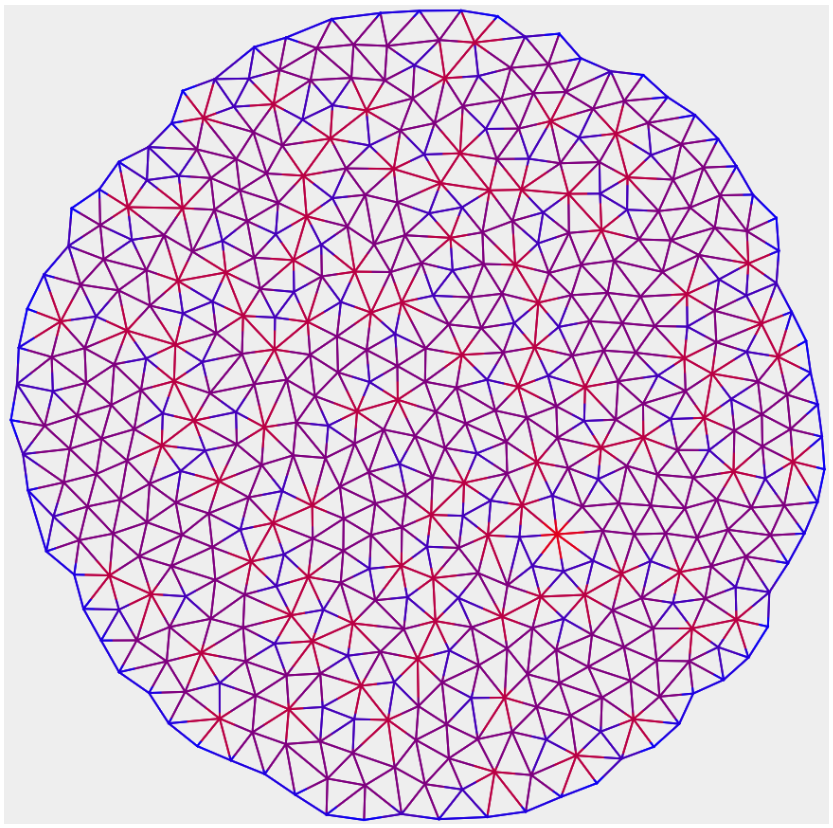
Canning for Amorphous Blob Computing

TER report

Auteur :
Lucas Labouret
M1 QDCS, Université Paris-Saclay
lucas.labouret@universite-paris-saclay.fr

Encadrant :
Frédéric Gruau
LISN
frederic.gruau@universite-paris-saclay.fr



Contents

Introduction **3**

A What is Blob Computing ? **3**
- A.1 Definition and Structure of a Computing Medium . . . 3
- A.2 An Example of Computation : the Voronoï Diagram . . . 5
- A.3 Objective : Cannings . . . 6

B Preliminary Work **7**
- B.1 Delaunay Triangulation . . . 7
- B.2 Farthest Point Optimization . . . 8

1 Borders and Media **8**
- 1.1 Soft Borders . . . 8
- 1.2 Hard Borders . . . 8
- 1.3 About Hybrid Borders . . . 9

2 Canning and Evaluation **9**
- 2.1 Vertex Cannings . . . 9
- 2.2 From Vertex to Total Cannings . . . 10
- 2.3 Evaluation . . . 13
  - 2.3.1 Definition . . . 13
  - 2.3.2 Some results . . . 13

3 GUI **14**

Conclusion **15**

References **15**

Introduction

This TER took place at the LISN from January to April 2025 under the supervision of Frédéric Gruau. It is the continuation of a research project I did under him last year as part of my Licence degree's curriculum. It is part of a long term research project about the concept of "blob computing"[4]. During this TER, I performed research about the simulation of blob computing on classical computers, and wrote a GUI application that allows to visualize and interact with my results.

A What is Blob Computing ?

To understand the exact subject of this TER, it is important to first introduce what blob computing means and how it works.

The ultimate goal of the project is to develop a new paradigm of computation that is inherently arbitrarily scalable through the use of *space* as a resource. This kind of computation is becoming increasingly useful, for example with the recent development of swarm robotic[7].

We aim to achieve this through the use of arbitrarily many processing elements (PEs) distributed through space and locally connected. These PEs, which can be severely lacking in power on their own, create a computing medium where virtual "blobs" can form and evolve. These blobs are the main primitive we use for making computations.

A.1 Definition and Structure of a Computing Medium

In its most general form, a computing medium is simply a set of processing elements distributed in some space which are locally connected. Here, we focus exclusively on a specific kind of media called "triangulated computing media", hence the term "computing medium" in the rest of this report will refer to those unless specified otherwise. A triangulated medium is represented as a weakly Delaunay-triangulated graph where each vertex, edge, and face is associated with a processing element capable of executing minimal computation, storing information, and communicating with its neighbors.

