Recomendaciones de formato y contenido de la memoria https://www.fi.upm.es/?pagina=2031

ADJUSTING PLANE PARTITIONS THROUGH VORONOI DIAGRAMS

Supervisor Manuel Abellanas mabellanas@fi.upm.es
Student Guillermo Alonso Núñez guillermo.alonso.nunez@alumnos.upm.es

INDEX

Title

Title of the project: Adjusting plane partitions through Voronoi Diagrams.

Name of the student.

Name of the supervisor.

Name of the university.

Logo of the university.

Image of the software?

Index

Summary

Summary/Resumen: Problem, methods of investigation, general conclusions (no more than 2 pages)

Introduction and objectives

What is a partition. Voronoi diagrams.

Introduction to symmetric dyfference. How we will calculate it (substracting area from the square). Different methods to calculate it.

Objectives: Minimize the total difference between a given partition and the one we search in a reasonable execution time.

We would like to be able to find the generating points of perfect Voronoi diagrams (it is not guaranteed we will be able to).

Development

Why Processing. Pros and cons of Java.
First (naive) idea about the development.
Line and polygon clipping. Divide and conquer. Robustness.
Input data.
Output data.
Gradient method.
Simmulated annealing.
Getting rid of the "repetition" problem: randomizing point order in each state.

Results and conclusions

Comparing different configurations, their results as well as their execution times.
Different ways of comparing the results.
Total minimum area.
Most average area reduced per step.
Conclusions.

Appendices (optional)

Bibliography

SUMMARY

The inverse Voronoi diagram problem is about detecting whether a given plane partition is a Voronoi diagram and finding the initial points that would generate such a partition. For a partition which does not come from a Voronoi diagram, it would be an interesting question to find the best fitting Voronoi diagram. That is, finding the diagram whose total sum for each polygons symmetric difference is minimized. The aim of this project is to develop a program to adjust plane partitions through the use of Voronoi diagrams.

RESUMEN

El problema inverso del diagrama de Voronoi consiste en reconocer si una partición es un diagrama de Voronoi y calcular, en caso afirmativo, los generadores del diagrama. Para una partición que no sea un diagrama de Voronoi, cabe plantearse cuál es el diagrama de Voronoi que mejor se ajusta a la partición. Es decir, cuál es el diagrama para el que la suma de las diferencias simétricas entre sus regiones y las respectivas regiones de la partición se minimiza. En este proyecto se propone desarrollar una herramienta software para ajustar particiones planas mediante diagramas de Voronoi.

Part 1. PROBLEM DATA

1. First version

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 points with two coordinates given with a negative orientation (clock-wise). At first, we know we will be working only with convex polygons (that is, all internal angles are less or equal than 180 degrees).

2. Subtasks

In order to solve the problem, we need to solve different subtasks in which the solution will rely on. Those are:

- Calculating a set of starting seed points from which the solution will be approximated.
- Obtaining the Voronoi diagram from an arbitrary set of points.
- Adjusting the Voronoi diagram to the square unit.
- Calculating the symmetric difference between the Voronoi diagram and the given partition.
- Choosing a criterion for searching the optimal solution.
- Calculating a set of starting seed points from which the solution will be approximated.

For calculating the seed points, a simple method we can use is, knowing that all polygons are convex, simply calculate each seed point as the average of each coordinates for all points of that polygon.

- Obtaining the Voronoi diagram from an arbitrary set of points.

- For obtaining the Voronoi diagram of a given set of points, an external library is being used, Mesh - http://leebyron.com/mesh/ . That library is designed specifically for obtaining the Voronoi diagram in a non complex way for the programmer.
- Adjusting the Voronoi diagram to the square unit.

The first method I used for trimming the Voronoi diagram to the square unit was a bit naive. I tackled the problem directly instead of applying a "divide and conquer" algorithm, which resulted in an extra complexity in the logic behind it, which had an effect directly in the complexity of the code.

TODO: EXPLAIN MY FIRST METHOD

Another strategy consists of taking advantage of the line clipping algorithm. Once we have that problem solved, we simply need to clip each polygon to the four segments that make up the square.

TODO: EXPLAIN THE LINE CLIPPING ALGORITHM

- Calculating the symmetric difference between the Voronoi diagram and the given partition.

The symmetric difference of two polygons A and B can be defined as the area that belongs to either A or B, but not both. There are various ways in which the symmetric difference can be calculated.

The first method I thought of took advantage of knowing that all polygons would be convex (at least for the first version). We will be calculating the symmetric difference as the area of the union minus the area of the intersection.

$$SD(A, B) = A(A \bigcup B) - 2 \cdot A(A \bigcap B)$$

For each pair of polygons (one from the given partition and its corresponding one from the Voronoi diagram), we first need to calculate all of their intersection points in some order. Then, we need to find a point contained inside both polygons, which will be used as an auxiliary point to calculate triangle areas.

We know that between each two intersections, there are two different paths connecting them, one from the partition polygon border and another one from the Voronoi polygon border. Note that these paths could be the same for extreme cases, which need to be handled specifically.

If from the arbitrary point we calculated previosly, we now obtain the areas of each path from them, it is clear that the bigger area belongs to the union and the smaller area belongs to the intersection. Then, we simply have to iterate through each consecutive pair of intersections and keep track of the total.

Note that with this method we are not explicitly calculating the intersection nor the union.

The method (NOT) currently implemented, suggested by Manuel, calculates the symmetric difference by substracting from the square unit area, the area of all the intersection polygons.

- Choosing a criterion for searching the optimal solution.

First approximation: Gradient Method

Simulated annealing

El pseudo-código del algoritmo es el siguiente: 1. Hallar un punto de cada región (elegimos el baricentro) 2. Hallar el diagrama de Voronoi para esos puntos 2.1. Hallar la intersección de ese diagrama con el cuadrado sobre el que estamos trabajando 3. Calcular la discrepancia entre el diagrama obtenido y la partición

dato 4. Desplazamos los puntos calculados en (1) mínimamente y comparamos los resultados contra los obtenidos 4.1 Si la discrepancia es menor, quiere decir que tenemos una solución mejor [1], por lo que la guardamos y seguimos probando 4.2 Si la discrepancia es mayor, la solución actual es peor, por lo que la descartamos

Part 2. 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