

INDEX

Summary
Introduction and objectives
Development
Results and conclusions
Appendices
Bibliography

Part 1. SUMMARY

Voronoi diagrams have practical and theoretical applications to a large number of fields, mainly in science, technology and visual art. The aim of my project is to study the inverse Voronoi diagram problem and design, compare and analyze different strategies for its solving. The inverse Voronoi diagram problem consists on detecting whether a given plane partition is a Voronoi diagram and finding the seed points that would generate such a partition. For a partition which does not come from a Voronoi diagram, it would be interesting to find the best fitting Voronoi diagram. At this point, we need a way to measure how good a candidate solution is. We will be using the total symmetric difference between the two partitions for that.

Note that despite the search space is bounded, it being continuous grants an infinite number of solution candidates. Therefore, we will try to solve the problem using different metaheuristics, which makes it impossible to tell whether the obtained solution is optimum.

The basic steps of all the strategies are:

- For each input, calculate a set of seed points from which we will start.
- Move each point slightly following some criteria and check if we improved the current best solution.
- Repeat last step until we cannot keep improving or we are satisfied with the result.

RESUMEN

Los diagramas de Voronoi tienen aplicaciones tanto prácticas como teóricas en muchos ámbitos, la mayoría relacionados con la ciencia y tecnología, aunque también se aplica en otros como el arte visual. El objetivo de mi proyecto es estudiar el problema inverso del diagrama de Voronoi y diseñar, comparar y analizar diferentes estrategias para su resolución. Dicho problema consiste en detectar si una partición dada es o no un diagrama de Voronoi, y calcular las semillas que lo generan en caso afirmativo. Para particiones que no lo son, sería interesante encontrar el diagrama de Voronoi que mejor se aproxima. Para ello, necesitaremos alguna forma de evaluar como de buena es una solución. Usaremos la diferencia simétrica entre las dos particiones para tal fin.

Nótese que pese a que el espacio de búsqueda es acotado, al ser continuo hay infinitos candidatos que habría que probar, lo cual es inviable. Por tanto, trataremos de resolver el problema mediante diferentes metaheurísticas, aunque en ningún caso podremos asegurar que la solución obtenida es óptima.

El esquema básico de todas las estrategias es el siguiente:

- Para cada datos de entrada, calcular un conjunto de puntos semilla a partir de los cuales comenzaremos la búsqueda.
- Desplazamos los puntos ligeramente según algún criterio y comprobamos si hemos conseguido mejorar nuestra solución, actualizando ésta en ese caso.
- Repetimos el paso anterior hasta que no podamos seguir mejorando o hasta que la solución obtenida se considere lo suficientemente buena.

Part 2. INTRODUCTION

There are some concepts the reader should be familiar with in order to follow this document without problems. Those are:

- Partition of a set. Partition of the unit square.
- Voronoi diagrams.
- Symmetric difference.

1. PARTITION OF A SET. PARTITION OF THE UNIT SQUARE

A partition of a set is a grouping of the set's elements into non-empty subsets, in such a way that every element is included in one and only one of the subsets.

For example, if we consider the set $X = \{1, 2, 3, 4, 5\}$, three different partitions could be:

- $\{1, 3\}, \{2, 4\}, \{5\}$
- $\{1, 2, 3\}, \{4, 5\}$
- $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$

In our case, we will be working with partitions of the unit square. That is, a set of polygons such that the union is the unit square and the intersection of two polygons is, at most, a segment. Note that there might be points contained in more than one polygon.

2. VORONOI DIAGRAMS

A Voronoi diagram is a partitioning of a plane into regions (called Voronoi cells) based on distance to points (called seeds or generators) in a specific subset of the plane. In this project we will only be using the simplest case of Voronoi diagrams: the seeds are given as a finite set of points in the Euclidian plane.

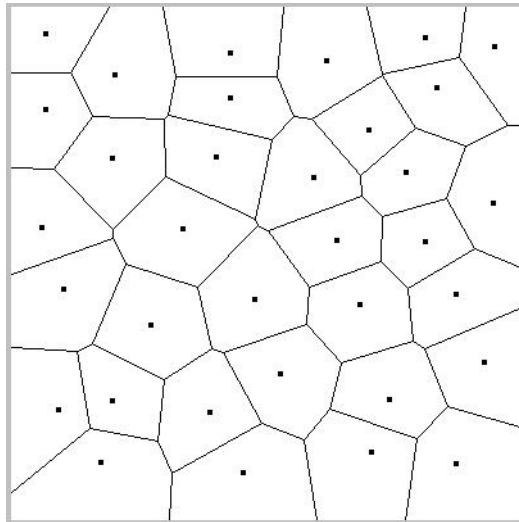


FIGURE 2.1. Visual representation of a Voronoi diagram inside the unit square.

