Geometria Computacional

Laboratório de Programação Competitiva I - 2020

Pedro Henrique Paiola (paiola@fc.unesp.br)
Giulia Moura Crusco (giulia@fc.unesp.br)
João Pedro Marin Comini (joaocomini@gmail.com)
Unesp Bauru

Geometria Computacional

- Problemas envolvendo geometria costumam ser o calcanhar de Aquiles dos competidores.
- Requisito importante para resolver problemas de geometria: domínio dos fundamentos de geometria.
- Geometria Analítica e Álgebra Linear.
- Inicialmente, implementar as soluções também parece uma tarefa complexa.

Geometria Computacional

- Características gerais de problemas geométricos:
  - Entrada de dados: pontos, vetores, retas, polígonos...
  - Objetos geométricos definidos por coordenadas ou equações.
  - Originalmente são contínuos, mas possuem representações discretas. OBS: normalmente vamos lidar com floats e doubles, e, consequentemente, com os erros de aproximação decorrentes.

Tipos de problemas geométricos

- **Seletivos**: busca-se selecionar um subconjunto da entrada, descobrindo possivelmente relações topológicas (adjacência entre os elementos da geometria). Exemplos:
    - Fecho convexo (Convex Hull)
    - Triangulação
    - Árvore geradora mínima

Tipos de problemas geométricos

- **Construtivos**: procura-se construir novos objetos geométricos, além das relações topológicas entre eles. Exemplos:
    - Intersecção de polígonos
    - Círculo mínimo
    - Diagrama de Voronoi
    - Geração de malhas
    - Suavização de curvas e superfícies

Tipos de problemas geométricos

- **Decisão**: tem-se como objetivo decidir se certo objeto satisfaz ou não certa(s) propriedade(s). Exemplos:
    - Decidir de um polígono é convexo
    - Verificar se um ponto pertence a um polígono
    - Verificar se um ponto está no interior de um polígono

Tipos de problemas geométricos

- **Consulta**: procura-se processar os objetos para efetuar consultas repetidas de forma eficiente. Exemplos:
    - Determinar dentre um conjunto de pontos aquele que se encontra mais próximo de um ponto específico
    - Qual o par de pontos mais próximos

Abordagem para solução

1. Definição matemática do problema
2. Definição precisa do que é resolver o problema algoritmicamente
3. Identificação de teoremas que ajudem na solução algorítmica
4. Representação computacional do problema
5. Estudo de soluções algorítmicas para o problema
6. Identificação de primitivas geométricas adequadas
7. Identificação e tratamento de casos especiais/degenerados
8. Análise de desempenho

Técnicas normalmente usadas

- Técnicas convencionais
    - Dividir para conquistar
    - Programação dinâmica
- Técnicas próprias para algoritmos geométricos
    - Varredura (line/plane sweep)
    - Construções randomizadas incrementais
    - Transformações duais

Ponto e Vetor

- Um **ponto** determina uma posição no espaço, ele não possui volume, área ou comprimento. Em um plano cartesiano, podemos representar um ponto por suas coordenadas x e y.
- Um **vetor** no plano R^2 é uma classe de objetos matemáticos (segmentos) com a mesma direção, sentido e módulo. Um vetor pode ser definido pelas coordenadas da extremidade e da origem:

Origem:	(x_i, y_i)
Extremidade:	(x_f, y_f)
Vetor:	$v = (x_f, y_f) - (x_i, y_i) = (x_f - x_i, y_f - y_i)$

Ponto e Vetor

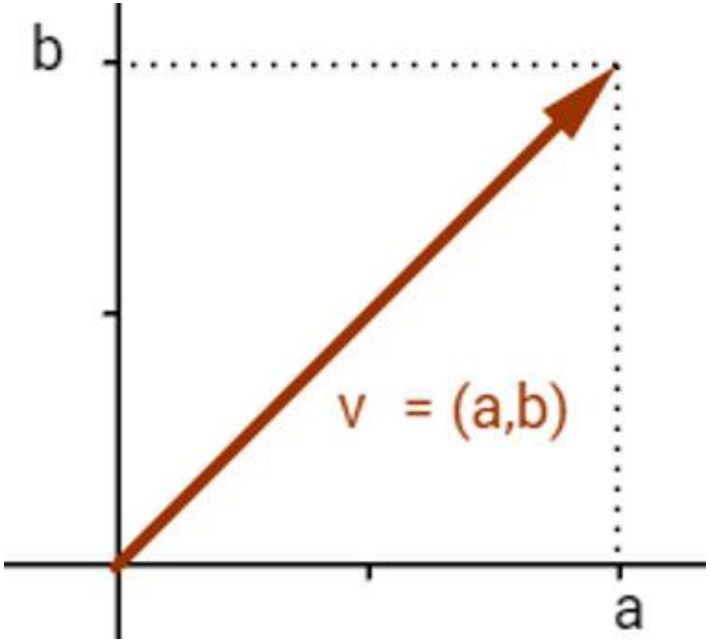
- Para a representação de pontos e vetores pode ser conveniente implementar uma classe (ou struct), na qual podemos incluir métodos úteis, inclusive com sobrecarga de operadores. O intuito é trabalhar com estes objetos geométricos com um pouco mais de facilidade, em um nível um pouco mais abstrato.

```
struct Point{
    double x;
    double y;
}
```

Ponto e Vetor

- Módulo ou comprimento de um vetor v = (a, b)

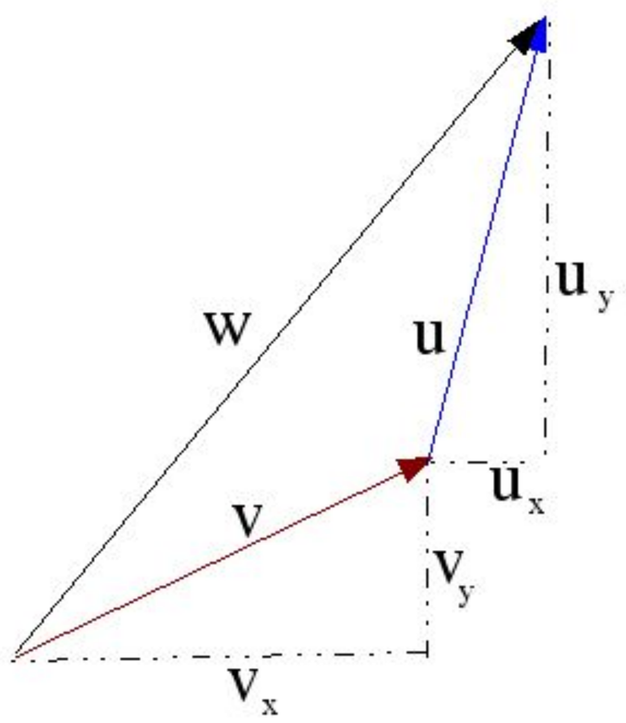
$$|v| = \sqrt{a^2 + b^2}$$



Operações com vetores

- Soma de vetores

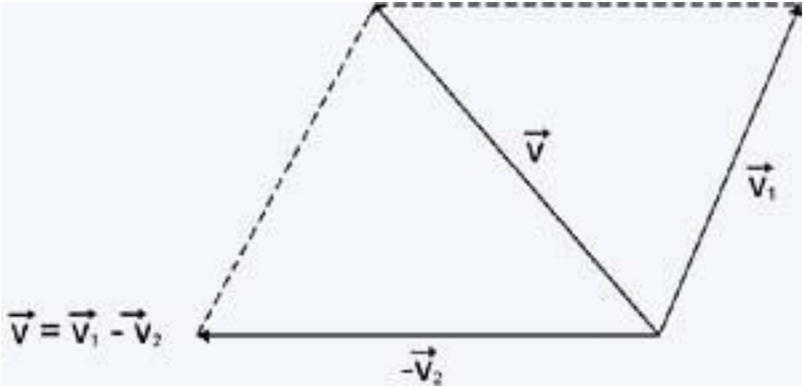
$$v = (v_x, v_y)$$
$$u = (u_x, u_y)$$
$$w = v + u = (v_x + u_x, v_y + u_y)$$



Operações com vetores

- Diferença de vetores: soma pelo oposto

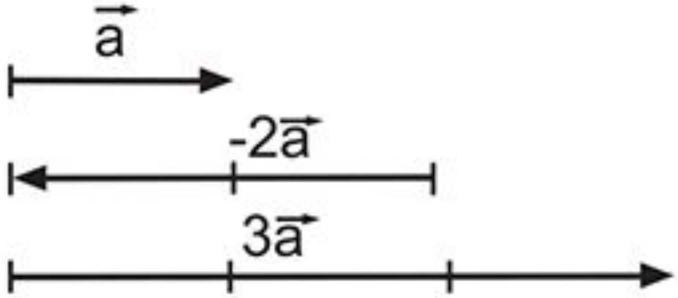
$$w = v - u = v + (-u)$$
$$w = (v_x - u_x, v_y - u_y)$$



Operações com vetores

- Multiplicação por escalar

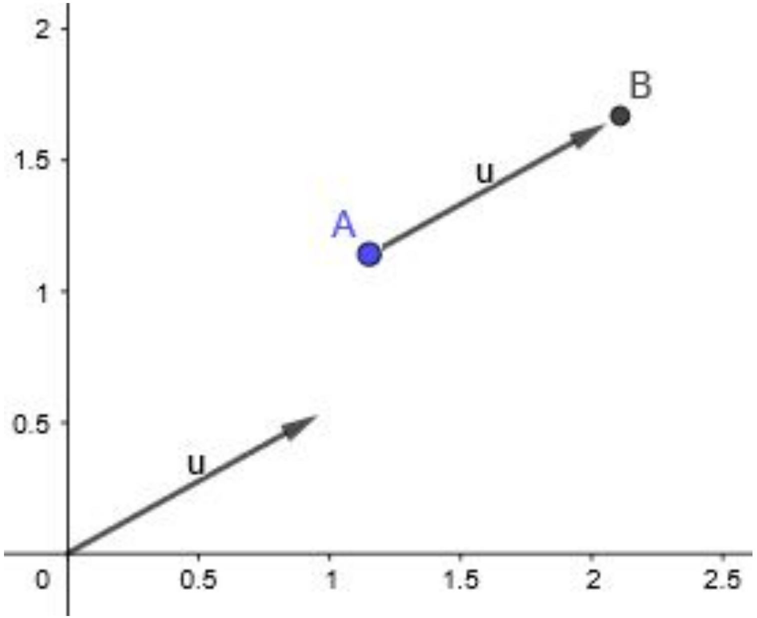
$$u = k.v = (k.v_x, k.v_y)$$



Operações com vetores

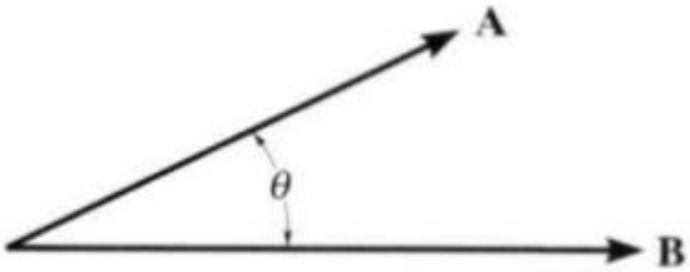
- Ponto + vetor: a soma de um ponto A e um vetor v resulta em ponto B

$$A + u = (A_x + u_x, A_y + u_y)$$



Operações com vetores

- Produto escalar (dot)



$$\boxed{\mathbf{A}\cdot\mathbf{B} = AB\cos\theta}$$

$$\boxed{\mathbf{A}\cdot\mathbf{B} = A_xB_x + A_yB_y}$$

Operações com vetores

- Produto escalar (dot)

