

# Interseção entre Segmentos e entre Caixas Alinhadas

## INF2604 – Geometria Computacional

Waldemar Celes  
celes@inf.puc-rio.br

Departamento de Informática, PUC-Rio



# Interseção entre Segmentos



# Interseção de segmentos

## Problema

- ▶ Dado um conjunto de segmentos de reta  $S = \{s_1, \dots, s_n\}$ :
  - ▶ **Detecção**: determinar se há cruzamentos
  - ▶ **Identificação**: determinar quais são os cruzamentos



# Interseção de segmentos: **Caso base**

Interseção entre 2 segmentos: **ab** e **cd**

- ▶ Algoritmo de detecção e identificação
  - ▶ Determinar o ponto de cruzamento das retas definidas pelos segmentos
  - ▶ Verificar se o ponto está dentro de ambos os segmentos



# Interseção de segmentos: **Caso base**

Interseção entre 2 segmentos: **ab** e **cd**

- ▶ Algoritmo de detecção e identificação
  - ▶ Determinar o ponto de cruzamento das retas definidas pelos segmentos
  - ▶ Verificar se o ponto está dentro de ambos os segmentos

$$r_1 = \mathbf{a} + t(\mathbf{b} - \mathbf{a})$$

$$r_2 = \mathbf{c} + s(\mathbf{d} - \mathbf{c})$$



## Interseção de segmentos: **Caso base**

Interseção entre 2 segmentos: **ab** e **cd**

- ▶ Algoritmo de detecção e identificação
  - ▶ Determinar o ponto de cruzamento das retas definidas pelos segmentos
  - ▶ Verificar se o ponto está dentro de ambos os segmentos

$$r_1 = \mathbf{a} + t(\mathbf{b} - \mathbf{a})$$

$$r_2 = \mathbf{c} + s(\mathbf{d} - \mathbf{c})$$

Recai num sistema linear  $2 \times 2$ :

$$x_{\mathbf{a}} + t(x_{\mathbf{b}} - x_{\mathbf{a}}) = x_{\mathbf{c}} + s(x_{\mathbf{d}} - x_{\mathbf{c}})$$

$$y_{\mathbf{a}} + t(y_{\mathbf{b}} - y_{\mathbf{a}}) = y_{\mathbf{c}} + s(y_{\mathbf{d}} - y_{\mathbf{c}})$$



## Interseção de segmentos: **Caso base**

Interseção entre 2 segmentos: **ab** e **cd**

- Em forma matricial:

$$\begin{bmatrix} x_b - x_a & x_c - x_d \\ y_b - y_a & y_c - y_d \end{bmatrix} \begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} x_c - x_a \\ y_c - y_a \end{bmatrix}$$



# Interseção de segmentos: **Caso base**

Interseção entre 2 segmentos: **ab** e **cd**

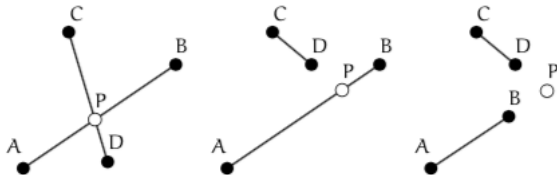
► Em forma matricial:

$$\begin{bmatrix} x_b - x_a & x_c - x_d \\ y_b - y_a & y_c - y_d \end{bmatrix} \begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} x_c - x_a \\ y_c - y_a \end{bmatrix}$$

► Verificação

$$t \in [0, 1]$$

$$s \in [0, 1]$$





## Interseção entre 2 de segmentos

Se for só **detecção**, pode ser mais eficiente?



## Interseção entre 2 de segmentos

Se for só **detecção**, pode ser mais eficiente?

- ▶ Os dois segmentos se cruzam se:
  - ▶ **c** e **d** estão em lados opostos de **ab**, e
  - ▶ **a** e **b** estão em lados opostos de **cd**
  - ▶ Lembrando:

$$orient < \mathbf{p}, \mathbf{q}, \mathbf{r} > = \begin{vmatrix} 1 & x_p & y_p \\ 1 & x_q & y_q \\ 1 & x_r & y_r \end{vmatrix} \begin{cases} > 0 & \text{se à esquerda} \\ < 0 & \text{se à direita} \end{cases}$$



## Interseção entre 2 de segmentos

Se for só **detecção**, pode ser mais eficiente?

