SANDIA REPORT

SAND2004-3858
Unlimited Release
Printed August 2004

Performance of a Streaming Mesh Refinement Algorithm

David C. Thompson
Philippe P. Pébay

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/ordering.htm

SAND2004-3858
Unlimited Release
Printed August 2004

Performance of a Streaming Mesh Refinement Algorithm

David C. Thompson
Sandia National Laboratories
M.S. 9012, P.O. Box 969
Livermore, CA 94550, U.S.A.
dcthomp@sandia.gov

Philippe P. Pébay
Sandia National Laboratories
M.S. 9051, P.O. Box 969
Livermore, CA 94550, U.S.A.
pppebay@ca.sandia.gov

Abstract

In SAND report 2004-1617 [8], we outline a method for edge-based tetrahedral subdivision that does not rely on saving state or communication to produce compatible tetrahedralizations. This report analyzes the performance of the technique by characterizing (a) mesh quality, (b) execution time, and (c) traits of the algorithm that could affect quality or execution time differently for different meshes. It also details the method used to debug the several hundred subdivision templates that the algorithm relies upon. Mesh quality is on par with other similar refinement schemes and throughput on modern hardware can exceed 600,000 output tetrahedra per second. But if you want to understand the traits of the algorithm, you have to read the report!

Contents

Figures

Tables

Performance of a Streaming Mesh Refinement Algorithm

1 Introduction

Given an initial tessellation of an element's parametric domain, we apply an adaptive triangulation technique similar to [1], [9], and [7]. The main difference between our technique and the first two is that we handle tetrahedra as well as triangles and currently use chord error at the parametric midpoint of each edge rather than the angle between normal vectors at each endpoint. Our approach is a direct extension of [7], but ensures compatibility in all conditions without neighborhood information. As with the previous work, we assume that the initial tessellation is fine enough that no large changes in the error metric occur interior to the simplices; the adaptive tessellation is intended to improve detail, not to handle understanding.

The key design point of our implementation is that there are two tasks performed by an edge-subdivision based tessellation algorithm:

1. making a decision about whether an edge should be subdivided, and
2. applying a template to produce new elements based on which edges of the initial element require subdivision.

We split these two tasks into separate C++ classes so that the same templates for subdivision could be applied to many different subdivision decision algorithms. The algorithms that decide whether edges require subdivision vary depending on

1. the interpolation algorithm used for the geometric map,
2. the criteria used in the decision (geometric distance, scalar field nonlinearity),
3. the purpose of the overall task requiring a tessellation.

Item 3 requires some further explanation; if the tessellation is being produced simply for display purposes, then a view-dependent subdivision may be performed. This greatly reduces the amount of work required, since no function evaluation need take place if both edge endpoints are safely outside the viewing frustum[1]. On the other hand, if the tessellation is produced as input to some further post-processing step, then no regions may be excluded from the calculation.

[1] We assume that some safety margin is included so that edges which curve into the frustum are not excluded.

The templates that produce new simplices given a starting simplex σ and the edges of σ requiring subdivisions deserve some discussion. The problem of triangulating the new set of points (*i.e.*, the original vertices of σ and the mid-edge nodes being introduced by the subdivision) is not unique – there may be one or many possible triangulations of the point set. With a streaming algorithm, we must guarantee that each simplex may be processed without any information about its neighbors. Since we want to maintain a compatible tessellation, this means that any simplices that are shared as a boundary between two higher-dimensional simplices must be tessellated identically when all the higher-dimensional simplices are divided, even where there are several distinct possibilities. For example, any triangle, τ, shared by two tetrahedra, σ_0 and σ_1, must be tessellated the same way when σ_0 and σ_1 are subdivided.

We have presented in [8] a new scheme for refining tetrahedral meshes that does not require neighborhood information. This makes it viable for streaming large datasets and for parallel processing, where communication would be required to process elements on boundaries between processes. We explain here how to practically implement the method described in [8], since there is a long way to go from describing the canonical configurations to implementing all the situations each of them represents.

2 Parallel streaming mesh refinement

2.1 Unambiguous Cases

We use the same nomenclature as [7], so this section is just a brief review of their results. When a tetrahedron, σ, is to be subdivided, we are given a list of edges of σ that will be divided. First, the vertices of σ are permuted into σ', a positive arrangement of σ that matches one of 12 cases ([7] present 11 cases but we divide their case 3c into 3c and 3d so that σ' will always be a *positive* arrangement of σ). Cases are called out with

- the number of edges of a tetrahedron, σ, that should be subdivided, and
- a letter representing a unique configuration of those edges relative to each other.

Then, a collection of points, P, is created that includes σ' and the midpoints of edges in σ' that must be subdivided. This set of points, P, must be tessellated in a consistent manner so that simplices adjacent to σ will be compatible at the boundary they share with σ. Let's say we can produce such a tessellation. Call it σ''. For each of the 12 cases, there will be edges in σ'' that are constrained to be present and possibly some edges of σ'' that are not constrained. [7] use geometry – the length of the edges of σ' – to decide how to connect points in P to form σ''. They choose edges for σ'' that produce tetrahedra with the best possible aspect ratio given P. Each case with unconstrained edges in σ'' will have several

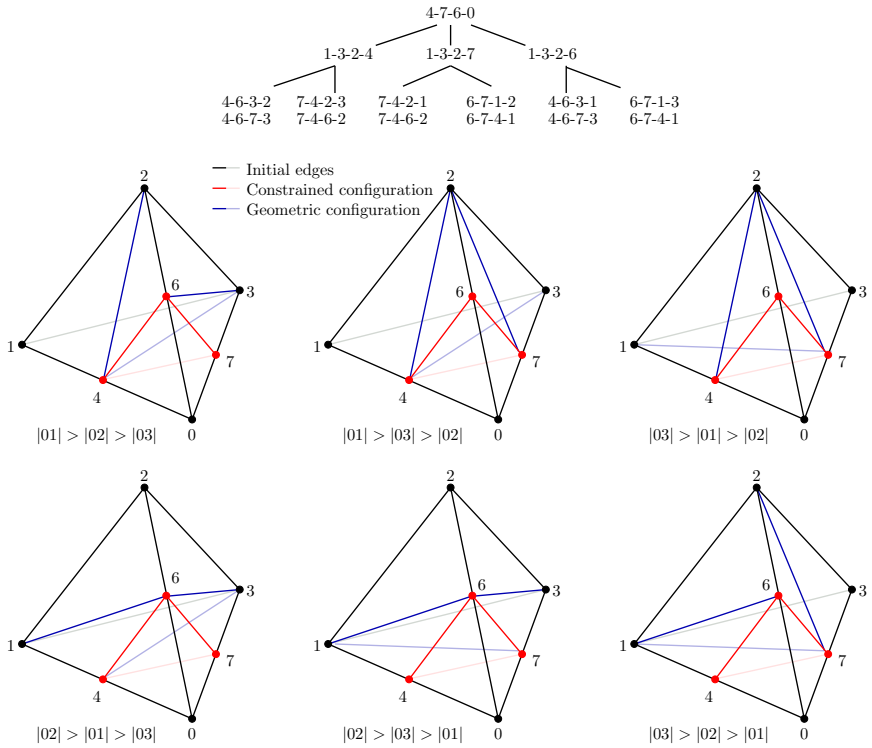

Figure 1. Subdivision of unambiguous case 3a.

variants based on which edges of σ' are longer than the others (relative to which edges must be subdivided). Figure 1 shows all 6 variants for Case 3a.

Unfortunately, when edges of σ' are of equal length, the edge length criterion gives no answer for how σ'' should be obtained. This leaves several possibilities for σ'' and it is ambiguous which one we should use so that σ'' remains compatible with its neighbors. There are two ways to resolve these ambiguities:

1. the algorithm stores the state of each element as it is refined, and subsequent communication will insured compatibility; or,

2. the algorithm chooses σ'' using a criterion that will the same for all simplices sharing that face.

We will take the second approach, because it requires less states for the algorithm, and eliminates communication. In the next section, we devise such an algorithm.

2.2 Ambiguous Cases

Ambiguous cases occur when a face can be split in two different ways, *i.e.*, whenever at least one face has exactly two edges of equal length that must be split. A simple enumeration shows that all such situations can be summarized by the means of reference configurations 2a, 3a, 3c, 4a, 4b or 5 (see Figure 2). To resolve the ambiguity when edges of

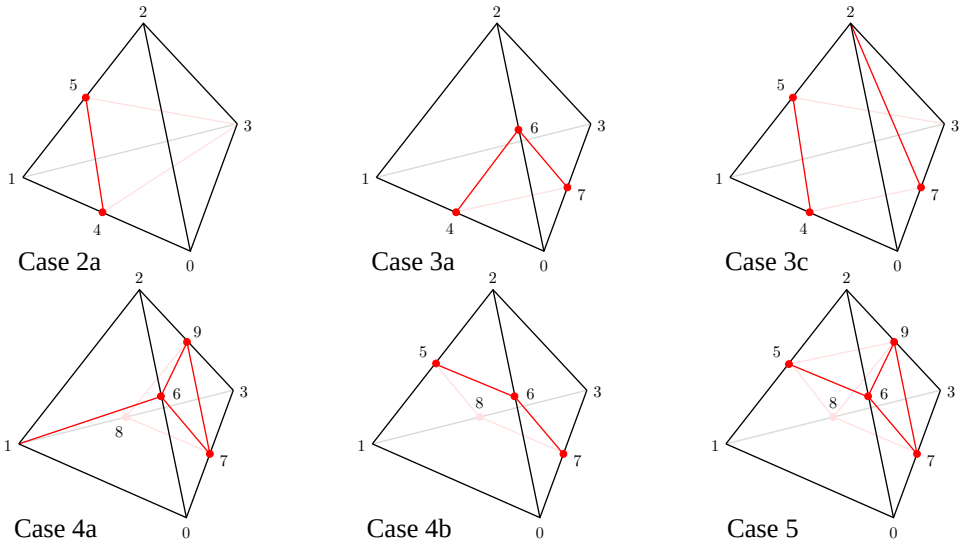

Figure 2. Potentially ambiguous configurations.

σ' are of equal length, we propose adding a new point, a, to each face with an ambiguous triangulation. More precisely, the face must be split unequivocally into a triangle and an isosceles trapezoid, the latter having two possible triangular subdivisions, both of them being acceptable according to the subdivision algorithm (*cf.* Figure 3(a)). By placing a on the angle bisector of the vertex opposite the base of the trapezoid, as shown in Figure 3(b), there exists a triangulation that is symmetric about the angle bisector and will be identical for σ and any tetrahedron that shares the face with σ. We discuss the placement of a along the angle bisector for the best resulting tetrahedra in [6]. The end result is that we place a at $\frac{1}{4}$ of the triangle's altitude. In [8], we proposed lifting the face ambiguities by the means of point(s) insertion(s) in the interior of the ambiguous faces. A complete set of compatible tetrahedral subdivisions for each canonical case has been proposed, and it is recalled here in 2.2.

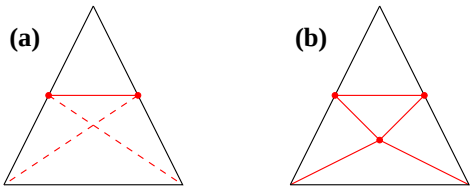

Figure 3. Ambiguous isosceles face refinement: (a) two different subdivisions are possible, and (b) unambiguous subdivision thanks to a point insertion.