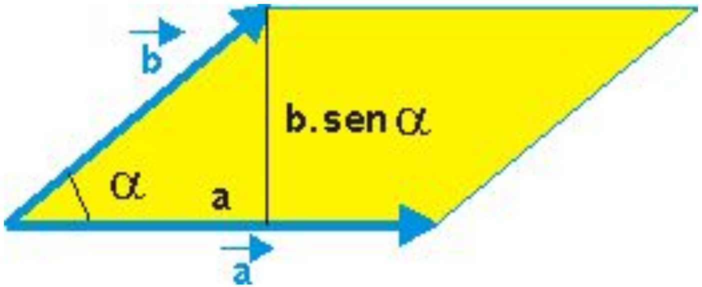
- Observação:
  - Se $v.u = 0 \Rightarrow \theta = 90^\circ$
  - Se $v.u > 0 \Rightarrow \theta < 90^\circ$
  - Se $v.u < 0 \Rightarrow \theta > 90^\circ$

Operações com vetores

- Produto vetorial (cross): o produto vetorial costuma ser definido para vetores no $\mathbf{R}^3$, cujo o resultado de v x u é um **vetor ortogonal** ao plano determinado pelos vetores v e u, e |v x u| pode ser interpretado como a **área do paralelograma** definido por v e u.

$$\vec{u} \times \vec{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$$

$$|v \times u| = |v||u| \sin\theta$$

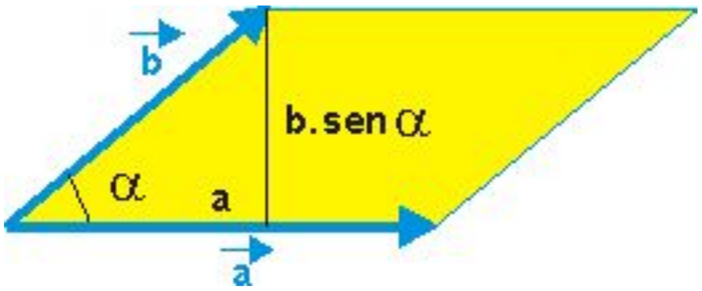


Operações com vetores

- Produto vetorial (cross): já no plano, obtemos um valor escalar, definido a seguir, que também representa a **área do paralelogramo** definido pelos vetores.

$$v \times u = \begin{vmatrix} v_x & v_y \\ u_x & u_y \end{vmatrix} = v_x . u_y - u_x . v_y$$

$$|v \times u| = |v||u| \sin \theta$$



Operações com vetores

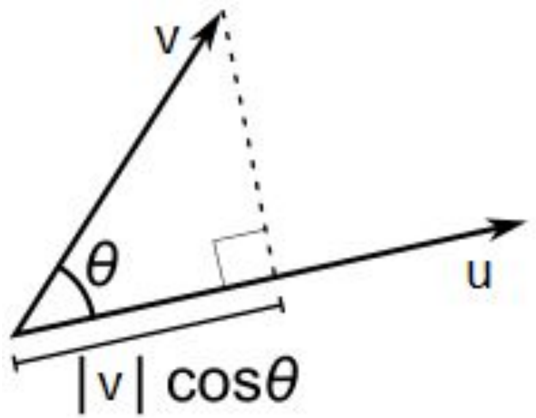
- Produto vetorial (cross):

- Observação:
  - Se v x u = 0 => vetores colineares
  - Se v x u > 0 => vetor u à esquerda de v (sentido anti-horário)
  - Se v x u < 0 => vetor u à direita de v (sentido horário)

Operações com vetores

- Projeção

$$proj_u v = \left(\frac{v.u}{|u|^2}\right) u$$



Representação de outros objetos

- Segmento de reta

```
struct Segment{
    Point p1, p2;
}
```

- Polígono

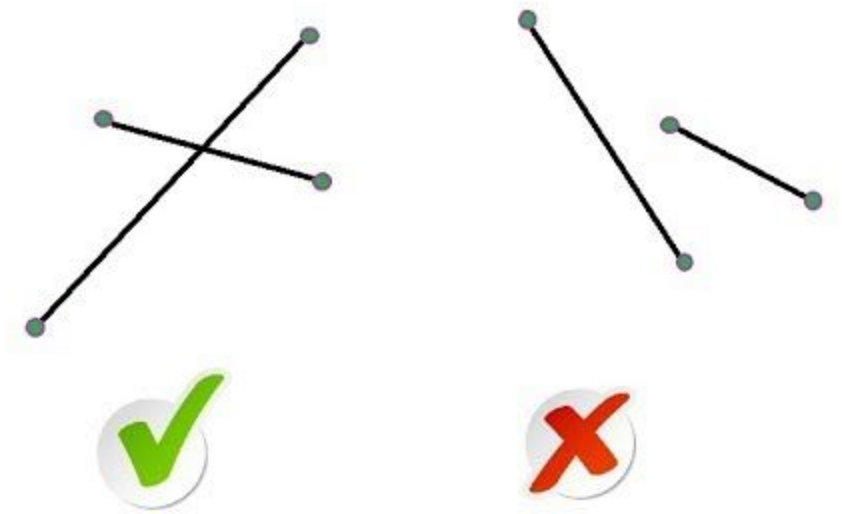
```
struct Polygon{
    int n;                    //Número de pontos no polígono
    Point p[MAX_POLY];        //Vetor de pontos (ordenados)
}
```

Problema de Intersecção

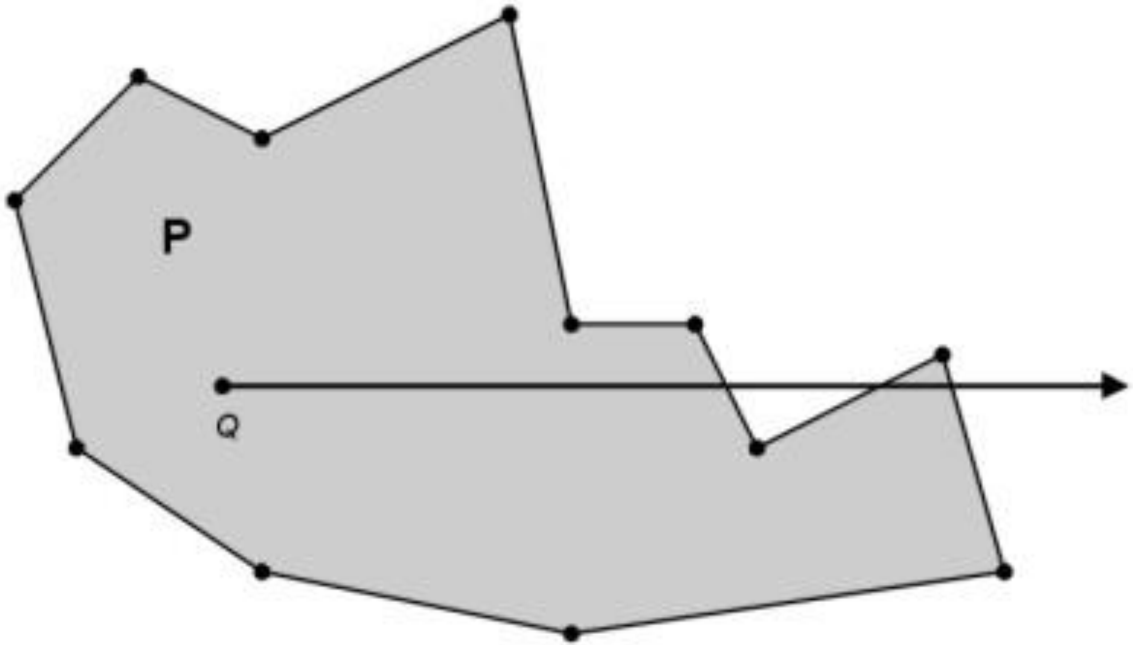
- Esse problema consiste em, dados dois ou mais objetos:
    - Determinar se eles se interceptam (predicado)
    - Determinar qual sua intersecção (objeto ou objetos na intersecção)
- Os dois problemas são relacionados mas NÃO são idênticos
    - Para determinar se 2 segmentos de reta se interceptam, basta fazer 4 testes de orientação
    - Para determinar o ponto de intersecção resolve-se um sistema de equações

Intersecção entre segmentos

- É comum precisarmos descobrir se dois segmentos de reta se intersectam, disponibilizamos uma função para isso no código disponibilizado no Moodle.
- Tal verificação se baseia, principalmente, no produto vetorial.
- Se for necessário descobrir qual o ponto em que ocorre essa intersecção, podemos descobrir o ponto de intersecção entre as duas retas subjacentes e verificar se esse ponto pertence aos dois segmentos.

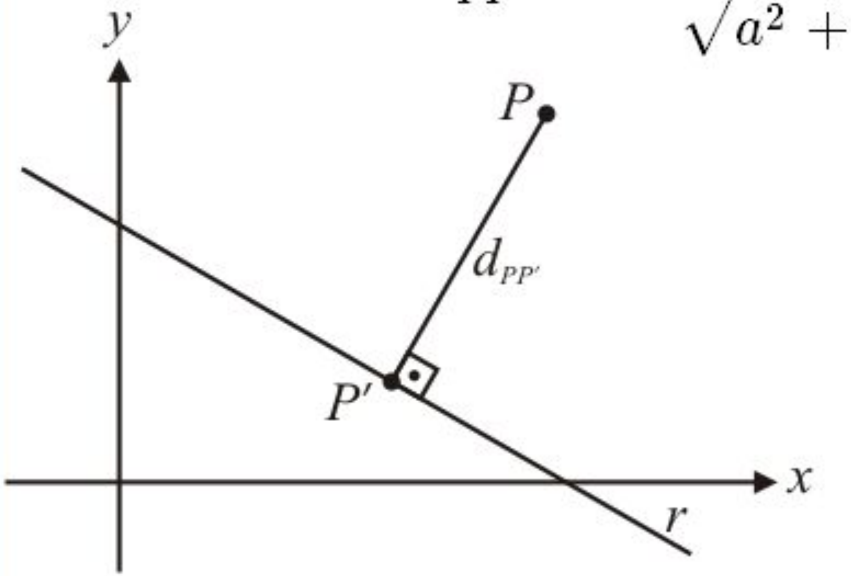


Ponto no interior de polígono



Distância de ponto a reta

$$d_{PP'} = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

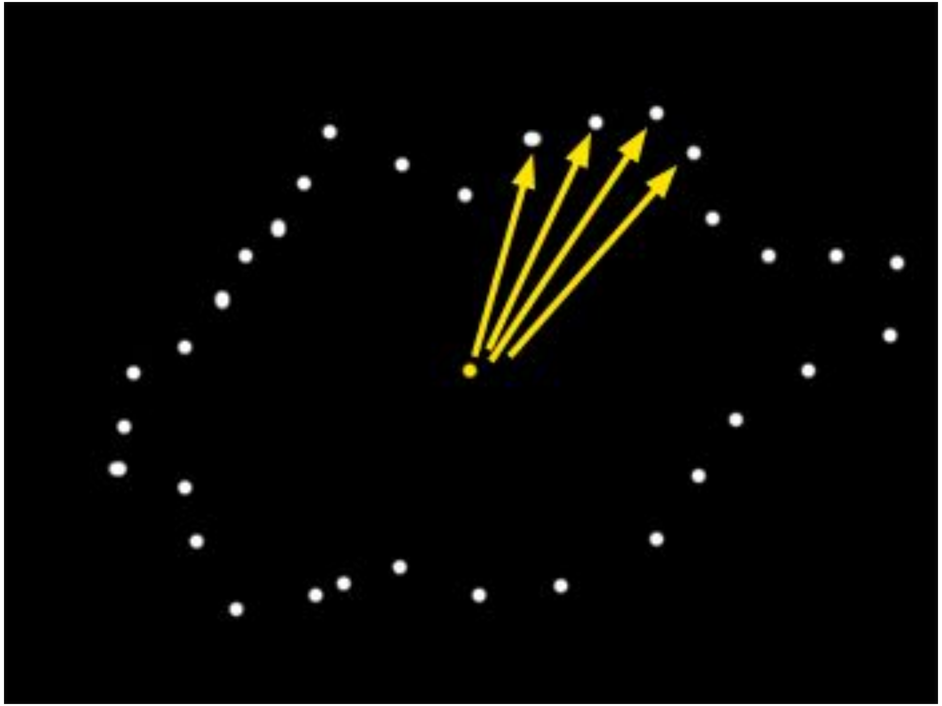


Distância de ponto a segmento

- Primeiro, descobrimos qual o ponto da reta subjacente ao segmento que está mais próximo do nosso ponto.
- Se esse ponto está dentro do segmento, a menor distância do ponto à reta é a menor distância do ponto ao segmento.
- Senão, a menor distância do ponto ao segmento é o menor valor dentre as distâncias do ponto as extremidades do segmento.

