Exploring polyhedral mesh generation from Delaunay tetrahedral meshes

Sergio Salinas-Fernández* Magdalena Alvarez† Nancy Hitschfeld-Kahler‡

Abstract

In this paper, we present the conceptual framework and a new algorithm for building polyhedral cells by the union of tetrahedrons. Two tetrahedrons are joined if the shared face fulfills some criteria. As preliminary results, statistics of the polyhedral cells generated by using the face area and the face in-circle radius metrics as joining criteria, are shown and discussed.

1 Introduction

Due to the requirements to count with polyhedral mesh generators that allow researchers to evaluate how general polyhedral cells can be and still give accurate simulation results in VEM [1], we present a first attempt to study the kind of polyhedra that can be generated by joining tetrahedra from a tetrahedral mesh. We present an algorithm and describe the properties of the built polyhedral cells. The preliminary experimental evaluation shows that the generated polyhedra are very far from convex cells and need to be further study too see if these kind of cells are useful and provide some advantages in the context of some specific simulation in comparison to Voronoi cells. A mesh composed of proper non-convex polyhedral cells should require less cells and points to model complex geometries than meshes composed of convex cells.

2 Basic concepts

We want to extend the concept of the terminal-edge region, defined in [3], to faces in 3D. One of the problems of doing this extension, is what is "the largest face of a tetrahedron". There are several criteria to define which face of a tetrahedron is the largest or the smallest, we are going to call them "Joining criteria". Given a tetrahedral mesh $\tau = (V, E, F)$, we can define:

DEFINITION 1. ***Joining criterion*** *For any tetrahedron t_i, the Joining criterion is a metric used to rank the faces of t_i ordered from the largest to the smallest. There is only one largest-face; if two faces have the same size, then one of them is chosen as the largest arbitrarily.*

Examples of joining criteria for a tetrahedron $t_i \in \tau$ include the area criterion, where the largest and smallest faces are determined by the maximum and minimum areas of the t_i faces, respectively, and the in-circle radius criterion, where the largest and smallest faces are identified based on the maximum and minimum in-circle radius of the t_i faces. With this already defined, we can extend the concept of terminal-edge and longest-edge propagation path [2] to faces in a tetrahedral mesh τ with a Joining criterion J:

DEFINITION 2. ***Terminal-face*** *A face is a terminal-face f_i if two adjacent tetrahedrons t_a, t_b to f_i share their respective (common) largest-face, according to J. This means that f_i is the largest-face of both tetrahedrons that share f_i. If $t_b = \emptyset$, then f_i is called border terminal-face.*

DEFINITION 3. ***Largest-face propagation path (Lfpp)*** *For any tetrahedron t_0 of any tetrahedralization τ, the Largest-Face Propagation Path of t_0 ($Lfpp(t_0)$) is the ordered list of all the tetrahedrons $t_0, t_1, t_2, ..., t_{n-1}$, such that t_i is the neighbor tetrahedron of t_{i-1} by the largest face of t_{i-1}, for $i = 1, 2, ..., n-1$.*

DEFINITION 4. ***Terminal-face region*** *A* terminal-face region *R is a region formed by the union of all tetrahedrons t_i such that $Lfpp(t_i)$ has the same terminal-face. Figure 1 displays an example of a terminal-face region.*

In order to accelerate the generation of terminal-face regions we develop a classification system for each face of τ. For each face $f_i \in \tau$, a face can be a frontier-face or internal-face.

DEFINITION 5. ***Frontier-face*** *A frontier-face f_i is a face that is shared by two tetrahedron t_1, t_2, each one belonging to a different terminal-face region, that means that f_i is not the largest-face of neither t_1 nor t_2. If $t_2 = \emptyset$ then f_i is a frontier-face even if ef_i is a border terminal-face.*

*ssalinas@dcc.uchile.cl. DCC, Universidad de Chile, Chile.

†magdalena.alvarez@ug.uchile.cl. DCC, Universidad de Chile, Chile.

‡nancy@dcc.uchile.cl. DCC, Universidad de Chile, Chile.

