Intersección entre segmentos de recta

comp-420

Propiedades de segmentos de recta

- Una combinación convexa de dos puntos $p_1=(x_1,y_1)$ y $p_2=(x_2,y_2)$ es cualquier punto $p_3=(x_3,y_3)$ tal que para una α en el rango de $0 \leq \alpha \leq 1$, tenemos:

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

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

- Dados dos puntos distintos p_1 y p_2, el segmento de recta $\overline{p_1p_2}$ es el conjunto de combinaciones convexas de p_1 y p_2.

- Llamamos a p_1 y p_2 los puntos extremos del segmento $\overline{p_1p_2}$.

- A veces nos importa el orden del segmento por lo que nos referimos al segmento dirigido $\overrightarrow{p_1p_2}$.

Propiedades de segmentos de recta

- Si p_1 es el origen (0,0), podemos tratar al segmento dirigido $\overrightarrow{p_1p_2}$ como el vector p_2.
- Exploraremos las preguntas siguientes:
  - Dados los segmentos dirigidos $\overrightarrow{p_0p_1}$ y $\overrightarrow{p_0p_2}$, ¿está $\overrightarrow{p_0p_1}$ en el sentido de las manecillas del reloj de $\overrightarrow{p_0p_2}$ respecto a su punto extremo común p_0?
  - Dados dos segmentos $\overrightarrow{p_0p_1}$ y $\overrightarrow{p_1p_2}$, si atravesamos $\overrightarrow{p_0p_1}$ y luego $\overrightarrow{p_1p_2}$, ¿hacemos una vuelta a la izquierda en el punto p_1?
  - ¿Intersectan los segmentos p_1p_2 y p_3p_4?
  - Estas preguntas se pueden responder en tiempo $O(1)$.
  - Los métodos para responder estas preguntas son solamente sumas, multiplicaciones y comparaciones.

Sentido de un segmento de recta respecto a otro

- Para determinar si un segmento dirigido $\overrightarrow{p_0p_1}$ está o no en sentido de las manecillas del reloj de un segmento $\overrightarrow{p_0p_2}$ respecto a un punto extremo común p_0:
- Translادamos para usar p_0 como origen.
- Hacemos que p_1-p_0 denote el vector $p_1'=(x_1',y_1')$ donde:

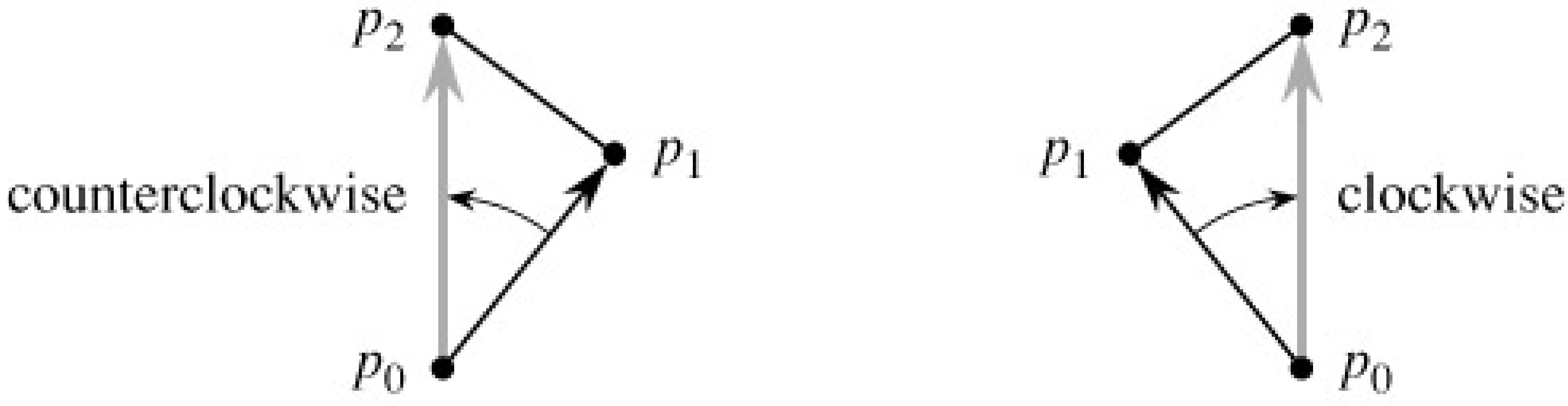
$$x_1' = x_1-x_0$$

$$y_1'=y_1-y_0.$$

- Definimos p_2-p_0 de la misma manera.
- Calculamos el producto vectorial:

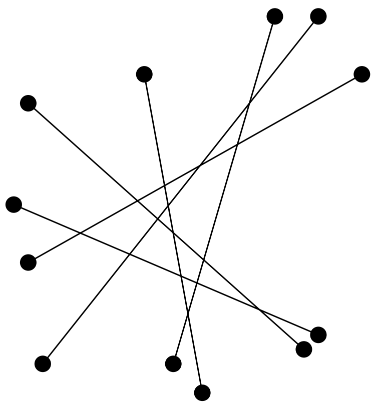
$$(p_1-p_0) \times (p_2-p_0) = (x_1-x_0)(y_2-y_0) - (x_2-x_0)(y_1-y_0)$$