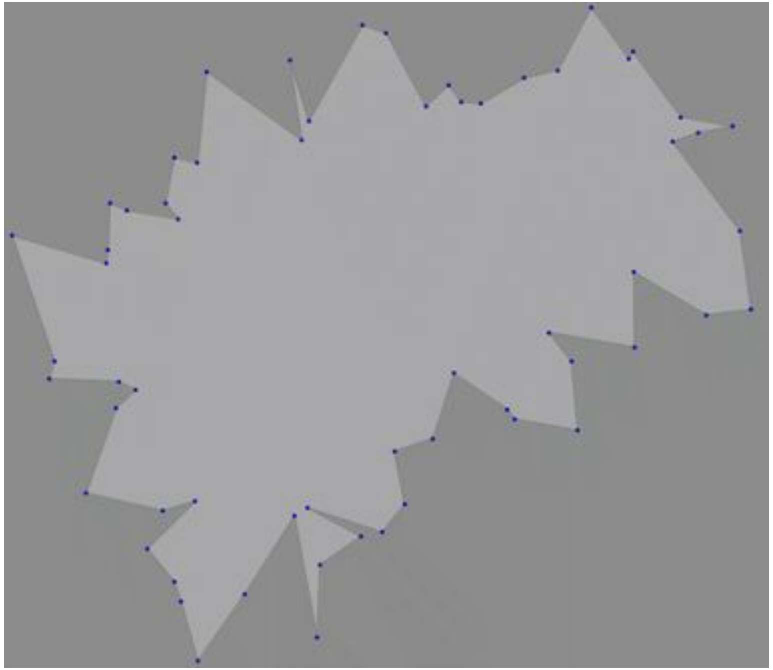
Radial Sort (Ordenação angular)

- Para alguns problemas é necessário ordenar os pontos de forma conveniente.
- Uma forma usual de fazer isso é ordenar os pontos no sentido horário ou anti-horário a partir de um ponto central.
- Isso pode ser feito a partir dos ângulos formados (a função atan2 é particularmente útil) ou utilizando o produto vetorial.



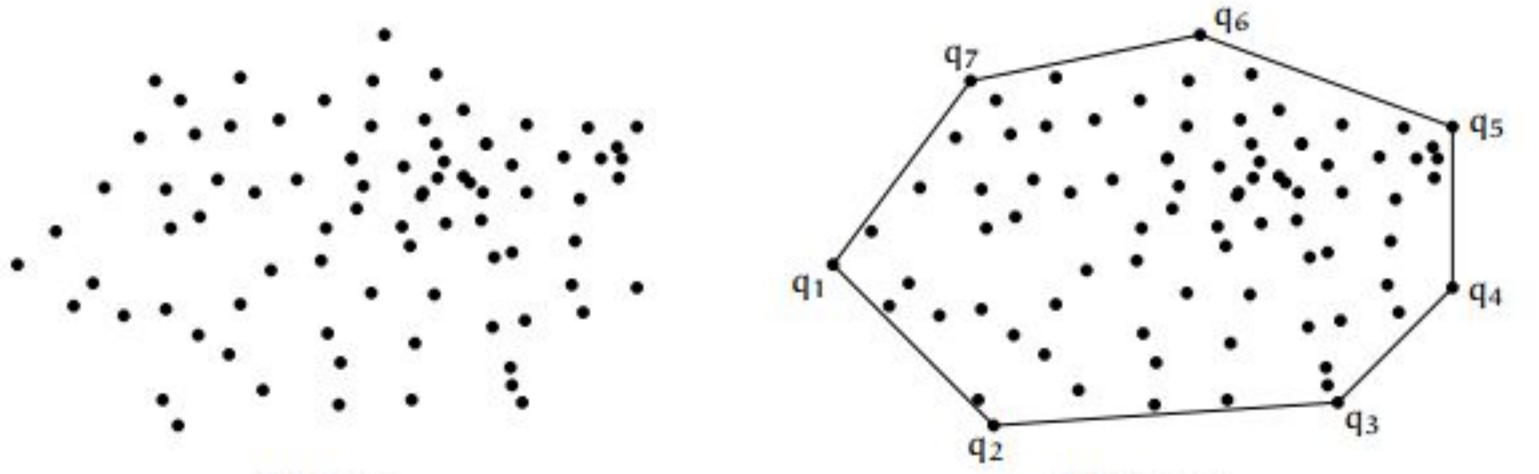
Radial Sort (Ordenação angular)

- Exemplo de uso: desenhar um polígono côncavo a partir de uma nuvem de pontos.



Convex Hull

- O fecho convexo (convex hull) de um conjunto de pontos é o menor polígono convexo que contém todo o conjunto de pontos.



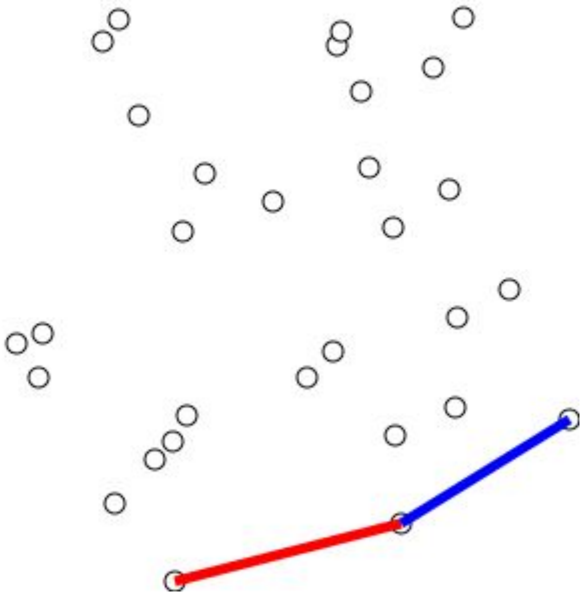
(a) Input.   (b) Output.

Convex Hull

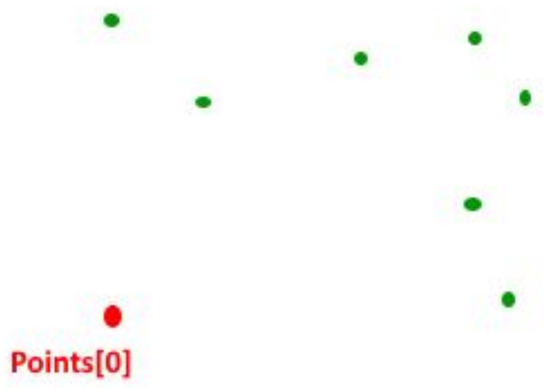
- Motivação:
    - O fecho convexo de um conjunto de pontos é uma aproximação simples
    - Necessariamente, não ocupa mais espaço do que o próprio conjunto de pontos
        - No pior caso, o polígono tem o mesmo número de vértices que o próprio conjunto
    - Computar o fecho convexo muitas vezes é um passo que precede outros algoritmos sobre conjuntos de pontos

Convex Hull

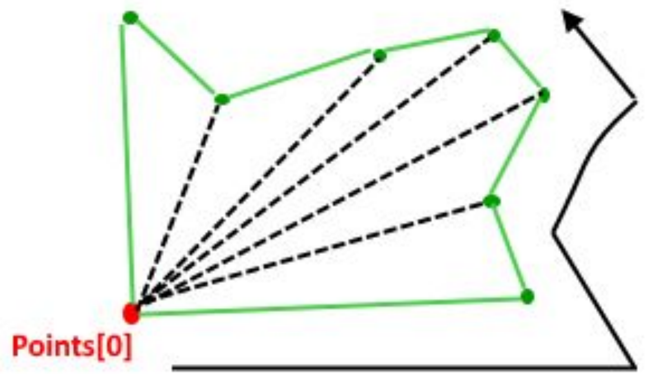
- Algoritmo (Graham Scan):
  - Escolher um ponto extremo p (ponto mais a esquerda, por exemplo)
  - Ordenar os outros pontos com o radial sort (com p como centro e no sentido anti-horário)
  - Percorremos os pontos ordenados:
    - Para cada tupla (p - anterior, c - atual, n - próximo), o ponto c só entra no fecho convexo se a orientação desses pontos for no sentido anti-horário.
    - Se c for rejeitado, damos um passo para trás



Convex Hull

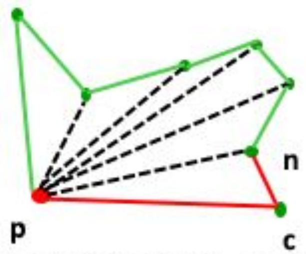


Given points

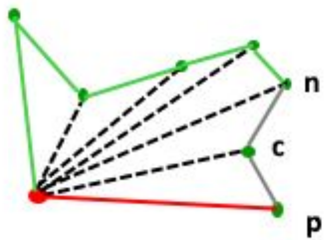


Points consider according to increasing angle
with respect to points[0] from a simple closed path

