REVIVING THE SEARCH FOR OPTIMAL TETRAHEDRALIZATIONS

Célestin Marot[1] Kilian Verhetsel[1] Jean-François Remacle[1]

[1] *Université catholique de Louvain, iMMC, Avenue Georges Lemaitre 4, bte L4.05.02, 1348 Louvain-la-Neuve, Belgium*

ABSTRACT

This paper revisits a local mesh modification method known as the *Small Polyhedron Reconnection* (SPR) [1]. The core of the SPR operation is a branch and bound algorithm which computes the best 3D triangulation (tetrahedralization) of a polyhedron through an efficient exploration of the set of all its triangulations. The search can accomodate for additional geometric constraints and will inevitably find the highest quality triangulation of the polyhedron if a triangulation exists. This paper focuses on the design of an optimized SPR operator and its application to improving the quality of finite element meshes. Compared to the original algorithm, a speed-up of 10 million is obtained by changing the heuristics determining the search space exploration order. This enables the integration of the SPR operator into standard mesh generation procedures. We show quality improvements obtained by applying this operation to meshes that have already been optimized using smoothing and edge removal techniques.

Keywords: SPR, reconnection, mesh generation, topological transformation, optimal triangulation

1. INTRODUCTION

Unstructured tetrahedral mesh generation generally begins by creating a surface mesh of a given geometric model. Afterwards, a tetrahedral mesh constrained by the surface mesh is created using one of the few available 3D mesh generation algorithms: advancing front, Delaunay or octree-based techniques. Unfortunately, all of those 3D mesh generation methods tend to produce elements (tetrahedra) that are not readily suitable for finite element computations. For example, the Delaunay tetrahedralization doesn't exclude the creation of nearly flat tetrahedra, called *slivers*, that cause tremendous numerical errors. It is possible to detect slivers through various quality measures [2]. Any tetrahedral mesh generation procedure is followed by a so-called *mesh improvement* step that aims at optimizing the overall quality of the mesh, which is mainly driven by the quality of its worst element.

Mesh improvement methods can be classified into two categories: *vertex relocation (or smoothing) methods* and *topological transformations*. Smoothing is the act of modifying the coordinates of vertices without changing the connectivity of the mesh. On the other hand, topological transformations treat the mesh as a graph in which the vertex coordinates are fixed inputs.

Smoothing methods have been extensively studied in the past 25 years [3, 4, 5, 6]. The objective of mesh smoothing is twofold: obtain a better overall quality and space out vertices harmoniously so that subsequent topological transformations improve the mesh. Indeed, smoothing and topological transformations are most effective when combined [7]. While most smoothing methods optimize the mesh directly and are thus dependent of the current triangulation, some methods are able to find good placement for points relatively independently of the triangulation, using moving mesh partial differential equations or central Voronoi tesselations [8, 6, 9].

In this paper, a good distribution of points, obtained using one or a combination of these smoothing techniques, is assumed. The main focus of this paper is

thus on the improvement of the efficiency of topological transformations. Topological transformations operates on *cavities*, which are the polyhedral holes formed by the removal of a face-connected subset of tetrahedra. In 3D, the most simple cavity has four vertices, and only one possible tetrahedralization. The most basic operations, called bistellar flips, operates on a cavity of five vertices formed by the removal of two or three tetrahedra. These are the 2-3 and 3-2 flips illustrated on Figure 1. These simple local reconnection schemes are effective at removing most low-quality tetrahedra from the mesh.

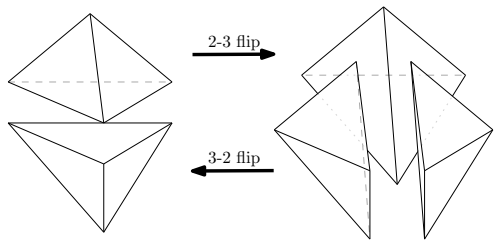


Figure 1: Two bistellar flips, namely the 2-3 and 3-2 flips, consisting in switching between two triangulations of a triangular bipyramid.

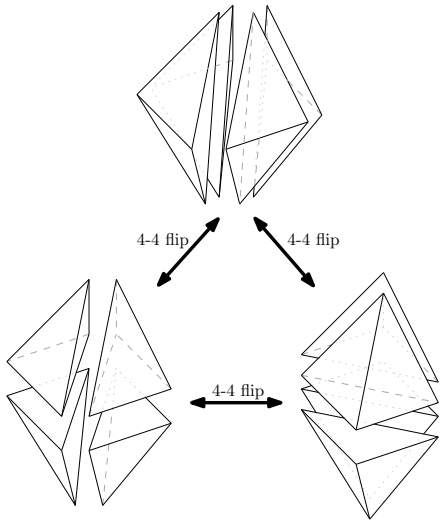


Figure 2: The 4-4 flip retriangulates an octahedron by changing the edge around which the four tetrahedra are placed.

