Engineering with Computers manuscript No.
(will be inserted by the editor)

Polygon meshing algorithm based on terminal-edge regions

Sergio Salinas · Nancy Hitschfeld-Kahler · Hang Si · Alejandro Ortiz-Bernardin

Received: date / Accepted: date

Abstract This paper is a study of a new kind of polygon mesh obtained from a Delaunay triangulation using the concept of terminal-edge region. An algorithm to generate those meshes is proposed. The algorithm is divided into three phases and takes as input a Delaunay triangulation. The first phase is to label each edge and triangle of the input triangulation; the second phase is to build polygons (simple or not) from terminal-edges region using the label system; and the third phase is to transform each non simple polygons into simple ones. The final mesh contains polygons with convex and non convex shape. Due to the similarities with the Voronoi diagram, this paper also explores empirical properties and comparisons between our proposed meshes and Clipped Voronoi meshes are studied. Finally, we validate these new terminal-edge based polygon meshes by solving a Laplace equation on an L-shaped domain using the Virtual Element Method (VEM) and demonstrate the optimal convergence rate of the numerical solution.

Keywords First keyword · Second keyword · More

1 Introduction

Meshes based on triangles and quadrilaterals are common in simulations using the Finite Element Method (FEM). The problem is that polygons(elements) in FEM need to obey specific quality criteria, to avoid angles that are too large or small, or with sides of graded length (aspect ratio criteria), etc. To fulfill these criteria, sometimes the insertion of a large number of points and elements is required in order to properly model a domain, increasing the time needed to make the simulation. New methods as Virtual Element Method (VEM) [25,26] can use any polygon

Sergio Salinas
University of Chile
E-mail: ssalinas@dcc.uchile.cl

Nancy Hitschfeld
University of Chile
E-mail: nancy@dcc.uchile.cl

Alejandro Ortiz-Bernardin
University of Chile
E-mail: aortizb@uchile.cl

Hang Si
Weierstrass Institute for Applied Analysis and Stochastics (WIAS)
E-mail: si@wias-berlin.de

as basic cell. So, (i) the domain geometry can be fitted using less elements than if only triangles and quadrilaterals are used, (ii) the required point density distribution is just that required by the simulation problem, and (iii) it should not be necessary to further improve the quality of the elements. We are currently researching how far the VEM can allow the simulation of more complex problems, in both 2D and 3D, in comparison to FEM [28].

In this paper we propose an algorithm to generate meshes with polygons of arbitrary shape (convex and non-convex) based on the label system described in [12]. These meshes might contain non-simple polygons, so we also propose an algorithm to repair them. We experiment to show the properties of these polygons, compare these meshes with Clipped Voronoi meshes, and we validate our mesh to shows that it works with the VEM. The algorithm needs an initial triangulation as input, therefore we choose uses the Delaunay triangulation as input, as Delaunay triangulation is the triangulation with the maximal angles, then the resulting mesh has the maximal angles too.

Our motivation is the generation of meshes that adapt to a geometric domain using polygons of any shape, but respecting the required point density. To do this, we propose to use the concept of terminal-edge region. Our main research questions are: Can terminal-edge regions be adapted to be used as good basic cells for polygon numerical methods such as VEM? Do these kinds of meshes need less polygons to model the same problem than polygon meshes based on the Voronoi diagram?

Thus, the main contributions of this paper are: A simple and automatic way to generate polygon meshes of arbitrary shape, without adding additional points to the initial input, and a evaluation of our meshes in tested simulations with the VEM.

This paper is organized as follows: Section 2 shows the state of art; Section 3 introduces the basic concepts to understand the algorithm; in Section 4 there is a description of the algorithm divided in three main phases and the data structure used in its implementation in C++; Section 5 shows experiments of the algorithm to do a comparison with Clipped Voronoi meshes; Section 6 shows a assessment of the meshes in the Virtual Element Method (VEM) and Section 7 explains the conclusions and ongoing work.

2 Related work

To our knowledge, there are no algorithms to automatized generation of 2D meshes with arbitrary shape for VEM. There have been interesting methods of generating meshes [22], where a packing-based mesh was proved functional to work with VEM.

In the case of the standard finite element method, the most used 2D meshes are triangulations [4,16,18] or quadrilateral meshes [3,9,14]. 2D Mixed meshes composed of triangles and quadrilaterals have also been used, but are not so common [6] as the previous ones. Other methods use the Voronoi diagram as the polygon/polyhedron mesh, and the Voronoi cells as mesh elements are presented in [31,5,19].

The generalization of finite element methods to include polygons/polyhedra as part of the mesh elements was started in the last decade [20,21,11]. Polygonal elements are usually generated by using a quadtree approach and from Voronoi cells. The VEM was introduced seven years ago [25,26] and since then, several research groups have been developing computational frameworks for using the VEM, both in 2D and 3D, in order to explore how far the VEM can be used to solve new problems. To mention some examples, the VEM has been formulated and applied to solve linear elastic and inelastic solid mechanics problems [27], in fluid mechanics [24], in the optimization of a fluid problem through a discrete network [2], for compressible and in-

compressible nonlinear elasticity [30], for finite elasto-plastic deformations [29] and brittle crack propagation [8].

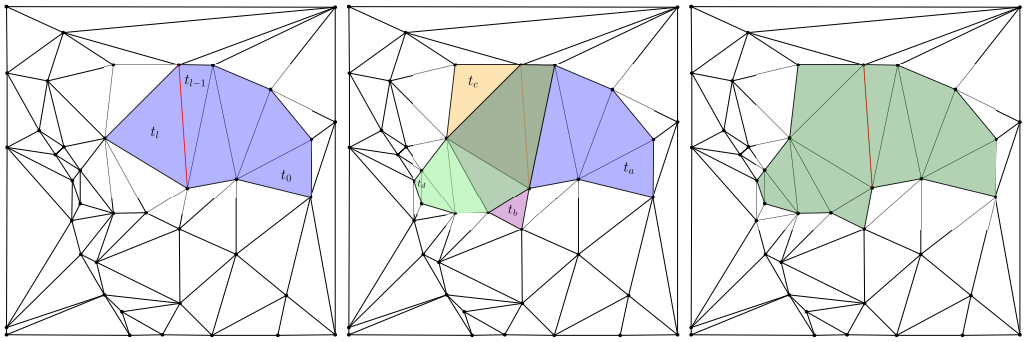
3 Basic concepts

The proposed polygon meshing algorithm is based on two concepts: Longest-edge propagation path (Lepp) introduced in [15] and terminal-edge regions defined in [1]. These concepts and some related properties are briefly reviewed in this section.

