Universidad de las Ciencias Informáticas

Facultad 6

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Biblioteca de algoritmos para la obtención de particionados 2D y su problema inverso

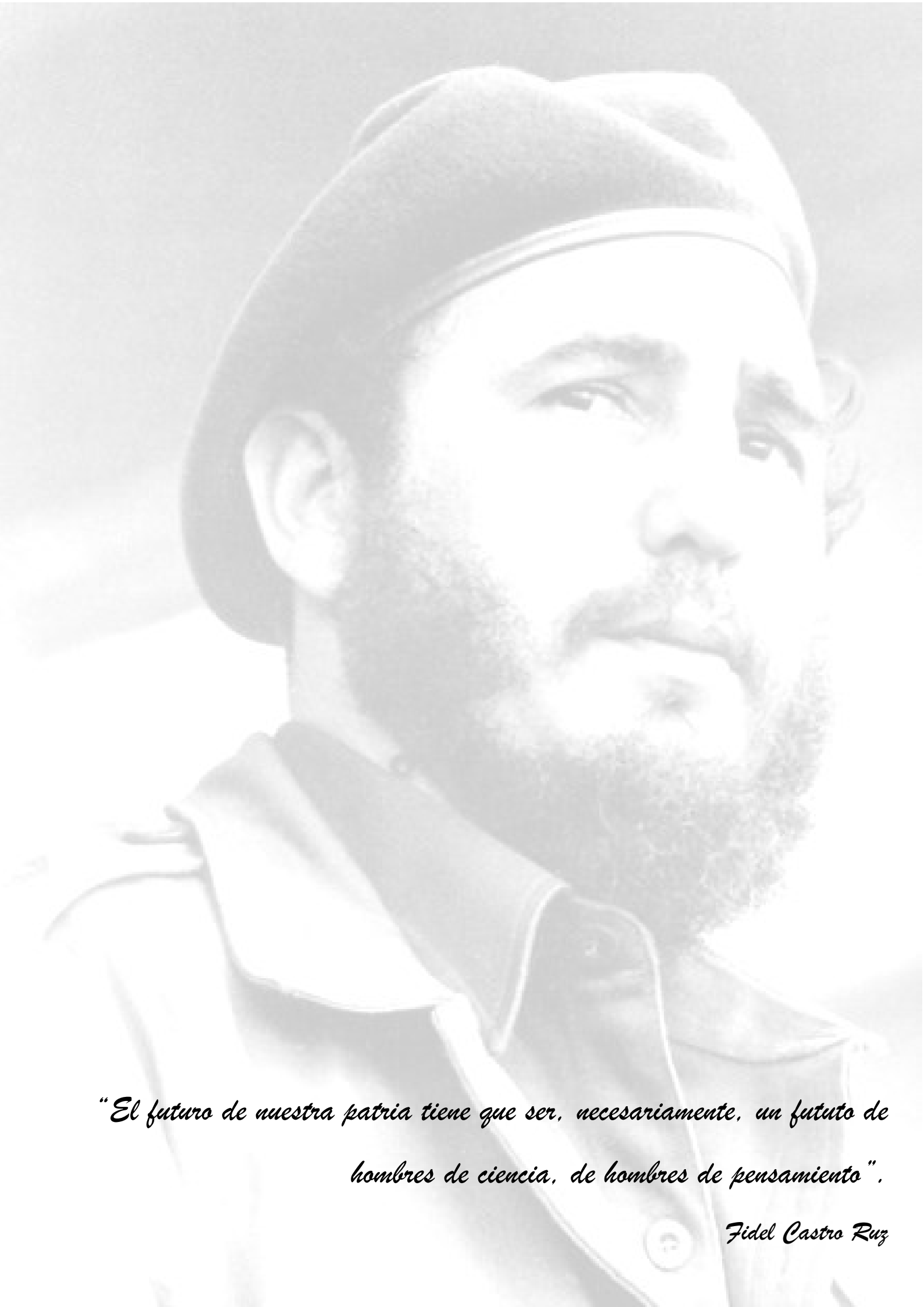
Autor: Yaisel Siverio Santa Teresa

Tutores: MsC. Romanuel Ramón Antunez

MsC. Lidisy Hernández Montero

La Habana, Julio de 2016

"Año 58 de la Revolución"



"El futuro de nuestra patria tiene que ser, necesariamente, un fututo de hombres de ciencia, de hombres de pensamiento".

Fidel Castro Ruz

Declaro ser el legítimo autor del trabajo titulado: “Biblioteca de algoritmos para la obtención de particionados 2D y su problema inverso”, y reconozco a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____

Firma del Autor

Yaisel Siverio Santa Teresa

_____

Firma del Tutor

MsC. Romanuel Ramón Antunez

_____

Firma del Tutor

MsC. Lidisy Hernández Montero

Síntesis del Tutor

Nombre y Apellidos: MsC. Romanuel Ramón Antunez

Correo electrónico: rramon@uci.cu

Categoría Docente: Asistente

Año de graduado: 2008

Profesión: Ingeniero en Ciencias Informáticas

Máster en Informática Aplicada: 2011

Síntesis del Tutor

Nombre y Apellidos: MsC. Lidisy Hernández Montero

Correo electrónico: lhernandez@uci.cu

Categoría Docente: Instructor

Año de graduado: 2008

Profesión: Ingeniero en Ciencias Informáticas

Máster en Informática Aplicada: 2015

Síntesis del Autor

Nombre y Apellidos: Yaisel Siverio Santa Teresa

Correo electrónico: ysiverio@estudiantes.uci.cu

A nuestro comandante en jefe Fidel Castro Ruz, ya que sin él no hubiera sido posible que me encontrara hoy aquí cumpliendo uno de mis grandes sueños, ser Ingeniero en Ciencias Informáticas.

A mis padres, gracias por apoyarme en todo momento y estar pendientes de mí.

A Jose, por cuidar de mi mamá y mis viejitas mientras yo no estaba en la casa, por ser mi segundo padre, por todos los consejos y por su apoyo incondicional, gracias.

A mi hermano y a mis primos, por ser mi motor impulsor, por ese cariño que desprenden, por su inocencia, gracias.

A mi familia, gracias por apoyarme y darme ánimo en todo momento.

A mi novia Yanary, por ser especial, por aconsejarme y guiarme en todo momento, gracias.

A mi cuñadita, por su guía y su apoyo en todo este proceso y por traernos una gran alegría, Dieguito.

A mis amigos Dayron y Yoel, a los que conocí en el transcurso de la carrera y se han vuelto como hermanos para mí, gracias por soportarme durante estos 5 años.

A Julio, Yanara y Roberto Carlos, por ser como unos hermanos para mi desde el primer año de la carrera, gracias por los momentos de alegría y tristeza compartidos, nunca los olvidaré.

A mis otros amigos y amigas…. Kielam, Daynelis, Rachel, Felipe, Darián, Andy, Dennis, Luis Alberto, Asley, Jesus, Nestor y el piquete de Cumanayagua. Gracias por su apoyo y por todos los momentos que hemos compartido.

A mis amigos de la FEU, Yoelvis, Yanitza, Lionel, Yoisbel, Yosvany, Osviel y Bernardo, gracias por todos los momentos compartidos.

A Ariel, por todo el tiempo y el apoyo que dedicaste en la realización de esta tesis, muchas gracias.

A mis tutores, gracias por apoyarme y guiarme en este importante proceso, la tesis, por ser mis amigos y preocuparse de mis problemas, por darme aliento en todo momento, por todos los momentos donde me han enseñado a ser un buen profesional.

Al tribunal por todas sus recomendaciones y sugerencias, gracias.

A todos los que hicieron posible y contribuyeron para que este sueño se pudiera hacer realidad. Gracias.

Este trabajo de diploma va dedicado de forma especial a mi abuelo Humberto que aunque hoy no se encuentre entre nosotros, siempre lo llevo presente en mi mente y en mi corazón.

A mi mamá, por todos esos momentos de felicidad que me ha entregado desde que vine al mundo, por ser mi amiga y mi guía en los peores y mejores momentos, por tenerme siempre presente y tratar de darme lo mejor sin tener las posibilidades, por tratar de que nunca me faltara nada, por su gran confianza en mí aunque las posibilidades fueran mínimas, por siempre darme aliento con su frase "… mi corazón de madre me dice que vas a salir bien…", por todas esas cosas en las que una vida entera no me alcanzarían para decirle.

A mi papá, por enseñarme a ser un hombre de bien y por motivarme aunque no fuera su intención a estudiar esta maravillosa profesión, por regalarme ese gran tesoro que es Roy, el hermano que nunca pensé tener, por siempre creer en mí y estar orgulloso de su hijo mayor.

A mi abuela Carmen, por ser esa segunda mamá, mi mami Yiya, gracias por consentirme y sacarme de tantos aprietos, por apoyarme y siempre enseñarme a decir la verdad por sobre todas las cosas, por ser tan especial para mí.

A mi abuelita linda, Zoila, la viejita de mi corazón, la que siempre me espera con tanta alegría, la que me ha visto crecer y me ha inculcado tantos valores necesarios para ser una buena persona, la que siempre cuando me tenía que ir me decía "… quédate si esa carrera la puedes estudiar aquí en Cienfuegos…", esta tesis también es para ti mi abuelita linda.

A mis abuelos Leo y Félix, muchas gracias por todos los momentos de alegría compartidos, a mi abuela por hacerme esos dulces tan ricos cada vez que voy de pase, a mi abuelo por tratar de enseñarme un poco de todo lo que sabe, por aconsejarme y darme ánimos en los momentos difíciles.

A Yany, por soportarme durante este proceso largo y obstaculizado que es la Universidad, por todos sus consejos, por todos los momentos que hemos compartido juntos, por ser mi amiga, mi hermana y mi novia, por apoyarme en todas mis decisiones aunque estuvieran erradas, te quiero mucho amor.

Resumen

El uso de los Diagramas de Voronoi y sus diversas aplicaciones en las distintas ramas de la ciencia ha promovido el surgimiento de propuestas de su inclusión en diversas soluciones asociadas a la compresión de imágenes, reducción de grafos y análisis de proximidad. La presente investigación describe la realización de una biblioteca de algoritmos para obtener particionados 2D y modelar componentes en el entorno de un Sistema de Información Geográfica 2D, SIG-2D. Se utilizan como base para el desarrollo de la misma, los algoritmos de Voronoi directo, inverso e inverso generalizado. La biblioteca propuesta permite obtener particionados de planos 2D, se destaca como algoritmo fundamental el inverso generalizado de Voronoi 2D, mediante el que se pueden obtener los posibles generadores de un particionado 2D. Para guiar el proceso de desarrollo se emplea la metodología *Agile Unified Process* y se realiza una combinación entre las arquitecturas N-Capas y Orientada a Objeto, con el objetivo de lograr una adecuada comunicación entre las capas de la biblioteca. Se define para su implementación el uso de tecnologías de escritorio, con el lenguaje de programación C++. Todas las tecnologías empleadas en su desarrollo son libres.

Palabras Clave: biblioteca, modelado bidimensional, particionado bidimensional, Voronoi.

Abstract

The use of the diagrams of Voronoi and its various applications in the different branches of science has promoted the surging of proposals of its inclusion in various solutions correlated to the compression of images, reduction of graphs and analysis of proximity. The present investigation describes the realization of a library of algorithms to obtain partitions 2D and modeling components in the surroundings of a Geographical Information System 2D, SIG 2D. It is used like developmental base of her same one, blunt, inverse Voronoi's algorithms and generalized reverse. The proposed library enables getting partition from 2D plane, stands out like fundamental algorithm the reverse generalized of Voronoi 2D, the one by means of which a partitions possible generators can be obtained 2D. In order to direct the process of development Agile Unified Process uses the methodology himself and comes true a combination between architectures N-Layers and Oriented to Object, for the sake of achieving an adequate communication between the capes of the library. The use of technologies of desk is defined for your implementation, with the programming language C++. All technologies used in their development are free.

Keywords: *dimensional modeling, library, two-dimensional partitioning, Voronoi.*

Índice

Introducción **1**

Capítulo 1. Fundamentos teóricos sobre el particionado 2D y su problema inverso **5**

1. 1 Introducción del capítulo 5
1. 2 Estado actual de la temática 5
1. 3 Análisis de las soluciones existentes 7
1.3.1 Biblioteca de algoritmos CGAL 8
1.3.2 Biblioteca de algoritmos Triangle 8
1.3.3 Biblioteca de algoritmos Fade2D 9
1.3.4 Biblioteca de algoritmos Triangulation 9
1. 4 Definiciones y notaciones 9
1.4.1 Conceptos básicos de la teoría de grafos 10
1. 5 Objeto de estudio 13
1.5.1 Algoritmos para particionado del plano y su problema inverso. 14
1.5.1.1. Algoritmo Fortune 14
1.5.1.2. Tesalación inversa de Laguerre 16
1.5.1.3. Problema inverso generalizado de Voronoi (PIVG) 23
1. 6 Metodología de desarrollo de software 26
1. 7 Tecnologías a utilizar 27
1.7.1 Lenguaje de modelado 27
1.7.2 Herramienta de modelado 28
1.7.3 Lenguaje de programación 28
1.7.4 Marco de trabajo 29
1.7.5 Entorno de desarrollo integrado 30
1. 8 Conclusiones parciales 31

Capítulo 2. Descripción de la biblioteca de algoritmos para la obtención de particionados 2D y su problema inverso **32**

2. 1 Introducción del capítulo 32
2. 2 Modelo conceptual 32
2.2.1 Diagrama de clases del modelo de dominio 32
2.2.2 Análisis de los conceptos del diagrama de clases del dominio 33
2. 3 Solución Propuesta 34
2. 4 Descripción de las funcionalidades del sistema propuesto 34
2.4.1 Requisitos funcionales del sistema 34
2.4.2 Requisitos no funcionales del sistema 35
2. 5 Modelación del sistema 36
2.5.1 Determinación y justificación de los actores de sistema 36
2.5.2 Diagrama de casos de uso del sistema 37
2.5.3 Descripción extendida de los casos de uso del sistema 37

2. 6 Conclusiones parciales 45

Capítulo 3. Implementación de la biblioteca de algoritmos para la obtención de particionados 2D y su problema inverso 46

3. 1 Introducción del capítulo. 46

3. 2 Arquitectura de software 46

3.2.1 Patrones arquitectónicos 46

3. 3 Modelo de diseño 48

3.3.1. Diagrama de paquetes 48

3.3.2. Diagrama de clases del diseño 48

3.3.3. Patrones de diseño 49

3.3.4. Modelo de implementación 50

3. 4 Verificación y validación 52

3.4.1. Pruebas de software 52

3.4.1.1. Pruebas de Caja Blanca o Estructurales 53

3.4.1.2. Pruebas de Caja Negra 56

3.4.1.3. Pruebas de rendimiento 59

3.4.2. Resultados de las pruebas 60

3. 5 Conclusiones parciales 60

Conclusiones generales 61

Recomendaciones 62

Referencias Bibliográficas 63

Anexo 1. Brote de cólera en Londres 66

Anexo 2. Geografía de los dialectos de Suabia 67

Anexo 3. Estudios de área de mercado por Bogue 68

Anexo 4. Diagrama del CUS obtener generadores PIVG 69

Índice de tablas

Tabla 1.5.1.2: Estructura de la teselación Laguerre 18
Tabla 2.2.2. Conceptos del diagrama de clases del dominio. 33
Tabla 2.4.1. Requisitos funcionales. 34
Tabla 2.5. Prioridad de Casos de Uso. 36
Tabla 2.5.1. Descripción del actor del sistema. 36
Tabla 2.5.3a. Descripción del CUS Obtener particionado. 37
Tabla 2.5.3b. Descripción del CUS Obtener generadores PIV. 40
Tabla 2.5.3c. Descripción del CUS Obtener generadores PIVG. 43
Tabla 3.5.1.2a. Caso de prueba del CUS Obtener particionado. 57
Tabla 3.5.1.2b. Variables del CUS Obtener particionado. 58

Índice de figuras

Fig 1.1: Círculo más grande que contiene a los puntos de un conjunto, Shamos y Hoey. 7
Fig 1.4.1a: Grafo Simple 10
Fig 1.4.1b: Multigrafo 11
Fig 1.4.1c: Pseudografo 11
Fig 1.4.1d: Digrafo 11
Fig 1.4.1e: Multigrafo Digrafo 12
Fig 1.4.1f: Diagrama de la DCEL 13
Fig 1.5: Triangulación Delaunay y Diagrama de Voronoi para 9 puntos 14
Fig 1.5.1.1a: Representación de la línea de playa 14
Fig 1.5.1.1b: Evento sitio, aparece un nuevo arco en la línea de playa 15
Fig 1.5.1.1c: Evento círculo, un arco de parábola desaparece y se genera un vértice del diagrama 15
Fig 1.5.1.1d: Estructura de datos para almacenar la línea de playa 16
Fig 1.5.1.2a: Teselación Laguerre para 15 círculos 18
Fig 1.5.1.2b: Teselación Laguerre para 6 círculos 19
Fig 1.5.1.2c: Teselación Laguerre generado por dos conjuntos diferentes de círculos 20
Fig 1.5.1.2d: Dos generadores determinan el tercero 20
Fig 1.5.1.2e: Teselación Laguerre generada por dos conjuntos diferentes de los círculos 23
Fig 1.5.1.3a: Circulo inicial para el vértice u 24
Fig 1.5.1.3b: Arista $e = uw$ cubierta por círculos 24
Fig 1.5.1.3c: Arista cubiertas por círculos de radio ε 26
Fig 2.1: Diagrama de clases del dominio 33
Fig 2.2: Diagrama de casos de uso del sistema. 37
Fig 3.3.1: Diagrama de paquetes. 48
Fig 3.3.2: Diagrama de clases del diseño. CU Obtener particionado 49
Fig 3.2.4a: Diagrama de componente del código fuente 51
Fig 3.2.4b: Diagrama de componentes del código compilado. 51
Fig 3.5.2: Método construir_Circulos de la clase CC_DV_IG_Voronoi. 54
Fig 3.5.2.1: Grafo de flujo del método construir_Circulos de la case CC_DV_IG_Voronoi 54
Fig 3.4.1.3: Resultado de las pruebas de rendimiento. 60
Fig 3.4.2: Resultados de las pruebas realizadas a la BP2DIN. 60
Fig 4.1: Análisis del brote de cólera en Londres por John Snow, 1854 66
Fig 4.2: Geografía de los dialectos de Suabia. 67
Fig 4.3: Estudio de áreas de mercado por Bogue. Copyright: Universidad de Florida. 68
Fig 4.4: Diagrama de clases CU obtener generadores PIVG. 69

Introducción

La GC[1] es una rama de las ciencias que se encarga del diseño y análisis sistemático de algoritmos y estructuras de datos, necesarios para la solución eficiente de problemas que implican como entrada y salida objetos geométricos. El primer algoritmo de GC nace luego de que una serie de pasos correctos, no ambiguos y con un final, resuelven un problema geométrico; el precursor: Euclides. Es solo a principios de 1970 que Michael Shamos comienza un estudio sistematizado de problemas, obsesionado con la idea de crear una nueva disciplina de las ciencias de computación, a la cual llamó GC (Shamos, 1999).

Dentro de la GC el DV[2] es una de las estructuras más estudiadas, debido al amplio dominio de aplicación que tiene. Esta estructura se define como: Sea $\boldsymbol{P} = \{\boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3}, \dots, \boldsymbol{p_n}\}$ un conjunto de $\boldsymbol{n}$ puntos en el plano denominados sitios, se llama DV al conjunto de regiones o teselas $\boldsymbol{Vor}\ (\boldsymbol{p_i})$, una para cada $\boldsymbol{p_i}$ que pertenece $\boldsymbol{P}$, de forma tal que cualquier punto $\boldsymbol{q} \in \boldsymbol{Vor}\ (\boldsymbol{p_i})$ cumple que $\|\boldsymbol{q} - \boldsymbol{p_i}\| < \|\boldsymbol{q} - \boldsymbol{p_j}\| \ \forall\ \boldsymbol{p_j} \in \boldsymbol{P}$, $\boldsymbol{j} \neq \boldsymbol{i}$.

En la actualidad muchas son las aplicaciones de dicha estructura, ejemplo de ellas se pueden encontrar en las distintas ramas de ciencias, tal es el caso de la: Astronomía, Biología, Química, Geografía, Geofísica, Geología, Hidrodinámica, Matemática, Mecánica Computacional y la Robótica, concretamente se pude ver el resultado de su aplicación en el descubrimiento de galaxias insólitas (Elyiv, y otros, 2009), en la construcción de prótesis del cuerpo humano tales como: los dedos, muñecas, brazos (Martínez, 2015), ha sido aplicado también en la compresión, procesamiento y generación de mosaicos e imágenes (Martínez, y otros, 2007), (Pérez Rosé, y otros, 2011), (Dobashi, y otros, 2002).

Una aplicación inmediata de esta estructura se aprecia en los SIG[3], principalmente en el análisis de proximidad, meteorológicos o de influencias en entornos 2D, véase (de Breg, y otros, 2008), al ser estas funcionalidades básicas que deben proveer estos sistemas. Esta última se ve reflejada, en la cobertura hospitalaria, cercanía de estaciones de bomberos o de trenes, centros comerciales, control del tráfico aéreo o telefonía móvil. No son estas sus únicas aplicaciones ya que el campo de utilidad de esta estructura es más basto de lo que puede imaginarse, por ejemplo, en palabras de José Ramón Gómez:

> *"Las aplicaciones en el área de astronomía, mediante la partición basada en el análisis de métodos del modelo de puntos para la investigación de la estructura espacial de varias poblaciones estelares, o en el área médica podemos hallar la reconstrucción*

[1] GC, Geometría Computacional.

[2] DV, Diagrama de Voronoi.

[3] SIG, Sistema de Información Geográfica.

tridimensional computarizada y análisis cuantitativo de neuronas y de haces dendríticos en la corteza cerebral." (Gómez, 2011)

Actualmente en los centros GEYSED[4] y CEMC[5], pertenecientes a la UCI[6], están definidos proyectos asociados a SIG y procesamiento de imágenes. En GEYSED es de especial atención la Plataforma de desarrollo SIG-WEB denominada GeneSIG, la cual no cuenta actualmente con soporte para la obtención de DV, lo que dificulta el desarrollo de aplicativos con capacidad para análisis de problemas de proximidad. Mientras que en CEMC se trabaja en un proyecto internacional con el CIMNE[7] para el desarrollo de un SIG-3D con la restricción de no usar ningún componente desarrollado por terceros, y la necesidad de realizar análisis de proximidad e influencias, a la vez que se optimiza tiempo de procesamiento y capacidad de almacenamiento.

A partir de la problemática descrita anteriormente se deriva el siguiente **problema de investigación**: ¿Cómo facilitar la obtención de particionados 2D y la resolución de su problema inverso, sin usar componentes de terceros, en aplicaciones SIG de forma que permitan realizar análisis de proximidad y favorezcan la compresión de datos?

De acuerdo al problema planteado se define como **objeto de estudio** los algoritmos para el particionados 2D y la solución a su problema inverso delimitado por el **campo de acción** técnicas de compresión de grafos y algoritmos para el análisis de proximidad. El **objetivo general** es desarrollar una biblioteca de algoritmos para la obtención de particionados 2D y su problema inverso.

Para dirigir correctamente el proceso investigativo se definen las siguientes **preguntas de investigación**:

- ¿Qué es un particionado del espacio?
- ¿Cómo se define su problema inverso?
- ¿Qué algoritmos permiten obtener particionados en espacios 2D?
- ¿Qué algoritmos permiten resolver el problema inverso?
- ¿Qué es la compresión de grafos?
- ¿Qué algoritmos permiten la compresión de grafos?
- ¿Qué es el análisis de proximidad?
- ¿Qué algoritmos permiten realizar análisis de proximidad?

[4] GEYSED, Geoinformática y Señales Digitales.

[5] CEMC, Centro de Estudios de Matemática Computacional.

[6] UCI, Universidad de las Ciencias Informáticas.

[7] CIMNE, Centro Internacional de Métodos Numéricos en Ingeniería.

- ¿Cómo desarrollar una biblioteca de algoritmos para realizar particionados 2D directos y utilizar su problema inverso?
- ¿La biblioteca de algoritmos desarrollada garantiza el particionado 2D y su problema inverso?

Para dar cumplimiento al objetivo general se proponen las siguientes **tareas investigativas**:

- ✓ Análisis de las bibliotecas que posibilitan el particionado 2D de objetos para definir características y tendencias del manejo de las mismas.
- ✓ Fundamentación de la metodología de software, las herramientas y tecnologías a utilizar en el proceso de desarrollo.
- ✓ Descripción de los requisitos funcionales y no funcionales de la propuesta de solución para definir sus características.
- ✓ Diseño de la propuesta de solución para guiar el proceso de implementación.
- ✓ Implementación de la propuesta de solución para dar cumplimiento al objetivo planteado.
- ✓ Validación de la propuesta de solución a través de pruebas de caja blanca y caja negra para demostrar su correcto funcionamiento.

Para darle cumplimiento a las tareas investigativas anteriormente mencionadas se emplean los siguientes **métodos de la investigación científica**:

Métodos teóricos:

Se basan en el empleo del pensamiento en sus funciones de deducción, análisis y síntesis (Barchini, 2005). En la presente investigación los métodos teóricos utilizados son:

- **Analítico - sintético:** se aplica al consultar la bibliografía especializada en el tema de algoritmos para la obtención de particionados 2D e identificar elementos claves que contribuyan a la solución del problema científico planteado, permite sintetizar conceptos que ayudan a comprender la solución del problema.
- **Sistémico:** en el estudio de los algoritmos para la obtención de particionados 2D y la comprensión de su estructura, forma de trabajo y complejidad, con el objetivo de definir la forma de implementación que se debe llevar a cabo en el desarrollo de la biblioteca propuesta.
- **Modelado:** para lograr una abstracción y modelar los objetos del mundo real, primero mediante el modelo de clases persistentes y después con la creación del modelo físico.

Métodos empíricos:

Se aproximan al conocimiento del objeto mediante sus conocimientos directos y el uso de la experiencia (Barchini, 2005). En la presente investigación el método empírico empleado es:

- **Estudio de caso:** se determina su uso, pues se requiere la utilización de los algoritmos para la obtención de particionados 2D como base para la implementación de la biblioteca propuesta, y la posterior realización de pruebas para determinar su efectividad.

Se espera obtener como **posibles resultados**:

1. Biblioteca de algoritmos para la obtención de particionados 2D y su problema inverso.
2. Documentación referente al proceso de desarrollo de la biblioteca, así como los algoritmos recopilados.
3. Aplicación de ejemplo de uso de la biblioteca desarrollada.