Sentido de un segmento de recta respecto a otro



Intersección entre segmentos de recta

- Dados dos conjuntos de segmentos de rectas, calcular las intersecciones entre un segmento de un conjunto A y un segmento del otro conjunto B.
- Consideraremos un segmento de A cuyo punto extremo esté sobre un segmento en B como segmentos que intersectan (los segmentos son cerrados).
- Para encontrar todas las intersecciones creamos un conjunto $S = A \cup B$.
- La especificación del problema es la siguiente:
  - Dado un conjunto S de n segmentos cerrados en el plano, reportar todos los puntos de instersección entre los segmentos en S.



Intersección entre segmentos de recta

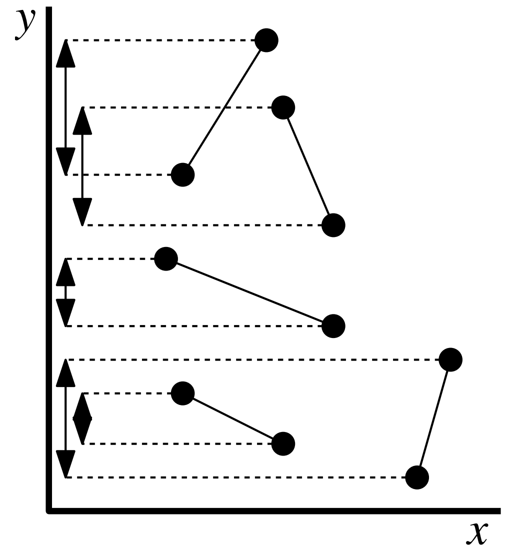
- Algoritmo de fuerza bruta:
  - tomar cada par de segmentos,
  - calcular si intersectan o no,
  - reportar la intersección
- Este algoritmo requiere un tiempo de ejecución de $O(n^2)$.
- Cuando cada par de segmentos está intersectando, cualquier algoritmo toma $\Omega(n^2)$ porque tiene que reportar todas las intersecciones
- En el caso general el número total de puntos de intersección es mucho menor que una cota cuadrática.

Intersección entre segmentos de recta

- Nos interesa un algoritmo que dependa:
    - número de segmentos de entrada,
    - número de puntos de intersección.
- Algoritmo sensible a la salida (output-sensitive algorithm)
- Para evitar probar todos los pares de segmentos hay que aprovechar la geometría del conjunto:
    - segmentos cercanos son candidatos a intersectar,
    - segmentos lejanos no son condidatos a intersectar.

Intersección entre segmentos de recta

- Definimos el intervalo y de un segmento como su proyección ortogonal en el eje y:



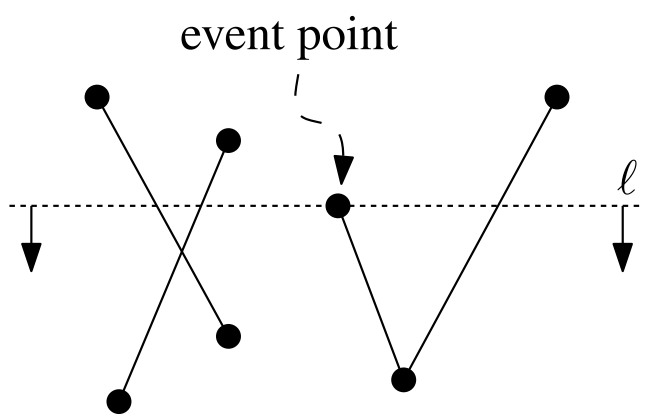
- Cuando los intervalos y de un par de segmentos no se sobreponen, podemos decir que están lejos y que no pueden intersectar.
- IDEA:
  - probar los pares de segmentos cuyos intervalos y se sobreponen (que haya una línea horizontal que intersecte los segmentos)

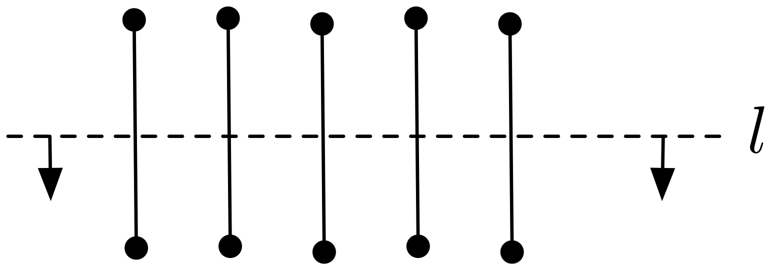
Intersección entre segmentos de recta

- Para encontrar los pares imaginemos una línea l que barre el plano de arriba hacia abajo.
- El algoritmo analiza los segmentos que intersectan esta línea.
- Este tipo de algoritmo es conocido como barrido de plano (sweep-plane) y la línea l se conoce como línea de barrido (sweep-line).
- El estado de la línea de barrido es el conjunto de segmentos que la intersectan.
- El estado cambia mientras la línea de barrido se mueve hacia abajo, pero no en forma contínua.
- Solo en puntos particulares es necesario actualizar el estado. Estos puntos se conocen como puntos evento (event points) en el algoritmo.
- Los puntos evento son los puntos extremos del segmento.

