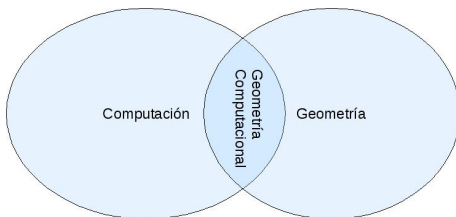


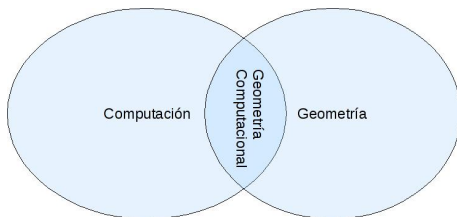
# Geometría Computacional. Envolvente convexa e intersección de líneas

Jose Luis Bravo Trinidad

# Introducción a la Geometría computacional



# Introducción a la Geometría computacional



- CAD/CAM
- GIS
- Robótica
- Visión artificial
- Simulación y optimización
- Animación por ordenador
- Videojuegos
- Etc

# Geometría Computacional y Diseño

Los programas de diseño/fabricación por computador (CAD/CAM) son herramientas esenciales para el diseño que están sustentadas por métodos de Geometría Computacional. Ejemplos de problemas con los que se pueden encontrar:

- Estudiar las restricciones de la fabricación sobre la geometría de la figura <sup>1</sup>
- Analizar los diseños
- Utilizar la geometría como base de los diseños <sup>2</sup>

---

<sup>1</sup>The Metal Casting

<sup>2</sup>The Very Many

## Restricciones de los procesos de fabricación

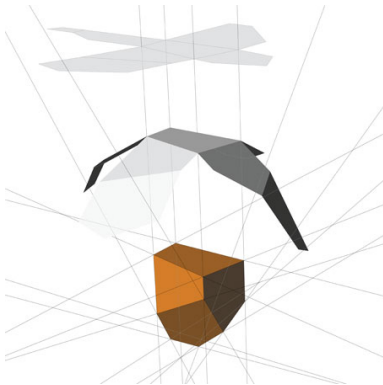


Figura: Piezas de hormigón diseñadas por ordenador<sup>3</sup>

<sup>3</sup>De <http://www.dezeen.com/>

## Análisis y optimización de diseños

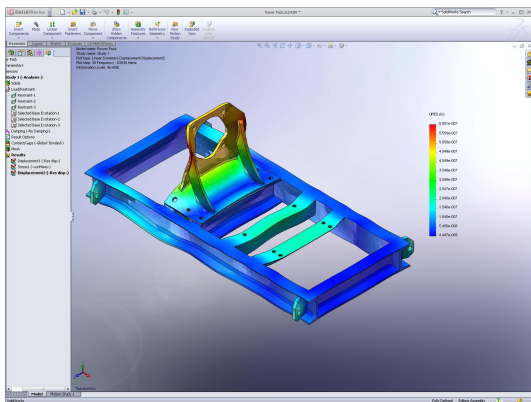


Figura: Análisis de las tensiones sobre una pieza<sup>4</sup>

<sup>4</sup>De <http://www.solidworks.com/>

## Geometría como inspiración para el diseño

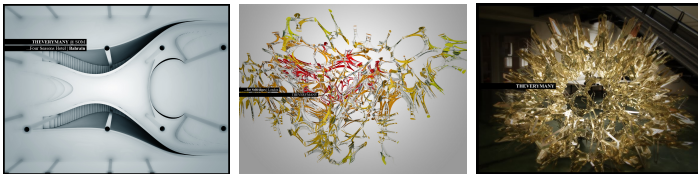


Figura: Conceptos de Geometría Computacional aplicados al diseño<sup>5</sup>

<sup>5</sup>De <http://www.theverymany.net/>

# Geometría Computacional y Topografía

Los Sistemas de Información Geográfica (GIS) son el equivalente topográfico a los CAD/CAM del diseño. Algunos problemas que se suelen encontrar:

- Calcular las intersecciones de varios contornos determinados por las características.
- Generar de mallados a partir de datos.
- Calcular funciones sobre las superficies (cantidad de luz, sombras, volumen de presas/taludes, ...)



