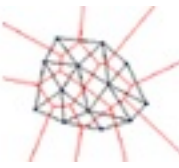


## Intersección de segmentos

# Índice



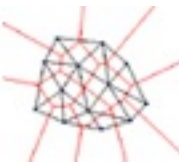
- Modelos geométricos básicos
- Un primer algoritmo
- Algoritmo avanzado

# Aplicaciones



- GIS: Sistema de Información Geográfica
- Uso de capas para representar información
  - ciudades, carreteras, rios, líneas de ferrocarril, ..
  - regiones de precipitación, de vegetación, ...
- Es interesante buscar intersecciones entre elementos y entre regiones

# Algoritmo de fuerza bruta



- Fuerza bruta:

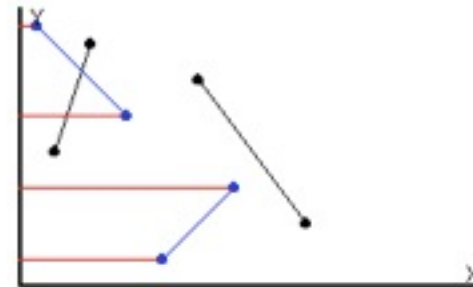
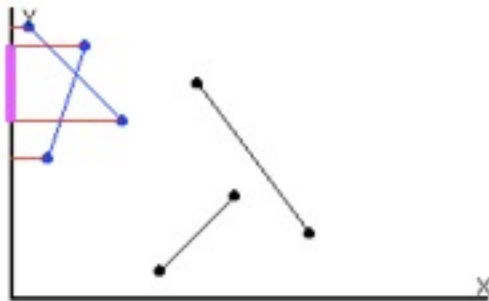
```
Para cada par de segmentos (s1,s2)
  si Intersectan(s1,s2)
    anotar PuntoDeInterseccion(s1,s2)
```

- Alto coste debido a que se comprueban todas las posibles combinaciones:  $O(n^2)$
- Mejora: hacer el coste del algoritmo dependiente de las **posibles intersecciones**

# Posibles intersecciones



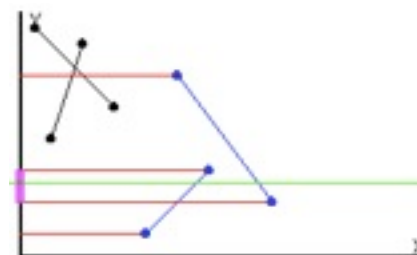
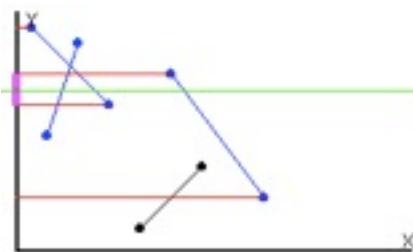
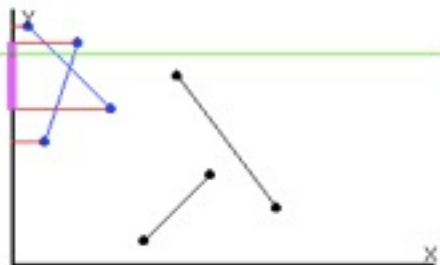
- Dos segmentos sólo son candidatos de intersectarse si se solapan sus proyecciones en el eje Y



# Línea de barrido



- Una línea de barrido imaginaria, moviéndose de arriba abajo, permite ir detectando los segmentos cuyas proyecciones se solapan



# Estado de la línea de barrido



- Consideramos que el **estado** de la línea de barrido está formado por los segmentos que ésta intersecta.
- El estado no cambia de forma continua, sólo lo hace en ciertos puntos discretos: **puntos evento**, que son los puntos inicio y final de segmento.