Table 1. Subdivisions of canonical ambiguous cases

Case	Ambiguity	Tetrahedra
2a	$\|01\| = \|12\|$	04*a*3 0*a*23 4153 45*a*3 *a*523
3aα	$\|01\| = \|02\| > \|03\|$	0467 4367 *a*123 *a*263 *a*643 *a*413
3aβ	$\|01\| = \|02\| < \|03\|$	0467 1327 *a*127 *a*267 *a*647 *a*417
3aγ	$\|01\| = \|02\| = \|03\|$	0467 26*ad* 37*db* 41*ab* *b*6*a*4 *b*6*da* *b*67*d* *b*647 2*abd* 1*ab*2 2*b*3*d* 321*b*
3cα	$\|01\| = \|12\| > \|03\|$	4153 *a*047 *a*207 *a*743 *a*273 *a*523 *a*453
3cβ	$\|01\| = \|12\| < \|03\|$	7153 7523 *a*047 *a*207 *a*527 *a*457 1547
3cγ	$\|01\| = \|12\| = \|03\|$	415*b* *b*153 *a*047 *a*207 *a*523 *a*273 *a*74*b* *a*7*b*3 *a*45*b* *ab*53
4aα	$\|03\| = \|13\| > \|23\|$	7893 670*b* 601*b* 6978 67*b*8 6*b*18 1268 2689
4aβ	$\|03\| = \|13\| < \|23\|$	7893 670*b* 601*b* 6978 67*b*8 6*b*18 1269 1689
4aγ	$\|03\| = \|13\| = \|23\|$	7893 670*b* 601*b* 6978 67*b*8 6*b*18 612*c* 629*c* 698*c* 681*c*
4bα	$\|02\| = \|12\| < \|13\| < \|03\|$	7823 *a*607 *a*158 *a*017 *a*718 67*a*8 6*a*58 6278 6528
4bβ	$\|02\| = \|12\| > \|13\| > \|03\|$	6523 *a*607 *a*158 *a*018 *a*708 67*a*8 6*a*58 6378 6538
4bγ	$\|03\| < \|02\| = \|12\| < \|13\|$	6238 *a*607 *a*158 *a*018 *a*708 67*a*8 6*a*58 6378 6528
4bδ	$\|02\| = \|12\| < \|03\| = \|13\|$	7823 *a*607 *a*158 *a*01*b* *ab*18 *a*0*b*7 *a*7*b*8 67*a*8 6*a*58 6278 6528
4bε	$\|02\| = \|12\| = \|03\| < \|13\|$	*a*607 *a*158 *a*018 *a*708 *d*625 *d*378 *d*238 *d*285 67*a*8 6*a*58 6*d*78 65*d*8
4bζ	$\|02\| = \|12\| = \|03\| > \|13\|$	*a*607 *a*158 *a*018 *a*708 *d*625 *d*378 *d*235 *d*385 67*a*8 6*a*58 6*d*78 65*d*8
4bη	$\|02\| = \|12\| = \|03\| = \|13\|$	*a*607 *a*158 *a*01*b* *ab*18 *a*0*b*7 *a*7*b*8 *d*625 *d*378 *d*23*c* *d*2*c*5 *dc*38 *d*5*c*8 67*a*8 6*a*58 65*d*8 6*d*78
5α	$\|02\| = \|12\|, \|03\| > \|13\|$	6529 7893 *a*607 *a*158 *a*017 *a*718 *a*859 *a*679
5β	$\|02\| = \|12\|, \|03\| = \|13\|$	6529 7893 *a*607 *a*158 *a*01*b* *ab*18 *a*0*b*7 *a*7*b*8 *a*859 *a*679

3 Practical Implementation Thanks To $\mathfrak{S}_4$

The algorithm outlined in the previous section implies that the tetrahedron σ' must be permuted into one of the canonical configurations of Table 2.2, prior to deciding σ''. In other words, the algorithm requires the composition of two permutations; this results in a great deal of complexity.

3.1 `Python` Implementation

We use `Python` to generate C++ code for two reasons: it helps manage the complexity of the tessellation process and it allows a compact array of output tetrahedra to be generated for fast execution.

To illustrate the complexity of the process, the basic flow of the adaptive tessellation code is:

1. An input tetrahedron, $\sigma_i = \{v_0, v_1, v_2, v_3\}$, is supplied by the user,
2. A subdivision algorithm is applied to each edge of σ_i, possibly resulting in additional vertices (some subset, $E \subseteq \{v_4, v_5, \ldots, v_9\}$).
3. A map $\Lambda : (\sigma_i, E) \rightarrow (\sigma_i, E)$ is applied to permute the vertices of σ_i and E into one of 12 reference configurations.
4. All faces of σ_i that contain exactly 2 subdivided edges of exactly the same length have a vertex added to the face, resulting in another set of additional vertices, $F \subseteq \{v_{10}, v_{11}, v_{12}, v_{13}\}$.
5. The lengths of edges of σ_i are used to determine a set of output tetrahedra, T consisting of vertices from σ_i, E, and F. This is a second map whose domain is the vertices in σ_i, E, and F; and whose range is a set of simplices composed of vertices in σ_i, E, and F, i.e., $\Upsilon : (\sigma_i, E, F) \rightarrow \{[v_i, v_j, v_k, v_\ell], \ldots\}$.
6. Each $\sigma_o \in T$ is used as an input tetrahedron for this algorithm until no further subdivision is required or the maximum number of recursions has been made.

The map Λ is implemented as a simple lookup table with 64 entries (the number of possible unique subsets E). Υ is more complex because the map relies on geometric relations between vertices of E and because the range is sets of simplices rather than a permutation of the input vertices. We could create a list of sets of all the simplices in the range of Υ, but this list would be extremely long. Also, it would contain many tetrahedralizations of σ_i that were simply rotated, mirrored, or otherwise transformed versions of some other set in the list. So, we decompose Υ into a pair of maps ($\Upsilon = \Delta \circ \Omega$):

1. A permutation of the input vertices (i.e., a second map, Δ, of the same form as Λ), and

2. A map from the input vertices to a set of tetrahedra (i.e., a map, Ω, of the same form as Υ but with a smaller domain).

Even after the decomposition, there are still approximately 280 tetrahedra grouped into 50 sets. Rather than hand-write the C++ code that selects the proper set of tetrahedra and permutations to apply, we have created a Python script that assembles an array of tetrahedra and permutations and then writes C++ code to apply the proper maps as table lookups.

3.2 Brief Reminder About Symmetric Groups

Definition 3.1. Let E be a set. A bijection from E onto itself is called a *permutation of* E. Equipped with the composition operation (also called *product* in this context), the set of all permutations of E forms a group, called the *symmetric group of* E, and denoted as $\mathfrak{S}(E)$. For all $n \in \mathbb{N}^*$, the symmetric group of $\{1,\dots,n\}$ is denoted $\mathfrak{S}_n$.

Remark 3.1. It is clear that the symmetric group of any ordered finite set E with $n \in \mathbb{N}^*$ elements is isomorphic to $\mathfrak{S}_n$, thanks to the canonical increasing bijection between E and $\{1,\dots,n\}$. Therefore, we will make use of $\mathfrak{S}_4$ to denote the symmetric group of the four vertices of a tetrahedron, indexed from 0 to 3.

In all that follows, E and will denote a finite set with cardinality $n \in \mathbb{N}^*$.

Definition 3.2. Let $s \in \mathfrak{S}(E)$. For any given $x \in E$, the *s-orbit of* x is defined as follows:

$$O_s(x) = \{s^p(x), p \in \mathbb{N}\}.$$

A *s-orbit* is a part of E that is a the s-orbit of at least one $x \in E$. s is said to be a *cycle* if there is unique s-orbit O_s with nonzero cardinality; in this case, denoting p the cardinal number of the orbit, s is said to be a *p-cycle* with *support* O. A *transposition* is a 2-cycle. A p-cycle s with support $\{a_1,\dots,a_p\}$, where $a_i = s^{i-1}(a_1)$ will be denoted $(a_1 \dots a_p)$.

Remark 3.2. The p-cycle notation is not unique: for example, a transposition $(a_1 a_2)$ can also be written $(a_2 a_1)$.

Example 3.1. $\mathfrak{S}_4$ contains $\binom{4}{2} = 6$ transpositions. In the case where the permuted set is $\{0,1,2,3\}$, these are (01), (02), (03), (12), (13) and (23). The composition of any two of them with non-disjoint support is a 3-cycle, and there are $4 \times 2 = 8$ three-cycles. In addition, $\mathfrak{S}_4$ contains 3 permutations formed of 2 permutations with disjoint support, namely $(01)(23)$, $(02)(13)$ and $(03)(12)$; those, along with the identity ι_4, form the KLEIN *Vier-gruppe*, the smallest finite group with element orders all smaller than the group's cardinal. The remaining $4! - 18 = 6$ elements are the 4-cycles, such as (0312).

We recall the following essential results, see [3] for proofs:

Theorem 3.1. *All non-identity permutations of E can be written as a product of permutations with pairwise disjoint supports. In addition, this product is unique, up to the factors ordering.*

Corollary 3.2. *Any permutation of E can be written as a product of transpositions.*

Definition 3.3. Let $s \in \mathfrak{S}(E)$ and $m(s)$ the number of s-orbits in its decomposition as defined in Theorem 3.1. The *signature* of s is then defined as:

$$\varepsilon(s) = (-1)^{n-m(s)}.$$

Theorem 3.3. *If $s \in \mathfrak{S}(E)$ is a product of m transpositions, then $\varepsilon(()\,s) = (-1)^m$.*

Example 3.2. The 4-cycle (0312) can be written as $(03)(31)(12)$:

$$(0,1,2,3) \overset{(12)}{\longmapsto} (0,2,1,3) \overset{(31)}{\longmapsto} (0,2,3,1) \overset{(03)}{\longmapsto} (3,2,0,1)$$

This decomposition is not unique since we also have, *e.g.*,

$$(0312) = (12)(20)(03) = (03)(31)(12)(02)(02).$$

Its signature is $\varepsilon((0312)) = (-1)^{4-1} = (-1)^3 = (-1)^5 = -1$.

3.3 Application to Tetrahedron Subdivision

The idea developed hereafter is to retrieve directly from the canonical configuration the decomposition of any particular configuration pertaining to a given subcase, by the means of vertex permutations. More precisely, given a particular configuration, vertices will be permuted while leaving the overlying topology unchanged, so that the canonical configuration is retrieved. In other words, vertex indices might be changed, but the connectivity is not transported throughout the process. In order that the relative vertex locations remain constant, midpoints and face points must be transported consistently. We therefore must extend vertex permutations to midpoints and faces, as illustrated in the following example:

Example 3.3. Consider the transposition (01), which switches 0 and 1, while keeping the other vertices unchanged: for instance, the doubles $(1,2)$ and $(0,2)$ are switched. In order to preserve midpoint consistency, 5 and 6 must therefore be switched, too. In fact, the generalized (01) is the following map:

$$(0,1,2,3,4,5,6,7,8,9,a,b,c,d) \longmapsto (1,0,2,3,4,6,5,8,7,9,a,b,d,c).$$

In particular, edge 01 is left globally unchanged, but if, *e.g.*, the canonical configuration has edge $c8$, then the configuration obtained thanks to the generalized (01) permutation has edge $d7$ instead.