Intersección entre segmentos de recta

- Si el punto evento es el extremo superior del segmento, el segmento es añadido al estado de la línea de barrido.
- Este segmento será probado con los segmentos que ya están en el estado.
- Si el punto evento es el extremo inferior del segmento, este es retirado del estado de la línea.





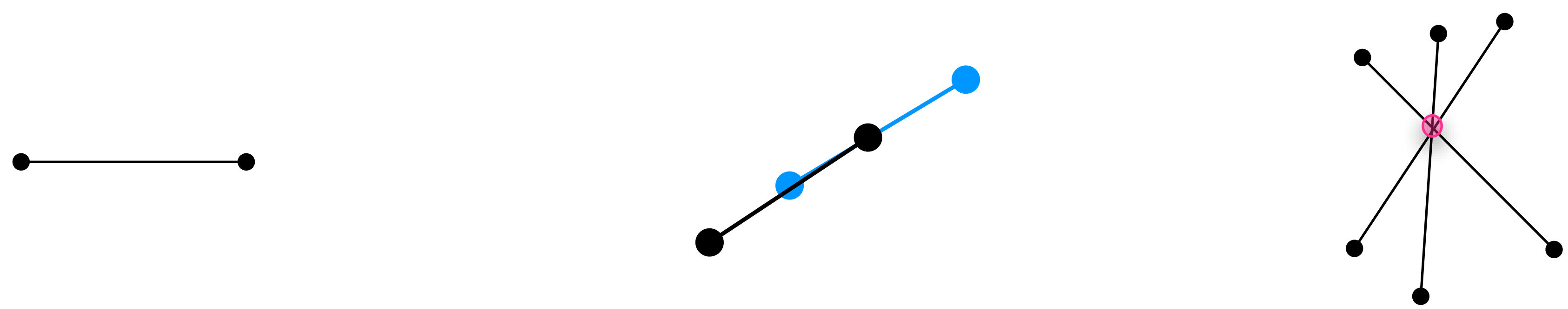
- ¡¡Todavía no es sensible al número de intersecciones!!

Intersección entre segmentos de recta

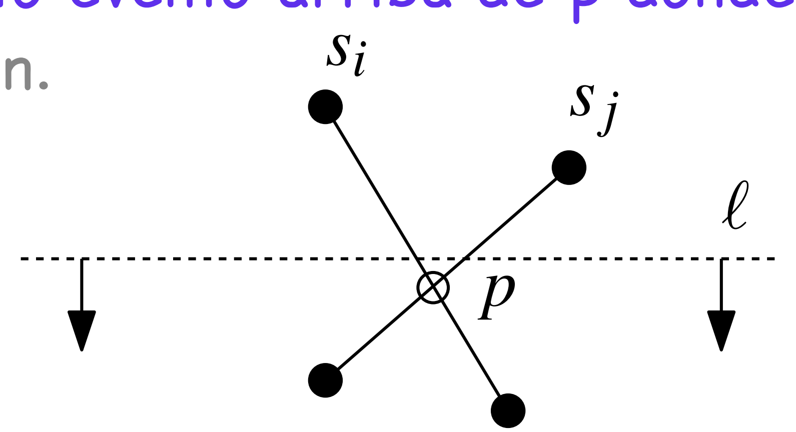
- Ordenar los segmentos de izquierda a derecha como intersectan la línea de barrido para incluir la idea de cercanía en la dirección horizontal.
- Se verificarán los segmentos adyacentes en el ordenamiento horizontal.
- Mientras baja la línea de barrido puede cambiar la adyacencia de los segmentos. Esto debe reflejarse en el estado de la línea de barrido.
- El nuevo estado esta formado, además de los puntos extremo, de los puntos de intersección (cambios de adyacencia).
- Con esta estratégia se reducen los pares de segmentos que verifican pero ¿se encuentran todas las intersecciones?
- Si dos segmentos s_i y s_j intersectan ¿habrá siempre una posición en la línea de barrido l donde s_i y s_j sean adyacentes sobre l?