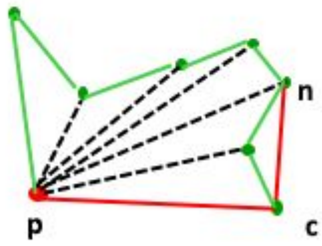
Convex Hull



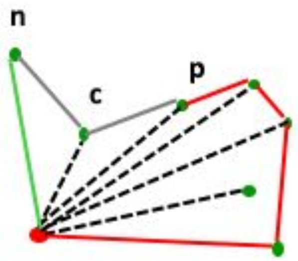
(p,c,n) is a left turn , accept c



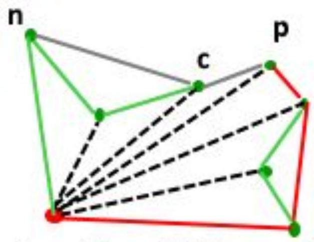
(p,c,n) is a right turn , reject c



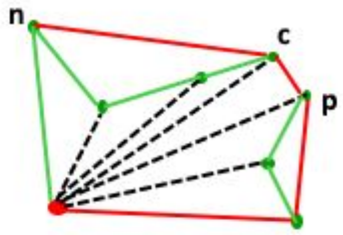
(p,c,n) is a left turn,accept c



(p,c,n) is a left turn, accept c



(p,c,n) is a right turn , reject c

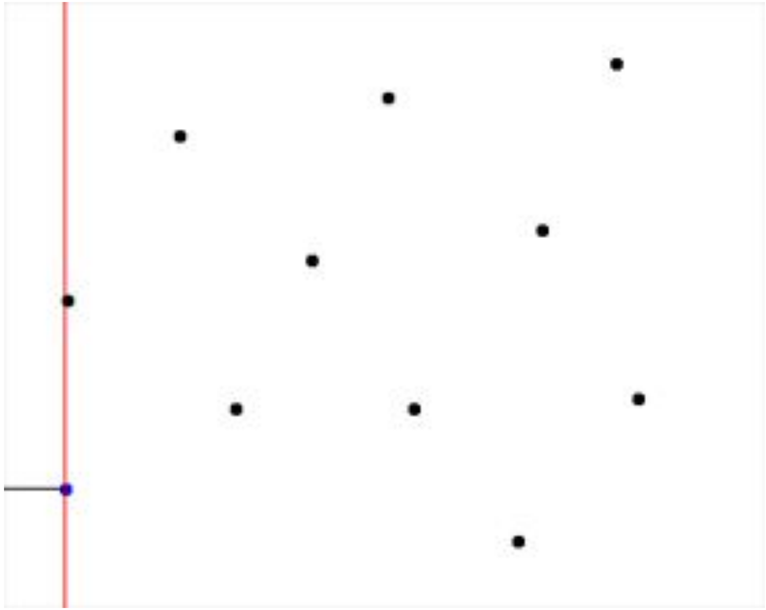


(p,c,n) is a left turn , accept c

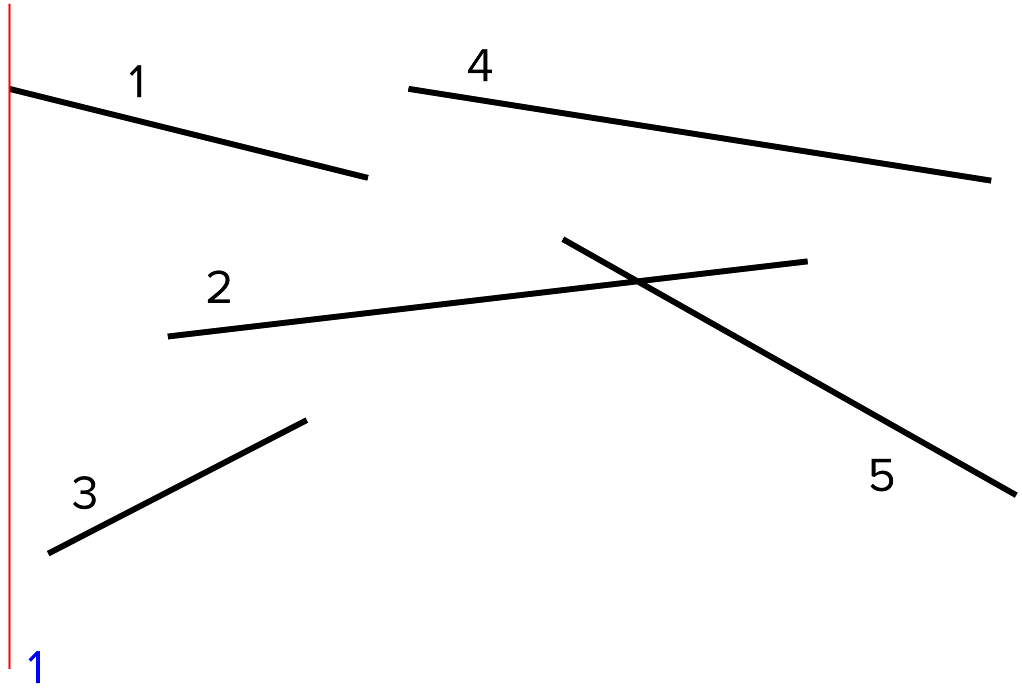
P : previouse c : current n : next

Line Sweep

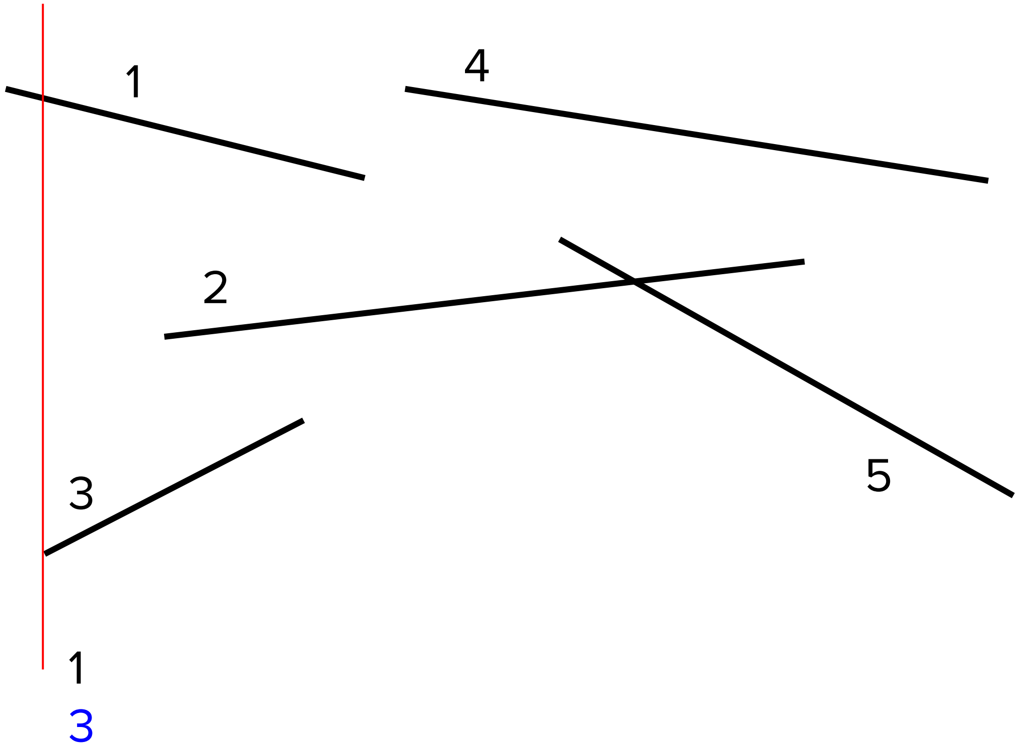
- O Line Sweep é uma técnica de varrer os pontos a partir de uma reta (normalmente vertical) que pode ser aplicado com diversas finalidades.



