Universidade Federal do Espírito Santo
Centro de Ciências Agrárias – CCA UFES
Departamento de Computação

Geometria Computacional

Tópicos Especiais em Programação

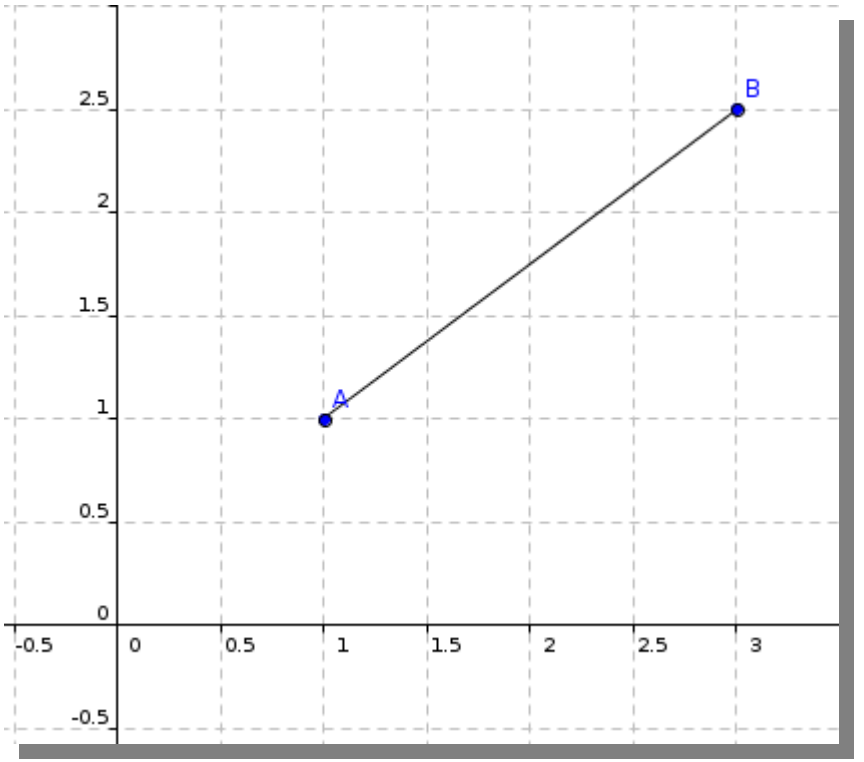
Site: http://jeiks.net
E-mail: jacsonrcsilva@gmail.com

Geometria Computacional

- É Importante em aplicações, como Gráficos computacionais, Robótica, Design, etc.
- No Mundo real:
  - A maioria dos objetos não é feita por linhas que tendem ao infinito;
- Já os objetos no computador:
  - Em sua maioria representam a geometria como arranjos de segmentos de linha.
  - Curvas fechadas ou formas são representadas por coleções ordenadas de segmentos de linha ou polígonos (discretos).
- Geometria computacional:
  - A geometria de segmentos de linhas discretas e polígonos.
  - É um divertido e interessante assunto, mas não tipicamente ensinado em cursos universitários necessários.
  - O aluno que aprende um pouco de geometria computacional tem vantagem e uma janela aberta para uma área fascinante de algoritmos ainda sob investigação nos dias de hoje.

Segmentos de Linha

- Um segmento de linha “s” é uma reta formada por dois pontos:
  - A = (1 ; 1)
  - B = (3 ; 2,5)
- Assim, podem ser representados por um par de pontos:




```
typedef struct {
   point p1,p2;
} segmento;
```

Segmentos de Linha e Interseção

- Primitiva mais importante na geometria sobre segmentos:
  - Testar se um determinado par deles se cruzam.
- Porém:
  - Dois segmentos podem ser linhas paralelas, o que significa que não se cruza.
  - Eles podem se cruzar nas extremidades, ou no meio do segmento.
  - Isso pode ser um verdadeiro problema.
  - Leia qualquer especificação do problema com cuidado para ver se ele não terá linhas paralelas ou segmentos sobrepostos.
  - No entanto, é melhor fazer o programa preparado para lidar com eles.

Segmentos de Linha e Interseção

- Nós utilizaremos nossas rotinas de interseção para encontrar se a interseção de um ponto existe.

```
bool point_in_box(point p, point b1, point b2){

return
  (p.X >= min(b1.X,b2.X)) && (p.X <= max(b1.X,b2.X))
  &&
  (p.Y >= min(b1.Y,b2.Y)) && (p.Y <= max(b1.Y,b2.Y))
  ;
}
```

```
bool segments_intersect(segment s1, segment s2) {
    line l1,l2;    /* linhas contendo os seguimentos de entrada */
    point p;       /* ponto de intersecao */

    points_to_line(s1.p1, s1.p2, l1);
    points_to_line(s2.p1, s2.p2, l2);
    if (same_lineQ(l1,l2)) /* segmentos sobrepostos ou disjuntos */
        return( point_in_box(s1.p1,s2.p1,s2.p2) ||
                point_in_box(s1.p2,s2.p1,s2.p2) ||
                point_in_box(s2.p1,s1.p1,s1.p2) ||
                point_in_box(s2.p1,s1.p1,s1.p2) );

    if (parallelQ(l1,l2)) return false;
    if (! intersection_point(l1,l2,p) ) return false;

    return  point_in_box(p,s1.p1,s1.p2) &&
            point_in_box(p,s2.p1,s2.p2);
}
```

