ANÁLISIS Y CODIFICACIÓN DE ALGORITMOS BÁSICOS PARA EL ESTUDIO DE LOS DIAGRAMAS DE VORONOI Y SUS APLICACIONES EN GEOMETRÍA COMPUTACIONAL

NILSON JAVIER ALEMÁN MOROCHO
WILSON ALBERTO ORDÓÑEZ BOLAÑOS
NELCY ROMERO ROJAS

UNIVERSIDAD DEL CAUCA
FACULTAD DE CIENCIAS NATURALES, EXACTAS Y DE LA EDUCACIÓN
DEPARTAMENTO DE MATEMÁTICAS
POPAYÁN
2004

ANÁLISIS Y CODIFICACIÓN DE ALGORITMOS BÁSICOS PARA EL ESTUDIO DE LOS DIAGRAMAS DE VORONOI Y SUS APLICACIONES EN GEOMETRÍA COMPUTACIONAL

Seminario de grado presentado como requisito

parcial para optar al título de:

Licenciados en Educación con Especialidad en Matemáticas a

NILSON JAVIER ALEMÁN MOROCHO y

NELCY ROMERO ROJAS

y Matemático a WILSON ALBERTO ORDÓÑEZ BOLAÑOS

Director

ORLANDO RODRÍGUEZ BUITRAGO

UNIVERSIDAD DEL CAUCA

FACULTAD DE CIENCIAS NATURALES, EXACTAS Y DE LA EDUCACIÓN

DEPARTAMENTO DE MATEMÁTICAS

POPAYÁN

2004

Nota de aceptación

_____

_____

_____

Director

_____
Especialista **Orlando Rodríguez B.**

Comité Evaluador

_____
Especialista **Martha Romero Rojas**

_____
Especialista **Gersaín Dagua**

_____
Ing. Civil. **Gilberto Fernández**

Fecha de sustentación: Popayán 20 de Agosto del 2004

A nuestros padres y hermanos
con todo nuestro amor.

AGRADECIMIENTOS

Los autores expresan sus agradecimientos a:

Orlando Rodríguez Buitrago, profesor del Departamento de Matemáticas de la Facultad de Ciencias Naturales, Exactas y de la Educación de la Universidad del Cauca y director del trabajo.

Martha Judith Romero Rojas, profesora del Departamento de Matemáticas de la Facultad de Ciencias Naturales, Exactas y de la Educación de la Universidad del Cauca.

Gersain Dagua, profesor del Departamento de Matemáticas de la Facultad de Ciencias Naturales, Exactas y de la Educación de la Universidad del Cauca.

Gilberto Fernández, profesor del Departamento de Matemáticas de la Facultad de Ciencias Naturales, Exactas y de la Educación de la Universidad del Cauca.

Diego Correa, profesor del Departamento de Matemáticas de la Facultad de Ciencias Naturales, Exactas y de la Educación de la Universidad del Cauca.

Y a todos los que de una u otra forma contribuyeron a la realización del presente trabajo.

CONTENIDO

LISTA DE ANEXOS

Pag.

INTRODUCCIÓN

La *Geometría Computacional* (bautizada por Shamos en 1975) es una disciplina relativamente joven de ahí su desconocimiento y escasa difusión, surge como respuesta a un gran número de aplicaciones y problemas geométricos que requieren algoritmos eficientes para su resolución, se podría decir que esta disciplina es el enlace de unión entre la geometría y el mundo de la computación. Hoy en día es utilizada como herramienta básica no solo en disciplinas inmediatamente relacionadas con la Geometría sino que es aplicada en diversas áreas como: Informática Gráfica, Robótica, Sistemas de Información Geográficos, entre otros.

El campo de la Geometría Computacional es extenso y variado, se pueden encontrar entre sus temas desde Triangulación hasta Proximidad pasando por Localización y otros muchos, y por supuesto por los Diagramas de Voronoi que constituyen el tema central del trabajo.

El contenido del presente informe incluye dos temas principales: La Triangulación y los Diagramas de Voronoi. La *Triangulación* de una nube de puntos o de un polígono es una partición del dominio que definen (el cierre convexo en el caso de nube de puntos o el propio polígono en el otro caso) en triángulos y *los Diagramas de Voronoi* que son una de las estructuras fundamentales, ya que de alguna forma ellos almacenan toda la información

referente a la proximidad entre puntos. De esta forma, se puede encontrar el punto más próximo a otro dado dentro de una nube de puntos.

Al profundizar en el estudio de los fundamentos teóricos de la geometría computacional aparece el término *algoritmo*, el cual es una secuencia de pasos que solucionan un problema determinado, por ejemplo la construcción euclidea se puede considerar como el primer algoritmo completo, pues cumple con los requerimientos: da una colección de instrucciones y reglas, es correcto, no ambiguo y tiene un final. Además se habla del término complejidad (no computacional), en el sentido de la dificultad para medir la complicación de la realización del algoritmo. Esto es una aproximación a lo que hoy se entiende por complejidad temporal. A medida que se profundizó en el estudio de la complejidad, se intentaba reducir el número de pasos para resolver un problema, deduciendo que todo problema posee un esfuerzo mínimo de realización, actualmente conocido como "*Complejidad del algoritmo*".

1. TRIANGULACIÓN DE POLÍGONOS

1.1 POLÍGONOS

La geometría computacional realiza cálculos sobre objetos conocidos como polígonos. Un polígono es una representación conveniente para muchos objetos del mundo real y se puede manipular fácilmente en el computador.

1.1.1 Definición de un polígono Un polígono es la región del plano limitada por una colección finita de segmentos que forman una curva cerrada simple.

Esta definición es difícil de manipular mediante el computador, por tal razón, se presenta la siguiente definición equivalente: sean $v_0, v_1, \ldots, v_{n-1}$ n puntos en el plano, y $e_0 = v_0v_1$, $e_1 = v_1v_2, \ldots,$ $e_{n-1} = v_{n-1}v_0$ n segmentos conectando los puntos. Se dice que estos segmentos limitan un polígono si y sólo si se satisfacen las siguientes condiciones.

1. Para todo $i = 0,1,\ldots,n-1$ se tiene que :

$$e_i \cap e_{i+1} = v_{i+1}$$

2. Para todo $j \neq i+1$ se tiene que:

$$e_i \cap e_j = \phi$$

1.1.2 Notas

a. Los puntos $v_0, v_1, \ldots, v_{n-1}$ que aparecen en la definición se llamarán *vértices* del polígono y los segmentos $e_0, e_1, \ldots, e_{n-1}$ se llamarán *lados* o *aristas* del polígono.

b. De ahora en adelante, debe tenerse en cuenta que todo índice aritmético será modulo n $(mod\ n)$, implicando así un ciclo ordenado de los vértices en el polígono. Esto resultará necesario para su respectiva implementación computacional.

c. En este documento los vértices se conectarán en sentido contrario a las manecillas del reloj (ó antihorario), mas adelante se observará la importancia de esta convención.

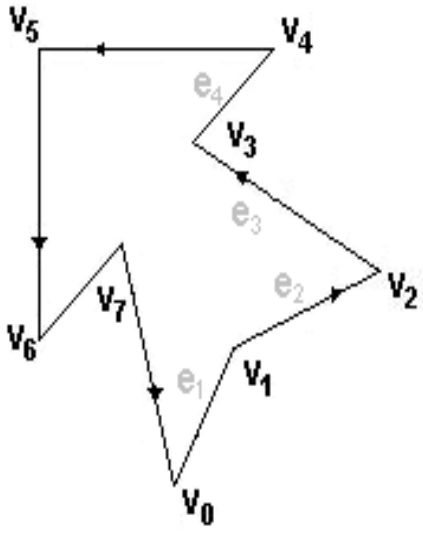


Figura 1.1 Polígono con vértices orientados en sentido antihorario.

1.1.3 Definición (Visibilidad) Sean x, y dos puntos arbitrarios de un polígono P. Se dice que x puede ver a y si y sólo si el segmento cerrado xy esta contenido en P, es decir $xy \subseteq P$.

Se dice que x puede ver claramente a y si y sólo si

$$xy \subseteq P \wedge xy \cap Fr(P) \subseteq \{x, y\};$$

donde $Fr(P)$ denota la frontera de P.

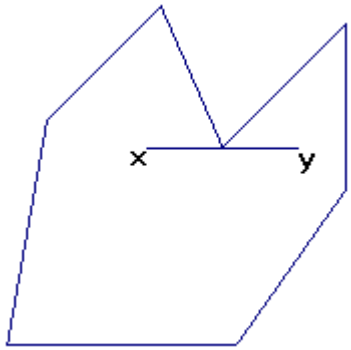


Figura 1.2 Bloqueo de una línea de visión

1.1.4 Definición (Diagonal) Sean a y b dos vértices de un Polígono P se dice que ab es una diagonal de P si y sólo si a y b son claramente visibles entre sí.

1.2 TEORÍA DE LA TRIANGULACIÓN

En la Teoría de la Triangulación se prueba que todo polígono admite una triangulación y se establecen algunas propiedades básicas de la misma.

1.2.1 Existencia de una Diagonal La clave para demostrar la existencia de una triangulación es probar la existencia de una diagonal. A continuación se precisan algunos conceptos necesarios para probar la existencia de una diagonal.

1.2.2 Definición de Vértice Convexo y Reflejo Un vértice es llamado *reflejo* (o cóncavo) si su ángulo interno es estrictamente mayor que π, si el vértice no es reflejo se dice que es *convexo*.

1.2.3 Nota Imaginemos que un caminante recorre los lados de un polígono en sentido antihorario. Si el caminante al llegar a un vértice realiza un giro a la izquierda, podemos decir que el vértice es estrictamente convexo; si el giro es a la derecha, el vértice es reflejo. Además, el interior del polígono esta siempre a la izquierda del caminante hipotético.

1.2.4 Lema Todo polígono tiene al menos un vértice estrictamente convexo.

Prueba: Sea L una línea horizontal que pasa por el vértice mas bajo de P. Si existen varios vértices más bajos, escoja v el vértice más a la derecha. Entonces el interior de P estará siempre por encima de L y además el lado que sigue a v estará también por encima de L.

Por el cumplimiento de las dos condiciones anteriores, tenemos que el caminante realiza un giro hacia la izquierda. Por lo tanto v es un vértice estrictamente convexo de P. Vea la *Figura 1.3.*

■

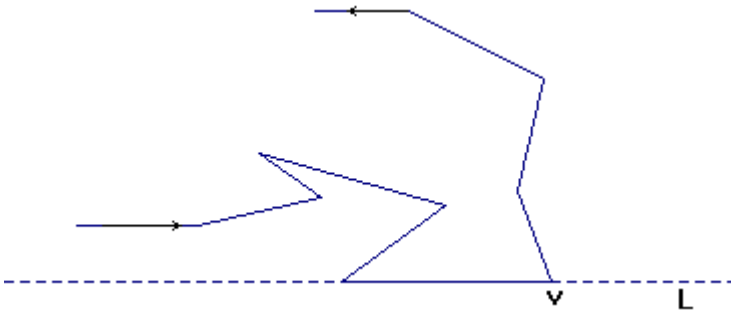


Figura 1.3 El vértice inferior más a la derecha es estrictamente convexo

1.2.5 Lema (De Meister) Todo polígono de $n \geq 4$ vértices tiene una diagonal.

Prueba: Sea v un vértice estrictamente convexo, cuya existencia es garantizada por el *lema 1.2.4.* Sean $a\ y\ b$ vértices adyacentes a v. Si ab es una diagonal entonces se finaliza la prueba.

Supongamos que ab no es una diagonal; entonces ab es exterior a P o intercepta $Fr(P)$. En ambos casos el triángulo Δavb contiene al menos un vértice de P diferente de $a, v\ y\ b$. Sea x el vértice de P en Δavb que es más aproximado a v. Sean l_1 la recta que pasa a través de los vértices $a\ y\ b$, y l_2 la recta paralela a l_1 movida de v hacia $a\ y\ b$, entonces x es el primer vértice de Δavb encontrado por la recta l_2. Véase la *figura 1.4.*

Por lo tanto se afirma que la intersección del semiplano limitado por l_2 que incluye a v con el interior del triángulo Δavb no contiene puntos de $Fr(P)$ excepto por $v\ y\ x$, luego vx es una diagonal.

■

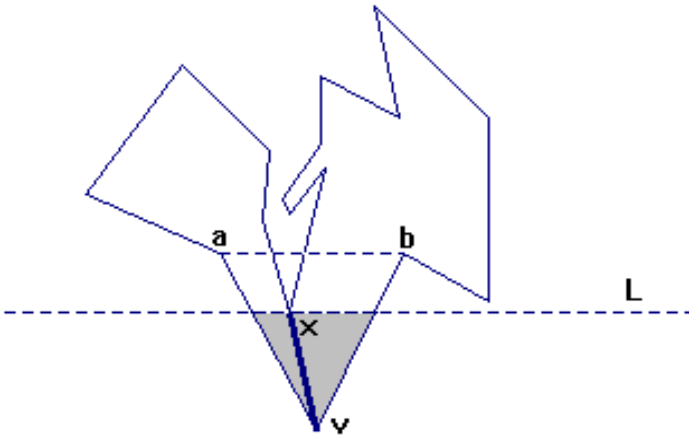


Figura1.4. *vx* puede ser una diagonal

1.2.6 Teorema (Triangulación) Todo polígono P de n vértices puede ser dividido en triángulos por adición de (cero o mas) diagonales.

Prueba: La prueba se hará por inducción. Si $n=3$, el polígono es un triángulo y el teorema se satisface.

Supongamos que el teorema se cumple para todo polígono de $3,4,5,\ldots,n-1$ vértices. Sea P un polígono de n vértices y $d=ab$ una diagonal de P; aplicando el *lema 1.2.5,* d divide a P en dos subpolígonos P_1 y P_2 donde d es un lado común a ambos; véase la *figura 1.5*.

Si n_1 y n_2 es el número de vértices de cada uno respectivamente, $n_1 < n$ y $n_2 < n$. Se tendría que la hipótesis de inducción se cumple sobre cada subpolígono, de esta manera queda demostrado el Teorema. ∎

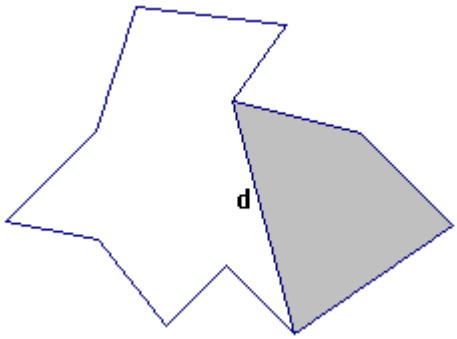


Figura 1.5. Una diagonal divide al polígono en dos subpolígonos.

Aunque en general existen varias formas de triangular un polígono dado, todas ellas tienen el mismo número de diagonales y triángulos, esto se formaliza en el siguiente teorema.

1.2.7 Teorema (número de diagonales) Toda triangulación de un polígono P de n vértices utiliza $n-3$ diagonales y consiste de $n-2$ triángulos.

Prueba: La prueba es por inducción. Si $n=3$, P es un triángulo y ambas afirmaciones se satisfacen.

Supongamos que $n \geq 4$ y que el teorema se satisface para todo polígono con un número de vértices inferior a n. Sea $d=ab$ una diagonal de P, entonces d divide a P en dos polígonos P_1 y P_2 donde d es un lado común a ambos. Sean n_1 y n_2 el número de vértices de P_1 y P_2 respectivamente. Entonces: $n_1+n_2=n+2$; porque a y b se cuentan como vértices en ambos subpolígonos. Aplicando la hipótesis de inducción a

P_1 y P_2 se tiene que P_1 utiliza n_1-3 diagonales y P_2 utiliza n_2-3 diagonales. Como d es una diagonal de P entonces P utiliza:

$(n_1-3)+(n_2-3)+1=n_1+n_2-5=n+2-5=n-3$ diagonales.
Luego P utiliza $n-3$ diagonales.

Ahora probemos que P consta de $n-2$ triángulos. Como n_1 y n_2 es el número de vértices de P_1 y P_2 respectivamente, el número de triángulos de P seria el número de triángulos de P_1 mas el número de triángulos de P_2. Es decir $(n_1-2)+(n_2-2)=n_1+n_2-4=n+2-4=n-2$ triángulos.

■

1.2.8 Corolario (Suma de ángulos) La suma de los ángulos internos de un polígono de n vértices es $(n-2)\,\pi$.

Prueba: Por el *Teorema 1.2.6*, el polígono se divide en $n-2$ triángulos. Cada uno contribuye π a los ángulos internos.

■

El dual de un grafo es un concepto muy importante en teoría de grafos, aunque no se utilizará en su total generalidad, se definirá a continuación por ser una estructura útil en la triangulación.