Intersección entre segmentos de recta

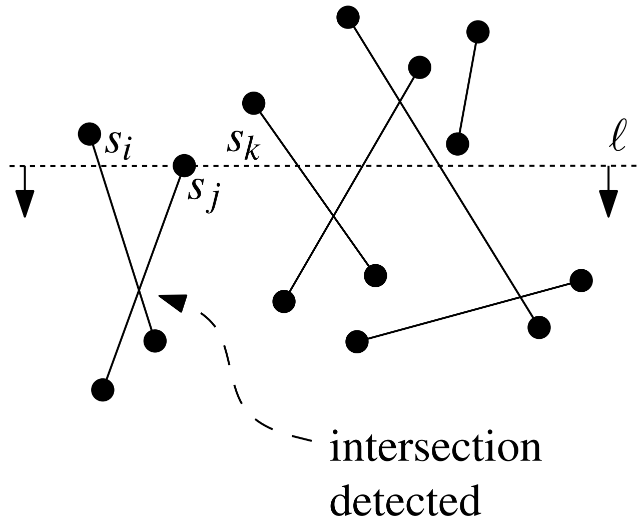
- Ignoremos primero los casos degenerados:

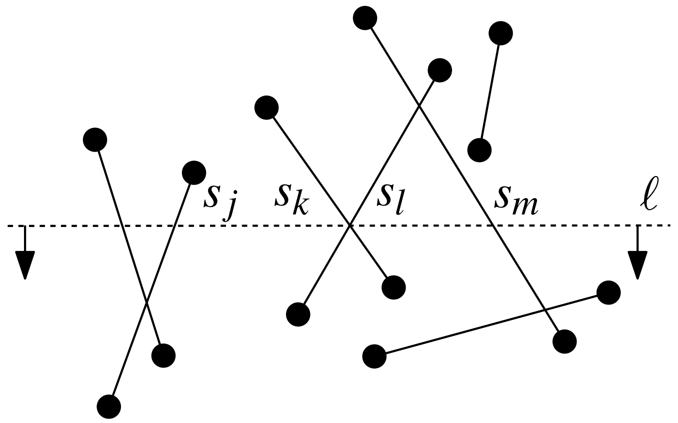


- Las intersecciones en puntos extremos se detectan fácilmente cuando están sobre la línea de barrido.
- Sean s_i y s_j dos segmentos no-horizontales cuyos interiores intersectan en un solo punto p,
- Supongamos que no hay un tercer segmento que pase por p.
- Entonces hay un punto evento arriba de p donde s_i y s_j son adyacentes y se probó si intersectaban.