Vertices, edges and faces are called "simplicial loci" (or "S-loci" for short). They determine the overall structure of their medium. Each S-locus controls a set of "transfer loci" (or "t-loci"). T-loci come in pairs that handle the communication between two S-loci. For example, if a vertex was connected to an edge, there would be a pair of t-loci (one belonging to the vertex and the other belonging to the edge) between them.

Obviously, there are 3 types of S-loci : Vertex, Edge, Face. T-loci are divided into 6 types :

- Ve linking a Vertex to an Edge
- Vf linking a Vertex to a Face

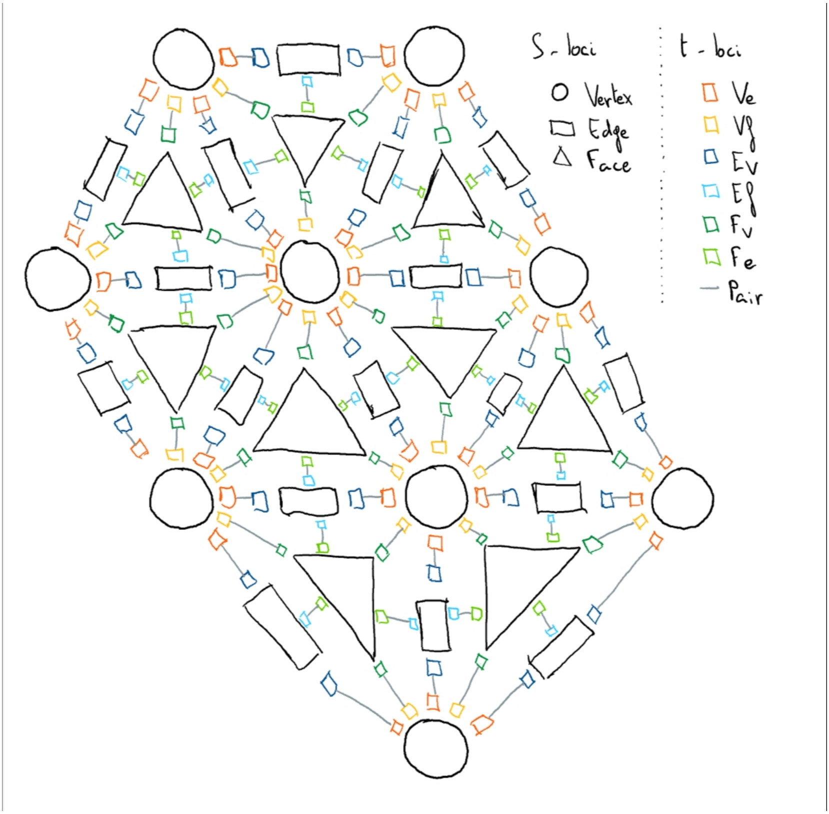


Fig. 1: Example of triangulated medium.

- Ev linking an Edge to a Vertex
- Ef linking an Edge to a Face
- Fv linking a Face to a Vertex
- Fe linking a Face to an Edge

Figure 1 show an example of how all these loci are put together to form a medium. For reference, a vertex controls any number of Ve's and Vf's, an edge controls two Ev's and two Ef's, and a face controls three Fv's and three Fe's.

A medium is said to be crystalline if a repeating structure emerges in the placement if its vertices. It is amorphous when no such structure emerges. Additionally, it is homogeneous if the average point density is approximately constant and there are no "holes" or "clusters", and it is isotropic if the directions of the edges of the medium follow a uniform distribution over $[0, \pi[$. This report mainly focuses on amorphous homogeneous isotropic (AHI) media.

For now, we consider PEs to be immobile and synchronous, although lifting these two constraints will probably be the subject of future research.

