Geometria Computacional

Retas e Vetores

Prof. Edson Alves

2018

Faculdade UnB Gama

Sumário

Definição de reta

Definição

- Reta também é um elemento primitivo da Geometria
- No primeiro livro dos elementos, Euclides define reta como "*a line is breadthless length*", que numa tradução livre diz que "linha é comprimento sem largura"
- As linhas são elementos unidimensionais
- Em C/C++, pontos podem ser representados através ou de sua equação geral, ou de sua equação reduzida
- A equação reduzida de uma reta é a mais conhecida e utilizada nos cursos de ensino médio
    1. tem a vantagem de facilitar comparações entre retas e identificar paralelismo
    2. não é capaz de representar retas verticais
- A equação geral, como o próprio nome diz, pode representar qualquer reta do plano

Equação reduzida da reta

- A equação reduzida da reta é dada por

$$y = mx + b,$$

  onde m é o coeficiente angular da reta e b é o coeficiente linear da reta
- O primeiro coeficiente representa a taxa de variação da reta: consiste no número de unidades que y varia para cada unidade de variação de x no sentido positivo do eixo horizontal
- O segundo coeficiente é o valor no qual a reta intercepta o eixo y

```cpp
template<typename T>
struct Line {
    T m, b;

    Line(T mv, T bv) : m(mv), b(bv) {}
};
```

Equação reduzida a partir de dois pontos dados

- Dados dois pontos $P = (x_p, y_p)$ e $Q = (x_q, y_q)$ tais que $x_p \neq y_p$, a inclinação da reta é dada por

$$m = \frac{y_q - y_p}{x_q - x_p}$$

- Deste modo, a equação reduzida da reta será dada por

$$y = m(x - x_p) + y_p = mx + (y_p - mx_p)$$

- Se $x_p = y_p$, a reta é vertical
- Retas verticais podem ser tratadas por meio de uma variável booleana que indica se a reta é vertical ou não
- Caso seja, o coeficiente b indica o ponto que a reta intercepta o eixo horizontal

Implementação da reta através da equação reduzida

```cpp
// Definição da função de comparação equals e da classe Point

template<typename T>
struct Line {
    T m, b;
    bool vertical;

    Line(const Point& P, const Point& Q) : vertical(false)
    {
        if (equals(P.x, Q.x))
        {
            vertical = true;
            b = P.x;
        } else
        {
            m = (Q.y - P.y)/(Q.x - P.x)
            b = P.y - m * P.x
        }
    }
};
```

Equação geral da reta

- A equação geral da reta é dada por

$$ax + by + c = 0$$

- Como dito, a equação geral pode representar retas verticais ($b = 0$)
- Nos demais casos, é possível obter a equação reduzida a partir da equação geral

```cpp
template<typename T>
struct Line {
    T a, b, c;

    Line(T av, T bv, T cv) : a(av), b(bv), c(cv) {}
};
```

Equação geral da reta a partir de dois pontos

- Dados dois pontos distintos $P = (x_p, y_p)$ e $Q = (x_q, y_q)$, é possível obter os coeficientes da equação geral da seguinte maneira:
    1. Substitua as coordenadas de um dos dois pontos na equação geral
    2. Encontrado o valor de c, substitua as coordenadas de ambos pontos na equação geral, obtendo um sistema linear
    3. Os valores de a e b são a solução deste sistema linear
- Este processo pode ser simplificado através do uso de Álgebra Linear: se três pontos $P = (x_p, y_p), Q = (x_q, y_q)$ e $R = (x, y)$ são colineares (isto é, pertencem a uma mesma reta), então

$$\det \begin{bmatrix} x_p & y_p & 1 \\ x_q & y_q & 1 \\ x & y & 1 \end{bmatrix} = 0$$

- A soluçaõ da equação acima é

$$a = y_p - y_q, b = x_q - x_p, c = x_p y_q - x_q y_p$$

Implementação da reta através da equação geral

```cpp
// Definição da classe Point

template<typename T>
struct Line {
    T a, b, c;

    Line(const Point& P, const Point& Q)
        : a(P.y - Q.y), b (Q.x - P.x), c(P.x * Q.y - Q.x * P.y)
    {
    }
};
```

