



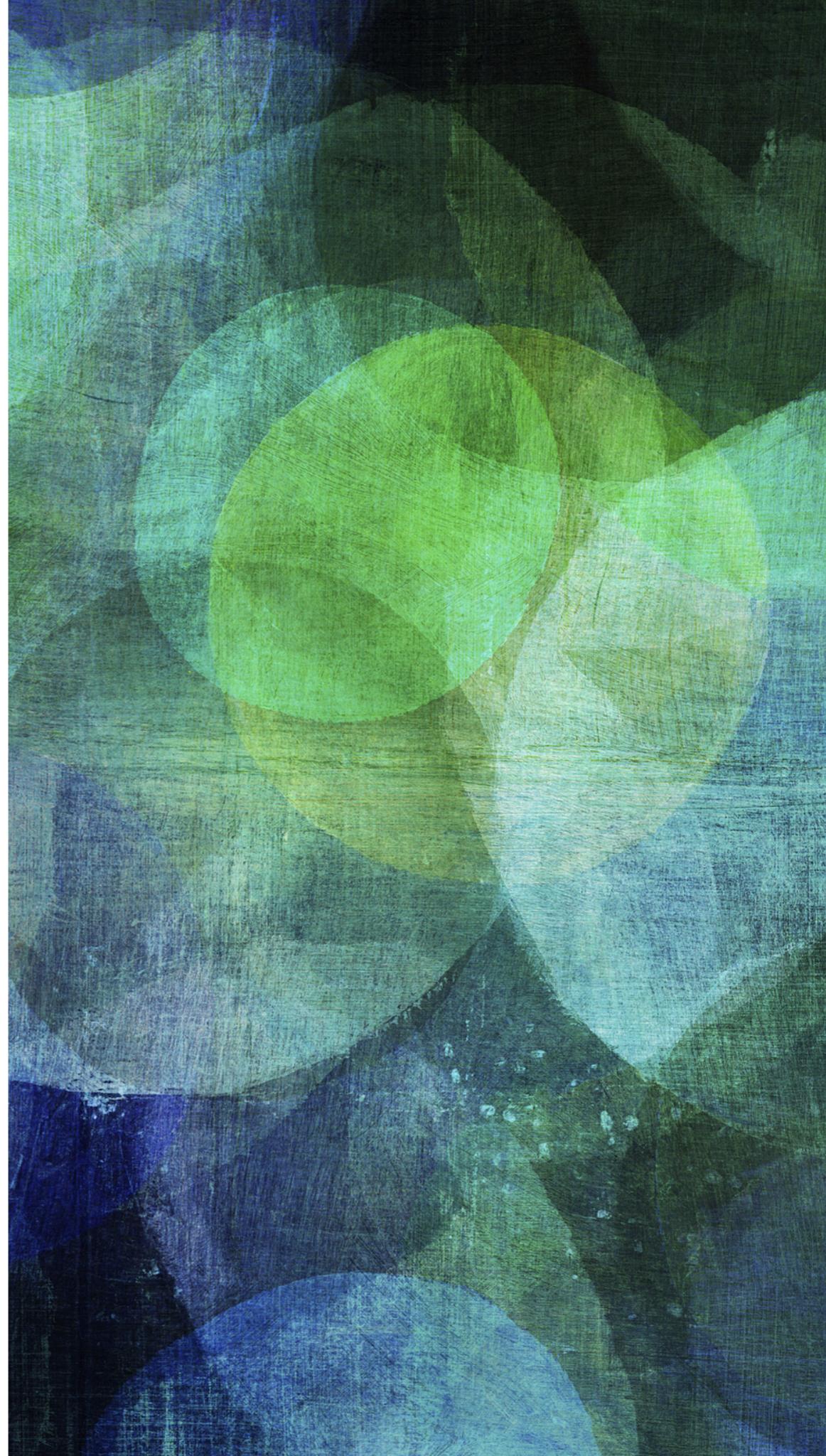
省选入门-计算几何

说明

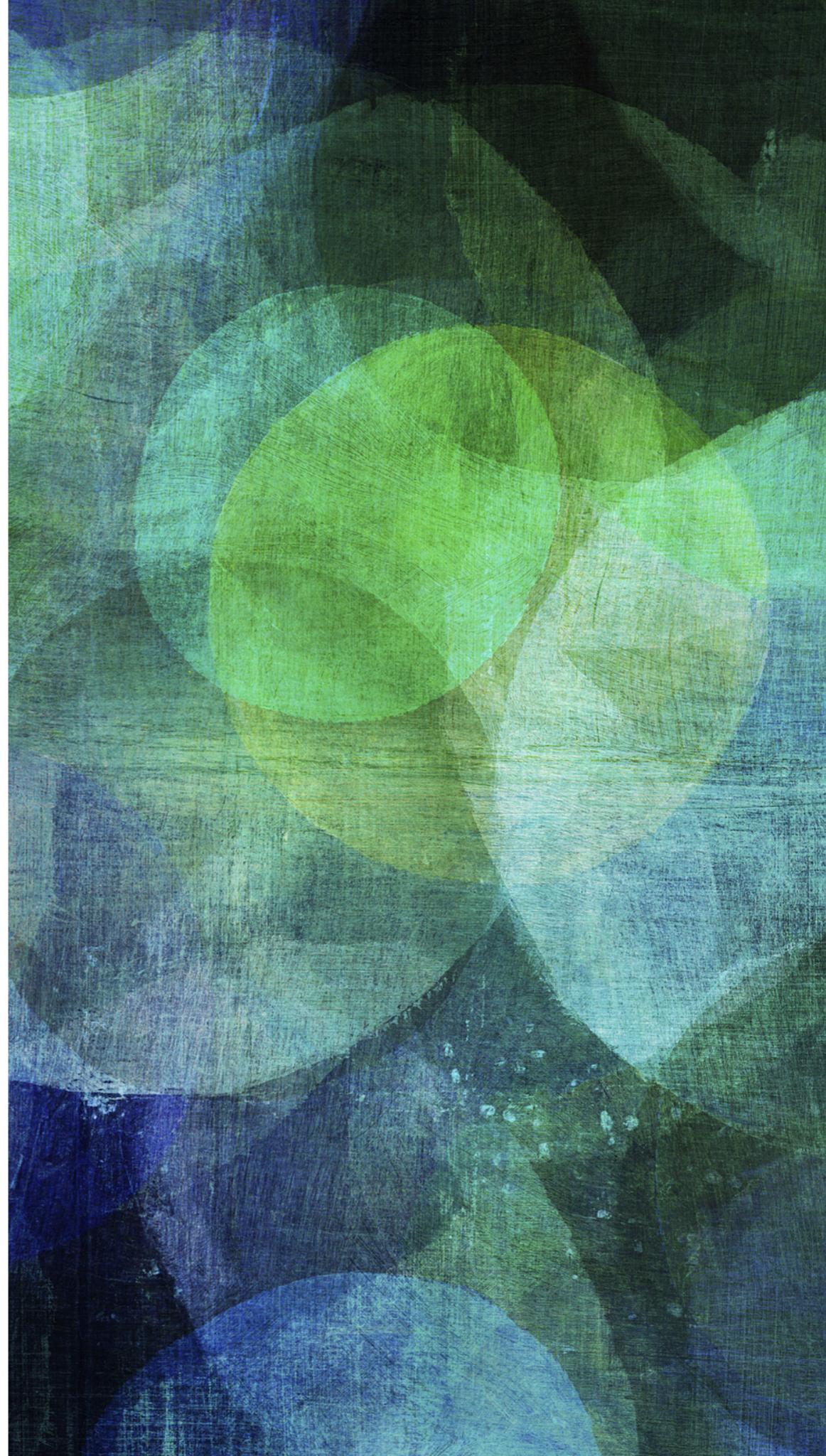
- 此课件重点讨论的是二维几何
- 主要内容有：
- 1: 基础知识：向量，点积，叉积，直线（线段）相交
- 2: 凸包与半平面交
- 3: 其他：三角形，多边形，坐标旋转

```
struct Point;
typedef Point Vector;
struct Point
{
    double x, y;
    void in() {
        scanf("%lf%lf", &x, &y);
    }
    void print() {
        printf("%.2lf %.2lf\n", x, y);
    }
    Point(double x = 0, double y = 0) : x(x), y(y) {}
    inline Vector rotate(double ang) {
        return Vector(x * cos(ang) - y * sin(ang), x * sin(ang) + y * cos(ang));
    }
    inline double dot(const Vector &a) {
        return x * a.x + y * a.y;
    }
    inline bool operator == (const Point &a) const {
        return sgn(x - a.x) == 0 && sgn(y - a.y) == 0;
    }
    inline bool operator < (const Point &a) const {
        return sgn(x - a.x) < 0 || sgn(x - a.x) == 0 && sgn(y - a.y) < 0;
    }
    inline Vector operator + (const Vector &a) const {
        return Vector(x + a.x, y + a.y);
    }
    inline Vector operator - (const Vector &a) const {
        return Vector(x - a.x, y - a.y);
    }
    inline double operator * (const Vector &a) const {
        return x * a.y - y * a.x;
    }
    inline Vector operator * (double t) const {
        return Vector(x * t, y * t);
    }
    inline Vector operator / (double t) {
        return Vector(x / t, y / t);
    }
    inline double vlen() {
        return sqrt(x * x + y * y);
    }
    inline Vector norm() {
        return Point(-y, x);
    }
};
```

