INDEX

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.

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.

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 square unit.
- Voronoi diagrams.
- Symmetric difference.

Partition of a set. Partition of the square unit.

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 square unit. That is, a set of polygons such that the union is the square unit and the intersection of two polygons is, at most, a segment. Note that there might be points contained in more than one polygon.

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.

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.

Symmetric difference.

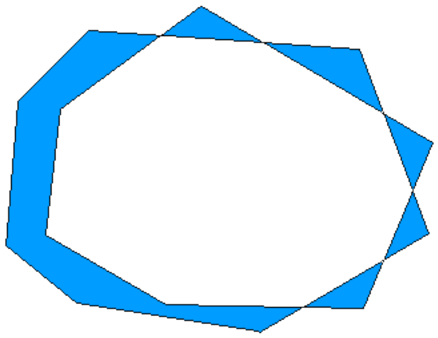
For measuring how good a given solution is, we will rely on the symmetric difference as an indicator. For two given polygons A and B, the symmetric difference can be calculated as follows:

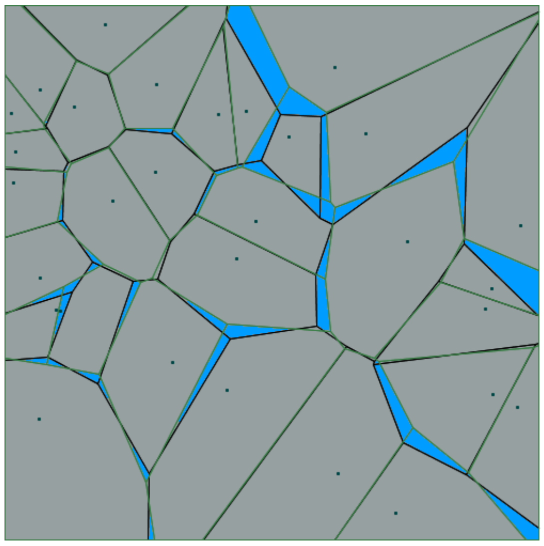
$$SD(A,B) = Area(A \bigcup B) - Area(A \bigcap B)$$

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.

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 substract from the square unit 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.

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





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...

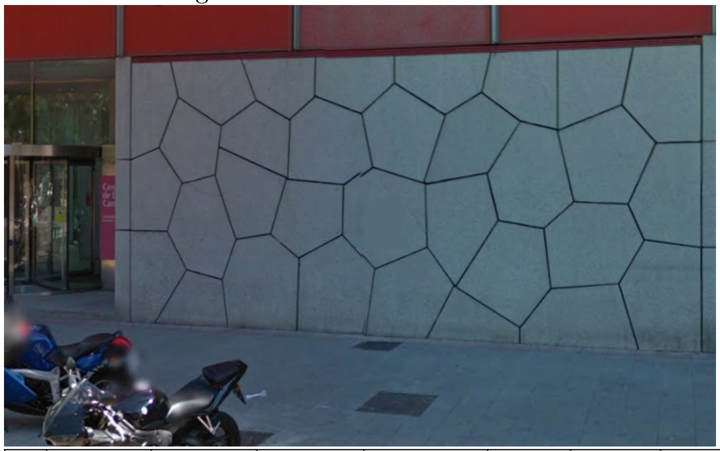
But what happens when the partition does not come from a Voronoi diagram? Even if it did, due to computers inaccuracy when representing floating-point numbers, the input data we have will not perfectly be the one that would be generated from a Voronoi diagram, but rather an approximation of it. Thus,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

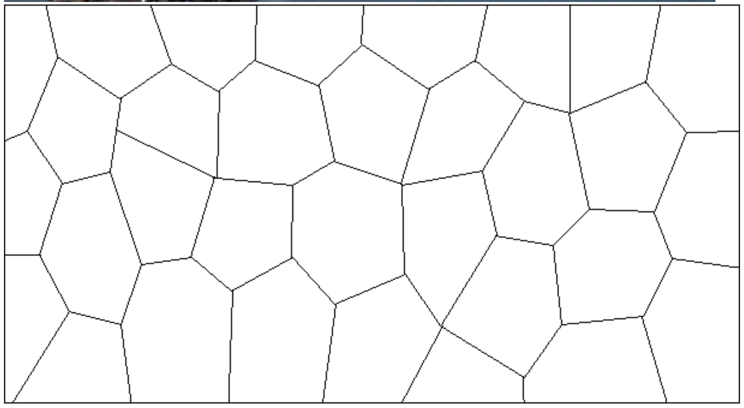
Generalized inverse Voronoi problem.
Another problem related to the ones exposed, but beyond of the scope of this project, is the generalized inverse Voronoi problem. This problem consists of, given a plane partition, calculate a Voronoi diagram such that the given partition is

contained on it. That is, add the necessary seeds so you get a partition which contains the one given.