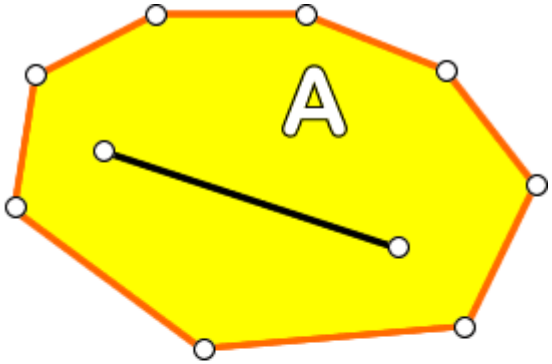
Computação de Polígonos e Ângulos

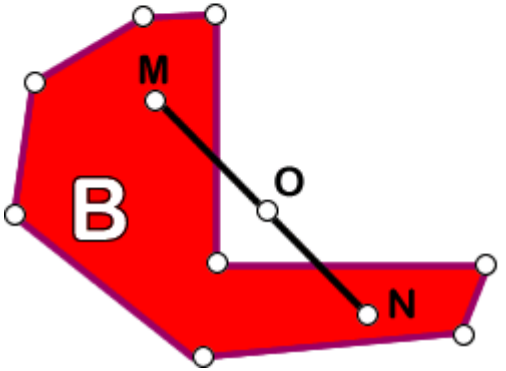
- **Polígonos:**
  - São um conjunto de segmentos de linha sem interseção que fecham-se em um vértice, formando uma figura geométrica.
  - O primeiro vértice do conjunto de segmentos é igual ao último.
  - Suas arestas não possuem interseção:
    - Somente pares de segmentos terão encontros em um de seus pontos (ponto final do segmento).
  - São a estrutura básica para descrever formas no plano.
- **Representação:**
  - Não listar os segmentos (arestas) do polígono.
  - Fazer a listagem dos n vértices ordenados que formam as extremidades do polígono

```
typedef struct {
  int n;              /* número de pontos no polígono */
  point p[MAXPOLY];   /* array de pontos do polígono */
} polygon;
```

Polígono Convexo

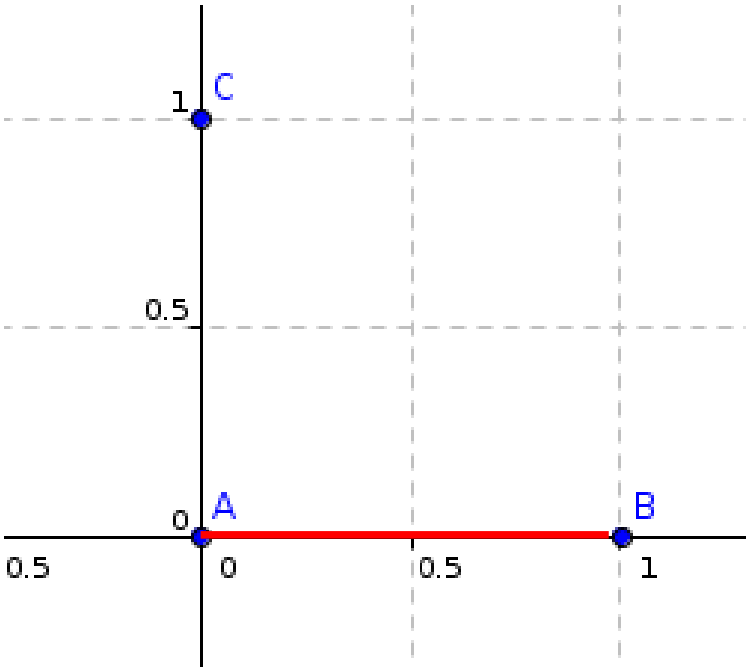
- A Região Convexa se e somente se todo segmento de reta reside na mesma região.
- A região A é convexa:
  - Pois qualquer segmento de reta que for escolhido só tem pontos na mesma região A.
- O polígono B já não é convexo:
  - Pois existe pelo menos um segmento de reta que tem extremidades na região B, mas tem pontos fora da região.
  - Repare que os pontos M e N estão em B, mas O é um ponto fora da região. Neste caso a região B é chamada de Região Não-Convexa.
- Assim, todos os ângulos devem ter no máximo 180º ou π radianos.





