Geometria Computacional para Maratona de Programação

Jéferson Coêlho

Instituto Tecgraf/PUC-Rio

jcoelho@tecgraf.puc-rio.br

October 25, 2015

Overview

1. Revisão - Conceitos Básicos
   - Vetores
   - Produtos Entre Veteores
   - Codificação
2. Interpolação Linear
3. Áreas
   - Área Orientada do Triângulo e Coordenadas Baricêntricas
   - Área De Polígonos
4. Interseções e Distâncias
5. Ângulos
   - Ângulo Entre dois vetores
   - Ponto em Polígono
6. Convex Hull

Revisão - Conceitos Básicos

Vetores

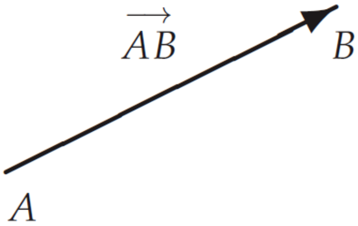


Figure : Dados dois pontos A e B, o vetor $\overrightarrow{V} = \overrightarrow{AB}$ é dado pela diferente B - A.

Vetores - Propriedades

Sejam $\overrightarrow{U}$, $\overrightarrow{V}$ e $\overrightarrow{W}$ três vetores e α e β escalares. Então são válidas as seguintes propriedades:

1. $\overrightarrow{U} + \overrightarrow{V} = \overrightarrow{V} + \overrightarrow{U}$
2. $(\overrightarrow{U} + \overrightarrow{V}) + \overrightarrow{W} = \overrightarrow{U} + (\overrightarrow{V} + \overrightarrow{W})$
3. $\alpha(\beta\overrightarrow{U}) = (\alpha\beta)\overrightarrow{U}$
4. $\alpha(\overrightarrow{U} + \overrightarrow{V}) = \alpha\overrightarrow{U} + \alpha\overrightarrow{V}$
5. $(\alpha + \beta)\overrightarrow{U} = \alpha\overrightarrow{U} + \beta\overrightarrow{U}$

Vetores - Multiplicação por escalar

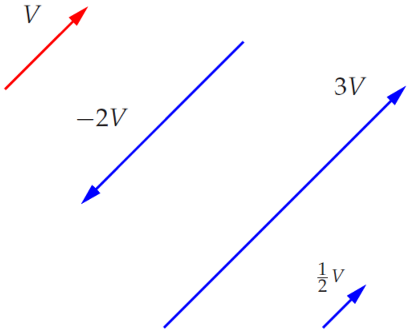


Figure : Ao multiplicar um vetor por um escalar, este pode ter sua norma e sentido alteradas, mas nunca sua direção.

Vetores - Soma e diferença de vetores

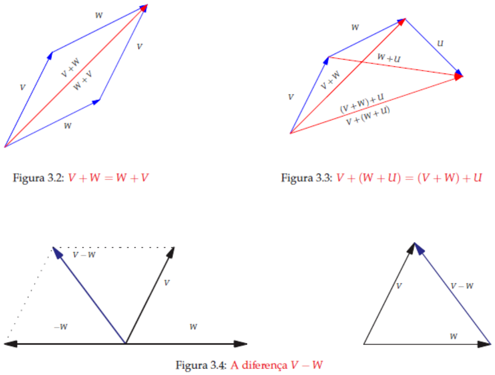


Figure : Soma e diferença entre vetores.

Vetores - Norma

Todo vetor que tem uma norma, onde este valor denota o seu comprimento. Dado um vetor $\overrightarrow{V} = (x, y, z)$, a sua norma é dada por:

$$d = ||\overrightarrow{V}|| = \sqrt{x^2 + y^2 + z^2} \tag{1}$$

No entanto, tentaremos sempre usar o quadrado da norma para evitar o uso de raízes, uma vez que esta operação é lenta e gera erros numéricos. A norma ao quadrado do vetor $\overrightarrow{V} = (x, y, z)$ é dada por:

$$d^2 = ||\overrightarrow{V}||^2 = x^2 + y^2 + z^2. \tag{2}$$

Um vetor unitário é aquele que tem norma igual a 1, logo:

$$\overrightarrow{V}^U = \frac{\overrightarrow{V}}{||\overrightarrow{V}||} \tag{3}$$