3.1 Terminal-edge regions

In any triangulation, triangles can be grouped under the Longest-edge propagation path concept defined as follows:

Definition 1 Longest-edge propagation path [15]: For any triangle t_0 in any triangulation Ω, the Lepp(t_0) is the ordered list of all the triangles t_0, t_1, t_2, ..., t_{l-1}, t_l, such that t_i is the neighbor triangle of t_{i-1} by the longest-edge of t_{i-1}, for $i = 1, 2, \ldots, l$. The longest-edge shared by t_{l-1} and t_l is a terminal-edge and t_{l-1} and t_l are terminal-triangles. If the longest-edge An example of the lepp of a triangle is shown in Figure 1(a).



(a) Lepp example (b) Multiple Lepp with same terminal-edge (c) Terminal-edge region

Fig. 1: Generation of terminal-edge regions **(a)** Longest-edge propagation of t_0, red line is a terminal-edge. **(b)** Four diferents lepp, Lepp(t_a), Lepp(t_b), Lepp(t_c) and Lepp(t_d), with the same terminal-edge. **(c)** Union of Lepp(t_a),..., Lepp(t_d) to generate a terminal-edge region.

Moreover, triangle edges can be classified according to their edge lengths inside the two triangles that share them. Therefore, given an edge e and two triangles t_1, t_2 that share e, we can label e as:

- Terminal-edge [15], if e is the longest-edge of t_1 and t_2.
- Frontier-edge [1], if e is neither the longest-edge of t_1 nor t_2.
- Internal-edge, if e is the longest edge of t_1 but not of t_2 or vice-versa.

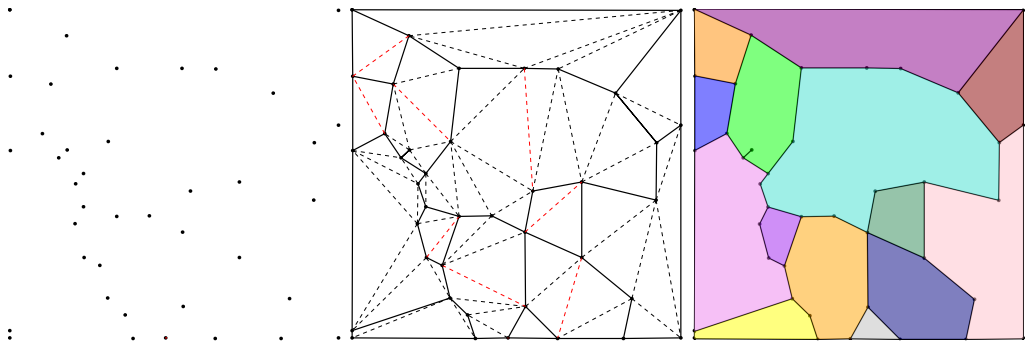
In case of equilateral or Isosceles triangles, one edge is chosen arbitrary as the longest-edge. Boundary edges, by simplicity, are considered as frontier-edge.

Definition 2 Terminal-edge region [1]: A *terminal-edge region* R is a region formed by the union of all triangles t_i such that Lepp(t_i) has the same terminal-edge. In case that the terminal-edge region is delimited by a boundary-edge the region will be called *boundary terminal-edge region*. Figure 1 shows the generation of terminal-edge region. Figure 1(b) shows four lepp with same terminal-edge, the union of those lepp generates a terminal-edge region in Figure 1(c).

Terminal-edge regions have some properties already proven. Important properties for the understanding of the algorithm proposed below, are:

- Terminal-edge regions are surrounded by frontier-edges [1].
- Terminal-edge regions cover the whole domain without overlapping [1] [12].
- Terminal-edge-regions might include frontier-edges in their interior. We have called this kind of frontier-edge a **Barrier-edge** [1] [12].

Figure 2 shows those properties. Figure 2(a) shows a set of points; Figure 2(b) shows the Delaunay triangulation of this point set when terminal-edges are drawn using red dashed lines: internal-edges using black dashed lines and frontier-edges using solid lines and Figure 2(c) shows each terminal-edge region using a different color. We observe in Figure 2(c) terminal-edge regions are enclosed by frontier-edges and the green region includes a barrier-edge.



(a) Initial point set (b) Delaunay triangulation (c) Terminal-edge regions

Fig. 2: (a) Initial random point set. (b) Delaunay Triangulation: Solid lines are frontier-edges, dashed black lines are internal-edges and red dashed edges are terminal-edges. (c) Terminal-edge regions

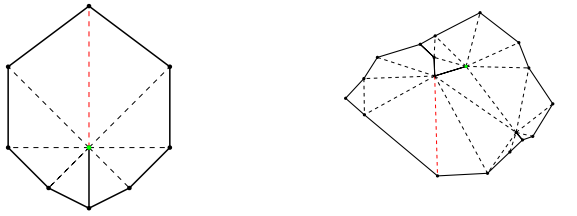
3.2 Terminal-edge regions as polygons

Because one of their proprieties of terminal-edge regions is that they covert a geometric domain without overlapping, means that they can be used to generate polygon meshes, representing each terminal-edge regions as polygons, simple and non-simple polygon. Non-simple polygons appear when terminal-edge regions include barrier-edges. As observed in Figure 2, the domain can be tessellated into 14 polygons where only the green region is represented by a non-simple polygon: it includes one barrier-edge.
For simplicity, in the case e is a domain boundary edge, e will be considered a frontier-edge too (see the edges belonging to the square in Figure 2(b) and (c)).

In order to build a conforming polygon tessellation from the partition generated from terminal-edge regions, non-simple polygons must be divided into simple polygons. That requires the integration of barrier-edges as part of the boundary of new simple polygons. Since this requirement is part of the meshing algorithm we are proposing in Section 4, we need, beforehand, to introduce some definitions and prove some properties that will sustain the construction of the algorithm.

Definition 3 Barrier-edge tip: A barrier-edge tip in a terminal-edge region R is a barrier-edge endpoint shared by no other barrier-edge

In Figure 3 we can see two polygons with barrier-edges, barrier-edge tips are shown in the color green. Figure 3(a) shows a basic case of polygon with barrier-edge tips and Figure 3(b) a case with two barrier-edge tips, there is no limit to the maximum number of barrier-edges for a polygon. Each point of the input is an endpoint of a frontier-edge or barrier-edge. Terminal-edge regions have none isolated interior points. Barrier-edge tips belong to a barrier-edge. Lemma 1 demonstrates this property.



(a) One barrier-edge and one tip

(b) Four barrier-edges and two tips tip

Fig. 3: Examples of non-simple polygons with barrier-edges. Black lines are frontier-edges, dashed black lines are internal-edges and red edges are terminal-edges. **(a)** One barrier-edge tip **(b)** Two barrier-edges tips.