Polígonos

- Calcular o ângulo definido entre três pontos ordenados é um problema complicado.
- Nós podemos evitar a necessidade de saber ângulos reais na maioria dos algoritmos geométricos utilizando o atributo "sentido anti-horário" ccw(a, b, c).
  - Esta rotina testa se o ponto C está posicionado à direita da linha dirigida que vai do ponto A para o ponto B.
  - Caso positivo, o ângulo formado pela varredura de A até C em sentido anti-horário em torno B é agudo.
  - Caso negativo, o ponto ou encontra-se à esquerda de AB ou os três pontos são colineares.

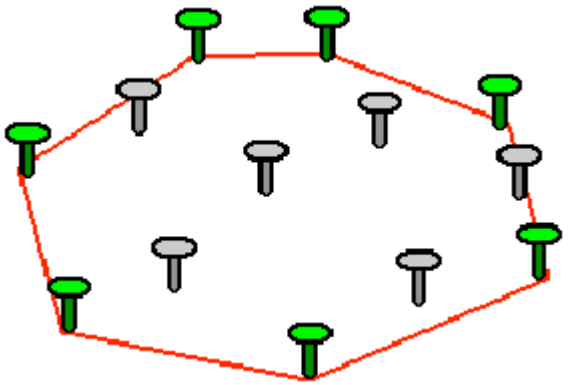


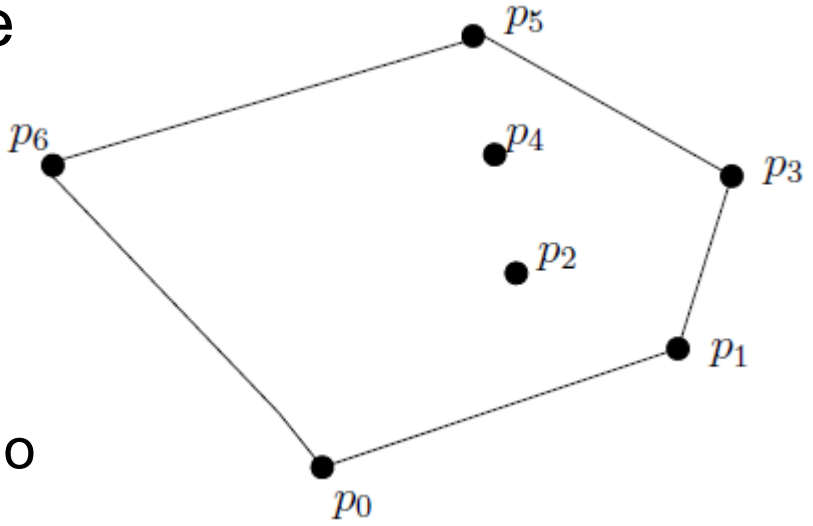
Polígonos

```
bool ccw(point a, point b, point c) {
  return (signed_triangle_area(a,b,c) > EPSILON);
}

bool cw(point a, point b, point c) {
  return (signed_triangle_area(a,b,c) < EPSILON);
}

bool collinear(point a, point b, point c) {
  return (fabs(signed_triangle_area(a,b,c)) <= EPSILON);
}
```

Casco Convexo (*Convex Hulls*)

- O casco convexo de um conjunto Q de pontos no plano é o menor polígono convexo que os envolve.
  - Todos os pontos de Q devem estar dentro do polígono ou sobre sua borda.
  - Para visualizar esse polígono:
    - Imagine que os pontos são pregos saindo do plano;
    - Imagine uma linha esticada de modo a conter todos os pontos.
    - Essa linha, após ser colocada prendendo-se aos pregos mais externos, delimitará o casco convexo dos pontos.





Varredura de Graham

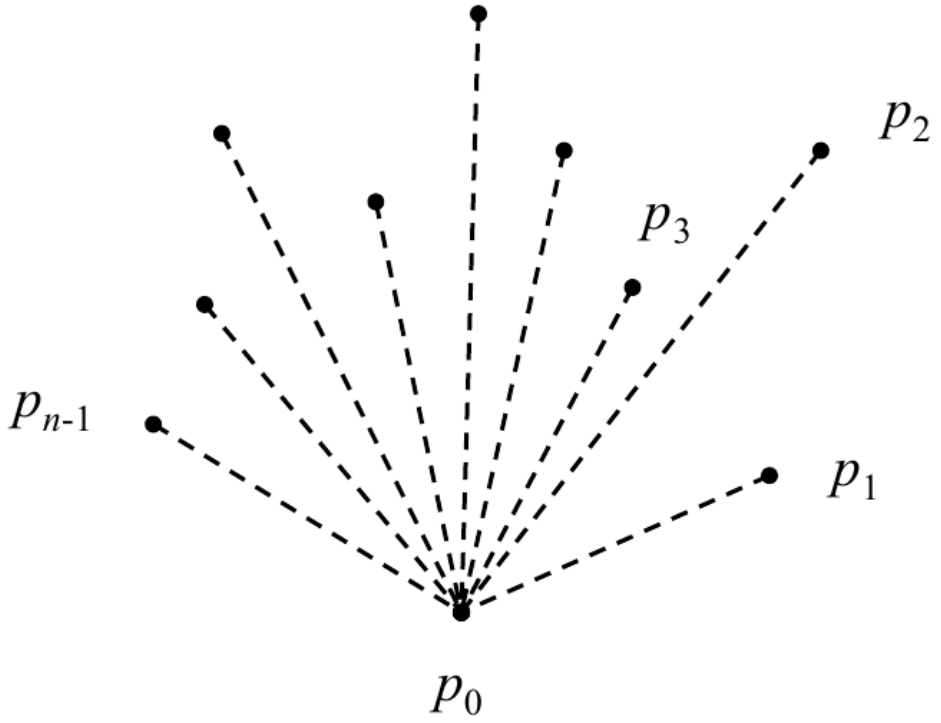
- Considerado o primeiro algoritmo de Geometria Computacional (1972), há quase tantos algoritmos diferentes para envoltória convexa quanto há para ordenação.
- O algoritmo de Graham:
  - Ordena os pontos do conjunto:
    - em ordem angular ou
    - da esquerda para a direita
  - E depois insere os pontos na envoltória nesta ordem.

