Engineering with Computers (2006) 22: 75–93
DOI 10.1007/s00366-006-0020-3

ORIGINAL ARTICLE

David C. Thompson · Philippe P. Pébay

Embarrassingly parallel mesh refinement by edge subdivision

Received: 22 September 2004 / Accepted: 8 November 2005 / Published online: 23 June 2006

Abstract We have previously proposed a new technique for the communication-free adaptive refinement of tetrahedral meshes that works for all configurations. Implementations of the scheme must deal with all possible geometric configurations, which results in a large number of cases that in turn result in practical programming issues. In this article, we address this issue with a `Python` script that generates C++ code using the symmetric group $\mathfrak{S}_4$ acting over canonical topological and geometric configurations. We then analyze the performance of the technique by characterizing (a) mesh quality, (b) execution time and parallel speedup, and (c) traits of the algorithm that could affect quality or execution time differently for different meshes and different mesh refinement strategies. This article also details the method used to debug the many subdivision templates that the algorithm relies upon. Mesh quality is on par with other similar refinement schemes, and we suggest a more elaborate technique that may substantially improve mesh quality. We show that throughput on modern hardware can exceed 600,000 output tetrahedra per second per processor, and that the method is embarrassingly parallel—assuming the application has partitioned the input properly.

Keywords Adaptive tetrahedral tessellation · Parallel mesh refinement · Streaming subdivision · Symmetric groups

1 Introduction: streaming mesh refinement

In [14], we presented the theory and algorithm for a communication-free tetrahedral edge-subdivision refinement scheme. In the present article, we focus on:

1. The practical implementation of the refinement technique in a way that allows its application to arbitrary subdivision schemes, and
2. The results obtained with the actual implementation of the method, both in terms of mesh quality and of execution speed.

In addition, we provide insight about the specifics of the method that leave room for improvement. Prior to that, we recall the context of *streaming mesh refinement*, along with a summary of the algorithm we have proposed.

1.1 Edge-based tetrahedral mesh refinement

The initial motivation for this work is the visualization of higher order finite element numerical simulations. Finite element techniques that use higher order (i.e., nonlinear) polynomial maps, $\Phi: R \rightarrow F$ and $\Xi: R \rightarrow X$, from some parametric space, R, into the model's geometric space, X, and solution space, F, of some set of differential equations are becoming more common. Although visualization techniques that use Φ and Ξ directly are under development, being able to take advantage of the huge number of techniques aimed at linear maps is highly desirable. Our approach to do this is through an adaptive simplicial tessellation of the parameter space, R, into regions where Φ and/or Ξ are approximately linear. Specifically, this article describes a subdivision scheme based on error metrics evaluated at edge midpoints of some initial (crude) tessellation, as shown in Fig. 1. Nevertheless, the method we propose is not limited to visualization, since it takes the edge refinement criterion as an input and is therefore application independent; e.g., it may also be used as a mesh refinement scheme for numerical simulations.

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 capture large changes.

D. C. Thompson (✉) · P. P. Pébay
Sandia National Laboratories, MS 9152,
P.O. Box 969, Livermore, CA 94550, USA
E-mail: dcthomp@sandia.gov
E-mail: pppebay@ca.sandia.gov

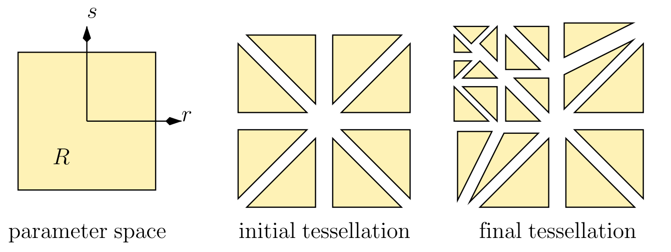


Fig. 1 Given some initial tessellation of a finite element's parameter space, we subdivide edges until some application-dependent criterion is met and we are left with a new, refined simplicial complex

Tetrahedral mesh refinement is twofold: first, one needs a criterion to decide which elements should be refined; second, subdivision operators must be applied to these elements while maintaining mesh conformity. See [4] for a detailed survey of subdivision operators; for this paper, the most important property of a subdivision operator is locality. If a subdivision operator is only allowed to insert points interior to a single element (so that edges and faces remain unchanged), refinement is clearly localized, but the quality of the resulting tetrahedra will be extremely poor. As the subdivision operator is allowed to make non-local changes (such as moving nodes, splitting edges, or splitting faces shared by other elements), some form of communication or storage is required to produce a conforming mesh. However, if both the subdivision criterion and the subdivision operator must make the same decisions for each edge and face across all tetrahedra that share it, no communication or storage is required even though the operation is non-local. This is the approach we will pursue, even though it rules out any subdivision criterion/operator pair that tessellates a boundary face or edge using element information not on that face or edge. For instance, a subdivision criterion based on the aspect ratio of a tetrahedron, T, will not work because when neighbors of T are processed, the aspect ratio of T will not be available to them and any boundary they share with T may not be compatible with the refinement of T.

In practice, numerous techniques for computing error metrics or size specifications along edges have been proposed (e.g., Lo [9] define a target edge length over all of space and subdivide a sorted list of edges exceeding the target length, while Labbé et al. [7] evaluate the difference between a metric that transforms each element to an equilateral triangle and a continuous "size specification" metric describing an ideal element shape, using node relocation, edge swapping, and node insertion to minimize the metric difference) while face-based error metrics are much harder to devise and compute. It is therefore natural to use an edge-based refinement approach; this means that there are two tasks to be performed by the 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 an initial template require subdivision.

It is therefore natural to implement these tasks separately: they exist as separate C++ classes so that the same templates for subdivision could be applied to many different subdivision decision algorithms. Therefore, the refinement task per se is application-independent, while the criteria that decide whether edges require subdivision (e.g., geometric distance, scalar field non-linearity, error metrics, etc.) are to be specified by the user, and are not discussed in this article.

This is not the first time adaptive tessellation using edge-based subdivision has been considered; Velho [19] and later Chung and Field [2] present a method for the streaming refinement of triangular meshes using edge subdivision, but do not extend the result to tetrahehedra. Ruprecht and Müller [15] propose a tetrahedral edge-subdivision algorithm but either do not enforce compatibility or require neighborhood information and storage space to hash shared output geometry. Oliker et al. [12] use three templates for tetrahedral edge subdivisions and propagate information across processor boundaries, changing the templates until the mesh is compatible.

Since our applications are intrinsically 3-D (e.g., volume rendering and isosurfacing), refinement methods that only address simplicial complexes in lower dimensions are inadequate. In addition, collections of simplices that do not meet the requirements of a simplicial complex (in particular, compatibility) are insufficient since algorithms such as isosurfacing will produce inconsistent output. Solutions to that problem that involve hashing output geometry, so that simplices share common vertices, edges, and faces, can require storage on the order of the size of the output and, more importantly, can require communication between processes when tessellation is performed in parallel. In cases where higher order finite elements are being rendered for visualization and not analyzed, the subdivided simplices are not ever stored—they are sent to a graphics card for immediate rendering via OpenGL. Because OpenGL does not require output geometry to share vertices, it is a waste of resources to spend time insuring the subdivided simplices are in a shared form. This is especially true for view-dependent rendering techniques, where a tessellation of the mesh is produced for each frame rendered. In the case of mesh refinement combined with parallel computing of numerical solutions, it is critical to limit communication between processors, since network bandwidth is the bottleneck and will so remain in the foreseeable future. For these reasons, devising a refinement scheme that produces compatible elements without any communication is highly desirable. This would, in particular, make it viable for streaming large datasets and for parallel processing.

1.2 Streaming mesh refinement

Our approach consists of *streaming* the data set through a refinement filter, as with [2] and [19]. However, contrary

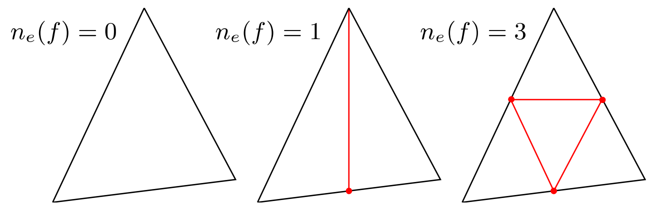


Fig. 2 Unambiguous face subdivision when 0, 1 or 3 edges are split

to these two techniques, we stream tetrahedra as well as triangles. Our refinement filter is a direct extension of [15], but insures compatibility in all conditions without neighborhood information: [15] was left with face ambiguities that could only be resolved via inter-element communication.

Streaming refinement requires each tetrahedron to be processed independently of its neighbors while remaining compatible with them. In order to be compatible, the edge subdivision algorithm must make the same decision about whether subdivision should occur for a given edge regardless of the tetrahedron it bounds—but as we explained earlier, this is application dependent. Additionally, the subdivision of any face f of a tetrahedron must be identical to the subdivision for the same face of any neighboring tetrahedron. As illustrated, there is no ambiguity regarding the subdivision of f when the number $n_e(f)$ of its split edges is either 0, 1 or 3 (Fig. 2). However, a topological ambiguity does arise when exactly two edges are split, as two distinct face subdivisions are possible (cf. Fig. 3). [15] used geometry—the longest edge criterion—for deciding on a subdivision template so that adjacent simplices produce compatibly-tessellated boundaries, as shown in Fig. 3. This decision removes the requirement of communication, because the same criterion is applied to any tetrahedron that shares the face, while producing the best possible triangle aspect ratio among the two possible solutions. Given a tetrahedron, there are $2^6 = 64$ possible choices for the set E of edges to be split. [15] showed that these possibilities reduce, via vertex permutations, to ten topological cases, denoted—and we will use the same nomenclature—0, 1, 2a, 2b, 3a, 3b, 3c, 4a, 4b, 5 and 6: the number is the cardinality n of E (i.e., the number of edges, n, of the tetrahedron that must be subdivided), followed by a letter when different topological configurations exist for a same n. Different topological configurations exist when one configuration of n edges to be split cannot be transformed into another by a vertex permutation. For example, compare the tetrahedra for case 3a and 3c shown[1] in Fig. 4. In case 3a, all three edges to be split share a common vertex while in 3c, they do not. There is

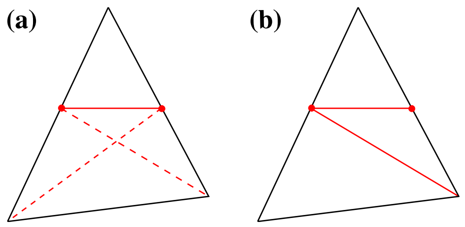


Fig. 3 The topological ambiguity when $n_e(f) = 2$ (**a**) is resolved by deciding to connect the longest edge midpoint to the opposite corner (**b**)

no transformation between the two cases such that a proper tessellation of one would be transformed into a proper tessellation of the other. Ruprecht and Müller showed that a valid tetrahedralization exists for every set of boundary triangles produced by this scheme [15]. However, the longest-edge criterion does not always specify how to subdivide faces with two split edges.

1.3 Geometric ambiguities

Indeed, geometrically ambiguous cases occur when a face can be split into two different ways despite the

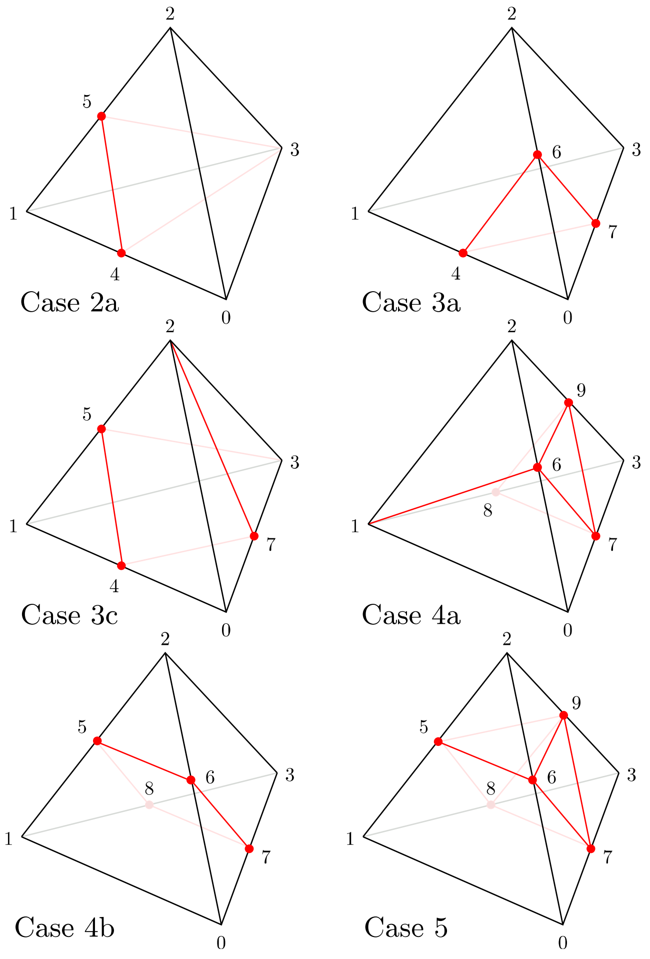


Fig. 4 Potentially ambiguous configurations

[1]In the online (color) version, all figures will display edges interior to a displayed face in red, edges interior to a tetrahedron in green, and others in black; black and red respectively turn into grey and pink when the edge is not in the foreground. In the print version, tetrahedra will be illustrated with unobscured bounding edges in bold lines, obscured bounding edges in narrow lines, and interior edges in dashed lines.

longest edge criterion, i.e., whenever at least one face has exactly two edges of equal length that must be split. These are the cases that are not handled by [15] and therefore, in all that follows, they will be simply referred to as *ambiguous cases*. Such ambiguities may only arise in cases 2a, 3a, 3c, 4a, 4b or 5, depicted in Fig. 4. Ambiguous faces can be triangulated in two different ways, and [15] do not propose a solution to resolve that ambiguity, other than allowing either for incompatible neighboring elements or communication between them. Since none of these solutions are acceptable, we presented a scheme for refining tetrahedral meshes that does not require neighborhood information in [14]. This article focuses on the practical aspects of implementing the technique and on measuring the performance of the algorithm.