The lines that appear in a Voronoi diagram are the points of the plane that are equidistant to two or more of the nearest seeds.

3. SYMMETRIC DIFFERENCE

For measuring how good a given solution is, we will rely on the symmetric difference as an indicator. For two given polygons A and B, the symmetric difference can be calculated as follows:

$$SD(A, B) = Area(A \cup B) - Area(A \cap B)$$

Applying this concept to two partitions, we can define the symmetric difference of two partitions as the sum of all the symmetric differences for each pair of related polygons.

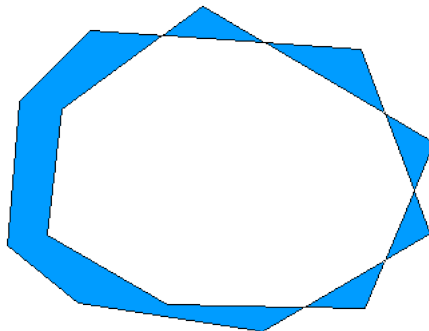


FIGURE 3.1. In blue, we can see a representation of the symmetric difference of two polygons. The highlighted area is the one that belongs to the symmetric difference. As we can see, it is the area that belongs to either polygon, but not both.

Since in this project we will only be interested in working with convex polygons, we can use that to our advantage and calculate the symmetric difference in a different way: we will subtract from the unit square area, the areas of all the intersections of each pair of related polygons. This will be explained more in detail later, in the development chapter.

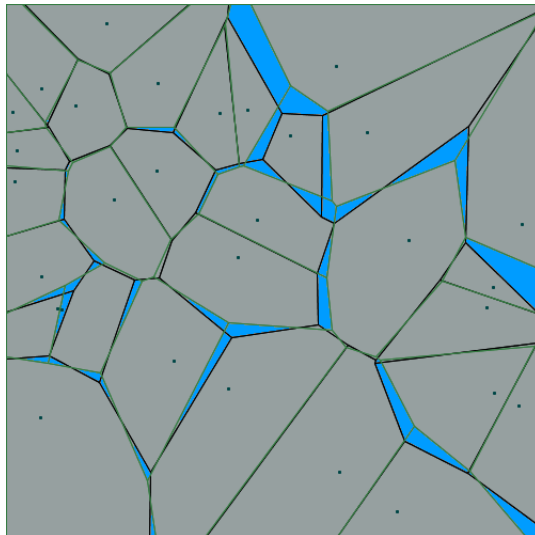


FIGURE 3.2. The new method would start with whole square painted in blue. We then subtract all the intersections of each pair of polygons. Graphically, the symmetric difference would be the total blue area of the picture in proportion to the area of the whole square.

In order to avoid issues with scaling, we will be treating the symmetric difference as a ratio with the total area of the unit square, so a 0% value will mean the two partitions are exactly the same.

4. INVERSE VORONOI PROBLEM

As mentioned earlier, the inverse Voronoi diagram problem consists on detecting whether a given plane partition is a Voronoi diagram and finding the seed points that would generate such a partition. From the mathematical point of view, it is interesting that, for partitions that do come from a Voronoi diagram, the problem can be solved in $O(n)$ time, n being the number of seeds of such a diagram.

Proof. We know the partition comes from a Voronoi diagram. TODO: Proof...

But what happens when the partition does not come from a Voronoi diagram? Even if it did, due to computers inaccuracy when representing floating-point numbers, the input data we have will not perfectly be the one that would be generated from a Voronoi diagram, but rather an approximation of it. Thus, we want to explore different methods of fitting an arbitrary partition, whether it comes from a Voronoi diagram. \square

5. GENERALIZED INVERSE VORONOI PROBLEM

Another problem related to the ones exposed, but beyond of the scope of this project, is the generalized inverse Voronoi problem. This problem consists of, given a plane partition, calculate a Voronoi diagram such that the given partition is contained on it. That is, add the necessary seeds so you get a partition which contains the given one.

6. VORONOI DIAGRAMS IN REAL LIFE

TODO: Talk about the mouthbreeder fish example of the book.

If one has the chance to go for a walk in Madrid, he/she might find itself in front of the “Teatros del Canal”. If he is familiar with Voronoi diagrams, something will catch his eye: some parts of the decoration of that building resemble those diagrams! Further in this project, we will adjust them and see how well they fit as a Voronoi diagram.



FIGURE 6.1. “Teatros del Canal” facade. As it can be seen, it looks very similar to a Voronoi diagram.