1.2.9 Definición El dual de una triangulación de un polígono es un grafo donde cada nodo se asocia a un triángulo y hay un arco entre dos nodos si y sólo si sus triángulos comparten una diagonal. Véase la *figura 1.6.*

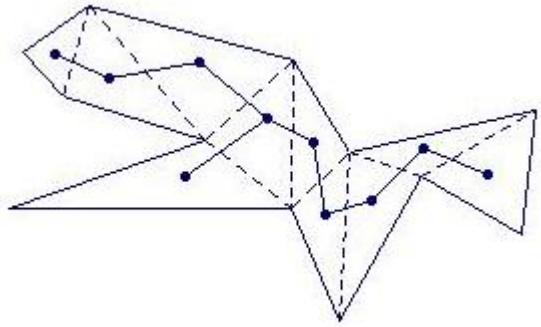


Figura1.6. Triangulación Dual

1.2.10 Nota

a) El dual *T* de una triangulación es un árbol cuyos nodos son de grado a lo más tres.

b) Un árbol es un grafo conectado sin ciclos.

c) Los nodos de grado uno son hojas de *T.*

d) Los nodos de grado dos están situados sobre la trayectoria del árbol.

e) Los nodos de grado tres son puntos de ramificación.

f) La correspondencia entre triangulaciones duales y árboles binarios se puede aprovechar con frecuencia.

1.2.11 Definición Sean $a,\ b,\ c$ tres vértices consecutivos en un polígono P. Se dice que $a,\ b,\ c$ forman *una oreja* si ac es una diagonal; b se llama *vértice oreja.* Dos orejas no están *superpuestas* si el interior de sus triángulos es disjunto.

1.2.12 Teorema (Las dos Orejas de Meister) Todo polígono de $n \geq 4$ vértices tienen al menos dos orejas no superpuestas.

Prueba: Todo nodo hoja en una triangulación dual corresponde a una oreja y un árbol de dos o más nodos tiene al menos dos hojas; por *teorema 1.2.7* (el árbol tiene $(n-2) \geq 2$ nodos).

1.3 IMPLEMENTACIÓN

1.3.1 Representación de un punto Todos los puntos serán representados por arreglos. Para las coordenadas se utilizarán enteros en lugar de reales, para evitar problemas de redondeo, facilitar la representación gráfica de polígonos y por su conveniencia al utilizarlos en algoritmos que comparan.

El siguiente código muestra una forma de definir un *tipo punto* en Delphi.

```
tipo
  TPunto = registro
    x,y: Entero;
  fin;
```

Código 1.1 Tipo Punto.

1.3.2 Representación de un Polígono Un polígono se puede representar por arreglos o por listas enlazadas. En nuestro caso se opto por arreglos dinámicos debido a la claridad en los códigos.

A continuación se muestra la manera de definir un *tipo polígono* en Delphi.

```
Tipo
Tlista_Vertices = arreglo de TPunto;
TPoligono = registro
   Poli: Tlista_vertices;
   tam: Entero;
 fin;
```

Código 1.2 Tipo Polígono.

1.4 ÁREA DE UN POLÍGONO

1.4.1 Área del Triángulo La forma usual para calcular el área de un triángulo: (base por altura)/2 no es muy conveniente en nuestro caso; esto debido a que la altura no se obtiene directamente de las coordenadas de los vértices. Por tal razón, se utilizará el producto cruz.

Del Álgebra Lineal se tiene que la magnitud del producto cruz de dos vectores es el área del paralelogramo determinado por ellos.

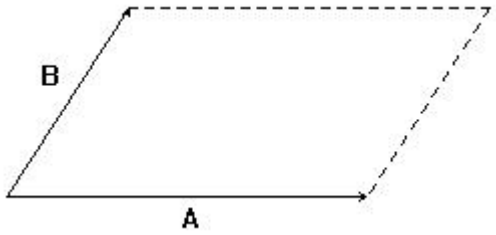


Figura 1.7 Paralelogramo del producto cruz

Tenga en cuenta que si A y B son dos vectores, el producto cruz $A \times B$ se puede calcular mediante el siguiente determinante:

$$\begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ A_0 & A_1 & A_2 \\ B_0 & B_1 & B_2 \end{vmatrix} = (A_1B_2 - A_2B_1)\hat{i} + (A_2B_0 - A_0B_2)\hat{j} + (A_0B_1 - A_1B_0)\hat{k} \quad (1.1)$$

1.4.1.1 Lema Si $T=(a,b,c)$ es un triángulo. Se denota el área de T como: $A\,(T)$ donde:

$$2A\,(T) = \begin{vmatrix} a_0 & a_1 & 1 \\ b_0 & b_1 & 1 \\ c_0 & c_1 & 1 \end{vmatrix} \quad (1.2)$$

Prueba: Sean $A=b-a$ y $B=c-a$. Como A y B son vectores del plano euclideo entonces de la ecuación 1.1 se tiene que $A_2 = B_2 = 0$, luego:

$$\begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ A_0 & A_1 & A_2 \\ B_0 & B_1 & B_2 \end{vmatrix} = (A_0B_1 - A_1B_0)\hat{k}$$

Por lo tanto la magnitud del determinante seria el área del paralelogramo de lados A y B, obteniéndose de esta manera que: $2A\,(T) = A_0\,B_1 - A_1B_0$.

Reemplazando los valores de A y B se tiene:

$$2A\,(T) = a_0b_1 - a_1b_0 + a_1c_0 - a_0c_1 + b_0c_1 - c_0b_1 = \begin{vmatrix} a_0 & a_1 & 1 \\ b_0 & b_1 & 1 \\ c_0 & c_1 & 1 \end{vmatrix}$$

■

1.4.2 Área de un Polígono desde un centro arbitrario A continuación se centrará el interés en determinar el área de un polígono cualquiera (convexo o no convexo) a partir de las coordenadas de sus vértices. Inicialmente se estudiarán los triángulos para luego generalizar a polígonos de n vértices.

Sea $T=(a,b,c)$ un triángulo cuyos vértices están orientados en sentido antihorario, y sea q un punto cualquiera en el plano (interno o externo al triángulo). Considérese la *figura 1.8*.

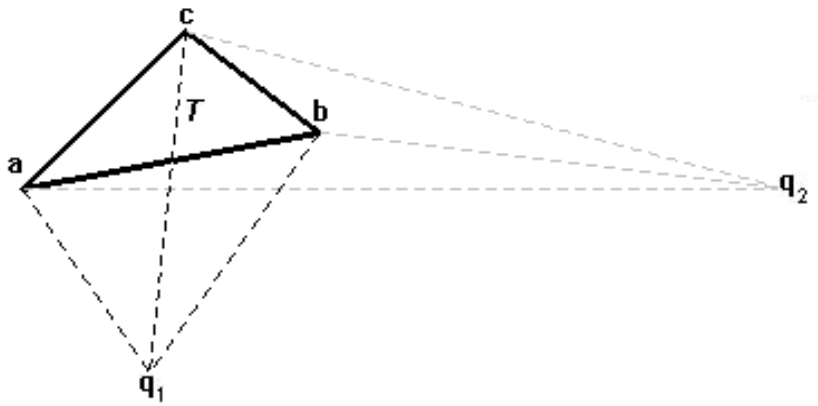


Figura 1.8 Área de T basada en dos puntos externos q_1 y q_2.

Se puede observar que: $A(T)=A(q,a,b)+A(q,b,c)+A(q,c,a)$ (1.3)

Caso 1. Si $q=q_1$, el primer término de la *ecuación 1.3* es negativo porque los vértices están en sentido horario y los dos términos restantes son positivos por encontrarse en sentido antihorario. Observe que $A(q_1,a,b)$ se resta porque es el área de la porción del cuadrilátero (q_1,b,c,a) que está por fuera del triángulo T, quedando así la suma total $A(T)$ como se afirmó anteriormente.

Caso 2. Si $q = q_2$, $A(q_2, a, b)$ y $A(q_2, b, c)$ son ambas negativas y son removidas de $A(q_2, c, a)$ la cual es positiva, quedando la suma total $A(T)$ como se afirmó.

Cualquier otra posición para q en el plano que no sea interna a T es equivalente a q_1 ó q_2 por simetría. Por supuesto la ecuación se mantiene cuando q es interno a T. Por lo tanto, se establece el siguiente lema:

1.4.2.1 Lema Si $T = (a, b, c)$ es un triángulo con vértices orientados en sentido antihorario y q es un punto arbitrario del plano, entonces $A(T) = A(q, a, b) + A(q, b, c) + A(q, c, a)$.

1.4.2.2 Teorema (Área de un polígono) Sea P un polígono de n vértices orientados en sentido antihorario y sea q un punto arbitrario en el plano, entonces:

$$A(P) = A(q, v_0, v_1) + A(q, v_1, v_2) + \ldots + A(q, v_{n-2}, v_{n-1}) + A(q, v_{n-1}, v_0) \qquad (1.4)$$

Prueba: La prueba se hará por inducción. Si $n = 3$, el polígono es un triángulo y el teorema se satisface; por el *lema 1.4.2.1.*

Supongamos que la *ecuación 1.4* es válida para todo polígono de $3, 4, 5, \ldots, n-1$ vértices. Por hipótesis P tiene n vértices, aplicando el *teorema 1.2.12* P tendría una oreja. Reenumere los vértices de P de modo que $E = (v_{n-2}, v_{n-1}, v_0)$ sea una oreja. Si P_{n-1} es un polígono obtenido al recortar la oreja E. Por hipótesis de inducción se tiene que:

$$A(P_{n-1}) = A(q, v_0, v_1) + A(q, v_1, v_2) + \ldots + A(q, v_{n-3}, v_{n-2}) + A(q, v_{n-2}, v_0)$$

Además, por el *lema 1.4.2.1* se tiene que:

$$A(E)=A(q,v_{n-2},v_{n-1})+A(q,v_{n-1},v_0)+A(q,v_0,v_{n-2})$$

Por todo lo anterior: $A(P)=A(P_{n-1})+A(E)$.

Como $A(q,v_0,v_{n-2})=-A(q,v_{n-2},v_0)$, entonces se obtiene la *ecuación 1.4*. ■

1.4.2.3 Nota Si $v_i=(x_i,y_i)$, la *ecuación 1.4* es equivalente a:

$$2A(P)=\sum_{i=0}^{n-1}(x_i\,y_{i+1}-y_i\,x_{i+1}) \qquad (1.5)$$

1.4.3 Código para calcular el área de un polígono El problema de calcular el área de un polígono se reduce a realizar una implementación directa de la *ecuación 1.4* donde $q=v_0$.

```
Función Area2(a,b,c: TPunto):Entero;
Inicio
  Area2 :=  a.x*b.y-a.y*b.x+a.y*c.x-a.x*c.y+b.x*c.y-c.x*b.y;
Fin;

Función AreaPoli2(P: TPoligono; n:Entero):Entero;
Inicio
  sum:= 0;
  Para  i:=1  hasta  (P.tam -2)  haga
     sum:= sum + Area2(P.poli[0],P.poli[i],P.poli[i+1]);
  AreaPoli2:= sum;
Fin;
```

Codigo1.3 Area2 y AreaPoli2.

La *función Area2* puede presentar un desbordamiento en la capacidad para almacenar enteros, cuando se multiplica coordenadas muy grandes. Una posible solución para este problema es implementar rutinas para sumar y multiplicar enteros grandes.

1.5 INTERSECCIÓN DE SEGMENTOS

1.5.1 Diagonales El paso clave para triangular un polígono es encontrar una diagonal (una línea recta entre dos vértices claramente visibles).

El siguiente lema es consecuencia inmediata de la definición de diagonal y será fundamental en la construcción del algoritmo de triangulación.

1.5.2 Lema El segmento $s = v_i\, v_j$ es una diagonal de P si y sólo si:

1. Para todo lado e del polígono P que no incida a uno u otro v_i ó v_j, s y e no se interceptan: $s \cap e = \phi$.
2. s es interno a P en un a vecindad de v_i y v_j.

Ahora se desarrollará un código que permite verificar la condición 1 del *lema 1.5.2.*

1.5.3 Función Izquierda La función izquierda se utilizará para decidir si dos segmentos se interceptan y establece si un punto está o no a la izquierda de una recta dirigida (una recta dirigida esta determinada por dos puntos dados en un orden particular (a, b)), si un punto c está a la izquierda de la recta

ab , la tripleta (a,b,c) forma un circuito antihorario: Esto significa estar a la izquierda de una recta.

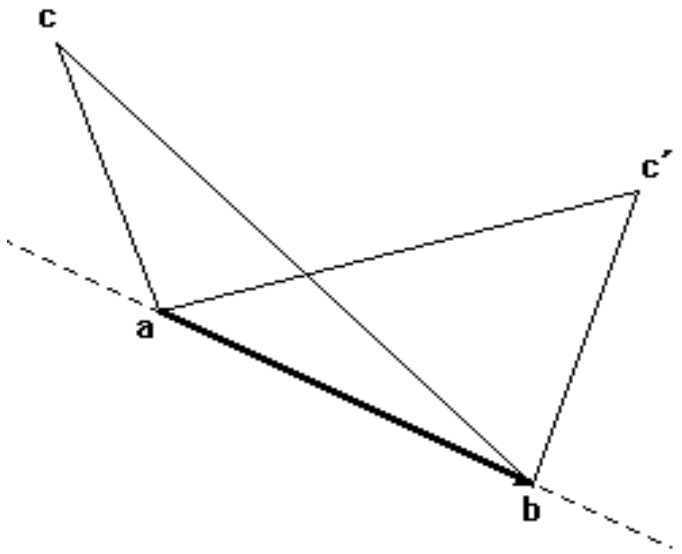


Figura 1.9 c esta a la izquierda de ab si y sólo si el triangulo abc tiene área positiva; también $abc´$ tiene área positiva.

Se puede implementar la función *Izquierda* mediante un llamado simple de *Area2*.

```
Función Izquierda(a,b,c: Tpunto): Booleano;
Inicio
  Si Area2(a,b,c)> 0 entonces
    Izquierda :=  verdadero
  Sino
    Izquierda :=  falso;
Fin;
```

Codigo1.4. función Izquierda.

La función *Izquierda* también puede ser implementada encontrando la ecuación de la recta que pasa a través de a y b, y sustituyendo las coordenadas del punto c en la ecuación. Este método podría ser directo, pero sujeto a casos especiales, mientras que el código que utiliza el área no tiene casos especiales.

Cuando el triángulo determinado por a, b y c tiene área cero, se dice que c es colineal con ab, se escribirá una función *Colineal* para esto. Se utilizará además una función *IzqSobre* para determinar si c está a la izquierda o sobre la recta que contiene al vector ab.

```
Funcion IzqSobre(a,b,c :Tpunto ): booleano;
Inicio
  Si Area2(a,b,c) ≥ 0 entonces
    IzqSobre :=  verdadero
  Sino
    IzqSobre :=  falso;
Fin;
```

```
Funcion Colineal(a,b,c :Tpunto ): booleano;
Inicio
  Si Area2(a,b,c)= 0 entonces
    Colineal :=  verdadero
  Sino
    Colineal :=  falso;
Fin;
```

Código 1.5 Funciones IzqSobre y Colineal.

Observación: En las rutinas *Izquierda, IzqSobre* y *Colineal* se realizaron comparaciones utilizando dos veces el área en lugar del área misma, esto con el fin de trabajar en el confortable dominio de los enteros.

1.5.4 Intersección Booleana Sean L_1 la recta que contiene al vector ab y L_2 la recta que contiene al vector cd. Los segmentos ab y cd se interceptan propiamente si c y d son partidos por L_1 (c esta a un lado de L_1 y d esta al otro lado), y además a y b son partidos por L_2. A continuación se mostrará el código para la intersección propia.

```
Funcion IntersectProp(a,b,c,d :Tpunto ): booleano;
Inicio
  si Colineal(a,b,c) y/o Colineal(a,b,d) y/o Colineal(c,d,a) y/o Colineal(c,d,b)
  entonces
    IntersecProp :=  falso
 sino
   IntersecProp := (Izquierda(a,b,c) ó Izquierda(a,b,d)) y (Izquierda(c,d,a) ó
                    Izquierda(c,d,b));
fin;
```

Código 1.6 Función IntersecProp.

1.5.4.1 Intersección Impropia La intersección impropia ocurre cuando un punto extremo de un segmento (sea c) se encuentra sobre otro segmento ab. Esto solamente ocurre cuando a, b y c son colineales, sin embargo la colinealidad no es suficiente para garantizar la intersección.

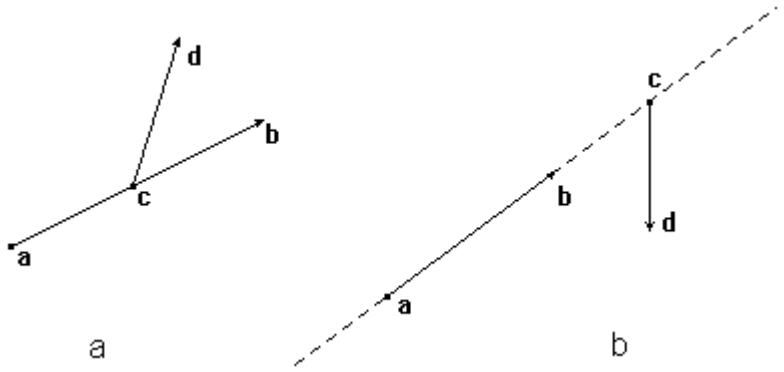


Figura 1.10 (a) Intersección impropia entre dos segmentos; (b) La colinealidad no es suficiente.

