Geometria Computacional

Círculos: Algoritmos

Prof. Edson Alves

2018

Faculdade UnB Gama

Sumário

Relação entre pontos e círculos

Relação de pertinência de um ponto P

- Dado um ponto P e um círculo de centro C e raio r, uma (e apenas uma) das três afirmações abaixo será verdadeira:
    1. P está dentro do círculo
    2. P está sobre o círculo
    3. P está fora do círculo
- Para determinar qual é a relação válida, basta computar a distância entre o ponto P e o centro C do círculo
- Caso esta distância seja menor, igual ou maior que r, P estará dentro, sobre e fora do círculo, respectivamente
- O conjunto de pontos que estão dentro do círculo é denominado disco de raio r e centro C

Implementação da posição do ponto em um círculo em C++

```cpp
// Definição da classe Point e da função equals()

template<typename T>
struct Circle {
    Point<T> C;
    T r;

    enum { IN, ON, OUT } PointPosition;

    PointPosition position(const Point& P) const
    {
        auto d = dist(P, C);

        return equals(d, r) ? ON : (d < r ? IN : OUT);
    }
};
```

Construção de círculos a partir de dois pontos

- É possível identificar o(s) círculo(s) que interceptam um conjunto de N pontos dados
- No caso $N = 1$, existem infinitos círculos (com infinitos raios possíveis) que passam por um dado ponto P
- O caso $N = 2$ se torna mais interessante se o raio r for pré-determinado
- Dados dois pontos P e Q e o um raio r, os cenários possíveis são:
  1. $P = Q$: esta situação é idêntica ao caso $N = 1$
  2. $\mathrm{dist}(P, Q) = 2r$: se a distância entre os dois pontos dados é igual ao diâmetro do círculo, existe um único círculo de raio r que passa por P e Q, cujo centro será o ponto médio do segmento PQ
  3. $\mathrm{dist}(P, Q) > 2r$: neste caso, nenhum círculo de r pode passar por ambos pontos simultaneamente
  4. $\mathrm{dist}(P, Q) < 2r$: neste caso, exatamente dois círculos passam por P e Q com raio r

Implementação da identificação de um círculo a partir de dois pontos e o raio

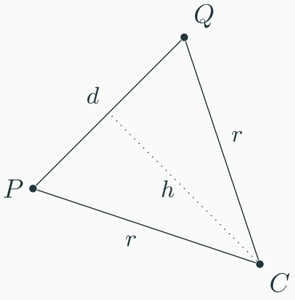
```cpp
// O código abaixo, adaptado do livro Competitive Programming 3.
// A função retorna um dos círculos possíveis: o outro pode ser
// encontrado invertendo os parâmetros P e Q na chamada da função

#include <optional>

// Definição da class Point

template<typename T>
struct Circle {
    // Membros e construtores

    static std::optional<Circle>
    from_2_points_and_r(const Point<T>& P, const Point<T>& Q, T r)
    {
        double d2 = (P.x - Q.x) * (P.x - Q.x) + (P.y - Q.y) * (P.y - Q.y);
        double det = r * r / d2 - 0.25;

        if (det < 0.0)
            return { };
```

Implementação da identificação de um círculo a partir de dois pontos e o raio

```

        double h = sqrt(det);

        auto x = (P.x + Q.x) * 0.5 + (P.y - Q.y) * h;
        auto y = (P.y + Q.y) * 0.5 + (Q.x - P.x) * h;

        return Circle { Point(x, y), r };
    }
}
```

Demonstração do algoritmo

- A implementação do algoritmo anterior, embora simples, é baseada em fatos não-triviais
- Sejam P e Q dois pontos distintos tais que $\mathrm{dist}(P, Q) < 2r$, onde r é o raio dado
- Neste cenário, o centro C do círculo não pertence ao segmento PQ
- Assim, é formado um triângulo PCQ