We now recall in Table 1 the summary of the tetrahedral subdivisions for canonical unambiguous configurations as we implemented them, based on [15]. For the sake of brevity, in this table and for the rest of the article, we index edges of a generic tetrahedron from 0 to 5, in the following order: 01, 12, 02, 03, 13 and 23, and to denote by $\underline{i}$ the length of edge i.

1.4 A streaming solution to geometric ambiguities

By definition, any face f that remains ambiguous despite the longest edge criterion is isosceles. The face f can

Table 1 Subdivisions of canonical unambiguous cases (topological classification Ruprecht and Müller [15], geometric implementation Pébay and Thompson [14])

Case	Geometry	Tetrahedral subdivision
1		0423 4123
$2a^u\alpha$	$\underline{0} < \underline{1}$	0453 0523 4153
$2a^u\beta$	$\underline{0} > \underline{1}$	0423 4523 4153
2b		0493 4193 0429 4129
$3a^u$	$\underline{0} > \underline{2} > \underline{3}$	0467 4123 4263 4673
3b		0742 4782 4812 7382
$3c^u\alpha$	$\underline{1} < \underline{0} > \underline{3}$	4275 4207 4153 5743 5273
$3c^u\beta$	$\underline{1} > \underline{0} < \underline{3}$	0527 0574 7145 7153 5273
$3c^u\gamma$	$\underline{1} < \underline{0} < \underline{3}$	4275 4207 7145 7153 5273
$3c^u\delta$	$\underline{1} > \underline{0} > \underline{3}$	0527 0574 4153 5743 5273
$4a^u\alpha$	$\underline{3} < \underline{4} < \underline{5}$	7893 6978 6018 6708 1269 1689
$4a^u\beta$	$\underline{3} < \underline{4} > \underline{5}$	7893 6978 6018 6708 1268 2689
$4a^u\gamma$	$\underline{3} > \underline{4} < \underline{5}$	7893 6978 6718 6701 1269 1689
$4b^u\alpha$	$\underline{1} < \underline{2} < \underline{3} < \underline{4}$	6815 6801 6708 8732 6852 6827
$4b^u\beta$	$\underline{1} < \underline{2} < \underline{3} > \underline{4} > \underline{1}$	6815 6871 6701 8732 6852 6827
$4b^u\gamma$	$\underline{1} < \underline{2} > \underline{3} < \underline{4} > \underline{1}$	6815 6801 6708 8736 6852 6823
$5^u\alpha$	$\underline{1} < \underline{2},\underline{3} > \underline{4}$	7893 6529 5718 5701 5760 5796 7859
$5^u\beta$	$\underline{1} < \underline{2},\underline{3} < \underline{4}$	7893 6529 0567 0578 0581 5796 5789
6		7893 6529 4158 0467 6458 6598 6978 6748

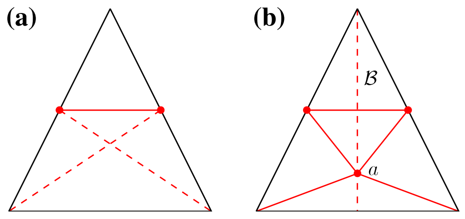


Fig. 5 Ambiguous face refinement: **a** two different subdivisions are possible, and **b** unambiguous subdivision thanks to a point insertion

therefore be unequivocally subdivided into a smaller isosceles triangle and an isosceles trapezoid with two possible triangulations of the latter, as illustrated in Fig. 5. We propose resolving this ambiguity by inserting a new point, a, onto the orthogonal bisector $\mathcal{B}$ of the unsplit edge (which is also the median, the angle bisector and the altitude, because the face is isosceles), the angle bisector of the vertex opposite the base of the trapezoid, as shown in Fig. 5. This point insertion yields an unambiguous subdivision of f with five triangles and, moreover, this triangulation is symmetric about b; therefore, if f is shared by two tetrahedra σ_1 and σ_2, this strategy guarantees that the triangulations of σ_1 and σ_2 will be compatible across f. In other words, we have resolved the geometric ambiguity while preserving both element compatibility and our streaming requirement (communication free). Applying this technique to each geometrically ambiguous face that arises during the refinement raises two questions: exactly where to put the point, and, once the point has been placed, how to decompose the tetrahedron.

In terms of topological compatibility, the point may be placed anywhere along the part of the base edge bisector that is interior to the isosceles trapezoid. However, geometrically speaking, not all placements are the same, since triangle quality will vary, which will in turn have an effect on tetrahedral quality [4]. Rather than relying on intuition, we assessed in [14] the quality of the resulting face subdivision in terms of triangle symmetrized aspect ratio [13]. The outcome is that the optimal position varies with the initial face shape but that, nonetheless, a good approximation is obtained by inserting a at one-fourth of the altitude from the unsplit edge. In general, note that this is *not* the intersection of the two possible edges (dotted) in Fig. 5.

In [14], we categorized all ambiguous cases, by the means of canonical configurations as had been done by [15] for unambiguous cases. In fact, we showed that there are exactly 19 such canonical cases, that we call out with their topological case number, the exponent ^{a}, and a Greek letter appended when several geometric configurations exist. Note that a tetrahedron (cases 3a, 3c, 4a, 4b, and 5) may have from 0 to 3, 0 to 2, 0 to 2, 0 to 4, or 0 to 2 ambiguous triangular faces, respectively. The exact number of subcases is determined by the particular edges shared between ambiguous triangles on the tetrahedron and their relative lengths. Where a face needs an additional point, we use a letter to reference that point. We use the letter a, b, c, or d for points on faces 012, 031, 132, or 023, respectively. Note that the position of a point inserted to resolve a geometric ambiguity depends on which edges of that face are subdivided, but this is always unequivocal as at most one additional point per face is required. With these conventions, the proposed tetrahedral subdivisions of canonical ambiguous cases are recalled in Table 2, see [14] for a detailed discussion.

2 Implementation

Consider a given input tetrahedron, σ, along with a given set of edges, $E \subset \{01, 12, 02, 03, 13, 23\}$, to be subdivided. E may take on any of 64 possible edge-patterns that partially specify how its faces must be subdivided during refinement. For faces with exactly two edges to be split, the relationships between edge lengths is also required to fully specify the subdivision of any face. Thus the number of cases is increased dramatically from 64 to 634 when the edge length relationships are considered. We will show how this is computed below, but first consider the following.

Remark 2.1 Case 1 occurs when E is a singleton. No edge length relationships are necessary to fully specify the face subdivisions.

Example 2.1 Case 3a occurs when $E = \{01, 02, 03\}$ or $E = \{01, 12, 13\}$ or $E = \{12, 02, 23\}$ or $E = \{03, 13, 23\}$. But for each choice of E, edge length comparisons yield 13 possible face subdivisions. For instance, if $E=\{01, 02, 03\}$, the lengths of edges 01 and 02, 01 and 03, and 02 and 03 must be compared. Each comparison yields three possible results: $<$, $>$, or $=$. The possible combinations are $\underline{0} > \underline{2} > \underline{3}$, $\underline{2} > \underline{3} > \underline{0}$, $\underline{3} > \underline{0} > \underline{2}$, $\underline{0} > \underline{3} > \underline{2}$, $\underline{2} > \underline{0} > \underline{3}$, $\underline{3} > \underline{2} > \underline{0}$, $\underline{0} = \underline{2} > \underline{3}$, $\underline{3} > \underline{0} = \underline{2}$, $\underline{0} > \underline{2} = \underline{3}$, $\underline{2} = \underline{3} > \underline{0}$, $\underline{0} = \underline{3} > \underline{2}$, $\underline{2} > \underline{0} = \underline{3}$, and $\underline{0} = \underline{2} = \underline{3}$.

Note that each of the ten Ruprecht–Müller cases is associated with one or more values of E. Example 2.1 shows that there are four values of E for case 3a. For any case x, the corresponding values of E may be grouped into a set that we will call $\tilde{E}_x$, the *edge splits* of x. Every entry of $\tilde{E}_x$ may have any set of relevant edge length relationships, so the total number of configurations that must be handled for case x is the product of the cardinality of $\tilde{E}_x$ and the number N_x of relevant edge length relationships for case x. The number of relevant edge length relationships, N_x, may be divided into geometrically unambiguous relationships, N_x^u, and geometrically ambiguous relationships, N_x^a. For the example above, $N_{3a} = 13 = N_{3a}^u + N_{3a}^a = 6 + 7$. Table 3 shows the total number of edge subdivision and relative edge length relationship combinations for all cases.

In this section, we discuss how we practically dealt with this combinatorial complexity, and how we implemented the method. We then illustrate the efficiency of our approach by considering how it performs in terms of speed and mesh quality with three different meshes and their subdivisions governed by two different refinement criteria. After having acknowledged how experimental results confirm that handling unambiguous cases was indeed a necessity, and not only a topic for academic contemplation, we discuss how the method might be made even better.

In Table 3, note that $|\tilde{E}_x|$ is always some divisor of 24. This is because the edge splits of any case x form a subgroup of $\mathfrak{S}_4$. The subdivisions indicated in Table 2 being for canonical cases only, this implies that the ini-

Table 2 Subdivisions of canonical ambiguous cases [14]

Case	Geometry	Tetrahedral subdivision
$2a^a$	$\underline{\mathbf{0}} = \underline{1}$	04a3 0a23 4153 45a3 a523
$3a^a\alpha$	$\underline{0} = \underline{2} > \underline{3}$	0467 4367 a123 a263 a643 a413
$3a^a\beta$	$\underline{0} = \underline{2} < \underline{3}$	0467 1327 a127 a267 a647 a417
$3a^a\gamma$	$\underline{0} = \underline{2} = \underline{3}$	0467 26ad 37db 41ab b6a4 b6da b67d b647 2abd 1ab2 2b3d 321b
$3c^a\alpha$	$\underline{0} = \underline{1} > \underline{3}$	4153 a047 a207 a743 a273 a523 a453
$3c^a\beta$	$\underline{0} = \underline{1} < \underline{3}$	7153 7523 a047 a207 a527 a457 1547
$3c^a\gamma$	$\underline{0} = \underline{1} = \underline{3}$	415b b153 a047 a207 a523 a273 a74b a7b3 a45b ab53
$4a^a\alpha$	$\underline{3} = \underline{4} > \underline{5}$	7893 670b 601b 6978 67b8 6b18 1268 2689
$4a^a\beta$	$\underline{3} = \underline{4} < \underline{5}$	7893 670b 601b 6978 67b8 6b18 1269 1689
$4a^a\gamma$	$\underline{3} = \underline{4} = \underline{5}$	7893 670b 601b 6978 67b8 6b18 612c 629c 698c 681c
$4b^a\alpha$	$\underline{1} = \underline{2} < \underline{4} < \underline{3}$	7823 a607 a158 a017 a718 67a8 6a58 6278 6528
$4b^a\beta$	$\underline{1} = \underline{2} > \underline{4} > \underline{3}$	6523 a607 a158 a018 a708 67a8 6a58 6378 6538
$4b^a\gamma$	$\underline{3} < \underline{1} = \underline{2} < \underline{4}$	6238 a607 a158 a018 a708 67a8 6a58 6378 6528
$4b^a\delta$	$\underline{1} = \underline{2} < \underline{3} = \underline{4}$	7823 a607 a158 a01b ab18 a0b7 a7b8 67a8 6a58 6278 6528
$4b^a\varepsilon$	$\underline{1} = \underline{2} = \underline{3} < \underline{4}$	a607 a158 a018 a708 d625 d378 d238 d285 67a8 6a58 6d78 65d8
$4b^a\zeta$	$\underline{1} = \underline{2} = \underline{3} > \underline{4}$	a607 a158 a018 a708 d625 d378 d235 d385 67a8 6a58 6d78 65d8
$4b^a\eta$	$\underline{1} = \underline{2} = \underline{3} = \underline{4}$	a607 a158 a01b ab18 a0b7 a7b8 d625 d378 d23c d2c5 dc38 d5c8 67a8 6a58 65d8 6d78
$5^a\alpha$	$\underline{1} = \underline{2}, \underline{3} > \underline{4}$	6529 7893 a607 a158 a017 a718 a859 a679
$5^a\beta$	$\underline{1} = \underline{2}, \underline{3} = \underline{4}$	6529 7893 a607 a158 a01b ab18 a0b7 a7b8 a859 a679