Blobs form the computational basis of blob computing. A blob of a given

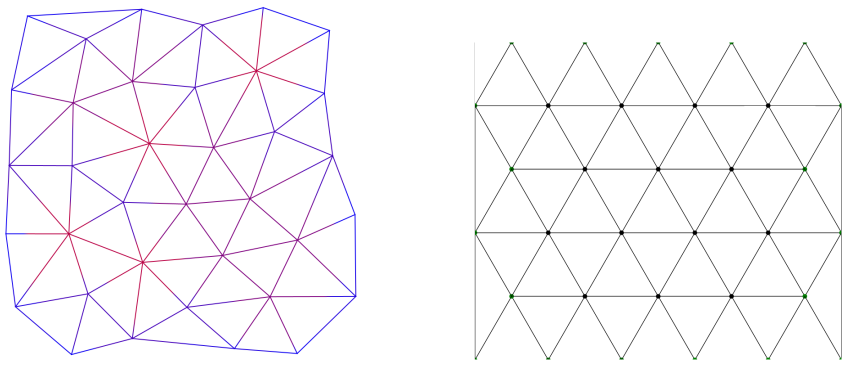


Fig. 2: An AHI medium (left) VS a crystalline hexagonal medium (right).

medium M is a connected sub-graph M' of M such that all the vertices of M' share a common property (e.g.: all their values are even, or all their values are 1, etc.). Formally, it is a "True" connected component of a boolean field over the vertices (see [2]).

Importantly, blobs are persistent through time, meaning that two blobs at two different points in time can be considered one and the same under the right conditions, even though blobs are able to change their shape over time. This is possible for two reasons :

1. Blobs maintain their connexity, they typically don't merge or divide unless instructed otherwise.
2. Blobs only change shape at their border : at each time step they can grow to unoccupied vertices adjacent to their border, or they can lose vertices that are part of their border.

Therefore, two blobs at two consecutive timestep are "the same blob" if they only differ at their border. We can then extend the notion to any number of timestep through its transitive closure.

A.2 An Example of Computation : the Voronoï Diagram

[2] develops tools to program on computing media. As an example, it shows how to build a (discrete) Voronoï diagram from a set of "seeds" in a medium.

The basic idea is the following : we start with a set of "seeds" distributed over the vertices of the network. Each seed is now the center of a blob. Then, at each synchronous step, the blobs will spread to their adjacent vertices if this wouldn't cause them to merge with another blob, until no blob can spread further. It can be implemented with very few types of local operations that are applied simultaneously to every S-locus in the network :

- Multi-cast
  The S-locus copies its value into its t-loci of a given type.

- Rotation

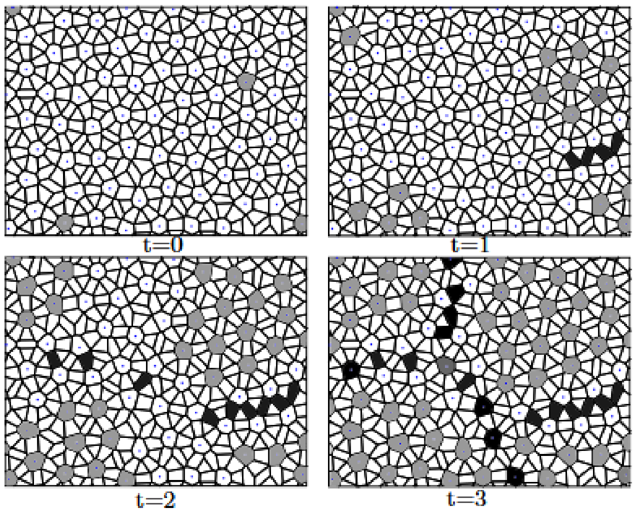


Fig. 3: Computation of a discrete Voronoï diagram.
Cells marked with a dot are the vertices of the medium. Gray cells form blobs. Black cells mark the frontiers between blobs. At t=0, blobs only cover the seeds. At t=3, blobs cover the Voronoï regions of the medium.
Source : [2]

Since t-loci are spatially arranged around their S-loci, we can define a clockwise and counter-clockwise neighbor. T-loci can perform logical operations on their and their neighbors' values.

- Reduction
  The S-locus stores the result of a commutative and associative operation applied to the values of its t-loci of a given type.
- Transfer
  Each of the S-locus' t-loci exchanges its value with the value of its pair.

In fact, clever association of these primitives allows to program everything a computing medium is capable of, such as sorting arrays of integer or multiplying matrices[5].

A.3 Objective : Cannings

Up until now, Gruau mainly focused on hexagonal crystalline media. He built a plateform allowing efficient simulation of blob computation on classical hardware using SIMD[2,3].

The point of this TER is to allow the use of SIMD to optimize the simulation of AHI media as well. We do this through the development of the notion of "total canning", or simply "canning". Given a medium M, a canning is a set of 9 injective maps which assign some integer coordinates to each locus of M :

1. $\{\text{Vertex}_M\} \mapsto \mathbb{N}^{a_1}$
2. $\{\text{Edge}_M\} \mapsto \mathbb{N}^{a_2}$
3. $\{\text{Face}_M\} \mapsto \mathbb{N}^{a_3}$
4. $\{\text{Ve}_M\} \mapsto \mathbb{N}^{a_4}$
5. $\{\text{Vf}_M\} \mapsto \mathbb{N}^{a_5}$
6. $\{\text{Ev}_M\} \mapsto \mathbb{N}^{a_6}$
7. $\{\text{Ef}_M\} \mapsto \mathbb{N}^{a_7}$
8. $\{\text{Fv}_M\} \mapsto \mathbb{N}^{a_8}$
9. $\{\text{Fe}_M\} \mapsto \mathbb{N}^{a_9}$

   where $a_1, ..., a_9 \geq 1$ are specific to each canning method and designate the number of coordinates assigned to each type of locus.