1.5.4.2 Función Entre ¿ Qué se necesita para decidir si c esta entre a y b ? Si el segmento ab no es vertical y c es colineal a ab , c sobre ab si y sólo si la coordenada x de c , se encuentra en el intervalo determinado por las coordenadas x de a y b . Si ab es vertical, una verificación similar de las coordenadas y determina si c está entre a y b . Esto conduce al siguiente código.

```
Funcion Entre(a,b,c :Tpunto ): booleano;
Inicio
  Si  no(Colineal(a,b,c)) entonces
   Entre :=  falso
  sino
    Si (a[x]≠b[x]) entonces
        Entre:=  (a.x<=c.x) y (c.x<=b.x ) y/o (a.x>=c.x) y (c.x>=b.x);
    sino
        Entre :=  (a.y<=c.y) y (c.y<=b.y) y/o (a.y>=c.y) y (c.y>=b.y);
Fin;
```

Código 1.7 Función Entre.

1.5.5 Código para la intersección de segmentos Finalmente se puede presentar un código para calcular la intersección de segmentos. Dos segmentos se interceptan si y sólo si un extremo de un segmento se encuentra entre los dos extremos del otro segmento o si ellos se interceptan propiamente. La verificación de la intersección impropia puede hacerse realizando cuatro llamados a la *función Entre,* como se muestra en el siguiente código.

```
Funcion Intersec(a,b,c,d :Tpunto ): booleano;
Inicio
  Si IntersecProp(a,b,c,d) entonces
    Intersec := verdadero
  sino
     Si Entre(a,b,c) y/o Entre(a,b,d) y/o Entre(c,d,a) y/o Entre(c,d,b)
entonces
       Intersec := verdadero
     sino
       Intersec := falso;
Fin;
```

Codigo 1.8 Función Intersec.

1.6 IMPLEMENTACIÓN DE LA TRIANGULACIÓN

1.6.1 Diagonales Internas ó Externas Si se ignora la distinción entre diagonales internas y externas, encontrar diagonales es una aplicación directa y repetida de la *función intersec:*

Sea S un segmento que une dos vértices no consecutivos en un polígono P (una diagonal potencial). S es una diagonal (interna o externa) de P si

y sólo si para todo lado e de P que no incide con los extremos de s se tiene que $e \cap s = \phi$. Esto se observará claramente en el siguiente código.

```
Funcion DiagExtInt(i,j: entero; P: Tpoligono ): booleano;
Inicio
bandera:= verdadero;
 para  k:=0  hasta (P.tam -1) haga
      Si  bandera  entonces
        k1:= (k+1) mod P.tam;
        Si  no ((k=i)  y/o  (k1=i)  y/o  (k=j)  y/o  (k1=j))  entonces
          Si  Intersect(P.poli[i],P.poli[j],P.poli[k],P.poli[k1])  entonces
            bandera:= falso;
 DiagExtInt :=  bandera;
Fin;
```

Código 1.9 Función DiagExtInt.

1.6.2 Función IntCono Dada una diagonal (interna o externa) de un polígono, el objetivo es diferenciar una diagonal interna de una externa.

Sea $s = v_i v_j$ una diagonal interna o externa de un polígono P , entonces s no intercepta a ningún lado de P . Si s es interior a P en una vecindad de v_i y en una vecindad de v_j , entonces s es interior a P a lo largo de su longitud. De esta manera, verificar la interioridad de s se restringe a vecindades de los extremos de s . Además, como estas vecindades pueden ser arbitrariamente pequeñas, la interioridad depende solamente de los lados de P que inciden con los puntos extremos de s , ningún otro lado de P se involucra en la decisión.

Una segunda observación importante es que solamente es necesario examinar uno de los puntos extremos de s: si s es interior a P en una vecindad de v_i, entonces s debe ser interior en una vecindad de v_j; similarmente para el caso exterior. De esta manera, el problema se reduce a verificar si s es interior en una vecindad de v_i.

Caso 1. Supongamos que v_i es convexo como se ilustra en la *figura 1.11(a).* s es interno a P si y sólo si s está en el interior del cono cuyo ápice es v_i, y cuyos lados pasan a través de v_{i-1} y v_{i+1}. Esto se puede verificar utilizando la *función Izquierda*: v_{i-1} debe estar a la izquierda $v_i v_j$ y v_{i+1} debe estar a la izquierda $v_j v_i$; de acuerdo a la definición de diagonal ambos llamados deben realizarse a la función *Izquierda* estricta.

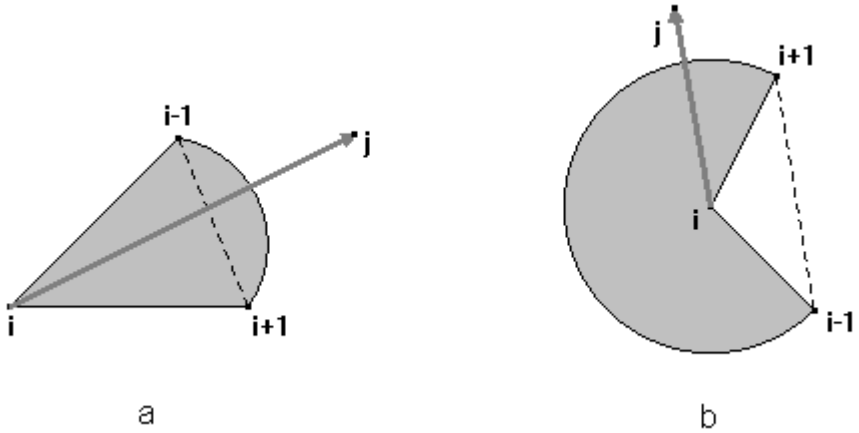


Figura 1.11 La Diagonal $s = ij$ está en el cono determinado por v_{i-1}, v_i, v_{i+1}: (a) convexo, (b) reflejo. En (b) ambos v_{i-1} y v_{i+1} están a la derecha de ij.

Caso 2. Supongamos que v_i es reflejo como se muestra en la *figura 1.11(b)*. Basta observar que el exterior de una vecindad de v_i es como en el caso

convexo, si el interior y el exterior se intercambian. Así, S es interior a P si y sólo si S no está en el exterior de P.

La distinción entre el caso convexo y el caso reflejo es fácilmente establecida por un llamado a la *función IzqSobre:* v_i es convexo si y sólo si v_{i+1} está a la izquierda o sobre $v_{i-1}v_i$.

Recuerde que si v_{i-1}, v_i, v_{i+1} son colineales entonces el ángulo interno a v_i es π el cual se ha definido como convexo. Las ideas anteriores son implementadas en el siguiente código.

```
Funcion IntCono(i,j: entero; P: Tpoligono): booleano;
Inicio
  i1 := (i+1)mod P.tam;
  in1 := (i+n-1) mod P.tam;
  Si IzqSobre(P.poli[in1],P.poli[i],P.poli[i1]) entonces
    IntCono := Izquierda(P.poli[i],P.poli[j],P.poli[in1]) y
               Izquierda(P.poli[j],P.poli[i],P.poli[i1])
  sino
    IntCono := no (IzqSobre(P.poli[i],P.poli[j],P.poli[i1]) y
                   IzqSobre(P.poli[j],P.poli[i],P.poli[in1]));
Fin;
```

Código 1.10 Función IntCono.

La *función IntCono* retorna verdadero si y sólo si la diagonal v_iv_j es estrictamente interna al polígono P en una vecindad de v_i.

1.6.3 Función Diagonal $v_i v_j$ es una diagonal interna de P si y sólo si *DiagExtInt(i,j,P)* y *IntCono(i,j,P)* son ambas verdaderas.

```
Funcion Diagonal(i,j: entero; P: Tpoligono): booleano;
Inicio
  Diagonal :=  IntCono(i,j,P) y DiagExtInt(i,j,P);
Fin;
```

Código 1.11 Función Diagonal.

Obsérvese que en el código anterior se llama primero a la *función IntCono*, esto debido que tiene tiempo de complejidad *O(1)*, mientras que la *función DiagExtInt* es de tiempo $O(n^2)$. Así, si *IntCono* es falsa, no se verifica *DiagExtInt.* La *función Diagonal* retorna verdadero si y sólo si $v_i v_j$ es una diagonal interna de P .

1.6.4 Triangulación mediante el cortado de orejas Para triangular un polígono primero se encuentra una oreja, se corta y se aplica recursividad sobre el polígono restante. Estas ideas se pueden ver en el siguiente algoritmo.

```
Triangulación
Si n>3 entonces
  Para cada diagonal oreja potencial (i,i+2) haga
    Si diagonal(i,i+2,n,P) entonces
      Imprima diagonal
      Cortar oreja en v_i
     Aplicar recursividad al resto del polígono
```

Algoritmo 1.1 Triangulación.

1.6.4.1 Recortado de oreja En el *algoritmo 1.1* se requiere una rutina que recorte una oreja. El procedimiento SuprimOreja es implementado para suprimir un vértice oreja v_i de un polígono P, dando como resultado un nuevo polígono con un vértice menos.

```
Procedimiento SuprimOreja(i:entero; P: Tpoligono);
Inicio
  Para k := i  hasta (P.tam -2)  haga
    P.poli[k] :=   P.poli[k+1];
Fin;
```

Codigo 1.12 Procedimiento SuprimOreja.

1.6.4.2 Trazado de diagonales En el *algoritmo 1.1* después de encontrar una diagonal, es necesario que ésta se trace, para esto, las coordenadas correspondientes a cada diagonal son almacenadas en un arreglo de diagonales. Otra posible alternativa es almacenar los índices de los vértices. Ahora se definirá un tipo para arreglos de diagonales.

```
Tipo
TDiagonal = Registro
     v1: Tpunto;
     v2: Tpunto;
  Fin;
  TArreglo_Diagonales = Arreglo  de  Tdiagonal;
  TDiagonales_Triangulacion = registro
    lista: TArreglo_Diagonales;
    tam: entero;
  fin;
```

Codigo 1.13 Tipo TarrDiagonal.

1.6.5 Procedimiento Triangular Ahora se presenta el código para triangular un polígono arbitrario de n vértices, teniendo en cuenta las consideraciones anteriores.

```
Procedimiento Triangular(var P:Tpoligono; varL:Tdiagonales_Triangulacion;
var j: entero);
Inicio
  Si  (P.tam >3)  entonces
     Para i:=0  hasta  (P.tam -1)  haga
        i1:= (i+1) mod P.tam;
        i2:= (i+2) mod P.tam;
        Si  Diagonal(i,i2,P)  entonces
           tempo.v1:= P.poli[i];
           tempo.v2:= P.poli[i2];
           L.tam:= j+1;
           Asignar Longitud (L.lista,L.tam);
           L.lista[j]:= tempo;
           j:= j+1;
           SuprimOreja (i1,P);
           P.tam:= P.tam -1;
           Triangular(P,L,j);
Fin;
```

Codigo 1.14 Procedimiento Triangular.

1.6.6 Análisis de complejidad del algoritmo de triangulación Sea $T(n)$ el tiempo de complejidad del algoritmo de triangulación operando sobre un polígono P de n vértices.

El peor caso que se puede presentar en el algoritmo, sucede cuando todas las diagonales potenciales resultan ser diagonales oreja, una vez encontrada la primera diagonal oreja, la oreja es cortada, el polígono reducido es triangulado mediante un llamado recursivo y el ciclo *Para* es abandonado.

Triangulación	$[T(n)]$
Si *n>3* ***entonces***	
Para *cada diagonal oreja potencial (i,i+2)* ***haga***	$[O(n)]$
Si *diagonal(i,i+2,n,P)* ***entonces***	$[O(n)]$
Imprima diagonal	$[O(1)]$
Cortar oreja en v_i	$[O(n)]$
Aplicar recursividad al resto del polígono	$[T(n-1)]$

Algoritmo 1.2 Algoritmo 1.1 con los respectivos tiempos de complejidad.

De todo lo anterior se tiene que:

$T(n) = O(n) \times O(n) + O(1) + O(n) + T(n-1)$

$T(n) = O(n^2) + O(1) + O(n) + T(n-1)$; Por la regla del producto.

$T(n) = O(n^2) + T(n-1)$; Por la regla del máximo.

Una forma de resolver esta relación de recurrencia es expandiéndola:

$T(n) = O(n^2) + T(n-1)$

$T(n) = O(n^2) + O((n-1)^2) + T(n-2)$

$T(n) = O(n^2) + O((n-1)^2) + O((n-2)^2) + T(n-3)$

$T(n) = O(n^2) + O((n-1)^2) + O((n-2)^2) + \cdots + T(3)$

Por lo tanto,

$$T(n)=\sum_{k=3}^{n} O(k^2)=O((2n^3+3n^2+n)/6)=O(n^3)$$

En conclusión el tiempo de complejidad del algoritmo de triangulación es $O(n^3)$.

Otra forma de solucionar la relación de recurrencia $T(n)=O(n^2)+T(n-1)$ es:

Sea $T(n)=a_{n+1}$, entonces la relación de recurrencia es equivalente a

$$a_{n+1}=a_n+n^2,\ \ a_3=1,\ \ n\geq 3. \qquad (1.6)$$

En este caso, la relación homogénea asociada es $a_{n+1}-a_n=0$, para la cual $a_n^{(h)}=c(1)^n=c$, donde c es una constante arbitraria. Por otro lado, podemos suponer que la relación particular es de la forma $a_n^{(p)}=A_2 n^3+A_1 n^2+A_0 n$. Sustituyendo este resultado en la ecuación (1.6) se obtiene

$$A_2(n+1)^3+A_1(n+1)^2+A_0(n+1)=A_2 n^3+A_1 n^2+A_0 n+n^2.$$

Al comparar los coeficientes de las potencias semejantes de n se tiene que

(n^3): $A_2=A_2$

(n^2): $3A_2+A_1=A_1+1$

(n^1): $3A_2+2A_1+A_0=A_0$

(n^0): $A_2+A_1+A_0=0$

Resolviendo este sistema de ecuaciones: $A_2 = \frac{1}{3}$, $A_1 = \frac{-1}{2}$ y $A_0 = \frac{-1}{6}$.

Por lo tanto, $a_n^{(p)} = \frac{1}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n$. y la solución general de la relación de recurrencia es $a_n = a_n^{(h)} + a_n^{(p)} = c + \frac{1}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n$. Como $a_3 = 1$, entonces $1 = c + \frac{1}{3}3^3 - \frac{1}{2}3^2 - \frac{1}{6}3$ y $c = -3$.

De todo lo anterior se tiene que $T(n) = O\left(\frac{1}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n - 3\right) = O(n^3)$.

2. DIAGRAMAS DE VORONOI

Los Diagramas de Voronoi son una de las estructuras geométricas más importantes para fraccionar el espacio; almacenan toda la información que se desea conocer acerca de la proximidad de un conjunto finito de puntos (u objetos), respondiendo a preguntas como: ¿Quién es más cercano a quien? y ¿quién es más lejano?.

2.1 DEFINICIONES Y PROPIEDADES BÁSICAS DE LOS DIAGRAMAS DE VORONOI.

2.1.1 Definiciones Sea $P = \{p_1, p_2, ..., p_n\}$ un conjunto de n puntos distintos en el plano llamados *sitios o posiciones*.

Se define la *célula o región de Voronoi* de p_i, denotada por $V(p_i)$, al lugar geométrico de los puntos del plano más próximos a p_i que a cualquier otro sitio.

$$V(p_i) = \{x : d(x, p_i) \leq d(x, p_j);\ j \neq i\} \qquad (2.1)$$

Note que este conjunto se ha definido cerrado. Algunos puntos de P no tienen una única posición próxima (o vecino próximo). El conjunto de todos los puntos que tienen más de un vecino próximo forman el Diagrama de Voronoi de P, denotado $Vor(P)$, en otras palabras, el Diagrama de Voronoi

de P se define como la partición del plano en n regiones, una por cada posición de P. Los lados de las regiones de Voronoi se llaman *lados de Voronoi* y aquellos puntos que equidistan de tres o más posiciones se llaman *vértices de Voronoi*.

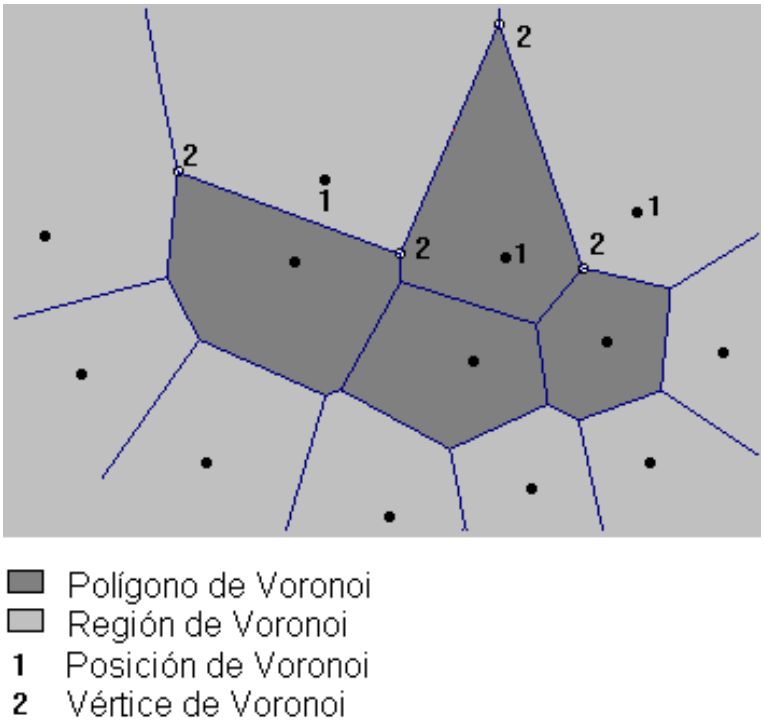


Figura 2.1 Partes del Diagrama.