Table 3 The total number of edge subdivision and relative edge length relationship combinations

Case	$\|\tilde{E}_x\|$	$N_x^u + N_x^a = N_x$	Total
0	1	1 + 0 = 1	1
1	6	1 + 0 = 1	6
2a	12	2 + 1 = 3	36
2b	3	1 + 0 = 1	3
3a	4	6 + 7 = 13	52
3b	4	1 + 0 = 1	4
3c	12	8 + 10 = 18	216
4a	12	4 + 5 = 9	108
4b	3	14 + 37 = 51	153
5	6	4 + 5 = 9	54
6	1	1 + 0 = 1	1
Sum	64	108	634

tial tetrahedron σ must be permuted into one of the canonical configurations, prior to subdivision. Then, after the proper subdivision template has been applied, the inverse of this permutation must be performed to transform the canonical tetrahedral subdivision into a subdivision of the given σ. Formally, (σ, E) belongs to the set of all possible tetrahedra and edge splits, $T \times \tilde{E}$, and it is transformed into an element of $T_0 \times \tilde{E}_0$, the set of all canonical configurations, as illustrated by the following diagram:

$$\begin{tikzcd}
T \times \tilde{E} \arrow[r, "\mathtt{D}"] \arrow[d, "s^{-1}"'] & T^{\mathbb{N}} \\
T_0 \times \tilde{E}_0 \arrow[r, "\mathtt{D}_{|T_0 \times \tilde{E}_0}"] & T^{\mathbb{N}} \arrow[u, "s"']
\end{tikzcd}$$

$\mathtt{D}$ maps an input tetrahedron and list of edge splits to a set of output tetrahedra $\{\sigma_1, \ldots, \sigma_n\} \in T^{\mathbb{N}}$. As detailed in Table 3, $\mathtt{D}$ must handle 634 configurations, each of which has multiple output tetrahedra. Assuming that each configuration has five to six tetrahedra[2], this table requires 15 to 17 kB of storage—23 to 27% of an Opteron's 64 kB L1 cache. Given the number of floating point values interpolated at each vertex (a minimum of six when no scalar fields are present), it is unrealistic to expect the table to stay in the L1 cache while the tests required to identify a configuration are performed. On the other hand, [15] and Table 2 provide the hash tables of a restriction of $\mathtt{D}$ to a subset of canonical cases T_0, as illustrated in Sect. 2.1. This restricted map $\mathtt{D}_{|T_0 \times \tilde{E}_0}$ requires much less memory. However, this will require the use of permutations in order to close the commutative diagram.

In Sect. 2.1, we explain how decomposing s into two permutations (both from the same 1.3 kB lookup table) reduces the table containing output tetrahedra to 50 entries (about 1.3 kB) for a total of 2.6 kB. That means we can free up approximately 20% of the L1 cache by using a smaller table and permutations. So, we are taking the longer path in Sect. 2.1 that we may obtain shorter code and execution times.

2.1 Take the long cut

The longer path in 2.1 requires a bijective map s, such that s^{-1} transforms any arbitrary (σ,E) into the canonical configuration to which it pertains. In fact, s is a re-indexing (i.e., a permutation) of the tetrahedron vertices and edges and, as such, it is bijective. Therefore, an intuitive way to understand the process is to use a graph of (σ,E) that is dual of the usual topological representation, where the nodes of the graph are tetrahedral vertices and the edges of the graph correspond to the split edges of the tetrahedron, as illustrated for a $(\sigma,\{(01),(02),(03)\})$ configuration in Fig. 6 for case 3a. Omitting nonsplit edges does not create any ambiguity, as each vertex of a tetrahedron is connected to each of the three other vertices by either a split (graph edge) or nonsplit (not shown on the graph) edge.

By renumbering the nodes of the graph, we get a class of entries that are congruent—they may be interchanged with one another without altering the structure of the edge colorings. This is illustrated in Fig. 7 for case 3a; the maps denoted s_0, s_1, s_2 and s_3 are indeed index permutations, and we examine later how we can take advantage of a particular group of permutations, namely $\mathfrak{S}_4$. In the general case, we denote Θ the permutation that maps a canonical topological case into the desired topological configuration. By definition, Θ relies solely on the edge splits E independent of the geometry of σ: the edge length criterion is not invoked at this stage.

Once a canonical topological configuration $\Theta^{-1}(E)$ has been identified, the longest edge criterion must then be applied to $(s^{-1}(\sigma), \Theta^{-1}(E))$ in order to decide the triangulation of faces that have exactly two split edges[3]. After edge length comparisons, each topological configuration results in several geometric configurations; more precisely the number of such geometric configurations for each canonical topological case is given in the N_x column of Table 3, with a total of 108 of them. Fortunately, some case regrouping can be done thanks to index permutation, resulting in a much smaller number of canonical geometric configurations, 39, that is the sum of all cases listed in Tables 1 and 2, along with case 0.

Example 2.2 As explained in Example 2.1, case 3a has six unambiguous configurations and given seven ambiguous configurations. However, all unambiguous configurations can be obtained by re-indexing the vertices of a single canonical geometric configuration 3a^u, as illustrated with the weighted graph representation in Fig. 8, thus resulting in a geometric equivalence class. On the contrary, the ambiguous configurations of case

[2]The average in our implementation is 5.6 tetrahedra per configuration.

[3]Cases 0, 1, 2b, 3b and 6 are exempt from the longest-edge criterion because they have no ambiguous faces.

3a cannot be retrieved from configuration 3a^u, nor can they all be retrieved from a unique ambiguous configuration, but pertain to three distinct geometric equivalence classes, 3a$^a\alpha$, 3a$^a\beta$, and 3a$^a\gamma$ (cf. [14]).

As with our previous map, Θ, that created topological equivalence classes, we now define the permutation Γ that maps a canonical geometrical case (i.e., that is contained in either Tables 1 or 2) into the considered geometric configuration. Combining these two maps, we can then rewrite the subdivision map D using the long cut, as follows:

$$\mathrm{D} = \Theta \circ \Gamma \circ \mathrm{D}_{|T_0 \times \tilde{E}_0} \circ \Gamma^{-1} \circ \Theta^{-1}. \qquad (2)$$

A summary of this process—the basic flow of the adaptive tessellation algorithm—is provided in Algorithm 1.

Algorithm 1 Streaming Tetrahedral Refinement

```
1:  An input tetrahedron, $\sigma = \{v_0, v_1, v_2, v_3\} \in T$, and a
    recursion depth $n \in \mathbb{N}$ are supplied by the user
2:  if $n > 0$ then
3:    apply the edge subdivision algorithm to each edge of
      $\sigma$, resulting in a set of split edges $E$
4:    if $E = \varnothing$ then
5:      exit
6:    else
7:      Apply $\Theta^{-1}$ to $(\sigma, E)$, in order to permute it into one
        of the topological canonical cases (note that this also
        permutes, formally, edge midpoints and face points)
8:      For each face of $(\Theta^{-1}(\sigma), \Theta^{-1}(E))$ with geometric
        ambiguity, compute the placement of the face point
        (this depends on what pair of edges are split)
9:      Apply $\Gamma^{-1}$ to $(\Theta^{-1}(\sigma), \Theta^{-1}(E))$, in order to per-
        mute it into one of the geometric canonical cases (in
        practice, this step is skipped for cases 1, 2b, 3b and 6,
        as those do not need the the longest edge criterion)
10:     Apply $\mathsf{D}_{|T_0 \times \tilde{E}_0}$ using Tab. 1 and Tab. 2, resulting
        in a set of $p \in \mathbb{N}^*$ output tetrahedra, $\mathcal{O}$, whose
        vertices are amongst vertices, edge midpoints, and
        additional face points of $\Gamma^{-1} \circ \Theta^{-1}(\sigma)$.
11:     Apply $\Theta \circ \Gamma$ to $\mathcal{O}$ to obtain the tetrahedral refine-
        ment $\cup_{i=1}^{p} \{v_0^i, v_1^i, v_2^i, v_3^i\}$ of $\sigma$.
12:     for $i = 1$ to $p$ do
13:       compute Streaming Tetrahedral Refinement of
          $\{v_0^i, v_1^i, v_2^i, v_3^i\}$ with recursion depth $n - 1$
14:     end for
15:   end if
16: end if
```

Even with this decomposition of D, Tables 1 and 2 still yield 273 tetrahedra distributed across 38 cases[4] to be handled by $\mathrm{D}_{|T_0 \times \tilde{E}_0}$. Rather than hand-write the C++ code that selects the proper set of tetrahedra and permutations to apply, we have created a Python (cf. [18]) script that assembles an array of tetrahedra and permutations and then writes C++ code to apply the proper maps as table lookups.

[4]our slightly different implementation, as will be explained further, actually regroups the tetrahedra in 50 sets.

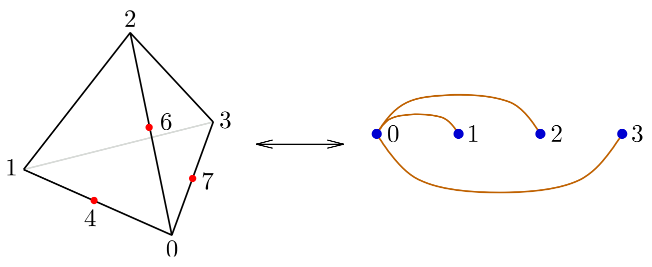


Fig. 6 A tetrahedron (*left*) may be thought of as a graph with four nodes (*right*). Graph edges correspond to split edges of the tetrahedron; the nonsplit tetrahedron edges do not appear on the graph. The example shown here is case 3a

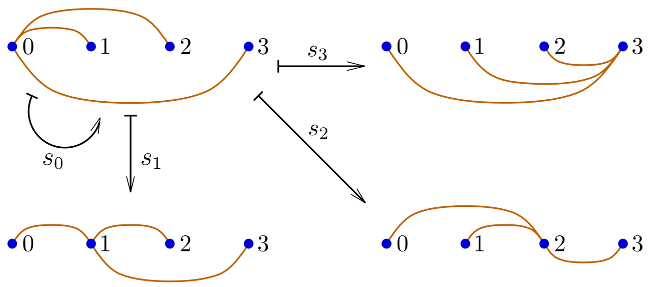


Fig. 7 These edge configurations are topologically equivalent: they form the 3a class—canonically represented by the upper left configuration

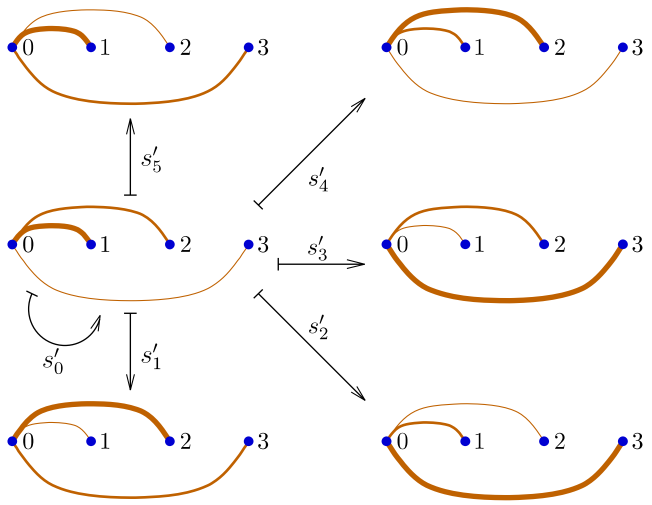


Fig. 8 When considering edge length relationships as needed (here shorter tetrahedral edges are represented with thicker graph edges), a unique topological configuration can have several geometric configurations. All possible unambiguous geometric configurations of case 3a are shown, and they are all identical, up to a permutation, to our chosen canonical graph $\underline{0} > \underline{2} > \underline{3}$

2.2 A reminder on symmetric groups

For each geometric equivalence class, we must choose a canonical representation (i.e., a class representative) and define how to map an arbitrary entry to that representation. In fact, these maps can be viewed as elements of the symmetric group $\mathfrak{S}_4$ which we now briefly recall. The interested reader will find an extensive introduction to symmetric groups for example in [6].