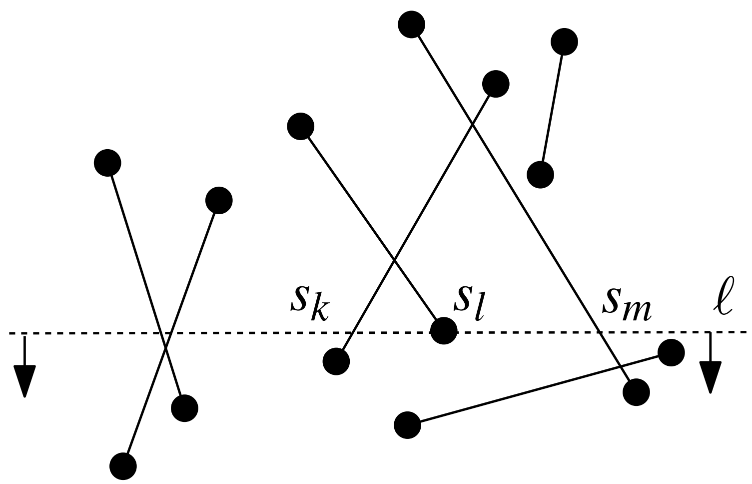


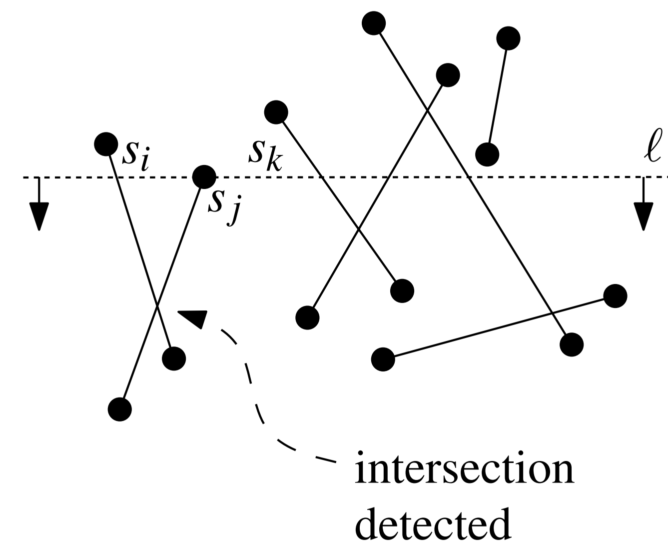
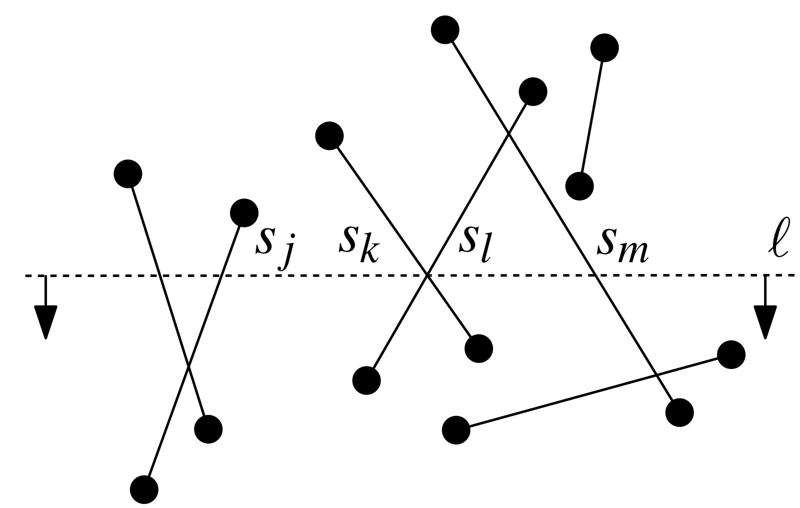
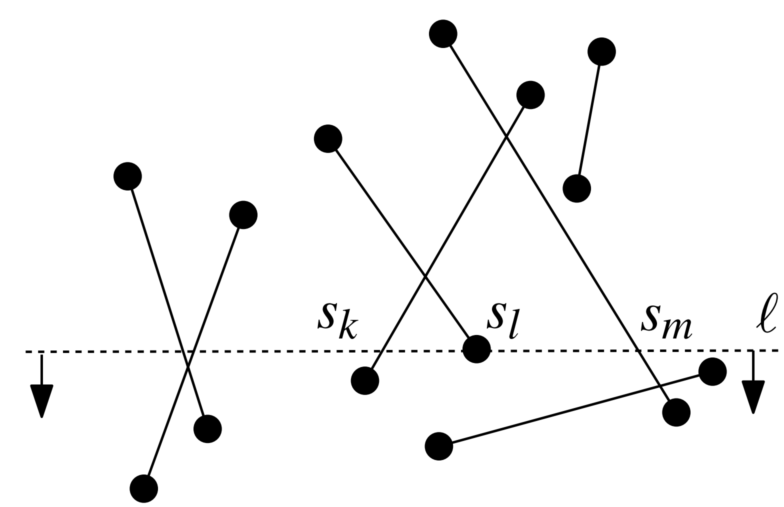
Intersección entre segmentos de recta





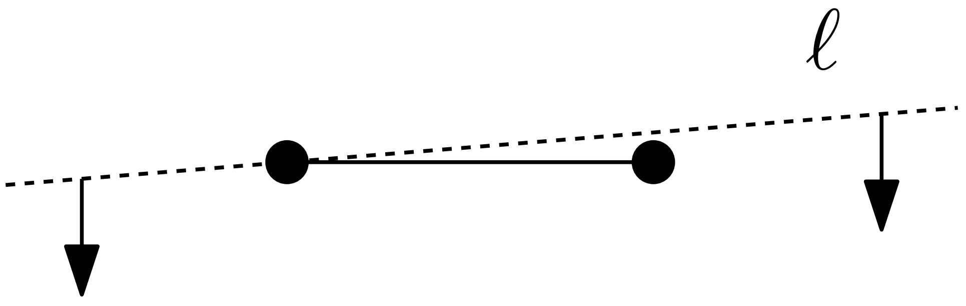
- Solo nos interesan las intersecciones abajo de la línea de barrido.



Evento	Acción	Ejemplo
extremo superior alcanzado	probar el segmento contra sus dos vecinos sobre la línea de barrido.	
cambio de adyacencia entre segmentos	cada segmento toma un nuevo vecino a lo más contra quién debera ser probado.	
extremo inferior alcanzado	sus dos vecinos se hacen adyacentes y deben ser probados.	

Intersección entre segmentos de recta: Estructuras de datos

- ¿Qué estructuras de datos se necesitan para implementar este algoritmo?
- cola de eventos Q.
- Operaciones:
- Eliminar el próximo evento (el más alto abajo de la línea de barrido) en Q y regresar el punto.
- Si dos puntos evento tienen la misma coordenada y, entonces regresar aquel con la coordenada x más pequeña.
- En una línea horizontal el punto a la izquierda será el extremo superior.



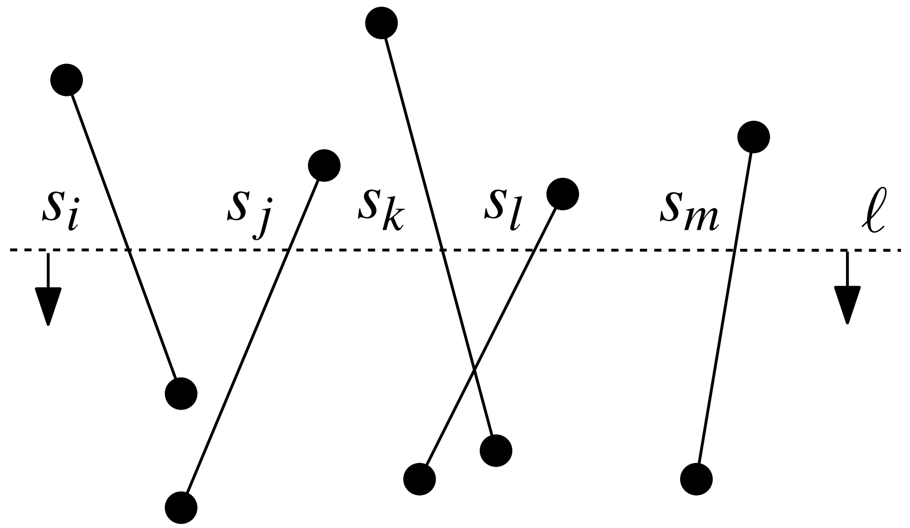
Intersección entre segmentos de recta: Estructuras de datos