One can accordingly generalize all the elements of $\mathfrak{S}_4$ but, since this is both easy and lengthy to be described, it is left to the reader as an exercise. In everything that follows, all permutations will be implicitly meant in the "generalized sense", except otherwise mentioned.

Given a particular configuration K, once both the canonical configuration K_0 and the generalized permutation s necessary to go from K_0 to K are known, then the subdivision of K is immediately obtained by applying s to the element vertices in the decomposition of K_0, as detailed in Example 3.4.

Example 3.4. Consider the ambiguous case 4bα, which arises when in ambiguous case 4*b*, exactly two edges among 12, 02, 13 and 03 have equal lengths, while being shorter than the two other ones. To represent the class of all such configurations, the canonical case 4bα in [8], illustrated in Figure 4(a), is defined by:

$$|02| = |12| < |13| < |03|. \tag{1}$$

According to Table 2.2, the corresponding subdivision is as follows (oriented tetrahedra):

$$0123 = 7823 \cup a607 \cup a158 \cup a017 \cup a718 \cup 67a8 \cup 6a58 \cup 6278 \cup 6528. \tag{2}$$

Let us consider now another configuration which pertains to the same topological equivalence class, but with a different ordering of the edge lengths:

$$|02| = |12| < |03| < |13|, \tag{3}$$

as illustrated in Figure 4(c). Thanks to the help of $\mathfrak{S}_4$, it is straightforward to generate the decomposition of this particular configuration from the canonical (2): in fact, (3) is simply obtained by applying the transposition (01) to (1). The new configuration, shown in Figure 4(b), has therefore the following subdivision, obtained by applying (01) to (2):

$$1023 = 8723 \cup a518 \cup a067 \cup a108 \cup a807 \cup 58a7 \cup 5a67 \cup 5287 \cup 5627, \tag{4}$$

that might be reoriented, because $\varepsilon((01)) = -1$, such that the oriented configuration reads:

$$0123 = 7823 \cup 5a18 \cup 0a67 \cup 1a08 \cup 8a07 \cup 85a7 \cup a567 \cup 2587 \cup 6527. \tag{5}$$

Now, consider the particular configuration defined as follows:

$$|02| = |03| < |12| < |13|, \tag{6}$$

which can be deduced from the canonical configuration by applying the permutation (0312), that transforms (2) into (see Figure 5(b)):

$$3201 = 8501 \cup d738 \cup d265 \cup d328 \cup d825 \cup 78d5 \cup 7d65 \cup 7085 \cup 7605. \tag{7}$$

Finally, because $\varepsilon((0312)) = -1$, the latter subdivision needs to be reoriented, which leads to the proper (oriented) decomposition of the configuration (6) (see Figure 5(c)):

$$2301 = 5801 \cup 7d38 \cup 2d65 \cup 3d28 \cup 8d25 \cup 87d5 \cup d765 \cup 0785 \cup 6705. \tag{8}$$

Note that tetrahedra 2301 and 0123 have the same orientation; the subdivision provided in (8) is therefore a valid oriented subdivision of the tetrahedron in configuration (6).

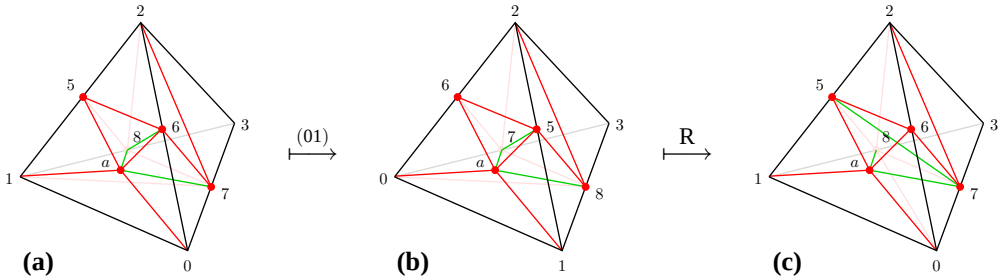

Figure 4. Ambiguous case 4bα: (4(a)) canonical configuration ($|02| = |12| < |13| < |03|$), (4(b)) image by (01), and (4(c)) reorientation to obtain the oriented subdivision of configuration $|02| = |12| < |03| < |13|$.

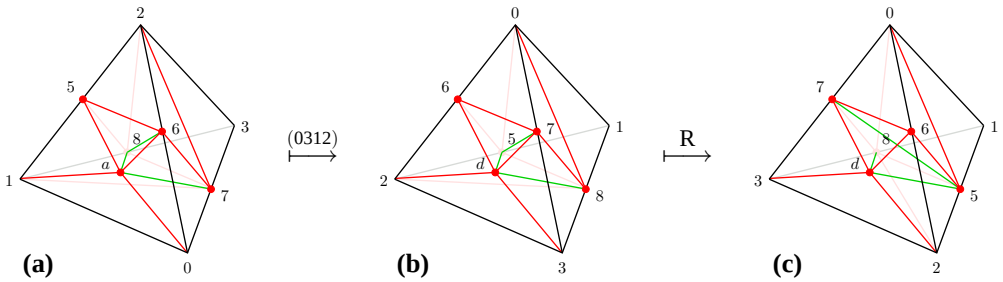

Figure 5. Ambiguous case 4bα: (5(a)) canonical configuration ($|02| = |12| < |13| < |03|$), (5(b)) image by (0312), and (5(c)) reorientation to obtain the oriented subdivision of configuration $|02| = |03| < |12| < |13|$.

As expected, obtaining the subdivision of the particular configuration is immediate, as soon as the $s \in \mathfrak{S}_4$ necessary to transform the canonical into the particular configuration is known. For instance, all the configurations that can be deduced from the canonical representation of case 4bα are detailed in Table 3.3. If the permutation $s \in \mathfrak{S}_4$ used to

Table 2. Case 4bα: permutations from the canonical representation to all possible configurations.

$s \in \mathfrak{S}_4$	Configuration	$s(0123)$	$\varepsilon(s)$
ι_4	$\|02\| = \|12\| < \|13\| < \|03\|$	0123	1
(01)	$\|02\| = \|12\| < \|03\| < \|13\|$	1023	-1
(23)	$\|03\| = \|13\| < \|12\| < \|02\|$	0132	-1
$(01)(23)$	$\|03\| = \|13\| < \|02\| < \|12\|$	1032	1
$(02)(13)$	$\|02\| = \|03\| < \|13\| < \|12\|$	2301	1
$(03)(12)$	$\|12\| = \|13\| < \|02\| < \|03\|$	3210	1
(0213)	$\|12\| = \|13\| < \|03\| < \|02\|$	2310	-1
(0312)	$\|02\| = \|03\| < \|12\| < \|13\|$	3201	-1

obtain the desired configuration has a negative signature, then the subdivision is composed of negatively oriented tetrahedra. This can be a problem, depending on the application using the refined mesh. As a general rule, it is good practice for mesh refinement software to be orientation-preserving. Therefore, whenever needed, *i.e.*, whenever $\varepsilon(s) = -1$, a final reorientation step must be performed, as done for instance in Figure 4(c) and Figure 5(c). This reorientation operation R can be interpreted as the application of any transposition $\tau \in \mathfrak{S}_4$ *to the whole configuration*, including the topological features, so that connectivities are left unchanged. A more geometric point of view is to regard R as a symmetry across any arbitrary plane, followed by a rotation and a transposition to retrieve the initial vertex coordinates.

Remark 3.3. One can observe that the permutations listed in Table 3.3, combined with the composition operation, form a group, and hence a subgroup of $\mathfrak{S}_4$. This is indeed not surprising, since the permutations do not change the topology of the tetrahedron (including edges imprinted onto it), and thus all configurations pertaining to 4bα can be deduced from each other. In particular, the configuration chosen to be canonical does not matter. The same argument applies to all cases, and thus each set of permutations (including the identity) associated with a canonical configuration is a subgroup of $\mathfrak{S}_4$, whose cardinal thus divides 24. Therefore, there can be only (including the canonical case itself) 1, 2, 3, 4, 6, 8, 12, or 24 particular configurations associated with a given case (*e.g.*, 8 with 4bα). This property is used to verify the code.

4 Results!

The results are presented as follows: first, we compare the theoretical decompositions of each ambiguous case to the subdivision as computed by the tessellator. Note that the theoretical decompositions are a matter of topology only, and therefore all of them can be presented over a unique generic tetrahedron. The actual instances of such ambiguous cases, on the contrary, can be encountered only for specific geometries, and therefore each case is represented by a particular tetrahedron that has the desired geometric properties, *i.e.*, edge length ratios.

4.1 Individual Elements

We first test the tessellator results for each of the cases we newly introduced in [8] and reminded in this report, in Table 2.2.

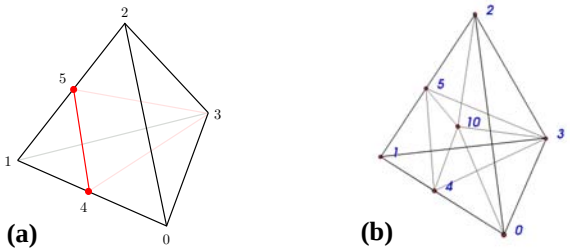

Figure 6. Ambiguous case 2a: theoretical (a) and computed (b) subdivisions.

Figure 6 compares the theoretical and computed decompositions of ambiguous case 2a. Similar comparisons are provided for all other ambiguous cases but, for the sake of saving space, all subcases of ambiguous cases 3a, 3c, 4a and 5 are gathered, respectively, in Figures 7, 8, 9 and 13. Due to the number of subcases, ambiguous case 4b is split across Figures 10, 11 and 12.

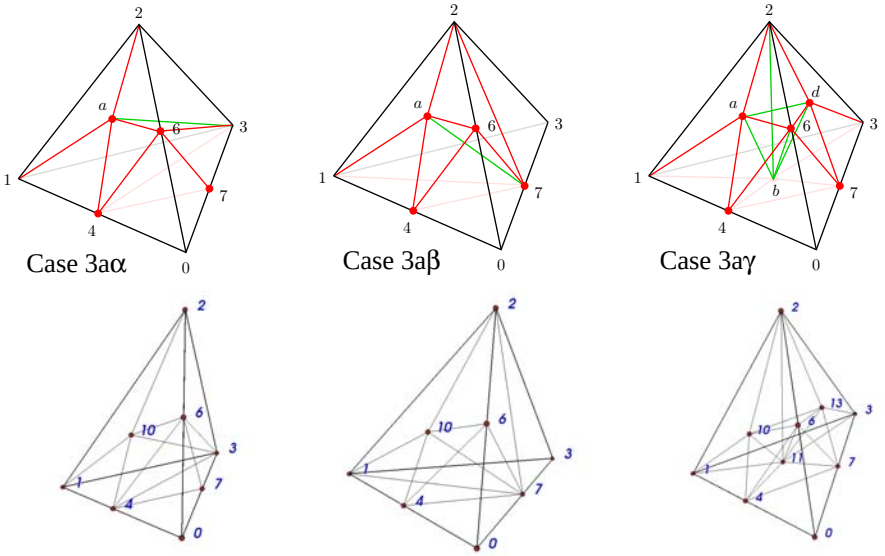

Figure 7. Ambiguous cases 3a: theoretical (upper) and computed (lower) subdivisions.

