Tetrahedral Grid Refinement

J. Bey, Tübingen

Abstract — Zusammenfassung

Tetrahedral Grid Refinement. We present a refinement algorithm for unstructured tetrahedral grids which generates possibly highly non-uniform but nevertheless consistent (closed) and stable triangulations. Therefore we first define some *local* regular and irregular refinement rules that are applied to single elements. The *global* refinement algorithm then describes how these local rules can be combined and rearranged in order to ensure consistency as well as stability. It is given in a rather general form and includes also grid coarsening.

AMS Subject Classifications: 65N50, 65N55

Key words: Tetrahedral grid refinement, stable refinements, consistent triangulations, green closure, grid coarsening.

Verfeinerung von Tetraeder-Gittern. Es wird ein Verfeinerungsalgorithmus für unstrukturierte Tetraeder-Gitter vorgestellt, der möglicherweise stark nicht-uniforme aber dennoch konsistente (d.h. geschlossene) und stabile Triangulierungen liefert. Dazu definieren wir zunächst einige *lokale* reguläre bzw. irreguläre Verfeinerungsregeln für einzelne Elemente. Der *globale* Verfeinerungsalgorithmus beschreibt dann, wie diese lokalen Regeln kombiniert und umgeordnet werden können, so daß sowohl Konsistenz als auch Stabilität garantiert sind. Die Formulierung des globalen Algorithmus ist sehr allgemein gehalten und erlaubt auch Gitter-Vergröberungen.

1. Introduction

The numerical treatment of partial differential equations includes the solution of large systems of equations. For three-dimensional problems, several millions of unknowns are no rarity, and – although computational power has grown exponentially during the last decades – many interesting real life problems could not be solved today, had not the development of efficient algorithms been similarly successful.

Modern applications make use of adaptive techniques to optimize the number of unknowns by fitting the corresponding discretization to the present approximate solution. For this purpose, the underlying discretization mesh is refined locally in regions where improved accuracy is needed, for example, near singularities, internal or boundary layers, re-entrant corners, etc. Those regions where the solution is expected to be smooth are not refined or can even be coarsened.

Multigrid or multi-level methods have been proven to be of optimal or nearly optimal complexity for the solution of discrete systems arising from a wide range

of partial differential equations, in particular the elliptic ones. Because these methods are based on discretization hierarchies obtained from successively refined meshes, they can be embedded especially well in an adaptive framework. This fact is also represented by recent multigrid convergence proofs (see [9] or [18]), which no longer require the underlying meshes to be quasi-uniform, in contrast to the early theory, e.g. in [12].

In industrial practice, however, at least in the three-dimensional case, adaptive and/or multi-level methods are rarely applied, because possible users are often deterred from the necessity to generate and manage hierarchies of possible highly non-uniform meshes. In particular the refinement of tetrahedral grids requires special care to avoid degenerated elements which may lead to deteriorating convergence rates.

In the present paper we want to show that tetrahedral grid refinement can be realized in a very efficient way. Our algorithm includes adaptive refinement as well as partial coarsening, and the resulting triangulations are consistent and stable. Therefore the algorithm may be useful also in other applications, for example in the area of computer graphics for the approximation of smooth surfaces.

We start from the following assumptions. Let $\Omega \subset \mathbb{R}^3$ be a polyhedral domain, and assume that an initial triangulation $\mathcal{T}_0$ of Ω is given, that is, a set of non-degenerated tetrahedral elements which cover Ω and are of mutually disjoint interior. Usually such initial triangulations should be as coarse as possible, just fine enough to resolve the shape of Ω and the coefficient jumps of the problem under consideration. Note that in engineering practice the generation of useful initial triangulations is a complex and expensive task, but this is no subject of the present paper. Here we are concerned with successive refinements of $\mathcal{T}_0$, that is, we want to generate sequences $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k, \ldots$ of triangulations of Ω, which satisfy the following conditions:

(C1) **Nestedness**: *Each element $T \in \mathcal{T}_k$, $k > 0$ is covered by exactly one element $T' \in \mathcal{T}_{k-1}$, and any corner of T is either a corner or an edge midpoint of T'. T' is called the father element of T, and T is a son of T'.*

(C2) **Consistency**: *Each triangulation $\mathcal{T}_k$ is consistent, which means that the intersection of any two tetrahedra in $\mathcal{T}_k$ is either empty, a common face, a common edge or a common corner.*

(C3) **Stability**: *The sequence $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k, \ldots$ is stable in the sense that the interior angles of all elements are uniformly bounded away from zero.*

The consistency condition prevents so called *hanging nodes* which for different reasons are undesired in many applications. In numerical algorithms, for example, hanging nodes do not represent degrees of freedom and are somewhat difficult to handle, because the local correlation pattern of the stiffness matrix is disturbed. On the other hand, the asymptotic convergence properties of multigrid or other iterative methods are not essentially affected by the existence of

some isolated hanging nodes. For reasons that will become clear later, consistent triangulations are also called *closed*.

The stability condition, (C3), is equivalent to the requirement that some *measure of degeneracy*, $\delta(T)$, is uniformly bounded for all elements T. Such measures $\delta(T)$ can be defined in various ways. We prefer to set $\delta(T) = h(T)/\varrho(T)$, where $h(T)$ is the length of the longest edge and $\varrho(T)$ is the diameter of the biggest inscribed ball of T. This $\delta(T)$ enters directly into approximation estimates for finite element spaces (see [10]), and therefore also into the convergence theory of multi-level methods. Thus – in contrast to (C2) – stability is essential with respect to applicability of the triangulations in numerical algorithms.

Unfortunately, adaptivity, consistency, and stability are not compatible with each other, because locally refined grids require elements of higher degeneracy to be closed. Therefore our refinement algorithm is constructed in three steps. We first define a basic strategy for the subdivision of a single tetrahedron into eight subtetrahedra of equal volume in such a way that successive refinement of any initial element T produces stable and consistent triangulations of T. Refinement strategies of this type are called *regular*. Then we choose a set of *irregular* refinement rules for elements that are not refined regularly but share a refined edge or side of another element. These irregular refinements are only used for the *closure*, i.e. to satisfy the consistency condition.

Both regular and irregular refinement rules are *local*[1] in the sense that they are applied to single elements. In contrast, the third and final step consists in a superior *global* refinement algorithm that describes how the local rules can be combined and rearranged in order to ensure consistency and stability at the same time. For this purpose, we introduce some additional global conditions. Before studying the local and global strategies in detail, we briefly discuss some well known methods for the 2D case and the difficulties arising in three dimensions.

1.1 Local Refinements in 2D

For two-dimensional triangular grids, refinement methods satisfying (C1–3) are well known. Perhaps the most popular one is the combination of *red* (regular) and *green* (irregular) refinements which has been proposed by R. E. Bank et al. in [3] and then implemented into the well known multigrid code *PLTMG* [2]. A red refinement subdivides a given triangle into four congruent ones by connecting the midpoints of its edges. Green refinements are only used to close triangulations and consist of simple bisections connecting one edge midpoint and the opposite corner.

---

[1]These should not be confused with usual local refinements in the sense of adaptivity.

A second major class of 2D refinement methods is based on bisections only. These bisection methods can be distinguished by their way of preserving stability. M. C. Rivara, for example, uses the longest edge for bisection [17]. Although the number of generated congruence classes may be infinite, her method can be shown to be stable. In contrast, the algorithm of W. F. Mitchell produced triangles of at most four congruence classes by dividing the edge in opposition to the newest vertex ([16]).

Note that two successive *newest-vertex-bisections* – provided the second step is applied to both sons – divide a given element into four subtriangles of equal volume and thus can be interpreted as one step of a regular 2D refinement method, since (C1–3) are also satisfied.