Through misuse of language, the term ”canning” will designate either this set of maps, or the algorithm used to build it.

The objective is twofold : firstly, we want to find a way to measure the efficiency of a canning on a given medium. Secondly, we want to create a canning that is efficient on most -if not all- media.

B Preliminary Work

In this section, I briefly introduce some work I did last year during another internship with Gruau to generate AHI media. While it isn't the main focus of this year's TER, it is what made it possible in its current form.

B.1 Delaunay Triangulation

A Delaunay triangulation is a particular type of triangulation where a triplet of vertices form a face iif no other vertex can be found inside the circumcircle of the triplet. We relax this definition slightly by allowing the hull of the graph not to be convex. It presents interested properties for our purpose as the edges of the triangulation tend to remain ”localized” (i.e. vertices that are connected tend to be spatially close).

I used the triangulation algorithm described in [6]. However, there is a bug in my implementation where vertically-aligned are often handled incorrectly, which I couldn't fix before the end of my internship last year. Since this problem wasn't particularly consequential this year either, I didn't take the time to fix it this year either.

Additionally, the FPO algorithm (see B.2) performs a lot of removals and insertions to the set of vertices. Currently, each time this happens, the entire set is retriangulated. This is a vastly inefficient approach to the problem. If I had the time, I would've liked to implement the removal and addition algorithms described in [6, 1].

B.2 Farthest Point Optimization

The Farthest Point Optimization algorithm[8] allows the construction of the irregular yet homogeneous sets of points that we need for blob computing. It yields better results than more common algorithm like Poisson-Disk sampling or Lloyd algorithm, and fits naturally into the project as it relies heavily on the Delaunay triangulation.

The implementation of FPO in my case wasn't straightforward. This is because the article applied the algorithm to sets of points in the unit torus, while the media use bounded subsets of the Euclidean plane instead. In particular, I had to adapt the algorithm to work around the existence of borders with no predetermined shape, and that can contain fixed points that cannot be moved even though they can still influence the placement of other points.

1 Borders and Media

The border of a medium is the set composed of both the edges separating the outer face of the graph from the inner faces of the graph, and the vertices at the ends of those edges.

The method we chose to handle the borders of our media is crucial, as impacts how the media are optimized, how they are canned afterward, and decide some useful properties they have for performing computations. Ultimately, we decided to study two broad types of borders : "soft" and "hard" borders.

1.1 Soft Borders

Soft borders are the least restrictive of the two types. A medium has a soft border if all of its vertices are allowed to be moved anywhere in a bounded, connected region of the Euclidean plane during FPO. This region can take any shape.

This method of handling the border tends to connect vertices that would normally be too far apart from each other if they were on the inside of the medium, breaking its locality. To solve this issue, we allow the iterative removal of the edges of the border that connects points which would otherwise be "too far apart".

We initially considered two types of soft bordered media : soft square media, where the vertices are confined into a square, and soft circle media, where the vertices are confined into a circle. However, we quickly abandoned soft circle media as they currently don't have any application for computing.

1.2 Hard Borders

A hard bordered medium is a media whose border (both the edges and the vertices) is fixed even before FPO. Hard borders are a lot more restrictive than soft borders, but they also offer many advantages over them. In particular, since we can decide the placement of the vertices of the borders, it means they can arranged in a way that makes them easy to mirror or to fold into tori. This has interesting properties once the media is used for computation.

However, one must be careful when constructing such media, as over/under-filling can lead to a loss of homogeneity or locality near the border.

We focused on two types of hard border : a hard square with evenly placed vertices, and a hard rectangle with ratio 1: $\sqrt{3}$. Initially, we also tried to work with a border following a hexagonal lattice to approach previous work, but we ended dropping the idea as we would either experience the loss of homogeneity/locality mentioned above, or FPO would end converging back to a hexagonal crystal, losing the isotropicism of the media.

1.3 About Hybrid Borders

We also briefly considered using a hybrid type of border by allowing the vertices of the borders to move in a restricted manner, we ultimately decided against it as it made our hypothesis more restrictive than pure soft borders while bringing non of the advantages that hard borders have.

