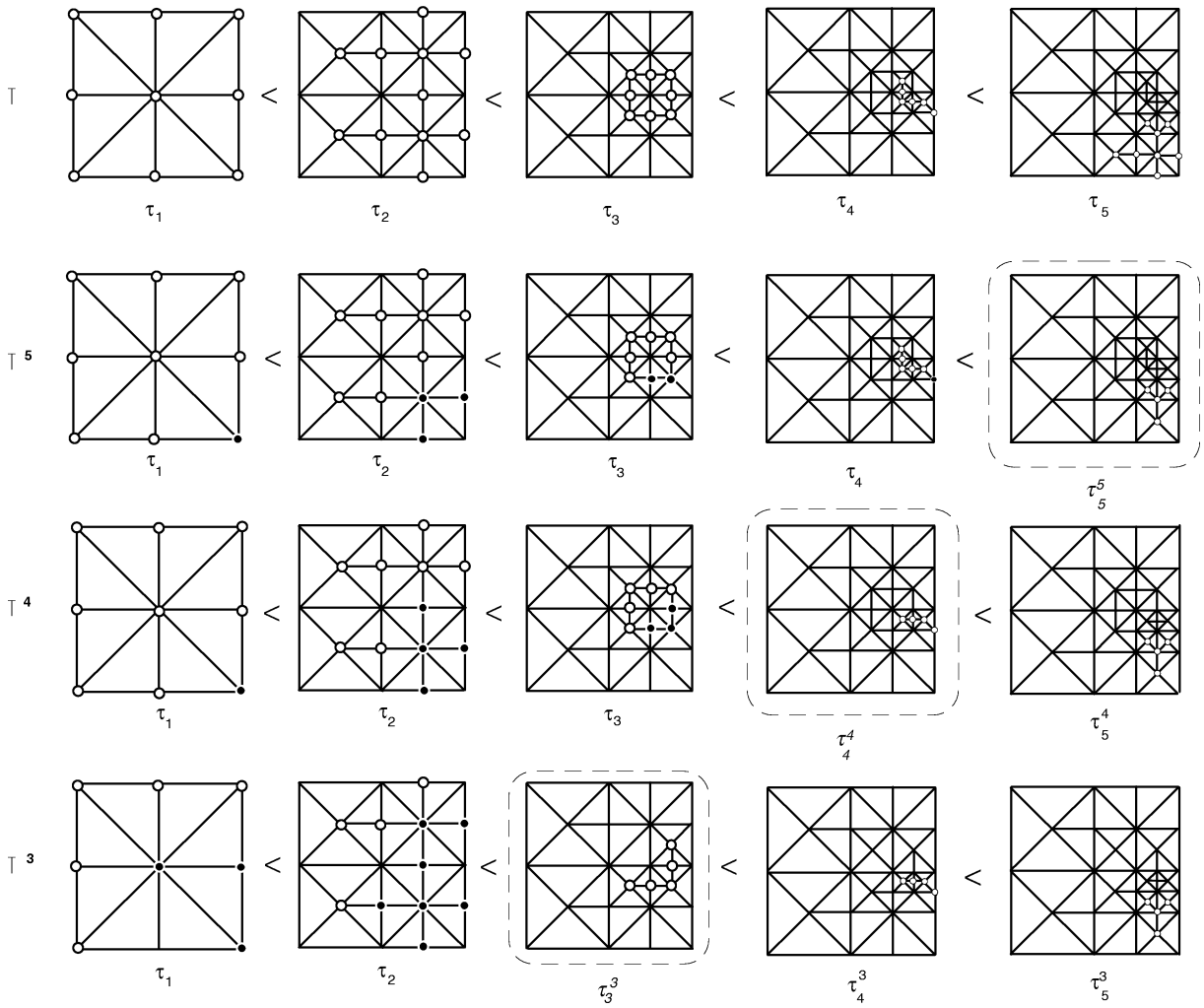


Fig. 9. Derefinement process in 2 dimensions (see the text for an explanation).

3.3. The derefinement algorithm in 3D

As in the 2D case, the concept of adjacency is the central idea in the algorithm in 3D. In order to check the elements belonging to the *hull* of the edge e we go from that edge to a face f, called the *supporting face* for e, in which e is an edge, and from this face to the neighboring elements of the face. In this way we can take all the elements in $h(e)$, as it has been explained in Section 2.3.