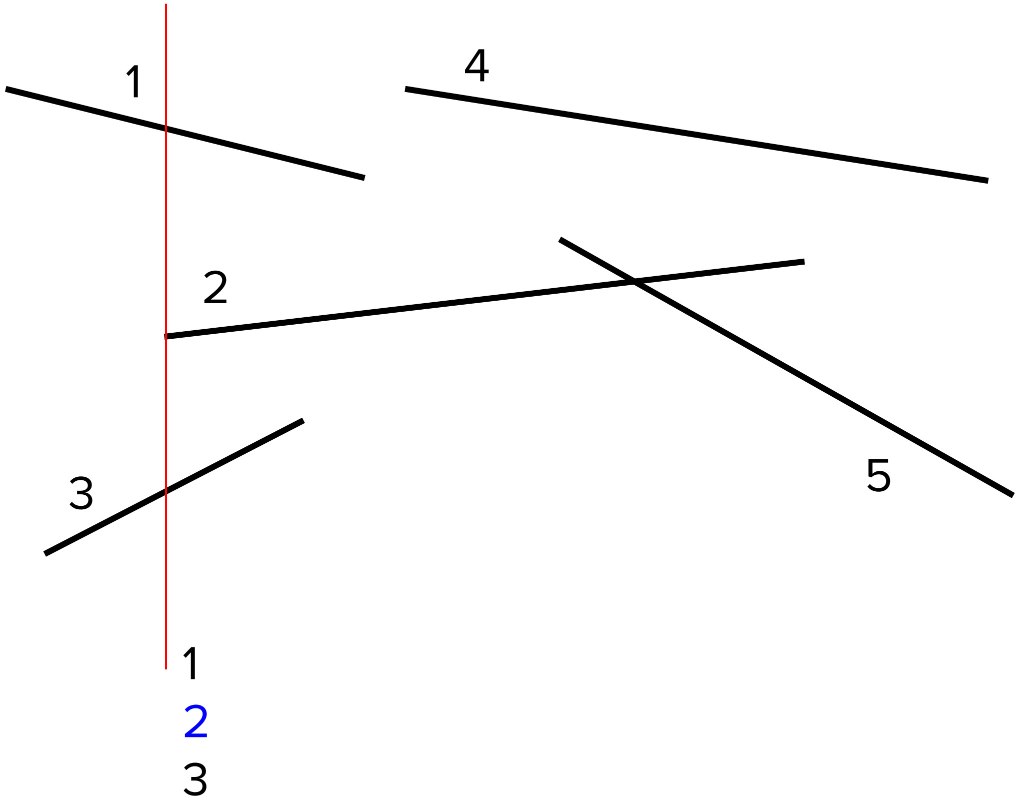
Line Sweep



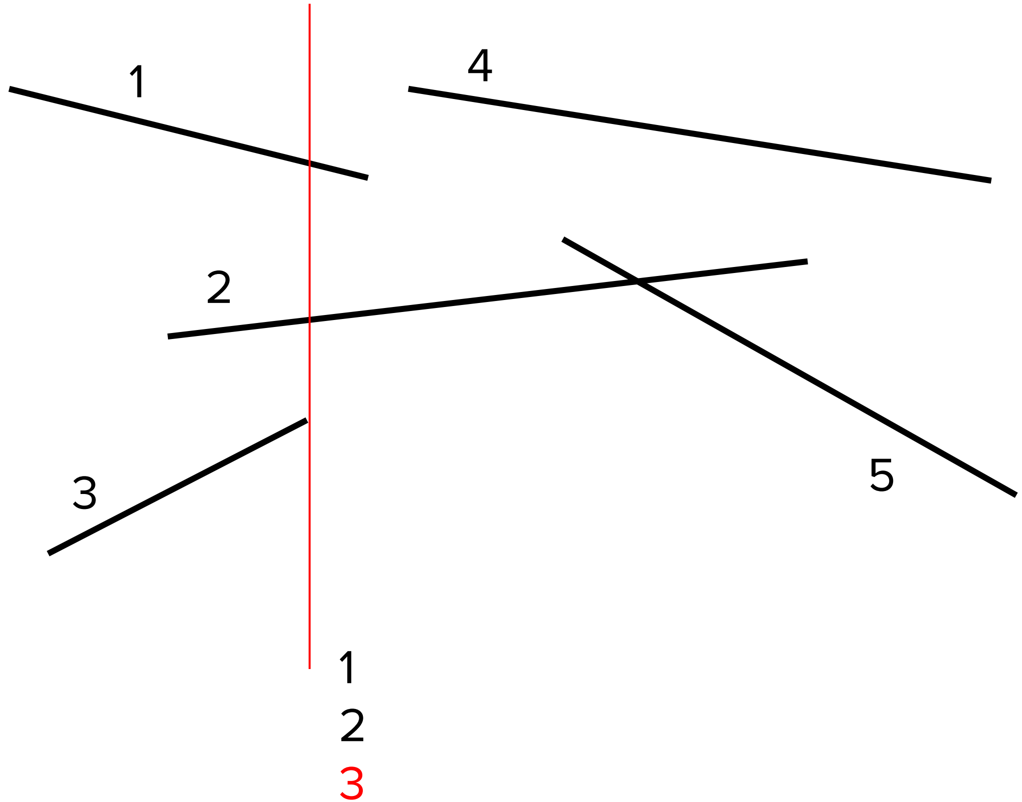
Line Sweep



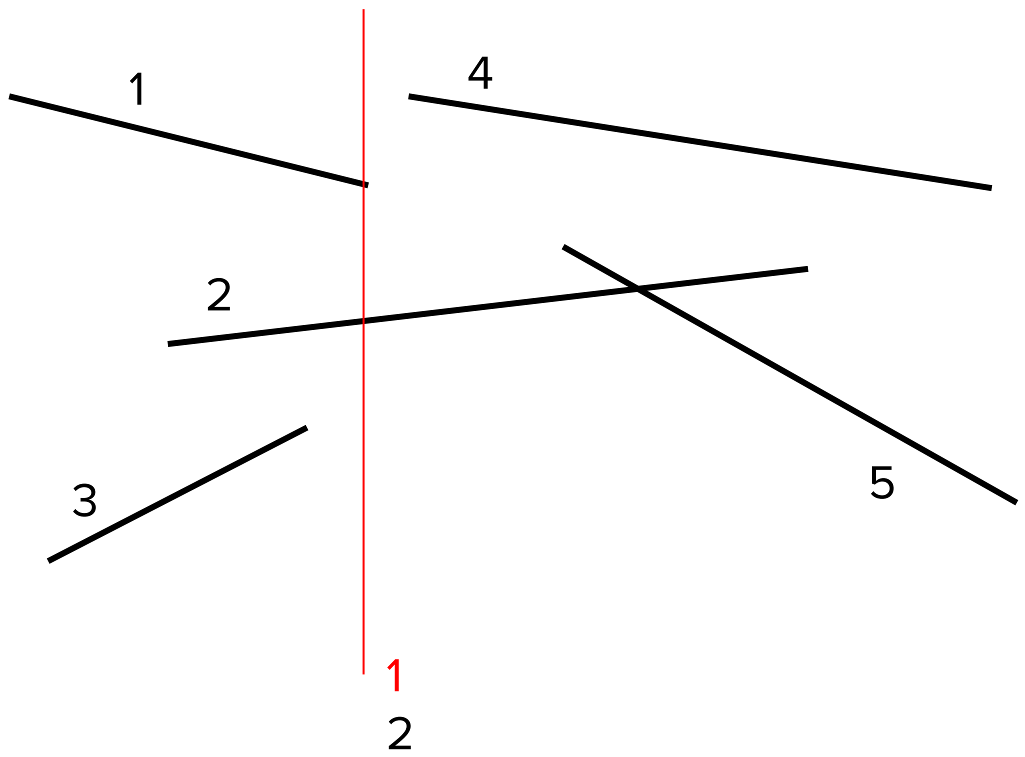
Line Sweep



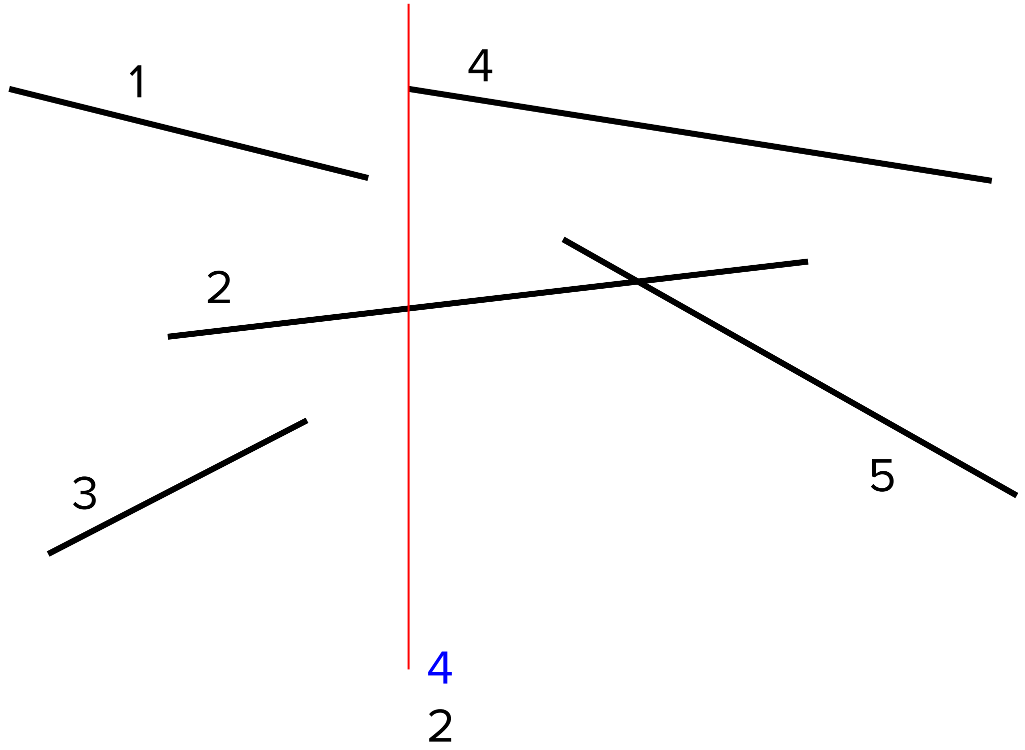
Line Sweep



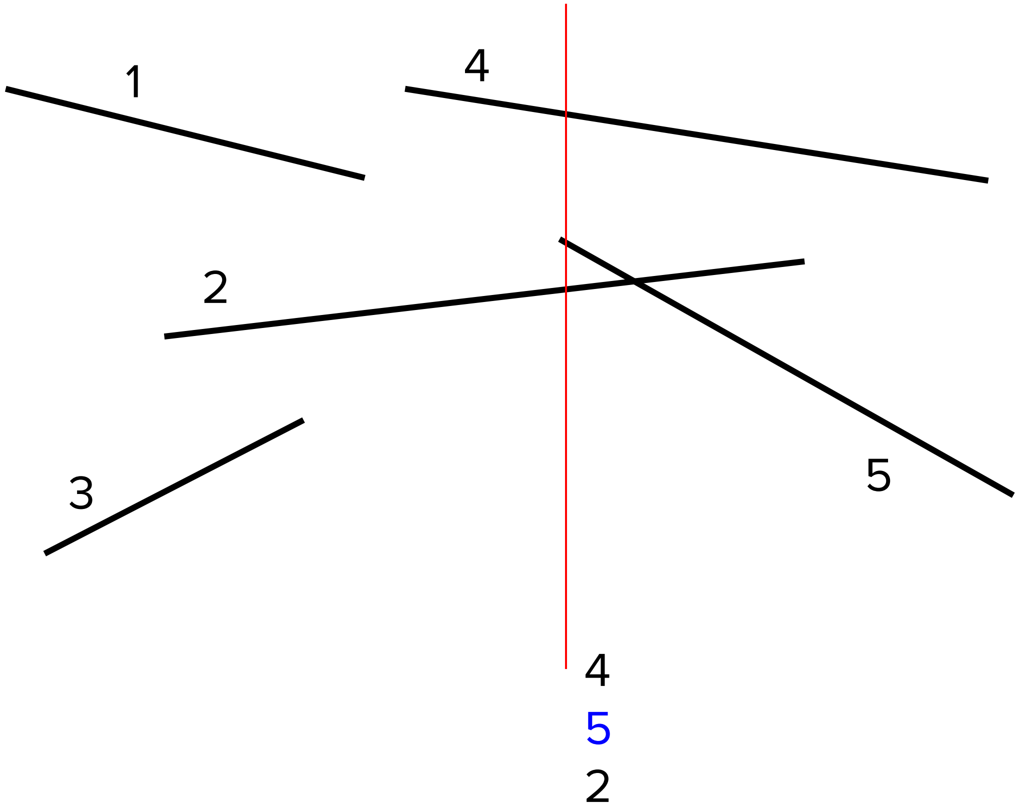
Line Sweep



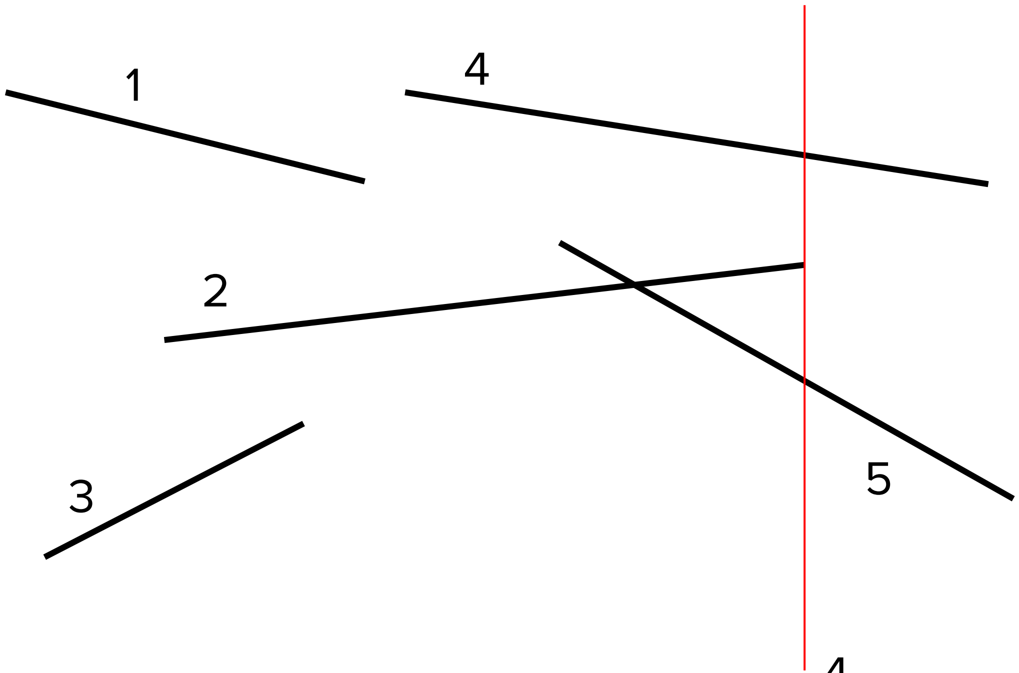
Line Sweep



Line Sweep

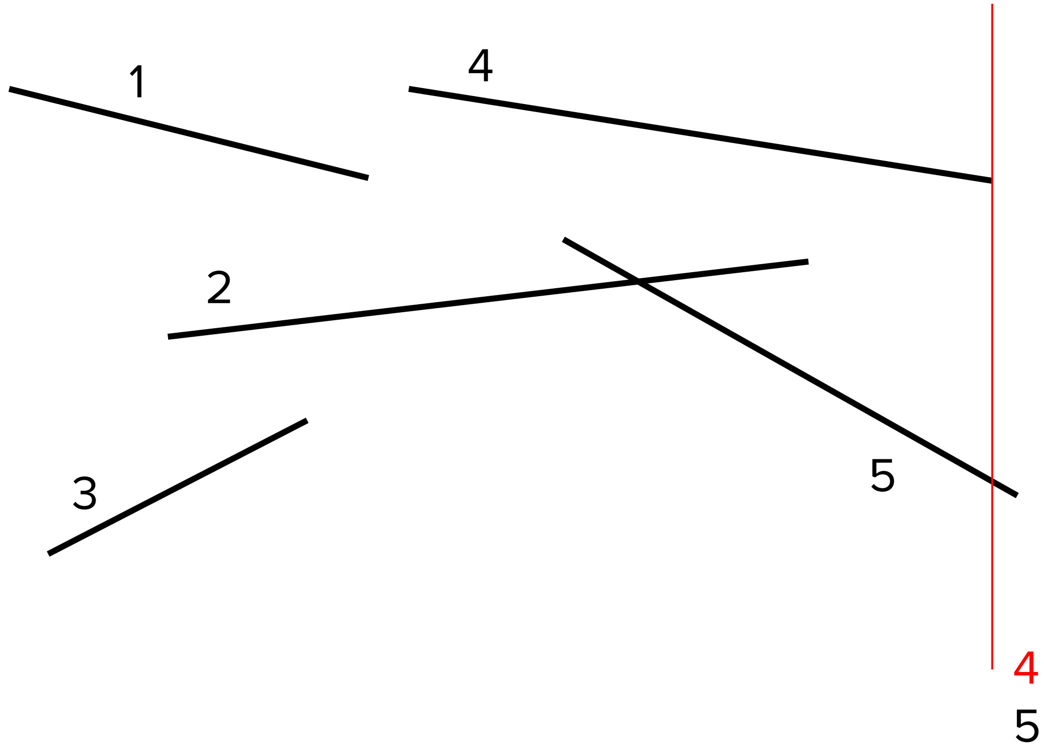


Line Sweep

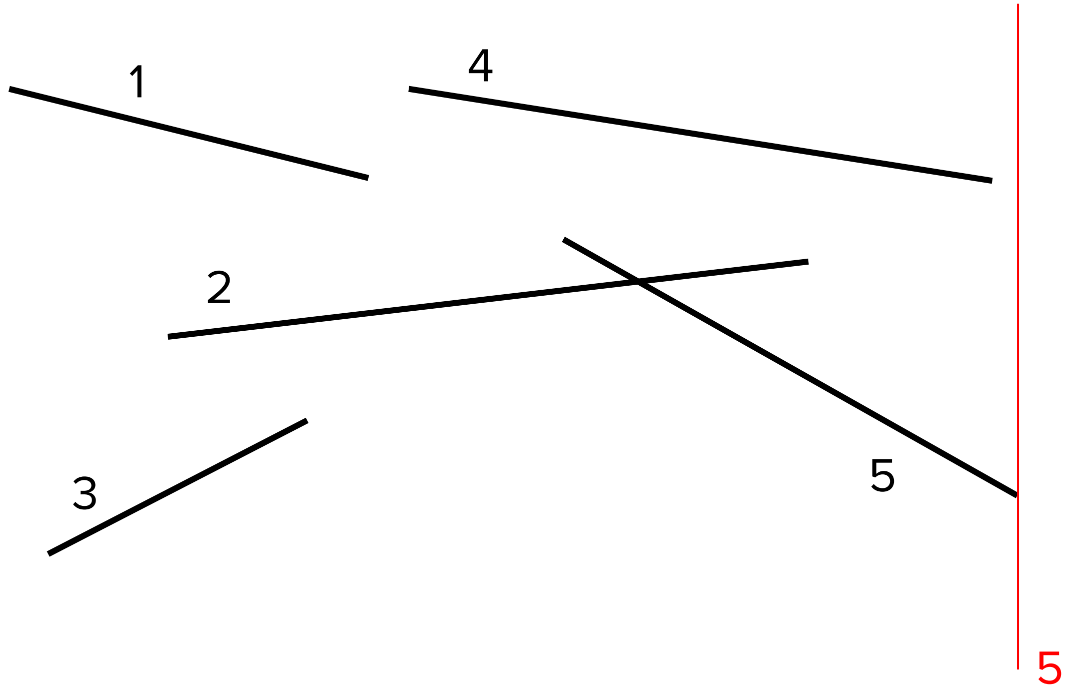


4
2 Houve inversão, entre
5 2 e 5 => Intersecção

Line Sweep



Line Sweep



Line Sweep

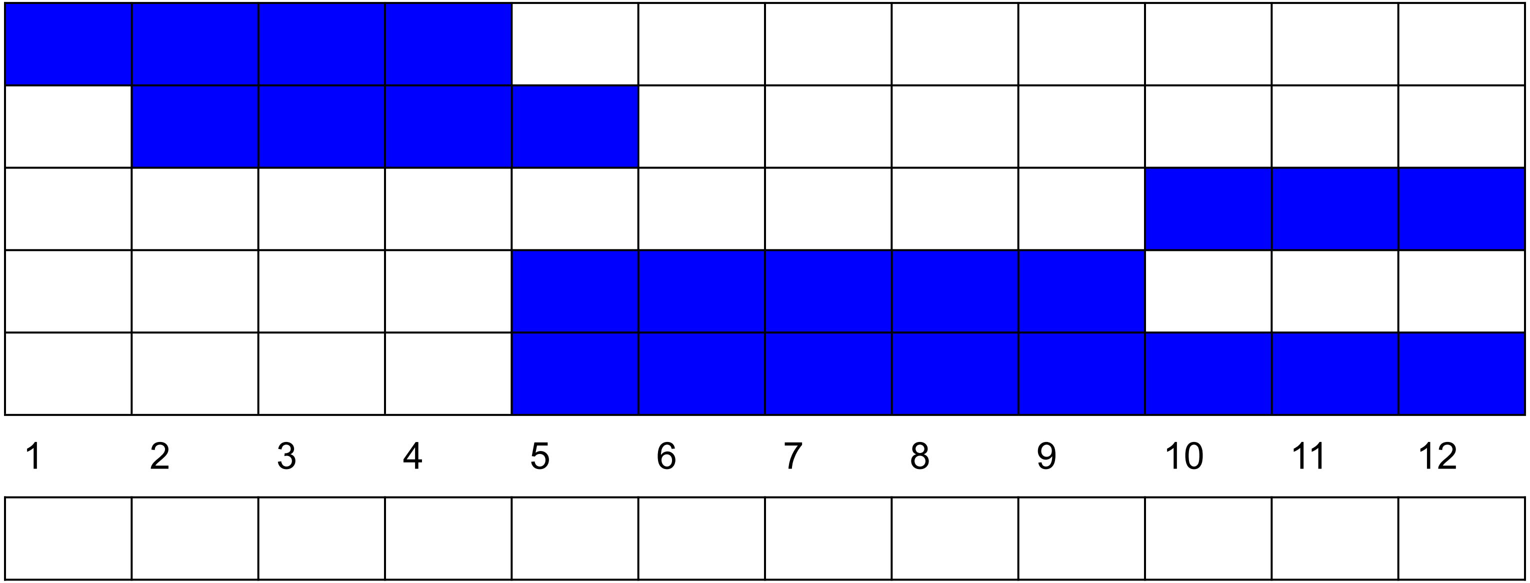
- Esta estratégia de varredura também pode ser aplicada em problemas que não são necessariamente geométricos. Por exemplo: Maximum Intervals Overlap
