



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

Diagramas de Voronoi

Librería para el cálculo y gestión de Diagramas de Voronoi

Autora

María Oliver Balsalobre (alumna)

Director

Daniel Sánchez Fernández (tutor)



Facultad de
Ciencias



FACULTAD DE CIENCIAS
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, Junio de 2017

Índice general

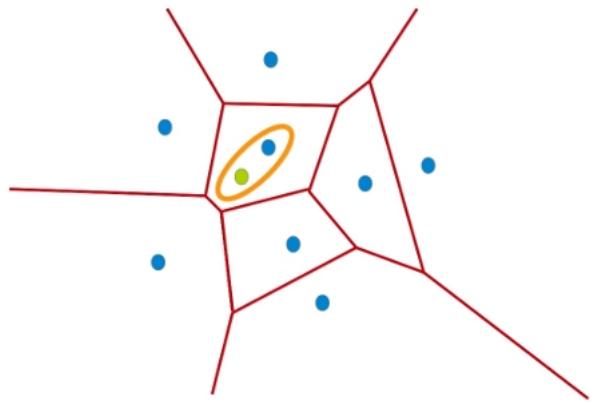
Resumen	3
Abstract	5
1. Introducción	15
2. Objetivos	19
3. Diagramas de Voronoi	21
3.1. Definiciones y resultados previos	21
3.2. Formalización y Propiedades	23
3.2.1. Caracterización de los Diagramas de Voronoi	25
3.3. Dimensiones más altas	28
3.4. Aplicaciones	29
4. Triangulación de Delaunay	33
4.1. Formalización	33
4.2. Propiedades	36
4.2.1. Propiedades de optimización	37
4.2.2. El vecino más cercano	39
5. Algoritmos para el cálculo de Diagramas de Voronoi	41
5.1. Algoritmo de Fuerza Bruta	42
5.2. Algoritmo Incremental	42
5.3. Algoritmo de Bowyer-Watson	45
5.4. Algoritmo Divide y Vencerás	48
5.5. Algoritmo de Fortune	50
5.6. Problemas de implementación	54
6. Desarrollo de la Librería	57
6.1. Análisis de requisitos	57
6.1.1. Requisitos específicos	58
6.1.2. Requisitos funcionales	58
6.1.3. Requisitos de datos	59

6.1.4. Restricciones semánticas	60
6.2. Diseño de la solución	60
6.3. Manual de usuario	65
6.4. Código desarrollado	76
7. Conclusiones y Trabajos Futuros	77
Bibliografía	79

Índice de figuras

3.1. Curiosidades de los Diagramas de Voronoi.	31
4.1. La unión de los circuncentros de la Triangulación de Delaunay produce el Diagrama de Voronoi.	34
4.2. Triangulación de Delaunay de 10 puntos en el plano, mostrando la circunferencia de cada triángulo.	36
5.1. Método Incremental del Diagrama de Voronoi.	44
5.2. Algoritmo Bowyer-Watson.	47
5.3. Algoritmo Divide y Vencerás. [11]	49
5.4. Mecanismo de barrido del Algoritmo Fortune.	51
5.5. Creación de una arista de Voronoi.	52
5.6. Desaparición de un punto del frente parabólico.	53
6.1. Información mostrada en el terminal del fichero de entrada. .	66
6.2. Información mostrada en el fichero de salida para dos dimensiones.	66
6.3. Información mostrada en el fichero de salida para diez dimensiones.	67
6.4. Pantalla inicial de la aplicación.	67
6.5. Diagrama de Voronoi de 10 puntos aleatorios.	68
6.6. Pruebas del algoritmo de Dijkstra.	69
6.7. Triangulación de Delaunay del Diagrama de Voronoi dibujado anteriormente con 10 puntos aleatorios.	69
6.8. Circunferencias circunscritas de la Triangulación de Delaunay. .	70
6.9. Hacemos zoom en la barra de desplazamiento de la parte superior de nuestra aplicación.	70
6.10. Añadimos un punto indicando sus coordenadas en la parte superior derecha de la aplicación.	71
6.11. Aplicación sin la barra de herramientas superior.	71
6.12. Guardar archivo.	72
6.13. Exportar archivo.	73
6.14. Exportar archivo.	74
6.15. Abrir archivo.	75

6.16. Mostramos información completa.	76
---	----



Diagramas de Voronoi

Librería para el cálculo y gestión de Diagramas de Voronoi.

Autora

María Oliver Balsalobre (alumno)

Director

Daniel Sánchez Fernández (tutor)

Librería para el cálculo y gestión de Diagramas de Voronoi

María Oliver Balsalobre (alumna)

Palabras clave: Diagrama de Voronoi, Triangulación de Delaunay, Algoritmo, Puntos, Vértices

Resumen

En este Trabajo Fin de Grado se aborda el problema de los Diagramas de Voronoi y su dual, es decir, la triangulación de Delaunay. Dado que nuestros objetivos principales eran elaborar una librería para el cálculo y gestión de estos diagramas en espacios euclídeos y su previo estudio en diversas tipologías, propiedades y algoritmos de obtención, comenzamos el trabajo con una serie de resultados previos sobre los que hablaremos durante el transcurso del proyecto.

Tras definir estos conceptos de vital importancia, en el tercer capítulo daremos una normalización del Diagrama de Voronoi, una de las estructuras fundamentales dentro de la Geometría Computacional, que se define como la descomposición de un espacio métrico en regiones de forma que a cada objeto se le asigna una región del espacio métrico formada por los puntos que están más cerca de él que de ningún otro objeto. Comentaremos algunas de sus propiedades y las numerosas aplicaciones que utilizan estos diagramas además de situaciones de nuestro día a día en las que podemos encontrarnos esta estructura. Además, en la elaboración práctica y el desarrollo informático de este trabajo llevaremos a cabo una de estas aplicaciones: robótica; los robots intentan llegar a un lugar indicado evitando obstáculos.

También hablaremos de la Triangulación de Delaunay, el dual del Diagrama de Voronoi, ya que los circuncentros que quedan tras hacer esta triangulación, son los vértices de las aristas que forman dicho Diagrama de Voronoi. Mencionaremos también algunas de sus propiedades básicas ya que esta triangulación posee varias características acerca de sus propiedades de optimización, sus estructuras como subgrafo y su flexibilidad para adaptarse a todo tipo de dimensiones. Comentaremos el algoritmo del vecino más

cercano, un problema de distancia que se resuelve directamente con el Diagrama de Voronoi y veremos que podemos calcular un par de puntos cuya distancia es mínima si el vecino más cercano para cada punto dentro de nuestro conjunto es conocido.

A continuación analizaremos algunos de los algoritmos estudiados para el cálculo tanto de los Diagramas de Voronoi como para la triangulación de Delaunay y veremos que este capítulo se encuentra a caballo entre los dos temas principales de nuestro trabajo: la geometría y la computación. Estos algoritmos nos serán muy útiles en el desarrollo y realización de la librería, para su cálculo y gestión y tendremos en cuenta en todo este apartado el concepto de orden de complejidad definido en el capítulo 3, para entender a qué nos referimos al hablar de tiempo empleado en los distintos algoritmos.

En la siguiente sección haremos, en primer lugar, un análisis de los requisitos donde describiremos detalladamente el comportamiento de nuestro sistema, los datos que recibe, trata, almacena y devuelve y cualquier consideración sobre su funcionamiento. Detallaremos cómo hemos resuelto el problema planteado, elaborando una librería mediante el lenguaje de programación Java cuya utilización por un lado, nos va a permitir, al pasársela un fichero de texto con la entrada correspondiente, obtener la salida deseada que será la dimensión, el número de puntos y las coordenadas de dichos puntos, que serán los vértices de Voronoi y por último, el número de regiones (polígonos) de Voronoi. Por otro lado, en otro proyecto realizaremos el diseño gráfico del Diagrama de Voronoi, incluyendo la librería realizada con una serie de clases que serán la base de la implementación. Para finalizar este capítulo elaboraremos un manual de usuario en el que se explicará paso a paso y de forma detallada y amigable para cualquier usuario, el funcionamiento de las dos aplicaciones desarrolladas.

Por último, en la finalización de este trabajo, haremos una autoevaluación en la que veremos si se han cumplido los objetivos planteados al inicio del proyecto y comentaremos la aportación personal que ha supuesto su realización, así como la mención de algunas vías futuras interesantes que son consecuencia de la investigación realizada.

Voronoi Diagrams: Library for the calculation and management of Voronoi Diagrams.

María Oliver Balsalobre (student)

Keywords: Voronoi Diagrams, Delaunay Triangulation, Algorithm, Points, Vertex

Abstract

This project verses about the known problem of Voronoi Diagrams along with its dual counterpart, Delaunay triangulation. One of the main objectives has been to create a library for the calculation and handling of these diagrams in Euclidean spaces, including preliminary analysis on different typologies, properties and building algorithms. A series of previous results has been used as a starting point to develop the whole topic discussion along this project.

As a starting point, key concepts are described and detailed, providing a normalization for the Voronoi Diagram which is one of the fundamental structures in Computational Geometry. Its origins in the Western literature date back to at least 17th century. René Descartes claims that the solar system consists of vortices. His illustrations show a descomposition of space into convex regions, each consisting of mater revolving around one of the fixed stars. Even thoug Descartes has not explicitly defined the extension of these regions, he gave us the underlying idea. Formally, Voronoi Diagrams are defined as the decomposition of a metric space in regions, in a way that every object is allocated a region of the metric space, composed by the closest points to that specific object. A Voronoi region is then defined as the group of closest points to any given point, and a Voronoi edge will be the boundary between two Voronoi regions. One of the main properties discussed implies that a convex partition of the plane defines a Voronoi diagram if and only if there is only one unique point for each region.

Indeed, this fundamental concept has emerged independently, and proven useful, in various fields of science. It is important to highlight how present

Voronoi Diagrams are in many different fields and disciplines such as architecture, design, urban building or meteorology. Indeed the practical side of this project uses one specific example of Voronoi Diagrams application: robotics, where robots follow certain pre-established path, avoiding obstacles along the way.

When talking about Delaunay Triangulation, the link with its dual counterpart, Voronoi Diagram, is drew, as building one of them immediately enables the construction of the other: the circumcentres obtained by the triangulation would be the vertexes for the edges of the Voronoi Diagram. Formally, a network of triangles becomes a Delaunay Triangulation if all the circumscribed circumferences being part of the triangles of the network are empty, meaning that no other vertexes are included, apart from the three vertexes defining them. As part of the dissertation, some basic properties of this triangulation will be presented, related to optimization capabilities, subgraph structures and flexibility to adapt itself to any number of dimensions. The nearest neighbour algorithm will be tackled, a known distance problem that can be directly resolved using the Voronoi diagrams. One important aspect is how to calculate a pair of points separated by the minimum distance, giving that the nearest neighbour for each point of the group is known. And this will be done in a lineal space and time.

Geometry and its link to computation complexity is an important part of this exercise, focusing on the variety of algorithms that can be used to calculate both the Voronoi Diagrams and Delaunay Triangulations. The brute force algorithm is explained, as a first approach, being a way to explore the geometry of each Voronoi region. Then the incremental algorithm is discussed, generally based in building the k points diagram and then move to the $k+1$ points version. Additionally, Bowyer-Watson algorithm is described, detailing how it can be used to calculate the Delaunay triangulation of a finite set of points for any number of dimensions. One further step will deal with the divide and conquer algorithm, explaining its associated high complexity while being the optimum method for the deterministic worst case. And finally the Fortune algorithm, based on a plain sweep technique, which on one hand can calculate the Voronoi Diagram for a cloud of points in the plane, but on the other hand it is by far the most complex approach. These algorithms will be a key part of the library being built, and the complexity measurement will be a very important input for its design (including the constant / logarithmic / lineal order), when talking about time taken by each of the algorithms to solve the given problem. A full set of functions will be created, sharing the same asymptotic behaviour that will lead to understanding which algorithms are the most efficient ones among the selection.

Once the algorithm screening is finished, the focus goes to the system that has been built, analysing the initial requirements and the expected behaviour, including inputs expected, how they are handled and stored, and outcomes. The solution to the initially defined problem is detailed, using

the library programmed in Java, that potentially could be used by anyone to resolve this specific problem or adapting it for other use cases. Some defined classes are key to understand how the library has been built: “Punto”, where the points defining the Euclidean space are grouped, as well as basic geometric calculations and the representation of matrixes and convex hulls; “Conjunto”, which is an abstract class built to create the skeleton of the interface, essentially being the body of the implementation; “Triangulo”, which inherits from the previous class, enabling the calculation of the neighbour triangles, the circumcentres and the opposite side to an any given vertex; “Grafo”, which will sustain the graph development, adding or subtracting both nodes and edges in the undirected graph, and will provide the set of nodes for the specific graph; “Triangulacion2D” will calculate the 2D Delaunay triangulation itself using the incremental algorithm, as, though not being the optimum or fastest approach, facilitates an interactive way of understanding and seeing how the algorithm works; and finally “Dijkstra”, specific class implementing the Dijkstra algorithm, calculating the shortest way from one point of the graph to any other point, taking into account the weight of each edge.

The system will work as follows: as an input we will use a text file that will contain the expected dimension, the number of points and their coordinates; and the outcome will be the Voronoi vertexes coordinates and the number of Voronoi regions (polygons). Additionally, the graphic representation of the Voronoi diagram in 2D has been implemented, including a library built over JFrame and using JPanel (Java tools). Within the JPanel, all the different methods available to build the Voronoi Diagrams and its associated Delaunay Triangulations have been implemented, along with the circumscribed circumferences. Different basic and specific drawing approaches are available, apart from a list of methods to calculate the shortest path between two random points. Using JFrame, the whole interactive framework is provided, enabling the user to work with the tool in real time.

As a complement, a user manual has been written, including a step by step guideline for both developed applications, that is, the graphic representation of the Voronoi Diagram and tools to support the input and output of terminal data. Screenshots for each part have been included as a way to facility the usage of the different options available. It is important to highlight that there are two different modes of operation: Voronoi, being the default option, to calculate the Voronoi Diagram for the set of points provided as an input; and Dijkstra, that will provide the option of choosing two random points, calculating the shortest path between them through the Voronoi vertexes and edges. Additionally, extra information can be displayed to describe the Delaunay Triangulation or the circumscribed circumferences to each triangle, associated to the created Voronoi Diagram. Basic interaction with the files has been included for simplicity, so open, save and export functions have been added to the interface. Furthermore, other options ha-

ve been developed to facilitate user interaction such as erasing the whole palette to start from scratch, seeing or hiding the tool bar, zooming in the diagram, or provide new points by entering its coordinates instead of using the click button. As already mentioned, both applications are available for anyone to use them or build up on them to expand the use cases or problems that could provide a solution for.

As a set of next steps, based on the outcome of this study, different options to continue the investigation are described. The body of literature on Voronoi Diagrams is vast and continuously growing, and many of the arising both theoretical and practical problems have found elegant and efficient solutions. Naturally, new questions have emerged from the exhaustive research conducted, and quite a few problems eluded satisfactory settlement until today. Especially those problems related to both the combinatorial complexity of simple structures and their efficient algorithm construction are still to be further investigated and resolved when dealing with Voronoi Diagrams. Therefore, as an interesting example, future work could be focused on Voronoi Diagrams of circles, and its application to the visualization of the growth of particles, studying Peter Gärdenfors' theory of conceptual representations, as a bridge between the symbolic and connectionist approaches, just to mention one of the multiple interesting ways forward.

Yo, **María Oliver Balsalobre**, alumna de la titulación Doble Grado en Ingeniería Informática y Matemáticas de la **Facultad de Ciencias** y la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 75719191-V, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.



Fdo: María Oliver Balsalobre

Granada a 26 de Junio de 2017 .

Daniel Sánchez Fernández, Profesor del Área de Bases de Datos Inteligentes y Sistemas de Información de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Librería para el cálculo y gestión de Diagramas de Voronoi***, ha sido realizado bajo su supervisión por **María Oliver Balsalobre (alumna)**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 26 de Junio de 2017.

El tutor:



Daniel Sánchez Fernández

Agradecimientos

Gracias a mi tutor Daniel, por ayudarme en todo lo posible desde el primer momento y haber estado disponible cada vez que lo he necesitado para contribuir con sus conocimientos a la realización de este trabajo.

Gracias a mi familia, en especial a mis padres, mis hermanos y mis abuelos, por el apoyo incondicional y constante. Gracias por haberme enseñado, tanto en lo personal como en lo profesional, y por el amor que recibo a diario.

Gracias a mis compañeros por recorrer juntos este camino hacia el final del Doble Grado y por pasar conmigo momentos inolvidables. Solís, eres la persona con las mejores ideas para resolver problemas que he conocido durante estos años y fundamental para mí. Marc estoy orgullosa de ti por todo lo que has conseguido y sigues consiguiendo. Me encanta verte crecer en lo que te gusta. Torres sabes que soy tu fan número uno, eres una de las mejores personas que conozco. Ramón, espero que de todos estos años te lleves un buen recuerdo de lo que hemos compartido durante la carrera tal y como lo haré yo. Miki, me alegra mucho de haberte conocido y haber compartido tantos momentos juntos, tanto profesionales como personales durante la carrera.

Gracias a mis amigos, los de siempre (CarmenAG, Mmar, Tina, Eli, Ajo, Paquito, Dani), por aguantar mis cambios de humor, estar siempre dispuestos a escucharme y ser un pilar imprescindible en mi vida. Dicen que los amigos son la familia que se elige y sin duda, yo no podría haber elegido mejor.

Y a mi mejor amigo, compañero de vida y de experiencias, y mi primer y último pensamiento en todo lo que hago, gracias por todo, espero pasar toda una vida juntos y seguir viéndote crecer y consiguiendo todas tus metas y objetivos.

Sin vosotros terminar el Doble Grado en Ingeniería Informática y Matemáticas no habría sido posible.

Capítulo 1

Introducción

El trabajo está dedicado a una estructura geométrica muy importante e influyente llamada **Diagrama de Voronoi**, cuyos orígenes, en la literatura occidental, nos llevan, al menos, al siglo XVII. Un Diagrama de Voronoi es una descomposición de un espacio métrico en regiones, de tal forma que, en dicha descomposición, a cada objeto se le asigna una región del espacio métrico formada por los puntos que están más cerca de él que de ninguno de los otros objetos. Es decir, divide el espacio en tantas regiones como puntos u objetos tengamos de tal forma que a cada punto se le asigna la región formada por todo lo que está más cerca de él que de ningún otro. En este proyecto abordaremos el estudio y evolución de los Diagramas de Voronoi, una de las estructuras fundamentales dentro de la Geometría Computacional.

Varios nombres han sido dados a estos diagramas, dependiendo del dominio (geografía, meteorología, biología, psicología). El matemático ruso Gueorgui Voronoi, creador de estos diagramas, no fue el primero en estudiarlos de una manera más formal y sistemática, siendo el meteorólogo americano Alfred H. Thiessen el que desarrolló esta construcción geométrica que permitía construir particiones del plano euclídeo; los llamó **polígonos de Thiessen**. Estos polígonos también fueron estudiados por el matemático alemán Gustav Lejeune Dirichlet, que les dio el nombre de **teselación de Dirichlet**.

Esta estructura es tan lógica e intuitiva que aparece cada vez que nos planteamos cuestiones relacionadas con problemas de proximidad y por eso muchos matemáticos se han dedicado a su estudio. Muchos autores señalan que esta estructura fue utilizada, incluso antes de Thiessen, en trabajos realizados por Descartes en 1644 sobre el cielo, cuando afirmaba que el Universo estaba formado por vórtices y plagado de éter y materia. René Descartes, en su trabajo sobre los principios de la filosofía, afirma que el sistema solar está constituido por vórtices y en sus ilustraciones muestra una descomposición del espacio en regiones convexas, cada una con una materia giratoria alrededor de una estrella fija. Aunque no especificó explícitamente la extensión

de estas regiones, la idea parece ser la siguiente: tomemos un espacio y un conjunto de lugares o sitios dados en ese espacio junto con una noción de la influencia que un lugar ejerce sobre cada localización del espacio. Por tanto, la región de cada lugar abarca todos los puntos en los que la influencia es más fuerte. Fue John Snow, para muchos padre de la epidemiología moderna, el que los utilizó cuando se produjo el brote de cólera en Londres en 1854. John dedujo que la causa de la enfermedad era el consumo de aguas contaminadas por heces. Para ello, en un mapa, señaló la distribución de muertes por cólera y estudió dónde se encontraban las fuentes de agua potable de la ciudad y delimitó las “regiones de Voronoi” de cada una de esas bombas. Calculando la distancia entre la residencia de cada difunto y la bomba de agua más cercana, llegó a la conclusión de que la zona más afectada por la enfermedad se correspondía con la región de Voronoi asociada a la bomba de Broad Street, ya que en dicha región se dieron 73 de los 83 casos. Al retirar la fuente de agua de dicha zona, el brote de cólera se extinguió.