基础知识



向量

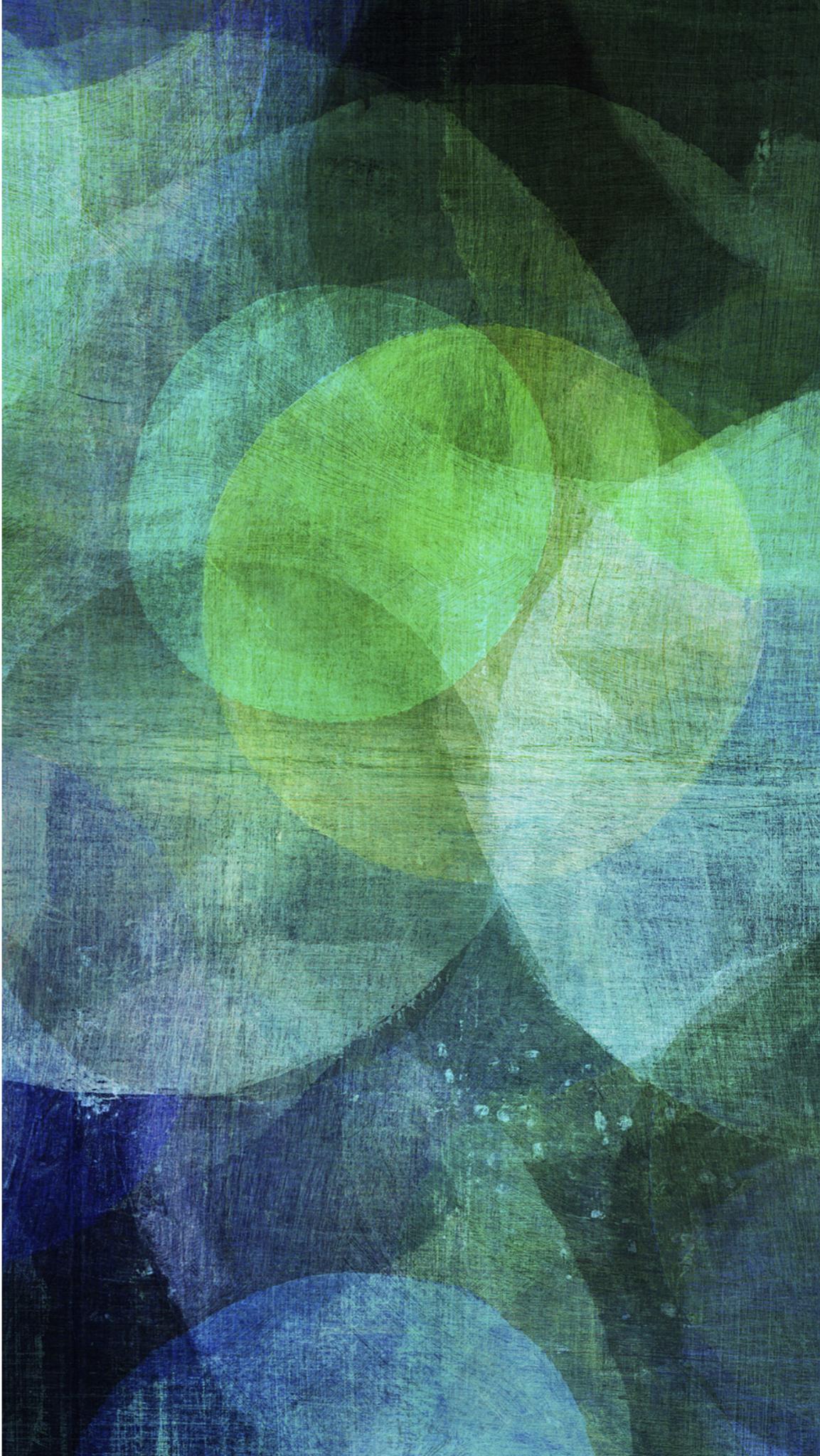


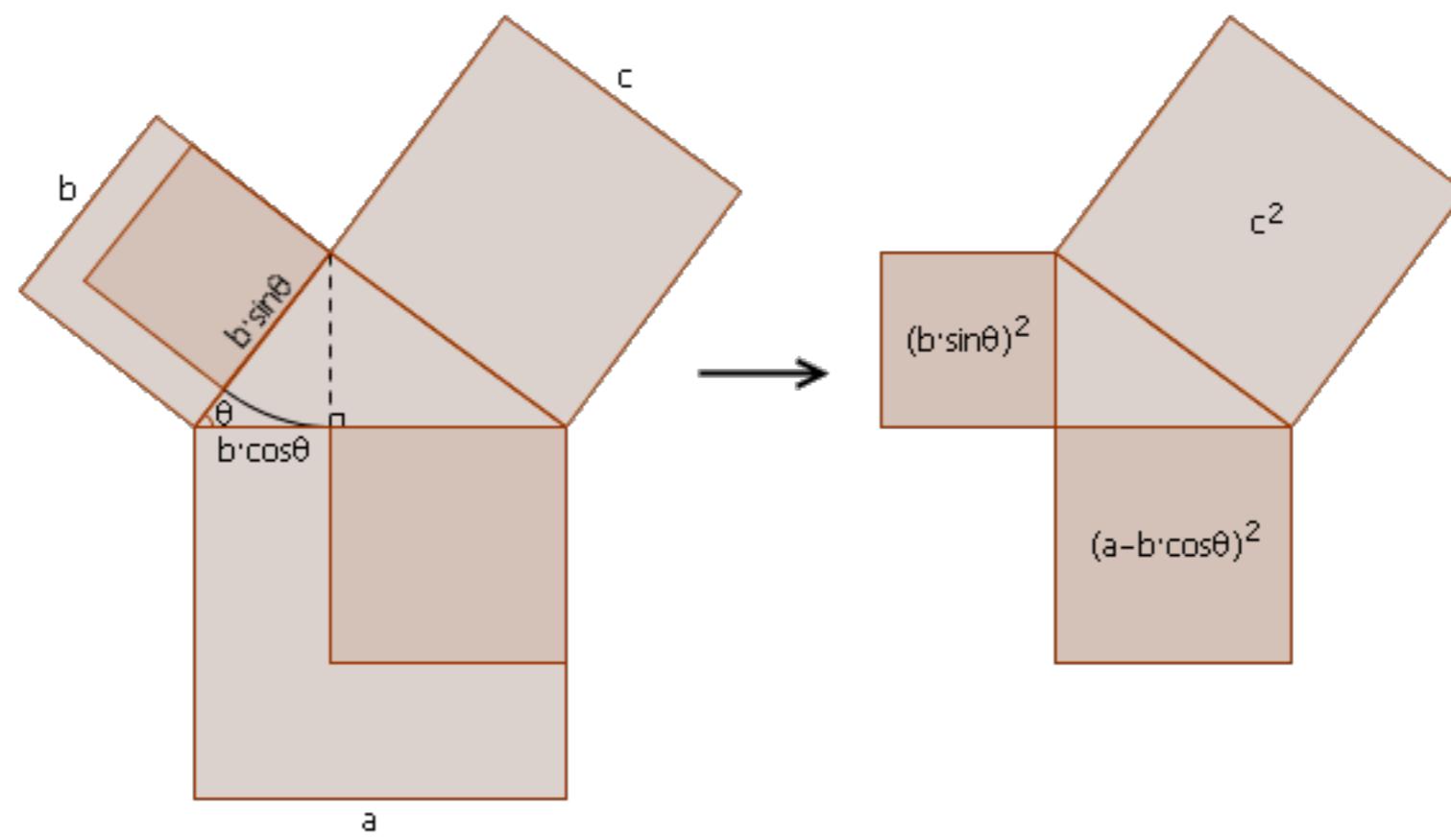
向量

- 点A(x₁,y₁)指向B(x₂,y₂)的向量为(x₂-x₁,y₂-y₁)
- 一个向量可以表示两个点之间的方向，以及长度

点积

内积





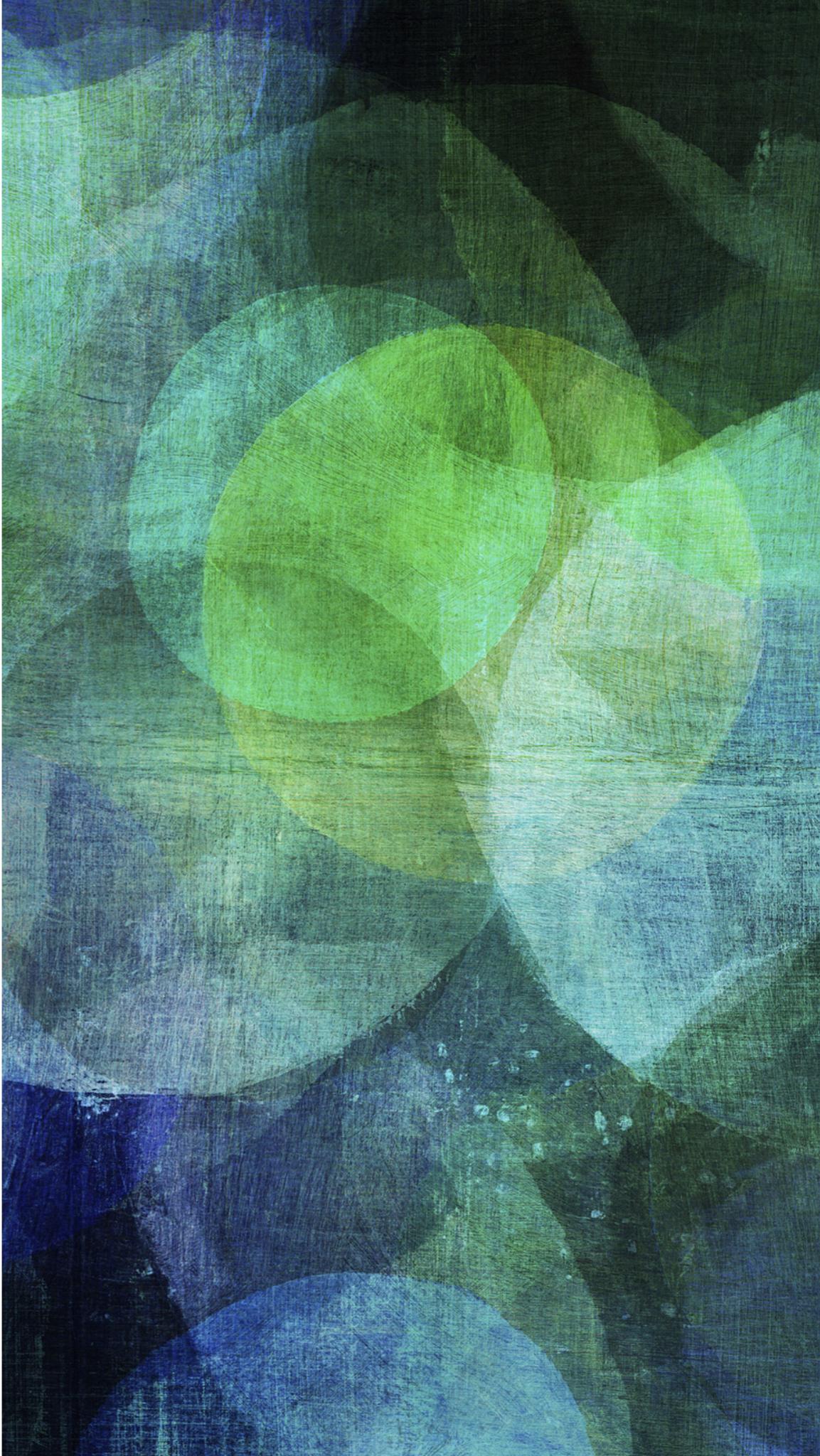
$$\begin{aligned}c^2 &= (b \cdot \sin\theta)^2 + (a - b \cdot \cos\theta)^2 \\&= a^2 + b^2 - 2ab \cdot \cos\theta\end{aligned}$$

点积

- 两个向量做点积会返回一个值
- 两个向量A(x1,y1) , B(x2, y2)做点积的结果为 $x1 * x2 + y1 * y2$
- 两个向量A B做点积等价于
- $x1 * x2 + y1 * y2 = |A| |B| * \cos(\text{夹角})$
- 这个等式可以利用余弦定理推导
- 所以点积也用来求两个向量的夹角
- 还可以用来求一个向量在另一个向量上的投影

