

GEOMETRÍA COMPUTACIONAL: EL PROBLEMA DEL CERCO CONVEXO

Recepción: Noviembre de 2005 / Aceptación: Diciembre de 2005

⁽¹⁾ Edgar Ruiz Lizama
⁽²⁾ Eduardo Raffo Lecca

RESUMEN

El artículo tiene por objetivo presentar los conceptos fundamentales de la geometría computacional y mostrar una solución al problema del cerco convexo, utilizando el algoritmo de Graham. Para la implementación de la solución, los autores usan el lenguaje de programación Java.

Palabras Clave: Geometría computacional, diagramas de Voronoi, planeamiento de movimiento, problema del cerco convexo, algoritmo de Graham.

COMPUTATIONAL GEOMETRY: THE CONVEX HULL PROBLEM ABSTRACT

The objective of this work is to show fundamental concepts of computational geometry and one solution for the convex hull problem, using the Graham algorithm. The authors use the Java language programming in order to implement the solution.

Key words: Computational geometry, Voronoi diagrams, motion planning, convex hull problem, Graham algorithm.

INTRODUCCIÓN

La geometría computacional es una rama de la ciencia de la computación y emergió en la década de 1970. Estudia los algoritmos que resuelven problemas geométricos. Su aplicación, va desde el diseño de los VLSI, hasta el CAD (*Computer-Aided Design*), el CAM, computación gráfica y GIS (sistemas de información geográfica)[1].

Imagine los teléfonos públicos en una urbanización y como contar con un mapa que muestre las regiones en donde se encuentre el teléfono más cercano. Una distribución posible se presenta en la Figura 1.

A la subdivisión de un área o localidad en regiones, se conoce como diagrama de Voronoi; y para hacer esto se hace uso de algoritmos geométricos.

Sea un conjunto de obstáculos geométricos (como los mostrados en la Figura 2), y asuma que se desea encontrar la conexión más corta entre dos puntos geométricos, salvando obstáculos. Este problema que es crucial en robótica se conoce como planeamiento del movimiento (*motion planning*), y el cual hace uso de algoritmos geométricos.

Un clásico problema en CG (del inglés *Computational Geometry*) es el cerco convexo o CH (*Convex Hull*); este crea un subconjunto S de un plano denominado convexo si y solo si, cualquier par de puntos $p, q \in S$; es decir el segmento pq está contenido completamente en S (ver Figura 3).

El Cerco convexo $CH(S)$ (se lee CH de S) es el más pequeño conjunto convexo en S .

Veamos el siguiente ejemplo:

entrada : $p_1, p_2, p_3, p_4, p_5, p_6$
salida : p_4, p_5, p_6, p_2 (ver Figura 4)

MÉTODOS GEOMÉTRICOS ELEMENTALES

Las aplicaciones en geometría computacional actúan sobre objetos geométricos simples: punto, línea y polígono [2].

Puntos. Es el objeto fundamental, y es considerado como un par de enteros: las coordenadas habituales en el plano cartesiano.

(1) Ingeniero Industrial. Profesor del Departamento de Ingeniería de Sistemas e Informática, UNMSM.
E-mail: eruilz@unmsm.edu.pe
(2) Ingeniero Industrial. Profesor del Departamento de Ingeniería de Sistemas e Informática, UNMSM.
E-mail: eraffol@unmsm.edu.pe

