Particiones Convexas, su aplicación en los Sistemas de Información Geográfica.

Yurai Núñez Rodríguez

16 de junio de 2000

Agradecimientos.

La realización de este trabajo ha sido posible gracias al esfuerzo de muchas personas a quienes no puedo dejar de agradecer. Primeramente debo mencionar a los estudiantes de tercer año Alexander Sánchez y Alejandro Rodríguez por haber trabajado conmigo en este sistema, sacrificando parte de su tiempo y de su sueño por la creación e implementación de varios algoritmos geométricos utilizados por mí posteriormente.

Debo agradecer también a los estudiantes y trabajadores del grupo de redes de la universidad, por permitirme compartir sus recursos, y brindarme asesoramiento técnico ante tantas dificultades a la hora de escribir este trabajo. Por mencionar algunos aspectos en los que me han ayudado destacaría el acceso a Internet, tiempo de máquina, impresoras y papel, facilitado por parte de Mario Díaz, jefe del grupo. Además las super-clases de Latex impartidas por Javier.

Agradezco a Adis María, Karel Alfonso y Adrián Sánchez por haberme ayudado en la revisión y corrección del trabajo escrito; también a Osvaldo Villegas y María Mercedes Vera por facilitarme algunos materiales para encuadernar el mismo.

Agradezco a todos aquellos profesores que me han enseñado diversos temas y me han formado en mi carrera, y más aun a aquellos que, además, me han dado el impulso necesario para confiar en mí mismo y seguir adelante .

Agradezco sin falta a todos mis amigos y familiares que me han brindado su apoyo y me han proporcionado las condiciones necesarias para poder dedicarle el mayor tiempo posible

al desarrollo de este trabajo de diploma.

Agradezco a todas aquellas personas, a quienes mi pésima memoria no haya logrado recordar; que de una forma u otra han aportado algún detalle o alguna idea que me haya permitido llevar hasta este punto mis esfuerzos.

Y por supuesto agradezco inmensamente a los dos "Tony", mi tutor Antonio Mesa por todas las cosas que me ha enseñado y por haberme ayudado tanto durante toda mi membresía en nuestro grupo de investgación y no sólo ahora, al realizar mi tesis, donde sus ideas fueron mi principal guía. Y por último Antonio Hernández, quien me ha dado una gran ayuda en cuanto a los enfoques que he debido dar a mi tesis, y demás aspectos para llevar esta por un camino de formalización y corrección, al cual espero haberme acercado.

Resumen

El presente trabajo está dirigido a la investigación de estructuras de datos y algoritmos eficientes para implementar Sistemas de Información Geográfica(SIG) urbanos. Los algoritmos y estructuras de datos referidos son aquellos dentro del campo de la Geometría Computacional, lo cual constituye una parte importante en estos sistemas. Principalmente se ha desarrollado una estructura denominada convex DCEL, que no es más que una partición del plano donde todas sus regiones tienen la forma de polígonos convexos, en los cuales quedan contenidos los objetos de la ciudad. Sobre esta estructura se han desarrollado un conjunto de algoritmos que permiten su creación y la realización de consultas sobre la misma, la visualización tridimensional de los objetos, y la actualización y modificación de estos. Los objetos que han sido tratados son las manzanas y los edificios, aunque se han tenido en cuenta otros. Las consultas fundamentales son: la identificación de puntos, tanto en el plano como en el espacio, además, la identificación de áreas dentro de la ciudad. Se demuestran en este trabajo aspectos positivos de las estructuras de datos y algoritmos utilizados, obteniendo algunos resultados significativos.

Palabras claves:
SIG
Geometría Computacional
convex DCEL
algoritmos
tridimensional
particiones convexas

Índice General

Índice de Figuras

Capítulo 1

Introducción

Este trabajo está orientado a la investigación de algoritmos geométricos y estructuras de datos eficientes dentro de la geometría computacional, específicamente dentro del área de la implementación de Sistemas de Información Geográfica(SIG); y más específico aun en los SIG urbanos.

El término Sistema de Información Geográfica o SIG se aplica actualmente a los sistemas computarizados de almacenamiento, elaboración y recuperación de datos con equipos y programas específicamente designados para manejar los datos espaciales de referencia geográfica y los correspondientes datos cualitativos o atributos. La tecnología del SIG puede ayudar a establecer la comunicación de varios sectores proporcionando no solamente las herramientas de gran alcance para el almacenaje y el análisis de datos espaciales y estadísticos multisectoriales, sino que también integra las bases de datos de los diversos sectores en un mismo formato, estructura y mapa en el SIG. Los sistemas de información geográfica tienen tres componentes principales: los equipos, los programas informáticos, los recursos humanos y la organización que hace que el sistema funcione [dSdlF99]. Estos sistemas son de gran utilidad en un número elevado de proyectos urbanos relacionados con las comunicaciones, las construcciones, las redes de transporte, de vías, de servicios, y muchos más; en sentido general se consultan hoy en día estos sistemas para cualquier trabajo que se realice dentro de la ciudad siempre que este lleve un estudio profundo y planificación adecuada.

La componente que nos interesa a nosotros es la constituida precisamente por los programas informáticos y el almacenamiento de la información. Dentro de esta componente se puede hacer una distinción entre los sistemas de tipo "Raster" y los sistemas de tipo "Vectorial", en nuestro caso hemos trabajado en la implementación de un sistema Vectorial, el cual presenta algunas ventajas, entre las que se encuentran: la necesidad de menor capacidad de almacenamiento que en los sistemas Raster, el mapa original puede representarse en su resolución original, y además múltiples atributos pueden ser fácilmente representados. Aunque se presentan ciertas desventajas las cuales nos afectan directamente a nosotros como desarrolladores de estos sistemas y esto se debe a que los algoritmos para las funciones realizadas son más complejos que los de los sistemas Rasters.

En el mundo en general existen varios sistemas ya creados, los cuales son comerciales y presentan una alta calidad, entre los cuales se pueden destacar MAPINFO, ARCVIEW, ILWIS entre otros, los cuales tienen disponible versiones para PC. A pesar de todos estos trabajos anteriormente realizados queda mucho por investigar y por hacer dentro de esta materia con vistas a mejorar los algoritmos y así ganar en rapidez tanto al consultar como al introducir datos, ganar en precisión, en ahorro de memoria, en utilidad y facilidad para el usuario, que en este caso, no tiene que ser un especialista en computación. En nuestra facultad se viene trabajando en este tema desde hace ya algunos años dentro del grupo de investigación de Geometría Computacional. Se han realizado varios trabajos con respecto al tema urbano incluso, por lo que se cuenta con cierta experiencia. Se puede mencionar, por ejemplo, los trabajos de diploma de varios estudiantes graduados en años anteriores, los cuales fueron parte significativa de la bibliografía utilizada para llevar a cabo este trabajo.

Dentro de un SIG, y para ser más específico dentro la componente software del mismo pudiéramos distinguir partes que serían la información espacial y gráfica, que nosotros tratamos como información geométrica, y el tratamiento de datos descriptivos no gráficos, como la información estadística, conjuntamente con los datos espaciales a los que están relacionados. La parte que concierne a este trabajo es precisamente la parte geométrica del problema, siendo esta una parte fundamental y la más complicada desde el punto de vista algorítmico. Además siempre se ha pensado en una solución que permita tratar todas las partes del problema lo mejor posible.

En el pequeño sistema que hemos implementado se trabajó en una estructura de datos que soporta parte de los objetos fundamentales que se encuentran en una ciudad, como son los edificios y las manzanas y que de forma eficiente realiza varias consultas geométricas[1] y permite una visualización también muy eficiente en tercera dimensión(3D) de los edificios de la ciudad. El haber obtenido algunos logros en cuanto a las estructuras de datos y los algoritmos no sólo tienen un valor teórico, sino que al lograr mejoras en tiempo y espacio permiten realmente que en la práctica se puedan mejorar los sistemas ya existentes y crear otros que permitan obtener mejores resultados con menos recursos de hardware y en un menor tiempo para el usuario.

Para lograr estos resultados se han utilizado algoritmos basados en el trabajo con polígonos fundamentalmente y la estructura de datos utilizada es básicamente un grafo que particiona el espacio(plano), lo cual permite tener de forma implícita relaciones geométricas entre todos los objetos que se contienen en el mapa, introduciendo nosotros la utilización de una partición convexa del plano, lo cual permite, de forma sencilla y eficiente, realizar una ordenación en los objetos del mapa y efectuar algoritmos de búsqueda. Otro resultado que hemos introducido es el haber trabajado con una aritmética de valores enteros casi en la totalidad del proyecto, lo que trae como ventajas una mayor rapidez en los cálculos y menos problemas de precisión. Para llegar a este resultado hemos consultado varios libros de Geometría Computacional incluso algunos más particulares que tratan de su aplicación dentro de los SIG. Estos libros han ayudado mucho en la obtención de algoritmos eficientes para el trabajo con los objetos más primitivos que se tratan en este trabajo como son los puntos, los segmentos, las líneas o rectas, rayos o semirectas y polígonos. Además en ellos se ofrecen ideas acerca de como afrontar este tipo de problemas y se muestra el trabajo con otras estructuras ampliamente estudiadas, tales como: las redes irregulares de triángulos, triangulaciones de Delaunay y Diagramas de Voronoi.

Como objetivo principal se encuentra el desarrollar las bases para un SIG que cuente con nueva funcionalidad e incluya todo un trabajo teórico e investigativo que haga de este, un

[1] Tipo de consultas que se refieren a la relación geométrica que guardan los objetos. Ejemplo: objetos vecinos de algun objeto determinado.

sistema eficiente. Se desea que el sistema a desarrollar no tenga un uso particular, sino que tenga la suficiente flexibilidad y adaptabilidad para poder incorporar en el mismo nuevas funcionalidades. Además de desarrollar todo este conjunto de algoritmos y estructuras de datos para darlos a conocer con el fin de que puedan ser utilizados en un futuro por cualquier otra entidad, la cual les pueda dar un uso completamente positivo. Como proyecto futuro también se viene manejando entre varias entidades de la provincia, la construcción de un SIG de la Ciudad de La Habana donde nuestro sistema podría jugar un importante papel.

El trabajo escrito está formado por 3 capítulos, excluyendo la introducción y las conclusiones, los cuales tratan, en ese mismo orden, las estructuras de datos más comunmente usadas para afrontar estos tipos de problemas, y las particiones convexas; así como las bases teóricas fundamentales en las que se basa el proyecto. El que le sigue constituye la explicación de las propiedades y operaciones que se utilizan para la formación de la partición convexa, analizando previamente un conjunto de preliminares en cuanto a estructuras de datos y algoritmos empleados. El siguiente, muestra las aplicaciones de la partición convexa, incluyendo un conjunto de algoritmos que operan sobre esta y muestran sus aspectos positivos. Posteriormente tenemos el capítulo que trata de la estructura que presenta el sistema implementado, su diseño de clases y su relación directa con las estructuras de datos. Lógicamente continúa el trabajo con las conclusiones y las propuestas para seguir desarrollando y complementar estas ideas, también problemas que quedan abiertos para todos aquellos que estén interesados en el tema.

Capítulo 2

Estructuras de Datos.

En este capítulo nos adentraremos ya en el problema que nos ha ocupado, así como las soluciones alternativas que han sido estudiadas e implementadas con anterioridad por muchos sistemas y la solución dada por nosotros a las estructuras de datos necesarias para la implementación de los SIG. Además veremos algunas comparaciones entre todas estas soluciones.

Existen varias técnicas que se utilizan para particionar un espacio determinado en diferentes partes o regiones y de esta forma tener a los objetos existentes en el espacio contenidos en diferentes regiones, lo cual proporciona una cantidad importante de información geométrica, al poder establecer relaciones de adyacencia y topológicas, en general, sobre estos objetos. El caso que nos interesa es el caso en que dicho espacio es el plano o para restringirlo más aun una región, digamos rectangular, de un plano; o sea, que nuestro espacio será el área de un rectángulo, llamémoslo R y nuestros objetos serán polígonos convexos, llamemos P al conjunto de estos polígonos convexos.

2.1 Estructuras que constituyen soluciones anteriores para implementaciones de SIG.

Dos de los tipos más importantes de mapas son los de tipo "choropleth" y los de tipo "isoline". Un mapa choropleth es básicamente una subdivisión en regiones, donde los límites separan regiones con diferentes atributos o propiedades. Por ejemplo, un mapa que muestra cierta cantidad de países, los límites siempre tienen diferentes países en sus dos lados. Un mapa tipo isoline también es una subdivisión en regiones, pero en este caso los límites muestran localizaciones donde los atributos tienen un mismo valor. En este caso podríamos tener un mapa que muestre las precipitaciones, donde las líneas que constituyen los límites, muestran localizaciones donde se recoge un determinado valor, sea de 800mm o 850mm de precipitación por ejemplo. Generalmente estos tipos de mapas son usados para representar modelos de elevación [vKNRW97].

Existe una gran diversidad de estructuras que constituyen técnicas de división del espacio y que son ampliamente utilizadas en los SIG, en este caso voy a explicar las más utilizadas, sin entrar en detalles de implementación o de los algoritmos para su construcción.

2.1.1 Diagramas de Voronoi.

Una de las técnicas antes mencionadas son los Diagramas de Voronoi. Veamos qué es un Diagrama de Voronoi, en este caso, estático, o sea, para un conjunto prefijado de objetos a los cuales llamaremos generadores, para ser coherentes con la literatura al respecto. Sea S el conjunto de generadores, que en este caso pudieran ser polígonos convexos degenerados en líneas o puntos, aunque generalmente se trata en la literatura consultada con líneas. Haremos esta extensión para poder hacer posteriormente una comparación adecuada. El conjunto S en este caso coincide con el conjunto P de polígonos antes mencionado, que están contenidos en el espacio, los cuales se quieren separar en regiones. Utilicemos como d la distancia Euclidiana y supongamos, para mayor simplicidad, que no hay 3 generadores a la misma distancia [vKNRW97].

Veamos ahora las particiones que se realizan de acuerdo a la regla del vecino más cercano. Definamos el conjunto de bisectores:

$$\mathrm{B}_{ij} = \{x \in \Re^2 | d(x, p_i) = d(x, p_j)\}$$

para dos polígonos p_i, p_j que pertenecen a P. Cada B_{ij} constituye una curva diferenciable formada por rectas y parábolas. Definamos además las celdas o regiones:

$$v_i = \{x \in \Re^2 | d(x, p_i) \leq d(x, p_j), \forall j | p_j \in P\}$$

Estas celdas estarán acotadas o delimitadas por los bisectores y las intersecciones entre ellos, donde estas intersecciones serán los vértices del diagrama, denotando V(S) como el conjunto de todos los vértices. De esta forma habrá un vértice que pertenece a V(S) por cada tres generadores p_i, p_j, p_k tal que exista una circunferencia que coincida con cada uno de ellos en exactamente un punto y esta a su vez no se intersecte con ningún otro generador.