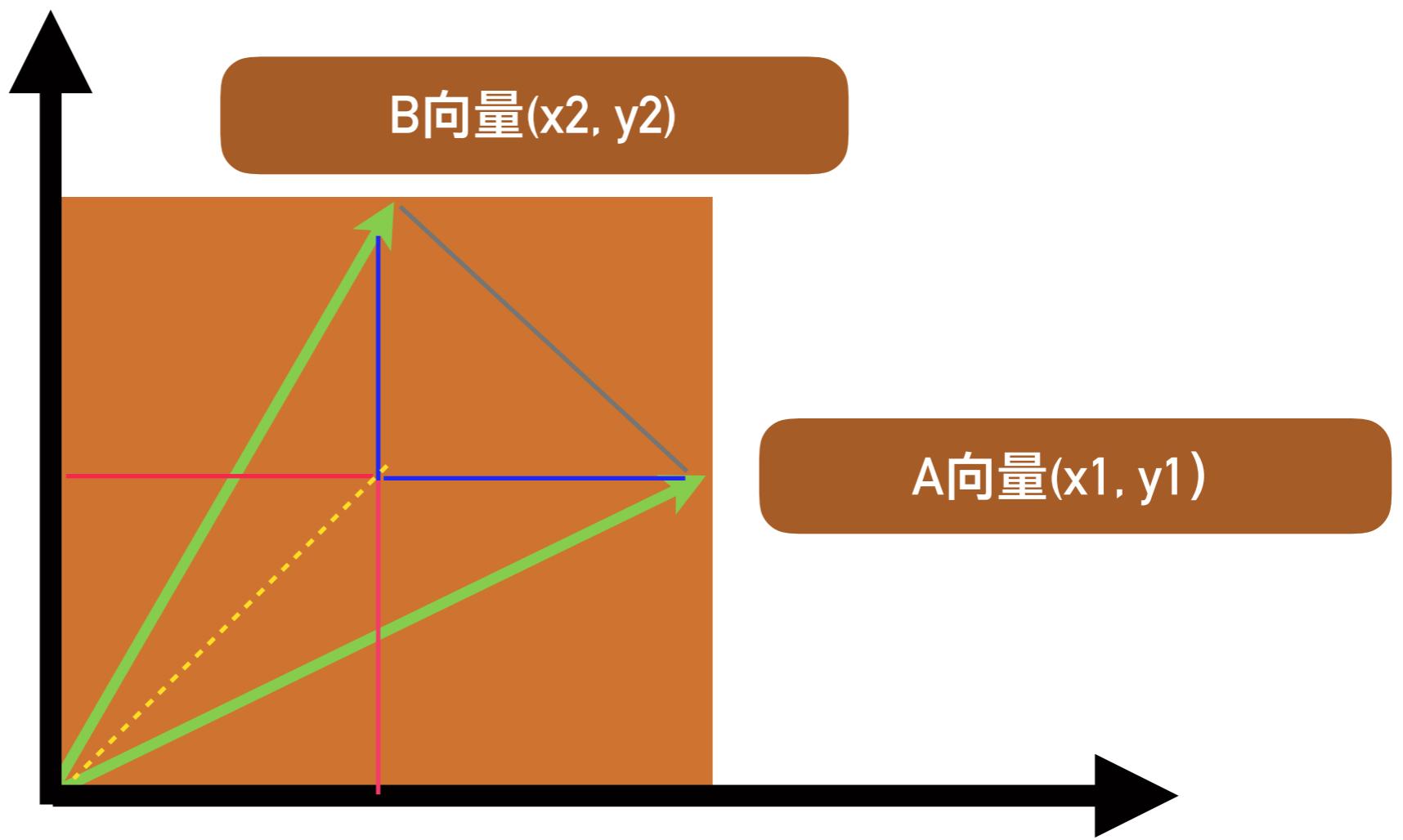
叉积

外积



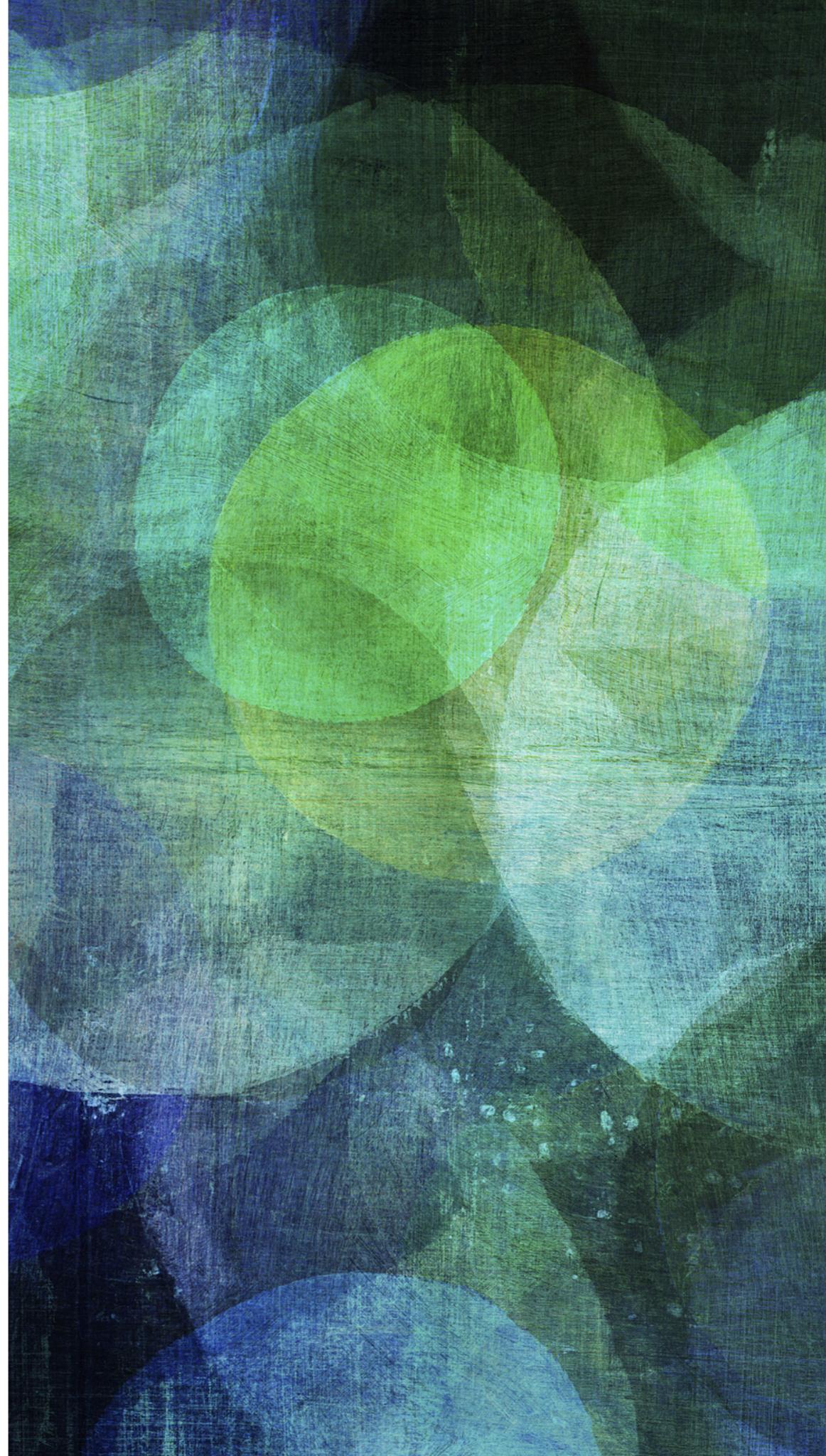
叉积

- ▶ 两个向量A(x₁,y₁), B(x₂,y₂)的叉积可以表示为x₁*y₂-x₂*y₁
- ▶ 叉积的值如果为正表示从B向量在A向量的左边，如果为负表示B向量在A向量的右边，即假设A, B向量都从原点O出发，那么从OA到OB如果在顺时针180度范围内，叉积就为正，否则为负