El presente trabajo de diploma está estructurado en 3 capítulos:

Capítulo 1, titulado **"Fundamentos teóricos sobre el particionado 2D y su problema inverso"**, se realiza un análisis detallado de la situación problemática; es definido el marco conceptual de la investigación y se estudian otras soluciones existentes asociadas al dominio de la problemática que pueden dar respuesta al problema a resolver. Se analizan las tecnologías y herramientas empleadas en la elaboración de la biblioteca propuesta para dar solución al problema planteado.

Capítulo 2, titulado **"Descripción de la biblioteca de algoritmos para el particionado 2D y su problema inverso"** se realiza el modelado del dominio del problema, se plantean los requisitos funcionales y no funcionales, así como los artefactos generados en el modelado del sistema.

Capítulo 3, titulado **"Implementación de la biblioteca de algoritmos para el particionado 2D y su problema inverso"**, este capítulo está centrado en la definición de las clases de diseño, sus relaciones y la estructura de los diagramas de clases del diseño. Se establece el modelo de implementación y se concluye con la validación de la biblioteca a partir de las pruebas necesarias para demostrar el correcto funcionamiento de la misma.

Además se brindan en este documento conclusiones, recomendaciones, bibliografía y anexos; que complementan la investigación realizada.

Capítulo 1. Fundamentos teóricos sobre el particionado 2D y su problema inverso

1. 1 Introducción del capítulo

Son analizados en este capítulo los fundamentos teóricos que constituyen la base sobre la cual se realiza este trabajo investigativo, se definen conceptos que giran alrededor del objetivo general de la investigación y que permiten sintetizar y relacionar los temas vinculados con el problema a resolver. Además se analizan algunas de las soluciones existentes a nivel nacional e internacional y posibles herramientas a utilizar para el desarrollo de la aplicación.

1. 2 Estado actual de la temática

Los DV se utilizan en investigaciones donde es necesario determinar áreas de influencia, como por ejemplo, en la cobertura hospitalaria, cercanía de estaciones de bomberos o del metro, centros comerciales, control del tráfico aéreo o telefonía móvil (Martínez, 2015). Otras de las aplicaciones de los DV son el procesamiento de imágenes (Martínez, y otros, 2007) y la topología digital (Francés, y otros, 2009).

Los DV son estructuras geométricas que aparecen con frecuencia en la naturaleza, por esta razón se han redescubierto varias veces a lo largo de la historia: reciben el nombre del matemático ruso Georgy Voronoi, también son conocidos como Polígonos de Thiessen (por el meteorólogo estadounidense Alfred Thiessen), Teselación[8] de Dirichlet (en honor al matemático alemán Peter Gustav Lejeune Dirchlet), Celdas de Wigner-Seitz (por el físico matemático húngaro Eugene Wigner y el físico estadounidense Frederick Seitz) o Zonas de Brillouin (por el físico francés León Nicolás Brillouin).

A pesar de que gran parte del desarrollo y muchas de las aplicaciones iniciales que ocurrieron en el campo de las Ciencias Naturales referente a los DV, según (Okabe, y otros, 1999), la primera aplicación conocida del concepto de DV aparece en un mapa incluido en el reporte sobre el brote de Cólera de St. James, Westminster, durante el otoño de 1854. Este mapa muestra una línea discontinua, descrita como frontera con igual distancia entre la bomba de la calle Broad y otras bombas, la cual encierra un área alrededor de la bomba de la calle Broad (ver Anexo 1).

Aunque no hay alguna atribución asociada a este mapa, es más probable que esta obra de John Snow, donde se muestra la distribución de muertes alrededor de la bomba de la calle Broad en 1854, se haya convertido en el mapa más famoso de alguna enfermedad del siglo XIX que se reproduce en gran variedad de textos, tanto en epidemiología como en cartografía. Hay que notar que la distancia en el

[8] El término teselado o teselación hace referencia a una regularidad o patrón de figuras que recubren o pavimentan completamente una superficie plana que cumple con dos requisitos: que no queden huecos y que no se superpongan las figuras.

mapa no es medida en términos de distancia euclidiana[9], sino en términos de distancia a lo largo de la red de calles de Westminster, se logró una red de diagrama de área-Voronoi. Este concepto aparecería ciento cincuenta años después (Okabe, y otros, 1999).

Otra aplicación aparece en el área de las Ciencias Sociales, en el trabajo del germanista Carl Hagg, que utilizó el DV como un medio para visualizar la variación dialectal y así mismo la identificación de Isoglosas (barreras lingüísticas). En su estudio de dialectos al sureste de Alemania (1898) él definió el equivalente a un DV de un conjunto de localidades en las que se habían recogido datos de su dialecto. Posteriormente pintó los bordes de Voronoi compartidos por dos localidades si diferían en términos de características de dialecto, así, bordes muy remarcados indicaban la presencia de Isoglosas (ver Anexo 2) (Haag, 1898).

Además de estos trabajos, se han desarrollado otras aplicaciones en las Ciencias Sociales durante los últimos cincuenta años, una de las primeras fue el desarrollada por Donald Bogue en 1949, quien usó los polígonos de Voronoi definidos alrededor de centros metropolitanos en Estados Unidos (representados como puntos en el mapa) como sustitutos de sus áreas de mercado (ver Anexo 3) (Bogue, 1949).

Apoyado por otros estudios, como los de Snyder y Dacey (Okabe, y otros, 1999) esta ha logrado ser la principal área de aplicación con el trabajo que se extiende hasta niveles de tiendas minoristas. Aunque también geógrafos, antropólogos y arqueólogos han usado el concepto para modelar diversos tipos de sistemas territoriales humanos. Varios antecedentes relacionados se encuentran en la economía espacial, con la Ley de las Áreas de Mercado, la cual considera la forma de la frontera entre las áreas de mercado de dos centros que compiten en diversas condiciones de precios de mercado y costos de transporte. Argumentos similares pueden hallarse al definir varios tipos de DV ponderados (Okabe, y otros, 1999).

Las aplicaciones empíricas permanecieron limitadas debido a la falta de un medio sencillo y eficaz de construcción de DV, al depender de métodos que implicaban el uso de regla y compás, así como lleno de ambigüedades. Aunque se trataron de mostrar métodos alternativos para su creación, tal es el caso de Richard Kopec, quien propuso un método alternativo para la construcción de polígonos de Thiessen (Kopec, 1963). Esto provocó la búsqueda de soluciones desde el área en desarrollo de la informática y en los 70 ya se habían desarrollado una serie de algoritmos para construir DV en dos y tres dimensiones, que a su vez motivó desarrollos en Ciencias de la Computación que contribuyeron al campo de la naciente Geometría Computacional.

Este esfuerzo se ve unificado por la obra pionera de Michael Shamos y Dan Hoey en 1975, *Closest-point Problems* (Problemas de punto más cercano), donde no solo presentan un algoritmo para la construcción del DV, sino que también muestran la forma en la que este se puede utilizar para resolver lo que entonces

[9] En matemáticas, la distancia euclidiana o euclídea es la distancia "ordinaria" (que se mediría con una regla) entre dos puntos de un espacio euclídeo, la cual se deduce a partir del teorema de Pitágoras (Bourbaki, 1987).

eran una serie de problemas relativos a un conjunto finito de puntos distintos en el espacio euclídeo. Como hallar el árbol de expansión mínimo, identificar el vecino más cercano de cada punto y encontrar el círculo más grande que contiene a los puntos de un conjunto, cuyo centro está en el interior de la envolvente convexa de ese conjunto de puntos (ver Fig 1.1).

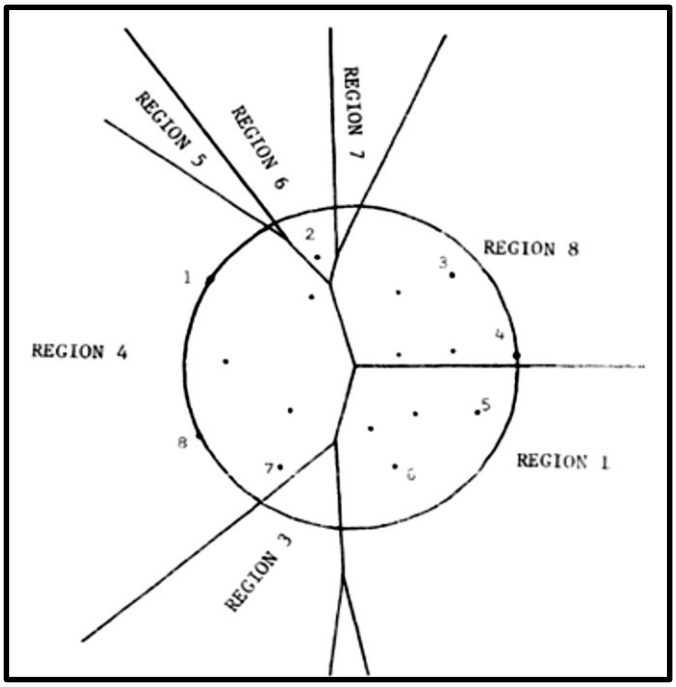


Fig 1.1: Círculo más grande que contiene a los puntos de un conjunto, Shamos y Hoey.

También propusieron formas de generalizar los DV, al ser consideradas las regiones de Voronoi asociadas con subconjuntos de k puntos de todo el conjunto de puntos en lugar de puntos individuales (Shamos, y otros, 1975). Como consecuencia de esta y otras iniciativas de aquella época, el concepto básico se ha extendido en los últimos años en una gran variedad de formas.

Su concepto dual, la TD[10] también tiene una historia marcada por redescubrimientos, aunque no tan frecuentes como el caso de los DV. La idea se origina con Voronoi en 1908, que la define por medio de relaciones entre vecinos, refiriéndose a la estructura resultante como *lensamble* de *simplexes*. Sin embargo, fue Deloné quien define la teselación (mosaico) al usar el método de la esfera vacía (Gruber, 2007). Una de las propiedades de la TD en dos dimensiones es que los triángulos individuales son tan equiláteros como sea posible.

1. 3 Análisis de las soluciones existentes

Existen disímiles soluciones que permiten realizar el particionado del plano, cada una de ellas con sus características y peculiaridades. Para la presente investigación se toman en cuenta las características

[10] TD, Teselación Delaunay.

de cuatro de ellas, que puedan ser útiles en el desarrollo de la biblioteca que se espera obtener. A continuación se describen dichas soluciones y sus principales características.

1.3.1 Biblioteca de algoritmos CGAL

CGAL es una biblioteca de algoritmos muy completa de código abierto que permite realizar el particionado del plano 2D y 3D mediante la utilización de los DV y la TD. La biblioteca está desarrollada sobre el lenguaje de programación C++ y se distribuye bajo un esquema de doble licencia. CGAL se puede utilizar junto con el software de código abierto de forma gratuita. Al utilizar CGAL en otros contextos se puede hacer mediante la obtención de una licencia comercial de *Geometry Factory*. (CGAL, 1995)

Características:

- Posee una gran adaptabilidad, gracias a su programación en C++ CGAL puede coexistir y trabajar de conjunto con otras bibliotecas en un mismo Software.
- CGAL cuenta con una clara estructura de módulos con un bajo número de dependencias, de manera tal que se pueda centrar su atención en aquellos módulos que sean de interés, lo que hace posible una mejor comprensión y uso por parte del usuario.
- La biblioteca ofrece soporte de generadores geométricos de objetos y funciones de clasificación espacial, así como un marco de búsqueda de matriz y un solucionador de programas lineales y cuadráticos. Además, ofrece interfaces para software de terceros tales como las bibliotecas Qt GUI, *Geomview*, y el *Boost Graph Library*.
- CGAL no permite que los usuarios puedan añadir nuevos módulos.
- No cuenta con soporte para realizar particionados del plano mediante DV inversos o generalizados.

Plataformas:

Linux, Mac OS, Windows.

1.3.2 Biblioteca de algoritmos Triangle

Triangle es una biblioteca de algoritmos de código abierto desarrollada mediante el lenguaje de programación C. Permite realizar el particionado del plano 2D mediante la utilización de los DV, la TD y la utilización de mallas triangulares de alta calidad. Estas últimas se generan sin importar el tamaño de los ángulos, por lo que es adecuada para el análisis de elementos finitos (Shewchuk, 2005).

- No cuenta con soporte para realizar particionados del plano mediante DV inversos o generalizados.

Plataformas:

Linux y Windows.

1.3.3 Biblioteca de algoritmos Fade2D

Fade2D es una biblioteca de algoritmos de código abierto que permite realizar el particionado del plano 2D mediante TD. La biblioteca está desarrollada sobre el lenguaje de programación C++ y se distribuye bajo las licencias comerciales y de soporte, es libre para la investigación científica. (Software, 2012)

Características:

- Es una de las bibliotecas más rápidas programadas en C++ para realizar particionados del plano mediante la TD.
- Es de fácil uso, cuenta con una buena documentación y variados ejemplos para una mejor compresión por el usuario.
- Es capaz de generar mallas de TD.
- Proporciona los ficheros *.dll y *.lib para las arquitecturas de x64 y x32 bits del sistema operativo Windows, al igual que los *.so para Linux.
- No cuenta con soporte para realizar particionados del plano mediante DV inversos o generalizados.

Plataformas:

Linux y Windows.

1.3.4 Biblioteca de algoritmos Triangulation

Triangulation es una biblioteca de algoritmos de código abierto que permite realizar el particionado del plano 2D mediante TD. La biblioteca está desarrollada sobre el lenguaje de programación C++ y se distribuye bajo la licencia pública general reducida de GNU (GNU LGPL), (Triangulation, 2011).

Características:

- Es capaz de leer información de una malla 1D, 2D o 3D que se encuentre definida en un fichero.
- Su código también se puede obtener para C, Fortran 77, Fortran 90 y Matlab, lo que le permite ser aplicada a una amplia gama de software.
- No cuenta con soporte para realizar particionados del plano mediante DV inversos o generalizados.

Plataformas:

Linux y Windows.

1. 4 Definiciones y notaciones

Antes de entrar en una descripción más detallada de algunos elementos, se definen varios conceptos y notaciones que se emplean a lo largo del documento.

1.4.1 Conceptos básicos de la teoría de grafos

La teoría de grafos es una disciplina antigua con muchas aplicaciones modernas. Las ideas básicas fueron introducidas por el matemático suizo Leonhard Euler en el siglo XVIII. Euler utilizó los grafos para resolver el famoso problema de los puentes de *Königsberg* (Euler, 1741).

$\boldsymbol{Definición\ 1.1}$ $(\boldsymbol{Grafo})$: Un grafo $\boldsymbol{G}$ es un par $\boldsymbol{G} = (\boldsymbol{V}; \boldsymbol{E})$, donde $\boldsymbol{V}$ es un conjunto finito (vértices, nodos) y $\boldsymbol{E}$ es un multiconjunto de pares no ordenados de vértices, denotados por $\{\boldsymbol{x}, \boldsymbol{y}\}$, que se denominan aristas (lados).

Se puede concluir que los grafos son estructuras discretas que cuentan con vértices y aristas unidas entre sí a los vértices.

$\boldsymbol{Definición\ 1.2}$ $(\boldsymbol{Subgrafo})$: Sea $\boldsymbol{G} = (\boldsymbol{V}; \boldsymbol{E})$ un grafo, si $\boldsymbol{H} = (\boldsymbol{W}; \boldsymbol{F})$ es un grafo tal que $\boldsymbol{W} \subseteq \boldsymbol{V}\ \boldsymbol{y}\ \boldsymbol{F} \subseteq \boldsymbol{E}$ se dice que $\boldsymbol{H}$ es un subgrafo de $\boldsymbol{G}$.

Se dice que $\boldsymbol{x}$ y $\boldsymbol{y}$ son extremos de $\{\boldsymbol{x}, \boldsymbol{y}\}$. Se denota por $\boldsymbol{V} = (\boldsymbol{G})$ el conjunto de vértices del grafo $\boldsymbol{G}$ y por $\boldsymbol{E} = (\boldsymbol{G})$ el conjunto de lados del grafo $\boldsymbol{G}$. Además $\boldsymbol{v}(\boldsymbol{G})$ y $\boldsymbol{e}(\boldsymbol{G})$ denotan el número de vértices y el número de aristas de $\boldsymbol{G}$ respectivamente. Puesto que $\boldsymbol{E}$ es un multiconjunto, es posible que existan pares repetidos y se dice que $\boldsymbol{G}$ tiene lados múltiples. También es posible que algún par no ordenado de $\boldsymbol{E}$ tenga el mismo vértice repetido, en este caso se dice que el lado es un bucle (*loop*). Si existen lados múltiples y/o lazos se dice que $\boldsymbol{G}$ es un multigrafo. Si no hay lados múltiples ni lazos se dice que es un grafo simple. Un digrafo $\boldsymbol{G}$ es un par $\boldsymbol{G} = (\boldsymbol{V}; \boldsymbol{E})$ donde $\boldsymbol{V}$ es un conjunto de vértices y $\boldsymbol{E}$ es un multiconjunto de pares ordenados. Los lados se denotan por pares ordenados, $(\boldsymbol{u}; \boldsymbol{v})$ denota el lado dirigido que tiene como vértice inicial a $\boldsymbol{u}$ y como vértice terminal a $\boldsymbol{v}$, (Martínez, 2015).

A continuación se presentan algunas definiciones extraídas del libro de Matemática Discreta y sus aplicaciones (Rosen, 2004)

$\boldsymbol{Definición\ 1.3}$ $(\boldsymbol{Grafo\ simple})$: Un grafo simple $\boldsymbol{G} = (\boldsymbol{V}; \boldsymbol{E})$ consta de $\boldsymbol{V}$, un conjunto no vacío de vértices, y de $\boldsymbol{E}$, un conjunto de pares no ordenados de elementos distintos de $\boldsymbol{V}$, a esos pares se les llama aristas o lados (Fig 1.4.1a).

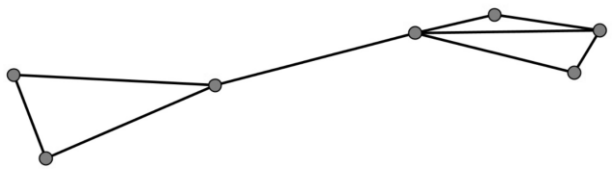


Fig 1.4.1a: Grafo Simple (Martínez, 2015).

En algunos casos lo grafos simples no bastan para modelar ciertas situaciones en las cuales se requiere de la existencia de múltiples aristas entre un par de vértices. En este caso no es suficiente definir las aristas como un par de vértices; en su lugar se utilizan los multigrafos, que constan de vértices y de aristas no dirigidas entre ellos, pero admiten la existencia de aristas múltiples entre pares de vértices.

***Definición* 1.4 (*Multigrafo*)**: Un multigrafo $G = (V; E)$ consta de un conjunto V de vértices, un conjunto E de aristas y una función f de E en $\{\{u, v\} | u, v \in V, u \neq v\}$. Se dice que las aristas $e1, e2$ son aristas múltiples o paralelas si $f(e1) = f(e2)$. (Fig 1.4.1b).

Los multigrafos definidos no admiten ciclos o lazos (aristas que conectan un vértice consigo mismo). Se usa en este caso, pseudografos que son más generales que los multigrafos.

Fig 1.4.1b: Multigrafo (Martínez, 2015).

***Definición* 1.5 (*Pseudografo*)**: Un pseudografo $G = (V; E)$ consta de un conjunto V de vértices, un conjunto E de aristas y una función f de E en $\{\{u, v\} | u, v \in V, u \neq v\}$. Se dice que una arista e es un bucle o lazo si $f(e) = \{u, u\} = \{u\}$ para algún $u \in V$. (Fig 1.4.1c).

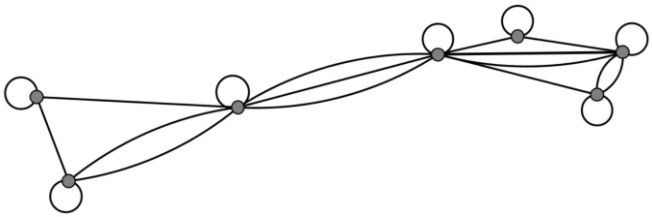


Fig 1.4.1c: Pseudografo (Martínez, 2015).

***Definición* 1.6 (*Digrafo*)**: Un grafo dirigido o digrafo $G = (V; E)$ consta de un conjunto V de vértices y un conjunto E de aristas que son pares ordenados de elementos de V (Fig 1.4.1d).

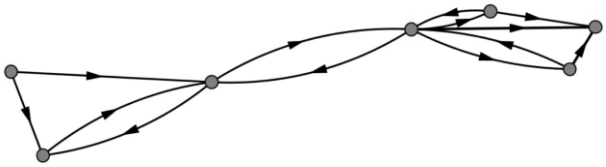


Fig 1.4.1d: Digrafo (Grafo Dirigido) (Martínez, 2015).

Al no tener en cuenta las direcciones de las aristas en G, el grafo que se obtiene se llama grafo subyacente de G. Finalmente, pueden existir multigrafos que tengan aristas dirigidas múltiples desde un vértice a un segundo vértice (que, ocasionalmente, puede coincidir con el primero).

***Definición* 1.7 (*Multigrafo dirigido*)**

Sea $G = (V; E)$ un grafo dirigido, donde V es un conjunto y E es un multiconjunto de pares ordenados de VxV, G es llamado un multigrafo dirigido y geométricamente puede representarse como un conjunto de vértices V y un conjunto de aristas E entre los vértices, donde no existe restricción en el número de aristas de un vértice a otro (ver Fig 1.4.1e).

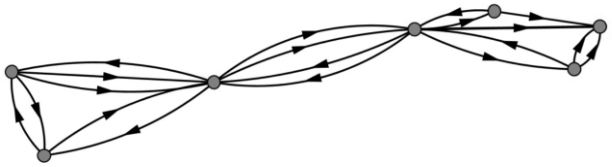


Fig 1.4.1e: Multigrafo Digrafo (Martínez, 2015).

***Definición* 1.8 (*GP*[11])**: Un grafo $\boldsymbol{G} = < \boldsymbol{V}, \boldsymbol{E} >$ se dice plano o planar si y solo si satisface cualquiera de las siguientes propiedades:

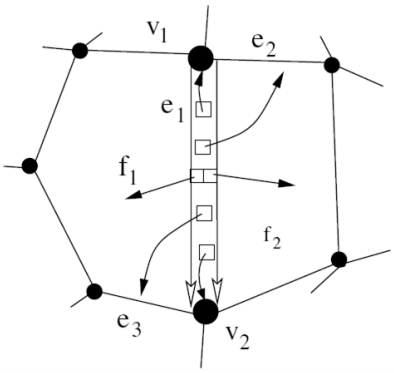
- Puede graficarse de una manera que ninguna arista se cruce con otra.
- Teorema de Kuratowski: $\boldsymbol{G}$ no contiene ningún subgrafo homeomorfo de los llamados grafos de Kuratowski: $\boldsymbol{K_5}$ *y* $\boldsymbol{K_{3,3}}$.

***Definición* 1.9 (*Double Connected Edge List* (*DCEL*))**: La DCEL para un GP considera que cada arco de $\boldsymbol{E}$ entre $(\boldsymbol{v_a}, \boldsymbol{v_b})$ contiene la siguiente información:

1. $\boldsymbol{V_0}$, que representa el vértice origen (v_a).
2. $\boldsymbol{V_d}$, que representa el vértice destino (v_d).
3. $\boldsymbol{F_l}$, que representa la cara izquierda según la dirección V_0 a V_d.
4. $\boldsymbol{F_r}$, que representa la cara derecha según la dirección V_0 a V_d.
5. $\boldsymbol{CCW_0}$, que representa el sucesor del arco $\boldsymbol{e}$ en el sentido antihorario alrededor de V_0.
6. $\boldsymbol{CCW_d}$, que representa el sucesor del arco $\boldsymbol{e}$ en el sentido antihorario alrededor de V_d.

La Fig 1.4.1f representa un ejemplo de DCEL para el arco $\boldsymbol{e_1}$ de la subdivisión dada. Según esta estructura de datos, es posible conocer en complejidad lineal las aristas alrededor de una cara dada (en el sentido horario) y las aristas alrededor de un vértice (en sentido antihorario). Para disponer de esta misma información pero en otro orden específico se deben duplicar los arcos en el GP con orientaciones opuestas a la inicial y de la misma forma construir la DCEL. La complejidad de almacenamiento para esta estructura de datos es $\boldsymbol{O}(|\boldsymbol{V}| + |\boldsymbol{F}| + |\boldsymbol{E}|)$, al ser $|\boldsymbol{V}|, |\boldsymbol{F}|, |\boldsymbol{E}|$ el número de vértices caras y aristas de la malla, respectivamente (Rivero, 2001).

[11] GP, Grafo Planar.



	v_0	v_d	F_l	F_r	CCW_o	CCW_d
e_1	v_1	v_2	f_2	f_1	e_2	e_3
e_2	v_3	v_1	f_2	f_0		
e_3	v_4	v_2	f_1	f_0		
...					...	

Fig 1.4.1f: Diagrama de la DCEL (Rivero, 2001).

1. 5 Objeto de estudio

En la actualidad existen diversas formas de obtener el particionado del plano, entre las que se encuentran la TD y el DV. Mediante estos métodos se pueden realizar distintos particionados con características relativamente parecidas. La TD es una red de triángulos que cumple la condición de Delaunay. Esta condición dice que la circunferencia circunscrita de cada triángulo de la red no debe contener ningún vértice de otro triángulo. Se usa la TD en la geometría por ordenador, especialmente en gráficos 3D por computadora. Se le denomina así por el matemático ruso Boris Nikolaevich Delone (Борис Николаевич Делоне, 1890 - 1980) quien lo crea en 1934; el mismo Delone usa la forma francesa de su apellido, «Delaunay», como apreciación a sus antecesores franceses (Wormser, 2008).

El DV es una estructura de datos que se han estudiado de manera profunda, fundamentalmente en la GC. Un DV se puede definir como el diagrama de la reducción al mínimo de un conjunto finito de funciones continuas. Por lo general, cada una de esas funciones se interpreta como la función de la distancia a un objeto. Se subdivide en regiones, cada región puede contener puntos, estos siempre van a estar más cercanos al punto generador de la región que a los restantes puntos de la subdivisión.Se pueden definir muchas variantes de DV, estas dependen de la clase de objetos, las funciones de distancia y la incrustación del espacio. Diagramas cuyas regiones son politopos[12] convexos, se conocen bien y sus propiedades se pueden deducir de las propiedades de los politopos y pueden ser construidos de manera eficiente (Wormser, 2008).