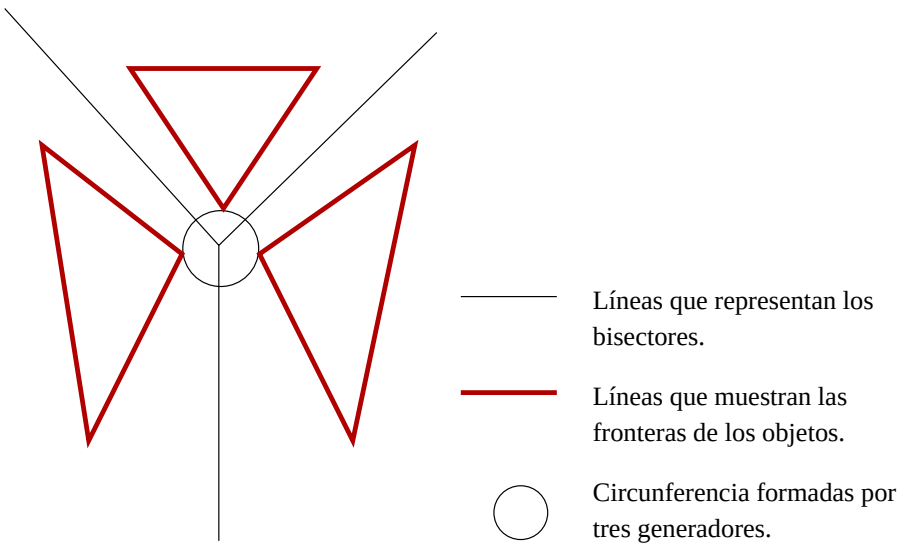

Figura 2.1: Diagrama de Voronoi.

A partir de un Diagrama de Voronoi podemos saber cual es el objeto más cercano a

un punto determinado, dependiendo de: a qué celda v_i pertenece el punto. De igual forma es fácil obtener los objetos vecinos de un objeto determinado y realizar diversas consultas sobre el mapa en cuestión. Además se pueden realizar operaciones de extracción, inserción y mezclar o intersectar con otros diagramas o conjuntos de objetos.

2.1.2 Triangulaciones.

También relacionados con estos diagramas tenemos las Redes Irregulares Trianguladas o TIN que proviene de sus siglas en Inglés (Triangulated Irregular Networks). Estas no son más que triangulaciones del espacio basadas en un conjunto de vértices. En este caso nos referimos a una triangulación plana, donde, por supuesto, no pueden intersectarse los lados de los triángulos, ver fig.2.2.

Un caso particular de estas triangulaciones son las triangulaciones de Delaunay las cuales tienen la particularidad de ser la triangulación donde la amplitud del menor de los ángulos, en todos los triángulos que conforman dicha triangulación, es la máxima posible, pero además de esto si vemos esta triangulación como un grafo Gt = < V, A > donde sus vértices(V) no son más que el conjunto de vértices sobre los cuales se ha construido la triangulación, y las aristas A son los lados de los triángulos, entonces este grafo es dual del grafo constituido por el Diagrama de Voronoi sobre ese conjunto de puntos. Esto significa que esta triangulación tiene varias características que pueden ser de mucha utilidad a la hora de tratar con objetos de un plano, puesto que un Diagrama de Voronoi puede ser almacenado en memoria como una triangulación de Delaunay y a su vez tiene las propiedades de una triangulación. Por las razones expuestas anteriormente esta estructura es ampliamente utilizada en los SIG.

La forma de ver un Diagrama de Voronoi como una triangulación de Delaunay es la siguiente: Sea un Diagrama de Voronoi Gd y su grafo dual Gt. Dos puntos están en celdas vecinas en Gd si y sólo si son adyacentes en Gt y existe un vértice en Gd que significa la intersección en la frontera de tres celdas en Gd que contienen a los objetos v_i, v_j, v_k si y sólo si v_i, v_j y v_k son adyacentes dos a dos en Gt, o sea que forman un triángulo.

2.2 Partición convexa, comparación con las anteriores.

Nuestra partición del espacio consiste en un conjunto de regiones convexas, las cuales tienen forma de polígonos y cada una de estas regiones tiene en su interior exactamente un objeto o ninguno, lo cual significa que no hay objetos que queden compartidos en varias regiones y que no hay regiones que contengan más de un objeto en su interior. La estructura de datos que soporta esta partición será analizada en el, ep.3.1.5 y la denominaremos convex DCEL.

Para hacer una comparación entre esta nueva estructura, que acabo de describir de forma sencilla, y las descritas anteriormente, basémonos entonces en que los objetos en nuestro caso son los polígonos que pertenecen a P, como lo habíamos nombrado inicialmente, los cuales constituyen las manzanas de la ciudad. Si en un SIG urbano se tratara de hacer una triangulación del espacio ocurriría que dada una manzana, su interior quedara dividido en triángulos, lo cual no sólo es innecesario y sería un costo de memoria adicional, sino que sería inconveniente debido a que una manzana debe tratarse como una unidad, que no debe dividirse salvo en extremas circunstancias. Además los edificios que quedan contenidos en ella quedarían igualmente fragmentados, de esta forma los objetos se vuelven prácticamente

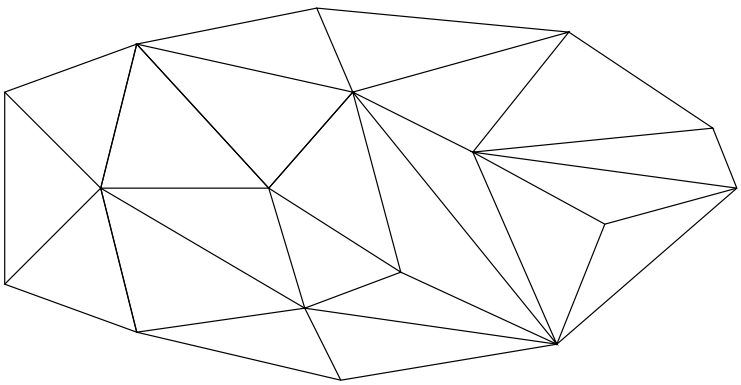

Figura 2.2: Ejemplo de triangulación.

intratables, y la complejidad de los algoritmos aumentaría considerablemente. Otra desventaja de la triangulación es que no sólo en el interior de las manzanas, sino que el área exterior a estas quedaría excesivamente fragmentada, cuyas consecuencias serían de igual forma un consumo de memoria totalmente innecesario y un aumento en el tamaño de las estructuras de datos, lo cual a su vez conllevaría a que el tiempo de ejecución de las operaciones sobre dicha estructura o demás funciones que la tomen como datos, aumentaría en un orden que depende ya del algoritmo en que estén basadas.

Obviamente las triangulaciones son estructuras que se emplean para representar relieves, al construir las mismas sobre un conjunto de puntos, a los cuales se les asigna una altura determinada. Además pueden ser usadas en otros tipos de mapas pero fundamentalmente en mapas de tipo "isoline".

Pasemos entonces a comparar nuestra estructura con los Diagramas de Voronoi. Es fácil darse cuenta que ambas constituyen mapas de tipo "choropleth", o sea que ambas estructuras están compuestas por regiones delimitadas dentro del espacio, en las cuales quedan contenidos integralmente los objetos. Pero también existen marcadas diferencias tales como que en la convex DCEL podemos encontrar regiones vacías, lo cual no entra en contradicción con la definición que se ha dado de la misma, mientras que esto no ocurre en los Diagramas de Voronoi, pero en estos a su vez podemos encontrar regiones no convexas, para ser más preciso, puede observarse que donde exista una celda cuyos límites o fronteras contengan un bisector en forma parabólica, se estará generando una región no convexa a uno de los lados del bisector. Esto, lógicamente, no ocurre en nuestra estructura debido a su propia definición. El hecho de que la convexidad de la estructura sea una ventaja lo aclararemos a continuación.

Como bien todos sabemos, las regiones convexas son un caso particular de las regiones en sentido general, y también es sabido que la convexidad de un conjunto u objeto cualquiera, es un elemento de gran importancia por la gran cantidad de propiedades que aparecen en el mismo. En nuestro caso podremos ver que al ser convexas las regiones se facilitan grandemente los algoritmos que expondremos en el cap.3, para la localización de puntos y para la visualización en tercera dimensión de los objetos contenidos en la ciudad. De las

propiedades particulares de las regiones convexas y en nuestro caso de los polígonos convexos, podemos destacar algunas que son las más utilizadas por nuestros algoritmos, estas son:

Propiedades que cumplen los polígonos convexos.

1-Una recta atraviesa dos veces a lo sumo la frontera de un polígono convexo.

2-Si tomamos dos puntos que pertenecen al polígono convexo entonces el segmento que los une también pertenece al polígono convexo.

3-Todos los ángulos interiores de un polígono convexo son menores o iguales que π.

4-En un plano con un punto prefijado, y un ángulo determinado a partir de dicho punto, se puede establecer un orden total $\preceq$ (Visitado antes o al mismo tiempo que) al visitar los polígonos que queden contenidos en este ángulo, partiendo del punto, siempre y cuando la intersección de los polígonos dos a dos sea vacía, ver fig.4.1.

Explicaré y demostraré la última propiedad debido a que esta es la más compleja de ellas.

Lo que quiere decir es sencillamente que si estuviéramos parados en un punto p en el plano nos quedarían unos polígonos detrás de otros para cualquier dirección determinada en la que nos movamos dentro de un ángulo determinado.

Demo: Probemos que si trazamos un rayo r desde un punto en el plano, en cualquier sentido y dirección, puede establecerse un orden total entre los polígonos convexos intersectados por el rayo. Primero vamos a asumir que el punto p prefijado no está en el interior de algún polígono, o si lo está, excluimos a este polígono del conjunto y después lo tomamos como el más próximo al observador.

Sean dos polígonos convexos p_i, p_j, veamos que al avanzar por el rayo r que corta en su interior a p_i y p_j, primero intersectamos la frontera de de uno de ellos, supongamos que fue la de p_i sin perder generalidad, en el punto $p_i f_1$, si antes de alcanzar el otro punto frontera de p_i que se intersecta con r, llamémoslo $p_i f_2$, encontraramos la intersección de r con algún punto de p_j, p_jr, entonces tendríamos que el punto $p_j r$ pertenece al segmento ($p_i f_1$, $p_i f_2$), o sea que pertenece a p_i y a p_j a la vez, lo cual es una contradicción porque $p_i \bigcap$pj=$\emptyset$. Lo que prueba que primero se intersectan los puntos de p_i y después los de p_j, por lo que se puede decir que $p_i \preceq r\ p_j$, donde $\preceq r$ significa "Visto antes o al mismo tiempo que, según el rayo r".

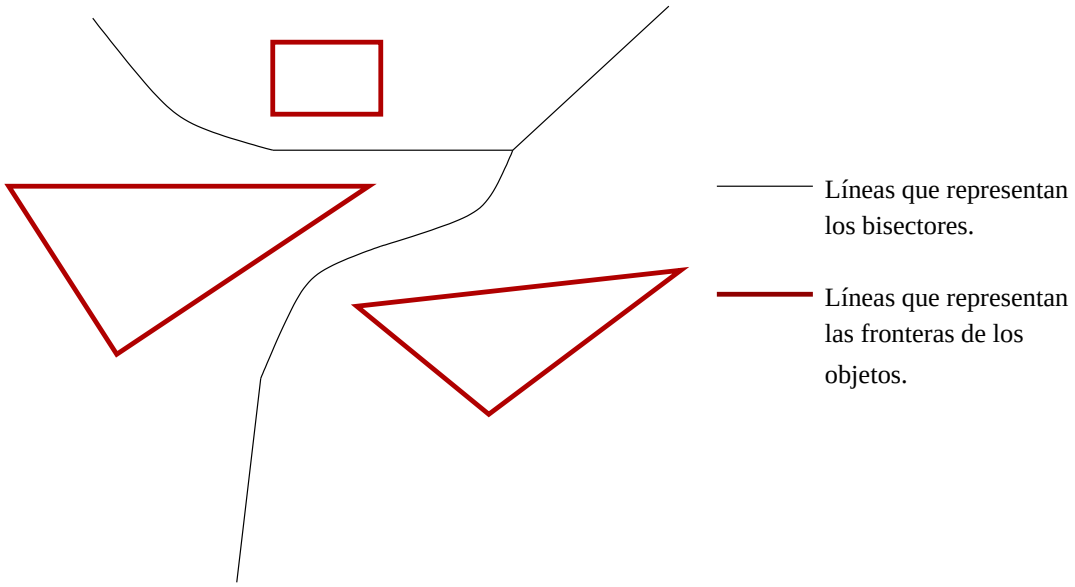

Figura 2.3: Diagrama de Voronoi con regiones no convexas.

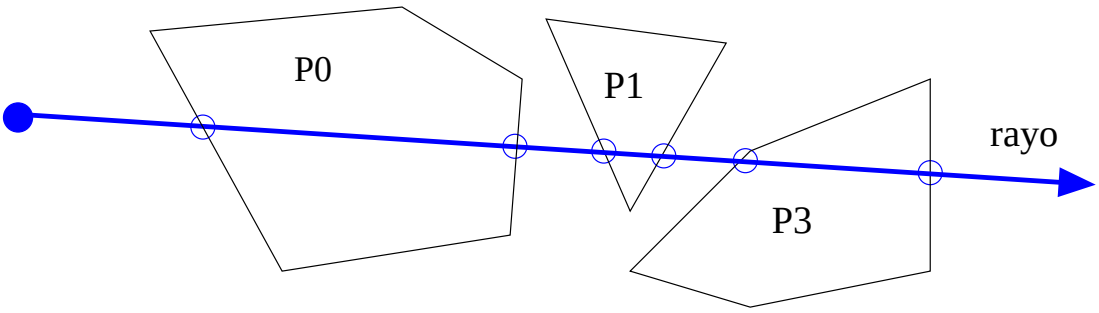

Figura 2.4: Ordenación entre polígonos atravesados por un rayo.

Es fácil ver que esta relación es reflexiva y antisimétrica, porque a no ser que el polígono sea el mismo no podemos decir que se intersecta una frontera primero que la otra y viceversa. Y si seguimos avanzando igualmente por el rayo y quedan involucrados 3 polígonos p_i, p_j, p_k cualesquiera también es fácil ver que la relación es transitiva.

Demostremos ahora que para cualesquiera dos rayos r_1, r_2 que parten de p y cortan a los polígonos p_i y p_j, si $p_i \preceq r_1$pj entonces $p_i \preceq r_2$ p_j. Supongamos que sea falso, o sea que $p_i \preceq r_1 p_j$ y que $p_j \preceq r_2$ p_i. Escojamos un punto de intersección de $r_1$con p_i y con p_j, llamémoslos $p_i r_1$y $p_j r_1$respectivamente, de igual forma escojamos $p_i r_2$ y $p_j r_2$. Tenderíamos que d($p_i r$, p) $<$ d($p_j r$, p) dado que por r_1, p_i está más cerca que p_j, y por suposición tendríamos que d(p_jr2,p) $<$ d(p_ir2,p) entonces podemos formar un cuadrilátero simple convexo dado el siguiente orden de sus vértices: $p_i r_1$, $p_j r_2$, $p_i r_2$, $p_j r_1$, cuyas diagonales estarían formadas por extremos de un mismo polígono $p_i r_1$, p_i r_2y $p_j r_2$, $p_j r_1$, las cuales, además, se cortarían por ser diagonales de un cuadrilátero convexo, y esto vuelve a ser una contradicción, porque la intersección de las diagonales no puede pertenecer a p_i y a p_j al mismo tiempo.