When applying bistellar flips in a hill-climbing manner, meaning a flip is only performed if it improves the overall quality, one often reaches local minima where no bistellar flip can possibly improve the quality. To overcome this problem, increasingly complex methods have been implemented. A 4-4 flip (Figure 2) can be

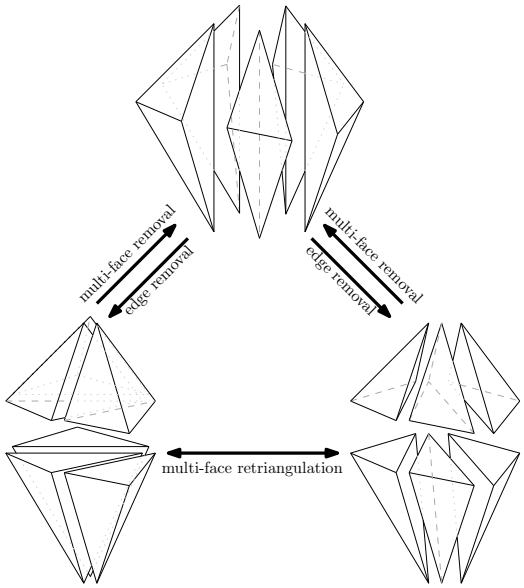


Figure 3: Edge removal, a more general operation than the 3-2 flip, replaces the cavity around an edge by choosing a triangulation of the vertices that are not on the edge.

obtained by a combination of a 2-3 and a 3-2 flip [10]. Therefore, adding this operation to the basic bistellar flips enables new optimizations that a hill-climbing technique would not find when the first 2-3 flip does not increase the mesh quality.

A more general transformation, called *edge removal*, *edge swap* or n-to-m flip, operates on a cavity formed by the the set of tetrahedra surrounding an edge [11, 12] (Figure 3). The vertices that are not the endpoints of the edge form an annulus around that edge. For each 2D triangulation of that annulus, we can create a 3D triangulation of the cavity by linking each triangle to both end points of the edge. The edge removal operation considers all these possibilities and chooses the best one if it increases the quality of the current mesh. This operation supersedes the 3-2 and 4-4 flips and can also be implemented robustly as a series of bistellar flips [12, 10]. The inverse of edge removal is called *Multi-face removal* [12]. It is less useful than edge removal in practice, and is particularly tedious to implement. Multi-face retriangulation, which is a combination of *multi-face removal* and edge removal can be used to overcome a valley in the objective function in some cases where multi-face removal cannot [13](Figure 3).

However, even with this large repertoire of topological operation, mesh improvement is still subject to

local minima [12, 13]. Instead of adding more and more operations to this zoo of topological transformation, it is possible to use an operation that generalizes all of them, called the *small polyhedron reconnection* (SPR) [1]. This operation considers the problem of finding the optimal triangulation of a cavity. A *cavity*, in this context, is a set of volumes defined by constrained closed surfaces, with possible interior constrained vertices, edges and triangles. The core idea of the SPR algorithm is to compute the best triangulation by searching the set of all possible triangulations using a branch and bound algorithm. The cavity is then replaced using its highest-quality triangulation to contain all constrained triangles and edges. This method is highly flexible and independent of the chosen quality measure. However, using the SPR operation in practice presents great technical challenges and subsequent mesh improvement procedures implemented in Tetgen [10], CGAL [14, 15] or MMG3D [16] do not use this method. Indeed, the performance of the SPR search algorithm varies dramatically based on the parameters and heuristics that it relies upon. A poor choice makes the algorithm completely inadequate for large meshes, which can now be created at a rate of several million tetrahedra per second [17]. The main contribution of this paper is to present efficient heuristics and implementation strategies that enable the use of SPR to optimize large meshes.

We propose improvements to the SPR algorithm by exploring the space of possible triangulations in a different order, aiming at a significant reduction to the number of triangulations that need to be considered during the search (Section 2.2). Repeated computations of expensive quality measures are also avoided by storing their results (Section 2.4). These choices affect the time taken to optimize cavities by several orders of magnitudes (Section 2.5). The optimized SPR procedure that is proposed has been integrated into our mesh improvement framework. The benefits of our approach are shown by applying it to large meshes (section 3).

2. QUICKLY COMPUTING OPTIMAL TRIANGULATIONS

The core of our method is an efficient algorithm to search for the best triangulation of a small polyhedral cavity $\mathcal{C}$. Let $\mathcal{T}(\mathcal{C})$ denote the set of all triangulations of this cavity. The best triangulation is defined as the triangulation of $\mathcal{C}$ that maximizes the quality of its worst element according to a quality measure q:

$$T_{\text{opt}} = \arg\max_{T \in \mathcal{T}(\mathcal{C})} \min_{t \in T} q(t)$$

The set of triangulations considered by the algorithm may be restricted to only those that contain a given set of edges and triangles if certain features must be preserved.

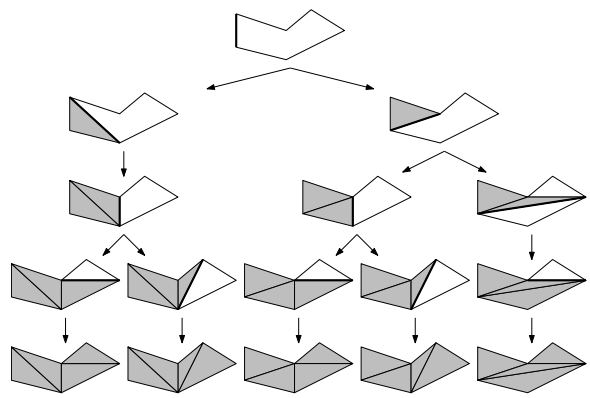


Figure 4: Enumerating the triangulations of a polygon. At each step, a boundary edge is picked, and a branch is created for each possible triangle that contains that edge. The untriangulated area remaining after a triangle insertion is filled by applying this procedure recursively.