The stable refinement of tetrahedral grids is more complex. In contrast to the 2D case, there is obviously no way to divide any given tetrahedron into eight congruent ones. Nevertheless, it is possible to extend both of the regular refinement strategies to three dimensions. For this purpose algorithms have been developed by several authors during the last years. Three-dimensional bisection methods have been presented, e.g. by E. Baensch [1], and J. M. L. Maubach [15], whereas the 3D analogon of Bank's red refinement strategy was introduced independently by S. Zhang, in [19] and the author in [5].

The second strategy forms the basis of this paper. It is marked by the fact that for any initial tetrahedron it produces elements of at most three congruence classes, no matter how many successive refinement steps are performed. Note that the N-dimensional generalization of this method was proposed already in 1942 by H. Freudenthal [11].

1.2 Global Refinement Algorithm

The global algorithm describes how the local rules can be combined in order to generate stable sequences of consistent triangulations. It is based on the following two restrictions:

(C4) Irregular elements are never refined.
(C5) $T \in \mathcal{T}_k \cup \mathcal{T}_{k+1}$ *implies* $T \in \mathcal{T}_l$ *for all* $l \geq k$, *that is, if an element is not refined passing from* $\mathcal{T}_k$ *to* $\mathcal{T}_{k+1}$, *then it remains unrefined in any* $\mathcal{T}_l$, $l > k$.

Here *irregular* elements are those resulting from irregular refinements. All other elements are called *regular*. Condition (C4) prevents the irregular refinements from destroying the stability of the regular ones and induces smooth transitions between regions of varying refinement depth. Condition (C5) does not really restrict the set of possible triangulations but allows the unique reconstruction of the complete sequence $\mathcal{T}_0, \ldots, \mathcal{T}_k$, if only $\mathcal{T}_0$ and $\mathcal{T}_k$ are known. Thus it makes sense to assign to each element T a level index l which is given by $l = \min\{k | T \in \mathcal{T}_k\}$.

There are mainly two reasons to require (C5). First, the size of any given element can be estimated by the size of an ancestor using the corresponding level distance, and thus (C5) is a usual requirement in the theory of multi-level methods (cf. [18]). On the other hand, the representation of the global algorithm is essentialy simplified, because it can be formulated in terms of operations that refer to all elements of a common level (see Section 3).

Conditions (C4,5) often lead to misunderstandings, which are caused by the wide-spread assumption that every refinement step is applied to the momentary finest triangulation $\mathscr{T}_k$ to produce the next finer triangulation $\mathscr{T}_{k+1}$ in such a way that $\mathscr{T}_0,\ldots,\mathscr{T}_k$ and $\mathscr{T}_{k+1}$ satisfy (C1-5). An algorithm of this type would be less useful because, first, it does not take into consideration partial coarsening, and second, (C4,5) prohibit supplementary refinements of temporarily unrefined regions. Both will be necessary at least in time-dependent numerical applications.

Our view of a refinement algorithm is more general and not restricted to the finest grid only. Given any input sequence $\mathscr{T}_0,\ldots,\mathscr{T}_k$, it produces another output sequence $\mathscr{T}_0',\ldots,\mathscr{T}_l'$ that satisfies the above conditions but does not have to contain any of the input triangulations with exception of $\mathscr{T}_0' = \mathscr{T}_0$. In addition, we do not require $l = k + 1$, but allow $l \in \{k, k \pm 1\}$ to catch also pure coarsening.

Under these circumstances, (C4) and (C5) lose their restrictive character. To maintain (C4), irregular refinements can be substituted by regular ones if an irregular element or one of its neighbors is assigned for further subdivision. Temporarily unrefined elements can be refined in agreement with (C5) by making sure that the sons are inserted at the correct level. These rearrangements, as well as possible coarsenings, are also tasks of the global algorithm.

From this point on, the paper is organized as follows. In Section 2 we present the basic regular refinement method for tetrahedral elements and add some irregular rules for the closure. Some advantages of our method in comparison with the bisection methods ([1],[15]) are discussed. In Section 3 we then describe the global refinement algorithm. It is formulated independently of any specific programming language and consequently does not use prescribed data structures as, for example, records or pointers. Instead, it operates on abstract objects like elements, edges, and nodes. Finally, in the Appendix, the stability proof for the regular refinement method from Section 2 is delivered subsequently.

2. Tetrahedron Refinement

In this section we present the local refinement strategies for single elements. The basic regular refinements, as proposed in [5], [19], produce consistent and

stable triangulations of any given initial tetrahedron. Irregular refinement rules are used for the closure only.

2.1 Regular Refinement

Let T be any given tetrahedron. We are looking for a subdivision of T into eight subtetrahedra $T_1,...,T_8$ of equal volume, in such a way that each corner of a son T_i coincides with either a corner or an edge midpoint of T.

Therefore we first connect the edges of each face triangle of T in the same way as in the two-dimensional regular refinement (Fig. 1). Then we cut off four subtetrahedra at the corners which are congruent with T. In the interior of the remaining octahedron there are three parallelograms, as shown in Fig. 2. Cutting the octahedron along two of these parallelograms, we obtain four more subtetrahedra. Each choice of two cut parallelograms corresponds to one of three possible diagonals, as shown in Fig. 3.
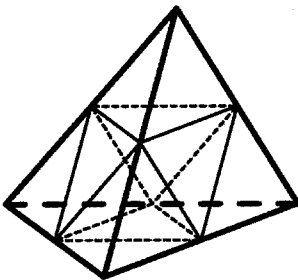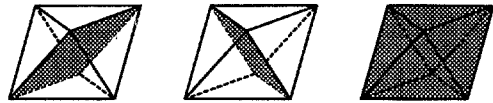

Figure 1. Regular refinement of the faces

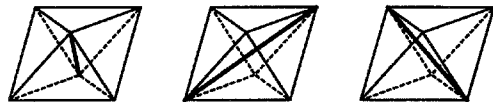**Figure 2.** Interior parallelograms

Figure 3. Interior diagonals

Note that the eight subtetrahedra are of equal volume, but the interior ones are not congruent with T in general. Therefore the question arises, which of the possible diagonals should be chosen in successive refinement steps. It has been shown in [19] that the wrong choice may lead to degenerated elements. In [5], however, we introduced a simple algorithm which for any initial element generates subtetrahedra of at most three congruence classes, no matter how many successive refinements are performed. Thus the stability of the generated triangulations is preserved.

To describe the algorithm we assume any tetrahedron T to be given by an ordered sequence of its vertices: $T = [x_0, x_1, x_2, x_3]$. For $0 \leq i, j \leq 3, i \neq j$, we

denote by $\mathbf{x}_{ij} := (\mathbf{x}_i + \mathbf{x}_j)/2$ the edge midpoint $\mathbf{x}_i$ and $\mathbf{x}_j$. Then the regular refinement algorithm can be formulated as follows:

Algorithm RegularRefinement (T)
{

> *divide* $T = [\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3]$ *into the subtetrahedra* T_i, $1 \le i \le 8$, *given by*
>
> $$T_1 := [\mathbf{x}_0, \mathbf{x}_{01}, \mathbf{x}_{02}, \mathbf{x}_{03}], \qquad T_5 := [\mathbf{x}_{01}, \mathbf{x}_{02}, \mathbf{x}_{03}, \mathbf{x}_{13}],$$
> $$T_2 := [\mathbf{x}_{01}, \mathbf{x}_1, \mathbf{x}_{12}, \mathbf{x}_{13}], \qquad T_6 := [\mathbf{x}_{01}, \mathbf{x}_{02}, \mathbf{x}_{12}, \mathbf{x}_{13}],$$
> $$T_3 := [\mathbf{x}_{02}, \mathbf{x}_{12}, \mathbf{x}_2, \mathbf{x}_{23}], \qquad T_7 := [\mathbf{x}_{02}, \mathbf{x}_{03}, \mathbf{x}_{13}, \mathbf{x}_{23}],$$
> $$T_4 := [\mathbf{x}_{03}, \mathbf{x}_{13}, \mathbf{x}_{23}, \mathbf{x}_3], \qquad T_8 := [\mathbf{x}_{02}, \mathbf{x}_{12}, \mathbf{x}_{13}, \mathbf{x}_{23}].$$