Observações sobre a equação geral da reta

- Um mesma reta pode ter infinitas equações gerais associadas: basta multiplicar toda a equação por uma número real diferente de zero
- Para normalizar a representação, associando uma única equação a cada reta, é necessário dividir toda a equação pelo coeficiente a (ou por b, caso a seja igual a zero)
- Esta estratégia permite a simplificação de algoritmos de comparação entre retas
- Por outro lado, não uniformizar a representação permite manter os coeficientes inteiros, caso as coordenadas dos pontos sejam inteiras
- Importante notar que, em ambas representações, pode acontecer da reta resultante ser degenerada
- Isto ocorre quando os pontos P e Q são idênticos: neste caso, a reta se reduz a um único ponto
- O tratamento deste caso especiais nos demais algoritmos aumenta o tamanho e a sofisticação dos códigos
- Porém o não tratamento de casos especiais pode levar ao WA

Relação entre ponto e reta

- Seja r uma reta com equação geral $ax + by + c = 0$ e $P = (x_p, y_p)$ um ponto qualquer
- $P \in r$ se, e somente se, $ax_p + by_p + c = 0$
- Esta relação pode ser verificada diretamente a partir da equação geral da reta, ou através do determinante apresentado anteriormente, conhecidos dois pontos Q e R da reta

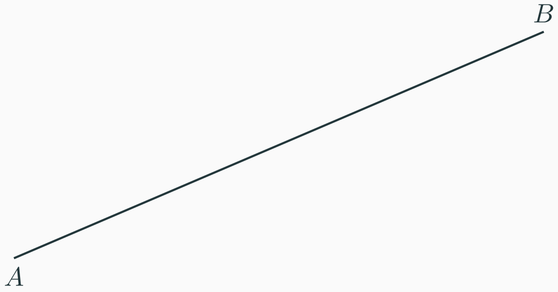
```cpp
// Definição da classe Point e da função de comparação equals

template<typename T>
struct Line {
    // Membros e construtor

    bool contains(const Point& P) const
    {
        return equals(a*P.x + b*P.y + c, 0);
    }
};
```

Segmentos de reta

- Sejam A e B dois pontos pertencentes à reta r. O segmento de reta AB é o conjunto de pontos de r que estão entre os pontos A e B
- O comprimento de um segmento de reta é a distância entre os pontos A e B



Distância entre dois pontos

- A definição de distância depende da norma utilizada
- A distância euclidiana entre dois pontos $A = (x_a, y_a)$ e $B = (x_b, y_b)$ é dada por

$$d(A, B) = \sqrt{(x_a - y_a)^2 + (x_b - y_b)^2}$$

- A distância do motorista de táxi é dada por

$$d(A, B) = |x_a - x_b| + |y_a - y_b|$$

- Segunda a norma do máximo, a distância entre A e B é dada por

$$d(A, B) = \max\{|x_a - x_b|, |y_a - y_b|\}$$

Observações sobre distância entre dois pontos

- Embora a distãncia euclidiana seja a mais comum, a raiz quadrada que aparece na sua definição leva a resultados em ponto flutuante
- Por este motivo, em geral é implementada a função que computa o quadrado da distância, o que elimina a raiz quadrada e permite a aritmética de inteiros
- O quadrado da distância pode ser usado para comparar os pontos por distância, pois ela preserva esta relação
- A distância do motorista de táxi considera que os movimentos no plano só podem ser feitos na horizontal e vertical
- Um exemplo prático da norma do máximo acontece no tabuleiro do xadrez: ela define o raio de ação do rei (todas as casas que estão a uma unidade de distância dele)

Exemplo de implementação das distâncias

```
#include <cmath>
#include <iostream>

template<typename T>
struct Point {
    T x, y;

    Point(T xv = 0, T yv = 0) : x(xv), y(yv) {}
};

template<typename T>
double dist(const Point<T>& P, const Point<T>& Q)
{
    return hypot(P.x - Q.x, P.y - Q.y);
}

template<typename T>
T dist2(const Point<T>& P, const Point<T>& Q)
{
    return (P.x - Q.x)*(P.x - Q.x) + (P.y - Q.y)*(P.y - Q.y);
}
```