An optimal triangulation is computed using a branch and bound algorithm (Algorithm 1). Starting from the boundary $\partial\mathcal{C}$ of the cavity $\mathcal{C}$ as input, a triangle $F \in \partial\mathcal{C}$ is selected. Any mesh of the cavity $\mathcal{C}$ must include a tetrahedron t that contains the triangle F. Each possible tetrahedron is considered, by *branching* on all possible choices for its fourth vertex. After inserting a new tetrahedron, the boundary $\partial\mathcal{C}$ is replaced by the boundary of the part of the cavity which has not yet been filled with tetrahedra (see Figure 4). The best mesh of this new cavity is computed by recursively applying the algorithm. This corresponds to the exploration of a tree whose nodes correspond to triangulations and whose edges correspond to the insertion of tetrahedra.

Throughout this process, the best triangulation found so far is tracked, as well as its quality q^*. After each branch, an upper bound on the quality of the best solution that could be obtained is computed by finding the minimum quality element that has already been added to the solution. If the upper bound is worse than q^*, this part of the search tree is skipped. By the end of the algorithm, the optimal triangulation T_{opt} will have been found.

The rest of this section discusses important design choices in order to achieve an efficient algorithm:

1. for a given triangle, how to compute the set of tetrahedra that can be built on top of it;
2. the selection of the triangle F on top of which the

next tetrahedron should be built;

3. the order in which the tetrahedra containing F are considered for insertion into the mesh;

4. how to avoid repeated evaluations of expensive geometric predicates.

Input: B: the target boundary;
T: a partial triangulation;
T^*: the best triangulation found;
q: a quality function
Output: The best triangulation of the cavity
if $B = \emptyset$ **then return** T;
$F \leftarrow$ *some triangle of* B;
foreach *vertex* v **do**
  $t \leftarrow$ *the tetrahedron formed by joining* v *to each vertex of* F;
  if $q(t) > \min_{t' \in T^*} q(t')$ **then**
    $T' \leftarrow T \cup \{t\}$;
    $B' \leftarrow B - \partial t$;
    $T^* \leftarrow$ Optimize-Cavity(B', T', T^*, q);
  end
end
return T^*;
Algorithm 1: Optimize-Cavity: search for the best triangulation with a given boundary.

2.1 Computing candidate tetrahedra

At each branching step, the algorithm considers a triangular face F before trying to add each tetrahedron t that contains F to the current triangulation. Because three of the vertices of t must be the vertices of F, only one vertex needs to be chosen. Many of those choices need to be filtered out, because inserting the corresponding tetrahedra would result into an invalid triangulation. Below are the conditions used to eliminate such candidates.

Geometric validity Each candidate tetrahedron t must have a positive orientation and a positive quality $q(t)$ greater than the quality q^* of the best triangulation found so far.

Geometric intersections The solution must not include intersecting tetrahedra. The new faces and edges of each candidate t are tested for intersection with the edges and triangles of the boundary of the unmeshed region as well as with constrained features. Tetrahedra that completely enclose a vertex are also rejected. All intersection tests are performed exactly by relying only on exact computations of the orientations of tetrahedra.

2.2 Mesh construction order

Changing the order in which tetrahedra are inserted into the mesh drastically affects the number of triangulations that need to be considered by the algorithm. This behavior is common in difficult optimization problems [18, 19]. Heuristics are used to choose a favorable order for most cases.

First, a triangular face F must be selected from the boundary of the unmeshed region. The algorithm then branches on the set of all tetrahedra containing F that can be added to the current triangulation. Faces are selected by attributing a cost to each of them. This cost is computed by summing the number of candidate tetrahedra containing the face and their qualities. A lower cost means either fewer candidates, hence a smaller search tree, or worse candidates, hence a tighter upper bound allowing for more pruning.

2.3 Ordering candidate tetrahedra

Once the face F has been selected, a second heuristic defines the order in which the different candidate tetrahedra are inserted into the mesh. If a good solution is found early, subtrees that provably cannot contain a better solution need not be explored. Hence, candidate tetrahedra are evaluated based on criteria used to determine how likely they are to be part of a good solution:

1. the number of faces shared with the boundary, since cavities with fewer boundary faces are generally easier to fill;

2. whether or not the candidate has a higher quality than the tetrahedra that have already been inserted into the mesh;

3. whether or not the candidate contains a new vertex, since any solution must contain all vertices present in the cavity.

Each tetrahedron is given a score based on the number of criteria that it meets. Candidates with a higher score are tested first.

2.4 Reusing results of geometric predicates

During the search, the orientations and qualities of many tetrahedra need to be evaluated. A robust algorithm requires these to be evaluated using adaptive precision in order to obtain consistent results despite numerical errors. As a result, these evaluations are computationally intensive. This effect is compounded by the need to evaluate the same tetrahedra many

times, if it is considered as a candidate at multiple occasions during the search.

Computing the qualities of all $\binom{n}{4}$ tetrahedra of an n-vertex point ahead of time would avoid repeated computations, but this solution is inadequate: the orientations and qualities of some tetrahedra are never needed. In many cases, the search ends early after only evaluating a small fraction of all possible tetrahedra.

Instead, our approach is to memoize the computation of quality values. To evaluate a tetrahedron T, it is first normalized as T' by sorting the indices of its four vertices. While sorting, the parity of the number of permutations that were performed is tracked. A table is then accessed to test whether or not the quality of T' is known. If not, it is computed and stored in the table. The quality of T is the same as that of T' for an even number of permutations, and the opposite otherwise.

In addition, if the tetrahedron intersects a constrained edge or fully encloses a vertex, a null quality is stored in the look-up table. This prevents the insertion of these tetrahedra into the mesh without requiring the reevaluation of the intersection tests.

