



Convex Polygon Packing Based Meshing Algorithm for Modeling of Rock and Porous Media

Joaquín Torres^{1(✉)}, Nancy Hitschfeld^{1(✉)}, Rafael O. Ruiz²,
and Alejandro Ortiz-Bernardin³

¹ Departamento de Ciencias de la Computación,
Universidad de Chile, Santiago, Chile
{jtorres,nancy}@dcc.uchile.cl

² Departamento de Ingeniería Civil, Universidad de Chile, Santiago, Chile
rafaelruiz@uchile.cl

³ Departamento de Ingeniería Mecánica, Universidad de Chile, Santiago, Chile
aortizb@uchile.cl

Abstract. In this work, we propose new packing algorithm designed for the generation of polygon meshes to be used for modeling of rock and porous media based on the virtual element method. The packing problem to be solved corresponds to a two-dimensional packing of convex-shape polygons and is based on the locus operation used for the advancing front approach. Additionally, for the sake of simplicity, we decided to restrain the polygon rotation in the packing process. Three heuristics are presented to simplify the packing problem: density heuristic, gravity heuristic and the multi-layer packing. The decision made by those three heuristic are prioritizing on minimizing the area, inserting polygons on the minimum Y coordinate and pack polygons in multiple layers dividing the input in multiple lists, respectively. Finally, we illustrate the potential of the generated meshes by solving a diffusion problem, where the discretized domain consisted in polygons and spaces with different conductivities. Due to the arbitrary shape of polygons and spaces that are generated by the packing algorithm, the virtual element method was used to solve the diffusion problem numerically.

Keywords: Polygonal meshes · Geometric packing · Virtual element method · Computational geometry

1 Introduction

In the last decade, the use of finite element methods (FEM) have been the common engineering practice to design and evaluate the performance of different

Supported by the Chilean National Fund for Scientific and Technological Development (FONDECYT) through grants CONICYT/FONDECYT No. 1181506, No. 11180812, and No. 1181192.

systems. However, in many applications, the FEM has shown some limitations related to the complexity involved in the mesh generation, specially for problems in which the domain is defined by an arrangement of irregular sub-domains. In particular, these situations are found in problems related to the flux of fluid and heat in porous media, fracture mechanics of conglomerate rocks, stability of tailing dams, concrete modeling, amount others. Here, one of the issues is to deal with the random nature of the sub-domains, requiring the statistical study of the problem with multiple simulations. As consequence, the computational burden increases since it is required to draw the geometry and create the mesh several times.

With the advent of the virtual element method (VEM) [15] very general polytopal meshes (polygons can even be non-convex) can now be used to simulate problems based on Galerkin methods in a manner similar to FEM. In porous media, microstructure, rock accumulation, among others, the bidimensional domain is composed naturally of arbitrary polytopal shapes, which in a simulation can be represented by virtual elements. In this regard, the mentioned issues could be solved using the packing perspective together with numerical methods that employ polytopal meshes (i.e., Virtual Element Method). In a general perspective, packing is an optimization problem on how to organize the content of a container as densely as possible. A particular example of packing is the geometric packing; this packing comprises fitting geometric figures as much as possible inside a container. For example, packing polygons inside a rectangle container, or packing tetrahedra inside a cube container. Then, these polygons/polyhedrons could be used to define the sub-domains at the same time that could be used as a mesh under VEMs schemes.

Different packing strategies have been developed in the past employing different packing geometries. For example, adopting a circle and sphere packing [3, 8, 10] and exploring applications of circle packing through simulations of discrete earthquakes [16] or employing packing geometries based on square-like or rectangular-like shapes [6, 7]. However, the use of convex polygon shapes had been received a limited attention, being the advancing front approach [4] one of the seminal works in this matter.

In this work, we propose a new packing algorithm designed for the generation of polygonal meshes to be used for modeling of rock and porous media based on VEM. Here, the packing problem to be solved corresponds to a two-dimensional packing of convex-shape polygons, where the designed algorithm is based on the locus operation used for the advancing front approach [4]. Additionally, for the sake of simplicity, we decided to restrain the polygon rotation in the packing process. In the following sections, the locus operation principle is presented first. Then, it is explained how the new algorithm works, and subsequently, to demonstrate the potential and the feasibility of the proposed algorithm, a diffusion problem is solved using the virtual element method on a domain discretized with a packing-based mesh.