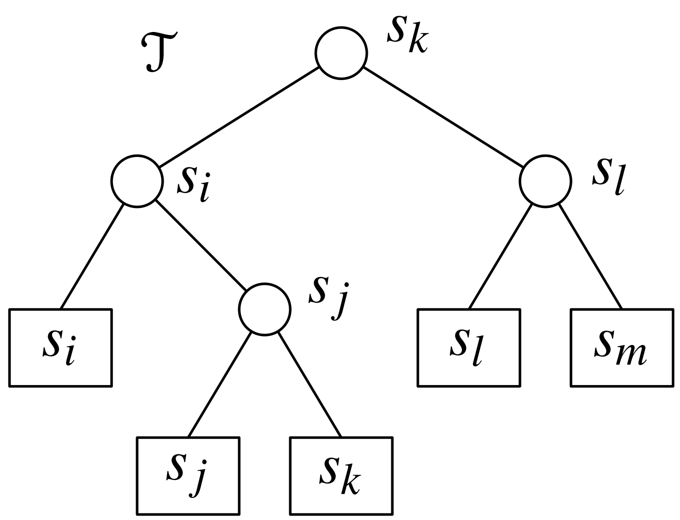
- Insertar un evento.
- Verificar si un segmento está dentro de Q.
- Definir un orden $\prec$ en los puntos evento.
- Si p y q son puntos evento, $p \prec q$ si y solo si $p_y > q_y$ o si $p_y = q_y$, $p_x < q_x$.
- Guardar los puntos evento en un árbol binario balanceado, ordenado de acuerdo a . $\prec$
- Con cada punto evento p en Q se deben almacenar también los segmentos que empiecen en p.
- Ambas operaciones toman O(log m) donde m es el número de eventos en Q.
- No se utiliza un montículo porque hay que verificar si un evento ya está presente en Q.

Intersección entre segmentos de recta: Estructuras de datos

- Se debe mantener un estado del algoritmo,: una secuencia de segmentos ordenados que intersecten la línea de barrido.
- La estructura del estado T, se usa para acceder a los vecinos de un segmento dado s, de tal manera que se pueda probar si intersecta con s.
- La estructura debe ser dinámica ya que los segmentos empiezan o terminan de intersectar a la línea de barrido (se añaden y eliminan).
- Como hay un orden bien definido en los segmentos dentro de la estructura de estado, se puede usar un árbol binario de búsqueda balanceado.
- Los segmentos que intersectan la línea de barrido se encuentran en el mismo orden en las hojas del árbol binario de búsqueda.

Intersección entre segmentos de recta: Estructuras de datos





- El orden de izquierda a derecha sobre la línea de barrido corresponde al orden de izquierda a derecha de las hojas de T.
- Los nodos internos mantienen la información necesaria para guiar la búsqueda hacia abajo.
- En cada nodo interno, almacenamos el segmento más a la derecha en el subárbol izquierdo.

Intersección entre segmentos de recta: Estructuras de datos

- Supongamos que buscamos en T al segmento inmediatamente a la izquierda de un punto p sobre la línea de barrido.
- En cada nodo interno v, probamos si p se encuentra a la izquierda o a la derecha del segmento almacenado en v.
- Dependiendo de estas prueba bajamos hacia el subárbol izquierdo o al derecho hasta llegar a una hoja.
- El segmento buscado estará almacenado en esta hoja o en la inmediata izquierda.
- Cada actualización y búsqueda de vecino toma O(log n).
- Las únicas estructuras que necesitamos entonces son:
    - La cola de eventos Q.
    - El estado de la línea de barrido T.

Intersección entre segmentos de recta

Algorithm FINDINTERSECTIONS(S)
Input. A set S of line segments in the plane.
Output. The set of intersection points among the segments in S, with for each intersection point the segments that contain it.