Varredura de Graham

- Algoritmo incremental (2D)
  - Pontos são pré-ordenados de forma conveniente;
  - Cada ponto é adicionado ao fecho convexo e testado.
- Precisamos de um ponto inicial p_0 que faz parte do casco convexo de forma garantida.
  - Solução: Tomar o ponto com menor coordenada *x* ou *y* como p_0.
  - Na verdade, um ponto extremo em qualquer direção serve.

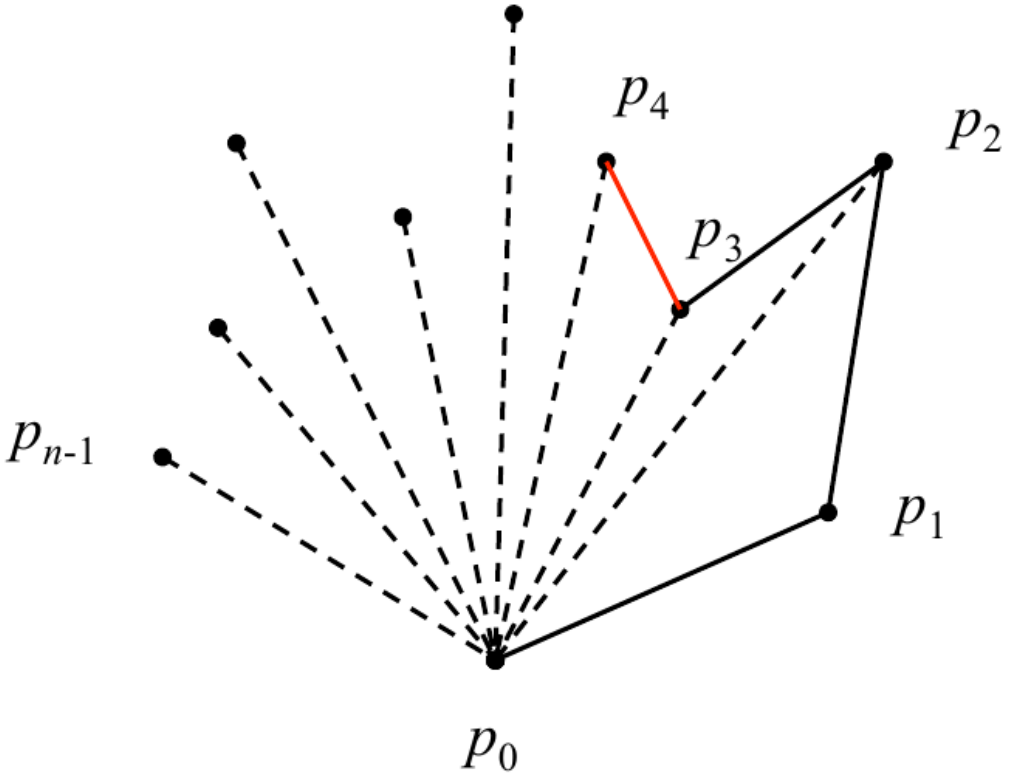
Varredura de Graham

- Os pontos restantes são ordenados de forma cíclica com respeito aos ângulos formados pelas retas p_0p_i.
- Pontos colineares são removidos nesse processo.