# Sistemas de Información Geográfica

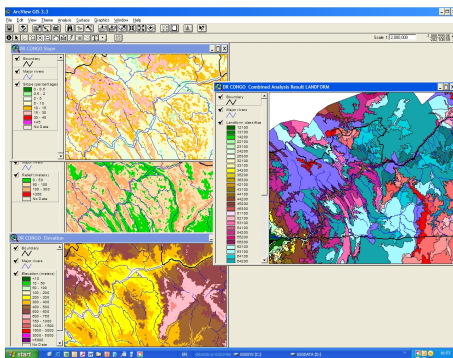


Figura: Ejemplo de datos proporcionados por un GIS<sup>6</sup>

<sup>6</sup>De ArcView

# Geometría Computacional e Informática

Geometría Computacional se considera como parte de las Ciencias de la Computación. Otras áreas que requieren la Geometría Computacional son:

- Gráficos por ordenador
- Visión artificial
- Robótica

## Gráficos por ordenador

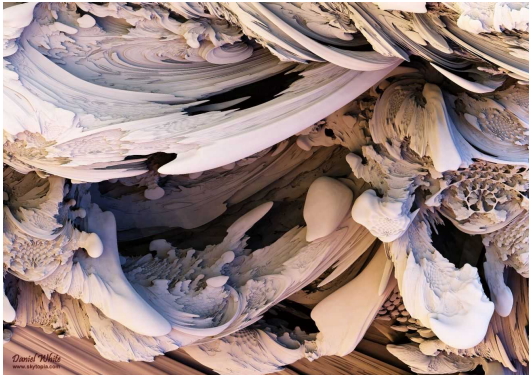


Figura: Gráfico generado por ordenador<sup>7</sup>

<sup>7</sup>De <http://www.skytopia.com/project/fractal/mandelbulb.html>

# Visión artificial

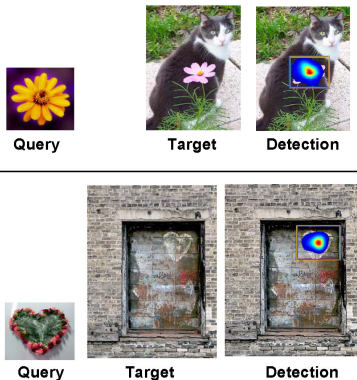


Figura: Reconocimiento de patrones en una imagen<sup>8</sup>

<sup>8</sup>De <http://users.soe.ucsc.edu/~milanfar/>

# Robótica

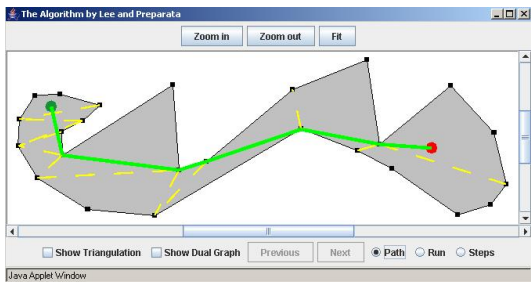


Figura: Camino óptimo entre dos puntos

# Geometría Computacional y Telemática

Ejemplos de problemas específicos:

- Optimizar la distribución de recursos (antenas, servidores, etc).
- Determinar la región de cobertura de una antena.