[12] En geometría politopo significa, en primer lugar, la generalización a cualquier dimensión de un polígono bidimensional, o un poliedro tridimensional.

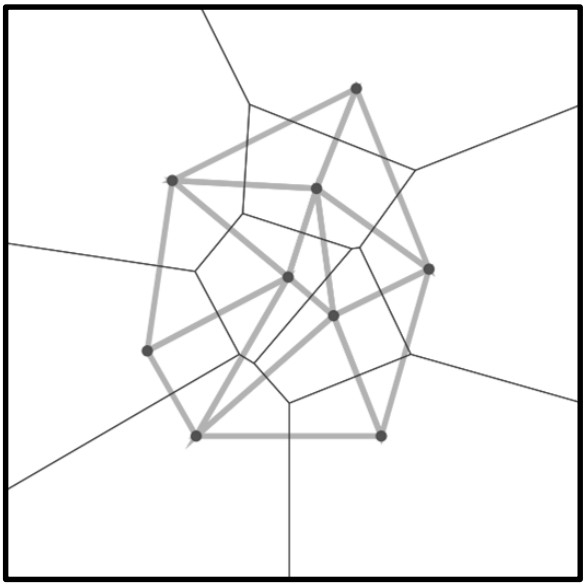


Fig 1.5: Triangulación Delaunay y Diagrama de Voronoi para 9 puntos (Wormser, 2008).

En la presente investigación se mencionan solamente los algoritmos de particionado del plano basados en las clasificaciones de los DV, directa, inversa e inversa generalizada.

1.5.1 Algoritmos para particionado del plano y su problema inverso.

1.5.1.1. ***Algoritmo Fortune*** (Marbete, y otros, 2013)**,** (Ramón, 2013)

En 1985, Fortune desarrolló un algoritmo para la construcción del DV. Este utiliza la estrategia de barrido de línea implementada por muchos algoritmos de gráficos por computadora y GC. La idea del algoritmo es muy simple y su complejidad tiempo es $\boldsymbol{O(n \log_2 n)}$ en el peor caso, donde $\boldsymbol{n}$ es el número de puntos de entrada. En la Fig 1.5.1.1a, se puede observar que una estructura topológica con la parte frontal en forma de parábola se cambia al deslizar la línea de exploración sobre el plano $\boldsymbol{xy}$**,** esta estructura es conocida como línea de playa (Marbete, y otros, 2013).

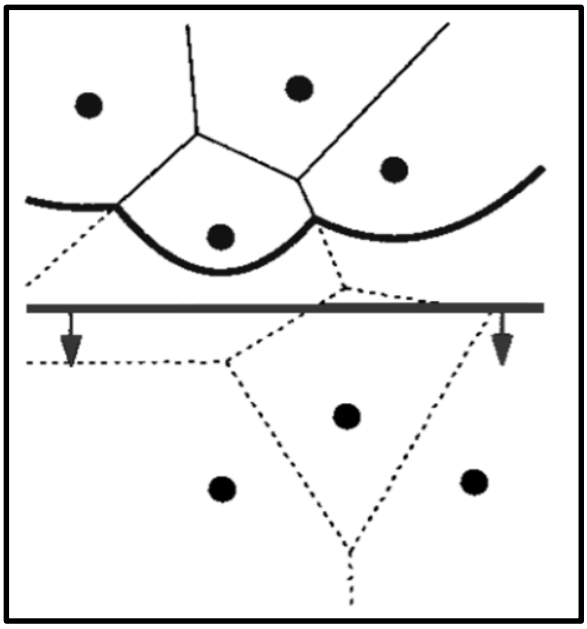


Fig 1.5.1.1a: Representación de la línea de playa (Ramón, 2013).

Mientras se realiza un barrido del plano se hacen las siguientes acciones en dependencia del tipo de evento:

Evento de Sitio: Si no es el primer evento de sitio, aparece un nuevo arco en la línea de playa que divide en dos el arco justo arriba del sitio al ocurrir un evento. En ese momento una arista del diagrama comienza a trazarse. Es necesario chequear si el arco que se divide tiene asociado un evento de círculo, en cuyo caso sería una falsa alarma (Ramón, 2013). Ver Fig 1.5.1.1b.

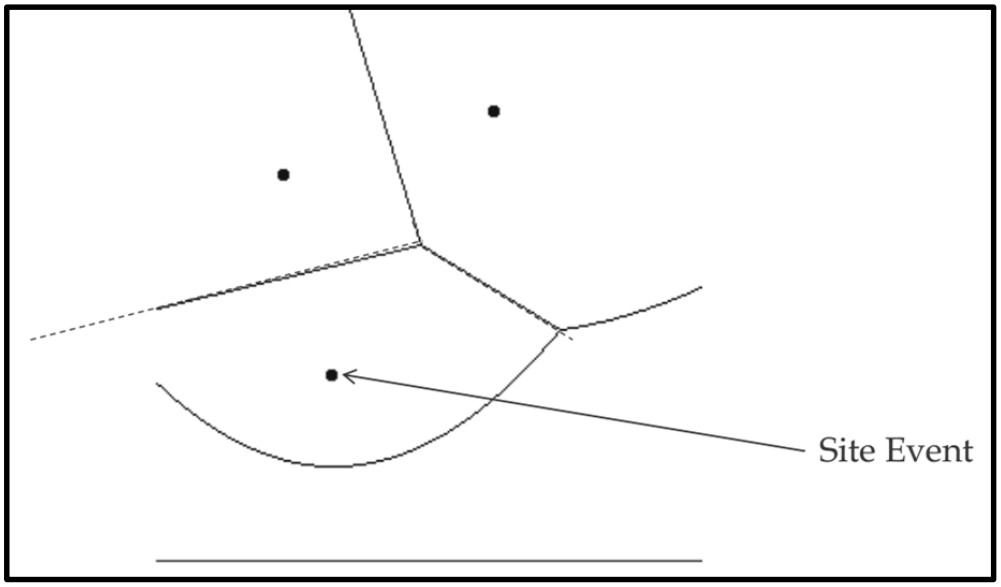


Fig 1.5.1.1b: Evento sitio, aparece un nuevo arco en la línea de playa (Ramón, 2013).

Evento de Círculo: Un arco desaparece y se genera un vértice del diagrama. Estos eventos se generan dinámicamente, al chequear que siempre aparezcan tres arcos consecutivos en la línea de playa, lo cual puede ocurrir después de cualquier evento (Ramón, 2013). Ver Fig 1.5.1.1c.

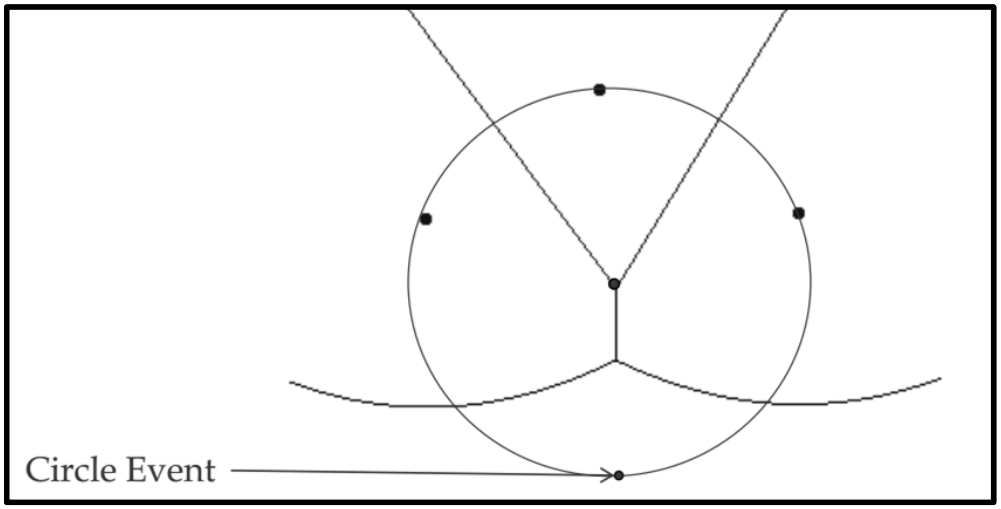


Fig 1.5.1.1c: Evento círculo, un arco de parábola desaparece y se genera un vértice del diagrama (Ramón, 2013).

Estructura de datos

Para almacenar la línea de playa es necesario crear un árbol binario de búsqueda $\boldsymbol{T}$**.** Ver Fig 1.5.1.1d.

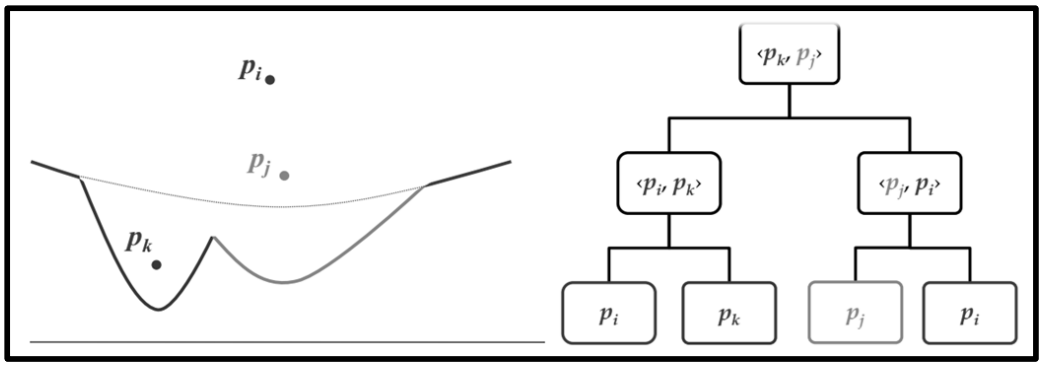


Fig 1.5.1.1d: Estructura de datos para almacenar la línea de playa (Ramón, 2013).

Nótese que un arco puede contribuir en más de una ocasión en la línea de playa y los nodos internos del árbol representan los puntos de intersecciones de los arcos parábola (Ramón, 2013).

Algoritmo de Fortune

VoronoiDiagram(P)

Entrada: Un conjunto $\boldsymbol{P}$ de sitios en el plano.

Salida: Diagrama de Voronoi de $\boldsymbol{P}$ guardado en la DCEL $\boldsymbol{D}$.

1. Inicializar una cola de eventos $\boldsymbol{Q}$ con todos los eventos de los sitios, $\boldsymbol{T} = \varnothing$, $\boldsymbol{D} = \varnothing$.
2. **Mientras** $\boldsymbol{Q}$ no está vacía **hacer**
3. Eliminar el evento con mayor coordenada $\boldsymbol{y}$ en $\boldsymbol{Q}$.
4. **if** el evento es de sitio $\boldsymbol{p_i}$ **then**
5. Mejorar evento sitio ($\boldsymbol{p_i}$)
6. **else**

   Mejorar evento círculo ($\boldsymbol{y}$) donde $\boldsymbol{y}$ es la hoja en $\boldsymbol{T}$ que representa el arco desaparece.
7. Los nodos internos en $\boldsymbol{T}$ aún representan la mitad de los medios bordes infinitos, calcular un cuadro delimitador para unir los bordes dentro del mismo.
8. Atravesar los bordes $\boldsymbol{D}$ para fijar la **DCEL** mediante el establecimiento de los puntos entre las caras y los bordes.

1.5.1.2. Tesalación inversa de Laguerre (Duan, y otros, 2014)

Una teselación Laguerre, también llamada diagrama de potencia o diagrama de Laguerre, es una versión ponderada de la conocida teselación de Voronoi. A continuación se explican algunas de las notaciones matemáticas por las que esta se rige, además de describir un algoritmo que permite obtener dicha teselación. Para más detalles (Okabe, y otros, 1999).

Sea $\boldsymbol{p} \in \mathbb{R}^{\boldsymbol{d}}$ y $\boldsymbol{p} \in \mathbb{R}$ un valor fijo, llamado peso del punto $\boldsymbol{p}$. Se denomina al par $(\boldsymbol{p}, \boldsymbol{w})$ un punto ponderado. Para toda $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{d}}$, se define el poder de $\boldsymbol{x}$ con respecto a $(\boldsymbol{p}, \boldsymbol{w})$ como $\boldsymbol{pow}\big(\boldsymbol{x}, (\boldsymbol{p}, \boldsymbol{w})\big) = \|\boldsymbol{x} - \boldsymbol{p}\|^{\boldsymbol{2}} - \boldsymbol{w}$.

Si se considera que $\boldsymbol{P} = \{(\boldsymbol{p_1}, \boldsymbol{w_1}), (\boldsymbol{p_2}, \boldsymbol{w_2}), \dots, (\boldsymbol{p_n}, \boldsymbol{w_n})\}$ son un conjunto de puntos finitos ponderados (generadores) en la Red. La teselación Laguerre $\boldsymbol{P}$ divide a $\mathbb{R}^{\boldsymbol{d}}$ en células, mediante el poder de dichos punto. La celda asociada con el i-ésimo punto generador, $\boldsymbol{C_i}$, se define por $\boldsymbol{C_i} = \{\boldsymbol{X} \in \mathbb{R}^{\boldsymbol{d}} : \boldsymbol{pow}\big(\boldsymbol{x}, (\boldsymbol{p_i}, \boldsymbol{w_i})\big)\ \boldsymbol{6}\ \boldsymbol{pow}\Big(\boldsymbol{x}, \big(\boldsymbol{p_j}, \boldsymbol{w_j}\big)\Big), \boldsymbol{i} \neq \boldsymbol{j}\}$.

Tenga en cuenta que si todos los pesos son iguales, la teselación Laguerre se reduce a la teselación de Voronoi estándar.

Observación 1. La teselación Laguerre está formada por un conjunto de puntos finitos localmente, pero un conjunto de puntos infinitos de generadores pueden definirse de la misma manera.

Al ser todos los pesos positivos, cada punto generador ponderado $(\boldsymbol{p}, \boldsymbol{w}) \in \boldsymbol{P}$ se puede interpretar y visualizar como una esfera (denotada por $\boldsymbol{S}(\boldsymbol{p}, \boldsymbol{r})$) con un radio $\boldsymbol{r} = \sqrt{\boldsymbol{w}} \geq \boldsymbol{0}$ con centro en el punto $\boldsymbol{p}$. El poder de un punto $\boldsymbol{x}$ con respecto a la esfera $\boldsymbol{S}(\boldsymbol{p}, \boldsymbol{r})$ está dado por $\boldsymbol{pow}(\boldsymbol{x}, \boldsymbol{S}(\boldsymbol{p}, \boldsymbol{r})) = \|\boldsymbol{x} - \boldsymbol{p}\|^{\boldsymbol{2}} - \boldsymbol{r_2}$.

Geométricamente, esto significa que para un punto $\boldsymbol{x}$ fuera de la esfera $\boldsymbol{S}(\boldsymbol{p}, \boldsymbol{r})$, el valor de $\boldsymbol{pow}(\boldsymbol{x}, \boldsymbol{S}(\boldsymbol{p}, \boldsymbol{r}))$ es igual a la longitud al cuadrado de la línea tangente de $\boldsymbol{x}$ a $\boldsymbol{S}(\boldsymbol{p}, \boldsymbol{r})$. El límite entre dos células adyacentes generadas por las esferas $\boldsymbol{S_1} = \boldsymbol{S}(\boldsymbol{p_1}, \boldsymbol{r_1})$ y $\boldsymbol{S_1} = \boldsymbol{S}(\boldsymbol{p_2}, \boldsymbol{r_2})$, se compone de todos los puntos $\boldsymbol{z} \in \mathbb{R}^{\boldsymbol{d}}$ tal que $\boldsymbol{pow}(\boldsymbol{z}, \boldsymbol{S_1}) = \boldsymbol{pow}(\boldsymbol{z}, \boldsymbol{S_2})$. Estos puntos forman un hiperplano $\boldsymbol{H}(\boldsymbol{S_1}, \boldsymbol{S_2})$, donde $\boldsymbol{H}(\boldsymbol{S_1}, \boldsymbol{S_2}) = \{\boldsymbol{z} \in \mathbb{R}^{\boldsymbol{d}} : \boldsymbol{2}\langle \boldsymbol{z}, \boldsymbol{p_1} - \boldsymbol{p_2} \rangle = \|\boldsymbol{p_1}\|^{\boldsymbol{2}} - \|\boldsymbol{p_2}\|^{\boldsymbol{2}} + \boldsymbol{r_2^2} - \boldsymbol{r_1^2}\}$.

Este límite es perpendicular a la línea que une $\boldsymbol{p_1}$ y $\boldsymbol{p_2}$ y se denomina eje radical de $\boldsymbol{S_1}$ y $\boldsymbol{S_2}$. En esta investigación, solo se va a considerar la teselación Laguerre en $\mathbb{R}^{\boldsymbol{2}}$. En este caso, los generadores pueden interpretarse como círculos.

Una teselación Laguerre se puede representar matemáticamente como un gráfico geométrico o una colección de vértices $\boldsymbol{V} = \{\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_m}\}$ (también $\boldsymbol{V}(\boldsymbol{P})$) y los bordes $\{(\boldsymbol{v_i}, \boldsymbol{v_j})\}$, donde los vértices son asignados a posiciones en el espacio. Al tener en cuenta que algunas células no solo están limitadas por segmentos, sino también por las rectas que se extienden hasta el infinito en una dirección determinada, porque se consideraron conjuntos finitos de generadores. Tales bordes se representan a menudo al hacer uso de un vértice "ficticio". Es decir, un vértice del borde está dispuesto a ser un punto arbitrario en la recta que se extiende hasta el infinito, véase (Schoenberg, y otros, 2003) (Aurenhammer, 1987). Los vértices interiores de la teselación tienen el mismo poder con respecto a tres círculos. Por el contrario, los vértices “ficticios” solo tienen igual poder con respecto a dos círculos. El uso de la

representación anterior, los vértices y los bordes de la teselación Laguerre se pueden almacenar en el formato dado en la Tabla 1.5.1.2.

Tabla 1.5.1.2: Estructura de la teselación Laguerre (Duan, y otros, 2014).

cell i	cell j	edge $e_{i,j}$			
		$v_1(x)$	$v_1(y)$	$v_2(x)$	$v_2(y)$
1	4	126.14	138.02	140.86	156.07
1	11	98.75	150.55	126.14	138.02
1	14	93.26	164.39	98.75	150.55
2	6	100.97	84.85	107.45	89.69
2	10	107.45	89.69	112.99	113.46
2	11	112.99	113.45	83.46	126.78
⋮	⋮	⋮		⋮	

Las células se marcan de $\mathbf{1}$ a $\boldsymbol{n}$. Las dos primeras columnas de la Tabla 1 corresponden a la etiquetas de células adyacentes. Por ejemplo, la célula $\mathbf{1}$ es adyacente a las células $\mathbf{4}$, $\mathbf{11}$, y $\mathbf{14}$. Las coordenadas de los vértices del borde que separa dos células se dan en las columnas $\mathbf{3}-\mathbf{6}$. Tenga en cuenta que una representación más compacta se puede lograr mediante el almacenamiento de las coordenadas de cada vértice, junto con los índices de los vértices adyacentes.

Cuando las células comparten bordes y vértices, es decir, se encuentran cara a cara (2D), se les denomina teselación normal. En este caso, cada borde limita con exactamente dos células, y cada vértice es compartido por exactamente tres células. Un ejemplo de un teselado normal Laguerre 2D, generado por 15 círculos, se muestra en la Fig 1.5.1.2a. Los límites entre las células son los bordes de la gráfica geométrica. Las competencias de los puntos en cada límite son iguales con respecto a los dos círculos vecinos. Observe que el grado de cada uno (interior) es el vértice 3. Los poderes de los vértices son, por tanto, igual con respecto a los tres círculos vecinos.

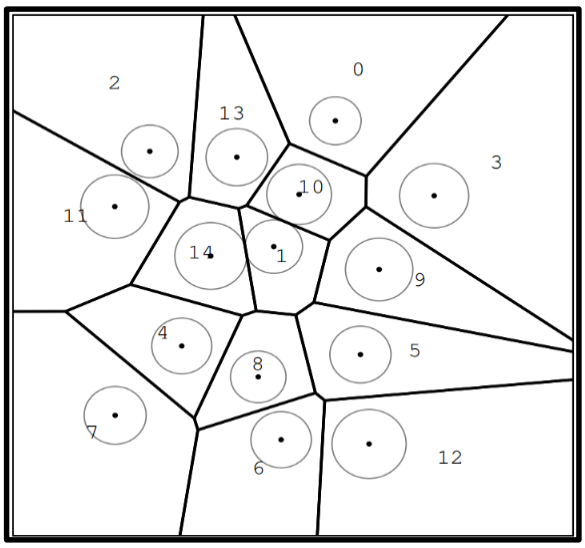


Fig 1.5.1.2a: Teselación Laguerre para 15 círculos (Duan, y otros, 2014).

A partir de ahora, se considera solamente la teselación normal Laguerre 2D, por ejemplo, los puntos del generador están al azar y uniformemente elegidos dentro de un área de muestreo limitada, la teselación es normal con probabilidad 1.

Propiedades

1. Una propiedad de la teselación Laguerre no contiene necesariamente su generador y un generador no necesariamente genera una célula.

Esta propiedad es bien conocida (véase, por ejemplo, (Aurenhammer, 1987)). La Fig 1.5.1.2b (a partir de (Aurenhammer, 1987), (Lautensack, 2007)), muestra que el punto de una célula Laguerre generador puede estar fuera de su celda; en particular, $\boldsymbol{p_4}$ se encuentra fuera de la célula $\boldsymbol{4}$. La misma figura muestra un círculo generador, $\boldsymbol{S(p_6, r_6)}$, para el que la célula Laguerre correspondiente está vacía. Una consecuencia de la propiedad 1 es que el generador dado para una teselación no es único. Es decir, se puede añadir círculos que no generan células adicionales, y el nuevo sistema brinda como resultado la misma teselación Laguerre que el original. Sin embargo, incluso al generar cada círculo una célula, los generadores de una teselación Laguerre no son necesariamente únicos.

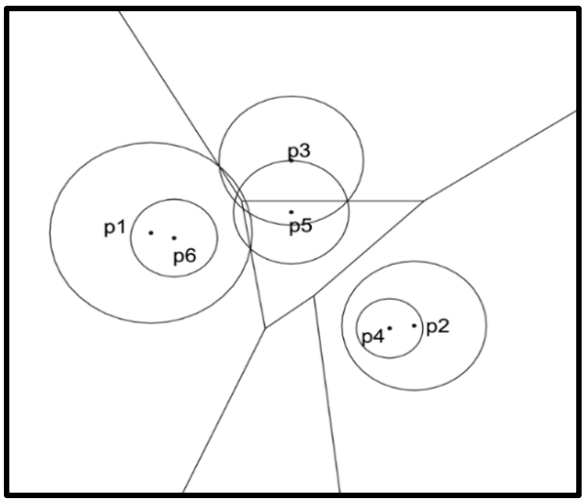


Fig 1.5.1.2b: Teselación Laguerre para 6 círculos (Duan, y otros, 2014).

2. Dos conjuntos completamente diferentes de los círculos pueden generar la misma teselación Laguerre.

Un ejemplo extremo se muestra en Fig 1.5.1.2c, donde tanto los círculos grises y círculos de color amarillo dan la misma teselación. La propiedad 2 se ha mencionado en la literatura (véase (Lautensack, 2007), (Aurenhammer, 1987)). Sin embargo, no es sorprendente que haya recibido poca atención. En particular, no se ha hecho hincapié en que dos conjuntos completamente diferentes (tanto en localización y radios) de círculos pueden generar la misma teselación.

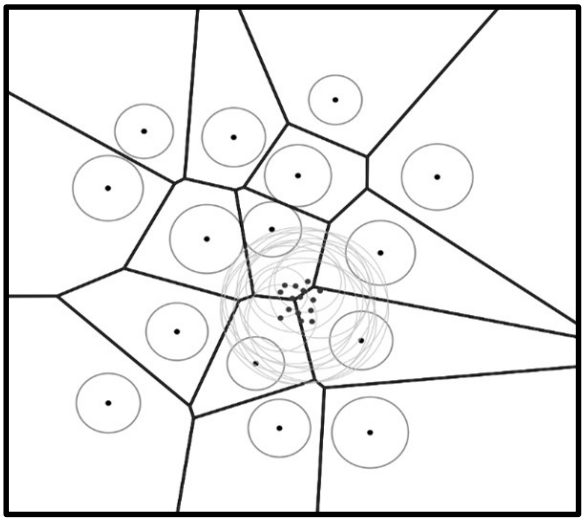


Fig 1.5.1.2c: Teselación Laguerre generado por dos conjuntos diferentes de círculos (Duan, y otros, 2014).

Generación de una teselación Laguerre mediente puntos ponderados

Se comienza a describir cómo, por una teselación dada (normal), se pueden determinar un conjunto de puntos ponderados generadores al especificar solo las coordenadas y el peso de un punto generador ponderado, además de una coordenada y del punto generador ponderado de una célula vecina.

Teorema 1.5.1 Los puntos generadores ponderados de una teselación normal Laguerre 2D dados, pueden ser completamente determinados desde el punto generador ponderado de una celda interior, y una coordenada del punto generador ponderado de una celda adyacente.

Prueba. Por razones de simplicidad se asume que los pesos de los puntos de generador son positivos, aunque la prueba no utiliza esta suposición. La ventaja es que los puntos de generador ponderados se pueden interpretar como círculos; Véase la Fig 1.5.1.2d. Sea $\boldsymbol{S_1(p_1, r_1)}$ el círculo generador de alguna celda interior $\boldsymbol{C_1}$, y $\boldsymbol{S_2(p_2, r_2)}$ el círculo generador de una celda adyacente a $\boldsymbol{C_2}$. Con las coordenadas de $\boldsymbol{P_1}$ y $\boldsymbol{P_2}$, $(\boldsymbol{x_1}; \boldsymbol{y_1})$ y $(\boldsymbol{x_2}; \boldsymbol{y_2})$, respectivamente. Se supone que $\boldsymbol{x_1, y_1, r_1}$ y $\boldsymbol{x_2}$ se tienen como datos.