2 Advancing Front Approach

From the computational complexity theory, the packing problem is considered as a NP problem [2], which indicates the imperative use of an heuristic to establish a proper solution. The heuristic employed here is based on the so called Advancing Front Technique [4] to allocate a new polygon on the boundary of the current polygon cluster. The algorithm determines the position of the new polygon by applying an operation to all active polygons in the cluster. The concept of active polygons is used to identify the polygons that belong to the top layer of polygons in the packing container. In particular, the algorithm builds a locus over each of these active polygons, which is defined as the resulting polygon after sliding the new polygon around the active polygons. This operation is similar to the Minkowski addition [1]. However, the Minkowski addition extends the polygon in a particular direction, contrary to the locus that extends the polygon in all directions. The locus generated from the active polygons of the layer (loci) help to identify the possible positions of the new polygon on the locus. In this regard, the intersection of two loci results in the possible positions to allocate the center of the new polygon. Note that this approach enforce the intersection of the polygon with the active layer. Figure 1 shows a scheme to exemplify the algorithm steps to build the locus polygons. In the figure, it is possible to observe the locus of each polygon and the possible positions of the new polygon defined by the intersection between the locus. The computational cost to obtain the locus is of order $O(n + m)$ using a cross product to determine intersection on each step with n and m the number of vertices of P and Q , respectively.

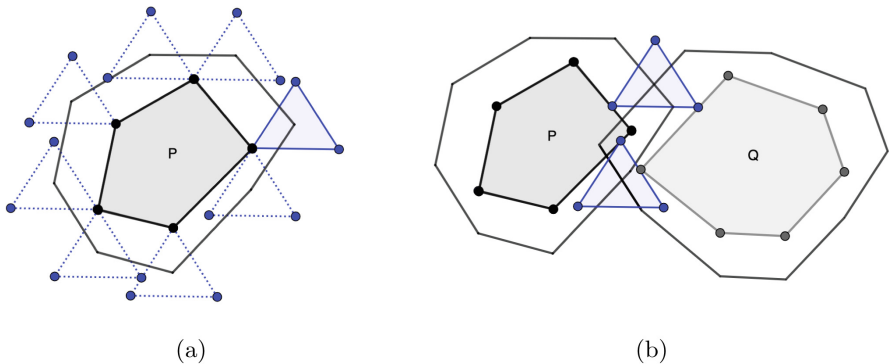


Fig. 1. Example of the locus algorithm with two polygons P and Q . a) Positions where the polygon moves around and the resulting locus. b) Intersection points of the two loci and the possible positions to allocate the center of the new polygon.

Note that the advancing front approach has been used before for packing circles, specifically using the locus of polygons to fill the domain [5]. Similarly, the algorithm has also been used to perform a sequential sphere packing [9].

3 Convex Polygon Packing Algorithm

As it was stated before, the packing algorithm is considered a NP problem which requires an heuristic approach. For this purpose, we propose three different approaches on how to generate the mesh. These approaches are:

1. Density heuristic: Aims to place the polygons on a position that minimize the area generated between the packed polygons.
2. Gravity heuristic: Prioritizes the Y position overall. It tries to place the polygon as low as it can.
3. Multi-layer packing: This packing works differently, groups the input on different batches of polygons, so it simulates multiple layers piled up. In each of these layers, the multi-layer packing uses either the density heuristic or the gravity packing heuristic.

Figure 2 shows a scheme to exemplify a comparison between the density heuristic and the gravity heuristic. We show the decision the algorithm takes when inserting a polygon on the layout. On both images, the polygons with striped lines are the positions where the next polygon could be placed. The green polygon the position selected to place the polygon.

