Method for local refinement of tetrahedral meshes

T. Kuutela*

January 14, 2022

1 Introduction

Partial or "local" refinemenet of tetrahedral mesh is sometimes useful in e.g. finite element computations. However, local refinement of tetrahedrons without remeshing the whole domain turns out to be a surprisingly difficult problem. The difficulty is well explained on page 3 of [1], which also describes one approach, especially well suited for cases where the resulting mesh is not intended to be saved. In following, we describe an algorithm that is similarly vectorizable and embarrassingly parallel - assuming the availability of a parallel integer sorting algorithm - and has predictable computation time. The predictable speed is achieved by having no (non-fixed-length) iterative parts in the algorithm, unlike classical edge bisection algorithms. The main difference to [1] is that we do not explicitly handle faces and instead directly split tetrahedra.

*Aalto University, Department of Mathematics and Systems Analysis, P.O. Box 11100, FI-00076 Aalto, Finland (topi.kuutela@aalto.fi). The work of TK was supported by the Academy of Finland (decision 312124).

2 Algorithm

Inputs:
p: list of node positions,
t: list of tetrahedra,
edge_splits: list of edges to be split

Algorithm:

1: Add new nodes at centers of edges to split. Keep track of "edge-to-new-point" map.
2: Classify tetrahedra into classes with topologically equivalent edge split patterns. (See below for details)
3: Add center nodes to tetrahedra in 3-triple, 4-run and five split patterns (3a, 4b, 5). Split these into four parts by creating news faces from all edges to the center node. The resulting tetrahedra can be split more easily. (See below for details)
4: Split each split class separately. Keep track of which faces had two splits and also record alternative topologies for these. (See Section 3 for details)
5: Fix compatibility between tetrahedra connected to twice split faces (See Section 2.1 for details)

Outputs:
p: New list of node positions,
t: New list of tetrahedrons,
new_to_old: Mapping from new tetrahedra to old tetrahedra indices.

Step 2: Tetrahedron classification can be done uniquely by counting how many split edges are connected to each vertex of a tetrahedron. The four counts then uniquely map to the split class. Note that these counts are not sufficient to uniquely determine which node corresponds to each node in the split tetrahedron in the 3-chain (3c) case.
Step 3: Compatibility between tetrahedra sharing a twice-split face is a little trickier problem. In the step 3, we filtered out edge split patterns that result in tetrahedron splits such that faces may be impossible to fix without also modifying tetrahedrons that are not connected to the twice-split face. In fact, it is possible to construct an arbitrarily long chain of necessary corrections unless these cases are filtered in advance.

The remaining tetrahedra with twice split faces always have exactly two configurations for the sub-tetrahedra connected to the twice split face(s). Furthermore, the configurations are such that swapping between them never affects the topology of any other tetrahedra.

The split of 3a, 4b and 5 split classes is always done so that a newly created center node is connected to the original four nodes, and the tetrahedron is then split to four tetrahedra corresponding to the faces and the center node. The split patterns from the edge splits in the resulting four tetrahedron can then be processed along with the rest of the mesh.

2.1 Fixing face compatibility

The compatibility correction can be handled neatly with the following datastructures:

`Bisplit faces`: A collection of 5-tuples $(f_0, f_1, f_2, f_3, f_4)$ corresponding to the nodes of twice split faces. f_0, f_1 and f_2 are the nodes in the original mesh in increasing node index order, while f_3 and f_4 are the edge centers. The order of f_3 and f_4 is defined so that f_3 is the node with three connecting edges (in the face), and f_4 is the node with four connecting edges. Note that

this way (f_0, f_1, f_2) triplet is a unique indicator for faces in the mesh, while f_3 and f_4 indicates which splitting topology resulted from the tetrahedron splitting. Note that for the interior of the mesh, the two tetrahedra corresponding to each side of the face are separately split. Therefore for the interior, there are always two `bisplit face` entries, while for the faces on the boundary of the mesh, there is only one.*

`Primary tetrahedra`: The pair of tetrahedra corresponding to the topology defined by the `bisplit faces`.

`Secondary tetrahedra`: The pair of tetrahedra which swaps the face topology on the corresponding `bisplit face`.