# Estructuras de datos



- **lista-evento**: conjunto de puntos iniciales y finales de los segmentos, ordenados por coordenada y.  
Implementación: array, Vector, lista enlazada
- **lista-estado**: lista de segmentos que intersectan la línea de barrido  
Implementación: array, Vector, lista enlazada

# Algoritmo de barrido (v. 0.5)

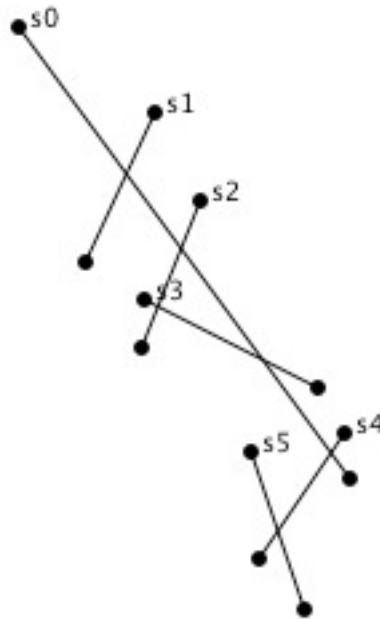


1. Insertar todos los puntos inicio y final en *lista-evento* de forma ordenada según su coordenada *y*
2. Mientras que *lista-evento* esté llena
3.      $p := \text{QuitarMin}(\textit{lista-evento})$
4.     Caso 1:  $p$  es un punto inicio del segmento  $s$
5.         Para cada segmento  $t$  de *lista-estado*
6.             Si  $\text{Intersectan}(s,t)$  anotar  $\text{Interseccion}(s,t)$
7.             Añadir  $s$  a *lista-estado*
8.     Caso 2:  $p$  es un punto final del segmento  $s$
9.     Eliminar  $s$  de la *lista-estado*

# Ejemplo



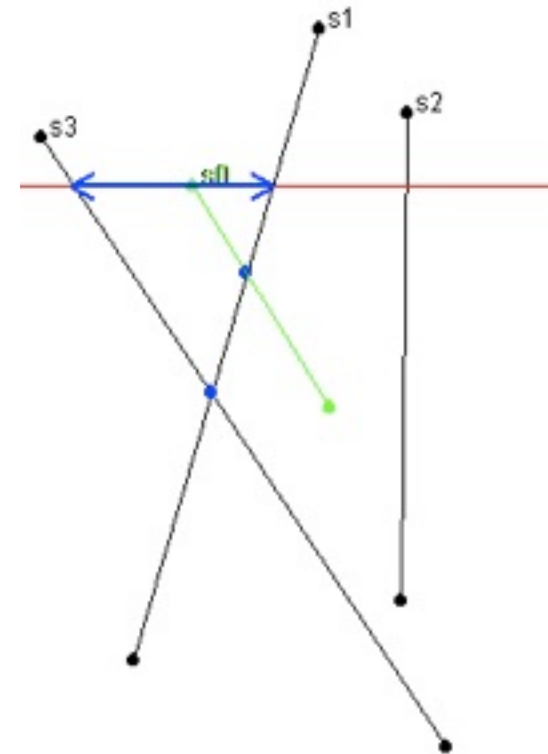
- Hacemos una traza del algoritmo con el siguiente ejemplo de segmentos



# Mejora: comprobar con vecinos



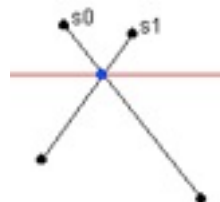
- Ordenamos de izquierda a derecha los segmentos que intersectan la línea de barrido según **coordenada x de la intersección con la línea de barrido**
- Sólo comprobaremos segmentos adyacentes al nuevo segmento



# Nuevos puntos evento



- Los puntos intersección son nuevos puntos evento: en ellos se cambia la ordenación de izquierda a derecha de los segmentos.



lista-estado: (s0,s1)



lista-estado: (s1,s0)

# Estructuras de datos



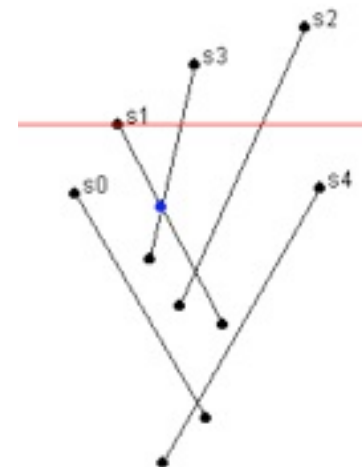
- **lista-evento**: conjunto de puntos iniciales, finales, y puntos de intersección  
Implementación: array, Vector, lista enlazada
- **lista-estado**: lista de segmentos que intersectan la línea de barrido, ordenados según la coordenada x del punto de intersección del segmento con la línea de barrido  
Implementación: array, Vector, lista enlazada, árbol