## Optimización de la distribución de recursos

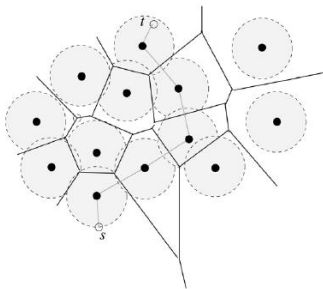


Figura: Optimización de la cobertura de una red inalámbrica<sup>9</sup>

<sup>9</sup>De Li et al, "Coverage in Wireless AdHoc Networks", IEEE Transactions on Computers, 52-6, 2003, 1-11

## Alcance de una señal

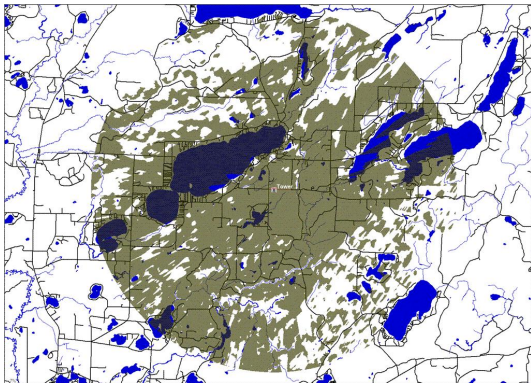


Figura: Cobertura de una torre de telecomunicaciones<sup>10</sup>

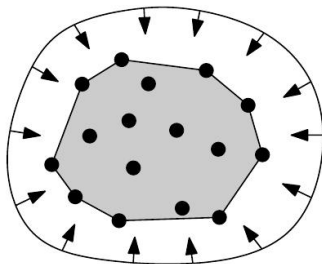
<sup>10</sup>De <http://www.sonicnet.us/>



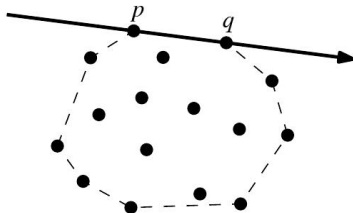
# Envolvente convexa

## Definición

*Se denomina envolvente convexa de una serie de puntos al menor convexo que los contiene.*



## Cálculo de la envolvente convexa



Un procedimiento posible es el siguiente:

- 1 Consideramos todas las parejas de puntos
- 2 Nos quedamos con aquellas tales que “a la izquierda” no quede ningún punto
- 3 Las ordenamos en el sentido de las agujas del reloj, comenzando por una arbitraria

## Complejidad algorítmica

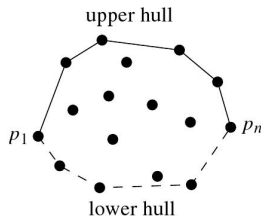
### Definición

*Se denomina complejidad algorítmica al coste de un algoritmo en función de un parámetro (por ejemplo, el número de puntos en la envolvente convexa).*

Una manera aproximada de obtener la complejidad es considerar la operación que más veces se repite.

En el algoritmo anterior es “mirar la posición de cada punto en relación a (cada una) de las parejas de puntos”, ya que si  $n$  es el número de puntos, el número de parejas será  $n^2$  y para cada pareja hay que revisar el resto de los puntos. La complejidad de ese algoritmo se denota  $O(n^3)$  que quiere decir que cuando  $n$  es muy grande el tiempo que se tarda es una constante (que depende del ordenador) por  $n^3$ .

## Cálculo de la envolvente convexa II



Un procedimiento mejor [2] es ordenar los puntos según el valor de la coordenada  $x$  y calcular la envolvente convexa en dos mitades, la superior y la inferior.

## Cálculo de la envolvente convexa II

Para el cálculo de la mitad superior:

- 1 Ordenamos los puntos en función de la coordenada  $x$ . Nos quedamos con el más a la izquierda y el más a la derecha.
- 2 Tomamos el siguiente punto a la izquierda. Para cada punto considerado debemos ir “girando a la derecha”.
- 3 En caso de que al añadir un punto dejemos de “girar a la derecha”, eliminamos puntos anteriores hasta que sea en efecto un giro a la derecha.
- 4 Cuando llegamos al extremo derecho terminamos.