Tras la solución de este problema, el Diagrama de Voronoi ha tenido cada vez más uso y se ha ido desarrollando tanto en el plano como en el espacio (en dos y tres dimensiones). Es importante, tanto en la teoría como en la práctica, y es por eso que su éxito está presente en campos muy diversos como, por ejemplo, en Robótica, donde se puede usar para marcar los caminos que puede seguir un robot ajustando los puntos del diagrama a los obstáculos que podría encontrarse durante su trayectoria. Pero ésta no es la única aplicación en la que nos podemos encontrar presentes estos diagramas, ya que en casi cualquier ámbito en el que pensemos seguramente se estén usando de manera más o menos visible estas estructuras, teniendo cada una su propia noción de espacio, lugar e influencia: en el caso de la organización territorial a nivel de España para la división de comunidades autónomas; a nivel deportivo, estudiando las zonas más próximas a cada jugador para planear las jugadas; en la naturaleza podemos encontrarlos en las pieles de los animales y las hojas de los árboles o en zonas desérticas, donde las grietas del terreno se ajustan según los puntos de máxima humedad; en Arquitectura, cada vez están más demandadas a la hora de decorar espacios debido a su geometría, etc.

Además de sus aplicaciones directas en todos los campos de la ciencia que hemos mencionado, los Diagramas de Voronoi pueden usarse para la resolución de numerosos problemas geométricos y teóricos. Por su estrecha relación con politopos (generalización a cualquier dimensión de un polígono bidimensional, o un poliedro tridimensional) e incluso con hiperplanos en dimensiones superiores, muchas preguntas (y respuestas) de conversión y geometría discreta se trasladan a estos diagramas. Además, la **Triangulación de Delaunay**, vista como un grafo combinatorio, está relacionada con varios grafos conectados.

Por todo esto, parece razonable mantener la idea, durante todo lo que nos sigue, de que cada uno de nosotros quizás nos encontremos en una región

que llamaremos nuestra, pero Voronoi estará presente en la de todos. Esta es la principal intención de este trabajo, guiarnos al fascinante mundo de los Diagramas de Voronoi.

Capítulo 2

Objetivos

Los Diagramas de Voronoi están estrechamente relacionados con la Geometría Computacional por lo que su temática es apropiada para un trabajo final de este Doble Grado. Abarca tanto aspectos matemáticos de geometría ya que, de alguna forma, ellos almacenan toda la información referente a la proximidad entre puntos, como aspectos informáticos en cuanto a la implementación de algoritmos que se basan en su utilización.

Aunque es casi imposible presentar todos los resultados conocidos en los Diagramas de Voronoi y todo lo relacionado con ellos, nuestro objetivo es estudiar los fundamentos principales relacionados con los puntos de vista estructurales y algorítmicos. Además de ser una estructura de división de espacio versátil, los Diagramas de Voronoi también son estéticamente agradables y muchas personas se sienten atraídas por ellos, incluso con respecto a sus aspectos artísticos.

Para la consecución de este objetivo, es necesario profundizar en la comprensión de los conceptos teóricos asociados, algunos de los cuales están actualmente en desarrollo, y así adquirir los conocimientos necesarios que permitan desarrollar la librería.

A continuación mostramos cuáles son los objetivos generales de ambas partes, detallando los más específicos.

Desde el punto de vista matemático, hemos definido una serie de objetivos a alcanzar a lo largo del trabajo. Éstos se centran en el estudio de los Diagramas de Voronoi de diversas tipologías en espacios euclídeos, propiedades y algoritmos de obtención.

- ★ Comprender el concepto de Diagrama de Voronoi y algunas de sus propiedades más conocidas.
- ★ Conocer la Triangulación de Delaunay junto con el algoritmo del vecino más cercano.

- ★ Estudiar los algoritmos empleados para la elaboración de los Diagramas de Voronoi.

Desde el punto de vista de la ingeniería informática marcamos como objetivo general la realización de una librería para el cálculo y gestión de los Diagramas de Voronoi en espacios euclídeos, incluyendo la entrada de puntos, la realización de la partición y otras funciones relevantes en diversas aplicaciones prácticas.

- ★ Especificar los requisitos funcionales y no funcionales.
- ★ Elaborar una librería donde incluyamos lo necesario para implementar los Diagramas de Voronoi trabajando con el lenguaje de programación Java.
- ★ Ser capaz de realizar una interfaz gráfica, con el uso de Java y estudiando las principales características de la programación de gráficos usando la tecnología Java2D, para hacer una aplicación en la que podemos ver, de manera visual, tanto lo explicado teóricamente como uno de los múltiples usos de estos diagramas.

Entender las propiedades de estos diagramas es la clave para su aplicación efectiva y para el desarrollo de la rápida construcción de algoritmos. Este problema es uno de los métodos de interpolación más simples que se basa en la distancia euclídea matemática.

Para analizar y explicar con claridad este concepto, utilizaremos una base matemática relacionada directamente con la geometría y, sobre todo, con la *distancia euclídea* y la *Triangulación de Delaunay*. Por otro lado veremos varias formas de calcular una representación geométrica tanto de los Diagramas de Voronoi como de su dual (Triangulación de Delaunay) haciendo uso de algoritmos básicos de la informática.

Para entender estas ideas hemos hecho uso de [1], ya que plasma toda la temática de una forma comprensible y bien estructurada. Además, nos ha facilitado la aproximación a nuevos conceptos que de otra forma hubiesen resultado más difíciles de interpretar.

Decidimos que la mejor forma de plasmar todo lo expuesto era realizar la implementación de una librería que nos permitiera visualizar lo referente al trabajo y entender de manera más dinámica todo lo relacionado con el cálculo y la gestión de estos diagramas. Para hacerlo, hemos usado el lenguaje de programación *Java*.

Capítulo 3

Diagramas de Voronoi

En este capítulo veremos, en primera instancia, algunas definiciones y conceptos previos relacionados con la geometría computacional y con la informática y enunciaremos varios resultados que nos serán muy útiles para comprender cómo funcionan estos diagramas. Además, comentaremos algunas propiedades interesantes para los Diagramas de Voronoi y los generalizaremos a dimensiones más altas, donde veremos que hay ciertas propiedades que se conservan. Por último, hablaremos de algunas de las numerosas aplicaciones que tienen debido a su gran importancia. Nos daremos cuenta, por tanto, de qué manera se manifiestan en la naturaleza y cómo y dónde podemos encontrarlos en nuestro día a día.

3.1. Definiciones y resultados previos

Daremos, en primer lugar, algunas notaciones y definiciones básicas que nos serán muy útiles para comprender mejor todo lo referente a los Diagramas de Voronoi. Notaremos por \mathcal{S} al conjunto de puntos en el **plano Euclídeo** \mathcal{R}^2 .

Definición 3.1. *Para los puntos $p = (p_1, p_2)$ y $x = (x_1, x_2)$, la **distancia euclídea** viene dada como*

$$d(p, x) = \sqrt{(p_1 - x_1)^2 + (p_2 - x_2)^2}.$$

Definición 3.2. *La **mediatriz** es el segmento que notaremos como \bar{pq} o simplemente como pq y se define como la recta perpendicular al segmento formado por p y q que se traza en su punto medio y lo divide en dos partes iguales.*

Definición 3.3. El **circuncentro** para S es el punto donde se cortan las tres mediatrices de un triángulo y es el centro de la circunferencia que pasa por los tres vértices, llamada **circunferencia circunscrita**.

Definición 3.4. Para cada $p, q \in S$ llamaremos **bisector**, $B(p, q)$, al lugar de todos los puntos en \mathbb{R}^2 que equidistan de p y de q . Es la línea perpendicular que pasa por el punto medio del segmento de línea \bar{pq} y separa el semiplano

$$D(p, q) = \{x | d(p, x) \leq d(q, x)\}$$

más cercano a p del semiplano $D(q, p)$ más cercano a q .

Definición 3.5. Una **partición** de un conjunto es una división del mismo en trozos separados y no vacíos. Esta división se representa mediante una colección o familia de subconjuntos de dicho conjunto que lo recubren.

Definición 3.6. La **envolvente convexa** de un conjunto de puntos es el polígono convexo que contiene todos los elementos de los puntos con menor área (o perímetro) posible.

Definición 3.7. La **geometría computacional** es la unión de la geometría clásica con la informática. Es decir, es una rama de las ciencias de la computación dedicada al estudio de algoritmos que pueden ser expresados en términos de la geometría. Partiendo de la abstracción de un problema de otras áreas, trata de desarrollar herramientas y técnicas para resolver problemas de naturaleza principalmente geométrica, con especial atención en el diseño eficiente de algoritmos y estructuras de datos.

Definición 3.8. A un conjunto de funciones que comparten un mismo comportamiento asintótico le denominaremos un **orden de complejidad**. Habitualmente estos conjuntos se denominan O , existiendo una infinidad de ellos. Para cada uno de estos conjuntos se suele identificar un miembro $f(n)$ que se utiliza como representante de la clase, hablándose del conjunto de funciones g que son del orden de $f(n)$. Básicamente, $O(f(n))$ está formado por aquellas funciones $g(n)$ que crecen a un ritmo menor o igual que el de $f(n)$. Así, tendremos: $O(1)$ orden constante; $O(\log(n))$ orden logarítmico; $O(n)$ orden lineal, etc.

3.2. Formalización y Propiedades

En esta sección, empezaremos considerando el caso más simple: puntos en el plano teniendo en cuenta la distancia euclídea entre ellos. Las propiedades que discutiremos son de vital importancia ya que la mayoría nos permitirán continuar con otros tipos de Diagramas de Voronoi y sus relativos, que veremos más adelante. Antes de nada, daremos una definición más general de Diagrama de Voronoi e introduciremos un par de conceptos importantes (como son, región de Voronoi y arista de Voronoi) que utilizaremos y hablaremos de ellos a lo largo de todo el proyecto.

Sea $S := s_1, \dots, s_n$ un conjunto de n puntos distintos en el plano denominados **sitios**, definimos el **Diagrama de Voronoi** de S como la subdivisión del plano en n celdas y lo denotaremos como $V(S)$.

Además, un punto p pertenece a la región de un sitio s_i si y sólo si

$$d(p, s_i) < d(p, s_j), \forall s_i \in S \text{ con } j \neq i.$$

Definición 3.9. La **región de Voronoi**, $VR(p, S)$, de p entre el conjunto dado, S , de sitios es la intersección de los $n-1$ semiplanos $D(p, q)$, donde q se extiende sobre todos los otros sitios en S . De manera más formal, podemos escribir

$$VR(p, S) = \bigcap_{q \in S, q \neq p} D(p, q).$$

$VR(p, S)$ consiste en todos los puntos $x \in \mathcal{R}^2$ para los que p es el vecino más cercano.

Está compuesta por la intersección de $n-1$ semiplanos que conforman una región poligonal convexa que puede ser abierta o cerrada. Esta región puede tener como máximo $n-1$ vértices y $n-1$ aristas.

Definición 3.10. La parte fronteriza común entre dos regiones de Voronoi se llama **arista de Voronoi** si contiene más de un punto. El Diagrama de Voronoi de S , $V(S)$, está definido como la unión de todas las aristas de Voronoi. Además, los puntos finales de las aristas serán los **vértices de Voronoi** que pertenecen al límite común de tres o más regiones de Voronoi.

Tras estos conceptos de vital importancia, veamos que existe una manera intuitiva de ver el Diagrama de Voronoi $V(S)$. Sea x un punto arbitrario en el plano, centramos un círculo, C , en x y dejamos que su radio crezca, desde 0 en adelante. En algún momento la expansión del círculo, por primera vez, alcanzará uno o más sitios de S . Tenemos entonces tres casos diferentes que veremos a continuación.

El siguiente Lema muestra que la región de Voronoi forma una descomposición del plano. Las regiones no se superponen, y para cada punto x hay al menos un sitio más cercano en \mathcal{S} , lo cual quiere decir que x está cubierto por esta región.

Lema 3.1. (i) *Si el círculo C , en su expansión desde el punto x , alcanza exactamente un sitio, p , entonces x pertenece al interior de la región $VR(p, \mathcal{S})$.*

(ii) *Si C alcanza exactamente dos sitios, p y q , entonces x es un punto interior de la arista de Voronoi que separa las regiones de p y q .*

(iii) *Si C alcanza tres o más sitios simultáneamente, entonces x es un vértice de Voronoi adyacente a aquellas regiones cuyos sitios hayan sido alcanzados.*

Demostración. (i) Si únicamente el sitio p es alcanzado, entonces p es el único elemento de \mathcal{S} que está más cerca de x . Por tanto, $x \in D(p, r)$ se mantiene para cada sitio $r \in \mathcal{S}$ con $r \neq p$.

(ii) Si C alcanza exactamente p y q , entonces x está contenido en cualquier semiplano $D(p, r), D(q, r)$, donde $r \notin p, q$, y también en $B(p, q)$ (límite común de $D(p, q)$ y $D(q, p)$). Por la definición 3.10 x pertenece al final de las regiones de p y q , pero de ningún otro sitio de \mathcal{S} .

(iii) En este caso, el argumento es análogo al anterior. □

Desde el Diagrama de Voronoi de \mathcal{S} podemos derivar fácilmente en la **envolvente convexa** de \mathcal{S} , que ya hemos visto en la definición 3.6 que se define como el conjunto convexo más pequeño contenido en \mathcal{S} . Como \mathcal{S} es finito, su envolvente convexa es un polígono convexo, abarcado por los $h \leq n$ puntos extremos de \mathcal{S} .

Lema 3.2. *Un punto p de \mathcal{S} se encuentra en el límite de la envolvente convexa de \mathcal{S} si y sólo si su región de Voronoi, $VR(p, \mathcal{S})$, es ilimitada.*

Demostración. La región de Voronoi de p es ilimitada si y sólo si existen algunos puntos $q \in \mathcal{S}$ tales que $V(S)$ contiene una pieza ilimitada de $B(p, q)$ como una arista de Voronoi. Sean $x \in B(p, q)$ y $C(x)$ el círculo de p y q centrado en x . El punto x pertenece a $V(S)$ si y sólo si $C(x)$ no contiene otro sitio. A medida que movemos x a la derecha a lo largo de $B(p, q)$, la parte de $C(x)$ contenida en el semiplano R sigue creciendo. Si hay otro sitio r en R , será alcanzado eventualmente por $C(x)$, haciendo que la arista de Voronoi acabe en x . De lo contrario, todos los demás sitios de \mathcal{S} deberán estar contenidos en el cierre del semiplano izquierdo L . Entonces p y q se encontrarán en la envolvente convexa de \mathcal{S} . □

Además, otra virtud del Diagrama de Voronoi es su pequeño tamaño.

Lema 3.3. *El número medio de aristas en el límite de una región de Voronoi es menor que 6.*

Demostración. Esto puede demostrarse de manera directa usando la fórmula de Euler para poliedros.

$$\text{vértices} - \text{aristas} + \text{caras} = 1 + \text{componentes conectadas}.$$

Aplicamos esta fórmula al diagrama finito de Voronoi: cada vértice tiene al menos tres aristas conectadas por lo que, sumando, obtenemos $\text{aristas} \leq \text{vértices}/2$ ya que cada arista se cuenta dos veces. Sustituyendo esta desigualdad con $\text{componentes conectadas} = 1$ y $\text{caras} = n + 1$ obtenemos:

$$\text{vértices} \leq 2n - 2 \text{ y } \text{aristas} \leq 3n - 3$$

La suma del número de aristas contenidas en los límites de todas las caras $n + 1$, da como resultado $2 * \text{aristas} \leq 6n - 6$, porque cada arista, de nuevo, se cuenta dos veces. Por lo tanto, el número medio de aristas que limitan una región es, como máximo $(6n - 6)/(n + 1) < 6$. \square

Es decir, de esta fórmula de Euler, se desprende que el número de vértices de Voronoi es como máximo $2 * \text{caras} - 5$ y el número de aristas es a lo más $3 * \text{caras} - 6$. Por lo tanto, la complejidad de $V(S)$ es $O(n)$.

3.2.1. Caracterización de los Diagramas de Voronoi

Hasta ahora las pocas propiedades mencionadas son básicas para el Diagrama de Voronoi de los puntos en el plano. Veamos ahora algunas características avanzadas del Diagrama de Voronoi clásico.

El proceso de construcción del Diagrama de Voronoi puede ser visto como una asignación de una región convexa plana a cada uno de los sitios, de acuerdo con la regla del vecino más cercano que veremos más adelante en la sección 4.2.2. Abordamos ahora la pregunta siguiente: dada una partición del plano en n regiones convexas (que son entonces necesariamente poligonales), ¿existen sitios, uno para cada región, de tal manera que la regla del vecino más cercano se cumple? En otras palabras, dado el Diagrama de Voronoi de un conjunto de sitios, ¿cuándo podemos decir que es una partición convexa?

Un determinado conjunto de sitios induce una partición convexa dado que su Diagrama de Voronoi es, por supuesto, fácil de decidir explotando propiedades de simetría entre los sitios. A continuación nos vamos a centrar en el reconocimiento de los Diagramas de Voronoi sin conocer los sitios.

Las cuestiones de este tipo se plantean en la localización de las instalaciones y en el reconocimiento de los modelos de crecimiento biológico: cuando los sitios se consideran como lugares de votación y la ley electoral requiere que cada persona vote en la encuesta respectiva más cercana al lugar, los distritos electorales forman un Diagrama de Voronoi. Si la legislatura dibuja las líneas del distrito en primer lugar, ¿cómo podemos saber si la ley electoral es satisfactoria?

Sean R_i y R_j dos de las regiones dadas. Supongamos que comparten una arista común, y sea h_{ij} la recta que contiene esa arista. Además, sea σ_{ij} la reflexión en la línea h_{ij} .

Lema 3.4. *Una partición convexa R_1, \dots, R_n del plano define un Diagrama de Voronoi si y sólo si existe un punto p_i para cada región R_i tal que:*

- (1) $p_i \in R_i$ (condición de contención).
- (2) $\sigma_{ij}(p_i) = p_j$ si R_j es adyacente a R_i (condición de reflexión).

*Demuestra*ción. Si tenemos un Diagrama de Voronoi entonces sus sitios de definición existen y obviamente cumplen (1) y (2). Para demostrar lo contrario, supongamos que los puntos p_1, \dots, p_n satisfacen ambas condiciones. Tomando cualquier región R_i y cualquier punto x en ella. Demostramos que $d(x, p_i)$ es mínimo. Para obtener una contradicción, supongamos que p_j , $j \neq i$, es la más cercana a x . Consideremos una arista de R_j que está intersectada por el segmento de línea $x\bar{p}_j$, y sea R_k la región adyacente a R_j en esa arista. Obsérvese que $k = i$ puede ocurrir. Por la convexidad de R_i y por (1), la recta h_{jk} separa p_j de x . Por lo tanto, por (2), obtenemos $d(x, p_k) < d(x, p_j)$, que es una contradicción. Podemos concluir entonces que p_i es la más cercana a x entre p_1, \dots, p_n lo que implica que R_i es la región de p_i en el Diagrama de Voronoi $V(p_1, \dots, p_n)$. \square

Basado en este Lema 3.4, el problema de reconocimiento ahora puede ser formulado como un problema de programación lineal. Primero explotamos la condición de reflexión para obtener un sistema de ecuaciones lineales. La reflexión en una línea es una transformación afín, por lo que podemos escribir $\sigma_{ij}(x)$ como $A_{ij}x + b_{ij}$, para una matriz apropiada A_{ij} y el vector b_{ij} . Consideremos un orden de búsqueda en profundidad de las regiones, de tal manera que para cada región R_i se conozca una región adyacente R_{i+1} . Para obtener un sistema lineal en x , escribimos

$$\begin{aligned} p_1 &= x, \\ p_2 &= A_{12}x + b_{12} := C_2x + d_2, \\ p_3 &= A_{23}(A_{12}x + b_{12}) + b_{23} := C_3x + d_3 \end{aligned}$$

y así sucesivamente. Esto expresa todos los puntos p_i en términos de p_1 usando $n - 1$ adyacencias entre las regiones. Cada una de las adyacencias restantes ahora da una ecuación de la forma

$$A_{ij}(C_i x + d_i) + b_{ij} = C_j x + d_j$$

Este sistema tiene como máximo la ecuación $3n - 3 - (n - 1) = 2n - 2$ por el Lema 3.3. Si no tiene solución, o una solución única, entonces ya lo tenemos hecho. En el primer caso, no podemos tener un Diagrama de Voronoi. En este último, se indican las coordenadas del primer sitio candidato $p_1 = x$. Los correspondientes sitios se obtienen simplemente por reflexión.

La configuración del sistema, su resolución y las pruebas de contención pueden realizarse en tiempo $O(n)$ mediante métodos estándar. Obsérvese que sólo se necesitan las ecuaciones de las líneas que delimitan las regiones y la información de adyacencia entre las regiones. No se requieren coordenadas de los vértices de la región. Esto es particularmente interesante para el problema de reconocimiento en las dimensiones superiores, en las que el método anterior se generaliza naturalmente.

Teorema 3.1. *Sea C una partición del plano en n regiones convexas dadas por planos medios que soportan las regiones y por adyacencias entre regiones. Un tiempo $O(n)$ es suficiente para decidir si C es un Diagrama de Voronoi y también para restaurar un conjunto adecuado de sitios en caso de su existencia.*

Este resultado es óptimo y el método subyacente es fácil de programar. Una generalización a dimensiones más altas es sencilla. Sin embargo, el método tiene que usarse con cuidado, ya que incluso una ligera desviación del Diagrama de Voronoi correcto (derivado de una medición imprecisa o un error numérico) hará que el método clasifique C como no-Voronoi.

Además, si la partición C es poligonal, entonces las condiciones de contención y reflexión pueden relajarse a:

- (1b) Ortogonalidad: si los polígonos R_i y R_j comparten una arista, e , entonces el segmento de línea $p_i \bar{p}_j$ es ortogonal a e , y
- (2b) Orientación: cualquier rayo que sea paralelo al vector $p_j p_i$ e intersecta e se encuentra con R_i primero.

