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

AJUSTE DE PARTICIONES PLANAS MEDIANTE DIAGRAMAS DE VORONOI

TUTOR: Manuel Abellanas mabellanas@fi.upm.es

ALUMNO: Guillermo Alonso Núñez [guillermo.alonso.nunez@alumnos.upm.es](mailto:guillermo.alonso.nunez@alumnos.upm.es)

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.

SUMMARY

The inverse Voronoi diagram problem is about detecting whether a given partition is a Voronoi diagram and finding the initial points that would generate such a partition if it were. 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.

Datos del problema

En un principio, trabajaremos con una partición del cuadrado unidad, es decir, una serie de elementos tales que su unión es el propio cuadrado unidad y la intersección de cualesquiera dos de ellos es, a lo sumo, una recta. Más concretamente, se nos facilitará una lista de polígonos dados por las coordenadas de sus puntos. Sobre los puntos, sabemos que estarán ordenados en sentido negativo para cada polígono (esto es, en el sentido de las agujas del reloj).

Primera aproximación: Método del descenso

Para la primera versión, nos centraremos en particiones que contengan polígonos convexos y/o estrellados.

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

Cálculo de la discrepancia

Para calcular la discrepancia o error, lo que hacemos será calcular la diferencia simétrica entre todos los polígonos. La suma de todas estas diferencias, será la discrepancia entre ambas particiones.

[1] Nótese que pese a ser una solución mejor que la anterior, no significa que tomándola vayamos a llegar a la solución óptima del problema, ya que podemos quedarnos estancados en un mínimo local, no necesariamente global.

Como calcular la intersección del diagrama con el cuadrado sobre el que trabajamos.

Pese a que de manera gráfica simplemente habría que recortar por el cuadrado, computacionalmente es algo más complejo de lo que parece. En un principio, lo que implementé, simplemente miraba si un punto estaba fuera del cuadrado, y si lo estaba, lo metía dejándolo en la arista correspondiente, sobre la misma recta. El problema de ésto, era que al ser un algoritmo lineal, surgían discrepancias [Imagen de un triangulo con un vértice fuera y recortado a capón]. Lo que hice entonces fue implementarlo de manera que se hacían dos barridos, primero un preprocesado de los puntos, y después limpiabamos según hiciera falta. En el primer barrido, lo que hacemos es crear una nueva estructura en la cual metemos ordenadamente los puntos que estaban dentro del cuadrado junto con los puntos de intersección. Después, tenemos que tener cuidado, porque si resulta que una línea se va por una arista del cuadrado y entra por otra diferente, habría que añadir también el punto de la esquina a la solución. Por tanto, hacemos otra pasada sobre el primer barrido mirando esos casos, y añadiendo los puntos que necesitemos. Nótese que pese a hacer dos barridos, la complejidad de este algoritmo es O(n), siendo n el número de puntos del polígono sobre el que hallamos la intersección. Éste algoritmo hay que aplicarlo sobre un número constante de polígonos que ya sabemos de ante mano, por tanto la complejidad sigue siendo O(n).

Bibliografía

[https://processing.org](https://processing.org/) – Herramienta basada en Java sobre la cual se ha realizado todo el trabajo

<http://leebyron.com/mesh/> - Librería de Processing que nos permite realizar operaciones relacionadas con el diagrama de Voronoi