我们可以发现 $x1 * y2$ 表示整个矩形的面积， $x2 * y1$ 表示小矩形的面积，大矩形减去小矩形就是三角形面积的两倍

直线相交



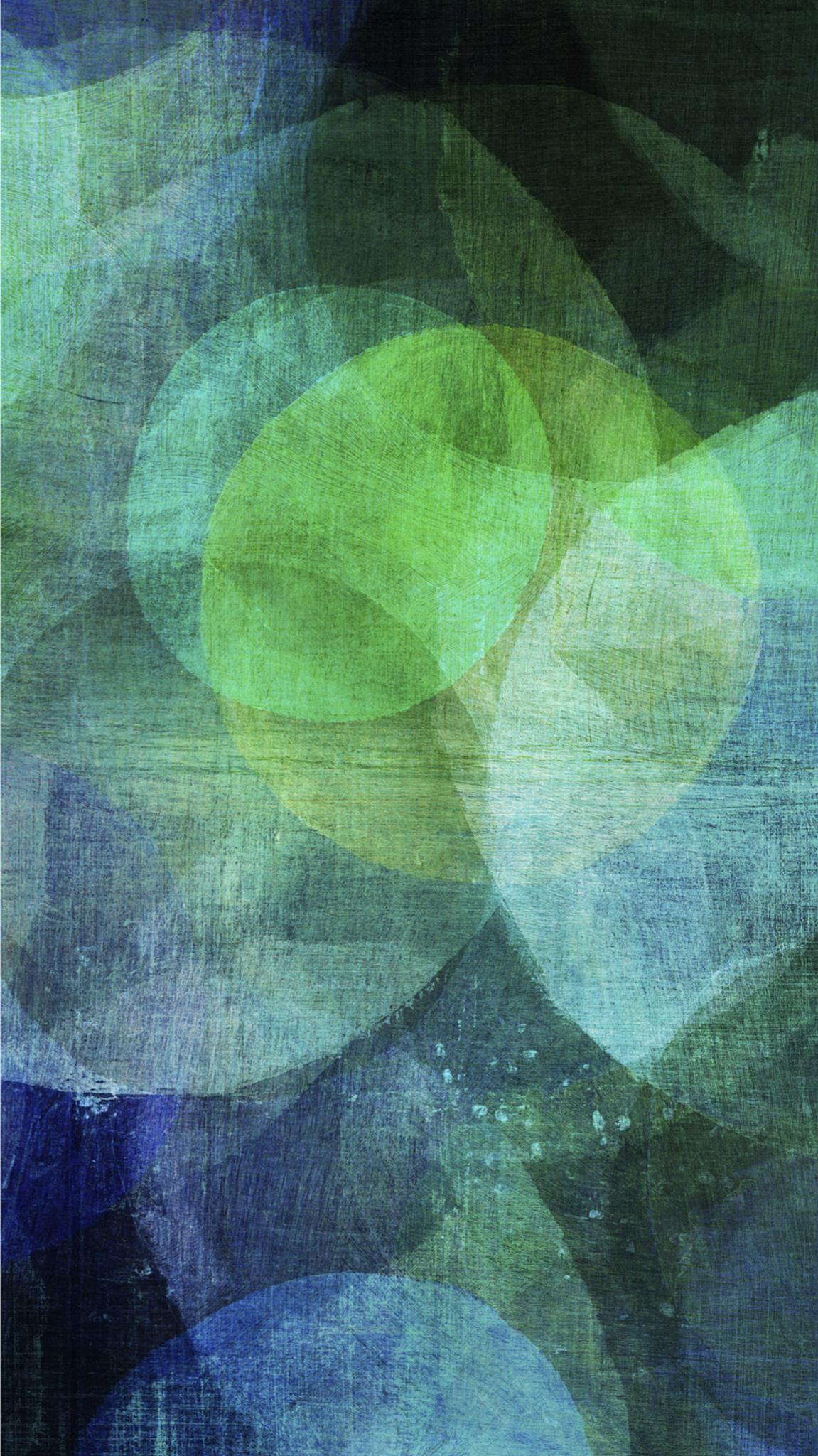
直线相交

- 求两直线交点需要先判断是否平行或者共线
- 求交点时可以利用参数方程来做

```
Point intersect(Point P, Vector v, Point Q, Vector w)  
{  
    Point ret;  
    Vector u = P - Q;  
    if(sgn(v * w) == 0) return false;  
    double t = w * u / (v * w);  
    ret = P + v * t;  
    return ret;  
}
```