Exemplo de implementação das distâncias

```
template<typename T>
T taxicab(const Point<T>& P, const Point<T>& Q)
{
    if (std::is_floating_point<T>::value)
        return fabs(P.x - Q.x) + fabs(P.y - Q.y);
    else
        return llabs(P.x - Q.x) + llabs(P.y - Q.y);
}

template<typename T>
T max_norm(const Point<T>& P, const Point<T>& Q)
{
    if (std::is_floating_point<T>::value)
        return std::max(fabs(P.x - Q.x), fabs(P.y - Q.y));
    else
        return std::max(llabs(P.x - Q.x), llabs(P.y - Q.y));
}
```

Exemplo de implementação das distâncias

```cpp
int main()
{
    Point<int> P, Q(2, 3);

    std::cout << "Euclidiana: " << dist(P, Q) << '\n';
    std::cout << "Quadrado: " << dist2(P, Q) << '\n';
    std::cout << "Motorista de táxi: " << taxicab(P, Q) << '\n';
    std::cout << "Norma do máximo: " << max_norm(P, Q) << '\n';

    return 0;
}
```

Vetores

Definição de vetores

- Vetores são segmentos de retas orientados
- Os vetores são caracterizados pela sua direção (inclinação da reta que contém o segmento), orientação (ponto de partida e de chegada) e tamanho (distância entre os dois pontos)
- Dois vetores são iguais apenas se coincidirem nestas três características
- Dados dois pontos A e B, $\vec{u} = \overrightarrow{AB}$ é o vetor que parte do ponto A em direção ao ponto B
- Observe que $\overrightarrow{AB}$ e $\overrightarrow{BA}$ tem mesma direção e comprimento, mas orientações distintas)

Vetor posição

- O vetor posição de um ponto P é o vetor que une a origem O ao ponto P ($\overrightarrow{OP}$)
- Na prática, trabalha-se apenas com vetores-posição: o vetor posição que equivale ao vetor $\overrightarrow{AB}$ é o vetor $\vec{v} = (x_b - x_a, y_b - y_a)$
- Deste modo, embora seja possível definir um tipo de dado para representar vetores, é possível utilizar pontos para representar vetores
- Esta estratégia pode dificultar a leitura das rotinas, pois embora usem a mesma memória a semântica é diferente
- Há porém a vantagem da velocidade de codificação, devido a eliminação de código redundante

Exemplo de implementação de vetores em C++

```cpp
// Definição da classe Point

template<typename T>
struct Vector
{
    T x, y;

    Vector(T xv, T yv) : x(xv), y(yv) {}

    Vector(const Point& A, const Point& B)
        : x(B.x - A.x), y(B.y - A.y) {}
};
```

Direção de um vetor

- A direção de um vetor pode ser caracterizada também pelo ângulo que vector posição equivalente faz com o eixo-x positivo
- Este ângulo pode ser computado pela função `atan2` da biblioteca `cmath`
- Esta função recebe dois parâmetros: a coordenada y e a coordenada x do vetor posição
- Esta função não lança exceções nem erros e tem retorno no intervalo $[-\pi, \pi]$
- A função `atan` difere no número de argumentos e no intervalo do retorno

Translações

- Um ponto P pode ser transladado no espaço, conhecidos os deslocamentos dx e dy nas direções paralelas aos eixos x e y, respectivamente
- Transladar ambos pontos que delimitam o vetor mantém o vetor inalterado
- Contudo, transladar apenas o ponto final P de um vetor posição pode alterar todas as três características de um vetor

```cpp
// Definição da estrutura Point

template<typename T>
Point<T> translate(const Point<T>& P, T dx, T dy)
{
    return Point<T> { P.x + dx, P.y + dy };
}
```

Rotações

- Um vetor posição pode ser rotacionado em θ graus no sentido anti-horário através da multiplicação da matriz de rotação R_θ e o vetor $\vec{v}$, onde

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \quad \vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$$

