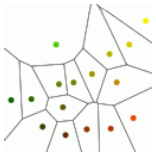
O WEB-site da disciplina de **Geometria Computacional**

http://www2.mat.ua.pt/pessoais/leslie/geocom/

Introdução à Geometria Computacional

António Leslie Bajuelos
Departamento de Matemática
Universidade de Aveiro



Mestrado em Matemática e Aplicações

2

O que é Geometria Computacional?

- **Uma boa definição:**

  Geometria Computacional é o estudo sistemático de algoritmos eficientes para problemas geométricos.

- **Raízes - bem antigas**

  Começou com a Geometria Euclidiana Clássica, que com seus axiomas, determinava construções geométricas (algoritmos) baseadas em operações simples.

3

O que é Geometria Computacional?

- **Alguns factos:**
  - O estudo dos algoritmos eficientes para problemas geométricos ficou parado por algum tempo.
  - A moda eram as *provas por contradição*, que não gerava nenhum método ou algoritmo de construção.
  - Por conta dos computadores voltaram a aparecer *provas construtivas* ⇨ os algoritmos reapareceram nos estudos geométricos
  - Em 1985 foi publicado o primeiro livro sobre o assunto, escrito por Preparata e Shamos

    F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introdution*. Springer-Verlag, New York, USA, 1985.

4

O que é Geometria Computacional?

- **Explorando a definição:**
  - O que são **algoritmos**?
  - O que é **eficiência algoritmica**?
  - O que é um **problema geométrico**?

5

O que é Geometria Computacional?

- **Problema Geométrico:**
  - **Características gerais:**
    - A ***entrada*** de um problema, isto é, os seus *dados*, é um conjunto finito de objectos geométricos (pontos, rectas, segmentos, triângulos, círculos, polígonos, etc.).
    - A ***saída*** ***(solução)*** do problema:
      - ***calcular*** algum número relacionado com a entrada (por exemplo, a área de um polígono)
      - ***construir*** outros objectos geométricos (por exemplo, a intersecção de dois polígonos),
      - ***decidir*** se a entrada tem uma certa propriedade (por exemplo, se um polígono é convexo).

6

O que é Geometria Computacional?

- **Problema Geométrico:**
  - Os objectos geométricos são geralmente definidos por ***números reais***, através de coordenadas ou equações.
  - Os objectos geométricos são ***contínuos***, mas têm um descrição ***discreta***, o que permite que eles sejam representados num computador.

Exemplo: um ***polígono*** é um região do plano, com um número infinito de pontos, mas que pode ser representada pela sua fronteira, que por sua vez pode ser representada pela sequência finita dos vértices.

7

O que é Geometria Computacional?

- **Problema Geométrico:**
  - A solução de um problema geométrico envolve:
    - aspectos **geométricos** (a localização e forma dos objectos)
    - aspectos **topológicos** (as relações de adjacência e incidência entre os objectos).

Um **algoritmo geométrico** envolve partes numéricas, correspondentes ao cálculo da geometria, e partes discretas, correspondentes ao cálculo da topologia.

Saber escolher como tratar a parte geométrica de forma simples, mas eficiente e robusta, é uma das chaves principais na resolução computacional de problemas geométricos.

8

O que é Geometria Computacional?

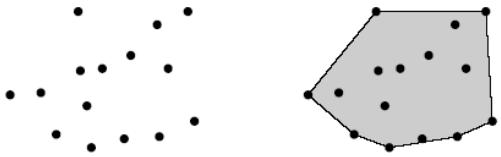
- **Problema Geométrico:**
  - A combinação dos aspectos geométricos & topológicos implicam em certas particularidades dos problemas geométricos que tornam a sua **solução robusta uma tarefa difícil**.
  - Essas particularidades estão ausentes na maioria dos problemas e algoritmos estudados em ciência da computação.

A mistura de técnicas geométricas, numéricas e discretas, aliada à motivação proveniente de problemas práticos dão à **Geometria Computacional** uma riqueza característica.

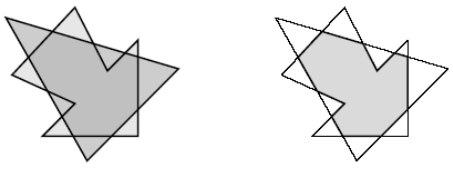
9

O que é Geometria Computacional?