- **Objetivo**: Dado os horários de entrada e saída dos convidados em uma festa, determinar qual o número máximo de convidados que estiveram na festa ao mesmo tempo.

Line Sweep

- Solução: os eventos de entrada e saída serão ordenados conforme o tempo, permitindo a aplicação do line sweep. Tomando como exemplo um caso de teste dado pelo exercício com 5 convidados:
- Entradas: 1 2 10 5 5
- Saídas: 4 5 12 9 12

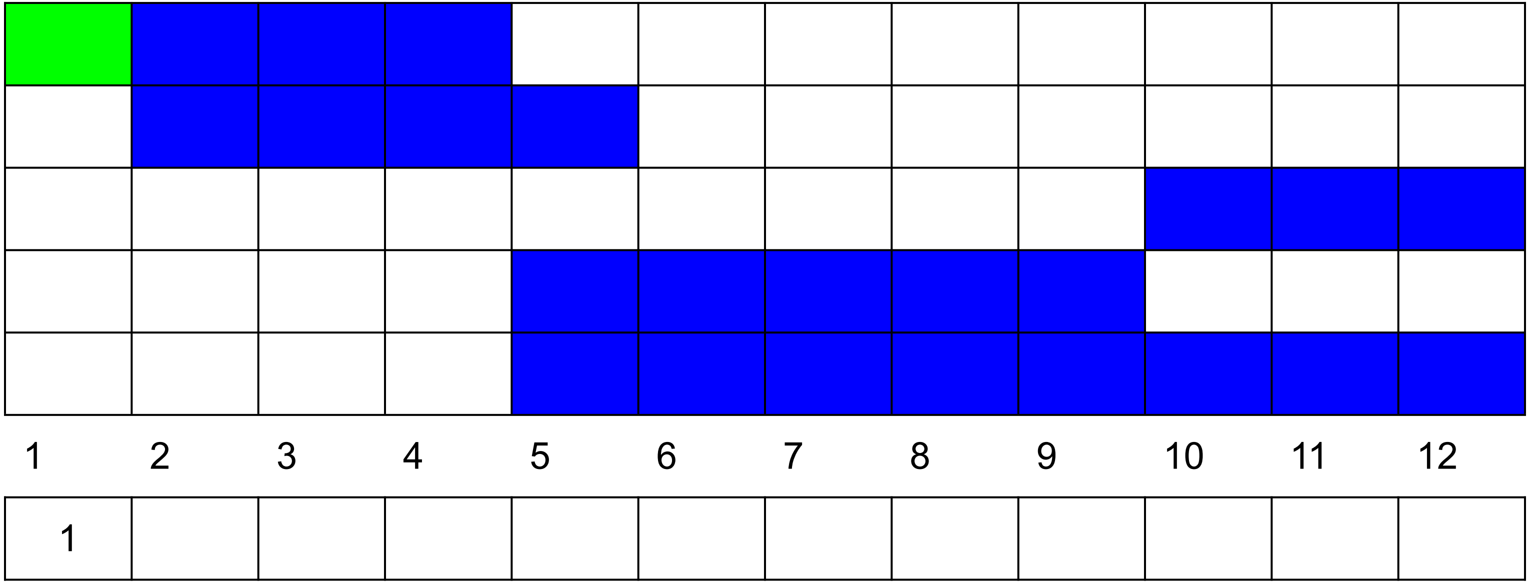
Line Sweep

- Entradas: 1 2 10 5 5   v = {(1,E), (2,E), (4,S), (5,E), (5,E), (5,S),(9,S),(10,E),(12,S),(12,S)}
- Saídas: 4 5 12 9 12



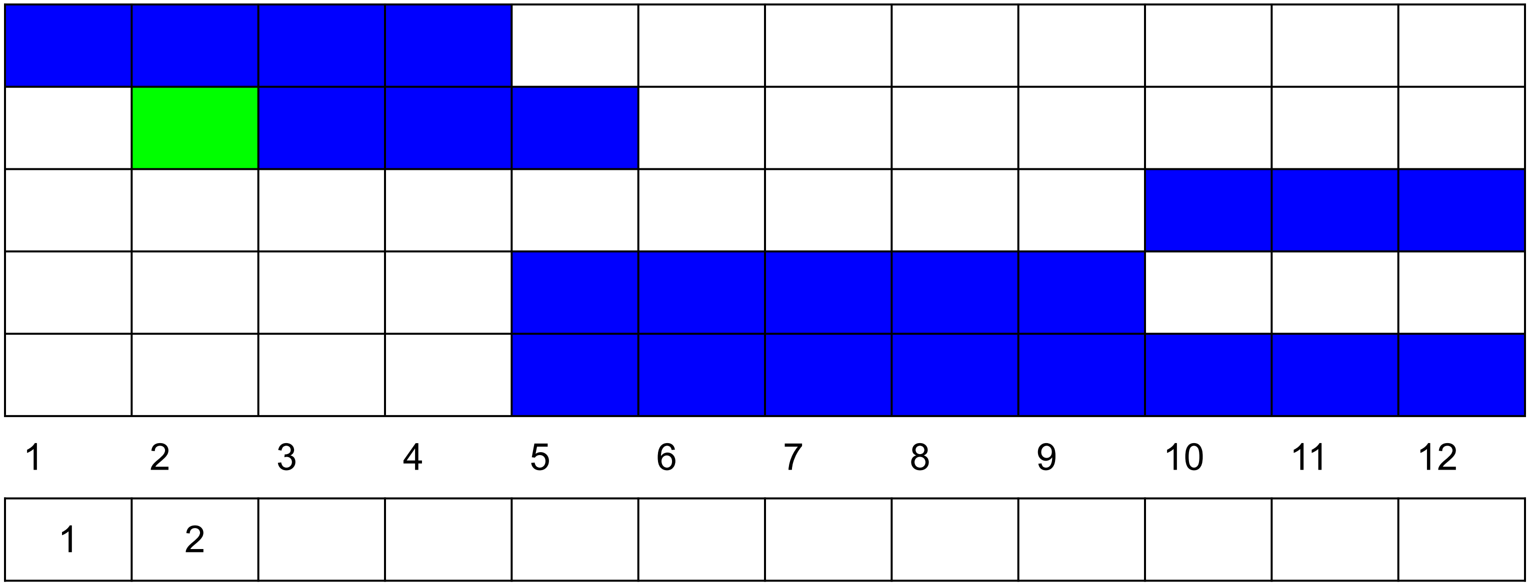
Line Sweep

- Entradas: 1 2 10 5 5 v = {**(1,E)**, (2,E), (4,S), (5,E), (5,E), (5,S),(9,S),(10,E),(12,S),(12,S)}
- Saídas: 4 5 12 9 12