Curiosamente, la existencia de una figura recíproca p_1, \dots, p_n para C , que ahora se denomina también dual ortogonal, equivale a la propiedad de que C puede realizarse como el estado de equilibrio de una tela de araña: los bordes de C permiten tensiones positivas que equilibran en toda verticalidad de C (propiedad de uso en la estática de estructuras planas). Por tanto, para el Diagrama Voronoi $V(S)$, su conjunto \mathcal{S} de sitios es un posible doble ortogonal.

3.3. Dimensiones más altas

Una importante generalización del Diagrama de Voronoi para los sitios de puntos, es el espacio de d dimensiones euclíadiano \mathbb{R}^d con $d \geq 3$. Naturalmente, el espacio tridimensional (el espacio en que vivimos) tiene un papel predominante, especialmente en lo que se refiere a las aplicaciones en las ciencias naturales. Pero también las dimensiones superiores son fuertemente relevantes, ya sea en aplicaciones directas como la agrupación de datos multivariados, o indirectamente, como la incorporación de un Diagrama Voronoi generalizado en un complejo de celdas poliédricas o una dimensión más grande, con propósitos analíticos o algorítmicos.

Se conservan varias propiedades agradables en dimensiones superiores (por ejemplo, la convexidad de las regiones y con ello, la estructura lineal por piezas), mientras que otras se pierden (por ejemplo, el tamaño del caso más bajo del diagrama). Además, los Diagramas de Voronoi en \mathbb{R}^d ($d \geq 3$) están estrechamente relacionados con objetos geométricos en diferentes dimensiones.

A continuación, mencionaremos algunas características que creemos interesantes.

Consideremos primero el Diagrama de Voronoi clásico en \mathbb{R}^3 . Sea S un conjunto de n sitios en el espacio de tres dimensiones. La bisectriz de dos sitios $p, q \in S$ es el plano perpendicular a través del punto medio del segmento de línea \bar{pq} . La región $VR(p, S)$ de un sitio p es la intersección de los semiplanos limitados por las bisectrices y, por lo tanto, es un poliedro convexo tridimensional. El límite de $VR(p, S)$ consiste en caras (subconjuntos máximos dentro de la misma bisectriz), de bordes o aristas (segmentos de línea máxima en el límite de caras) y de vértices (extremos de los bordes). Las regiones, caras, aristas y vértices de $V(S)$ definen un **conjunto de celdas poliédricas** en tres dimensiones. Este conjunto es *cara a cara*: si dos regiones tienen una intersección no vacía f , entonces f es una cara (borde o vértice) de ambas regiones. El número de caras de $VR(p, S)$ es como máximo $n - 1$; a lo sumo uno por cada sitio $q \in S \setminus p$. Por lo tanto, por la fórmula del poliedro de Euler, el número de aristas y vértices de $VR(p, S)$ es $O(n)$ también. Esto muestra que el número total de componentes del diagrama $V(S)$ en el espacio \mathbb{R}^3 es $O(n^2)$. De hecho, hay configuraciones en S que obligan a cada par de regiones de $V(S)$ a compartir una cara, logrando así su máximo número posible $\binom{n}{2}$.

Por otro lado, los Diagramas de Voronoi de orden superior son generalizaciones naturales y útiles del Diagrama de Voronoi clásico. Dado un conjunto S de n puntos en espacio d -dimensional y un entero k entre 1 y $n - 1$, el Diagrama de Voronoi de orden k de S , para $V_k(S)$, divide el espacio en regiones tales que cada punto dentro de una región fija tiene los mismos

k sitios más cercanos. Por tanto, $V(S)$ sólo es el Diagrama de Voronoi de \mathcal{S} .

Muchas de las aplicaciones del Diagrama de Voronoi clásico son significativas en sus versiones del “orden-k”: la búsqueda del k-vecino más cercano y la colocación más grande del círculo “casi” vacío; o los Diagramas de Voronoi de orden superior aplicados directamente, así como para los problemas de agrupamiento y clasificación. Las regiones de $V_k(S)$ son poliedros convexos, ya que surgen de la intersección de semiplanos de \mathbb{R}^d limitada por simetría de hiperplanos de los sitios. Un subconjunto M de k sitios en \mathcal{S} tiene una región no vacía en $V_k(S)$ si hay una esfera que encierra a M pero ningún sitio en el conjunto de centros de todas esas esferas. Dos diferencias entre un Diagrama de Voronoi de orden 2 y el Diagrama de Voronoi clásico son evidentes. Una región no necesita contener sus sitios de definición, y la bisectriz de dos sitios puede aportar más de un borde (o cara, en dimensiones más altas) al diagrama.

Como ya hemos comentado anteriormente, los Diagramas de Voronoi están estrechamente relacionados con los objetos geométricos en diferentes dimensiones. Estas relaciones y sus implicaciones estructurales y algorítmicas son discutidas (más profundamente y con más detalle) en infinidad de libros y por diferentes autores (Schmitt y Spehner [14], Dewdney y Vranch[16], Joe [17], Edelsbrunner y Seidel[18], Aurenhammer e Imai[19]) y resulta interesante leer sobre ello.

3.4. Aplicaciones

Las aplicaciones de los Diagramas de Voronoi son muchas y por eso se han desarrollado tanto aunque aún hay mucho trabajo por hacer en este terreno ya que es un campo muy activo de investigación. Existe mucha bibliografía al respecto e incluso una web de tipo wiki [22] donde se pueden consultar sus diversas aplicaciones.

Entre sus aplicaciones geométricas se encuentra la construcción de la Triangulación de Delaunay dualizando el Diagrama de Voronoi o la búsqueda del punto $s_i \in \mathcal{S}$ más cercano a un punto del plano, que veremos en el próximo Capítulo 4. Entre sus aplicaciones no geométricas, se encuentran usos ecológicos, como la de determinar la supervivencia de organismos en competencia por alimentos o luz (por ejemplo, árboles en un bosque), y usos en redes sociales para representar las relaciones entre personas.

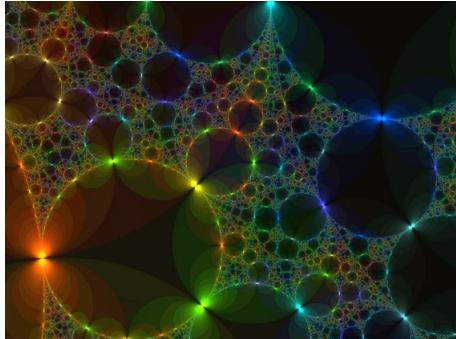
Los grafos aproximados o Diagramas de Voronoi, han sido usados y son fundamentales para estudios de proximidad en espacios vectoriales, son una estructura fundamental en la geometría computacional para solucionar el problema de puntos más cercanos y para determinar áreas de influencia

respecto unos puntos fijados. Aunque es realmente un desafío generalizarlo a espacios métricos porque los algoritmos a construir dependen fuertemente de la información de las coordenadas. Consisten en subdividir un plano, por ejemplo el mapa de una ciudad, en zonas de proximidad a unos ciertos puntos importantes o especiales con los que se generan los diagramas. Una aplicación puede ser la parcelación de la ciudad en zonas en relación a la proximidad de los colegios o, en el caso de la telefonía móvil, la división del espacio en función de la conexión a la antena que los usuarios tienen más próxima. En este sentido, los Diagramas de Voronoi a veces se utilizan para realizar una planificación, como la asignación de hospitales o centros de salud en las ciudades.

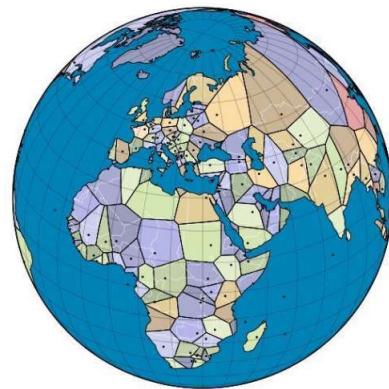
En definitiva, este método científico está muy involucrado en la vida tanto cotidiana como no cotidiana. Algunos ejemplos de problemas de proximidad, aparte de los ya mencionados, son los siguientes:

- Detección de aviones en peligro de colisión, control aéreo, encontrando el par de puntos más cercanos de entre todos los puntos iniciales fijados.
- Ubicación de servicios de urgencia, antenas de telefonía, repetidores de radio y televisión, encontrando el centro de radio mínimo que contiene todos los puntos fijados.
- Posicionamiento de torretas de telefonía móvil o de visión de zonas forestales para el control de fuegos.
- En ecología, para estudiar la supervivencia de especies en competencia en alimentos o luz.
- En economía, para estudiar áreas de mercado de centros en competición.
- En meteorología, para determinar las áreas de precipitación (polígonos de Thiessen).
- En robótica, para la creación de robots que evitan obstáculos.
- En arquitectura, las aplicaciones arquitectónicas de este diagrama, generalmente se encuentran en el diseño o en la urbanística.
- En espacios conceptuales (Geometría del pensamiento) Gärdenfors ([12]) estudia cómo los Diagramas de Voronoi permiten generar una partición de categorías sobre conceptos que resultan dinámicos.
- También son utilizados para obtener espacios de color difusos a partir de escalados adecuados de las celdas de Voronoi.

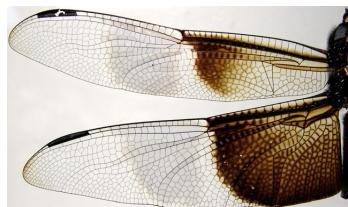
A continuación mostraremos algunas fotografías curiosas en las que veremos distintos ejemplos de aparición de estos diagramas.



(a) Las burbujas de jabón se agrupan buscando la manera de ocupar el menos espacio posible.



(b) El mundo, dividido en Diagramas de Voronoi, en base a las capitales más cercanas.



(c) Podemos encontrarlos en la naturaleza.



(d) Mapa político de España, cuyos puntos son las provincias y las aristas las comunidades autónomas.



(e) Edificio Airespace en Tokyo.

Figura 3.1: Curiosidades de los Diagramas de Voronoi.

Capítulo 4

Triangulación de Delaunay

En este capítulo consideramos una estructura muy relacionada con los Diagramas de Voronoi que tiene una relevancia importante en el campo de la geometría computacional, especialmente en gráficos 3D por computadora. Se trata de una red de triángulos conexa y convexa que cumple la condición de Delaunay. Hablaremos de algunas propiedades importantes que posee y concluiremos hablando del grafo del vecino más cercano, que es un subgrafo de las aristas de la Triangulación de Delaunay.

Este concepto dual, también tiene una historia marcada por redescubrimientos, aunque no tan frecuentes como el caso de los Diagramas de Voronoi. Aunque fue Delaunay quien definió la Teselación (mosaico) usando el método de la esfera vacía, la idea se originó con Voronoi en 1908, que la definió por medio de relaciones entre vecinos, refiriéndose a la estructura resultante como el ensamble de simplices (envolventes convexas). Una de las propiedades importantes de la Teselación de Delaunay en dos dimensiones es que los triángulos individuales son tan equiláteros como sea posible. Dicha triangulación ha sido ampliamente estudiada en el ámbito de la geometría computacional debido a sus aplicaciones en diferentes áreas de la ciencia y la ingeniería. Nos va a ser muy útil para la construcción del Diagrama de Voronoi ya que antes de dibujar cualquier región de Voronoi, primero trazaremos su respectiva Triangulación de Delaunay en la que nos apoyaremos marcando el punto central de cada circunferencia y trazando perpendiculares a las aristas de los triángulos.

4.1. Formalización

La Triangulación de Delaunay (Boris Nikolaevich Delaunay, 1934), $DT(S)$, es el dual del Diagrama de Voronoi con todos sus circuncentros: la circunferencia circunscrita de un triángulo es la circunferencia que contiene los tres vértices del triángulo. Luego, los circuncentros que quedan tras hacer la

Triangulación de Delaunay son los vértices de los segmentos del Diagrama de Voronoi. La unión de estos circuncentros produce el Diagrama de Voronoi, tal y como podemos ver en las siguientes figuras:

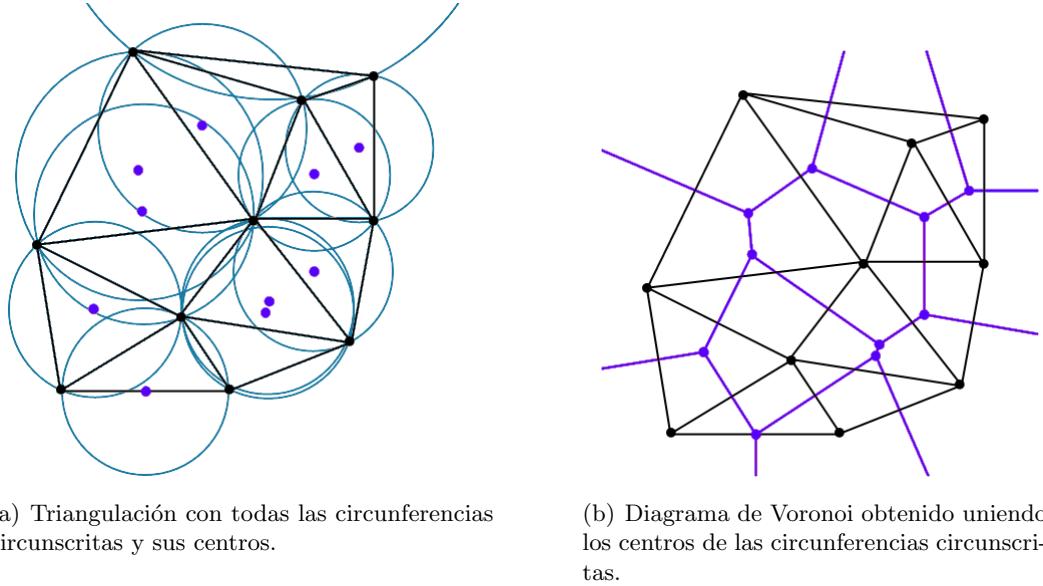


Figura 4.1: La unión de los circuncentros de la Triangulación de Delaunay produce el Diagrama de Voronoi.

Definición 4.1. *La **triangulación** (o red triangular), T , de \mathcal{S} es un conjunto máximo de segmentos de línea no cruzados por los sitios en \mathcal{S} . En otras palabras, T define una partición de la envolvente convexa de \mathcal{S} en triángulos cuyo conjunto de vértices es exactamente \mathcal{S} .*

El número de diferentes particiones de este tipo es grande (de hecho, exponencial) para cada conjunto de puntos de $n \in \mathcal{S}$ en posición general, lo que hace que los resultados de optimidad sean aún más valiosos. Diremos, además, que:

Definición 4.2. *T es **localmente Delaunay** si, para cada uno de sus cuadriláteros convertidos, Q , los correspondientes dos triángulos (adyacentes) tienen circunferencias vacías de vértices de Q .*

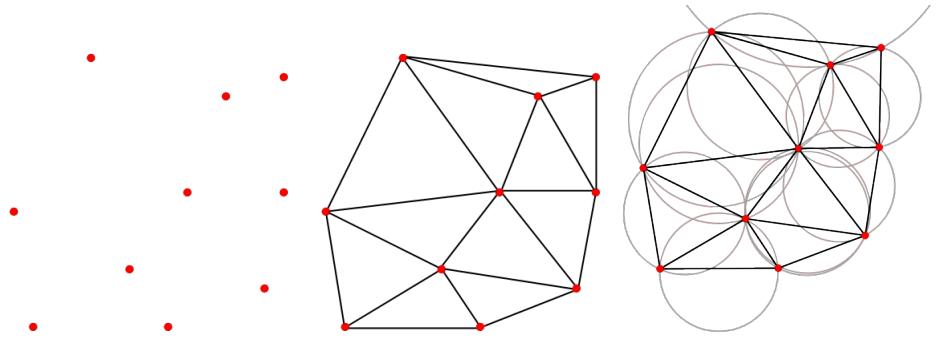
Claramente, $DT(S)$ es localmente Delaunay porque las circunferencias para sus triángulos están totalmente vacías de sitios (ver Teorema 4.1). Curiosamente, la propiedad local también implica la global.

La Triangulación de Delaunay es una de las triangulaciones más interesantes por ser aplicable para la resolución de multitud de problemas aparentemente sin relación entre sí, debido a sus propiedades geométricas, y por contar con algoritmos bastante eficientes para su cálculo. Además, tiene

una serie de propiedades que detallaremos a continuación y hacen que este diagrama pueda servir para conocer el área de influencia de cada punto, encontrar el lugar del plano más alejado de los puntos que actualmente forman la nube, encontrar el punto más cercano a un lugar del plano cualquiera, averiguar dónde situar un punto para que su área de influencia sea mayor teniendo en cuenta los puntos ya existentes en el plano... Con todas estas características, podemos darnos cuenta de por qué las aplicaciones (3.4) en la vida real de este diagrama son incontables y sobre campos muy diferentes.

De manera más formal, diremos que una red de triángulos es una **Triangulación de Delaunay** si todas las circunferencias circunscritas de todos los triángulos de la red son vacías, es decir, no contienen otros vértices aparte de los tres que la definen (condición de Delaunay). Esto es, no debe contener ningún vértice de otro triángulo (definición original para espacios bidimensionales). Si esta definición queremos ampliarla para espacios tridimensionales (para dimensiones más grandes se puede hacer pero no se usa en la práctica), podemos hacerlo usando la esfera circunscrita. Esta condición nos asegura que los ángulos del interior de los triángulos son lo más grandes posibles, por lo que maximiza la extensión del ángulo más pequeño de la red. Además, la triangulación forma la envolvente convexa del conjunto de puntos y es unívoca si no hay más de tres vértices en ningún borde de la circunferencia circunscrita. Esto es muy práctico porque los errores de redondeo son mínimos y es por ésto que las triangulaciones de Delaunay se utilizan en aplicaciones gráficas.

Las siguientes figuras muestran que si queremos construir una triangulación a partir de algunos puntos, es fácil hacerlo simplemente conectando los vértices. Además, con la condición de Delaunay, podemos estudiar si la triangulación a realizar es útil.



(a) De algunos puntos (b) Conectando los vértices (c) Con la condición de De requerimos construir una podremos obtener cualquier launay podemos examinar si la triangulación. launay podemos examinar si la triangulación es útil.

Figura 4.2: Triangulación de Delaunay de 10 puntos en el plano, mostrando la circunferencia de cada triángulo.

4.2. Propiedades

Una vez conocido el Diagrama de Voronoi, pueden definirse una serie de propiedades que permiten calcular una Triangulación de Delaunay a partir de dicho diagrama de forma directa e inequívoca. Es por eso que se dice que el Diagrama de Voronoi y la Triangulación de Delaunay son duales.

Cada triangulación de \mathcal{S} contiene las aristas de la envolvente convexa de \mathcal{S} y sus caras acotadas son triángulos. Su número, como ya vimos, es exactamente igual a $2n - h - 2$, donde h representa los vértices de la envolvente convexa. El número de aristas es $3n - h - 3$. Obsérvese que ambos números, que se derivan fácilmente de la fórmula de poliedro de Euler, son independientes de la forma de triangular el conjunto de puntos \mathcal{S} , por las características de esta fórmula.

Llamaremos a un subconjunto conectado de aristas de una triangulación una **teselación** de \mathcal{S} si contiene las aristas de la envolvente convexa y si cada punto de \mathcal{S} tiene al menos dos aristas incidentes.

La **teselación de Delaunay**, $DT(\mathcal{S})$, es obtenida conectando con un segmento de línea cualesquiera dos puntos p, q de \mathcal{S} para los cuales existe un círculo que pasa por p y q pero no contiene ningún otro sitio de \mathcal{S} en su interior o límite. Las aristas (o bordes) de $DT(\mathcal{S})$ las llamaremos **aristas de Delaunay**.

Lema 4.1. *Dos puntos de \mathcal{S} están conectados por una arista de Delaunay si y sólo si sus regiones de Voronoi tienen una arista adyacente.*

Dos regiones de Voronoi pueden compartir, al menos, una arista de Voronoi, por convexidad. Además, este lema implica que $DT(\mathcal{S})$ es el grafo

teórico dual de $V(S)$ creado mediante aristas rectas. De forma biyectiva, las aristas de Delaunay se corresponden con las aristas de Voronoi; las caras de Delaunay serán los vértices de Voronoi; y los vértices de Delaunay serán las regiones de Voronoi.

Teorema 4.1. *Si el conjunto de puntos \mathcal{S} está en posición general (no hay cuatro puntos coplanares, no hay cinco puntos co-esféricos), entonces $DT(\mathcal{S})$, el dual del Diagrama de Voronoi $V(\mathcal{S})$, es una triangulación de \mathcal{S} , llamada Triangulación de Delaunay. Tres puntos de \mathcal{S} dan como resultado exactamente un triángulo de Delaunay si el círculo que definen no contiene ningún otro punto de \mathcal{S} .*

La Triangulación de Delaunay posee varias características acerca de sus propiedades de optimización, sus estructuras como subgrafos y su flexibilidad para adaptarse a dimensiones más altas. Contiene un tetraedro para cada vértice, un triángulo para cada arista y una arista para cada cara de $V(\mathcal{S})$. Equivalentemente, $DT(\mathcal{S})$ puede definirse usando la propiedad de esfera vacía, declarando un tetraedro extendido por \mathcal{S} como Delaunay si su circunferencia está vacía de sitios en \mathcal{S} . Los centros de estas esferas vacías son sólo los vértices de $V(\mathcal{S})$. $DT(\mathcal{S})$ es una partición de la envolvente convexa de \mathcal{S} en tetraedros, siempre que \mathcal{S} esté en posición general. Obsérvese que, para configuraciones especiales de puntos en el espacio de tres dimensiones, las aristas de $DT(\mathcal{S})$ ya pueden formar el grafo completo en \mathcal{S} , es decir, el grafo donde todos los pares de vértices $\binom{n}{2}$ coinciden con un borde. El conjunto de celdas $DT(\mathcal{S})$ es llamado la **tetraedrización de Delaunay** para \mathcal{S} .