- Esta matriz pode ser deduzida observando-se que as coordenadas do ponto P do vetor posição $\vec{v}$ podem ser expressas como

$$x = r\cos\omega, \quad y = r\sin\omega,$$

onde r é o tamanho do vetor $\vec{v}$ e ω é o ângulo que $\vec{v}$ faz com o eixo-x positivo

- Assim, as coordenadas do ponto resultante da rotação são

$$x' = r\cos(\omega + \theta), \quad y' = r\sin(\omega + \theta)$$

Rotações

- Utilizando as fórmulas para soma de ângulos do seno e do cosseno obtemos
$$x' = r\cos\omega\cos\theta - r\sin\omega\sin\theta$$
e
$$y' = r\sin\omega\cos\theta + r\cos\omega\sin\theta,$$
o que corresponde ao resultado do produto matricial já citado

```cpp
// Definição da estrutura Point

template<typename T>
Point<T> rotate(const Point<T>& P, T angle)
{
    auto x = cos(angle) * P.x - sin(angle) * P.y;
    auto y = sin(angle) * P.x + cos(angle) * P.y;

    return Point<T> { x, y };
}
```

Rotação em torno de um ponto arbitrário

- Caso se deseje rotacionar o ponto P em torno de outro ponto C que não seja a origem (mais precisamente, outro eixo paralelo ao eixo-z que passe pelo ponto dado), basta seguir os três passos abaixo:
  1. transladar o ponto com deslocamentos iguais aos opostos das coordenadas de C, obtendo-se o ponto P'
  2. rotacionar o ponto transladado P'
  3. transladar P', com deslocamentos iguais às coordenadas de C
- A translação inicial muda o sistema de coordenadas do problema, o levando a um novo sistema onde C é a origem
- Assim, pode-se utilizar a rotina de rotação em torno da origem e, ao final do processo, retornar ao sistema original, aplicando a translação inversa
- Importante notar que as três operações devem ser realizadas exatamente na ordem descrita

Implementação da rotação em torno de um ponto arbitrário

```
// Definição da classe Point e das funções translate() e rotate()

template<typename T>
Point<T> rotate(const Point<T>& P, T angle, const Point<T>& C)
{
    auto Q = translate(P, -C.x, -C.y);
    Q = rotate(Q, angle);
    Q = translate(Q, C.x, C.y);

    return Q;
}
```

Rotações tridimensionais

- A mesma ideia da rotação pode ser aplicada em pontos tridimensionais
- As matrizes R_x, R_y e R_z abaixo rotacionam o ponto tridimensional $P = (x_p, y_p, z_p)$ em θ graus no sentido anti-horário

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \quad R_y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Observações sobre translações e rotações

- Dado um conjunto de pontos C, se aplicadas a todos pontos $P \in C$, as operações de translação e rotação não alteram as distâncias entre os pares de pontos
- Desta forma, se uma figura é descrita por um conjunto de pontos, todas as suas características que são baseadas em distâncias (ângulos internos, perímetro, área, volume, etc) são invariantes a estas duas transformações
- Este importante fato pode ser utilizado para simplificar problemas, como exemplificado no caso da rotação em torno de um ponto arbitrário

Escala

- Outra transformação possível em um ponto (ou vetor posição) é a escala
- A escala consiste na multiplicação das coordenadas de um ponto por um escalar
- Se o mesmo escalar é utilizado em todos os produtos a escala é dita uniforme
- Ao contrário das transformações anteriores, a escala não preserva as distâncias

```
// Definição da classe Point

template<typename T>
Point<T> scale(T sx, T sy)
{
    return Point<T> {sx * P.x, sy * P.y};
}
```

Normalização de vetores

- Uma aplicação comum da escala é a normalização de vetor
- Um vetor é dito unitário se o seu comprimento é igual a 1
- Dado um vetor $\vec{v}$ qualquer, é possível determinar um vetor unitário $\vec{u}$, na mesma direção e sentido de $\vec{v}$, dividindo-se as coordenadas de $\vec{v}$ pelo tamanho $|\vec{v}|$ de $\vec{v}$
- Observe que a escala com constantes positivas preserva a direção e o sentido do vetor

```
// Definição da classe Vector

template<typename T>
Vector<T> normalize(const Vector<T>& v)
{
    auto len = v.length();
    return Vector<T>(v.x / len, v.y / len);
}
```