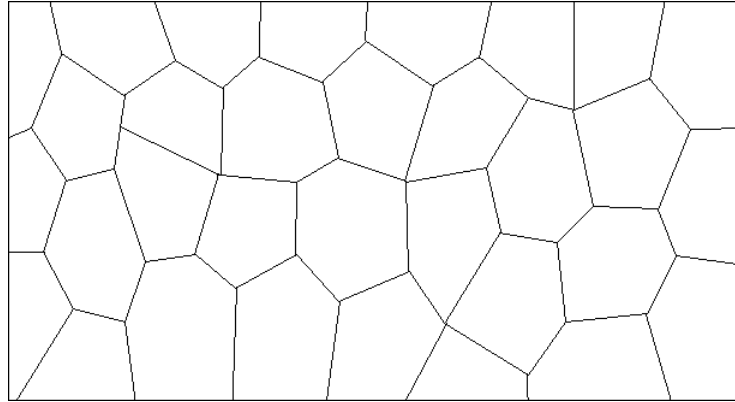


FIGURE 6.2. “Teatros del Canal” facade transcription. We will later use this to see and compare the results of the different fitting methods.

OBJECTIVES

The objectives of this project are:

- Fit a given partition of the unit square as much as we can through Voronoi diagrams. Therefore, the variable we want to minimize is the resulting

symmetric difference between the input and the solution given. We want to implement different techniques and compare their results in a set of test cases.

- If the given partition is in fact a Voronoi diagram, we would like to arrive to that conclusion. Even though this looks trivial, it will be very difficult to get there for non trivial input, since the number of local minimums that have to be avoided to find the seeds increases in a higher magnitude compared to the number of polygons.

Part 3. DEVELOPMENT

1. WHY PROCESSING. PROS AND CONS OF JAVA

Despite Java not being the most efficient language out there for crunching raw numbers, I decided to develop my program in that programming language for a number of reasons:

- First of all, I had already used it in the past for a different project, and one of the biggest advantages of it is the ease to display information in the screen.
- The aim of this project is not to design an algorithm that can finish execution in the least amount of time, but rather, study and compare different techniques and their results.
- It is extremely portable across the different operative systems.
- Java's data structures come in handy for our problem. ArrayList's are used constantly in the code to contain information about the partitions and polygons.
- The number of regions we are going to be treating in our research will be 200 maximum, a reasonable number for any modern programming language.

2. CALCULATING THE VORONOI DIAGRAM FROM A SET OF SEED POINTS

Even though I am using an external library for calculating the Voronoi diagrams, here I will briefly explain the most common methods for calculating a Voronoi diagram.

//TODO

3. LINE CLIPPING. POLYGON CLIPPING

There are two situations in the problem in which we have to calculate the intersection of two polygons: when clipping a polygon to the unit square and when clipping two polygons.

The first one is needed because, theoretically, there will always be some Voronoi region whose area will be infinite. In order to get rid of that issue, we must treat our Voronoi diagram in order to clip all regions into a bounded area, in our case, the unit square.

The second one is useful when we want to calculate the area of the intersection of two polygons for calculating the symmetric difference. This is doable since we will be working with convex polygons. An easy way to do it is calculate the actual intersection, and then, get the area of that polygon.

There is a simple, common operation in both tasks, to which these two problems can be reduced to: clip a polygon with a single line. If we have a solid function that can do this task, then we can rely on it to calculate any intersection of convex polygons (note that the unit square is a convex polygon too), repeating the process for all lines of one of the polygons against the other.

The algorithm works as follows:

Suppose we want to clip the polygon ABCDEFG with the blue line. Note that this polygon is not convex, but for explanation purposes, we will apply the algorithm to it. Also note that the line has a direction, given by the blue arrow.

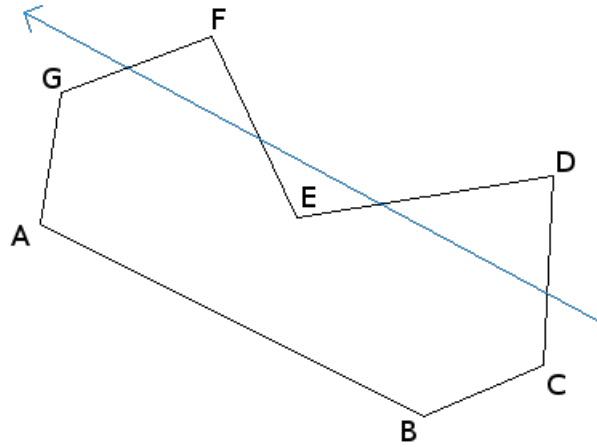


FIGURE 3.1. Starting point. We will clip the polygon with the given line (in blue). Starting point will be A.

The first step of the clipping algorithm is to set a starting vertex. We will set A as our starting point. We must check in which side of the blue line it lies. We can do this using the cross product of two points of the line and our point. Since in the implementation we will be working with positively oriented polygons, we are interested only in points which lie on the left of the line. The point A is lying on the left, so we add it to the solution and continue.

