TRIANGULACIÓN DE POLÍGONOS

RESUMEN

Una de las principales herramientas en geometría Computacional la constituyen las triangulaciones, tanto de una nube de puntos como la de polígonos. En este artículo veremos los temas más destacados de estas estructuras.

PALABRAS CLAVES:
Polígono, geometría, vértices, nubes, segmento triangulación.

ORLANDO RODRÍGUEZ BUITRAGO
Especialista
Matemáticas Computacional.
Profesor Asistente.
Universidad del Cauca.
orodriguez@unicauca.edu.co

GUILLERMO SOLARTE MARTINEZ
Ingeniero de Sistemas
Profesor Auxiliar
Universidad Tecnológica de Pereira.
roberto@utp.edu.co

ABSTRACT

One of the main tools of the Geometry Computing is triangulations both for a cloud of points and for polygons. This article will deal with the most outstanding subjects of such structures.

KEYWORDS: Polygonos, geometry, cloud, triangulation

1. TRIANGULACIÓN DE POLÍGONOS

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.

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 con el uso del 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 $e_0 = V_0V_1$, $e_1 = V_1V_2$,.... $e_{n-1} = V_{n-1}V_0$, n segmentos conectando los puntos. Se dice que estos segmentos limitan el polígono si y sólo si se satisfacen las siguientes condiciones.

1. Para todo i= 0,1,...n-1 se tiene que:
   $e_i \cap e_{i+1} = V_{i+1}$
2. Para todo $j \neq i+1$ se tiene que:
   $e_i \cap e_j = \varnothing$

a. Los puntos $V_0,V_1,V_2....Vn-1$ que aparecen en la definición se llamarán vértices del polígono y los segmentos $e_0,e_1,...,e_{n-1}$ se llamarán lados ó aristas del polígono.

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

c. En este documento los vértices se conectarán en sentido antihorario, más adelante se observará la importancia de esta convención.

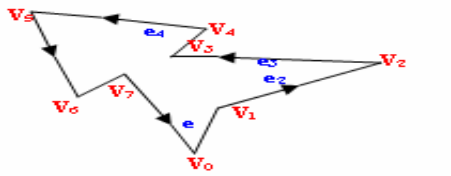


Figura 1. Polígono con vértices orientados en sentido antihorario.

Definición (visibilidad) Sean x, y dos puntos arbitrarios de un polígono P. Se dice que x puede ver a y si y solo si el segmento cerrado xy ésta contenido en P, es decir $xy \subseteq P$.
Se dice que x puede ver claramente a y, si y solo si, $xy \subseteq P \wedge xy \cap Fr(P) \subseteq \{ x, y \}$; donde
Fr (P) de nota la frontera de P.

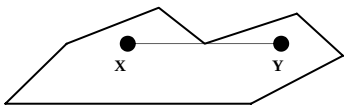


Figura 2. Bloqueo de una línea de visión

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 si.

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.
Existencia de una diagonal. La clave para demostrar la existencia de la triangulación es probar la existencia de

Fecha de Recepción: 01 Agosto de 2006
Fecha de Aceptación: 08 Febrero de 2007

una diagonal. A continuación se precisan algunos conceptos necesarios para probar la existencia de una diagonal.

Definición de vértices convexos y reflejo. Un vértice es llamado reflejo (ó cóncavo), si su ángulo interno es estrictamente mayor que π, si el vértice no es reflejo, se dice que es convexo.

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

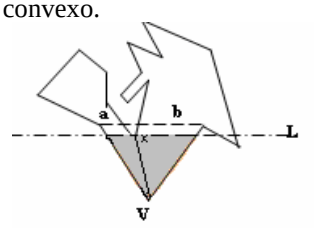


Figura 3. vx puede ser una diagonal

Proposición (triangulación). Todo polígono P de n vértices puede ser dividido en triángulos por adición de (cero o más) diagonales

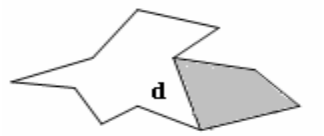


Figura 4. Una diagonal divide al polígono en dos subpolígonos

En general existen varias formas de triangular un polígono dado. Todas ellas tienen el mismo número de diagonales y/o de triángulos, esto se formaliza en el siguiente Proposición.

Proposición (número de diagonales) Toda triangulación de un polígono P, de n vértices, utiliza n-3 diagonales y consiste de n-2 triángulos.
Corolario (suma de ángulos) la suma de ángulos internos de un polígono de n vértices es (n-2)π

Definición. El dual de un triángulo de un polígono es un grafo donde cada nodo se asocia a un triángulo, hay un arco entre dos nodos si y solo si sus triángulos comparten una diagonal.