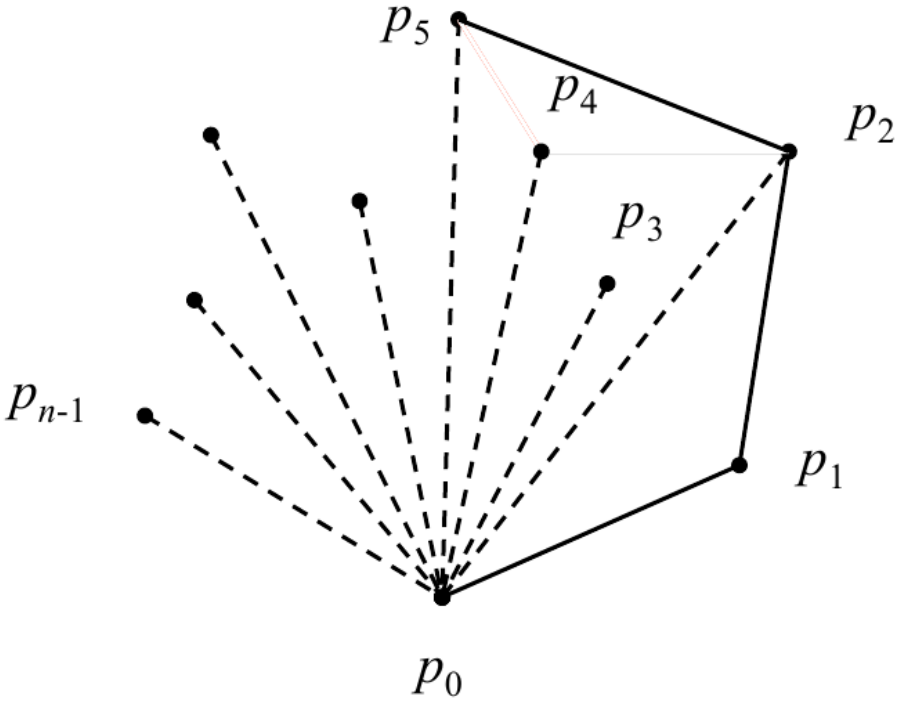
Varredura de Graham

- Cada ponto considerado deve estar à esquerda da aresta anteriormente computada (teste de orientação)
  - Ou o ponto anterior faz uma curva para esquerda.

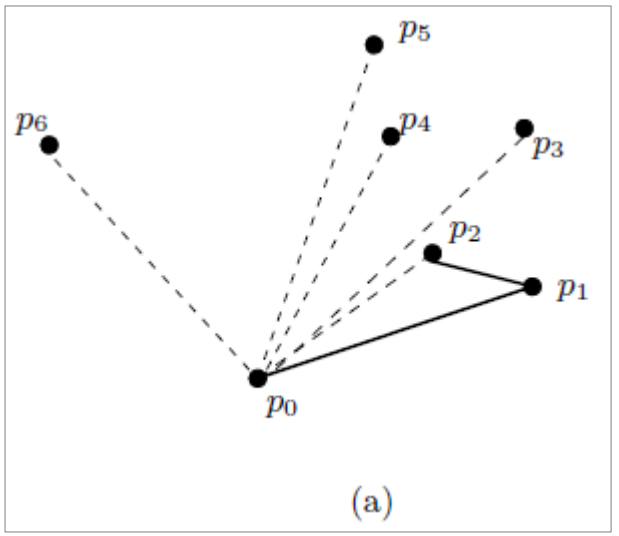


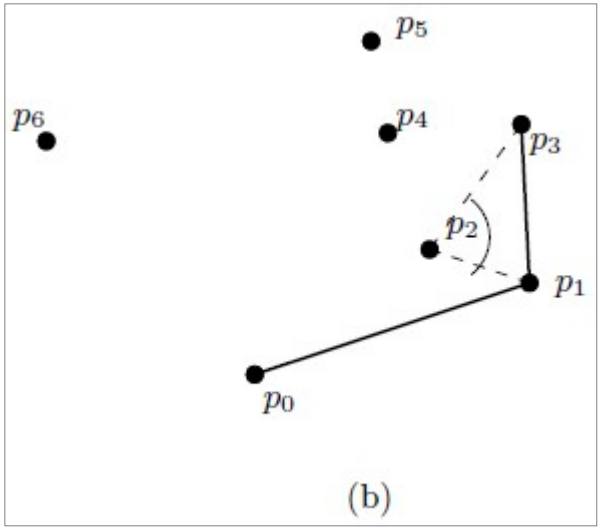
Varredura de Graham

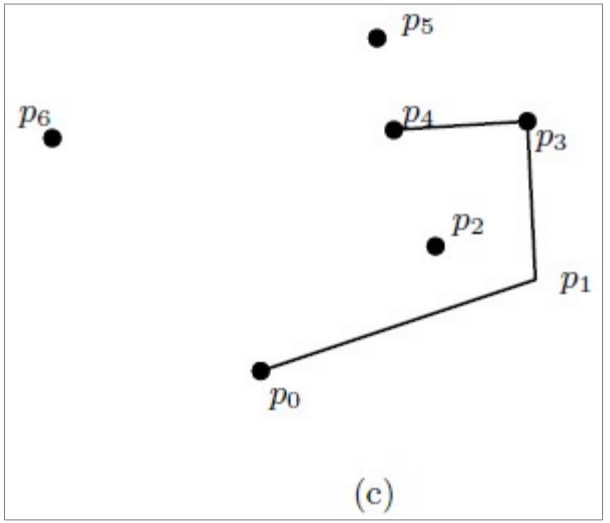
- O fecho convexo é mantido como uma pilha de pontos.
  - Percorre-se os pontos enquanto o ponto no topo da pilha não fizer uma curva para à esquerda,
  - Quando se considera o novo ponto, ele é desempilhado.
  - Em seguida estende-se a cadeia empilhando-se o novo ponto .