- ▶ Os dois segmentos se cruzam se:
  - ▶ **c** e **d** estão em lados opostos de **ab**, e
  - ▶ **a** e **b** estão em lados opostos de **cd**
  - ▶ Lembrando:

$$orient < \mathbf{p}, \mathbf{q}, \mathbf{r} > = \begin{vmatrix} 1 & x_p & y_p \\ 1 & x_q & y_q \\ 1 & x_r & y_r \end{vmatrix} \begin{cases} > 0 & \text{se à esquerda} \\ < 0 & \text{se à direita} \end{cases}$$

Logo:

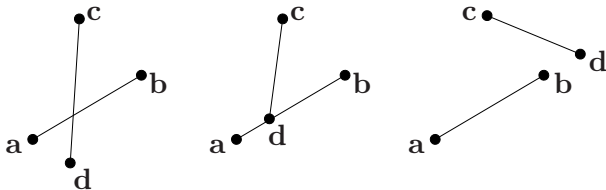
$$\begin{aligned} orient < \mathbf{a}, \mathbf{b}, \mathbf{c} > & orient < \mathbf{a}, \mathbf{b}, \mathbf{d} > < 0 \quad \text{and} \\ orient < \mathbf{c}, \mathbf{d}, \mathbf{a} > & orient < \mathbf{c}, \mathbf{d}, \mathbf{b} > < 0 \end{aligned}$$



## Detecção entre 2 de segmentos: casos degenerados

Conceito de “lados opostos” pode ser robusto

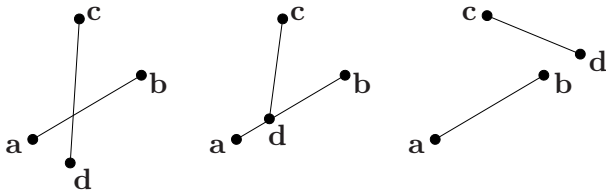
- ▶ Segmentos não colineares: condição  $\leq 0$  para tratar contato



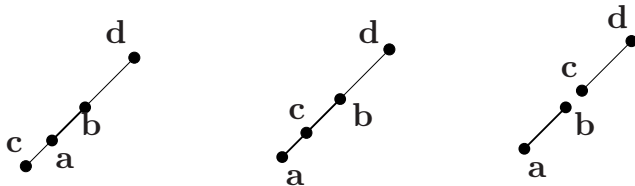
## Detecção entre 2 de segmentos: casos degenerados

Conceito de “lados opostos” pode ser robusto

- ▶ Segmentos não colineares: condição  $\leq 0$  para tratar contato



- ▶ Segmentos colineares: pontos em lados opostos (antes/depois)



# Interseção de segmentos

Caso geral:  $n$  segmentos

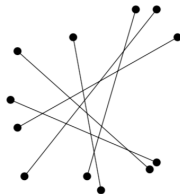
- ▶ Algoritmo força bruta:  $\binom{n}{2} = O(n^2)$



# Interseção de segmentos

Caso geral:  $n$  segmentos

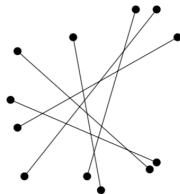
- ▶ Algoritmo força bruta:  $\binom{n}{2} = O(n^2)$
- ▶ Não temos como melhorar o limite inferior do tempo estimado
  - ▶ Pois podem existir  $n^2$  interseções



# Interseção de segmentos

Caso geral:  $n$  segmentos

- ▶ Algoritmo força bruta:  $\binom{n}{2} = O(n^2)$
- ▶ Não temos como melhorar o limite inferior do tempo estimado
  - ▶ Pois podem existir  $n^2$  interseções
- ▶ Por que buscar métodos melhores?
  - ▶ Podemos melhorar detecção
  - ▶ Na prática, podemos melhorar identificação





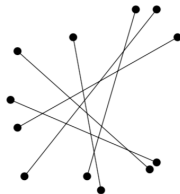
# Interseção de segmentos

Caso geral:  $n$  segmentos

- ▶ Algoritmo força bruta:  $\binom{n}{2} = O(n^2)$
- ▶ Não temos como melhorar o limite inferior do tempo estimado
  - ▶ Pois podem existir  $n^2$  interseções
- ▶ Por que buscar métodos melhores?
  - ▶ Podemos melhorar detecção
  - ▶ Na prática, podemos melhorar identificação

Tempos esperados ótimos:

- ▶ Detecção:  $O(n \log n)$
- ▶ Identificação:  $O(n \log n + \nu)$ 
  - ▶ Onde  $\nu$  representa o número de interseções existentes



# Interseção de $n$ segmentos

Solução: explorar geometria dos segmentos

- ▶ Testar apenas segmentos próximos



# Interseção de $n$ segmentos

Solução: explorar geometria dos segmentos

- ▶ Testar apenas segmentos próximos

Algoritmo de varredura

- ▶ Segmentos sem sobreposição na projeção em  $y$ , não se interceptam



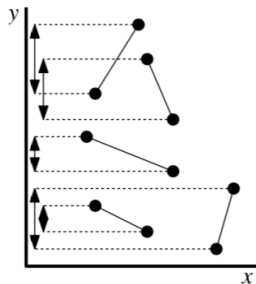
# Interseção de $n$ segmentos

Solução: explorar geometria dos segmentos

- ▶ Testar apenas segmentos próximos

Algoritmo de varredura

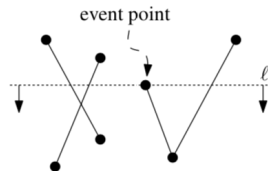
- ▶ Segmentos sem sobreposição na projeção em  $y$ , não se interceptam
- ▶ Simplificação (a princípio):
  - ▶ Não existem segmentos horizontais
  - ▶ Segmentos só se interceptam em um ponto
  - ▶ Não há interseção de 3 ou mais segmentos