Lemma 1 *Let us Ω be a triangulation with a set of vertices V in general position. Then, each vertex v is an end-point of at least one of the frontier-edge or barrier-edges and there is no isolated interior points (vertices incident only to internal-edges).*

Proof: Let v be a vertex associated to the terminal-edge region R generated by the terminal-edge e and T the set of the triangles that share v. By contradiction, lets assume that v is an interior point of R as shown in Figure 4. Since the triangles in T are part of R, they must share their longest-edge around v. Given that T is finite, there should exist a triangle t_0 (see Figure 4) that shares their longest-edge with two triangles of R in order to maintain v interior point in R. This is not possible because a triangle has just one edge labeled as its longest-edge. This contradicts our assumption, so v has to be an endpoint of at least one frontier- or barrier-edge of R. As Ω is partitioned into terminal-edge regions R_i without overlap [12], then there can not exists isolated points in Ω. □

It is worth mentioning that Lemma 1 is important, since it means that the initial points used to represent the geometric domain and the ones inside the domain to fulfill point density requirements, are maintained after the polygon mesh is generated. Moreover, it allows for the use of interior-edges that contain barrier-edge tips as endpoints to split a non-simple polygon into simple polygons.

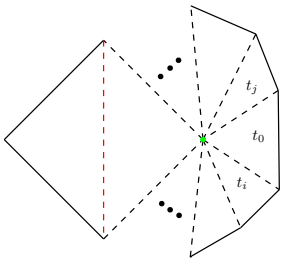


Fig. 4: The vertex in green is an interior point.

There is a degenerate case that does not respect Lemma 1, when points are not in general position. Given a vertex v of a triangulation Ω and T the set of triangles incident to v. If all triangles in T are equilateral, it may happen that the algorithm chooses all the left edges or all right edges as longest-edge in triangles T, generating a circular lepp without terminal-edge and the final mesh would have a hole. This rare case can be solve changing the order in that arbitrary longest-edge is selected in the algorithm.

4 The algorithm

In this section, we describe the main steps of the proposed algorithm, its computational cost and the data structure used for its implementation. The algorithm receives an initial triangulation as input, that can be generated by any known triangulator. The triangulation can be Delaunay or not, but we are using Delaunay triangulations because, as Delaunay triangulation is the triangulation with the maximal angles, then the resulting mesh has the maximal angles too. Also several correct and robust triangulators such as Detri2 [18] and Triangle [17] are available for free. We use Detri2 [18] to generate the constrained Delaunay triangulations used as initial mesh. The whole process applied to the initial triangulation is divided in three phases:

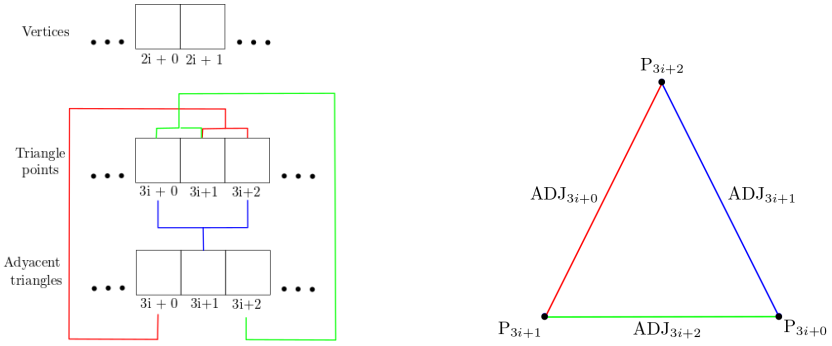
i) Label phase: Each edge is labeled as longest-, terminal or frontier edges. Also the algorithm labels seed triangles used in the next phase.
ii) Traversal phase: Generation of polygons from seed triangles. In this phase frontier-edges of a terminal-edge region are store as part of the new polygon. Non-simple polygons generate in this phase are sent to the partition phase.
iii) Partition phase: Polygon with barrier-edges are partitioned into simple polygons.

4.1 Data structures

In this subsection we show the data structures used in a implementation in C++ and its highlights to contribute to a better understanding of the algorithm.

To represent the Delaunay triangulation, three one-dimensional arrays are used to represent vertices, triangles and adjacent triangles respectively.

Vertices are saved in pairs (x, y), where each two elements of the array are the coordinates x and y of a vertex. Triangle array is set of indices to the Vertex array. Each 3 values is a triangle in the array. Since the algorithm needs to know the neighbor of each triangle, an array is used to save the indices to adjacent triangles: each 3 values in this array $3i + 0$, $3i + 1$, $3i + 2$ are the indices to the triangle of adjacent triangle i.



(a) Data structure (b) Triangle

Fig. 5: Data structure: **(a)** Relation between the order in Triangle array and the elements in Adjacent array. **(b)** The parts of a triangle represented in Triangle array and Adjacent array. In the case of Adjacent array, the elements store both the edges and the adjacent triangle to that edge.

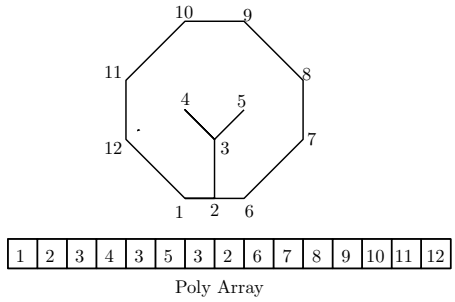


Fig. 6: Representation of polygon as set of vertices. Sequences $3-4-3$ and $3-5-3$ indicate the existence of barrier-edge tips.

To facilitate the implementation of the algorithm, indices in the triangle array are ordered based on adjacent triangles, in such a manner that is easy find out, in a side triangle, whose points are shared with the adjacent triangle. The order is shown in the Figure 5 and it is similar to that used by Detri2 [18]. The first two point indices $3i+0$, $3i+1$ in Triangle array are shared with its adjacent triangle in the position $3i+2$ of Adjacent array, points $3i+1$, $3i+2$ in Triangle array with element $3i+2$ in Adjacent array and elements $3i+1$, $3i+2$ in Triangle array with element $3i+1$ in Adjacent array. This is also an implicit way to identify each edge in a triangle, being the red edge of Figure 5 the edge 0, blue the edge 1 and green the edge 2. To label edges as frontier-edge the algorithm mark the adjacency as -1. For example, in Figure 5, to label red edge as frontier edge, ADj_{3i+0} must change its value by -1.

In the label phase, a seed lists is defined, it is a integer list that contains the indices of the triangles labeled as seed at the end of first phase. In the case of the reparation phase, one open hash table H, without repeat elements, is used instead of a seed list to store the indices of triangles that will generate new polygons.