Demonstração do algoritmo

- A Lei dos Cossenos diz que, num triângulo de lados a, b, c cujo ângulo oposto a a é α, vale a igualdade

$$a^2 = b^2 + c^2 - 2ab\cos\alpha$$

- Aplicando a Lei dos Cossenos ao triângulo PCQ, e considerando θ o ângulo oposto ao lado d, têm-se que

$$d^2 = r^2 + r^2 - 2r^2\cos\theta = 2r^2(1 - \cos\theta)$$

- Como $|\cos\theta| \leq 1$, vale a desigualdade

$$d^2 \leq 4r^2,$$

  a qual pode ser reescrita como

$$\frac{r^2}{d^2} \geq \frac{1}{4}$$

Demonstração do algoritmo

- Defina o discriminante

$$\Delta = \frac{r^2}{d^2} - \frac{1}{4}$$

- Para que exista um círculo que passe por P e Q é preciso que $\Delta \geq 0$
- Como PCQ é um triângulo isóceles, sua altura (cuma medida é h) coincide com a mediana
- Seja M o ponto médio de PQ. Aplicando o Teorema de Pitágoras ao triângulo PMC obtêm-se

$$r^2 = h^2 + \left(\frac{d}{2}\right)^2,$$

isto é,

$$h = \sqrt{r^2 - \frac{d^2}{4}} = d\sqrt{\frac{r^2}{d^2} - \frac{1}{4}} = d\sqrt{\Delta}$$

Demonstração do algoritmo

- Sejam $\vec{P}$ e $\vec{Q}$ os vetores-posição dos pontos P e Q. O vetor unitário $\vec{u}$ que parte de P em direção a Q é dado por

$$\vec{u} = \frac{\vec{Q} - \vec{P}}{\|\vec{Q} - \vec{P}\|} = \left(\frac{x_Q - x_P}{d}, \frac{y_Q - y_P}{d} \right)$$

- O vetor unitário $\vec{n}$, normal a $\vec{u}$, é dado por

$$\vec{n} = \left(\frac{y_P - y_Q}{d}, \frac{x_Q - x_P}{d} \right)$$

- Assim, o vetor posição do centro C do círculo pode ser encontrado pela soma vetorial

$$\vec{C} = \vec{P} + \frac{d}{2}\vec{u} + h\vec{n}$$

- As duas soluções possíveis diferem pelo sentido do vetor $\vec{n}$

Demonstração do algoritmo

- Portanto,

$$
\begin{aligned}
\vec{C} &= \vec{P} + \frac{d}{2}\vec{u} + h\vec{n} \\
&= (x_P, y_P) + \frac{d}{2}\left(\frac{x_Q - x_P}{d}, \frac{y_Q - y_P}{d}\right) + h\left(\frac{y_P - y_Q}{d}, \frac{x_Q - x_P}{d}\right) \\
&= (x_P, y_P) + \left(\frac{x_Q - x_P}{2}, \frac{y_Q - y_P}{2}\right) + d\sqrt{\Delta}\left(\frac{y_P - y_Q}{d}, \frac{x_Q - x_P}{d}\right) \\
&= (x_P, y_P) + \left(\frac{x_Q - x_P}{2}, \frac{y_Q - y_P}{2}\right) + \sqrt{\Delta}\,(y_P - y_Q, x_Q - x_P) \\
&= \left(\frac{x_P + x_Q}{2}, \frac{y_P + y_Q}{2}\right) + \sqrt{\Delta}\,(y_P - y_Q, x_Q - x_P)
\end{aligned}
$$

Construção de círculos a partir de três pontos

- Para o caso $N = 3$ há uma interessante relação: se os pontos P, Q, R não são colineares, a equação do círculo que passa por estes três pontos pode ser expressa pelo determinante

$$D = \begin{vmatrix} x^2 + y^2 & x & y & 1 \\ P_x^2 + P_y^2 & P_x & P_y & 1 \\ Q_x^2 + Q_y^2 & Q_x & Q_y & 1 \\ R_x^2 + R_y^2 & R_x & R_y & 1 \end{vmatrix}$$

- Este determinante também pode ser utilizado para determinar se 4 pontos são cocirculares, substituindo as coordenadas do quarto ponto nas variáveis da primeira linha
- Contudo, a implementação desta determinante não é trivial, uma vez que é preciso recorrer a cofatores, e o resultado final não fica na forma canônica, de onde são extraídas as informações sobre o raio e o centro

Construção de círculos a partir de três pontos

- Uma outra abordagem é observar que a distância entre os três pontos e o centro do círculo são iguais, isto é,

$$\mathrm{dist}(P, C) = \mathrm{dist}(Q, C) \ \text{ e } \ \mathrm{dist}(P, C) = \mathrm{dist}(R, C)$$