# Interseção de $n$ segmentos

## Algoritmo de varredura

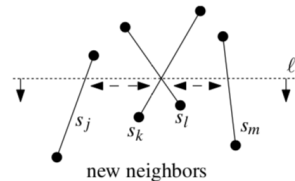
- ▶ Ordena segmentos em  $y$ 
  - ▶ Ordena pontos de início e fim de cada segmento
- ▶ Passa linha horizontal de varredura  $\ell$
- ▶ Mantém conjunto de segmentos ativos  $A$ 
  - ▶ Pontos dos segmentos são *eventos*
    - ▶ Ponto de início: segmento é inserido em  $A$
    - ▶ Ponto de fim: segmento é removido de  $A$
- ▶ Só precisa testar segmentos ativos
  - ▶ Mas ainda pode ser  $O(n^2)$



# Interseção de $n$ segmentos

## Algoritmo de varredura

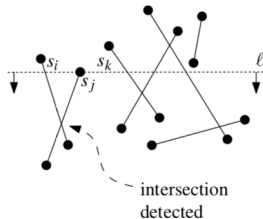
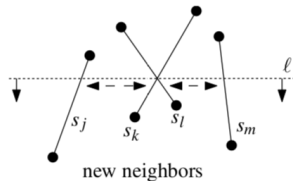
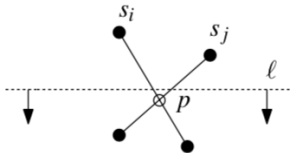
- ▶ Matém segmentos ordenados em  $A$ 
  - ▶ Ordenação por  $x$ , interseção do segmento com  $l$
  - ▶ Ponto de interseção passa a ser *evento*



# Interseção de $n$ segmentos

## Algoritmo de varredura

- ▶ Matém segmentos ordenados em  $A$ 
  - ▶ Ordenação por  $x$ , interseção do segmento com  $\ell$
  - ▶ Ponto de interseção passa a ser *evento*
  - ▶ Só precisa testar segmentos vizinhos imediatos
  - ▶ Só considera interseções abaixo de  $\ell$



# Interseção de $n$ segmentos

## Algoritmo de varredura: **Inicialização**

- ▶ Estrutura para armazenar pontos ordenados em  $y$ : *Fila de Eventos E*
  - ▶ Inserção de pontos iniciais e finais de segmentos
  - ▶ Inserção dinâmica de pontos de interseção
- ▶ Estrutura para armazenar segmentos ativos: *Conjunto de Ativos A*
  - ▶ Inserção/remoção ordenada
  - ▶ Acesso a vizinhos imediatos
  - ▶ Troca de ordem





# Interseção de $n$ segmentos

Algoritmo de varredura: **Tratamento de eventos**

## 1. Ponto de início de segmento

- ▶ Insere segmento em  $A$ , em ordem
- ▶ Verifica interseção com segmentos vizinhos
  - ▶ Reporta interseção, se for o caso
  - ▶ Insere interseção na fila de eventos, se for o caso

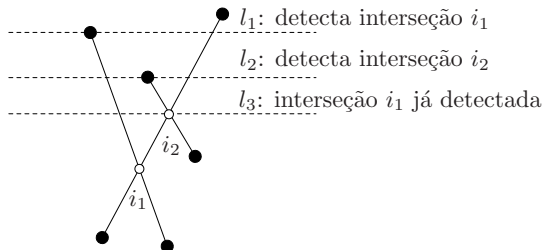


# Interseção de $n$ segmentos

Algoritmo de varredura: **Tratamento de eventos**

## 2. Ponto de interseção

- ▶ Troca ordem dos segmentos em  $A$
- ▶ Verifica interseção com segmentos vizinhos
  - ▶ Reporta interseção, se for o caso
  - ▶ Insere interseção na fila de eventos, se for o caso
  - ▶ Evitar re-inserção de evento já detectado



# Interseção de $n$ segmentos

Algoritmo de varredura: **Tratamento de eventos**

## 3. Ponto de fim de segmento

- ▶ Remove segmento em  $A$ , em ordem
- ▶ Verifica interseção entre vizinhos
  - ▶ Reporta interseção, se for o caso
  - ▶ Insere interseção na fila de eventos, se for o caso