In the travel phase, polygons are temporally stored in an array with the vertices that conform it as is shown in Figure 6, store one edge in a polygon means adding their to endpoints in counter-clock wise to the temporal array that represents that polygon. After generate each non-simple

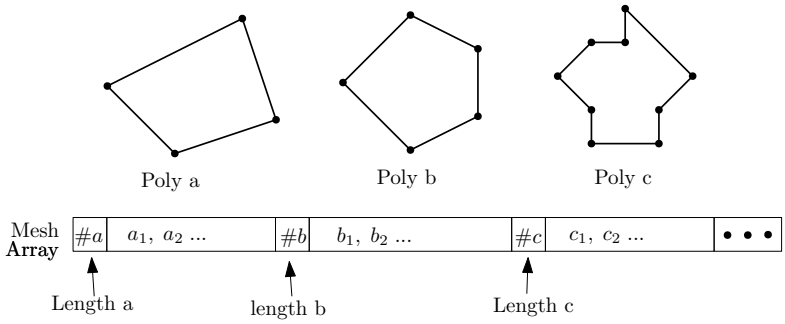


Fig. 7: Mesh array with three polygons.

polygons, they are stored in a mesh. The mesh consists in just one array, where each polygon stores first its length (number of vertices) and after the index of their vertices as is shown in Figure 7. To improve the compute of $deg(b_i)$ we added an array that pair each vertex to one triangle adjacent to that vertex to the algorithm.

All mentioned arrays, list and tables uses $O(n)$ of memory space, where n is the number of points.

4.2 Label Phase

The Label phase refers to labeling each edge of a triangle as a frontier edge or not and looking for triangles to use in the next phase to generate polygons. Those triangles are called seed triangles.

The algorithm first cycles over each triangle of the initial triangulation to compute which edge is the longest-edge; in the case of equilateral and isosceles triangles the algorithm chooses randomly any edge as the longest-edge. This step avoids having a triangle belong to two terminal-edge regions at the same time.

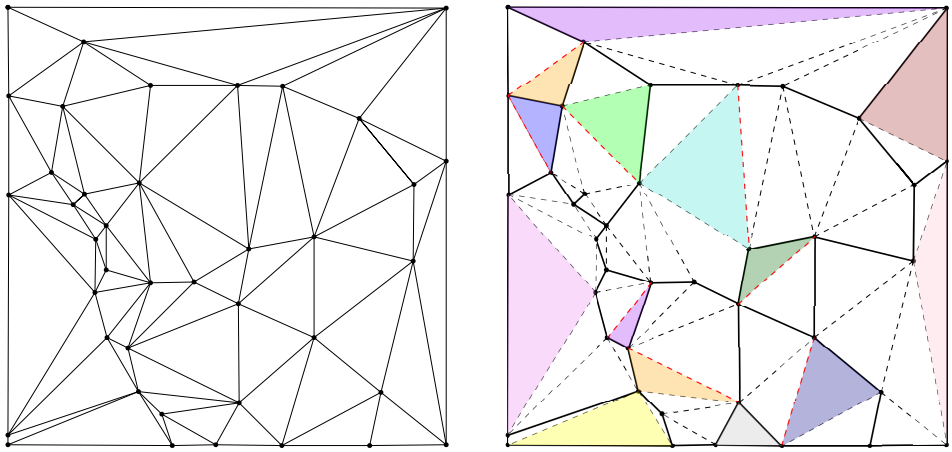
Afterward, the algorithm does a second iteration over the edges. In the case when an edge e is not the longest-edge of any of the two triangles that share it, e is labeled as a frontier-edge. In this iteration the algorithm also labels seed triangles. In the case when e is a terminal-edge, the algorithm sees the triangles adjacent to e and saves one of them (the triangle with lower index in the triangle array) in the seed list. In the case of boundary terminal-edges, the adjacent triangle if saved in the seed list.

The final result of this step is shown in Figure 8. The colorful triangles will be used in the next phase as seed triangles to generate the polygons. In Figure 8(b) it can be also observed that terminal-edge regions are already formed and delimited by frontier-edges.

4.3 Travel Phase

In this phase polygons are recognized and represented as a closed polyline. The main idea behind this phase is travel through adjacent triangles inside a terminal-edge region and save their frontier-edge as edges of the new polygon in counter clock wise order (ccw).

For this purpose, the algorithm uses each triangle t in the seed list created in the previous phase as the starting triangle to build each polygon. Note that every triangle can be used to generate a polygon, but we use one triangle adjacent to a terminal-edge to avoid generating the same polygon twice.



(a) Delaunay Triangulation

(b) Triangulation after applying the label phase

Fig. 8: Label phase: (a) Input Delaunay triangulation. (b) Solid lines are the frontier-edges, dashed lines are the internal-edges and colorful triangles are seed triangles.

Let be t a triangle of the seed list and P a temporary array to store a polygon, if t has 3 frontier-edges, then t is saved as a polygon in P, and the algorithm goes to the next triangle in the seed list, else, the algorithm stores the tree or two continuous endpoints of the edges of t labeled as frontier-edge in P in ccw, saves t as a initial triangle, choose the first stored endpoint v_{init} in P as initial vertex (by lemma 1 each vertex of a terminal-edge region is part of a frontier-edge) and the last stored endpoint v_{end} in P. After, the algorithm travels to the next adjacent triangle t' in ccw that have v_{end} as vertex. There are three cases for t':

i) The triangle t' has just 1 continuous frontier-edge e that contains v_{end}. Then e is store in P, v_{end} is updated with last stored endpoint and the next triangle t' is the next non-visited triangle that contains the new v_{end}.
ii) The triangle t' is an ear triangle (a triangle with 2 frontier-edges). In this case, those both edges are stored in ccw in P, v_{end} is updated with last stored endpoint and the last triangle visited is the new t' because the other two triangles are adjacent by frontier-edges (i.e. They are in other terminal-edge region).
iii) The triangle t' has no frontier edge. t' shares an internal-edge with the last triangle visited and with other two triangles that can be visited, but just one of these contains the endpoint v_{end} as vertex, so that triangle is the new t'. This case can occur when a barrier edge is found, when a triangle is between two ear triangles or when the algorithm surrounds an ear triangle of another terminal-edge region.

The travel phase ends when v_{end} reaches the initial point v_{init} and the initial triangle t is the same as $'t$. Experiments showed that both conditions are necessary, the first one due to a triangle can be visited multiple times in flat polygons and the second one due to barrier-edges can be omitted if only the first condition is used. An example of this phase is shown in Figure 9.

After generating the polygon P, the algorithm checks if P has barrier-edge tips. This can be easily done by just checking repetition of consecutive vertices in P. Given three consecutive vertices of P, v_i, v_j and v_k. If v_i and v_k are equal, then v_j is a barrier-edge tip. If P does not have barrier-edge tips, then the polygon is saved as part of the mesh, else, the polygon is sent to the next of non-simple polygon reparation.

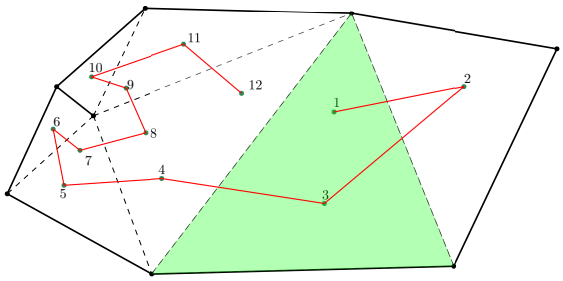


Fig. 9: Travel of the algorithm inside a terminal-edge region. The green triangle is a seed triangle, the numbers indicate in which step of the iteration each triangle is visited. Note that triangles with no frontier-edge are visited 3 times, with 1 frontier-edge 2 times and with 2 frontier-edges 1 time.