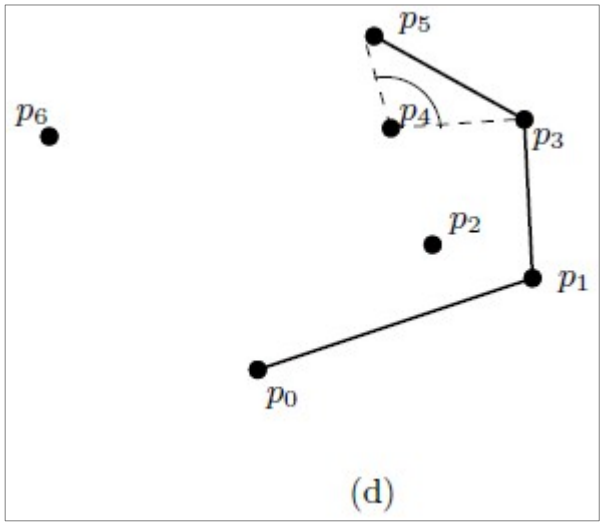


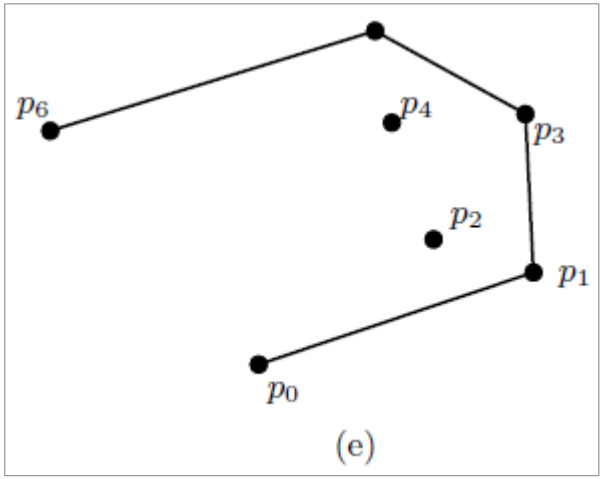
Varredura de Graham

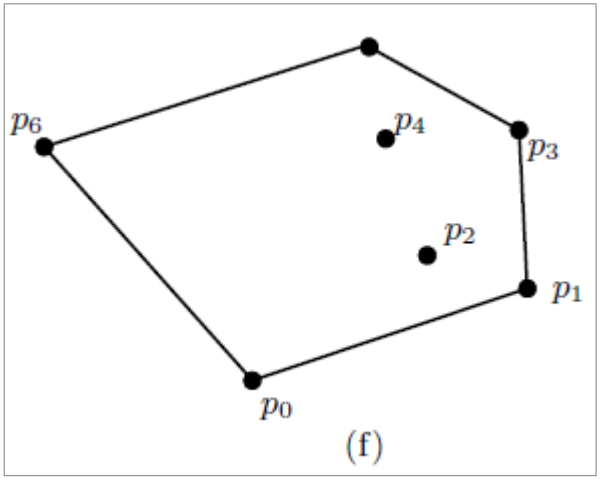












Triangulação

- Achar o perímetro de um polígono é fácil:
  - Basta calcular o comprimento de cada lado usando a fórmula da distância euclidiana e somar tudo ao final.
- Computar a área de “manchas” irregulares já é mais difícil.
- A ideia é dividir o polígono em triângulos não sobrepostos e somar suas áreas.
- Esta operação é chamada de triangulação.

Triangulação

- Triangular um polígono convexo é fácil:
  - basta conectar um dado vértice v a todos os outros (n - 1) vértices.
- Desta forma, o polígono é dividido em P triângulos usando arestas que não se cruzam e que se situam completamente dentro de P.
- Mas e quando o polígono não é convexo?

