# Geometria Computacional
# Interseção de Segmentos

*Claudio Esperança*

*Paulo Roma Cavalcanti*





# O Problema de Interseção

- Consiste em, dados dois ou mais objetos geométricos
  - Determinar *se* ele se interceptam (predicado)
  - Determinar qual sua interseção (objeto ou objetos na interseção)
- Os dois problemas são relacionados mas não são idênticos
  - Para determinar se 2 segmentos de reta se interceptam, basta fazer 4 testes de orientação
  - Para determinar o ponto de interseção resolve-se um sistema de equações





# Motivação

- Avaliação de bordo (CSG)
  - Requer computar curvas de interseção entre primitivas
- Robótica e planejamento de movimento
  - Detecção / prevenção de colisão
- Sistemas de Informações Geográficas (SIG/GIS)
  - Operações de superposição de Mapas
- Computação Gráfica
  - Traçado de raios





# Interseção de Segmentos de Reta

- Dada uma coleção de $n$ segmentos de reta, computar todos os pontos de interseção
- Quantos pontos de interseção podemos esperar?
  - No mínimo, 0
  - No máximo $\binom{n}{2} = O(n^2)$
- Portanto, um algoritmo $O(n^2)$ seria de certa forma, ótimo
  - Fácil de obter: testa-se todos os pares
- Em muitos casos, no entanto, espera-se poucas interseções
  - O ideal é um algoritmo sensível à saída





# Complexidade do Problema

- A complexidade esperada do problema é $O(n \log n + I)$ onde $I$ é o tamanho da saída
  - Precisamos de tempo $O(I)$ para reportar todos os pontos de interseção
  - Qual a razão para o termo $n \log n$ ?
    - Problema associado $\Omega(n \log n)$:
      - Determinar se $n$ números dados são todos distintos
    - Problema pode ser mapeado no problema de interseção de segmentos de reta
      - Construir $n$ retas verticais, cada uma com $x$ igual a um dos números
    - Se o problema de interseção pudesse ser resolvido em tempo $o(n \log n)$ – isto é, estritamente menor – então o problema associado também poderia



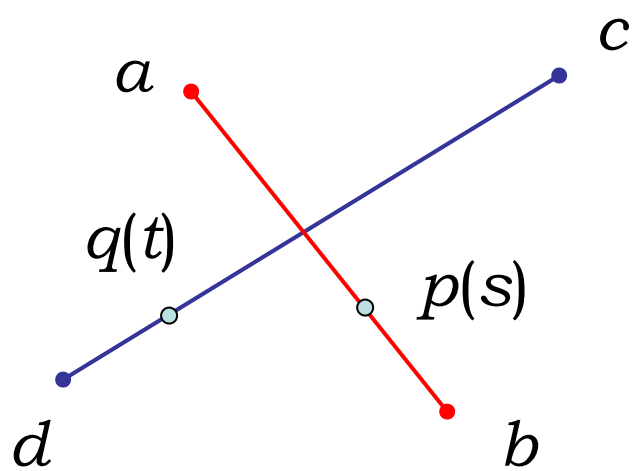


# Interseção de 2 segmentos

- Segmentos representados em forma paramétrica

$p(s) = (1 - s)\ a + s\ b \quad$ para $0 \le s \le 1$

$q(t) = (1 - t)\ c + t\ d \quad$ para $0 \le t \le 1$



# Interseção de 2 segmentos

- Interseção ocorre sse existem $s$ e $t$ tais que $p(s) = q(t)$ e $0 \leq s,t \leq 1$
- 2 equações e 2 incógnitas:

$$(1 - s)\, a_x + s\, b_x = (1 - t)\, c_x + t\, d_x$$
$$(1 - s)\, a_y + s\, b_y = (1 - t)\, c_y + t\, d_y$$