>>> Geometría computacional: el problema del cerco convexo

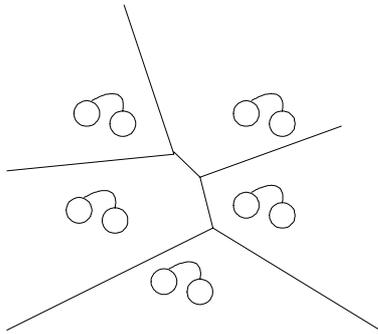


Figura 1. Área dividida en regiones

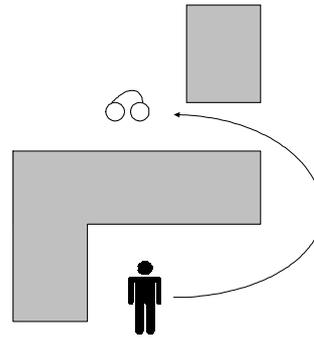


Figura 2. Esquema del planeamiento del movimiento

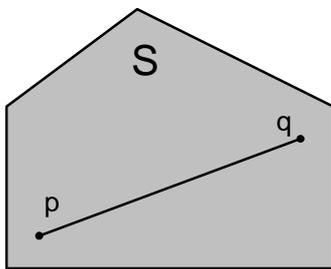


Figura 3. El problema del cerco convexo

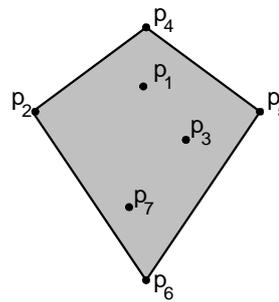


Figura 4. Un cerco convexo

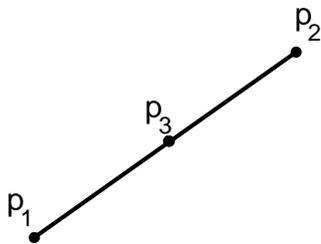


Figura 5. Una línea

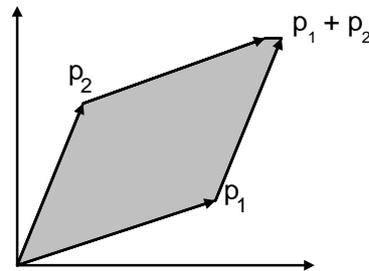


Figura 6. Vectores y el producto cruz

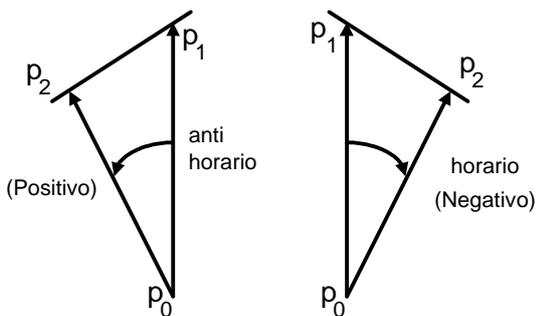


Figura 7. Sentido horario $p_1 \times p_2$

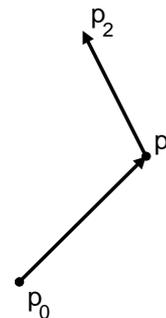


Figura 8. Producto cruzado negativo

Línea. Una línea es un par de puntos, que se supone están unidos por un segmento de línea recta.

Para especificar las propiedades de un segmento de línea recta, se define la combinación convexa de $p_1 = (x_1, y_1)$ y $p_2 = (x_2, y_2)$, como un punto $p_3 = (x_3, y_3)$; existiendo un α , tal que $0 \leq \alpha \leq 1$, y $p_3 = \alpha p_1 + (1-\alpha) p_2$, siendo p_3 un punto que está contenido en el segmento de línea $p_1 p_2$ (ver figura 5).

$$x_3 = \alpha x_1 + (1-\alpha) x_2$$

$$y_3 = \alpha y_1 + (1-\alpha) y_2$$

Considere los vectores p_1 y p_2 ; su producto cruzado $p_1 \times p_2$ se interpreta como el área del paralelogramo formado por $(0,0)$, p_1 , p_2 y $p_1 + p_2$ (ver figura 6); y se define como:

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix}$$

$$= x_1 y_2 - x_2 y_1$$

Si p_1 y p_2 es positivo, entonces p_1 es sentido horario de p_2 en caso que el producto cruzado sea cero, se dice que son colineales.