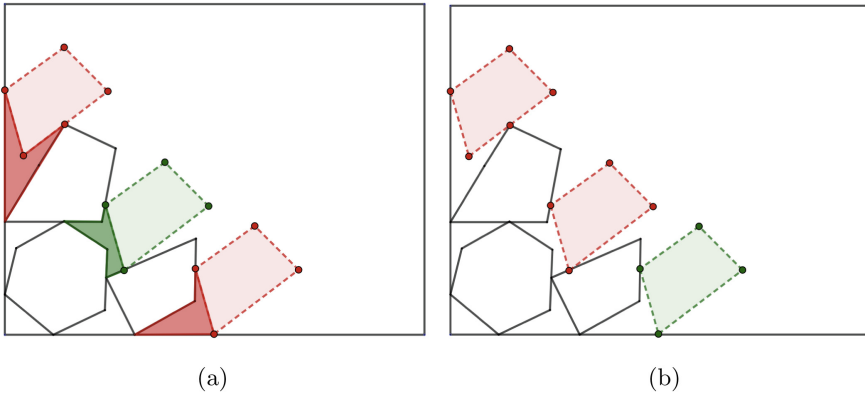


Fig. 2. Gravity heuristic and density heuristic decision example. (a) Decision made by the area heuristic choosing the middle position because the empty space left by the polygon has the smallest area. (b) Decision made by the gravity heuristic choosing the bottom position.

3.1 Input Construction

The input construction refers to the definition of any new polygon that will be introduced in the container. Here, two restrictions are imposed: only convex-shape polygons are generated, and the rotation is restrained. Additionally, two variables are introduced to control the roundness and size. Then, the polygons are generated adopting a strategy based on circumscribed circles [13] following three steps: (1) create a circle, (2) placing points randomly on the circle perimeter, and (3) connecting the points in a counterclockwise order. As a summary, the input of this algorithm is a list of radius and number of vertices, while the output corresponds to a list of convex polygons.

3.2 Inserting a New Polygon

Once the new polygon is defined, it is necessary to define the algorithm to allocate it. The first step corresponds to insert the first polygon, which comprises of inserting the polygon at the bottom left corner of the layout (container). For this purpose, the algorithm searches for the minimum x and y coordinates of the polygon. Then, the polygon is translated to the corner applying a translation. In the second step, the process is repeated until there is no polygon left to pack or cannot pack the remaining polygons. Inside the loop, the next polygon on the list is take and located in the best position it could be placed depending on the heuristic decision. If the polygon cannot be inserted, the algorithm returns null, and it is discarded.

Additionally, it is important to highlight that at the moment to insert a new polygon, the algorithm iterates over the active polygons to identify if the new element suits. Thus, the algorithm does the following steps to determine the best position:

1. It looks for the neighbors of the polygons and checks if the distance between them is less than the longest diagonal of the inserted polygon.
2. Then, it obtains the locus polygon of both packed polygons and intersects those loci to get the intersection points.
3. Finally, it tests the obtained position with the used heuristic. If the result of the evaluation is better than the current best one, it saves the position; otherwise, the position is discarded.

After the algorithm allocates the polygon, it checks if the neighbor polygons of the inserted polygon are not suitable for following iterations. For example, if none of the next polygons can be place near them, then that polygon is not suitable. This check helps the algorithm to discard polygons and also improve the computational cost.

3.3 Neighborhood Data Structure

The allocation of new polygons could have a significant computational cost. In order to alleviate this computational burden, we propose the construction of a neighborhood data structure that is consulted in each iteration.

In particular, the data structure consists of a graph with nodes defined by the centroid of each polygon and the links between the polygon and its neighbors. There are concave or convex polygons between polygons, those polygons are named “spaces” and also those spaces are inserted to the graph. The graph is initialized inserting the container. The container is accounted employing two nodes, one that represents the container itself and another that represents the empty space. After inserting the container, the graph is updated in the same fashion after each polygon insertion: (1) identify the space that contains the inserted polygon, (2) generate two new spaces from allocating the polygon, (3) add the new spaces and the polygon as new nodes, and (4) update the neighborhood links. Figure 3 shows (a) the mesh of an intermediate iteration of the algorithm and (b) the graph build from the polygon insertions.

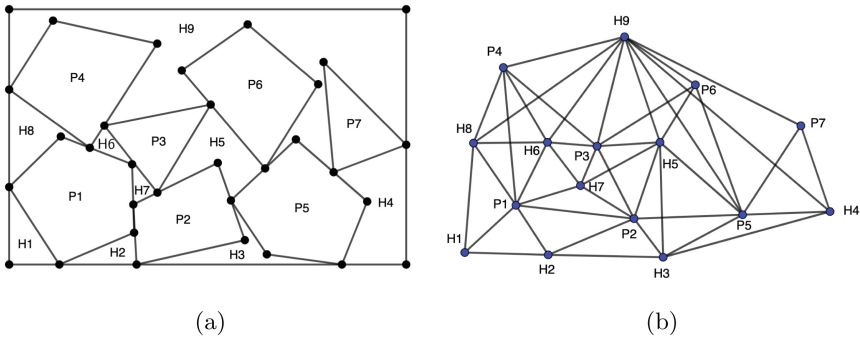


Fig. 3. Packing mesh and its graph data structure. Each of the polygons is named with a P, and the spaces with H. (a) Intermediate stage of the algorithm. (b) Graph representation of the intermediate stage (a)