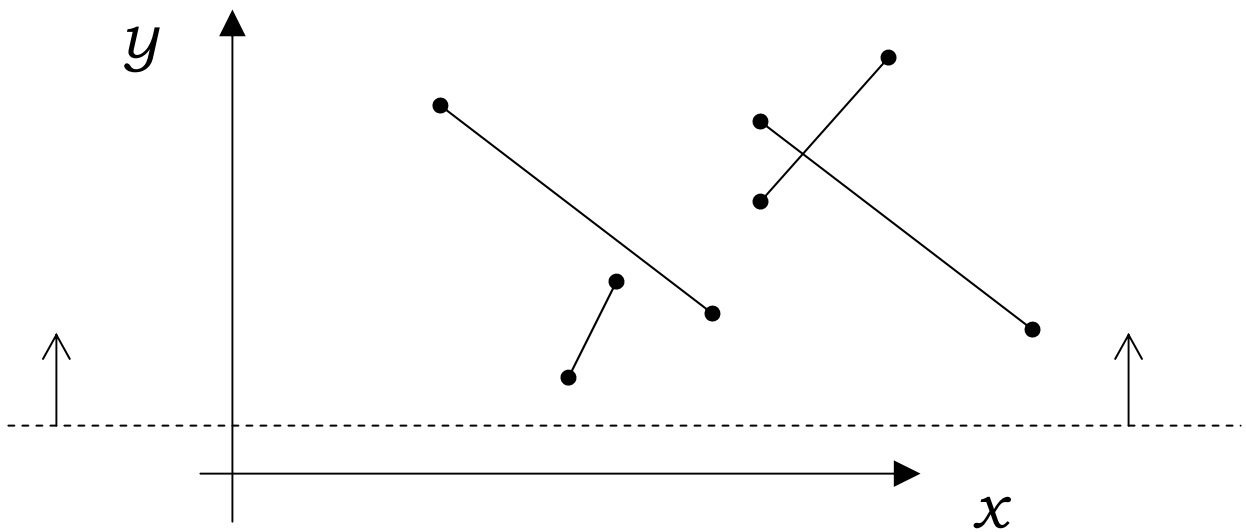
- Ao resolver o sistema, teremos que fazer uma divisão
  - Se o divisor for 0, as retas de suporte são paralelas ou coincidentes





# Algoritmo de Varredura

- Problema é resolvido percorrendo o plano com uma linha de varredura
  - Usaremos uma linha horizontal varrendo o plano desde $y = -\infty$ até $y = +\infty$
  - Na verdade, não precisamos examinar todos os valores de $y$, apenas aqueles para os quais alguma coisa acontece (eventos)



# Algoritmo de Varredura

- Eventos:
  - Linha passa por ponto extremo de um segmento
  - Linha passa por ponto de interseção

  (os primeiros podem ser ordenados, mas os segundos, não)
- Tratamento dos eventos
  - Atualização das estruturas de dados
  - Reportar pontos de interseção