# Evento: Punto inicial



Se añade segmento a lista-estado

Se comprueba con los segmentos adyacentes (sólo  $s_3$ ). Se detecta un punto de intersección que se inserta, de forma ordenada, en la lista-evento



lista-estado: ( $s_3, s_2$ )



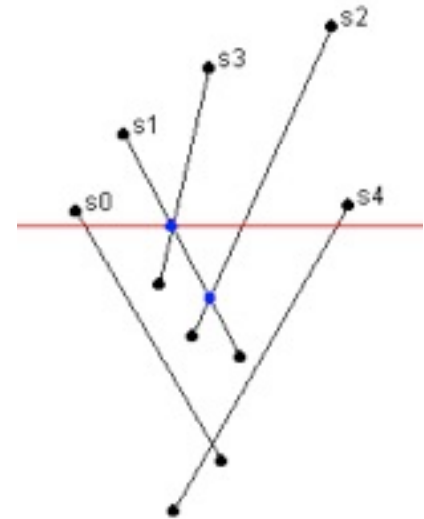
lista-estado: ( $s_1, s_3, s_2$ )

# Evento: Punto de intersección



Se intercambia en la lista-estado el orden de los segmentos

Al cambiar el orden entre  $s_3$  y  $s_1$ , se debe comprobar intersección con sus nuevos vecinos:  $s_3$  con  $s_0$  y  $s_1$  con  $s_2$ . Se detecta otra intersección entre  $s_1$  y  $s_2$ .



lista-estado:  $(s_0, s_1, s_3, s_2, s_4)$



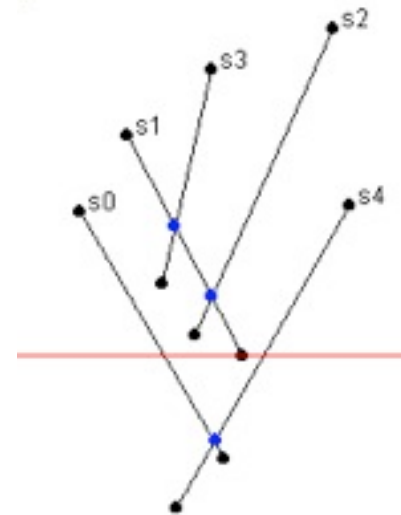
lista-estado:  $(s_0, s_3, s_1, s_2, s_4)$

# Punto final



Se elimina segmento de lista-estado.

Al eliminar  $s_1$ , aparecen dos nuevos segmentos vecinos que hay que comprobar:  $s_0$  y  $s_4$ , generando la última intersección.



lista-estado:  $(s_0, s_1, s_4)$



lista-estado:  $(s_0, s_4)$

# Algoritmo de barrido (v. 0.9)



1. Insertar los puntos inicio y final en la *lista-evento* de forma ordenada según su coordenada y
2. Mientras que *lista-evento* esté llena
3.      $p := \text{QuitarMin}(\textit{lista-evento})$
4.      $\text{TratarPuntoEvento}(p, \textit{lista-estado})$
5.     Si se encuentran nuevas intersecciones
6.     Añadir los puntos de intersección en la *lista-evento*

# TratarPuntoEvento( $p$ , $l$ -estado)



## Caso 1

$p$  es un punto inicio del segmento  $s$   
Insertar  $s$  de forma ordenada en *lista-estado*  
Comprobar si intersecciona con los segmentos adyacentes

## Caso 2

$p$  es un punto final del segmento  $s$   
Eliminar  $s$  de la *lista-estado*  
Comprobar si interseccionan los nuevos segmentos adyacentes

## Caso 3

$p$  es un punto de intersección de los segmentos  $s_1$  y  $s_2$   
Intercambiar el orden de  $s_1$  y  $s_2$  en *lista-estado*  
Comprobar si interseccionan con los nuevos adyacentes

# Casos degenerados



- La versión vista del algoritmo no funciona bien con los siguientes casos degenerados:
  - Más de un segmento intersectan en el mismo punto
  - Más de un segmento comienza en el mismo punto
- Para considerar estos casos, consultar la versión del algoritmo propuesta en el **de Berg**