El problema de determinar si dos segmentos consecutivos se encuentran a la izquierda o a la derecha: $p_0 p_1$ y $p_1 p_2$ con respecto a p_1 ; equivale a encontrar el ángulo entre $p_0 p_1 p_2$. Utilizando el producto cruzado se tiene que

$$(p_2 - p_0) \times (p_1 - p_0)$$

En caso que el producto cruzado sea negativo $p_0 p_2$ es *antihorario* con respecto a $p_0 p_1$ y se encuentra a la izquierda p_2 . Un producto cruzado positivo es tomado como sentido horario con respecto a $p_0 p_1$ y p_2 es a la derecha. Al respecto ver la Figura 8.

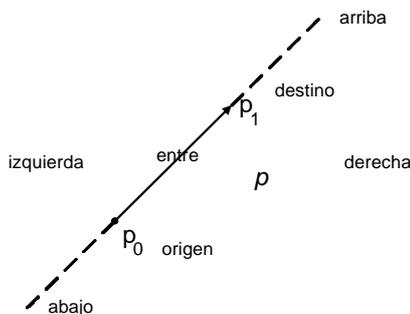


Figura 9. Clasificación de P

Se puede clasificar el punto P con respecto a una región.

Así puede un punto P, encontrarse: a la izquierda, derecha, origen, destino, entre los dos puntos, arriba o abajo (véase la Figura 9).

SEGMENTOS DE LÍNEA

Un segmento de línea $\overline{p_0 p_1}$, consiste de los puntos finales p_0 y p_1 (véase Figura 10).

Si p_0 es el origen de un segmento de línea y p_1 es el destino, entonces es un segmento de línea dirigido $p_0 p_1$; denominado arco o borde (edge). Un arco posee dos puntos finales: origen y destino; por defecto el punto p_0 es $(0,0)$ y destino es $(1,0)$.

La rotación de un arco es mover a 90° en sentido horario los puntos finales (véase la Figura 11).

donde:

$$m = 0.5 * (p_1 - p_0)$$

$$p = p_1 - p_0$$

y como

$$n \cdot (p_1 - p_0) = 0, \text{ con } p_1 - p_0 = (x,y)$$

$$n = p(y,-x)$$

$$\text{origen} = m - 0.5 * n$$

$$\text{destino} = m + 0.5 * n$$

Se observa que una doble notación o twice es la rotación de una rotación (Ver figura 12)

Para determinar el punto en t, se tiene que este es igual a $p_0 + t * (p_1 - p_0)$

Para ínter seccionar un objeto arco con otro denominado f, se tiene que:

$$P(t) - C \text{ coincide con } \overline{cd}$$

$$n \cdot (P(t) - c) = 0$$

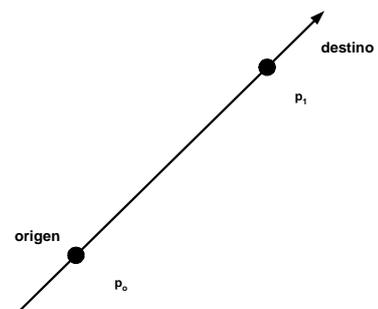


Figura 10. Segmento de línea $\overline{p_0 p_1}$

>>> Geometría computacional: el problema del cerco convexo

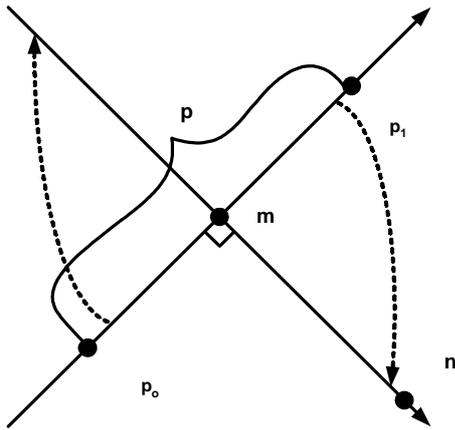


Figura 11. Rotación o arco

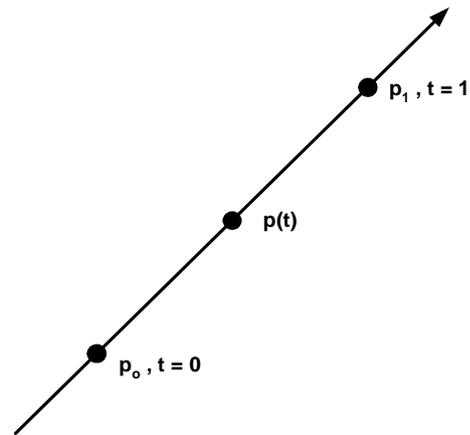


Figura 12. Rotación de rotación

$$\begin{aligned} n \cdot ((a + t(b-a)) - c) &= 0 \\ n \cdot (a - c) + t n \cdot (b - a) &= 0 \end{aligned}$$

$$t = -\frac{n \cdot (a - c)}{n \cdot (b - a)}$$

Si $n \cdot (b - a)$ es cero; se tiene que $b - a$ y $d - c$ son perpendiculares al vector n . Al clasificar el arco f , si este es LEFT o RIGHT retorna PARALELO; en otro caso COLINEAL.

El valor de t es $\frac{n \cdot (a - c)}{n \cdot (b - a)}$

que indica que hay intersección, retornando INTERSECCION (véase la Figura 13).

Para evaluar la pendiente (o slope, en inglés), la con-

dición es $p_0.x$ sea diferente a $p_1.x$ (porque en otro caso es $\frac{0}{0}$):

$$\text{pendiente} = \frac{p_1.y - p_0.y}{p_1.x - p_0.x}$$

El valor de la ordenada y en base a la abscisa x (según la figura 14), es:

$$y = \text{pendiente} * (x - p_0.x) + p_0.y$$

EL PROBLEMA DEL CERCO CONVEXO

Un *polígono convexo*[5], es un polígono con la propiedad de que cualquier segmento de recta cuyos extremos estén en el polígono cae por completo dentro