2.5 Performance results

The SPR algorithm can be used with any chosen quality measure. For the purpose of this analysis we use

$$\gamma = \frac{\sqrt{24}\ 3V}{|e_{max}|(A_1 + A_2 + A_3 + A_4)} = \frac{\sqrt{24}\ r_{in}}{|e_{max}|}, \quad (1)$$

where V is the volume of the tetrahedron, $|e_{max}|$ is the length of the longest edge, A_i is the area of the ith face and r_{in} is the inradius of the tetrahedron. The factor $\sqrt{24}$ is added such that the optimal tetrahedron, which is a regular tetrahedron, has a quality $\gamma = 1$. This quality measure penalize all tetrahedra according to their associated interpolation error [2].

For each cavity featured in Figure 5, we measured the running time of the SPR operation (Table 1). All 5 cavities were extracted from real-world, non-optimized meshes. We measured the running time in two different settings:

1. with no initial lower bounds given to the algorithm, as is the case when SPR is used for boundary recovery and no initial triangulation is known;

2. with the quality of the initial triangulation γ_{orig} as a lower bound, as is done for mesh optimization.

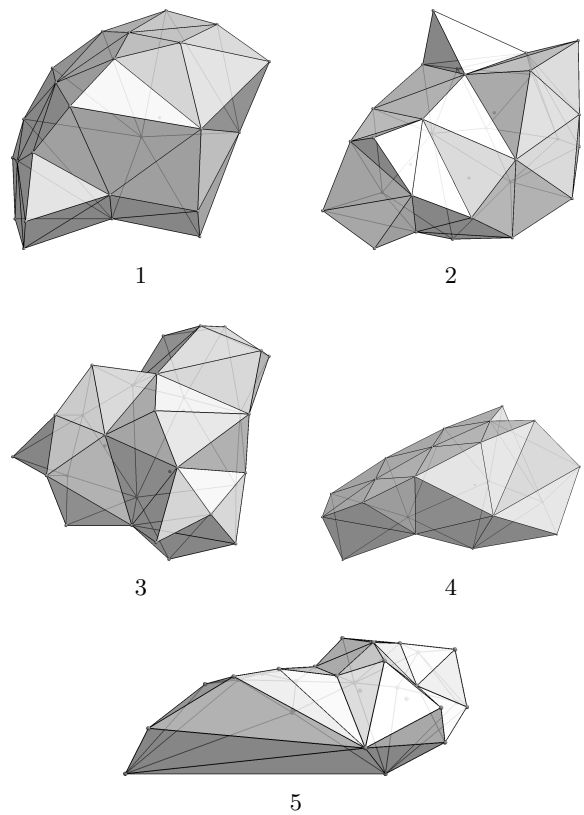


Figure 5: Different cavities on which the tests were conducted. № 1,3 and 4 were randomly extracted from the mesh improvement procedure of a torus or sphere. № 2 looks ordinary but is particularly slow without cleverly chosen heuristics, whereas № 5 comes from a mesh with bad elongated triangles on its surface.

The execution time in the second case is always strictly less than in the first, because the lower bound allows the algorithm to prune triangulations containing a tetrahedron with a quality worse than γ_{orig}. All cavities were optimized within 3.6 milliseconds when the lower bound was used, and within 13.3 seconds otherwise.

We measured the speedups offered by the different improvements to the algorithm by disabling each optimization independently:

1. the face selection heuristic (Section 2.2);

2. the candidate ordering heuristic (Section 2.3);

3. the reuse of previously computed qualities of tetrahedra (Section 2.4).

The combined effect of these improvements was measured by disabling all of them simultaneously. The improved algorithm is between 10^4 and 10^7 times faster

Cavity	V	$\lvert\partial C\rvert$	Quality		Time (ms)		Speedup			
			γ_{orig}	γ_{after}	$q(t) > 0$	$q(t) > \gamma_{\text{orig}}$	Section 2.2	Section 2.3	Section 2.4	Combined
1	25	44	0.28	0.52	3.1	2.5	25	1.4	0.68	4×10^4
2	31	52	0.31	0.53	13.3	3.0	3.5×10^4	3.1	5×10^4	3×10^6
3	31	52	0.25	0.51	9.6	3.6	10^6	10	2072	$> 10^7$
4	22	38	0.23	0.55	4.8	1.0	501	2.6	0.97	2×10^4
5	29	48	0.29925	0.29938	7.3	1.3	5×10^4	1.1	1.5	8×10^4

Table 1: Summary of the results obtained by optimizing the cavities of Figure 5 using SPR. Speedups are given for $q(t) > \gamma_{\text{orig}}$.

than a naive implementation. In the case of the third cavity, the execution with all optimization disabled was stopped after 20 hours, and no improved solution found (Table 1).

All experiments were performed on an Intel® Core™ i7-6700HQ CPU. Timings and speed ups are the average of 100 runs, or of two runs for entries that required more than one minute of computation time. Our implementation uses only a small amount of memory, although asymptotically proportional to n^4 were n is the number of points of the cavity. For the cavities tested, the maximum RAM usage did not exceed 3 MB.

3. APPLICATION TO MESH IMPROVEMENT

Computing the optimal triangulation of a cavity is an expensive operation. When optimizing large meshes, this operator should be used alongside other mesh improvement techniques in order to achieve good performances. The optimization framework we proposes combines SPR with mesh smoothing and edge-removal.

Mesh smoothing We use Laplacian smoothing, improved by a golden-section search on the segment between the original position and the average of neighboring vertices. This approach is effective in practice even though the objective function is not unimodal. Freitag et al. give a review of this combined Laplacian and optimization-based technique [4].