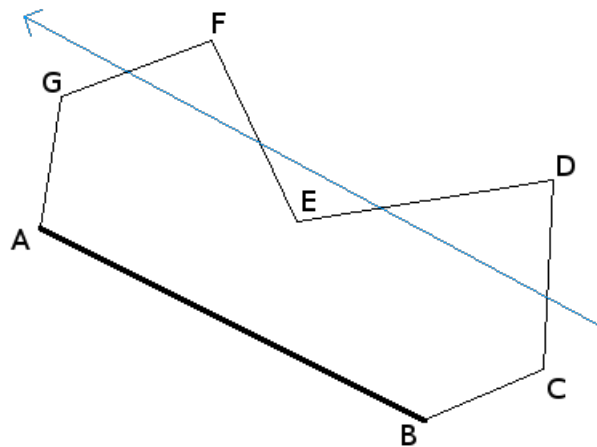


FIGURE 3.2. Since point B lies on the left of the line, add it to the result.

The next step is to select the next point and see whether it lies in the same side of the blue line. Since B also lies on the same side than A, we add it to the solution too and keep going.

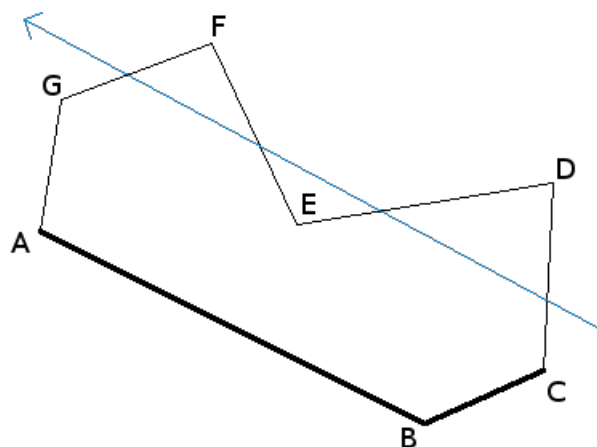


FIGURE 3.3. Since C is still on the left of the line, we add it to the final result too.

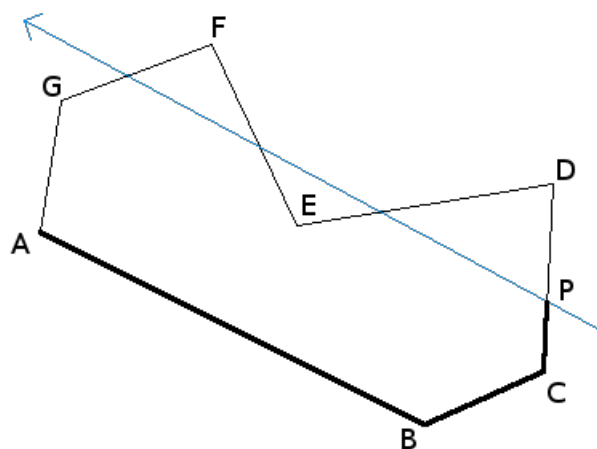


FIGURE 3.4. The next point, D, does not lie on the left of the blue line. Since last one did, it means an intersection point P has to exist. Add P to the final result.

When we check D, we see that it is lying on the right side of the line. Then, we know that this point will be clipped by the line. Therefore, we need to find point P, which is the intersection of CD with the blue line. We will be adding this point to the solution.

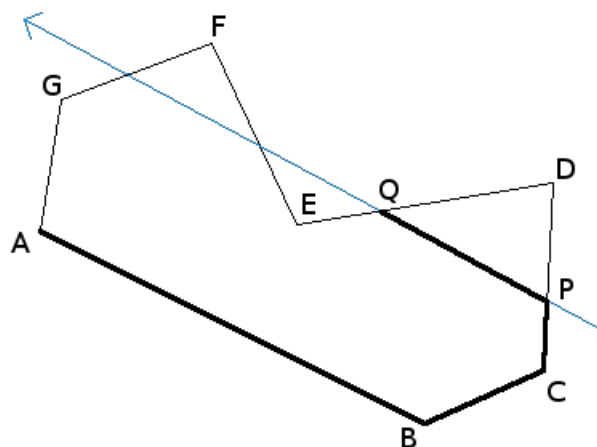


FIGURE 3.5. Current point D lies on the right of the line. Since E lies on the left, it means another intersection Q must exist. Add Q to the final result.

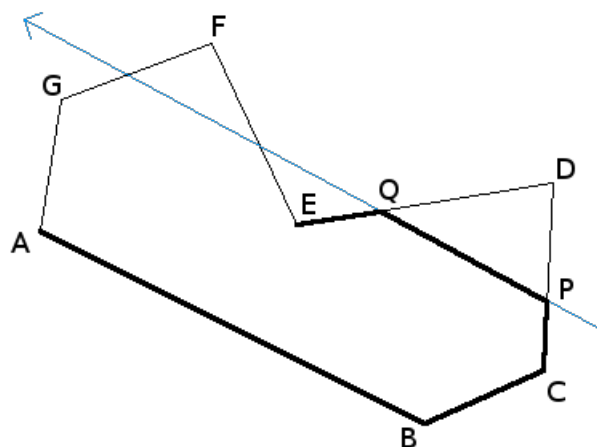


FIGURE 3.6. Point E lies on the left. Add it to the solution.