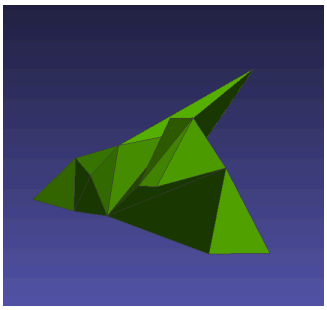


Figure 1: Terminal-face region generated with the incircle criterion

DEFINITION 6. ***Internal-face*** *A internal-face f_i is a face that is shared by two tetrahedrons t_1, t_2, each one belonging to the same terminal-face region. In other words, f_i is an internal-face if f_i is neither a terminal-face nor frontier-face.*

For the context of this work, we only need the frontier-faces, as those faces will be faces of the polyhedral mesh at the end of the algorithm, internal-faces will be removed during the process of joining tetrahedrons.

One property that terminal-face regions must obey, for their use as polyhedrons in a polyhedral mesh, is that they must not overlap.

LEMMA 2.1. *Let τ be a tetrahedral mesh of any set of points P with a Joining criterion J. Then the set of terminal-face regions in τ do not overlap.*

As in 2D, terminal-edge regions can contain frontier-edges in its interior, in 3D, terminal-face regions can contain frontier-faces in their interior. We will refer to this kind of frontier faces as barrier-faces.

DEFINITION 7. ***Barrier-face*** *Given a terminal-face region R_i, any frontier-face $f \in R_i$ that is not part of the boundary δR_i is called a barrier-face.*

DEFINITION 8. ***Barrier-edge tip*** *A barrier tip in a terminal-face region R_i is an edge incident to only one frontier-face (particularly a barrier-face), and the rest of the faces are internal-faces.*

3 Algorithm

In this section, we present our algorithm to generate polyhedral meshes. The algorithm takes as input a tetrahedral mesh $\tau = (V, E, F)$, a Joining criterion J, and return a polyhedral mesh $\tau' = (V, E', F')$. The algorithm has 3 main phases: Label phase, Traversal phase and Repair phase.

For the understanding of the algorithm, we are going to assume that we have a data structure with all geometrical information of the tetrahedral mesh. For the output we use a list of polyhedrons, and each polyhedron is represented as a list of faces.

3.1 Label phase The first step to generate the new polyhedral mesh is to define whose faces are going to be faces of the output mesh (i.e. the frontier-faces), and choose one tetrahedron t_i per terminal-face region R_i, to be used in the Traversal phase to generate the new polyhedron, we are going to name to t_i as seed tetrahedron, and it is a tetrahedron adjacent to a terminal-face. For this reason, as equal as the 2D algorithm, we first with the Label phase.

The Label phase takes as input the mesh $\tau = (V, E, F)$ and outputs two auxiliary arrays containing mesh information: the `Seed array`, which is the size of the number of tetrahedrons and stores the indices of all seed tetrahedrons in τ, and the `Frontier-face bitvectors`, an array of size $|F|$ where each element i is set to true if the face f_i is a frontier-face and false otherwise. Those arrays will be use in the Traversal phase and the Repair phase.

The algorithm first calculates the faces that meet the joining criterion J of each tetrahedron $t_i \in \tau$. Afterwards, the algorithm labels the seed tetrahedrons, those are the tetrahedrons adjacent to a terminal-face, and that are used in the traversal phase to generate the polyhedrons. Finally, the algorithm labels the frontier-faces, those are the faces of the final mesh τ'. For each face $f_i \in \tau$, the algorithm gets the both tetrahedrons t_i, t_j, that shares f_i, if f_i is not the largest face of neither t_i nor t_j, or if f_i is a border face, then f_i is labeled as frontier-edge (sets as true) in the `frontier_Bitvector`. This process is exemplified in Figure 2.

With the tetrahedrons and faces already labeled, the algorithm continues to the Traversal phase.

3.2 Traversal phase In this phase the algorithm converts terminal-face regions into polyhedral cells. To do this, starting from a seed tetrahedron $t_i \in$ `seed_array`, the algorithm build the polyhedron P by calling adepth first search (DFS) algorithm. This algorithm travels inside the terminal-face region using the faces of the seed tetrahedron t_i. For each tetrahedron t_j adjacent to t_i by its face, the algorithm checks if t_j contains a frontier-face f_i, if it is true, then f_i is stored in P, as part of the polyhedron, and if its is not the case, then f_i is a internal-face. Consequently, the DFS travel to the neighbors of t_j looking for others frontier-faces.