Tomemos la relación $\preceq = \bigcup_i \preceq r_i$. Aunque no cumple que sea un orden parcial por no ser necesariamente transitiva, es antisimétrica y reflexiva.

Ahora podemos ampliar esta relación de forma tal que la hagamos transitiva, o sea, si $p_i \preceq$p$_j$ y $p_j \preceq p_k$, con p_i, p_j, p_k diferentes, hacer $p_i \preceq$p$_k$ en la relación, no siendo posible porque se estaría formando un ciclo de precedencia entre los polígonos, lo cual sólo sería posible si se recorriera el plano alrededor del punto, pero como estamos trabajando dentro de un ángulo determinado esto no va a ocurrir. Finalmente para los pares de polígonos que no son comparables establecemos un orden cualquiera y de esta forma queda un orden topológico entre todos los polígonos, lo cual nos permite que $\preceq$sea un orden total.

Lqqd.

Analizemos ahora la desventaja de tener polígonos vacíos. Si nos imaginamos que se pudiera hacer una partición del plano en regiones convexas, donde no hayan regiones vacías, entonces sería bueno analizar el contraejemplo en la fig. 2.6.

Por lo tanto, en una estructura como esta es necesario tener regiones que queden vacías, lo cual, como es lógico, no es un aspecto conveniente. Analicemos entonces el orden de las regiones vacías en función de la cantidad de polígonos convexos en el espacio. Veamos que la cantidad máxima de regiones vacías necesarias depende de la cantidad de casos críticos, como el del contraejemplo, que aparezcan en el espacio. Esta cantidad es O(n) donde n es la cantidad de objetos en el espacio como aparece en la fig.2.7, es el caso bien crítico:

Tengamos en cuenta ahora que, debido a problemas de la aritmética utilizada y los algoritmos empleados, la cantidad de regiones vacías obtenidas es un poco mayor que la mínima necesaria. Esto lo veremos con más detalles en el próximo capítulo donde veamos aspectos de la aritmética empleada y los algoritmos para insertar manzanas u objetos, así como la optimización de la estructura, pág.31. Lo siguiente será demostrar que aun en el peor de los algoritmos utilizados, y sin optimización alguna, la cantidad de regiones vacías sigue siendo O(n).

Supongamos que tenemos un algoritmo que particiona el mapa en regiones convexas pero sin adicionar vértices a la estructura, tan sólo los vértices que existen en los objetos, que no son más que polígonos. Esto es posible debido a que una triangulación del espacio, dado este conjunto de vértices, pero sin separar el interior de los polígonos en triángulos, ya sería una división en regiones convexas, donde las regiones vacías son triángulos y las regiones que contienen objetos coinciden con los polígonos contenidos. Esta división, que sería el resultado de este sencillo algoritmo, pero que produce el máximo número de regiones vacías posibles, al cual llamaremos t, sigue siendo O(n).

Esto se debe a que si acotamos la cantidad de vértices máxima que puede tener un polígono por una constante suficientemente grande, lo cual es posible, porque la entrada de datos en este problema, que son los objetos, puede crecer todo lo que quiera, que siempre se puede encontrar un valor razonable para acotar los vértices de un polígono de la entrada, basándonos en un valor estadístico obtenido a partir de un número grande de mapas reales;

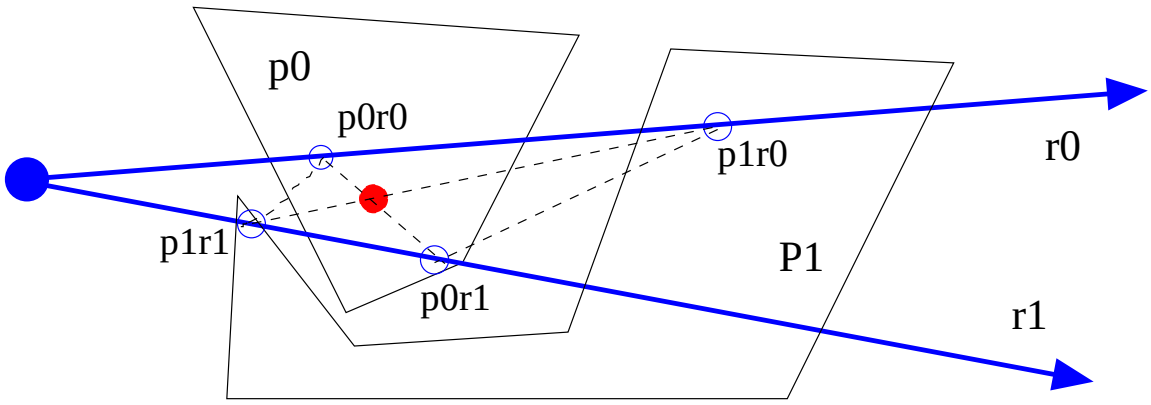

Figura 2.5: Ilustración de la demostración por el absurdo, en la ordenación de regiones sobre dos rayos distintos.

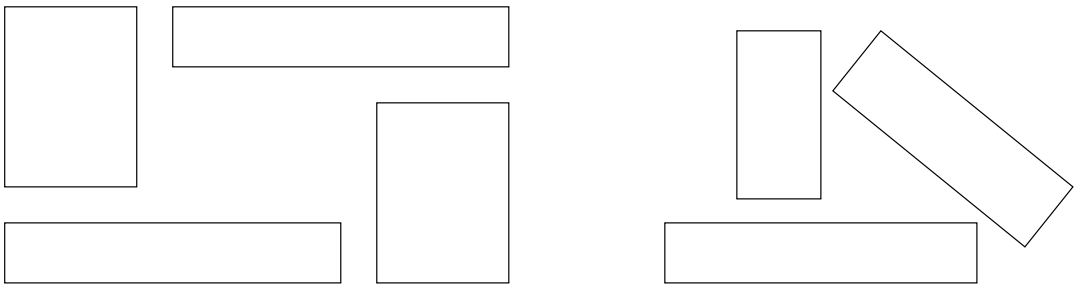

Figura 2.6: Mapa donde no es posible la separación en regiones convexas con exactamente un objeto en su interior.

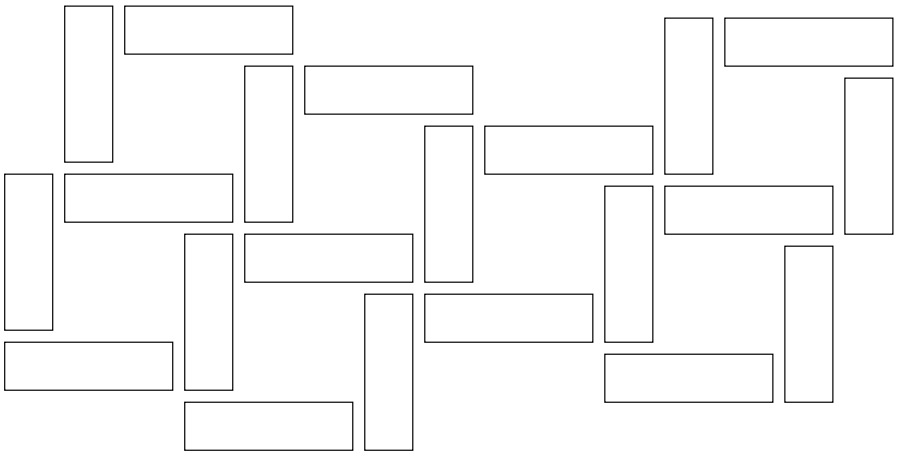

Figura 2.7: Caso crítico en cuanto a cantidad de regiones vacías.

entonces podemos obtener que si V es el conjunto de vértices |V|=O(n) y suponiendo que todos los polígonos también queden divididos en triángulos, obtendríamos una cantidad tt de regiones, que cumpliría $tt \geq t$. Pero resulta que por el teorema de Euler tenemos que |A|-|V|+2=tt donde A es el conjunto de aristas para la triangulación, pero además tenemos que $|A| \leq 3 \cdot tt - 6$ por lo que |A|=O(tt), pero además $|A| \geq tt$ ya que evidentemente para tener tt regiones se necesitan más de tt aristas entonces se cumple que |A|=$\Omega(tt)$, por lo que |A|=$\Theta(tt)$. Quedando entonces |V|=$\Theta(tt)$+2, o sea tt=Θ(|V|). Por lo que se deduce que t=O(|V|), o sea, que la respuesta para este algoritmo sencillo planteado anteriormente es también de orden lineal. Finalmente tenemos que cualquier algoritmo que se emplee para hacer la partición en regiones convexas generaría una cantidad de regiones vacías que sería de un orden lineal.

En este trabajo se han obtenido varias técnicas para realizar esta partición del espacio y finalmente se ha utilizado una técnica combinada, donde se combina la separación de los objetos cuando están contenidos en una región convexa mediante la partición de esta región convexa utilizando un segmento cuyos extremos puede que no coincidan con vértices de los objetos del espacio, pero si se garantiza que tengan coordenadas enteras. Ver algoritmo en el ep.3.4. La otra técnica que se combina es emplear sólo los vértices de los objetos para realizar la partición. Todo esto puede verse con más detalle en el ep.3.3, donde se trata la operación de inserción en la convex DCEL. Estas técnicas de particionado se mezclan con varias técnicas de optimización, las cuales pueden ser utilizadas periódicamente o en etapas de postdigitalización[1] finalmente podiéramos obtener una solución que tenga casi tan pocas, o tan pocas, regiones vacías como las necesarias. Habiendo utilizado una combinación de todas las técnicas anteriormente mencionadas se puede llegar a muy buenos resultados.

Una característica muy importante que cumplen todas estas técnicas de partición del espacio es que en cada momento toda la estructura es completamente consistente y esta preparada para hacer inserciones de nuevos objetos dentro del espacio, así como eliminar alguno de los ya existentes.

[1]Etapa en la que se reajustan los datos de forma conveniente. Es un trabajo automático que realiza el sistema.

Capítulo 3

Formación de la partición convexa.

3.1 Preliminares

Hemos utilizado un gran número de preliminares en lo que respecta a estructuras de datos y algoritmos muy conocidos por las personas que han incursionado en esta materia, entre los cuales tenemos ordenaciones, búsquedas binarias o dicotómicas, colas, pilas, árboles binarios balanceados de búsqueda, y muchos otros [AHU83]. Aun así merece la pena explicar aquellos que son más específicos dentro de la Geometría Computacional. Veamos entonces algunos a continuación.

3.1.1 Trazado de rayos (Ray Trace)

Esta técnica consiste en partir de un punto en una dirección determinada e ir analizando diferentes eventos que ocurren sobre el rayo trazado, el rayo a veces se traza hasta otro punto conocido, o se asume sencillamente que se traza hasta infinito. De forma general se pudiera describir este tipo de algoritmo de la siguiente forma:

P1 Trazar el rayo desde un punto inicial

P2 Hacer incidir eventos sobre el rayo, los cuales suelen ser intersecciones con otros segmentos. En ocasiones estos eventos avanzan a lo largo del rayo, lo cual brinda una convergencia y por tanto una condición de parada al algoritmo, aunque otras veces no existe este avance de forma monótono.

3.1.2 Barrido del plano (Plane Sweep)

Este es uno de los paradigmas más ampliamente utilizada en la geometría computacional plana, o sea, trabajando sobre espacios de dos dimensiones. Consiste en una línea de frente que barre todo el espacio donde están contenidos los datos. Durante el barrido se va pasando por diferentes estados, los cuales cambian con la aparición de eventos que ocurren en la línea de frente. Los algoritmos de barrido se basan en una característica invariante de los algoritmos incrementales: En cada momento la línea de frente ha acumulado la respuesta al problema para el subconjunto de datos que se ha analizado. Una vez barridos todos los datos se tiene la respuesta del problema en las manos.

El éxito de los algoritmos de barrido se debe a un truco ingenioso, tratando las abscisas X como la dimensión-tiempo, por lo que un problema de 2 dimensiones(2-d) es reducido a un problema de 1 dimensión. Para datos de una dimensión conocemos varias estructuras de datos eficientes, gracias a esto el algoritmo tiene éxito al resolver n problemas de 1 dimensión, cada uno de los cuales puede involucrar n elementos en tiempo $\mathrm{O}(n \log n)$ en vez de $\mathrm{O}(n^2)$ o $\mathrm{O}(\log n)$ en vez de $\mathrm{O}(n)$, entre otros. Desafortunadamente esta idea no podemos generalizarla eficientemente a otras dimensiones. Si barremos un espacio de 3 dimensiones con un plano, transformaríamos el problema en una secuencia de problemas de 2-d, lo cual generaría búsquedas en el espacio 2-d, donde es difícil encontrar estructuras de datos que respondan a dichas búsquedas en tiempo logarítmico [MN99], [vKNRW97].

Descripción de forma general de un algoritmo que utilice el barrido del plano.

P1 Tomar un punto inicial y una línea imaginaria que pase por este punto. Esta línea es la que realiza el barrido del espacio.

P2 Avanzar hasta el próximo evento y realizar un análisis del estado.

P3 Repetir P2 hasta que se acaben los eventos o el espacio analizado.

En casos especiales de barrido se completa un ciclo, como ejemplo tenemos el caso del barrido circular, con lo cual termina el algoritmo. Aunque estos no son los tipos más comunes de barrido, también son considerados como tal.

3.1.3 Inundación(Flood Fill)

Esta técnica más bien corresponde al tema de grafos aunque es muy empleada en la geometría computacional, por el gran uso que hace de estos. Dada la relación de adyacencia que podemos encontrar en un grafo entre distintos elementos como son las regiones, en el caso de los grafos planares, los vértices y las aristas y según los tipos de recorridos en grafo podemos hacer varios tipos de inundación, los cuales pueden ser primero en profundidad o por niveles. Describamos entonces de forma general como sería un algoritmo de Inundación.

P1 Se asume un elemento inicial

P2 Se analiza el elemento actual

P3 Se realizan marcas en la estructura o estructuras auxiliares para que no se repitan elementos

P4 Se expande hacia los vecinos o adyacentes si cumplen con una determinada condición y se repite desde el paso P2 para cada uno de ellos.

3.1.4 Determinante de Giro

Dados 3 puntos en el plano $p_1=(x_1,\ y_1)$, $p_2=(x_2,\ y_2)$ y $p_3=(x_3,\ y_3)$.

$$G = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

G es el valor del determinante de la matriz representada. Al calcular el determinante de esta forma se obtiene que G < 0 si al ir en la dirección p_1, p_2 se hace un giro a la derecha para llegar a p_3, G > 0 si se hace un giro a la izquierda, y G $= 0$ si no se realiza giro o se realiza un giro de 180^o. Este Giro toma en cuenta estos valores tomando en cuenta las coordenadas de los vértices en un sistema de coordenadas cartesianas. Si lo analizamos en un sistema de coordenadas como el sistema de coordenadas de la pantalla, donde el (0, 0) está en la esquina superior izquierda y las abscisas aumentan hacia la derecha y las ordenadas aumentan hacia abajo, entonces los valores de los giros son de signo opuesto a los antes vistos.