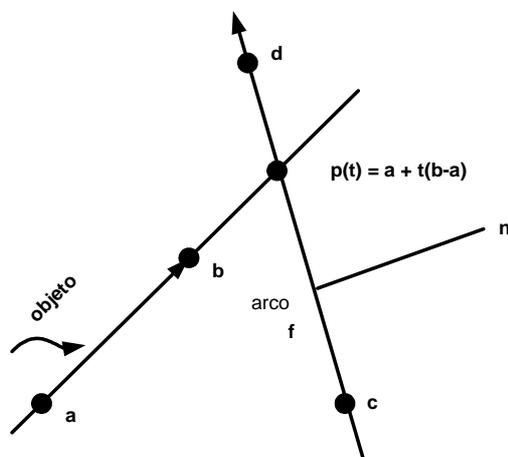


Figura 13. Intersección

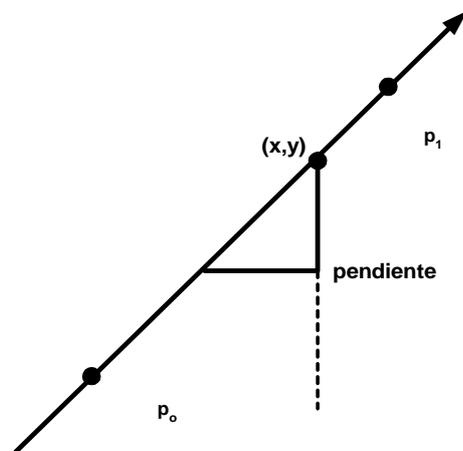


Figura 14. El concepto de pendiente de una recta

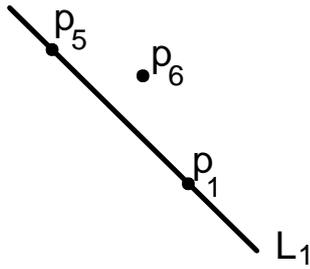


Figura 15. p_6 no forma parte del cerco

del polígono. El problema del cerco convexo consiste en encontrar el área más pequeña de un polígono convexo que encierre un conjunto de puntos en el plano. Como se vio anteriormente, las figuras 3 y 4 presentan un cerco convexo, así también la Figura 16.

Sea un conjunto S en el plano; y P un punto en S . Se dice que P es un punto del cerco, si existe una línea L a través de P , tal que todos los puntos en S excepto P , se encuentran a un lado de L (excepto P , ninguno pertenece a la línea).

En la Figura 15, p_6 no es un punto del cerco por encontrarse a un lado de L_1 .

El cerco convexo de un conjunto finito de puntos S , consiste de puntos del cerco ordenados que se encuentran en el borde de S . Para la Figura 16 el cerco convexo es la secuencia de los puntos p_1, p_2, p_3, p_4, p_5 .

El cerco convexo de un conjunto finito de puntos S en el plano, es la secuencia p_1, p_2, \dots, p_n de puntos de cercos de S en el siguiente orden:

1. El punto p_1 es el punto con la coordenada mínima y (p_1 es un punto de cerco).

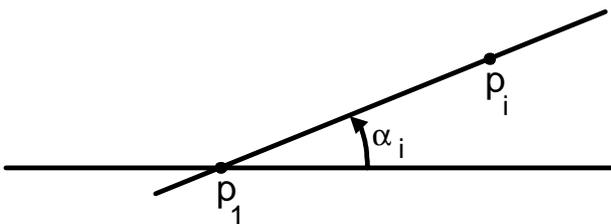


Figura 17. p_6 no forma parte del cerco

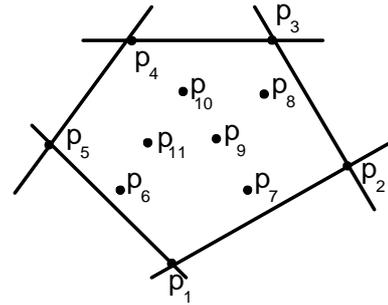


Figura 16. Un cerco convexo

2. Para $i \geq 2$, sea α_i el ángulo desde la horizontal al segmento de línea p_1, p_i . Los puntos p_2, p_3, \dots, p_n están ordenados tal que $\alpha_1, \alpha_2, \dots, \alpha_n$ están en una secuencia ordenada. (Ver figura 17)

En la Figura 18; p_1 tiene la mínima coordenada y ; entonces este es el primer punto listado en el cerco convexo. Como se muestra, los ángulos desde la horizontal a las líneas segmentos $p_1, p_2; p_1, p_3; p_1, p_4; p_1, p_5$ crecen.

ALGORITMO DE GRAHAM

En el algoritmo de Graham (R.L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, *Information Processing letters* 1, 132-133; 1972) los puntos son ordenados de acuerdo al ángulo tomando un punto inicial. La secuencia o pasos del algoritmo se presentan a continuación.

- a. Primero encontrar el punto p_1 con mínima coordenada y ; en el caso que varios puntos tengan el mismo valor, tomar el del mínimo valor de x .
- b. Ordenar todos puntos de P de S de acuerdo al ángulo desde la horizontal al segmento de línea p_1, p_i .