Vetores - Produto Escalar

Dados dois vetores $\overrightarrow{V} = (v_1, v_2, v_3)$ e $\overrightarrow{U} = (u_1, u_2, u_3)$, o produto escalar (*dot product*) entre eles é dado por:

$$\overrightarrow{V} \bullet \overrightarrow{U} = ||\overrightarrow{V}||.||\overrightarrow{U}||. \cos\theta \tag{4}$$

Onde θ é o menor ângulo formado pelos dois vetores.

O produto escalar pode ser definido ainda como:

$$\overrightarrow{V} \bullet \overrightarrow{U} = v_1 u_1 + v_2 u_2 + v_3 u_3 \tag{5}$$

O fato de o produto escalar poder ser definido de duas formas, faz com que essa operação seja de grande utilidade, uma vez que ela pode ser usada para encontrar ângulos, normas...

Vetores - Produto Escalar: Propriedades

Sejam $\overrightarrow{U}$, $\overrightarrow{V}$ e $\overrightarrow{W}$ três vetores e α e β escalares. Então são válidas as seguintes propriedades:

1. $\overrightarrow{U} \bullet \overrightarrow{V} = \overrightarrow{V} \bullet \overrightarrow{U}$
2. $\overrightarrow{U} \bullet (\overrightarrow{V} + \overrightarrow{W}) = \overrightarrow{U} \bullet \overrightarrow{V} + \overrightarrow{V} \bullet \overrightarrow{W}$
3. $\alpha(\overrightarrow{U} \bullet \overrightarrow{V}) = (\alpha\overrightarrow{U}) \bullet \overrightarrow{V} = \overrightarrow{U} \bullet (\alpha\overrightarrow{V})$
4. $\overrightarrow{V} \bullet \overrightarrow{V} = ||\overrightarrow{V}||^2$
5. $\overrightarrow{V} \bullet \overrightarrow{W} = 0 \Leftrightarrow \theta = 90^0$

Vetores - Projeção Ortogonal

A projeção ortogonal de um vetor $\overrightarrow{V}$ sobre um vetor $\overrightarrow{W}$ é dada por:

$$proj_{\overrightarrow{W}} \overrightarrow{V} = \frac{(\overrightarrow{V} \bullet \overrightarrow{W})\overrightarrow{W}}{||\overrightarrow{W}||^2} \tag{6}$$

Ou, de forma equivalente:

$$proj_{\overrightarrow{W}} \overrightarrow{V} = \frac{(\overrightarrow{V} \bullet \overrightarrow{W})\overrightarrow{W}}{\overrightarrow{W} \bullet \overrightarrow{W}} \tag{7}$$

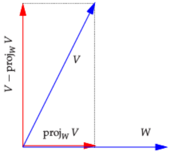


Figure : Projeção Ortogonal de Vetores.

Dados dois vetores $\overrightarrow{V} = (v_1, v_2, v_3)$ e $\overrightarrow{W} = (w_1, w_2, w_3)$ em $\mathbb{R}^3$, o produto vetorial (*cross product*) entre eles é dado por:

$$\overrightarrow{V} \times \overrightarrow{U} = \left(\begin{vmatrix} v_2 & v_3 \\ w_2 & w_3 \end{vmatrix}, - \begin{vmatrix} v_1 & v_3 \\ w_1 & w_3 \end{vmatrix}, \begin{vmatrix} v_1 & v_2 \\ w_1 & w_2 \end{vmatrix} \right) \tag{8}$$

Além disso, a norma do produto vetorial é dada por:

$$||\overrightarrow{V} \times \overrightarrow{W}|| = ||\overrightarrow{V}||.||\overrightarrow{W}||.\sin\theta \tag{9}$$

onde θ é o ângulo formado entre $\overrightarrow{V}$ e $\overrightarrow{W}$.