In each step of the packing algorithm, the next inserted polygon or the target polygon is processed in the same way. After we insert the target polygon, a space inside is divided into two spaces. So we need to look for that space to update the graph. Note that this space can be a concave or convex polygon. So to find this space we use the Shimrat’s algorithm [14], which finds if a point lies inside a polygon. The Shimrat’s algorithm uses ∞ to cast a ray, for this purpose, we use a point far away from the layout and then casting a ray from there to the centroid of the target polygon. Then we use this algorithm on each space across the graph, if this ray intersects even times, the centroid of the target polygon is outside that space, and if the ray intersects odd times, the centroid of the target polygon is inside.

After getting the space to divide or the overlapping space, the algorithms divide this space into two new ones. Thus for this division, we need to find the intersection points between the target polygon and the overlapping space. For this intersection, we use a simple algorithm of order $O(n^2)$, but it can be improved with an algorithm that uses a sweep line approach [17] to order

$O(n \log(n))$. Then the algorithm uses intersection points to build the new spaces, iterating across the overlapping hole in clockwise order from one intersection point to the other intersection point. Then calculates the same route but starting with the second intersection point. The same algorithm is applied to the target polygon generating 4 routes. Although we have these four routes, the algorithm does not know how to connect these routes correctly. For example, it can merge two routes and have a wrong representation of the layout generating a space containing the target polygon. This error can cause an incorrect construction of the mesh. We solve this problem efficiently with the Shimrat's algorithm. In this way, the algorithm connects two random routes, one from the target polygon and from the overlapping space. If the space built contains the centroid of the target polygon, this means we are connecting the wrong routes, so we change one of these and get the correct polygon. We get the other space from the two remaining routes.

Finally, with the new two spaces, we update the graph structure linking the spaces to the polygons linked with the overlapping space. We get the neighbor polygons with the intersection points between the space and the neighbor polygons. Also, we add the target polygon into the graph, link it with the neighbor polygons, and the two new spaces. This concludes an insertion step of the algorithm.

4 Complexity Analysis

In this section, we analyze the computational order of the algorithm by computing the computational complexity of each step and, finally, the order of the whole algorithm. Being p the number of packed polygons and n the number of vertices of the resulting mesh, the order of each step is:

1. First, we look for pairs of polygons and test first if they are close. This test costs $c * p$ being c the number of neighboring polygons, but the number of neighbors is negligible compared to the number of polygons of the mesh. So looking for pairs of polygons cost $O(p)$.
2. After we have a pair of polygons, we get the locus of the two chosen polygons. Getting the locus costs $O(n)$ [4].
3. We got the loci for the possible positions for the target polygon. Then, we intersect the loci and get the points where we can allocate the centroid of the target polygon. The intersection of the loci costs $O(n)$ because the loci are convex.
4. Now that we got a possible position, we test the position on each heuristic:
 - Density Heuristic: For this heuristic, we need to first find the overlapping space with the target polygon. This search cost $O(n^2)$. Then, we build two spaces generated when inserting the target polygon. This step costs $O(r)$ with r the number of segments of the route. In the worst case, the route generated includes all the links of the graph, so it costs $O(n)$. In consequence, this step cost $O(n^2)$.

- Gravity Heuristic: This heuristic uses the Y coordinate to test. When we create a polygon, we previously stored the centroid on the polygon representation. In consequence, getting the Y coordinate is constant, so this step is order $O(1)$.
 - Multi-layer packing: This packing uses one of the previous heuristics to test the position. So the order of this heuristic is the same as the one used to pack multi-layers.
5. After we got the best position, we insert the target polygon on the mesh and we update the graph. To update the graph, we need to find the overlapping space, build the two new spaces when placing the target polygon, and update the links. Because the spaces of the layout can be concave, we can not use the intersection algorithm between convex polygons. We use the simplest algorithm of order $O(n^2)$, and then updating the graph is just updating the neighbors of the divided space. This update can cost at most the number of segments of the graph that is order $O(n)$. In conclusion, this step cost $O(n^2)$.
 6. Finally, checking if the target polygon closed the surrounding of their neighbors costs $O(c)$, and as the number of neighbors is negligible compared to the number of polygons of the mesh, then the step is order $O(1)$.