- **Problema Geométrico:**
  - **Exemplos:**
    - **Invólucros Convexos**



    - **Intersecção de Polígonos**



10

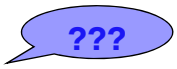
O que é Geometria Computacional?

✓ **Problema Geométrico**

11

Algoritmos e Eficiência Algorítmica

- **Algoritmo:**
  - **Uma definição muito informal:**

    Um algoritmo é uma receita matemática de como resolver um problema.
  - O **custo de um algoritmo** será medido pela quantidade de recursos computacionais utilizados na sua execução, tipicamente:
    - Quanto **tempo** ele leva para resolver um problema?
    - Quanta **memória** foi gasta pelo algoritmo.
  - Então...

    Dados dois ou mais algoritmos para solucionar um mesmo problema, é mais sensato escolher aquele que obtém uma solução no menor tempo possível e que utiliza a menor quantidade de espaço para representação dos dados do problema.

12

Algoritmos e Eficiência Algorítmica

- **Eficiência de Algoritmos:**
  - Vamos denotar por:
    - **S(n)** – quantidade de memória exigido por um algoritmo em função do tamanho (n) do problema
    - **T(n)** – tempo de execução em função do tamanho (**n**) do problema
  - Infelizmente na prática é bastante difícil (e em muitos casos impossível) prever com rigor o tempo de execução de um algoritmo.
  - Uma possível solução...
    - **Identificar no algoritmo as operações elementares (primitivas) e determinar o número de vezes que elas são executadas (o tempo "real" de execução de cada primitiva seria uma constante multiplicativa)**

Iremos simplificar a análise de um algoritmo concentrando-nos na eficiência **assimptotica** (análise do pior dos casos).

13

Algoritmos e Eficiência Algorítmica

- **Eficiência de Algoritmos:**
  - **Exemplo:**
    - Suponhamos que a **complexidade temporal** é expressa em termos de uma função $f(n)$.
    - Assim, interessa estudar a taxa de crescimento de $f(n)$
    - Suponhamos, por exemplo, que

      $f(n) = an^2+bn+c$

      onde a, b e c são constantes
    - À medida que n aumenta os termos de grau 1 e grau 0 tornam-se insignificantes pelo que podem ser desprezados. Mas, também a torna-se insignificante à medida que n aumenta.
    - Assim, encontraremos a função $T(n) = n^2$

14

Algoritmos e Eficiência Algorítmica

- **Eficiência de Algoritmos:**
  - Resumindo, ao expressarmos o tempo de execução de um algoritmo passaremos a trabalhar, apenas, com o termo dominante.
  - A **notação assimptótica** fornece-nos uma linguagem conveniente para expressarmos a taxa de crescimento de uma função de **complexidade temporal** em função do tamanho do problema.
  - As notações que usamos para descrever o **tempo de execução assimptótica** de um algoritmo são definidas em termos de funções cujos domínios são o conjunto de números $\aleph_0 = \{0, 1, 2, ...\}$.

15

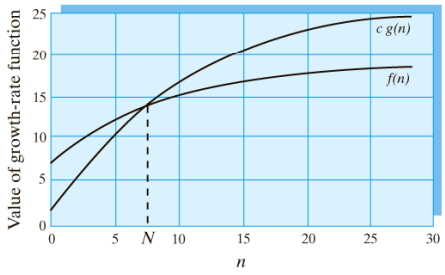
Algoritmos e Eficiência Algorítmica

- **Ordem de Complexidade:**
  - A notação *O* (*ordem de complexidade*)
    - Sejam *f* e *g* duas funções de domínio $\aleph$.

      Dizemos que a função *f* é $O(g(n))$ se,

      $\exists\, c \in \Re^+, \exists\, N \in \aleph: \ |f(n)| \leq c|g(n)|, \forall\, n \geq N$

      Assim, usamos a notação *O* quando queremos um **limite *superior* assimptótico**.