Para dos puntos p y q en el plano se define su bisector, denotado por $B(p,q)$ como la mediatriz del segmento pq. Sea $H(p_i, p_j)$ el semiplano cerrado con frontera $B(p_i, p_j)$ que contiene a p_i, entonces $H(p_i, p_j)$ es el conjunto de todos los puntos mas próximos a p_i que a p_j. De donde se tiene que la región de Voronoi $V(p_i)$ es la intersección de $(n-1)$ semiplanos:

$$V(p_i) = \bigcap_{j=1,\, j\neq i}^{n} H(p_i, p_j) \qquad (2.2)$$

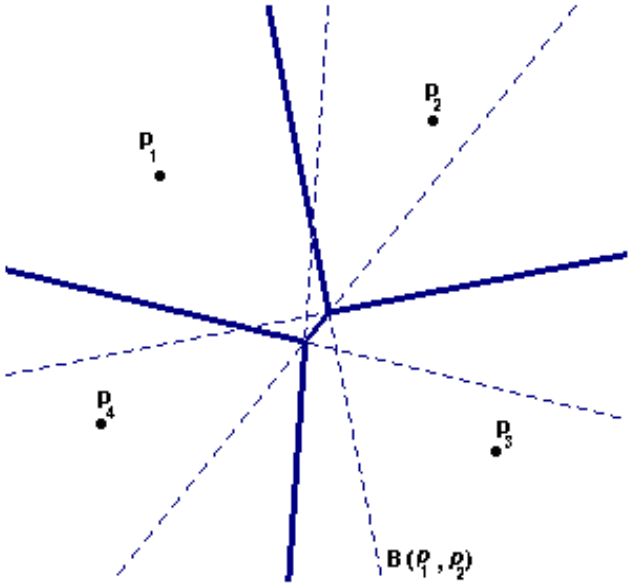


Figura 2.2 Intersección de Semiplanos.

De la *ecuación 2.2*, se deduce que toda región de Voronoi tiene a lo más $(n-1)$ lados y es convexa, dado que los semiplanos son conjuntos convexos y la intersección finita de conjuntos convexos es convexa. Si la región es limitada se denomina polígono de Voronoi.

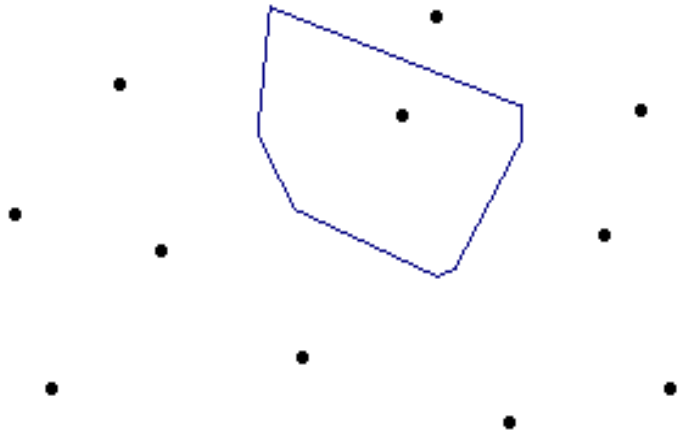


Figura 2.3 Polígono de Voronoi

2.1.2 Diagramas de Voronoi para pocas posiciones.

Dos posiciones Supongamos que $P=\{p_1, p_2\}$. Entonces todo punto x sobre el bisector $B(p_1, p_2)$ es equidistante de p_1 y p_2. Esto puede verse dibujando el triángulo (p_1, p_2, x) como se muestra en la *figura 2.4*. Por el teorema lado-ángulo- lado (a, θ, b) de Euclides, se tiene que $d(x, p_1)=d(x, p_2)$. Por lo tanto, el Diagrama de Voronoi de P esta constituido únicamente por $B(p_1, p_2)$.

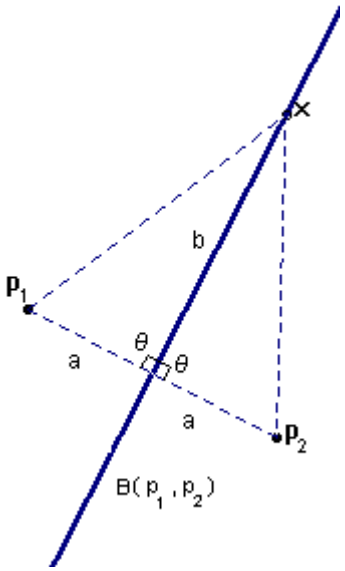


Figura 2.4 Diagrama de Voronoi para dos posiciones

Tres posiciones Para $P=\{p_1, p_2, p_3\}$ el Diagrama de Voronoi consta de los bisectores $B(p_1, p_2)$, $B(p_2, p_3)$ y $B(p_3, p_1)$. Por Euclides, estos se intersectan en un punto v llamado circuncentro, el cual equidista de las tres posiciones y es llamado *vértice de Voronoi.*. Así el Diagrama de Voronoi de P aplicando la *ecuación 2.2* es como aparece en la *figura 2.5.*

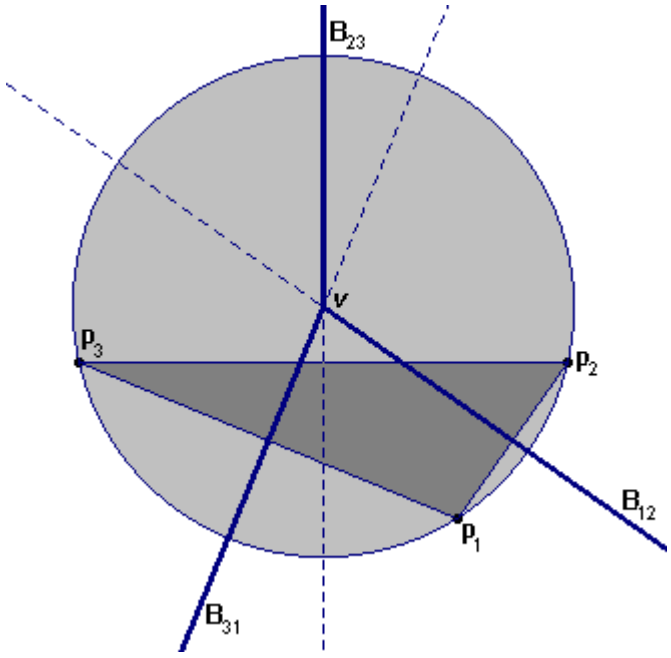
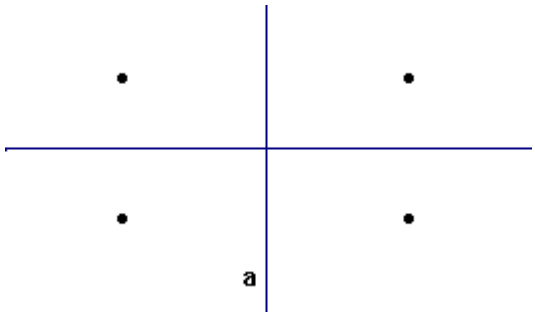


Figura 2.5 Diagrama de Voronoi para tres posiciones

Cuatro posiciones. El Diagrama de Voronoi para cuatro posiciones cocirculares (las esquinas de un rectángulo) se muestra en la *figura 2.6.a.* Observe que el vértice de Voronoi es de grado cuatro. Este Diagrama es considerado como un caso *degenerado* o *anormal.*

Suponga ahora que una posición es movida bruscamente como en la *figura 2.6.b* (cuatro posiciones no cocirculares). En este caso se considera el diagrama como *normal.*



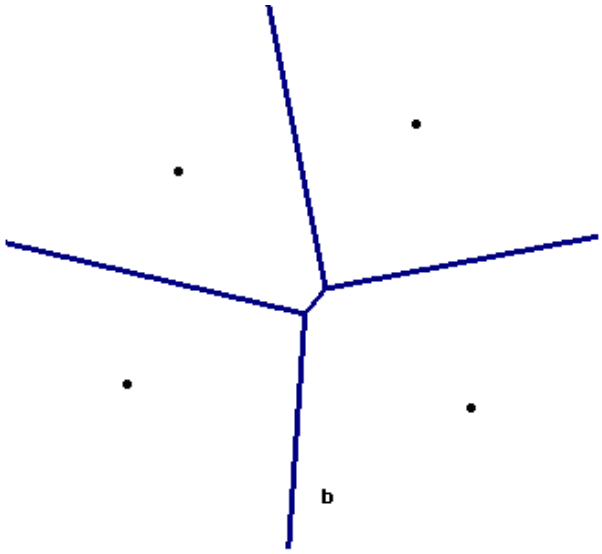


Figura 2.6 a) Diagrama de Voronoi para cuatro puntos cocirculares y b) Diagrama de Voronoi para cuatro posiciones no cocírculares

Se dice que las posiciones o sitios del conjunto P están en posición genérica si no hay tres de ellos alineados o cuatro cocirculares.

Se puede comprobar que la construcción de un diagrama de Voronoi hasta con seis posiciones, aplicando la *ecuación 2.2* es una tarea fácil, pero cuando el conjunto tiene un número de puntos considerable, nos encontramos ante un problema que es sólo factible si se realiza con un computador, de aquí que la generación de dichos diagramas ha sido motivo de estudio, a fin de obtener diseños algorítmicos para calcularlo encontrándose de varios tipos y distintas medidas asintóticas.

En la *figura 2.7* se ilustra el diagrama de Voronoi para 16 posiciones, donde se puede observar que el grado de dificultad en la construcción manual del diagrama aumenta.

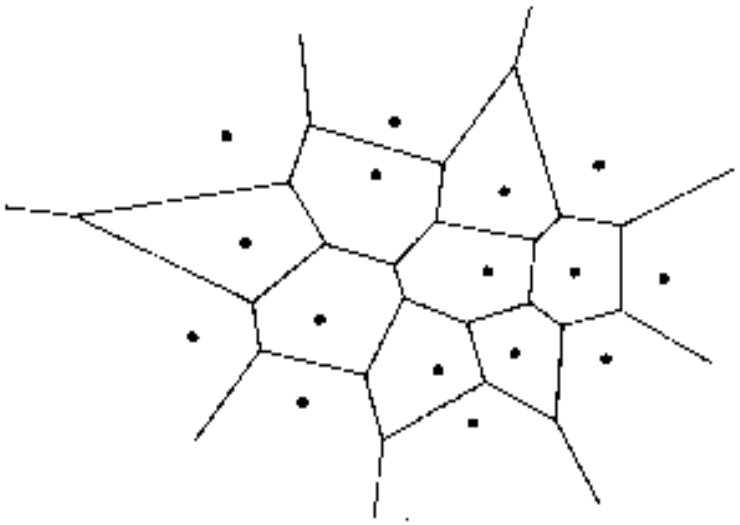


Figura 2.7. Diagrama de Voronoi para 16 posiciones.

2.2 TRIANGULACIÓN DE DELAUNAY

La Teoría de Grafos juega un papel importante en la fundamentación matemática de las Ciencias de la Computación. Los grafos constituyen una herramienta básica para modelar fenómenos discretos y son fundamentales para la comprensión de las estructuras de datos y el análisis de algoritmos.

Formalmente: Un grafo es una pareja $G = (V, A)$, donde V es un conjunto de puntos llamados vértices y A es un conjunto de pares de vértices, llamadas aristas. Para simplificar, la arista $\{a, b\}$ se denota ab.

Se dice que un grafo es *plano* si admite una inmersión en el plano (es decir, si se puede dibujar en un plano sin que haya cruces). Véase *figura 2.8*.

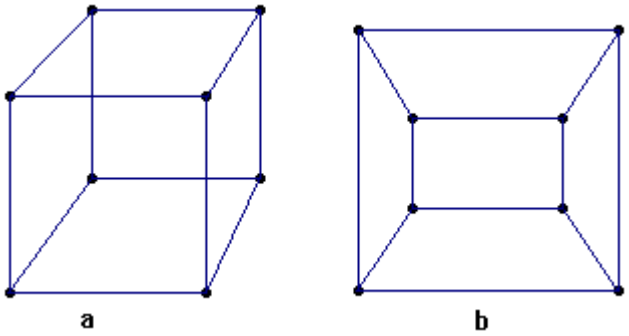


Figura 2.8 a) Grafo no planar, b) Grafo planar

Grafo dual: Dado un grafo plano $G = (V, A)$, se define el grafo dual de G denotado por $G' = (V', A')$ como:

1) Cada cara de G se identifica con un vértice de G'.
2) Cada arista e de G da lugar a una arista de G'; entre los dos vértices identificados por las caras que separa la arista e.

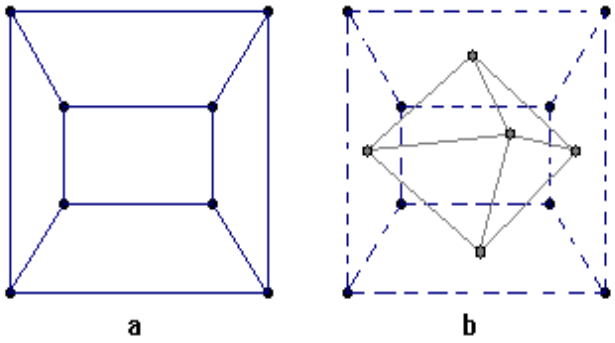


Figura 2.9 a) Grafo plano, b) Grafo dual superpuesto al grafo plano.

2.2.1 Definición (Triangulación de Delaunay) Dado un conjunto de n puntos en el plano $P = \{p_1, p_2, ..., p_n\}$ todos distintos, $Vor(P)$ es un grafo plano con n caras (las regiones de Voronoi). Se define la triangulación de Delaunay de P, denotada por $TD(P)$, como el grafo dual de $Vor(P)$. Los vértices de $TD(P)$ son las posiciones de P. Si las regiones de Voronoi

correspondientes a p_i y p_j comparten un lado entonces estos dos vértices se conectan.

Si suponemos que los puntos de P están en posición genérica (no hay tres de ellos alineados, ni cuatro cocirculares) se tiene que $TD(P)$ es una triangulación de P (lo que significa que es un grafo rectilíneo plano cuyos vértices son los puntos de P y cuyas caras son triángulos). En la *figura 2.10* se muestra la triangulación de Delaunay para el Diagrama de Voronoi de la *figura 2.7* y en la *figura 2.11* se muestra la triangulación de Delaunay superpuesta sobre el Diagrama de Voronoi.

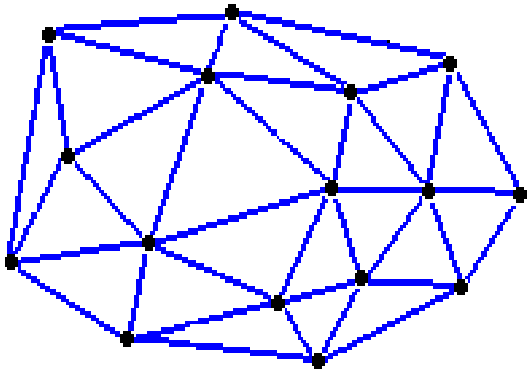


Figura 2.10 Triangulación de Delaunay.

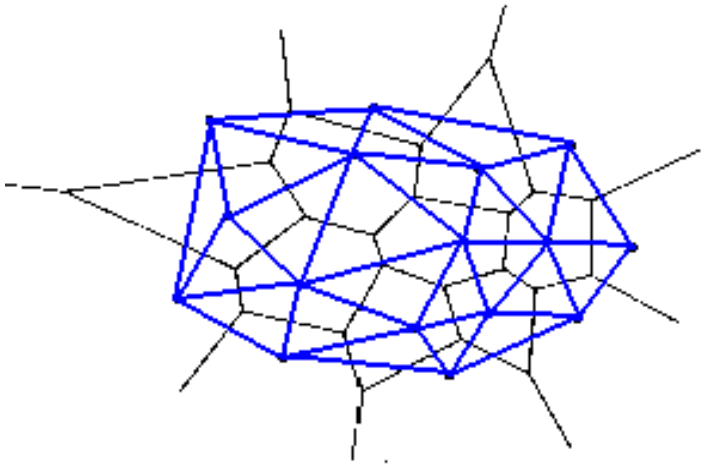


Figura 2.11 Triangulación de Delaunay superpuesta al Diagrama de Voronoi.