Note que a norma do produto vetorial é igual ao drobro da área do triângulo, logo:

$$Area_{triangulo} = \frac{||\overrightarrow{V} \times \overrightarrow{W}||}{2} \tag{10}$$

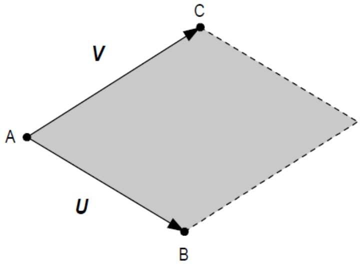


Figure : Metade da norma do produto vetorial retorna a área do triângulo.

Vetores - Produto Vetorial: propriedades

Dados os vetores $\overrightarrow{U}$, $\overrightarrow{V}$ e $\overrightarrow{W}$ no espaço e α escalar, as seguintes propriedades são válidas:

1. $\overrightarrow{V} \times \overrightarrow{W} = -\overrightarrow{W} \times \overrightarrow{V}$
2. $\overrightarrow{V} \times \overrightarrow{W} = 0 \Leftrightarrow \overrightarrow{V} = \alpha\overrightarrow{W}$, ou seja, os três pontos que formam o vetor estão sobre a mesma reta.
3. $(\overrightarrow{V} \times \overrightarrow{W}) \bullet \overrightarrow{V} = (\overrightarrow{V} \times \overrightarrow{W}) \bullet \overrightarrow{W} = 0$
4. $\alpha(\overrightarrow{V} \times \overrightarrow{W}) = (\alpha\overrightarrow{V}) \times \overrightarrow{W} = \overrightarrow{V} \times (\alpha\overrightarrow{W})$
5. $\overrightarrow{V} \times (\overrightarrow{U} + \overrightarrow{W}) = \overrightarrow{V} \times \overrightarrow{U} + \overrightarrow{V} \times \overrightarrow{W}$

Para utilizar o produto vetorial em 2D, basta supor que se esteja no plano $z = 0$, uma vez que esta operação é definida apenas para $\mathbb{R}^3$. Neste caso, o produto vetorial pode ser redefinido como:

$$\vec{V} \times \vec{U} = \left(\begin{vmatrix} v_2 & 0 \\ w_2 & 0 \end{vmatrix}, - \begin{vmatrix} v_1 & 0 \\ w_1 & 0 \end{vmatrix}, \begin{vmatrix} v_2 & v_3 \\ w_2 & w_3 \end{vmatrix} \right) = (0, 0, v_1 w_2 - v_2 w_1) \tag{11}$$

Logo, em 2D, o produto vetorial retorna um escalar, assim:

$$\vec{V} \times \vec{U} = v_1 w_2 - v_2 w_1 \tag{12}$$

Esse escalar é positivo se obedecida a regra da mão direita, e negativo caso contrário.

Dados os vetores $\overrightarrow{U}$, $\overrightarrow{V}$ e $\overrightarrow{W}$ no espaço, o produto misto é um escalar que representa o volume do paralepípedo determinado pelos três vetores. Este produto **é comumente utilizado para verficar se 4 pontos estão no mesmo plano**. O produto misto é dado por:

$$(\overrightarrow{V} \times \overrightarrow{W}) \bullet \overrightarrow{U} \tag{13}$$

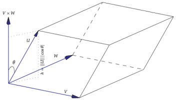


Figure : Paralelepípedo formado por três vetores.

Quando o produto misto de três vetores é igual a zero, os quatro pontos que geraram o vetor estão no mesmo plano.

Vetores - Codificação

Toda esta álgebra básica **deve estar codificada em uma pequena classe de forma simples a ser usada**, com sobrecarga de operadores.

A função **cmp** encapsula a comparação de dois doubles, seguindo a convenção adotada para strings. Desta forma a função retorna -1 se $x < y$, 0 se $x = y$ e 1 se $x > y$. **Esta função deve sempre ser usada ao comparar dois doubles**.

Atenção para a tolerância. Ela pode mudar dependendo do problema.