4.4 Non-simple polygon reparation

As we have mentioned, the generated polygons might be non-simple; this is the fact when they contain barrier-edges tips. In this section, we describe the algorithm to transform non-simple polygons into simple-ones by using the barrier-edges inside polygons and some additional internal-edges of the initial triangulation to build the new simple polygons. The use of any internal-edge allows us to split polygons as stated in Lemma 1.

The process of reparation works similar to label phase and travel phase but inside a polygon P, its exploits the lemma 1, as all vertices inside the triangulation are part of a frontier-edge, then labeling as frontier-edge an internal-edge that contains a barrier-edge tip will generate two polytopes that can be used to generate two new polygons. So the algorithm changes internal-edge to frontier-edges, saves seed triangles adjacent to those new frontier-edges, and uses them to generate new polygons, but instead of using a seed list, the algorithm generates a hash table H with seeds triangle to generate polygons based on the original non-simple polygon P.

The process is the following: For each barrier-edge tip b_i in a polygon P, the algorithm count the number $deg(b_i)$ of incident internal-edges to b_i in the original triangulation, if $deg(b_i)$ is odd then the algorithm labels the middle edge as frontier-edge, else, the algorithm chooses any of the middle edges to label as frontier-edge. In both cases, two triangles adjacent to the new frontier-edge are saved in a hash table H to use one of them a seed triangle. Figure 10(a) demonstrate an example of a polygon with three barrier-edge tips (vertices in green color). Figure 10(b) shows the as the middle internal-edge of all three barrier-edge tips is changed to a frontier-edge in order to generate the delimitation of the news polygons and six seeds triangles are saved in H.

Later, the algorithm uses each triangle t in the hash table H to generate a new polygon, for each $t \in H$ the algorithm repeats the travel phase, but each triangle t' reached during the travel is verified if t' is in H. If it is the case, then the $'t$ is removed from H to avoid generate the same polygon several times. This can be seen in Figure 10(b), there are three purple triangles, but just one is chosen from H to generate the new polygon, during its generation two purple triangles are removed from H to avoid generate the same polygon again. Figure 10(b) shows the new polygon after the split and the seed triangles that generate them.

The number of polygons generated after the split is at the most $(\#b+1)$, with $\#b$ the number of barrier-edge tips in a polygon P. Figure 10 explains the whole process.

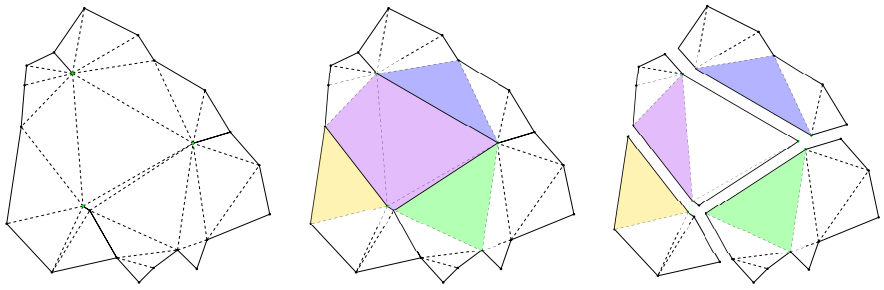


Fig. 10: Example of polygon split using barrier-edge tips. (a) Original polygon to split. (b) Middle edges incident to barrier-edge tips are label as frontier-edge and seed triangles (colorful triangles) are store in a hash table H (c) Seeds triangles are used to generate new polygons without barrier-edge tips,

4.5 Computational Complexity Analysis

In this section, we analyze the computational complexity of each phase and of the whole algorithm. Let n be the initial number of points, m the number of triangles of the Delaunay triangulation, k the number of triangles of the terminal-edge region used to generate the polygon P and $\#b$ the number of barrier-edge tips of P.

0. **Initial Triangulation** The cost of generate a Delaunay Triangulation from a random set of points is $O(n\,log\,n)$.
1. **Label Phase** This phase uses 3 iterations of cost $O(m)$, one to search max edges, other to label edges and another to save seed triangles.
2. **Travel Phase** Build any polytope has cost $O(k)$, each triangle is visited at least 3 times in the worst case (when a triangle does not have a frontier-edge). As each terminal-edge region covers the whole domain without overlapping, then this phase has cost $O(m)$.
3. **Non-simple polygon reparation phase** Check if a polygon has barrier-edge tips and calculates $deg(b_i)$ has cost $O(k)$ since the first one is an iteration over the number of vertices of the polygons and the second one the maximum number of triangles to count is k. The max number of elements to save in the hash table is $2 * \#b < k$, so building the hash table has a cost of $O(k)$, using an open hash, each search and deletion has a cost of $O(1)$. The cost of building the $O(\#b)$ new polygons cost $O(k)$, since new polygons uses the same k numbers of triangles to be built, each triangle is just visited at least 3 times in the worst case. If all the polygons generated in travel phase have barrier-edge tips, this process has a final cost of $O(n)$.

Finally the cost of the algorithm is $O(m)$, with $O(n\,log\,n)$ the cost of generating a initial triangulation.

5 Terminal-edge regions based meshes vs Voronoi based meshes

To show statistics about terminal-edge regions based meshes in contrast to a Voronoi diagram based mesh, we designed a simple experiment using a implementation of the algorithm in C++.

The input of the experiment was generated with a initial point sets, those points are random points inserted inside a $2x2$ square. The number of points ranges from 10^1 to 10^5. A tolerance

Input points	Triangle number	Terminal-edges number	Terminal-edge polygon	Max bet in non-simple Polygons	Total Bet	Triangles per polygon	Edges per polygon
10	14	5	5	0	0	2.80	5.80
10^2	192	34	38	2	4	5.05	7.18
10^3	1949	283	310	2	27	6.29	8.30
10^4	19618	3012	3192	4	180	6.15	8.15
10^5	197976	29965	31902	4	1938	6.21	8.21

Table 1: Geometric information of polygon mesh

Input Points	Voronoi vertices	Voronoit Regions	Voronoi Edges	Edges per Region
10	22	10	31	4.9
10^2	202	100	301	5.75
10^3	2002	1000	3001	5.9
10^4	20001	10000	30001	5.9618

Table 2: Geometric information of Clipped Voronoi diagram

$\pm\gamma$ is defined in case of a point p is too close to one edge the square; in that case the point p is inserted in the border edge instead. With those initial point sets were generates a set of Delaunay triangulation using Detri2 [18]. Results of the experiment are summarized in table 5. In the case of Voronoi meshes, Deldir [23] was used to generate the geometric information of Clipped Voronoi diagram in Table 2, using the same initial point sets as Voronoi sites.