Definition 2.1 *Let S be a set. A bijection from S onto itself is called a* permutation of *S. Equipped with the composition operation (also called* product *in this context), the set of all permutations of S forms a group, called the* symmetric group of *S, and denoted as* $\mathfrak{S}(S)$. *For all* $n \in \mathbb{N}^*$, *the symmetric group of* $\{1,...,n\}$ *is denoted* $\mathfrak{S}_n$.

Remark 2.2 It is clear that the symmetric group of any ordered finite set S with $n \in \mathbb{N}^*$ elements is isomorphic to $\mathfrak{S}_n$, thanks to the canonical increasing bijection between E and $\{1,...,n\}$. 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.

Henceforth, S will denote a finite set with cardinality $n \in \mathbb{N}^*$.

Definition 2.2 *Let* $s \in \mathfrak{S}(S)$. *For any arbitrary* $x \in E$, *the* s-orbit of x *is a subset of S obtained by applying permutation s to element x any number of times, more formally denoted:*

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

An s-orbit *is a part of S that is the* s-orbit *of at least one x* $\in S$. *s is said to be a* cycle *if there is a unique* s-orbit $s{\cdot}x$ *with nonzero cardinality; in this case, denoting p the cardinal number of the orbit, s is said to be a* p-cycle *with* support $s{\cdot}x$. *A* transposition *is a two-cycle. A p-cycle s with support* $\{a_1,...,a_p\}$, *where* $a_i = s^{i-1}(a_1)$ *will be denoted* $(a_1 \ldots a_p)$. *Finally, the* $\mathfrak{S}_4$ – orbit *of x is defined as follows:*

$$\mathfrak{S}_4 \cdot x = \cup_{s \in \mathfrak{S}_4} s \cdot x. \tag{3}$$

Remark 2.3 The p-cycle notation is not unique: for example, a transposition $(a_1\ a_2)$ can also be written $(a_2\ a_1)$.

The following example introduces the most important symmetric group for us, since we are going to make an extensive use of $\mathfrak{S}_4$ to generate the code that handles all 634 cases outlined in Table 3.

Example 2.3 $\mathfrak{S}_4$ 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 three-cycle, and there are $4{\times}2 = 8$ three-cycles. In addition, $\mathfrak{S}_4$ contains three permutations formed of two transpositions with disjoint support, namely (01)(23), (02)(13) and (03)(12); those, along with the identity ( ) form the Klein *Vierergruppe*, the smallest finite group with element orders all smaller than the group's cardinal. The remaining $4! - 18 = 6$ elements are the four-cycles, such as (0312).

Another useful feature of symmetric groups is the signature of a map, which we will employ to preserve the orientation of tetrahedra as they are processed:

Definition 2.3 *Let* $s \in \mathfrak{S}(S)$ *and* $m(s)$ *the number of distinct s-orbits. The* signature *of s is then defined as* $\varepsilon(s) = (-1)^{n - m(s)}$.

Example 2.4 There are three distinct (01)-orbits in {0,1,2,3}: {0,1}, {2} and {3}, while there is a single (0312)-orbit: {0,1,2,3} itself. Therefore, $\varepsilon((01)) = (-1)^{4-3} = -1$, and $\varepsilon((0312)) = (-1)^{4-1} = -1$. This implies that neither (01) nor (0312) are orientation-preserving. More generally, the only orientation-preserving permutations in $\mathfrak{S}_4$ are either three-cycles or members of the *Vierergruppe* (i.e., 12 permutations among 24).

We can then reformulate the ideas introduced intuitively in Sect. 2.1 within this rigorous and powerful framework.

2.3 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 underlying 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. Therefore, we must extend vertex permutations to midpoints and faces, as illustrated in the following example:

Example 2.5 The transposition (01) switches 0 and 1, leaving the other corner vertices unchanged: for instance, the doubles (1,2) and (0,2) are switched. In order to preserve midpoint consistency, vertices 5 and 6 must therefore be switched, too. In fact, the generalized (01) is the following map:

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

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 $\mathfrak{S}_4$; in fact, these generalized permutations are elements of $\mathfrak{S}_{14}$, but the converse is not true as the image of any element in $\{4,5,6,7,8,9,a,b,c,d\}$ has to be consistent with the images of the corresponding edge or face vertices. Indeed, these generalized permutations are obtained by unambiguous injection of $\mathfrak{S}_4$ into $\mathfrak{S}_{14}$, and as this is unambiguous, we will use the same notation for a permutation in $\mathfrak{S}_4$ and its counterpart in $\mathfrak{S}_{14}$. Fully describing each of these 24 generalized permutations is both easy and lengthy, and is therefore left to the reader as an exercise. In everything that follows, all permutations will be implicitly meant in the "generalized sense"

Table 4 Case 3a: permutations from the canonical representation to all possible configurations

Fig. 7 map	Possible choices in $\mathfrak{S}_4$
s_0	() (12) (13) (23) (123) (132)
s_1	(01) (012) (013) (0123) (0132) (01)(23)
s_2	(02) (021) (023) (0213) (0231) (02)(13)
s_3	(03) (031) (032) (0312) (0321) (03)(12)

unless otherwise mentioned. We now examine in details how Algorithm 2.1 can be nicely formulated using this framework, that will then provide an excellent basis for implementation.

Θ: topology-based permutations. The first step consists in expressing the map Θ introduced in Sect. 2.1 in terms of permutations. For instance, one can easily check that the maps s_0, s_1, s_2 and s_3 in Fig. 7 are elements of $\mathfrak{S}_4$ that can be freely chosen as indicated in Table 4.

Remark 2.4 One shall notice that, *primo*, as indicated in Table 3 there are four elements in the (topological) equivalence class 3a; *secundo*, as shown in Table 4, there are six possible choices for s_0, i.e., six permutations that leave the class representative (topologically) unchanged; *tertio*, the cardinality of $\mathfrak{S}_4$ is 24 and we have $6{\times}4=24$. In fact, this is not a coincidence, but directly results from a well-known property about symmetric groups[5], according to which

$$(\forall E \in \tilde{E}) \quad |\mathfrak{S}_4 \cdot E| \times |\mathfrak{S}_{4E}| = |\mathfrak{S}_4|, \tag{4}$$

where $\mathfrak{S}_{4E}$ denotes the *stabilizer of* E in $\mathfrak{S}_4$, i.e., the set (indeed, a subgroup of $\mathfrak{S}_4$) of all permutations in $\mathfrak{S}_4$ that leave E unchanged. Therefore, counting the permutations for which a given canonical configuration is left topologically unchanged immediately yields the number of subcases that pertain to it. This property permits us to easily devise Table 5 and then verify it.

This choice of a permutation for each element of $\tilde{E}$ has to be made as well, and thanks to the relatively small cardinality—64—of $\tilde{E}$, this problem can be exhausted by enumeration. A convenient notation to represent any arbitrary split edge configuration E in to use the corresponding *bitcode*, where the bit of a split (resp. nonsplit) is 1 (resp. 0). This notation is indeed consistent (bijective); e.g., the bitcode of canonical case 3a, as shown in Fig. 6, is 101100. With these conventions, the map Θ can be fully specified, by arbitrarily picking one permutation that performs the desired topological transformation. The particular choices we made are indicated in Table 5. The implementation of Θ is simply a 64-entry lookup table that also provides direct access to Θ^{-1}. It is important to note that all permutations indicated in Table 5 have positive signature, and are therefore orientation-preserving, with the exception of four transpositions ((01), (12), (02) and (13)) and two 4-cycles ((0132) and (0213)) used to retrieve six particular configurations in the 3c equivalence class. In fact, it is easy to check that none of these six cases can be obtained from the canonical case with an even permutation. Therefore, we have modified Ruprecht and Müller's classification by introducing the case 3d, with class representative bitcode 101010, to avoid using any orientation-changing permutation in Θ; with this new case, we retrieve the five other configurations as indicated in Table 6.

Table 5 The map Θ

Case	Bitcode	Permutations and permuted bitcodes		
0	000000			
1	100000	$\xrightarrow{(012)}$ 010000	$\xrightarrow{(021)}$ 001000	$\xrightarrow{(03)(12)}$ 000001
		$\xrightarrow{(013)}$ 000010	$\xrightarrow{(031)}$ 000100	
2a	110000	$\xrightarrow{(021)}$ 101000	$\xrightarrow{(012)}$ 011000	$\xrightarrow{(01)(23)}$ 100100
		$\xrightarrow{(031)}$ 001100	$\xrightarrow{(032)}$ 100010	$\xrightarrow{(03)(12)}$ 010001
		$\xrightarrow{(132)}$ 000110	$\xrightarrow{(023)}$ 010010	$\xrightarrow{(02)(13)}$ 000101
		$\xrightarrow{(123)}$ 001001	$\xrightarrow{(013)}$ 000011	
2b	100001	$\xrightarrow{(012)}$ 010100	$\xrightarrow{(021)}$ 001010	
3a	101100	$\xrightarrow{(012)}$ 110010	$\xrightarrow{(031)}$ 000111	$\xrightarrow{(021)}$ 011001
3b	100110	$\xrightarrow{(123)}$ 111000	$\xrightarrow{(012)}$ 010011	$\xrightarrow{(03)(12)}$ 001101
3c	110100	$\xrightarrow{(012)}$ 011010	$\xrightarrow{(01)}$ 101010	$\xrightarrow{(03)(12)}$ 010101
		$\xrightarrow{(021)}$ 101001	$\xrightarrow{(12)}$ 011100	$\xrightarrow{(0132)}$ 010110
		$\xrightarrow{(031)}$ 001110	$\xrightarrow{(02)}$ 110001	$\xrightarrow{(0213)}$ 001011
		$\xrightarrow{(032)}$ 100011	$\xrightarrow{(13)}$ 100101	
4a	001111	$\xrightarrow{(013)}$ 111100	$\xrightarrow{(123)}$ 110110	$\xrightarrow{(02)(13)}$ 111010
		$\xrightarrow{(021)}$ 010111	$\xrightarrow{(012)}$ 100111	$\xrightarrow{(03)(12)}$ 101110
		$\xrightarrow{(132)}$ 111001	$\xrightarrow{(023)}$ 101101	$\xrightarrow{(01)(23)}$ 011011
		$\xrightarrow{(031)}$ 110011	$\xrightarrow{(032)}$ 011101	
4b	011110	$\xrightarrow{(021)}$ 110101	$\xrightarrow{(012)}$ 101011	
5	011111	$\xrightarrow{(013)}$ 111101	$\xrightarrow{(031)}$ 111011	$\xrightarrow{(03)(12)}$ 111110
		$\xrightarrow{(021)}$ 110111	$\xrightarrow{(012)}$ 101111	
6	111111			

Γ: geometry-based permutations The same method is now applied to specify the map Γ in the permutation framework; e.g., the maps s'_0, s'_1, s'_2, s'_3, s'_4 and s'_5 in Fig. 8 are, respectively, ( ), (12), (13), (23), (123) and (132), and they are the only possible choices.

Remark 2.5 At this point, the attentive reader will likely have noticed that these s'_0, s'_1, s'_2, s'_3, s'_4 and s'_5 are exactly the possible choices for s_0 in Fig. 7, as indicated in Table 7. This is not a matter of coincidence: indeed, (12), (13), (23), (123) and (132) leave canonical configuration 3a *topologically unchanged*, but *geometrically changed*. More generally, Γ is topology-preserving

Table 6 The map Θ for case 3d

Case	Bitcode	Permutations and permuted bitcodes		
3d	101010	$\xrightarrow{(021)}$ 011100	$\xrightarrow{(032)}$ 010110	$\xrightarrow{(03)(12)}$ 001011
		$\xrightarrow{(012)}$ 110001	$\xrightarrow{(031)}$ 100101	

[5] and, more generally, about groups acting over sets.

but modifies geometry (except for ( ) instances, when the target geometry is that of the canonical case itself).

For the sake of brevity, we do not specify Γ for all cases here, as this can be readily done by simple imitation, *ab uno disce omnes*, of the Γ specification for case 3a. In addition, this is very similar to what has been extensively described for Θ. The interested reader can find the full specification of Γ in our C++ code generator written in `Python` called `vtkStreamingTessellator.py` that is available as part of `ParaView` (cf. [5]). Just note that, generally, not all geometric configurations can be retrieved from a single configuration: unambiguous case 3a is an exception, as one can see in Tables 1 and 2.

2.4 Wrap-up example

We finally illustrate the somewhat abstract description of the whole streaming refinement process as summarized in Sect. 2.2 by applying it to a particular case. In order to make it as explanatory as possible, we choose one of the most complex configurations: $4b^a\alpha$. For this case, all geometrical configurations that can be deduced from the canonical representation of case $4b^a\alpha$ are detailed in Table 7.