# Posição Geral

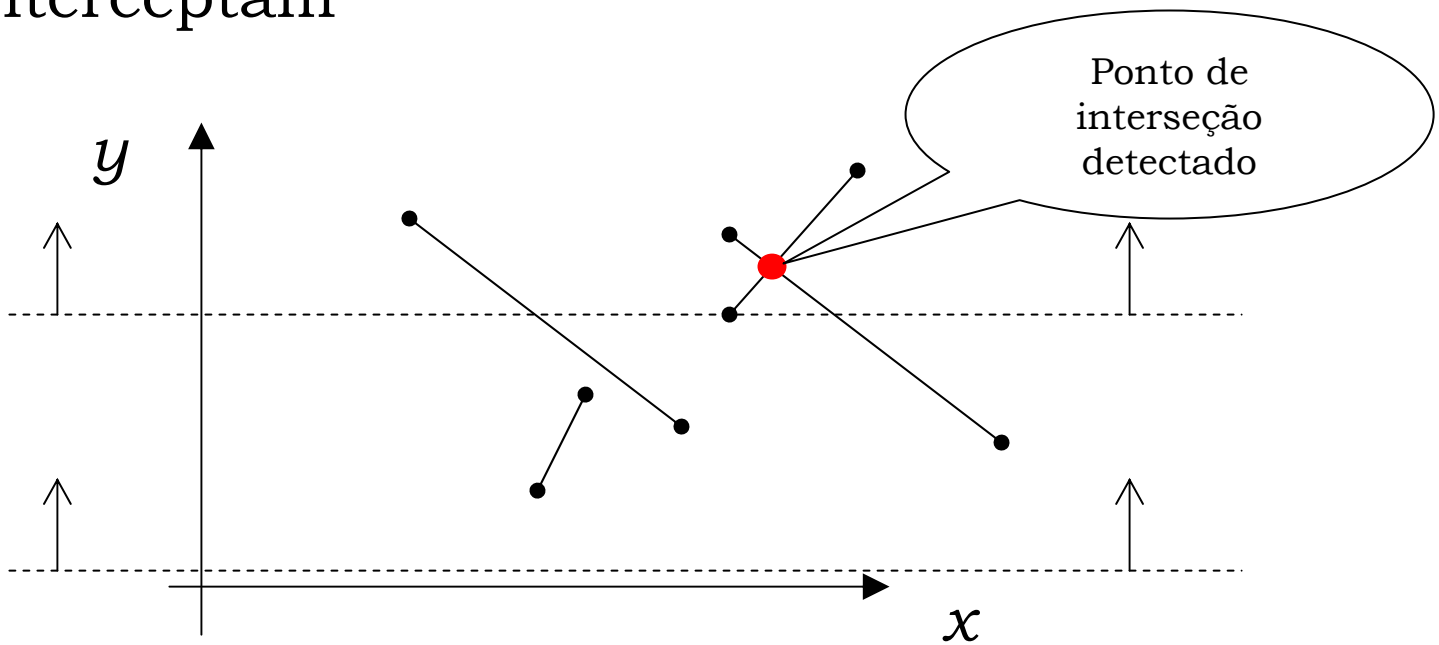
- Existem muitos casos degenerados que podem acontecer na prática e que iremos ignorar por enquanto:
  - Segmentos horizontais
  - Segmentos colineares
  - Interseção de mais de dois segmentos num mesmo ponto
- Esses casos serão abordados na implementação mas atrapalham na concepção e análise do algoritmo
- Este tipo de raciocínio é muito comum em Geometria Computacional. Diz-se que “os dados de entrada estão em posição geral” (*general position assumption*)