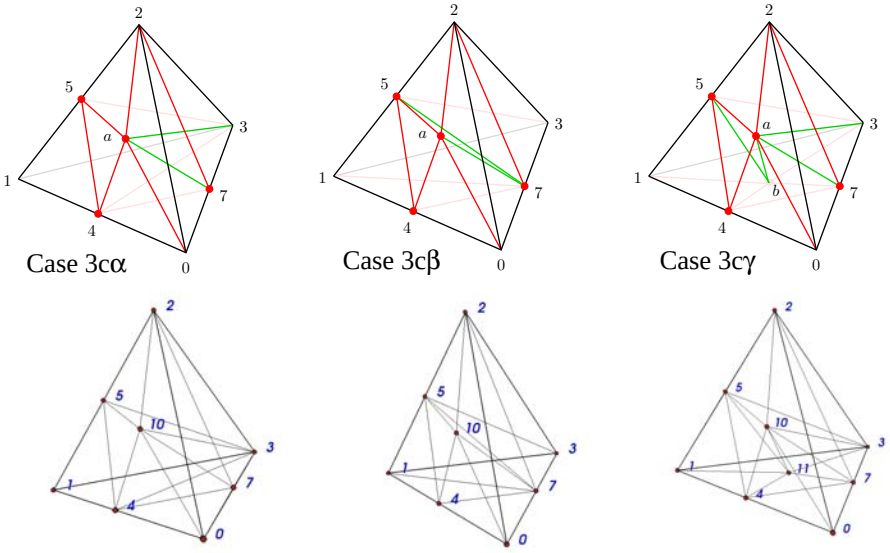

Figure 8. Ambiguous cases 3c: theoretical (upper) and computed (lower) subdivisions.

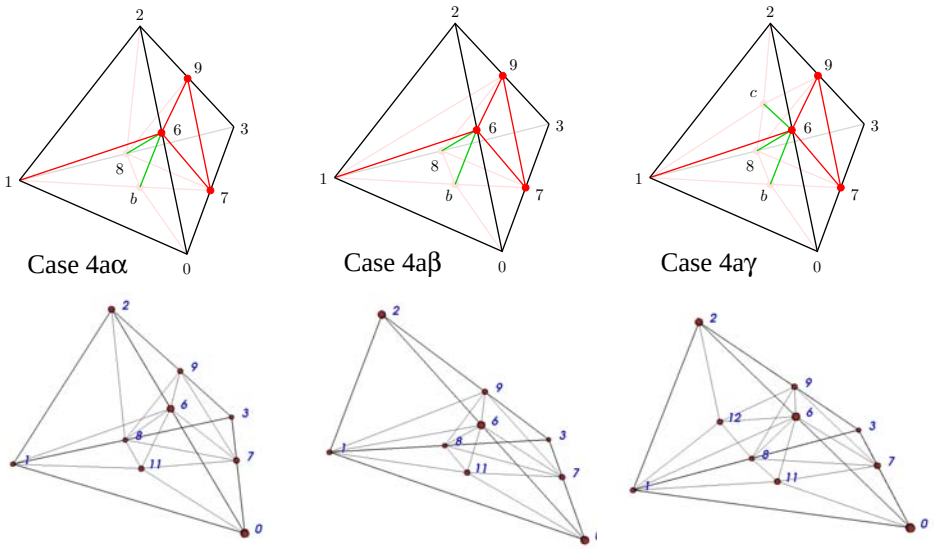

Figure 9. Ambiguous cases 4a: theoretical (upper) and computed (lower) subdivisions.

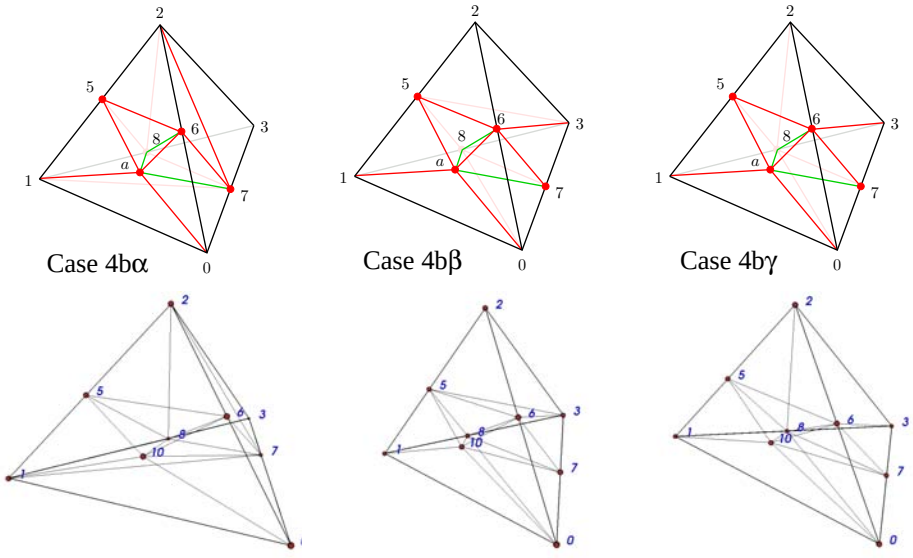

Figure 10. Ambiguous cases 4bα, 4bβ and 4bγ: theoretical (upper) and computed (lower) subdivisions.

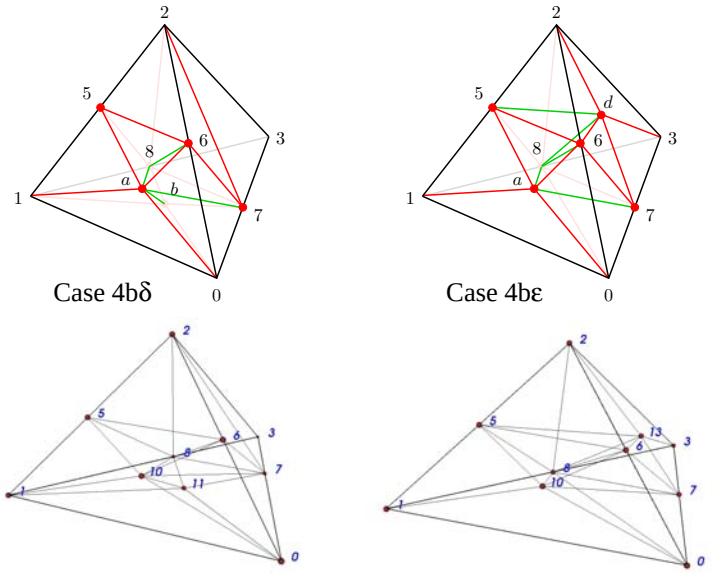

Figure 11. Ambiguous cases 4bδ and 4bε: theoretical (upper) and computed (lower) subdivisions.

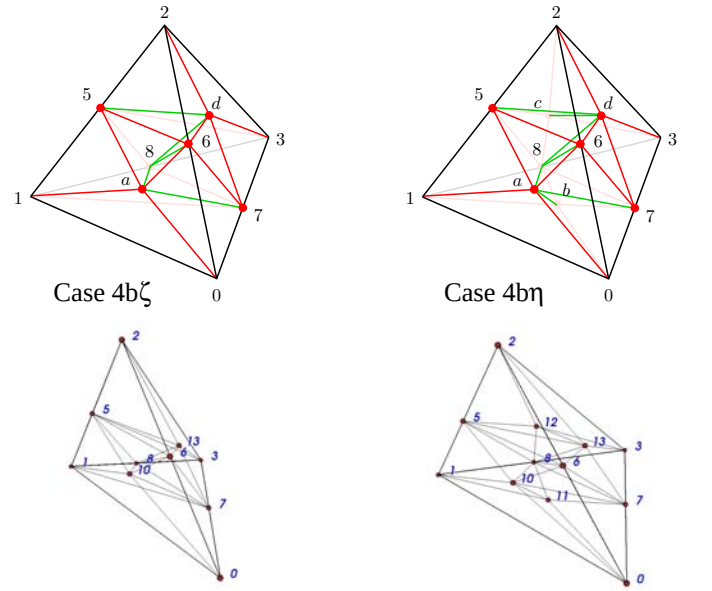

Figure 12. Ambiguous cases 4bζ and 4bη: theoretical (upper) and computed (lower) subdivisions.

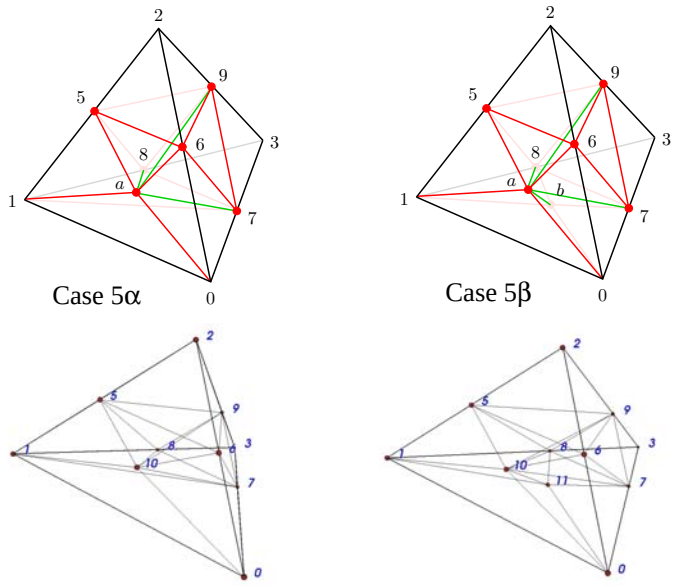

Figure 13. Ambiguous cases 5: theoretical (upper) and computed (lower) subdivisions.

4.2 Entire Meshes

Now that the tessellator is validated for all canonical configurations, we test it, *sine die*, with entire tetrahedral meshes.