Example 2.6 Consider two tetrahedra σ_1 and σ_2, with the same set of split edges E, written 011110 in bitcode notation. In addition, suppose that the geometries of σ_1 and σ_2 are such that $\underline{2} = \underline{1} < \underline{3} < \underline{4}$ and $\underline{2} = \underline{3} < \underline{1} < \underline{4}$, respectively. We now examine how σ_1 and σ_2 are streamed through Algorithm 2.1.

1. As indicated in Table 5, E is the class representative for case 4b, and therefore $\Theta = \Theta^{-1} = (\,)$.
2. According to Tables 1 and 2, neither σ_1 nor σ_2 are in canonical geometrical configuration. Nonetheless, they both pertain to the ambiguous 4*b* case that arises when exactly two edges among the images of 12, 02, 13 and 03 by Θ have equal lengths while being shorter than the two other ones. The geometric equivalence class for all such configurations is represented by canonical case $4b^a\alpha$, with $\underline{2} = \underline{1} < \underline{4} < \underline{3}$. As indicated in Table7 and illustrated in Fig. 9, the respective values of Γ are (01) and (0312).

Table 7 Case $4b^a\alpha$: permutations from the canonical representation to all possible geometric configurations

$s \in \mathfrak{S}_4$	Configuration	$\varepsilon\,(s)$
()	$\underline{2} = \underline{1} < \underline{4} < \underline{3}$	1
(01)	$\underline{2} = \underline{1} < \underline{3} < \underline{4}$	−1
(23)	$\underline{3} = \underline{4} < \underline{1} < \underline{2}$	−1
(01)(23)	$\underline{3} = \underline{4} < \underline{2} < \underline{1}$	1
(02)(13)	$\underline{2} = \underline{3} < \underline{4} < \underline{1}$	1
(03)(12)	$\underline{1} = \underline{4} < \underline{2} < \underline{3}$	1
(0213)	$\underline{1} = \underline{4} < \underline{3} < \underline{2}$	−1
(0312)	$\underline{2} = \underline{3} < \underline{1} < \underline{4}$	−1

3. Having found Θ and Γ, and thus Θ^{-1} and Γ^{-1}, from the lookup tables, one knows that in both cases (without having to actually compute it), the image of $\Gamma^{-1} \circ \Theta^{-1}$ is a tetrahedron in canonical configuration $4b^a\alpha$. Therefore, this tetrahedron belongs to $T_0 \times \tilde{E}_0$ and its image by $\mathrm{D}_{|T_0 \times \tilde{E}_0}$ is known explicitly from Table 2:

$$7823, a607, a158, a017, a718, 67a8, 6a58, 6278, 6528. \tag{5}$$

4. Applying $\Theta \circ \Gamma$ to (5) in the case of σ_1 (thus with $\Gamma = (01)$) yields the following subdivision:

$$8723, a518, a067, a108, a807, 58a7, 5a67, 5287, 5627, \tag{6}$$

when for σ_2 (with $\Gamma = (0312)$), one obtains

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

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. Actually, in our implementation of the scheme, an additional function is embedded in the process. In the example above, the two particular values of Γ are not orientation-preserving, as $\varepsilon\,((01)) = \varepsilon\,((0312)) = -1$. More generally, If the permutation $s \in \mathfrak{S}_4$ used to 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 $\varepsilon\,(s) = -1$, a final reorientation step must be performed. This reorientation operator $\mathcal{R}$ 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 $\mathcal{R}$ as a symmetry across any arbitrary plane, followed by a rotation and a transposition to retrieve the initial vertex coordinates. Accordingly, this slightly refined version of the subdivision process can be summarized as:

$$\mathrm{D} = \mathcal{R} \circ \Theta \circ \Gamma \circ \mathrm{D}_{|T_0 \times \tilde{E}_0} \circ \Gamma^{-1} \circ \Theta^{-1}. \tag{8}$$

Example 2.7 As shown in Fig. 9, after the reorientation operator $\mathcal{R}$ has been applied to (6) and (7), the positively-oriented configurations respectively read

$$7823, 5a18, 0a67, 1a08, 8a07, 85a7, a567, 2587, 6527 \tag{9}$$

and

$$5801, 7d38, 2d65, 3d28, 8d25, 87d5, d765, 0785, 6705. \tag{10}$$

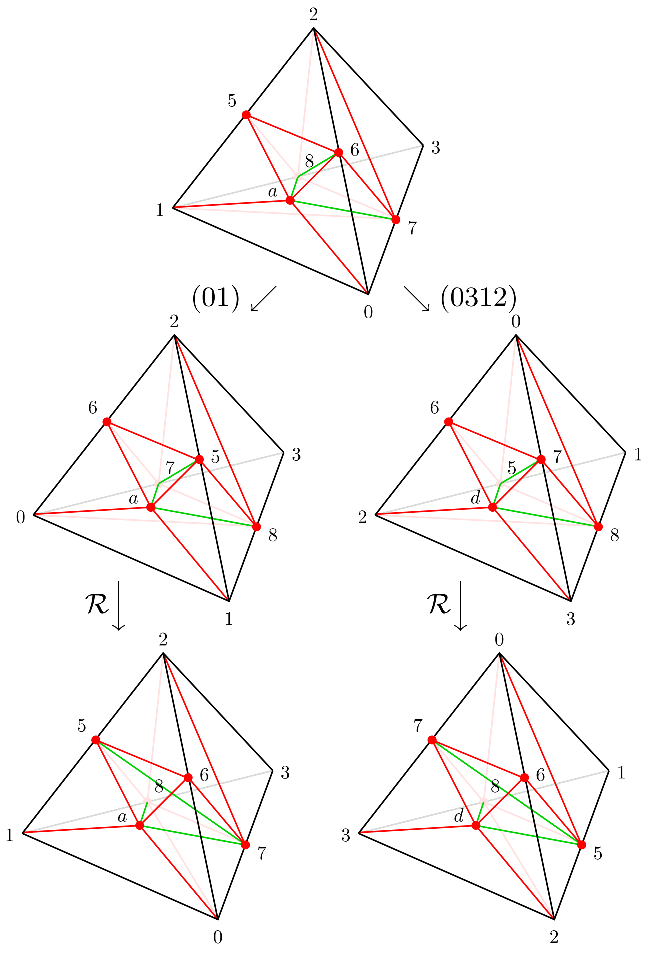


Fig. 9 *Above*: canonical configuration of ambiguous case $4b^a\alpha$; *center*: images by (01) (left) and (0312) (*right*); *below*: re-orientations to obtain configurations $\underline{2} = \underline{1} < \underline{3} < \underline{4}$ (*left*) and $\underline{2} = \underline{3} < \underline{1} < \underline{4}$ (*right*)

Remark 2.6 One can observe that the permutations listed in Table 7, 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 a given canonical case 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 cardinality 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 $4b^a\alpha$). This property is used to verify the code.

3 Results

The results are presented as follows: first, we present two different refinement strategies, that rely on analytical edge size specifications. Then, successive steps of streaming mesh refinement governed by these subdivision criteria are performed over a set of sample tetrahedral meshes. The intrinsically 4D objects that result from this process are then illustrated by the means of the 2D format; quantitative information such as the number and quality of output tetrahedra is also presented. Furthermore, we analyze the effects of the scheme on mesh quality, by examining how the statistics of two widely accepted tetrahedral quality measures evolve throughout the process. Finally, we assess the execution speed of our implementation: first, single processor, and then on multiple processors, for both ideally balanced and naively balanced computational loads.

The edge subdivision criterion that will be used to refine that mesh relies on the principle of an edge size map specification [4]. Since the goal is to demonstrate the efficiency and versatility of our technique, we use arbitrary analytical functions; for practical applications, other criteria would obviously be used, in particular based on error metrics. The edge refinement criterion used here can be simply stated as follows: given a trivariate real function f and a given constant $k \in \mathbb{R}^+$, an edge with length e and midpoint with coordinates (x_m, y_m, z_m) is refined if and only if

$$[f(x_m, y_m, z_m)]^2 < ke^2. \tag{11}$$

Remark 3.1 The constant k is to be set, depending on the desired level of refinement progressivity. Note that (11) is expressed in a squared form, because it is computationally easier to compute e^2 than e.

3.1 Distance-to-a-plane-based refinement

We start with f in (11) being the distance to the plane with unit normal $\frac{1}{\sqrt{3}}(1, 1, 1)$ passing through the center of the mesh bounding box with coordinates (x_0,y_0,z_0). Setting $k = 2$, (11) thus becomes:

$$(x_m + y_m + z_m - x_0 - y_0 - z_0)^2 < 6e^2. \tag{12}$$

This edge refinement criterion is to be applied to two meshes: *primo*, to a cuboid: the initial mesh $\mathcal{T}^0_{\text{box}}$ has 553 points and 1,627 tetrahedra, and is displayed in Fig. 10;

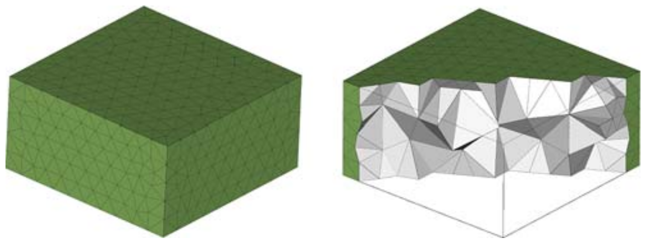


Fig. 10 Initial mesh $\mathcal{T}^0_{\text{box}}$ of a cuboid: boundary (*left*) and clip across the mesh to show interior detail (*right*)

secundo, to a more organic mesh $\mathcal{T}^0_{\text{tri}}$ with 4,209 points and 16,910 tetrahedra, shown in Fig. 12, left.

Four successive refinement steps are performed from the initial meshes $\mathcal{T}^0_x$, leading to the refined meshes $\mathcal{T}^1_x$, $\mathcal{T}^2_x$, $\mathcal{T}^3_x$, and $\mathcal{T}^4_x$. Columns n_v and n_t in Table 8, left, indicate the numbers of points and tetrahedra of these meshes, and cut snapshots are shown in Figs. 11 and 12.

3.2 It gets curvy

We now refine in a somehow more aesthetic way the mesh of a mechanical part (cf. Fig. 13, left), with 28,694 points and 150,779 tetrahedra. An interesting family of smooth planar curves is that of *Lissajous curves*, see e.g. [8]. One of the simplest such curves in the xy plane can be parametrized as follows:

$$\begin{cases} x(t) = A\cos t \\ y(t) = B\sin(2t), \end{cases} \tag{13}$$

where A and B are two real positive constants, and $t \in [0, 2\pi[$. An agreeable generalized cylinder can then be obtained by using this curve as directrix, and the lines perpendicular to the xy plane as director curves. In all that follows, this generalized cylinder will be denoted $\mathcal{L}_{A,B}$; a section of $\mathcal{L}_{1,1}$, for $0 \le z \le 2$, is displayed by Fig. 14. An implicit equation of $\mathcal{L}_{A,B}$ is (cf. [17] for details):

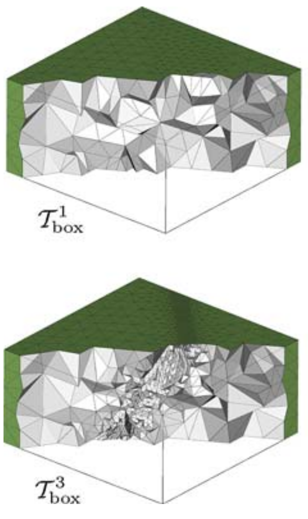
$$f(x,y,z) = \frac{4x^4}{A^4} - \frac{4x^2}{A^2} + \frac{y^2}{B^2} = 0$$

and thus this definition of f in $\mathbb{R}^3$ is substituted in (11). 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 (11). Therefore, the mesh refinement process should “track” $\mathcal{L}_{A,B}$, and the tracking should become tighter as the recursive refinement level increases. Finally, we set A and B so that a truncation of $\mathcal{L}_{A,B}$ complete is contained in $\mathcal{T}^0_{\text{part}}$ (in practice, $A = 1/30$ and $B = 1/50$ are used).