4.2.1. Propiedades de optimización

Veamos ahora algunas características avanzadas del diagrama dual de Voronoi.

La Triangulación de Delaunay, $DT(\mathcal{S})$, de un conjunto \mathcal{S} de n sitios en el plano posee una serie de propiedades agradables y útiles, muchas de las cuales son bien conocidas y comprendidas hoy en día. Al ser el dual geométrico del Diagrama de Voronoi $V(\mathcal{S})$, $DT(\mathcal{S})$ comprende la información de proximidad inherente a \mathcal{S} de una manera compacta. Aquí nos fijamos en $DT(\mathcal{S})$ como una triangulación por sí misma y nos concentramos en los parámetros que son optimizados por $DT(\mathcal{S})$ sobre todas las posibles triangulaciones del punto establecido de \mathcal{S} . Debemos tener en cuenta la definición más teórica de triangulación:

Teorema 4.2. *Una triangulación de \mathcal{S} es localmente Delaunay si y sólo si es igual a la Triangulación de Delaunay, $DT(\mathcal{S})$.*

Demostración. Sea T una triangulación de \mathcal{S} y supongamos que T es localmente Delaunay. Se muestra que, para cada triángulo Δ de T , su circunferencia $C(\Delta)$ está vacía de sitios en \mathcal{S} .

Asumiendo el contrario, dejemos s dentro de $C(\Delta)$ para algunos $s \in \mathcal{S}$ y algunos Δ en T . Observamos que $s \notin \mathcal{S}$ y sea e la arista de Δ más cercana a s . Supongamos, sin pérdida de generalidad, que (Δ, e, s) maximiza el ángulo extendido por e en s , para todos los triples (triángulo, arista, sitio).

Debido a s , la arista e no puede ser una arista de la envolvente convexa de \mathcal{S} . Dejemos que el triángulo Δ' sea adyacente a Δ en e , y sea s' el tercer vértice de Δ' . Como T es localmente Delaunay, s' está fuera de $C(\Delta)$, por lo tanto $s \neq s'$. Ahora, observamos que s está encerrado por $C(\Delta')$, y sea e' la arista de Δ' más cercana a s . El ángulo en s para (Δ, e', s) es mayor que el de (Δ', e, s) , lo que da una contradicción. \square

Definición 4.3. Un *giro de arista* en una triangulación T de \mathcal{S} es el intercambio de las dos diagonales en uno de los cuadriláteros de conversión de T . Lo llamaremos *giro bueno* si, después del giro, la triangulación dentro del cuadrilátero (que consta de sólo 2 triángulos) es localmente Delaunay. El intercambio repetido de diagonales en el mismo cuadrilátero siempre produce una secuencia alternada de buenos malos giros.

Por tanto, el anterior teorema 4.2 ahora puede usarse para probar que $DT(S)$ optimiza varias medidas de calidad, mostrando que cada buena inversión incrementa la calidad. Cualquier secuencia de buenos giros termina entonces en el óptimo global, la Triangulación de Delaunay. La longitud de una secuencia de este tipo está limitada por $\binom{n}{2}$, ya que una arista que se desvía no puede volver a aparecer.

Una de las medidas de calidad más prominentes se refiere a los ángulos que ocurren en una triangulación. Recordemos que el número de aristas (y por tanto de los triángulos) no depende del camino de las triangulaciones \mathcal{S} , y sea t el número de triángulos de \mathcal{S} . La **angulosidad** de una triangulación se define como la lista ordenada de ángulos $(\alpha_1, \dots, \alpha_{3t})$ de sus triángulos, en orden ascendente.

Definición 4.4. Una triangulación se llama *equiangular* si posee lexicográficamente la mayor angularidad entre todas las triangulaciones posibles para \mathcal{S} . De hecho, cada buen giro aumenta la angularidad. En el caso de cocircularidades entre los sitios, $DT(S)$ no es una triangulación completa, como vimos en las propiedades básicas al comienzo de esta sección 4.2.

Teorema 4.3. Sea \mathcal{S} un conjunto finito de sitios en el plano. Una triangulación de \mathcal{S} es equiangular solamente si es una terminación de $DT(S)$.

La triangulación equiangular maximiza, obviamente, el ángulo mínimo que ocurre en todos los triángulos. Esta propiedad es deseable para aplicaciones de generación de malla en modelado de terreno o para el método de elementos finitos. Por este teorema 4.3, tales triangulaciones pueden ser computadas en tiempo $O(n \log(n))$ con los algoritmos de Triangulación de Delaunay que veremos, más adelante, en el Capítulo 5.

4.2.2. El vecino más cercano

La Triangulación de Delaunay $DT(S)$ de un conjunto S de n puntos contiene, como subgrafos, diversas estructuras con diversas aplicaciones (como por ejemplo árboles y ciclos mínimos).

Nosotros nos centraremos en uno de los ejemplos más importantes, el **grafo del vecino más cercano** de S , problema de distancia que el Diagrama de Voronoi resuelve directamente: para cada punto del conjunto S , se requiere un vecino más cercano en S . Esto puede responderse construyendo en primer lugar un Diagrama de Voronoi para luego localizar la celda del diagrama que contiene a dicho punto. La solución ofrecida por este diagrama se basa en el siguiente hecho.

Lema 4.2. *Sea $S = P \cup Q$ una descomposición disjunta del conjunto de puntos S , y $p_0 \in P$ y $q_0 \in Q$ sean tales que:*

$$d(p_0, q_0) = \min_{p \in P, q \in Q} d(p, q) \quad (4.1)$$

Entonces, las regiones de Voronoi de p y q son adyacentes a la arista en $V(S)$.

Demostración. En caso contrario, el segmento de línea p_0q_0 contiene un punto z que pertenece al cierre de alguna región de Voronoi $VR(r, S)$, donde $r \neq p_0, q_0$. Supongamos que r pertenece a Q ; El caso $r \in P$ es simétrico. Desde $z \in VR(r, S)$ se sigue que $d(z, r) \leq d(z, q_0)$, de ahí

$$d(p_0, r) \leq d(p_0, z) + d(z, r) \leq d(p_0, z) + d(z, q_0) = d(p_0, q_0) \leq d(p_0, r)$$

Por minimalidad de $d(p_0, q_0)$. Por lo tanto, la igualdad debe mantenerse, y obtenemos $d(p_0, r) = d(p_0, q_0)$ y $d(z, r) = d(z, q_0)$.

Pero como p_0 está en la bisectriz $B(q_0, r)$, cada punto z en el interior de $q_0\bar{p}_0$ es estrictamente cercano a q_0 que a r , luego llegamos a una contradicción. \square

Si aplicamos este lema 4.2 a los subconjuntos q y $S \setminus p$, se sigue que el vecino más cercano de p está incluido en una región vecina de Voronoi (por

ejemplo, es un vecino de Delaunay). Por lo tanto, es suficiente examinar, para cada $p \in \mathcal{S}$, todos los vecinos de p en el Diagrama de Voronoi, y seleccionar el más cercano de ellos. De esta manera, cada arista de $V(S)$ se accede dos veces. Dado que su número total es lineal (ver Lema 3.3), podemos resolver el problema del vecino más cercano construyendo el Diagrama de Voronoi.

Teorema 4.4. *Dado un conjunto \mathcal{S} de n puntos en el plano, un tiempo $O(n\log(n))$ y espacio lineal son suficientes para determinar, para cada $p \in \mathcal{S}$, un vecino más cercano en \mathcal{S} .*

Si para cada p su vecino más cercano en \mathcal{S} es conocido, podemos determinar fácilmente un par de puntos cuya distancia es mínima.

Corolario 4.1. *El par más cercano entre n puntos en el plano se puede determinar dentro de un tiempo $O(n\log(n))$ y espacio lineal.*

El par más cercano en \mathcal{S} corresponde a la arista más corta de la Triangulación de Delaunay $DT(\mathcal{S})$, como lo muestra el Lema 4.2. El segundo par más cercano, debe estar en $DT(\mathcal{S})$ también, ya que ambas aristas están en el árbol de expansión mínimo de \mathcal{S} , que es un subgrafo de $DT(\mathcal{S})$.

Teorema 4.5. *Los k pares más cercanos de un conjunto \mathcal{S} de n puntos se pueden calcular en tiempo $O((n+k)\log(n))$ y espacio $O(n+k)$.*

A parte de este ejemplo, existen algoritmos eficientes para programar estas estructuras que se siguen del hecho de que $DT(\mathcal{S})$ tiene un tamaño $O(n)$ y pueden ser construidos en tiempo $O(n\log(n))$ en el plano. Este efecto positivo se pierde parcialmente en dimensiones más altas, ya que $DT(\mathcal{S})$ puede ser el grafo completo en \mathcal{S} ya en tres dimensiones.

Podemos concluir diciendo que las triangulaciones de Delaunay son una herramienta muy útil para generar mallas irregulares a partir de un conjunto de puntos. Existen buenos algoritmos para su cálculo que han sido implementados con éxito. Sin embargo, en general, las implementaciones que se emplean no utilizan la potencia y capacidad de las tarjetas gráficas. Por ejemplo, en el desarrollo de una aplicación informática para emular la evolución de bosques, se calculan Diagramas de Voronoi de forma gráfica como medio de solventar el problema del cálculo de Diagramas de Voronoi generalizados con pesos tanto aditivos como multiplicativos. A continuación, hablaremos de algunas de las posibles estrategias para calcular la Triangulación de Delaunay a partir de un conjunto de puntos de entrada.

Capítulo 5

Algoritmos para el cálculo de Diagramas de Voronoi

Este capítulo se encuentra a caballo entre la geometría y la computación. Explicaremos, de manera más teórica, algunos de los algoritmos más conocidos y utilizados para la implementación de los Diagramas de Voronoi. Estos algoritmos nos serán muy útiles en el desarrollo y realización de la librería para su cálculo y gestión, que veremos en el siguiente capítulo 6.

El Diagrama de Voronoi, $V(S)$, y la Triangulación de Delaunay, $DT(S)$, muestran la influencia de proximidad que \mathcal{S} ejerce de una manera simple e intuitiva, con una descripción de tamaño lineal solamente. Ya hemos revelado en los Capítulos 3 y 4 el potencial intrínseco oculto que poseen. Ahora, mostraremos que se prestan a varios métodos eficientes de construcción.

Conviene recordar la definición 3.8 que nos será útil para entender a qué nos referimos cuando hablamos de complejidad o tiempo empleado en los distintos algoritmos. Veremos distintas maneras de computar una representación geométrica del Diagrama de Voronoi y su dual. Por simplicidad, vamos a asumir que, de los n puntos de \mathcal{S} , no hay cuatro de ellos que sean cocirculares y no hay tres que sea colineales. Como vimos en el Teorema 4.1, podemos referirnos a $DT(S)$ como la Triangulación de Delaunay. Todos los algoritmos que presentaremos pueden funcionar sin esta suposición de posición general. Además, pueden ser generalizados para otras métricas distintas a la euclídea y a otros lugares que no sean puntos.

Cualquiera de las dos estructuras permite desplazar eficientemente los bordes incidentes a un vértice dado, o bordeando una cara dada. La estructura de cuatro bordes ofrece la ventaja adicional de describir, al mismo tiempo, un gráfico plano y su dual, de modo que puede usarse para construir tanto el Diagrama de Voronoi como la triangulación de Delaunay. De la DCEL (Double Connected Edge List) de $V(S)$ podemos derivar el con-

junto de triángulos que constituyen la Triangulación de Delaunay en tiempo lineal. Por el contrario, del conjunto de todos los triángulos de Delaunay el DCEL del Diagrama de Voronoi se puede construir en el tiempo $O(n)$. Por lo tanto, cada algoritmo para calcular una de las dos estructuras se puede utilizar para calcular el otro, dentro de un esfuerzo $O(n)$ de tiempo extremo.

En cada estructura, un registro está asociado con cada borde e que almacena la siguiente información: los nombres de los dos extremos de e ; referencias a los bordes en el sentido de las agujas del reloj o en sentido contrario a las agujas del reloj junto a e alrededor de sus extremos; finalmente, los nombres de las caras a la izquierda y a la derecha de e . La necesidad de espacio de ambas estructuras es $O(n)$.

5.1. Algoritmo de Fuerza Bruta

Este algoritmo es el más simple posible y consiste en explotar la geometría de cada región de Voronoi y es una primera aproximación para la construcción del Diagrama de Voronoi. Mediante el cálculo explícito de los $n - 1$ semiplanos originados debido a los bisectores trazados con respecto a los demás sitios, se construye su región de Voronoi para cada lugar. A continuación, se computa la intersección de estos $n - 1$ semiplanos para dar origen a la región de Voronoi. Es decir, consiste en probar todas las opciones posibles.

Este algoritmo tiene muchas desventajas debido al cálculo explícito de los semi-planos. Además, su intersección puede provocar problemas de precisión en la computadora que pueden ser generados por una versión incorrecta del Diagrama de Voronoi $V(S)$. Por otro lado, no podemos obtener información inmediata de los vecinos de cada sitio que se pueda aprovechar. Dado que se trata de un algoritmo ineficiente, también su complejidad computacional es bastante alta ya que está en el orden de $O(n^2 \log(n))$.

En las siguientes secciones describimos varios algoritmos que calculan todo el Diagrama de Voronoi dentro de este tiempo. Estos algoritmos son (asintóticamente) los peores casos óptimos.

5.2. Algoritmo Incremental

Este algoritmo se basa, de manera general, en, supuesto construido el diagrama para k puntos, construir el diagrama para $k+1$. Emplea un tiempo de $O(n)$ en la inserción de cada nuevo punto, con una complejidad total de $O(n^2)$ ya que la región de p puede tener hasta $n - 1$ aristas, para $n = |S|$ (donde $||$ indica la cardinalidad, es decir, el número de elementos en el

conjunto). A pesar de su complejidad cuadrática, este ha sido el método más popular y utilizado para construir el Diagrama de Voronoi.

Básicamente, la idea es que la secuencia cíclica de aristas de la nueva región de Voronoi $VR(p, S)$ tiene que ser construida y las partes invalidadas del diagrama $V(S \setminus p)$ que estén dentro de $VR(p, S)$, borradas.

Suponemos tener un Diagrama de Voronoi $V(S)$ para n sitios y queremos construir el diagrama $V(S')$ después de agregar un sitio s . Además, suponemos que s pertenece a uno de los círculos asociados con más de un vértice de Voronoi teniendo en cuenta que no pueden ser vértices de $V(S')$ ya que no se cumpliría la condición de Delaunay.

Sea $V(S_k)$ el Diagrama de Voronoi para los k primeros sitios s_1, s_2, \dots, s_k , el algoritmo comienza con $V(S_k)$ cuando $k = 2$ o $k = 3$ sitios y se modifica incrementalmente agregando los sitios uno a uno. La idea del algoritmo ([10]) es convertir $V(S_k - 1)$ a $V(S_k)$ para cada nuevo sitio s_k que se agregue.

Por tanto, el algoritmo (pseudocódigo) es el siguiente: Supóngase construido $V(S_k - 1)$ y que se agrega un nuevo sitio s_k :

- Encontrar el sitio s_i cuya celda de Voronoi contenga a s_k .
- Dibujar la mediatrix entre s_i y s_k .
- La mediatrix cruza la frontera de $V(S_i)$ en dos puntos: x_1 y x_2 , entrando en la región de Voronoi adyacente.
- Encontrar el punto en el que la bisectriz cruza el límite de la región de Voronoi adyacente.
- Encontrar la secuencia de segmentos perpendiculares de los sitios vecinos hasta llegar al punto de partida.

Veámoslo gráficamente para entender de manera visual lo siguiente:

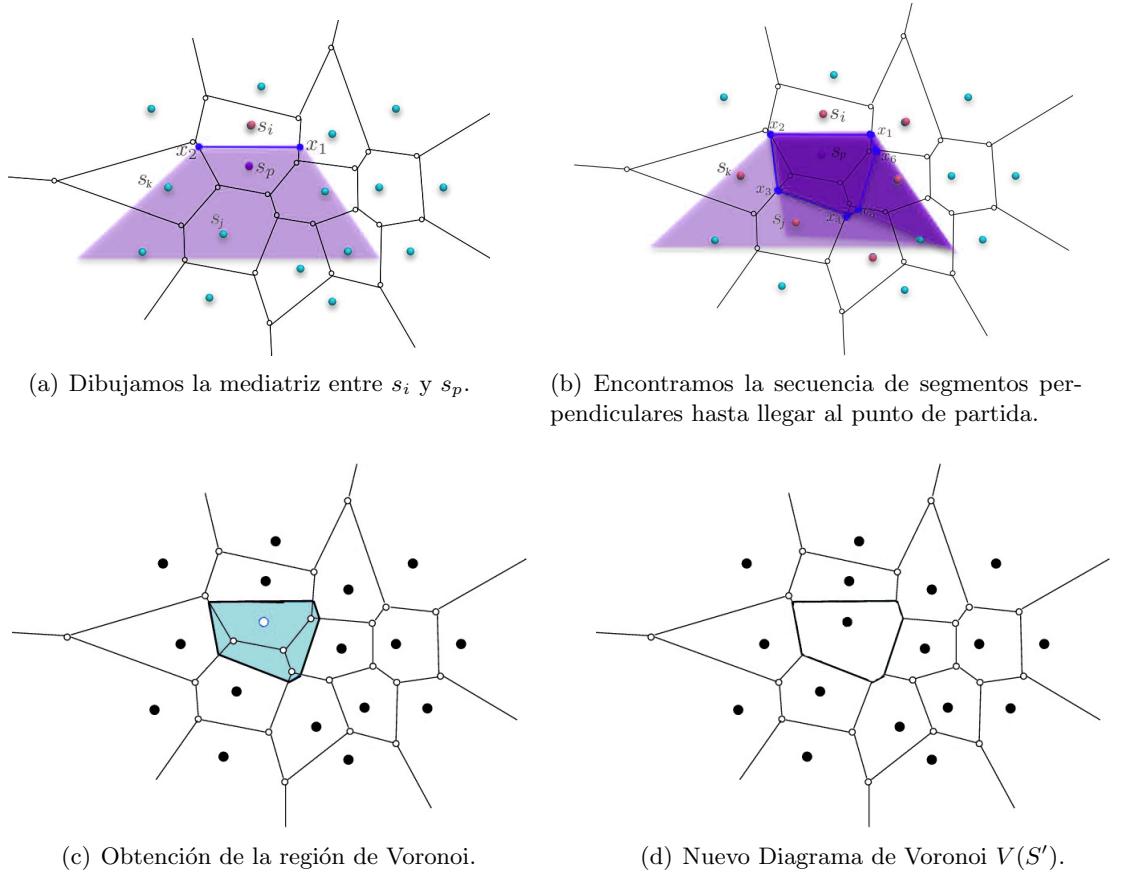


Figura 5.1: Método Incremental del Diagrama de Voronoi.

El sitio s_p está a la izquierda del segmento dirigido x_1x_2 . Dicho segmento x_1x_2 divide a la celda convexa en dos partes: la de la izquierda pertenece a la nueva celda $V(s_p)$ y, además, la mediatrix $x_1\bar{x}_2$ es una arista de la frontera del nuevo polígono $V(s_p)$. Entonces, para encontrar el punto de intersección x_3 habría que trazar la mediatrix $s_p\bar{s}_j$ hasta unirla con la frontera de $V(s_j)$. Por tanto, la nueva arista de la región sería el segmento $x_2\bar{x}_3$.

De manera similar encontramos la secuencia de segmentos de mediatrixes de s_p y sus sitios vecinos hasta llegar al punto de intersección inicial x_1 . La secuencia $x_1x_2, x_2x_3, \dots, x_{m-1}x_m, x_mx_1$ es la frontera en orden contrario a las manecillas del reloj de la celda de Voronoi del nuevo sitio s_p , $V(s_p)$. Finalmente, borramos de $V(S)$ la estructura que queda dentro del nuevo polígono $V(s_p)$ para obtener $V(S')$.

La inserción incremental funciona bien en el caso de dimensiones superiores y también para ciertos tipos de Diagramas de Voronoi generalizados y sus duales. Una técnica similar a la inserción incremental es la búsqueda in-

cremental. Se inicia con un solo trazo de Delaunay y luego descubre nuevos, de manera incremental, creando triángulos a partir de bordes de triángulos previamente descubiertos.

5.3. Algoritmo de Bowyer-Watson

Este algoritmo es conocido a veces como el Algoritmo de Bowyer o el Algoritmo de Watson ya que Adrian Bowyer y David Watson lo idearon independientemente uno del otro al mismo tiempo, y cada uno publicó un papel en la misma edición del *"The Computer Journal"* .