}

The subtetrahedra $T_1, ..., T_4$ are those from the corners which are congruent with T, whereas $T_5, ..., T_8$ originate from the remaining octahedron. The chosen diagonal is given by the vertices $\mathbf{x}_{02}$ and $\mathbf{x}_{13}$ that are common to all interior sons. Consequently the diagonal is implicitly characterized by the order of vertices, and thus the latter one is essential for the algorithm. The maximum number of generated congruence classes depends on the definition of *congruence* :

Definition. *Two tetrahedra T_1, T_2 are defined to be congruent, if they can be made to coincide by a rigid motion and a positive or negative scaling, i.e., if there exists a scaling factor $c \ne 0$, a translation vector $\mathbf{x}$, and an orthogonal matrix $\mathbf{Q}$ such that*

$$T_1 = \mathbf{x} + c\mathbf{Q}T_2 := \{\mathbf{x} + c\mathbf{Q}\mathbf{x}' \mid \mathbf{x}' \in T_2\}.$$

Of course congruence represents an equivalence relation and therefore all tetrahedral elements can be divided into congruence classes. In the following theorem we state that for any initial tetrahedron the above algorithm produces elements of at most three congruence classes, no matter how many successive refinement steps are performed. This fact immediately implies stability.

Theorem 1. *For any initial tetrahedron T, recursive application of algorithm Regular-Refinement produces consistent and stable triangulations of T. Moreover, all generated elements belong to one of at most three congruence classes.*

Proof: We just sketch the proof at this point and refer to the Appendix for details. It is based on the dissection of the unit cube into six tetrahedra passing into each other by permutation of their co-ordinates. It turns out that applying algorithm *RegularRefinement* to each of them yields the same triangulation as when we first divide the cube into eight subcubes which again are subdivided into six tetrahedra as indicated above. This process can be repeated, and the assertion follows by induction and the usual affine transformation argument. □

Remark 1. Although the vertex numbering is given for generated elements per definition, there is some freedom in choosing the vertex order of initial elements. In [19], S. Zhang has shown that the maximum measure of degeneracy of the sons is minimized if the vertices of any initial tetrahedron T are numbered in such a way that the diagonal between x_{02} and x_{13} is as short as possible. Extending this idea, he investigates a second strategy that chooses always the shortest diagonal for refinement. He could show that this *shortest-interior-edge* strategy is equivalent to the method presented above, as long as it is applied to initial elements with non-obtuse faces and suitable vertex order. Otherwise, for elements with at least one obtuse triangle, *shortest-interior-edge* subdivision may generate any number of congruence classes. In this case stability remains to be proven.

Remark 2. In 1942, H. Freudenthal proposed a method for the stable refinement of N-dimensional simplicial grids [11]. It turns out that the three-dimensional case of his method is equivalent to the algorithm presented above. Bank's red refinements correspond to the case $N = 2$. It can be shown that for general $N \geq 2$ the number of generated congruence classes is given by $N!/2$. In addition, one can prove that this number is optimal in the sense that any other regular refinement method based on (C1) produces at least $N!/2$ congruence classes for almost all initial N-simplices, provided that sufficiently many refinement steps are performed (see [6]). Furthermore, the proof of Theorem 1 in the Appendix – which also carries over to the N-dimensional case – indicates that all generated elements of a certain congruence class can be made to coincide just by translation and scaling. A rotation – as allowed by Definition 1 – is not necessary. These facts can be used to increase the efficiency of various algorithms based on such grids. In finite-element or finite-volume applications, for example, the assembling of local stiffness matrices represents one of the most time-consuming tasks that can be considerably improved by calculating and storing just once data that depend on the elements' level and congruence class only.

Remark 3. *The newest-vertex-bisection* of Mitchell, [16], has been generalized to $N \geq 2$ dimensions by J. M. L. Maubach [15]. It can be shown that this method produces $2^{N-2} \cdot N!$ congruence classes provided that N successive bisections (each applied to all possible sons) are considered as one (regular) refinement step. Moreover, these $2^{N-2} \cdot N!$ congruence classes contain those $N!/2$ classes generated by Freudenthal's algorithm. Therefore – in comparison with 3D *newest-vertex-bisection* – algorithm *RegularRefinement* produces only a quarter of the number of congruence classes but at most the same maximum measure of degeneracy.

A second advantage of our method is given by the symmetric subdivision of the triangular faces[2], which allows regular refinement of adjacent elements without

---

[2]This argument applies to the case $N = 3$ only.

consistency problems. In contrast, bisection methods may be also classified by their way of maintaining consistency even in the case of non-adaptive, global refinements (cf. [1, 15–17]). On the other hand, bisection methods are usually preferred in combinatorial algorithms (e.g. for fixpoint approximation), where N is very large and a refinement into 2^N sons per element makes no sense.

2.2 Irregular Refinements

As mentioned above, algorithm *RegularRefinement* can be applied to adjacent elements without consistency problems. In the case of adaptive refinements, however, only a subset of the given tetrahedra will be assigned for regular subdivision, and thus triangulations have to be closed by further irregular refinements in order to satisfy the consistency condition. We call this procedure the *green closure* in analogy to Bank's algorithm for the two-dimensional case, [3].

The 2D green closure can be realized by simple bisections if elements with two or more subdivided edges are refined regularly. This strategy is also used in *PLTMG* [2], but has the disadvantage that under certain circumstances regular refinements can propagate as in a game of dominoes. This expansion of the refined area can be prevented by providing a complete set of refinement rules, that is, one for every possible edge refinement pattern. In 2D there are $2^3 = 8$ different patterns for the three edges of a triangle. The full pattern corresponds to a regular refinement and the empty pattern indicates no refinement at all. The remaining six patterns can be divided into two types with one and two refined edges, respectively. The latter case can be closed by connecting the midpoints of these edges and then one of them to the opposite corner.

In three dimensions there are $2^6 = 64$ possible edge refinement patterns. Using symmetry arguments, the 62 irregular cases can be divided into 9 different types. For practical reasons we do not specify a complete set of irregular rules for these 9 types – which indeed is possible – but restrict ourselves to the four types shown in Fig. 4.
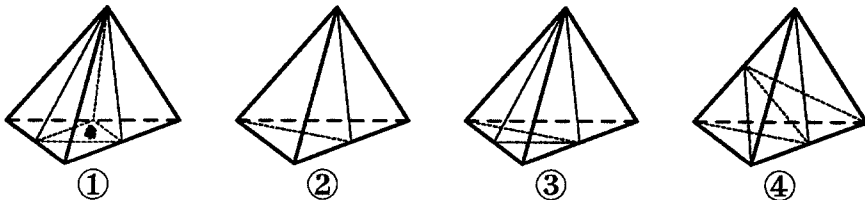

Figure 4. Irregular refinements for the green closure in 3D

These four types are considered also in [8]. Type (1) is applied if one neighboring element is refined regularly but no other edge is subdivided. Type (2)

corresponds to elements with exactly one refined edge, and types (3) and (4) apply to two refined edges on a common face and in opposition, respectively. The remaining cases are handled according to the following instruction:

(C6) If three or more edges are refined and do not belong to a common face, then the element is refined regularly.

Remark 4. Note that type (3) refinements require special care in order to preserve consistency even over the critical side with two refined edges. Therefore it is necessary to coordinate the refinements of both elements sharing this side.

Remark 5. In consequence of (C6), a domino effect may also occur in certain situations. In practice, however, we observed that the expansion of regularly refined regions caused by (C4) is rather dominating.

3. Global Refinement Algorithm

The global refinement algorithm describes how the local rules from Section 2 can be combined and rearranged in order to generate stable sequences of consistent triangulations. It is formulated independently of any specific programming language and consequently does not use prescribed data structures as, for example, records or pointers. For these, we refer to [7], [14], and the references therein. Instead, our algorithm operates on some basic abstract objects.