After three subdivision steps governed by that criterion, the expected refinement pattern appears clearly, as

Table 8 Numbers of points (n_p) and tetrahedra (n_t) of meshes $\mathcal{T}^0_{\text{box}}, \mathcal{T}^1_{\text{box}}, \mathcal{T}^2_{\text{box}}, \mathcal{T}^3_{\text{box}}$ and $\mathcal{T}^4_{\text{box}}$ (left), and $\mathcal{T}^0_{\text{tri}}, \mathcal{T}^1_{\text{tri}}, \mathcal{T}^2_{\text{tri}}, \mathcal{T}^3_{\text{tri}}$ and $\mathcal{T}^4_{\text{tri}}$ (*right*)

	n_p	n_t		n_p	n_t
$\mathcal{T}^0_{\text{box}}$	553	1,627	$\mathcal{T}^0_{\text{tri}}$	4,209	16,910
$\mathcal{T}^1_{\text{box}}$	1,742	7,351	$\mathcal{T}^1_{\text{tri}}$	6,675	29,561
$\mathcal{T}^2_{\text{box}}$	6,517	32,887	$\mathcal{T}^2_{\text{tri}}$	16,249	82,186
$\mathcal{T}^3_{\text{box}}$	25,691	140,588	$\mathcal{T}^3_{\text{tri}}$	55,253	304,412
$\mathcal{T}^4_{\text{box}}$	103,253	587,283	$\mathcal{T}^4_{\text{tri}}$	215,723	1,235,745



$\mathcal{T}^1_{\text{box}}$
$\mathcal{T}^2_{\text{box}}$
$\mathcal{T}^3_{\text{box}}$
$\mathcal{T}^4_{\text{box}}$

Fig. 11 Streaming refinement of $\mathcal{T}^0_{\text{box}}$ governed by a distance-to-a-plane metric: *1* ($\mathcal{T}^1_{\text{box}}$), *2* ($\mathcal{T}^2_{\text{box}}$), *3* ($\mathcal{T}^3_{\text{box}}$), and *4* ($\mathcal{T}^4_{\text{box}}$) levels of refinement; some elements have been removed to show interior detail

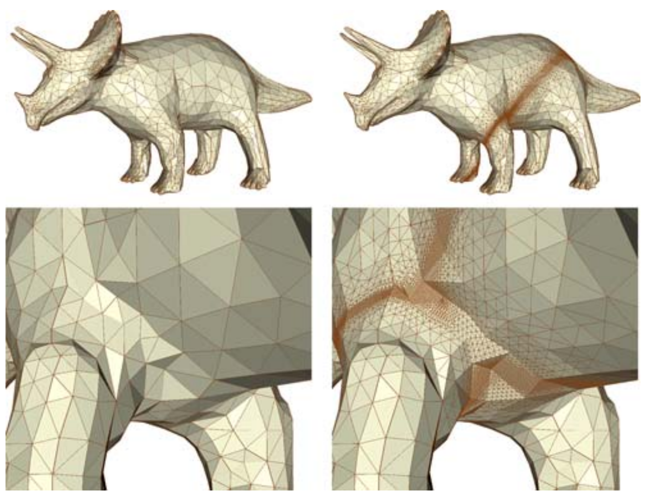


Fig. 12 Overviews of the boundaries of the initial mesh of a triceratops (*left*), and of the beauty queen (*right*) obtained after four steps of refinement governed by a distance-to-a-plane metric

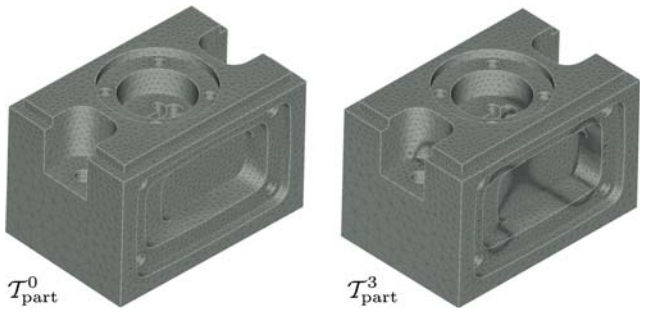


Fig. 13 Boundary overviews of the initial mesh ($\mathcal{T}^0_{\text{part}}$) of a mechanical part, and of the mesh ($\mathcal{T}^3_{\text{part}}$) obtained after three refinement steps

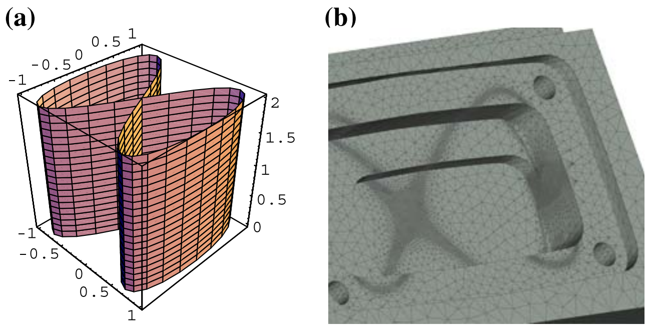


Fig. 14 a The generalized cylinder with directrix the Lissajous curve (cos t, sin $(2t)$, 0), and **b** close-up of the final refined mesh $\mathcal{T}^3_{\text{part}}$

shown in the overview (Fig. 13, right) and the close-up (Fig. 14, right) of the boundary of $\mathcal{T}^3_{\text{part}}$. The mesh topologies of the four levels of refinement governed by that edge subdivision strategy are summarized in Table 9, and Fig. 15 displays cuts across the corresponding meshes.

3.3 Mesh quality

It is well known (cf. [3, 4]) that mesh quality is crucial to the accuracy of finite element computations; this is almost as equally important in the context of visualization. In particular:

1. Rendering that uses Phong shading on triangles with significant changes in their vertex normals incorrectly models the physics of lighting and will appear blemished; this is because the normal vectors at each vertex are linearly interpolated (which does *not* maintain unit length) over the interior. The approximation results in abrupt changes in lighting along triangle edges and inconsistent lighting on the triangle interiors
2. Isosurfacing even decent cells can generate bad triangles when the isosurface intersects cell edges near vertices, in which case one gets many very small triangles right next to large ones (and that is a problem, as explained in Item 1). In fact, when this situation occurs, the resulting triangles may also have extremely poor normals, as Bloomenthal and Ferguson [1] note. They then propose a technique to eliminate such triangles while Moore and Warren [10] alter the mesh vertices in the neighborhood of such triangles to produce better results. Cells that are slivers are much more likely to generate triangles with these problems.
3. Unstructured volume rendering of poorly shaped elements can also cause trouble, especially depending on how the elements are oriented relative to the screen; many volume rendering techniques approximate the lighting integral [11], for which there is no analytical solution, with an integral that *does* have an analytical solution. Or, they use lookup tables. Usually, the approximations assume that the power of an exponential function is relatively constant over the limits of integration. This can be bad in general and worse for degenerate tetrahedra.

Table 9 Numbers of points (n_p) and tetrahedra (n_t) 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}
$\mathcal{T}^0_{\text{part}}$	28,694	150,779
$\mathcal{T}^1_{\text{part}}$	62,854	345,529
$\mathcal{T}^2_{\text{part}}$	211,480	1,207,377
$\mathcal{T}^3_{\text{part}}$	865,695	5,034,682

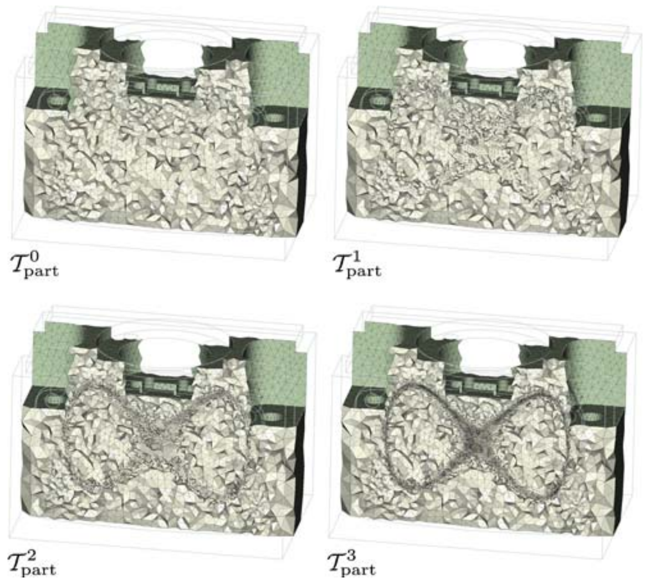

$\mathcal{T}^0_{\text{part}}$
$\mathcal{T}^1_{\text{part}}$
$\mathcal{T}^2_{\text{part}}$
$\mathcal{T}^3_{\text{part}}$

Fig. 15 Streaming refinement of $\mathcal{T}^0_{\text{part}}$ governed by an analytical size map: *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

Therefore, an important aspect of mesh modification is the effect of the considered scheme over mesh quality. Indeed, mesh quality degradation is a typical weak point of mesh subdivision without subsequent mesh smoothing; this issue is discussed, e.g., in [4] for mid-edge based tetrahedral subdivisions, where substantial mesh quality reduction is reported. We therefore discuss the qualitative impact of our method by the means of two tetrahedral quality measures: aspect ratio and radius ratio; more precisely, we examine how the range, average and standard deviations thereof evolve throughout several steps of streaming refinement.

Aspect ratio: denoted ι, it appears the most accepted estimate in the context of tetrahedral finite element analysis, in particular because it explicitly appears in a priori error estimates (see, e.g., [3]). This measure is defined for each tetrahedron σ by

$$\iota_\sigma = \alpha \frac{h_{\max}}{r}, \tag{14}$$

where $h_{\max}$, r and α respectively denote the longest edge length and the inradius of σ, and a normalization coefficient such that $\iota_\sigma = 1$ when σ is a regular tetrahedron. An elementary geometric calculation shows that $\alpha = 1/12\sqrt{6}$. Indeed, 1 is the absolute *minimum* of ι and is reached only by regular tetrahedra. We have in particular written a `Visualization ToolKit` (`VTK`, cf. [16]) class[6] that computes this quality measure.

Radius ratio: denoted ρ, we also use it, mostly because there is an existing `VTK` class that computes it and because some in the meshing community seem to prefer it. This measure is defined for each tetrahedron σ as follows:

$$\rho_\sigma = \beta \frac{R}{r}, \tag{15}$$

where R is the circumradius of σ, and β is a normalization coefficient, such that $\rho_\sigma = 1$ when σ is regular. Since $R = 3r$ for any regular tetrahedron, it follows immediately that $\beta = 1/3$.

Remark 3.2 In the case of triangular meshes, it is shown in [13] 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, which has a salient point instead. In the case of 3D simplicial (tetrahedral) meshes, this result has not been proven but it is reasonable to expect similar behaviors.

The results of quality assessments throughout the mesh refinement processes outlined above are provided in Tables 10, 11 and 12. Similar trends in quality are observed in the three cases. Neither averages nor standard deviations are dramatically degraded, in the sense of aspect ratio as well as of radius ratio. 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 [13]. Overall, the reduction in average quality is comparable to what is described in [4] 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 four subfaces created when a point is inserted to remove an ambiguity have better aspect ratios than the two subfaces of either possible decompositions that use existing mid-edge and corner vertices.

Table 10 Best, average, worst, and standard deviation of ι (aspect ratio) and ρ (radius ratio) qualities of meshes $\mathcal{T}^1_{\text{box}}$, $\mathcal{T}^2_{\text{box}}$, $\mathcal{T}^3_{\text{box}}$, and $\mathcal{T}^4_{\text{box}}$

	n_t	$\bar{\iota}$			σ_ι	$\bar{\rho}$			σ_ρ
			$-$	$+$			$-$	$+$	
$\mathcal{T}^0_{\text{box}}$	1,627	**1.85**	1.05	12.2	0.73	**1.7**	1.01	37.5	1.78
$\mathcal{T}^1_{\text{box}}$	7,351	**1.97**	1	12.2	0.79	**1.91**	1	35.4	1.83
$\mathcal{T}^2_{\text{box}}$	32,887	**2.16**	1	18	0.94	**2.3**	1	116	2.63
$\mathcal{T}^3_{\text{box}}$	140,588	**2.4**	1	18	1.15	**2.82**	1	121	3.6
$\mathcal{T}^4_{\text{box}}$	587,283	**2.71**	1	32.6	1.44	**3.58**	1	384	5.38

Table 11 Best, average, worst, and standard deviation of ι (aspect ratio) and ρ (radius ratio) qualities of meshes $\mathcal{T}^1_{\text{tri}}$, $\mathcal{T}^2_{\text{tri}}$, $\mathcal{T}^3_{\text{tri}}$, and $\mathcal{T}^4_{\text{tri}}$

	n_t	$\bar{\iota}$			σ_ι	$\bar{\rho}$			σ_ρ
			$-$	$+$			$-$	$+$	
$\mathcal{T}^0_{\text{tri}}$	16,910	**1.67**	1.04	13.13	0.64	**1.5**	1.01	64.4	1.19
$\mathcal{T}^1_{\text{tri}}$	29,561	**1.7**	1.04	13.13	0.59	**1.54**	1.01	64.4	1.11
$\mathcal{T}^2_{\text{tri}}$	82,186	**1.87**	1.04	16.4	0.66	**1.81**	1.01	123	1.52
$\mathcal{T}^3_{\text{tri}}$	304,412	**2.14**	1.04	28	0.87	**2.28**	1.01	335	2.68
$\mathcal{T}^4_{\text{tri}}$	1,235,745	**2.46**	1.04	44.4	1.16	**2.95**	1.01	636	4.61