2 Canning and Evaluation

Before we begin our search for total cannings, let us define the notion of partial canning. A partial canning of a medium M is simply a subset of a total canning of M (or the algorithm used to build it). In particular, we focus on vertex cannings, i.e. partial cannings which only contain the map from the vertices of M to their coordinates. From there, we will exhibit a generic method for build total cannings using vertex canning as a base. Finally, we will show how we can evaluate the efficiency of different cannings so we can compare them.

2.1 Vertex Cannings

To align with the way crystaline media are hendled, we set $a_1 = 2$, i.e. we want to assign 2D integer coordinates to each vertex of the medium. The problem now becomes equivalent to placing the vertices in a 2D grid. The vertex canning is valid if all the vertices have been placed in the grid and if each cell of the grid contains at most one vertex. The coordinates of each vertex then becomes the 2D index of its cell. Based on this idea, I have developped two diffrent vertex cannings with opposite philosophies :

1. **TopDistanceXStorted** vertex canning
   This vertex canning was developed to rely as much as possible on the graph structure of the medium, making as little use as possible of its embedding. It was developed quite early in the project, before soft-bordered media appeared, so it only works on hard-bordered media. It works in the following way :
   - First, compute the minimum graph-distance of each vertex to any vertex on the topside of the border. All vertices at distance Y from the top are

placed in line Y of the grid.
- Second, place the leftmost vertex of each line in the first cell of its line, the second leftmost vertex of each line in the second cell of its line, repeating until all vertices are placed in the grid.

2. **RoundedCoordinates** vertex canning
This vertex canning was developed to rely exclusively on the spatial embedding of the medium, completing ignoring its graph structure. It works in the following way :
- First set a counter to 1.
- Multiply the spatial coordinates of each vertex by the counter and round the result. Use the two integers obtained as a 2D index to put each vertex in the grid.
- If two vertices share the same cell, increment the counter by one and go back to the previous step. Otherwise, we have a vertex canning.

Regarding the TopDistanceXStorted algorithm, I first tried sorting the lines using the distance to the leftside border, however it resulted in invalid vertex cannings where two vertices would share the same cell.

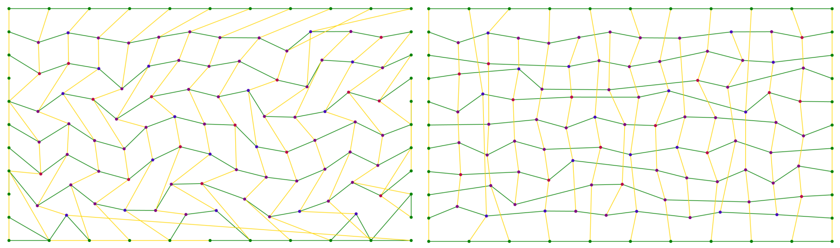


Fig. 4: TopDistanceXSorted (left) next to RoundedCoordinates (right) applied to the same medium.
Two vertices are joint by a green line if they are consecutive on a line of the grid, and a yellow line if they are consecutive in a column.

Intuitively from figure 4, we can infer that RoundedCoordinates is better as it creates a more "grid-like" canning, whereas the columns in TopDistanceXSorted seem to "drift" to the left. We will confirm this intuition later.

2.2 From Vertex to Total Cannings

Now that we can build vertex cannings, we want a method to convert them into total cannings.

To do this, we first define the "belonging" relation between loci, such that we can say that some locus A belongs to some other locus B. We do this so that can "index" B relative to A. Given some vertex canning VC, we define the relation in the following way :

- Vertices belong to no one.

- We iterate over the lines of VC, going from the top-most line to the bottom-most line in order.
  Then we iterate over each vertex V of the line, going from the left-most to the right-most in order.
  Each Edge and Face connected to V now belong to V, unless it already belongs to another vertex.

- Each t-locus belongs to the S-locus that has control over it.

Now, given a vertex v, an edge e and a face f we write
- y_v and x_v the coordinates of v in VC,
- $E(v)$ the set of edges belonging to v,
- $F(v)$ the set of faces belonging to v,
- $Ve(v)$ the set of Ve t-loci belonging to v
- $Vf(v)$ the set of Vf t-loci belonging to v
- $Ev(e)$ the set of Ev t-loci belonging to e
- $Ef(e)$ the set of Ef t-loci belonging to e
- $Fv(f)$ the set of Fv t-loci belonging to f
- $Fe(f)$ the set of Fe t-loci belonging to f