4.2.1 Refinement According to the Distance to a Plane

The first example is that of a cuboid[2]. The initial mesh $\mathcal{T}^0_{\text{box}}$ has 553 points and 1627 tetrahedral elements, and is displayed in Figure 14. The mesh refinement, using the scheme

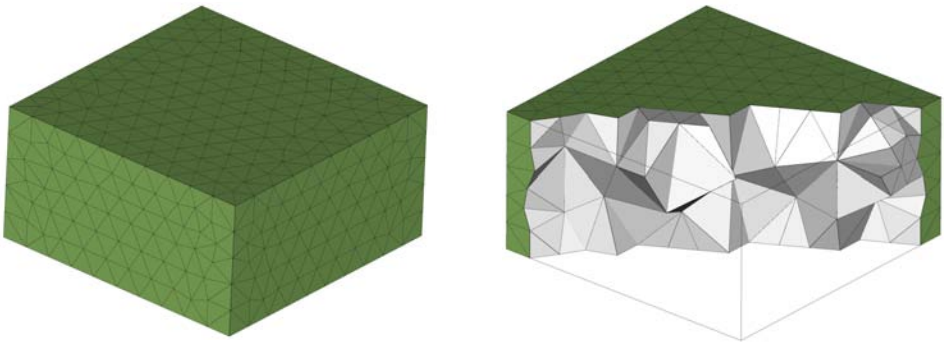

Figure 14. Initial mesh $\mathcal{T}^0_{\text{box}}$ of a cuboid: boundary (left) and clip across the mesh to show interior detail (right).

described in the previous sections, is performed according to the following criterion: An edge of length e and whose midpoint coordinates are (m_x, m_y, m_z) is subdivided if

$$\frac{(m_x + m_y + m_z - 2)^2}{3} < 2e^2. \tag{9}$$

Geometrically, this occurs when the distance between the edge midpoint and the plane with unit normal $\frac{1}{\sqrt{3}}(1,1,1)$ passing through the center of the mesh, $(0,0,2)$, is smaller than $\sqrt{2}$ times the edge length. Four successive refinement steps are performed from the initial mesh $\mathcal{T}^0_{\text{box}}$, leading to the refined meshes $\mathcal{T}^1_{\text{box}}$, $\mathcal{T}^2_{\text{box}}$, $\mathcal{T}^3_{\text{box}}$, and $\mathcal{T}^4_{\text{box}}$. Columns n_v and n_t in Table 3 respectively indicate the numbers of points and tetrahedra of these meshes, and cut snapshots are shown in Figure 15. It is also interesting to examine whether the scheme degrades mesh quality or not, since this is a typical weak point of mesh subdivision without subsequent mesh smoothing. In this goal, we are making use of the aspect-ratio measure, denoted ι, which is the most accepted estimate in the context of tetrahedral finite element analysis (*cf.* [2]). This measure is defined for each tetrahedron K by

$$\iota(K) = \alpha \frac{h_{\max}(K)}{r(K)}, \tag{10}$$

[2] also known, more formally but less conveniently, as *rectangular parallelepiped*.

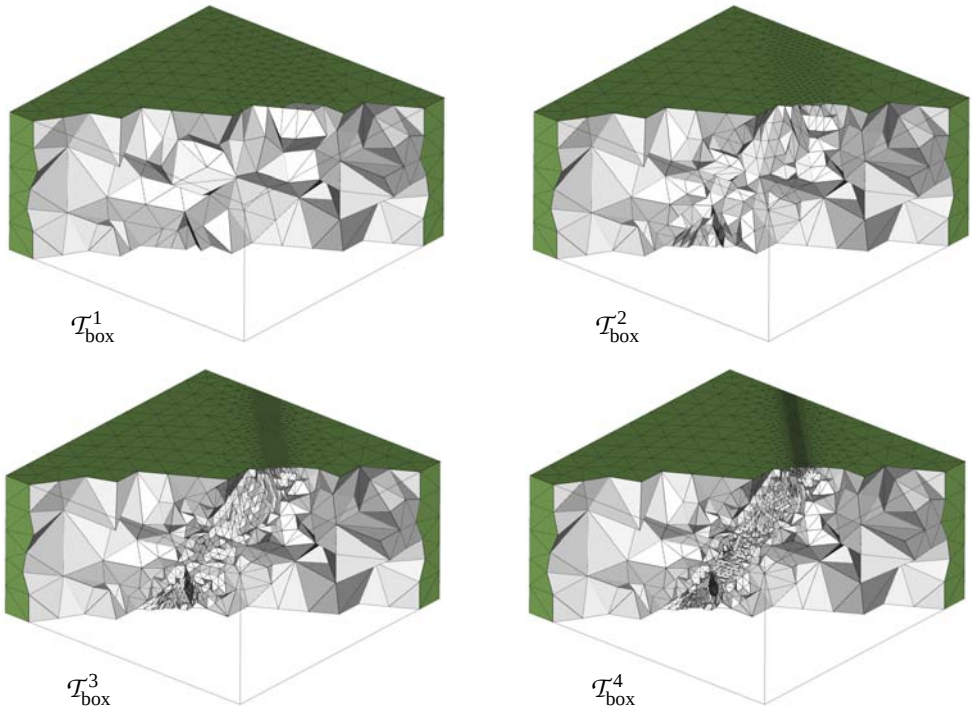

Figure 15. Streaming refinement of $\mathcal{T}^0_{\text{box}}$ according to the distance to a plane: 1 $\left(\mathcal{T}^1_{\text{box}}\right)$, 2 $\left(\mathcal{T}^2_{\text{box}}\right)$, 3 $\left(\mathcal{T}^3_{\text{box}}\right)$, and 4 $\left(\mathcal{T}^4_{\text{box}}\right)$ levels of refinement; some elements have been removed to show interior detail.

where $h_{\max}(K)$ and $r(K)$ respectively denote the longest edge length and the inradius of K. α is a normalization coefficient, so that $\iota(K) = 1$ when K is a regular tetrahedron. An elementary geometric calculation shows that $\alpha = \frac{1}{12}\sqrt{6}$. Indeed, 1 is the absolute *minimum* of ι and is reached only by regular tetrahedra. We have also written a VTK class that computes ι qualities: `vtkCalculateMeshQuality`.

Quality is also assessed by the means of the radius-ratio ρ, since there is an existing VTK quality that already computes it: `vtkMeshQuality`. This measure is defined for each tetrahedron K as follows:

$$\rho(K) = \beta \frac{R(K)}{r(K)}, \tag{11}$$

where $R(K)$ is the circumradius of K, and β is a normalization coefficient, such that $\rho(K) = 1$ when K is regular. Since, for any regular tetrahedron K_0, $R(K_0) = 3r(K_0)$, it follows immediately that $\beta = \frac{1}{3}$.

Remark 4.1. In the case of 2D simplicial (triangular) meshes, it is shown in [5] that ι and ρ have essentially similar behaviors, except for the fact that the latter has a critical point at its *minimum*, which might make it less suitable for iterative optimization procedures than the former, who has a salient point instead. In the case of 3D simplicial (tetrahedral) meshes, this result has not been proven but it is reasonable to also expect similar behaviors.

Table 3. Best, average, worst, and standard deviation of ι (aspect-ratio) and ρ (radius-ratio) qualities of meshes $\mathcal{T}^0_{\text{box}}$, $\mathcal{T}^1_{\text{box}}$, $\mathcal{T}^2_{\text{box}}$, $\mathcal{T}^3_{\text{box}}$ and $\mathcal{T}^4_{\text{box}}$.

	n_p	n_t	ι^-	$\bar{\iota}$	ι^+	σ_ι	ρ^-	$\bar{\rho}$	ρ^+	σ_ρ
$\mathcal{T}^0_{\text{box}}$	553	1627	1.05	1.85	12.2	0.73	1.01	1.7	37.5	1.78
$\mathcal{T}^1_{\text{box}}$	1742	7351	1	1.97	12.2	0.79	1	1.91	35.4	1.83
$\mathcal{T}^2_{\text{box}}$	6517	32887	1	2.16	18	0.94	1	2.3	116	2.63
$\mathcal{T}^3_{\text{box}}$	25691	140588	1	2.4	18	1.15	1	2.82	121	3.6
$\mathcal{T}^4_{\text{box}}$	103253	587283	1	2.71	32.6	1.44	1	3.58	384	5.38

The results of mesh quality assessments are provided in Table 3. Average qualities are not dramatically worsened, and the dispersion of elements is not substantially worse. Radius ratios (ρ) exhibit far greater dispersion than aspect ratios (ι), but that was expected, due to the nature of the estimates themselves. Indeed, in the case of triangle elements, this property has been established theoretically in [5]. The reduction in average quality is comparable to what is described in [2] for mid-edge based tetrahedral refinement. The fact that we are treating some of the isosceles faces as ambiguous cases might reduce the number of elements with bad aspect ratios (or radius ratio), because the 4 subfaces as shown in Figure 3(b) have better aspect ratios than the 2 subfaces of either possible decompositions in Figure 3(a).

4.2.2 Refinement Governed by an Analytical Size Map

The second example will consist in the refinement of the tetrahedral mesh of mechanical part, illustrated in Figure 16(a), left. The initial mesh, denoted $\mathcal{T}^0_{\text{part}}$ (*cf.* Figure 16(b)), has 28694 points and 150779 elements.

The edge subdivision criterion that will be used to refine that mesh relies on the principle of an edge size map specification [2]. In fact, the previous example can also be formulated in that way, but it is certainly more intuitive to present it in terms of distance to a plane. In the case of the current example, we are going to make use of a more evolved analytical size specification, based on the 3D lifting of a 2D closed smooth curve. An interesting category of such curves is the family of *LISSAJOUS curves*, see *e.g.* [4]. One of the simplest LISSAJOUS curves in the *xy* plane can be parametrized as follows:

$$\begin{cases} x(t) &= A\cos t \\ y(t) &= B\sin(2t), \end{cases} \tag{12}$$

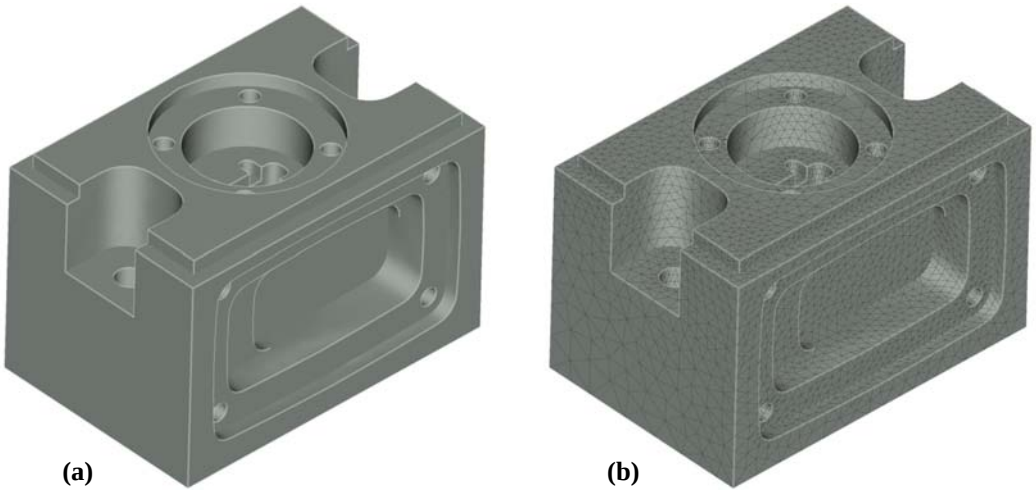

Figure 16. Boundaries of a mechanical part (a) and of the initial mesh $\mathcal{T}^0_{\text{part}}$ (b). Thanks to Pr Pascal Frey (University P. & M. Curie, Paris-6) for the mesh.

where A and B are two real positive constants, and $t \in [0, 2\pi[$. The curve obtained by taking $A = B = 1$ in (12) is shown in Figure 17(a).