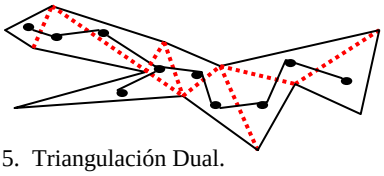


Figura 5. Triangulación Dual.

Definiciones

a) El dual T de una triangulación es un árbol cuyos nodos son de grado a lo más tres
b) Un árbol T es un grafo conectado sin ciclos
c) Los nodos de grado uno son hojas de T
d) Los nodos de grado tres son puntos de ramificación
e) Los nodos de grado dos están situados sobre la trayectoria del árbol.

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.

Proposición (las dos orejas de Meister) Todo polígono de n>=4 vértices tiene al menos dos orejas no superpuestas

2. IMPLEMENTACION

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

El siguiente código muestra una forma de definir un tipo punto en Phython

```
from record import record
clases Tpunto(record)
x = 0
y = 0
```

Representación de un polígono. Un polígono se puede representar por arreglos o por listas en lazadas. En este caso se opto por arreglos dinámicos debido a la claridad del código.
Ejemplo

```
from record import record
class  Tlista_vertice (record):

Tlista_vertices = Tpunto[ ]
Tpoligono=" "
Poli:tlista_vertice;
Tam=0
```

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).

Proposición (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:

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

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*, donde $q= v_0$

```
def Area2(a, b, c):
  Area = a.x * b. y-a.y * b.x+a.y*c.x-a.x*c.y+b.x*c.y-
          cx*by;
return Area

def  AreaPoli2( p, n) :
   sum = 0
   i = 1
while  i: <   Ptam-2:
  sum = sum+Area2(P.poli[0], P.poli[i], P.poli[i+1])
  i=i+1
  return  sum
```

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

Intersección de segmentos

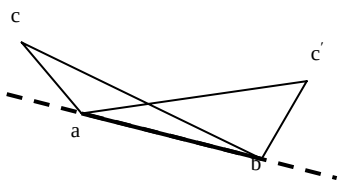
Diagonales. El paso clave para triangular un polígono es encontrar una diagonal (una línea recta entre dos vértices claramente visible).

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

Lema. El segmento $s = v_i v_j$ es una diagonal de P si y solo si:

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

Función izquierda. Se utiliza para decidir si dos segmentos se interceptan y establece si un punto esta 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 esta 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.



En la Figura 6. C esta a la izquierda de ab si y solo si el triángulo abc tiene área positiva; También abc′ tiene área positiva.

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

```
def  izquierda(a, b, c):
   if  Area ( a, b ,c ) > 0:
       Izq = true
   else:
     Izq = false
 return  Izq
```

La función izquierda también puede ser implementada encontrado 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.

Cuándo el triangulo 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 la función izqSobre para determinar si c está a la izquierda o sobre la recta que contiene al vector ab.

```
def  izqsobre(a, b, c) :
   if Area2(a ,b ,c ) >= 0 :
       izqsob = true
   else
       izqsob = false
 return izqsob


def  Colineal(a, b, c):
       if Area2(a,, b, c) = = 0:
         Coli = true
       else
          Coli = false
return Coli
```

Observación, en la rutinas izquierda, izqsobre y Colineal se compararon dos veces el área en lugar del área misma, esto con el fin de trabajar en el dominio de los enteros.

Intersección Booleana Sean L_1 la recta que contiene el 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 L_1 y d esta al otro lado) y además a y b son partidos por L_2. A continuación se muestra el código para intersección propia.

```
def  intersectProp(a, b, c, d):
    if Colineal (a, b, c) and /or Colineal (a ,b ,d) and /or
      Colineal(c, d, a) and /or  Colineal(c, d, b) :
      interProp = flase
    else:
      interProp=((( izquierda(a,b,c)  or izquierad(a,b,d))
      and  (izquierda(c,d,a) or izquierad(c,d,b)))

return interProp
```

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

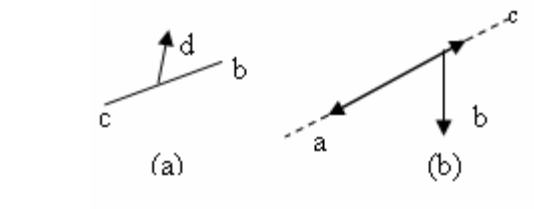


Figura 7. (a) Intersección Impropia entre Segmentos.
(b) La Colinealiadad no es suficiente.