/* Input variables: Refined sequence of meshes $T = \{\tau_1 < \tau_2 < \cdots < \tau_k\}$
Output variables: New derefined sequence $T^m = \{\tau_1 < \tau_2' < \cdots < \tau_m'\}$
Internal variables: N, node; e, edge; $h(e)$, hull of e; f, triangular face; t, tetrahedron;
derefinement indicators */
/* Loop in levels of T */

For $j = k$ to 2, **do**
  For each *eligible proper* node $N \in \tau_j$, **do**
    /* 1. *The derefinement condition is evaluated* */
    1.1. The derefinement condition is checked
    1.2. The nodes and edges are pointed out
    /* 2. *The conformity is ensured locally* */
    /* let e be surrounding edge of N, and $h(e)$ the $hull$ of e */
    For each tetrahedron $t \in h(e)$, **do**
      make N conforming in t
    End For
  End For
  /* 3. *The sequence of meshes is re-defined* */
  For each $f \in \mathrm{skt}(\tau_{j-1})$, **do**
    3.1 Subdivide f by the 4-T partition of Rivara
  End For
  For each $t \in \tau_{j-1}$, **do**
    3.2 Perform the new subdivision of t
  End For
End For.

Note that the algorithm is comprised of two main stages: the application of the derefinement algorithm to the skeleton, and the actual redefinition of the interior of the tetrahedra. For this second task, the corresponding part of the 3D refinement algorithm is used.

An example of application of the 3D derefinement algorithm is presented in Fig. 10. There the first column corresponds to the input sequence in the algorithm and the second one to the final sequence. Note that each level of mesh in the derefined sequence is coarser than the analogous one in the input sequence.

4. The refinement/derefinement combination: Algorithm III

By combining the above two strategies we obtain the composite refinement/derefinement scheme. This algorithm can be outlined as follows:

Initial mesh generation and set up of the parameters for refining and derefining
For nsteps $= 1$ to $N_{\max}$ **do**
  For $i = 1$ to N_r **do**
    1. Computation of the new timestep $\Delta_i(t)$
    2. Solution of the corresponding system of equations
    3. Computation of the refinement error indicator
    4. Local refinement (Algorithm I)
  End For
    5. Derefinement of the mesh, with tolerance ε (Algorithm II)
End For.