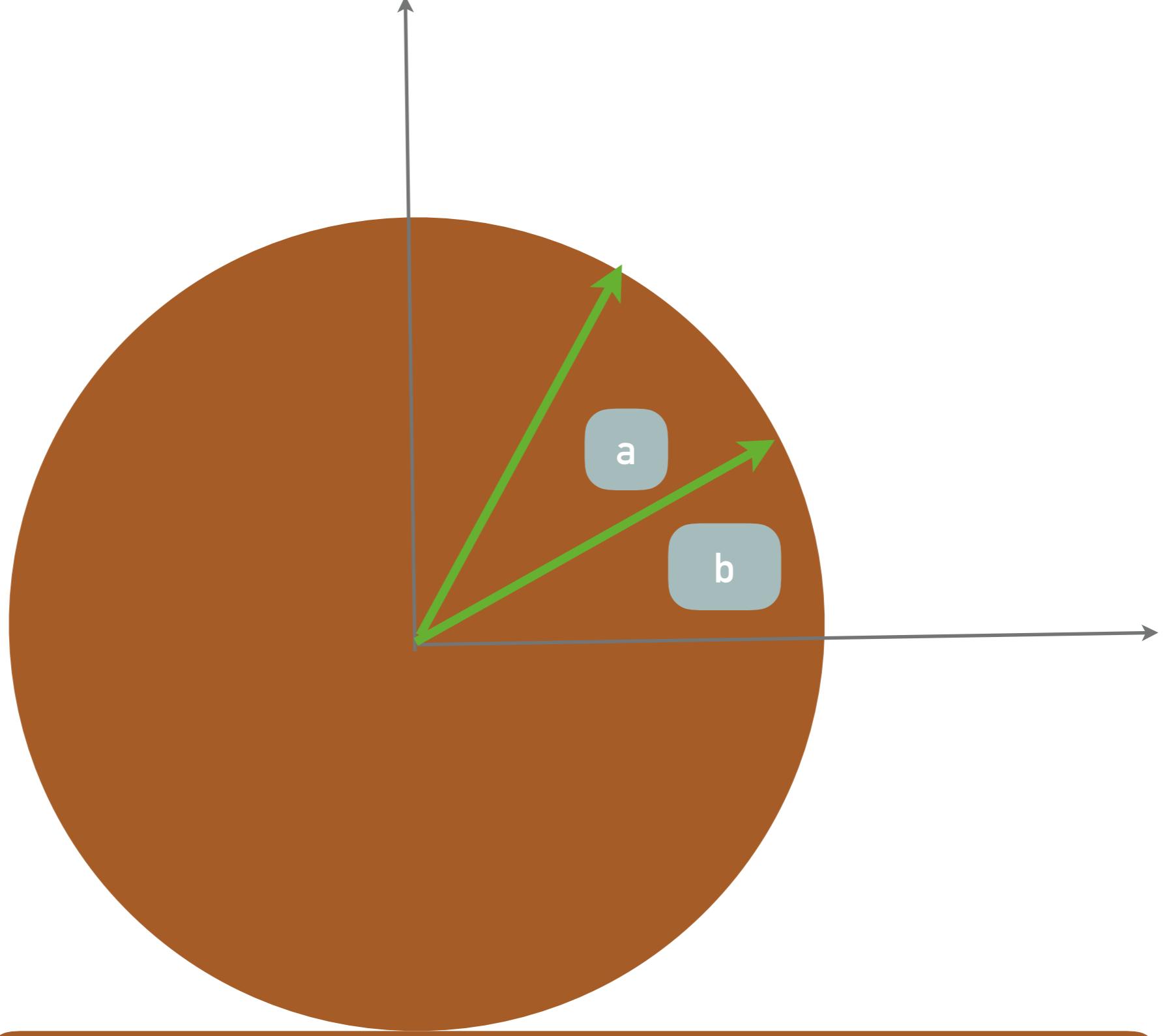
- 将第一条直线上的点表示成($P.x + v.x * t_1, P.y + v.y * t_1$)的形式
- 将第二条直线上的点表示成($Q.x + w.x * t_2, Q.y + w.y * t_2$)的形式
- 两项分别相等可以解出 t_1 和 t_2 ，就可以求出交点
- 求线段交点可以先求直线交点再判断交点是否在线段上

坐标旋转



坐标旋转

- 点 (x_1, y_1) 以 (x_0, y_0) 为中心逆时针旋转 a 角度
- 等价于 (x_1-x_0, y_1-y_0) 绕原点旋转，求出旋转后的坐标加上 (x_0,y_0) 即可
- (x,y) 绕原点逆时针旋转 a 角度会得到
- $(x * \cos(a) - y * \sin(a), x * \sin(a) + y * \cos(a))$