In order to specify a size map, an implicit definition of the curve is needed. Using the fact that $\sin(2t) = 2\sin t \cos t$, it follows immediately that any point that belongs to the parametric curve defined in (12) satisfies the following implicit equation:

$$\frac{4x^4}{A^4} - \frac{4x^2}{A^2} + \frac{y^2}{B^2} = 0. \tag{13}$$

Conversely, any real double (x,y) that satisfies (13) also has

$$\frac{y^2}{B^2} = \frac{4x^2}{A^2} - \frac{4x^4}{A^4} = \frac{4x^2}{A^2}\left(1 - \frac{x^2}{A^2}\right), \tag{14}$$

and thus $1 - \frac{x^2}{A^2} \geq 0$, whence $-A \leq x \leq A$. Therefore, there exists a unique $\varphi \in [0, 2\pi[$ such that $x = A\cos\varphi$ and (14) then becomes:

$$\frac{y^2}{B^2} = 4\cos^2\varphi \sin^2\varphi, \tag{15}$$

hence $y = \pm B\sin(2\varphi)$. This means that, by taking t as either φ or $-\varphi$ (thanks to the parity of cos), any real double (x,y) that satisfies the implicit equation (13) can be parametrized as in (12). Since the double inclusion has been shown, it follows that (12) and (13) are equivalent. It is now therefore immediate to deduce a 3D surface by considering (13) in $\mathbb{R}^3$, where the generic coordinates are denoted (x,y,z). In that case, (13) is thus a generalized cylinder with directrix the LISSAJOUS curve (12) in the xy plane, and director curves the lines perpendicular to that plane. In all that follows, this generalized cylinder will be denoted $\mathcal{L}_{A,B}$; a section of $\mathcal{L}_{1,1}$, for $0 \leq z \leq 2$, is displayed by Figure 17(b).

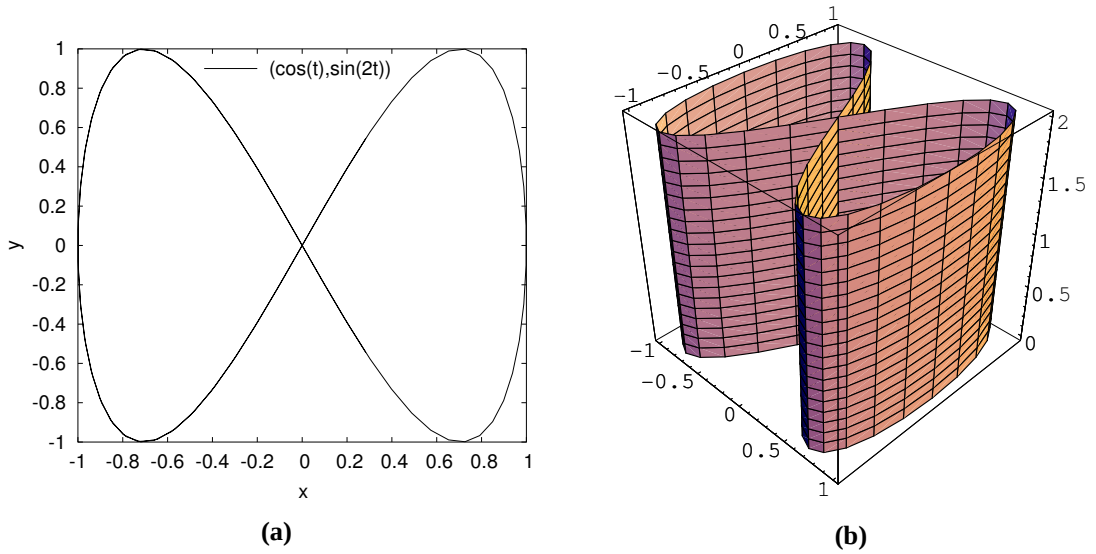

Figure 17. (a): the LISSAJOUS curve $(\cos(t), \sin(2t))$; (b): a section of the generalized cylinder with the latter as directrix, and director lines perpendicular to the *xy* plane.

Now, the right-hand side of (13) will be used to specify the edge size map, by defining, for all $(x,y,z,)$ in $\mathbb{R}^3$,

$$f(x,y,z) = \frac{4x^4}{A^4} - \frac{4x^2}{A^2} + \frac{y^2}{B^2},$$

and the refinement criterion will therefore be:

$$[f(m_x, m_y, m_z)]^2 < ke^2, \tag{16}$$

where, as previously, (m_x, m_y, m_z) are the coordinates of the midpoint of an edge with length e, and k is a positive real constant to be set, depending on the desired level of refinement progressivity. By definition, $\mathcal{L}_{A,B}$ is the 0-level set of f, which means that the edge size target on this surface is 0. On the other hand, the further an edge midpoint from $\mathcal{L}_{A,B}$, the looser the size specification implied by (16). Therefore, the mesh refinement process should "track" $\mathcal{L}_{A,B}$, and the tracking should become tighter when iterating refinement steps. Finally, we set A and B so that a complete z-section of $\mathcal{L}_{A,B}$ is intercepted by the initial mesh[3].

Mesh topology and quality results of 3 refinement steps are summarized in Table 4. Similar trends in quality were observed for the mechanical part as with the cuboid. A boundary overview and a close-up on some interesting features of the final mesh $\mathcal{T}^3_{\text{part}}$ are presented in Figures 19 and 20 display cuts across initial and refined meshes.

[3] the exact values used are $A = \frac{1}{30}$ and $B = \frac{1}{50}$.

Table 4. Best, average, worst, and standard deviation of ι (aspect-ratio) and ρ (radius-ratio) qualities of meshes $\mathcal{T}^0_{\text{part}}$, $\mathcal{T}^1_{\text{part}}$, $\mathcal{T}^2_{\text{part}}$ and $\mathcal{T}^3_{\text{part}}$.

	n_p	n_t	ι^-	$\bar{\iota}$	ι^+	σ_ι	ρ^-	$\bar{\rho}$	ρ^+	σ_ρ
$\mathcal{T}^0_{\text{part}}$	28694	150779	1.01	1.5	12	0.27	1	1.29	40.9	0.3
$\mathcal{T}^1_{\text{part}}$	62854	345529	1.01	1.63	19.9	0.39	1	1.44	168	0.65
$\mathcal{T}^2_{\text{part}}$	211480	1207377	1.01	1.85	33.2	0.59	1	1.75	427	1.42
$\mathcal{T}^3_{\text{part}}$	865695	5034682	1.01	2.11	46.4	0.81	1	2.19	890	2.84

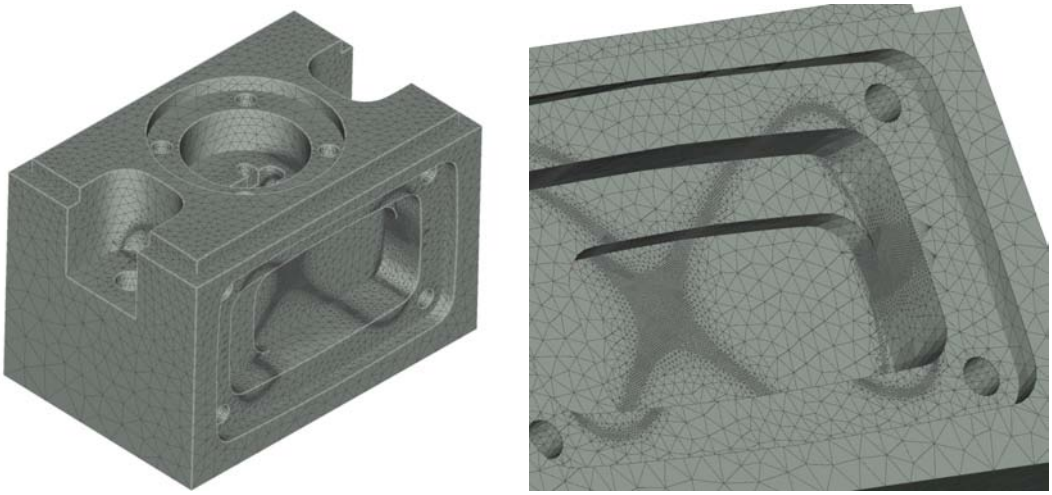

Figure 18. Boundary overview (left) and close-up (right) of the final refined mesh $\mathcal{T}^3_{\text{part}}$.

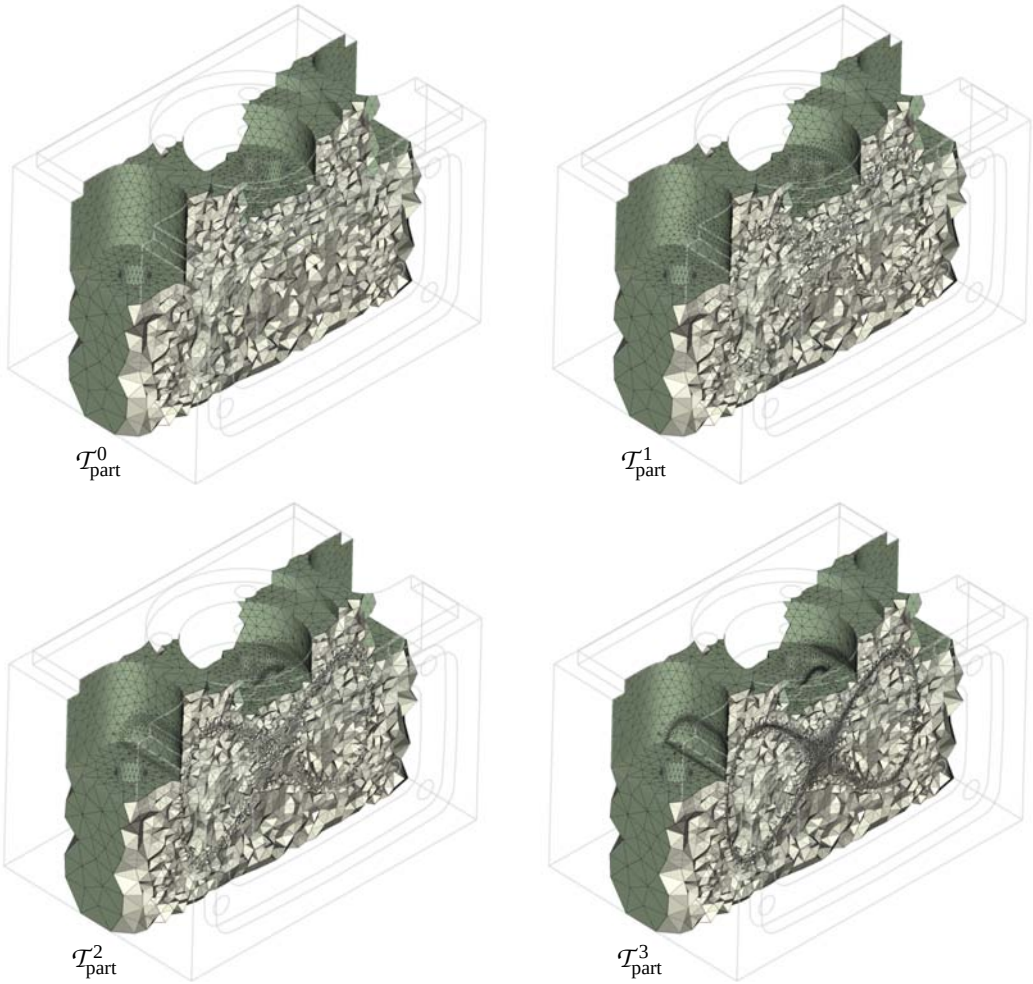

Figure 19. Streaming refinement of $\mathcal{T}^0_{\text{part}}$ according to the distance to a plane: 0 $\left(\mathcal{T}^0_{\text{part}}\right)$, 1 $\left(\mathcal{T}^1_{\text{part}}\right)$, 2 $\left(\mathcal{T}^2_{\text{part}}\right)$, and 3 $\left(\mathcal{T}^3_{\text{part}}\right)$ levels of refinement; some elements have been removed to show interior detail.