We keep repeating the same process until we arrive to the starting point.

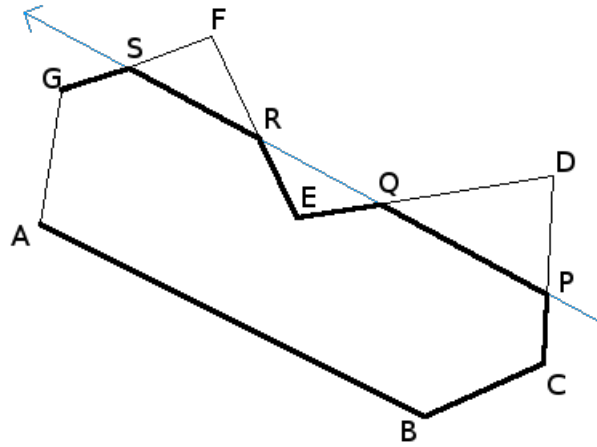


FIGURE 3.7. On the left again. Add G to the result.

When we arrive at the starting point, we have finished clipping the polygon with the line.

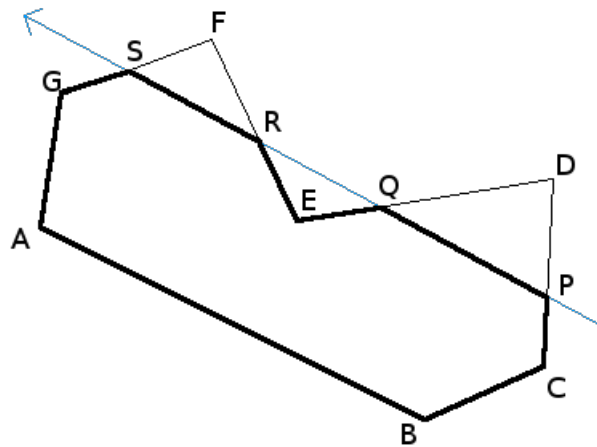


FIGURE 3.8. We arrived to A, our starting point.

For iterating a polygon A with another polygon B, given that both of them are correctly oriented, we can do it clipping A with all lines formed by B.

4. INPUT DATA

The input data for the problem is a partition of the unit square. That is, a set of polygons whose union is the unit square and, for every two polygons, its intersection is at most a straight line. More precisely, each polygon is given as a set of ordered 2D points. At first, we know we will be working only with convex polygons (that

is, all internal angles are less or equal than 180 degrees). The input data will be read from a *.txt file in which each line will contain a whole partition.

More in detail, this is the structure of the input data files:

5. OUTPUT DATA

The program generates various files after being executed. The files will be located in the “voronoi” folder, and will be the following:

- output.csv: CSV file in which each line represents each iteration of the algorithm. Each line contains information of where each seed point was at that stage and the symmetric difference in that precise step.
- before.png: Image that displays the first approximation of the Voronoi diagram.
- after.png: Image that displays the final approximation of the Voronoi diagram.

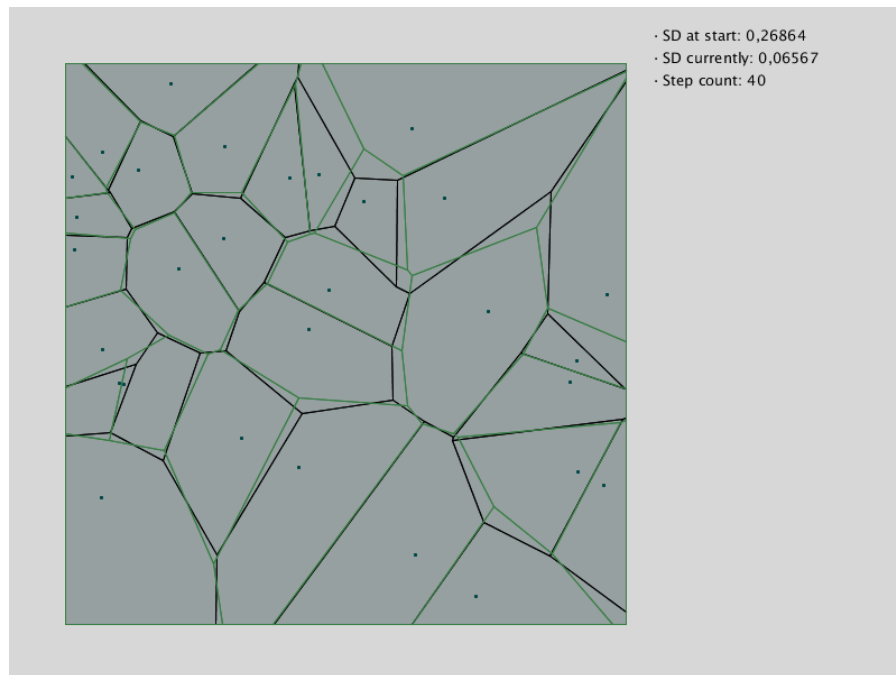


FIGURE 5.1. Output image generated by the program. It shows the final result, as well as some other information of the process.