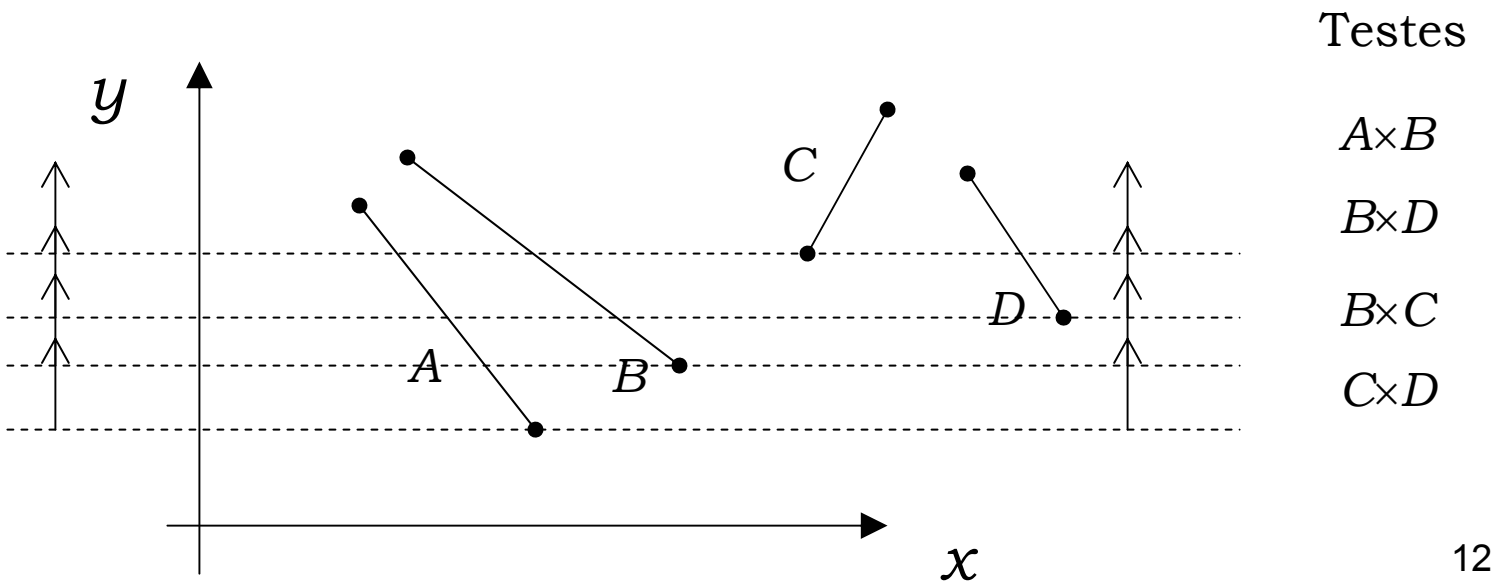
# Detecção de eventos de interseção

- Queremos detectar eventos de interseção antes que ocorram
  - Estrutura de dados que descreve a linha de varredura contém os segmentos interceptados ordenados por *x* do ponto de interseção
  - Sempre que há alteração nessa ordem, os segmentos afetados são testados para ver se se interceptam



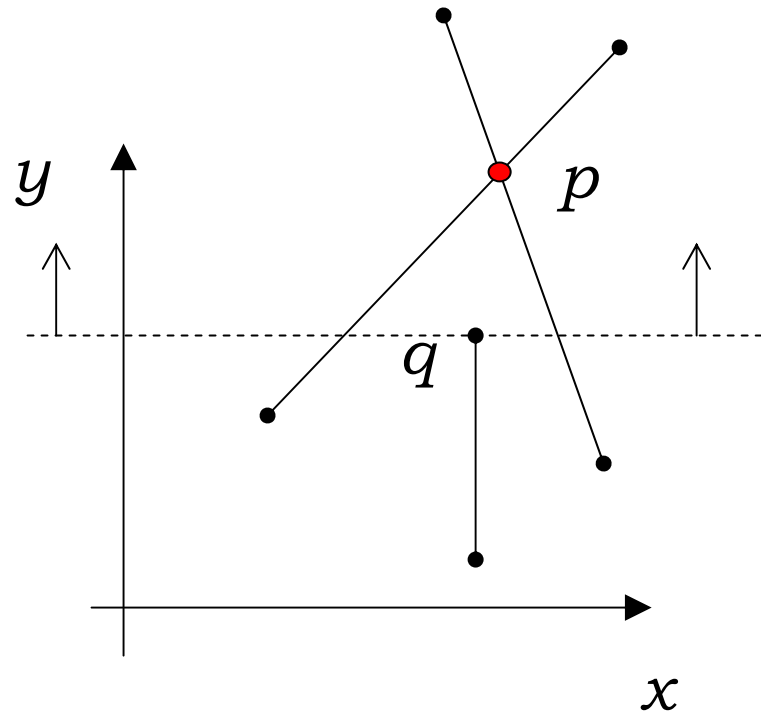
# Detecção de eventos de interseção

- Não queremos testar 2 a 2 todos os segmentos que interceptam a linha de varredura
  - No pior caso, isso levaria a complexidade $O(n^2)$
- A idéia é testar apenas pares de segmentos consecutivos com respeito à linha de varredura. Por exemplo:



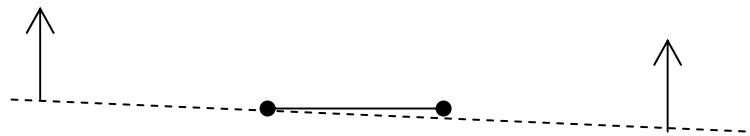
# Detecção de eventos de interseção

- Lema:
  - Se 2 segmentos $s_i$ e $s_j$ se interceptam num ponto $p$ então $s_i$ e $s_j$ foram adjacentes com relação à linha de varredura em alguma posição da mesma anterior a $p$
- Prova:
  - Num ponto infinitesimalmente abaixo de $p$, $s_i$ e $s_j$ eram adjacentes
    - Estamos admitindo posição geral, i.e., 3 segmentos não podem se interceptar em $p$
  - Considere o evento $q$ acontecido imediatamente antes de $p$
    - Necessariamente, logo após $q$, $s_i$ e $s_j$ são adjacentes