3.1.5 DCEL.

La principal estructura de datos que hemos utilizado es una estructura que tiene cierta complejidad pero que almacena una gran cantidad de información en ella, la cual es suficiente para soportar una partición del plano en regiones convexas y poder realizar operaciones sobre ella de forma muy eficiente. Esta estructura es la denominada por nosotros como convex DCEL y constituye un tipo de grafo especial con un conjunto de operaciones definidas sobre la misma.

Una DCEL(doubly connected edge list) es un grafo orientado a sus aristas, o sea, básicamente es una lista de aristas en las que se almacenan las regiones que tienen a ambos lados, los vértices que tienen en sus extremos, y se enlazan a otras aristas de la siguiente forma: enlaces a la anterior y la próxima arista si recorremos cada una de las dos regiones a sus lados, en contra de las manecillas del reloj. Este sentido de dirección es sólo un convenio adoptado por nosotros, pero en realidad puede tomarse cualquiera de los dos sentidos siempre y cuando se sea consecuente con este. Sin entrar en detalles de implementación es válido destacar que generalmente, como en nuestro caso, se trata las aristas como dos medio-aristas

que contienen los datos relacionados con una de las regiones que está a un lado de la misma. Mediante estas medio-aristas y estos enlaces se pueden hacer recorridos de forma eficiente por todas las aristas alrededor de un vértice, o alrededor de una región, así como hacer otros recorridos de forma eficiente sobre las distintas regiones del espacio. Los vértices tienen coordenadas en el plano, las aristas topológicamente son segmentos que unen los puntos del plano representados por los vértices, entonces las regiones no son más que polígonos.

La estructura consta entonces de una lista de medio-aristas.

Las medio-aristas a su vez contienen referencias a sus vértices de origen, su otra medio-arista, su próxima medio-arista en sentido a favor de las manecillas del reloj y en contra, y por último una referencia a la región que ella limita.

Las regiones pueden tener en su interior un objeto del plano.

Los vértices no son más que puntos en el plano.

Aunque estas estructuras pueden tener otros atributos o propiedades que sean necesarias desde el punto de vista de implementación, no es necesario esclarecerlo aquí para la explicación de los algoritmos, sino en el próximo capítulo, cap.5, donde tratamos los detalles principales de diseño e implementación.

Veamos ahora las operaciones principales que se han implementado sobre esta DCEL.

-Inserción de aristas: Consiste en añadir las dos medio-aristas a la lista y actualizar las referencias involucradas entre las aristas, además, actualizar las referencias a regiones, tener en cuenta el caso en que se divide una región en dos. El costo temporal de este algoritmo pudiera considerarse constante $\Theta(1)$ ya que no depende de la cantidad de objetos en el espacio.

Algoritmo A:

Entrada de datos: Dos vértices extremos, v_1, v_2.

Salida: La lista de medio-aristas actualizada, con la nueva arista.

P1 Crear 2 nuevas medio-aristas y añadirlas a la lista

P2 Establecer una relación de referenciación entre ellas

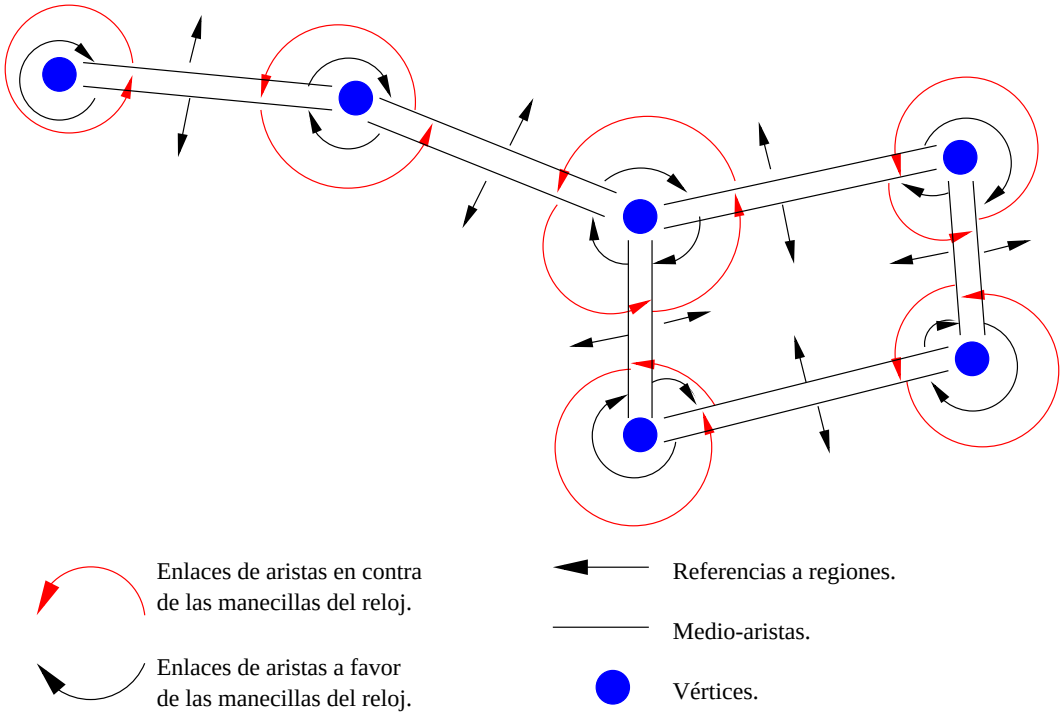

Figura 3.1: Ejemplo de DCEL.

P3 if($v_1$está en la estructura)
buscar entre qué otras medio-aristas incidentes a $v_1$quedaría la nueva arista
actualizar los enlaces dependiendo de cual este a la izquierda y cual a la derecha
P4 if(v_2 está en la estructura)
buscar entre qué otras medio-aristas incidentes a v_2 quedaría la nueva arista
actualizar los enlaces dependiendo de cuál este a la izquierda y cuál a la derecha
P5 if(v_1y v_2 estaban en la estructura)
Crear una nueva región a la derecha de la arista v_1, v_2.
Actualizar las referencias a regiones de las demás medio-aristas de la región
else
Actualizar las referencias a regiones de ambas medio-aristas como la región a la que pertenecen.

-Borrado de aristas: Es el proceso inverso a la inserción, se eliminan las dos medio-aristas asociadas, pero antes se vuelven a arreglar los enlaces de las demás medio-aristas en la lista, tal como si estas no existieran, se eliminan los vértices en sus extremos en caso que su grado sea 1, y se analiza si hay dos regiones que se convierten en una sola. El costo temporal de este algoritmo también pudiera considerarse constante $\Theta(1)$ ya que no depende de la cantidad de objetos en el espacio.

Algoritmo:
Entrada de datos: Posición m de una medio-arista a en la lista.
Salida: La lista de medio-aristas actualizada, con una arista menos.

P1 if(el vértice asociado a a tiene grado 1)
Eliminar dicho vértice
P2 if(el vértice asociado a la otra medio-arista de a tiene grado 1)
Eliminar dicho vértice
P3 Restaurar los enlaces de las medio-aristas cuyas referencias a próximas en contra y a favor de las manecillas del reloj coincidan con a o su medio-arista asociada.
P4 If(la arista formada por a y su asociada separa regiones distintas)
Eliminar la región asociada a a

P5 Eliminar a
P6 Eliminar la medio-arista asociada a a

Veamos, sin entrar a analizar los algoritmos, otras operaciones sobre la estructura:
-División de una arista dado un vértice: Consiste en dividir la arista, o lo que es lo mismo las dos medio-aristas que la forman de manera que quede insertado el vértice v, o sea, si la arista era el segmento comprendido entre v_1 y v_2 ahora quedaran dos aristas de la forma (v_1, v) y (v, v_2).

-Salvar la estructura: Consiste en escribir a disco todas las aristas, los vértices y las regiones, para lo cual se realizan recorridos de inundación sobre la estructura.

-Recuperar la estructura: Consiste en el proceso inverso a salvar, pues este es leer de disco las aristas, los vértices y las regiones. Este algoritmo al igual que el anterior tienen un costo temporal Θ(n) donde n es la cantidad de objetos que hay en el espacio, ya que habíamos visto que tanto la cantidad de vértices, como de aristas y regiones era Θ(n). Pudiera parecer algo lento desde el punto de vista de que n puede ser muy grande, pero hay que recordar que este no es un algoritmo que tenga que dar respuesta en tiempo real como otros que tienen que ver con la digitalización, las consultas y las visualizaciones.

3.2 Aritmética empleada

Cuando hablo de la aritmética empleada me refiero al tipo de números que se va a almacenar y posteriormente con el que se va a realizar los cómputos. En nuestro caso nos interesa fundamentalmente las coordenadas en las que se ubican los puntos en el plano. Estas coordenadas van a ser de tipo enteras, usando la representación estándar de C++ de enteros de 32 bits. Esta aritmética conlleva a que no se creen ni propaguen errores de redondeo como los que ocurre al trabajar con números de punto flotante. Sólo hay que tener cuidado de que no ocurra "overflow"[1] al realizar alguna operación y si es necesario, trabajar con algún entero de

[1] A veces conocido en español como desbordamiento, ocurre al hacer una operación sobre un tipo de dato cuya cantidad de bits no es suficiente para almacenar el resultado.

mayor tamaño para los cálculos intermedios. Trabajando con una escala adecuada al nivel de detalle que se quiere representar en los objetos de la ciudad, se puede hacer la entrada y la salida de datos directamente con coordenadas enteras transformadas directamente de la pantalla u otro medio de digitalización o vectorización, para ser más específicos, lo cual constituye otra ventaja. Y ya que hemos visto la robustez de esta aritmética y su correspondencia con la entrada y la salida, debo destacar que una amplia ventaja que presenta el cómputo con enteros sobre el cómputo con números de punto flotante es la rapidez con que ocurren las operaciones. Por esto, salvo en algunas pocas operaciones realizadas ya en la visualización en tres dimensiones todos los cálculos ocurren con números enteros.

El trabajo con esta aritmética es debido al hecho de que en los algoritmos no se generan nuevos vértices en el espacio, salvo en escasas ocasiones con algunas técnicas que veremos más adelante para generar vértices de puntos enteros con el objetivo de minimizar regiones vacías. No se calculan intersecciones, sino generalmente sólo importa si ocurre o no alguna intersección. Estos algoritmos para ver si ocurre intersección como tantos otros que explicaré más adelante hacen un amplio uso del determinante de giro explicado previamente en este capítulo.

3.3 Propiedades y operaciones de la partición convexa (convex DCEL).

Analicemos entonces ya la convex DCEL que particiona el mapa, cuya estructura es la misma, pero cuenta con un mayor conjunto de operaciones. Estas operaciones aprovechan la particularidad de que las regiones son polígonos convexos, aunque pueden ser instrumentadas de otra forma en una DCEL común. Las equivalentes de las operaciones que se verán a continuación, pero aplicadas a una DCEL común, no fueron tratadas en este trabajo, por eso no se explicaron anteriormente.

-Localización de un punto determinado: Consiste en localizar un punto dado como dato, o sea, ver en que región se encuentra situado dicho punto.

Algoritmo:

Entrada de datos: Punto p que quiere ser localizado.

Salida: Región en la que se encuentra el punto.

P1 p_0 = punto inicial, que sea vértice de alguna medio-arista.

P2 while($\exists p_1$ | d(p_1,p) < d(p_0,p), con p_1 adyacente a p_0)

$p_0 = p_1$

P3 Realizar un trazado de rayo desde p_0 hasta p, donde los eventos son: salir de una región. Se sale de una región una sola vez dada la propiedad 2 mencionada en ep.2.2. La búsqueda del punto de salida se realiza recorriendo la frontera de la región desde la arista o punto por donde se entró. Si al recorrer la frontera, se vuelve al punto por donde se entró a la región, acabar el trazado de rayo.

P4 Retornar la última región visitada.

En cuanto a costo temporal se puede ver que si se escoge un punto inicial, que sea el vértice más alejado, lo cual constituiría el caso peor; entonces habría que atravesar tantas regiones como puedan intersectar una línea recta en la estructura. Si viéramos la estructura como un espacio cuadrado y cuadriculado con n cuadrículas, una línea recta podría atravesar $\sqrt{n}$ regiones. Como esto lo podemos ver como un caso aproximado a lo que ocurre en realidad. Se puede decir que el costo temporal del algoritmo es $\Theta(\sqrt{n})$. Aun siendo este, un algoritmo que tiene un buen comportamiento, para resolver este problema se pueden obtener un algoritmo en tiempo $\Theta(\log^2 n)$ con estructuras adicionales que se añaden a la DCEL, lo que significa memoria adicional, y que no es necesaria con el algoritmo antes descrito pues su costo espacial es constante.

-Cubrimiento (Solapamiento) de un polígono cualquiera por regiones de la convex DCEL: Consiste en ver cuales son las regiones que se intersectan con una zona de forma poligonal cualquiera en la estructura, o sea, finalmente es un problema de solapamiento de polígonos. Analizando que, lo que hace este algoritmo es local a la zona que abarca el polígono dado como dato, y que este pudiera abarcar a toda estructura, el costo temporal del mismo es Ω(n) debido a que está acotado inferiormente en tiempo por su complejidad combinatorial que es Θ(n). Finalmente quedaría que en dependencia del tamaño del polígono dado por datos

el costo temporal sería $\Theta(\max(\sqrt{n},\mathrm{M}))$, M es la complejidad combinatorial de los objetos involucrados, y $\sqrt{n}$ debido a que hay que localizar un primer punto al menos, ver decripción del algoritmo en la fig.3.3.

Algoritmo:

Entrada de datos: polígono simple p.

Salida: Conjunto C de medio-aristas que se cruzan con p o están contenidas dentro de p.

A partir de estas aristas fácilmente podemos obtener las regiones.

P1 Dividir el polígono p en polígonos convexos, obteniendo un conjunto de polígonos convexos PC.

P2 $\forall pc_i$ tal que $pc_i \in$PC, hacer:

P2.1 $\forall$segmento en pc_i, nombrémoslo (pc_{i_j}, $pc_{i_{j+1}}$), hacer:

P2.1.1 Hacer un trazado de rayo desde pc_{i_j}hasta $pc_{i_{j+1}}$donde los eventos son salir de una región, Añadir al conjunto C cada una de las medio-aristas cortadas por el rayo, o aquellas que el rayo intersecta en un extremo y quedan hacia el interior del polígono.

P2.2 Hacer una inundación para obtener todas las aristas en el interior de pc_i, y Añadir estas aristas al conjunto C.

P3 retornar C

-Inserción de objetos: Consiste en insertar nuevos objetos, o sea, polígonos, en las regiones de la estructura, llamemos O al conjunto de objetos que han sido insertados en el espacio. El costo temporal en este caso sería del mismo orden que el del algoritmo de cubrimiento analizado anteriormente, todo lo que hay que analizar al respecto es que la intersección de polígonos no depende de n donde n es la cantidad de objetos en el espacio, y que el algoritmo de eliminación de huecos empleado es lineal, o sea, de orden Θ(M) donde M es la cantidad de objetos contenidos en la gran región cuyo interior contiene los huecos u objetos. A su vez esta cantidad de huecos y la cantidad de aristas que conforman esta gran región es lineal con respecto a la cantidad de aristas en el cubrimiento. Ver figura 3.4. Para mayor simplicidad veamos hasta el momento los objetos como polígonos convexos.