1. Initialize an empty event queue $\mathcal{Q}$. Next, insert the segment endpoints into $\mathcal{Q}$; when an upper endpoint is inserted, the corresponding segment should be stored with it.
2. Initialize an empty status structure $\mathcal{T}$.
3. **while** $\mathcal{Q}$ is not empty
4.   **do** Determine the next event point p in $\mathcal{Q}$ and delete it.
5.   HANDLEEVENTPOINT(p)

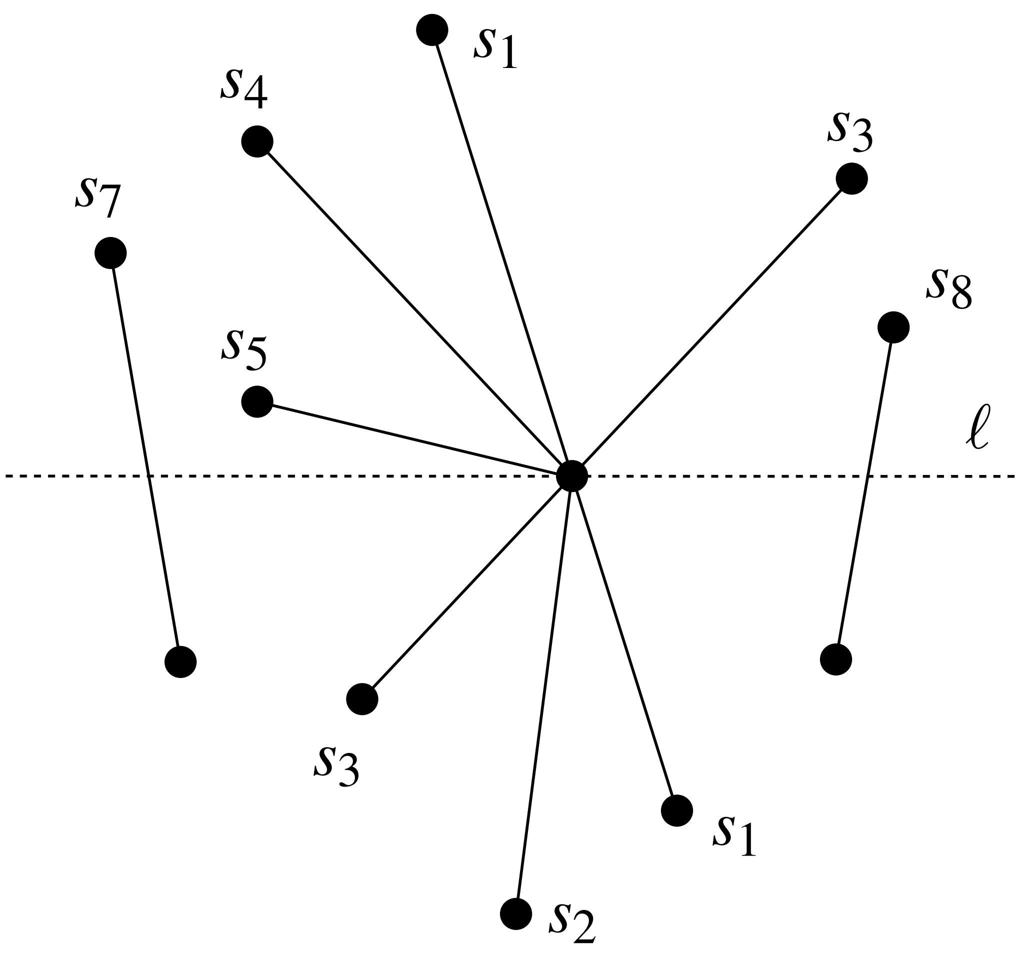
HANDLEEVENTPOINT(p)