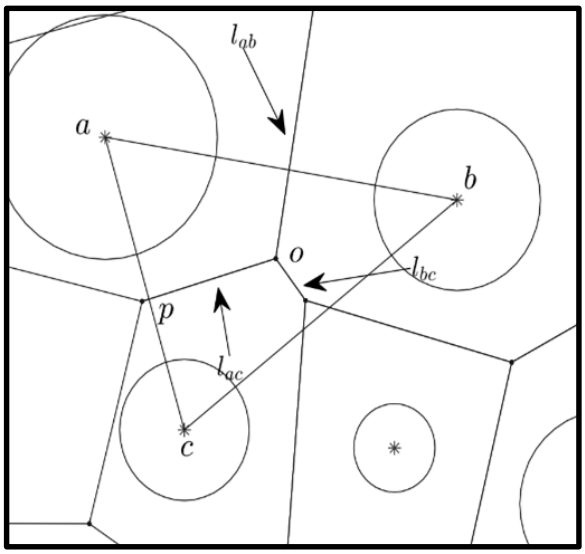


Fig 1.5.1.2d: Dos generadores determinan el tercero (Duan, y otros, 2014).

Al ser $\boldsymbol{e_{1,2}}$ el borde de la teselación de separación $\boldsymbol{C_1}$ y $\boldsymbol{C_2}$. Para la teselación Laguerre, el segmento de línea que une $\boldsymbol{p_1}$ y $\boldsymbol{p_2}$ es perpendicular a $\boldsymbol{e_{1,2}}$. Usar $\boldsymbol{m_{1,2}}$ como la pendiente de $\boldsymbol{e_{1,2}}$. De ello se deduce que $\boldsymbol{y_2}$ es determinada por $\frac{y_2-y_1}{x_2-x_1} = -\frac{1}{m_{1,2}}$ (1), (Tenga en cuenta que es posible que $\boldsymbol{m_{1,2}}$, esto no es relevante en aplicaciones prácticas. El problema puede ser resuelto, por ejemplo, mediante la rotación de la teselación. Por lo tanto, se supone la pendiente sea diferente de cero). A partir de esto se puede determinar $\boldsymbol{r_2}$ mediante $\|\boldsymbol{p_1}-\boldsymbol{q}\|^2 - \boldsymbol{r_1^2} = \|\boldsymbol{p_2}-\boldsymbol{q}\|^2 - \boldsymbol{r_2^2}$, (2), para cualquier punto $\boldsymbol{q}$ en la línea que contiene el segmento $\boldsymbol{e_{1,2}}$. En particular, se puede tomar $\boldsymbol{q} = \boldsymbol{p_{1,2}}$, como la intersección de la línea a través $\boldsymbol{p_1}$ y $\boldsymbol{p_2}$, y la línea que contiene el borde $\boldsymbol{e_{1,2}}$.

Debido a que la teselación se supone que es normal y $\boldsymbol{C_1}$ es una célula interior, hay una célula adyacente tanto a $\boldsymbol{C_1}$ y $\boldsymbol{C_2}$, como a $\boldsymbol{C_3}$, la generación del círculo $\boldsymbol{C_3}$, $\boldsymbol{S_3} = (\boldsymbol{p_3}, \boldsymbol{r_3})$, se determina de la siguiente manera. El punto $\boldsymbol{P_3} = (\boldsymbol{x_3}, \boldsymbol{y_3})$ es la intersección de (1) la línea que pasa a través de $\boldsymbol{p_1}$, es perpendicular a $\boldsymbol{e_{1,2}}$ (el borde entre $\boldsymbol{C_1}$ y $\boldsymbol{C_3}$) y (2) la línea que pasa a través de $\boldsymbol{p_2}$ y es perpendicular a $\boldsymbol{e_{1,2}}$ (el borde entre $\boldsymbol{C_2}$ y $\boldsymbol{C_3}$).

De ello se deriva que las coordenadas $\boldsymbol{x_3}$ y $\boldsymbol{y_3}$ de $\boldsymbol{p_3}$ satisfacen $\frac{y_3-y_1}{x_3-x_1} = -\frac{1}{m_{1,3}}$ $\boldsymbol{y}$ $\frac{y_3-y_2}{x_3-x_2} = -\frac{1}{m_{2,3}}$, (3) donde $\boldsymbol{m_{1,2}}$ y $\boldsymbol{m_{2,3}}$ son las pendientes de $\boldsymbol{e_{1,2}}$ y $\boldsymbol{e_{2,3}}$, respectivamente. Por lo tanto, $\boldsymbol{x_3} = \frac{m_{2,3}(m_{1,3}y_1+x_1)-m_{1,3}(m_{2,3}y_2+x_2)}{m_{2,3}-m_{1,3}}$, $\boldsymbol{y_3} = \frac{m_{1,3}y_1+x_1-(m_{2,3}y_2+x_2)}{m_{1,3}-m_{2,3}}$. (4). El radio $\boldsymbol{r_3}$, se determina entonces como en (2); es decir,$\|\boldsymbol{p_1}-\boldsymbol{q}\|^2 - \boldsymbol{r_1^2} = \|\boldsymbol{p_3}-\boldsymbol{q}\|^2 - \boldsymbol{r_3^2}$, (5), donde $\boldsymbol{q}$ es cualquier punto de la línea que contiene el borde $\boldsymbol{e_{1,3}}$. También es posible determinar $\boldsymbol{r_3}$ al considerar el par $\boldsymbol{C_2}$, $\boldsymbol{C_3}$ en lugar de $\boldsymbol{C_1}$, $\boldsymbol{C_3}$, con $\|\boldsymbol{p_2}-\boldsymbol{q}\|^2 - \boldsymbol{r_2^2} = \|\boldsymbol{p_3}-\boldsymbol{q}\|^2 - \boldsymbol{r_3^2}$, (6) donde $\boldsymbol{q}$ es cualquier punto de la línea que contiene el borde $\boldsymbol{e_{2,3}}$.

Para probar que (5) y (6) dan el mismo valor para $\boldsymbol{r_3^2}$, es suficiente para demostrar que $\boldsymbol{r_1^2} - \|\boldsymbol{p_1}-\boldsymbol{u}\|^2 = \boldsymbol{r_2^2} - \|\boldsymbol{p_2}-\boldsymbol{u}\|^2$, (7) donde $\boldsymbol{u}$ es el vértice en la intersección de las líneas que contienen los bordes $\boldsymbol{e_{1,2}}$ y $\boldsymbol{e_{2,3}}$. Pero esto se induce directamente de la definición de la teselación Laguerre, y el hecho de que $\boldsymbol{u}$ se encuentra también en la línea a través del borde $\boldsymbol{e_{1,2}}$. Al proceder de esta manera, es posible determinar de forma iterativa el círculo generador de cada celda.

Observación 2. El teorema 1.5.1 se plantea en 2D, pero su concepto es aplicable para las dimensiones más altas mediante el uso de vectores normales de hiperplanos separados por células, por ejemplo, los bordes en 2D o caras planas en 3D, donde los generadores deben estar en líneas con la misma orientación. Un ejemplo similar se puede encontrar en (Aurenhammer, 1987), donde se expone una construcción no única para la ortogonal, dual de un teselado Laguerre (en el caso de Voronoi esto corresponde a la triangulación de Delaunay).

Observación 3. La demostración del Teorema 1.5.1 muestra que si se añade la misma constante a todos los radios al cuadrado, la teselación no cambia. Al conocer, si $\|\boldsymbol{p_i} - \boldsymbol{q}\|^2 - \boldsymbol{r_i^2} = \|\boldsymbol{p_j} - \boldsymbol{q}\|^2 - \boldsymbol{r_j^2}$ para cada $\boldsymbol{q}$ en un borde que separa las células adyacentes $\boldsymbol{C_i}$ y $\boldsymbol{C_j}$ y al ser $\boldsymbol{r_i^2}$ y $\boldsymbol{r_j^2}$ sustituidos por $\boldsymbol{r_i^2 + c}$ y $\boldsymbol{r_j^2 + c}$. Se obtiene como consecuencia de ello, que siempre es posible encontrar un conjunto de generadores todos con pesos positivos. Si existen pesos negativos, simplemente se determina el mínimo de estos y se resta este valor a todos los pesos.

El teorema 1.5.1 y la observación 3 sugieren el siguiente algoritmo para determinar los círculos generadores de un teselado Laguerre, dado $\boldsymbol{x_1}$, $\boldsymbol{y_1}$, $\boldsymbol{r_1}$ y $\boldsymbol{x_2}$. Se observa que el algoritmo puede empezar con un par de células (internas) ($\boldsymbol{C_1, C_2}$). Estas pueden ser reemplazadas por cualquier par de células internas durante el reetiquetado.

Algoritmo 1. Construción de Generadores

Entrada: $\boldsymbol{x_1}$, $\boldsymbol{y_1}$, $\boldsymbol{r_1}$ $(\geq \boldsymbol{0})$, $\boldsymbol{x_2}$ y los datos de la teselación como se encuentran en la Tabla 1.5.1.2.

Salida: Un conjunto de generadores $\boldsymbol{P} = \{(\boldsymbol{p_k}, \boldsymbol{r_k^2})\}$ con radios mínimos no negativos.

1: Inicializar $\boldsymbol{P} = \{(\boldsymbol{p_1}, \boldsymbol{r_1^2}, \dots, \boldsymbol{p_n}, \boldsymbol{r_n^2})\}$ para $\boldsymbol{NAN_{nx3}}$.
2: Calcular $\boldsymbol{y_2}$ y $\boldsymbol{r_2^2}$ mediante (1) y (2). Utilizar una bandera para indicar que $\boldsymbol{C_1}$ y $\boldsymbol{C_2}$ han sido asignados a los generadores.
3: **mientras** no todas las células tengan generadores **hacer**
4: **para** $\boldsymbol{k = 1{:}n}$ **hacer**
5: **si** $(\boldsymbol{p_k}, \boldsymbol{r_k^2})$ no han sido asignados y más de dos células adyacentes han sido asignadas.
entonces
6: Elegir dos de las células adyacentes a $\boldsymbol{C_k}$ con los generadores asignados.
7: Calcular $\boldsymbol{p_k}$ mediante la ecuación (3).
8: Calcular $\boldsymbol{r_k^2}$ mediante la ecuación (2).
9: **fin del si**
10: **fin del para**
11: **fin del mientras**
12: **si** $\boldsymbol{\min\{r_k^2\} < 0}$ **entonces**
13: $\boldsymbol{\text{se t}\{r_k^2\} = \{r_k^2\} - min\{r_k^2\}}$
14: **fin del si**

El algoritmo proporciona un método para recuperar los círculos de generación de una teselación dado un pequeño número de entradas, es decir, $\boldsymbol{x_1}$, $\boldsymbol{y_1}$, $\boldsymbol{r_1}$ y $\boldsymbol{x_2}$. Sin embargo, al depender de cómo se

seleccionen estas entradas, se podrán obtener resultados muy diferentes. Aunque el algoritmo garantiza pesos positivos, en algunos casos los círculos de generación pueden estar lejos fuera de las células que generan. Un ejemplo se da en la Fig 1.5.1.2e. En otros casos, los radios de un círculo generador pueden ser mucho más grandes que las células o incluso la ventana de observación; véase la Fig 1.5.1.2e.

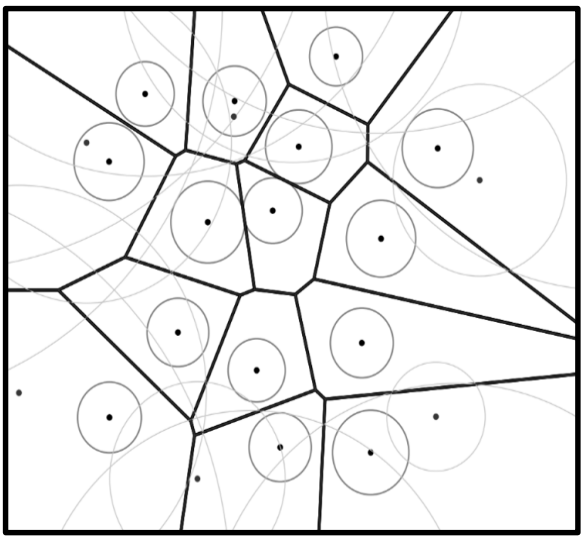


Fig 1.5.1.2e: Teselación Laguerre generada por dos conjuntos diferentes de los círculos (Duan, y otros, 2014).

Como se ha indicado anteriormente, muchas aplicaciones basadas en la teselación Laguerre atribuyen significado a los círculos de generación. Por esta razón, es importante que se tenga un método que elija una solución que satisfaga criterios como los enumerados anteriormente. Algunos de estos criterios son dependientes del modelo; otros sin embargo, son bastante universales. En particular, es casi siempre deseable tener puntos de generación que se encuentren dentro de las células que generan y es casi siempre deseable tener radios de valor real.

1.5.1.3. Problema inverso generalizado de Voronoi (PIVG) (Almaguer, y otros, 2007)

La idea seguida por el algoritmo consiste en poner pares de puntos (centinelas) a través de cada arista del GP (un centinela por cada sitio) con el objetivo de "garantizar" la arista. El número de centinelas necesarios para proteger una arista depende de su longitud y de su posición respecto a sus aristas vecinas, pues son situados a lo largo de las aristas para proteger, siempre a cierta distancia fijada por sus vecinas. Cada par de centinelas es situado sobre la circunferencia de algún círculo, cuyo centro yace en la arista y este no toca a ninguna otra arista. A continuación se detalla el proceso formalmente.

Supongan que se tiene un grafo planar $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ al que se le desea aplicar el algoritmo, este trabaja con dos fases fundamentales: primero se construyen círculos iniciales centrados en cada vértice de $\boldsymbol{G}$, luego se procede a cubrir cada arista $\boldsymbol{e} \in \boldsymbol{E}$ por círculos interiores no solapados cuyo centros tienen origen en $\boldsymbol{e}$.

Sea $\boldsymbol{u}$ un vértice de $\boldsymbol{G}$, y $\boldsymbol{\lambda}$ la longitud de la menor arista $\boldsymbol{G}$ incidente en $\boldsymbol{u}$. Denótese como $\boldsymbol{\xi_G(u)}$ a ambos lados de cada arista incidentes en $\boldsymbol{u}$, uno por cada lado de $\boldsymbol{e}$ (a una distancia adecuada $\boldsymbol{\varepsilon}$).

Sea $\boldsymbol{w}$ la intercepción entre $\boldsymbol{\xi_G(u)}$ y $\boldsymbol{e}$, ahora $\boldsymbol{p}$ y $\boldsymbol{q}$ protegen al segmento $\boldsymbol{\overline{uw}}$ de $\boldsymbol{e}$, lo que significa que $\boldsymbol{\overline{uw}}$ aparecerá en el DV que será construido, mientras no sean incluidos próximamente, nuevos puntos dentro de $\boldsymbol{\xi_G(u)}$.

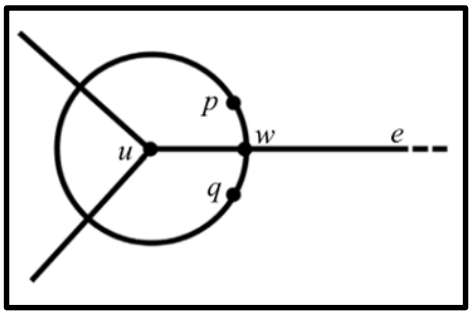


Fig 1.5.1.3a: Circulo inicial para el vértice $\boldsymbol{u}$ (Almaguer, y otros, 2007).

Sea $\boldsymbol{e} = \boldsymbol{\overline{uw}}$ una arista de $\boldsymbol{G}$, y $\boldsymbol{w_1}, \boldsymbol{w_2}$ los puntos de intersección de $\boldsymbol{\xi_G(u)}$ y $\boldsymbol{\xi_G(v)}$ con $\boldsymbol{e}$ respectivamente. Los segmentos $\boldsymbol{\overline{uw_1}}$ y $\boldsymbol{\overline{w_2v}}$ están ahora garantizados, mientras que el segmento $\boldsymbol{\overline{w_1w_2}}$ (quizás vacío) permanece desprotegido. Con el propósito de proteger $\boldsymbol{\overline{w_1w_2}}$ es necesario cubrirlo con círculos (interiores) centrados en él, mientras que estos no intercepten alguna arista distinta de $\boldsymbol{e}$ y no incluyan algún centinela perteneciente a otro círculo. Entonces se pueden escoger pares de centinelas en cada círculo a una distancia $\boldsymbol{\varepsilon}$ de $\boldsymbol{e}$.

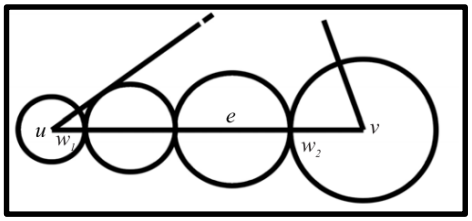


Fig 1.5.1.3b: Arista $\boldsymbol{e} = \boldsymbol{\overline{uw}}$ cubierta por círculos (Almaguer, y otros, 2007).

Como consecuencia del Lema 1 el segmento $\boldsymbol{e}$ queda protegido en toda su longitud, miestras que no sea incluido algún punto posterirmente dentro de cualquiera de los círculos centrados en $\boldsymbol{e}$ se acerquen a una distancia $\boldsymbol{\acute{\varepsilon}}$ de otra arista $\boldsymbol{f}$, pues los centinelas de $\boldsymbol{f}$ pueden caer dentro de algún círculo perteneciente a $\boldsymbol{e}$. Al tener en cuenta lo anterior se puede garantizar que los centinelas de $\boldsymbol{e}$ no interfieran con las aristas, ya que estos no van a estar incluidos en ningún otro círculo perteneciente a otra arista.

El algoritmo puede describirse de la siguiente manera:

Algoritmo PIVG

Entrada: Teselación del plano definida por un grafo planar $\boldsymbol{G = (V, E)}$

Salida: Conjunto de generadores de Voronoi $\boldsymbol{S}$

1. $\boldsymbol{S = \Phi}$.
2. **Para cada vértice** $\boldsymbol{u \in G}$

   Contruir el círculo $\boldsymbol{\xi_G(u)}$ **centrado en** $\boldsymbol{u}$**.**
3. **Escoger un valor adecuado para** $\boldsymbol{\varepsilon}$**.**
4. Para **cada vértice** $\boldsymbol{u \in V}$ **y cada arista** $\boldsymbol{e \in E}$ **incidente en** $\boldsymbol{u}$**.**

   Situar un par $\boldsymbol{(p, q)}$ **de centinelas en** $\boldsymbol{\xi_G(u)}$**, simétricos cada uno a** $\boldsymbol{e}$**, a una distancia** $\boldsymbol{\varepsilon}$

   $$\boldsymbol{S = S \cup \{p, q\}}$$
5. Para **cada arista** $\boldsymbol{e \in E}$

   Mientas **existan** $\boldsymbol{f}$ **desprotegidos en** $\boldsymbol{e}$

   Cubrir $\boldsymbol{f}$ **mediante círculos interiores creados en** $\boldsymbol{e}$ **(se sitúan los centinelas correspondientes a cada círculo y se agregan a** $\boldsymbol{S}$**). Situar un par** $\boldsymbol{(p, q)}$ **de centinelas en** $\boldsymbol{\xi_G(u)}$**, simétricos cada uno a** $\boldsymbol{e}$**, a una distancia** $\boldsymbol{\varepsilon}$**.**
6. Devolver $\boldsymbol{S}$ **y** Terminar

Este algoritmo, en efecto, muestra una estrategia general que brinda una variada gama de posibilidades a la hora de ejecutar los pasos 3 y 5. Con el objetivo de probar que el algoritmo funciona se debe demostrar que:

1. El algoritmo termina.
2. Una vez ejecutado toda arista de la teselación queda protegida.

Para mostrar que el algoritmo termina se deben dejar claros algunos aspectos. Sea $\boldsymbol{p_0}$ el radio, el menor círculo inicial; es claro (por definición de círculo inicial) que $\boldsymbol{p_0 > 0}$. Sea $\boldsymbol{\alpha}$ el menor de los ángulos formados por las aristas incidentes de $\boldsymbol{G}$, digase $\boldsymbol{e}$ y $\boldsymbol{f}$. Al tomar a $\boldsymbol{\varepsilon \leq p_0 sen(\alpha/2)}$ se puede estar seguros que dos pares de centinelas, pertenecientes, uno a $\boldsymbol{e}$ y otro a $\boldsymbol{f}$, no se mezclarán, lo que quiere decir, que los pertenecientes a $\boldsymbol{e}$ no estarán a una distancia menor que $\boldsymbol{\acute{\varepsilon}}$ de $\boldsymbol{f}$, lo que garantiza que un centinela de una arista no interfiera en otra. Esto es válido para todos los círculos iniciales.

Una vez que los círculos iniciales han sido construidos y sus correspondientes conjuntos de centinelas encontrados, si permanece algún segmento intermedio de alguna de las aristas, sin proteger, dicho segmento debe ser cubierto por un número finito de círculos interiores.

Tómese una arista, dígase $\boldsymbol{e}$, con un segmento intermedio sin cubrir de longitud $\boldsymbol{\delta}$, y un círculo de radio $\boldsymbol{\varepsilon}$ para cubrirlo, se puede asegurar que ninguno de ellos se intercepta con algún otro perteneciente a otra arista. Exactamente $[{}^{\boldsymbol{\delta}}/_{\mathbf{2}}\,\boldsymbol{\varepsilon}+\mathbf{1}]$ círculos son suficintes para cubrir completamente el segmento desprotegido, donde quizás el último de estos tendrá radio $\boldsymbol{p_1} < \boldsymbol{\varepsilon}$. Para este último los centinelas pueden ser situados a una distancia $\acute{\boldsymbol{\varepsilon}} < \boldsymbol{\varepsilon}$ de $\boldsymbol{e}$.

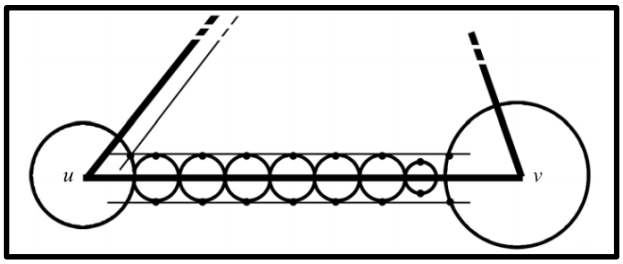


Fig 1.5.1.3c: Arista cubiertas por círculos de radio $\boldsymbol{\varepsilon}$ (Almaguer, y otros, 2007).

Usar círculos de radio $\boldsymbol{\varepsilon}$ es, entre las variantes a considerar, la que produce mayor cantidad de círculos y como consecuencia, de centinelas (generadores del DV). Al suponer que todos los círculos interiores tienen un radio igual a la $\boldsymbol{\varepsilon}^{-}$ vecindad escogida, si $\boldsymbol{\delta}$ es la longitud de $\boldsymbol{e}$ (mayor arista de $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$) y $\boldsymbol{p_0}$ el radio de menor círculo inicial construido en $\boldsymbol{G}$, $\boldsymbol{S}$ el conjunto de puntos (sitios) generados y $\boldsymbol{F}$ el conjunto de regiones de $\boldsymbol{G}$, entonces $\boldsymbol{e}$ puede cubrirse con $\left[\frac{\delta/_2 p_0}{2\varepsilon}\right]+\mathbf{1}$ círculos no solapados de $\boldsymbol{\varepsilon}$ radio, generándose $2\left(\left[\frac{\delta/_2 p_0}{2\varepsilon}\right]+\mathbf{1}\right)+\mathbf{4}$ puntos, si se generaliza este análisis entonces $|\boldsymbol{S}| \leq |\boldsymbol{E}|(\mathbf{2}\left(\left[\frac{\delta/_2 p_0}{2\varepsilon}\right]+\mathbf{1}\right)+\mathbf{4})$, pero se puede encontrar una cota aún inferior si se hace el análisis arista por arista: $|\boldsymbol{S}| \leq \mathbf{2}\sum_{\boldsymbol{e}\in\boldsymbol{E}}\left[\frac{\delta_e - p_0 - p_1}{2\varepsilon}\right]+\mathbf{6}|\boldsymbol{E}|$ donde $\boldsymbol{\delta_e}$ es la longitud de $\boldsymbol{e}$, $\boldsymbol{p_0}$ y $\boldsymbol{p_1}$ el radio de los círculos iniciales centrados en el vértice que define $\boldsymbol{e}$. Se puede asegurar ahora que $|F| \leq |S| \leq 2\sum_{\boldsymbol{e}\in\boldsymbol{E}}\left[\frac{\delta_e - p_0 - p_1}{2\varepsilon}\right]+\mathbf{6}|\boldsymbol{E}|$.

1. 6 Metodología de desarrollo de software

Las metodologías de desarrollo de software constituyen el conjunto de procedimientos, técnicas, herramientas y soporte documental que ayuda a los desarrolladores a realizar un nuevo software. Representan un marco de trabajo que tiene entre sus principales funciones: guiar, planificar, estructurar, controlar, manipular y dirigir el proceso de desarrollo de sistemas de información (Jacobson, y otros, 2000).

Se define AUP[13] como la metodología más adecuada para el desarrollo de la biblioteca. Esta se basa en la gestión de riesgos, al proponer que aquellos componentes con alto riesgo tengan más prioridad que los demás y sean desarrollados en etapas tempranas del proyecto. Desarrolla prototipos ejecutables

[13] *Agile Unified Process* (AUP, por sus siglas en inglés, Proceso Unificado Ágil)

durante la fase de elaboración del producto, demuestra la validez de la arquitectura para los requisitos claves del producto y determina los riesgos técnicos.

En AUP se establecen cuatro fases: Concepción, Elaboración, Construcción y Transición, que transcurren de manera consecutiva y que acaban con hitos claros alcanzados.

1. Concepción: Identificación del alcance y dimensión del proyecto, propuesta de la arquitectura y del presupuesto del cliente.
2. Elaboración: Confirmación de la idoneidad de la arquitectura.
3. Construcción: Desarrollo incremental del sistema, siguiendo las prioridades funcionales de los implicados.
4. Transición: Validación y despliegue del sistema.

Se elige esta metodología porque permite guiar proyectos de una complejidad y volumen no muy altos y que necesitan una rápida implementación, los cuales son los aspectos fundamentales a tener en cuenta para el desarrollo del producto a obtener. AUP proporciona un desarrollo del software rápido y eficiente, con una generación de artefactos media que satisface los requerimientos para la construcción de la biblioteca, a la vez que permite un ahorro de tiempo considerable. De igual forma se tiene en cuenta que es la metodología utilizada en los LPS[14] Aplicativos SIG, por lo que se debe mantener en aras de garantizar que los especialistas comprendan los artefactos generados y el proceso seguido en el desarrollo.

1. 7 Tecnologías a utilizar

1.7.1 Lenguaje de modelado

Además de lo esencial que resulta utilizar una metodología de software como hilo conductor de todo el ciclo de vida del sistema, también es importante tener en cuenta que en un proceso de desarrollo de software es necesario contar con algún elemento que describa el aspecto y la conducta del producto, estos elementos son llamados lenguajes de modelado.