Voronoi diagrams in real life.
TODO: Contar lo de los japoneses y los peces del libro de Manuel.

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 that person 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.





OBJECTIVES

The objectives of this project are:

- Fit a given partition of the square unit 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.

DEVELOPMENT

Why Processing. Pros and cons of Java.

Despite Java not being the most efficent language out there for crunching raw numbers, I felt like I had to go with Processing for the development 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.

Calculating the Voronoi diagram from a set of seed points.

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

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 square unit and when clipping two polygons.

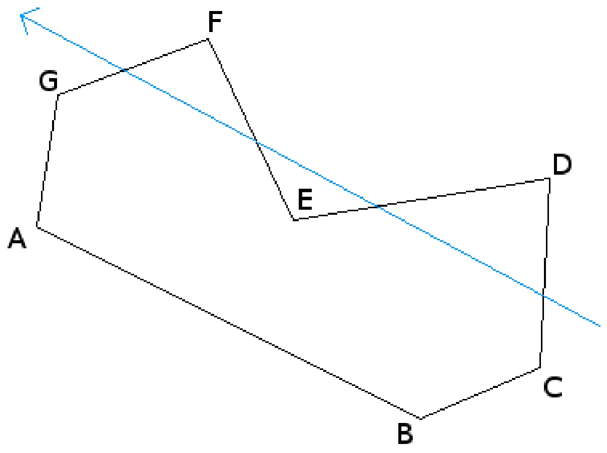
The first one is needed because, theoretically, there will always be some Voronoi region whose area will be infinite. In order to get rid of that issue, we must treat our Voronoi diagram in order to clip all regions into a bounded area, in our case, the square unit.

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.

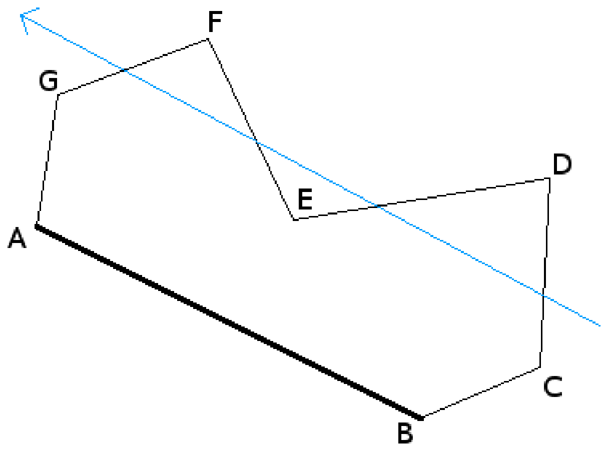
There is a simple, common operation in both tasks, to which these two problems can be reduced to: clip a polygon with a single line. If we have a solid function that can do this task, then we can rely on it to calculate any intersection of convex polygons (note that the square unit is a convex polygon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

The algorithm works as follows:

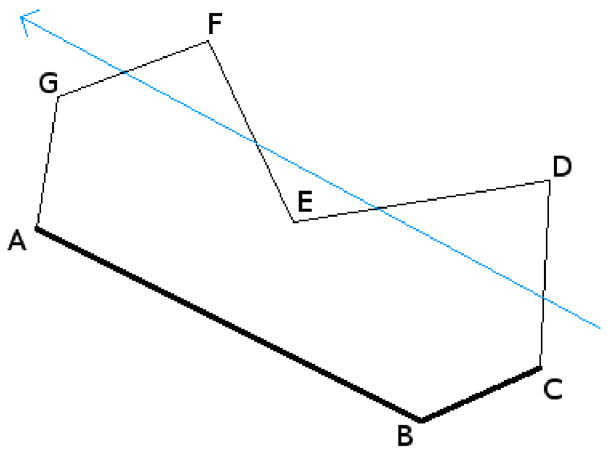
Suppose we want to clip the polygon ABCDEFG with the blue line. Note that this polygon is not convex, but for explanation purposes, we will apply the algorithm to it. Also note that the line has a direction, given by the blue arrow.