Edge removal The edge removal technique used here is based on tables containing all triangulations of the disk with up to seven points. We iterate through each triangulation in order to determine the most appropriate one.

Using SPR Throughout the optimization process, a list of low-quality tetrahedra is maintained. The threshold on the value of γ that determines what is a "low-quality" tetrahedron is given as a parameter to our program. A threshold of 0.5 was used for all the results given in this paper. We first attempt to improve each bad tetrahedron using mesh smoothing and edge removal, as these techniques are much faster than SPR. SPR is then used to optimize cavities that surround the low quality tetrahedra that remain after applying other optimization techniques.

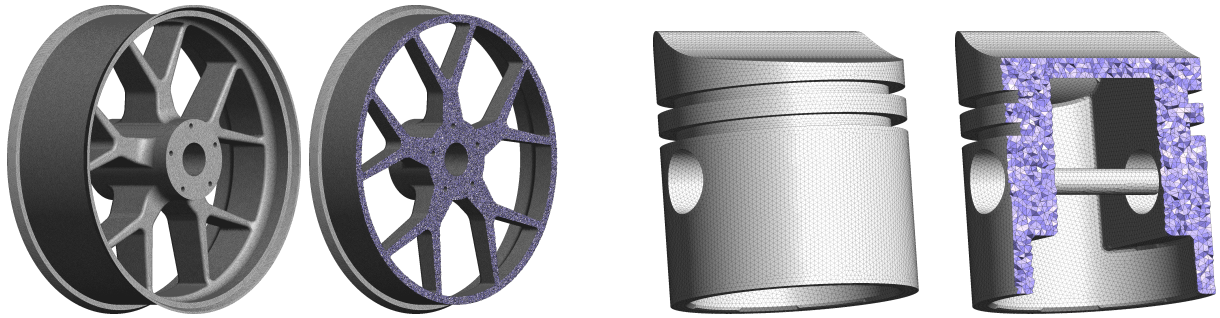
The cavity used to optimize a given tetrahedron is obtained by selecting all tetrahedra that contain at least one of its vertices[1]. There are about 50 tetrahedra in an average cavity (Figure 6a). To save time, the search only explores the first 2000 nodes of the search tree, since it is rarely beneficial to spend more time on individual cavities.

If the mesh quality is still insufficient after this step, the entire optimization process is repeated. Because of the modifications performed by applying the SPR operator, mesh smoothing and edge-removal may remove some of the remaining bad elements. This is done iteratively until the mesh can no longer be improved.

To avoid optimizing the same cavities multiple times, a modification flag is maintained for each tetrahedron. The SPR operator is only run on a cavity if at least one of its tetrahedra is marked as modified. The new tetrahedra obtained from any of the three optimization techniques are marked as modified. This modification flag is removed after running SPR if the solution has not improved.

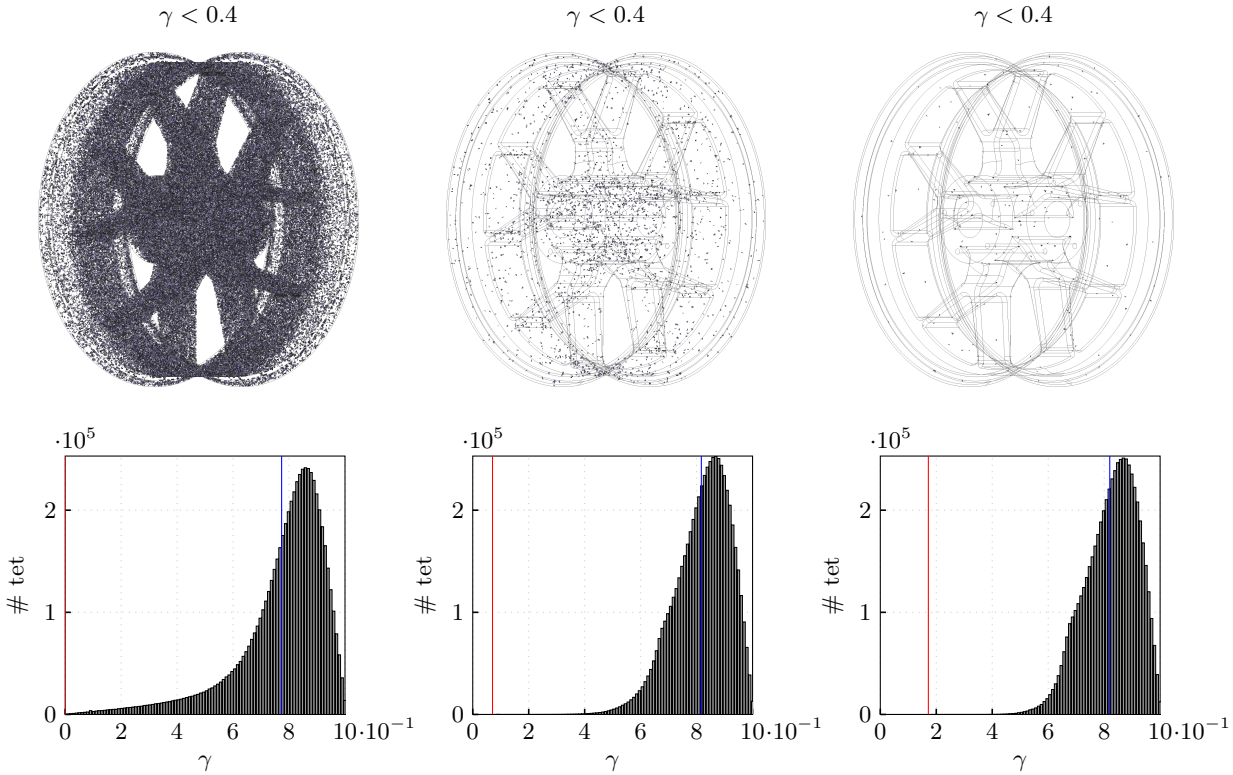
SPR efficiency SPR is approximately 1000× slower than edge removal and smoothing, as Table 2 shows. The first SPR pass has a success rate above 85%. In other words, the first time edge removal and smoothing failed to improve the mesh further, the SPR operation could improve 85% of bad tetrahedra. The efficiency of each operation decreases after each iteration of the mesh improvement process, until every operation fails for all bad tetrahedra. All iteration combined, SPR improved the mesh 63.1% of the times. This is particularly impressive considering that