UML[15] es uno de los lenguajes de modelado de gran utilidad para el desarrollo, pues ofrece un modo estándar de visualizar, especificar, construir y documentar los artefactos de un sistema (Jacobson, y otros, 2000). Se utiliza la versión 2.0 de este lenguaje para la realización de los entregables que propone la metodología seleccionada.

[14] LPS, Laboratorios de producción de software.

[15] *Unified Modeling Language* (UML, por sus siglas en inglés, Lenguaje Unificado de Modelado).

1.7.2 Herramienta de modelado

Con el objetivo de realizar una correcta planificación, diseño, implementación y documentación del proceso de desarrollo de la biblioteca se hace uso de herramientas CASE[16]. Estas herramientas son utilizadas con frecuencia para lograr una mejor productividad, eficiencia y eficacia de los productos informáticos, así como la reducción del tiempo de construcción de los mismos y posibles errores que estos puedan poseer.

Visual Paradigm en su versión 8.0, se considera una de las herramientas CASE más adecuada para realizar el proceso de modelado de un software. Esta propiedad es básicamente la que constituye un hecho determinante en su selección para ser utilizada en el modelado de la propuesta de solución. Posee una interfaz fácil de utilizar y es una herramienta CASE que soporta todo el ciclo de desarrollo del software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue.

1.7.3 Lenguaje de programación

Los lenguajes de programación son herramientas que permiten crear software. Están constituidos por un conjunto de símbolos, reglas sintácticas y semánticas que definen la estructura, el significado de los elementos y las expresiones de estos. Son utilizados para controlar el comportamiento físico y lógico de una máquina, permitiéndole a uno o más programadores especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo realizado a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, representado en forma simbólica y en manera de un texto los códigos que podrán ser leídos y escritos por una persona que no necesariamente posea muchos conocimientos sobre programación (Louden, 2004) (Lenguajes de programación, 2015).

C++

C++ es un lenguaje de programación derivado del C que soporta la programación orientada a objetos y la estructurada. Fue creado a mediados de los años 80 por Bjarne Stroustrup con el fin de agregar funcionalidades y características de las que carecía su antecesor, mantiene ventajas en cuanto a riqueza de operadores, expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original. Actualmente es uno de los lenguajes más potentes a nivel mundial, en gran medida esto se debe a que es un lenguaje compilado. Desde el punto de vista de los lenguajes orientados a objetos, C++ es un lenguaje híbrido y existen varios IDE que lo soportan como: NetBeans, Eclipse, Qt Creator, entre otros (Lenguajes de programación, 2015) (Louden, 2004) (Programación en Castellano, 2015).

[16] *Computer Aided Software Engineering* (CASE, por sus siglas en inglés, Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software.

Debido a las necesidades existentes, se decide seleccionar un lenguaje de programación rápido y eficiente con el cual el equipo de desarrollo se encuentre familiarizado y que brinde las condiciones óptimas para la construcción exitosa de la biblioteca, se decide utilizar como lenguaje de programación C++.

Para su elección se tuvieron en cuenta varias características dentro de las que se encuentran (Rodríguez, 2011):

1. Es considerado por expertos como el lenguaje más potente para el trabajo con el hardware de la máquina.
2. En tiempo de ejecución, supera a su equivalente en otros lenguajes como C# y Java.
3. Es un lenguaje de propósito general, con el que los desarrolladores alcanzan acceso a recursos primarios como solo este lenguaje puede ofrecer.
4. Potencia un mejor rendimiento de la caché de las aplicaciones al permitir que estas sean más rápidas que las desarrolladas en otros lenguajes de programación.

1.7.4 Marco de trabajo

En el proceso de desarrollo de software un marco de trabajo, también definido como infraestructura digital o *framework*, es una estructura de software conceptual y tecnológica compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. Puede ser considerado como una aplicación genérica incompleta y configurable a la que se puede añadir las últimas piezas para construir una aplicación concreta. Puede incluir lenguaje interpretado, bibliotecas, soporte a programas y otras herramientas para ayudar a desarrollar y unir los diferentes componentes de un proyecto (Gutiérrez, 2012) (Rodríguez, 2011).

Dentro de los objetivos principales que persigue un *framework* se encuentran:

- Acelerar el proceso de desarrollo.
- Reutilizar código ya existente.
- Promover buenas prácticas de desarrollo como el uso de patrones.

QT

Qt es un *framework* de desarrollo multiplataforma para aplicaciones con interfaz gráfica de usuario o de consola que viene acompañado de un conjunto de herramientas para facilitar su uso. Incluye una biblioteca de clases intuitiva, integra herramientas de desarrollo y un IDE multiplataforma. Permite realizar aplicaciones avanzadas, desplegarlas en escritorios y sistemas operativos integrados sin tener que reescribir el código fuente, contribuyendo de esta forma con el tiempo y el costo de desarrollo del producto. Es fácil de usar, aprender, mantener y de código reutilizable, además posee un alto

rendimiento en tiempo de ejecución y ocupa poco espacio en disco. Posee sus propias herramientas, ideales para los programadores, como es el caso de Qt Creator para C++ y Qt Designer para el desarrollo de interfaces de usuario (Garrido, 2009).

Algunas de las características asumidas a la hora de seleccionar este marco de trabajo son:

1. Cuenta con abundante documentación y posee una herramienta para visualizar la misma, Qt Assistant (Rodríguez, 2011).
2. Internacionalización de aplicaciones con la ayuda de la herramienta Qt Linguist (Rodríguez, 2011).
3. Hasta el momento ha sido liberado bajo dos licencias: LGPL y comercial.
4. Incluye un *framework* de animación que ayuda a construir animaciones fluidas, interfaces gráficas de alto rendimiento sin la molestia de la gestión de estructuras complejas y temporizadores (Rodríguez, 2011).

1.7.5 Entorno de desarrollo integrado

Un IDE[17], es un programa informático compuesto por un conjunto de herramientas de programación, que proveen facilidades a los programadores para escribir códigos y agilizar el proceso de desarrollo de software. Pueden ser aplicaciones por sí solas o ser parte de aplicaciones existentes, además permiten ser utilizadas tanto con un único lenguaje de programación como con varios de estos. Las herramientas que normalmente componen un entorno de desarrollo integrado son las siguientes: editor de texto, compilador, intérprete, herramientas para la automatización, depurador, diseñador para la construcción de interfaces gráficas de usuario y, opcionalmente, un sistema de control de versiones (Álvarez, 2010) (Rodríguez, 2011) (IDE, 2015).

Para la selección del entorno integrado de desarrollo a utilizar en la construcción de la biblioteca se definieron los siguientes parámetros a cumplir:

1. Facilidad de uso y abundante documentación.
2. Compatibilidad con GNU\Linux y Windows.
3. Integración con el *framework* de desarrollo escogido para la implementación de la biblioteca.
4. Soporte para el lenguaje C++.
5. Integración con bibliotecas de Qt.

[17] IDE, en inglés, *Integrated Development Environment.*

Qt Creator

Qt Creator es un entorno integrado de desarrollo creado por Trolltech y diseñado para hacer que el desarrollo en C++ de la aplicación Qt sea más rápido y fácil. Está basado en la biblioteca Qt, biblioteca multiplataforma de interfaces gráficas de usuario. Pensado especialmente para el desarrollo en varias plataformas como: Windows, Linux y Mac OSX. Adaptado a las necesidades de los desarrolladores, permite crear aplicaciones de escritorio y plataformas de dispositivos móviles. Proporciona dos editores visuales integrados: Qt Designer para la creación de interfaces de usuario y Qt Quick Designer para el desarrollo de interfaces de usuario con el lenguaje QML (Rodríguez, 2011).

Algunas de las características que se destacan de este IDE son (Programación en Castellano, 2015):

1. Avanzado editor de código C++.
2. Tiene integrado diseñadores de interfaz gráfica de usuario.
3. Herramientas para la administración de proyectos.
4. Ayuda sensible al contexto.
5. Depurador visual.
6. Resaltado y autocompletado de código.

1. 8 Conclusiones parciales

Se evidencia cómo a pesar de existir distintas herramientas para realizar el particionado del plano, estas no resuelven el problema planteado inicialmente por lo que se hace beneficioso el desarrollo de la biblioteca para su uso en futuras aplicaciones. Con la investigación de las características y ventajas que proporcionan las herramientas y tecnologías para el desarrollo de software, dígase ventajas del Qt Designer para crear las interfaces de usuario y las bondades de C++ como lenguajes de programación, quedan sentadas las bases teóricas y tecnológicas, de gran ayuda en el diseño e implementación de la biblioteca.

Capítulo 2. Descripción de la biblioteca de algoritmos para la obtención de particionados 2D y su problema inverso

2. 1 Introducción del capítulo

En este capítulo se realiza una descripción de la propuesta a defender, en vista a solucionar la situación actual en el campo de acción. Se muestran los requisitos, tanto los funcionales como los no funcionales, y los casos de uso generados a partir de dichos requisitos con sus respectivas especificaciones. La propuesta se elabora con el objetivo de facilitar los particionados del plano y agilizar el desarrollo de aplicaciones informáticas que requieran de su uso, ejemplo de ellas los SIG.

2. 2 Modelo conceptual

Luego de un exhaustivo análisis y debido a no tener una estructura definida de los procesos de negocio (fronteras bien establecidas, donde se logren ver claramente quiénes son las personas que lo inician, quiénes son los beneficiados, pero además quiénes son las personas que desarrollan las actividades en cada uno de estos procesos), se plantea un modelo de dominio. Se realiza a través de un diagrama de clases de UML al identificar los conceptos representados en el diagrama en un glosario de términos; de manera tal que todo el interesado adquiera un entendimiento mínimo del contexto en que se emplaza la biblioteca.

2.2.1 Diagrama de clases del modelo de dominio

Según (Jacobson, y otros, 2000) un modelo de dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las cosas que existen o los eventos que suceden en el entorno en el que trabaja el sistema. Todo ello se representa a través de clases relacionadas, mediante el lenguaje UML, con el objetivo de tener una mejor comprensión de la estructura y dinámica de la organización, los problemas actuales dentro de esta, e identificar las mejoras potenciales.

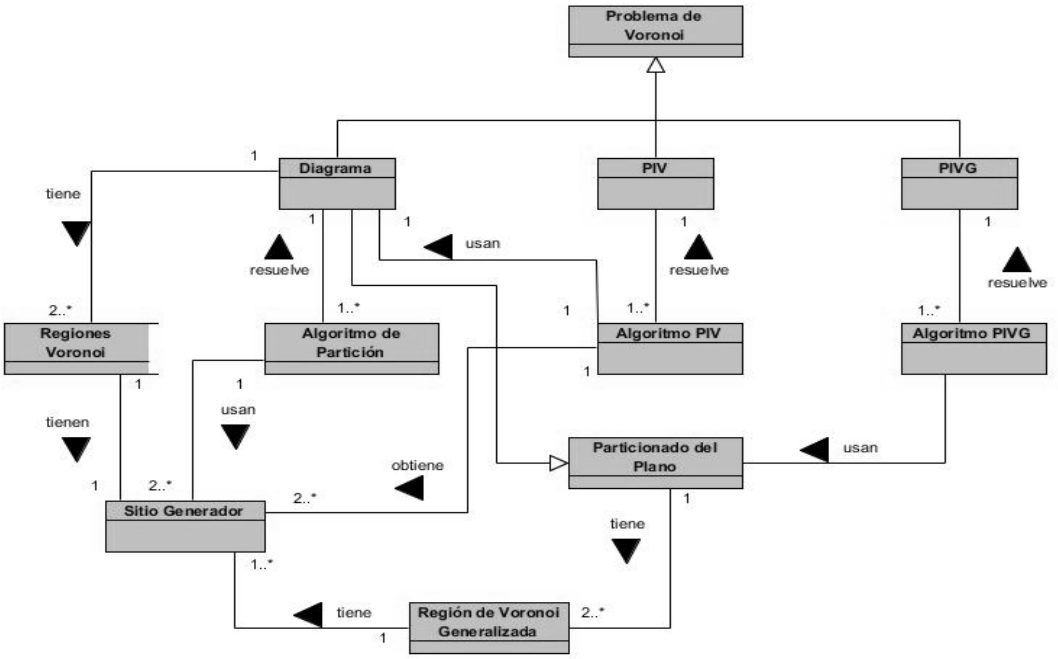


Fig 2.1: Diagrama de clases del dominio.

2.2.2 Análisis de los conceptos del diagrama de clases del dominio

Mediante un breve glosario de términos se definen los conceptos identificados en el diagrama de dominios, estos son:

Tabla 2.2.2. Conceptos del diagrama de clases del dominio.

Conceptos	Descripción
Problema de Voronoi	Particionado del plano donde cada punto se encuentra dentro de una región específica.
Diagrama	Elemento que está formado por sitios, caras y aristas.
PIV	Problema Inverso de Voronoi.
PIVG	Problema Inverso Generalizado de Voronoi.
Regiones de Voronoi	Conjunto de particiones en el plano que contienen el punto que las generó.
Algoritmo de partición	Conjunto de reglas y procedimientos que hacen posible el particionado del plano.
Algoritmo PIV	Conjunto de reglas y procedimientos que dan solución al Problema Inverso de Voronoi.
Algoritmo PIVG	Conjunto de reglas y procedimientos que dan solución al Problema Inverso Generalizado de Voronoi.
Particionado del plano	Divisiones que se realizan sobre una superficie plana.
Sitio Generador	Punto en el plano mediante el que se genera una región de Voronoi.
Región de Voronoi generalizada	Región de una partición del plano que puede ser generada a partir de uno o varios puntos generadores.

2. 3 Solución Propuesta

Como solución propuesta se tiene la creación de una biblioteca para obtener el particionado 2D y su problema inverso (BP2DIN), que agrupe algoritmos que permitan realizar el particionado del plano y la resolución de su problema inverso. La BP2DIN es el resultado de la unión de un conjunto de algoritmos, cada uno encargado de solucionar los temas antes mencionados. Esta distribución se hace con el objetivo de ganar en simplicidad y portabilidad de la BP2DIN.

2. 4 Descripción de las funcionalidades del sistema propuesto

La especificación de requisitos del software es una descripción completa del comportamiento del sistema a desarrollar. Incluye un conjunto de casos de uso que describen todas las interacciones que se estima el usuario tendrán con el software. Contiene requisitos no funcionales (o suplementarios). Estos son los requisitos que imponen restricciones al diseño o funcionamiento del sistema (tal como requisitos de funcionamiento, estándares de calidad, o requisitos del diseño).

2.4.1 Requisitos funcionales del sistema

Los requisitos funcionales describen las funcionalidades que se quiere realice un sistema informático, facilitan el mecanismo apropiado para saber lo que quiere el cliente, llegar a un acuerdo y llevar a cabo el proceso de construcción del *software* sin que existan ambigüedades (Pressman, 2015).

Tabla 2.4.1. Requisitos funcionales.

Referencia	Requisito funcional
R 1	**Obtener el particionado del plano dado un conjunto de puntos iniciales.**
R 1.1	Obtener el particionado del plano dado un conjunto de puntos iniciales mediante el algoritmo de Voronoi.
R 1.1.1	Obtener el particionado del plano dado un conjunto de puntos iniciales mediante el algoritmo Voronoi con peso.
R 1.1.2	Obtener el particionado del plano dado un conjunto de puntos iniciales mediante el algoritmo Voronoi ponderado.
R 1.2	Obtener el particionado del plano dado un conjunto de puntos iniciales mediante el algoritmo de Thiessen.
R 1.3	Obtener el particionado del plano dado un conjunto de puntos iniciales mediante el algoritmo de Dirichlet.
R 1.4	Obtener el particionado del plano dado un conjunto de puntos iniciales mediante el algoritmo Delaunay.
R 1.5	Obtener el particionado del plano dado un conjunto de puntos iniciales mediante el algoritmo de Fortune.
R 1.6	Obtener el particionado del plano dado un conjunto de puntos iniciales mediante el algoritmo de Laguerre.
R 2	**Obtener el conjunto de puntos generadores dado el Diagrama de Voronoi.**

R 2.1	Obtener un conjunto de puntos generadores dado el particionado del plano mediante el algoritmo inverso de Voronoi.
R 2.1.1	Obtener un conjunto de puntos generadores dado el particionado del plano mediante el algoritmo inverso de Voronoi con peso.
R 2.1.2	Obtener un conjunto de puntos generadores dado el particionado del plano mediante el algoritmo inverso de Voronoi ponderado.
R 2.2	Obtener un conjunto de puntos generadores dado el particionado del plano mediante el algoritmo inverso de Thiessen.
R 2.3	Obtener un conjunto de puntos generadores dado el particionado del plano mediante el algoritmo inverso de Dirichlet.
R 2.4	Obtener un conjunto de puntos generadores dado el particionado del plano mediante el algoritmo inverso de la Triangulación Delaunay.
R 2.5	Obtener un conjunto de puntos generadores dado el particionado del plano mediante el algoritmo inverso de Fortune.
R 2.6	Obtener un conjunto de puntos generadores dado el particionado del plano mediante el algoritmo inverso de Laguerre.
R 3	**Obtener el conjunto de puntos generadores dado el particionado del plano.**
R 3.1	Obtener un conjunto de puntos generadores dado el Diagrama de Voronoi mediante el algoritmo inverso generalizado de Voronoi.

2.4.2 Requisitos no funcionales del sistema

Los requisitos no funcionales describen las cualidades, características y propiedades que el producto a construir debe tener. Estos se clasifican en diferentes categorías, algunas de estas son:

Requerimiento de usabilidad

La herramienta será utilizada por programadores, el producto debe estar concebido para que el usuario piense qué desea hacer, y no cómo hacerlo.

Requerimiento de Soporte

Para garantizar el soporte de esta herramienta se crea un manual de usuario con el objetivo de esclarecer posibles dudas que puedan surgir durante con la misma. Además de un manual de uso y trabajo con los algoritmos de particionado del plano, pues el usuario no tiene que ser un especialista en este tema.

Requerimiento de portabilidad

La biblioteca propuesta puede ser utilizada en sistemas operativos como Windows o distribuciones GNU/Linux.

Requerimiento de rendimiento

Se debe realizar una biblioteca de algoritmos eficiente y precisa, que permita dar respuesta en un tiempo que oscile entre 0 y 32 segundos, para conjuntos de datos menores a 14 500 puntos.

Requerimiento legal

La biblioteca se desarrolla bajo la licencia GNU/GPL.

2. 5 Modelación del sistema

El diagrama de casos de uso del sistema contiene actores, casos de uso del sistema y sus relaciones; describe el comportamiento del sistema al ser utilizado por un usuario y bajo qué restricciones. Permite que los desarrolladores y clientes lleguen a un acuerdo sobre los requerimientos, y proporciona la entrada fundamental para el análisis, diseño y las pruebas. Documenta el comportamiento de un software desde el punto de vista del usuario. Por tanto los requisitos funcionales del sistema determinan los casos de uso, los cuales representan las funcionalidades que un sistema puede ejecutar (Wesley, 2004). La Tabla 2.5 muestra la prioridad definida para cada uno de los CUS.

Tabla 2.5. Prioridad de Casos de Uso.

Caso de uso del Sistema	Prioridad
Obtener particionado.	Crítico
Obtener generadores PIV.	Crítico
Obtener generadores PIVG.	Crítico

2.5.1 Determinación y justificación de los actores de sistema

Un actor no es parte del sistema, es un agente que interactúa con el sistema que se construye, rol de un usuario. Puede intercambiar información o puede ser un recipiente pasivo de información y representa a un ser humano, a un software o a una máquina que se relaciona con el sistema (Pressman, 2015). En este caso con el sistema interactúa un único actor ver Tabla 2.5.1.

Tabla 2.5.1. Descripción del actor del sistema.

Actor	Justificación
Usuario	Representa a un sistema (ejemplo: SIG) o a un programador que utiliza la BP2DIN.

2.5.2 Diagrama de casos de uso del sistema

Los diagramas de casos de uso del sistema (DCUS) se utilizan para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Una relación es una conexión entre los elementos del modelo, por ejemplo la especialización y la generalización son relaciones. Los DCUS se utilizan para ilustrar los requisitos del sistema al mostrar cómo reacciona una respuesta a eventos que se producen en el mismo (Pressman, 2015).

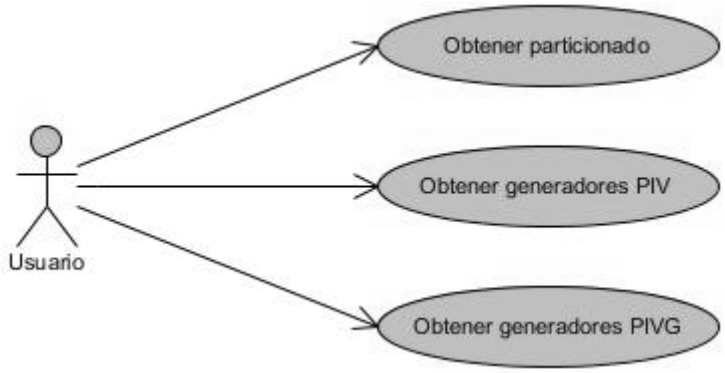


Fig 2.2: Diagrama de casos de uso del sistema.

2.5.3 Descripción extendida de los casos de uso del sistema

Tabla 2.5.3a. Descripción del CUS Obtener particionado.

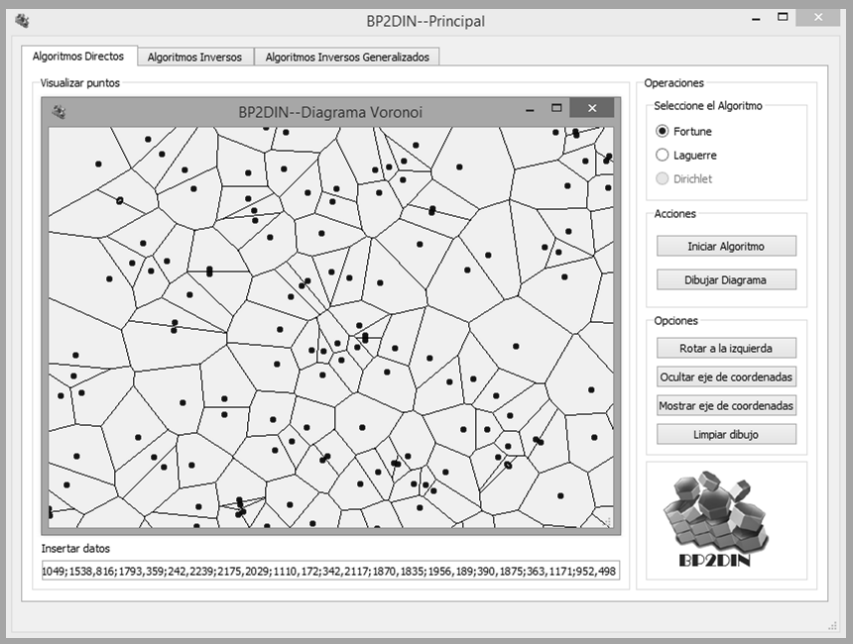
Caso de Uso:	Obtener particionado
Actores:	Usuario
Propósito:	Obtener el particionado mediante el DV.
Descripción:	El caso de uso inicia cuando el usuario selecciona la opción obtener el particionado del plano, luego inserta los valores necesarios según el método escogido. El caso de uso termina cuando el sistema valida los datos además de calcular y mostrar el particionado del plano.
Complejidad:	Media.
Prioridad:	Crítico.
Referencia:	RF 1.1 RF 1.1.1 RF 1.1.2 RF 1.2 RF 1.3 RF 1.4 RF 1.5 RF 1.6

Precondiciones:	
Poscondiciones:	El sistema calcula el particionado del plano.

Flujo normal de eventos

	Acción del actor	Respuesta del sistema
1.	El caso de uso inicia cuando se selecciona la opción calcular el particionado del plano por el método de Fortune.	
2.	Inserta los valores y da clic en el botón “Iniciar Algoritmo”.	
3.		Valida los valores introducidos.
4.	Da clic en el botón “Dibujar Diagrama”.	
5.		El sistema calcula y muestra el particionado del plano.

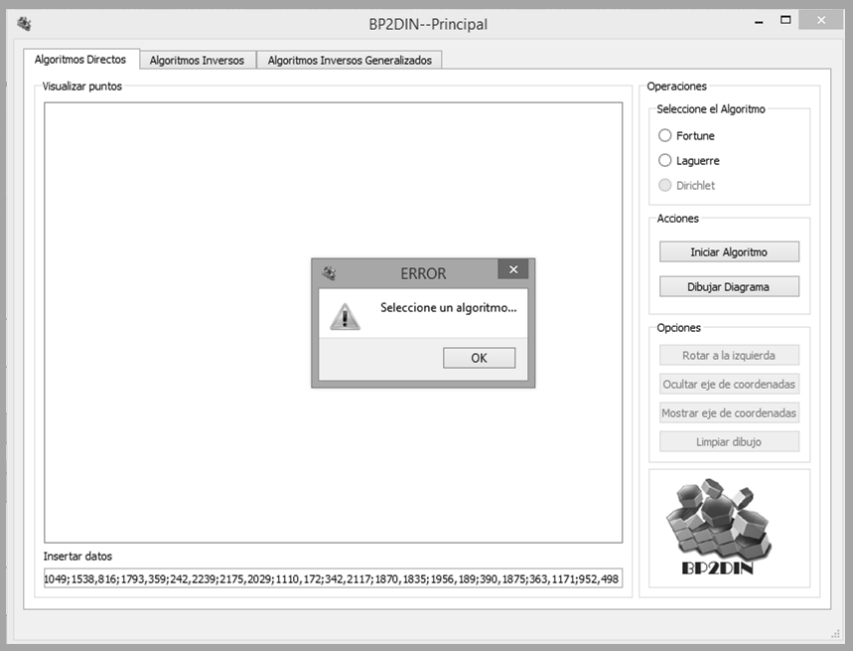
Prototipo de interfaz 1.1



Flujo alterno de eventos “Algoritmo no seleccionado”

	Acción del actor	Respuesta del sistema
1.1	El caso de uso inicia cuando no se selecciona la opción calcular el particionado del plano por el método de Fortune.	
2.1	Inserta los valores y da clic en el botón “Iniciar Algoritmo”.	
3.1		El sistema muestra un mensaje de error “Seleccione un algoritmo…”.