6. GRADIENT METHOD

The gradient method is a simple heuristic used in optimization problems where the search space is not treatable. It consists on, at any given state, compare neighbour solutions against the current best one. If they are better, mark it as the new best solution and repeat. If the minimum or maximum we are looking for is bounded, there will be a point at which we will not be able to continue improving. That will be our solution.

For example, consider you want to find the maximum value of a function. You start at a random point with a corresponding value. You will have to move to the left if the value at some distance to the left is higher, or to the right if the value at some distance to the right is higher. The problem with this is that we can easily get stuck in a local maximum.



FIGURE 6.1. The starting point will be the left arrow. As we can see, this method would indicate us to go right in order to get to a better solution.

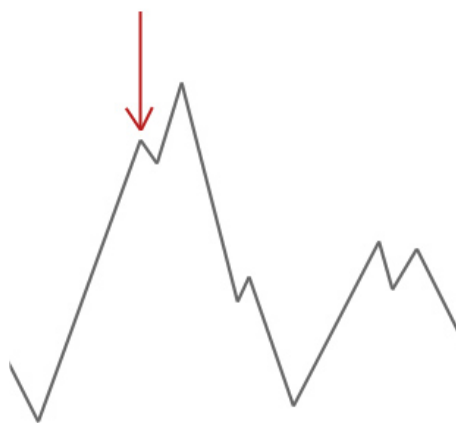


FIGURE 6.2. Ending point using a simple gradient method.

However, when reaching this point, it would check for its neighbours and see that whatever we do, we would get a worse solution. Therefore, the peak pointed by the red arrow would be our solution.

7. IMPROVED GRADIENT METHOD: IMPLEMENTING STEPS

One step closer to the simulated annealing, but still very similar to the gradient method is the gradient method with variable step. It consists on iterating different

gradient methods, but varying the step size we are taking on each one. We start with a high value for the step, and when we cannot improve anymore for that step, we reduce it to other value. This way, we will adjust our solution more than with using the gradient method. We will also be able to get further away from the initial solution. However, lots of solutions are lost and this method is greatly dependent on the step values chosen.

8. SIMULATED ANNEALING

Simulated annealing copies a phenomenon in nature—the annealing of solids—to optimize a complex system. Annealing refers to heating a solid and then cooling it slowly. Atoms then assume a nearly globally minimum energy state. //TODO: Explain more and better.

8.1. Getting rid of the “repetition” problem: randomizing point order in each state. Given the nature of our problem, it is easy to see that the order in which we will move our points will play a huge role in the final result. In order to avoid the cyclic effect, what we want to do is move them in a pseudo-random mannner every iteration.

Part 4. Results and conclusions

The following sets have been used during the study of the different methods.

$$A = \{0.05, 0.04, 0.01, 0.005, 0.001\}$$

$$B = 0.05, 0.03, 0.01$$

$$C = 0.1, 0.01, 0.005$$

PARTITION EXAMPLE 8

	Parameters	Starting SD	Ending SD	Total adjustment	Steps	Adj/step
Grad.	0.05	0.26864	0.12694	0.14170	6	0.02116
Grad.	0.01	0.26864	0.10517	0.16347	20	0.00526
Grad.	0.005	0.26864	0.10109	0.16755	32	0.00316
Grad. steps	Set A	0.26864	0.06518	0.20346	29	0.00225
Grad. steps	Set B	0.26864	0.07564	0.19300	15	0.00504
Grad. steps	Set C	0.26864	0.06018	0.20847	37	0.00163
Sim. Annealing	Set A	0.26864	0.06449	0.20416	31	0.00208
Sim. Annealing	Set B	0.26864	0.07564	0.19300	15	0.00504
Sim. Annealing	Set C	0.26864	0.06788	0.20076	27	0.00251

TABLE 1. Results obtained for lineNumber = 8. This partition is made up of 30 polygons.