In summary, being p the number of polygons inserted by the algorithm, the algorithm costs $O(p^2n)$ for the gravity heuristic, and costs $O(p^2n^2)$ for the density heuristic. The number of polygons p is directly related to the number of vertices n so we can replace it. The final order for the gravity heuristic packing is $O(n^3)$, and for the density heuristic packing is $O(n^4)$.

5 Performance Experiments

To test the performance of the algorithm, we designed a variety of tests. The first batch of tests comprised a comparison between the two heuristics. The experiments considered an increment on the number of vertices of the packed polygons, changing the size of the packed polygons, packing polygons close to a regular polygon vs convex polygons generated randomly. These experiments test time and efficiency of the heuristics, with efficiency the percentage of space of the container covered by the inserted polygons. We ran all these experiments on an Intel Core i5-8400 CPU.

5.1 Heuristic Comparison

The results of the experiments were that the algorithm with the density heuristic reaches a maximum efficiency of 75% and with the gravity heuristic reaches a maximum efficiency of 80%. Also, the density algorithm takes 4000s when packing polygons of 20 vertices, but the gravity heuristic takes 40s to pack the same list. That difference in time happens in all the experiments, concluding that the gravity heuristic reaches its goal in less time. Figure 4 shows two examples of resulting meshes of each heuristic. The packing on the left shows the result

of using the density heuristic and on the right the mesh resulting of applying the gravity heuristic. The clear differences are that the generated meshes using the density heuristic contains spaces that have more area but zones of the mesh where it is more dense. Instead of using the gravity heuristic produces meshes with more uniformly distributed polygons. These differences reflect the results of the experiments showing why the density heuristic reaches less efficiency.

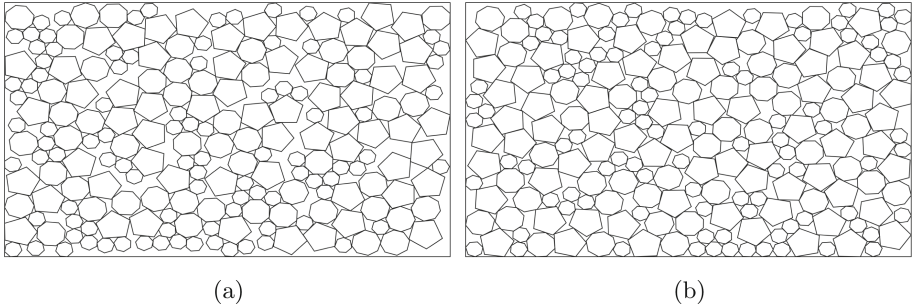


Fig. 4. Generated meshes when packing three different polygons of different size and number of vertices. (a) Mesh generated using the density heuristic and (b) mesh generated using the gravity heuristic.

5.2 Multi-Layer Packing

The last approach designed is the Multi-layer packing. This algorithm inserts layer by layer the polygons on the container. We inserted the polygon in ascending order by size of the polygon, i.e., the first layer comprised the polygons with the biggest radius and on each following layer the following biggest polygon. We tested the same experiments on this approach, reaching a maximum of 78% using the density heuristic for each layer. Figure 5 shows an example of a mesh got by the multi-layer packing algorithm. On that example we packed the polygons on three layers each with different size and a different number of vertices. The size of the polygon decrease on each passing layer. In a qualitative analysis these new meshes seems to be worse than the one layered meshes in space efficiency.

We compare a one layer container versus a multi-layer container, both experiments using the density heuristic. We made five different experiments: (a) packing with polygons of 5 vertices all same size, (b) packing with polygons with 5 or 7 vertices all same size, (c) packing polygons with 5 vertices and two different sizes, (d) packing polygons with 20 vertices all same size, and (e) packing polygons with 20 vertices and two different sizes. The multi-layer packing used different polygons on each of the layers, we used the polygons on ascending order by size, decreasing the size on each layer, i.e., on experiment (c) we used on the first layer the bigger polygons and then on the second layer used the smaller

ones. Figure 6 show the result of the experiments described and it show a similar behaviour of both approaches. The difference is that the multi-layer packing needs a bigger container to be more efficient in covered space. The multi-layer packing algorithm generates different meshes but with a 2% loss in efficiency.