3.1 Abstract Data Structures

The basic objects we are dealing with are *elements, nodes*, and *edges*. These are distributed over the *grids* G_k of level $k = 0,..., k_{max}$ in the following way: G_0 contains the elements, nodes, and edges of triangulation $\mathcal{T}_0$, whereas G_k, $k > 0$ consists of those elements in $\mathcal{T}_k$ which do not already belong to $\mathcal{T}_{k-1}$, as well as their edges and nodes. The objects of grid G_k are said to be of level k. Condition (C5) implies that the elements of level k contain those of level $k + 1$.

For any level k node N belonging to at least one refined level k element, there is a succeeding node N' with equal co-ordinates on level $k + 1$, which in this case is called the *son* of N. Two nodes of the same level k are defined to be *neighbors*, if they are endnodes of a common level k edge. Two elements of the same level are *neighbors*, if they share a common triangular face.

The basic objects are now connected by specifying some required references between them. These are summarized in the following list. Given any element,

for example, we assume to have access to its nodes, edges, neighbors, and sons. The (maximum) number of references to objects of a special type is given in parentheses. ($\leq n$) items indicate that some or all of these references may be missing. Unrefined elements, for example, have no sons. Note that the number of edges starting from a given node is not restricted a priori but of course is bounded due to stability.

Reference list:

- **Element** $\rightarrow$ Nodes (4), Edges (6), Sons ($\leq$ 8), Neighbors ($\leq$ 4).
- **Edge** $\rightarrow$ Endnodes (2), Midnode ($\leq$ 1).
- **Node** $\rightarrow$ Edges (∞), Son ($\leq$ 1).

For a simple representation of the refinement algorithm, we describe the current state of an element T by some self-explaining, symbolic constants. For example, "T is *refined-regularly*" means that T has been refined by algorithm *RegularRefinement*. Otherwise T is either *refined-irregularly* or *unrefined*. Unrefined elements are called *leaf elements*.

On the other hand, each element is marked for a certain refinement rule that may or may not coincide with its actual refinement. Let R be one of the local refinement rules of Section 2, then "T is *marked-for-refinement-by-R*" indicates that T shall be refined according to R. Mostly, it will be sufficient to know whether R is regular or irregular. In this case, we say that T is *marked-for-regular-refinement* and *marked-for-irregular-refinement*, respectively. In addition, T may be *marked-for-no-refinement* or even *marked-for-coarsening*. In the latter case, T is allowed to be removed. Finally, T is *marked-according-to-refinement*, if mark and actual refinement rule of T coincide. This is just the state that all elements are expected to have after termination of the global algorithm.

In a similar way, we describe the state of an edge E of T. If T is *marked-for-refinement-by-R* where R is a rule that is going to refine E, then we say that E is *marked-for-refinement-by-T*. E is just *marked-for-refinement*, if there is at least one element T with edge E such that E is *marked-for-refinement-by-T*.

Remark 6. Given any edge E, we assume to be able to decide immediately – without looking at the elements around it – whether E is *marked-for-refinement* or not. In practice, this requirement can be efficiently satisfied by associating to E a counter $N(E)$ which is updated each time the mark of an element T around E is changed.

3.2 The Algorithm

We start the presentation of the global refinement algorithm by specifying the expected input, action, and output.

Input: We assume that the global algorithm is applied to a sequence $G_0,...,G_{k_{max}}$ of grids, which – in the way indicated above – correspond to triangulations that satisfy conditions (C1–5). Each leaf element is either *marked-for-regular-refinement, marked-for-no-refinement,* or *marked-for-coarsening.* These leaf element marks are usually determined by application of an error estimator. All other elements are assumed to be *marked-according-to-refinement.*

Action: The action of the algorithm is determined by the leaf element marks. These are evaluated in the following way: Regular leaf elements that are *marked-for-regular-refinement* are refined regularly. Irregular elements *marked-for-regular-refinement* are substituted by regular elements. Other marks of irregular elements are ignored. Regularly refined elements are *marked-for-no-refinement* if all sons are *marked-for-coarsening.* Finally, the green closure is performed on conditions (C4–6).

Output: The output sequence $G'_0,..., G'_{k'_{max}}$, $k'_{max} \geq 0$ with $G'_0 = G_0$ and $k'_{max} \in \{k_{max}, k_{max} \pm 1\}$ is required to satisfy the input specification, but in addition we expect all leaf elements to be *marked-for-no-refinement.* Consequently, applying the global algorithm to the output sequence again will be of no effect.

The basic structure of the global refinement algorithm is shown in Fig. 5. It mainly consists of two phases: In Phase I, i.e. loop (1), the different grid levels are visited in a top-down manner. The leaf element marks are evaluated (2) and the closure of the next finer level is computed between the regular elements (3). In Phase II, (4), the grids are visited in reverse order (bottom-up). The closure is completed (5), unused objects are removed (6), and new objects are generated (7). Finally, the number of output levels is determined by updating the maximum level number k_{max} (8), (9).

Algorithm GlobalRefinement $(G_0, \ldots, G_{k_{max}})$
{
   for $k := k_{max}$ down to 0 do   (1)
   {
     EvaluateMarks (G_k);   (2)
     CloseGrid (G_k);   (3)
   }
   For $k := 0$ to k_{max} do if $(G_k \neq \emptyset)$ then   (4)
   {
     if $(k > 0)$ CloseGrid (G_k);   (5)
     UnrefineGrid (G_k);   (6)
     RefineGrid (G_k);   (7)
   }
   if $(G_{k_{max}} = \emptyset)$then $k_{max} := k_{max} - 1$;   (8)
   else if $(G_{k_{max}+1} \neq \emptyset)$ then $k_{max} := k_{max} + 1$ (9)
}

Figure 5. Global refinement algorithm

In Phase II, the execution of (5) for $k = 0$ is omitted because level 0 elements are regular per definition. Since only leaf elements can be removed, $G_k = \emptyset$ in line (4) is possible for $k = k_{max}$ only. In this case, loop (4) is prematurely terminated and k_{max} is decremented by one (8). Otherwise, k_{max} is incremented if there are new elements of level $k_{max} + 1$ (9).

The two phase top-down/bottom-up structure of algorithm *GlobalRefinement* goes back to P. Bastian, who implemented a two-dimensional version for his *UG* code [4]. The structure of the subroutines, however, is different. These are explained in the following subsection.

3.3 The Subroutines

Algorithm *GlobalRefinement* works element-based in the sense that all subroutines step over the elements of a given level. Function *EvaluateMarks* is responsible for evaluation and manipulation of the element marks according to the above specified action of the algorithm. To satisfy (C4), irregularly refined elements are *marked-for-regular-refinement* if at least one son has an edge that is *marked-for-refinement* (6). This includes the case that the son itself is *marked-for-regular-refinement*.

Function EvaluateMarks(G_k):
{
    for all elements $T \in G_k$:   (1)
    {
        if (T is refined-regularly and all sons of T are marked-for-coarsening)   (2)
            then T is marked-for-no-refinement;   (3)

        if (T is refined-irregularly) then   (4)
            if (at least one edge E of a son of T is marked-for-refinement)   (5)
                then T is marked-for-regular-refinement;   (6)
                else T is marked-for-no-refinement;   (7)

    }
}

After termination of *EvaluateMarks*, all marks for irregular refinements have been removed (in either (6) or (7)). If not substituted by a regular one, any irregular refinement must be confirmed by the following closure that is computed in function *CloseGrid*.