En la geometría computacional, el algoritmo de Bowyer-Watson es un método para calcular la Triangulación de Delaunay de un conjunto finito de puntos en cualquier número de dimensiones aunque también puede usarse para obtener un Diagrama de Voronoi de los puntos. Es un algoritmo incremental que funciona agregando puntos, uno cada vez, a una Triangulación de Delaunay válida de un subconjunto de los puntos deseados. Después de cada inserción, se suprimen los triángulos cuyas circunferencias contienen el nuevo punto, dejando un agujero poligonal en forma de estrella que se vuelve triangular con el nuevo punto. Al utilizar la conectividad de la triangulación para localizar los triángulos a eliminar de manera eficiente, el algoritmo puede tomar operaciones $O(n \log(n))$ para triangular n puntos, aunque existen casos degenerados especiales donde esto llega hasta $O(n^2)$.

La base de este algoritmo se basa en la propiedad del círculo vacío circunscrito. Concretamente, en el hecho de que la circunferencia circunscrita de cada triángulo no puede contener otro nodo de la triangulación.

El siguiente pseudocódigo describe una implementación básica del algoritmo Bowyer-Watson ([13]). La eficiencia se puede mejorar de varias maneras. Por ejemplo, la conectividad triangular puede usarse para localizar los triángulos que contienen el nuevo punto en su circunferencia, sin tener que comprobar todos los triángulos.

- La función Bowyer-Watson recibe como parámetros las coordenadas de los puntos que definen una triangulación.
- Crear un triángulo lo suficientemente grande como para contener completamente todos los puntos de entrada.
- Para cada punto de la triangulación, agrega todos los puntos de uno en uno.
- Buscar todos los triángulos que ya no son válidos debido a la inserción.
- Si el punto está dentro de la circunferencia del triángulo, poner triángulo como inválido.

- Elimina los triángulos de la estructura creando un hueco.
- Para cada borde en triángulo, si el borde no es compartido por ningún otro triángulo de los inválidos, calcula la frontera del hueco.
- Para cada triángulo inválido, eliminarlos de la estructura de datos y eliminar el triángulo de la triangulación.
- Para cada arista en polígono se vuelve a triangular el hueco.
- Para cada triángulo en triangulación, limpiar los triángulos externos a la lista de vértices.

Veámoslo de manera gráfica:

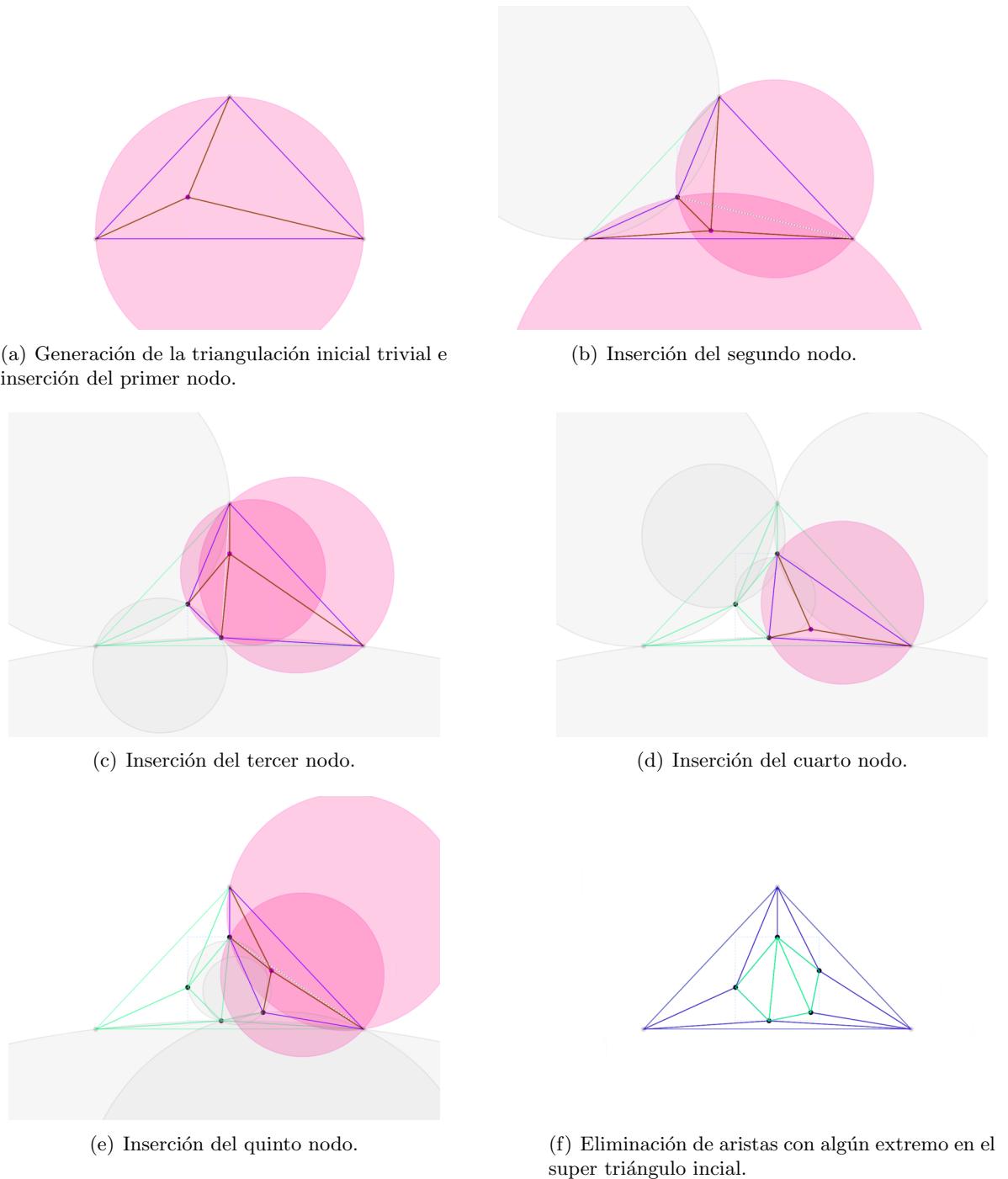


Figura 5.2: Algoritmo Bowyer-Watson.

5.4. Algoritmo Divide y Vencerás

Otra manera de construir el Diagrama de Voronoi es con el algoritmo divide y vencerás, que es el algoritmo óptimo del peor caso determinista, en tiempo $O(n \log(n))$. Aunque el algoritmo resulta bastante difícil de implementar, su complejidad es asintóticamente óptima.

El conjunto de puntos de sitios, \mathcal{S} , está dividido por una línea en dos subconjuntos L y R de igual tamaño. Entonces, los Diagramas de Voronoi $V(L)$ y $V(R)$ están computados de manera recursiva. Esto es, se aplica la misma estrategia para los conjuntos de puntos (más pequeños) L y R . Si sólo dos o tres puntos se dejan en un conjunto, sus diagramas se construyen directamente en un tiempo $O(1)$ (orden constante). La parte esencial se basa en encontrar la cadena o línea divisoria, es decir, el conjunto de todos los bordes de Voronoi de $V(S)$ que separan regiones de sitios en L de regiones de sitios en R , y mezclar $V(L)$ y $V(R)$ para obtener $V(S)$. Durante la recursión, las líneas divisorias, tanto verticales como horizontales, pueden ser fácilmente encontradas si los sitios en \mathcal{S} están ordenados por sus coordenadas x e y de antemano.

Veamos ahora cuáles son los pasos fundamentales de dicho algoritmo de manera gráfica:

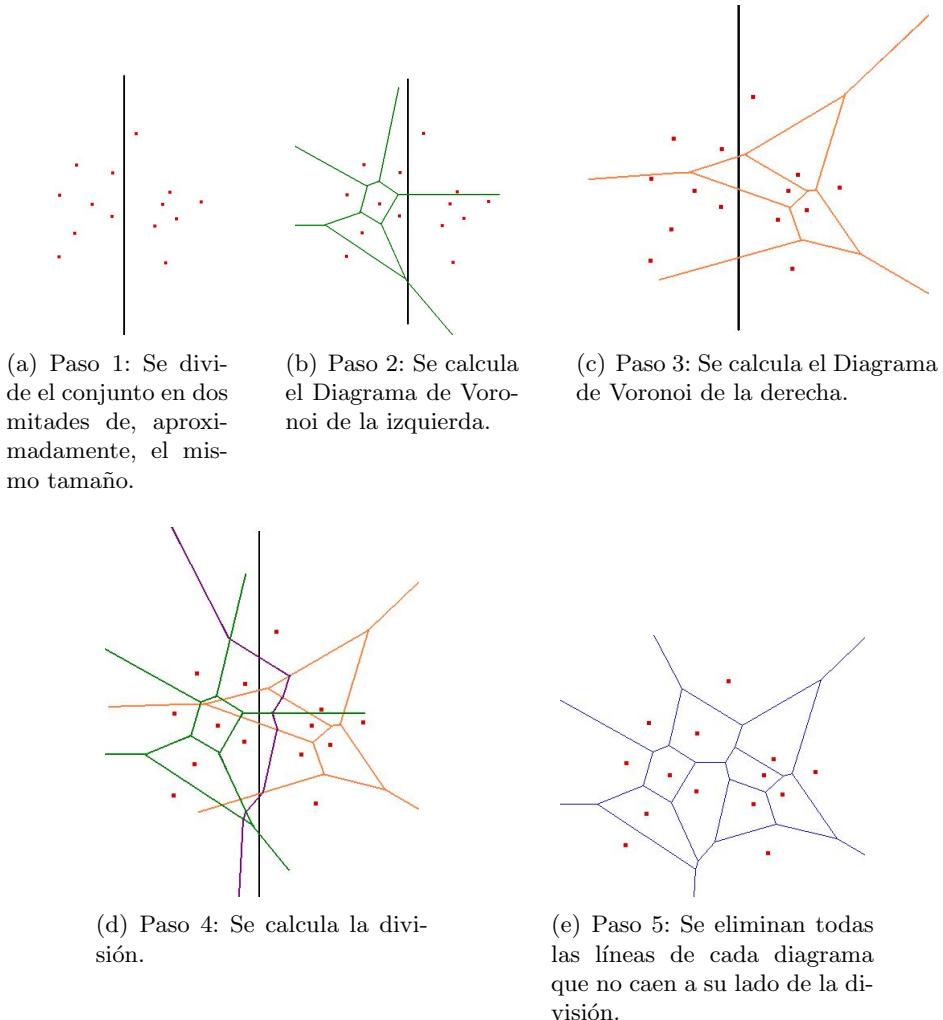


Figura 5.3: Algoritmo Divide y Vencerás. [11]

Los algoritmos de Divide y Vencerás permiten la paralelización. De hecho, algoritmos eficientes para calcular en paralelo el Diagrama de Voronoi o la Triangulación de Delaunay han sido propuestos por diferentes autores (Blelloch, Edelsbrunner, Shi). Estos autores destacan un algoritmo que utiliza un mapa de levantamiento para \mathcal{S} para construir una cadena de bordes Delaunay que divide \mathcal{S} . Demuestran, experimentalmente, que su implementación es comparable en trabajo a los mejores algoritmos secuenciales.

En todo lo referente a este algoritmo, hemos hecho énfasis en que lo complejo está en la fase ascendente (cómo realizar el paso de la fusión). Este paso se complica considerablemente para los Diagramas de Voronoi en un ajuste más general, porque la cadena de fusión puede ciclar e incluso

desconectarse.

Una aproximación alternativa, con énfasis en la fase que va desde arriba hacia abajo para realizar el paso de división, se basa en la construcción del eje mediano de una forma general y explota la estructura en árbol de un eje mediano para calibrar la etapa de división.

5.5. Algoritmo de Fortune

Hasta mediados de los ochenta, la mayoría de las implementaciones para computar el Diagrama de Voronoi usaban el algoritmo incremental cuadrático, admitiendo su mayor lentitud para evitar la complejidad del código del algoritmo Divide y Vencerás. En 1985, Fortune inventó un inteligente algoritmo de barrido del plano que resulta tan simple como el incremental, pero en tiempo $O(n \log(n))$ y es capaz, con esta complejidad óptima, de calcular el Diagrama de Voronoi de una nube de puntos en el plano. Es bastante más complicado que cualquier otro algoritmo de barrido del plano en los que la clave es la capacidad para descubrir todos los eventos próximos de manera eficiente. Por ejemplo, en el algoritmo de barrido del plano para encontrar la intersección de segmentos de línea, éstas son descubiertas antes de que la línea de barrido llegue a ellas (son agregados como eventos futuros).

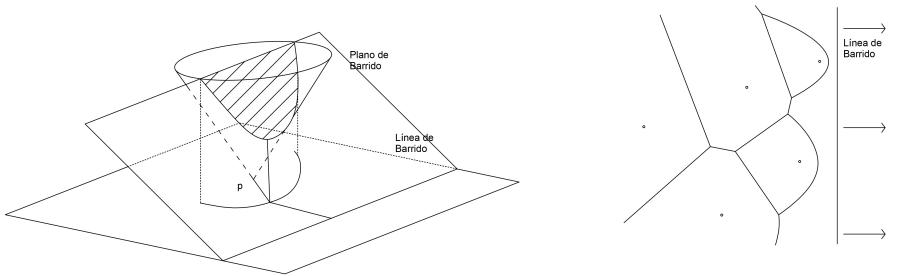
El problema con el Diagrama de Voronoi es el de predecir cuándo y dónde se producirán los próximos eventos. Imaginemos que detrás de la línea de barrido ya se ha construido el Diagrama de Voronoi basándose en los sitios que se han encontrado hasta ahora en el barrido. La dificultad es que un sitio que queda por delante de la línea de barrido podría generar un vértice de Voronoi que se encuentre detrás de la línea de barrido.

Fortune hizo la inteligente observación de que podía calcularse el Diagrama de Voronoi mediante barrido del plano construyendo una versión “distorsionada” de éste pero que es topológicamente equivalente. Esta versión distorsionada del diagrama se basa en una transformación que modifica la forma en que las distancias son medidas en el plano. El diagrama resultante tiene la misma estructura topológica que el Diagrama de Voronoi, pero sus aristas son arcos parabólicos, en vez de segmentos de línea recta [20].

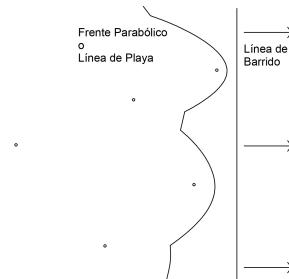
En primer lugar, habrá una línea de barrido horizontal, moviéndose de arriba a abajo. También vamos a mantener una curva x-monótona llamada **línea de playa**. A medida que la línea de barrido se mueve hacia abajo, la línea de playa va justo detrás siguiéndola.

Veamos ahora el mecanismo de barrido de este algoritmo: partimos de un plano de barrido que se mueve de izquierda a derecha sobre el plano $x - y$ con un ángulo de inclinación igual al de los conos.

El plano de barrido intersecta con el plano $x - y$ en lo que hemos llamado línea de barrido. La intersección del plano de barrido con el cono proyectada sobre el plano $x - y$ es una parábola con foco en el punto p y cuya directriz (línea que determina las condiciones de generación de otra línea) es la línea de barrido. Si hacemos opaco el plano de barrido, entonces éste oculta los conos de la derecha de la línea de barrido; los conos visibles que todavía no han sido totalmente barridos por el plano de barrido proyectarán sobre el plano $x - y$ un conjunto de parábolas apuntando hacia la izquierda, con ejes paralelos y distintas anchuras.



(a) Círculos de elevación sobre el cono.

(b) Proyección en el eje $x - y$.

(c) La frontera de la unión de todas las parábolas es lo que se denomina frente parabólico o línea de playa.

Figura 5.4: Mecanismo de barrido del Algoritmo Fortune.

Nos damos cuenta que a la izquierda, el plano de barrido está por encima de algunos conos y no bloquea sus vistas desde abajo. Sin embargo, hacia la derecha, como el plano de barrido es opaco, oculta todos los conos y no permite su visión desde abajo. Si observamos por debajo del plano $x - y$, cada arco del frente parabólico cae en una región de Voronoi y cada punto de corte entre dos arcos consecutivos del frente parabólico cae en uno de

los lados del Diagrama de Voronoi. Es decir, cada punto de un lado del Diagrama de Voronoi será un punto de corte entre dos arcos consecutivos del frente parabólico.

En la evolución del frente parabólico surgen dos tipos de eventos ([7]):

1. Eventos de Sitio

Cuando la línea de barrido pasa justo por encima de un punto, una parábola degenerada se une al frente parabólico y parte un arco de dicho frente en dos. La evolución posterior de la parábola durante el proceso de barrido dará lugar a una nueva arista del Diagrama de Voronoi. Por tanto diremos que hay un evento de punto cuando la línea de barrido pasa justo por encima de uno de los puntos de la nube. Este es el único modo de añadir un nuevo arco al frente parabólico y provoca la aparición de un nuevo lado del Diagrama de Voronoi.

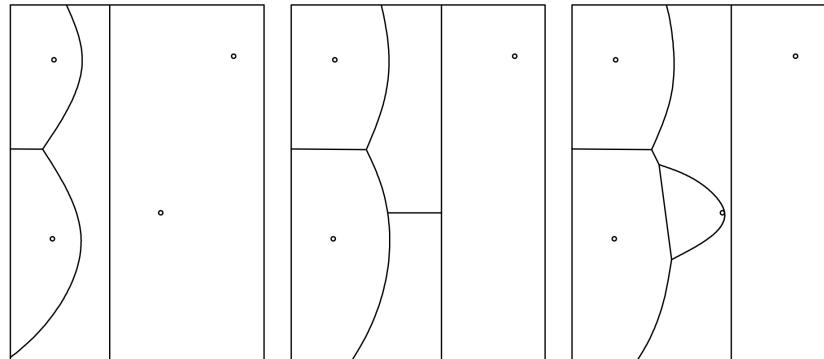


Figura 5.5: Creación de una arista de Voronoi.

2. Eventos de Círculo

Ocurre cuando un arco va decreciendo en el frente parabólico hasta convertirse en un punto y desaparecer de dicho frente.

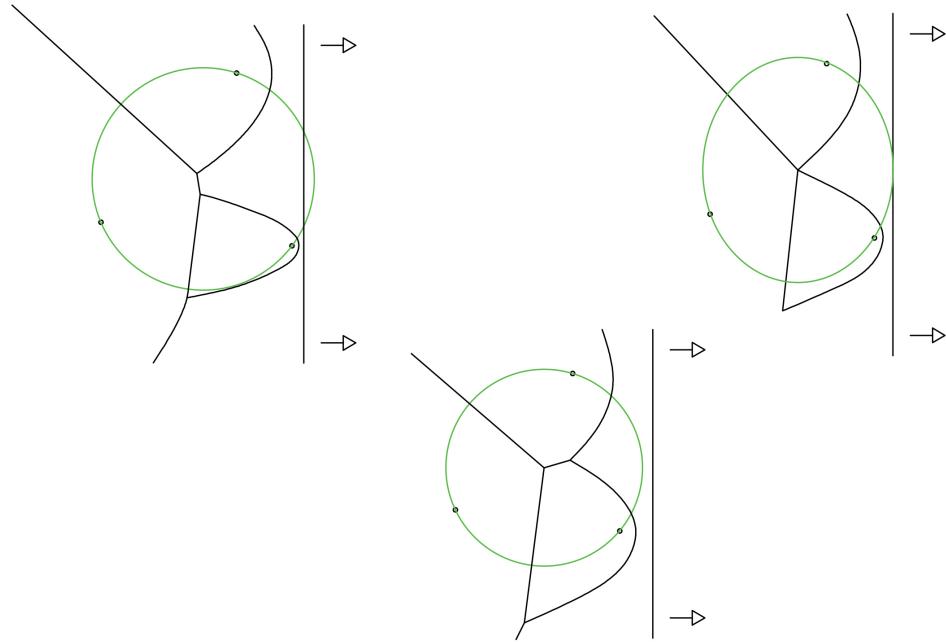


Figura 5.6: Desaparición de un punto del frente parabólico.

El pseudocódigo de este algoritmo, que necesita tres tipos de estructuras de datos (estructura para guardar la parte del diagrama calculada hasta el momento; cola de prioridades que permita guardar los eventos de sitio y de círculo que debe procesar la línea de barriado; árbol de búsqueda binario que guarde la línea de playa) sería el siguiente ([7]):

Entrada: $S := s_1, \dots, s_n$ un conjunto de n puntos en el plano. Salida: El Diagrama de Voronoi $V(S)$ dentro de un rectángulo de restricción.

- Inicializar la cola de eventos Q con todos los puntos.
- Mientras Q no esté vacío.
- Hacer considerar el evento con mayor coordenada y de Q .
- Si el evento es un Evento de Sitio s_i , entonces TratarEventoSitio(s_i).
- En caso contrario TratarEventoCírculo(s_l), donde s_l es el punto más bajo del círculo que causa el evento.
- Eliminar el evento de Q .

- Los nodos internos presentes todavía en T (Árbol binario) pertenecen a las medio-aristas infinitas del Diagrama de Voronoi.

Calcular un rectángulo que contenga todos los vértices del Diagrama de Voronoi en su interior, uniéndole las semi-aristas infinitas.

Podemos observar, finalmente, que la técnica de barrido del plano reduce un problema estático bidimensional (por ejemplo, la construcción de un Diagrama de Voronoi en el plano) a un problema dinámico unidimensional (aquí, el mantenimiento de la representación de frente parabólico en la línea de barrido). Esto nos permite utilizar eficientes estructuras de datos unidimensionales como árboles binarios balanceados y colas de prioridad.