16

Algoritmos e Eficiência Algorítmica

- **Ordem de Complexidade :**
  - Ao usarmos a notação *O*, que pode ser referida como **ordem de complexidade**, podemos descrever o tempo de execução de um algoritmo inspeccionando, simplesmente, a estrutura do algoritmo para o *pior caso*.
  - **Exemplos:**

    $(n+1) / 2 = O(n)$

    $2n^3 + 4n^2 + 3 = O(n^3)$

    $n^2 + 1/n + 3 = O(n^2)$

    $n \log_2 n + n + 2 = O(n \log_2 n)$

    $\ln n = O(\log_2 n)$ $\qquad$ $\log_b a = \ln a / \ln b$

17

Algoritmos e Eficiência Algorítmica

- **Ordem de Complexidade :**
  - Assim podemos definir Classes de Complexidade Algorítmica (*ordem de complexidade*)

$O(1)$	Constante
$O(\log n)$	Logarítmica
$O(n)$	Linear
$O(n^2)$	Quadrática
$O(n^3)$	Cúbica
$O(n^k)$	Polinomial
$O(a^n)$	Exponencial
$O(n!)$	Factorial

  - Estas classes estão enumeradas por ordem crescente de esforço. Assim, os algoritmos "melhores", do ponto de vista de tempo de execução, são os algoritmos que têm ordem de complexidade constante.

18

Algoritmos e Eficiência Algorítmica

- **Ordem de Complexidade :**

Alguns exemplos de **Tempos de Execução** de Programas, em função da **Dimensão** n da **Instância** do **Problema** a resolver:

	10	100	10^3	10^4	10^5	10^6
$\log_2 n$	3	6	9	13	16	19
n	10	100	1000	10^4	10^5	10^6
$n \log_2 n$	30	664	9965	10^5	10^6	10^7
n^2	100	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{300}	10^{300}	10^{3000}	10^{300000}

(com $2^{10} \approx 10^3$)

1 ano = $365 \times 24 \times 60 \times 60 \approx 3 \times 10^7$ segundos
1 século $\approx 3 \times 10^9$ segundos
1 milénio $\approx 3 \times 10^{10}$ segundos

19

Algoritmos e Eficiência Algorítmica

- **Ordem de Complexidade :**
  - **Três algoritmos para determinar:**

1 + 2 + 3 + …+n

Algorithm A	Algorithm B	Algorithm C

`sum = 0`
`for i = 1 to n`
`  sum = sum + i`
	`sum = 0`
	`for i = 1 to n`
	`{ for j = 1 to i`
	`  sum = sum + 1`
	`}`
		`sum = n * (n + 1) / 2`

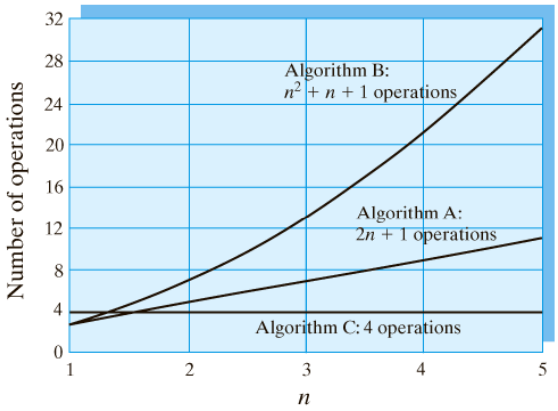
	Algorithm A	Algorithm B	Algorithm C
Assignments	$n + 1$	$1 + n(n+1)/2$	1
Additions	n	$n(n+1)/2$	1
Multiplications			1
Divisions			1
Total operations	$2n + 1$	$n^2 + n + 1$	**4**
	$O(n)$	$O(n^2)$	$O(1)$

20

Algoritmos e Eficiência Algorítmica

- **Ordem de Complexidade :**
  - **Três algoritmos para determinar:**

1 + 2 + 3 + ...+ n



21

Algoritmos e Eficiência Algorítmica

- **Ordem de Complexidade :**
  - Algoritmos de ordem de complexidade logarítmica são os mais eficientes.
  - Algoritmos de ordem de complexidade exponencial geram tempos de execução incomportáveis.
  - Um algoritmo só é viável se puder ser calculado em tempo útil.
  - **Na prática só interessam algoritmos de complexidade polinomial ou inferior**.