Since the edges, faces, Ve's and Vf's will be indexed relative to vertices, we set $a_2 = a_3 = a_4 = a_5 = a_1 + 1 = 3$. Similarly, Ev's and Ef's will be indexed relative to edges so we set $a_6 = a_7 = a_2 + 1 = 4$, and Fv's and Fe's will be indexed relative to faces so we set $a_8 = a_9 = a_3 + 1 = 4$. Now, the attribution of coordinates works in the following way :

- **Edges** :
  For each vertex v
  - Select a first edge. If it exists and it is in $E(v)$, it is the edge joining v to the next vertex in its line. Otherwise, start on the left of v and walk clockwise around it, selecting the first edge in $E(v)$ that you cross.
  - Sort $E(v)$ clockwise around v starting from your first edge.
  - Number your edges, starting at 0 and incrementing by 1 in the order of your sorted $E(v)$.
  - Edge number p now has coordinates (p, y_v, x_v).

- **Faces** :
  For each vertex v
  - Select a first face. If it exists and it is in $F(v)$, it is the face joining v to the next vertex in its line and to v's next neighbor clockwise. Otherwise, start on the left of v and walk clockwise around it, selecting the first face in $F(v)$ that you cross.
  - Sort $F(v)$ clockwise around v starting from your first face.
  - Number your faces, starting at 0 and incrementing by 1 in the order of your sorted $F(v)$.
  - Face number p now has coordinates (p, y_v, x_v).

- **Ve's** :
  For each vertex v
  - Select a first Ve. If it exists, it is the Ve linking v to the edge joining v to the next vertex in its line. Otherwise, start on the left of v and walk clockwise around it, selecting the Ve in $Ve(v)$ linking v to the first edge in $E(v)$ that you cross.
  - Sort $Ve(v)$ clockwise around v starting from your first Ve.
  - Number your Ve's, starting at 0 and incrementing by 1 in the order of your sorted $Ve(v)$.
  - Ve number p now has coordinates (p, y_v, x_v).

- **Vf's** :
  For each vertex v
  - Select a first Vf. If it exists, it is the Vf linking v to the face joining v to the next vertex in its line and to v's next neighbor clockwise. Otherwise, start on the left of v and walk clockwise around it, selecting the Vf in $Vf(v)$ linking v to the first face in $F(v)$ that you cross.
  - Sort $Ve(v)$ clockwise around v starting from your first Ve.
  - Number your Vf's, starting at 0 and incrementing by 1 in the order of your sorted $Vf(v)$.
  - Vf number p now has coordinates (p, y_v, x_v).

- **Ev's** :
  For each Edge e
  - We note (p, y_e, x_e) its coordinates.
  - The Ev linking to the vertex e belongs to has coordinates $(0, p, y_e, x_e)$.
  - The other Ev has coordinates $(1, p, y_e, x_e)$.

- **Ef's** :
  For each Edge e
  - We note (p, y_e, x_e) its coordinates, and v the vertex it belongs to.
  - We consider that e "faces" v, which let us distinguish between a right Ef and a left Ef.
  - The right Ef has coordinates $(0, p, y_e, x_e)$.
  - The left Ef has coordinates $(1, p, y_e, x_e)$.

- **Fv's** :
  For each face f
  - We note (p, y_f, x_f) its coordinates, and v the vertex it belongs to.
  - We call "first Fv" the Fv linking to v
  - Starting from the first Fv, we number the Fv's 0 through 2 clockwise around f.
  - Fv number q now has coordinates (q, p, y_f, x_f)

- **Fe's** :
  For each face f
  - We note (p, y_f, x_f) its coordinates, and v the vertex it belongs to.
  - We call "first Fe" the Fe clockwise-next to the Fv linking to v
  - Starting from the first Fe, we number the Fe's 0 through 2 clockwise around f.
  - Fe number q now has coordinates (q, p, y_f, x_f)

2.3 Evaluation

2.3.1 Definition

Finally, now that we have total cannings, we want a way to evaluate them. First we remember that the point of a canning is to allow the use of SIMD to simulate the execution of a computing media. Therefore, we want can decide how good a canning is by deciding how many operations are needed to simulate each an execution step, and how well they can be parallelized. While it is possible to compute this exactly, for now we only focus on finding an upper bound. We do this by noticing that since

1. each t-locus in a canning is indexed relative to vertex and
2. all communications are uniquely determined by the triangulated graph, which is itself (with probability 1 in general) uniquely determined by the placement of the vertices,

we can find an upper bound for each communication type with a single computation over the canning of the vertices, which will be accurate up to a constant factor that depends on the type of communication[1].

This upper is given by a Δy, which designates the maximum distance between two lines of vertices that interact with each other, and a Δx, which designates the maximum distance between two columns of vertices that interact with each other. Together, they allow computing $\mathcal{A}_{max} = (2\Delta y + 1)(2\Delta x + 1)$ the maximum area of interaction around any vertices.