Algoritmo:

Entrada de datos: el nuevo polígono p a insertar.

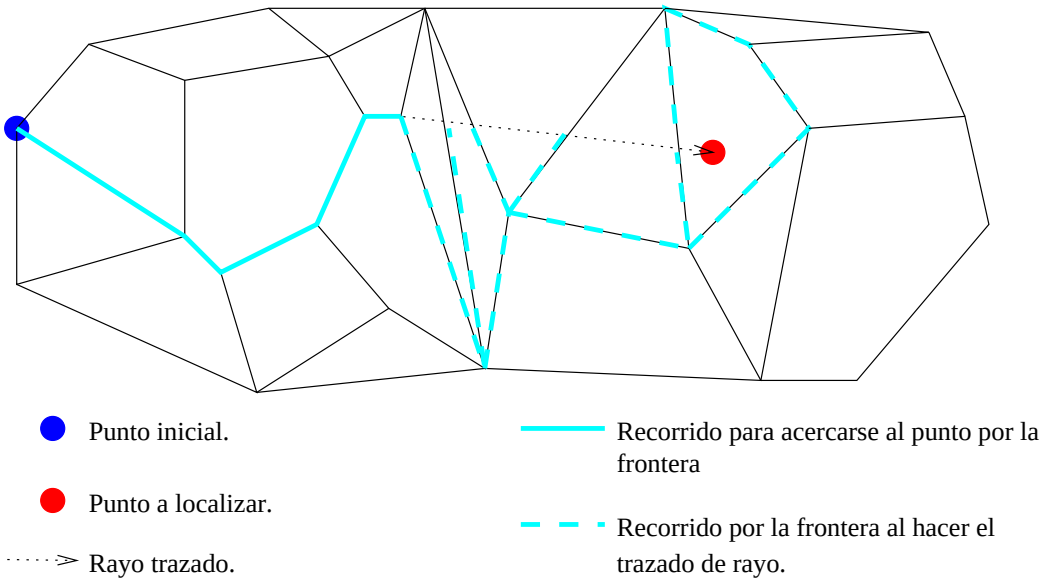

Figura 3.2: Localización de puntos.

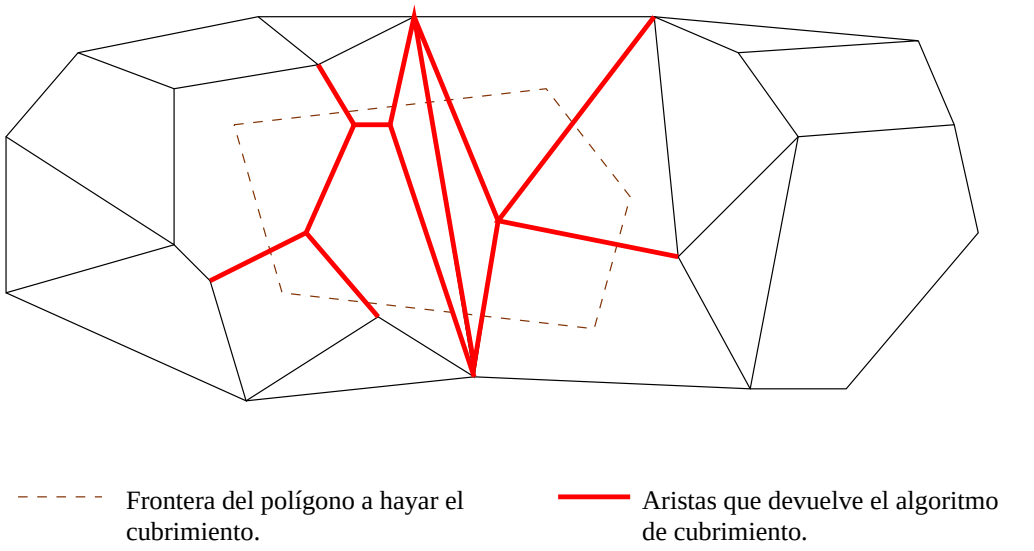

Figura 3.3: Cubrimiento de un polígono mediante regiones, aristas intersectadas.

Salida: La estructura modificada con el nuevo polígono añadido.

Se hace uso de algoritmos que veremos más adelante en el ep.3.4.

P1 L = lista de medio-aristas obtenidas del cubrimiento de p en la estructura
P2 if(L ==$\emptyset$)
P2.1 if((reg = localizar p_1, primer punto de p) == región vacía)
P2.1.1 Insertar el objeto p en reg
else
P2.1.2 if(Se intersecta p con el polígono existente en reg)
P2.1.2.1 Abortar
P2.1.3 if(Se puede separar p de polígono existente en reg)//Usando el algoritmo de búsqueda de soluciones enteras para separar dos polígonos convexos contenidos en otro polígono convexo pág.37.
P2.1.3.1 Insertar el segmento que los separa como arista en la estructura
P2.1.3.2 retornar
else
P2.1.3.3 goto P3
else
P2.2 if(Se intersecta p con algún polígono existente en alguna región asociada a alguna medio-arista de L)
P2.2.1 Abortar
P2.3 Borrar todas las medio-aristas que pertenecen a L
// En este punto p está dentro de una región poligonal donde hay varios otros objetos por separar unos de otros en regiones diferentes, ver figura 3.4.
P3 Tratamos el problema como una región con huecos y aplicamos el algoritmo de la eliminación de huecos en forma de polígonos convexos dentro de un polígono simple. Insertar todas los segmentos resultantes como aristas en la estructura.
P4 Particionar todos los polígonos no convexos obtenidos en el paso anterior en convexos e insertar los segmentos resultantes como aristas en la estructura

Es posible emplear este algoritmo debido a que al separar un convexo, en dos partes,

mediante un segmento cada una de las partes constituye un nuevo polígono convexo, por lo cual aplicar este algoritmo conserva las propiedades de la estructura.

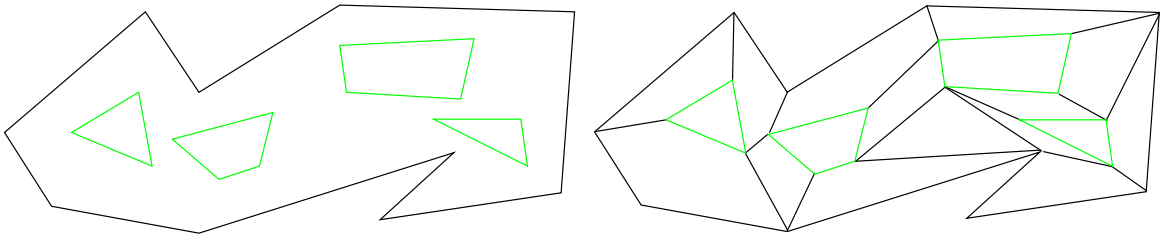

Figura 3.4: División en convexos de una región no convexa con huecos convexos.

En los casos en que los objetos son polígonos no convexos, sólo habría que hacer el siguiente reajuste al algoritmo anterior:

P1 Sea HC el conjunto de las envolturas convexas del conjunto H de huecos.

P2 $\forall$ par hc_i, hc_j de polígonos convexos, o envolturas hacer:

P2.1 if(Se intersectan hc_i y hc_j)

P2.1.1 Hacer h_iC y h_jC los conjuntos de polígonos convexos obtenidos al particionar h_i y h_j en polígonos convexos.

P2.1.2 Hacer H = (H $\{h_i, h_j\}$) $\bigcup$ h_iC $\bigcup$ h_jC

else

P2.1.3 Hacer H = (H $\{h_i, h_j\}$) $\bigcup$ $\{ hc_i, hc_j\}$

Veamos ahora también, sin entrar a analizar el algoritmo, otra operación sobre la estructura:

-Optimización: Consiste en analizar las regiones vacías que pueden ser eliminadas de la estructura y proceder a eliminarlas. Este algoritmo podría volverse más complejo debido a la aritmética empleada, lo que significa trabajo con coordenadas enteras. Existen muchos casos en los que se podría minimizar la cantidad de regiones vacías si escogiéramos vértices de la DCEL en puntos de coordenadas racionales situados en alguna posición conveniente del espacio. Es evidente que al restringir a enteros el conjunto de coordenadas la cantidad de particiones posibles del espacio se reduce también, por esta razón se puede perder optimalidad en la estructura.

Entre las posibles estrategias de funcionamiento de un algoritmo de este tipo podemos destacar algunas más sencillas, donde sólo se eliminen aristas, garantizando siempre la convexidad de las regiones y la unicidad de los objetos en su interior(Esta estrategia ya ha sido implementada), y pudiéramos tener casos más complejos donde se pueden eliminar y crear vértices y aristas. El problema de la creación de vértices o su cambio de posición, lo cual pudiera verse como una eliminación y una creación del mismo, adolece del problema de la aritmética antes descrito, aun así, se puede, en muchos casos, realizar cambios de la estructura de este tipo, lo cual permitiría finalmente llevar a cabo reducciones de polígonos, o sea, de regiones a simples vértices o aristas.

Su costo temporal de ejecución no sería lo más importante puesto que su objetivo es sólo buscar eficiencia, por lo que podría utilizarse solamente en etapas de postdigitalización, aun así, no debería resolverse con un algoritmo exponencial.

3.4 Otros algoritmos empleados en la inserción de objetos y formación de la estructura.

-Intersección de polígonos convexos: Este es es el método para resolver el problema de decidibilidad que consiste en chequear si dos polígonos convexos se intersectan, asumiendo que si la intersección de los mismos sólo ocurre en sus fronteras no habría intersección entre ellos, o sea, la intersección de dos polígonos convexos sería cierta si y sólo si el área de su intersección, AI, cumple que AI > 0. La complejidad temporal de este algoritmo es del orden de $\Theta(n+m)$ donde n y m son las cantidades de vértices que componen cada polígono, por lo que alcanza la cota mínima de tiempo. Siendo la complejidad espacial de orden $\Theta(n+m)$, lo que significa que es lineal también. El algoritmo se basa en que si dos polígonos convexos se intersectan es porque se intersectan los segmentos de uno con el otro o porque hay un punto del polígono que aparece segundo por las X, de menor a mayor, en el interior del primero.

Algoritmo:

Entrada de datos: dos polígonos p_0 y p_1

Salida del algoritmo: Si o No.

P1 Realizar un barrido del plano, donde los eventos son los vértices de ambos polígonos. Estos aparecen ordenados de acuerdo a su coordenada X en el plano. Llamemos p_{i_0} al primer polígono encontrado haciendo el barrido. Para cada evento v hacer:

P1.1 if($v \in p_{i_1}$ && v está entre los dos últimos segmentos por encima y por debajo de p_{i_0})

P1.1.1 retornar Si

else

P1.1.2 if(se intersecta algún par de segmentos de p_{i_0} con alguno de p_{i_1}, teniendo en cuenta que analizamos el último de arriba y de abajo por cada uno)

P1.1.2.1 retornar Si

P2 retorna No

Hay otros casos particulares que también quedan completamente cubiertos con nuestro algoritmo como son el caso en que coinciden vértices y lados de un polígono con vértices y lados del otro. Para estos casos se tiene en cuenta la cantidad de coincidencias vértice-vértice o vértice-polígono, analizando todos los casos posibles.

-Intersección de polígonos no convexos: Este problema es de una mayor complejidad aun que el mencionado para polígonos no convexos. Debido a que los polígonos empleados no presentan un número grande de vértices hemos optado por implementar un algoritmo sencillo basado en el algoritmo anterior para intersectar polígonos convexos, el problema es igualmente determinar si el área de la intersección entre los dos polígonos (Ai) cumple Ai>0.

Consiste en particionar ambos polígonos en polígonos convexos; así obteniendo los conjuntos de polígonos convexos C1 y C2 respectivamente para el 1^{er} y 2^{do} polígono P1 y P2, en la entrada de datos. Entonces se intersectan los polígonos dados como datos si y sólo si hay intersección entre los convexos c_1 y c_2 para algún c_1 que pertenece a C1 y c_2 que pertenece a C2. Este problema se puede resolver con un algoritmo de orden $\Omega(n \log n)$ lo cual es la cota mínima para el problema, la complejidad temporal de nuestro algoritmo depende del algoritmo empleado para particionar el polígono en convexos, además depende de la cantidad de polígonos obtenidos y la cantidad de puntos que tengan los mismos. Finalmente

quedaría un algoritmo de orden cuadrático en el caso peor. El algoritmo empleado para hacer la partición en convexos es de orden cuadrático, no de orden $\Theta(n \log n)$ como es la cota mínima para este problema, pero trabaja muy rápido en el caso de polígonos sencillos y con poca cantidad de puntos como los que se utilizan en nuestro caso. Una vez efectuada la división en convexos lo que resta es de orden cuadrático también, aunque de primera vista parece ser cúbico, o peor. A continuación está incluida la demostración.

Propiedad: Primero veamos que para cualquier polígono p de k vértices, al ser particionado en c polígonos convexos, obtendríamos a lo sumo *3k - 6* segmentos(incluyendo la frontera del no convexo), que sería la cota superior de segmentos obtenidos de una triangulación sobre el conjunto de vértices, en este caso, del polígono p. Cada uno de los segmentos puede pertenecer a lo sumo a 2 convexos, por lo que al sumar $\sum_{i=1}^{c} n_i \leq 6k - 12 = \Theta(k)$, donde n_i es la cantidad de segmentos que constituyen la frontera de *i-ésimo* polígono convexo, con *i=1..c.*

Veamos ahora el tiempo t que consume realizar todas las intersecciones, donde P1 y P2, de k_1 y k_2 vértices, quedan particionados en c_1y c_2 convexos respectivamente:

$t=\sum_{i=1}^{c_1} \sum_{j=1}^{c_2} t_{1_i 2_j}$, donde $t_{1_i 2_j}$ es el tiempo que consume la intersección del polígono *i-ésimo* de P1 y el *j-ésimo* de P2.

$t=\sum_{i=1}^{c_1} \sum_{j=1}^{c_2} \Theta(n_{1_i} + n_{2_j})$, ya que habíamos visto que el algoritmo de intersección de polígonos convexo es de orden lineal, donde n_{1_i}es la cantidad de segmentos que tiene el polígono *i-ésimo* de P1, y n_{2_j}es la cantidad de segmentos que tiene el polígono *j-ésimo* de P2.

$t \leq \sum_{i=1}^{c_1} \sum_{j=1}^{c_2} cte(n_{1_i} + n_{2_j}) =$

$= cte(\sum_{i=1}^{c_1} \sum_{j=1}^{c_2} n_{1_i} + \sum_{i=1} \sum_{j=1} n_{2_j}) =$

$= cte(\sum_{j=1}^{c_2} \sum_{i=1}^{c_1} n_{1_i} + \sum_{i=1}^{c_1} \sum_{j=1}^{c_2} n_{2_j}) =$

$= cte(\sum_{j=1}^{c_2} \Theta(k_1) + \sum_{i=1}^{c_1} \Theta(k_2)) =$

$= cte_1 \cdot c_2 \cdot k_1 + cte_2 \cdot c_1 \cdot k_2$, esta última igualdad se debe a la propiedad antes descrita, pág.34.

$t \leq cte_1 \cdot k_2 \cdot k_1 + cte_2 \cdot k_1 \cdot k_2 = \Theta(k_1 k_2)$.