Código global:

```
const double EPS = 1e-10;
int cmp( double x, double y = 0, double tol = EPS )
{
    return (x <= y + tol ) ? ( x + tol < y ) ? -1 : 0 : 1;
}
```

Operações básica de algebra:

```
struct Point
{
    double x, y;

    Point( double x = 0, double y = 0 ) : x( x ), y( y ){}//Construtor
    Point operator+( Point q ){return Point( x + q.x, y + q.y );}//Soma
    Point operator-( Point q ){return Point( x - q.x, y - q.y );}//Diferenca
    Point operator*( double t ){return Point( x * t, y * t );}//Mult. escalar
    Point operator/( double t ){return Point( x / t, y / t );}//Div. escalar
    double operator*( Point q ){return x * q.x + y * q.y;}//Produto escalar
    double operator^( Point q ){return x * q.y - y * q.x;}//Produto Vetorial.
```

Funções utilitarias e de debug:

```
    int cmp( Point q ) const
    {
        if ( int t = ::cmp( x, q.x ) ) return t;
        return ::cmp( y, q.y );
    }
    bool operator==( Point q ) const {return cmp( q ) == 0;}
    bool operator!=( Point q ) const {return cmp( q ) != 0;}
    bool operator<( Point q ) const {return cmp( q ) < 0;}
    friend ostream& operator<<( ostream& o, Point p )
    {
        return o << "(" << p.x << ", " << p.y << ")";
    }
    static Point pivot;
};
Point Point::pivot;
typedef vector<Point> Polygon;
```

Interpolação Linear

Dados pois pontos P_1 e P_2 quaisquer, a reta que passa por eles (mesmo aquelas verticais) pode ser definida por:

$$P(t) = (1 - t)P_1 + tP_2. \tag{14}$$

Por vezes essa equação é rearranjada da seguinte forma:

$$P(t) = P_1 + t(P_2 - P_1). \tag{15}$$

Estas equações são particularmente uteis para definir um segmento de reta. Note que para $P(0) = P_1$ e $P(1) = P_2$. Além disso, uma orientação é gerada pela reta, fazendo com que antes de P_1 $t < 0$ e depois de P_2, $t > 1$.

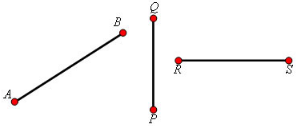


Figure : Segmentos de reta.

Áreas

Área orientada do triângulo

Como já sabemos, a área de um triângulo pode ser dada pelo produto vetorial. Mas em 2D, o produto vetorial é um escalar, e este escalar possui um sinal que está relacinado com a ordem da aplicação do produto vetorial.

No triângulo abaixo, $(\overrightarrow{C-A}) \times (\overrightarrow{B-A}) > 0$ e $(\overrightarrow{B-A}) X (\overrightarrow{C-A}) < 0$, mas ambos tem o mesmo módulo. **Esta polaridade de sinal é a base de muitos algoritmos.**

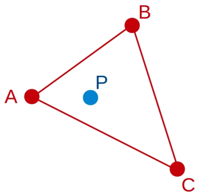


Figure : Area do triângulo.

Coordenadas Baricêntricas

Problema: como determinar se um ponto P está dentro, fora, sobre uma arestas ou sobre um ponto do triângulo ABC?

Se forem levados em considerações os triângulos PBC, APC e ABP, a soma de suas áreas tem que ser equivalente à soma da área do triângulo ABC. Assim:

$$S(PBC) + S(APC) + S(ABP) = S(ABC) \tag{16}$$

Dividindo todas a equação pela área total:

$$\frac{S(PBC)}{S(ABC)} + \frac{S(APC)}{S(ABC)} + \frac{S(ABP)}{S(ABC)} = 1 \tag{17}$$

Isso gera um sistema de coordendas, chamado de coordenadas bariêntricas, onde:

$$\lambda_1 = \frac{S(PBC)}{S(ABC)} \qquad \lambda_2 = \frac{S(APC)}{S(ABC)} \qquad \lambda_3 = \frac{S(ABP)}{S(ABC)} \tag{18}$$

Logo:

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \tag{19}$$

Todo ponto no plano pode ser escrito como uma combinação linear das coordenadas baricêntricas com os vérticees do triângulo:

$$P = \lambda_1 A + \lambda_2 B + \lambda_3 C \tag{20}$$

Para calcular as coordenadas baricêntricas, basta calcular áreas, logo uma função como a abaixo pode ser definida para fazer isso.

```
double area( Point a, Point b, Point c)
{
    return (c - a) ^ (b - a) /  2.0
}
```

Coordenadas Baricêntricas

Uma das aplicações mais elegantes das coordenadas baricêntricas, é que apenas analisando os seus sinais, sabe-se exatamente onde o ponto se encontra no plano.

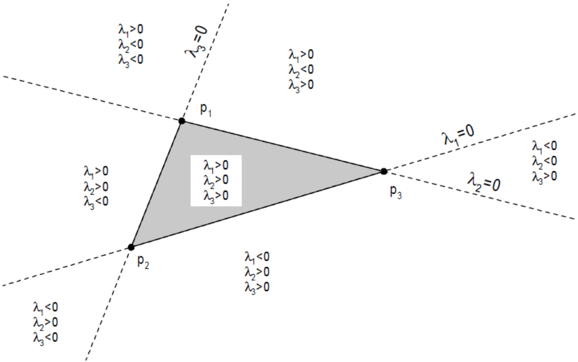


Figure : Subdivisão do plano induzida pelas coordenadas baricêntricas.

Exemplo

Uva Online Judge 143 - Orchard Trees

Área de Polígonos Convexos

Problema: Sabendo calcular a área de um triângulo, como calcular a área de um polígono convexo, como o abaixo:

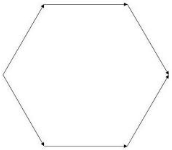


Figure : Polígono Regular

Área de Polígonos Convexos

Adiciona um ponto no meio e soma as áreas de todos os triângulos:

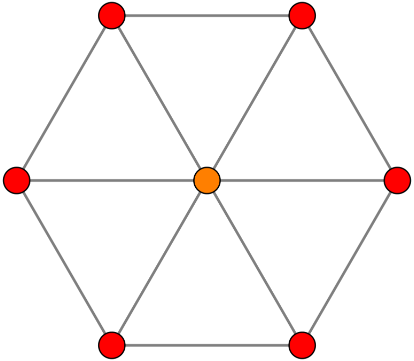


Figure : Polígono Regular

Área de Polígonos Convexos

O ponto adicionado precisa necessariamente estar dentro do polígono?

Área de Polígonos Convexos

O ponto adicionado precisa necessariamente está dentro do polígono?

Qual o melhor ponto a ser escolhido então??

O ponto adicionado precisa necessariamente está dentro do polígono?

Qual o melhor ponto a ser escolhido então??

Funciona para qualquer qualquer polígono??

A área de qualquer polígono pode ser dada por:

$$A = \frac{1}{2}\left|\sum_{i=0}^{n-1} p[i] \times p[(i+1)\%n]\right| \tag{21}$$

Código:

```
double polygonArea( Polygon& T)
{
    double s = 0.0;
    int n = T.size();
    for (int i = 0; i < n; i++)
    {
        s += T[i] ^ T[(i + 1) % n];
    }
    return s / 2.0; //Retorna a area com sinal
}
```

Interseções e Distâncias

Dado um ponto. Como saber se o ponto está sobre uma reta ou não?

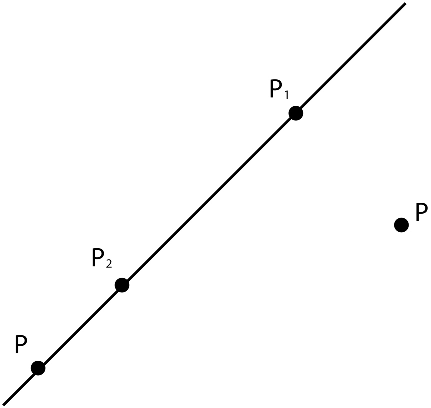


Figure : Ponto Sobre Reta

Ponto Sobre Reta

Se a área do triângulo formado pelos três ponto for nula, significa que o ponto está sobre a reta, ou seja:

$$Se\ (P - P_1) \times (P_2 - P_1) == 0 \tag{22}$$

Código:

```
inline int ccw( Point& p, Point &q, Point& r )
{
    return cmp((p - r) ^ (q - r));
}
bool pontoSobreReta( Point& p1, Point &p2, Point& p)
{
    return ccw( p1, p2, p ) == 0;
}
```

Dado um ponto. Como saber se o ponto está sobre um segmento de reta ou não?