Function CloseGrid(G_k):
{
    let Q be the set of all regular elements $T \in G_k$ having at least one edge E   (1)
        that is marked-for-refinement but not marked-for-refinement-by-T;

while $(Q \neq \emptyset)$ (2)

   {

      choose an element $T \in Q$;   (3)

      $Q := Q \setminus \{T\}$;   (4)

      CloseElement (T);   (5)

   }

}

In function *CloseGrid*, the candidates for the closure are stored in a set Q (1). Candidates are all regular elements T with an edge E that is *marked-for-re-finement* by another element but not by T. Thus, elements *marked-for-regular-re-finement* cannot be candidates. As long as Q is non-empty, an arbitrary element is poped out of Q and passed to function *CloseElement* to determine a suitable refinement rule.

Function CloseElement (T):

{

   Search for a refinement rule R refining exactly those edges E of T   (1)

      that are marked-for-refinement;

   if (R is found) then   (2)

   {

      if (R is of type (3) and $T' \notin Q$ *for the critical neighbor* T' *of T)*   (3)

         then fit R to the mark of T';   (4)

      T is marked-for-refinement-by-R;   (5)

   }

   else

   {

      for all edges E of T that are not marked-for-refinement: (6)

         $Q := Q \cup \{T' \neq T | E$ *is an edge of* $T'\}$;   (7)

      T is marked-for-regular-refinement;   (8)

   }

}

Given any element T, function *CloseElement* at first looks for a (regular or irregular) refinement rule R with an edge refinement pattern matching exactly the edges of T that are *marked-for-refinement* (1). If such an R exists, T is *marked-for-refinement-by-R* (5). Special care is necessary if R is of type (3) (cf. Remark 4 in Section 2.2). In this case, R must be fitted to the mark of the critical neighbor sharing both of the edges that are *marked-for-refinement* (4), provided this neighbor does not belong to Q and thus is also marked for a type (3) refinement.

If a suitable R cannot be found, rule (C6) is applied and T is *marked-for-regular-refinement* (9). In this case, all elements around those edges of T that previously have not been *marked-for-refinement* are now new candidates for the closure and hence added to Q (7). Note that the definition of Q as a set implies that every element occurs in Q not more than once.

Remark 7. In statement (7) of *CloseElement*, we refer to all elements T' sharing a given edge E. Of course, it is not necessary to store with each edge a list of surrounding elements. Since one of them, T, is known, we can make use of the element neighborhood relations to determine the others, provided that edges at the boundary belong to exactly two triangular boundary faces[3].

CloseGrid is called twice for every grid level. In Phase I, the closure of level $k + 1$ is computed between the present regular elements of level k. Irregular level k elements that would be candidates for the closure are recognized during the following execution of function *EvaluateMarks* at level $k - 1$ (statement (5)). In this case, their fathers are *marked-for-regular-refinement* in *EvaluateMarks*, (6), and thus, these irregular elements are substituted by regular ones in Phase II, function *RefineGrid*, just before *CloseGrid* is called a second time at level k.

Here the computation of the level $k + 1$ closure is extended to the additional regular elements of level k, which are initially *marked-for-no-refinement* (see *RefineGrid*, (6)). To prove consistency (Theorem 3), we will show later on that in this second phase (C6) is never applied, which means that no further edges are *marked-for-refinement*. Therefore, consideration of these new elements for the closure is sufficient.

Remark 8. *EvaluateMarks* and *CloseGrid* only manipulate the marks of elements in G_k but do not change their actual refinements, i.e. do not create or remove any objects in G_{k+1}. This is done in *UnrefineGrid* and *RefineGrid*, respectively.

UnrefineGrid removes unused elements, nodes, and edges from the following level (7). For this purpose, the remaining objects are *marked-for-re-use* (5). Of course, we assume that these marks do not effect the refinement marks of elements or edges. Re-used objects are all sons of elements that are *marked-according-to-refinement*, as well as their edges and nodes. At the beginning of function *UnrefineGrid*, a possible new grid level $G_{k_{max}+1}$ is initialized (1) and the re-use marks of all level $k + 1$ objects are reset (2), (3).

Function UnrefineGrid (G_k):
{

    if ($k = k_{max}$) then $G_{k+1} := \emptyset$;   (1)

    for all objects $O \in G_{k+1}$:   (2)

        O is not marked-for-re-use;   (3)

    for all refined elements $T \in G_k$ that are marked-according-to-refinement: (4)

        all sons of T, their edges and nodes are marked-for-re-use;   (5)

    for all objects $O \in G_{k+1}$ that are not marked-for-re-use: (6)

        remove O from G_{k+1};   (7)

}

---

[3] In fact this is a condition on Ω.

In function *RefineGrid*, the computed refinements are actually realized. At first, in (2), any remaining marks for coarsening are switched into marks for no refinement in order to let the concerned (leaf) elements be *marked-according-to-refinement*.

The main loop (3) is now executed for each element $T \in G_k$ that is not *marked-according-to-refinement*. When entering the loop, the state of T is unrefined – because all sons have been removed in function *UnrefineGrid* – and therefore T must be marked for either regular or irregular refinement. Now T is refined according to this mark which means that the corresponding son elements are created (4). All sons are *marked-for-no-refinement* (6) and added to G_{k+1} (7). Missing nodes and edges of level $k + 1$ must be created (9) in addition to those which have been *marked-for-re-use* in function *UnrefineGrid*. Finally, in (10), the existing neighbors of all sons are determined to update the neighborhood graph of level $k + 1$.

Function RefineGrid (G_k):
{

    for all elements $T \in G_k$ that are marked-for-coarsening: (1)
        T is marked-for-no-refinement;   (2)
    for all elements $T \in G_k$ that are not marked-according-to-refinement: (3)
    {

        refine T according to the rule that T is marked for;   (4)
        for all sons T' of T: (5)
        {

            T' is marked-for-no-refinement;   (6)
            $G_{k+1} := G_{k+1} \cup \{T'\}$;   (7)
            find existing nodes and edges of T' in G_{k+1};   (8)
            create missing nodes and edges of T' and add them to G_{k+1};   (9)
            find existing neighbors of T' in G_{k+1};   (10)
        }

    }

}

When function *RefineGrid* is finished, all elements of level k are *marked-according-to-refinement* and thus one requirement of the output specification is satisfied. (C1–5) are established with Theorem 3 in the following subsection. To prove the optimal complexity of algorithm *GlobalRefinement*, we have to be aware that the amount of work for (8) and (10) is bounded by a constant (i.e. $O(1)$):

Remark 9. Since we have access to the existing son nodes and edge midnodes of T, we can find the nodes of T' in (8) with constant amount of work. The same is also true for the edges of T' because we know all edges starting at a given node. In (10), the neighbors of T' can be found in $O(1)$ operations between the sons of T and the sons of neighbors of T. Here it is sufficient to consider those neighbors of T which are already *marked-according-to-refinement*.

3.4 Complexity and Correctness

Theorem 2. *The amount of work of algorithm GlobalRefinement is proportional to the number n^*_{out} of leaf elements in the output sequence $G'_0,...,G'_{k'_{max}}$.*

Proof: Let n_k be the number of elements of level k and $n = n_0 + ... + n_{k_{max}}$ be the total number of elements of the input sequence. Taking into account Remarks 6 and 9, it is clear that each of the functions *EvaluateMarks*, *UnrefineGrid*, and *RefineGrid* performs a limited number of operations for every element and thus the amount of work for these functions on level k is of order $O(n_k)$. The same is also true for function *CloseGrid*, because every element can be added to Q no more than six times – once for every edge – in statement (7) of function *CloseElement*. Therefore the complexity of algorithm *GlobalRefinement* is of order $O(n)$.

Let now n^*_{in} be the number of leaf elements of the input sequence. Then the geometric growth of the number of leaf elements with increasing refinement depth implies $n \leq 2n^*_{\text{in}}$. The local refinement and coarsening rules imply $1/8n^*_{\text{in}} \leq n^*_{\text{out}} \leq 8n^*_{\text{in}}$. In combination with the $O(n)$ result, these inequalities complete the proof. $\square$

If we consider the corresponding triangulations $\mathscr{T}'_k$ instead of the grids G'_k, then the leaf elements of the output sequence are just the elements of the finest triangulation $\mathscr{T}'_{k'_{max}}$. It follows that the amount of work of algorithm *GlobalRefinement* is even proportional to the number of nodes of $\mathscr{T}'_{k'_{max}}$, provided the generated triangulations are stable. The following theorem ensures stability as well as consistency.