In Table , we can observe that after 10^4 input points, the number of triangles per polygon is in average 6.5 and the number of edges per polygon is 8.5. The number of barrier-edges is less than 1% of the number of points, so the reparation phase just adds $\approx 0.5\%$ of polygons to the mesh. If we compare these terminal-edge meshes with the meshes generated from the Voronoi diagram, the clipped Voronoi diagram contain 3 times more polygons than the our meshes. Each Voronoi region is formed by max 6 edges and, on average, terminal-edge polygons by 8 edges.

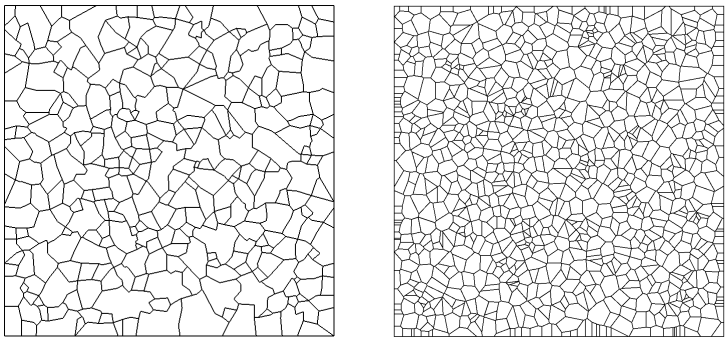
Additionally, the terminal-edge regions meshes use only the points given as input; in contrast, the Voronoi based meshes use new points, the Voronoi points, one per each triangle of the Delaunay mesh. Since the number of triangles is greater than the number of input points, the size of the Voronoi mesh is greater than the size of the terminal-edge region mesh not only in terms of polygons but also in terms of mesh points.

The terminal-edge region based mesh does not need to insert extra-points to the boundary to fit the device geometry; in contrast the constrained Voronoi based mesh needs to introduce new points at the boundary/interfaces to cut the Voronoi regions that go outside the domain.

Figure 11 is a comparison between terminal-edge mesh and Clipped Voronoi diagram. Both were generated with the same initial triangulation using DetriQT [18]. Figure 11(a) shows the polygon meshes generated by the proposed algorithm and Figure 11(b) a clipped Voronoi diagram.

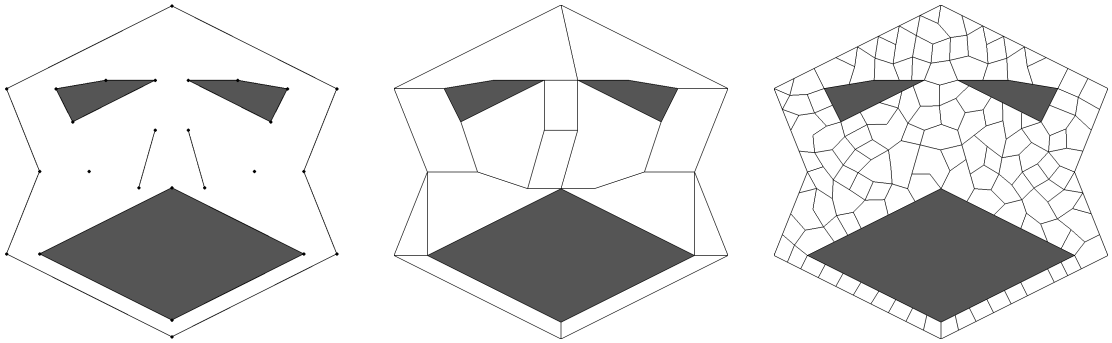
Figure 12 shows how more refinement triangulation as input affects to the output mesh. Figure 12(a) is a PLSG from [17] with 26 vertices. 12(b) is a terminal-edge mesh generate using a constrained Delunay triangulation of 12(b). Figure 12(c) uses a conforming Delaunay triangulation of 12(b) with a maximum triangle area constraint of 100.

Figure 13 compares the terminal-edge mesh (Figure 13(b) and the clipped Voronoi mesh (Figure 13) of the same triangulation in Figure 13(a). As Voronoi diagram mesh in Figure 13 has in total 100 vertices, Figure 13(d) shows the same unicorn generate from Delaunay refinement of Figure 13(d) with of 100 vertices in total.



(a) Polygon Mesh (b) Voronoi diagram

Fig. 11: **(a)** Polygon mesh generated by 1000 random points.**(b)** Clipped Voronoi diagram of the same 1000 random points generated with Detri2QT [18].



(a) Face PLSG from [17] (b) Face from Constrained Delaunay (c) Face from more refined triangulation

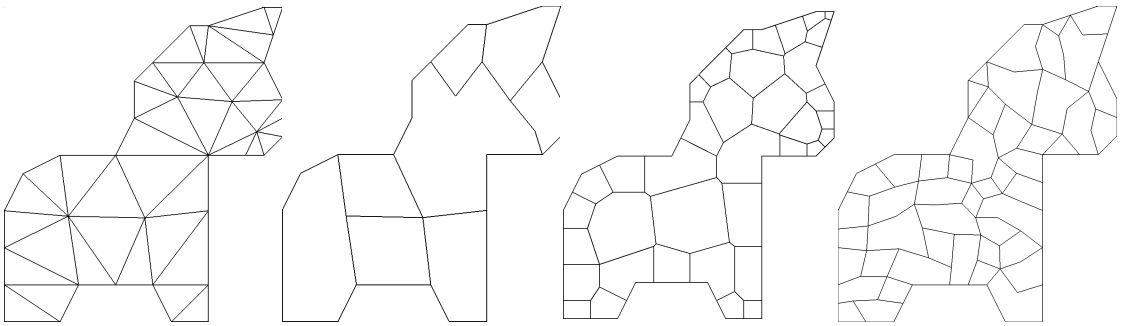
Fig. 12: **(a)** Original PLSG from [17] with 26 vertices, grey areas are holes. **(b)** Terminal-edge mesh version from a constrained Delaunay triangulation with the same 26 vertices. **(c)** Terminal-edge version of a refined Delaunay triangulation with 220 vertices.

6 Preliminary Simulation Results

In this section, we assess the terminal-edge region meshes in the virtual element method (VEM) [25]. To this end, an L-shaped domain is considered. Fig. 14 shows this domain meshed with a random and a semiuniform terminal-edge region sample mesh. The chosen problem is governed by the Laplace equation and its exact solution is given by [10]

$$u(x_1, x_2) = r^{2/3} \sin(2/3\,\theta), \quad r = \sqrt{x_1^2 + x_2^2}, \quad \theta(x_1, x_2) = \arctan(x_2/x_1).$$

The boundary conditions are of Dirichlet type with the exact solution imposed on the entire domain boundary. The re-entrant corner of the L-shaped domain introduces a singularity in the solution that manifests itself as unbounded derivatives of u at the origin. The numerical solution (denoted by u^h) is assessed through its convergence with mesh refinements. Figs. 15 and 16 present the L^2 norm and the H^1 seminorm of the error, where it is shown that the optimal rates of convergence of 2 and 1 for the L^2 norm and the H^1 seminorm of the error, respectively,



(a) Unicorn Triangulation (b) Unicorn Terminal-edge mesh (c) Clipped Voronoi unicorn (d) Unicorn from more refined triangulation

Fig. 13: Comparision uniform PLSG from [13] **(a)** Triangulation of unicorn PLSG with 36 vertices. **(b)** Terminal-edge mesh unicorn with 36 vertices. **(c)** Clipped Voronoi Unicorn PLSG with 100 vertices. **(d)** Terminal-edge mesh Unicorn from refined Delaunay triangulation with 100 vertices.

are delivered by the VEM with random and semiuniform terminal-edge region meshes. Finally, contour plots of the VEM solution for the u field are shown in Fig. 17 for a random and a semiuniform mesh.