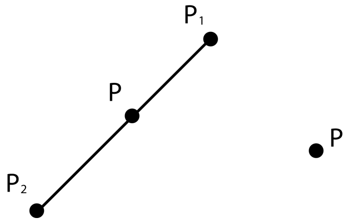


Figure : Ponto Sobre Segmento de Reta

Ponto Sobre Reta

Sabemos verificar se o ponto está sobre a reta. Para verificar se ele está sobre o segmento de reta, supondo que a primeira condição está satisfeita, basta verificar o ângulo entre os vetores $\overrightarrow{P_1 - P}$ e $\overrightarrow{P_2 - P}$. Note que se o ponto for interno, o ângulo é igual a 180 graus e 0 caso contrário.

Analisando o produto escalar, ele será negativo no primeiro caso e positivo no segeundo. Além disso, é zero quando o ponto P coincide com P_1 ou P_2. Logo:

```
bool between( Point& p1, Point &p, Point& p2)
{
    return ccw( p1, p2, p ) == 0 && cmp((p1 -  p) * (p2 - p)) <= 0;
}
```

Interseções e Distâncias

Dado um ponto. Como saber a distância do ponto uma reta?

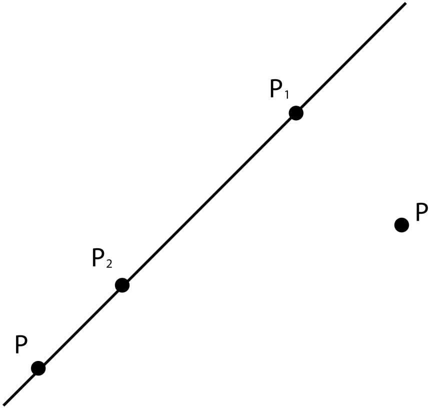


Figure : Distância de ponto à reta

Ponto Sobre Reta

Basta calcular a altura do triângulo formado pelos três pontos. Como sabe-se a base a área, basta fazer:

$$A = \frac{bh}{2} \tag{23}$$

$$h = 2 * A/b \tag{24}$$

$$h = \frac{|(p1 - p) \times (p2 - p1)|}{(p2 - p1) \bullet (p2 - p1)} \tag{25}$$

Código:

```
bool distanciaReta( Point& p1, Point &p2, Point& p)
{
    Point A = p1 - p, B = p2 - p1;
    return fabs(A ^ B) / sqrt( B * B );
}
```

Dado um ponto. Como a distância do ponto a um segmento de reta?

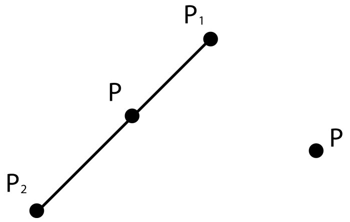


Figure : Distância de um ponto a um segmento de reta

Ponto Sobre Reta

Parecido com o anterior, mas deve-se verificar se a menor distância é a altura do triângulo ou um dos seus lados. Será um dos lados se algum dos ângulos da base for maior que 90 graus. Ou seja, se:

$$a^2 > b^2 + c^2 \tag{26}$$

onde a é o valor de um lado que não seja $||\overrightarrow{P_1 P_2}||$.

```
bool distanciaReta( Point& p1, Point &p2, Point& p)
{
    Point A = p1 - p, B = p2 - p1, C = p1 - p;
    double a = A * A, b = B * B, c = C * C;
    if (cmp(a, b + c) >= 0) return sqrt( c );
    else if (cmp(c, a + b) >= 0) return sqrt( a );
    else return fabs(A ^ C) / sqrt( b );
}
```