## Complejidad del nuevo algoritmo

Ordenar los puntos se puede hacer de modo que cueste  $O(n \log n)$ .

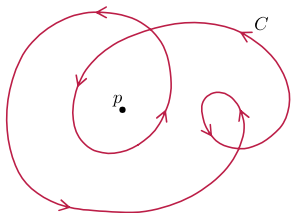
Recorrer los puntos cuesta  $O(n)$ .

Finalmente, cada vez que no “giremos a la derecha” habría que encontrar el último punto con el que nos podemos quedar y eso se puede hacer en  $O(\log n)$ .

Juntamos todo:

$$O(n \log n) + O(n)O(\log n) = O(n \log n)$$

## Número de rotación

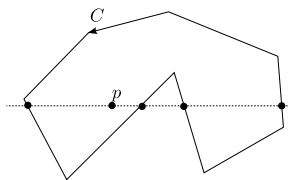


El número de rotación es un entero que representa el número de veces que una curva gira alrededor de un punto.

Se puede utilizar para determinar si un punto está dentro de una curva:

Está dentro de la curva si y sólo si su número de rotación es distinto de cero.

## Escaneo de línea



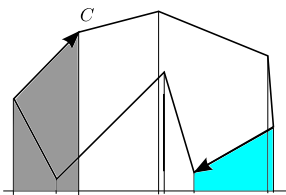
Para determinar si un punto  $P$  está dentro de la curva, trazamos la recta horizontal con la misma altura que  $P$ .

Contamos el número de cortes de la recta horizontal con la curva antes de llegar a  $P$ .

Si el número es impar, el punto pertenece a la curva y si es par, no pertenece.



## Cálculo de áreas de polígonos



Se traza una línea horizontal a una altura arbitraria.

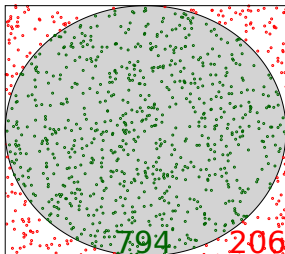
Para cada segmento orientado del polígono, de extremos  $p_1, p_2$  se hace lo siguiente:

Se define el paralelogramo limitado por el segmento, la línea horizontal y las verticales que pasan por  $p_1, p_2$ .

Si la coordenada  $x$  de  $p_1$  es menor que la de  $p_2$  se suma el área del paralelogramo.

Si la coordenada  $x$  de  $p_1$  es mayor que la de  $p_2$  se resta el área del paralelogramo.

## Método de Montecarlo



Encerramos la curva en un rectángulo de área  $A$ .

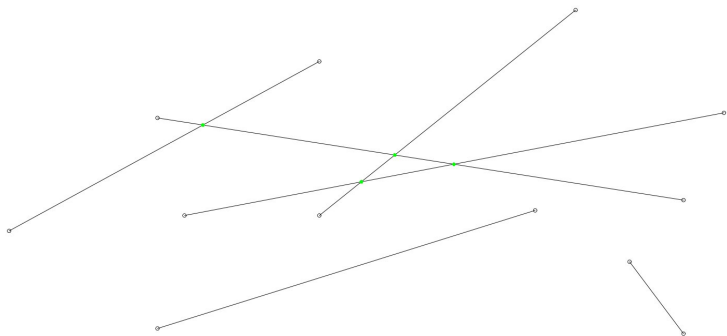
Generamos muchos puntos aleatorios en el rectángulo.

Calculamos el porcentaje de puntos en la región,  $R$ .

Aproximamos el área como  $RA$ .