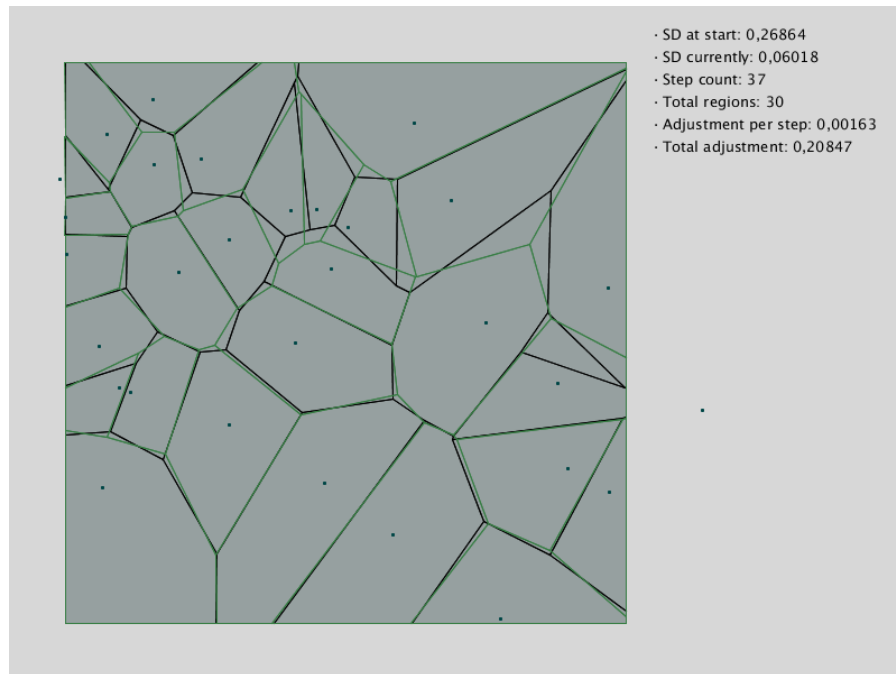


FIGURE 0.1. Result of the best tested configuration for lineNumber = 8.

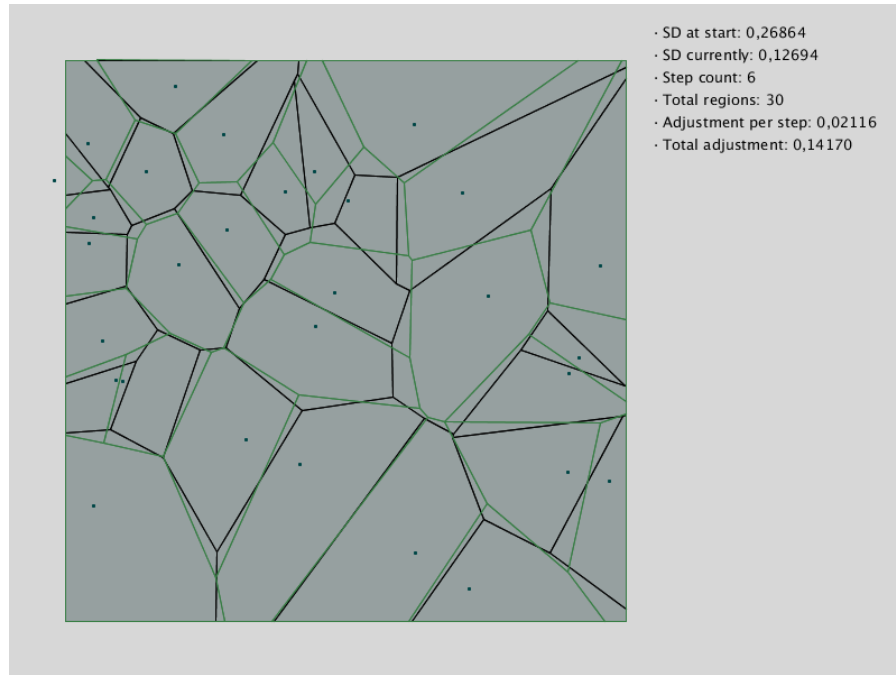


FIGURE 0.2. Result of the worst tested configuration for lineNumber = 8.

PARTITION EXAMPLE 1

	Parameters	Starting SD	Ending SD	Total adjustment	Steps	Adj/step
Grad.	0.05	0.21728	0.09680	0.12049	5	0.01936
Grad.	0.01	0.21728	0.06632	0.15097	13	0.00510
Grad.	0.005	0.21728	0.05689	0.16039	27	0.00211
Grad. steps	Set A	0.21728	0.04463	0.17625	63	0.00071
Grad. steps	Set B	0.21728	0.05221	0.16507	17	0.00307
Grad. steps	Set C	0.21728	0.03407	0.18321	29	0.00117
Sim. Annealing	Set A	0.21728	0.04463	0.17625	65	0.00069
Sim. Annealing	Set B	0.21728	0.05221	0.16507	17	0.00307
Sim. Annealing	Set C	0.21728	0.03407	0.18321	29	0.00117

TABLE 2. Results obtained for lineNumber = 1. This partition is made up of 20 polygons.