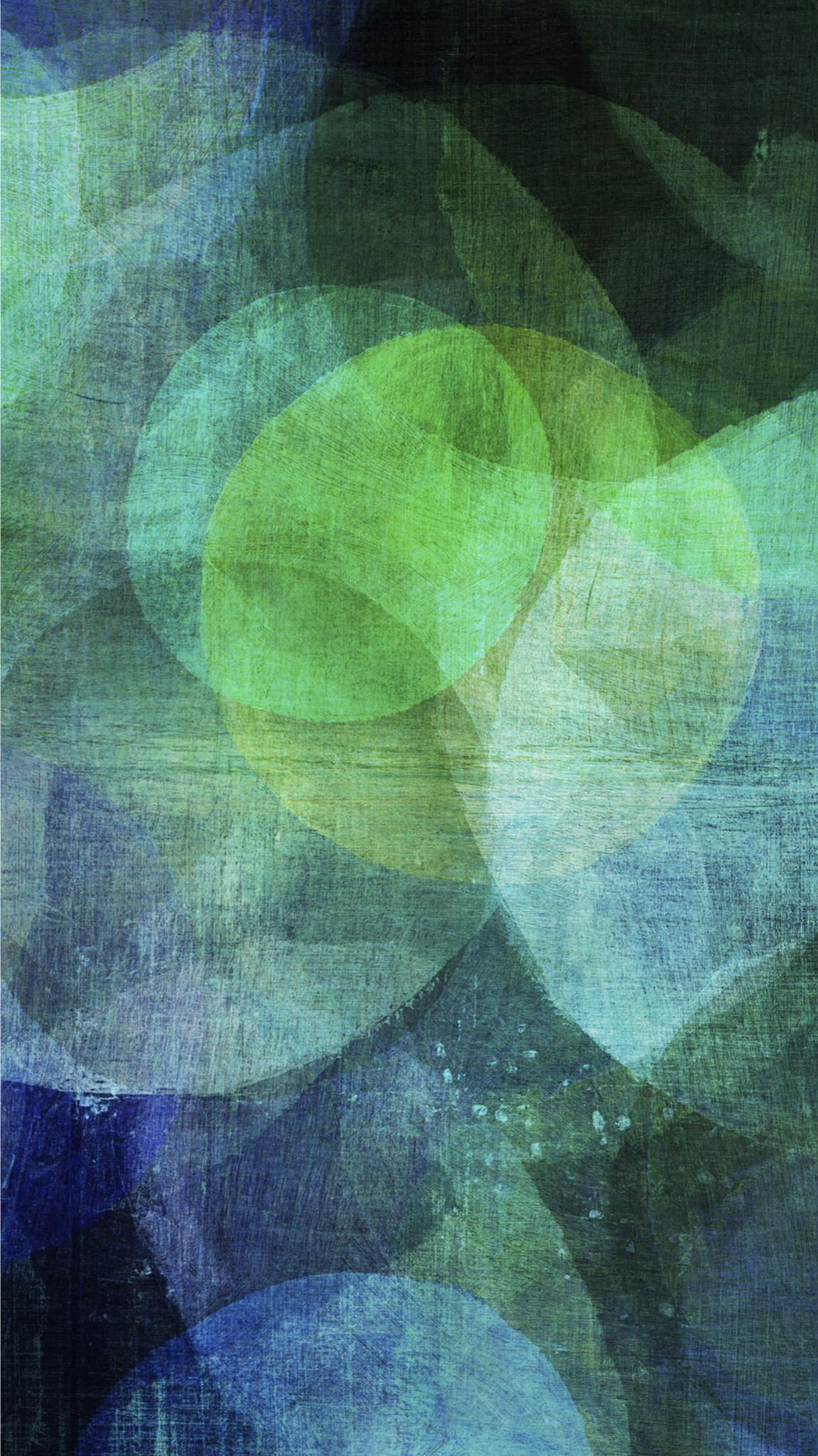
新的坐标为 $(R \cos(a+b), R \sin(a+b))$

根据 $\cos(a + b) = \cos(a)\cos(b) - \sin(a)\sin(b)$

$\sin(a+b) = \sin(a)\cos(b) + \cos(a)\sin(b)$

我们已知 $\sin b$, $\cos b$, 因此根据上述公式可以求出新的坐标

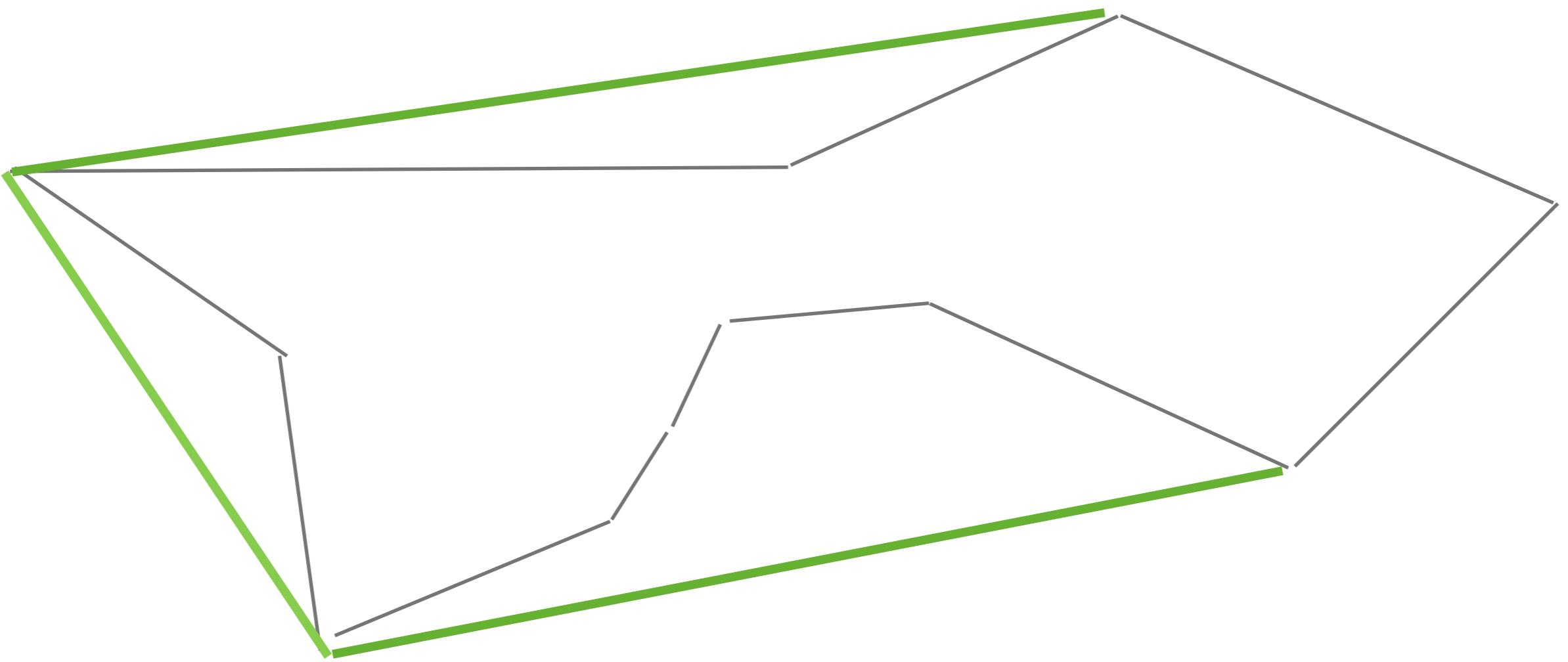
包



- 常见求凸包的方法有两种：水平序和极角序
- 水平序是先求下凸折线，再求上凸折线，最后拼成凸包
- 极角序是选择一个左下角的点当作原点，所有的点按照与原点连线后跟x轴正向的角度排序(此处可以学一下atan2函数)
- 排序之后的操作都差不多，下面以水平序为例
- 凸包的思想也经常用于其它与单调性有关的算法中，比如斜率优化

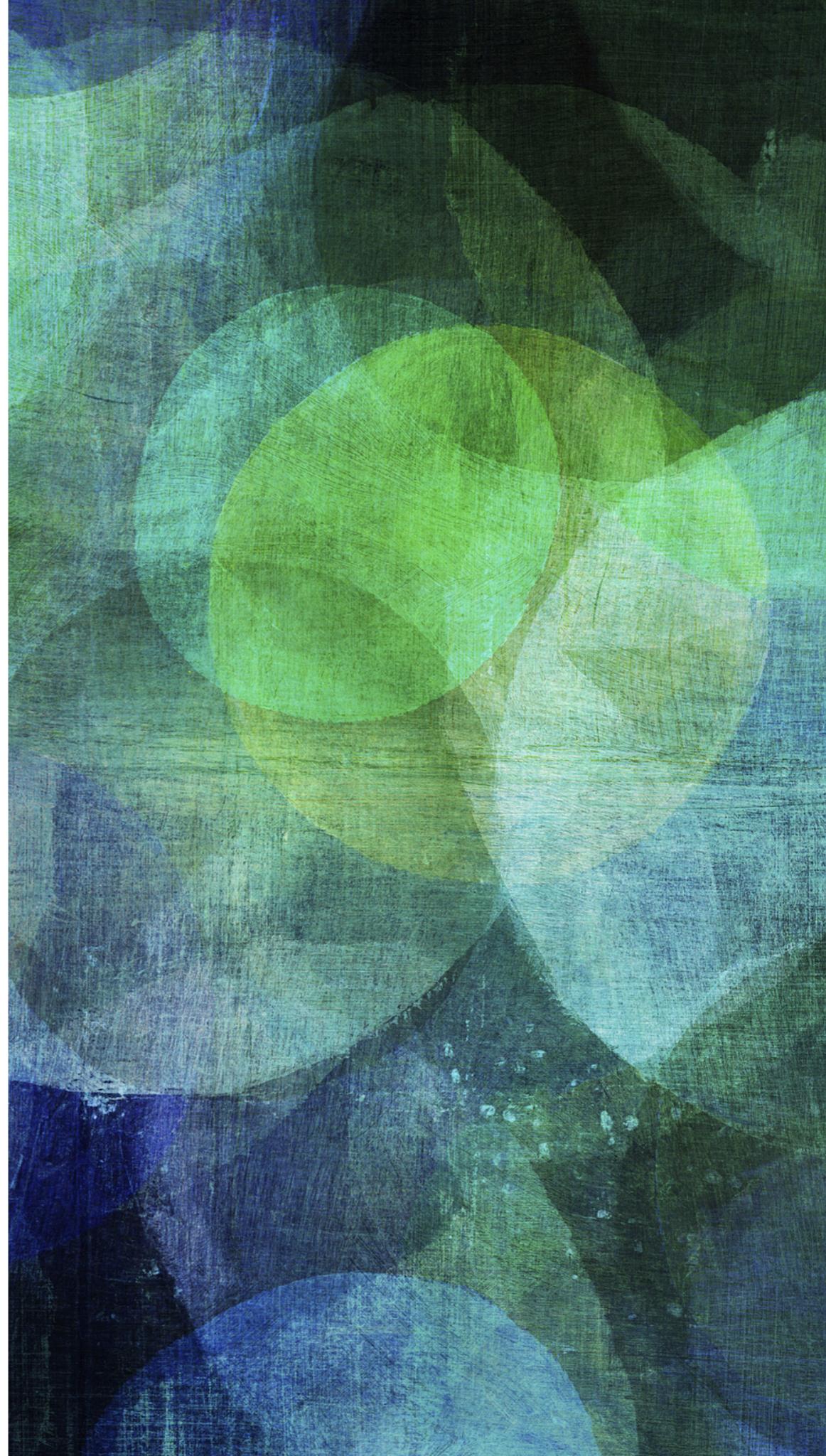
水平序凸包

- 将凸包分成上下两条凸折线
- 用一个栈记录折线上的点，假设栈顶的元素为b，前一个为a
- 假设当前点为c
- 如果 $\text{cross}(a, b, c) \geq 0$ 则直接将c加入栈中
- 否则将b退栈，一直到 $\text{cross}(a, b, c) \geq 0$



```
void gao() {
    if(n < 3) {
        return ;
    }
    std::sort(p, p + n);
    std::copy(p, p + n - 1, p + n);
    std::reverse(p + n, p + 2 * n - 1);
    int m = 0, top = 0;
    for(int i = 0; i < 2 * n - 1; i++) {
        while(top >= m + 2 && sgn((p[top - 1] - p[top - 2]) * (p[i] - p[top - 2])) <= 0) {
            top--;
        }
        p[top++] = p[i];
        if(i == n - 1) {
            m = top - 1;
        }
    }
    n = top - 1;
}
```