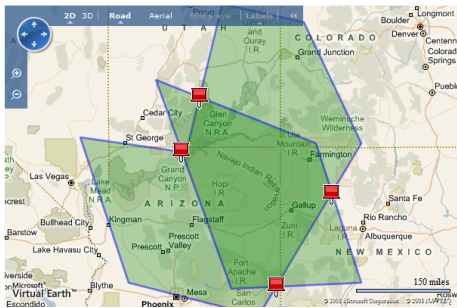
## Intersecciones de líneas y curvas

**Problema 1:** Calcula la intersección de un conjunto de segmentos entre sí.



## Intersecciones de líneas y curvas

**Problema 2:** Calcula la intersección de dos curvas entre sí.



## Cálculo de la intersección directamente

Una primera aproximación consiste en considerar todas las parejas de segmentos e ir calculando para cada pareja si se corta o no se corta.

Cada segmento lo numeraremos entre 1 y  $n$ , el número de segmentos. Como para una pareja no importa el orden, tomaremos siempre como primer elemento el que tenga menor número.

## Cálculo de la intersección directamente

- Tomamos cada uno de los segmentos como primer elemento de la pareja. Esto lo podemos realizar con un bucle que recorra los números entre 1 y  $n$  (luego veremos que basta llegar hasta  $(n - 1)$ ). Denotemos  $i$  a este segmento.
- Tomamos como segundo elemento uno que tenga mayor número de orden. Esto lo podemos hacer con un segundo bucle dentro del primero que recorra los números entre  $i + 1$  y  $n$ . Denotemos  $j$  a este segmento.
- Para cada pareja de segmentos  $i, j$ , generadas anteriormente, calculamos su intersección.

## Cálculo de la intersección de dos segmentos

Supongamos que tenemos un segmento que comienza en el punto  $(x_0, y_0)$  y termina en  $(x_1, y_1)$  y un segundo segmento que comienza en  $(x_2, y_2)$  y termina en  $(x_3, y_3)$ . ¿Dónde se cortan?

Los puntos del primer segmento los podemos representar como

$$(x_0, y_0) + \lambda(x_1 - x_0, y_1 - y_0) \quad 0 < \lambda < 1$$

y los del segundo

$$(x_2, y_2) + \mu(x_3 - x_2, y_3 - y_2) \quad 0 < \mu < 1$$

Ahora basta resolver el sistema

$$\begin{cases} x_0 + \lambda(x_1 - x_0) = x_2 + \mu(x_3 - x_2) \\ y_0 + \lambda(y_1 - y_0) = y_2 + \mu(y_3 - y_2) \end{cases}$$

y comprobar que la solución verifica las restricciones de  $\lambda$  y  $\mu$  anteriores.

## Complejidad algorítmica

Como hay que estudiar  $n(n - 1)/2$  parejas de segmentos y para cada pareja podemos suponer que se tarda un tiempo constante (no hay bucles), este algoritmo tiene un coste en tiempo del orden de  $n^2$ .



# Sweep-line

Los algoritmos de “sweep-line” son ampliamente utilizados en Geometría Computacional.

La idea es recorrer el plano mediante una línea que se desliza de arriba a abajo e ir calculando la estructura que queremos en función de su intersección con la línea.

## Intersección de la “sweep-line” con los segmentos

Al ir bajando la línea, el corte con el conjunto de segmentos es una serie de puntos (por simplicidad, vamos a asumir que no hay líneas horizontales, lo que también se asume en el algoritmo de R dado). Al ir bajando la línea, su “estructura” (en matemáticas diríamos topología) sólo cambia cuando aparece un nuevo segmento, cuando desaparece un segmento o cuando hay una intersección. Esto hace que sólo sea necesario calcularla en dichos puntos.

## Eventos

Vamos a denominar **eventos** a los tres casos anteriores, es decir, que aparezca un segmento en la “sweep-line”, que desaparezca o que haya una intersección.

Para cada evento, guardaremos la altura a la que se produce, el segmento que lo produce (en el caso de una intersección, guardaremos el segmento más a la izquierda) y el tipo de evento (1 para segmento que aparece, 2 para segmento que desaparece, 3 para intersección).