2.2.2 Propiedades de la triangulación de Delaunay Es importante resaltar la relación que existe entre la triangulación de Delaunay y su correspondiente Diagrama de Voronoi. Estas estructuras son duales, cada una contiene la misma información, aunque su representación es diferente. A continuación se enunciaran algunas propiedades de la triangulación de Delaunay, seguidas de una lista de propiedades del Diagrama de Voronoi.

Sea $P = \{p_1, p_2, ..., p_n\}$ un conjunto finito y fijo de posiciones.

D1. $TD(P)$ es el grafo dual de $Vor(P)$. (Esto es por definición)

D2. $TD(P)$ es una triangulación de P si y sólo si, los puntos de P están en posición genérica. Las caras de $TD(P)$ son llamadas *triángulos de Delaunay*.

D3. Cada cara de $TD(P)$ corresponde a un vértice de $Vor(P)$.

D4. Cada lado de $TD(P)$ corresponde a un lado de $Vor(P)$.

D5. Cada nodo de $TD(P)$ corresponde a una región de $Vor(P)$.

D6. La frontera de $TD(P)$ es la envolvente convexa de las posiciones.

D7. El interior de cada cara de $TD(P)$ no contiene posiciones.

Las propiedades *D6* y *D7* son tal vez las más interesantes y se observan en las *figuras 2.10* y *2.11.*

2.2.3 Propiedades del Diagrama de Voronoi.

V1. Cada región de Voronoi es un conjunto convexo.

V2. $V(p_i)$ es ilimitada si y sólo si, p_i esta sobre la envolvente convexa de P.

V3. Si v es un vértice de Voronoi de la unión de las regiones $V(p_1), V(p_2)$ y $V(p_3)$, entonces v es el centro del círculo $C(v)$

determinado por p_1, p_2 y p_3. (Esto es generalizado para los vértices de Voronoi de cualquier grado).

V4. $C(v)$ es el circuncírculo del triángulo de Delaunay correspondiente a v.

V5. El interior de $C(v)$ no contiene posiciones.

V6. Si p_j es el vecino más próximo a p_i, entonces (p_i, p_j) es un lado de $TD(P)$.

V7. Si existe un círculo que pasa a través de p_i y p_j que no contiene otras posiciones, entonces (p_i, p_j) es un lado de $TD(P)$. (El reciproco también se cumple: Para todo lado de Delaunay existe un círculo vacío).

2.2.4 Definición de Círculo Vacío Sean $a, b \in P$ y C un círculo cuya frontera pasa a través de a y b. Se dice que C es vacío si no contiene otras posiciones de P distintas de a y b.

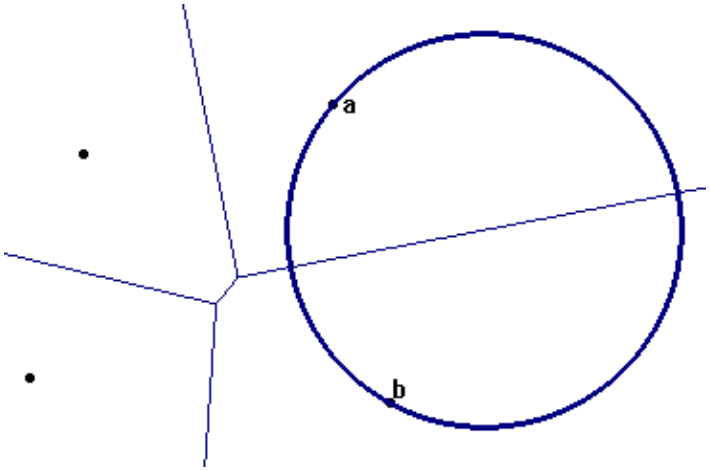


Figura 2.12 Círculo vacío a través de a y b

2.2.5 Teorema $ab \in TD(P)$ si y sólo si existe un círculo vacío a través de a y b.

Prueba: Supongamos que $ab \in TD(P)$ entonces $V(a)$ y $V(b)$ comparten un lado e de Voronoi. Sea x en el interior de e y $C(x)$ un círculo con centro en x y radio igual a $d(x,a)$, con $C(x)$ que contiene una posición d diferente de a y b. Entonces x esta en $V(a)$ y $V(b)$ únicamente. Luego no es posible que $C(x)$ contenga otra posición diferente de a y b. Por lo tanto $C(x)$ es un círculo vacío a través de a y b.

Recíprocamente, Supongamos que de $C(x)$ es un círculo vacío que pasa a través de a y b con centro en x, entonces $C(x)$ no contiene ninguna otra posición diferente de a y b. Luego $x \in V(a) \cap V(b)$, de donde se tiene que a es el vecino mas próximo de b. Aplicando la propiedad V6, se tiene que $ab \in TD(P)$.

2.3 IMPLEMENTACIÓN DEL ALGORITMO DEL DIAGRAMA DE VORONOI

2.3.1 Descripción del Método Incremental Green y Sibson en el año de 1977 calcularon el diagrama de Voronoi por inserción incremental. La idea básica es partir de un diagrama ya construido para un conjunto $S = \{p_1, p_{2,}, ..., p_k\}$ y obtener el Diagrama de Voronoi para el conjunto $P = S \cup \{p_{k+1}\}$ insertando un nuevo sitio p_{k+1}. Para insertar un nuevo sitio al diagrama $Vor(S)$ se deben seguir los siguientes pasos:

Paso 1: Encontrar la región de Voronoi que contiene al punto p_{k+1}. Para esto se busca la posición más próxima a p_{k+1}. Supongamos que p_{i1} es la posición de la región que contiene a p_{k+1}.

Paso 2: El bisector $B(p_{i1}, p_{k+1})$ corta la frontera de $V(p_{i1})$ en dos puntos x_1 y x_2 como se muestra en la *figura 2.13*, donde p_{k+1} está a la izquierda del vector x_1x_2. De esta manera el segmento x_1x_2 es un nuevo lado de Voronoi que divide la región $V(p_{i1})$ en dos partes.

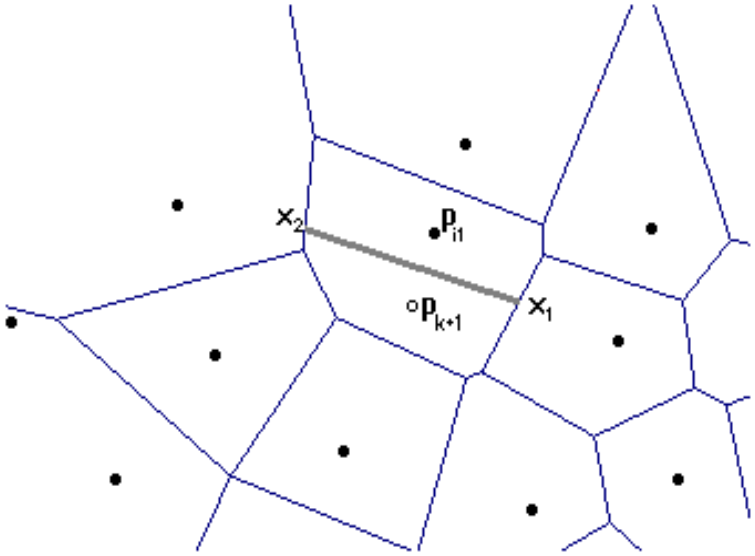


Figura 2.13 Construcción del primer lado de Voronoi en la nueva región.

Paso 3: partiendo del lado inicial x_1x_2 se expande la frontera de la nueva región de Voronoi $V(p_{k+1})$ mediante el siguiente procedimiento:

a) x_2 se encuentra también en una región adyacente a $V(p_{i1})$, digamos $V(p_{i2})$. Entonces el bisector $B(p_{i2}, p_{k+1})$ corta la frontera de $V(p_{i2})$ en dos puntos donde uno de ellos es x_2 y el otro es un punto x_3. Como se muestra en la *figura 2.14.*

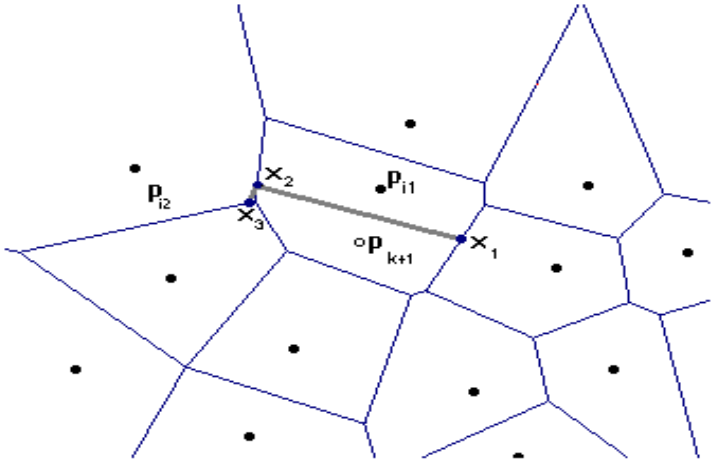


Figura 2.14 Segundo lado de Voronoi en la nueva región.

b) Repitiendo el procedimiento a), se obtiene una sucesión de vértices de Voronoi, hasta alcanzar el vértice x_1 (vértice de partida). Sea dicha sucesión $\{x_1, x_{2,}, ..., x_{m-1}, x_m = x_1\}$. Los nuevos vértices de Voronoi son: $\{x_1, x_{2,}, ..., x_{m-1}\}$. Estos vértices determinan la frontera de la nueva región $V(p_{k+1})$ la cual es un polígono convexo cuyos vértices están orientados en sentido antihorario. Esto se puede ver en la *figura 2.15*

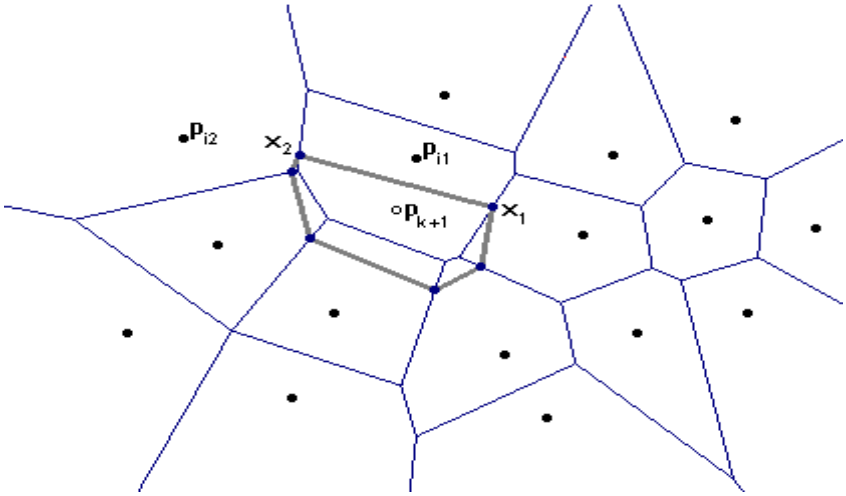


figura 2.15 Frontera de la nueva región.

Paso 4: Se borran los vértices y las porciones de los lados de Voronoi que están en el interior de la nueva región de Voronoi $V(p_{k+1})$. De esta manera se obtiene el Diagrama de Voronoi para el conjunto $P = \{p_1, p_2, ..., p_k, p_{k+1}\}$, como se muestra en la *figura 2.16.*

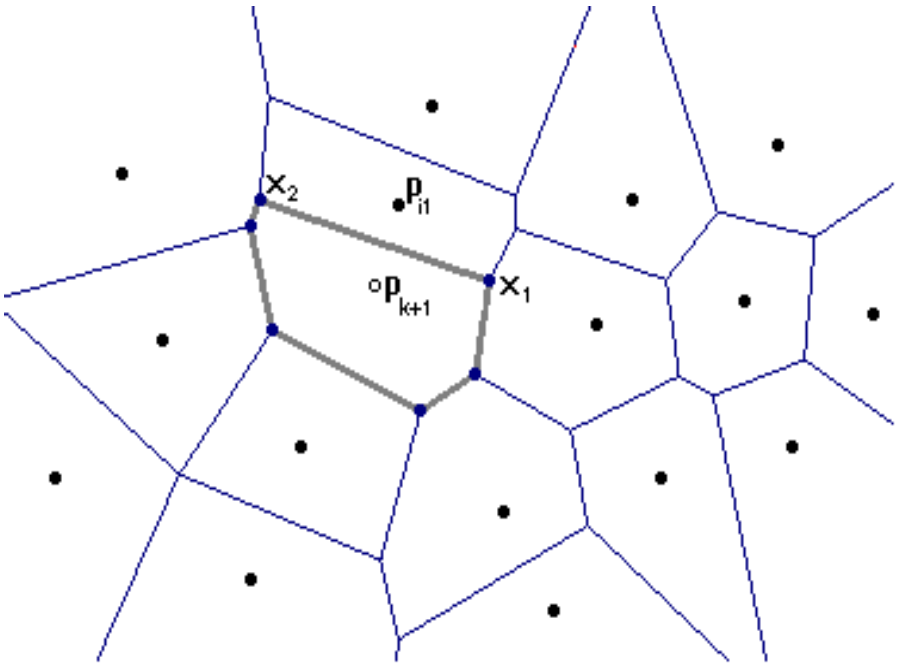


Figura 2.16 Nuevo Diagrama de Voronoi

2.4 IMPLEMENTACIÓN DEL MÉTODO INCREMENTAL EN DELPHI

2.4.1 Representación de posiciones Las posiciones se guardan en arreglos dinámicos y su estructura contiene: a) coordenadas x y y, b) una bandera de tipo booleano.

El siguiente código muestra una manera de definir las posiciones en Delphi.

```
Tipos
  Tpunto=Registro
    x,y : real;
  Fin;
  Tposicion= Registro
    Punto : Tpunto;
    Bandera : Booleano;
  Fin;
  Tpolipos= arreglo de Tposicion;
  Tposiciones_diagrama= Registro
    Lista : Tpolipos;
    Tam :entero;
  Fin;
```

Código 2.1 Representación de posiciones.

2.4.2 Representación de vértices de Voronoi Los vértices de Voronoi se pueden representar en arreglos y en su estructura contienen únicamente las coordenadas x y y. El siguiente código muestra una manera de definir vértices de Voronoi en Delphi.

```
Tipos
  Tpoliver= Arreglo de Tpunto;
  Tvertices_diagrama =Registro
    Lista : Tpoliver;
    Tam :Entero;
  Fin;
```

Código 2.2 Representación de los vértices de Voronoi.

2.4.3 Representación de las regiones de Voronoi en Delphi Las regiones de Voronoi en Delphi se pueden representar como polígonos. En su estructura, cada región de Voronoi consta: a) Posición, b) arreglo de vértices que delimitan la región y c) número de vértices de Voronoi que contiene el polígono. Por otro lado, el diagrama puede verse como la unión de todas las regiones de Voronoi. El siguiente código muestra una forma de definir las regiones y el diagrama de Voronoi en Delphi.

Tipos

*Tregion= **Registro***
Sitio : Tpunto;
Poli : Tpoliver;
*Tam : **entero**;*
Fin;
*Tdiagrama= **Arreglo de** Tregion;*

Código 2.3 Representación de las regiones y del diagrama de Voronoi en Delphi.

2.4.4 Procedimiento para iniciar la construcción del diagrama de Voronoi Para iniciar la construcción del diagrama de Voronoi, en nuestro caso se insertaron cuatro posiciones iniciales (Las esquinas de un rectángulo más grande que la pantalla). Esto es un artificio, el cual se realizó para evitar manejar regiones de Voronoi abiertas e ilimitadas, las cuales pueden dificultar la implementación del método. Las cuatro posiciones iniciales operan internamente, pero no se muestran gráficamente sobre el diagrama. El siguiente algoritmo muestra una manera de iniciar el diagrama de Voronoi en Delphi.

Procedimiento *Reiniciar_Diagrama*

Inicio

Se guarda en P las cuatro posiciones (vértices de rectángulo)

En Vor se guarda las regiones de cada posición.

Fin*;*

Algoritmo 2.1 Procedimiento Reiniciar_Diagrama.

Geométricamente, el resultado después de llamar a *Reiniciar Diagrama* es que se generan cuatro regiones de Voronoi como se muestra en la *figura 2.17,* quedando de esta manera $P = \{p_1, p_2, p_3, p_4\}$.

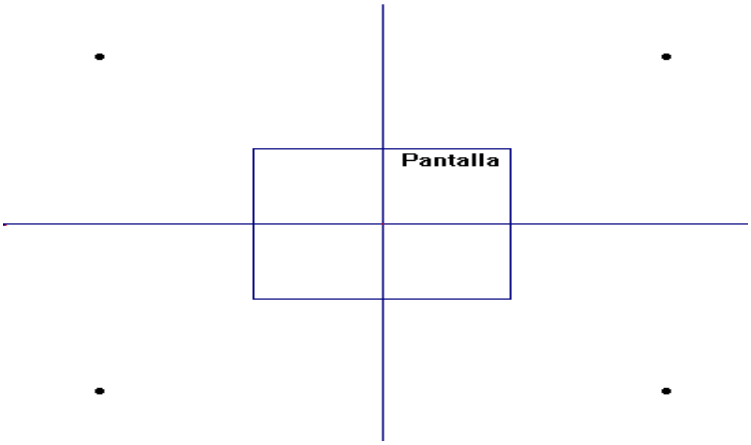


Figura 2.17 Cuatro primeras regiones de Voronoi.