# Estruturas de Dados – Fila de Eventos

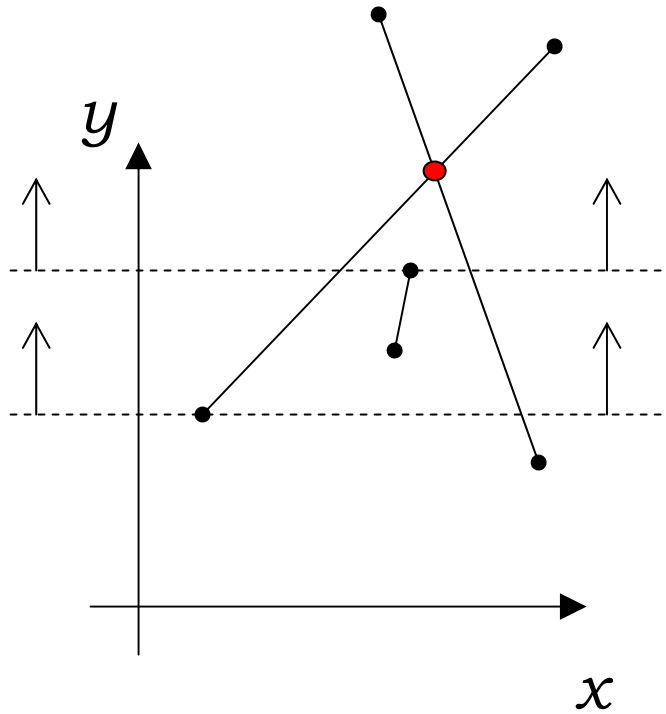
- Mantém os eventos (extremidades ou interseções) na ordem em que eles são encontrados pela linha de varredura:
  - Sejam $p$ e $q$ dois eventos na fila
  - Então $p$ precede $q$ sse
    - $p.y < q.y$ ou
    - $p.y = q.y$ e $p.x < q.x$
  - Pode-se pensar numa linha de varredura ligeiramente inclinada



- Cada evento contém
  - Posição $(x, y)$
  - Tipo (extremidade superior, inferior ou interseção)
  - Segmentos que participam no evento

# Estruturas de Dados – Fila de Eventos

- Operações
  - Retirar o evento seguinte
  - Inserir um evento
- Pode-se usar um *heap* , mas como eventos duplicados podem ocorrer, é melhor usar uma estrutura que suporte busca em tempo logarítmico, como uma árvore binária balanceada
  - Outra opção é usar um *heap* mas ao retirar o evento seguinte, testar se o próximo é uma duplicata. Caso positivo, este é descartado



# Estr. Dados – Estado da Linha de Varredura

- Indica os segmentos correntemente interceptando a linha de varredura ordenados por coordenada $x$ de interseção
- Operações
  - Inserir um segmento
  - Remover um segmento
  - Consultar os segmentos adjacentes a um dado segmento
- Usa-se uma estrutura que permita realizar essas operações em $O(\log n)$. Ex.:
  - Uma árvore binária de busca balanceada
  - Uma skip-list





# Algoritmo

- Inserir todos os eventos correspondentes a extremidades de segmentos na fila de eventos
- Repetir enquanto fila não vazia:
  - Retirar próximo evento da fila. Temos os casos
    - Extremidade Inferior de um segmento $s$
      - Inserir $s$ na linha de varredura
      - Computar interseções de $s$ com os segmentos adjacentes
    - Extremidade Superior de um segmento $s$
      - Testar a interseção dos 2 segmentos adjacentes
      - Retirar $s$ da linha de varredura
    - Ponto de interseção entre segmentos $s_i$ e $s_j$
      - Trocar $s_i$ e $s_j$ de posição na linha de varredura
      - Testar a interseção de $s_i$ e $s_j$ com seus novos vizinhos





# Complexidade

- Cada segmento é colocado na fila (2 vezes): $O(n \log n)$
- A cada momento, a fila pode ter no máximo $2n + I$ eventos e portanto inserir ou remover um evento tem complexidade $O(\log(2n + I)) = O(\log(2n + n^2)) = O(\log n)$
- A linha de varredura tem no máximo $n$ segmentos e portanto inserir ou remover um evento tem complexidade $O(\log n)$
- O processamento de um evento pode requerer até 2 testes de interseção, logo, se faz $O(2n + I)$ testes
- Portanto, o processamento todos os eventos tem complexidade $O((2n + I)(1 + \log n)) = O(n \log n + I \log n)$