6 Preliminary Simulation Results

In a recent work about polygonal meshes [12], a C++ library for the virtual element method [15] (VEM) was developed. Therein polygonal meshes were generated from a constrained Voronoi diagram [11] of the domain. The present work aims to extend the variety of meshes generated in [12] to be able to perform simulations of packing-based problems using the VEM.

To test a mesh generated using the proposed packing algorithm, the following diffusion problem is considered:

1. A unit square domain ($1\text{ m} \times 1\text{ m}$).
2. A 0°C applied to all the boundary nodes.
3. A conductivity of $1\text{ W}/(\text{mK})$ assigned to all the polygons.
4. A conductivity of $0.1\text{ W}/(\text{mK})$ assigned to all the spaces.
5. A heat source that varies according to the following equation:

$$b(x, y) = 32y(1 - y) + 32x(1 - x)$$

The foregoing diffusion problem is solved numerically using the VEM. This method can handle arbitrary polytopal meshes including non-convex polygons, and thus it is very appealing for simulations that use packing-based meshes.

Figure 7(a) shows the domain and mesh used in the diffusion problem. Figure 7(b) depicts the VEM temperature field. Finally, the heat flux is shown in Fig. 8.

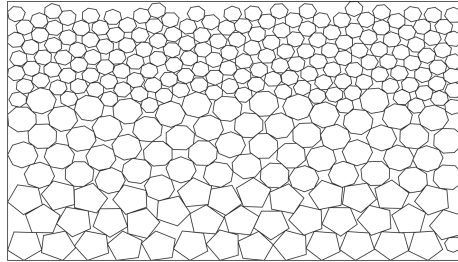


Fig. 5. Generated mesh when packing on a three layered container. The algorithm inserts the polygons depending in how we configure them. Here each layer has polygons with different size and a different number of vertices.

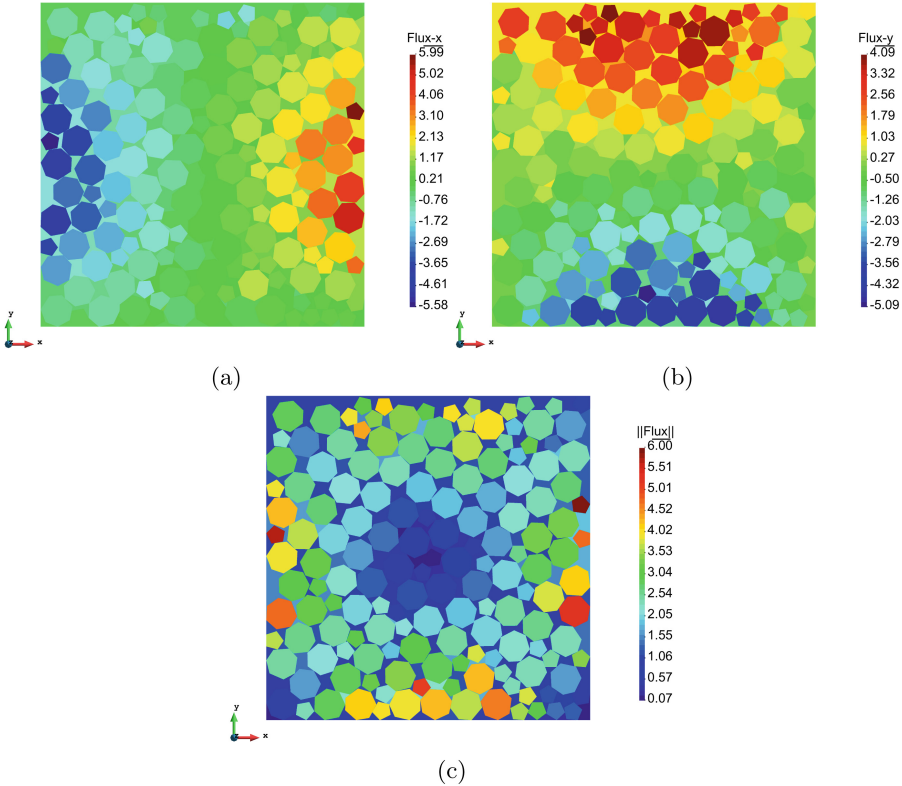


Fig. 8. Heat conduction in a unit square domain discretized with a packing mesh: (a) flux in the x -direction, (b) flux in the y -direction and (c) norm of the flux.