[1] An other choice of cavity might be more efficient. We still need to investigate that.



(a) rim mesh: 6 285 719 tet. (b) piston mesh: 277 648 tet.

Figure 6: Input tetrahedral meshes, both generated with our in-house 3D constrained Delaunay. The geometries come from the CAD models `00000040` and `00009733` from the ABC data set [20]. The surface meshes were generated with `gmsh -2 -algo meshadapt -clcurv -clmax 1.5` and `gmsh -2 -algo frontal -clscale 0.1` [21] respectively.



(a) unimproved mesh (b) ours without SPR (c) ours

Figure 7: On the first row, tetrahedra with a quality $\gamma < 0.4$ are displayed for the rim mesh (Figure 6a). On the second row is an histogram of qualities present in the rim mesh. The red and blue vertical lines indicate the minimum and average qualities respectively. First column correspond to the input mesh, without any mesh improvement. For the second column, only edge removal and smoothing were applied. For the last column, edge removal, smoothing and SPR were applied.

	number of calls	total time	average time per call	success rate	cavity size (# tet)
smoothing	5519647	33.0	6×10^{-6}	2.5%	26.5
edge removal	6297417	6.3	1×10^{-6}	7.8%	4.9
SPR	103602	397.9	3.8×10^{-3}	63.1%	51.2

Table 2: Statistics on the mesh improvement procedure of the rim mesh (Figure 6a). The fourth column shows the proportion of calls that lead to a modification of the mesh. The last column shows the average number of tetrahedra in the cavity before the operation was executed.

range of γ	0 .. 0.2	0.2 .. 0.4	0.4 .. 0.6	0.6 .. 1	total	time [s]
unimproved mesh	64 097	197 909	523 801	5 499 912	6 285 719	0
ours without SPR	193	3794	172 435	5 726 705	5 903 127	23.4
ours	14	344	95 614	5 718 753	5 814 725	441.5
Stellar	–	–	–	–	–	> 72 000

Table 3: Quality distribution of elements in the rim mesh, together with timings of the mesh improvement procedure.

the SPR operation is only used when both edge removal and smoothing are totally ineffective. However, the SPR operation still account for 90.1% of the total time.

In term of quality improvements, the SPR operation helps removing most bad tetrahedra as can be seen on Figure 7. An implementation based only on edge removal and smoothing still has 3987 tetrahedra with a quality below 0.4 when optimizing the large rim mesh. When using SPR, this number falls to 358 (Table 3). For the smaller mesh of a piston (Figure 6b), the results are similar (Table 4).

Comparison to Stellar Stellar is a tetrahedral mesh improvement program, elaborated by Klingner and Shewchuk in 2009[2]. The different methods used within this program are summarized in their paper [22] and best detailed in Klingner's Ph.D. dissertation [23]. To our best knowledge, no other program based on local modifications consistently optimizes meshes as far as Stellar does.

Indeed, Stellar implements many more operations than our simple edge removal and improved Laplacian smoothing [23]. First, Stellar uses a more sophisticated smoothing algorithm based on nonsmooth optimizations introduced by Freitag and Ollivier-Gooch [7]. Second, in addition to edge removal, it implements the inverse operation called multi-face removal[3] and *edge-contraction*, also called *edge-collapse*, which is an operation that removes an edge from the mesh, replacing its two endpoints with a single vertex. The position of the remaining vertex is chosen using their advanced smoothing method. Finally, Stellar uses *vertex insertion* in combination with all other mesh optimizations techniques, forming a complex, huge composite operation. In comparison, our algorithm does neither add nor remove any vertex from the mesh. Thanks to edge-contraction and vertex insertion, Stellar gets significantly better results than our program for small meshes. However, while our simple mesh improvement procedure took 7 minutes to optimize the rim mesh shown in Figure 6a, Stellar was still in the first stages of its mesh improvement procedure after 20 hours. We were thus not able to improve large meshes with Stellar. Furthermore, the simplicity of our approach makes the code far simpler to maintain: our whole code is ≈ 2500 lines long while Stellar is closer to 25 000.

For our tests, we disabled all operations of Stellar that were modifying the surface mesh, leaving only the operations explained previously. We can compare the results for the smaller piston mesh on the histograms given on Figure 8 and with the data in Table 4. Stellar improves the mesh further but is slower than our program.

4. CONCLUSION

Thanks to a careful and efficient implementation, we have shown that the SPR operator can effectively be used to remove most of the low-quality tetrahedra that are left by other mesh improvement operations. Such an efficient implementation is important not only for better mesh improvement methods, but because it enables the use of SPR, a very flexible tool, for a wide range of other applications. Indeed, the SPR can replace any cavity-based topological transformation.