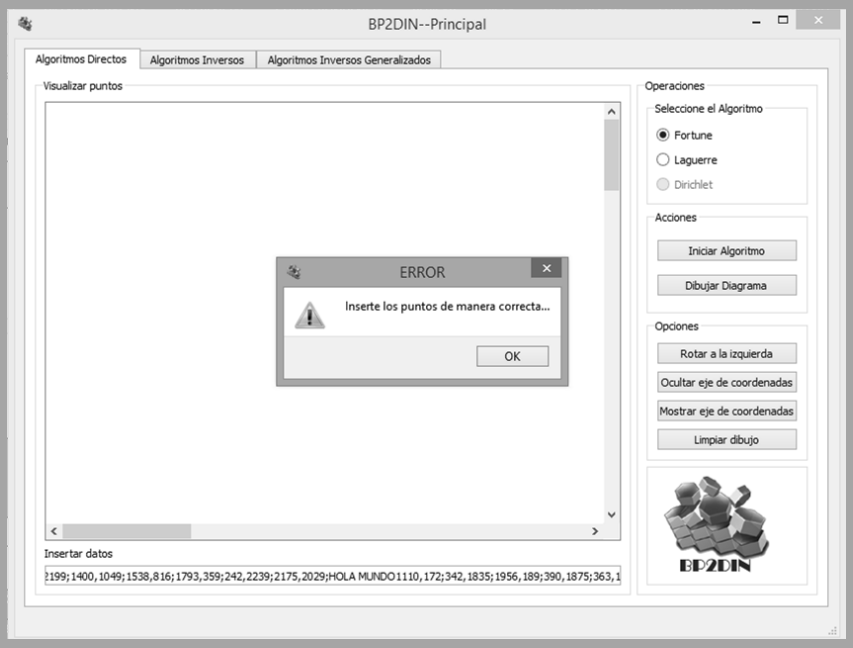
Prototipo de Interfaz 1.2



Flujo alterno de eventos "Datos incorrectos"

	Acción del actor	Respuesta del sistema
2.1.1	El actor inserta los datos de forma incorrecta y da clic en el botón "Iniciar Algoritmo".	
2.2.1		El sistema valida los datos y muestra un mensaje de error "Inserte los punto de manera correcta…".

Prototipo de interfaz 1.3



Flujo alterno de eventos "Datos vacíos"		
2.1.2	El actor no inserta los datos y da clic en el botón "Iniciar Algoritmo".	
2.1.2		El sistema valida los datos y muestra un mensaje de error "Debe introducir los puntos…".

Prototipo de interfaz 1.4

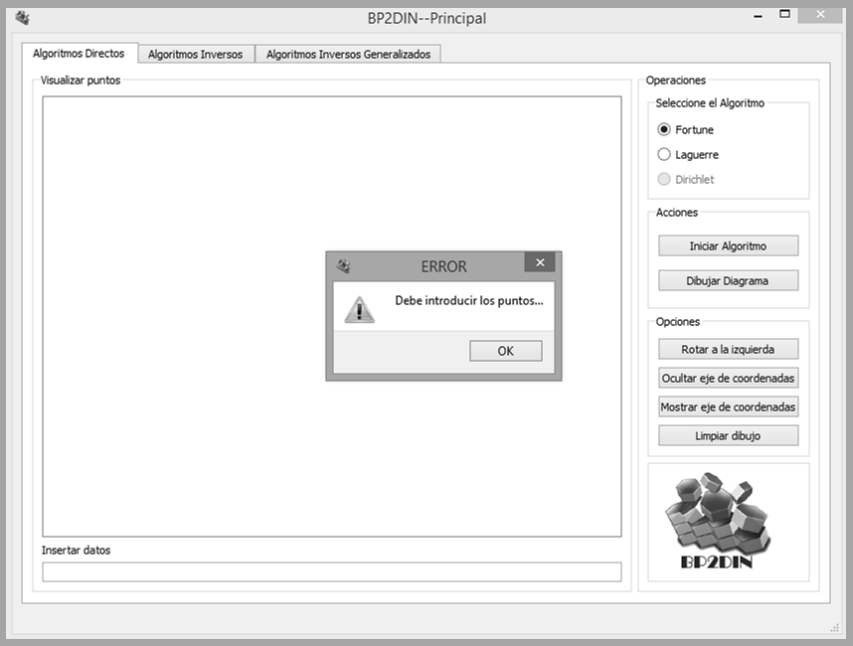


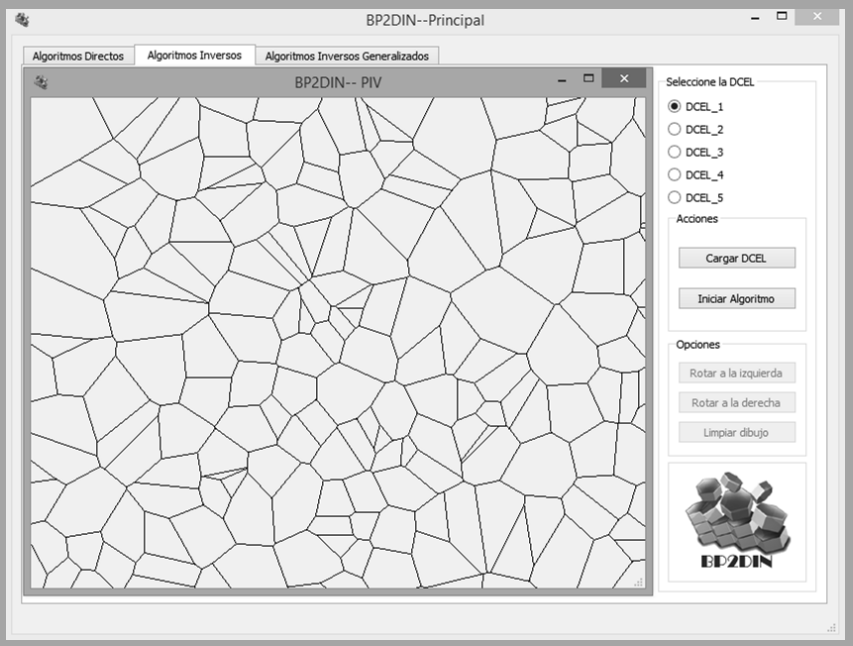
Tabla 2.5.3b. Descripción del CUS Obtener generadores PIV.

Caso de Uso:	Obtener generadores PIV.
Actores:	Usuario.
Propósito:	Obtener los generadores para un particionado del plano dado.
Descripción:	El caso de uso inicia cuando el usuario selecciona la opción obtener los generadores, luego selecciona los valores necesarios según el método escogido. El caso de uso termina cuando el sistema valida los datos y calcula los posibles generadores del particionado.
Complejidad:	Alta.
Prioridad:	Crítico.
Referencia:	RF 2.1
Precondiciones:	
Poscondiciones:	El sistema calcula los posibles generadores del particionado.

Flujo normal de eventos		
	Acción del actor	**Respuesta del sistema**

1.	El caso de uso inicia cuando se selecciona el particionado deseado y da clic en el botón “Cargar DCEL”.	
2.		El sistema comprueba que la DCEL haya sido cargada satisfactoriamente.
3.	Da clic en el botón “Iniciar Algoritmo”.	
4.		El sistema calcula y muestra los posibles generadores del particionado.

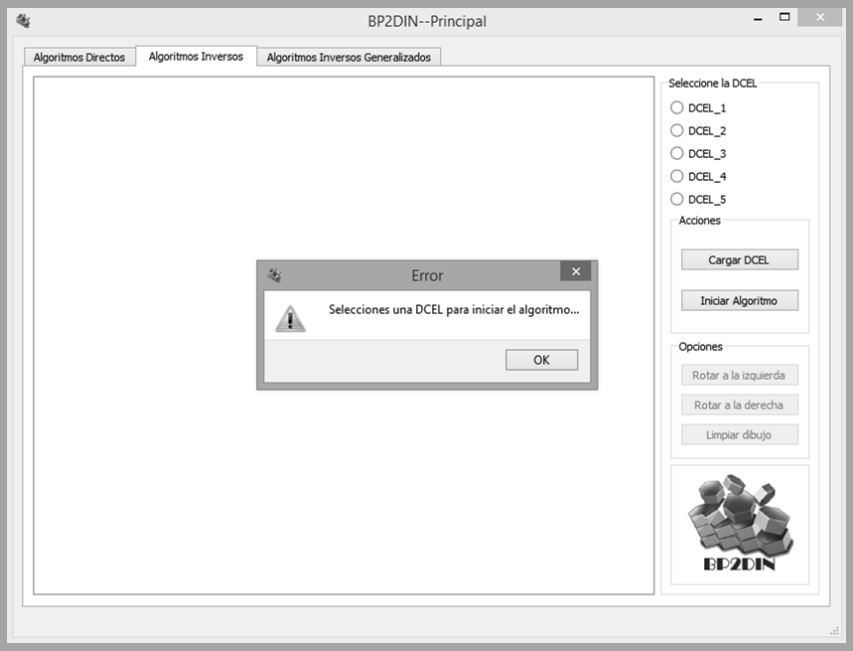
Prototipo de interfaz 2



Flujo alterno de eventos “DCEL no seleccionada”

	Acción del actor	Respuesta del sistema
1.1	El caso de uso inicia cuando no se selecciona un particionado y da clic en el botón “Cargar DCEL”.	
2.1		El sistema muestra un mensaje de error “Selecciona una DCEL para iniciar el algoritmo…”.

Prototipo de interfaz 2.1



Flujo alterno de eventos "DCEL no seleccionada"

	Acción del actor	Respuesta del sistema
1.2	El caso de uso inicia cuando no se selecciona un particionado y da clic en el botón "Iniciar Algoritmo".	
2.2		El sistema muestra un mensaje de error "Antes de iniciar el algoritmo seleccione un DCEL y cárguela…".

Prototipo de interfaz 2.2

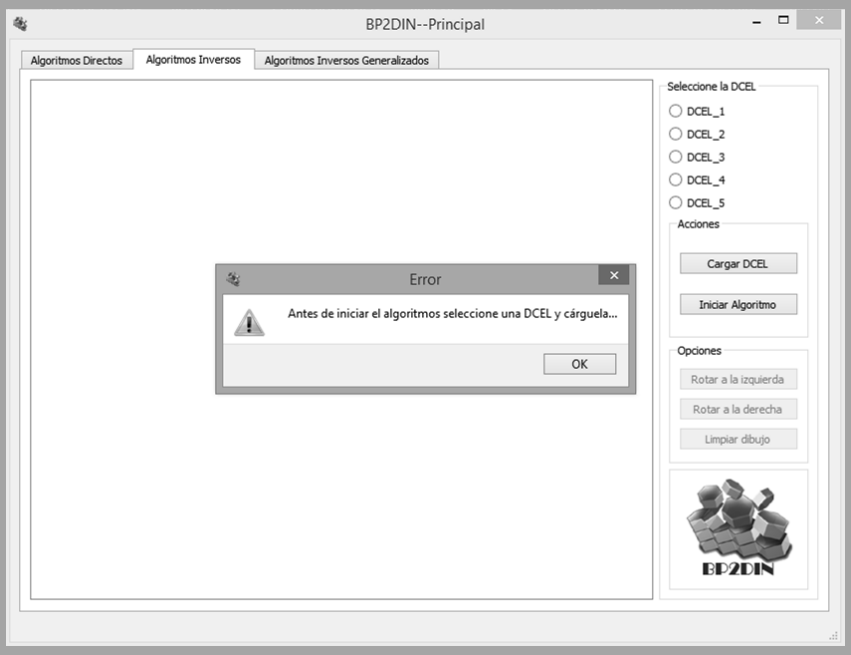
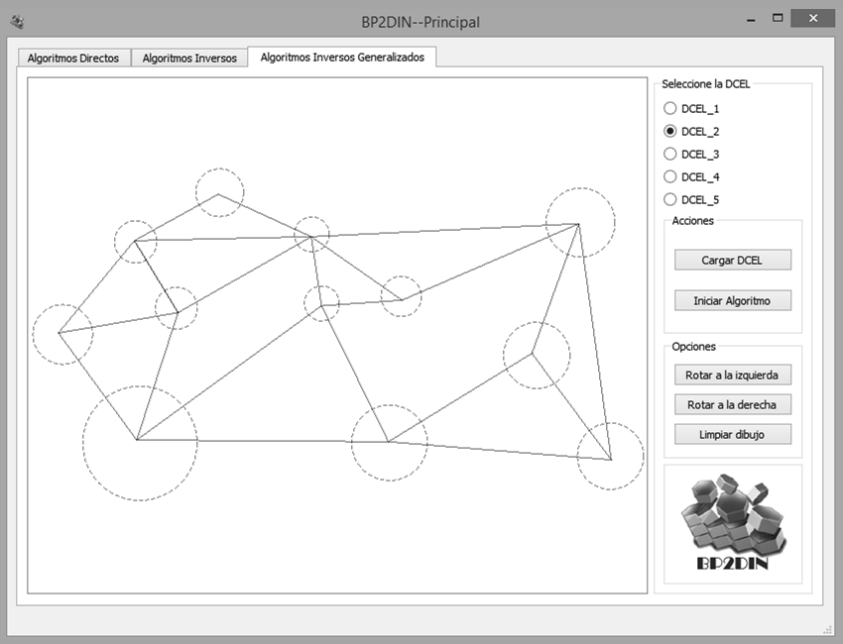


Tabla 2.5.3c. Descripción del CUS Obtener generadores PIVG.

Caso de Uso:	Obtener generadores PIVG.
Actores:	Usuario.
Propósito:	Obtener los generadores para cualquier particionado dado.
Descripción:	El caso de uso inicia cuando el usuario selecciona la opción obtener los generadores, luego escoge los valores necesarios según el método escogido. El caso de uso termina cuando el sistema valida los datos y calcula los posibles generadores del particionado.
Complejidad:	Alta.
Prioridad:	Crítico.
Referencia:	RF 3.1 RF 3.1.1 RF 3.1.2 RF 3.2 RF 3.3 RF 3.4 RF 3.5 RF 3.6
Precondiciones:	
Poscondiciones:	El sistema calcula los posibles generadores del particionado.

Flujo normal de eventos		
	Acción del actor	**Respuesta del sistema**
1.	El caso de uso inicia cuando se selecciona el particionado deseado y da clic en el botón “Cargar DCEL”.	
2.		El sistema comprueba que la DCEL haya sido cargada satisfactoriamente.
3.	Da clic en el botón “Iniciar Algoritmo”.	
4.		El sistema calcula y muestra los posibles generadores del particionado.

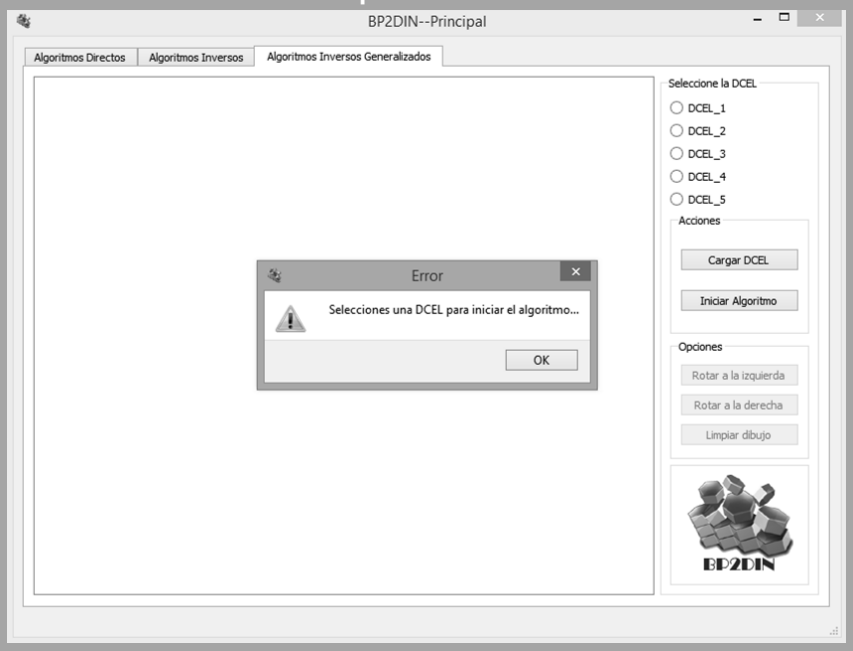
Prototipo de interfaz 3



Flujo alterno de eventos "DCEL no seleccionada"

	Acción del actor	Respuesta del sistema
1.1	El caso de uso inicia cuando no se selecciona un particionado y da clic en el botón "Cargar DCEL".	
2.1		El sistema muestra un mensaje de error "Selecciona una DCEL para iniciar el algoritmo…".

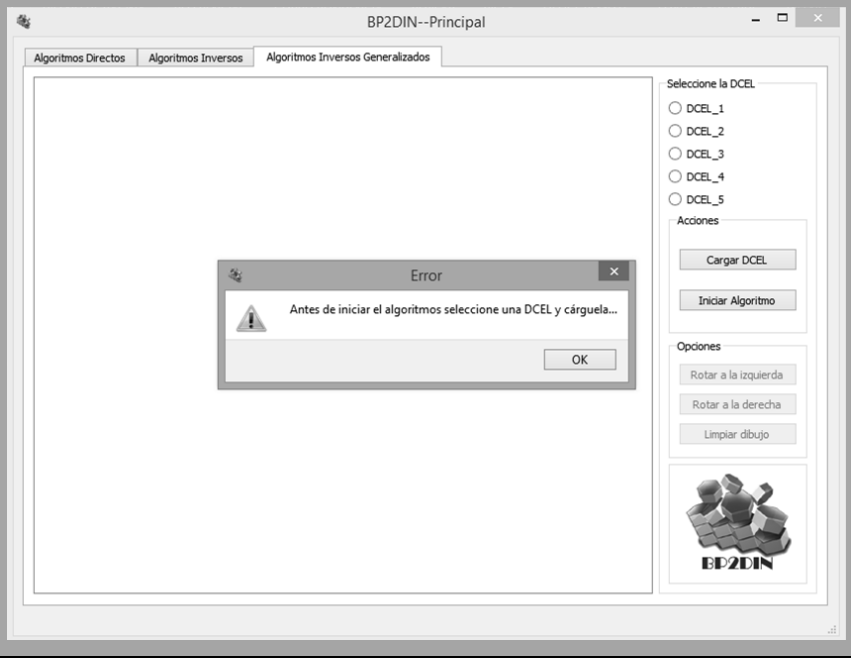
Prototipo de interfaz 3.1



	Acción del actor	Respuesta del sistema

1.2	El caso de uso inicia cuando no se selecciona un particionado y da clic en el botón “Iniciar Algoritmo”.	
2.2		El sistema muestra un mensaje de error “Antes de iniciar el algoritmo seleccione un DCEL y cárguela…”.

Prototipo de interfaz 3.2



2. 6 Conclusiones parciales

A partir de la utilización de las herramientas y la metodología escogida en el capítulo anterior se dio paso a la presentación de la solución propuesta, al realizar un esbozo detallado y un mayor acercamiento al objetivo general propuesto en la investigación. La obtención del modelo de dominio permitió tener una mejor comprensión de los conceptos asociados al objeto de estudio, se listaron los requisitos funcionales y no funcionales del sistema, además de determinar quién es al actor que interactuará con la BP2DIN y detallar los casos de uso que satisfacen los requerimientos del sistema.

Capítulo 3. Implementación de la biblioteca de algoritmos para la obtención de particionados 2D y su problema inverso

3. 1 Introducción del capítulo.

Al iniciar la construcción de un sistema de software, luego de conocer cuáles son las necesidades básicas del usuario final, el ambiente donde será desarrollado el mismo, las condiciones para ello, las tecnologías y recursos necesarios para lograr un ambiente de trabajo adecuado, entonces se sientan las bases del esqueleto arquitectónico del sistema. La arquitectura de software es la estructura de las estructuras del sistema, la cual comprende los componentes de software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos (Pressman, 2015).

El presente capítulo muestra los artefactos ingenieriles relacionados con el diseño, implementación y validación de la biblioteca a desarrollar. Entre los principales elementos a mostrar se encuentran la arquitectura de software seleccionada y el modelo de diseño, incluye además los patrones arquitectónicos y de diseño utilizados respectivamente. De igual manera se incluye el modelo de implementación y las pruebas realizadas al módulo.

3. 2 Arquitectura de software

La definición de arquitectura de software más usada, asumida como oficial y adoptada también por grandes compañías desarrolladoras de software, es la que brinda el estándar de la IEEE[18] que expresa lo siguiente: "*La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución"* (IEEE, 2000).

De forma general la arquitectura de software es un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un software, ofrece a los programadores, analistas y todo el conjunto de desarrolladores del software la posibilidad de compartir una misma línea de trabajo y cubrir todos los objetivos y restricciones de la aplicación. Es considerada como el nivel más alto en el diseño de la arquitectura de un sistema, puesto que establece la estructura, funcionamiento e interacción entre las partes del software (Pressman, 2015).

3.2.1 Patrones arquitectónicos

Un patrón arquitectónico impone una transformación en el diseño de la arquitectura, su alcance es más específico pues se concentra en un aspecto, en lugar de hacerlo en toda la arquitectura. Un patrón aplica una regla sobre la arquitectura, describe la manera en que el software maneja alguna característica de su funcionalidad al nivel de la infraestructura, abarca puntos específicos del comportamiento dentro del

[18] *Institute of Electrical and Electronics Engineers* (IEEE, por sus siglas en inglés, Instituto de Ingeniería Eléctrica y Electrónica). Es una asociación mundial de técnicos e ingenieros dedicada a la estandarización y el desarrollo en áreas técnicas

contexto de la arquitectura y es usado para determinar la forma de la estructura general de un sistema (Pressman, 2015).

Arquitectura en Capas

La arquitectura en capas soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite la partición de un problema complejo en una secuencia de pasos incrementales. Brinda una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente inferior. Esta arquitectura admite optimizaciones y refinamientos, además de proporcionar una amplia reutilización (Reynoso, y otros, 2004).

Casos representativos de esta arquitectura son muchos de los protocolos de comunicación en capas. En ellos cada capa proporciona un sustrato para la comunicación a algún nivel de abstracción, y los niveles más bajos suelen estar asociados con conexiones de hardware. El ejemplo más característico es el modelo OSI con los siete niveles: nivel físico, vínculo de datos, red, transporte, sesión, presentación y aplicación. El estilo también se encuentra en forma más o menos pura en arquitecturas de bases de datos y sistemas operativos.

Arquitectura Orientada a Objetos (AOO)

Los nombres alternativos para esta arquitectura han sido Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de esta son los objetos, o más bien instancias de los tipos de datos abstractos. En la caracterización clásica de David Garlan y Mary Shaw, los objetos representan una clase de componentes que ellos llaman *managers*, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante es la representación interna de un objeto, ya que este no es accesible desde otros. Entre sus características se puede destacar que los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son las unidades de modelado, diseño e implementación, los objetos y sus interacciones son el centro de las responsabilidades en el diseño de la arquitectura y en la estructura de la aplicación (Pressman, 2015), (Ecured, 2016).

Arquitectura de la BP2DIN

Para el desarrollo de la biblioteca se realiza la unión entre las arquitecturas antes mencionadas y para obtener de esa manera lo mejor de ambas. Por lo que se decide establecer una arquitectura conformada por dos capas, la primera denominada interfaz de comunicación y la segunda lógica de negocio de la BP2DIN, al ser la segunda capa mucho más compleja que la primera, se divide en subsistemas que se encargan del funcionamiento de la misma. Los subsistemas se organizan y comunican mediante la AOO.

3. 3 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el módulo, al ser la principal vía de acceso en la actividad de implementación (Jacobson, y otros, 2000).

3.3.1. Diagrama de paquetes

En el lenguaje UML, se modelan diferentes diagramas para presentar el diseño de la aplicación, entre los que se encuentra el diagrama de paquetes. Un diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas, muestra las dependencias entre esas agrupaciones (Pressman, 2015). Estos diagramas suministran una descomposición de la jerarquía lógica de un sistema. Los paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada uno y minimizar el acoplamiento externo entre ellos. Se propone de esta forma, el diagrama de paquetes para la BP2DIN.

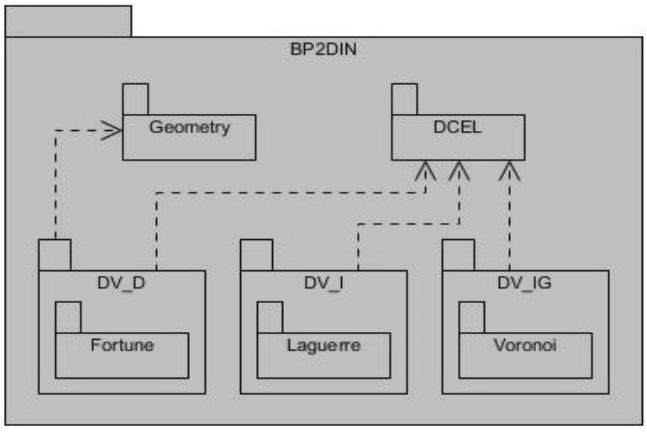


Fig 3.3.1: Diagrama de paquetes.

Breve descripción de los paquetes de la BP2DIN

Geometry/ contiene un conjunto de clases necesarias para trabajar con objetos geométricos.

DCEL/ contiene todas las clases que componen la estructura de datos DCEL.

DV_D/Fortune/ contiene todas las clases que componen al algoritmo Fortune.

DV_I/Laguerre/ contiene todas las clases que componen al algoritmo Laguerre.

DV_IG/Voronoi/ contiene todas las clases que componen al algoritmo Voronoi.

3.3.2. Diagrama de clases del diseño

Un diagrama de clases del diseño representa una abstracción de una o varias clases en la implementación del sistema. Responden al cumplimiento de los requisitos funcionales, son subproductos

del modelo de diseño que proporcionan una perspectiva estática que representa el diseño estructural del sistema, estos muestran un conjunto de clases y sus atributos (Larman, 2003).

Los diagramas de clases del diseño se emplean en el modelado de las vistas del diseño del sistema para describir las especificaciones de las interfaces y las clases del software, lo que facilita el trabajo de los desarrolladores. Tomándose como partida lo antes expuesto, se propone el siguiente diagrama de clases del diseño para el caso de uso Obtener particionado.