Por lo que podemos decir que el problema es de orden cuadrático.

Lqqd.

Algoritmo:

Entrada de datos: P1 y P2 polígonos simples.
Salida del algoritmo: Si o No.

P1 Particionar los polígonos P1 y P2 en convexos obteniendo C1 y C2
P2 $\forall p_{1_i} \in$C1 y $\forall p_{2_j} \in$C2 hacer:
  P2.1 if(Se intersectan p_{1_i} y p_{2_j})
    P2.1.1 retorna Si
P3 retorna No

-Envoltura convexa de un conjunto de puntos o polígono no convexo: Se entiende por envoltura convexa al menor polígono convexo, o sea, el de menor área, que contiene todos los puntos dados por datos. El costo temporal de este algoritmo es $\Theta(n \log n)$ por lo que es muy eficiente, donde n es la cantidad de puntos de la entrada.

Algoritmo:
Entrada de datos: Conjunto de puntos en el plano.
Salida: Envoltura convexa.

P1 Ordenar los puntos por las X´s.
P2 Formar un segmento s con el primer y último punto
P3 Para cada punto p que está sobre s, tomados de mayor a menor X hacer:
  P3.1 if(Giro(anterior de p, p, siguiente a p) == derecha)
    P3.1.1 Elimino p
    P3.1.2 p = anterior a p
P4 Para cada punto p que está bajo s, tomados de mayor a menor X hacer:
  P4.1 if(Giro(anterior de p, p, siguiente a p) == derecha)
    P4.1.1 Elimino p
    P4.1.2 p = anterior a p
P5 Formar un polígono P con los puntos que quedaron sobre el segmento y adicionarle los que quedaron por debajo, en ese orden.
P6 Retornar P

-Eliminación de huecos en forma de polígonos convexos dentro de un polígono simple:

Este algoritmo resuelve la partición de un polígono simple con huecos en su interior, que en este caso son convexos, en un conjunto de polígonos sin huecos, entre los cuales se encuentran los polígonos convexos dados como datos. Su costo temporal es $\Theta(n \log n)$ por lo que también es muy eficiente. Para esto hay que utilizar un método de ordenación interna cuyo orden sea $\Theta(n \log n)$, además de una estructura de ordenación que permita inserción, borrado, recuperación de información en tiempo $\Theta(\log n)$, por lo que la estructura adecuada debe ser un árbol binario balanceado de búsqueda en alguna de sus implementaciones.

Algoritmo:

Entrada de datos: Polígono contenedor, lista de polígonos convexos que constituyen los huecos

Salida: Conjunto de segmentos Cs que particionan al contenedor.

P1 Realizar un barrido del plano donde los eventos son los puntos de todos los polígonos involucrados, estos ocurren ordenados por su X. Llamemos PE al polígono externo o contenedor y PH al conjunto de huecos, además llamemos A al árbol binario de búsqueda utilizado para almacenar los segmentos involucrados en cada paso del barrido. Para cada evento p hacer

P1.1 Insertar él o los segmentos cuyo punto de menor X es p en A

P1.2 If(p es el primer punto a insertar de algún polígono interno)

P1.2.1 Cs = Cs $\bigcup$ { segmento (p, evento anterior) }

P1.3 Eliminar de A el o los segmentos cuyo punto de mayor X es p

// De esta forma quedan enlazados todos los primero puntos de todos los huecos.

P2 Realizar el mismo proceso desde P1, pero esta vez teniendo en cuenta que se hace el barrido de mayor a menor por las X.

// De esta forma quedan enlazados todos los últimos puntos de todos los huecos. Por lo que todos los huecos quedan conectados de alguna forma con algún otro polígono.

Observemos que si todos los huecos quedan conectados por su menor y mayor coordenada X a otros polígonos, que puede ser este el exterior o alguno interior, que a su vez estará conectado directa o indirectamente con el polígono exterior, entonces tenemos que no habrá ningún polígono que quede como un hueco aislado, por lo que ha quedado dividido el polígono exterior en polígonos no convexos y convexos, entre estos últimos se encuentran los mismos

huecos.

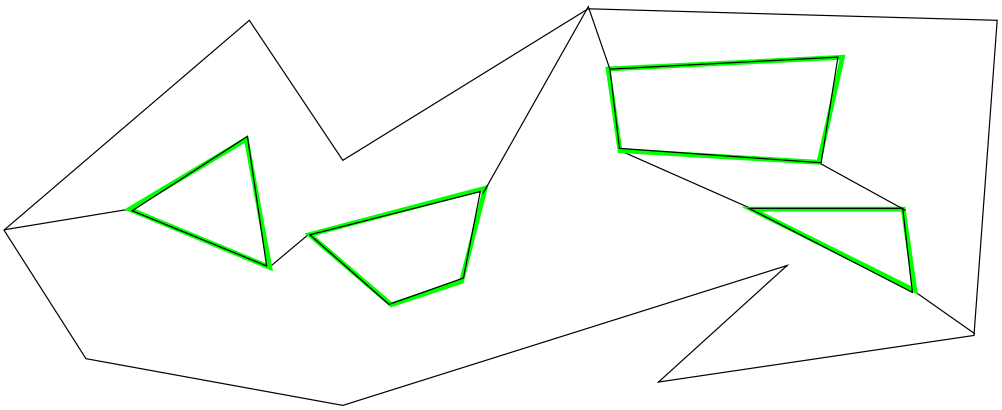

Figura 3.5: Muestra de como se enlazan los polígonos al eliminar los huecos.

-Búsqueda de soluciones enteras para particionar un polígono convexo que contiene dos polígonos convexos en su interior(huecos), con vértices enteros, en dos polígonos convexos tal que cada uno de ellos contenga un hueco convexo: Este algoritmo consiste en separar dos polígonos convexos contenidos dentro de un tercer convexo, para lo cual se busca un segmento, cuyos vértices tengan coordenadas enteras y estén sobre la frontera del polígono externo, tal que contenga a cada lado un polígono de los internos.

Algoritmo:

Entrada de datos: Polígono exterior PE, polígonos interiores p_0, p_1.

Salida: Si existe o No y el segmento que divide a PE en dos partes en caso de existir.

P1 Buscar un par de rectas r_0, r_1 tal que r_0, r_1 son las rectas que pasan por un vértice de p_0 y uno de p_1 tal que si cada uno de los vértice de p_0 quedan sobre o a un lado de la recta, entonces los de p_1 quedan sobre o a al otro lado de la misma. Lo antes explicado se muestra en la fig.3.6. Ver el resultado que muestra la existencia de r_0 y r_1, pág.38.

P2 if($r_0 == r_1$)

P2.1 Hallar intersección de r_1 con la frontera de PE, obteniendo pt_1, pt_2.

P2.2 if(pt_1 y pt_2 tienen coordenadas enteras)

P2.2.1 retornar el segmento pt_1, pt_2

else

P2.2.2 retornar No.

else

// Queda dividido PE en 4 cuadrantes, nombrémoslos C_0, C_1, C_2, C_3, donde quedarían p_0 en C_0 y p_1 en C_2, de forma alterna.

P2.3 Obtener los conjuntos F_0, F_1 de puntos frontera de coordenadas enteras en C_1 y C_3, donde F_0 es de menor cardinalidad.

P2.4 Para cada punto de F_0, f_{0_i}hacer

P2.4.1 Realizar una búsqueda binaria sobre los puntos de F_1, de un punto f_1 tal que el segmento (f_{0_i}, f_1), donde la relación de ordenación sobre los puntos de F1 establece: $f_{1_j} < f_{1_k}$si el segmento (f_{0_i}, f_{1_j}) corta a p_0 y el segmento (f_{0_i}, f_{1_k}) no, o si el segmento (f_{0_i}, f_{1_j}) no corta a p_1 y el segmento (f_{0_i}, f_{1_k}) sí. Terminar cuando se haya encontrado el punto f_1, o para el menor punto f_{1_j} y el mayor[2] f_{1_k}que no sean comparables mediante $<$.

P2.4.2 if(Se encontró f_1)

P2.4.2.1 retornar el segmento (f_{0_i}, f_1)

P2.5 Retornar No

Resultado: Siempre que existan dos polígonos convexos p_0, p_1 que no se intersecten, es posible encontrar r_0 y r_1 tales que r_0, r_1 son las rectas que pasan por un vértice de p_0 y uno de p_1 tal que si cada uno de los vértice de p_0 quedan sobre o a un lado de la recta, entonces los de p_1 quedan sobre o al otro lado de la misma.

Demo: Esto se puede ver si tomamos una recta cualquiera y al establecer un sentido de dirección sobre la misma tenemos p_0 a un lado y p_1 al otro, lo cual siempre es posible por no haber intersección entre p_0 y p_1, o lo que es lo mismo, son separables uno de otro. Escojamos un punto cualquiera pr_0 sobre la recta y rotémosla sobre este punto hasta que toque en algún punto a p_0 o a p_1, digamos que toca primero a p_0, sin perder generalidad. Ahora tomemos pr_1 como el punto de contacto entre la recta y p_0, el más alejado a pr_0 en caso que existan varios y rotemos la recta sobre el punto pr_1 hasta tocar a p_1 en un punto digamos pt_1, entonces tenemos una recta que la llamaremos r_0 que es la recta que pasa por

[2]En este caso el menor y el mayor hacen referencia al menor y mayor elemento que se manipulan al hacer una búsqueda binaria.

pr_1 y pt_1. Para encontrar la otra recta sólo tendríamos que rotar al inicio la recta escogida en el otro sentido sobre pr_0 y obtendríamos de forma similar la recta r_1, sólo que en caso en que los polígonos se toquen en algún punto, lo cual aceptamos como posible, quedaría $r_0 = r_1 =$ recta seleccionada en un principio.

Lqqd.

Este algoritmo pudiera tener un costo temporal elevado debido a que su solución depende de la cantidad de puntos enteros que aparezcan en los segmentos que conforman al polígono externo, aun así, se puede entrar a considerar que mientras mayor sea esta cantidad, mayor será la probabilidad de encontrar un segmento que satisfaga la solución de este problema. Además, no depende su tiempo de la cantidad de objetos que aparezcan en nuestro espacio. De las implementaciones obtenidas del mismo podemos decir que ha funcionado en un tiempo satisfactorio. Con respecto a su costo espacial podemos ver que si almacenamos los puntos en una forma conjuntual implícita, la memoria necesaria sería de un orden constante.

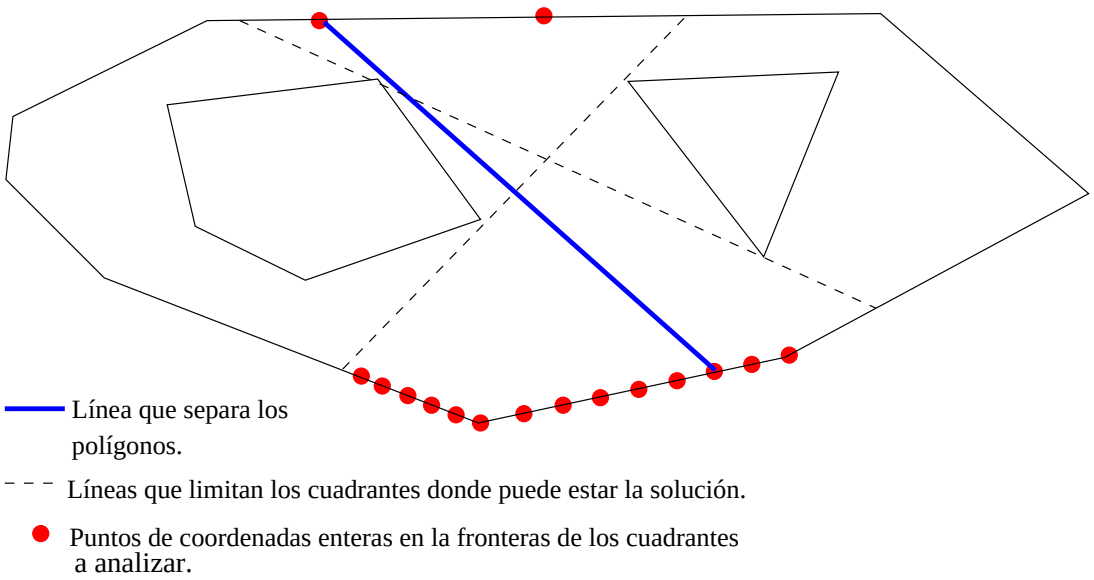

Figura 3.6: Separación de dos huecos en el interior de un convexo, usando un segmento con extremos enteros.

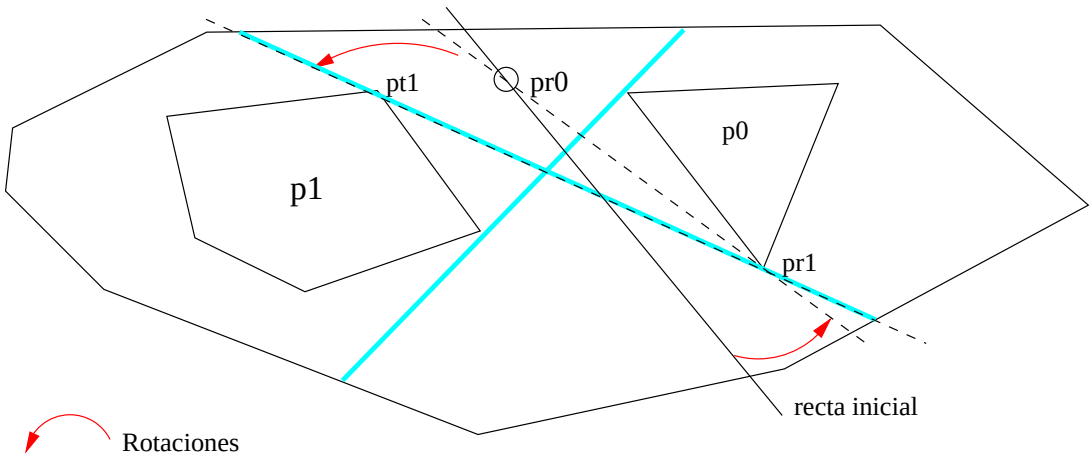

Figura 3.7: Prueba de separación entre convexos.

Capítulo 4

Aplicaciones de la partición convexa.

4.1 Consultas en el plano.

Este tipo de consultas son las que se realizan desde una vista superior del mapa de una ciudad, de la misma forma en que se fueron adicionando los objetos contenidos en la misma. Aquí podemos destacar la identificación de objetos en un punto determinado o en una zona que quede contenida en un polígono.

La identificación de objetos en un punto determinado no es más que la localización de dicho punto en la convex DCEL, cuyo algoritmo fue visto en el ep.3.3, de forma tal que obtengamos la región a la que pertenece y posteriormente chequear si el punto se encuentra en el interior del objeto existente en esa región, si la región no es vacía. De no encontrarse en el interior del objeto o estar vacía la región, no se identifica objeto alguno. Mediante esta consulta podemos preguntarnos por algun edificio en específico y ver sus características.

La identificación de objetos en una determinada zona consiste en realizar el cubrimiento del polígono contorno de la zona en la estructura (Ver algoritmo de cubrimiento en una convex DCEL en el ep.3.3) y posteriormente chequear la intersección de la zona con los objetos contenidos en el interior de las regiones del cubrimiento. Todos los objetos intersectados constituyen los identificados dentro de esa zona. Esta consulta es útil cuando queremos saber, por ejemplo, las manzanas afectadas por una inundación en una zona determinada.