2.4.4.1 Procedimiento para insertar una nueva posición en el diagrama para agregar una nueva posición en el diagrama se necesita tener en cuenta los siguientes pasos:

paso 1. Se agrega la nueva posición a la lista de posiciones del diagrama: $P = \{p_1, p_{2,}, ..., p_{nuevo}\}$.

Paso 2. Se crea una nueva región de Voronoi $V(p_{nuevo})$, calculando los nuevos vértices del diagrama como se describió en la *sección 2.3.1*, obteniéndose así los siguientes resultados, véase *figura 2.18*.

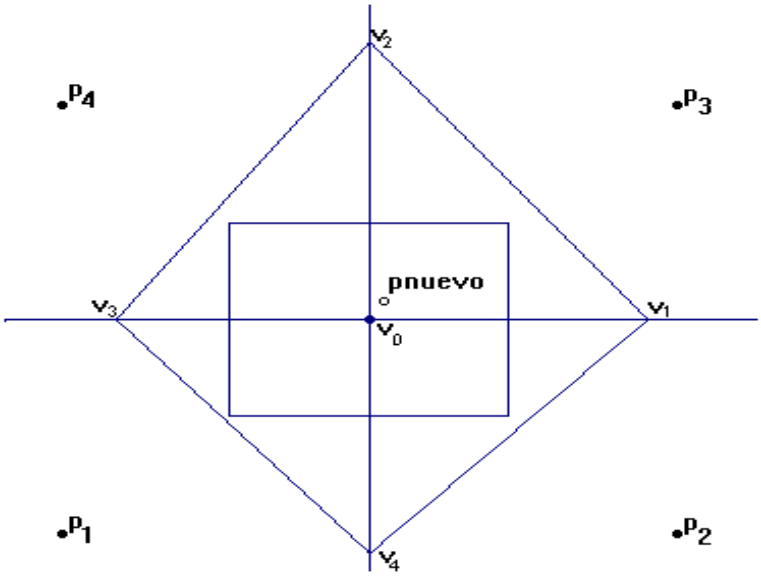


figura 2.18 Diagrama de Voronoi previo para una posición.

$vertnuevos = \{v_1, v_2, v_3, v_4\}$,

$V = \{(0,0)\}$,

$Diagrama = \{reg_1, reg_2, reg_3, reg_4, reg_{nueva}\}$, donde la nueva región es: $reg_{nueva} = \{v_1, v_2, v_3, v_4\}$

paso 3. Se obtiene la lista de los vértices a borrar, es decir, todos los vértices que están por dentro de la nueva región de Voronoi. Resultando así : $VertBorrar = \{(0,0)\} = \{v_0\}$.

Paso 4. Se modifican las regiones que fueron afectadas al crear la nueva región. En el ejemplo que se esta siguiendo, se deben modificar las cuatro primeras regiones, como se muestra en la *figura 2.19.*

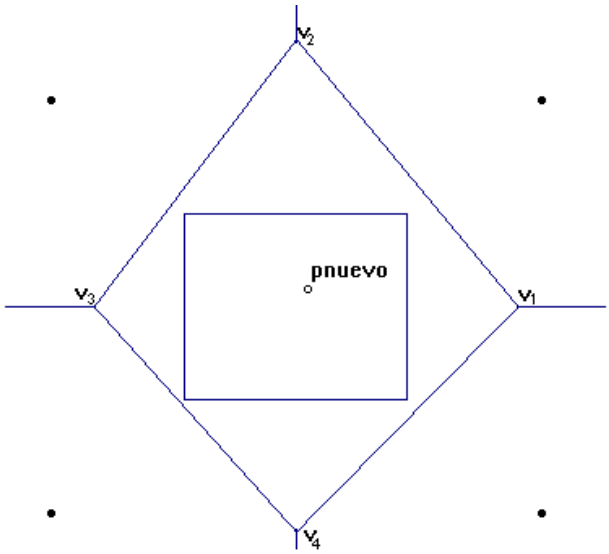


Figura 2.19 Diagrama de Voronoi para una posición.

Paso 5. Se actualizan los vértices, eliminando aquellos que forman parte de $VertBorrar$ y agregando los nuevos vértices: $V = \{v_1, v_2, v_3, v_4\}$. Así, cuando se agrega una nueva posición se tiene que repetir todo el mecanismo descrito anteriormente.

2.4.4.2 Procedimiento nueva_region Después de agregar la nueva posición a la lista de posiciones. El procedimiento nueva_region calcula la región de Voronoi asociada a la nueva posición y guarda los nuevos vértices.

Para la construcción de esté procedimiento se siguieron las ideas presentadas en la *sección 2.3.1*, presentándose el algoritmo a continuación.

Procedimiento *nueva_region(VN: Tvertices_diagrama; R: Tregion);*

Inicio

Se asignan a la bandera de cada una de las P.tam-1 Posiciones, falso;

La posición pnuevo (la P.tam) se le asigna verdadero;

Se encuentra la posición próxima de pnuevo, digamos p_i*;*

A la bandera de dicha posición se le asigna verdadero;

Se calculan los dos puntos en los cuales se interceptan la mediatriz de pnuevo

y p_i*, con la región de* p_i*,*

digamos x_1 *y* x_2*;*

Se agregan estos vértices a vertnuevos ;

Se calcula la posición próxima de x_2 *y se hace el mismo procedimiento anterior*

hasta que se llegue al vértice x_1*;*

A la nueva región se le asigna vertnuevos ;

Fin*;*

Algoritmo 2.2 Procedimiento nueva_region.

En el algoritmo anterior P es de tipo *Tposiciones_diagrama.*

2.4.4.3 Función Posicion_proxima La función Posición_proxima es muy útil para construir la nueva región asociada a la nueva posición. El código para posición próxima se presenta a continuación.

function *Posicion_Proxima(****var*** *LP: Tposiciones_diagrama; z: Tpunto):* ***entero****;*

inicio

k:= 1;

PosIni:= 0;

senal:=verdadero;

```
Mientras (senal y (k<= LP.tam) ) haga
   si (no (LP.lista[k].bandera)) entonces
        PosIni:= k;
        senal:= falso;
    k:=k+1;
si senal entonces
  i:= 0
sino
  i:= PosIni;
si ( (0<PosIni) y (PosIni<LP.tam) ) entonces
    Menor:= DistEuclidea(LP.lista[PosIni].punto,z);
    j:= PosIni;
    para k:=(PosIni+1) hasta LP.tam haga
      si ( no (LP.lista[k].bandera) ) entonces
          aux:= DistEuclidea(LP.lista[k].punto,z);
          si (aux<menor) entonces
              Menor:= aux;
              j:= k;
    i:= ;j
 Posicion_Proxima:= i;
Fin;
```

Código 2.4 función Posicion_proxima.

Esta función retorna el índice de la posición más cercana de la lista de posiciones al punto *z*. Si todas las banderas son verdaderas entonces la función retorna el índice cero.

2.4.4.4 Procedimiento Inter_poli_mediatriz Este procedimiento es útil para calcular los lados de la nueva región de Voronoi, como se presenta a continuación.

```
Procedimiento.Inter_Poli_Mediatriz(vor1: Tregion; p1: Tposicion; p2: Tposicion; var
x1,x2: Tpunto);
Inicio
  z1:= PuntoMedio(p1.punto,p2.punto);
  EcuacionMediatriz(p1.punto,p2.punto,a1,b1,c1);
  si (b1 <> 0) entonces
     z2.x:= (z1.x+5);
     z2.y:= (-a1*z2.x-c1)/b1;
  sino
      z2.x:= z1.x;
      z2.y:= z1.y+5;
  m:= 1;
  Para k:= 1 hasta Vor1.tam haga
     si (k<Vor1.tam) entonces
       j:= k+1
     sino
       j:= 1;
     bandera:= (leftOn(z1,z2,Vor1.poli[k]) y leftOn(z1,z2,Vor1.poli[j]))
              ó (leftOn(z2,z1,Vor1.poli[k]) y leftOn(z2,z1,Vor1.poli[j]));
     si no (bandera) entonces
        EcuacionRecta(Vor1.poli[k],vor1.poli[j],a2,b2,c2);
        ResolverSistema(a1,b1,c1,a2,b2,c2,z);
        En caso de m haga
          1:  x1:= z;
              m:=m+1;
          2:   x2:= z;
  senal:= no (izq(x1,x2,P2.punto));
  si senal entonces
      aux:= x1;
      x1:= x2;
      x2:= aux;
fin;
```

Código 2.5 Procedimiento Inter_poli_mediatriz.

El segmento x_1x_2 es una porción del bisector entre la posición de la región y el pnuevo. Este segmento divide la región R en dos piezas y es un lado de la nueva región asociada a pnuevo. Al finalizar el procedimiento, x_1 y x_2 se ordenan de tal manera que pnuevo.punto este a la izquierda del vector x_1x_2. Esto se puede apreciar en la *figura 2.20.*

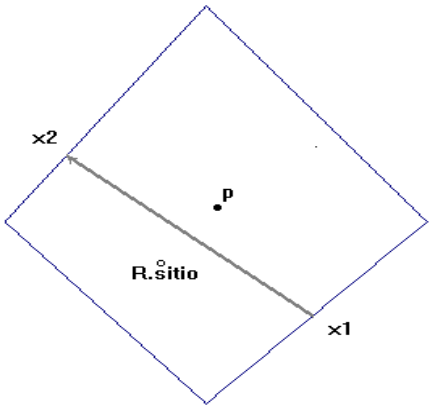


Figura 2.20 Lado de Voronoi para dos posiciones

2.4.4.5 Procedimiento Vértices_a_Borrar Después de crear la nueva región para insertar en el diagrama de Voronoi, se obtienen los vértices a borrar, es decir, aquellos que pasan a formar parte de la nueva región. A continuación se presenta el código del procedimiento Vertices_a_Borrar.

Procedimiento*.Vertices_a_Borrar(R :Tregion;* ***var*** *VB: Tvertices_diagrama);*
Inicio
VB.tam:= 0;
Asignar longitud (VB.lista,(VB.tam+1));
para *k:=1* ***hasta*** *V.tam* ***haga***
w:= V.lista[k];
senal:= verdadero;

```
    para  i:=1 hasta R.tam haga
       si (i<R.tam) entonces
         j:= i+1
       sino
         j:= 1;
          si (no (LeftOn(R.poli[i],R.poli[j],w))) entonces
         senal:= falso;
    si senal entonces
       VB.tam:=VB.tam+1;
       Asignar longitud (VB.lista,(VB.tam+1));
       VB.lista[VB.tam]:= w;
Fin;
```

Código 2.6 Procedimiento Vértices_a_Borrar.

2.4.4.6 Procedimiento Modificar_Region Si el segmento x_1x_2 es un nuevo lado de Voronoi y afecta a la región *R*, entonces se debe redefinir la región *R.* El *procedimiento Modificar_Region* realiza esto y a continuación se presenta el código.

```
Procedimiento  Modificar_Region (x1,x2: Tpunto; var R: Tregion);
Inicio
  Reg.tam:= 0;
  Asignar Longitud (reg.poli,(reg.tam+1));
  Reg.punto:= R.punto;
  senal:= Verdadero;
  Para  k:=1  hasta  R.tam  haga
     c:= R.poli[k];
     Si  (IzqSobre(x2,x1,c))  entonces
```

```
        Reg.tam := Reg.tam + 1;
        Asignar Longitud (reg.poli,(reg.tam+1));
        Reg.poli[Reg.tam]:= c;
    sino
        sino  senal  entonces
            Reg.tam := Reg.tam + 1;
            Asignar Longitud (reg.poli,(reg.tam+1));
            Reg.poli[Reg.tam]:= x2;
            Reg.tam := Reg.tam + 1;
            AsignarLongitud (reg.poli,(reg.tam+1));
            Reg.poli[Reg.tam]:= x1;
            senal:= false;
  R:=reg;
Fin;
```

Código 2.7 Procedimiento Modificar_Region.

2.4.4.7 Procedimiento Modificar_Regiones_Afectadas Después de que se ha calculado la nueva región de voronoi, se deben redefinir todas las regiones que fueron afectadas, esto se puede hacer realizando llamados al procedimiento Modificar_Region. A continuación se presenta el código:

```
Procedimiento  Modificar_Regiones_Afectadas (var vor: Tdiagrama; Rnuevo: Tregion);
Inicio
  P.lista[P.tam].bandera:= verdadero;
  Para  i:=1  hasta  (P.tam-1)  haga
     P.lista[i].bandera:= falso;
  Para  i:=1  hasta  (Rnuevo.tam)  haga
```

```
    x1:= Rnuevo.poli[i];
    Si  (i<Rnuevo.tam)  entonces
       x2:= Rnuevo.poli[(i+1)]
    sino
       x2:= Rnuevo.poli[1];
    z.punto:= PuntoMedio(x1,x2);
    j:= Posicion_Proxima(P,z.punto);
    P.lista[j].bandera:= Verdadero;
    Modificar_Region(x1,x2,Vor[j]);
Fin;
```

Código 2.8 Procedimiento Modificar_Regiones_Afectadas.

2.4.4.8 Procedimiento Agregar_Vértices_Nuevos Después de Modificar todas las regiones afectadas, se actualizan los vértices de Voronoi. Para esto se dejan aquellos vértices que no son vértices a borrar y se agregan los nuevos vértices que se calcularon. A continuación se presenta el código para actualizar los vértices de Voronoi.

```
Procedimiento  Agregar_Vértices_Nuevos (LP:Tposiciones_diagrama;
VN,VB: Tvertices_Diagrama;var LV:Tvertices_Diagrama);

Inicio
  LVaux.tam:= 0;
  Asignar Longitud(LVaux.lista,(LVaux.tam+1));
  Para  k:=1  hasta  LV.tam  haga
     w:= LV.lista[k];
     senal:=  no (EstarVB(w,VB));
     Si  senal  entonces
```

```
    LVaux.tam:= LVaux.tam + 1;
    LVaux.tam:= LVaux.tam + 1;
    Asignar Longitud(LVaux.lista,(LVaux.tam+1));
    LVaux.lista[LVaux.tam]:= w;
 Para K:=1 hasta VN.tam haga
   w:= VN.lista[K];
   LVaux.tam:= LVaux.tam + 1;
   Asignar Longitud(LVaux.lista,(LVaux.tam+1));
   LVaux.lista[LVaux.tam]:= w;
 LV:= LVaux;
Fin;
```

Código 2.9 Procedimiento Agregar_Vértices_Nuevos.

2.4.4.9 Procedimiento Agregar_Posición Ahora se tiene toda la maquinaria lista para desarrollar el código principal que permite insertar una nueva posición al diagrama de Voronoi. A continuación se presenta el código, el cual sigue las ideas descritas en la *sección 2*.3.

```
Procedimiento Agregar_Posición (pnuevo: Tposicion);
Inicio
   P.tam:= P.tam + 1;
   Asignar Longitud (P.lista,(P.tam+1));
   Asignar Longitud (Vor,(P.tam+1));
   P.lista[P.tam]:= pnuevo;
   vor[P.tam].punto:= pnuevo.punto;
   Nueva_Region (Vor,VerticesNuevos,vor[P.tam]);
   Vertices_a_Borrar (Vor[P.tam],verticesborrar);
   Modificar_Regiones_Afectadas (Vor,Vor[P.tam]);
   Agregar_Vértices_Nuevos (P,VerticesNuevos,VerticesBorrar,V);
Fin;
```

Código 2.10 Procedimiento Agregar_Posición.

En el código anterior: *P* es el arreglo de posiciones, *Vor* es el arreglo de las regiones y *V* es el arreglo de vértices. Para construir el diagrama de Voronoi para un conjunto finito de posiciones, se insertan una a una cada posición haciendo llamados al procedimiento *Agregar_Posicion*. Antes de llamar a *Agregar_Posicion* Se debe verificar que la nueva posición sea deferente de las que ya existen en el diagrama.

2.4.4.10 Procedimiento para graficar el diagrama de Voronoi Una vez se hallan insertado todas las posiciones y no se esperan mas, lo que falta por hacer es mostrar el gráfico del diagrama de Voronoi. Por eso se almacenaron las posiciones, los vértices y las regiones en arreglos. A continuación se presenta un código sencillo para mostrar el diagrama de Voronoi.

```
Procedimiento Graficar Diagrama;
Inicio
    Limpiar_Plano;
   Para  k:= 1  hasta  P.tam  haga
      Graficar Polígono  Vor[k] ;
      Graficar Punto  P.Lista[k].punto ;
   Para  k:= 1  hasta  V.tam  haga
      Graficar Punto  V.Lista[k]. ;
Fin;
```

Código 2.11 Procedimiento graficar diagrama.

2.4.5 Análisis de complejidad del algoritmo Agregar_Posición Sea $T(n)$ el tiempo de complejidad del algoritmo Agregar_Posición que opera sobre un conjunto P de n posiciones (incluyendo la nueva posición).

El peor caso que se puede presentar ocurre cuando la región asociada a la nueva posición tiene $(n-1)$ lados. Lo cual implica que se deben modificar $(n-1)$ regiones. Véase el *código 2.12* Con sus respectivos tiempos de complejidad.