Produto interno e produto vetorial

Produto interno

- O produto interno (ou escalar) entre dois vetores $\vec{u}$ e $\vec{v}$ é dado pela soma dos produtos das coordenadas correspondentes dos dois vetores
- O resultado deste produto não é um vetor, e sim um escalar
- É possível mostrar que este produto coincide com o produto do tamanho dos dois vetores e do cosseno do ângulo entre estes vetores, conforme mostra a expressão abaixo:

$$< \vec{u}, \vec{v} >= \vec{u} \cdot \vec{v} = u_x v_x + u_y v_y = |\vec{u}||\vec{v}| \cos \theta$$

- A relação acima permite computar o ângulo entre os vetores $\vec{u}$ e $\vec{v}$
- O sinal do produto interno $d = \vec{u} \cdot \vec{v}$ pode ser utilizado para interpretar a natureza do ângulo entre os dois vetores:
    1. se $d = 0$, os vetores são ortogonais (formam um ângulo de 90°)
    2. se $d < 0$, os vetores foram um ângulo agudo (menor que 90°)
    3. se $d > 0$, os vetores formam um ângulo obtuso (maior que 90°)

Implementação do produto interno e do ângulo entre vetores em C++

```cpp
// Definição da classe Vector

template<typename T>
T dot_product(const Vector<T>& u, const Vector<T>& v)
{
    return u.x * v.x + u.y * v.y;
}

// O retorno está no intervalo [0, pi]
template<typename T>
double angle(const Vector& u, const Vector& v)
{
    auto lu = u.length();
    auto lv = v.length();
    auto prod = dot_product(u, v);

    return acos(prod/(lu * lv));
}
```

Produto vetorial

- O produto vetorial entre dois vetores $\vec{u}$ e $\vec{v}$ é $\vec{w} = \vec{u} \times \vec{v}$ cujas coordenadas são obtidas através do determinante

$$u \times v = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix},$$

  onde $\vec{i}, \vec{j}, \vec{k}$ são vetores unitários com mesma direção e sentido que os eixos x, y, z, respectivamente
- Sendo definido por um determinante, o produto vetorial não é comutativo: a troca da ordem dos vetores altera o sentido do resultado
- Para computar o produto vetorial entre vetores bidimensionais, basta fazer a coordenada z de ambos vetores iguais a zero
- O vetor resultante é perpendicular tanto a $\vec{u}$ quanto a $\vec{v}$

Produto vetorial

- A magnitude deste vetor é igual ao produto dos comprimentos dos vetores $\vec{u}$ e $\vec{v}$ pelo seno do ângulo θ formado por estes dois vetores, isto é,

$$|\vec{u} \times \vec{v}| = |\vec{u}||\vec{v}| \sin \theta$$

- Este valor coincide com a área do paralelogramo formado por $\vec{u}$ e $\vec{v}$
- Se os vetores tiverem mesma direção, o produto vetorial terá comprimento zero (como não definirão um plano, não há um vetor normal)
- Vetores normais podem ser utilizados para definir a orientação de uma figura tridimensional (o lado interno e externo da figura)

Exemplo de implementação de produto vetorial em C++

```cpp
// Definição da classe Vector

template<typename T>
Vector<T> cross_product(const Vector<T>& u, const Vector<T>& v)
{
    auto x = u.y*v.z - v.y*u.z;
    auto y = u.z*v.x - u.x*v.z;
    auto z = u.x*v.y - u.y*v.x;

    return Vector<T>(x, y, z);
}
```

Referências

1. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
2. **LAAKSONEN**, Antti. *Competitive Programmer's Handbook*, 2018.
3. **De BERG**, Mark; **CHEONG**, Otfried. *Computational Geometry: Algorithms and Applications*, 2008.
4. David E. Joyce. *Euclid's Elements*. Acesso em 15/02/2019[1]
5. Wikipédia. *Geometria Euclidiana*. Acesso em 15/02/2019[2].

[1] https://mathcs.clarku.edu/ djoyce/elements/bookI/defI2.html
[2] https://pt.wikipedia.org/wiki/Geometria_euclidiana