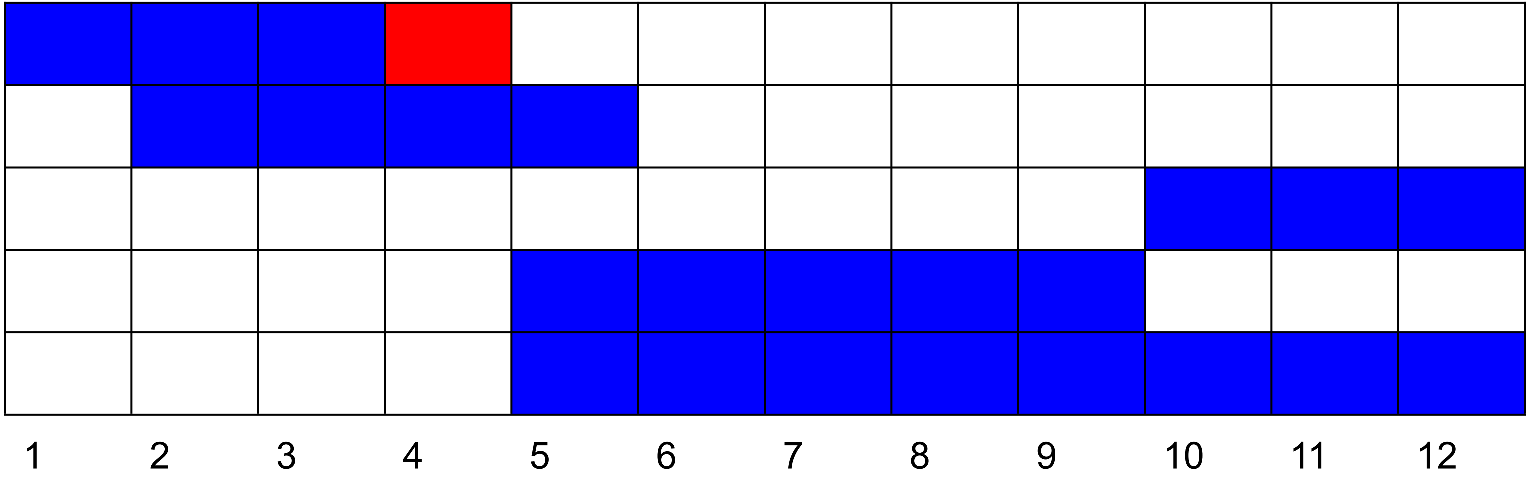
Line Sweep

- Entradas: 1 2 10 5 5   v = {(1,E), **(2,E)**, (4,S), (5,E), (5,E), (5,S),(9,S),(10,E),(12,S),(12,S)}
- Saídas: 4 5 12 9 12



Line Sweep

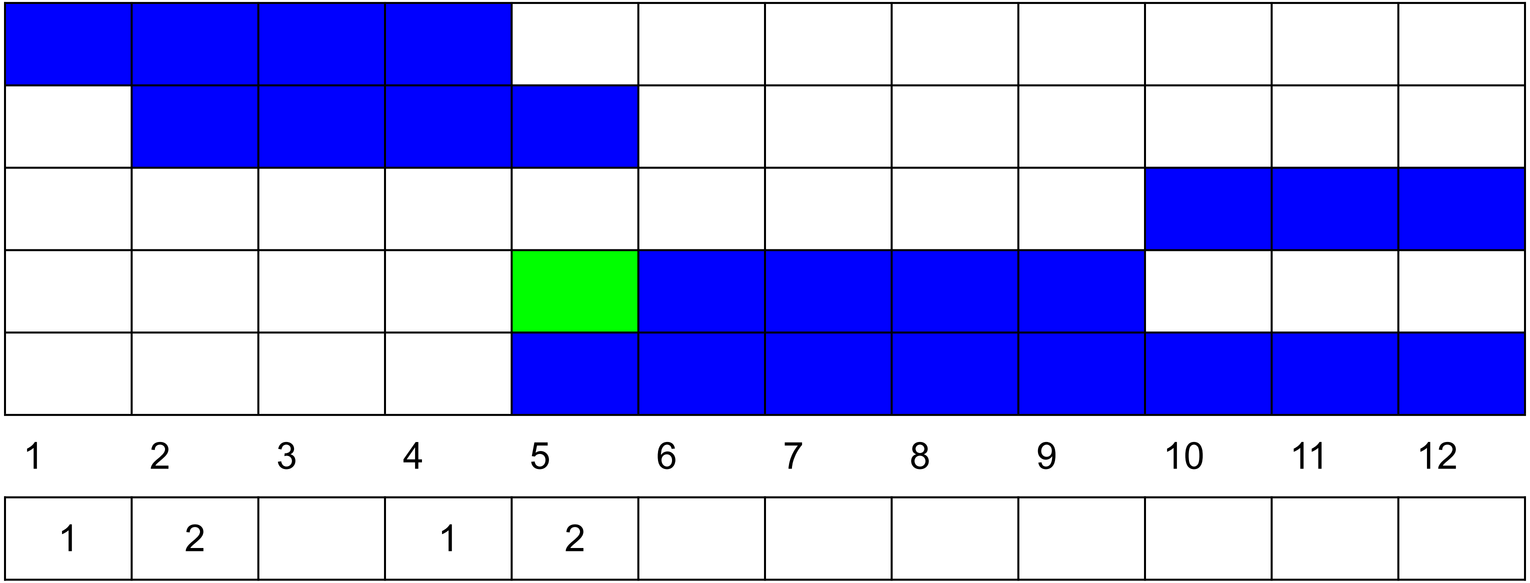
- Entradas: 1 2 10 5 5 v = {(1,E), (2,E), **(4,S)**, (5,E), (5,E), (5,S),(9,S),(10,E),(12,S),(12,S)}
- Saídas: 4 5 12 9 12



1	2		1								

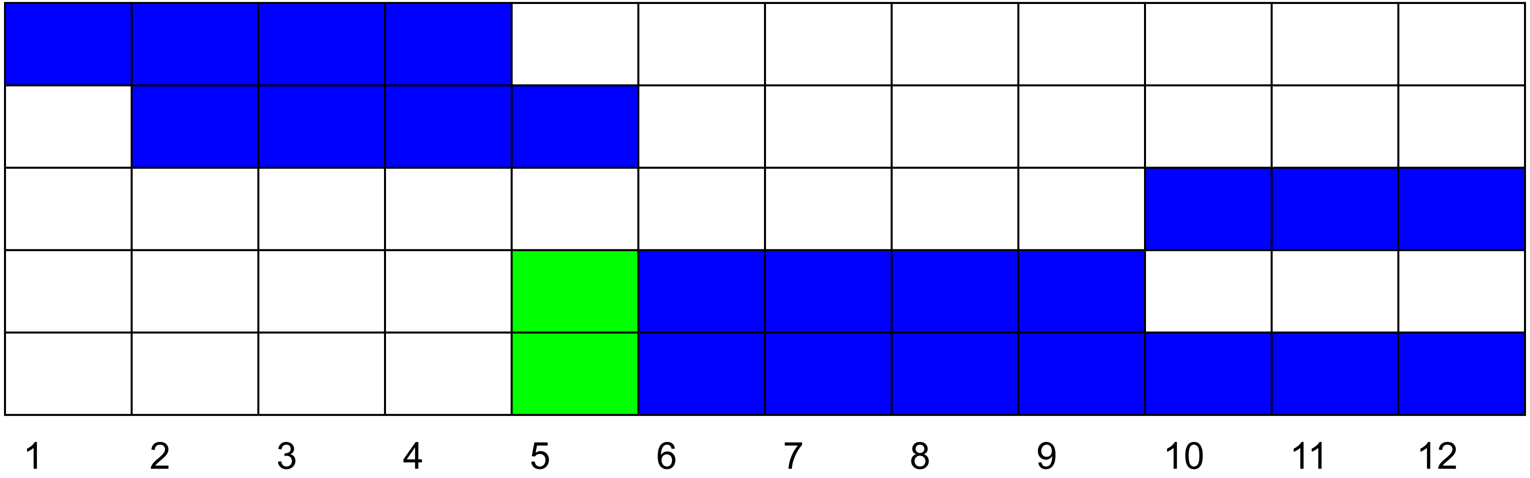
Line Sweep

- Entradas: 1 2 10 5 5 v = {(1,E), (2,E), (4,S), **(5,E)**, (5,E), (5,S),(9,S),(10,E),(12,S),(12,S)}
- Saídas: 4 5 12 9 12



Line Sweep

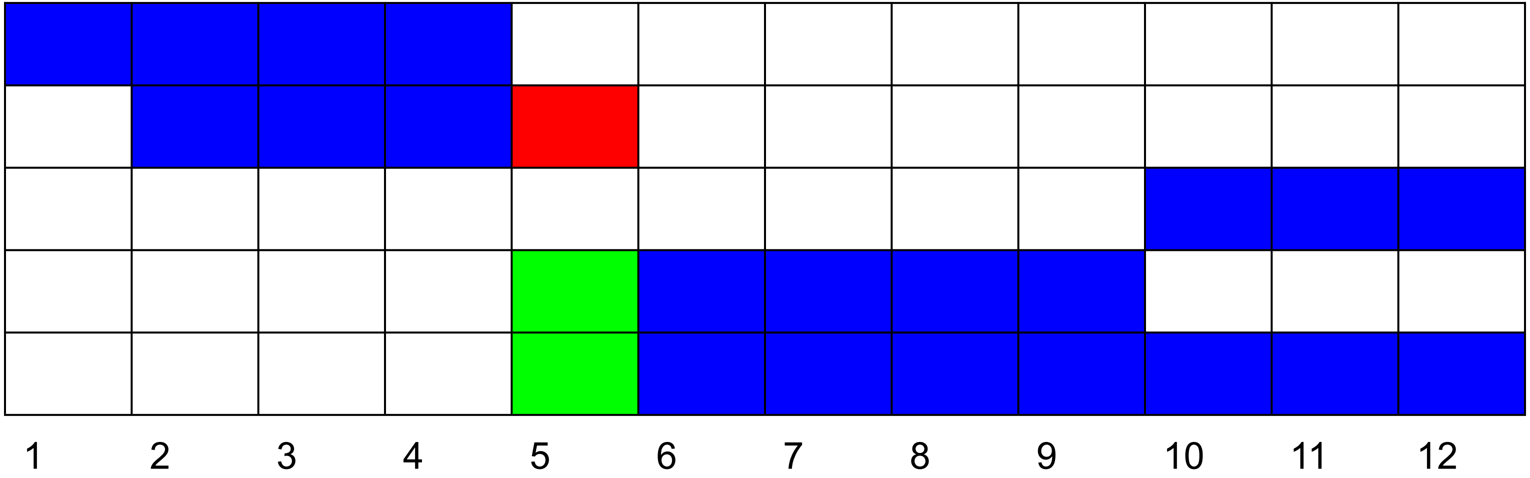
- Entradas: 1 2 10 5 5 v = {(1,E), (2,E), (4,S), (5,E), **(5,E)**, (5,S),(9,S),(10,E),(12,S),(12,S)}
- Saídas: 4 5 12 9 12