Its use for *boundary recovery* has already been studied in previous works [24, 25]. Theoretically, the SPR operation provides a strong guarantee: if a triangulation of a polyhedron contains the required triangles and edges, the SPR operation executed on this polyhedron

[2] `http://people.eecs.berkeley.edu/~jrs/stellar/`

[3] multi-face removal alone does not give Stellar any edge over our program because SPR supersedes it.

range of γ	0 .. 0.2	0.2 .. 0.4	0.4 .. 0.6	0.6 .. 1	total	time [s]
unimproved mesh	2758	8249	21 028	245 613	277 648	0
ours without SPR	1	170	7904	254 553	262 628	0.8
ours	0	27	6269	253 625	259 921	14.5
Stellar	0	10	820	257 193	258 023	263.9

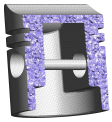
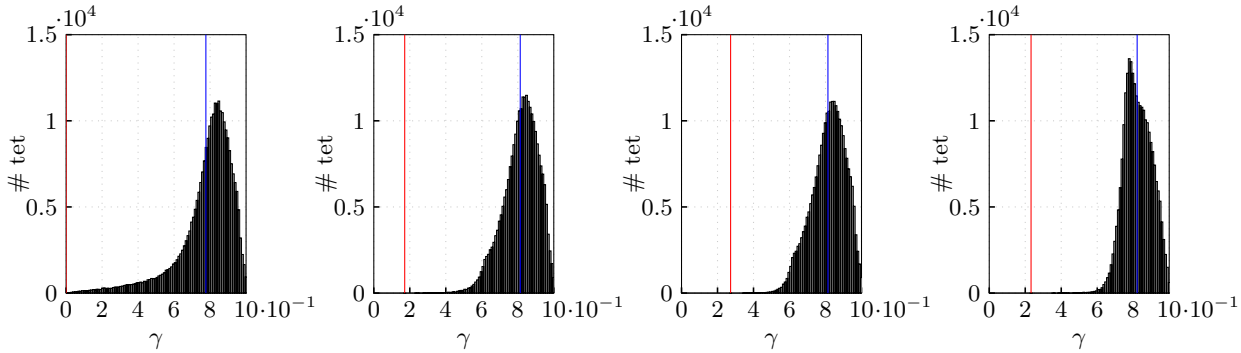


Table 4: Quality distribution of elements in the piston mesh, together with timings of the mesh improvement procedure.



(a) unimproved mesh (b) ours w/o SPR (c) ours (d) Stellar

Figure 8: Histogram of qualities present in the piston mesh. The red and blue vertical lines indicate the minimum and average qualities respectively.

will inevitably find it. Therefore, using the SPR with a large-enough cavity that contains the missing edges and triangles will allow to recover the right geometry most of the time. The rest of the time, the cavity is not meshable as such. The algorithm can add one or more Steiner points inside the polyhedron in order to make it meshable. The number and the position of Steiner points is a difficult subject. which was, in our opinion, best studied by Si and Gärtner [26]. It may also be possible to enlarge the cavity, widening the panel of possible triangulations in order to make the polyhedron meshable.

Another valuable use of the SPR operation concerns the removal of vertices. Such an operation is useful for adaptive meshing where parts of the mesh must be dynamically coarsened. Vertex removal can also be used to improve mesh quality, in the same way that vertex insertion is used [22]. SPR makes it easy to find the best triangulation to fill the cavity left after removing a vertex. By comparison, a previous solution generates a suboptimal triangulation by applying the ear-clipping algorithm to the same cavity [27].

A fast implementation of the SPR operator is needed in order to use these techniques in practical meshing software.

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement HEXTREME, ERC-2015-AdG-694020).

References

[1] Liu J., Sun S., Wang D. "Optimal Tetrahedralization for Small Polyhedron: A New Local Transformation Strategy for 3-D Mesh Generation and Mesh Improvement." p. 14, 2006

[2] Shewchuk J.R. "What Is a Good Linear Finite Element? - Interpolation, Conditioning, Anisotropy, and Quality Measures." Tech. rep., In Proc. of the 11th International Meshing Roundtable, 2002

[3] Amenta N., Bern M., Eppstein D. "Optimal Point Placement for Mesh Smoothing." *Journal of Algorithms*, vol. 30, no. 2, 302–322, Feb. 1999. URL `http://linkinghub.elsevier.com/retrieve/pii/S0196677498909841`

[4] Freitag L., Jones M., Plassmann P. "A Parallel Algorithm for Mesh Smoothing." *SIAM Journal on Scientific Computing*, vol. 20, no. 6, 2023–

2040, Jan. 1999. URL http://epubs.siam.org/doi/10.1137/S1064827597323208

[5] Freitag L.A., Knupp P.M. "Tetrahedral mesh improvement via optimization of the element condition number." *International Journal for Numerical Methods in Engineering*, vol. 53, no. 6, 1377–1391, Feb. 2002. URL http://doi.wiley.com/10.1002/nme.341

[6] Dassi F., Kamenski L., Farrell P., Si H. "Tetrahedral mesh improvement using moving mesh smoothing, lazy searching flips, and RBF surface reconstruction." *Computer-Aided Design*, vol. 103, 2–13, oct 2018. URL http://www.sciencedirect.com/science/article/pii/S0010448517302336