Procedimiento *Agregar_Posición (pnuevo: Tposicion);* $T(n)$

Inicio

P.tam:= P.tam + 1; $O(1)$

Asignar Longitud (P.lista,(P.tam+1)); $O(1)$

Asignar Longitud (Vor,(P.tam+1)); $O(1)$

P.lista[P.tam]:= pnuevo; $O(1)$

vor[P.tam].punto:= pnuevo.punto; $O(1)$

Nueva_Region (Vor,VerticesNuevos,vor[P.tam]); $O(n^2)$

Vertices_a_Borrar (Vor[P.tam],verticesborrar); $O(n^2)$

Modificar_Regiones_Afectadas (Vor,Vor[P.tam]); $O(n^2)$

Agregar_Vértices_Nuevos (P,VerticesNuevos,VerticesBorrar,V); $O(n^2)$

Fin*;*

Código 2.12 Procedimiento Agregar_Posición con su respectivos tiempos de complejidad.

Teniendo en cuentas las consideraciones anteriores se deduce que:

$$T(n)=O(1)+O(n)+O(n^2)+O(n^2)+O(n^2)+O(n^2)$$

$$T(n)=O(n^2)+O(n)+O(1)$$

$$T(n) = O\left(n^2\right) \tag{2.3}$$

Por lo tanto el tiempo de complejidad al agregar una nueva posición al diagrama es $O(n^2)$ en el peor caso.

2.4.6 Análisis de complejidad general del algoritmo Sea $P = \{p_1, p_2, ..., p_n\}$, el diagrama se construye aplicando el procedimiento Agregar_Posición a cada sitio de P, si $T(n)$ es el tiempo de complejidad del algoritmo que calcula el Diagrama de Voronoi para todo P, entonces aplicando la *ecuación (2.3)* se tiene que:

$$T(1) = O\left(5^2\right)$$

$$T(2) = O\left(5^2\right) + O\left(6^2\right)$$

$$T(3) = O\left(5^2\right) + O\left(6^2\right) + O\left(7^2\right)$$

$$\vdots$$

$T(n) = O\left(5^2\right) + O\left(6^2\right) + O\left(7^2\right) + ... + O\left((n+4)^2\right)$ por lo tanto:

$$T(n) = \sum_{i=1}^{n} O\left((i+4)^2\right)$$

Donde $T(n) = \sum_{i=1}^{n} O\left((i+4)^2\right) = \sum_{i=1}^{n} O\left(i^2\right)$ por la regla del máximo.

$$T(n) = \sum_{i=1}^{n} O\left(i^2\right) \leq \sum_{i=1}^{n} O\left(n^2\right)$$

$$T(n) = \sum_{i=1}^{n} O\left(i^2\right) \leq \sum_{i=1}^{n} O\left(n^2\right) = nO\left(n^2\right) = O\left(n^3\right).$$

Así el tiempo de complejidad del algoritmo para el peor de los casos es de $O(n^3)$.

2.5 APLICACIONES DE LOS DIAGRAMAS DE VORONOI.

Ahora se discutirán algunas aplicaciones de los Diagramas de Voronoi, cada uno en forma diferente: Vecino más próximo, todos los vecinos más próximos, triangulación maximizando el ángulo mínimo, el círculo vacío más grande, el árbol de recubrimiento mínimo y la trayectoria del agente viajero.

Aunque existe una teoría extensa sobre envolventes convexas, en nuestro caso, esta no es utilizada en la construcción del algoritmo que calcula los Diagramas de Voronoi, por lo cual nos limitaremos a presentar únicamente el concepto considerando que será de gran ayuda en la comprensión de algunas de las aplicaciones de los diagramas.

Se define el *cierre convexo* de P como el conjunto convexo de menor área que contiene a P; la frontera de este conjunto se denomina *envolvente convexa* de P y es denotada por $Conv(P)$.

El cierre convexo puede obtenerse a partir de la intersección de semiplanos cerrados (la intersección de todos los semiplanos que contienen a P). Si P tiene n puntos, basta con a lo sumo n semiplanos. Estos semiplanos son determinados por parejas de puntos de P tales que el resto de puntos

están contenidos en el semiplano. En consecuencia el cierre convexo de P es un polígono convexo cuyos vértices son puntos del conjunto. La *figura 2.21* muestra la envolvente convexa para un número particular de puntos.

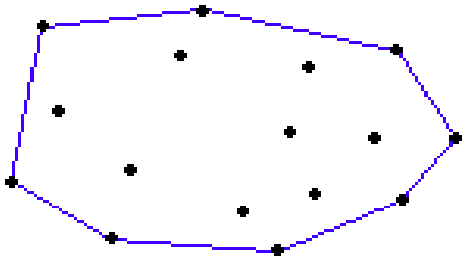


Figura 2.21 envolvente convexa

2.5.1 Vecino más próximo Partiendo del Diagrama de Voronoi de un conjunto fijo de puntos $P = \{p_1, p_{2,}, ..., p_n\}$, encontrar un vecino más próximo de un punto q, se reduce a encontrar en cual región de Voronoi cae. La posición de esta región es precisamente su vecino más próximo.

Encontrar el vecino más próximo tiene un gran número de aplicaciones en distintos campos como la Biología, Ecología, Geografía, física,…entre otros.
A continuación se presentarán algunos ejemplos particulares relacionados con el vecino más próximo.

2.5.1.1 Torres para la observación de Incendios Imagine un bosque que contiene un cierto número de torres para observar incendios. Cada guardabosque es responsable de extinguir cualquier incendio próximo a su torre (más que a cualquier otra torre). El conjunto de todos los árboles para los cuales un guardabosque particular es responsable constituye la región de Voronoi asociada a su torre. El Diagrama de Voronoi delimita las áreas de responsabilidad de los guardabosques.

2.5.1.2 Clasificador Euclídeo para el reconocimiento de formas Una técnica aplicada con frecuencia en el campo del reconocimiento de formas es hacer el mapa de un conjunto de objetos en un espacio caracterizado por reducir los objetos a vectores cuyas coordenadas son características medibles. Para ilustrar esto se plantea el siguiente ejemplo.

Se pretende discriminar dos tipos diferentes de tuercas: tipo A y tipo B. A con diámetro interno y externo de 2 y 3 cm respectivamente; B con diámetros 3 y 4 cm. Supongamos que en el deposito las tuercas se presentan sobre una cinta transportadora, donde se emplea una cámara de video, conectada mediante una tarjeta digitalizadora a un computador, que a su vez, actúa sobre un brazo mecánico encargado de almacenar las diferentes tuercas en sus respectivas cajas, como se muestra en la *figura 2.22.*

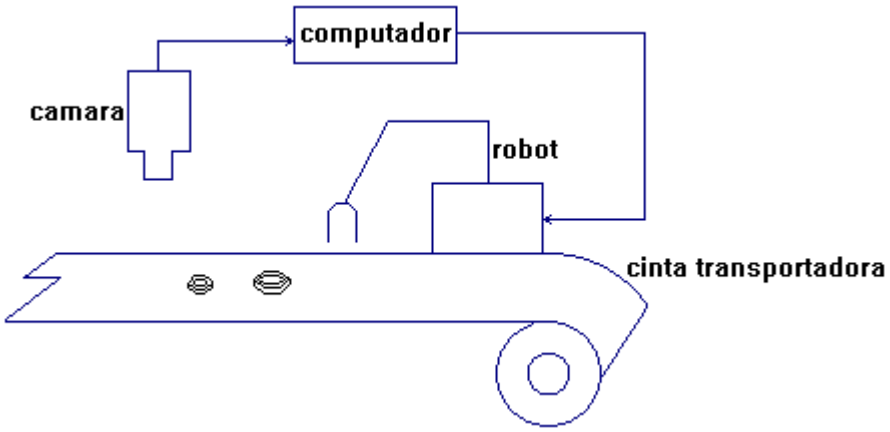


Figura 2.22 Sistema Robótico basado en visión para la selección de tuercas.

Sea x una tuerca cuyos diámetros son 2.8 y 3.7 cm. ¿Qué tipo de tuerca es x, teniendo en cuenta que sólo tuercas del tipo A y tipo B existen en el depósito?. Es más probable que sea una tuerca del tipo B porque su

distancia al prototipo B en el espacio de características es 0.36 cm; mientras que su distancia al prototipo A es 1.06 cm. En otras palabras el vecino más próximo de x es B; porque x esta en la región de Voronoi de B. Como se observa en la *figura 2.23.*

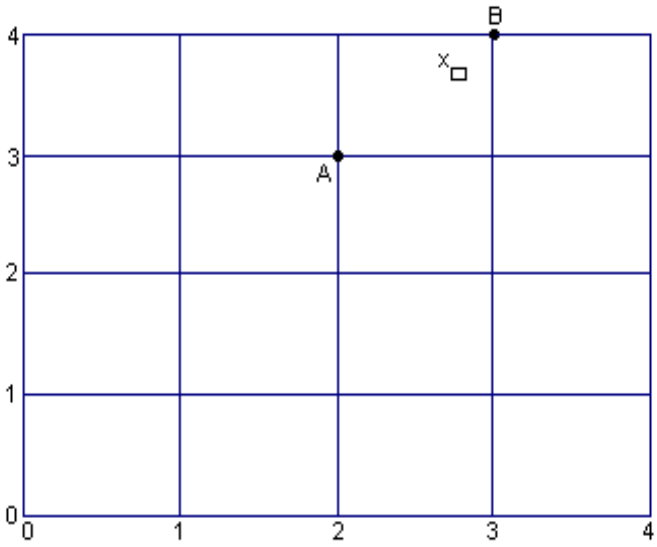


Figura 2.23 x es más cercano a B que a A

2.5.2 Todos los Vecinos más próximos El problema de todos los vecinos más próximos consiste en encontrar el vecino más próximo para cada punto de un conjunto finito dado.

La relación vecino más próximo en un conjunto finito P se define: b es el vecino más próximo de a si y sólo si $d(a,b) \leq mín_{c \neq a} d(a,c)$; donde $c \in P$. La notación $a \rightarrow b$ significa que el vecino más próximo de a es b. Si $a \rightarrow b$, no necesariamente $b \rightarrow a$. Esto se observa en la *figura 2.24*. Además, un punto puede tener varios vecinos igualmente próximos. Observe el punto d de la figura.

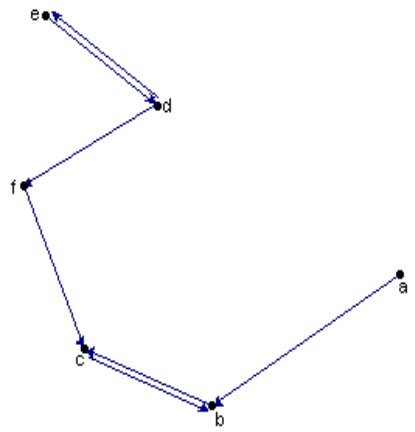


Figura 2.24 $a \to b$, pero $b \to c$; $d \to e$ y $d \to f$.

Se define el grafo de los vecinos mas próximos NNG asociando a un nodo cada punto de P . Se traza un arco entre dos punto si uno es el vecino más próximo del otro. Este grafo esta relacionado con $TD(P)$ al estar contenido, es decir $NNG(P) \subseteq TD(P)$.

2.5.3 Triangulación maximizando el ángulo mínimo Una triangulación de un conjunto finito de puntos P es un conjunto de segmentos cuyos puntos extremos están en P , se interceptan únicamente en los puntos extremos y dividen la envolvente convexa de P en triángulos.

La técnica llamada “Análisis de elementos finitos“ es utilizada, por ejemplo, para modelar los cuerpos de los automóviles. Para este propósito es mejor particionar en triángulos lo más gruesos posibles. Un camino para hacer esto preciso, es evitar los triángulos con ángulos pequeños. En este caso, la triangulación de Delaunay resulta ser óptima.

Sea T una triangulación de P y sea $\{\alpha_1, \alpha_{2,}, ..., \alpha_{3t}\}$ Su sucesión de ángulos ordenados del más pequeño al más grande; donde t es el número de triángulos de T. El número t es constante para cada conjunto P. Se puede definir una relación entre dos triangulaciones T y T' del mismo conjunto P, intentando capturar el grosor de los triángulos.

Se dice que T es más gruesa que T', denotada por $T \geq T'$ si la sucesión de ángulos de T es de manera lexicográfica más grande que la sucesión de ángulos de T'. La importancia de la triangulación de Delaunay se debe a que $TD(P) \geq T'$ para cualquier otra triangulación T' de P.

2.5.4 Círculo vacío más grande El problema de encontrar el círculo vacío más grande de un conjunto P de n posiciones consiste en encontrar un círculo cuyo centro se encuentre en la envolvente convexa de P, que éste no contenga posiciones en su interior y que sea el de mayor radio posible.

Sea q el centro del círculo vacío más grande, $f(q)$ el radio del círculo y $Conv(P)$ la envolvente convexa de P. Existen aparentemente un número infinito de puntos q candidatos para este máximo pero ahora se argumentará que solamente existe una lista finita de verdaderos candidatos para el máximo de f.

2.5.4.1 Centros interiores a la envolvente Supongamos que p es estrictamente interior a $Conv(P)$. Imaginémonos inflando un círculo desde p, el radio de $f(p)$ se obtiene cuando el círculo choca con una posición de P.

Si el radio $f(p)$ incluye solamente una posición s_1, entonces $f(p)$ no puede ser un máximo; porque si p es movido a lo largo de la semirrecta $s_1 p$ lejos de s_1 hacia p' entonces $f(p')$ es mayor, como se aprecia en la *figura 2.25*.

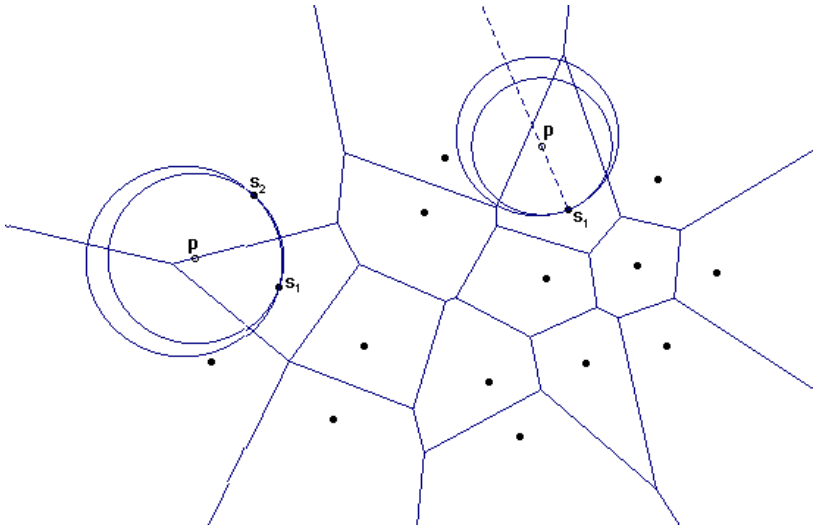


Figura 2.25 Centro en el interior, círculo a través de un punto (arriba) y a través de dos puntos (abajo).

Por lo tanto p no es un máximo de f porque existe un punto p' donde f es mayor (la existencia de p' se garantiza porque p es estrictamente interior en la envolvente).

Supongamos que el radio $f(p)$ incluye dos posiciones p_1 y p_2. Si p es movido hacia p' a través de bisector $B(p_1, p_2)$ entonces $f(p')$ es nuevamente mayor como se muestra en la *figura 2.26.* Solamente cuando el círculo incluye tres posiciones se puede tener un máximo de f. El movimiento de p en cualquier otra posición resulta al moverse hacia alguna posición cercana, resultando así que el radio disminuye. De todo lo anterior se tiene la siguiente proposición:

2.5.4.2 Proposición Si el centro p del círculo vacío más grande es estrictamente interior a la envolvente $Conv(P)$, entonces p coincide con un vértice de Voronoi.

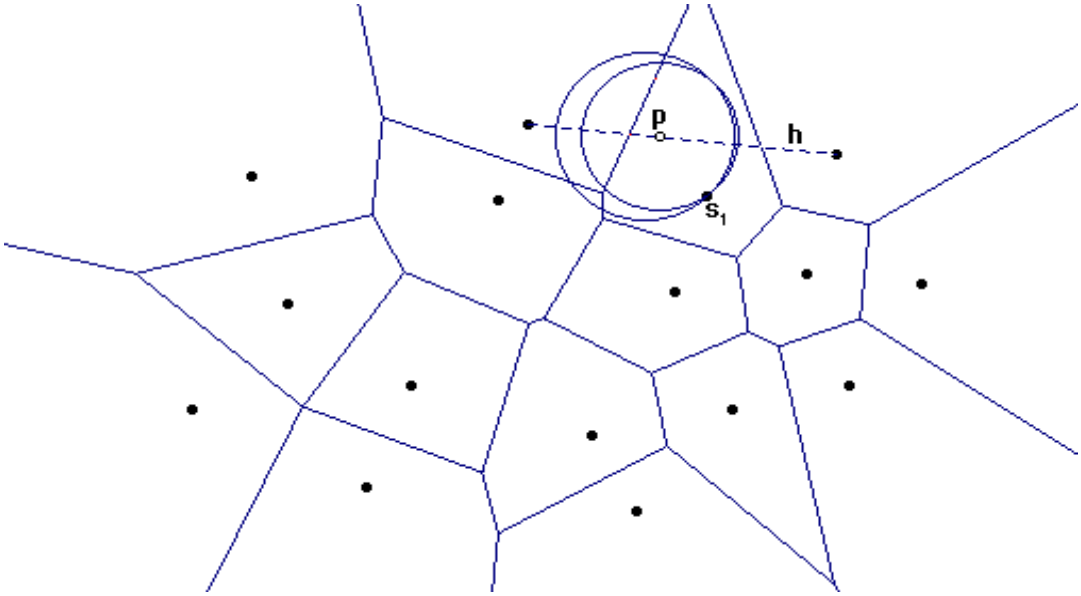


Figura 2.26 Centro sobre el lado h de Voronoi, círculo a través de una posición.