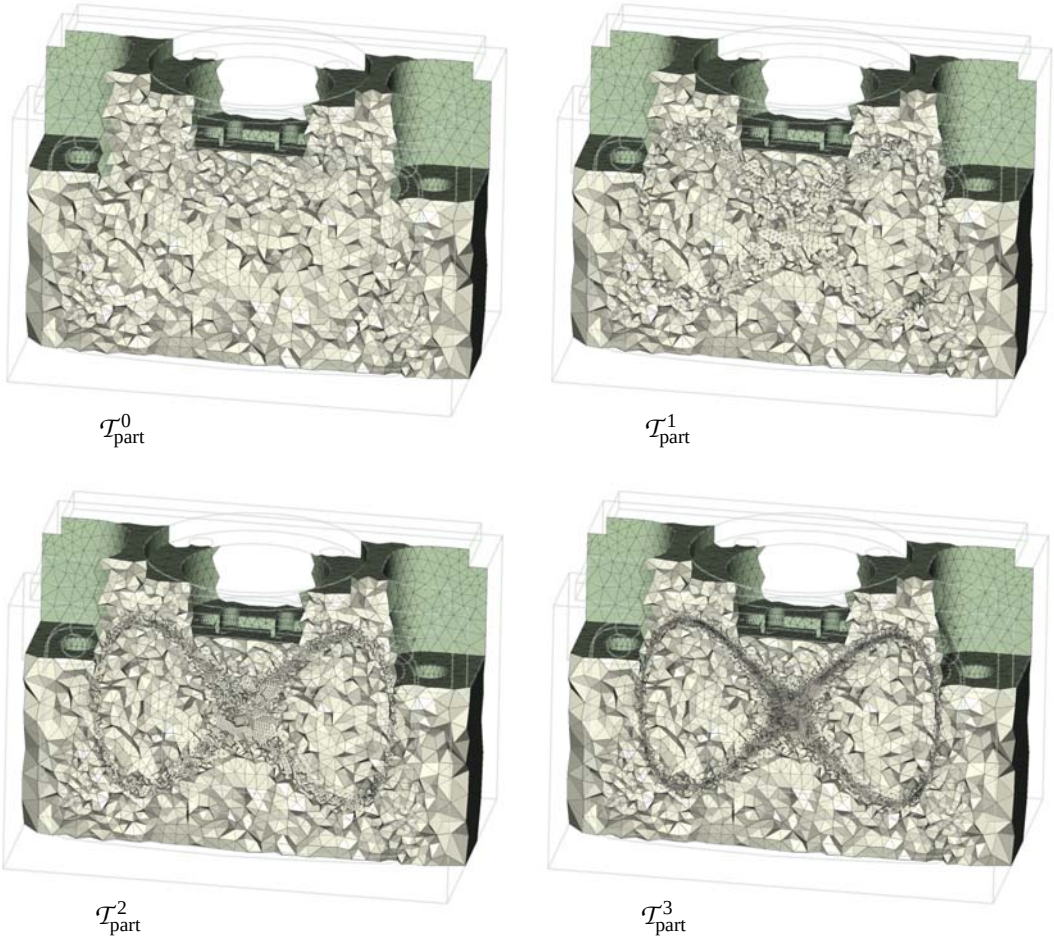

Figure 20. Streaming refinement of $\mathcal{T}^0_{\text{part}}$ according to the distance to a plane: 0 $\left(\mathcal{T}^0_{\text{part}}\right)$, 1 $\left(\mathcal{T}^1_{\text{part}}\right)$, 2 $\left(\mathcal{T}^2_{\text{part}}\right)$, and 3 $\left(\mathcal{T}^3_{\text{part}}\right)$ levels of refinement; some elements have been removed to show interior detail.

4.3 Speed!

The speed of execution of the algorithm is an important performance measure second only to adequate mesh quality. The mechanical part mesh was refined using the distance-from-a-plane subdivision metric with 0 through 4 levels of refinement, which generate increasingly large output meshes for the same input set (see Figure 21).

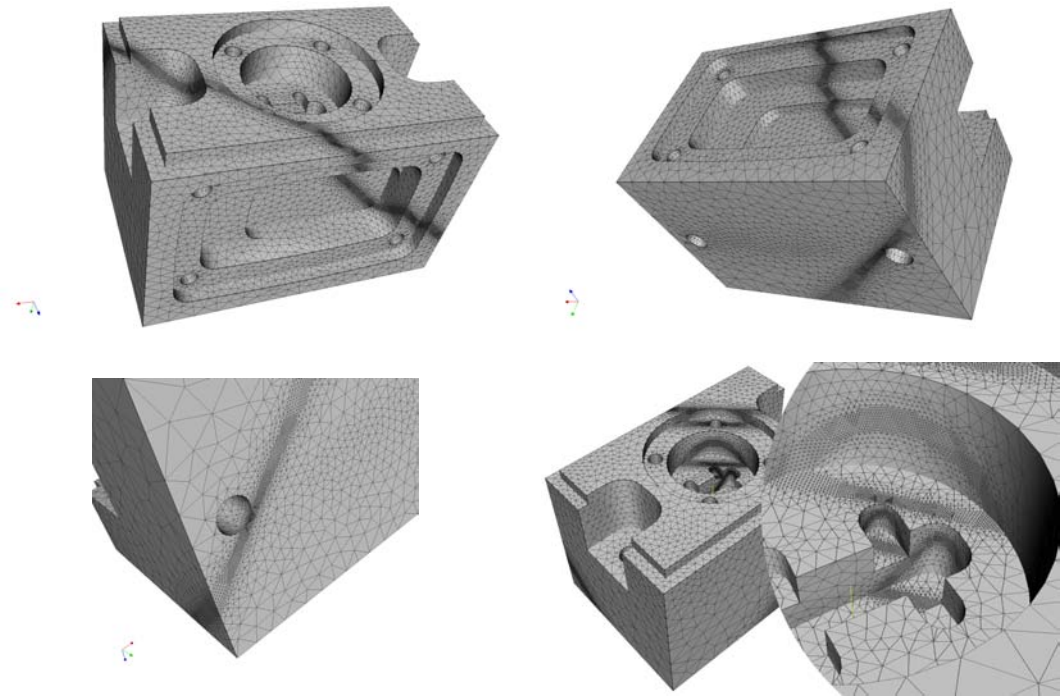

Figure 21. Several views of a 3-level refinement of the mechanical part's mesh $\mathcal{T}^0_{\text{part}}$ using the distance-from-a-plane subdivision metric.

The time required to perform the refinement and the throughput (in Mtets[4] per second) are shown in Table 4.3. The times are CPU times of the algorithm, compiled with gcc 3.3.3 on a single-CPU AMD Opteron 246 running Linux (Fedora Core 2) with 4 GB of memory. Mesh read and write times are not included, but the memory allocations required to store the output tetrahedra are included (as opposed to a true stream in which the results would be discarded after computation). Times are also shown for the cuboid in Table 4.3, but since the input is small and the mesh synthetic, they should probably be given less weight.

[4] millions of tetrahedra

Table 5. The speed of execution for varying levels of refinement of the mechanical part varies almost linearly with the number of output tetrahedra.

Refinements	Time $[s]$	Tetrahedra	Throughput $[\frac{\text{tets}}{s} \times 10^6]$
0	0.220	150779	0.687
1	0.458	277705	0.606
2	1.19	799149	0.674
3	4.36	2957365	0.679
4	17.2	11955114	0.697

Table 6. The speed of execution for varying levels of refinement of the cuboid varies almost linearly with the number of output tetrahedra.

Refinements	Time $[s]$	Tetrahedra	Throughput $[\frac{\text{tets}}{s} \times 10^6]$
0	0.00221	1627	0.737
1	0.0101	7351	0.726
2	0.0503	32887	0.654
3	0.214	140588	0.657
4	0.887	587283	0.662
5	3.757	2447890	0.652

4.4 Chance!

One of thing that affects the speed and quality of the output meshes is the chance that each subdivision case is encountered. We therefore present the case distribution histograms corresponding to 4 levels of distance-from-a-plane refinements for the cuboid and mechanical part meshes, respectively in Figures 23(a) and 23(b). Also shown, in Figure 23(c), is the histogram for a more organic mesh illustrated in Figure 22. Note that an additional case is mentioned: 3d; it corresponds to negatively oriented 3c configurations, that are treated independently only for practical implementation reasons.

Remark 4.2. It is interesting to notice that, for all non-ambiguous cases, any particular configuration can be obtained from the canonical one by *via* a positively oriented permutation, except for case 3c. This is why only this case enjoys the honor of a special treatment.

Global trends observed in Figure 23 are similar for the three meshes, although those exhibit substantially different topologies (*e.g.*, number of elements, genus) and geometric (*e.g.*, boundary curvature, quality) features. The fact that all cases are met is yet another global validation of the tessellator, since all cases outlined theoretically have been met and resolved. However, this does not mean that all subcases, and in particular ambiguous subcases, have been met. We therefore further refine the diagnostic, by examining the case counts for ambiguous cases at the end of the refinement process (level 4). Surprisingly enough, not a single ambiguous tetrahedron is met during the 4 steps of refinement of the