# Interseção de $n$ segmentos

Algoritmo de varredura: **Tratamento de eventos**

## 3. Ponto de fim de segmento

- ▶ Remove segmento em  $A$ , em ordem
- ▶ Verifica interseção entre vizinhos
  - ▶ Reporta interseção, se for o caso
  - ▶ Insere interseção na fila de eventos, se for o caso

**Tempo esperado do algoritmo**

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



## Interseção entre Caixas Alinhadas



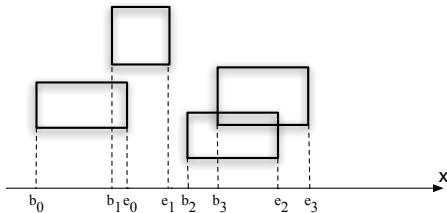
# Interseção entre Caixas Alinhadas

Ordenação e varredura

## Caso 1D

Ordenação:

Ordena pontos extremos ( $b$  &  $e$ )



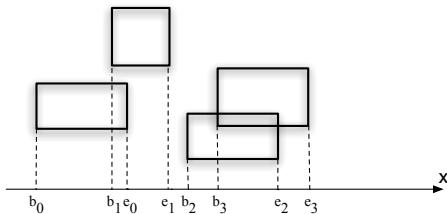
# Interseção entre Caixas Alinhadas

## Ordenação e varredura

### Caso 1D

#### Ordenação:

Ordena pontos extremos ( $b$  &  $e$ )



#### Varredura:

Conjunto de objetos ativos:  $S = \emptyset$

- ▶ Se achar  $b_i$ :
  - ▶ Reporta pares  $[o_i, s_j], \forall s_j \in S$
  - ▶  $S = S \cup \{o_i\}$
- ▶ Se achar  $e_i$ :
  - ▶  $S = S - \{o_i\}$

# Ordenação e varredura

## Extensão para 2D e 3D

- ▶ Ordena um vetor para cada dimensão
- ▶ Teste para reportar caixas colisão
  - ▶ Número de sobreposições tem que ser igual à dimensão

Implementação:

- ▶ Usa-se tabela para contar sobreposições entre objetos
- ▶ Ao encontrar sobreposição, acrescenta-se contador
- ▶ Se contador chegar a 3 (caso 3D), reporta-se par

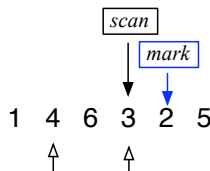
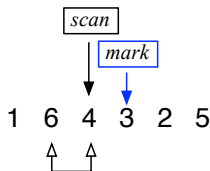




# Ordenação e varredura em cenas dinâmicas

## Ordenação

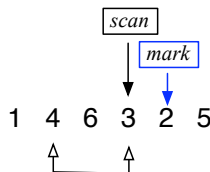
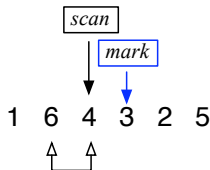
- Complexidade convencional:  $O(n \log n)$
- Se conjunto estiver quase ordenado, usando *insert-sort*:  $O(n)$



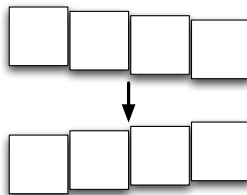
# Ordenação e varredura em cenas dinâmicas

## Ordenação

- ▶ Complexidade convencional:  $O(n \log n)$
- ▶ Se conjunto estiver quase ordenado, usando *insert-sort*:  $O(n)$



Contra-exemplo de coerência ( $O(n^2)$ ):



# Ordenação e varredura em cenas dinâmicas

Manutenção de lista  $R$  de pares em colisão

- ▶ Atualizar  $R$  no *insert-sort*
  - ▶ Se antes  $e_j < b_i$  e agora  $e_j > b_i$ 
    - ▶ Incrementa contador, se chegar a 3, insere par em  $R$
  - ▶ Se antes  $e_j > b_i$  e agora  $e_j < b_i$ 
    - ▶ Decrementa contador, se era 3, retira par de  $R$
  - ▶ Troca entre  $b_i$  e  $b_j$  ou entre  $e_i$  e  $e_j$ 
    - ▶ Nada a fazer



# Pseudo-código: Ordenação e varredura

```

update (L)
  scan = head(L)
  while L unsorted do
    mark = right(scan)
    left = left(scan)
    while scan < left do
      if scan = bi and left = ej then
        incr. counter (i,j)
        if counter(i,j) = 3 then
          report overlap (i,j)
        end if
      else if scan = ej and left = bi then
        decr. counter (i,j)
        if counter(i,j) = 2 then
          remove overlap (i,j)
        end if
      endif
      scan = left
      left = left(scan)
    end while
    scan = mark
  end while

```