Table 12 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_t	$\bar{\iota}$			σ_ι	$\bar{\rho}$			σ_ρ
			$-$	$+$			$-$	$+$	
$\mathcal{T}^0_{\text{part}}$	150,779	**1.5**	1.01	12	0.27	**1.29**	1	40.9	0.3
$\mathcal{T}^1_{\text{part}}$	345,529	**1.63**	1.01	19.9	0.39	**1.44**	1	168	0.65
$\mathcal{T}^2_{\text{part}}$	1,207,377	**1.85**	1.01	33.2	0.59	**1.75**	1	427	1.42
$\mathcal{T}^3_{\text{part}}$	5,034,682	**2.11**	1.01	46.4	0.81	**2.19**	1	890	2.84

3.4 Speed

The speed of execution of the algorithm is an important performance measure second only to adequate mesh quality. We provide speedups in Figs. 16 and 17 to illustrate that the algorithm scales as expected. The former figure represents a small, contrived example that places identical elements on all processes and subdivides them identically so that the load is perfectly balanced. The mesh contains a total of only 160 tetrahedra initially, arranged into a $48 \times 3 \times 3$ unit brick and refined wherever edges are near a plane with base point (0,0,1) and normal (0,0,1) using the same scheme as the cuboid and triceratops examples. The bottom lines of the speedup graph are far-removed from the theoretical linear speedup because the time required to pass the test

[6]http://www.vtk.org/doc/nightly/html/classvtkMeshQuality.html

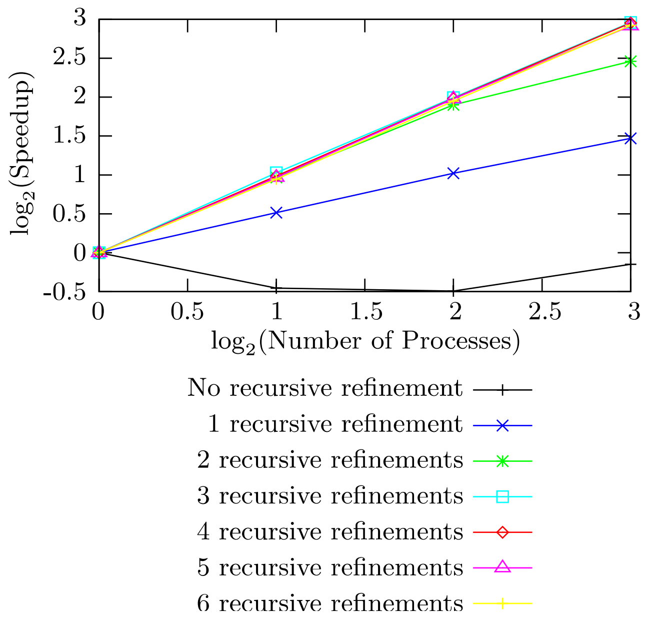


Fig. 16 Speedup for refining a perfectly load-balanced synthetic mesh with 2, 4, or 8 processes

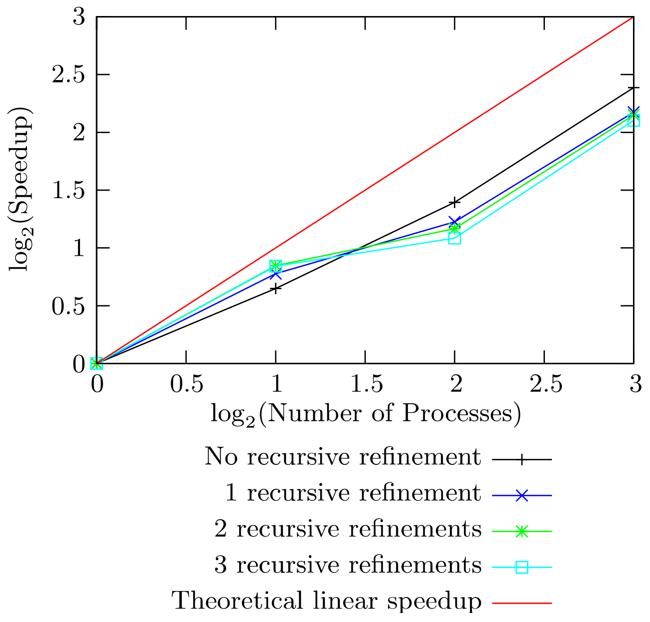


Fig. 17 Speedup for refining a naively load-balanced version of $\mathcal{T}_{\text{part}}$ with 2, 4, or 8 processes

mesh through the refinement algorithm was small relative to the noise in the timing measurements. The other lines show that with a well-balanced scheme, linear speedups are possible (because there is no communication).

The latter figure depicts the performance of the Lissajous refinement algorithm applied to the mechanical part with a naive z-slab-based decomposition. Although the speedup is not linear, it is still acceptable and it illustrates the importance of load balancing. Since load balancing is very application-dependent and since the refinement algorithm is typically *not* used as the sole criterion for load balancing, this example has been chosen with a deliberately imbalanced partition of elements. When used in real-life applications where such an imbalance exists, a repartitioning of the mesh will most probably be required after refinement.

Table 13 Throughput for varying levels of refinement of the cuboid mesh with the distance-to-a-plane subdivision criterion

Number of refinements	Number of tetrahedra	Throughput [Mtets/s]
0	1,627	0.737
1	7,351	0.726
2	32,887	0.654
3	140,588	0.657
4	587,283	0.662
5	2,447,890	0.652

Table 14 Throughput for varying levels of refinement of the triceratops mesh with the distance-to-a-plane subdivision criterion

Number of refinements	Number of tetrahedra	Throughput [Mtets/s]
0	16,910	0.666
1	29,561	0.624
2	82,186	0.637
3	304,412	0.628
4	1,235,745	0.627
5	5,135,168	0.620

Table 15 Throughput for varying levels of refinement of the mechanical part mesh with the Lissajous-based subdivision criterion

Number of refinements	Number of tetrahedra	Throughput [Mtets/s]
0	150,779	0.662
1	345,529	0.633
2	1,207,377	0.653
3	5,034,682	0.663
4	21,984,967	0.667

In addition to the speedups, some characterization of the absolute speed of the algorithm is in order. We measure the absolute speed by measuring the CPU time required to execute the algorithm on a single-CPU AMD Opteron 246 running Linux (Fedora Core 2) with 4 GB of memory. We compiled our code with gcc 3.3.3 and `-O2` optimizations. 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); this is why CPU time without refinement is nonzero. The times required to perform the refinements and the throughput (in Mtets[7] per second) are

[7]millions of tetrahedra

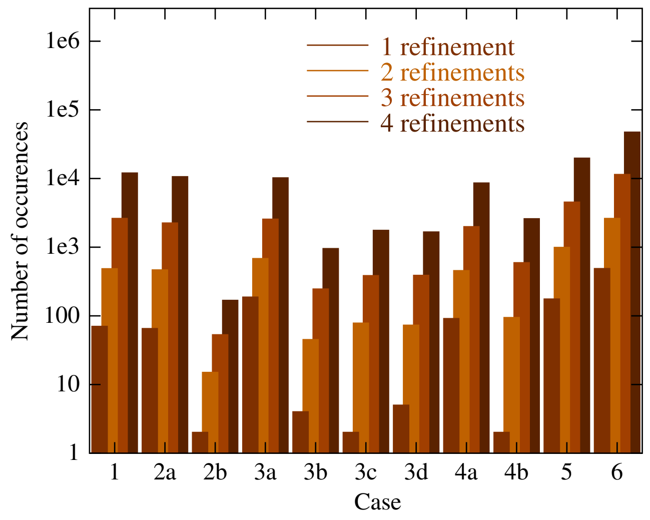


Fig. 18 Total counts of the cases encountered at each of the four steps of refinement of the cuboid mesh

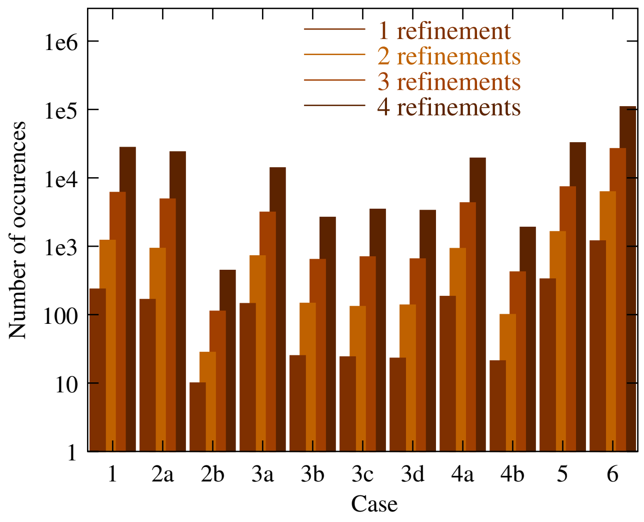


Fig. 19 Total counts of the cases encountered at each of the four steps of refinement of the triceratops mesh

shown in Tables 13, 14 and 15 for the three sample meshes from the previous sections. The same machines and technique were used to create the speedup tables above.

The speed results in the case of the cuboid should probably be given less weight than those of the mechanical part, since $\mathcal{T}^0_{\text{box}}$ is a synthetic mesh and the corresponding input is relatively small. Overall, we observe a nearly linear increase in time with output mesh size.

The following sections summarize characteristics of the algorithm discovered during our evaluation.

3.5 Algorithm characteristics

One property of our proposed technique is that, of the four triangles produced by tessellating the isosceles trapezoid, two of them will be isosceles triangles. This means that if further subdivisions are required on the two equal-length edges of either isosceles triangle, we will again have an ambiguous case. This is undesirable since detecting the ambiguity and placing point a is slightly more work than handling an unambiguous case.

3.6 A final note on ambiguous cases

One of the things that affect the size and quality of the output meshes, as well as the speed to compute them, is the probability that each subdivision case is encountered. Therefore, we have instrumented a version of the tessellator to count cases, and it was used to examine the refinement of several meshes. The distribution histograms corresponding to four levels of the previously described refinements for the cuboid, triceratops, and mechanical part meshes are shown respectively in Figs. 18, 19, and 20. Global trends observed in Figs. 18, 19 and 20 are similar for the three meshes, although

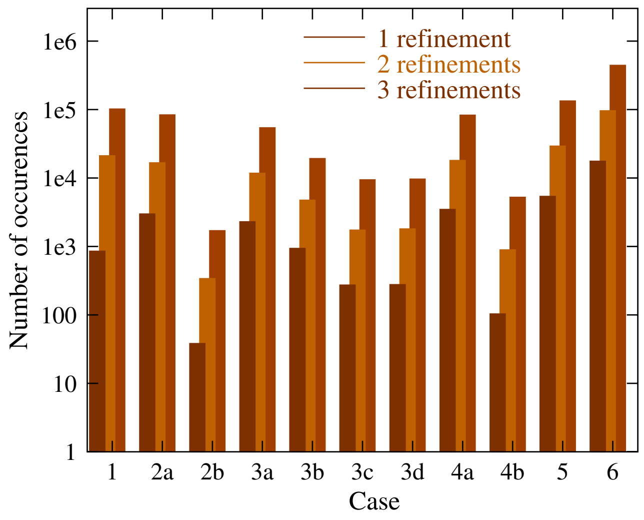


Fig. 20 Total counts of the cases encountered at each of the four steps of refinement of the mechanical part mesh

those exhibit substantially different topologies (e.g., number of elements, genus) and geometric (e.g., boundary curvature, quality) features. The fact that every case is encountered in these meshes and their refinements is yet another global validation of the tessellator. However, this does not mean that all subcases, and in particular ambiguous subcases, have been tested. We therefore refined the diagnostic by examining the case counts for ambiguous cases at the end of the refinement process. Surprisingly enough, not a single ambiguous tetrahedron is hit during the four levels of refinement of the mechanical part, and thus only the statistics for the cuboid and the triceratops are shown in Fig. 21. Case 3d, derived from 3c for practical implementation reasons (orientation-preserving Θ), is

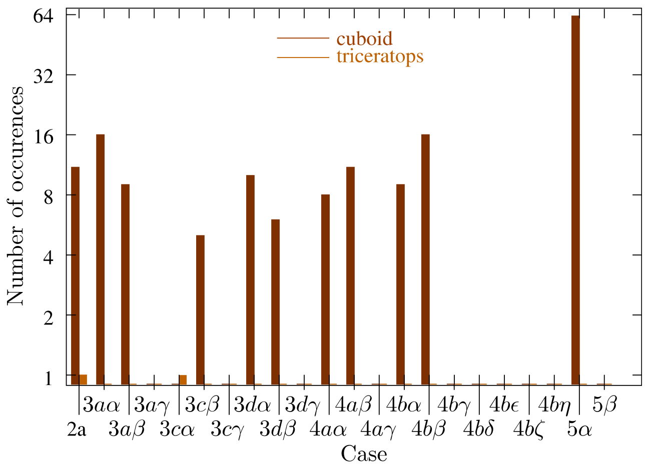


Fig. 21 Total counts of the ambiguous cases encountered during the four steps of refinement governed by a distance-from-a-plane metric of the cuboid and triceratops meshes

decomposed as 3c, in subcases 3dα, 3dβ and 3dγ; each of them is therefore only one odd permutation (in fact, a transposition) away from its canonical counterpart. It appears that the similarity previously observed for the distribution among the general cases no longer holds when considering ambiguous sub-configurations: the diversity of the original meshes is reflected by large variations among ambiguous cases, since the cuboid, mechanical part, and triceratops refinements respectively result in 164, 0, and 2 ambiguous cases. Regardless of the distribution of ambiguous cases, the fact that they are encountered means that handling them is not only an academic goal, but that is necessary to a robust communicationless mesh refinement code.