Together with the constant factor we defined earlier, this $\mathcal{A}_{max}$ forms the upper bound were looking for. Therefore $\mathcal{A}_{max}$ is a suitable measurement of the performance of a canning, which we wish to minimize.

2.3.2 Some results

Overall, as we guessed earlier, the RoundedCoordinates algorithm yields much better results than the TopDistanceXSorted algorithm. RoundedCoordinates produces an $\mathcal{A}_{max}$ oscillating between 25 and 49 regardless of the size the medium. Meanwhile TopDistanceXSorted produces an $\mathcal{A}_{max}$ that grows with the size of the medium. It outperforms RoundedCoordinates for very small media (less than 50 vertices) but quickly becomes much worse with an $\mathcal{A}_{max}$ reaching upward of 150 for 1000 vertices, and upward of 200 for 2000 vertices.

[1]For example, there is exactly 2 Ev per edge, and there are at most 3 times as many edges as there are vertices, so Ev-Ve communication will have a constant factor of 6.

RoundedCoordinates seems to perform overall better on (reasonably defined) hard bordered media. This aligns with our observation that soft borders join vertices that would normally be too far apart from each other, perhaps indicating that we are too lenient when initially correcting the border.

TopDistanceXSorted seems to get worse the wider the media get. This is further evidence that the "drift" in the columns we observed earlier is responsible for the poor performance of this method.

3 GUI

A large portion of the TER was focused on the development of a software equipped with a GUI that allows the user to

1. Randomly generate new media, save them, and load them.
2. Optimize media with different parameters.
3. Visualize different aspects of media.
4. Compute and visualize different cannings.

If developing a GUI for this application ended up being time-consuming, it was incredibly helpful all around while debugging everything else. Overall it was probably a net positive on development time alone, and that is still neglecting the advantage of being able to see what I am doing.

Below are some pictures of the kind of visualization the GUI provides.

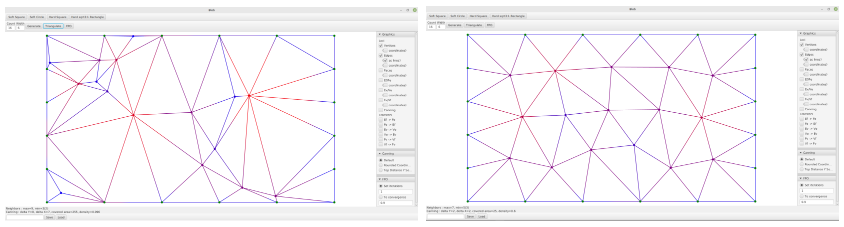
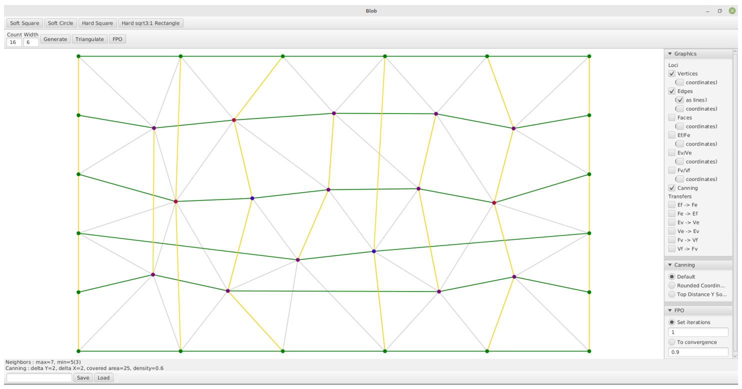
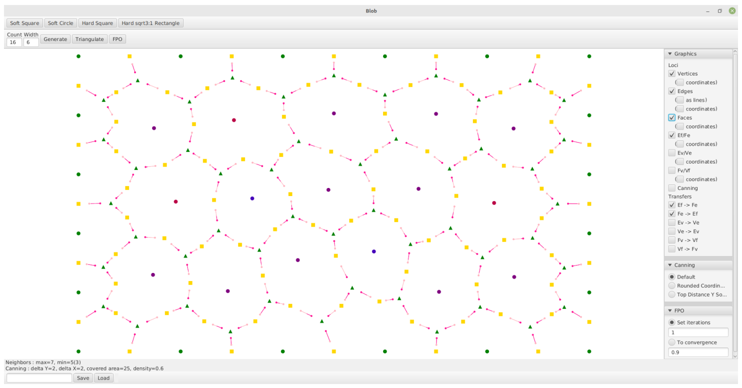