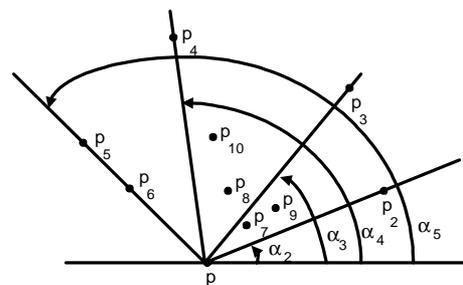


Figura 18. Un cerco convexo

>>> Geometría computacional: el problema del cerco convexo

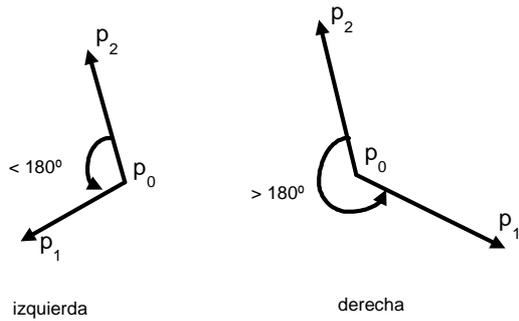


Figura 19. Comparación de ángulos

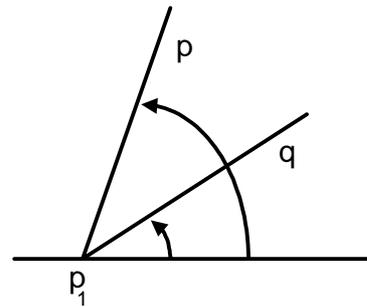


Figura 20. Comparando: $p > q$

c. Finalmente examinar los puntos en orden y descartar aquellos que no están en el cerco convexo.

Para implementar la estrategia usada en el algoritmo, se desarrolla un camino para comparar ángulos.

El producto cruzado determina cuando un punto no se encuentra en cerco convexo y deberá ser descartado.

Suponga que visita p_1, p_0 y p_2 en ese orden y los segmentos de línea p_0, p_2 y p_0, p_1 tienen ángulos menores que 180° entonces p_0 se mueve a la izquierda, en otro caso se mueve a la derecha.

El producto cruz de p_0, p_1, p_2 es:

$$\text{cross}(p_0, p_1, p_2) = (y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0)$$

esto, implica que se mueve a la izquierda si $\text{cross}(p_0, p_1, p_2) < 0$.

De igual manera p_0, p_1, p_2 se mueve a la derecha si $\text{cross}(p_0, p_1, p_2) > 0$. Cuando $\text{cross}(p_0, p_1, p_2) = 0$, entonces los puntos p_0, p_1, p_2 son colineales.

El algoritmo Graham toma sistemáticamente los puntos $p \neq q$ en el ordenamiento y compara p y q . Si el producto $\text{cross}(p_1, p, q)$ es < 0 entonces p, p_1, q se mueven a la izquierda; es decir $p > q$. Al respecto examine el seudo código de la Figura 21.

Sea el conjunto S ordenado de acuerdo a sus ángulos (véase Figura 22). El algoritmo examina p_1, p_3, p_4 , moviéndose hacia la izquierda, luego p_2 es retenido. Después se examina p_2, p_3, p_4 , reteniéndose p_3 (ver Figura 23).

```
Graham ( p, n, k )
  if ( n = 1 ) then
    k = 1
    retorno
  min = 1
  for i = 2 to n
    if ( p[ i ].y < p[ min ].y ) then
      min = i
  for i = 1 to n
    if ( p[ i ].y < p[ min ].y ) and ( p[ i ].x < p[ min ].x )
      then min = i
  Intercambiar ( p[ i ], p[ min ] )
  Ordenar p2, ..., pn de acuerdo al ángulo
  p[ 0 ] = p[ n ]
  k = 2
  for i = 3 to n
    while ( cross ( p[ k ], p[ k-1 ], p[ i ] ) > 0 )
      k = k - 1
      k = k + 1
    Intercambiar ( p[ i ], p[ k ] )
  End. Graham
```