Theorem 3. *Let $\mathscr{T}_0$ be a consistent initial triangulation of the polyhedral domain Ω. Then recursive application of algorithm GlobalRefinement – in alternation with any given strategy for the generation of the leaf element marks – produces sequences of triangulations that satisfy conditions (C1–5).*

Proof: Of course, (C1) follows from the definition of the local refinement rules in Section 2. (C4,5) are satisfied by construction of the algorithm. Stability (C3) follows from the stability of the regular refinements (Theorem 1) in combination with (C4).

Still, consistency remains to be proven. This is done by induction over the grid levels k. The consistency of $\mathscr{T}'_0 = \mathscr{T}_0$ is given by assumption. Let now $\mathscr{T}'_k$ be consistent for a fixed $0 \leq k < k'_{max}$. After application of function *CloseGrid* in Phase I, the already existing regular elements of G_k are marked for consistent refinement. In Phase II, *CloseGrid* operates on those additional regular elements – generated by the preceding *RefineGrid* call – which have an edge that is *marked-for-refinement*.

We now have to show that it is sufficient to consider these new elements. Therefore let E be any edge of level k that is *marked-for-refinement*. The consistency of $\mathscr{T}_k'$ implies that – at least inside $\Omega - E$ is completely surrounded by level k elements. Moreover, these elements are regular due to statement (6) of function *EvaluateMarks*. Thus the second call to *CloseGrid* yields a consistent triangulation $\mathscr{T}_{k+1}'$ of Ω, provided that no additional edges are refined by applying (C6) in *CloseElement*, (8).

To see that this cannot be the case, assume that (C6) is applied to a just created regular element T of level k. Then there are at least three edges of T that are *marked-for-refinement* but do not belong to a common face. Without loss of generality we may assume that (C6) is applied the first time on this level. Therefore the three edges of T did exist and have been *marked-for-refinement* already in Phase I. It follows that at this time also the father element $\mathscr{T}'$ of T has had at least three refined edges on more than one face. Thus T' was a regularly refined element, in contradiction to the assumption that T is a new regular element. Consequently (C6) is never applied in Phase II, which proves consistency.   □

3.5 The Adaptive Grid Manager AGM^{3D}

The AGM^{3D} code contains our actual implementation of the algorithm presented above. This *Adaptive Grid Manager* provides a set of problem independent tools for the adaptive numerical solution of PDE's in three space dimensions. It originates from an early three-dimensional version of Bastian's 2D code UG (*Unstructured Grids,* cf. [4]). Many of the basic concepts of UG also entered into AGM^{3D}, in particular the two phase top-down/bottom-up structure of the global refinement algorithm.

The AGM^{3D} code is written in ANSI C, and the basic data structures are especially designed for the application of multigrid and multi-level algorithms. The refinement procedure of AGM^{3D} is somewhat more advanced than the algorithm described in this paper. In particular, it includes the approximation of curved boundaries by projection of boundary edge midnodes. Graphical representation of generated grids is also possible. For this purpose, the multi-level structure is used for an efficient solution of the hidden-elements problem. For a more detailed description of AGM^{3D} and in particular the refinement routines, we refer to the manual ([7]). Both code and manual can be obtained from the author.

To finish the representation of the global algorithm, we show two pictures of adaptively refined triangulations that have been generated by AGM^{3D}. A refinement of the unit cube arising during the solution of a convection-diffusion problem is shown in Fig. 6. For an improved three-dimensional impression, the

cube has been cut open along its main diagonal. Here the smooth transitions between regions of varying refinement depth caused by (C4) can be observed especially well. Figure 7 shows the lower half of a refined torus. To approximate its curved boundary, every generated midnode of a boundary edge has been projected to the boundary. Note that projection to a concave part of the boundary may destroy stability if the initial triangulation $\mathscr{T}_0$ is not suitably chosen.
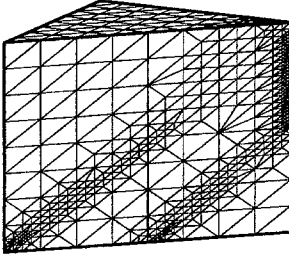

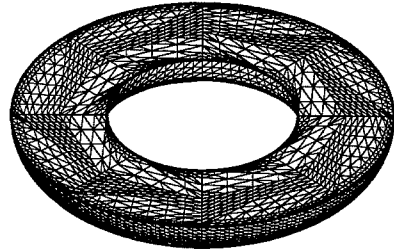**Figure 6.** Triangulation of the unit cube

Figure 7. Triangulation of a torus

Appendix

It remains to complete the proof of Theorem 1 in Section 2. Its leading part can be found implicitly in the early paper of H. Freudenthal, [11], who investigated the stable refinement of N-dimensional simplicial grids. At the beginning, we recall the assertion of Theorem 1:

Theorem 1. *For any initial tetrahedron T, recursive application of algorithm RegularRefinement produces consistent and stable triangulations of T. Moreover, all generated elements belong to at most three congruence classes.*

Proof: Of course, stability follows from the last statement. The proof of Theorem 1 is based on the dissection of the unit cube $C = [0,1]^3$ into six tetrahedra passing into each other by permutation of their co-ordinates. For any permutation $\pi \in S_3$, the tetrahedron $T_\pi = [\mathbf{x}_\pi^0, \mathbf{x}_\pi^1, \mathbf{x}_\pi^2, \mathbf{x}_\pi^3]$ is defined to be the closed convex hull of the corners

$$\mathbf{x}_\pi^0 = (0,0,0)^T, \quad \mathbf{x}_\pi^i = \mathbf{x}_\pi^{i-1} + e^{\pi(i)}, \quad i = 1,2,3, \tag{1}$$

where e^j denotes the jth standard unit vector in $\mathbb{R}^3$. The definition of the convex hull implies the representation

$$T_\pi = \left\{ \mathbf{x} \in C \mid 0 \le x_{\pi(3)} \le x_{\pi(2)} \le x_{\pi(1)} \le 1 \right\}, \quad \pi \in S_3. \tag{2}$$

Clearly $\mathcal{T}_0 = \mathcal{T}_0(C) := \{T_\pi \mid \pi \in S_3\}$ is a triangulation of C. Moreover, one may easily verify that for $\pi \neq \pi'$ the intersection of T_π and $T_{\pi'}$ is a common lower dimensional subsimplex, and thus $\mathcal{T}_0$ is consistent. According to [13], $\mathcal{T}_0$ is called the *Kuhn-triangulation* of C. It is shown in Fig. 8.
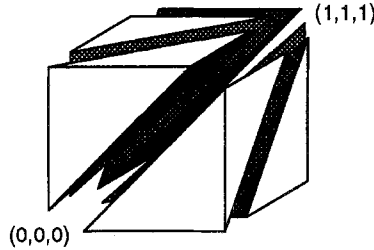

Figure 8. Kuhn-triangulation of the unit cube

Another triangulation $\mathcal{T}_1$ of C can be defined in the following way: Let $\mathcal{B}$ be the canonic subdivision of C into eight subcubes of edge length $1/2$, that is

$$\mathcal{B} := \left\{ C_{\mathbf{x}} \mid \mathbf{x} \in \{0,\tfrac{1}{2}\}^3 \right\}, \tag{3}$$

where $C_{\mathbf{x}}$, $\mathbf{x} \in \{0,1/2\}^3$ is given by

$$C_{\mathbf{x}} := \mathbf{x} + \tfrac{1}{2}C := \left\{ \mathbf{x} + \tfrac{1}{2}\mathbf{x}' \mid \mathbf{x}' \in C \right\}. \tag{4}$$

The Kuhn-triangulation of any subcube $C_{\mathbf{x}}$ is given by the tetrahedra $T_{\mathbf{x},\pi} := \mathbf{x} + 1/2\, T_\pi$, $\pi \in S_3$. Consequently,

$$\mathcal{T}_1 := \left\{ T_{\mathbf{x},\pi} \mid \mathbf{x} \in \{0,\tfrac{1}{2}\}^3, \pi \in S_3 \right\} \tag{5}$$

is a triangulation of C. The consistency of $\mathcal{T}_1$ follows from the consistency of $\mathcal{T}_0$ and the fact that Kuhn-triangulations of adjacent subcubes $C_{\mathbf{x}}$, $C_{\mathbf{x}'}$ induce a unique 2D-triangulation of the common face.