Ângulos

Ângulo Entre dois vetores

Apesar de simples, o cálculo de ângulos entre vetores é uma operação extremamente custosa. Uma forma simples de calcular um ângulo entre dois vetores é:

$$\theta = \arccos\left(\frac{\overrightarrow{V} \bullet \overrightarrow{W}}{||\overrightarrow{V}||.||\overrightarrow{W}||}\right) \tag{27}$$

Além de duas raízes, existe uma função trigonométrica inversa, que é bastante custosa e ainda por cima não dá qualquer orientação ao ângulo. Logo, normalmente, essa abordagem não é indicada para cálculos de ângulos.

Uma alternativa é a função *atan*2, que calcula o ângulo entre um vetor e eixo x, como ângulo resultante variando no intervalo de $-\pi$ a π.

Ângulo Entre dois vetores

Normalmente, em aplicações de geometria computacional, buscamos calcular o ângulo orientado, ou seja, o ângulo com o mesmo sinal do produto vetorial.

Suponha dois vetores $\overrightarrow{V}$ e $\overrightarrow{W}$. Sabemos que:

$$\overrightarrow{V} \bullet \overrightarrow{W} = ||\overrightarrow{V}||.||\overrightarrow{W}||.\cos\theta \tag{28}$$

$$\overrightarrow{V} \times \overrightarrow{W} = ||\overrightarrow{V}||.||\overrightarrow{W}||.\sin\theta \tag{29}$$

Note que podemos construir um vetor

$$A = (\overrightarrow{V} \bullet \overrightarrow{W}, \overrightarrow{V} \times \overrightarrow{W}) = (||\overrightarrow{V}||.||\overrightarrow{W}||.\cos\theta, ||\overrightarrow{V}||.||\overrightarrow{W}||.\sin\theta)$$

Se dividirmos ese vetor por $||\overrightarrow{V}||.||\overrightarrow{W}||$, seu ângulo com o vetor eixo x não muda. Logo, $B = (\cos\theta, \sin\theta)$ e A tem o mesmo ângulo com o eixo ângulo com o eixo x.

Ângulo Entre dois vetores

Desta forma, o ponto B, é o ponto sobre o círculo de raio unitário e ângulo *theta* em relação ao eixo x, logo, achar o ângulo desse vetor com o eixo o x, é equivalente a encontrar o ângulo orientado entre dois vetores $\overrightarrow{V}$ e $\overrightarrow{W}$. Este pode ser calculado da seguinte forma:

```
double angle( Point& p, Point &q, Point& r)
{
    Point u = p - r, w = q - r;
    return atan2( u ^ w, u * w );
}
```

Note que quando o produto vetorial for negativo, o ponto está no lado negativo de y, logo o ângulo será negativo.

Ponto em polígono

Um dos problemas mais corriqueiros em geometria computacional é determinar se um ponto está dentro, sobre ou fora de um polígono qualquer no plano.

Existem duas abordagens: teorema da curva de Jordan e por ângulos.

Em 2D, a abordagem por ângulos é mais simples. Se um ponto está dentro de um polígono, a soma dos seus ângulos orientados, ou seja, a soma de todos os ângulos formado por ele e cada par de pontos de cada aresta, é sempre igual a 2π. Se o ponto estiver fora, está soma é sempre nula.

Esta definição não vale para quando o ponto estiver sobre a aresta, mas neste caso, basta testar se o ponto está sobre o segmento de reta da aresta.

Ponto em polígono

Dito isso, o algoritmo fica da seguinte forma:

```
//-1 sobre, 0 dentro, 1 fora
int inpoly(Point& p, Polygon& T)
{
    double a = 0.0; int n = T.size();
    for (int i = 0; i < n; i++)
    {
        if (between(T[i], p, T[(i + 1) % n])) return -1;
        a += angle(T[i], p, T[(i + 1) % n]);
    }
    return cmp(a) != 0;
}
```

Exemplo

Uva Online Judge 881 - Points, Polygons and Containers

Convex Hull

Convex Hull

O problema de encontrar o feixo convexo de um conjunto de pontos P, consiste em encontrar um subconjunto de pontos tal que todos os conjuntos em P possam ser escritos como uma combinação convexa deste subconjunto.