Función entre. ¿Que se necesita para decidir si c esta entre a y b? Si el segmento ab no es vertical y c es colineal ab, c sobre ab si y solo 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 esta entre a y b, esto conduce al siguiente código.

```
def Entre(a, b, c):
   if   not (Colineal (a, b, c)) :
            entr = flase
  else:
       if  (a[x] != b[x]):
       entr = (a.x<=c.x) y(c.x<=b.x) and / or
              (a.x>=c.x)y(c.x>=b.x)
       else:
         entr = (a.y<=c.y) y(c.y<=b.y) and / or
         (a.y>=c.y) y(c.y>=b.y):
return entr
```

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

```
def  intersec(a, b, c, d):
   if intersecPro(a,b,c,d) :
     Intersec= verdadero
   else:
    if Entre(a,b,c) and/or Entre(a,b,d)and/or Entre(c,d,a)
           and/or Entre(c,d,b):
           inter = true
       else:
          inter = false
return  inter
```

Implementación de la triangulación de las diagonales internas o externas.
Si se ignoran las distinciones 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 solo si para todo lado e de P que no incide con los extremos de s se tiene que e $\cap$ s = $\varnothing$. Esto se observara claramente en el siguiente código

```
def  diagExtint(i , j, p):
    bandera = true
    for i in range(Ptam-1)):
          if  bandera = = true :
              Kk1= ((k+1) % ptam)
          else((k = i) y/o (k1 = i) and/or (k1 = j)):
                if intersect(P.poli[i], P.poli[j], P.poli[k],
                   P.poli[k1]) :
                   bandera = false
return  bandera;
```

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 ninguno lado de P. Si es interior a P en una vecindad de v_j, entonces s es interior a P a lo largo de su longitud. De esta manera, verifica 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 8. (a) s es interno a P si y solo si s esta 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.

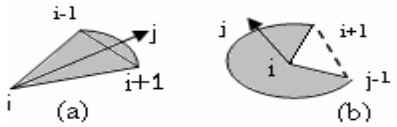


Figura 8 La diagonal s= ij esta en el cono determinando por v_{i-1}, v_i v_{i+1}: figura 8 (a) convexo y la figura 8. (b) reflejo. En (b) ambos v_{i-1} y v_{i+1} esta a la derecha de ij.

Caso 2 supongamos que v_i es reflejo como se muestra en ala figura 8. (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 solo si s no esta 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 solo si v_{i+1} esta a la izquierda o sobre v_{i-1} v_j , recuerde que si v_{i-v} , 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.

```
def  intCono(i, j, P) :
    I1= (i+1) % ptam
    In1=(i+n-1)%ptam
    if izqsobre((P.poli[in1], P.poli[i], P.poli[i1]):
      IntCo= izquierda((P.poli[i], P.poli[j],P.poli[in])
      and izquierda((P.poli[j], P.poli[i], P.poli[in1])
      return IntCo
    else:
      IntCo= not (izquierda((P.poli[i], P.poli[j],
      P.poli[in])  y izquierda((P.poli[j], P.poli[i],
      P.poli[in1]))
      return IntCo
```

La función intCono retorna verdadero si y solo si la diagonal v_iv_j es estrictamente interna al polígono P, en una vecindad de v_i. La función diagonal v_i v_j es una diagonal interna de P, si y solo si, DiagExtint(i,j,P) son ambas verdaderas.

```
def diagonal(i, j, P) :
return (intCono(i,j,P) y DiagExtint(i,j,P))
```

Obsérvese que en el código anterior se llama primero a la función intCono, y luego la función diagExtinit). Así si intCono es falsa, no se verifica DiagExitint.
La función diagonal retorna verdadero si y solo si v_iv_j es una diagonal interna de P.

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. Esta idea se pueden ver en el siguiente algoritmo de triangulación

```
if n > 3:
 for diagonal oreja potencial range(i,i+2)
      if diagonal(i,i+2,n,P) :
           print  diagonal
           Cortar oreja en v_i
Aplicar recursividad al resto del polígono
```

Algoritmo Triangulación.