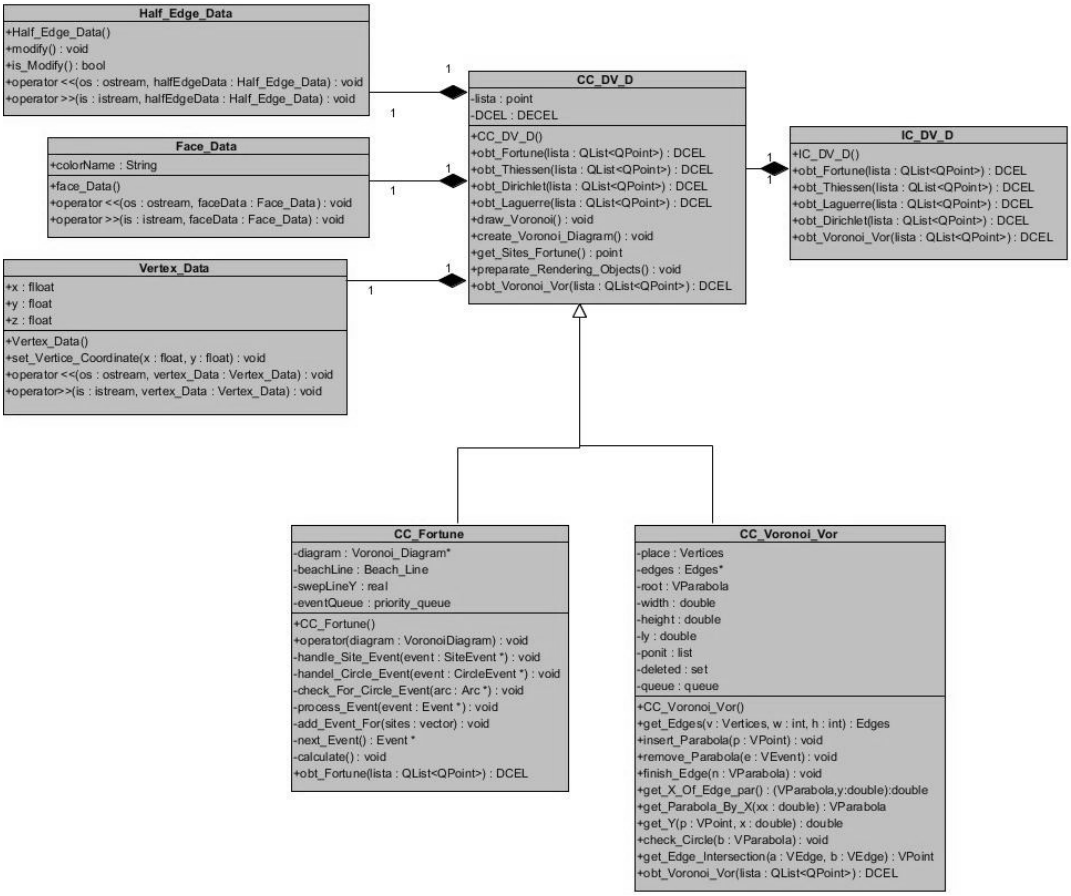


Fig 3.3.2: Diagrama de clases del diseño. CU Obtener particionado.

3.3.3. Patrones de diseño

Para el desarrollo de software un elemento fundamental y de buena práctica, lo constituye el uso de patrones. Representan una excelente manera de trabajar en base a la calidad. Además, un patrón de diseño es una buena práctica documentada o solución, que se ha aplicado con éxito en múltiples ambientes para erradicar problemas comunes de diseño de software, con una probada efectividad y con características de reutilización (Fowler, 2007).

Patrones GRASP[19]

Experto: Se encarga de asignar responsabilidades a las clases que cuentan con la información necesaria para efectuar la tarea que tiene encomendada. En la clase DCEL_Mesh se evidencia el uso del patrón, ya que dicha clase es la única que contiene la información para gestionar todo lo referente a dicha estructura de datos.

Creador: Es quien guía el proceso de asignación de responsabilidades relacionadas con la creación de objetos. Tiene el objetivo de asignarle a la clase B la responsabilidad de crear una instancia de la clase A, la intención de este patrón es encontrar un creador que necesite conectarse al objeto creado en alguna situación. Es utilizado durante la construcción de la biblioteca al ser necesario crear en una clase determinada una instancia de otra, por ejemplo en la clase CC_DV_IG_Voronoi se utiliza este patrón en la creación del objeto VUtil, (ver Anexo 4).

Controlador: Responsable de asignar a clases específicas la responsabilidad de controlar el flujo de eventos en el sistema, al facilitar la centralización de las actividades desarrolladas y permitir interactuar mediante estas con las demás clases de la biblioteca. La utilización de este patrón se evidencia en la clase CC_DV_D.

Patrón GoF[20]

***Layers*:** este patrón se utiliza con el objetivo de organizar en capas el modelo de diseño, los componentes de una capa solo pueden hacer referencia a componentes de capa inmediatamente inferiores. En la aplicación se evidencian las dos capas utilizadas, para delimitar las interfaces de comunicación y la lógica de negocio.

3.3.4. Modelo de implementación

El modelo de implementación describe cómo los elementos del modelo de diseño se implementan en términos de componentes (base de datos, ejecutables, módulos o ficheros). De igual manera describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados y cómo dependen los componentes unos de otros (Jacobson, y otros, 2000).

En el presente trabajo de diploma el modelo de implementación muestra la transición del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .cpp debido a su implementación en C++ para el caso del diagrama de componentes de código fuente (fig. 3.2.4a) y para

[19] *General Responsibility Assignment Software Patterns* (GRASP, por sus siglas en inglés, Patrones generales de software para asignar responsabilidades)

[20] *GoF (Gang of Four, en español: Pandilla de los Cuatro) se clasifican en 3 categorías basadas en su propósito: creacionales, estructurales y de comportamiento.*

el diagrama de código compilado (fig. 3.2.4b) se traducen en los ficheros .lib y .a para las bibliotecas estáticas y .dll y .so para las dinámicas.

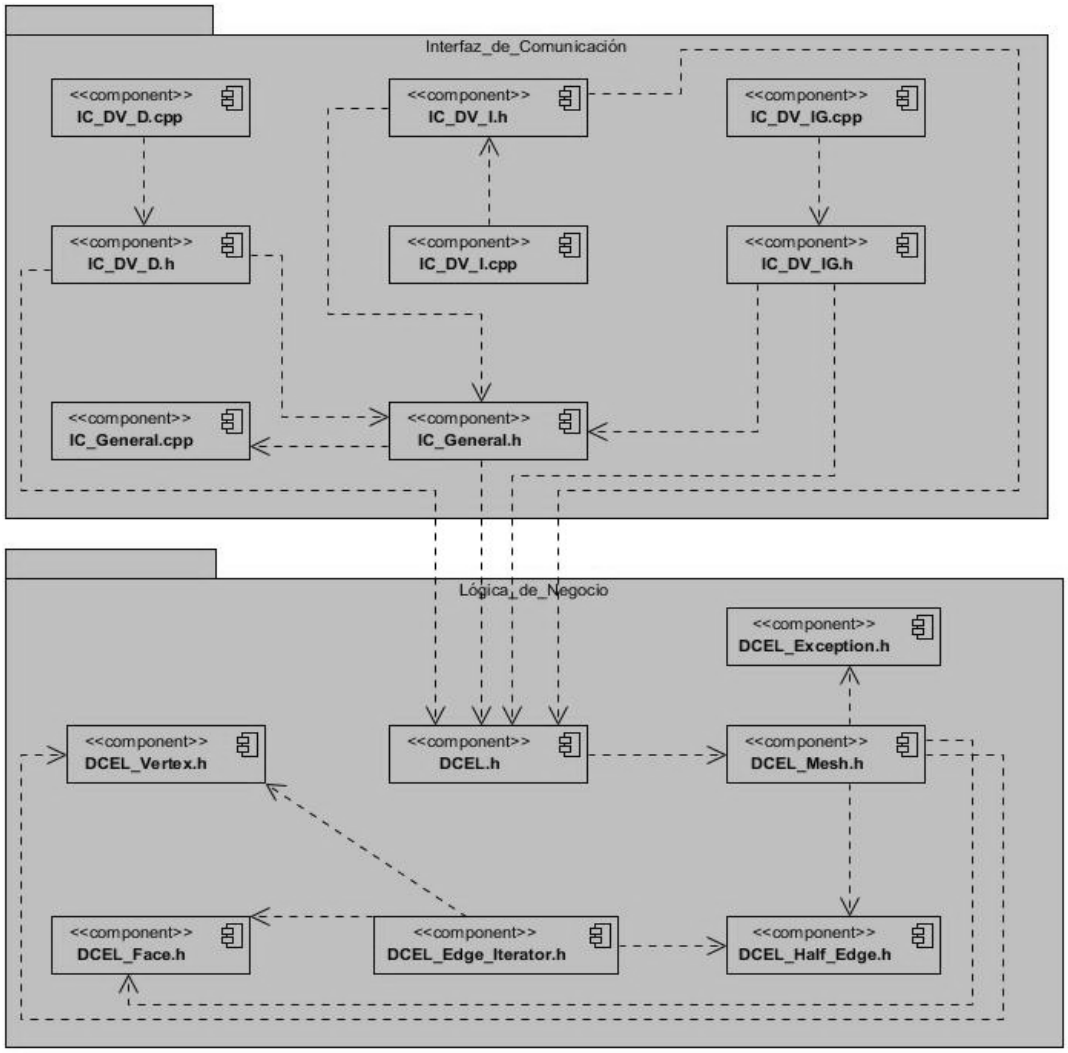


Fig 3.2.4a: Diagrama de componente del código fuente.

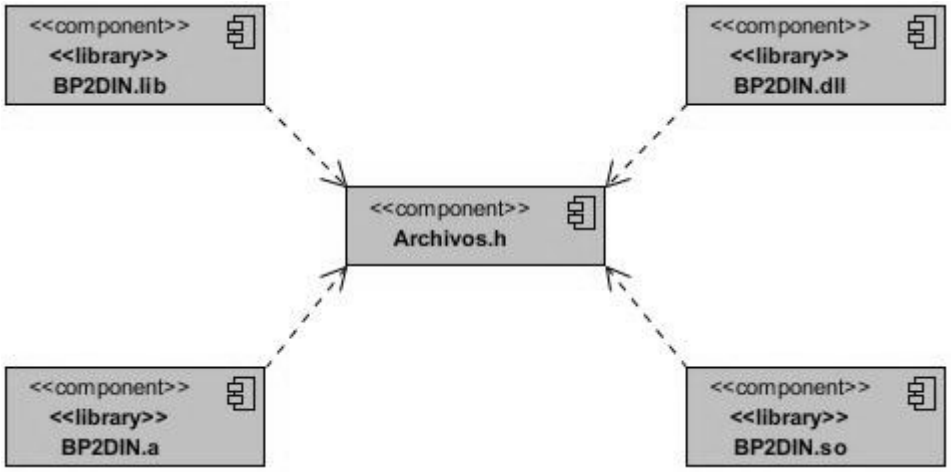


Fig 3.2.4b: Diagrama de componentes del código compilado.

3. 4 Verificación y validación

La verificación y validación es el nombre que se da a los procesos de comprobación y análisis que aseguran que el software que se desarrolla está acorde a su especificación y cumple las necesidades de los clientes (Drake, y otros, 2009). La validación y la verificación son términos diferentes, el papel de la verificación es comprobar que el software está realizado de acuerdo con su especificación. Se comprueba que el sistema cumple los requisitos funcionales y no funcionales que se han definido anteriormente. La validación es un proceso más general, pues se debe asegurar que el software cumple las expectativas del cliente. Va más allá de comprobar si el sistema está acorde con su especificación, para probar que el software hace lo que el usuario espera a diferencia de lo que se ha especificado.

Dentro del proceso de verificación y validación se utilizan dos técnicas de comprobación y análisis de sistemas, las inspecciones de software y las pruebas del software. La primera es una técnica estática que analiza y comprueba las representaciones del sistema como el documento de requisitos, los diagramas de diseño y el código fuente del programa. La segunda es una técnica dinámica que consiste en comprobar las respuestas de una implementación del software a series de datos de prueba y examinar las respuestas del mismo junto a su comportamiento operacional, para comprobar que se desempeñe conforme a lo requerido (Drake, y otros, 2009), (Pruebas_de_software, 2005).

Las técnicas estáticas solo pueden comprobar la correspondencia entre un programa y su especificación y no puede probar que el software es de utilidad operacional, ni si las características no funcionales del software son las correctas. Por lo tanto, para validar un sistema de software, es recomendable llevar a cabo ambos tipos de prueba.

3.4.1. Pruebas de software

Las pruebas de software son un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación. Las pruebas constituyen el último bastión desde el que se puede evaluar la calidad y, de forma más pragmática, descubrir los errores. Pero las pruebas no deben ser vistas como una red de seguridad (Pressman, 2015).

El único instrumento adecuado para determinar el status de la calidad de un producto software es el proceso de pruebas. En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos (Pruebas_de_software, 2005).

Se realizaron varias pruebas a la BP2DIN con el fin de lograr un producto de alta calidad, las mismas arrojaron que el sistema se encuentra con un desempeño aceptable de acuerdo a los requisitos especificados.

3.4.1.1. Pruebas de Caja Blanca o Estructurales

Con el objetivo de llevar a cabo la Estrategia de Pruebas de Unidad, que compruebe el correcto funcionamiento de la BP2DIN, entendida como una unidad funcional independiente y comprobar su correcta codificación, se realizan en esta investigación las pruebas de Caja Blanca o Estructurales.

A este tipo de prueba se le conoce también como Técnicas de Caja Transparente o de Cristal. Se centra en diseñar los casos de prueba fijándose en el comportamiento interno y la estructura del programa, examina la lógica interna del programa sin considerar los aspectos de rendimiento. El objetivo es diseñar casos de prueba que se ejecuten al menos una vez en todas las sentencias del programa, y todas las condiciones tanto en los casos verdaderos como en los falsos. Existen distintos criterios de cobertura lógica, que permiten decidir qué sentencias o caminos se deben examinar con los casos de prueba. El criterio de cobertura de caminos asegura que los casos de prueba diseñados permiten que todas las sentencias del programa sean ejecutadas al menos una vez y que sean probadas las condiciones para ambos casos.

La prueba de camino básico es una de las técnicas empleadas para aplicar este criterio. Propuestas inicialmente por Tom McCabe. El método del camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar una media como guía para la creación de un conjunto de caminos básicos de ejecución. Los casos de prueba obtenidos garantizan que durante la prueba se ejecuta al menos una vez cada sentencia del programa (Pressman, 2015).

Diseño de casos de prueba de Caja Blanca

A continuación se muestra la prueba de camino básico aplicada al método **construir_Circulos** de la clase **CC_DV_IG_Voronoi** de la BP2DIN, para la que se siguieron los siguientes pasos lógicos:

```
QList<DV_IG_Voronoi::circulos> DV_IG_Voronoi::construir_Circulos(MyDCEL dcel)
{
    QList<vertice> vertexPoints;
    QList<circulos> result;
    circulos circ;
    QPoint punto;
    for (int i=0; i< dcel.getNumVertices(); i++)
    {

        MyDCEL::Vertex* vertex = dcel.getVertex(i);
        MyDCEL::EdgeIterator iterator(vertex);

        vertice x1;
        x1.first = vertex->getData().x;
        x1.second= vertex->getData().y;

        while(iterator.hasNext())
        {
            MyDCEL::Vertex* vecinos = iterator.getNext()->getTwin()->getOrigin();
            vertice x2;
            x2.first = vecinos->getData().x;
            x2.second = vecinos->getData().y;
            vertexPoints.append(x2);
        }

        QList<float> distancias;

        for (int var = 0; var < vertexPoints.size(); var++)
        {
        float distance = sqrt(pow(((vertexPoints.at(var).first)- (x1.first)),2)
                         +pow(((vertexPoints.at(var).second)-(x1.second)),2));
        distancias.append(distance);
        }

        qSort(distancias);
        punto.setX(vertex->getData().x);
        punto.setY(vertex->getData().y);
        circ.first.append(punto);
        circ.second.append(distancias.at(0));
        result.append(circ);
        vertexPoints.clear();

    }

   return result;
}
```


Fig 3.5.2: Método construir_Circulos de la clase CC_DV_IG_Voronoi.

El método recibe por parámetros una DCEL donde está almacenada toda la información referente al particionado analizado. Selecciona los datos necesarios de la DCEL para situar los círculos en cada vértice del particionado. Retorna una estructura con el radio y las coordenadas del centro de cada círculo.

Representación del método en un grafo de flujo.

El grafo de flujo se utiliza para representar el flujo de control lógico del método. Para ello se utilizan los tres elementos siguientes:

- Nodos: representan cero, una o varias sentencias en secuencia. Cada nodo comprende como máximo una sentencia de decisión.
- Aristas: líneas que unen dos nodos.
- Regiones: áreas delimitadas por aristas y nodos.

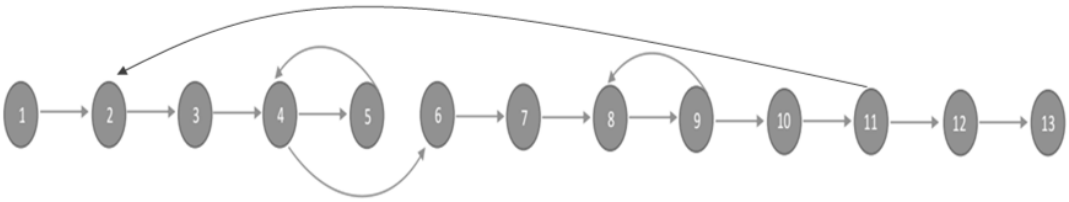


Fig 3.5.2.1: Grafo de flujo del método construir_Circulos de la case CC_DV_IG_Voronoi.

Cálculo de la complejidad ciclomática

Existen varias formas de calcular la complejidad ciclomática $\boldsymbol{V(G)}$ de un método a partir de un grafo de flujo G, las cuales deben arrojar el mismo resultado para asegurar que el cálculo de la complejidad sea correcto.

- $\boldsymbol{V(G) = Aristas - Nodos + 2}$
  $\boldsymbol{V(G) = 15 - 13 + 2}$
  $\boldsymbol{V(G) = 4}$
- $\boldsymbol{V(G) = Nodos\ de\ predicado + 1}$
  $\boldsymbol{V(G) = 3 + 1}$
  $\boldsymbol{V(G) = 4}$
- $\boldsymbol{V(G) = N}$úmero de regiones del grafo
  $\boldsymbol{V(G) = 4}$

Conjunto básico de caminos independientes.

El valor de la complejidad ciclomática da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Por tanto, a continuación se identifica el conjunto básico de caminos independientes.

Camino 1: 1-2-3-4-6-7-8-9-10-11-12-13
Camino 2: 1-2-3-4-5-4-6-7-8-9-10-11-12-13
Camino 3: 1-2-3-4-6-7-8-9-8-9-10-11-12-13
Camino 4: 1-2-3-4-5-4-6-7-8-9-8-9-10-11-12-13
Estos 4 caminos constituyen el camino básico para el grafo de flujo correspondiente.

Casos de prueba que fuerzan la ejecución de cada camino

A cuatro de los caminos obtenidos mediante la prueba de camino básico se le realiza un caso de prueba.

Número del camino: 1
Caso de prueba:
Caso de prueba:
DCEL = 0 vértices
Resultado esperado: No se construyeron los círculos.
Resultado de la prueba: Satisfactorio

Número del camino: 2
Caso de prueba:
DCEL = 7 vértices

Resultado esperado: Se construyeron los círculos centrados en cada vértice con radio ($\boldsymbol{r} = \mathbf{menos\ de\ las\ aristas\ /2}$).

Resultado de la prueba: Satisfactorio.

Número del camino: 3

Caso de prueba:

DCEL = 5 vértices

Resultado esperado: Se construyeron los círculos centrados en cada vértice con radio ($\boldsymbol{r} = \mathbf{menos\ de\ las\ aristas\ /2}$).

Resultado de la prueba: Satisfactorio.

Número del camino: 4

Caso de prueba:

DCEL = 15 vértices

Resultado esperado: Se construyeron los círculos centrados en cada vértice con radio ($\boldsymbol{r} = \mathbf{menos\ de\ las\ aristas\ /2}$).

Resultado de la prueba: Satisfactorio.

3.4.1.2. Pruebas de Caja Negra

Las pruebas de caja negra son las que se llevan a cabo sobre la interfaz del software. Examina algunos aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica del software. Las pruebas de caja negra también denominadas pruebas de comportamiento, se centran en los requisitos funcionales del software, esta prueba no es una alternativa a las técnicas de pruebas de caja blanca. Se trata más bien de otro enfoque que intenta descubrir diferentes tipos de errores que los métodos de caja blanca (Pressman, 2015).

Para confeccionar los casos de prueba de caja negra existen distintos criterios, el seleccionado para realizar las pruebas a la BP2DIN es el de Particiones de equivalencia; (Pressman, 2015) presenta la partición de equivalencia como un método de prueba que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba.

A continuación se muestra la descripción de las pruebas aplicadas al caso de uso Obtener particionado, las pruebas aplicadas al resto de los casos de uso pueden ser consultadas en el documento “Biblioteca de algoritmos para el particionado 2D y su problema inverso. Diseño de casos de prueba”.

Diseño del caso de prueba para el CUS Obtener particionado

Descripción general

El caso de uso inicia después que el usuario solicita obtener el particionado del plano directo, inserta los datos necesarios según el método escogido. El sistema valida los datos y calcula el particionado del plano.

Casos de prueba

Sección (SC): Obtener particionado.

Tabla 3.5.1.2a. Caso de prueba del CUS Obtener particionado.

Escenario	Descripción	Insertar datos	Seleccione algoritmo	Respuesta del sistema	Flujo central
EC 1.1: Obtener particionado correctamente.	El usuario selecciona el algoritmo deseado e inserta los puntos en el campo de texto. Luego da clic en el botón “Iniciar algoritmo” y posteriormente en el botón “Dibujar diagrama”.	V	V	El sistema dibuja en la pantalla los puntos y muestra el particionado del plano.	Seleccionar pestaña “Algoritmos Directos”. Seleccionar el algoritmo. Insertar los puntos deseados. Clic en el botón “Iniciar algoritmo”. Clic en el botón “Dibujar diagrama”.
		1,2;5,6;10,20	Fortune		
EC 1.2: Obtener particionado sin entrar datos.	El usuario selecciona el algoritmo deseado, no introduce los puntos y da clic en el botón “Iniciar algoritmo”.	I	V	El sistema muestra una notificación “Debe introducir los puntos…”.	Seleccionar pestaña “Algoritmos Directos”. Seleccionar el algoritmo. Clic en el botón “Iniciar algoritmo”.
		N/A	Fortune		
		V	I		

EC 1.3: Obtener particionado al insertar los datos correctamente y sin seleccionar el algoritmo deseado.	El usuario no selecciona el algoritmo e introduce los puntos y da clic en el botón "Iniciar algoritmo".	1,2;5,6;10,20	N/A	El sistema muestra una notificación "Seleccione un algoritmo…".	Seleccionar pestaña "Algoritmos Directos". Insertar los puntos deseados. Clic en el botón "Iniciar algoritmo".
EC 1.4: Obtener particionado al insertar datos incorrectos y con un algoritmo seleccionado.	El usuario selecciona el algoritmo deseado e introduce los datos de forma incorrecta y luego da clic en el botón "Iniciar algoritmo".	I	V	El sistema muestra una notificación "Inserte los puntos correctamente…".	Seleccionar pestaña "Algoritmos Directos". Seleccionar el algoritmo. Insertar los puntos deseados para el algoritmo. Clic en el botón "Iniciar algoritmo". Clic en el botón "Dibujar diagrama".
EC 1.5: Obtener particionado al dejar los campos vacíos.	El usuario no selecciona el algoritmo deseado y no introduce datos, luego da clic en el botón "Iniciar algoritmo".	I N/A	I N/A	El sistema muestras una notificación "Seleccione un algoritmo…".	Seleccionar pestaña "Algoritmos Directos". Clic en el botón "Iniciar algoritmo".

Descripción de las variables

Tabla 3.5.1.2b. Variables del CUS Obtener particionado.

No	Nombre de campo	Clasificación	Valor nulo	Descripción
1	Insertar datos	Campo de texto (textField)	No	Solo admite cadenas de caracteres compuestas por números, separados por "," y ";".
2	Seleccionar algoritmo	Campo de selección (radioButton)	No	Permite seleccionar el algoritmo deseado para la obtención del particionado.

3.4.1.3. Pruebas de rendimiento

Las pruebas de rendimiento están diseñadas para probar el rendimiento de un software en tiempo de ejecución dentro del contexto de un sistema integrado. Estas se realizan para determinar la velocidad de procesamiento de una o varias tareas en un sistema con condiciones particulares de trabajo. Son utilizadas también para validar y verificar algunos atributos en calidad del sistema: escalabilidad, fiabilidad y uso de los recursos. Las pruebas de rendimiento son un subconjunto de la ingeniería de pruebas, una práctica informática que se esfuerza por mejorar el rendimiento, englobándose en el diseño y la arquitectura de un sistema, antes incluso del esfuerzo inicial de la codificación (Pressman, 2015).

Pruebas de Carga

Las pruebas de carga se realizan generalmente para observar el comportamiento de un software bajo una cantidad de operaciones esperada. Esta carga puede ser el máximo número de datos que debe poder procesar el software al realizar un número específico de procedimientos durante el tiempo que dura la carga. Esta prueba puede mostrar los tiempos de respuesta de todas las operaciones de sistema en cuestión, además puede ayudar a detectar un posible cuello de botella en el software (Molyneaux, 2009).

Las pruebas fueron realizadas en 3 escenarios con características diferentes de software y hardware, se realizaron un total de 5 iteraciones por algoritmo con juegos de datos que oscilaron entre 0 y 15 000 puntos. Estas arrojaron los siguientes resultados:

Escenario 1, CPU Intel Core i3-2328M CPU @2.20GHz, 4GB RAM, sistema operativo Windows 8.1 x64. Se obtuvo como resultado para los algoritmos directo, inverso e inverso generalizado una respuesta promedio de 25.57, 15.00, 18.01 segundos.

Escenario 2, CPU AMD A8 @1.9GHz, 8GB RAM, sistema operativo Windows 8 x64. Se obtuvo como resultado para los algoritmos directo, inverso e inverso generalizado una respuesta promedio de 23.00, 13.00, 17.02, segundos.

Escenario 3, CPU Intel Core i7-3328M CPU @2.4GHz, 8GB RAM, sistema operativo Xubuntu 16.04 LTS x64. Se obtuvo como resultado para los algoritmos directo, inverso e inverso generalizado una respuesta promedio de 18.00, 9.20, 13.01 segundos.

A continuación se esbozan dicho resultados gráficamente para tener una mejor comprensión de los mismos, ver Fig 3.4.1.3.