22

Algoritmos e Eficiência Algorítmica

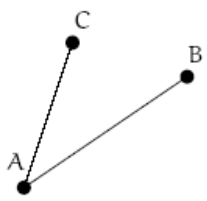
- **Algoritmo para problemas geométricos:**
  - Os **algoritmos eficientes** para problemas geométricos são fruto da combinação de
    - **Teoremas** sobre a geometria do problema:
      - Frequentemente teoremas clássicos, conhecidos há séculos, mas também teoremas novos inspirados pela abordagem algorítmica.
    - **Técnicas algorítmicas** tradicionais e especiais
    - **Estruturas de dados** adequadas
    - Boa **análise de desempenho**
    - **Primitivas geométricas** simples e poderosas

23

Algoritmos e Eficiência Algorítmica

- **Algoritmo para problemas geométricos :**
  - Em geral, basta usar um pequeno número de ***primitivas geométricas*** bem escolhidas; essas primitivas tendem a ser utilizadas em vários algoritmos.
  - Exemplo:

    Dados três pontos no plano, A, B e C, decidir se o ponto C está à esquerda do segmento orientado AB.



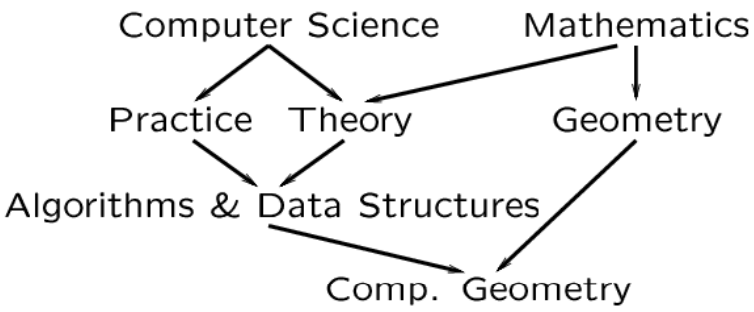
24

Algoritmos e Eficiência Algorítmica

✓ **Algoritmos e Eficiência**

25

Geometria Computacional



26

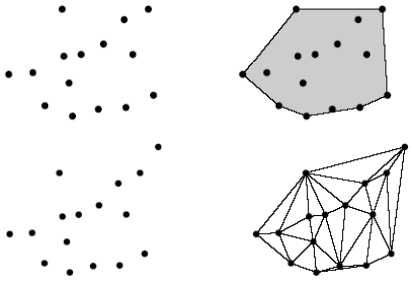
Geometria Computacional

- **Tipos de Problemas:**
  - Podemos classificar os problemas abordados em GC em quatro tipos:
    - ***Selectivos***
    - ***Construtivos***
    - ***Decisão***
    - ***Consultas***

27

Geometria Computacional

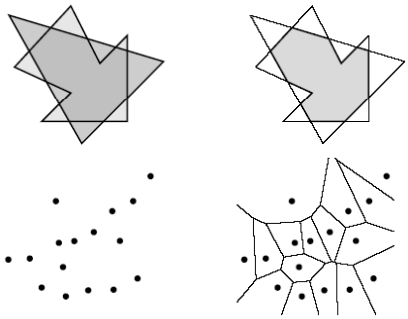
- **Tipos de Problemas:**
  - ***Selectivos:***
    - Nesses problemas queremos **seleccionar** um subconjunto da entrada de dados.
    - Não temos que construir nenhum objecto geométrico novo, mas possivelmente temos que **descobrir relações topológicas**.
    - Exemplos: **invólucros convexo**, **triangulação**, etc.



28

Geometria Computacional

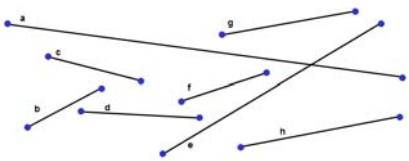
- **Tipos de Problemas:**
  - ***Construtivos:***
    - Nesses problemas temos que **construir** um ou mais objectos geométricos novos a partir da entrada, além de possivelmente relações topológicas envolvendo tanto objectos originais quanto objectos novos.
    - Exemplos: **intersecção de polígonos**, **diagrama de *Voronoi***, etc.

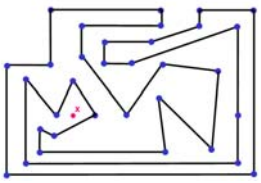


29

Geometria Computacional

- **Tipos de Problemas:**
  - ***Decisão:***
    - Nesses problemas temos somente que **responder sim ou não** a uma pergunta.
    - Não precisamos construir nada, nem novos objectos geométricos nem novas relações topológicas.
    - Exemplos:
      - Dado um conjunto de segmentos de recta, há algum par que se **intersecta**? (Não é necessário calcular o ponto de intersecção, somente identificar *um* par que se intersecta, se existir algum.)
      - Dado um polígono, ele é **convexo**?
      - Dado um polígono e um ponto, o ponto está fora do polígono?