Figura 21. Seudo código para el algoritmo de Graham

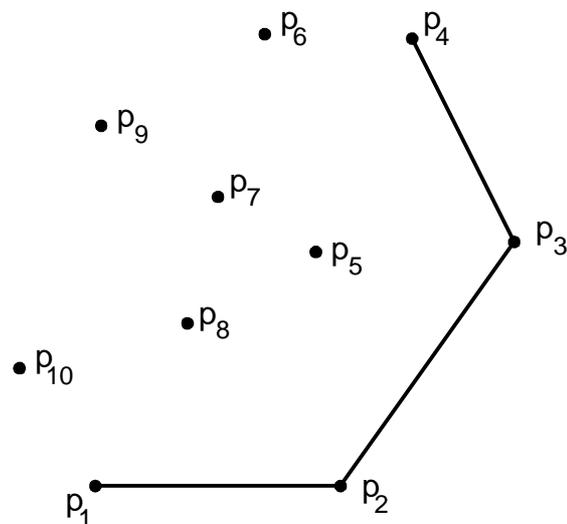


Figura 22. Conjunto S de puntos

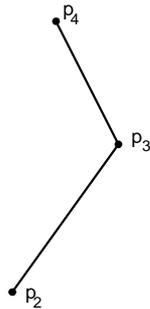


Figura 23. Encontrando p_2, p_3, p_4

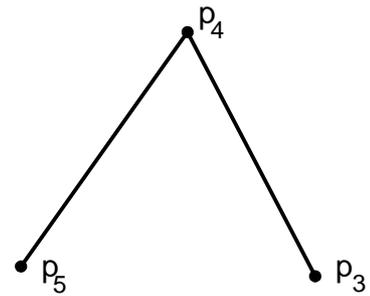


Figura 24. Examinando p_3, p_4, p_5

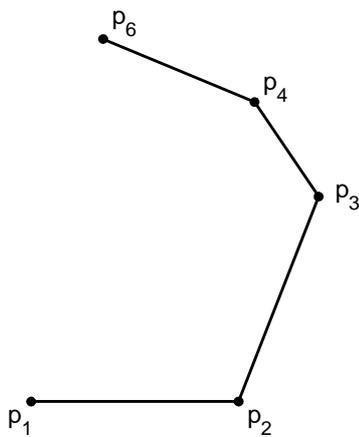


Figura 25. Obteniendo p_6

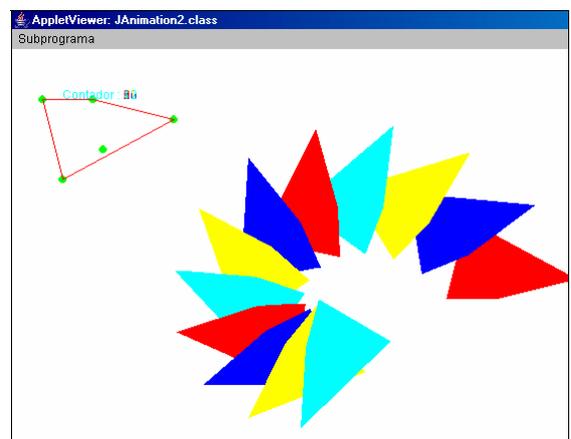


Figura 26. Salida para Graham.java

```

/**
 * Autor :E.Raffo Lecca – E. Ruiz Lizama
 * Fecha :1/05/2004
 */
import Point2D;

public class Graham {
    private int n;
    private int k;

    private Point2D puntos[]; // puntos en el conjunto S
    public static Point2D[] hull;

    public Graham(int m)
    {
        puntos=new Point2D[m+1];
        puntos[m]=new Point2D();
    }
    public Graham()
    {
        this(10); // por defecto
    }
}