5.6. Problemas de implementación

Veamos cuáles son las cuestiones de implementación, incluidas las configuraciones degeneradas que podrían ocurrir en la entrada o durante la ejecución de dichos algoritmos, junto con las preguntas numéricas generales que surgen en el cálculo de objetos geométricos.

Los problemas numéricos son las grandes (pero no las únicas) cuestiones críticas en este contexto, en la geometría computacional y en la computación geométrica y científica en general. Las decisiones basadas en las preguntas como las siguientes: ¿Son dos rectas exactamente paralelas, o se cruzan en un punto lejano remoto? ó ¿Podemos estar seguros de que este segmento de línea toca la circunferencia de ese círculo?, podrían influir críticamente en el posterior control del flujo de un algoritmo geométrico, y con él su exactitud o, peor aún, su corrección.

El trabajo teórico de los algoritmos geométricos está a menudo basado en las siguientes suposiciones idealistas y, a veces, poco realistas:

1. Las entradas están en posición general.

Se utiliza para descartar configuraciones de entrada degeneradas que se desvían del caso genérico y tienden a complicar las cosas. Por ejemplo, si asumimos que no hay tres puntos en el conjunto S que coinciden en una línea, y no cuatro puntos en un círculo, entonces la teselación de Delaunay es siempre una triangulación de S . Que las configuraciones descartadas están contenidas en un subconjunto de menor dimensión del espacio de configuración R^{2n} de las coordenadas del punto parece ser una justificación suficiente para ignorarlas. No obstante, en la aplicación de integración a escala muy grande (VLSI), el conjunto de entrada puede consistir en puntos de rejilla que no estén en posición general. En consecuencia, tenemos que implementar algoritmos de tal

manera que puedan hacer frente a todas las entradas posibles. Hay diferentes maneras de proceder.

Comenzando con un algoritmo que trabaja para la entrada en la posición general, uno podría también extenderse en un algoritmo que pueda manejar la entrada degenerada. A veces esto puede hacerse de una manera orgánica, sin implementar análisis de casos complicados. Por ejemplo, el algoritmo divide y vencerás se puede extender para manejar los vértices de Voronoi de grado mayor que 3. Calcula el Diagrama de Voronoi para n puntos arbitrarios en el plano S , por lo tanto por dualización.

Otra aproximación elegante se basa en la idea de que mover los puntos en S por una cantidad infinitesimal eliminará las degeneraciones como las mencionadas anteriormente. Por lo tanto, primero se aplica una perturbación a la entrada de tal manera que se consigue la posición general. Entonces el algoritmo se ejecuta en la entrada perturbada. Una dificultad consiste en recuperar la salida verdadera en la entrada S , a partir de la salida que A ha calculado sobre la entrada perturbada.

2. La aritmética del número real es exacta y puede realizarse en tiempo constante.

La mayoría de los lenguajes de programación ofrecen valores estandarizados de punto fijo o flotante, con un número predefinido de dígitos, números enteros de tamaño fijo y enteros de longitud (potencialmente) ilimitada. Mientras que la aritmética de coma flotante está soportada por hardware y, por lo tanto, es bastante rápida pero propensa a errores de redondeo. La computación con enteros largos, por otro lado, es exacta pero lenta. Un enfoque de filtro de punto flotante intenta evitar cálculos enteros largos, utilizando aritmética de punto rápido en combinación con límites de error.

Si los objetos de entrada se especifican mediante coma flotante o coordenadas racionales, pueden representarse exactamente por los tipos de números anteriores. Sin embargo, tan pronto como un algoritmo genera nuevos objetos geométricos a partir de los dados, sus coordenadas no necesitan ser números racionales. Por ejemplo, el algoritmo de barrido del plano genera vértices de Voronoi como puntos de intersección de parábolas. Sus coordenadas envuelven raíces cuadradas. Bisecciones de objetos curvos más generales pueden ser de mayor grado algebraico.

En general, es necesario encontrar los ceros reales de las ecuaciones polinomiales con coeficientes racionales a_i .

$$f(x) = \sum_{i=0}^n a_i x^i$$

Si el grado de n es mayor que 4, los ceros de f pueden, en general, no ser expresados por radicales anidados en los números a_i .

En su lugar, los ceros reales de f deben aislarse por intervalos racionales, y luego ser numéricamente aproximados. El problema se vuelve aún más difícil si los coeficientes a_i son números algebraicos, sólo disponibles por aproximación. Ambas cuestiones, degeneración y precisión, se entrelazan. A saber, muchos algoritmos geométricos proceden evaluando predicados geométricos.

Este tema sobre los problemas de implementación que pueden originar los diferentes algoritmos disponibles para la computación de los Diagramas de Voronoi es bastante amplio y se puede ahondar mucho más en él pero no es el tema central de este trabajo. No obstante, puede ser interesante la consulta de ciertos libros que se adentran en todas estas cuestiones que hemos mencionado, como pueden ser: Seidel [8] para la primera suposición; Mehlhorn y Näher [15] y Yap [9] para la segunda.

Capítulo 6

Desarrollo de la Librería

En este capítulo nos centraremos en hacer toda la representación gráfica ya que es la manera más vistosa para entender el funcionamiento de estos diagramas y todo lo referente a nuestro trabajo. La condición de Delaunay es posible ampliarla para espacios tridimensionales usando la esfera circunscrita en vez de la circunferencia circunscrita y, aunque también es posible ampliarla para espacios con más dimensiones, no se usa en la práctica. Por otro lado, la técnica del algoritmo incremental es la que utilizaremos para realizar la Triangulación de Delaunay (Edelsbrunner y Shah [21]). Un buen software en el que podemos ver ejemplos de cálculo de la envolvente convexa, la Triangulación de Delaunay, el Diagrama de Voronoi, la intersección de medio espacio alrededor de un punto, la Triangulación de Delaunay del sitio más lejano y el Diagrama de Voronoi del sitio más lejano es Qhull, [2], donde el código fuente se ejecuta en 2-d, 3-d, 4-d y dimensiones superiores.

6.1. Análisis de requisitos

Para comenzar con esta parte informática, elaboraremos un documento de descripción del sistema que recogerá detalladamente el comportamiento del mismo, los datos que recibe, trata, almacena y devuelve, y cualquier consideración sobre su funcionamiento. Además, haremos un análisis de requisitos donde habrá que distinguir los requisitos correspondientes a cada área funcional: los **requisitos de datos** serán los datos que hay que comunicar al sistema, qué almacena y qué devuelve; los **requisitos funcionales** que van a ser las funciones que tiene que realizar el sistema. Registraremos un requisito por cada función puntual del sistema; las **restricciones semánticas** recogen el funcionamiento especial del sistema.

Deseamos implementar una librería para la gestión de Diagramas de Voronoi mediante el lenguaje de programación Java ([4]) y haciendo uso, además, como complemento, de otro lenguaje de programación como es Pyt-

hon ([5]). Con ello pretendemos facilitar información a todo aquel interesado en el estudio de estos diagramas. Nuestro objetivo será realizar una aplicación multimedia que permita trabajar con la entrada de coordenadas de puntos (tanto con eventos de ratón, como indicando las coordenadas manualmente) para dibujar Diagramas de Voronoi, su correspondiente triangulación y las circunferencias circunscritas de cada triángulo formado. Debemos ilustrar gráficamente, si la dimensión nos lo permite, estas estructuras en tiempo real o, por el contrario, si la dimensión es elevada, la indicaremos junto con las coordenadas de los puntos mediante un fichero de texto y mostraremos el resultado en otro fichero generado por nuestra aplicación, indicando los vértices y las regiones de Voronoi formadas a partir de esos puntos.

6.1.1. Requisitos específicos

Será necesario disponer de una versión actualizada (en mi caso, lo he ejecutado siempre con la versión 1.8) de Java [3]. Además, será necesario tener instalado Python [5] así como su librería SciPy [6] para poder ejecutar la aplicación de terminal para dimensiones elevadas.

6.1.2. Requisitos funcionales

Contamos con una lista de las funciones que tiene que realizar nuestro sistema.

- ➡ Crear Ventana: El sistema debe generar una ventana vacía a la que añadirle un lienzo.
- ➡ Crear un lienzo vacío: El sistema generará un lienzo con el fondo vacío cuyo tamaño será el del triángulo inicial utilizado para el algoritmo incremental.
- ➡ Nuevo lienzo vacío: Poder generar una nueva ventana con un lienzo en blanco.
- ➡ Cuadro de diálogo para abrir un fichero: Mediante un cuadro de diálogo permitir abrir un fichero de texto que tenga la entrada requerida.
- ➡ Dibujar Voronoi en el Lienzo: El sistema dibujará los puntos indicados en el fichero de entrada y representará el Diagrama de Voronoi correspondiente.
- ➡ Cuadro de diálogo para guardar un fichero: El sistema abrirá un cuadro de diálogo en el que el usuario indica el nombre y la ruta donde almacenar el fichero y lo guardará con el mismo formato que el fichero de entrada, donde se indicará lo que tengamos actualmente dibujado.

- ➡ Cuadro de diálogo para exportar un fichero: El sistema abrirá un cuadro de diálogo en el que el usuario indica el nombre y la ruta donde almacenar el fichero y lo guardará con el formato necesario para poder reproducir lo que haya actualmente en el lienzo.
- ➡ Cambiar herramienta: El sistema cambiará la herramienta de Dibujo del lienzo por la nueva, que será: dibujar triangulación de Delaunay, dibujar circunferencias circunscritas o pintar el camino más corto entre dos puntos elegidos de manera aleatoria.
- ➡ Establecer escala: Aplicar escalado al diseño, aumentando o disminuyendo la escala y permitiendo seguir dibujando en él.
- ➡ Introducir coordenadas: Existirán dos cuadros para indicar las coordenadas de un punto y que éste se dibuje en el lugar indicado.

6.1.3. Requisitos de datos

A continuación detallamos una lista numerada de los datos que hay que comunicar al sistema, qué almacena y qué devuelve.

- ➡ Entrada: Dimensión.
- ➡ Entrada: Número de puntos sobre los que queremos que se construya el Diagrama de Voronoi.
- ➡ Entrada: Coordenadas de dichos puntos.
- ➡ Manejo: Mostrar los puntos indicados, dibujando su Diagrama de Voronoi, dando la oportunidad a añadir otros.
- ➡ Manejo: Abrir un archivo de texto en el que se indique como entrada la dimensión, el número de vértices y las coordenadas de cada punto.
- ➡ Manejo: Guardar la entrada de puntos realizada en un fichero de texto con el mismo formato que el de entrada (dimensión, número de puntos y coordenadas de los puntos).
- ➡ Manejo: Exportar en un documento la información necesaria para que cualquier persona pueda dibujar el Diagrama de Voronoi representado. Esto es, las coordenadas de los vértices de Voronoi, identificados con un índice, para los puntos introducidos y mostrar una matriz que indicará si están conectados dichos vértices (0 si no están conectados y 1 si lo están).
- ➡ Manejo: Mostrar en todo momento las coordenadas en las que nos encontramos.

- ➡ Manejo: Permitir acercar o alejar lo representado para verlo con claridad.
- ➡ Manejo: Opción de empezar de nuevo sin tener que reiniciar la aplicación diseñada.
- ➡ Salida: Representación gráfica del Diagrama de Voronoi, la Triangulación de Delaunay y sus circunferencias circunscritas.
- ➡ Salida: Establecer el camino más corto entre dos puntos elegidos de manera aleatoria, pasando por las aristas del Diagrama de Voronoi, a las que se accederá trazando líneas perpendiculares hacia ellas y alejándose de los puntos únicos de cada región (los que forman la Triangulación de Delaunay).
- ➡ Salida: Sendos ficheros de texto para utilizar en la aplicación y para que cualquier otra persona pueda elaborar el Diagrama de Voronoi representado.

6.1.4. Restricciones semánticas

- ➡ Para el diseño de nuestra solución además se nos exige elaborar la aplicación utilizando el lenguaje de programación Java, permitiéndonos utilizar todas las librerías disponibles.
- ➡ Utilizar el algoritmo incremental para generar la Triangulación de Delaunay.
- ➡ Además de realizar la interacción de ficheros con los botones necesarios, establecer un comando directo a través del teclado.
- ➡ El Diagrama de Voronoi de los puntos se dibujará siempre por defecto y en todo momento, representando todas las demás opciones (Triangulación de Delaunay, circunferencias y camino más corto) con dicho diagrama representado.
- ➡ Las regiones de Voronoi se diferenciarán representándolas de distintos colores establecidos de manera aleatoria.

6.2. Diseño de la solución

En primer lugar, se ha desarrollado una librería en Java cuya utilización, por un lado, nos va a permitir, al pasarle un fichero de texto con extensión .txt con la entrada correspondiente, la salida deseada también en un archivo "salida.txt" lo que haremos para la dimensión dos en el lenguaje de programación Java y para dimensiones superiores utilizando un fichero elaborado en el lenguaje de programación Python (incluido en nuestra aplicación de

Java). Es decir, partiremos de un problema en el que los datos de entrada serán: la dimensión, el número de puntos y las coordenadas de cada punto. Las salidas que obtendremos serán las coordenadas de los vértices de Voronoi y el número de polígonos creados (regiones de Voronoi). Ésta será la primera aplicación desarrollada. Por otro lado, dicha librería ha sido incluida en otro proyecto de Java donde se desarrolla, con la utilización de un JFrame y un JPanel, la visualización del Diagrama de Voronoi y una aplicación que hemos considerado que puede hacer que, de manera visual, se entienda perfectamente el cálculo de los diagramas y el gran uso que pueden tener.

Decidimos que ésta era la manera idónea de programar todo. Es decir, teniendo, por un lado, la librería necesaria para utilizar Voronoi, no sólo para las aplicaciones que hemos implementado en 2D, sino para usos futuros o para cualquier otro usuario interesado en utilizarla. Por otro lado, la aplicación para interactuar con la terminal. Y, por último, incluyendo la librería en un JFrame y con el uso de un JPanel incluidos en java, poder elaborar la interfaz gráfica.

En la librería realizada, se pueden distinguir una serie de clases, que serán la base de todo lo que hagamos. Además, esta librería podría ser utilizada para posibles aplicaciones futuras. Las clases son las siguientes:

- **Punto:** En esta clase, consideramos los puntos en el espacio Euclídeo y los implementamos como *double* (números reales). Incluiremos operaciones geométricas básicas que se pueden realizar con los puntos como son: obtener sus coordenadas y su dimensión, crear un nuevo punto a partir de otro añadiéndole coordenadas, producto escalar, norma, suma, resta, cálculo del ángulo y la bisectriz. Por otro lado representaremos las matrices y las envolventes convexas como un *array* (estructura de datos que nos permite almacenar una gran cantidad de datos de un mismo tipo) de puntos. Para las matrices calculamos tanto el determinante como el producto matricial. Para los simplices o envolventes convexas vemos qué relación mantienen con los puntos, es decir, si éstos están dentro, fuera o incluidos. Del mismo modo veremos qué relación tienen con las circunferencias circunscritas y calcularemos el circuncentro.
- **Conjunto:** Hereda de una clase abstracta, *AbstracSet*, que proporciona una implementación esquelética de la interfaz *Set* para minimizar el esfuerzo requerido para desarrollarla. Además, haremos uso de la clase *Collections*, también incluida en el paquete de Java, que consiste enteramente en métodos estáticos que se utilizan para operar en colecciones. Operaciones comunes como ordenar una lista o encontrar un elemento de una lista se pueden hacer fácilmente usando dicha clase. Con su uso, la clase Conjunto nos permite: crear un conjunto vacío de

tamaño 3; crear un conjunto con la capacidad que le digamos; crear un conjunto que contenga los elementos de la colección, eliminando los duplicados; devolver el elemento del índice que le especifiquemos; comprobar si algún elemento de la colección también pertenece a este conjunto.

- **Triángulo:** Hereda de la clase Conjunto. Un triángulo va a ser un conjunto no modificable de exactamente 3 puntos. Nos va a permitir calcular cuáles son los triángulos vecinos; obtener la cara opuesta a un vértice que le indiquemos y obtener el circuncentro calculado en la clase Punto.
- **Grafo:** En esta clase los nodos van a tener un tipo genérico G. Añadimos y eliminamos tanto nodos como aristas en el grafo no dirigido. Además, podemos obtener el conjunto de nodos de dicho grafo. Hemos empleado, para su implementación, la clase *Collections* explicada anteriormente para definir un conjunto de nodos; y las clases *HashMap*, que es una clase rápida y fácil de usar que representa una tabla hash (estructura de datos que asocia las claves con valores) y *Maps* cuya interfaz asigna las claves a los valores. Las claves son únicas y por lo tanto, no se permiten claves duplicadas. Estas últimas serán las que nos permitan crear la variable *vecinos* que nos ayudará a saber qué nodos son los más cercanos a uno dado.
- **Triangulación2D:** Realizamos la Triangulación de Delaunay en dos dimensiones mediante el algoritmo incremental 5.2. Ésta no es la manera más rápida de construirlo, pero es una manera razonable de construir una Triangulación de Delaunay incrementalmente y permite una buena visualización interactiva. Consideramos la triangulación como un conjunto de triángulos no modificable como conjunto. La única manera de cambiarla es añadiéndole sitios mediante uno de los métodos implementados. También podremos obtener el triángulo opuesto del vértice especificado; recorrer los triángulos alrededor de uno dado y podremos comprobar a qué triángulo pertenece un punto.
- **Dijkstra:** Clase específica para implementar el **Algoritmo de Dijkstra** que calcula el camino óptimal desde un punto hasta otro en un grafo, teniendo en cuenta los pesos de cada arista. En nuestro caso, los pesos serán la distancia euclídea entre ellos. La idea es buscar el punto más cercano posible a nuestro punto de origen, repitiendo el proceso hasta llegar a nuestro punto de destino, moviéndonos por los vértices y las aristas de Voronoi. Utilizaremos para implementarlo una cola de prioridad incluida en la clase *PriorityQueue* de Java.

En otro proyecto de Java hemos realizado, como ya hemos comentado,

el diseño de la visualización del Diagrama de Voronoi, incluyendo la librería anteriormente explicada. Además, hemos empleado el algoritmo de Dijkstra que nos calcula el camino más corto posible entre dos puntos elegidos al azar. Esta aplicación puede ser útil, por ejemplo, en Robótica. Si tenemos una escena con obstáculos (atractores) en las que un robot deba moverse, una forma de evitar las colisiones o el acercamiento sería diseñar la trayectoria del robot sobre las líneas del Diagrama de Voronoi, con lo que se movería siempre a medio camino entre dos de los atractores, intentando acercarse lo menos posible a ellos. Esto es precisamente una de las aplicaciones que podremos visualizar de manera gráfica. En este proyecto hemos incluido lo siguiente:

- **Lienzo:** Se trata de un JPanel en el que están implementados todos los métodos para poder dibujar tanto el Diagrama de Voronoi, como su respectiva Triangulación de Delaunay, además de las circunferencias circunscritas. Para ello, tendremos unos métodos básicos de dibujado con los que podremos dibujar puntos, aristas y los respectivos polígonos formados, añadiéndoles un color aleatorio que nos va a permitir distinguir con claridad las regiones de Voronoi. Por otro lado disponemos de otros métodos específicos de dibujado: para dibujar la Triangulación de Delaunay, recorremos todos los triángulos y vamos uniendo los diferentes vértices; para dibujar las circunferencias circunscritas recorremos los diferentes triángulos de la triangulación y calculamos sus circuncentros; en el dibujo de Voronoi debemos tener en cuenta que los circuncentros de las circunferencias serán nuestros vértices de Voronoi y calculamos la distancia euclídea entre vértices para saber cuáles de ellos están conectados. Además, tendremos en cuenta que en cada región de Voronoi sólo podrá haber un punto (vértice de los triángulos de la triangulación). Además, tenemos unos métodos específicos implementados para obtener el camino más corto posible entre dos puntos elegidos al azar utilizando el algoritmo de Dijkstra como son: la obtención del punto más cercano a otro punto; cálculo del punto más cercano a un segmento; añadir una arista que conecte los puntos de corte (que se encuentran en una misma arista de Voronoi) entre los dos puntos elegidos, en caso de que éstos se encuentren en regiones colindantes; añadir todas las perpendiculares al polígono en el que se encuentren dichos puntos, para después elegir la más adecuada para establecer el camino más corto. Por último, implementamos los métodos que nos van a permitir trabajar con ficheros (guardar, abrir, exportar) y un método que, según en el caso en el que nos encontremos, hará la función que le corresponda. Además aquí incluimos la gestión de eventos de ratón vinculados al proceso de dibujado.
- **VentanaPrincipal:** Se trata de un JFrame en el que incluimos el

JPanel Lienzo. Por tanto, aquí simplemente establecemos qué deben hacer los botones añadidos a nuestra interfaz.

A continuación detallamos las opciones que existen en nuestra interfaz y en la sección siguiente 6.4 veremos cómo se utilizan:

▪ **Modos:**

- Voronoi: Ésta será la opción por defecto. Cada vez que hagamos click con el ratón, se pintarán unos puntos (tantos como queramos) y su respectivo Diagrama de Voronoi. Estos puntos serán los vértices de los triángulos que construyen la Triangulación de Delaunay y sólo habrá uno de ellos en cada región de Voronoi.
- Dijkstra: Cuando este botón está seleccionado, automáticamente, el próximo click hecho con el ratón ya no será para construir el Diagrama de Voronoi, sino que se podrán elegir dos puntos aleatorios y se calculará el camino óptimo más corto entre ellos, moviéndose por las aristas y vértices de Voronoi. Si volvemos a hacer click con el ratón se eliminará el camino y podremos volver a seleccionar dos puntos para volver a calcular este algoritmo.