[7] Freitag L.A., Ollivier-Gooch C. "Tetrahedral mesh improvement using swapping and smoothing." *International Journal for Numerical Methods in Engineering*, vol. 40, no. 21, 3979–4002, Nov. 1997. URL http://doi.wiley.com/10.1002/%28SICI%291097-0207%2819971115%2940%3A21%3C3979%3A%3AAID-NME251%3E3.0.CO%3B2-9

[8] Huang W., Ren Y., Russell R. "Moving Mesh Partial Differential Equations (MMPDES) Based on the Equidistribution Principle." *SIAM Journal on Numerical Analysis*, vol. 31, no. 3, 709–730, Jun. 1994. URL https://epubs.siam.org/doi/abs/10.1137/0731038

[9] Du Q., Wang D. "Tetrahedral mesh generation and optimization based on centroidal Voronoi tessellations." *International Journal for Numerical Methods in Engineering*, vol. 56, no. 9, 1355–1373, mar 2003. URL http://doi.wiley.com/10.1002/nme.616

[10] Si H. "TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator." *ACM Trans. Math. Softw.*, vol. 41, no. 2, 11:1–11:36, feb 2015. URL http://doi.acm.org/10.1145/2629697

[11] de L'isle E.B., George P.L. "Optimization of Tetrahedral Meshes." I. Babuska, W.D. Henshaw, J.E. Oliger, J.E. Flaherty, J.E. Hopcroft, T. Tezduyar, editors, *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, The IMA Volumes in Mathematics and its Applications, pp. 97–127. Springer New York, 1995

[12] Shewchuk J.R. "Two Discrete Optimization Algorithms for the Topological Improvement of Tetrahedral Meshes." p. 11

[13] Misztal M.K., Bærentzen J.A., Anton F., Erleben K. "Tetrahedral Mesh Improvement Using Multi-face Retriangulation." B.W. Clark, editor, *Proceedings of the 18th International Meshing Roundtable*, pp. 539–555. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. URL http://link.springer.com/10.1007/978-3-642-04319-2_31

[14] Boissonnat J.D., Devillers O., Pion S., Teillaud M., Yvinec M. "Triangulations in CGAL." *Computational Geometry*, vol. 22, no. 1, 5–19, may 2002. URL http://www.sciencedirect.com/science/article/pii/S0925772101000542

[15] Jamin C., Alliez P., Yvinec M., Boissonnat J.D. "CGALmesh: A Generic Framework for Delaunay Mesh Generation." *ACM Transactions on Mathematical Software*, vol. 41, no. 4, 1–24, Oct. 2015. URL http://dl.acm.org/citation.cfm?doid=2835205.2699463

[16] Dobrzynski C. "MMG3D: User Guide." Technical Report RT-0422, INRIA, mar 2012. URL https://hal.inria.fr/hal-00681813

[17] Marot C., Pellerin J., Remacle J.F. "One machine, one minute, three billion tetrahedra." *International Journal for Numerical Methods in Engineering*, vol. 117, no. 9, 967–990, 2019. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.5987

[18] Gent I.P., Walsh T. "Easy Problems are Sometimes Hard." *Artif. Intell.*, vol. 70, no. 1-2, 335–345, 1994. URL https://doi.org/10.1016/0004-3702(94)90109-0

[19] Gomes C.P., Selman B., Crato N. "Heavy-Tailed Distributions in Combinatorial Search." *Principles and Practice of Constraint Programming - CP97, Third International Conference, Linz, Austria, October 29 - November 1, 1997, Proceedings*, pp. 121–135. 1997. URL https://doi.org/10.1007/BFb0017434

[20] Koch S., Matveev A., Jiang Z., Williams F., Artemov A., Burnaev E., Alexa M., Zorin D., Panozzo D. "ABC: A Big CAD Model Dataset For Geometric Deep Learning." *arXiv:1812.06216 [cs]*, dec 2018. URL http://arxiv.org/abs/1812.06216. ArXiv: 1812.06216

[21] Geuzaine C., Remacle J.F. "Gmsh: a three-dimensional finite element mesh generator with built-in pre facilities." p. 24, 2009

[22] Klingner B.M., Shewchuk J.R. "Aggressive Tetrahedral Mesh Improvement." M.L. Brewer, D. Marcum, editors, *Proceedings of the 16th*

International Meshing Roundtable, pp. 3–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. URL http://link.springer.com/10.1007/978-3-540-75103-8_1

[23] Klingner B.M. *Improving Tetrahedral Meshes*. Ph.D. thesis, EECS Department, University of California, Berkeley, Nov 2008. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-145.html

[24] Liu J., Chen B., Chen Y. "Boundary recovery after 3D Delaunay tetrahedralization without adding extra nodes." *International Journal for Numerical Methods in Engineering*, vol. 72, no. 6, 744–756, 2007. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2044

[25] Chen J., Zhao D., Huang Z., Zheng Y., Gao S. "Three-dimensional constrained boundary recovery with an enhanced Steiner point suppression procedure." *Computers & Structures*, vol. 89, no. 5-6, 455–466, Mar. 2011. URL https://linkinghub.elsevier.com/retrieve/pii/S0045794910002828

[26] Si H., Gärtner K. "3D boundary recovery by constrained Delaunay tetrahedralization." *International Journal for Numerical Methods in Engineering*, vol. 85, no. 11, 1341–1364, Mar. 2011. URL http://doi.wiley.com/10.1002/nme.3016

[27] Devillers O., Teillaud M. "Perturbations and Vertex Removal in a 3D Delaunay Triangulation." 2003. URL https://hal.inria.fr/inria-00166710