ACM/ICPC: Competitive Programming Notebook

Lucas Mattioli, Marlon Mendes, Simião Carvalho

Contents

Points

Comparing floating point values

Returns true if double values a and b are equal

```
const double EPS { 1e-9 };
bool equals(double a, double b)
{
    return fabs(a - b) < EPS;
}
```

Listing 1: equals

Lines

General equation of a line

Non-normalized form: ax + by + c = 0

```
class Line {
public:
    double a;
    double b;
    double c;

    Line(double av, double bv, double cv) : a(av), b(bv), c(cv) {}

    Line(const Point& p, const Point& q)
    {
        a = p.y - q.y;
        b = q.x - p.x;
        c = p.x * q.y - p.y * q.x;
    }
};
```

Listing 2: General equation of a line

General equation of a line normalized

```
class Line {
public:
    double a;
    double b;
    double c;

    Line(double av, double bv, double cv) : a(av), b(bv), c(cv) {}

    Line(const Point& p, const Point& q)
    {
        a = p.y - q.y;
        b = q.x - p.x;
        c = p.x * q.y - p.y * q.x;

        auto k = a ? a : b;

        a /= k;
        b /= k;
```

```
        c /= k;
    }
};
```

Listing 3: General equation of a line

Point on a line

Is the given point located on the given Line?

```
template<typename T>
struct Line {
    bool contains(const Point<T>& P) const
    {
        return equals(a*P.x + b*P.y + c, 0);
    }
};
```

Listing 4: Point on line

Vectors

Angle between vector and X-axis

Returns an angle in radians in the interval $[-\pi, +\pi]$. A positive angle means in the COUNTER-clockwise direction. A negative angle is measured in the clockwise direction. Note that the atan2 swaped the parameters.

```
inline double angle(double x, double y) {
    return atan2(y, x);
}
```

Listing 5: angle between X-axis and vectorx, y

Translation

```
Point translate(const Point& P, double dx, double dy)
{
    return Point { P.x + dx, P.y + dy };
}
```

Listing 6: Translate point

Rotation around origin

```
Point rotate(const Point& P, double angle)
{
    auto x = cos(angle) * P.x - sin(angle) * P.y;
    auto y = sin(angle) * P.x + cos(angle) * P.y;

    return Point { x, y };
}
```

Rotation around another point

```cpp
Point rotate(const Point& P, double angle, const Point& C)
{
    auto Q = translate(P, -C.x, -C.y);
    Q = rotate(Q, angle);
    Q = translate(Q, C.x, C.y);

    return Q;
}
```

Rotation around origin 3D

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

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

Scale

```cpp
Point scale(double sx, double sy)
{
    return Point(sx * P.x, sy * P.y);
}
```

Listing 7: Scale vector by a factor of sx and sy

Normalization

```cpp
Vector normalize(const Vector& v)
{
    auto len = v.length();
    auto u = Vector(v.x / len, v.y / len);

    return u;
}
```

Listing 8: Returns a unit vector with the same direction as the given vector

Dot product

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

```
double dot_product(const Vector& u, const Vector& v)
{
    return u.x * v.x + u.y * v.y;
}
```

Angle between vectors

```
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));
}
```

Cross product

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

- $|\vec{u} \times \vec{v}| = |\vec{u}||\vec{v}| \sin \theta$
- where $\vec{i}, \vec{j}, \vec{k}$ are unity vectors on the same direction and orientation as x, y, z, respectively
- the result vector $\vec{w}$ is orthogonal to both $\vec{u}$ and $\vec{v}$
- it is the area of the parallelogram formed by $\vec{u}$ and $\vec{v}$

```
Vector cross_product(const Vector& u, const Vector& 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(x, y, z);
}
```