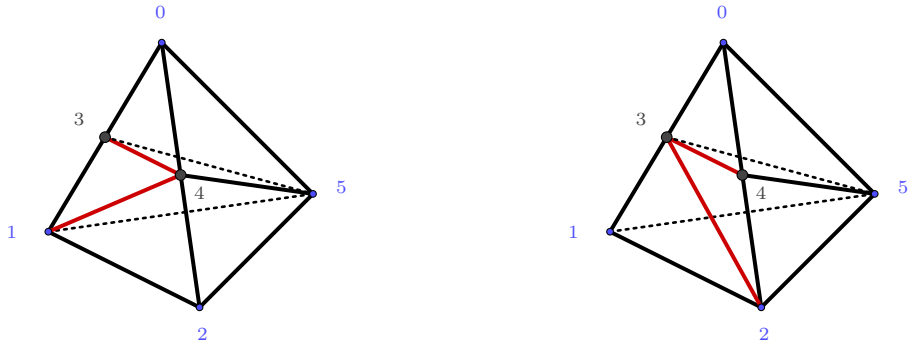


Figure 1: Two topologies for the face $(0, 1, 2)$. In the left figure the `bisplit face` 5-tuple is $(0, 1, 2, 3, 4)$, while in the right it is $(0, 1, 2, 4, 3)$. On the left tetrahedron, the corresponding `primary tetrahedra` are $(1, 3, 4, 5)$ and $(1, 2, 4, 5)$, while the `secondary tetrahedra` are $(2, 3, 4, 5)$ and $(1, 2, 3, 5)$. Note that for both configurations the third tetrahedra remains the same: $(0, 3, 4, 5)$.

Collecting the `bisplit face` 5-tuples in an array, the `bisplit face` can then be sorted by the keys defined by f_0, f_1 and f_2. Pairwise consecutive `bisplit face` then classify faces into three groups:

1. If two subsequent 5-tuples match entirely (i.e. all of f_0, f_1, f_2, f_3 and f_4 are the same), the topology is already matching.
2. If f_3 and f_4 are swapped in the second 5-tuple, the topology is mismatched and the corresponding tetrahedras on one side of the face should be swapped for the `secondary tetrahedra`.
3. If neither of the neighbouring 5-tuples have the same f_0, f_1 and f_2 entries, the face is on the boundary of the mesh, and therefore does not need swapping.

Note that the construction of these `bisplit face` is independent tetrahedronwise. Therefore `bisplit face` classification and pairing can be computed in parallel as it is just a sort and a series of pairwise comparisons. Finally, the swapping operation can also be carried out in parallel as the swap between `primary` and `secondary tetrahedra` can never affect the topology of neighboring tetrahedra with our choices of splitting patterns.

*In our implementation, we actually save `bisplit face` twice for each tetrahedra. This way the indexing can directly match the indexing of `Secondary tetrahedra`.

3 Split classes

In the following, we list all the topological cases for different splitting patterns that can occur in a tetrahedron.

For each pattern, we list the new tetrahedra set.

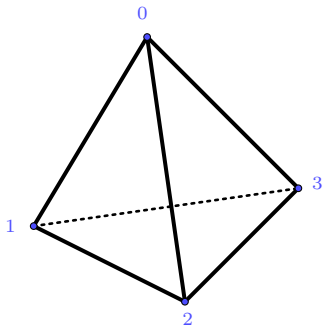
The names given to the patterns correspond to function names in our code, while the alphanumeric code corresponds to identifiers used in [1] and [2].

We have marked the bisplit faces' quadrilateral forming edges with red lines. Furthermore, where necessary, we mark the `bisplit faces` identifiers above the two tetrahedra that form the `primary tetrahedra` configuration. The alternative configuration is then included in the `secondary tetrahedra` table.

For split classes for which center node is added, we mark down the split classes for the resulting four tetrahedra. We always denote the center node with C.

Finally, for each class, we include the four-digit edge splits per vertex identifier as ID:xxxx.

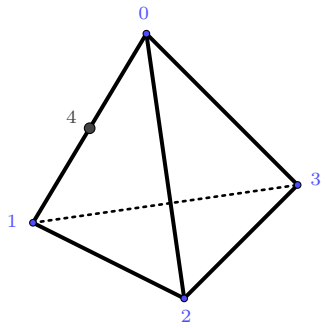
No splits (0):



Ignored

ID:0000

One split (1):

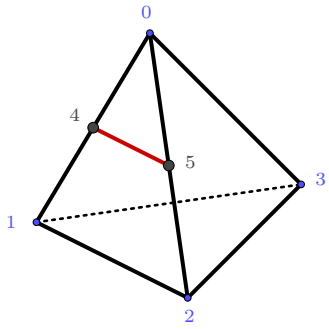


New tetrahedra

0	1
2	2
3	3
4	4

ID:0011

Two splits "pair" (2a):



New tetrahedra

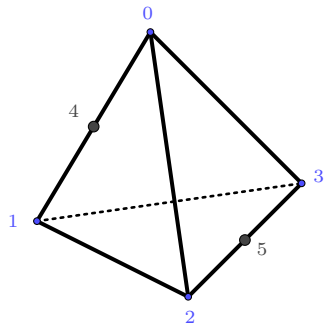
	F01254	
0	1	2
3	2	3
4	3	4
5	4	5

Secondary tetrahedra

F01254'
1
2
3
5

ID:0112

Two splits "opposite" (2b):