7 Conclusions

The experimental results showed that the gravity heuristic is better in CPU time and covers more area than the density heuristic. The gravity heuristic packing algorithm takes less CPU time than the density, behaving according to the theoretical order shown in Sect. 4. It also improves the efficiency by 5% with respect to the density heuristic: 80% vs 75%, respectively.

The experiments testing the multi-layer packing algorithm showed an interesting new type of resulting meshes. However, there is a decrease of 2% in efficiency compared to the gravity heuristic packing, but this decrease is not significant.

Thus, the meshes resulting from the packing algorithms can be useful for modeling of rock and porous media. As an example, we illustrated its potential by solving a diffusion problem, where the discretized domain consisted in polygons and spaces with different conductivities. Due to the arbitrary shape of polygons and spaces that are generated by the packing algorithm, the virtual element

method was used to solve the diffusion problem numerically. This shows that, for the virtual element method, a new variety of problems can be solved using packing-based meshes.

References

1. Agarwal, P.K., Flato, E., Halperin, D.: Polygon decomposition for efficient construction of Minkowski sums. *Comput. Geom.* **21**(1–2), 39–61 (2002)
2. Berkey, J.O., Wang, P.Y.: Two-dimensional finite bin-packing algorithms. *J. Oper. Res. Soc.* **38**(5), 423–429 (1987)
3. Collins, C.R., Stephenson, K.: A circle packing algorithm. *Comput. Geom.* **25**(3), 233–256 (2003)
4. Feng, Y., Han, K., Owen, D.: An advancing front packing of polygons, ellipses and spheres. In: *Discrete Element Methods: Numerical Modeling of Discontinua*, pp. 93–98. ASCE Library (2002)
5. Feng, Y., Han, K., Owen, D.: Filling domains with disks: an advancing front approach. *Int. J. Numer. Meth. Eng.* **56**(5), 699–713 (2003)
6. Hopper, E., Turton, B.C.: An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *Eur. J. Oper. Res.* **128**(1), 34–57 (2001)
7. Huang, W., Chen, D., Xu, R.: A new heuristic algorithm for rectangle packing. *Comput. Oper. Res.* **34**(11), 3270–3280 (2007)
8. Li, Y., Ji, S.: A geometric algorithm based on the advancing front approach for sequential sphere packing. *Granular Matter* **20**(4), 1–12 (2018). <https://doi.org/10.1007/s10035-018-0829-7>
9. Löhner, R., Oñate, E.: An advancing front technique for filling space with arbitrary separated objects. *Int. J. Numer. Meth. Eng.* **78**(13), 1618–1630 (2009)
10. Mohar, B.: A polynomial time circle packing algorithm. *Discr. Math.* **117**(1–3), 257–263 (1993)
11. Okabe, A., Boots, B., Sugihara, K., Chiu, S.N.: *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley Series in Probability and Statistics, 2nd edn. Wiley (2009)
12. Ortiz-Bernardin, A., Alvarez, C., Hitschfeld-Kahler, N., Russo, A., Silva-Valenzuela, R., Olate-Sanzana, E.: **veamy**: an extensible object-oriented C++ library for the virtual element method. *Numer. Algorithms* **82**, 1–32 (2018)
13. Pinelis, I.: Cyclic polygons with given edge lengths: existence and uniqueness. *J. Geom.* **82**(1–2), 156–171 (2005). <https://doi.org/10.1007/s11075-018-00651-0>
14. Shimrat, M.: Algorithm 112: position of point relative to polygon. *Commun. ACM* **5**(8), 434–435 (1962)
15. da Veiga, B.L., Brezzi, F., Cangiani, A., Manzini, G., Marini, L.D., Russo, A.: Basic principles of virtual element methods. *Math. Models Methods Appl. Sci.* **23**(01), 199–214 (2013)
16. Williams, G.B.: Earthquakes and circle packings. *J. D’Analyse Mathématique* **85**(1), 371–396 (2001)
17. Žalik, B.: Two efficient algorithms for determining intersection points between simple polygons. *Comput. Geosci.* **26**(2), 137–151 (2000)