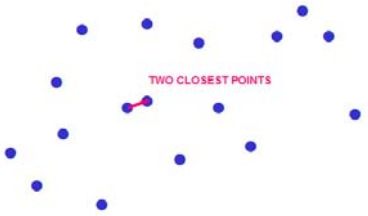




30

Geometria Computacional

- **Tipos de Problemas:**
  - ***Consulta:***
    - Dado um conjunto fixo de objectos geométricos, queremos processá-lo de modo a poder responder eficientemente a **consultas** repetidas sobre ele.
    - Exemplos:
      - Dado um conjunto de pontos, a consulta é encontrar o par de pontos mais próximos.
      - Dado um conjunto de rectângulos, a consulta é listar todos os rectângulos que intersectam um dado rectângulo.



31

Geometria Computacional

- **Aplicações:**
  - ***Computação Gráfica:***
    - Exemplos
      - Ao seleccionarmos um objecto numa interface gráfica, devemos **seleccionar dentre todos os objectos** desenhados na ecrã aquele que está mais próximo da posição do *mouse*.
      - Para **desenhar** de forma realista uma cena tridimensional, é necessário saber como os vários objectos se projectam na ecrã e tratar as suas oclusões.
      - Para fazer uma **animação realista**, é necessário detectar se há colisões entre os objectos que estão se movendo e o resto da cena.

32

Geometria Computacional

- **Aplicações:**
  - ***Robótica:***
    - Um dos problemas fundamentais em robótica é o ***planeamento de movimentos***: o robô precisa analisar o seu ambiente e descobrir uma forma de se mover de um ponto ao outro sem colidir com os objectos no ambiente.
    - Além disso, o robô quer fazer isso da maneira mais eficiente possivelmente, o que implica na necessidade de identificar o caminho mais curto (é viável!) entre os dois pontos.

33

Geometria Computacional

- **Aplicações:**
  - ***Sistemas de Informação Geográfica:***
    - Esses sistemas lidam com enormes quantidades de dados geométricos para poder representar fielmente a geometria de estradas, rios, fronteiras, curvas de nível, áreas verdes, etc.
    - Um problema típico nessa área é saber que objectos geográficos estão perto de outros.
    - Exemplo:
      - Se um rio ameaça transbordar, quais as cidades e estradas que serão afectadas?

34

Geometria Computacional

- **Aplicações:**
  - ***Tecnologia VSLI:***
    - Neste tipo de tecnologias os circuitos são compostos por dezenas de milhares de componentes electrónicos que não podem se sobrepor, para evitar curto-circuitos.
    - Durante o projecto desses circuitos é necessário identificar de forma eficiente as ***sobreposições***.
  - ***Bases de Dados:***
    - Uma consulta a uma base de dados com muitos campos numéricos é na verdade uma ***consulta multi-espacial***.
    - Exemplo
      - Se a base de dados de uma companhia armazena idade, salário, altura, peso de cada funcionário, então cada funcionário é representado por um ponto em $\mathbf{R}^4$.

35

Geometria Computacional

- **Configurações *degeneradas*:**
  - A principal particularidade dos problemas geométricos é a existência de ***configurações degeneradas***, que ocorrem quando os objectos geométricos não estão em ***posição geral***.
  - Exemplos:
    - *Três pontos numa mesma recta,*
    - *Quatro pontos sobre um mesmo círculo,*
    - *Três rectas passando pelo mesmo ponto.*
  - Situações deste género são chamadas de degeneradas porque elas ocorrem somente com probabilidade quase zero (o que não quer dizer que não aconteçam na prática!).
  - Algoritmos ***robustos*** deve estar preparado para lidar com casos especiais vindos de configurações degeneradas.

Fonte de complicações na implementação de algoritmos geométricos

36

Geometria Computacional

- **Configurações *degeneradas*:**
  - Uma consequência importante da existência de configurações degeneradas é que os erros de ***arredondamento da aritmética de ponto flutuante*** passam a ser relevantes em situações *quase* degeneradas.
  - Exemplo:
    - Um ponto que esteja à esquerda de um segmento de recta e muito perto desse segmento pode ser confundido com um ponto que está à direita, e vice-versa.