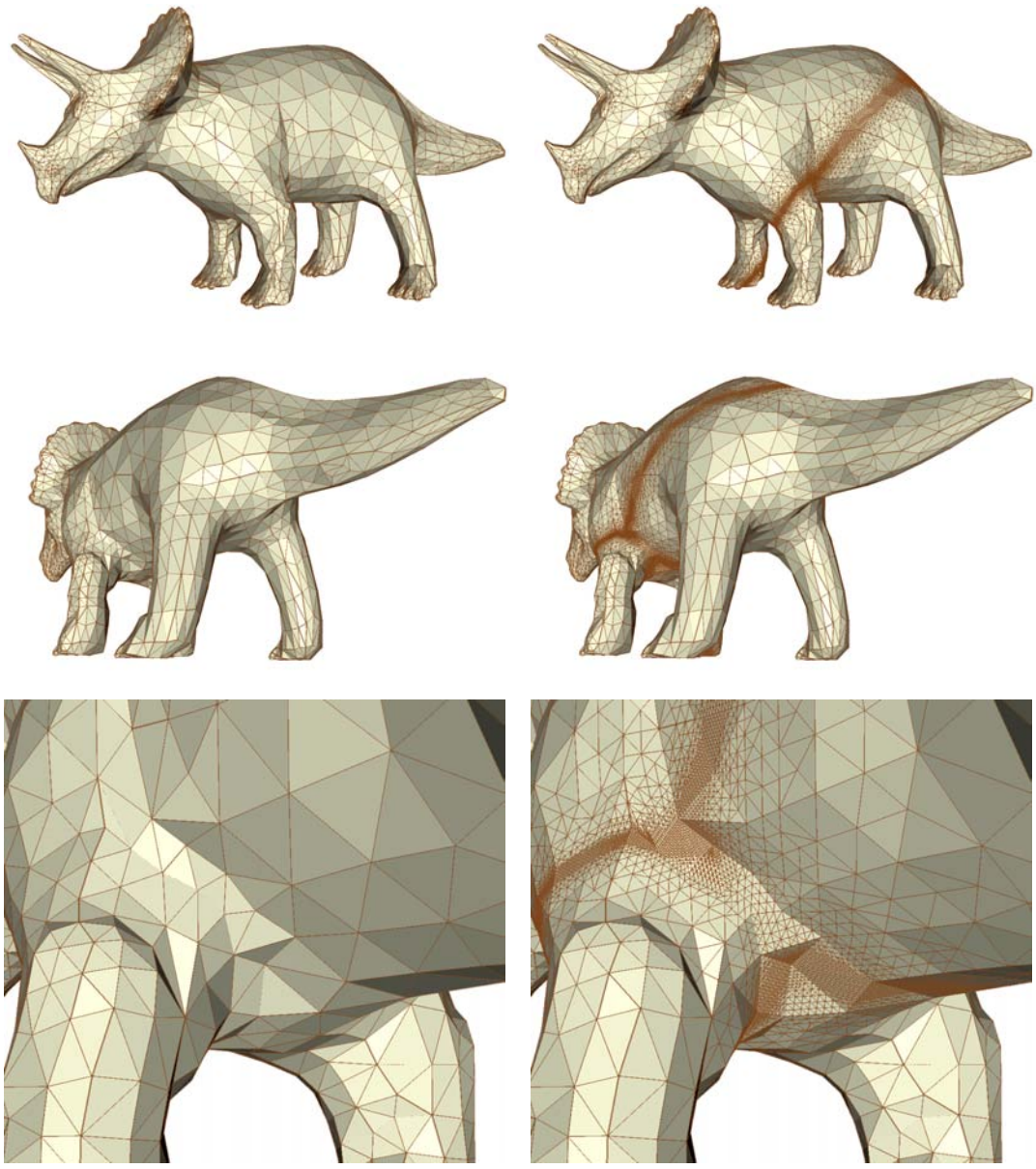

Figure 22. Several views of the boundaries of the initial mesh of a triceratops (left), and of the beauty queen (right) obtained after 4 steps of refinement governed by a distance-from-a-plane metric.

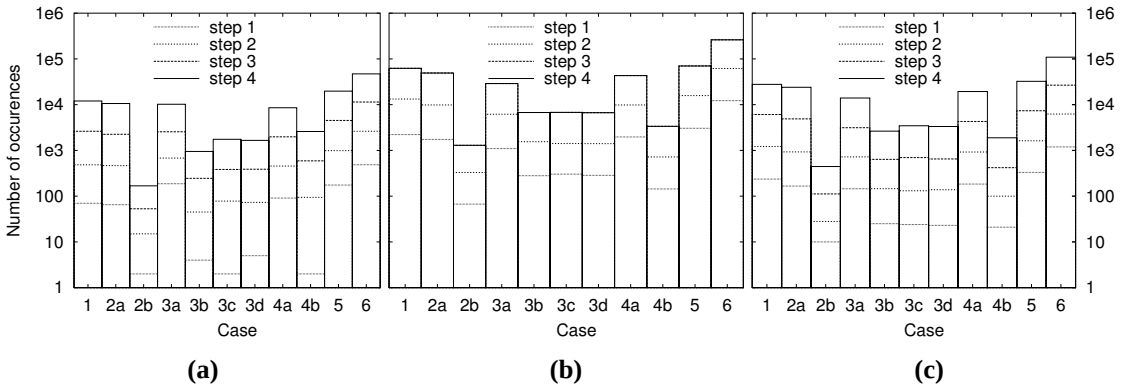

Figure 23. Total counts of the cases encountered at each of the 4 steps of refinement governed by a distance-from-a-plane metric of the cuboid (a), mechanical part (b), and triceratops (c) meshes; case 3d denotes negatively oriented 3c configurations.

mechanical part, and thus only the statistics for the cuboid and the triceratops are shown in Figure 24. Case 3d, introduced above for practical implementation reasons, is decomposed as the canonical 3c, in 3dα, 3dβ and 3dγ subcases; each of them is therefore only one negative permutation away from its canonical counterpart. It appears clearly that the similar trends previously observed for the distribution among the general cases no longer holds when considering ambiguous subconfigurations: the diversity of the original meshes is clearly reflected by great variations among ambiguous cases. Chances to hit an ambiguous case are much larger when dealing with a mesh that has been obtained from regular geometries, such as $\mathcal{T}^0_{\text{box}}$.

5 Conclusion

This report presents a technique for statelessly refining a tetrahedral mesh. Rather than coordinate face and edge subdivision information across elements of the mesh, subdivision templates and criteria are chosen so that they always yield identical results on shared faces and edges. Although this requires more computation (twice for each face and as many times per edge as there are tetrahedra that reference it), it bypasses any communication that might otherwise be required.

The main contribution of this paper above our previous work is that we now have an implementation, available in ParaView, while [8, 6] presented the theory of the new refinement scheme. This has allowed us in particular to assess mesh quality and algorithm speed. Two meshes were evaluated: a synthetic geometric mesh of a cuboid and the mesh of a mechanical part. The qualities of the refined meshes are comparable to other refinement schemes that use communication to resolve ambiguities.

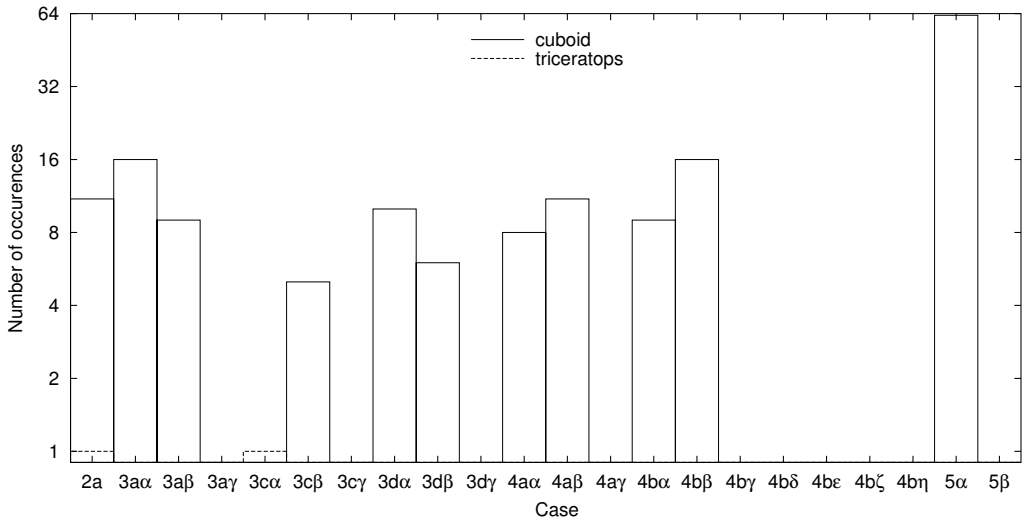

Figure 24. Total counts of the ambiguous cases encountered during the 4 steps of refinement governed by a distance-from-a-plane metric of the cuboid and triceratops; cases 3dα, 3dβ and 3dγ respectively denote negatively oriented 3cα, 3cβ and 3cγ configurations.

In addition to mesh quality, the time performance of the algorithm is characterized. The algorithm can generate approximately $600,000$ tetrahedra per second on modern hardware. More importantly, the algorithm scales linearly with the size of the output. Since the output vertices are stored on the heap during the recursive call to sample a tetrahedron, speed may likely be increased by preallocating a pool of memory for output vertices. This could also allow higher levels of refinement to be handled since the heap is not typically sized for such use.

Besides the performance of the algorithm, the relative frequency of ambiguous and unambiguous refinement templates presents an interesting trend. A version of the tessellator instrumented to count cases was used to examine the refinement of several meshes. Meshes of regular geometries tended to have higher numbers of ambiguous cases because they have many coplanar vertices and are often created by mesh generators that regularly sample space.

Possible future work includes allowing some canonical cases to choose between several subdivision templates to further optimize quality, depending on the particular geometry.

Acknowledgments

Thanks to Prof. P. J. FREY[5], Université P. & M. Curie, Paris, France, who provided the initial meshes of the mechanical part and triceratops used in this Report.

[5]URL: http://www.ann.jussieu.fr/~frey/

References

[1] A. J. Chung and A. J. Field. A simple recursive tessellator for adaptive surface triangulation. *Journal of Graphics Tools*, 5(3), 2000.

[2] P. J. Frey and P.-L. George. *Mesh Generation*. Hermes Science Publishing, Oxford & Paris, 2000.

[3] T. W. Hungerford. *Algebra*. Graduate Texts in Mathematics. Springer, 1997.

[4] J. D. Lawrence. *A Catalog of Special Plane Curves*. Dover Publications, New York, 1972.

[5] P. P. Pébay and T. J. Baker. Analysis of triangle quality measures. *Mathematics of Computation*, 72(244):1817–1839, 2003.

[6] P. P. Pébay and D. C. Thompson. Parallel mesh refinement without communication. In *Proc. 13th International Meshing Roundtable*, Williamsburg, VA, September 2004. Accepted.

[7] D. Ruprecht and H. Müller. A scheme for edge-based adaptive tetrahedron subdivision. In Hans-Christian Hege and Konrad Polthier, editors, *Mathematical Visualization*, pages 61–70. Springer Verlag, Heidelberg, 1998.

[8] D. C. Thompson, R. Crawford, R. Khardekar, and P. P. Pébay. Visualization of higher order finite elements. Sandia report, Sandia National Laboratories, April 2004.

[9] Luiz Velho. Simple and efficient polygonization of implicit surfaces. *Journal of Graphics Tools*, 1(2):5–24, 1996.

DISTRIBUTION:

1 Pr Richard Crawford
Mech. Engr., C2200
The Univ. of Texas at Austin
Austin, TX 78703

1 Patricia Howard
Comp. & Appl. Mathematics
Rice University
6100 Main St. - MS 134
Houston, TX 77005-1892

1 Rahul Khardekar
1634 Oxford Street
Apt 305
Berkeley, CA 94709

2 Will Schroeder
Kitware
28 Corporate Drive
Suite 204
Clifton Park, NY 12065

1 Timothy J. Baker
Mechanical & Aerospace Engineering Department
Engineering Quadrangle
Princeton University
Princeton, NJ 08544

1 Pr Jérôme Pousin
I.N.S.A. Lyon
Center for Mathematics
Bâtiment Léonard de Vinci
69621 Villeurbanne cedex, France

1 Pr Pascal Frey
Laboratoire J-L Lions
Université Pierre et Marie Curie
Boîte courrier 187
75252 Paris cedex 05, France

1 MS 9051
Andy McIlroy, 8351

4 MS 9051
Philippe P. Pébay, 8351

1 MS 9915
Curtis L. Janssen, 8961

1 MS 9915
Mike Koszykowski, 8961

1 MS 9915
Mitchel W. Sukalski, 8961

1 MS 9217
Paul T. Boggs, 8962

1 MS 9217
Kevin R. Long, 8962

1 MS 9012
Jerry A. Friesen, 8963

1 MS 9012
Gary J. Templet, 8963

4 MS 9012
David C. Thompson, 8963

1 MS 0382
Kevin D. Copps, 9143

1 MS 1110
David M. Day, 9214

1 MS 1110
Louis Romero, 9214

1 MS 0822
Brian N. Wylie, 9227

1 MS 0822
Lee Ann Fisk, 9227

3 MS 9018
Central Technical Files, 8940-1

1 MS 0899
Technical Library, 9616

1 MS 9021
Classification Office, 8511, for Technical Library, MS 0899, 9616 DOE/OSTI via URL