▪ **Más información:**

- Delaunay: Si este cuadro está seleccionado, podremos ver la Triangulación de Delaunay del respectivo Diagrama de Voronoi.
- Circunferencias: Al igual que el botón de Delaunay, si este botón está seleccionado, se mostrarán las circunferencias circunscritas correspondientes a la triangulación y al diagrama.

▪ **Interacción con ficheros:**

- Abrir (Alt+A): Permite seleccionar un fichero de texto, en el que la entrada será la dimensión, el número de vértices y sus correspondientes coordenadas. La interfaz dibujará esos puntos por defecto, permitiendo incluir algunos más si así lo deseamos.
- Guardar (Alt+G): Permite guardar en un fichero de texto los mismos datos de entrada (dimensión, número de vértices y coordenadas de dichos vértices) para, en caso de usar posteriormente el botón de abrir para seleccionar el archivo guardado, pudiéramos volver a mostrarlo.
- Exportar (Alt+E): Permite guardar un archivo con la misma extensión que los anteriores (.txt) que incluirá la información necesaria para dibujar el Diagrama de Voronoi que acabamos de mostrar en pantalla. Dicha información será una lista de los vértices

de Voronoi generados con un índice para identificarlos y una matriz de aristas que indicará si dichos vértices están relacionados o no (mostrando 1 ó 0 respectivamente).

- **Otras opciones:**

- Limpiar (Alt+L): Borra todo lo realizado, dejando el fondo blanco, dando la oportunidad de volver a dibujar de nuevo.
- Ver: Nos permite ocultar la barra de herramientas si así lo deseamos.
- Zoom: Habrá en nuestra interfaz una barra de desplazamiento que podemos deslizar para acercar o alejar el Diagrama de Voronoi que hayamos dibujado.
- Añadir: Podremos añadir puntos, además de haciendo click con el ratón, indicando las coordenadas X e Y y pulsando el botón, de manera automática, se nos dibujará el punto que hayamos introducido.
- Además, en la parte de abajo de nuestra interfaz se mostrarán las coordenadas de los puntos por los que vayamos pasando con el ratón.

Estas dos aplicaciones, tanto la de entrada y salida de ficheros como la que utiliza una interfaz gráfica, han sido incluidas en un archivo JAR, que permite ejecutar aplicaciones escritas en el lenguaje Java y Python. Con Java instalado en nuestro ordenador bastará con abrir la terminal y escribir por línea de comandos las siguientes órdenes para ejecutar los .jar correspondientes:

- Aplicación Voronoi + Dijkstra: `java -jar VoronoiDijkstraApp.pack.jar`
- Aplicación entrada y salida: `java -jar VoronoiTerminal.pack.jar entrada.txt`

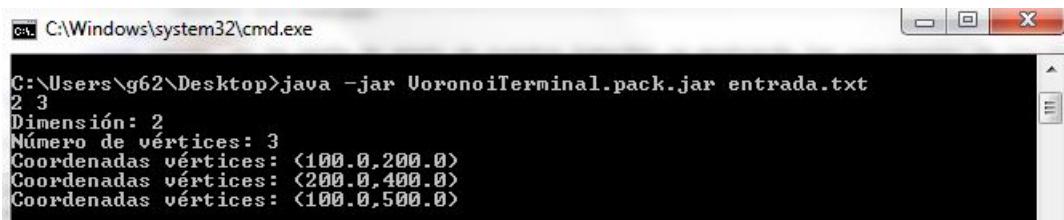
A continuación mostraremos ejemplos de uso de estas aplicaciones donde podremos ver su funcionamiento de manera gráfica.

6.3. Manual de usuario

Aunque el diseño gráfico es bastante intuitivo, vamos a ver con ejemplos cómo utilizar las dos aplicaciones diseñadas.

- Empezaremos por la que interactúa directamente con la terminal, sin hacer uso de ninguna interfaz gráfica.

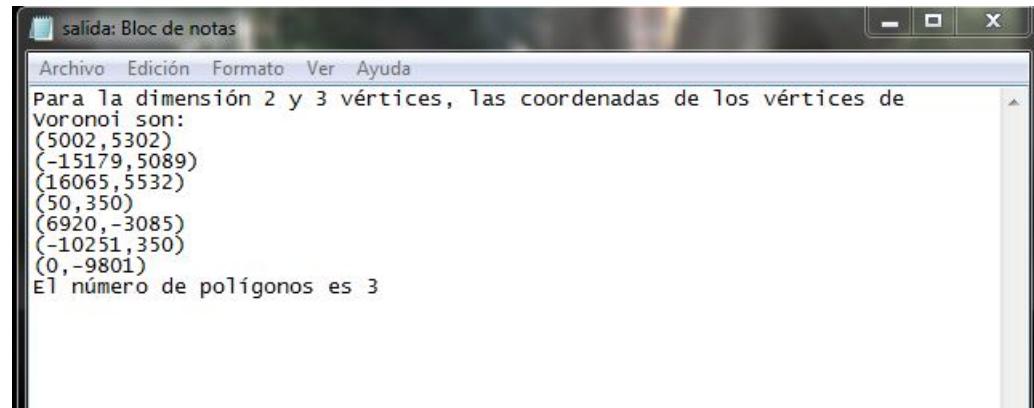
1. Tras abrir la terminal de Windows, nos ubicamos en el directorio donde se encuentra nuestra aplicación junto con el fichero de entrada que contendrá los datos necesarios para ejecutarla.
2. Una vez hecho esto, introducimos el comando `java -jar VoronoiTerminal.pack.jar entrada.txt` siempre y cuando nuestro fichero de entrada tenga este nombre intuitivo.
3. Se nos mostrará en terminal información sobre el fichero de texto leído:



```
C:\Windows\system32\cmd.exe
C:\Users\g62\Desktop>java -jar VoronoiTerminal.pack.jar entrada.txt
2 3
Dimensión: 2
Número de vértices: 3
Coordenadas vértices: <100.0,200.0>
Coordenadas vértices: <200.0,400.0>
Coordenadas vértices: <100.0,500.0>
```

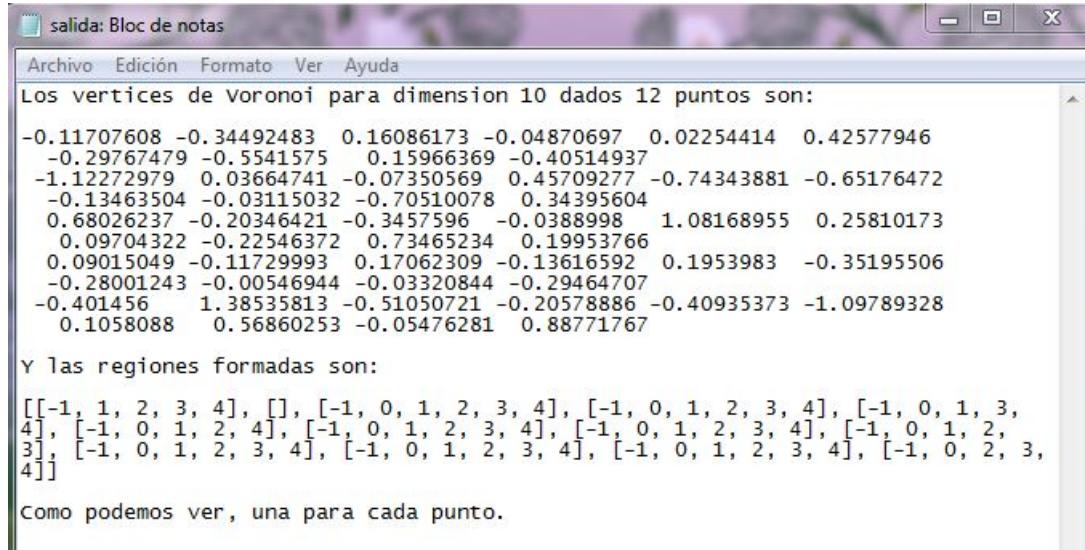
Figura 6.1: Información mostrada en el terminal del fichero de entrada.

4. A continuación, se generará un fichero de texto con extensión .txt ubicado en el mismo directorio donde teníamos el .jar y el fichero de entrada, que mostrará lo siguiente:



```
salida: Bloc de notas
Archivo Edición Formato Ver Ayuda
Para la dimensión 2 y 3 vértices, las coordenadas de los vértices de
Voronoi son:
(5002,5302)
(-15179,5089)
(16065,5532)
(50,350)
(6920,-3085)
(-10251,350)
(0,-9801)
El número de polígonos es 3
```

Figura 6.2: Información mostrada en el fichero de salida para dos dimensiones.



```

salida: Bloc de notas
Archivo Edición Formato Ver Ayuda
Los vertices de voronoi para dimension 10 dados 12 puntos son:
-0.11707608 -0.34492483 0.16086173 -0.04870697 0.02254414 0.42577946
-0.29767479 -0.5541575 0.15966369 -0.40514937
-1.12272979 0.03664741 -0.07350569 0.45709277 -0.74343881 -0.65176472
-0.13463504 -0.03115032 -0.70510078 0.34395604
0.68237 -0.20346421 -0.3457596 -0.0388998 1.08168955 0.25810173
0.09704322 -0.22546372 0.73465234 0.19953766
0.09015049 -0.11729993 0.17062309 -0.13616592 0.1953983 -0.35195506
-0.28001243 -0.00546944 -0.03320844 -0.29464707
-0.401456 1.38535813 -0.51050721 -0.20578886 -0.40935373 -1.09789328
0.1058088 0.56860253 -0.05476281 0.88771767

Y las regiones formadas son:
[[[-1, 1, 2, 3, 4], [], [-1, 0, 1, 2, 3, 4], [-1, 0, 1, 2, 3, 4], [-1, 0, 1, 3, 4], [-1, 0, 1, 2, 4], [-1, 0, 1, 2, 3, 4], [-1, 0, 1, 2, 3, 4], [-1, 0, 1, 2, 3, 4], [-1, 0, 1, 2, 3, 4], [-1, 0, 1, 2, 3, 4], [-1, 0, 1, 2, 3, 4]]]

Como podemos ver, una para cada punto.

```

Figura 6.3: Información mostrada en el fichero de salida para diez dimensiones.

- A continuación mostramos el funcionamiento en detalle de la aplicación más completa con interfaz gráfica.

1. Mostramos la interfaz nada más abrirla:

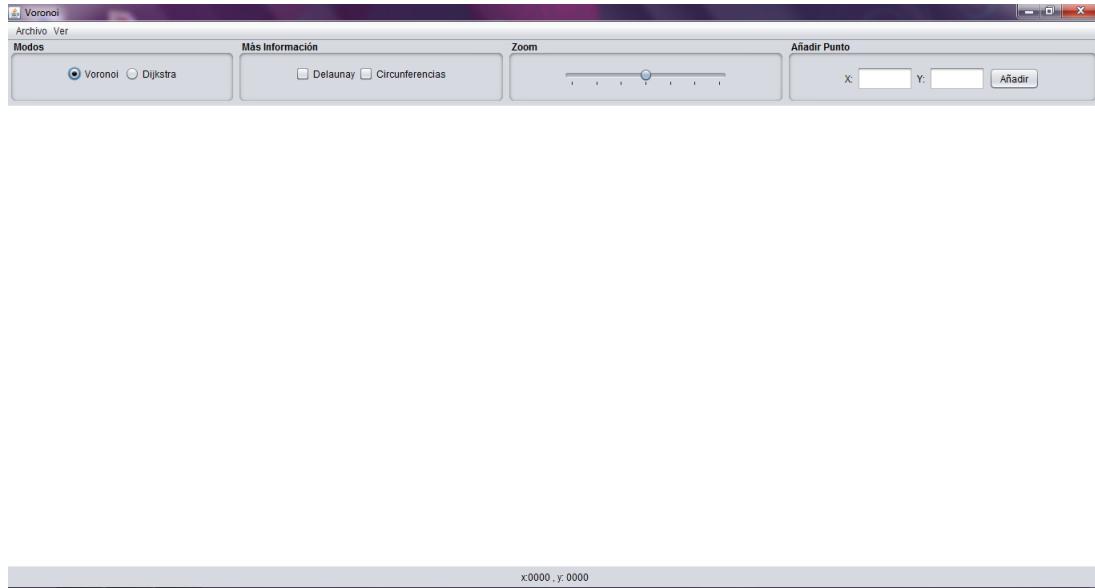


Figura 6.4: Pantalla inicial de la aplicación.

2. En primer lugar, podemos dibujar puntos con el ratón. Por defecto, se

nos dibujará el Diagrama de Voronoi ya que éste es el modo inicial, donde cada punto dibujado con el ratón sólo pertenecerá a una región de Voronoi. A continuación se muestra el resultado de 10 puntos dibujados.

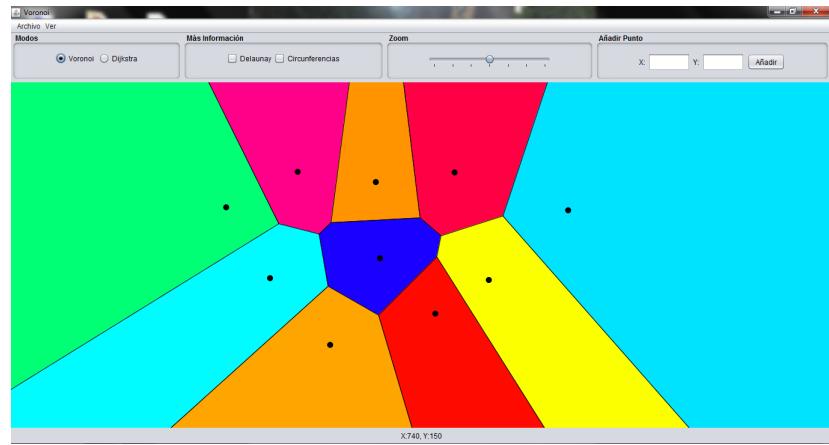


Figura 6.5: Diagrama de Voronoi de 10 puntos aleatorios.

3. Si cambiamos el modo a Dijkstra, podremos elegir dos puntos al azar y se nos trazará el camino óptimo y más corto posible entre ellos teniendo en cuenta dónde están colocados y evitando acercarse demasiado a los puntos negros. Estos puntos, en nuestra aplicación, serán nuestros atractores por lo que el recorrido de un robot para llegar desde un punto inicial al destino que le indiquemos, deberá alejarse lo máximo posible de ellos. Se calculará trazando perpendiculares desde los puntos seleccionados hasta las aristas de Voronoi para luego moverse por ellas.

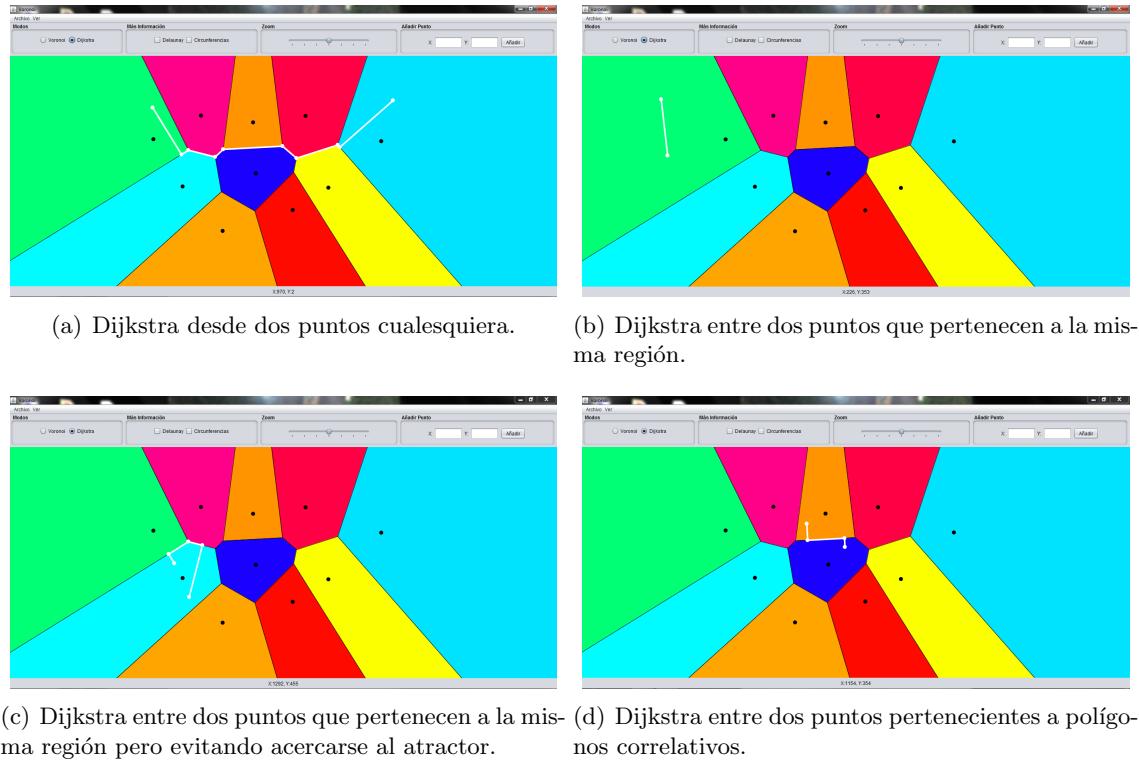


Figura 6.6: Pruebas del algoritmo de Dijkstra.

4. Además, ofrecemos más información: Si seleccionamos “Delaunay”, se dibujará la correspondiente Triangulación de Delaunay.

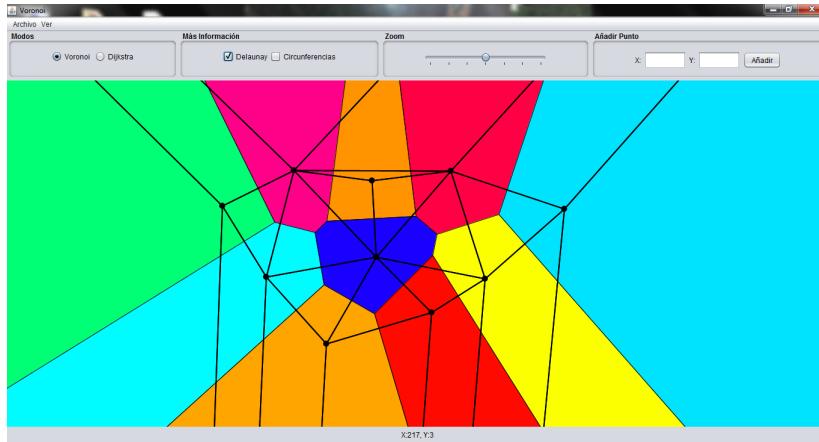


Figura 6.7: Triangulación de Delaunay del Diagrama de Voronoi dibujado anteriormente con 10 puntos aleatorios.

5. Otra información que podemos mostrar es, al seleccionar “Circunferencias”, se dibujará la correspondiente Triangulación de Delaunay.

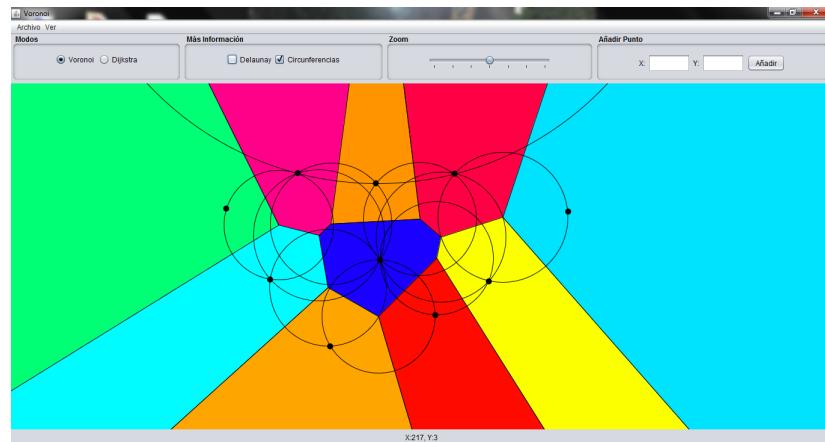


Figura 6.8: Circunferencias circunscritas de la Triangulación de Delaunay.

6. Podemos además darle y quitarle zoom a lo dibujado moviendo la barra de desplazamiento, y se nos permitirá seguir dibujando siempre que lo deseemos.

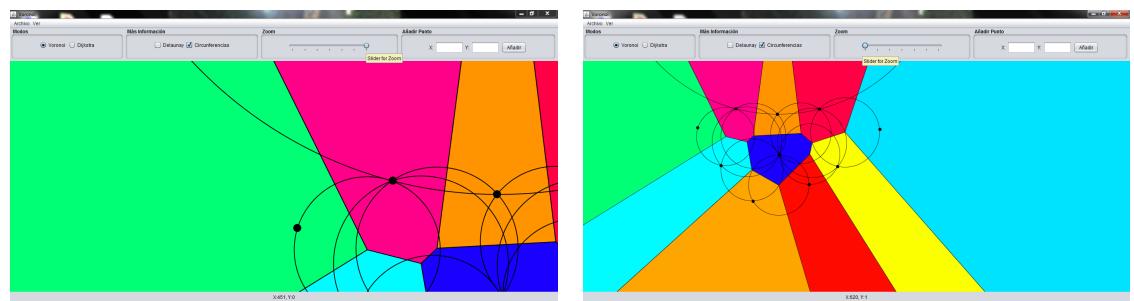


Figura 6.9: Hacemos zoom en la barra de desplazamiento de la parte superior de nuestra aplicación.