1	2		1	2,3							

Line Sweep

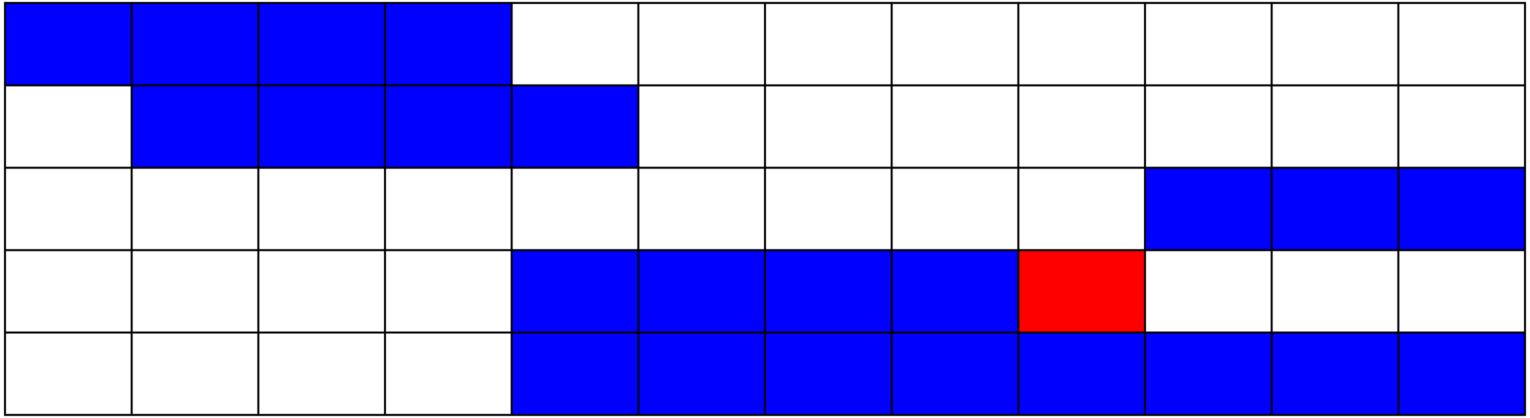
- Entradas: 1 2 10 5 5   v = {(1,E), (2,E), (4,S), (5,E), (5,E), **(5,S)**,(9,S),(10,E),(12,S),(12,S)}
- Saídas: 4 5 12 9 12



1	2		1	2,3,2							

Line Sweep

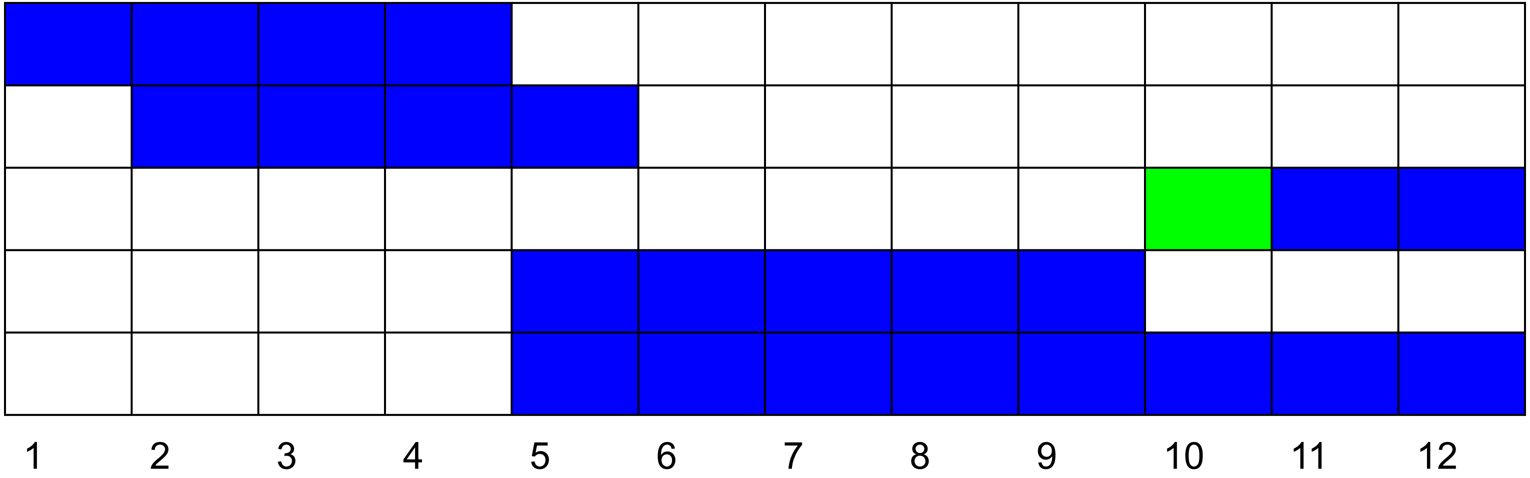
- Entradas: 1 2 10 5 5 v = {(1,E), (2,E), (4,S), (5,E), (5,E), (5,S),**(9,S)**,(10,E),(12,S),(12,S)}
- Saídas: 4 5 12 9 12



1	2	3	4	5	6	7	8	9	10	11	12
1	2		1	2,3,2				1			

Line Sweep

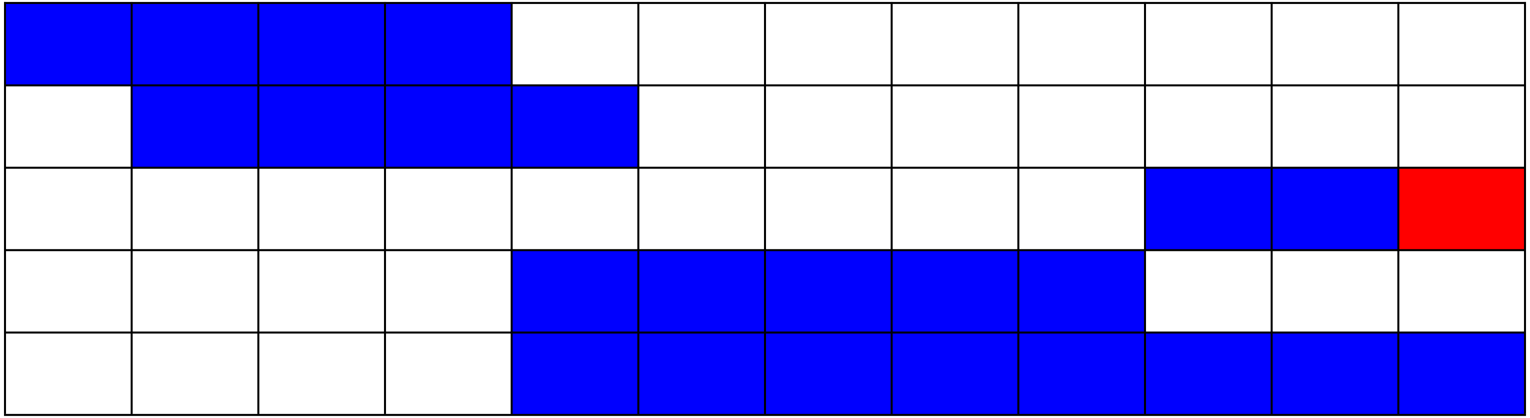
- Entradas: 1 2 10 5 5 v = {(1,E), (2,E), (4,S), (5,E), (5,E), (5,S),(9,S),**(10,E)**,(12,S),(12,S)}
- Saídas: 4 5 12 9 12



1	2		1	2,3,2				1	2		

Line Sweep

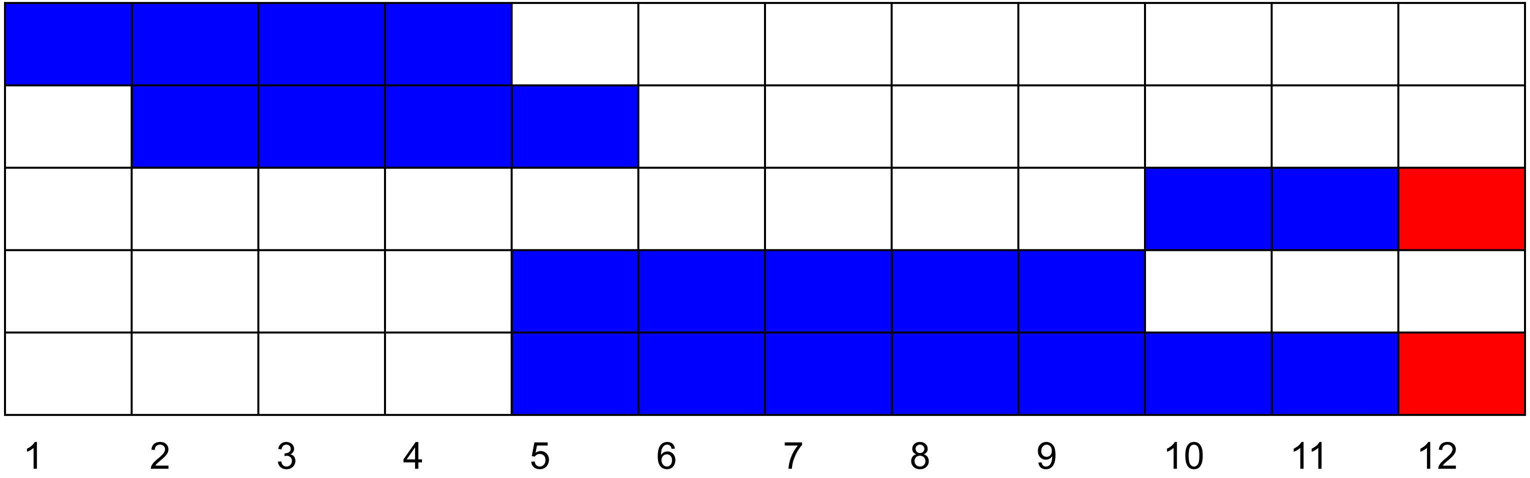
- Entradas: 1 2 10 5 5   v = {(1,E), (2,E), (4,S), (5,E), (5,E), (5,S),(9,S),(10,E),**(12,S)**,(12,S)}
- Saídas: 4 5 12 9 12



1	2	3	4	5	6	7	8	9	10	11	12
1	2		1	2,3,2				1	2		1

Line Sweep

- Entradas: 1 2 10 5 5 v = {(1,E), (2,E), (4,S), (5,E), (5,E), (5,S),(9,S),(10,E),(12,S),**(12,S)**}
- Saídas: 4 5 12 9 12



1	2		1	2,3,2				1	2		1,0

Referências

Biblioteca de códigos de Thiago Alexandre Domingues de Souza.

https://www.clis.com.br/unicamp2019/

https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/geometry/math-operations-on-points-and-vectors

https://stackoverflow.com/questions/13480135/operator-overloading-in-struct

http://wiki.maratona.dcc.ufmg.br/index.php/Roteiro_9

http://jeiks.net/wp-content/uploads/2014/08/TEP_Slides-14.pdf

https://www.youtube.com/watch?v=3ph6V32oja0

Referências

http://www.uel.br/projetos/matessencial/geometria/vetor2d/vetor2d.htm

http://www.uel.br/projetos/matessencial/geometria/vetor3d/vetor3d.htm

https://www.geeksforgeeks.org/convex-hull-set-2-graham-scan/

http://www.ic.uff.br/~anselmo/cursos/GeomComp/slides/GC_aula1(introducao).pdf

http://www3.decom.ufop.br/toffolo/site_media/uploads/2011-1/bcc402/slides/13._geometria_computacional.pdf