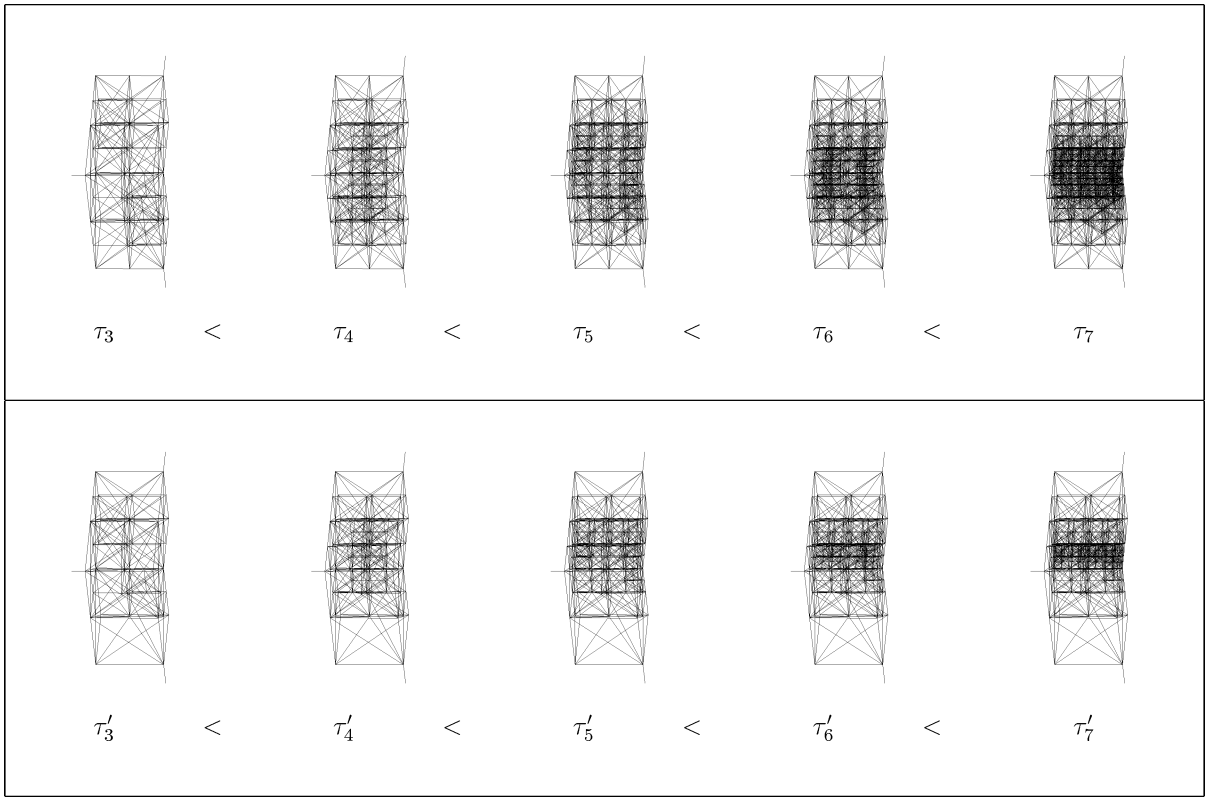


Fig. 10. Derefinement process in 3 dimensions. First row represents a sequence of refined meshes. Second row is for the corresponding derefined sequence.

4.1. Properties

Several properties are noteworthy:

(1) The algorithm only depends on a few parameters: $N_{\max}$, N_r, $\Delta_i(t)$, the refinement parameter γ, and the derefinement parameter ε.
(2) Both refinement and derefinement procedures are finite and exhibit linear complexity $O(N)$, where N is the number of nodes [22].
(3) The refinement procedure can be applied to any initial triangulation, even to strong non-convex regions, without any preprocessing [19].
(4) After the first applications of the refinement algorithm, the quality of the meshes seems to attain an asymptotic limit greater than zero, and the non-degeneracy is proved experimentally, but not yet mathematically.
(5) After some refinements around a particular vertex, the solid angles sharing that vertex are not divided any more. *Fractal property of the algorithm* [32].
(6) The derefinement condition is evaluated in a *minimum* number of nodes: the *eligible proper nodes*.

(7) Since the derefinement algorithm is the inverse algorithm of a refinement algorithm based on bisection, the non-degeneracy of the meshes obtained at derefining is proved. In other words, after derefining a mesh the minimum solid angle is not less than the minimum solid angle of the refined input mesh. *Non-degeneracy property at derefining* [32].
(8) The grids obtained at refining or derefining are nested. *Nestedness of the triangulations*. The nested nature of the grids makes the use of multigrid methods relatively easy [11,13].
(9) The derefinement algorithm can be combined with global refinement to obtain a local refinement procedure. Here the computed solutions on successively refined meshes determine a robust error indicator for local coarsening although computational overhead is higher.
(10) Many strategies can be devised for the derefinement condition. For example, the relative (or absolute) difference between the numerical solution in a particular node and the average of the numerical solution at the extreme nodes of its surrounding edge is a simple indicator. More complicated formulas involving second derivatives can also be applied.
(11) In the resolution of an evolutionary process, removing dupe nodes keeps the number of equations bounded. This is an important feature because in a finite element code most of the CPU time is employed in the resolution of the associated system of equations [11]. As it was proved in 2D, the derefinement procedure uses less than 1.0% of the total CPU time in an evolution (convection–diffusion) problem [11].
(12) The main idea in the development of these algorithms that is the use of the skeleton to find a refinement and derefinement algorithms in one dimension higher, could be applied to obtain similar algorithms in four dimensions.

5. Numerical examples

Many examples combining refinement and derefinement for two dimensional problems can be found in the literature (see, for example, [11,34]). We present here only one simulation example in 3D. Fig. 11 shows the evolution of meshes when a combination of (local) refinement followed by a derefinement algorithm is applied to get the moving refinement area. Note how the refinement area changes as the singularity moves.

6. Concluding remarks

The refinement and coarsening algorithms presented here provide a very useful tool for the treatment of unsteady problems in three dimensions. Adaptivity of the mesh is particularly important in three-dimensional problems because problem size and computational cost grow very rapidly as the mesh size is reduced. With the refinement/derefinement combination families of sequences of nested meshes are obtained. Furthermore, the fact of using nested grids enables us to use the multigrid method in an easy way to solve the system of equations associated with the finite-element method [13].

These ideas are not only important in developing efficient methods for solving partial differential equations (PDE), but are clearly relevant in other areas such as approximation of surfaces, visualization, data compression and solid modeling.