7. Podemos añadir puntos indicando sus coordenadas X e Y. Por ejemplo, añadimos a continuación el punto (100, 200).

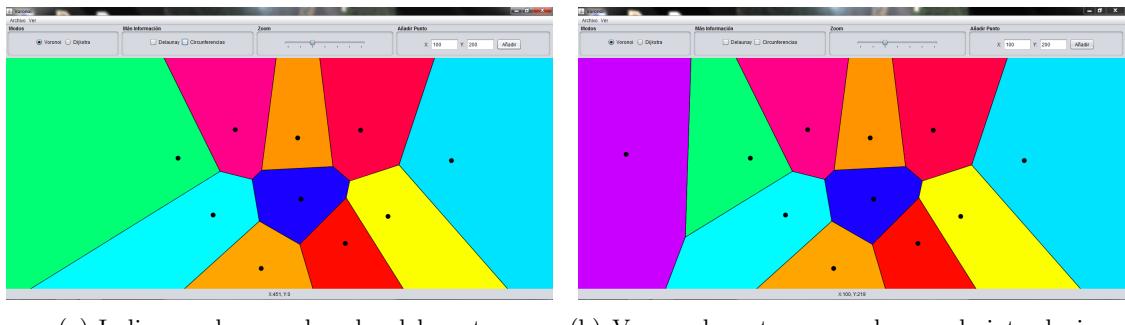


Figura 6.10: Añadimos un punto indicando sus coordenadas en la parte superior derecha de la aplicación.

8. Podemos añadir o quitar la barra de herramientas en la opción “Ver” del menú arriba a la izquierda:

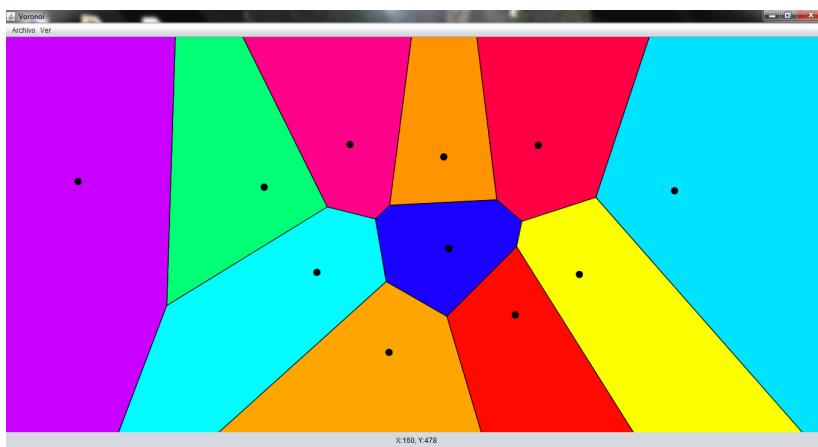


Figura 6.11: Aplicación sin la barra de herramientas superior.

9. Para trabajar con archivos tenemos diferentes opciones pulsando en “Archivo” del menú superior izquierdo de nuestra aplicación:
 - a) Guardar archivo: guardaremos el dibujo actual en un archivo con extensión *.txt*.

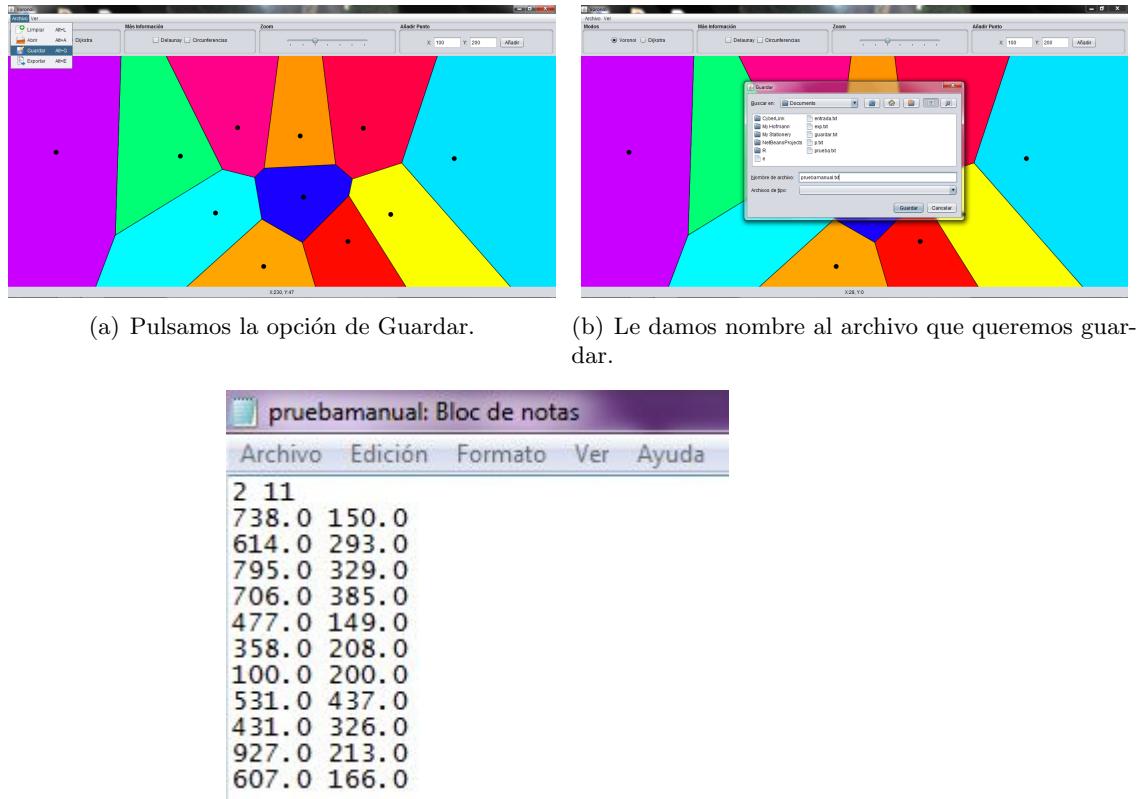
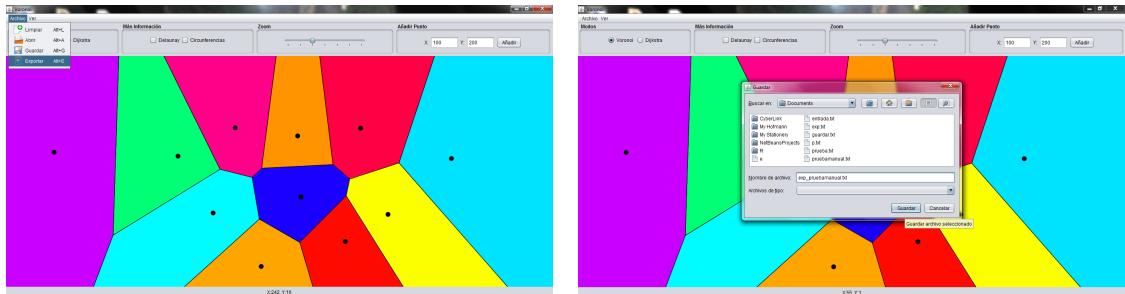


Figura 6.12: Guardar archivo.

- b) Exportar archivo: guardaremos el dibujo actual en un archivo con extensión *.txt* con los datos necesarios para que cualquier persona pueda representar nuestro Diagrama de Voronoi:



(a) Pulsamos la opción de Exportar.

(b) Le damos nombre al archivo que queremos exportar.

```

exp_pruebamainual: Bloc de notas
Archivo Edición Formato Ver Ayuda
Teniendo en cuenta que el triángulo inicial en el que dibujamos es:
Triángulo[(-10000.0,-10000.0), (10000.0,-10000.0), (0.0,10000.0)]

La lista de vértices de Voronoi con su índice:
0 - (715,255)
1 - (708,290)
2 - (240,-177)
3 - (643,-9237)
4 - (2076,5319)
5 - (-935,-8874)
6 - (223,372)
7 - (84,738)
8 - (818,222)
9 - (-14895,4947)
10 - (3195,-6908)
11 - (532,233)
12 - (-2300,5076)
13 - (611,387)
14 - (527,339)
15 - (18660,6830)
16 - (3959,5457)
17 - (680,225)
18 - (5532,5586)
19 - (0,-9840)
20 - (512,252)
21 - (445,235)
22 - (609,-358)

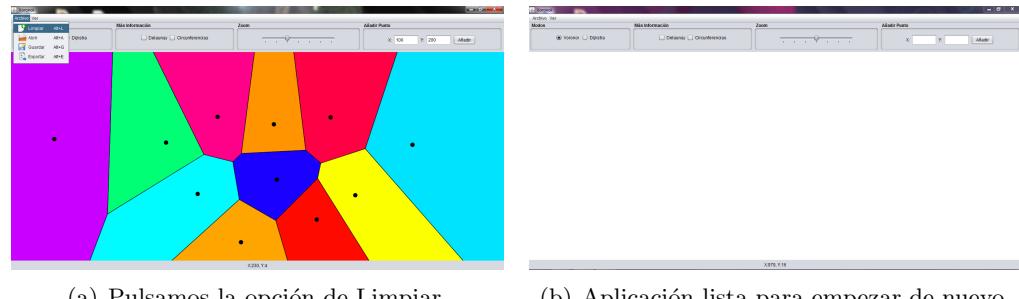
Y la matriz de aristas es:
* 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
2 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
3 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0
4 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0
5 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
6 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
7 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
8 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
9 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
10 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
11 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1
12 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
13 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
14 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0
15 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
16 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0
17 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1
18 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0
19 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
20 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0
21 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0

```

(c) El archivo exportado contiene esta información: lista de vértices de Voronoi y si están conectados o no (indicándolo con 0 y 1 respectivamente).

Figura 6.13: Exportar archivo.

- c) Limpiar: esta opción nos pone la pantalla en blanco, tal y como la encontramos al principio del todo.



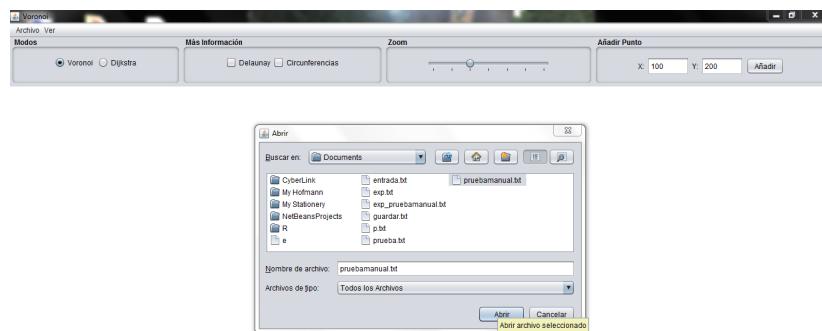
(a) Pulsamos la opción de Limpiar. (b) Aplicación lista para empezar de nuevo.

Figura 6.14: Exportar archivo.

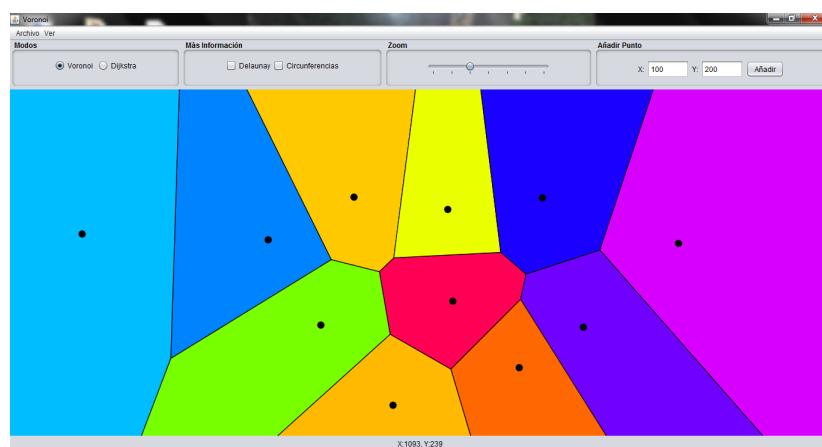
- d) Abrir archivo: podemos abrir un archivo que tendrá los datos necesarios para poder representarlo en nuestra aplicación, es decir: dimensión, número de vértices y coordenadas de los vértices. Lo probamos para el archivo anteriormente guardado, por lo que nos saldrá el mismo dibujo con colores diferentes ya que éstos se seleccionan de manera aleatoria:



(a) Pulsamos la opción de Abrir.



(b) Elegimos el archivo que queremos abrir.



(c) Se muestra el archivo seleccionado.

Figura 6.15: Abrir archivo.

10. Además, podemos mostrar varia información a la vez.

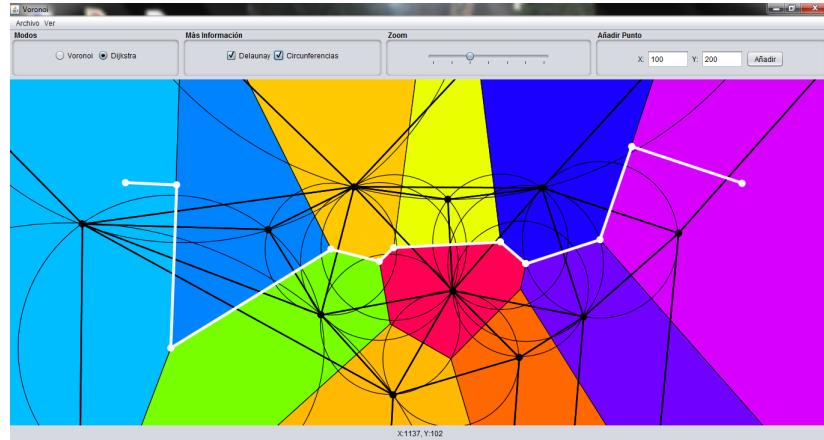


Figura 6.16: Mostramos información completa.

6.4. Código desarrollado

El código elaborado en dos proyectos de Java, uno para la aplicación desde terminal (en la carpeta *VoronoiTerminal*) y otro para la interfaz gráfica (dentro de la carpeta *VoronoiDijkstraApp*) así como la librería implementada (*Voronoi.MOB.Biblioteca*), puede encontrarse en el siguiente enlace:

<https://github.com/MariaOB/TFG>

Dentro de la carpeta *dist* de los directorios donde se encuentran ambas aplicaciones, podemos obtener los .jar que comentábamos anteriormente donde se pueden ejecutar las aplicaciones teniendo Java instalado en nuestra computadora.

Capítulo 7

Conclusiones y Trabajos Futuros

El objetivo general que ha marcado la realización de este Trabajo Fin de Grado ha sido doble: por un lado, el estudio de los Diagramas de Voronoi de diversas tipologías en espacios euclídeos, propiedades y algoritmos de obtención desde el punto de vista matemático; desde el punto de vista de la Ingeniería Informática, la realización de una librería para el cálculo y gestión de los Diagramas de Voronoi en espacios euclídeos de cualquier dimensión, incluyendo la entrada de puntos, la realización de la partición y otras funciones relevantes en diversas aplicaciones prácticas.

A lo largo del documento hemos podido comprobar que se han cumplido estos objetivos. Los primeros objetivos en completarse fueron los matemáticos, ya que sin la base matemática no hubiese podido realizar la aplicación. Aunque, conforme iba estudiando todos los conceptos matemáticos, fui pensando en cómo implementarlo todo utilizando el lenguaje Java con el que ya había trabajado durante la carrera aunque, quizás, no a tan gran escala como lo hemos hecho en este proyecto.

De forma más específica, estos objetivos se han alcanzado mediante la aportación de los siguientes resultados:

- En los Capítulos 3 y 4 se han introducido definiciones formales de los conceptos “Diagramas de Voronoi” y “Triangulación de Delaunay”, así como otras definiciones relevantes para entender estas ideas. Hemos estudiado algunas caracterizaciones, además de habernos concienciado del gran uso que se le puede dar a este tema y las numerosas aplicaciones que tienen en nuestro día a día.
- En el Capítulo 5 se ha llevado a cabo una revisión de las diferentes técnicas y algoritmos relacionados con los Diagramas de Voronoi y su dual. Aquí hemos podido comprobar que hay algoritmos que,

aunque tengan un orden de complejidad óptimo, la dificultad de implementación es muy elevada especialmente si los tenemos en cuenta en dimensiones elevadas.

- En el Capítulo 6 hemos explicado la implementación de una librería específica para los Diagramas de Voronoi y hemos puesto en práctica los conocimientos adquiridos para mostrarlos de manera gráfica.

Desde mi punto de vista, el trabajo ha resultado interesante, ya que he conseguido unificar en un mismo proyecto matemáticas e informática y poner en práctica todo el estudio realizado sobre los Diagramas de Voronoi. Con respecto a la parte matemática, el aprender un tema como los Diagramas de Voronoi y realizar un trabajo extenso de manera casi autosuficiente (ya que he contado siempre con la ayuda de mi tutor) me ha hecho ampliar mi conocimiento, capacidad de comprensión y estudio, además de enseñarme a afrontar un trabajo más amplio desde el principio y revisando la parte de documentación hasta el último detalle. El trabajar con temas con los que no he tratado a lo largo de la carrera y el contar con gran ambición para afrontar y aprender todo lo nuevo que se me presenta, me ha servido como motivación para el desarrollo de este trabajo. En cuanto a la parte informática, el seguir aprendiendo sobre los lenguajes de Java y Python me ha parecido de gran utilidad ya que es un lenguaje que se está usando en la actualidad del mundo empresarial.

Además, como consecuencia de la investigación realizada se nos han planteado una serie de líneas de investigación que consideramos interesantes para afrontar en el futuro:

- Ampliación del estudio de los Diagramas de Voronoi.
- Estudiar los Diagramas de Voronoi generados a partir de los centros de los círculos ([23], [12]).
- Extender la librería ya diseñada poder desarrollar distintas aplicaciones.

Bibliografía

- [1] Franz Aurenhammer, Rolf Klein y Der-Tsai Lee, *Voronoi Diagrams and delaunay Triangulations*, World Scientific, 2013.
- [2] Qhull, *Voronoi Diagram*. <http://www.qhull.org/html/qvoronoi.htm>,
<http://www.qhull.org/html/qdelaun.htm>
- [3] Java™ Platform, Standard Edition 7 API Specification <https://docs.oracle.com/javase/7/docs/api/>.
- [4] Programming Tutorials and Source Code Examples. <http://www.java2s.com/>
- [5] Python™. The Python Programming Language. <https://www.python.org/>
- [6] SciPy: Fundamental Python library for scientific computing. Last updated on Mar 09, 2017. <https://scipy.org/scipylib/index.html>
- [7] Binay Bhattacharya. *Notes from the book by de Berg, Van Krevald, Overmars, and Schwarzkpf. Fortune's Algorithm*, 2011. <http://www.cs.sfu.ca/~binay/813.2011/Fortune.pdf>
- [8] R. Seidel. *Small-dimensional lineal programming and convex hulls made easy*. Discrete & Computational Geometry 6, 1991.
- [9] C.K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, 2000.
- [10] Claudia Esteves Jaramillo, Johan Van Horebeek, Alonso Ramírez Manzanares. *Curso de Análisis de Algoritmos*. Geometría Computacional, Diagramas de Voronoi. 2013. http://www.cimat.mx/~alram/analisis_algo/07_DiagramasVoronoi.pdf
- [11] Geometría Computacional. Ingeniería Informática e Ingeniería Técnica de Gestión. Diagramas de Voronoi, 2000-2001. <http://asignatura.us.es/fgcitig/contenidos/gctem3ma.htm>

- [12] Peter Gärdenfors. *Conceptual Spaces. The Geometry of Thought*. MIT Press, 2004.
- [13] Bowyer-Watson algorithm, última edición en Marzo de 2017 https://en.wikipedia.org/wiki/Bowyer%20Watson_algorithm
- [14] D.Schmitt and J.-C. Spehner. *Angular properties of Delaunay diagrams in any dimensions*. Discrete & Computational Geometry 5, 2000, 17-36.
- [15] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
- [16] A.K. Dewdney and J.K. Vranckh. *A convex partition of R^3 with applications to Crum's problem and Knuth's post-office problem*. Utilitas Mathematica 12, 1977, 193-199.
- [17] B. Joe. Geompack. *A software package for the generation of meshes using geometric algorithms*. Advances in Engineering Software and Workstations 13, 1991, 325-331.
- [18] H. Edelsbrunner and R. Seidel. *Voronoi Diagrams and arrangements*. Discrete & Computational Geometry 1, 1986, 25-44.
- [19] F. Autenhammer and H. Imai. *Geometric relations among Voronoi diagrams*. Geometriae Dedicata 27, 1988, 65-75.
- [20] Dr. Eduardo A. Rodríguez Tello. CINVESTAV-Tamaulipas. 15 de marzo del 2013. <http://www.tamps.cinvestav.mx/~ertello/gc/sesion16.pdf>
- [21] H. Edelsbrunner and N.R. Shah. *Incremental topological flipping works for regular triangulations*. Algorithmica, 1996.
- [22] The Voronoi Web Site , última modificación en Junio de 2008. <http://www.voronoi.com>
- [23] François Anton, Darka Mioc, and Christopher Gold. *The Voronoi diagram of circles and its application to the visualization of the growth of particles*, 2009. http://www2.imm.dtu.dk/projects/graph/material/Anton_etal_2009.pdf