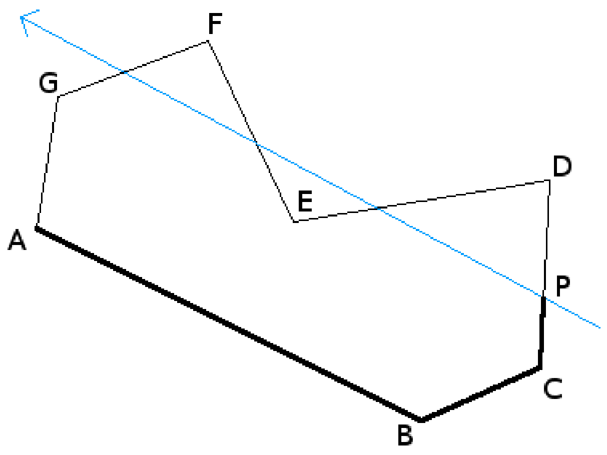
The first step of the clipping algorithm is to set a starting vertex. We will set A as our starting point. We must check in which side of the blue line it lies. We can do this using the cross product of two points of the line and our point. Since in the implementation we will be working with positively 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.



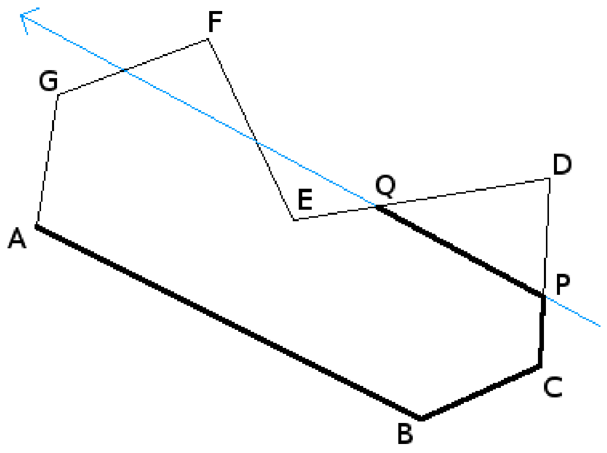
The next step is to select the next point and see whether it lies in the same side of the blue line. Since B also lies on the same side than A, we add it to the solution too and keep going.

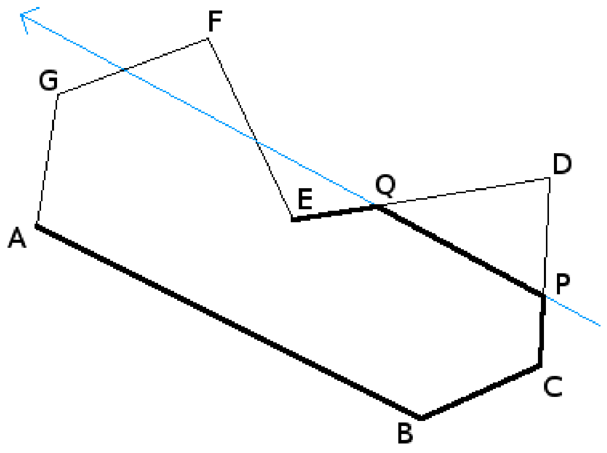


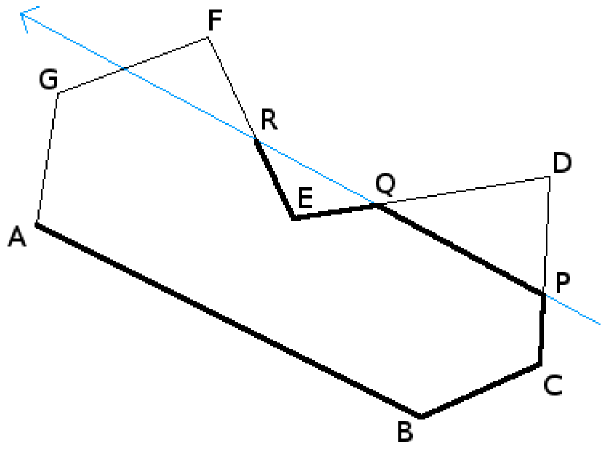
Same for C, still on the left.