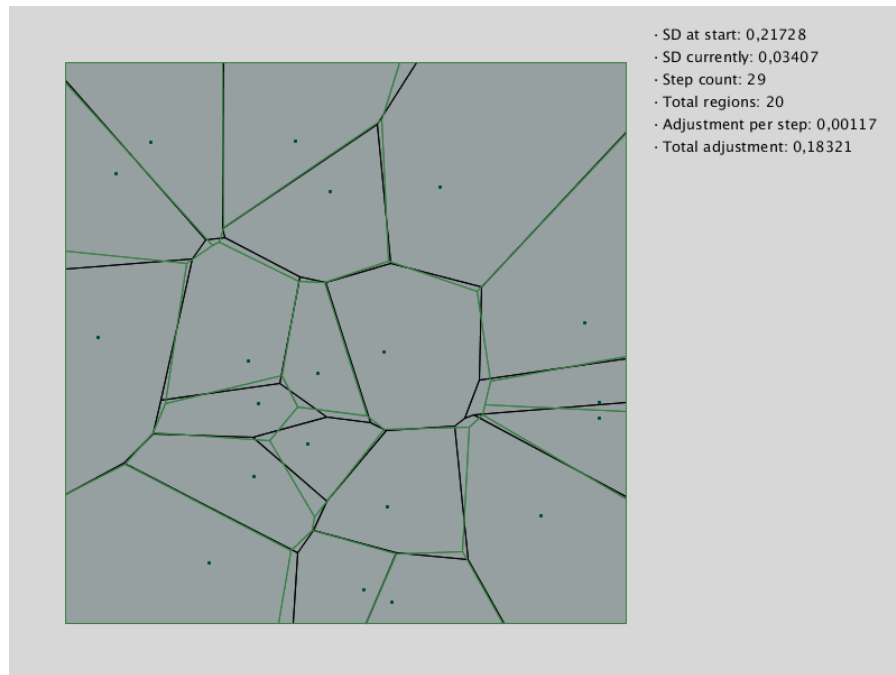


FIGURE 0.3. Result of the best tested configuration for lineNumber = 1.

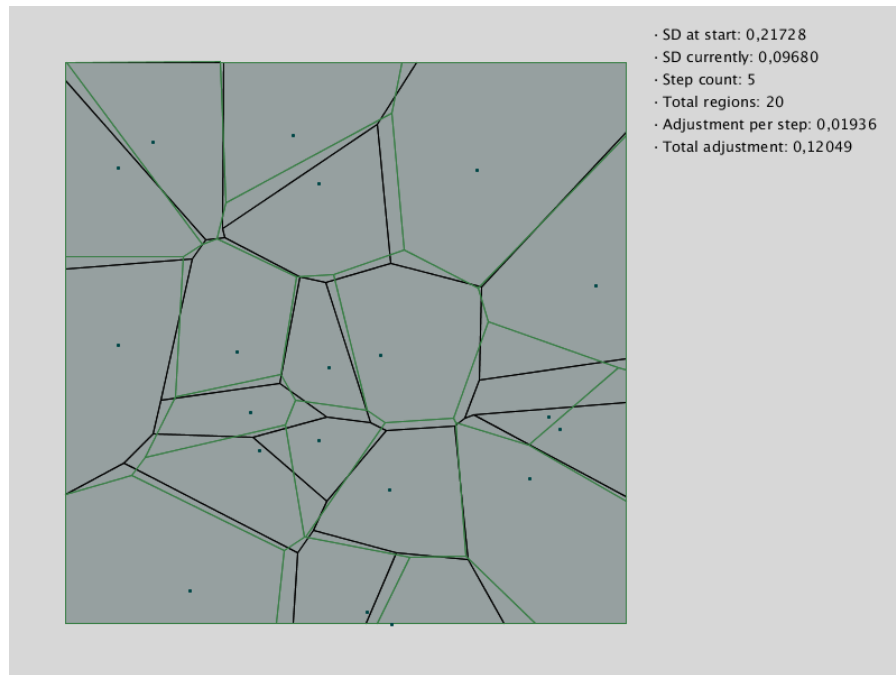


FIGURE 0.4. Result of the worst tested configuration for lineNumber = 1.

Part 5. Bibliografía

- Spatial tessellations: Concepts and Applications of Voronoi Diagrams (Second Edition) - Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, Sung Nok Chiu
- <https://processing.org> – Open source programming language and IDE in which the whole project is coded
- <http://leebyron.com/mesh/> - External processing library used for calculating the Voronoi diagrams
- <http://www.lyx.org/> - Document processor used for writing this thesis
- https://github.com/Flood1993/TFG_voronoi - Git repository containing everything about this project
- <http://apmonitor.com/me575/index.php/Main/SimulatedAnnealing> - Simulated annealing
- <http://www.exploringbinary.com/why-0-point-1-does-not-exist-in-floating-point/> - Inaccuracy of computers when dealing with floating-point numbers