There are still several open questions related to a mathematical proof of the non-degeneracy of the meshes obtained, and the existence of a bounded number of similarity classes (that, perhaps, depend

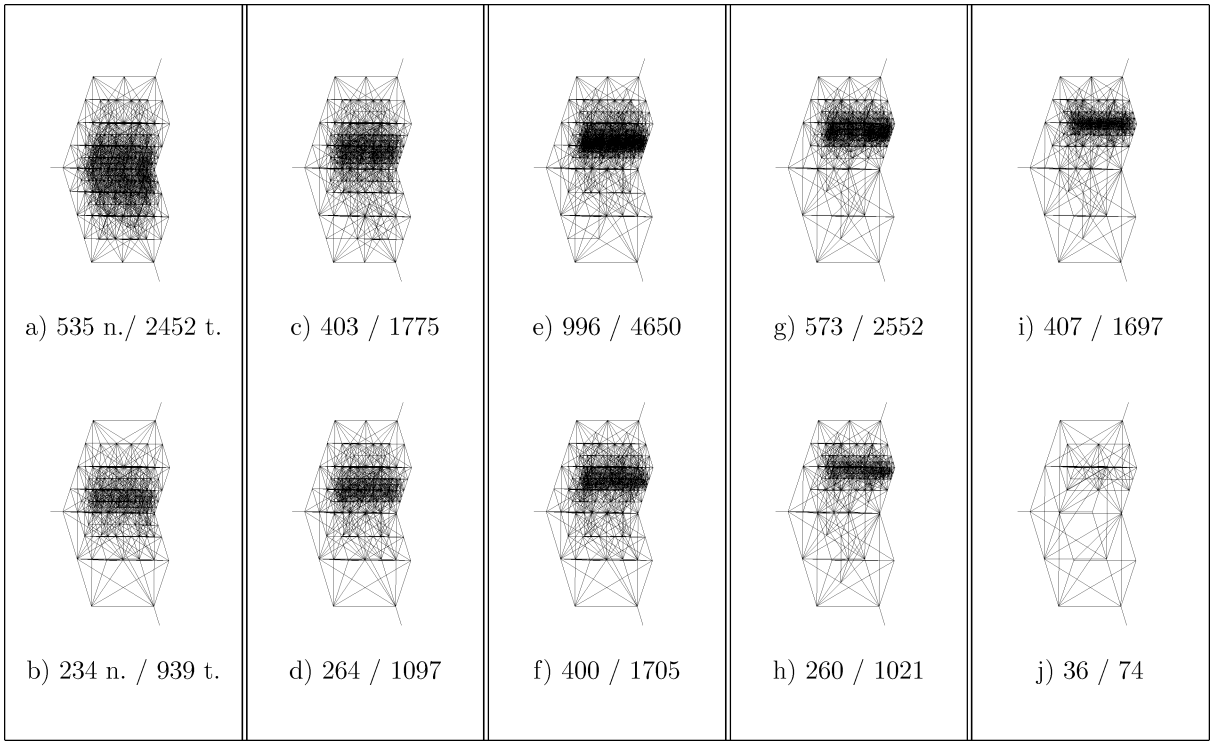


Fig. 11. Simulation example in 3 dimensions. After approximating the singularity the mesh is derefined. The sequence must be read from a) to j). The number of nodes & tets involved are indicated.

on the geometry of the initial 3D triangulation). Although theses properties have been proved in two-dimensions [30], the generalization to three dimensions is not yet solved.

References

[1] D.N. Arnold, A. Mukherjee, L. Pouly, Locally adapted tetrahedral meshes using bisection, SIAM J. Sci. Comput. (1997).
[2] I. Babuška, W.C. Rheinboldt, Error estimates for adaptive finite element computations, SIAM J. Numer. Anal. 15 (1978) 736–754.
[3] I. Babuška, W.C. Rheinboldt, A posteriori error analysis of finite element solutions for one-dimensional problems, SIAM J. Numer. Anal. 18 (1981) 565–589.
[4] E. Bänsch, Local mesh refinement in 2 and 3 dimensions, IMPACT Com. Sci. Engrg. 3 (1991) 181–191.
[5] M.W. Beall, M.S. Shephard, A general topology-based data structure, Internat. J. Numer. Methods Engrg. 40 (1997) 1573–1596.
[6] M. Berger, Geometry, Springer-Verlag, 1987.
[7] M. Bern, D. Eppstein, Mesh Generation and Optimal Triangulation, World Scientific, 1992.
[8] S.W. Bova, G.F. Carey, Mesh generation/refinement using fractal concepts and iterated function systems, Internat. J. Numer. Methods Engrg. 33 (1992) 287–305.
[9] G.F. Carey, Computational Grids: Generation, Refinement and Solution Strategies, Taylor and Francis, 1997.