Triangulação

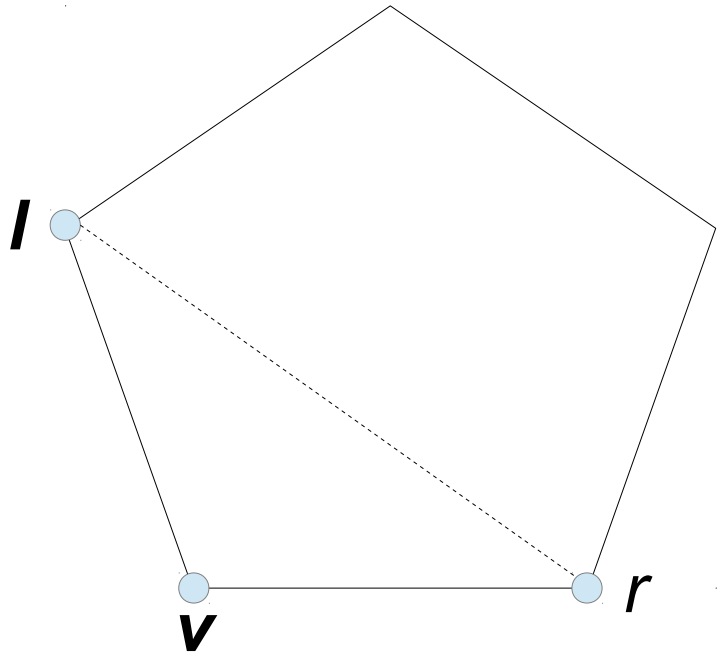
- Representação da triangulação:
  - Listando as arestas ou, como é feito aqui,
  - Com uma lista explícita dos índices dos vértices em cada triângulo:

```
typedef struct {
   int n;
   int t[MAXPOLY][3];
} triangulation;
```

- O algoritmo de triangulação mais eficiente “roda” em tempo linear do número de vértices.
- O algoritmo mais fácil de implementar é baseado no corte da orelha (**Algoritmo de Van Gogh**).

Algoritmo de Van Gogh

- Uma orelha de um polígono P é um triângulo definido por um vértice *v* e seus vizinhos da esquerda e da direita
  - (l e r), tal que o triângulo (v, l, r) se situa completamente dentro de P.
- Como *lv* e *vr* são arestas (limites) de P, o lado que define a orelha é *rl*.



Algoritmo de Van Gogh

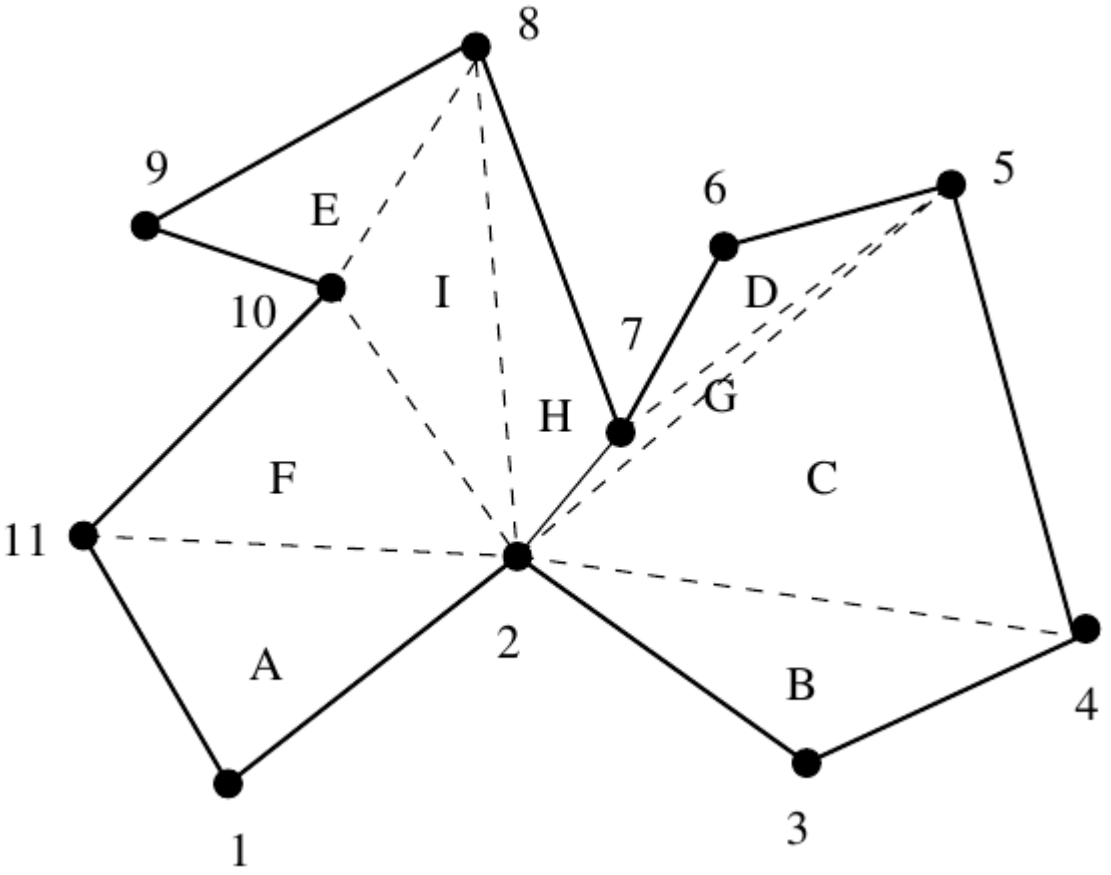
- *rl* deve se situar completamente no interior do polígono P.
  - Para isso, *lvr* deve definir um ângulo não-reflexo.
- Nenhum outro segmento do polígono pode ser cortado por este lado,
  - Senão um pedaço será retirado do triângulo.
- O fato importante é que todo polígono sempre contém uma orelha.
  - Na verdade, duas no mínimo para n ≥ 4.