- A expansão dos quadrados das duas igualdades acima leva a um sistema linear em relação as coordenadas do centro, pois

$$\begin{cases} (x - x_P)^2 + (y - y_P)^2 = (x - x_Q)^2 + (y - y_Q)^2 \\ (x - x_P)^2 + (y - y_P)^2 = (x - x_R)^2 + (y - y_R)^2 \end{cases}$$

corresponde a

$$\begin{cases} (2x_Q - 2x_P)x + (2y_Q - 2x_Q)y = (x_Q^2 + y_Q^2) - (x_P^2 + y_P^2) \\ (2x_R - 2x_P)x + (2y_R - 2x_R)y = (x_R^2 + y_R^2) - (x_P^2 + y_P^2) \end{cases}$$

- Determinado o centro, o raio será a distância entre qualquer um dos pontos e o centro

Construção de um círculo a partir de 3 pontos

```
#include <optional>

// Definição da class Point e das funções equals() e distance()

template<typename T>
struct Circle {
    // Membros e construtores

    static std::optional<Circle>
    from_3_points(const Point<T>& P, const Point<T>& Q, const Point<T>& R)
    {
        auto a = 2*(Q.x - P.x);
        auto b = 2*(Q.y - P.y);
        auto c = 2*(R.x - P.x);
        auto d = 2*(R.y - P.y);

        auto det = a*d - b*c;

        // Pontos colineares
        if (equals(det, 0))
            return { };
```

Construção de um círculo a partir de 3 pontos

```cpp

        auto k1 = (Q.x*Q.x + Q.y*Q.y) - (P.x*P.x + P.y*P.y);
        auto k2 = (R.x*R.x + R.y*R.y) - (P.x*P.x + P.y*P.y);

        // Solução do sistema por Regra de Cramer
        auto cx = (k1*d - k2*b)/det;
        auto cy = (a*k2 - c*k1)/det;

        Point<T> C { cx, cy };
        auto r = distance(P, C);

        return Circle<T>(C, r);
    }
};
```

Relação entre círculo e reta

Interseção entre círculo e reta

- Uma reta que passa pelos pontos P_1 e P_2 pode ser representada, de forma paramétrica, pela expressão vetorial $\vec{P} = \vec{P_1} + t(\vec{P_2} - \vec{P_1})$, onde t é uma variável real
- Assim, a coordenada x de P é dada por $x = x_1 + t(x_2 - x_1)$
- De forma semelhante, a coordenada y de P é dada por $y = y_1 + t(y_2 - y_1)$
- Se estas coordenadas forem levadas para a equação do círculo de centro C e raio r (isto é, $(x - x_C)^2 + (y - y_C)^2 = r^2$), o resultado é o polinômio

$$at^2 + bt + c = 0,$$

onde

$$\begin{aligned}
a &= (x_2 - x_1)^2 + (y_2 - y_1)^2 \\
b &= 2\left[(x_2 - x_1)(x_1 - C_x) + (y_2 - y_1)(y_1 - C_y)\right] \\
c &= C_x^2 + C_y^2 + x_1^2 + y_1^2 - 2(C_x x_1 + C_y y_1)
\end{aligned}$$

Interseção entre círculo e reta

- O discriminante $\Delta = b^2 - 4ac$ desta equação caracteriza as possíveis interseções
- se $\Delta < 0$, não há interseção entre o círculo e a reta
- se $\Delta = 0$, há um único ponto de interseção (a reta é tangente ao círculo)
- se $\Delta > 0$, há dois pontos distintos de interseção
- As coordenadas dos pontos de interseção podem ser obtidas substuíndos os zeros do polinômio nas equações paramêtricas de x e y

Implementação da interseção entre círculo e reta

```cpp
// Definição das classes Point e Circle e da função equals()

// Interseção entre o círculo c e a reta que passa por P e Q
template<typename T> std::vector<Point<T>>
intersection(const Circle<T>& c, const Point<T>& P, const Point<T>& Q)
{
    auto a = pow(Q.x - P.x, 2.0) + pow(Q.y - P.y, 2.0);
    auto b = 2*((Q.x - P.x) * (P.x - c.C.x) + (Q.y - P.y) * (P.y - c.C.y));
    auto d = pow(c.C.x, 2.0) + pow(c.C.y, 2.0) + pow(P.x, 2.0)
        + pow(P.y, 2.0) + 2*(c.C.x * P.x + c.C.y * P.y);
    auto D = b * b - 4 * a * d;

    if (D < 0)
        return { };
    else if (equals(D, 0))
    {
        auto u = -b/(2*a);
        auto x = P.x + u*(Q.x - P.x);
        auto y = P.y + u*(Q.y - P.y);
        return { Point { x, y } };
    }
```