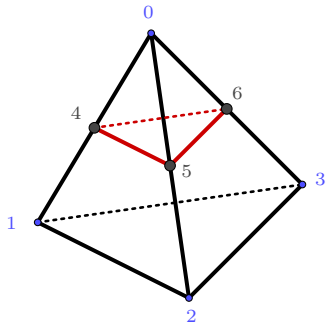


New tetrahedra

0	0	1	1
2	3	2	3
4	4	4	4
5	5	5	5

ID:1111

Three splits "triple" (3a):



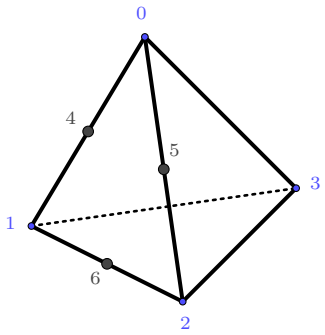
Center node C is added in this class.

Resulting split classes

2a	2a	2a	0
C	C	C	C
0	0	0	1
1	1	2	2
2	3	3	3

ID:1113

Three splits "loop" (3b):

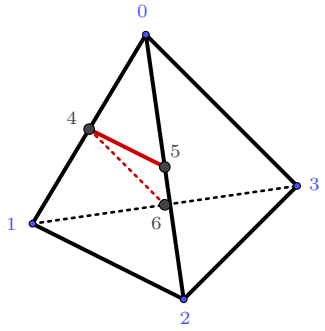


New tetrahedra

0	1	3	2
3	3	4	5
4	4	5	6
5	6	6	3

ID:0222

Three splits "chain" (3c):



New tetrahedra

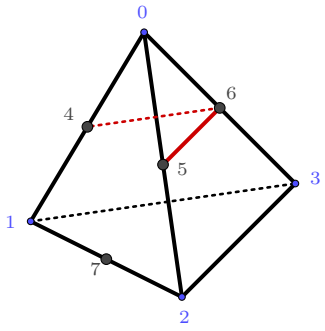
F01364		F01245		
0	3	1	1	2
3	4	2	4	3
4	5	5	5	5
5	6	6	6	6

Secondary tetrahedra

F01364'		F01245'	
0	0	1	2
4	3	2	4
5	5	4	5
6	6	6	6

ID:1122

Four splits "dangler" (4a):



New tetrahedra

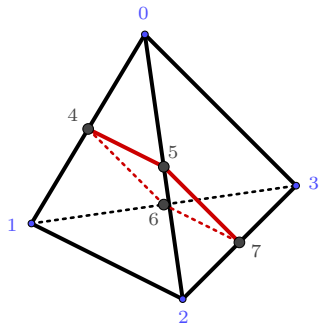
		F01346		F02356	
0	4	4	1	2	5
4	5	1	3	3	2
5	6	6	6	6	6
6	7	7	7	7	7

Secondary tetrahedra

F01364'		F02356'	
3	1	2	3
4	3	3	5
6	4	5	6
7	7	7	7

ID:1223

Four splits "run" (4b):



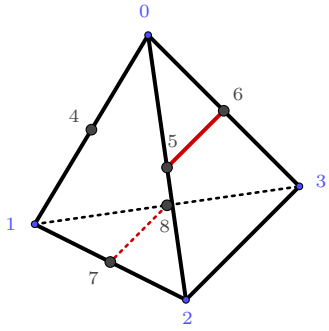
Center node C is added in this class.

Resulting split classes

2a	2a	2a	2a
C	C	C	C
0	0	0	1
1	1	2	2
2	3	3	3

ID:2222

Five splits (5):



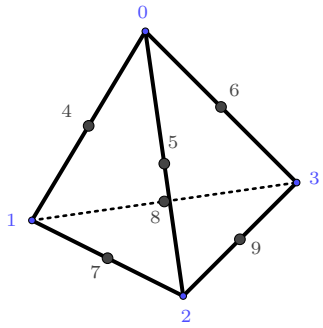
Center node C is added in this class.

Resulting split classes

3b	3b	2a	2a
C	C	C	C
0	0	0	1
1	1	2	2
2	3	3	3

ID:2233

Six splits (6):



New tetrahedra

0	6	5	4	4	4	4	4
4	8	7	1	9	9	9	9
5	9	9	7	6	6	5	7
6	3	2	8	8	5	7	8

ID:3333

References

[1] D. C. Thompson and P. P. Pébay, *Embarrassingly parallel mesh refinement by edge subdivision, Engineering with Computers 2006, vol. 2*, https://link.springer.com/content/pdf/10.1007/s00366-006-0020-3.pdf

[2] D. Ruprecht and H. Müller, *A Scheme for Edge-based Adaptive Tetrahedron Subdivision*, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.9474&rep=rep1&type=pdf