We now show that $\mathcal{T}_1$ is a refinement of $\mathcal{T}_0$ in the sense of condition (C1). Given any $\mathbf{x} \in \{0,1/2\}^3$ and $\pi \in S_3$, we are looking for a permutation $\pi^* = \pi^*(\mathbf{x},\pi)$ such that $T_{\mathbf{x},\pi} \subset T_{\pi^*}$. Therefore let $0 \leq k \leq 3$ be the number of entries x_i of $\mathbf{x}$ with $x_i = 1/2$. It follows that there are k unique indices $i_1,...,i_k \in \{1,2,3\}$ satisfying

$$1 \leq i_1 < \cdots < i_k \leq 3, \qquad x_{\pi(i_1)} = \cdots = x_{\pi(i_k)} = \tfrac{1}{2}, \tag{6}$$

whereas the remaining $3 - k$ indices $i_{k+1},...,i_3 \in \{1,2,3\}$ can be ordered such that

$$1 \leq i_{k+1} < \cdots < i_3 \leq 3, \qquad x_{\pi(i_{k+1})} = \cdots = x_{\pi(i_3)} = 0 \tag{7}$$

Here and in the following, for the case $k = 0$ and $k = 3$ we skip over those parts of the corresponding (in)equalities that make no sense. We now define π by $\pi^*(j) = \pi(i_j)$, $1 \le j \le 3$. From the right hand sides of (6) and (7), we conclude that

$$x_{\pi^*(1)} = \cdots = x_{\pi^*(k)} = \tfrac{1}{2}, \qquad x_{\pi^*(k+1)} = \cdots = x_{\pi^*(3)} = 0. \tag{8}$$

Further, for any $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3)^T \in 1/2\, T_\pi$ we have $0 \le \xi_{\pi(3)} \le \xi_{\pi(2)} \le \xi_{\pi(1)} \le 1/2$. Using the left hand sides of (6) and (7), we obtain

$$0 \le \xi_{\pi^*(k)} \le \cdots \le \xi_{\pi^*(1)} \le \tfrac{1}{2}, \qquad 0 \le \xi_{\pi^*(3)} \le \cdots \le \xi_{\pi^*(k+1)} \le \tfrac{1}{2}. \tag{9}$$

Combining (8), (9) with (2) proves $T_{\mathbf{x},\pi} \subset T_{\pi^*}$. Of course, by construction, any corner of $T_{\mathbf{x},\pi}$ corresponds to either a corner or an edge midpoint of T_{π^*} and thus $\mathcal{T}_1$ is in fact a refinement of $\mathcal{T}_0$ in the sense of (C1).

At this point, we have shown the existence of a refinement method for the elements of the Kuhn-triangulation $\mathcal{T}_0$ of C. This method yields the same triangulation $\mathcal{T}_1$ that is obtained if we first subdivide C into the eight subcubes $C_{\mathbf{x}} \in \mathcal{B}$ and these again by Kuhn-triangulation. Figure 9 illustrates these equivalent ways of generating $\mathcal{T}_1$.
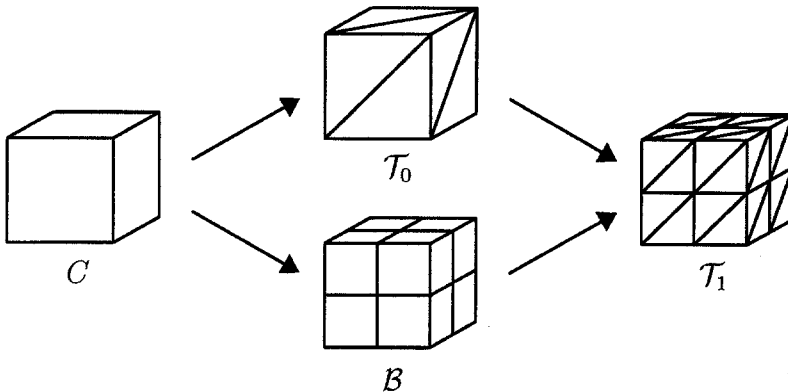

Figure 9. Two ways of generating $\mathcal{T}_1$

We now want to show that $\mathcal{T}_1$ is exactly the triangulation which is generated if algorithm *RegularRefinement* is applied to all elements $T_\pi \in \mathcal{T}_0$, provided their vertices are numbered according to (1). Therefore we first consider the *reference element* $T_0 := T_{\pi_{\text{id}}} = T_{(1,2,3)}$ with corners $(0,0,0)^T$, $(1,0,0)^T$, $(1,1,0)^T$ and $(1,1,1)^T$.

Using algorithm *RegularRefinement* to refine T_0, it is easily verified that the generated sons $T_{0,i}, 1 \leq i \leq 8$ can be represented by

$$T_{0,i} = \mathbf{x}_i + \tfrac{1}{2} T_{\pi_i}, \qquad \mathbf{x}_i \in \left\{0, \tfrac{1}{2}\right\}^3, \ \pi_i \in S_3, 1 \leq i \leq 8. \tag{10}$$

If the subsequent order of sons $T_{0,i}$ corresponds to the formulation of the refinement algorithm in Section 2, the start vertices $\mathbf{x}_i$ and permutations π_i, $1 \leq i \leq 8$ are given by

$$\mathbf{x}_1 = (0, 0, 0)^T, \qquad \mathbf{x}_2 = \mathbf{x}_5 = \mathbf{x}_6 = \left(\tfrac{1}{2}, 0, 0\right)^T,$$

$$\mathbf{x}_3 = \mathbf{x}_7 = \mathbf{x}_8 = \left(\tfrac{1}{2}, \tfrac{1}{2}, 0\right)^T, \qquad\qquad \mathbf{x}_4 = \left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right)^T, \tag{11}$$

and

$$\pi_1 = \pi_2 = \pi_3 = \pi_4 = \pi_{id},$$

$$\pi_5 = (2, 3, 1), \quad \pi_6 = (2, 1, 3), \quad \pi_7 = (3, 1, 2), \quad \pi_8 (1, 3, 2), \tag{12}$$

respectively. Representation (10) implies $T_{0,i} \in \mathscr{T}_1$ for $1 \leq i \leq 8$. For $i \neq j$, (11) resp. (12) show that either $\mathbf{x}_i \neq \mathbf{x}_j$ or $\pi_i \neq \pi_j$ is true. Therefore, $T_{0,i}, T_{0,j}$ correspond to different elements $T_{\mathbf{x}_i, \pi_i}, T_{\mathbf{x}_j, \pi_j} \in \mathscr{T}_1$, which are known to have mutually disjoint interior. Furthermore, we conclude from (10) that the volumes of all sons sum up to the volume of T_0, and thus the convexity of T_0 implies that the generated refinement of T_0 coincides with the one induced by $\mathscr{T}_1$.