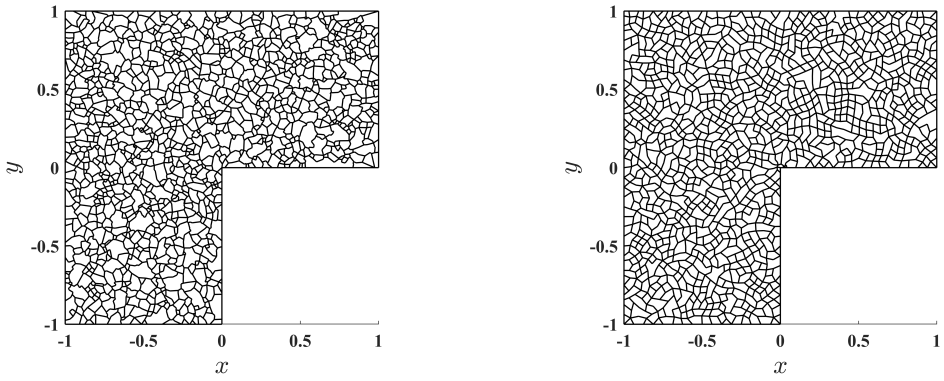
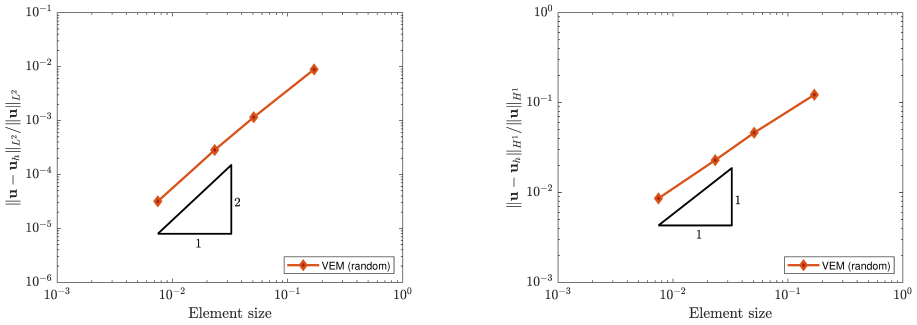


Fig. 14: L-shaped domain meshed with **(a)** a random and **(b)** a semiuniform terminal-edge region mesh.

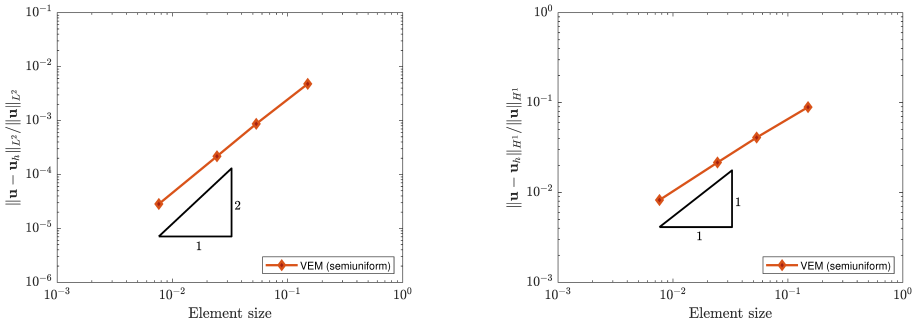
7 Conclusions and ongoing work

We have presented a preliminary formalization and statistical evaluation of a new kind of polygon mesh based on terminal-edge regions. We have observed that this kind of mesh contains three times less polygons and half the number of points than the standard polygon meshes based on the Voronoi diagram.



(a) (b)

Fig. 15: **(a)** L^2 norm and **(b)** H^1 seminorm of the error for the L-shaped domain problem using the VEM and random terminal-edge region meshes.



(a) (b)

Fig. 16: **(a)** L^2 norm and **(b)** H^1 seminorm of the error for the L-shaped domain problem using the VEM and semiuniform terminal-edge region meshes.

We have shown some preliminary simulation results to show that this kind of mesh can be as useful as Voronoi meshes with the VEM but using less points and polygons. In 3D, where the number of tetrahedra in the worst case can be $O(n^2)$, with n the number of input points, a stronger mesh point reduction should be obtained than in 2D.

Until reciently, the mesh refinement had to be done in the input triangulation and the presented algorithm used to build the polygon mesh from that input. Since we keep the underline triangulation, we are planning to allow further refinement inside the polygon mesh in the next version of this polygon mesh generator.

One of our hypothesis is that to take as input a Delaunay triangulation instead of any other triangulation should allow us to generate polygon meshes with less points and polygons. Thus, our ongoing work is to complete the theoretical formulation and determine the scope these kind of mesh can be used. Since terminal edge-regions exist in 3D [7], we hope to soon have a 3D polyhedron mesh generator.

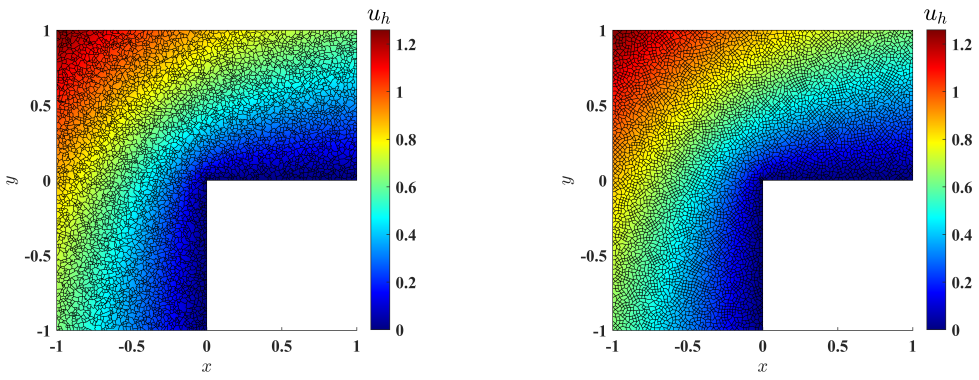


Fig. 17: Contour plots of the VEM solution for the u field. **(a)** A random and **(b)** a semiuniform terminal-edge region mesh.