[10] R.A. Dwyer, Higher-dimensional Voronoi diagrams in linear expected time, in: Proc. 5th Annual ACM Symp. on Comp. Geometry, ACM, 1989, pp. 236–333.
[11] L. Ferragut, R. Montenegro, A. Plaza, Efficient refinement/derefinement algorithm of nested meshes to solve evolution problems, Comm. Numer. Methods Engrg. 10 (1994) 403–412.
[12] J.P.S.R. Gago, D.W. Kelly, O.C. Zienkiewicz, A posteriori error analysis and adaptive processes in the finite element method, Part II: Adaptive mesh refinement, Internat. J. Numer. Methods Engrg. 19 (1983) 1621–1656.
[13] W. Hackbush, Multigrid Methods and Applications, Springer Verlag, 1985.
[14] I. Kossaczký, A recursive approach to local mesh refinement in two and three dimensions, J. Comput. Appl. Math. 55 (1994) 275–288.
[15] A. Liu, B. Joe, On the shape of tetrahedra from bisection, Math. Comp. 63 (1994) 141–154.
[16] A. Liu, B. Joe, Quality local refinement of tetrahedral meshes based on bisection, SIAM J. Sci. Comput. 16 (1995) 1269–1291.
[17] A. Mukherjee, An adaptive finite element code for elliptic boundary value problems in three dimensions with applications in numerical relativity, Ph.D. Thesis, Pennsylvania State University, 1996.
[18] E. Muthukrishnan, P.S. Shiakolas, R.V. Nambiar, K.L. Lawrence, Simple algorithm for adaptive refinement of three-dimensional finite element tetrahedral meshes, AIAA Journal 33 (1995) 928–932.
[19] M.A. Padrón, A 3D derefinement algorithm for tetrahedral nested meshes based on the squeleton, Ph.D. Thesis, University of Las Palmas de Gran Canaria, 1999. In Spanish.
[20] A. Plaza, The fractal behaviour of triangular refined/derefined meshes, Comm. Numer. Methods Engrg. 12 (1996) 295–302.
[21] A. Plaza, G.F. Carey, About local refinement of tetrahedral grids based on bisection, in: 5th Inter. Mesh. Roundtable, Sandia Corporation, 1996, pp. 123–136.
[22] A. Plaza, G.F. Carey, Refinement of simplicial grids based on the skeleton, Appl. Numer. Math. 32 (2) (2000) 195–218.
[23] A. Plaza, L. Ferragut, R. Montenegro, Derefinement algorithms of nested meshes, in: J. van Leeuwen (Ed.), Algorithms, Software, Architecture, Elsevier Science/North-Holland, 1992, pp. 409–415.
[24] A. Plaza, R. Montenegro, L. Ferragut, An improved derefinement algorithm of nested meshes, in: M. Papadrakakis (Ed.), Advances in Post and Preprocessing for Finite Element Technology, Civil-Comp Ltd., 1994, pp. 175–180.
[25] A. Plaza, M.A. Padrón, G.F. Carey, A 3d derefinement algorithm for tetrahedral grids, in: McNU'97, Trends in Unstructured Mesh Generation, 1997, pp. 17–23.
[26] M.C. Rivara, Mesh refinement based on the generalized bisection of simplices, SIAM J. Numer. Anal. 2 (1984) 604–613.
[27] M.C. Rivara, A grid generator based on 4-triangles conforming mesh refinement algorithms, Internat. J. Numer. Methods Engrg. 24 (1987) 1343–1354.
[28] M.C. Rivara, Selective refinement/derefinement algorithms for sequences nested triangulations, Internat. J. Numer. Methods Engrg. 28 (1989) 2889–2906.
[29] M.C. Rivara, Local modification of meshes for adaptive and/or multigrid finite-element methods, J. Comput. Appl. Math. 36 (1991) 79–89.
[30] M.C. Rivara, G. Iribarren, The 4-triangles longest-side partition of triangles and linear refinement algorithms, Math. Comp. 65 (1996) 1485–1501.
[31] M.C. Rivara, C. Levin, A 3-d refinement algorithm suitable for adaptive and multi-grid techniques, J. Comm. Appl. Numer. Methods 8 (1992) 281–290.
[32] M.C. Rivara, A. Plaza, Mesh refinement/derefinement based on the 8-tetrahedra longest-edge partition, Math. Comp. (1999 submitted).
[33] P.M. Knupp, Matrix norms and the condition number, in: 8th Inter. Mesh. Roundtable, Sandia Corporation, 1999, pp. 13–22.
[34] R. Montenegro, A. Plaza, L. Ferragut, M.I. Aseusio, Application of a nonlinear evolution model to fire propagation, Nonlinear Anal., Theory, Methods Appl. 30 (5) (1997) 2873–2882.