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

```
def SuprimOreja(i,P):
     for k range(i,ptam- 2):
          P.poli[k] = P.poli[k+1]
```

Trazado de diagonales. En el algoritmo después de encontrar una diagonal, es necesario que esta se trace, para esto, las coordenadas correspondientes a cada diagonal son almacenadas en un arreglo de diagonales. Otras posibles alternativas es almacenar a los índices de los vértices. Ahora se definirá un tipo arreglo de diagonales.

```
from record import record
clases TDiagonal (record)
V1: Tpunto;
V2: Tpunto;

Tarreglo_diagonales = Arreglo de Tdiagonal[ ]
Tdiagonales_Triangulacion = " "
Lista = Tarreglo_diagonales[]
Tam = 0
```

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

```
def Triangular( p,l, j):
    if ptam >3:
      for i range(0,ptam-1):
          i1:=(i+1)% ptam-1
          i2:=(i+2)% ptam
          if diagonal(i,i2,P):
             tempo.v1=Ppoli[i]
             tempo.v2=Ppoli[i2]
              Ltam =j+1
             SuprimOreja(i1,P)
             Ptam = ptam-1
             Tringular(p,l,j)
```

Análisis de complejidad del algoritmo de triangulación. Sean T(n) el tiempo de complejidad del algoritmo de triangulación operando sobre un polígono P de n vértices.
El peor de los caso que se puede presentar en el algoritmo, sucede cuando todas las diagonales potenciales resultan ser diagonales orejas, 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) ]
if n >3:
for diagonaloreja potencial rangoo (i, i+2):  [O(n) ]
  if diagonal(i,i+2n,P) :                     [O(n) ]
     print  diagonal                                [O(1) ]
     Cortar oreja en V_i                            [O(n) ]
Aplicar  recursividad al resto del polígono   [T(n-1) ]
```

De todo lo anterior se obtiene el algoritmo con los respectivos tiempos de complejidad

$T(n)= O(n) \times O(n) + O(1)+o(n) +T(n-1)$
$T(n)= O(n^2) + O(1) + O(n) +T(n-1)$; 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:

$T(n)= O(n^2) +T(n-1)$
$T(n)= O(n^2) + O((n-1)^2) +T(n-2)$
$T(n)= O(n^2) + O((n-1)^{2)} + O((n-2)^2)+o(n) + T(n3)$

$T(n)= O(n^2) + O((n-1)^{2)} + O((n-2)^2)+.....++T(3)$
Por lo tanto

$T(n)= \sum O(k^2)= O((2n^3 + 3n^2+n)/6) = +O(n^3)$
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 s equivalente a:

$a_{n+1 =} a_{n+n}{}^2$, $a_3= 1$. n>=3. Ecuación 2.

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 del forma $a_n^{(p)}= A_2n^3 + A_1n^2 + A_0n$. Sustituyendo este resultado en la ecuación 2.

Se obtiene $A_2 (n+1)^3 + A_1(n+1)^2 + A_0(n+1) = A_2 n^3 + A_1n^2 + A_0 n+n^2$.

Al compara los coeficientes de las coordenadas 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 sistemas de ecuaciones
$A_2 = 1/3$. $A_1= -1/2$ y $A_0 = -1/6$.
Por lo tanto:

$a_n^{(p)} =1/3n^3 - 1/2n^2 – 1/6n$. y la solución general de la relación de recurrencia es:

$a_n = a_n{}^{(h)} + a_n^{(p)}= c+1/3n^3 - 1/2n^2 – 1/6n$ como $a_3 =1$, entonces:
$1 = c + (1/3)\, 3^3 - (1/2)2^2 – (1/6)3$ y $c = -3$.
De todo lo anterior se tiene que $T(n)= O((1/3)\, n^3 - (1/2)n^2 – (1/6)n-3) = O(n^3)$.

3. CONCLUSIONES

Las posibles aplicaciones de este tema incluyen la vigilancia con cámaras de seguridad fijas o con robots ambulantes, la iluminación óptima de espacios, “rendering” de imágenes, posicionamiento de antenas para aplicaciones inalámbricas, etc. Interpolar funciones en una nube de puntos. Se emplea en la cartografía para obtener un modelo aproximado del terreno, a partir de las coordenadas espaciales de unos cuantos miles de puntos

4. BIBLIOGRAFÍA

[1] A.Okabe, et al. "Spatial Tessellations, Concepts and Applications of Voronoi Diagrams", John Wiley & Sons, 1992
[2] R. Klein "Concrete and Abstract Voronoi Diagrams" Springer-Verlag, J. O'Rourke: "*Computational Geometry in C*", Cambridge Univ. Press, 1994.
[3] M. de Berg y otros: "*Computational Geometry: Algorithms and Applications*", Springer, 1997.
[4] Nilson j, Wilson A, Nelcy R. Tesis de grado análisis de algoritmo geometría computacional.
[5] www-history.mcs.st-andrews.ac.uk/ history / Mathematicians/Delone.html
[6] wwwpi6.fernuni-hagen.de/Geometrie-Labor / VoroGlide/