4.2 Trabajo en tercera dimesión.

4.2.1 Algoritmos 2-d usados para el trabajo en tercera dimensión.

-Ordenación "front-end" desde un punto determinado de las regiones de la convex DCEL: Este algoritmo consiste en establecer cierto orden entre las regiones que se encuentren dentro de un ángulo determinado teniendo como vértice al punto p, de forma tal que si una región r_1 se encuentra detrás de otra r_0 con respecto a un punto p, r_1 se analiza después de r_0, dada cualquier dirección y el punto p, se puede establecer un orden topológico entre todas las regiones que quedan involucradas en esa dirección. Esto se basa en un resultado demostrado en el ep.2.2, lo cual significa que desde el punto p se puede determinar si una región está detrás de cualquier otra, o no tienen relación alguna. El costo temporal de este algoritmo es lineal, $\Theta(n)$ donde n es la cantidad de objetos involucrados. Esto se debe a que la cantidad de regiones es también Θ(n) como habíamos demostrado anteriormente, por lo tanto al visitar cada región un número acotado de veces se puede decir que el orden es lineal. Para este algoritmo utilizaremos una cola Q.

Algoritmo:

Entrada de datos: La convex DCEL, el punto p, el vector de dirección v y un ángulo a tal que v biseca el ángulo a.

Salida: Lista L de regiones donde se cumple que si $r_i \preceq r_j$ entonces r_i precede a r_j.

P1 *reg* = Localizar p en la estructura //*reg* será la región donde se encuentre el mismo.

P2 Insertar en Q las regiones vecinas a *reg* que se encuentren dentro del ángulo a

P3 while(Q no vacía) hacer

P3.1 Insertar el elemento cabeza de Q (r_i) en L, y extraerlo de Q

P3.2 $\forall$región r_j adyacente a r_i tal que $r_i \preceq r_j$ hacer

P3.2.1 if(r_j queda dentro del ángulo a && $\forall r_k$ tal que $r_k \preceq r_j$ se cumple que r_k está en L)

P3.2.1.1 insertar r_j en Q

Este algoritmo obviamente terminará porque la cantidad de regiones es finita, además al encontrarse con los límites del espacio no se puede continuar en esa dirección.

-Intersección entre poligonales monótonas: Consiste en intersectar dos poligonales planas y monótonas con respecto al eje X, p_0, p_1, de forma tal que se obtenga una poligonal que este descrita por la siguiente ecuación: $\forall x$, $f(x)=\min(p_0(x), p_1(x))$. El costo temporal es $\Theta(n_0+n_1)$ donde n_0 y n_1 son las cantidades de segmentos que componen a p_0 y p_1 respectivamente. Aunque su costo temporal es elevado, para poligonales con poca cantidad de segmentos como en nuestro caso, tiene un comportamiento aceptable. Podemos asumir que p_0 acota superior e inferiormente a p_1 por las X, o sea, la menor X de p_0 es menor que la menor X de p_1 y lo correspondiente ocurre con la mayor.

Algoritmo:

Entrada de Datos: Dos poligonales monótonas p_0 y p_1.

Salida: Poligonal p que constituye la intersección de las dos de la entrada.

P1 Hacer búsqueda binaria de la X del primer punto de p_1 para ver en que segmento s de p_0 corresponde

P2 Buscar cual poligonal está por encima(menor Y) en ese valor de X.

P3 Hacer un barrido del plano sobre p_0, a partir del segmento s, y p_1, donde los eventos son los vértices de ambas poligonales. Llamemos s_0 al segmento que comienza con el último vértice analizado de p_0 y Llamemos s_1 al segmento que comienza con el último vértice analizado de p_1. Para cada evento hacer:

P3.1 Insertar en p el punto al cual se avanzó en el último evento si este pertenece a la poligonal que está por encima

P3.2 if(Se intersectan s_0 y s_1 en el punto pt)

P3.2.1 Insertar el vértice pt en p

P3.2.2 Cambiar la poligonal que está por encima

4.2.2 Visualización tridimensional.

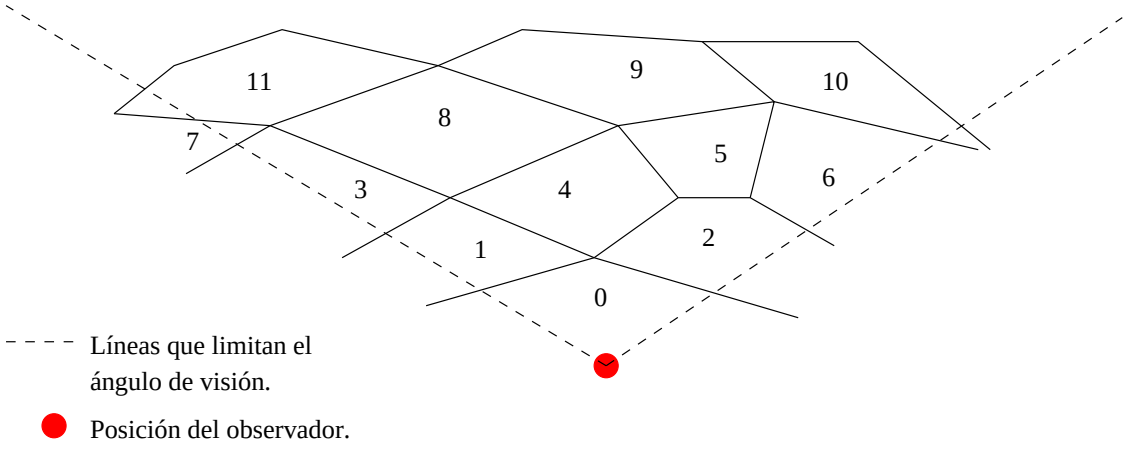

Figura 4.1: Ordenación de las regiones del plano.

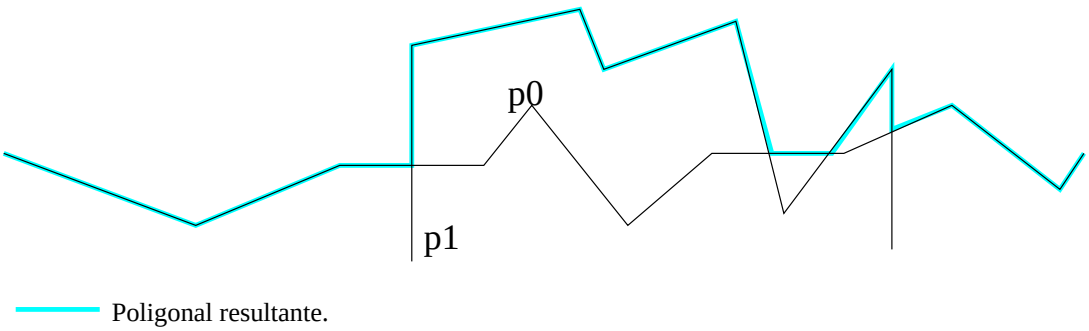

Figura 4.2: Intersección de poligonales.

-Visualización en 3 dimensiones de un conjunto de prismas en una partición convexa: Consiste en mostrar los objetos de la ciudad en forma de prismas rectos y crear una lista de poligonales donde cada poligonal representa el área de pantalla que se ha utilizado para representar los objetos, hasta el momento en que fue insertada en la lista. Este algoritmo tiene un costo temporal de orden $\Theta(n)$ donde n es la cantidad de objetos involucrados en la estructura. Para este algoritmo utilizamos la ordenación de las regiones del plano ya que una vez que tenemos las regiones ordenadas, al ser convexas las mismas y además por ser rectos los prismas a representar, podemos decir que se cumple la misma ordenación para las regiones que para todas las caras de diferentes objetos a representar en 3-d . Esto proporciona una gran ventaja ya que estas ordenaciones normalmente de un orden $\Theta(n \log n)$.

Algoritmo:

Entrada de datos: La convex DCEL, el punto p, el vector de dirección v y un ángulo a tal que v biseca el ángulo a.

Salida: La lista LP de poligonales, la visualización de los objetos.

Utilizar una pila P para pintar los prismas de atrás hacia adelante.

P1 Insertar la poligonal que equivale al borde inferior de pantalla en LP

P2 Obtener L = ordenación de las regiones de la estructura dado el punto p, el vector v y el ángulo a de visualización.

P3 $\forall$región r que está en L hacer: //Se va tomando r de acuerdo al orden en L

P3.1 if(r no es vacía)

P3.1.1 Obtener la perspectiva del objeto o contenido en r

P3.1.2 Obtener el prisma que se visualiza de o e insertarlo en la pila P

P3.1.3 Obtener la poligonal pp que constituye la parte superior del prisma

P3.1.4 Insertar en PL la intersección de la última poligonal insertada en PL y pp

P4 while(P no este vacía) hacer:

P4.1 Pintar en pantalla el prisma que representa el objeto en el tope de la pila y extraer el mismo

El algoritmo antes expuesto pudiera aprovechar la estructura de poligonales para realizar

un pintado en pantalla más eficiente desde el punto de vista de que las zonas solapadas en pantalla se pintan varias veces según este algoritmo. Pudiéramos introducirnos en un algoritmo más complejo y ver que resultados trae.

4.2.3 Consultas y modificaciones del mapa desde una vista tridimensional.

-Localización de puntos sobre la visualización tridimensional: Consiste en localizar un punto determinado de la pantalla, mostrando una visualización tridimensional, o sea, ver a que objeto pertenece un punto determinado. Este algoritmo tiene un costo temporal $\Theta(\log^2 n)$ donde n, en este caso, es la cantidad de objetos que se han mostrado en pantalla. Es fácil llegar a esta conclusión al ver que todo lo que se realiza es una búsqueda binaria dentro de otra. Este algoritmo nos muestra una gran rapidez al efectuar consultas sobre la vista tridimensional de la ciudad y es uno de los mayores logros que se han obtenido con este proyecto. Su orden de costo temporal puede ser reducido aun más hasta $\Theta(\log n)$, almacenando un poco más de información.

Algoritmo:

Entrada de datos: Lista de poligonales, lista de prismas en la visualización, punto p a localizar.

Salida: Objeto seleccionado.

P1 Hacer una búsqueda binaria sobre la lista ordenada de poligonales de forma tal que se obtenga exactamente debajo de qué poligonal Parr se encuentra. Para buscar si el punto se encuentra encima o debajo de una determinada poligonal P_i, hacer:

P1.1 Hacer una búsqueda binaria en P_i según la coordenada X de p y comparar con el segmento de la poligonal que comprende esta coordenada X.

P2 if(p pertenece al polígono frontera del prisma que generó la poligonal Parr)

P2.1 retornar objeto cuya perspectiva es el prisma analizado

else

P2.2 retornar NULL

Esta localización nos permite identificar un edificio determinado dentro de una visualización tridimensional, de esta forma podemos ver sus características, atributos y propiedades, y además, modificar las mismas, de forma tal que se permita una interacción mucho mayor con la ciudad. Por ejemplo, podemos seleccionar un edificio determinado y cambiar su altura, lo cual permite simular cambios en una ciudad desde una representacion más realista.

4.3 Robusticidad.

De estos algoritmos, como de ningún otro algoritmo geométrico, se puede asegurar que sean completamente robustos, por lo que se han implementado teniendo en cuenta una inmensa casuística, refiriéndome al gran número de casos particulares que pueden surgir de los datos del problema a resolver y su interacción con las estructuras de datos almacenadas. Una vez terminados, estos algoritmos fueron probados con un gran número de casos por diferentes personas pertenecientes al grupo y en algunos casos usando ejemplos creados aleatoriamente. Además, un gran número de estos algoritmos es empleado en la implementación de otros algoritmos, por lo que de esta manera son sometidos a prueba un número de veces del orden de los miles.

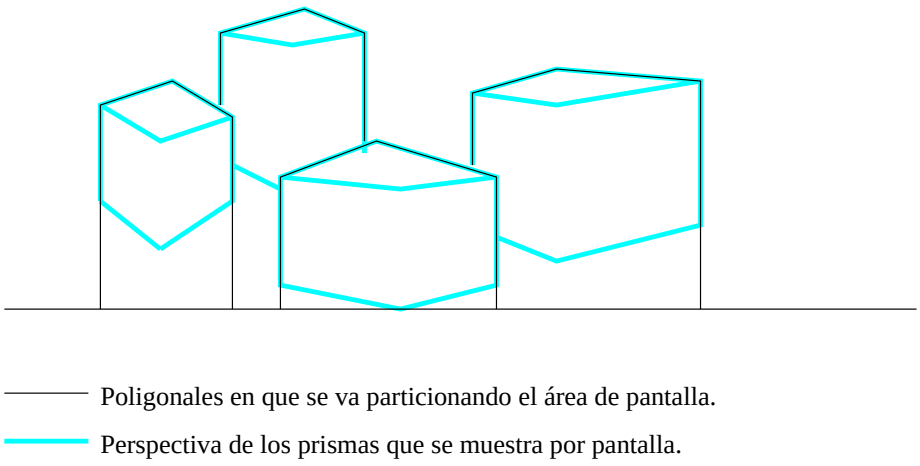

Figura 4.3: Regiones y prismas en pantalla dado una visualización tridimensional.

Capítulo 5

Diseño del sistema.

5.1 Qué hace el sistema?

Hasta el momento el sistema cuenta con una serie de algoritmos implementados en dos programas diferentes. Estos algoritmos incluyen los descritos anteriormente y otros más, que pueden no haber sido descritos por ser de una menor complejidad, las variaciones que puedan haber entre las implementaciones y los algoritmos, se deben precisamente a detalles de implementación, los cuales no entraré a explicar pues no creo que sean de gran interés.

Uno de estos programas trata con los objetos del mapa en el plano. Para esto se utiliza una clase que implementa una estructura convex DCEL. Hasta el momento se asume que los objetos son edificios dentro de la ciudad, esto lo hacemos para mayor simplicidad a la hora de probar otros algoritmos como el de visualización en 3-d, localización de puntos, etc. En realidad lo que se pretende representar, en versiones futuras y más cercanas a un SIG real, son las manzanas de la ciudad, las cuales en su interior tendrían a su vez una estructura convex DCEL en la que quedarían contenidos los edificios. La interfaz que proporciona este programa permite insertar edificios en la estructura, eliminarlos y cambiar sus características. Se muestran los edificios como polígonos que describen la forma de su base y se muestra además la estructura convex DCEL, lo que permite ver de forma incremental como se va creando dicha estructura. Evidentemente este programa es tan sólo utilizado para probar los algoritmos, pues la información de la convex DCEL es completamente transparente para

un usuario. Por último permite realizar optimizaciones en la estructura, salvarla en disco y recuperar o cargar alguna estructura previamente salvada.