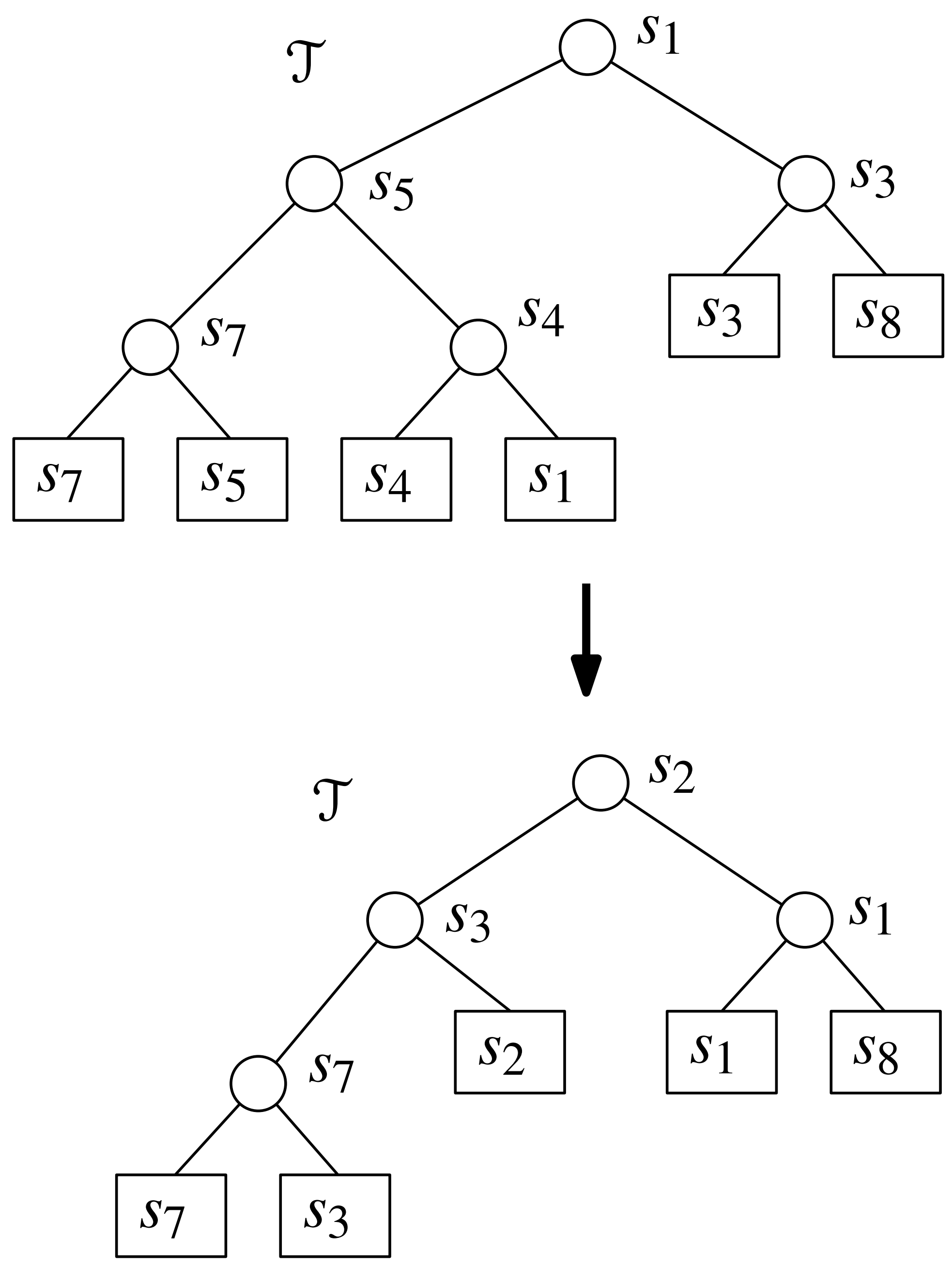
1. Let $U(p)$ be the set of segments whose upper endpoint is p; these segments are stored with the event point p. (For horizontal segments, the upper endpoint is by definition the left endpoint.)
2. Find all segments stored in $\mathcal{T}$ that contain p; they are adjacent in $\mathcal{T}$. Let $L(p)$ denote the subset of segments found whose lower endpoint is p, and let $C(p)$ denote the subset of segments found that contain p in their interior.
3. **if** $L(p) \cup U(p) \cup C(p)$ contains more than one segment
4.   **then** Report p as an intersection, together with $L(p)$, $U(p)$, and $C(p)$.
5. Delete the segments in $L(p) \cup C(p)$ from $\mathcal{T}$.
6. Insert the segments in $U(p) \cup C(p)$ into $\mathcal{T}$. The order of the segments in $\mathcal{T}$ should correspond to the order in which they are intersected by a sweep line just below p. If there is a horizontal segment, it comes last among all segments containing p.
7. ($*$ Deleting and re-inserting the segments of $C(p)$ reverses their order. $*$)
8. **if** $U(p) \cup C(p) = \emptyset$
9.   **then** Let s_l and s_r be the left and right neighbors of p in $\mathcal{T}$.
10.   FINDNEWEVENT(s_l, s_r, p)
11.   **else** Let s' be the leftmost segment of $U(p) \cup C(p)$ in $\mathcal{T}$.
12.   Let s_l be the left neighbor of s' in $\mathcal{T}$.
13.   FINDNEWEVENT(s_l, s', p)
14.   Let s'' be the rightmost segment of $U(p) \cup C(p)$ in $\mathcal{T}$.
15.   Let s_r be the right neighbor of s'' in $\mathcal{T}$.
16.   FINDNEWEVENT(s'', s_r, p)

Intersección entre segmentos de recta

FINDNEWEVENT(s_l, s_r, p)
1. **if** s_l and s_r intersect below the sweep line, or on it and to the right of the current event point p, and the intersection is not yet present as an event in $\mathcal{Q}$
2. **then** Insert the intersection point as an event into $\mathcal{Q}$.

Intersección entre segmentos de recta





Intersección entre segmentos de recta

- El algoritmo FINDINTERSECTIONS calcula todos los puntos y los segmentos que los contienen, correctamente.
- El algoritmo es correcto y además sensible a la salida, es decir, sensible al número de intersecciones.
- El tiempo de calculo del algoritmo es $O((n+k)\log n)$, donde k es el tamaño de la salida. Aún más que esto:
- El tiempo de cálculo del algoritmo FINDINTERSECTIONS para un conjunto S de n segmentos de recta en el plano es $O(n \log n + I \log n)$, donde I es el número de puntos de intersección de los segmentos en S.

Sea S un conjunto de n segmentos de recta en el plano. Se pueden reportar todos los puntos de intersección y los segmentos involucrados en ellos en un tiempo $O(n\log n + I \log n)$ y espacio $O(n)$, donde I es el numero de puntos de intersección.