Algoritmo de Van Gogh

- Algoritmo:
  - teste cada um dos vértices até achar uma orelha.
  - Adiciona-se a aresta associada e corta-se a orelha, decrementando o número de vértices.
- O corte da orelha necessita testar se um dado ponto se situa dentro de um triângulo.
- O polígono restante deve também ter uma orelha,
  - tal que continua-se a cortar até que apenas três vértices sejam mantidos,
  - ficando um triângulo.

Algoritmo de Van Gogh

- Deve-se testar se um vértice que define uma orelha tem duas partes.
- Para o teste do ângulo:
  - Utiliza-se novamente ccw/cw.
- Assume-se que os vértices estão ordenados em uma ordem anti-horária em volta do centro virtual.
- A reversão da ordem do polígono nos obriga a trocar o sinal no teste do ângulo.

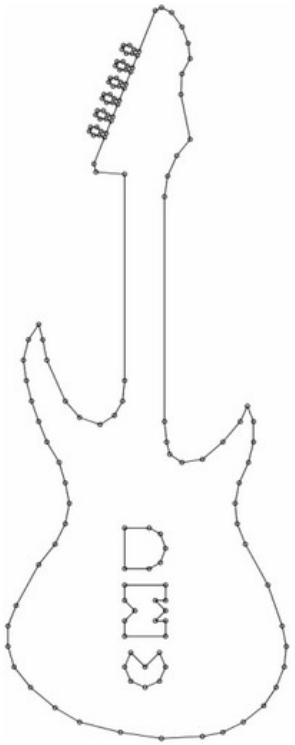
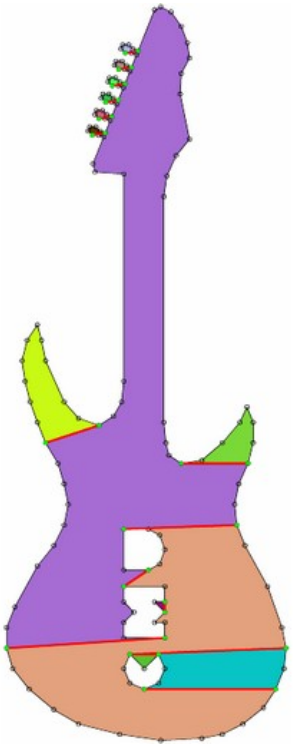
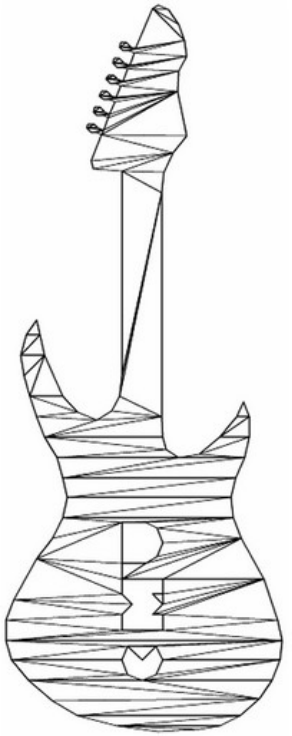
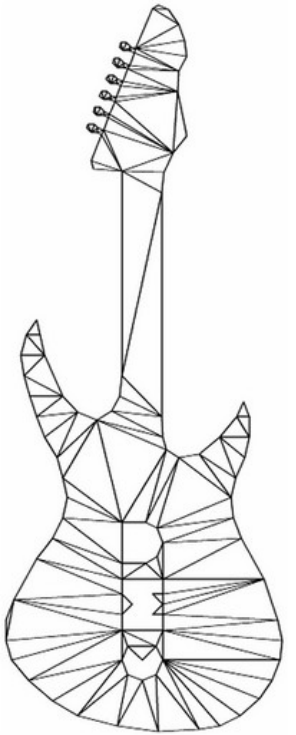


- Triangulação de um polígono via algoritmo de Van Gogh (corte da orelha), com triângulos rotulados na ordem da inserção (A-I)

Computação das áreas

- Pode-se então computar a área de qualquer polígono:
  - Basta triangular e somar as áreas de todos os triângulos formados.
- Somando as áreas com sinal dos triângulos
  - Estes definidos por um ponto arbitrário p com cada segmento do polígono P,
  - Chega-se na área de P, porque os triângulos com sinal negativo cancelam a área fora do polígono:

$$Área(P) = \frac{1}{2} \sum_{i=0}^{n-2} (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$$







http://sites-final.uclouvain.be/mema/Poly2Tri/

Problemas

- Fáceis:
  - *Chainsaw Massacre*;
  - *Useless Tile Packers*;
- Intermediários:
  - *Trees on My Island*;
- Difíceis:
  - *Nice Milk*.