Implementação da interseção entre círculo e reta

```cpp

    auto u = (-b + sqrt(D))/(2*a);

    auto x = P.x + u*(Q.x - P.x);
    auto y = P.y + u*(Q.y - P.y);

    auto P1 = Point { x, y };

    u = (-b - sqrt(D))/(2*a);

    x = P.x + u*(Q.x - P.x);
    y = P.y + u*(Q.y - P.y);

    auto P2 = Point { x, y };

    return { P1, P2 };
}
```

Relação entre dois círculos

Interseção entre dois círculos

- Dados dois círculos com centros C_1, C_2 e raios r_1, r_2, existem cinco cenários possíveis para suas interseções
- Seja $d = \text{dist}(C_1, C_2)$. Então:
  1. se $d > r_1 + r_2$, então os círculos não se interceptam
  2. se $d < |r_1 - r_2|$, então também não há interseção, pois um dos círculos (o de menor raio) está contido no outro (o de maior raio)
  3. se $d = 0$ e $r1 = r2$, então os círculos são idênticos: há infinitos pontos de interseção
  4. se $d = r1 + r2$, os círculos se interceptam em um único ponto
  5. nos demais casos, há dois pontos na interseção entre os círculos

Par de círculos com dois pontos de interseção

- Seja $C_1 = (x_1, y_1)$ e $C_2 = (x_2, y_2)$
- as coordenadas dos pontos de interseção P_1 e P_2 são dadas por

$$
\begin{aligned}
a &= \frac{r_1^2 - r_2^2 + d^2}{2d} \\
h &= \sqrt{r_1^2 - a^2} \\
P &= C_1 + \frac{a}{d}(C_2 - C_1) \\
P_1 &= \left(P_x + \frac{h}{d}(y_2 - y_1), P_y - \frac{h}{d}(x_2 - x_1)\right) \\
P_2 &= \left(P_x - \frac{h}{d}(y_2 - y_1), P_y + \frac{h}{d}(x_2 - x_1)\right)
\end{aligned}
$$

- A justificativa deste resultado segue de desenvolvimento semelhante ao da identificação de um círculo a partir de dois pontos e um raio

Implementação da interseção entre dois círculos

```cpp
#include <variant>
#include <vector>

const int oo { 2000000000 };

// Definição das classes Point e Circle, e das função equals()
// e distance()

template<typename T> std::variant<int, std::vector<Point<T>>>
intersection(const Circle<T>& c1, const Circle<T>& c2)
{
    double d = distance(c1.C, c2.C);

    if (d > c1.r + c2.r or d < fabs(c1.r - c2.r))
        return 0;

    if (equals(d, 0.0) and equals(c1.r, c2.r))
        return oo;

    auto a = (c1.r * c1.r - c2.r * c2.r + d * d)/(2 * d);
    auto h = sqrt(c1.r * c1.r - a * a);
```

Implementação da interseção entre dois círculos

```

    auto x = c1.C.x + (a/d)*(c2.C.x - c1.C.x);
    auto y = c1.C.y + (a/d)*(c2.C.y - c1.C.y);

    auto P = Point<T> { x, y };

    x = P.x + (h/d)*(c2.C.y - c1.C.y);
    y = P.y - (h/d)*(c2.C.x - c1.C.x);

    auto P1 = Point<T> { x, y };

    x = P.x - (h/d)*(c2.C.y - c1.C.y);
    y = P.y + (h/d)*(c2.C.x - c1.C.x);

    auto P2 = Point<T> { x, y };

    return std::vector<Point<T>> { P1, P2 };
}
```

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. QC.EDU.HK. *Equation of circle passing through 3 given points*. Acesso em 18/08/2016.[1]
5. Stack Exchange. *Mathematics: Get the equation of a circle when given 3 points*. Acesso em 18/08/2016.[2]

[1] http://paulbourke.net/geometry/circlesphere/
[2] https://math.stackexchange.com/questions/213658/get-the-equation-of-a-circle-when-given-3-points