半平面交



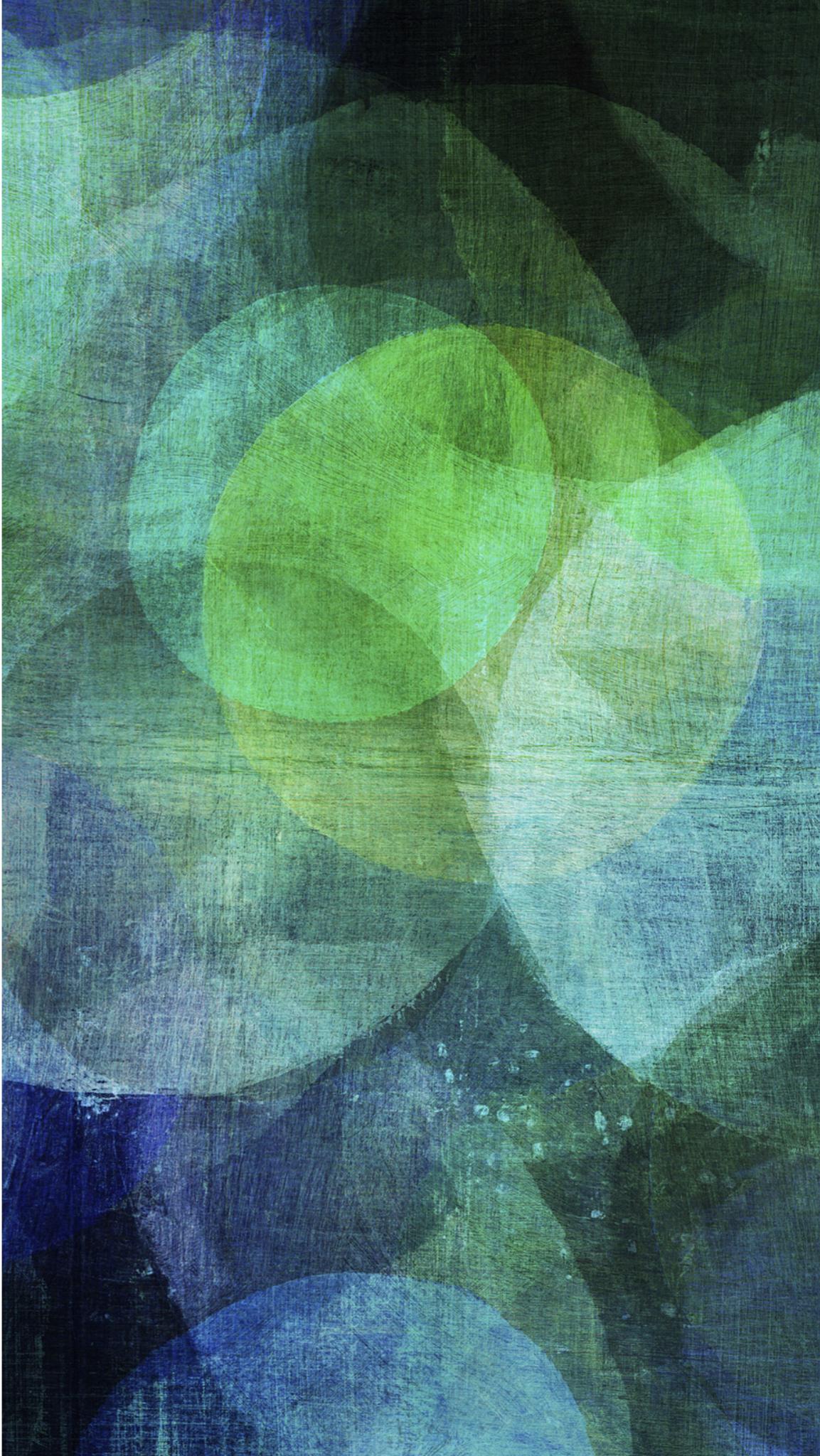
半平面交

- 有若干个 $ax+by+c = 0$ 的直线，每个直线有一侧是我们需要的一个半平面，求所有半平面的交集（一个凸多边形）

- 有 n^2 与 $n\log n$ 两种做法
- 此处介绍 n^2 做法
- 维护一个凸包，表示半平面的交，每次新来一条直线，暴力找到这条直线会与哪条凸包上的边相交（如果某条边两个端点在直线的异侧，肯定有交点）然后求出新的交点，维护新的凸包
- $n\log n$ 的算法使用双端队列优化，可参考刘汝佳训练指南

```
point intersect(point P, Vector v, point Q, Vector w) {
    point p;
    Vector u = P - Q;
    double t = cross(w, u) / cross(v, w);
    p = P + v * t;
    return p;
}
int inhalfplane(point p, point s, point e) {
    return sgn(cross(e - s, p - s));
}
std::vector<point> CUT(const std::vector<point> &p, point s, point e) {
    std::vector<point> q;
    int n = (int) p.size();
    for(int i = 0; i < n; i++) {
        int nowin = inhalfplane(p[i], s, e);
        int nextin = inhalfplane(p[(i + 1) % n], s, e);
        if(nowin >= 0) {
            q.push_back(p[i]);
        }
        if(nextin * nowin < 0) {
            q.push_back(intersect(p[i], p[(i + 1) % n] - p[i], s, e - s));
        }
    }
    return q;
}
```

多边形



多边形的常见应用

- 求多边形的面积（不一定是凸的）：随便按照一个方向求相邻三个点构成的有向三角形的面积之和，最后取绝对值
- logn判断点是否在凸多边形内部：二分判断点在哪一块三角形区域内，最后再判断点是否在三角形内部
- 判断点在任意一个多边形内部：射线法，做一条线，判断与多边形相交几次，奇数次就表示在多边形内部，否则在外部

```
bool point_in_polygon(Point o, Point *p, int n)
{
    int t;
    Point a, b;
    p[n] = p[0];
    for(int i = 0; i < n; i++) {
        if(dot_on_seg(o, Seg(p[i], p[i + 1]))) {
            return true;
        }
    }
    t = 0;
    for(int i = 0; i < n; i++) {
        a = p[i]; b = p[i + 1];
        if(a.y > b.y) {
            std::swap(a, b);
        }
        if(sgn((a - o) * (b - o)) < 0 && sgn(a.y - o.y) < 0 && sgn(o.y - b.y) <= 0) {
            t++;
        }
    }
    return t & 1;
}
```