⇩ ⇩

O algoritmo pode tomar decisões topológicas erradas ou inconsistentes

  - **Atenção:** A topologia muda descontinuamente e a geometria muda continuamente ⇨ portanto um pequeno erro na geometria pode acarretar um grande erro na topologia.

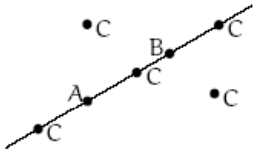
37

Geometria Computacional

- **Configurações *degeneradas*:**
  - Exemplo:

Primitiva geométrica para determinar se um ponto C está à esquerda de um segmento orientado AB.

    - Genericamente, C pode estar à esquerda ou à direita de AB. Entretanto, isso não cobre todos os casos, pois há configurações degeneradas, i.e. C pode estar:
      - ***Sobre a recta AB***
      - ***Antes de A,***
      - ***Em A***
      - ***Entre A e B***
      - ***Em B***
      - ***Depois de B***



⇩ ⇩

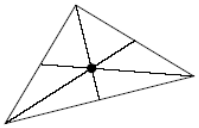
Uma *implementação robusta* desse predicado tem que tratar cuidadosamente todos esses casos

38

Geometria Computacional

- **Configurações *degeneradas*:**
  - ***Invariantes geométricos*** *- mesmo objectos geométricos em posição geral podem gerar objectos secundários em configurações degeneradas.*
  - Exemplo:

    As três medianas de um triângulo são concorrentes.



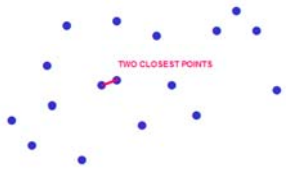
  - Esse teorema permite definir o ***baricentro*** de um triângulo como o ponto comum às suas três medianas.

Ainda que os vértices do triângulo estejam em posição geral no plano, as três medianas sempre são concorrentes e portanto não estão em posição geral

39

Problemas Clássicos da GC

- ***Par mais próximo*** (*the closest pair problem*)
  - **Enunciado:** *Dados n pontos queremos encontrar dois cuja distância entre eles é mínima.*

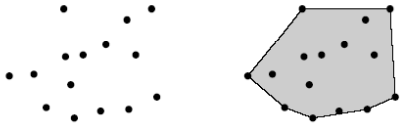


  - **Aplicação:** Controlo de tráfego aéreo: os dois aviões que estão em maior perigo de colisão são aqueles que estão mais próximos.
  - **Resolução:**
    - Este problema pode ser resolvido facilmente em $O(dn^2)$; onde d é a dimensão do espaço.
    - Numa forma mais eficiente este problema pode ser resolvido por um algoritmo do tipo divisão-e-conquista em tempo $O(dn \log n)$

40

Problemas Clássicos da GC

- ***Invólucros Convexos*** (*convex hull*)
  - **Enunciado:** *Dado um conjunto Q com n pontos; encontrar o invólucro convexo[1] desse conjunto de pontos.*



  - **Aplicação:** Robótica - Se o invólucro convexo de um robô não colide com obstáculos então o robô também não colide.
  - **Comentários:**
    - Historicamente, esse foi o primeiro **problema geométrico** a ser completamente analisado.
    - Tem uma estreita relação com o problema de ordenação: sua complexidade está intimamente ligada à de ordenação e muitos algoritmos para ordenação têm versões análogas para invólucros convexos.

[1] O *invólucro convexo* de um conjunto $Q \subset \boldsymbol{R}^d$ é o menor conjunto convexo de $\boldsymbol{R}^d$ que contém Q

41

Problemas Clássicos da GC

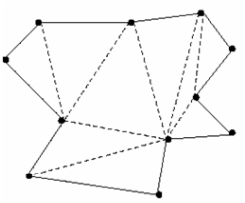
Applet

- ***Invólucros Convexos*** (*convex hull*)
  - **Resolução:**
    - Nos anos 60 uma aplicação da Bell Labs necessitava computar o ***invólucro convexo*** de aproximadamente 10000 pontos no plano e os algoritmos de complexidade de tempo $O(n^2)$ foram considerados muito lentos.
    - Tendo essa aplicação como motivação, no começo dos anos 70, *Graham* desenvolveu o primeiro algoritmo de complexidade de tempo $O(n \log n)$
    - O fecho convexo também pode ser construído em $O(n \log n)$ por um algoritmo de divisão-e-conquista

42

Problemas Clássicos da GC

- ***Triangulação*** (*triangulation*)
  - **Enunciado:** (para polígonos) *Dado um polígono P queremos adicionar o maior número possível de diagonais (que não se cruzem) em P, de tal forma que o interior do polígono fique dividido em triângulos com interiores disjuntos.*



  - **Aplicação:** na resolução do problema da galeria de arte.