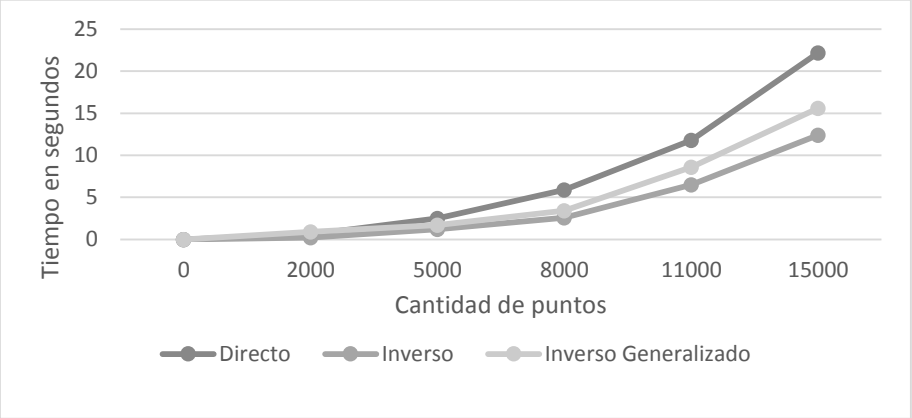


Fig 3.4.1.3: Resultado de las pruebas de rendimiento.

3.4.2. Resultados de las pruebas

La gráfica de la Fig 3.4.2 muestra los resultados obtenidos de acuerdo a la cantidad de no conformidades detectadas al aplicar las pruebas en cada una de las iteraciones realizadas. Una vez concluido el proceso de validación y verificación se comprueba que biblioteca cumple con las especificidades planteadas al inicio de la investigación, tras obtener los resultados esperados.

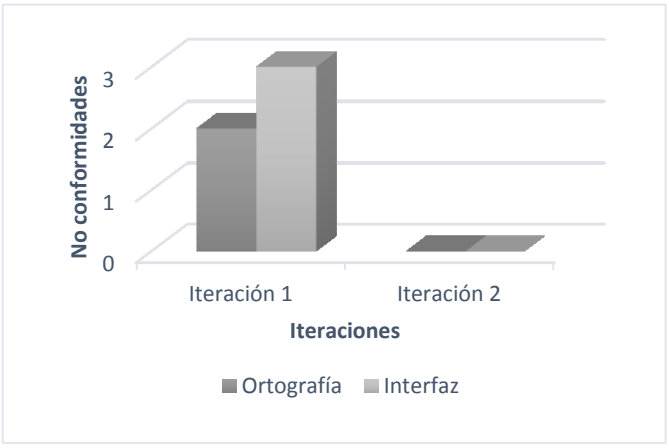


Fig 3.4.2: Resultados de las pruebas realizadas a la BP2DIN.

3. 5 Conclusiones parciales

Para esta primera versión de la BP2DIN, se concebido detalladamente el diseño íntegro de la misma, así como su arquitectura. Mediante el estudio y empleo de los patrones de arquitectura y de diseño se realiza una adecuada estructuración de la biblioteca, lo que hace más sencilla la integración de nuevos componentes, su actualización, corrección y mejora. Se expone además cómo queda expresada la biblioteca en componentes de implementación, se muestra el resultado de las pruebas realizadas a la biblioteca con varios conjuntos de datos que oscilan entre 0 y 15 000 puntos, retorna la respuesta esperada en un tiempo menor a 28 segundos, tiempo que cumple con lo establecido en los requisitos no funcionales descritos en el capítulo anterior.

Conclusiones generales

Al realizar un estudio detallado de los conceptos fundamentales, algoritmos y las herramientas similares a esta investigación, se dio cumplimiento a los objetivos planteados con la conclusión de la biblioteca para realizar el particionado 2D y problema inverso (BP2DIN). De manera que esta cumpliera los requisitos funcionales establecidos para lograr las metas trazadas con la realización de una biblioteca de algoritmos que muestra su funcionalidad con resultados satisfactorios.

Al tener la biblioteca de algoritmos un diseño adaptable a los posibles cambios dentro de su paquete de clases, posibilita que se puedan agregar nuevas funcionalidades en versiones posteriores sin hacer grandes modificaciones. Esto aumenta la utilidad de la biblioteca con vistas a confeccionar un producto cada vez más completo y eficaz.

El aporte principal es el desarrollo de una biblioteca de algoritmos y un manual de trabajo que brinda al desarrollador de aplicaciones informáticas, tales como SIG, software de diseño y compresión de imágenes, facilidades para implementar un módulo de particionado del plano, al utilizar específicamente, el cálculo de sitios, la generación de DV y la resolución de su PIV y PIVG para dar respuesta a las necesidades del usuario en menor tiempo. Además de mejorar, las aplicaciones que necesiten cálculos de proximidad.

Los programadores pueden concentrarse en los aspectos especializados de su aplicación al eliminar la dependencia parcial de especialistas en el tema de particionados del plano. Dicha biblioteca es la antesala para la confección de herramientas más complejas dedicadas a la compresión de datos para la construcción de modelos predictivos.

Recomendaciones

- Continuar el estudio de las aplicaciones de los DV.
- Añadir a la Biblioteca soporte para las TD.
- Añadir otros algoritmos que permitan determinar DV inversos y generalizados.
- Añadir técnicas de programación multi-hilo y programación en paralelo para ganar en el tiempo de respuesta de la biblioteca aprovechando de esta forma las arquitecturas multinúcleos.

Referencias Bibliográficas

Almaguer, Daniel Trinchet y Rosés, Hebert Pérez. 2007. *Algoritmo para solucionar el problema inverso generalizado de Voronoi.* La Habana : Revista cubana de Ciencias Informáticas, 2007.

Álvarez, Elaine Morales. 2010. *Módulo de Gestión de Errores de la Plataforma de Televisión Informativa PRIMICIA.* La Habana : s.n., 2010.

Aurenhammer, F. 1987. *A criterion for the affine equivalence of cell complexes in R^d and convex polyhedra R^d+1.* s.l. : Discrete and Computational Geometry, 1987. págs. 49-64.

Aurenhammer, F. 1987. *Power diagrams: properties, algorithms and applications.* s.l. : SIAM Journal on Computing, 1987. págs. 78-96.

Barchini, Graciela Elisa. 2005. *Métodos "I + D" de la Informática.* Santiago del Estero : s.n., 2005. Vol. Vol 2.

Blanco, Christian. 2014. *Medidas ergódicas como puntos extremos.* Xalapa : s.n., 2014.

Bogue, D. J.,. 1949. *The Structure of the Metropolitan Community: A Study of Dominance and Sub-dominance.* s.l. : University of Michigan: Contributions of the Institute for Human Adjustment, Social Science Research Project, University of Florida Libraries., 1949. pág. 14.

Castillo, Sebastián. 2014. *Correción automática de la geometría de fotogramas capturados desde plataformas aéreas no tripuladas.* Córdoba : Servicios de Publicaciones de la Universidad de Córdoba, 2014.

CGAL. 1995. CGAL: The Computational Geometry Algorithms Library. [En línea] 1995. [Citado el: 23 de 11 de 2015.] http://www.cgal.org/index.html.

de Breg, Mark, y otros. 2008. Computational Geometrics: Algorithms and Applications. Tercera. s.l. : Springer, 2008, 7.

Dobashi, Yoshinori, y otros. 2002. *A Method for Creating Mosaic Images Using Voronoi Diagrams.* Tokyo : The Eurographics Associations, 2002.

Drake, José M y López, Patricia. 2009. *Verificación y Validación.* 2009.

Duan, Qibin, y otros. 2014. *Inverting Laguerre Tessellation.* Australia : The Computer Journal, 2014.

Dyer, Ramsay, Zhang, Hao y Möller, Torsten. 2011. *Voronoi-Delaunay Duality and Delaunay Meshes.* Canada : s.n., 2011.

Ecured. 2016. EcuRed. *EcuRed.* [En línea] EcuRed, 2016. [Citado el: 24 de 3 de 2016.] http://www.ecured.cu/Estilos_arquitect%C3%B3nicos.

Elyiv, A, Melnyk, O y Vavilova, I. 2009. *High-order 3D Voronoi tessellation for identifying isolated galaxies, pairs and triplets.* s.l. : R. Astron. Soc., 2009.

Euler, L. 1741. *Solutio problematis ad geometriam situs pertinentis.* s.l. : Commentarii academiae scientiarum Petropolitanae 8, 1741. págs. 128-140.

Fowler. 2007. *UML destilled, A brief guide to standard object modeling languaje.* Tercera. s.l. : Pearson Education, 2007. ISBN: 978-8131-715-65-9.

Francés, A., y otros. 2009. *Diagramas de Voronoi y Topología Digital.* s.l. : Universidad de Rioja, 2009.

Garrido, Salvador Alemany. 2009. *Introduccion a QT. Programacion grafica en C++ con Qt4.* 2009.

Gómez, José Ramón. 2011. *Diagramas de Voronoi.* Matemática Aplicada, Universidad de Sevilla. Sevilla : s.n., 2011. págs. 8-12.

Gruber, P. M. 2007. *Convex and Discrete Geometry. A Series of Comprehensive Studies in Mathematics.* s.l. : Springer, 2007. pág. 467.

Gutiérrez, Javier J. 2012. ¿Qué es un framework web? [En línea] 2012. [Citado el: 28 de 9 de 2015.] http://www.cssblog.es/guias/Framework.pdf..

Haag, C. 1898. *Die Mundarten des oberen Neckar- und Donautales Schwbischalemannisches Grenzgebiet: Baarmundarten.* Reutlingen : Hutzler, 1898.

IDE. 2015. Entornos de desarrollo integrado. Concepto de IDE. [En línea] 2015. [Citado el: 25 de 9 de 2015.] http://petra.euitio.uniovi.es/~i1667065/HD/documentos/Entornos%20de%20Desarrollo%20Integrado.pdf.

IEEE. 2000. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.* s.l. : IEEE, 2000. ISBN 0-7381-2519-9.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Addison-Wesley, 2000.

Kopec, R. J. 1963. *An alternative method for the construction of Thiessen Polygons.* Quinta. s.l. : The Profesional Geographer, 1963. págs. 24-26. Vol. 5.

Larman, Craig. 2003. *UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos.* Segunda. s.l. : Prentice Hall, 2003. ISBN 9701702611.

Lautensack, C. 2007. *Random Laguerre Tessellations.* s.l. : PhD thesis Karlsruhe Institue of Technology, 2007.

Lenguajes de programación. 2015. Lenguajes de programación. [En línea] 2015. [Citado el: 12 de 10 de 2015.] http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml.

Louden, Kenneth C. 2004. *Lenguajes de programación: principios y prácticas.* Mexico : s.n., 2004.

Marbete, Punam y Gupta, Reetu. 2013. *Fortune's Method: An Efficient Method For Voronoi Diagram Construction.* s.l. : International Journal of Advanced Research in Computer and Communication Engineering, 2013. 2319-5940.

Martínez, Alondra, y otros. 2007. *Image Processing using Voronoi Diagrams.* s.l. : Spanish Ministry of Science and Education, 2007. TIN2005-08863-C03.

Martínez, Ulises. 2015. *Aplicación de la Geometría Computacional en la Reconstrucción 3D Basadas en diagramas de Voronoi.* Universidad Autónoma de Puebla. Puebla : s.n., 2015.

Okabe, A, y otros. 1999. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Segunda. s.l. : John Wiley and Sons, 1999. págs. 6-12.

Okabe, A, y otros. 1999. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Segunda. s.l. : John Wiley and Sons, 1999. págs. 221-224.

Okabe, A, y otros. 1999. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Segunda. s.l. : John Wiley and Sons, 1999. págs. 118-119.

Pérez Rosé, Hebert, y otros. 2011. *DIFFERENT APPROACHES TO VORONOI IMAGE COMPRESSION.* Santiago de Cuba : University of Oriente, Department of Computer Science, 2011.

Pressman, Roger S. 2015. *Software Engineering: A Practitioner's Approach.* Eighth. New York : Mc Graw Hill Interamericana, 2015. 978-0-07-802212-8.

Programación en Castellano. 2015. Programación en castellano. Qt Creator, un completo entorno de desarrollo. [En línea] 27 de 9 de 2015. [Citado el: 27 de 9 de 2015.] http://www.programacion.com/noticia/qt_creator-_un_completo_entorno_de_desarrollo_1723.

Pruebas_de_software. 2005. Pruebas de software. [En línea] 2005. [Citado el: 5 de Mayo de 2016.] http://pruebasdesoftware.com/laspruebasdesoftware.htm.

Ramón, Romanuel. 2013. *Geometría Computacional: Diagrama de Voronoi.* La Habana : s.n., 2013.

Reda, Yaagoubi, y otros. 2015. *HybVOR: A Voronoi-Based 3D GIS Approach for Camera Surveillance Network Placement.* Kingdom of Saudi : Chris Gold and Wolfgang Kainz, 2015. ISSN: 2220-9964.

Reynoso, Carlos y Kiccillof, Nicolás. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

Rivero, José Pablo Suárez. 2001. *ESTRUCTURAS DE DATOS TIPO GRAFO EN LOS ALGORITMOS DE REFINAMIENTO Y DESREFINAMIENTO BASADOS EN LA BISECCIÓN POR EL LADO MAYOR. APLICACIONES.* Las Palmas, Gran Canaria : Universidad de Las Palmas de Gran Canaria, 2001.

Rodríguez, Raúl Lugo. 2011. *Análisis, diseño e implementación del subsistema de Transmisión de la plataforma de televisión informativa PRIMICIA versión 2.0.* La Habana : s.n., 2011.

Rosen, K. H. 2004. *Matemática discreta y sus aplicaciones.* Quinta. s.l. : McGraw-Hill, 2004. págs. 503-581.

Schoenberg, F. P., Ferguson, T. y Li, C. 2003. *Inverting Dirichlet Tessellation.* s.l. : The Computer Journals, 2003. págs. 76-83.

Shamos, M. I. 1999. *The early years of Computational Geometry, a personal memoir.* s.l. : Contemporary Mathematics, 1999. págs. 313-332.

Shamos, M. I y Hoey, D. 1975. *Closest-point problems, 16th Annual Symposium on Foundations of Computer Science.* s.l. : IEEE, 1975. págs. 151-162.

Shewchuk, Jonathan Richard. 2005. Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator. [En línea] 28 de Julio de 2005. [Citado el: 16 de 11 de 2015.] http://www.cs.cmu.edu/~quake/triangle.html.

Software, Geom. 2012. Geom Software - C++ Programming and Geometry Libraries. [En línea] 9 de 2012. [Citado el: 28 de 11 de 2015.] http://www.geom.at/fade2d/html/.

Triangulation. 2011. Triangulation Library. [En línea] 9 de Septiembre de 2011. [Citado el: 18 de Marzo de 2015.] http://people.sc.fsu.edu/~jburkardt/cpp_src/triangulation/triangulation.html.

Wesley, Addison. 2004. *Software development for small team: A RUP - centric approach.* s.l. : Addison Wesley Object technology series, 2004. ISBN: 978-0321-199-50-8.

Wormser, Camille. 2008. *Generalized Voronoi Diagrams and Applications .* s.l. : Computer Science. University Nice Sophia Antipolis, 2008.

Anexo 1. Brote de cólera en Londres

Fig 4.1: Análisis del brote de cólera en Londres por John Snow, 1854.

Anexo 2. Geografía de los dialectos de Suabia

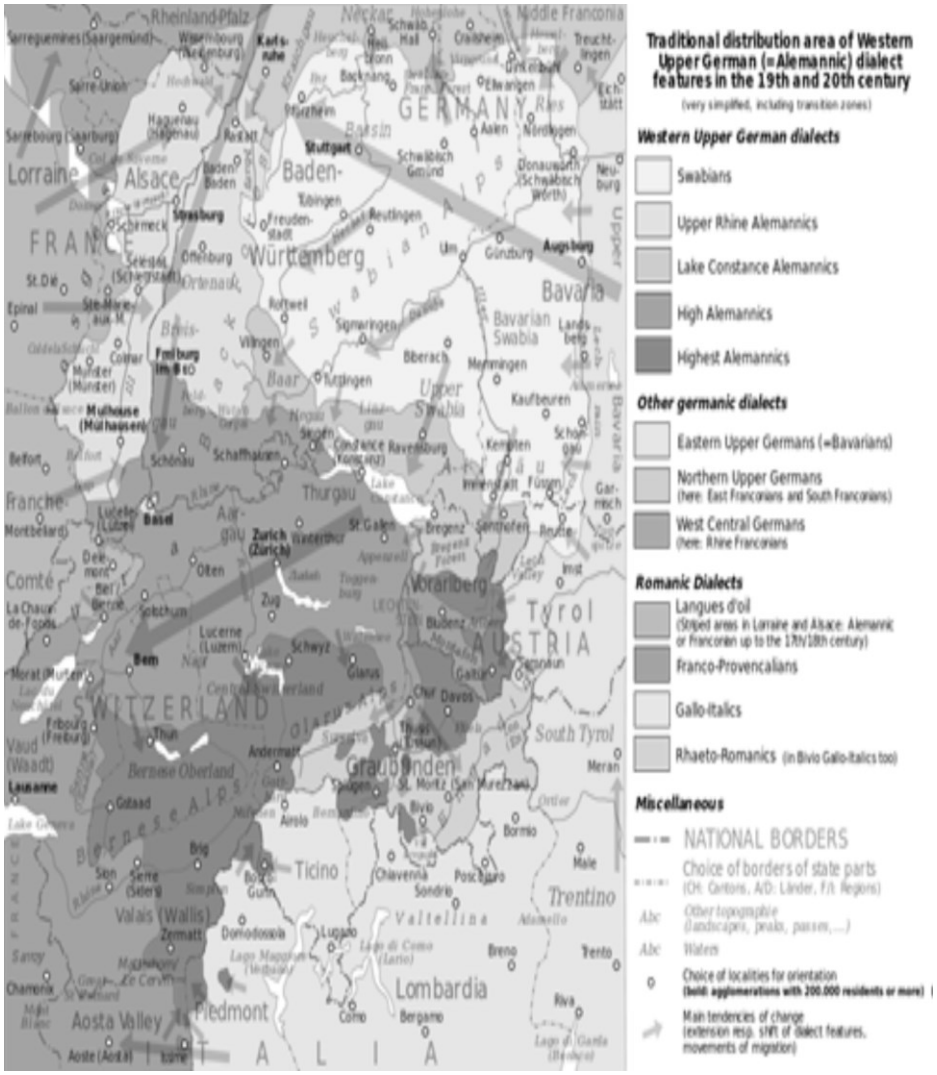
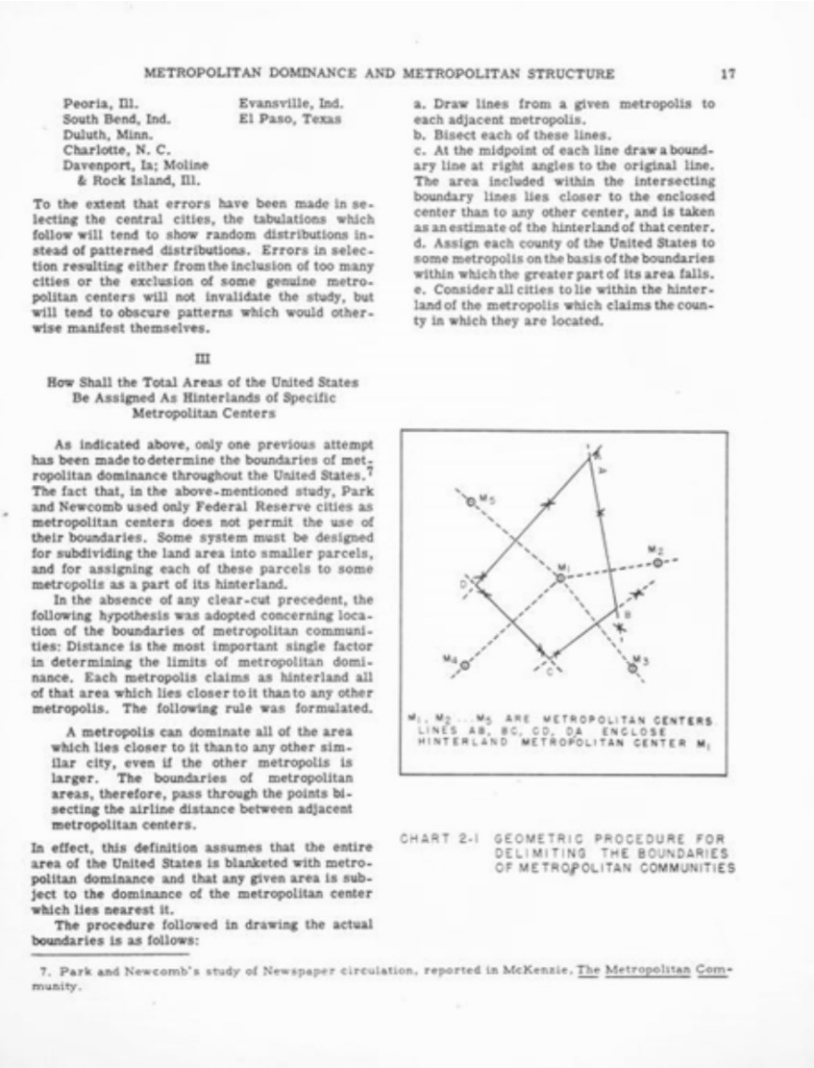


Fig 4.2: Geografía de los dialectos de Suabia.

Anexo 3. Estudios de áreas de mercado por Bogue



METROPOLITAN DOMINANCE AND METROPOLITAN STRUCTURE 17

Peoria, Ill.
South Bend, Ind.
Duluth, Minn.
Charlotte, N. C.
Davenport, Ia; Moline & Rock Island, Ill.
Evansville, Ind.
El Paso, Texas

To the extent that errors have been made in selecting the central cities, the tabulations which follow will tend to show random distributions instead of patterned distributions. Errors in selection resulting either from the inclusion of too many cities or the exclusion of some genuine metropolitan centers will not invalidate the study, but will tend to obscure patterns which would otherwise manifest themselves.

III

How Shall the Total Areas of the United States Be Assigned As Hinterlands of Specific Metropolitan Centers

As indicated above, only one previous attempt has been made to determine the boundaries of metropolitan dominance throughout the United States.[7] The fact that, in the above-mentioned study, Park and Newcomb used only Federal Reserve cities as metropolitan centers does not permit the use of their boundaries. Some system must be designed for subdividing the land area into smaller parcels, and for assigning each of these parcels to some metropolis as a part of its hinterland.

In the absence of any clear-cut precedent, the following hypothesis was adopted concerning location of the boundaries of metropolitan communities: Distance is the most important single factor in determining the limits of metropolitan dominance. Each metropolis claims as hinterland all of that area which lies closer to it than to any other metropolis. The following rule was formulated.

> A metropolis can dominate all of the area which lies closer to it than to any other similar city, even if the other metropolis is larger. The boundaries of metropolitan areas, therefore, pass through the points bisecting the airline distance between adjacent metropolitan centers.

In effect, this definition assumes that the entire area of the United States is blanketed with metropolitan dominance and that any given area is subject to the dominance of the metropolitan center which lies nearest it.

The procedure followed in drawing the actual boundaries is as follows:

a. Draw lines from a given metropolis to each adjacent metropolis.
b. Bisect each of these lines.
c. At the midpoint of each line draw a boundary line at right angles to the original line. The area included within the intersecting boundary lines lies closer to the enclosed center than to any other center, and is taken as an estimate of the hinterland of that center.
d. Assign each county of the United States to some metropolis on the basis of the boundaries within which the greater part of its area falls.
e. Consider all cities to lie within the hinterland of the metropolis which claims the county in which they are located.

$M_1, M_2 \ldots M_5$ ARE METROPOLITAN CENTERS. LINES AB, BC, CD, DA ENCLOSE HINTERLAND METROPOLITAN CENTER M_1

CHART 2-I GEOMETRIC PROCEDURE FOR DELIMITING THE BOUNDARIES OF METROPOLITAN COMMUNITIES

7. Park and Newcomb's study of Newspaper circulation, reported in McKenzie, The Metropolitan Community.

Fig 4.3: Estudio de áreas de mercado por Bogue. Copyright: Universidad de Florida.

Anexo 4. Diagrama del CUS obtener generadores PIVG

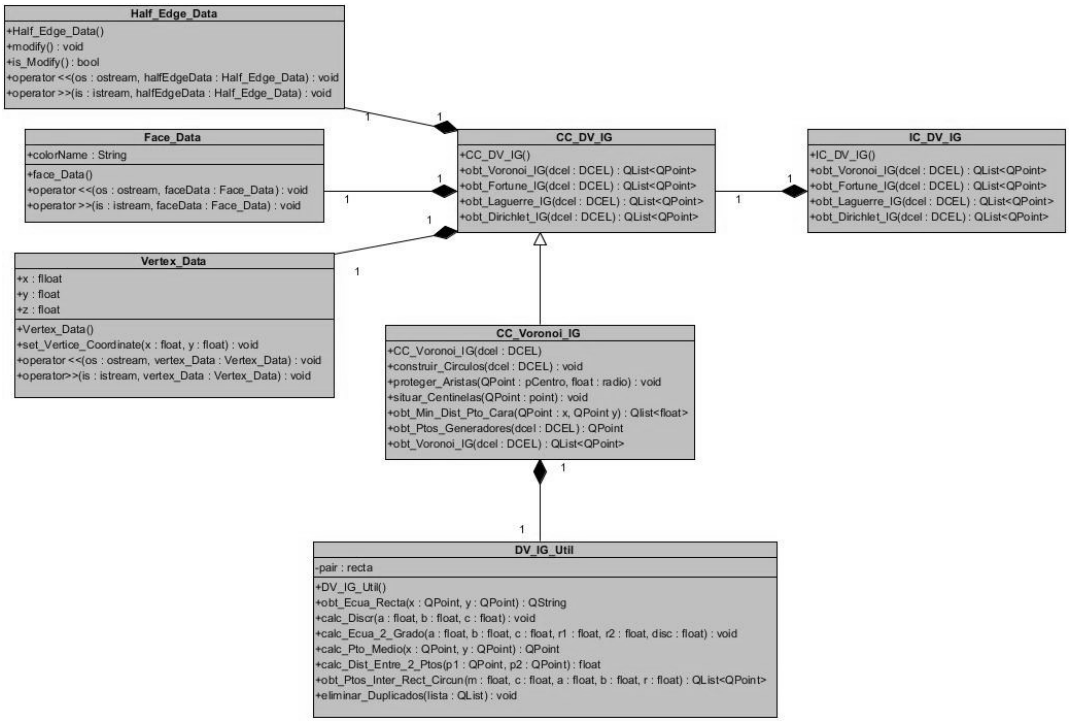


Fig 4.4: Diagrama de clases CU obtener generadores PIVG.