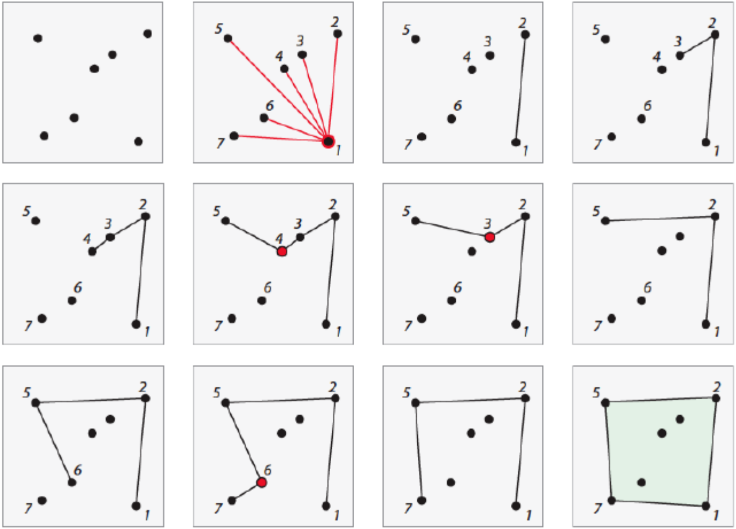
O problema de Convex Hull pode reduzido ao problema de ordenação, logo, para cada algoritmo de ordenação, existe um algoritmo de convex hull equivalente. Além desses, existem mais alguns...

O melhor algoritmo, em termos de custo benefície, é chamado de Grahan Scan.

Algoritmo de Grahan Scan:

1. Obtenha um elemento que certamente esteja no feixo convexo. Vamos assumir o elemento de menor x.
2. Ordene os pontos com base no angulo (com sinal) para o eixo x. Isso força com que os pontos sejam ordenados no sentido anti-horário.
3. A cada ponto adicionado, se o produto vetorial dos ultimos dois vetores for negativo, remova pontos anteriores ate que ele se torne positivo.

Grahan Scan



Grahan Scan

Neste algoritmo é provável que o cálculo do ângulo provoque um *time out*.

Note que não precisamos saber de fato o valor de cada ângulo, apenas os valores relativos para ordenação.

Note que isso pode ser feito usando apenas o produto vetorial. Se o produto vetorial entre dois vetores, em relação ao pivô, for positivo significa que os pontos estão na ordem correta, ser der negativo os dois pontos devem ser trocados. Esse algoritmo é conhecido como ordenação radial.

```
bool radial( Point& p, Point& q )
{
    Point P = p - Point::pivot, Q = q - Point::pivot;
    double R = P ^ Q;
    if ( cmp( R ) ) return R > 0;
    return cmp( P * P, Q * Q ) < 0;
}
```

Grahan Scan

Usando esta ordenação, o algoritmo pode ser implementado da seguinte forma:

```
Polygon convexHull( vector<Point>& T )
{
    int j = 0, k, n = T.size( );
    Polygon U( n );
    Point::pivot = *min_element( T.begin( ), T.end( ) );
    sort( T.begin( ), T.end( ), radial_lt );
    for ( k = n - 2; k >= 0 && ccw( T[0], T[n - 1], T[k] ) == 0; k-- );
    reverse( ( k + 1 ) + T.begin( ), T.end( ) );
    for ( int i = 0; i < n; i++ )
    {
        // troque o >= por > para manter pontos colineares
        while (j > 1 && ccw( U[j - 1], U[j - 2], T[i] ) >= 0)j--;
        U[j++] = T[i];
    }
    U.resize( j );
    return U;
}
```

Exemplo

Uva Online Judge 681 - Convex Hull Fiding

Erro númerico devido ao truncamento da mantissa

Uva Online Judge 10209 - Is this Integration?

End