43

Problemas Clássicos da GC

Applet

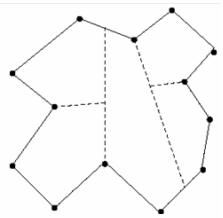
- ***Triangulação*** (*triangulation*)
  - **Resolução:**

Running Time	Designers	Year	Technique
$O(n^2)$	*Lennes*	1911	Recursive diagonal insertion
$O(n^3)$	*Meisters*	1975	Ear cutting
O(n log n)	*Garey, Johnson, Preparata & Tarjan*	1978	Decomposition into monotone pieces
O(n log n)	*Chazelle*	1982	Divide & Conquer
O(n + r log r) where r is the # of reflex vertices of the Polygon	*Hertel & Mehlhorn*	1983	-
O(n log s) where s is the sinuosity of P	*Chazelle*	1983	-
O(n log log n)	*Tarjan & Van Wyk*	1987	Using involved data structures
O(n (1 + to)) where to is the # of free triangles in the output triangulation.	*Toussaint*	1988	Sleeve-searching (Output sensitive)
$O(n^2)$	*ElGindy, Everett & Toussaint*	1990	Finding an ear in linear time via prune & search
O(n)	*Chazelle*	1990	-
O(kn)	Kong, Everett & *Toussaint*	1990	Graham Scan

44

Problemas Clássicos da GC

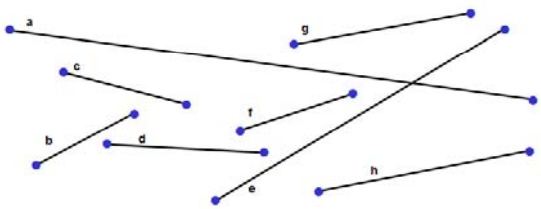
- ***Partição de Polígonos*** (*polygon partition*)
  - **Enunciado:** Alem de algoritmos eficientes para dividir um polígono em ***triângulos***, também é de interesse o desenvolvimento de algoritmos que dividam um polígono em ***polígonos monótonos***, ***trapezóides*** ou ***polígonos convexos***.



  - **Aplicação:** Uma motivação para particionar um polígono em polígonos convexos é o reconhecimento de caracteres: um *character* pode ser representado como um polígono particionado em partes convexas.

Problemas Clássicos da GC

- ***Problemas de Intersecção*** (*intersection problems*)
  - Um dos problemas mais básicos em geometria computacional é o de **computar intersecção**.
  - **Dois enunciados:**
    - Problema Nº1: *Dados n segmentos de rectas no plano;* ***determinar todos os pontos*** *de intersecção entre pares de segmentos.*
    - Problema Nº2: *Dados n segmentos de recta no plano;* ***decidir*** *se existem dois segmentos que se intersectam.*



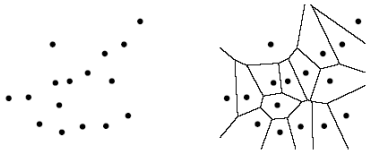
Problemas Clássicos da GC

Applet

- ***Problemas de Intersecção*** (*intersection problems*)
  - **Aplicações:**
    - em **robótica e planeamento de movimentos** é importante sabermos quando dois objectos se intersectam para evitarmos colisões;
    - em **computação gráfica**: *ray shooting* é um método importante para a *renderização* de cenas. A parte que é computacionalmente mais cara do *ray shooting* é justamente determinar a intersecção entre o raio com outros objectos.
  - **Resolução (Problema Nº 2):**
    - **Força bruta:** $O(n^2)$
    - **Linha de Varrimento:** $O(n \log n + I \log n)$

Problemas Clássicos da GC

- ***Diagrama de Voronoi*** (*Voronoi Diagrams*)
  - **Enunciado:** Dado um conjunto S de n pontos no plano queremos determinar para cada ponto p em S qual e a região $V(p)$ dos pontos do plano que estão mais perto de p do que de qualquer outro ponto em S. As n regiões $V(p)$ formam uma partição do plano chamada de ***Diagrama de Voronoi***.