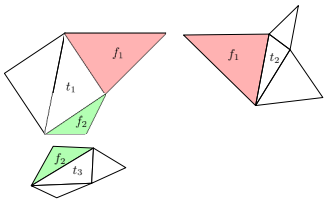


Figure 2: Example of labeling with adjacent 3 tetrahedrons, Red faces are the terminal-faces, and green faces are frontier-faces. Tetrahedron t_1 is connected to t_2 by face f_1, and tetrahedron t_1 is connected to t_3 by face f_2. According to the Joining criterion of the largest area, f_1 is the largest face of t_1 and t_2, meaning that f_1 is a terminal-face, thus t_1 is chosen as a seed tetrahedron. f_2 is not the largest face of t_1 and t_3, thus f_2 is labeled as a frontier-face.

For each P generated from the DFS, the algorithm checks for barrier edge tips. The algorithm counts the number of repeated faces in P, if there are repeated faces, it means that a face was stored two times during the DFS, indicating a barrier-face. In such cases, P is sent to the Repair phase.

3.3 Repair phase For any not simple polyhedron P_i, the algorithm uses the barrier edge tips to split a polyhedron in two. A barrier tips is an edge $e_i \in P_i$ that is adjacent to only one frontier-face of P_i, and the rest of faces adjacent to e_i are internal-faces. The first question to answer is how to know if edge e_i is a barrier edge tip.

THEOREM 3.1. *Given a terminal-edge region R_i, with F_f the set of frontier-faces of R_i, an edge e belonging to a barrier-face of R_i, and the set F_e of faces incident to e., e is a barrier tip if $||F_e \cap F_f| = 1$.*

Using Theorem 3.1 we define the following algorithm to get a list B_p with all the barrier edge tips of polyhedron P. The algorithm takes all edges e_i of the barrier-faces in P, and iterates over all them, to check if they are a barrier tip. The check is done by calculating, the formula of above, if it is true, then there is only a barrier-face adjacent to e_i, thus e_i is barrier tip.

Once the algorithm has computed the set of barrier tips B, we can use them to split the polyhedron P. This split consists of converting internal-faces f_i to frontier-faces, and using the two tetrahedra adjacent to f_i as seeds to repeat the traversal phase.

The algorithm of the repair phase first defines a `subseed list` L_p to store the seed tetrahedra that will be used as seeds to generate the new polyhedra. And the `usage bitarray` A that is used as a flag to check if a seed tetrahedron has been used during the creation of a new polyhedron, thus the algorithm can avoid creating duplicate polyhedra.

Then the algorithm iterates over all the barrier edge tips $b_i \in B$. For each b_i, the algorithm selects the barrier-face f_i incident to b_i, circles around the internal faces of b_i, and stores them in order of appearance in a sublist l. The middle internal face f_m of l is calculated. f_m is converted to a frontier-face by setting `frontier_bitvector`$[f_m]$ = `True`. The two tetrahedron t_1 and t_2 adjacent to f_m are stored in the list L_p to be used as seed tetrahedra, and they are also marked as `True` in the `usage bitarray` A.

Later the algorithm constructs the polyhedron. For each tetrahedron $t_i \in L_p$, the algorithm checks if t_i has been used during the generation of a tetrahedron. If this is not the case, then the algorithm proceeds to generate a new polyhedron P' by calling the traversal phase. However, for each tetrahedron t_j visited in the traversal phase, A$[f_m]$ is set to `False` to avoid using t_j to generate the same polyhedron P' again. This process is repeated until there are no more seed tetrahedra in L_p, at which point all the new polyhedrons are simple polyhedrons, and are added to τ'.

Finally, τ' is a mesh composed of simple polyhedra. Notice that a more simple strategy to generate simple polyhedra is just to eliminate the barrier faces because they are internal faces and do not represent geometrical aspects to be respected from the input domain.

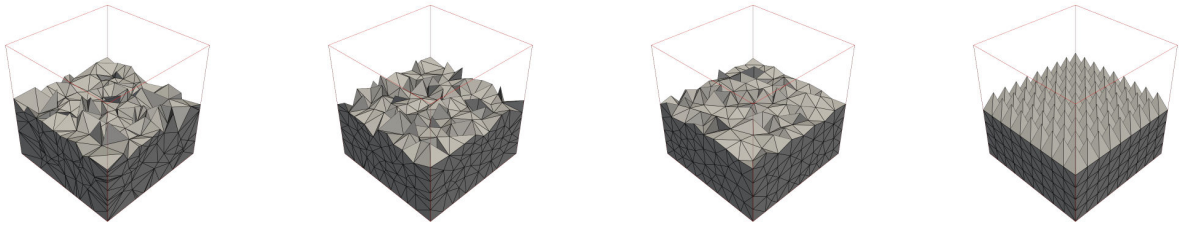
4 Experiments

The experiments consist in testing the algorithm with 4 different kinds of tetrahedral meshes in a cube: Random, Poisson, Grid, and Quality meshes. Examples of those meshes can be seen in 3.