3.7 Future work: improving mesh quality

Due to the importance of preserving mesh quality for visualization, and even more so for computation applications, we are considering different options for including that requirement within the streaming approach. In particular, we are considering taking advantage of the fact that some canonical cases (both ambiguous and unambiguous) allow for several subdivision templates. Therefore, possible future work includes allowing the algorithm to choose between several subdivision templates whenever possible, in order to optimize quality depending on the geometry of the tetrahedron to be refined. Here is an example of how one particular case might be handled.

Example 3.1 Consider the tetrahedron σ with type 4b, whose vertices 0, 1, 2, and 3 have the following coordinates, respectively: (3,0,0), (0,0,0), (1,1,1), and (1,2,0). Edges 01, 12, 02 and 03 then have respective lengths 3, $\sqrt{3}$, $\sqrt{6}$ and $2\sqrt{2}$, and thus σ pertains to case 4b$^u\beta$. The canonical decomposition we use is (1): 6815, 6871, 6701, 8732, 6852, 6827. However, there is an alternate decomposition (2): 7815, 6571, 6701, 8732, 7852, 6527. In fact, the two subdivisions templates differ, depending on how they split the quadrangle 5678: using either edge 57, or 68. Now, regarding the aspect ratio ι, one gets the results summarized in Table 16. It appears that the canonical subdivision template is not the best one. We might hope to find a heuristic less costly than computing all possible output tessellation qualities. For example, consider comparing |57| and |68| for the case 4b$^u\beta$ example above. Unfortunately, the diagonal lengths are equal $|57| = |68| = \frac{\sqrt{11}}{2}$ giving us no way to select one template over the other. This has proven true in general, so we simply compute the quality of all tetrahedra for each template and use statistics to choose the best template.

Preliminary results indicate that implementing only a few alternative templates can even improve mesh quality as it is refined, as shown in Tables 17 and 18. In any case, these results show a dramatic quality improvement versus what has been obtained in Tables 11 and 12. The performance penalty for this partial implementation is approximately 15% but needs to be more thoroughly characterized.

4 Conclusions

We have presented a new scheme for refining tetrahedral meshes that does not require neighborhood information. 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

Table 16 Best, average, and worst ι (aspect ratio) quality for the two possible subdivisions of a 4b$^u\beta$ tetrahedron

	n_t	$\bar{\iota}$ (− +)
4b$^u\beta - 1$ (canonical)	6	**2.42** 1.22 3.59
4b$^u\beta - 2$ (alternate)	6	**2.3** 1.22 3.01

Table 17 Best, average, and worst ι (aspect ratio) and *rho* (radius ratio) qualities of meshes $\mathcal{T}^0_{\text{tri}}$, $\mathcal{T}^1_{\text{tri}}$, $\mathcal{T}^2_{\text{tri}}$, $\mathcal{T}^3_{\text{tri}}$, and $\mathcal{T}^4_{\text{tri}}$ when using average quality driven adaptive templates (partial implementation)

	n_t	$\bar{\iota}$ (− +)	σ_ι	$\bar{\rho}$ (− +)	σ_ρ
$\mathcal{T}^0_{\text{tri}}$	16,910	**1.67** 1.04 13.13	0.64	**1.5** 1.01 64.4	1.19
$\mathcal{T}^1_{\text{tri}}$	29,561	**1.63** 1.04 13.13	0.54	**1.45** 1.01 64.4	0.96
$\mathcal{T}^2_{\text{tri}}$	81,213	**1.59** 1.04 13.13	0.41	**1.39** 1.01 64.4	0.64
$\mathcal{T}^3_{\text{tri}}$	291,252	**1.56** 1.04 13.13	0.34	**1.36** 1.01 64.4	0.44
$\mathcal{T}^4_{\text{tri}}$	11,335,558	**1.55** 1.04 13.13	0.31	**1.35** 1.01 64.4	0.34

Table 18 Best, average, and worst ι (aspect ratio) and *rho* (radius ratio) qualities of meshes $\mathcal{T}^0_{\text{part}}$, $\mathcal{T}^1_{\text{part}}$, $\mathcal{T}^2_{\text{part}}$, and $\mathcal{T}^3_{\text{part}}$, when using average quality driven adaptive templates (partial implementation)

	n_t	$\bar{\iota}$			σ_ι	$\bar{\rho}$			σ_ρ
			−	+			−	+	
$\mathcal{T}^0_{\text{part}}$	150,779	**1.5**	1.01	12	0.27	**1.29**	1	40.9	0.3
$\mathcal{T}^1_{\text{part}}$	345,529	**1.54**	1.01	12	0.3	**1.33**	1	40.9	0.33
$\mathcal{T}^2_{\text{part}}$	1,194,361	**1.56**	1.01	13.5	0.35	**1.36**	1	40.9	0.4
$\mathcal{T}^3_{\text{part}}$	4,851,010	**1.56**	1.01	14.7	0.38	**1.36**	1	65	0.45

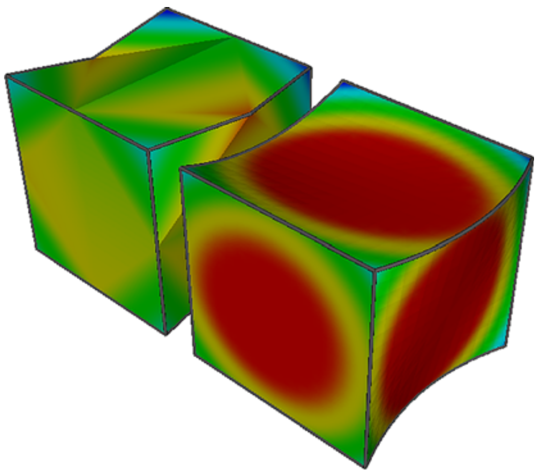


Fig. 22 Surface rendering of a scalar field interpolated over a quadratic hexahedron: linear interpolation with 20 tetrahedra (*left*) and after 3 levels of streaming mesh refinement (*right*)

each face and as many times per edge as there are tetrahedra that reference it), it bypasses any communication that might otherwise be required. This makes it viable for streaming large datasets and for parallel processing, where communication would be required to process elements on boundaries between processes.

Although we cannot make any conclusions about how well the algorithm will work with all possible applications—because balancing the load is dependent on application specifics such as the initial distribution of elements and the subdivision metric—we can say that the algorithm scales linearly with the number of processes in the load-balanced case. We have also presented evidence that the quality of the mesh is not significantly degraded during refinement. In addition, our initial goal of improving visualization of higher order elements has been attained, as illustrated in Figs. 22 and 23.

We have implemented the proposed approach, and the code is available as the `vtkStreamingTessellator` class[8] in `ParaView`. We have also added mesh

[8]http://www.vtk.org/doc/nightly/html/classvtkStreamingTessellator.html

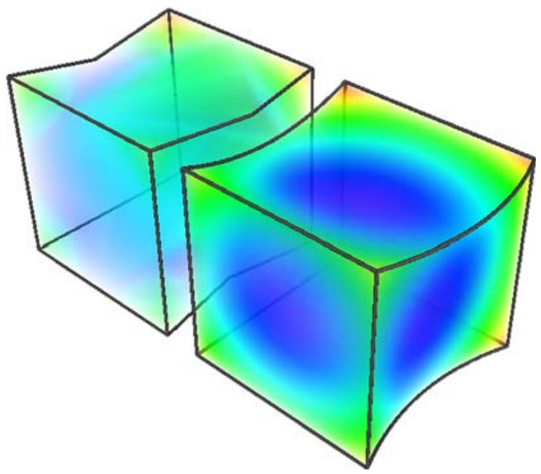


Fig. 23 Volume rendering of a scalar field interpolated over a quadratic hexahedron: linear interpolation with 20 tetrahedra (*left*) and after 3 levels of streaming mesh refinement (*right*)

quality tools in `VTK`. This has allowed us to test the method, and assess mesh quality and algorithm speed. Three meshes were evaluated: a synthetic geometric mesh of a cuboid, a mesh of an extinct reptile, and a mesh of a mechanical part. The qualities of the refined meshes are comparable to other refinement schemes that use communication to resolve ambiguities.

In addition to mesh quality, the time performance of the algorithm is characterized. The algorithm can generate approximately 600,000 tetrahedra per second per processor on modern hardware. More importantly, the algorithm scales roughly linearly with the size of the output. To be precise, we expect the algorithm to scale linearly with the number of function calls to the subroutine that subdivides a tetrahedron. When the maximum number of recursive calls is limited, this number and the number of output tetrahedra are roughly proportional. As recursion depth is increased, the number of function calls will grow faster than the number of output tetrahedra. However, for a mesh of significant size, recursing more than four or five times will quickly fill memory with output tetrahedra, as the mechanical part example reveals.

Finally, we have presented an approach for improving the quality of refinements produced with the algorithm in this paper and hope to generate alternative decompositions of all the cases that will admit them.

Acknowledgements David C. Thompson and Philippe P. Pébay were supported by the United States Department of Energy, Office of Defense Programs. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

References

1. Bloomenthal J, Ferguson K (1995) Polygonization of non-manifold implicit surfaces. Comput Graph, 29(Annual Conference Series):309–316

2. Chung AJ, Field AJ (2000) A simple recursive tessellator for adaptive surface triangulation. J Graph Tools 5(3):1–9
3. Ciarlet PG (1991) Handbook of numerical analysis, vol 2. North Holland, Amsterdam, pp 17–352
4. Frey PJ, George P-L (2000) Mesh generation. Hermes Science Publishing, Oxford
5. Henderson A (1995) The ParaView guide. Kitware, Inc.
6. Hungerford TW (1997) Algebra. Graduate texts in mathematics. Springer, Berlin Heidelberg New York
7. Labbé P, Dompierre J, Vallet M-G, Guibault F, Trépanier J-Y (2001) A measure of the conformity of a mesh to an isotropic metric. In: Proceedings of the 10th international meshing roundtable, Newport Beach
8. Lawrence JD (1972) A catalog of special plane curves. Dover Publications, New York
9. Lo SH (1998) 3d mesh refinement in compliance with a specified node-spacing function. Comput Mech 21:11–19
10. Moore D, Warren J (1991) Mesh displacement: an improved contouring method for trivariate data. Technical Report TR91-166, Rice University, September
11. Moreland KL (2004) Fast high accuracy volume rendering. PhD Thesis, University of New Mexico, July
12. Oliker L, Biswas R, Gabow HN (2000) Parallel tetrahedral mesh adaptation with dynamic load balancing. Parallel Comput J, Special Issue on Graph Partitioning, pp 1583–1608
13. Pébay PP, Baker TJ (2003) Analysis of triangle quality measures. Math Comput 72(244):1817–1839
14. Pébay PP, Thompson DC (2005) Communication-free streaming mesh refinement. ASME Trans J Comput Inf Sci Eng, Special Issue on Mesh-Based Geometry 5(4):309–316
15. Ruprecht D, Müller H (1998) A scheme for edge-based adaptive tetrahedron subdivision. In: Hege H-C, Polthier K (eds) Mathematical visualization. Springer, Berlin Heidelberg New york, pp 61–70
16. Schroeder W, Martin K, Lorensen B (1995) The visualization toolkit: an object-oriented approach to 3D graphics. Prentice Hall, Englewood Cliffs
17. Thompson DC, Pébay PP (2004) Performance of a streaming mesh refinement algorithm. Sandia Report SAND2004-3858, Sandia National Laboratories, August
18. van Rossum G (1995) Python language reference manual. Network Theory Ltd, Bristol
19. Velho L (1996) Simple and efficient polygonization of implicit surfaces. J Graph Tools 1(2):5–24