La proposición anterior establece que los vértices de Voronoi son buenos candidatos para ser el centro del círculo vacío más grande.

2.5.4.3 Centros sobre la envolvente Supongamos que p se encuentra directamente sobre la envolvente convexa de P.

Si el círculo vacío más grande incluye justamente una posición p_1. Primero no es posible que p sea un vértice de la envolvente porque esto implicaría que $f(p)=0$. Luego p esta sobre el interior de un lado h de

$Conv(P)$. Moviendo a p en una u otra dirección de h entonces $f(p)$ aumenta, por lo tanto $f(p)$ no puede ser máximo.

Si suponemos que el círculo centrado en p contiene dos posiciones p_1 y p_2, para que $f(p)$ aumente debe suceder que p se mueva a lo largo del bisector $B_{1,2}$ hacia el exterior de la envolvente. Así puede estar bien que $f(p)$ sea un máximo local. De esta manera se ha probado la siguiente proposición.

2.5.4.4 Proposición Si el centro p del círculo vacío más grande se encuentra sobre la envolvente $Conv(P)$, entonces p debe encontrarse sobre un lado de Voronoi.

2.5.4.5 Algoritmo para calcular el círculo vacío más grande Hasta el momento se ha encontrado una lista finita de puntos que son centros potenciales del círculo vacío más grande: los vértices de Voronoi y las intersecciones entre los lados de Voronoi y la envolvente convexa de las posiciones. Como se indica en el *algoritmo 2.3* que se presenta a continuación debido a Toussaint en 1983.

Círculo vacío más grande
Inicio
Calcular $Vor(P)$
Calcular $Conv(P)$
Para *cada vértice de Voronoi* v ***haga***

Si $v \in Conv(P)$ ***entonces***

Calcular el radio de círculo centrado en v *y actualizar el máximo.*

Para *cada lado de Voronoi* e ***haga***

Calcular $q = e \cap Conv(P)$

Calcular el radio de círculo centrado en q *y actualizar el máximo.*

Retornar el centro de radio máximo.

Fin*;*

Algoritmo 2.3

2.5.5 Árbol de recubrimiento mínimo Sea $P = \{p_1, p_2, \cdots, p_n\}$ un conjunto de puntos en el plano. El árbol de recubrimiento mínimo de P, denotado por $MST(P)$, se define como el árbol de longitud mínima que conecta todos los puntos de P. Un ejemplo es mostrado en la *figura 2.27*.

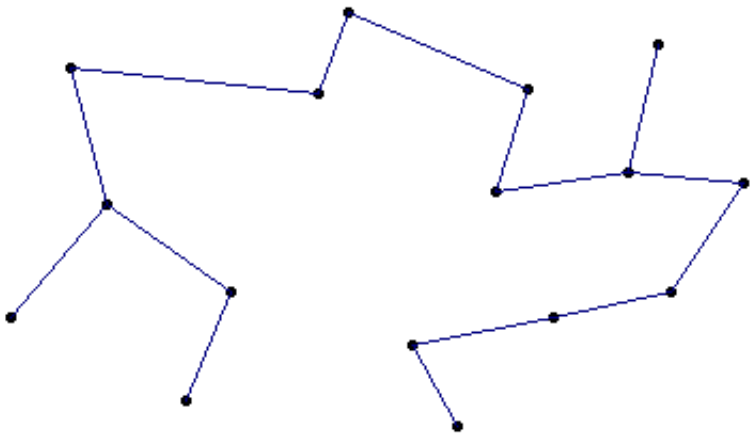


Figura 2.27 Árbol de recubrimiento mínimo euclidiano

2.5.5.1 Teorema Si P es un conjunto finito de puntos en el plano, entonces $MST(P) \subseteq TD(P)$.

Prueba: Supongamos que $ab \in MST(P)$ y que $ab \notin TD(P)$. Por el *teorema 2.2.4,* ningún círculo a través de a y b puede ser vacío. En particular el círculo con diámetro ab contiene por lo menos una posición $c \in P$ distinta de a y b.

Por otro lado, $d(a,c) < d(a,b)$ y $d(b,c) < d(a,b)$ como se aprecia en la *figura 2.27* Al quitar ab del $MST(P)$ se obtienen dos subárboles. Denotemos por T_a el árbol que contiene a a y T_b el árbol que contiene a b. Sin perdida de generalidad, supongamos que $c \in T_a$, si consideramos el árbol $T' = T_a + bc + T_b$. Entonces T' es un árbol más corto que conecta todos los puntos de P. Esto es una contradicción; Porque $MST(P)$ es el árbol más corto que conecta todos los puntos de P. Luego, no es posible que $ab \notin TD(P)$.

2.5.5.2 Algoritmo de Kruskal El algoritmo de Kruskal es útil para calcular el MST de un conjunto finito de puntos.

Sea G un grafo cuyos nodos son los puntos del conjunto P. El algoritmo de Kruskal utiliza una estrategia voraz basada en la intuición de que el $MST(P)$ debe contener los lados más cortos de G. Esto sugiere que el $MST(P)$ puede construirse incrementalmente por la adición del lado más corto que aun no ha sido explorado y que mantenga la propiedad acíclica del árbol.

Sea T el árbol construido incrementalmente. Las ideas anteriores nos conducen al siguiente algoritmo.

Algoritmo de Kruskal

Inicio

Ordenar los lados de G *según la longitud de menor a mayor:* $G = \{e_1, e_2, e_3, \cdots\}$

Inicializar T *en vacío*

i:= 1 ;

Mientras *(* $T \neq MST(P)$ *)* ***haga***

si *(* $T \cup \{e_i\}$ *es acíclico)* ***entonces***

T := $T \cup \{e_i\}$

i:= i+1;

fin*;*

Algoritmo 2.4 Algoritmo de Kruskal.

Para calcular el $MST(P)$, es suficiente con aplicar el *algoritmo 2.2* para $G = TD(P)$. Esto puede ser observado en el *algoritmo 2.3*.

Algoritmo Calcular $MST(P)$

Inicio

Calcular $Vor(P)$*;*

Calcular $TD(P)$*;*

Aplicar algoritmo de Kruskal sobre $TD(P)$*;*

Retornar $MST(P)$

Fin*;*

Algoritmo 2.5 Algoritmo que calcula el árbol de recubrimiento mínimo.

2.5.6 Trayectoria del Agente Viajero El problema del agente viajero consiste en: Dado un conjunto finito P de puntos, encontrar la trayectoria

cerrada más corta que visite cada punto de P, dicha trayectoria se denomina Trayectoria del agente Viajero y se denota por $TSP(P)$; en otras palabras, consiste en un agente de ventas que tiene que visitar n ciudades, comenzando y terminando en una misma ciudad, visitando solamente una vez cada ciudad, excepto el origen y realizar el recorrido mínimo, este recorrido puede estar en función del tiempo o de la distancia, es decir, recorrer el mínimo de kilómetros o realizar el menor tiempo posible.

El problema del agente viajero se puede modelar fácilmente como un grafo completo dirigido, en donde los vértices del grafo son las ciudades y las aristas son los caminos, dichas aristas deben tener un peso, y este peso representa la distancia que hay entre dos vértices que están conectados por medio de dicha arista.

Este problema es uno de los más estudiados por la ciencia computacional por su gran significado práctico; muchos otros problemas pueden ser reducidos a este, desafortunadamente se clasifica como *NP duro,* es decir que no se conoce un algoritmo polinomial que lo resuelva y tiene pocas posibilidades de ser encontrado, por tal razón se han buscado algoritmos heurísticos y aproximados efectivos. Uno de estos algoritmos está basado en la Triangulación de Delaunay mediante el árbol de expansión mínima.

Suponiendo que las posiciones de P son ciudades que un vendedor debe visitar antes de regresar a casa, para encontrar el $TSP(P)$, primero se calcula el árbol de expansión mínima $MST(P)$ y se sigue la ruta de ida y regreso como se ilustra en la *figura 2.28* . Es claro que $TSP(P)$ tiene exactamente dos veces la longitud del $MST(P)$.

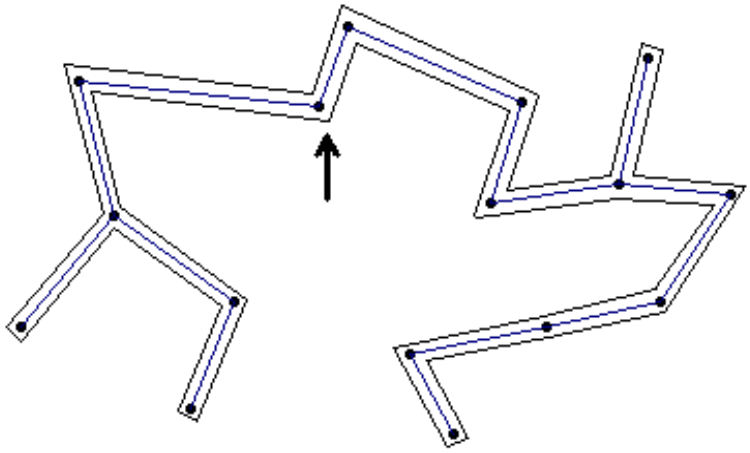


Figura 2.28 Recorrido formado siguiendo el MST

2.5.6.1 Proposición Sea P un conjunto finito de puntos. Si M es la longitud del $MST(P)$ y T es la longitud de cualquier $TSP(P)$ *entonces* $2M < 2T$.

Prueba: Sea T_1 la longitud del TSP removiendo un lado entonces $T_1 < T$. Como M es la longitud del $MST(P)$ entonces se tendría $M \leq T_1$ y de aquí $M < T$. Por lo tanto $2M < 2T$.

Si T es la longitud del TSP optimo, se tiene que $M < T < 2M < 2T$; por la proposición anterior. Luego $M < T < 2M$. Esto quiere decir que la longitud del doble recorrido del $MST(P)$ es una cota superior de la longitud del TSP optimo de P . Partiendo del $MST(P)$ se puede mejorar la trayectoria del agente viajero realizando la siguiente consideración: No

visitar dos veces la misma posición. Esto se puede observar en la *figura 2.29.*

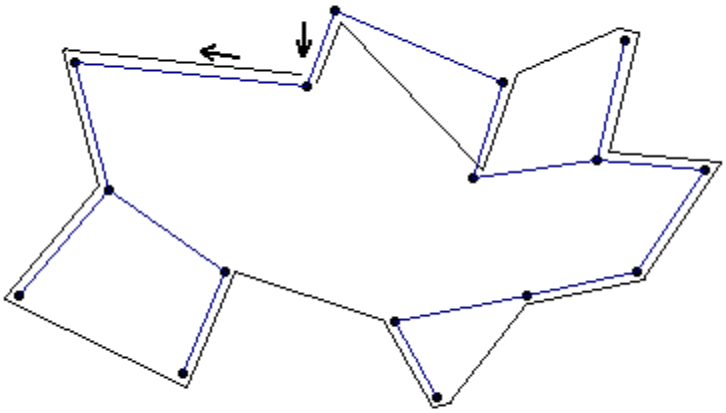


Figura 2.29 Recorrido acortado del doble MST de la *figura 2.28*

Note que no todo vértice de Voronoi está en el interior de la envolvente convexa. Por eso, es importante verificar si $v \in Conv(P)$ en el algoritmo.

2.5.6.2 Ejemplo Suponga que se desea ubicar una nueva tienda en un área donde ya existen varias tiendas rivales. Si la densidad de población es uniforme, ¿Dónde se debe ubicar la nueva tienda de tal manera que las ventas sean máximas?

Una idea natural para resolver este problema es ubicar la nueva tienda lo más lejos posible de las tiendas antiguas. Mas precisamente se debe escoger un sitio cuya distancia a la tienda más cercana sea lo más grande posible.

Este problema se puede resolver (matemáticamente) encontrando el círculo vacío más grande cuyo interior no contenga otras tiendas. El centro de este círculo es precisamente el mejor lugar (geográfico) para ubicar la nueva tienda.

3. CONCLUSIONES

1. El análisis e interpretación de la teoría manejada en este documento nos permitió conocer la importancia que tiene la geometría en general y específicamente la geometría computacional en la solución de problemas sencillos y de un alto grado de dificultad, utilizando construcciones geométricas que pueden resultar complicadas al carecer de una ayuda computacional.

2. Se implementaron dos códigos en Delphi: Uno que calcula la Triangulación de polígonos y otro que calcula el diagrama de Voronoi para una nube de puntos. Para esto se utilizaron conceptos básicos de la geometría como: Polígono, Área, Diagonal, Bisector, entre otros, vistos desde una perspectiva computacional.

3. Después de realizar un análisis minucioso de las aplicaciones se logró deducir que el lazo de unión entre las diversas disciplinas y la Geometría Computacional, hacen que esta sea práctica, útil e importante en los distintos campos del conocimiento.

4. RECOMENDACIONES

Los Diagramas de Voronoi tienen muchas aplicaciones, pero son poco conocidas. Se recomienda hacer estudios detallados de las Aplicaciones de los Diagramas de Voronoi que conlleven a la implementación computacional.

Analizar e implementar otros métodos que conlleven a dar solución a los problemas estudiados y a otros muchos que se presentan en la Geometría Computacional.

Estudiar los algoritmos propuestos y realizar implementaciones con estructuras dinámicas de datos como: listas enlazadas, pilas, colas, entre otras.

BIBLIOGRAFÍA

BRASSARD. G./ BRATLEY. P. Fundamentos de Algoritmia. Madrid 1997.

CORMEN. Thomas H./ LEISERSON. Charles E./ RIVEST. Ronald L. Algorithmics. New York 1994

EDWARDS. C.H./ PENNEY. David E. Calculo con Geometría Analítica. Mexico 1994.

GRIMALDI. Ralph P. Matemáticas Discreta y Combinatoria. Mexico D:F. 1998.

O'ROURKE. Joseph Computacional Geometry in C. New York.1994 .

ANEXOS

Anexo B. Manual de usuario del programa de Triangular

INSERTAR VERTICES DEL POLÍGONO El programa esta diseñado para que el usuario inserte los vértices manualmente con el Mouse. Para insertar los vértices debe presionar el boton ***insertar vértices***. Inmediatamente aparecerá el mensaje “Inserte los vértices en sentido antihorario”, esto es necesario para que el algoritmo de triangulación funcione eficientemente, de lo contrario el programa puede cometer errores. Presione el boton ***aceptar***, inmediatamente aparecerá un recuadro blanco sobre el cual el usuario puede insertar los vértices.

Observación: Si el usuario al insertar los vértices comete errores puede ir a ***nuevo polígono*** para volver a introducir los vértices deseados.

GRAFICAR EL POLÍGONO Una vez se hayan insertado los vértices presione el boton ***graficar***, seguidamente se conectaran los puntos en el orden en que fueron insertados, conectando el último con el primero para de esta manera cerrar la región del polígono.

CALCULAR ÁREA Al pulsar el boton ***Calcular área***, aparecerá el área correspondiente al polígono en unidades cuadradas (UC) en la parte superior izquierda de la ventana.

TRIANGULACIÓN Teniendo en cuenta que el objetivo principal del programa es triangular un polígono cualquiera, a presionar el boton ***Triangular*** aparecerá el polígono triangulado con sus respectivas diagonales en color rojo.

NUEVO POLÍGONO Si el usuario desea triangular un nuevo polígono, lo podrá hacer presionando el boton ***Nuevo polígono*****,** siguiendo cada uno de los pasos mencionados anteriormente.

Anexo A. Manual de usuario del programa Diagramas de Voronoi

INSERTAR POSICIONES DE VORONOI Este programa esta diseñado para que el usuario inserte las posiciones del Diagrama manualmente con el Mouse.

Para iniciar la construcción del Diagrama de Voronoi el usuario debe dar click sobre el boton ***insertar***, inmediatamente aparecerá un recuadro blanco en el cual debe insertar las posiciones. Cada vez que se inserta una nueva posición, el Diagrama de Voronoi se modifica, mostrando la estructura correspondiente a dichos puntos con la particularidad que los lados de la nueva región son de color verde para diferenciarla de las otras regiones.

NUEVO DIAGRAMA Para iniciar la construcción de un nuevo diagrama se debe presionar el boton ***Nuevo diagrama*** e iniciar de nuevo el procedimiento descrito anteriormente.