For each kind of mesh, we generate 5 different sizes, with 500, 1000, 5000, 10000 and 50000 vertices. The summary of the experiments can be seen in Table 1. In general, regardless of the point distribution of the meshes, the algorithm merges, on average, 3 tetrahedron per polygon, reducing the tetrahedron mesh by around 70% of the elements, and about one-fifth of the polyhedrons contain barrier faces. This is even true for points not in general position, as it is the case of Grid meshes.

We can see differences between the two joining criteria; the maximum face in-circle radius criterion tends to join more tetrahedrons than the maximum face area criterion. The resulting polyhedral cells include less barrier-edge tips, but the generated meshes contain in average more tetrahedrons.

The unique statistic that seems to vary depending on the point distribution is the percentage of polyhe-



(a) Random mesh (b) Poisson mesh (c) Quality mesh (d) Grid mesh

Figure 3: Example of the meshes generated for the experiments, all meshes have near 5000 vertices. The cubes were cut by a plane to show the interior of the mesh.

	In-circle criterion					Area max criterion				
	Random	Poisson	Quality	Grid	Total	Random	Poisson	Quality	Grid	Total
Reduction	70.2%	66.4%	66.6%	66.4%	67.4	70.7%	68.8%	68.3%	70.1%	69.5%
Avg Tetras	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.2	3.0
Barriers	25.0%	25.6%	24.9%	8.5%	21.0%	21.1%	18.8%	19.4%	21.1	20.1%
Tetrahedrons	24.8	23.5	22.0	2.3	18.1	18.6	16.0	17.6	21.9	18.5

Table 1: Summary of the results from the joining criteria In-circle and max area. The "Reduction" row shows the percentage of tetrahedrons removed by the algorithm. The "Avg tetras" row indicates the average number of tetrahedrons contained within each polyhedron. The "Barrier" row displays the percentage of polyhedrons containing barrier faces. And lastly, the "Tetrahedrons" row presents the total number of tetrahedrons remaining in the mesh. For each row, the colored numbers compare the maximum, in red, and minimum, in blue, value obtained in each distribution experiment.

drons that remain as tetrahedrons after the algorithm was applied. Poisson's meshes have the lowest percentage of tetrahedrons, while grid meshes have the highest. However, on average, all meshes retain approximately 18.6% of their elements as tetrahedrons.

5 Conclusions and future work

In this paper, we have presented the theoretical concepts and a preliminary version of an algorithm designed to convert a tetrahedral mesh into a polyhedral mesh. We have introduced the concept of terminal-face region in order to guide the polyhedral construction. Preliminary results show that the algorithm reduces the number of polyhedrons by approximately 70%, and that the average number of tetrahedrons per polyhedron is 3. The algorithm also retains approximately 20% of the tetrahedrons in the polyhedral mesh. Those results seem to be almost the same, independent of the chosen joining criterion. This means that the concept of terminal-face region as defined in this paper needs to be improved by considering joining strategies that also considers quality criteria of the generated polyhedron. Our ongoing work is taken in consideration of the latest findings. Moreover we have to test if these polyhedral cells are useful in the context of numerical simulations using the Virtual element method simulations [4].

6 Acknowledgments

This research is supported by Fondecyt Regular Project 1241596.

References

[1] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. Marini, and A. Russo, *Basic principles of virtual element methods*, Mathematical Models and Methods in Applied Sciences, 23 (2013), pp. 199–214.

[2] M.-C. Rivara, *New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations*, International Journal for Numerical Methods in Engineering, 40 (1997), pp. 3313–3324.

[3] S. Salinas-Fernández, N. Hitschfeld-Kahler, A. Ortiz-Bernardin, and H. Si, *POLYLLA: polygonal meshing algorithm based on terminal-edge regions*, Engineering with Computers, 38 (2022), pp. 4545–4567.

[4] T. Sorgente, S. Biasotti, G. Manzini, and M. Spagnuolo, *A survey of indicators for mesh quality assessment*, Computer Graphics Forum, 42 (2023), pp. 461–483.