To obtain the same result for the other elements in $\mathscr{T}_0$, we associate to each $\pi \in S_3$ the corresponding permutation matrix P_π which is given by $P_\pi = (\delta_{i, \pi(j)})^3_{i,j=1}$. We then have $T_\pi = P_\pi T_0$ and in particular for the corners $\mathbf{x}_\pi^j = P_\pi \mathbf{x}_{id}^j$, $0 \leq j \leq 3$. Applying algorithm *RegularRefinement* to T_π yields the sons $T_{\pi,i} = P_\pi T_{0,i}, 1 \leq i \leq 8$. Denoting by $\pi \circ \pi_i$ the composition of π, π_i within S_3, and using the fact that the associated permutation matrix is given by $P_{\pi \circ \pi_i} = P_\pi P_{\pi_i}$, the analogon to (10) is established by

$$T_{\pi,i} = P_\pi T_{0,i} = P_\pi \mathbf{x}_i + \tfrac{1}{2} T_{\pi \circ \pi_i}. \tag{13}$$

Now $P_\pi \mathbf{x}_i \in \{0, 1/2\}^3$ implies $T_{\pi,i} \in \mathscr{T}_1$ for each $1 \leq i \leq 8$, $\pi \in S_3$. Using the argumentation from above, it follows that the generated refinement of T_π coincides with the one induced by $\mathscr{T}_1$. Since this is true for any $\pi \in S_3$, $\mathscr{T}_1$ is in fact the triangulation generated from $\mathscr{T}_0$ by algorithm *RegularRefinement*.

In addition to (10), i.e. $T_{0,i} = \mathbf{x}_i + 1/2 \, T_{\pi_i}$, we observe that the vertex numbering assigned to $T_{0,i}$ by the refinement algorithm coincides with the one induced by T_{π_i}, i.e. the jth corner of $T_{0,i}$ is given by $\mathbf{x}_i + 1/2 \, \mathbf{x}_{\pi_i}^j$. This property is preserved under permutation and remains valid for any element of $\mathscr{T}_1$. If now algorithm *RegularRefinement* is recursively applied to the elements of $\mathscr{T}_1$, it follows by induction that the generated triangulations $\mathscr{T}_k, k \geq 0$ of C are given by

$$\mathscr{T}_k = \left\{ T_{\mathbf{x}, \pi} = \mathbf{x} + 2^{-k} T_\pi \mid \mathbf{x} \in \left\{0, 1 \cdot 2^{-k}, \ldots, (2^k - 1) \cdot 2^{-k}\right\}^3, \quad \pi \in S_3 \right\}, \tag{14}$$

and thus can also be obtained by first dividing C into 8^k subcubes of edge length 2^{-k}, which then are subdivided by Kuhn-triangulation.

To finish the proof, let $T = [\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3]$ be any non-degenerated tetrahedron, and let $\mathbf{F}: T_0 \to T$ be the affine transformation that maps T_0 one-to-one on T and in particular $\mathbf{x}^j_{\pi_{id}}$ to $\mathbf{x}^j$ for $0 \le j \le 3$. $\mathbf{F}$ maps edges and edge midpoints of T_0 to the corresponding edges and midpoints of T. Therefore recursive application of algorithm *RegularRefinement* to T yields triangulations

$$\mathscr{T}_k(T) = \left\{ \mathbf{F}(\hat{T}) | \hat{T} \in \mathscr{T}_k(C), \hat{T} \subset T_0 \right\}, \quad k = 0,1,2,\ldots \tag{15}$$

The consistency of $\mathscr{T}_k(C)$ implies the consistency of $\mathscr{T}_k(T)$ for all $k \ge 0$. Moreover, any $\hat{T} \in \mathscr{T}_k(C)$ of any level of $k \ge 0$ can be represented by $\hat{T} = \hat{\mathbf{x}} + 2^{-k} T_{\hat{\pi}}$ with suitable $\hat{\mathbf{x}} \in \mathbb{R}^3$, $\hat{\pi} \in S_3$. It follows that

$$\mathbf{F}(\hat{T}) = \mathbf{F}\hat{\mathbf{x}} - 2^{-k} x^0 + 2^{-k} \mathbf{F}(T_{\hat{\pi}}) \tag{16}$$

is congruent with $\mathbf{F}(T_{\hat{\pi}})$. We further observe that for any pair $\pi, \pi' \in S_3$ satisfying $\pi'(j) = \pi(4-j)$, $j = 1,2,3$ we have

$$-T_{\pi'} = -(1,1,1)^T + T_\pi, \tag{17}$$

implying that $\mathbf{F}(T_{\pi'})$ is congruent with $\mathbf{F}(T_\pi)$. Thus the elements of all triangulations $\mathscr{T}_k(T)$, $k \ge 0$ belong to at most three congruence classes. These arguments complete the proof. $\square$

References

[1] Baensch, E.: Local mesh refinement in 2 and 3, dimensions. Impact Comput. Sci. Eng. *3*, 181–191 (1991).

[2] Bank, R.E.: PLTMG: A software package for solving elliptic partial differential equations, Users' Guide 6.0. Philadelphia: SIAM, 1990.

[3] Bank, R.E., Sherman, A.H., Weiser, A.: Refinement algorithms and data structures for regular local mesh refinement, In: Scientific computing (Stepleman, R., ed.) pp. 3–17 Amsterdam: IMACS/North-Holland, 1983.

[4] Bastian, P.: Parallele adaptive Mehrgitterverfahren. PhD thesis, Univ. Heidelberg, 1994.

[5] Bey, J.: Analyse und Simulation eines Konjugierte-Gradienten-Verfahrens mit einem Multilevel Präkonditionierer zur Lösung dreidimensionaler elliptischer Randwertprobleme für massiv parallele Rechner. Master's thesis, Institut für Geometrie und Praktische Mathematik, RWTH Aachen, 1991.

[6] Bey, J.: A robust multigrid method for 3d convection-diffusion equations. PhD thesis, Univ. Tübingen (in preparation).

[7] Bey, J.: AGM^{3D} Manual. Tech. Rep., Univ. Tübingen, 1994.

[8] Bornemann, F.A., Erdmann, B., Kornhuber, R.: Adaptive multilevel methods in three space dimensions. Int. J. Numer. Meth. Eng. *36*, 3187–3203 (1993).

[9] Bramble, J.H., Pasciak, J.E., Wang, J., Xu, J.: Convergence estimates for multigrid algorithms without regularity assumptions. Math. Comp. *57*, 23–45 (1991).

[10] Ciarlet, P.G.: The finite element method for elliptic problems. Amsterdam: North-Holland, (1978).

[11] Freudenthal, H.: Simplizialzerlegungen von beschränkter Flachheit. Ann. Math. *43*, 580–582 (1992).

[12] Hackbusch, W.: Multigrid methods and applications. Berlin Heidelberg New York Tokyo: Springer, 1985.

[13] Kuhn, H.W.: Some combinatorial lemmas in topology. IBM J. Res. Dev. *45*, 518–524 (1960).

[14] Leinen, P.: Data structures and concepts for adaptive finite element methods. Computing *55*, 325–354 (1995).
[15] Maubach, J.M.L.: Local bisection refinement for N-simplicial grids generated by reflection. SIAM J. Sci. Comput. *16*, 210–227 (1995).
[16] Mitchell, W.F.: Adaptive refinement for arbitrary finite-element spaces with hierarchical basis. J. Comput. Appl. Math. *36*, 65–78 (1991).
[17] Rivara, M.C.: Design and data structure of a fully adaptive multigrid finite element software. ACM Trans. Math. Software *10*, 242–264 (1989).
[18] Yserentant, H.: Old and new convergence proofs of multigrid methods. Acta Numer. 285–326 (1993).
[19] Zhang, S.: Multi-level iterative techniques. PhD thesis, Research Report no. 88020, Dept. of Math., Pennstate Univ., 1988.

J. Bey
Mathematisches Institut
Universtität Tübingen
Auf der Morgenstelle 10
D-72076 Tübingen
Federal Republic of Germany
e-mail: bey@na.uni-tuebingen.de