El otro programa es el encargado de trabajar con la representacion en 3-d de la ciudad, específicamente el conjunto de edificios almacenados en el mapa, el cual debe haber sido construido y salvado con anterioridad utilizando el programa para trabajar con el mapa de forma plana, que es el descrito anteriormente. Este programa permite una visualización tridimensional de la ciudad desde un punto de observación y además cierto movimiento dentro de la estructura. Aunque realmente hay mucho que trabajar con respecto a la perspectiva utilizada, precisión y a los movimientos dentro del espacio, muestra lo suficiente para verificar la eficiencia y robusticidad con que funcionan los algoritmos antes explicados. También permite localizar o identificar puntos dentro del espacio tridimensional, y seleccionar edificios dentro del mismo, a los cuales se les puede realizar cambios en sus propiedades. Esta última funcionalidad permite una gran interactividad con la estructura, ya que se pueden modificar los objetos tal y como son vistos en la vida real, por ejemplo modificar la altura o el color de un edificio determinado. Como es lógico, al permitir modificaciones en la estructura, se da la posibilidad de salvar y recuperar la estructura.

Valdría la pena destacar que en los métodos utilizados para calcular la perspectiva de los objetos, se hacen algunos cálculos con operadores de punto flotante, pero la menor cantidad de veces posible, para aprovechar al máximo las propiedades de la estructura al haberse trabajado con una aritmética de enteros.

5.2 Lenguaje de programación en el que fue realizado el proyecto y bibliotecas empleadas.

El proyecto, incluyendo los programas antes explicados y en general la implementación de los algoritmos se realizo en C++ utilizando el compilador Visual C++ 5.0. Para las estructuras de datos sencillas se utilizaron los arreglos dinámicos y las listas doblemente enlazadas CArray y Clist respectivamente de la biblioteca de clases MFC(Microsoft Foundation Clas-

ses). Ambos programas constituyen proyectos basados en MFC por lo que se utilizan de esta biblioteca las clases de ventanas que constituyen la interfaz del proyecto, ya sean ventanas principales, cajas de diálogo comunes, etc. Además se utilizan algunos objetos geométricos sencillos que brinda esta biblioteca como es la clase CPoint, con la cual se tratan los puntos en las funciones geométricas.

5.3 Descripción de las clases principales del sistema.

Clases empleadas, propiedades y atributos, explicación de su utilización, operaciones sobre las mismas y relación entre ellas:

CPolygon: Clase que describe un polígono simple.

Atributos:

Points Lista de vértices del polígono, es un arreglo de CPoint.

Count Cantidad de vértices que forman el polígono.

Constructores: Tiene un constructor que crea el polígono a partir de una lista de puntos, y además tiene como parámetros dos indicadores que establecen si se quiere hacer el chequeo de simplicidad del polígono y que sus puntos estén ordenados en sentido contrario a las manecillas del reloj.

Funciones:

Convy Retorna una lista de polígonos convexos que constituyen la partición del polígono en cuestión en convexos. Además retorna, por si se necesita, los segmentos que particionan el polígono.

GetConvex Implementa el algoritmo de envoltura convexa visto en el ep.3.4.

Intersect Implementa el algoritmo de Intersección de polígonos visto en el ep.3.4.

Inside Verifica si un punto determinado se encuentra en el interior del polígono o no.

Write Es la encargada de salvar la región para disco, el polígono se salva a sí mismo, se le pasa una como parámetro referencia al fichero donde debe salvarse.

Read Es la encargada de recuperar la región desde disco, el polígono se recupera a sí mismo, se le pasa como parámetro una referencia al fichero donde debe salvarse.

CConvex: Clase que describe un polígono convexo. Hereda de la clase CPolygon dado que es un caso particular de la misma.

Atributos:

Tiene los mismos atributos heredados.

Constructores: Tiene un solo constructor que funciona de igual manera que el constructor de la clase CPolygon, sólo que adiciona la posibilidad de chequear si el polígono es convexo.

Funciones:

Inside Realiza la misma función que en la clase padre, pero la sobrescribe para computarla apropiadamente.

Intersect Realiza la misma función que en la clase padre, pero la sobrescribe implementando así el algoritmo de intersección de polígonos convexos visto en el ep.3.4.

CBlock: Es la clase que describe una manzana de la ciudad. No entraremos en detalles porque momentáneamente es tratada como un solo edificio, en vez de ser una convex DCEL que describa su interior como varios edificios. Un edificio contiene un polígono que describe su base, una altura, un color, entre otras características.

CVertex: Es la clase que describe los vértices de la DCEL.

Atributos:

X Coordenada X de tipo entera.

Y Coordenada Y de tipo entera.

Edge Enlace a una arista que contenga el vértice Edge como origen, de tipo Posición[1] .

Constructores: Consta de dos constructores, uno que crea el vértice a partir de sus coordenadas X y Y, y otro que lo crea a partir de un buffer que contiene la información de sus atributos. Este último usado para crearlo a partir de la información almacenada en un fichero.

Funciones:

Size Es una función de tipo static que retorna el tamaño del buffer de datos necesario para salvar y recuperar sus datos.

GetData Se le pasa un puntero a un buffer donde guarda sus datos.

[1]Es una referencia a una medio-arista de la lista.

CHalfEdge: Es la clase que describe las medio-aristas de la DCEL.

Atributos:

V vértice origen de la medio-arista, de tipo CVertex*.

Cw es la referencia a la medio-arista que se encuentra próxima en sentido de las manecillas del reloj y que tiene como origen también al mismo vértice V, de tipo Posición.

Ccw es la referencia a la medio-arista que se encuentra próxima en sentido contrario de las manecillas del reloj y que tiene como vértice destino el mismo vértice destino que la medio-arista en cuestión, de tipo Posición.

Reg referencia a la región que está a su derecha partiendo del vértice V, de tipo CRegion*.

La referencia a su medio-arista asociada no se incluye en los atributos pues esta referencia queda implícita en la posición en que se guardan las medio-aristas en la lista que se encuentra en la DCEL. Ver la lista de aristas en la clase DCEL.

Constructores: Tiene dos constructores, uno que crea la medio-arista a partir de un vértice y otro que la crea a partir de un buffer con los datos almacenados, para poderla crear a partir de la información almacenada en un fichero.

Funciones:

Size Idéntica a la de la clase CVertex.

GetData Idéntica a la de la clase CVertex.

CRegion: Es la clase que describe las regiones de la DCEL.

Atributos:

Edge Enlace a una arista que sea límite de la región en cuestión, de tipo Posición.

Block Enlace al objeto contenido dentro de la región, de tipo CBlock.

Constructores: Tiene dos constructores, uno que crea la región teniendo en cuenta la medio-arista que tiene asociada y el otro que la crea a partir de una referencia a un fichero donde se encuentra la información.

Funciones:

WriteTo Es la encargada de salvar la región para disco. Al igual que se recupera a sí misma, se salva a sí misma. Se le pasa como parámetro una referencia al fichero donde debe salvarse.

CDCEL: Es la implementación de la estructura DCEL.

Atributos:

List Es la lista que contiene las medio-aristas, las medio-aristas asociadas guardan la siguiente relación: Se encuentran en posiciones consecutivas, al ser guardadas por pares, la primera que se inserta en la lista tiene una posición par y la segunda la posición impar siguiente. Al tratar la lista como un arreglo como lo hemos hecho hasta ahora se puede decir que a partir de la posición pos de una arista su asociada se encuentra en pos xor (pos and 1), lo cual constituye una forma muy rápida de tener acceso a la asociada.

Constructores: Tiene uno que crea la estructura vacía.

Funciones:

Insert Implementa el algoritmo de inserción de aristas visto en el ep.3.1.5.

Remove Implementa el algoritmo de borrado de aristas visto en el ep.3.1.5.

Write Implementa el algoritmo de salvar visto en el ep.3.1.5.

Read Implementa el algoritmo de recuperar visto en el ep.3.1.5.

CCDCEL: Es la clase que implementa una convex DCEL. Hereda de la clase CDCEL.

Atributos:

Tiene los mismos atributos heredados.

Constructores: Tiene uno que crea la estructura vacía.

Funciones:

Locate Implementa el algoritmo de localización de un punto determinado en una convex DCEL, visto en el ep.3.3.

Cover Implementa el algoritmo de cubrimiento de un polígono simple por regiones de la convex DCEL, visto en el ep.3.3.

Insert Implementa el algoritmo de inserción de polígonos convexos en una convex DCEL, visto en el ep.3.3.

CBlockSpace: Es la clase que describe el mapa de manzanas, no es más que una convex DCEL donde las regiones son manzanas, de tipo CBlock.

CVector: Es la clase que define un vector en el plano.

Atributos:

P1 Vértice origen del vector, de tipo CPoint.

P2 Vértice destino del vector, de tipo CPoint.

Constructores: Tiene uno sólo que crea el vector a partir de dos puntos.

Funciones:

Ño tiene funciones por el momento.

CRay: Es la clase que describe un rayo en el plano.

Atributos:

P1 Vértice origen del rayo, de tipo CPoint.

A, B, C Son los valores que corresponden a la ecuación canónica de la recta[2] que contiene al rayo en cuestión, son de tipo entero.

AB_hyp Es una aproximación a la longitud de la hipotenusa de un triángulo rectángulo, donde A y B son las longitudes de los catetos. Es usada un gran número de veces, como es el caso en que se quiere saber la distancia de un punto al rayo en cuestión, por esto es que se calcula y se conserva su valor, su cálculo es costoso debido a que implica operaciones de punto flotante.

Constructores: Tiene un constructor, se crea a partir de un vector que constituye una porción de longitud finita del rayo en cuestión.

Funciones:

Ño tiene funciones por el momento.

CPolygonal: Es la clase que describe una poligonal monótona con respecto al eje de las X.

Atributos:

Points Lista de vértices de la poligonal, es un arreglo de CPoint.

Count Cantidad de vértices que forman la poligonal.

Constructores: Tiene un solo constructor que crea la poligonal a partir de una lista de puntos.

Funciones:

Find Localiza que segmento de la poligonal comprende la coordenada X pasada como parámetro.

[2]La ecuación canónica de la recta se escribe: $Ax + By + C = 0$.

Merge Implementa el algoritmo de intersección de poligonales visto en el ep.3.4.

CPrism: Es la clase que describe un prisma, básicamente se basa en un polígono que describe la frontera de su perspectiva y algunos otros detalles que se quieran mostrar del prisma que representa, tales como aristas que se ven, etc.

C3dBlockSpace: Es la clase que describe el mapa de la ciudad desde un punto de vista tridimensional. Hereda de CBlockSpace.

Atributos:

Viewer_pos Posición en que se encuentra el observador dentro de la ciudad, de tipo CPoint.

Viewer_heignt Altura con respecto al nivel del mar de dicho observador, de tipo entero.

Viewer_ang Indica cuál dentro de un rango de ángulos posibles es el dirección hacia donde mira dicho observador, de tipo entero.

Constructores: Tiene un constructor que crea la estructura vacía.

Funciones:

GetPerspective Método privado que calcula un polígono que constituye la representación en perspectiva del polígono pasado como argumento a la función, llevado a las coordenadas de pantalla.

GetPerspective Método privado que calcula un punto que constituye la posición en la pantalla que ocuparía un punto de la ciudad llevado a perspectiva.

Show Implementa el algoritmo de visualización en 3 dimensiones de la ciudad, visto en el ep.3.4.

Locate Implementa el algoritmo de localización de puntos sobre una visualización tridimencional, visto en el ep.3.4.

5.4 Mejoras que se pueden introducir en el diseño del sistema.

Debido a la forma en que se fue desarrollando el sistema, a la forma en que se implementaron los algoritmos y guiados por su facilidad para probarlos, el diseño de clases no es el más apropiado realmente, por lo que se podrían realizar algunos cambios como: la clase CPolygon pudiera heredar de la clase CPolygonal debido a que tienen una gran cantidad de características en común, además está claro que un polígono simple puede verse como un tipo particular de poligonal cerrada. Hacer varios arreglos para que realmente se pueda ajustar la convex DCEL al interior de las manzanas también, entre los que habría que hacer uso de la genericidad y el polimorfismo.

Capítulo 6

Conclusiones y propuestas.

Al analizar los resultados obtenidos después de la implementación de estos algoritmos, podemos estar satisfechos de chequear que su funcionamiento ha sido tan bueno como quisimos en un principio. A pesar de haber tropezado con un gran número de detalles que hacen un poco más complicada la implementación, que el algoritmo en sí, hemos logrado implementar los algoritmos tal y como fueron descritos, alcanzando muy buenos tiempos de ejecución de los mismos. El propio diseño de los algoritmos lo consideramos un logro fundamental, que en conjunto con la estructura de datos (objetivo principal de este proyecto), han dado un resultado realmente satisfactorio. Esto, complementado además por la aritmética empleada, la cual proporciona una gran rapidez en la ejecución de las funciones, y en cierta medida, un menor grado de problemas de precisión. Los resultados obtenidos son alentadores a la continuación del trabajo para llegar hasta el desarrollo total de esta y tantas otras partes de un SIG urbano que conciernen al desarrollo dentro de la Geometría Computacional y otros campos.

Como mejoras al trabajo que se ha realizado, se pueden efectuar, primeramente, algunos cambios del diseño del sistema como vimos en el capítulo correspondiente, y perfeccionar varios aspectos de implementación de los algoritmos tratados, en busca de que todos lleguen a alcanzar su cota mínima en cuanto a costo temporal y espacial. Además, lograr con estos obtener las soluciones optimas, como es el caso de obtener la menor cantidad de regiones vacías en la estructura, para lo cual habría que trabajar en la inserción de objetos

y optimización de la estructura.

Como trabajo futuro, podemos ampliar nuestros objetivos y continuar en varios aspectos como son el tratamiento de varias partes del SIG en forma de multicapas, para poder incorporar a nuestro sistema otras muchas funcionalidades tales como el relieve, las vías, redes técnicas, etc. Incluso pretendemos trabajar con otras estructuras que se pudieran incluir en la misma capa, como son los puentes, riveras de ríos y canales, túneles, etc.

Otros aspectos que opino que se deben desarrollar es la creación de rutinas encargadas de realizar el chequeo de integridad de las estructuras de datos; así como la creación de rutinas que chequean la robusticidad de los algoritmos implementados. Esto permite no sólo la detección de errores a la hora de ir implementando los diferentes algoritmos usados en el sistema, sino también caminar con pasos más seguros y firmes, cosa que no es del todo posible cuando tratamos con algoritmos que presentan una elevada complejidad.

Bibliografía

[AHU83] Alfred V. Aho, John E. Hopcroft, and Jefferey D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, 1983.

[Chr97] Nicholas Chrisman. *Exploring Geografic Information Systems.* John Wiley, 1997.

[CR93] David Comas and Ernest Ruiz. *Fondamentos de los Sistemas de Información Geográfica.* Ariel, 1993.

[dBvKOS97] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry Algorithms and Applications.* Springer-Verlag, 1997.

[dSdlF99] Grupo de SIG de la FAO. Sistemas de información geográfica en el desarrollo sostenible, Junio 1999.

[Her98] Gabor G. Herman. *Geometry of Digital Spaces.* Birkhauser, 1998.

[MN99] Kurt Mehlhorn and Stefan Näher. Leda a platform for combinatorial and geometric computing, Abril 1999.

[Vel91] José Veliz. Trabajo de diploma, 1991. Universidad de La Habana, Facultad de Matemática y Computación.

[vKNRW97] Marc van Kreveld, Jürg Nievergelt, Thomas Roos, and Peter Widmayer. *Algoritmic Foundation of Geografic Information Systems.* Springer-Verlag, 1997.