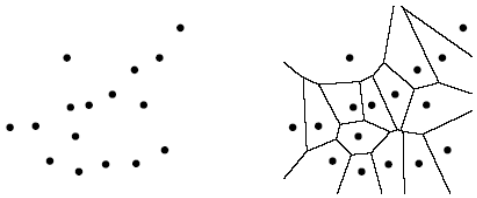


  - **Aplicação:** Imagine uma vasta floresta contendo vários pontos de observação de incêndio. O conjunto das árvores que estão mais próximas de um determinado posto p determina a região $V(p)$ das árvores que são de responsabilidade de ponto de observação p.

Problemas Clássicos da GC

Applet

- ***Diagrama de Voronoi*** (*Voronoi Diagrams*)
  - **Resolução:**
    - O ***diagrama de Voronoi*** de um conjunto de n pontos pode ser construído em $O(n \log n)$ por um algoritmo do tipo **divisão-e-conquista**
    - Em 1985, Fortune desenvolveu um algoritmo de varrimento (plane-sweep algorithm) muito elegante e simples cuja complexidade de tempo e $O(n \log n)$

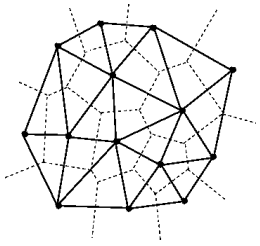


49

Problemas Clássicos da GC

Applet

- ***Triangulação de Delaunay*** (*Delaunay triangulations*)
  - **Enunciado:** O dual geométrico (utilizando rectas) de um **diagrama de *Voronoi*** para um conjunto *S* de pontos forma uma triangulação ao do conjunto *S*, chamada de ***triangulação de Delaunay.***

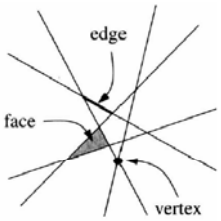


  - **Aplicações:**
    - Simulação do processo de produção de cristais
    - Metalurgia (para examinar as misturas)
    - Cartografia

50

Problemas Clássicos da GC

- ***Arranjos e dualidade*** (*Arrangements of lines*)
  - **Definição:** Dadas n rectas no plano, um ***arranjo*** é simplesmente o grafo que tem como vértices as intersecções das rectas e como as arestas os segmentos de rectas ligando estas intersecções.
  - Uma tal estrutura pode ser construída em tempo $O(n^2)$

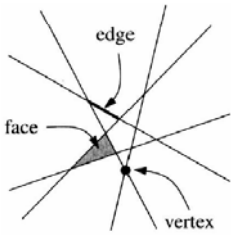


  - **Motivação:** muitos problemas envolvendo pontos podem ser transformados em problemas envolvendo rectas através do ***método de dualidade***.

51

Problemas Clássicos da GC

- ***Arranjos e dualidade*** (*Arrangements of lines*)
  - **Exemplo:** Suponha que desejemos determinar se existem ***três pontos colineares*** entre um conjunto de *n* pontos no plano



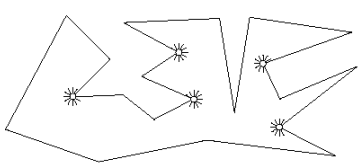
  - **Resuloção:**
    - Isto pode ser determinado por um algoritmo do tipo **força-bruta** em tempo $O(n^3)$
    - Entretanto, se os pontos são dualizados em rectas, então a questão é reduzida a decidir se existe um vértice de grau pelo menos 4 neste arranjo de rectas

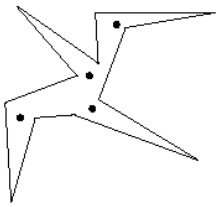
52

Problemas Clássicos da GC

Applet

- ***Galeria de Arte*** (*Art Gallery Problem)*
  - **Enunciado:** *Em 1973, Victor Klee propôs o problema de determinar o menor número de guardas que são suficientes para cobrir o interior de uma sala de uma* ***galeria de arte*** *com n paredes.*





  - **Resuloção:**
    - Teorema da Galeria de Arte (***Chvatal's Art Gallery Theorem ou Watchman Theorem***) que diz que $\lfloor n/3 \rfloor$ guardas são ocasionalmente necessários e sempre suficientes para cobrirmos uma sala com *n* paredes.

53