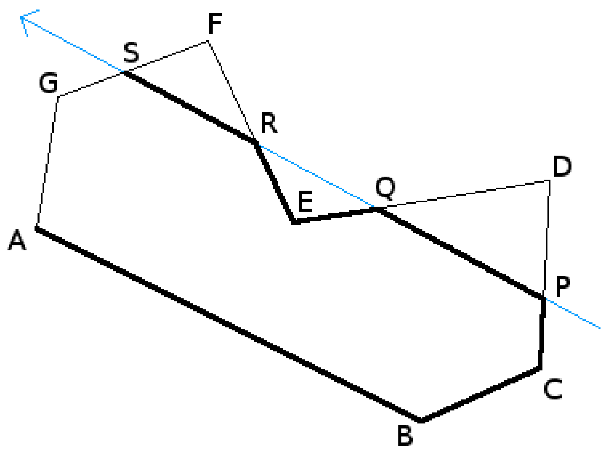


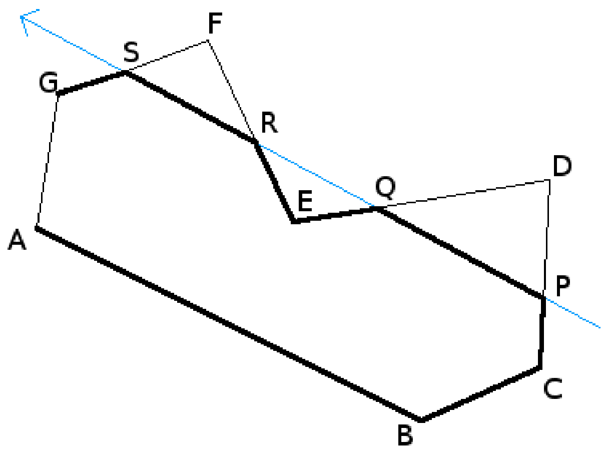
When we check D, we see that it is lying on the right side of the line. Then, we know that this point will be clipped by the line. Therefore, we need to find point P, which is the intersection of CD with the blue line. We will be adding this point to the solution.

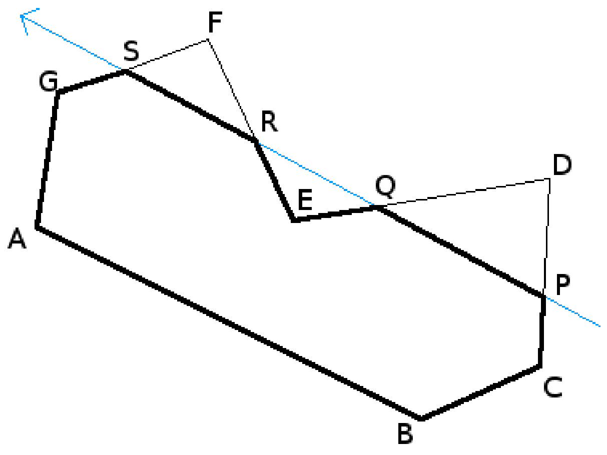

F
G
D
Q
E
A
P
C
B


F
G
D
Q
E
A
P
C
B


F
G
R
D
Q
E
A
P
C
B







Input data.
The input data for the problem is a partition of the square unit. That is, a set of

polygons whose union is the square unit 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:

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.

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 one. If they are better, take the best as the new current 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.

Improved gradient method: implementing steps
. One step closer to the simulated annealing, but still very similar to the gradient method. It consists on iterating different gradient methods, but varying the step size we are taking on each. This way, we will adjust our solution as much as possible, but this will not get us out of the local maximum.

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. In 1953 Metropolis created an algorithm to simulate the annealing process. The algorithm simulates a small random displacement of an atom that results in a change in energy. If the change in energy is negative, the energy state of the new configuration is lower and the new configuration is accepted. TODO: explain more and better.

Getting rid of the "repetition" problem: randomizing point order in each state.

Bibliografía

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