Fig. 5: "Canonical" view of a medium before (left) and after (right) optimization with FPO.



```
Count 16  Width 6
Graphics
Loci
[✓] Vertices
  ( [ ] coordinates)
[✓] Edges
  ( [✓] as lines)
  ( [ ] coordinates)
[ ] Faces
  ( [ ] coordinates)
[ ] Ef/Fe
  ( [ ] coordinates)
[ ] Ev/Ve
  ( [ ] coordinates)
[ ] Fv/Vf
  ( [ ] coordinates)
[✓] Canning
Transfers
[ ] Ef -> Fe
[ ] Fe -> Ef
[ ] Ev -> Ve
[ ] Ve -> Ev
[ ] Fv -> Vf
[ ] Vf -> Fv
Canning
(●) Default
( ) Rounded Coordin...
( ) Top Distance Y So...
FPO
(●) Set iterations
1
( ) To convergence
0.9
Neighbors : max=7, min=5(3)
Canning : delta Y=2, delta X=2, covered area=25, density=0.6
```

Fig. 6: View of a partial canning (see 2.1) of the vertices of the medium in figure 5



```
Count 16  Width 6
Graphics
Loci
[✓] Vertices
  ( [ ] coordinates)
[✓] Edges
  ( [ ] as lines)
  ( [ ] coordinates)
[✓] Faces
  ( [ ] coordinates)
[✓] Ef/Fe
  ( [ ] coordinates)
[ ] Ev/Ve
  ( [ ] coordinates)
[ ] Fv/Vf
  ( [ ] coordinates)
[ ] Canning
Transfers
[✓] Ef -> Fe
[✓] Fe -> Ef
[ ] Ev -> Ve
[ ] Ve -> Ev
[ ] Fv -> Vf
[ ] Vf -> Fv
Canning
(●) Default
( ) Rounded Coordin...
( ) Top Distance Y So...
FPO
(●) Set iterations
1
( ) To convergence
0.9
Neighbors : max=7, min=5(3)
Canning : delta Y=2, delta X=2, covered area=25, density=0.6
```

Fig. 7: View of the Ef-Fe pairs in the medium of figure 5.

Conclusion

References

[1] Olivier Devillers. "On Deletion in Delaunay Triangulations". In: *International Journal of Computational Geometry and Applications* 12 (2002), pp. 193–205. DOI: 10.1142/S0218195902000815. URL: https://inria.hal.science/inria-00167201.

[2] Frédéric Gruau. "A parallel data-structure for modular programming of triangulated computing media." In: *Natural Computing* 22.4 (2023), pp. 753–766. ISSN: 1572-9796. DOI: 10.1007/s11047-022-09906-1. URL: https://doi.org/10.1007/s11047-022-09906-1.

[3] Frédéric Gruau. *platform-CA2*. URL: https://github.com/fredgruau/platform-CA2.

[4] Frédéric Gruau et al. “BLOB computing”. In: *Proceedings of the 1st Conference on Computing Frontiers*. CF ’04. Ischia, Italy: Association for Computing Machinery, 2004, 125–139. ISBN: 1581137419. DOI: 10.1145/977091.977111. URL: https://doi.org/10.1145/977091.977111.

[5] Frédéric Gruau, Christine Eisenbeis, and Luidnel Maignan. “The foundation of self-developing blob machines for spatial computing”. In: *Physica D: Nonlinear Phenomena* 237.9 (2008). Novel Computing Paradigms: Quo Vadis?, pp. 1282–1301. ISSN: 0167-2789. DOI: https://doi.org/10.1016/j.physd.2008.03.046. URL: https://www.sciencedirect.com/science/article/pii/S0167278908001255.

[6] Leonidas Guibas and Jorge Stolfi. “Primitives for the manipulation of general subdivisions and the computation of Voronoi”. In: *ACM Trans. Graph.* 4.2 (Apr. 1985), 74–123. ISSN: 0730-0301. DOI: 10.1145/282918.282923. URL: https://doi.org/10.1145/282918.282923.

[7] Iñaki Navarro and Fernando Matía. “An Introduction to Swarm Robotics”. In: *International Scholarly Research Notices* 2013.1 (2013), p. 608164. DOI: https://doi.org/10.5402/2013/608164. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.5402/2013/608164. URL: https://onlinelibrary.wiley.com/doi/abs/10.5402/2013/608164.

[8] Thomas Schlömer, Daniel Heck, and Oliver Deussen. “Farthest-point optimized point sets with maximized minimum distance”. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG ’11. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2011, 135–142. ISBN: 9781450308960. DOI: 10.1145/2018323.2018345. URL: https://doi.org/10.1145/2018323.2018345.