## Eventos

Al comenzar, ya podemos calcular todos los eventos de tipo 1 y 2 que se van a producir, así que crearemos una lista  $Q$  con los eventos y guardaremos esos eventos, ordenados de mayor a menor altura.

También guardaremos la “topología” de la “sweep-line” en un vector, que principio estará vacío. Después iremos guardando los segmentos que estén “activos” en cada momento, ordenados según su coordenada “ $x$ ” a esa altura (de menor a mayor).

Comenzaremos entonces a tratar los eventos, comenzando por el que se produzca a mayor altura.

## Eventos de creación

Al producirse un evento de creación haremos lo siguiente:

- 1 Buscamos el nuevo segmento en qué posición se coloca en la “sweep-line” .
- 2 Lo incluimos en la “sweep-line” .
- 3 Tomamos sus dos vecinos y comprobamos si se corta con alguno de ellos.
- 4 Si se corta, añadimos un evento de intersección.

## Eventos de creación

Al producirse un evento de eliminación haremos lo siguiente:

- 1 Buscamos su posición en la “sweep-line” y lo eliminamos.
- 2 Comprobamos si sus dos vecinos (ahora adyacentes) se cortan.
- 3 Si se cortan, añadimos un evento de intersección.

## Eventos de intersección

Al producirse un evento de intersección haremos lo siguiente:

- 1 Guardamos la intersección.
- 2 Los dos segmentos que producen la intersección intercambian su posición en la “sweep-line”.
- 3 Para cada uno de ellos, comprobamos si tienen intersección con sus nuevos vecinos.
- 4 Por cada intersección encontrada añadimos un evento de intersección.

## Cálculo de la posición de un punto en la “sweep-line”

Para calcular la posición de un punto  $(x, y)$  correspondiente a un nuevo segmento respecto a un segmento  $(x_i, y_i) \rightarrow (x_f, y_f)$  basta comparar la intersección del segmento  $(x_i, y_i) \rightarrow (x_f, y_f)$  con la línea de altura  $y$ , y esta es:

$$\left( x_i + \frac{y - y_i}{y_f - y_i} (x_f - x_i), y \right).$$








## Complejidad algorítmica

El algoritmo, bien implementado, tiene una complejidad del orden de  $(n + m) \log(n)$ , donde  $n$  es el número de segmentos y  $m$  el de intersecciones:

- Se ejecutan  $2n + m$  eventos;  $2n$  de inicio y fin de un segmento y  $m$  de intersecciones.
- Para cada evento, hay que buscar o insertar en la “sweep-line” y esto se puede hacer con un coste  $\log(k)$  donde  $k$  es el número de elementos de la “sweep-line” (obviamente  $k \leq n$ ).

El segundo punto es cierto sólo si para guardar la “sweep-line” se utiliza un árbol de búsqueda binario. Como no ha sido el caso (la hemos guardado simplemente en un vector) la complejidad del algoritmo es mayor (concretamente,  $(n + m)k$ , donde  $k$  es el tamaño máximo de la “sweep-line”).

## Bibliografía

-  M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, “Computational Geometry Algorithms and Applications”, Springer, 2008.
-  A.M. Andrew, “Another Efficient Algorithm for Convex Hulls in Two Dimensions”, Info. Proc. Letters 9, **1979**, 216–219.
-  F. P. Preparata; Michael Ian Shamos, “Computational Geometry”, Springer-Verlag, 1985
-  J. O’Rourke, “Computational Geometry in C”, Cambridge University Press, 1998
-  M. Kreveld, J. Nievergelt, T. Roos, P. Widmayer, “Algorithmic Foundations of Geographic Information Systems”, Springer, 1997

## Enlaces relacionados

- The Very Many (diseño basado en geometría computacional)
- Grupo give (descripción de muchos problemas aplicados de geometría computacional que están actualmente investigando)
- Librería de Geometría Computacional en C++
- Diagramas de Voronoi: aplicaciones, tutoriales, publicaciones, etc