```

```

public void add(Point2D p)
{
    if(n<puntos.length-1)
        puntos[n++]=p;
}

private void swapTo(Point2D p,Point2D q)
{
    Point2D temp=new Point2D(p);
    p.assign(q);
    q.assign(temp);
}

public void convexHull()
{
    int i,j;
    Point2D p;
    if(n==1)
        return;
    // encontrar el punto inicial
    int min=1;
    for(i=1;i<n;i++)
        if(puntos[i].getY()<puntos[min].getY())

```

Figura 27a. Código del archivo7 Graham.java

>>> Geometría computacional: el problema del cerco convexo

```

        min=i;
        for(i=0;i<n;i++)
            if(puntos[i].getY()<puntos[min].getY()
                &&
                puntos[i].getX()<puntos[min].getX())
                    min=i;
        // intercambiar
        swapTo(puntos[0],puntos[min]);
        // ordenar
        for(i=0;i<n-1;i++)
            for(j=i+1;j<n;j++){
                p=Point2D.resta(puntos[i],puntos[0]);
                double angi=p.polar();

                p=Point2D.resta(puntos[j],puntos[0]);
                double angj=p.polar();

                if(angj<angi)
                    swapTo(puntos[i],puntos[j]);
            }
        // iteracion
        puntos[n].assign(puntos[0]);
        k=1;
        for(i=2;i<=n;i++){
            while(Point2D.cross(Point2D.resta(puntos[k],puntos[k-
1]),
                Point2D.resta(puntos[i],puntos[k-
1]))<0){
                k--;
            }
            k++;
            swapTo(puntos[i],puntos[k]);
        }
        // leer Puntos
        public Point2D[] getPuntos()
        {
            return puntos;
        }
        // hull: la solucion
        public Point2D[] hull()
        {
            int i;
            Point2D[] hull=new
            Point2D[k];
            for(i=0;i<k;i++)
                h u l l [ i ] = n e w
            Point2D(puntos[i]);
            return hull;
        }
    }

```

Figura 276. Código del archivo7 Graham.java

En la siguiente iteración examina p_3, p_4, p_5 y se retiene p_4 (ver figura 24).

Al examinar p_4, p_5, p_6 se descarta p_5 y se regresa a p_3, p_4, p_6 , se retiene p_4 .

Al examinar p_4, p_6, p_7 se retiene p_6 . En la siguiente se examinan p_6, p_7, p_8 se retiene p_7 . En p_7, p_8, p_9 se descarta p_8 . El algoritmo regresa a p_6, p_7, p_9 y se descarta p_7 . El algoritmo regresa a p_4, p_6, p_9 y se retiene p_6 .

A continuación se presenta el código JAVA para el archivo *Graham.java* (Figura 27: a y b). La animación desde un applet, se presenta en la Figura 26. Los archivos: *Point2D.java*, *JgrahamShow.java*, *JgrahamShow.html*, se encuentran en [3].

CONCLUSIONES

El artículo presenta los conceptos fundamentales de la geometría computacional y una solución a uno de los problemas clásicos en este campo: el cerco convexo.

La solución es implementada en un applet de Java,

el mismo que aprovecha la interfase grafica para presentar una animación de un polígono (cerco convexo) en la salida estándar.

BIBLIOGRAFÍA

1. Laszlo, M. J. (1996). *Computational Geometry and Computer Graphics in C++*. Prentice-Hall, Inc, U.S.A.
2. Preparata, F. P.; Michael Ian Shamos. (1988). *Computational Geometry An Introduction*. Springer-Verlag. U.S.A.
3. Raffo Lecca, E. (2005). *Curso: Mi Primer Java*. En: <http://industrial.unmsm.edu.pe/org/apya/daisi/eraffol/website/index.htm>
4. Cormen, Thomas; Leiserson, Charles; Rivest, Ronald, and Stein, Clifford. (2001). *Introduction To Algorithms*. Second Edition. McGraw-Hill Book Company and MIT Press. U.S.A.
5. Weiss, Mark A. (2002). *Data Structures and Problem Solving Using Java*. Second Edition. Adisson-Wesley. U.S.A.