References

1. Alonso, R., Ojeda, J., Hitschfeld, N., Hervías, C., Campusano, L.: Delaunay based algorithm for finding polygonal voids in planar point sets. Astronomy and Computing **22**, 48 – 62 (2018)
2. Benedetto, M.F., Berrone, S., Pieraccini, S., Scialò, S.: The virtual element method for discrete fracture network simulations. Computer Methods in Applied Mechanics and Engineering **280**, 135–156 (2014)
3. Canann, S., Tristano, J., Staten, M.: An aproach to combined laplacian and optimization-based smoothing for triangular, quadrilateral and quad-dominant meshes. In: 7th International Meshing Roundtable, pp. 479–494 (1998)
4. Chew, L.: Constrained delaunay triangulation. In: Algorithmica, vol. 4, pp. 97–108 (1994)
5. Ebeida, M.S., Mitchell, S.A.: Uniform random voronoi meshes. In: Proceedings of the 20th International Meshing Roundtable, IMR 2011, October 23-26, 2011, Paris, France, pp. 273–290 (2011)
6. Garretón, G.: A hybrid approach to 2d and 3d mesh generation for semiconductor device simulation. Ph.D. thesis, ETH Zürich. Series in Microelectronics, Vol. 80 (1999)
7. Hervías, C., Hitschfeld-Kahler, N., Campusano, L.E., Font, G.: On finding large polygonal voids using Delaunay triangulation: The case of planar point sets. In: Proceedings of the 22nd International Meshing Roundtable, pp. 275–292 (2013)
8. Hussein, A., Aldakheel, F., Hudobivnik, B., Wriggers, P., Guidault, P.A., Allix, O.: A computational framework for brittle crack-propagation based on efficient virtual element method. Finite Elements in Analysis and Design **159**, 15 – 32 (2019)
9. Lee, K.Y., Kim, I.I., Cho, D.Y., wan Kim, T.: An algorithm for automatic 2d quadrilateral mesh generation with line cons traints. Computer-Aided Design **35**(12), 1055 – 1068 (2003)
10. Mitchell, W.F.: A collection of 2d elliptic problems for testing adaptive grid refinement algorithms. Applied Mathematics and Computation **220**, 350–364 (2013)
11. Natarajan, S., Ooi, E.T., Saputra, A., Song, C.: A scaled boundary finite element formulation over arbitrary faceted star convex polyhedra. Engineering Analysis with Boundary Elements **80**, 218 – 229 (2017)
12. Ojeda, J., Alonso, R., Hitschfeld-Kahler, N.: A new algorithm for finding polygonal voids in delaunay triangulations and its parallelization. In: The 34th European Workshop on Computational Geometry, EuroCG, pp. 56:1–6 (2018)
13. Ortiz-Bernardin, A., Álvarez, C., Hitschfeld-Kahler, N., Russo, A., Silva-Valenzuela, R., Olate-Sanzana, E.: Veamy: an extensible object-oriented C++ library for the virtual element method. Numerical Algorithms **82**(4), 1–32 (2019)
14. Owen, S.J., Staten, M.L., Canann, S.A., Saigal, S.: Advancing front quadrilateral meshing using triangle transformations. In: Proceedings, 7 th International Meshing Roundtable 98, pp. 409–428 (1998)
15. Rivara, M.C.: New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. Int. Jour. for Num. Meth. in Eng. **40**, 3313–3324 (1997)
16. Shewchuk, J.R.: Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In: ACM (ed.) First Workshop on Applied Computational Geometry, pp. 124–133. (Philadelphia, Pennsylvania) (1996)

17. Shewchuk, J.R.: Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In: M.C. Lin, D. Manocha (eds.) Applied Computational Geometry Towards Geometric Engineering, pp. 203–222. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
18. Si, H.: An introduction to unstructured mesh generation methods and softwares for scientific computing. Course (2019). 2019 International Summer School in Beihang University
19. Sieger, D., Alliez, P., Botsch, M.: Optimizing voronoi diagrams for polygonal finite element computations. In: Proceedings of the 19th International Meshing Roundtable, IMR 2010, October 3-6, 2010, Chattanooga, Tennessee, USA, pp. 335–350 (2010)
20. Sukumar, N., Malsch, E.A.: Recent advances in the construction of polygonal finite element interpolants. Archives of Computational Methods in Engineering **13**(1), 129–163 (2006)
21. Tabarraei, A., Sukumar, N.: Extended finite element method on polygonal and quadtree meshes. Computer Methods in Applied Mechanics and Engineering **197**(5), 425–438 (2008)
22. Torres, J., Hitschfeld, N., Ruiz, R.O., Ortiz-Bernardin, A.: Convex polygon packing based meshing algorithm for modeling of rock and porous media. In: V.V. Krzhizhanovskaya, G. Závodszky, M.H. Lees, J.J. Dongarra, P.M.A. Sloot, S. Brissos, J. Teixeira (eds.) Computational Science – ICCS 2020, pp. 257–269. Springer International Publishing, Cham (2020)
23. Turner, R.: deldir: Delaunay Triangulation and Dirichlet (Voronoi) Tessellation (2021). URL https://CRAN.R-project.org/package=deldir. R package version 0.2-10
24. V, E.C.: Mixed Virtual Element Methods. Applications in Fluid Mechanics. B.S. Thesis, Universidad de Concepción, Concepción, Chile (2015)
25. Beirão da Veiga, L., Brezzi, F., Cangiani, A., Manzini, G., Marini, L., Russo, A.: Basic principles of virtual element methods. Mathematical Models and Methods in Applied Sciences **23**(01), 199–214 (2013)
26. Beirão da Veiga, L., Brezzi, F., Marini, L.D.: Virtual elements for linear elasticity problems. Siam Journal on Numerical Analysis **51**(2), 794–812 (2013)
27. Beirão da Veiga, L., Lovadina, C., Mora, D.: A virtual element method for elastic and inelastic problems on polytope meshes. Computer Methods in Applied Mechanics and Engineering **295**, 327–346 (2015)
28. Wriggers, P., Aldakheel, F., Hudobivnik, B.: Application of the virtual element method in mechanics. Tech. rep., Report number: ISSN 2196-3789. Leibniz Universität Hannover (2019)
29. Wriggers, P., Hudobivnik, B.: A low order virtual element formulation for finite elasto-plastic deformations. Computer Methods in Applied Mechanics **327**, 459–477 (2017)
30. Wriggers, P., Reddy, B.D., Rust, W.T., Hudobivnik, B.: Efficient virtual element formulations for compressible and incompressible finite deformations. Computational Mechanics **60**, 253–268 (2017)
31. Yan, D.M., Wang, W., Lévy, B., Liu, Y.: Efficient computation of 3d clipped voronoi diagram. In: GMP, pp. 269–282 (2010)