第5章　计算几何

计算几何是计算机科学中研究解决几何问题算法的一个分支。在现代工程与数学中，计算几何应用于计算机图形学、机器人技术、大规模集成电路设计、计算机辅助设计以及统计学等多种领域。计算几何问题的输入往往是一个描述性的几何对象的集合，诸如点的集合、直线段的集合或是按时针方向排列的多边形顶点。输出通常是对那些对象询问的回应，如各条直线是否相交？或是另一个新的几何对象，如点集的凸壳（最小的封闭凸多边形）。

本章讨论一些平面上的计算几何算法。每一个输入的对象表示为点的集合$\{p_1, p_2, p_3, \cdots\}$，其中每一个$p_i=(x_i, y_i)$且$x_i, y_i \in \mathbf{R}$。例如，$n$个顶点的多边形$P$表示为其边界上的$n$个顶点按出现的顺序构成序列$\{p_0, p_1, p_2, \cdots, p_{n-1}\}$。计算几何也可以用于三维，甚至更高维的空间，然而这样的高维空间中的问题及其解都很难形象化，而在二维空间中却可以看到一个很好的例子。

5.1　线段的性质

本章的几个计算几何算法都要求回答关于线段的性质问题。两个不同的点$p_1=(x_1, y_1)$和$p_2=(x_2, y_2)$，任何满足$0 \leqslant \alpha \leqslant 1$，$x_3=\alpha x_1+(1-\alpha)x_2$，$y_3=\alpha y_1+(1-\alpha)y_2$的点$p_3=(x_3, y_3)$，称为$p_1$、$p_2$的**凸组合**。也可以写成$p_3=\alpha p_1+(1-\alpha)p_2$。直观地说，$p_3$是$p_1$和$p_2$连线上介于$p_1$和$p_2$之间的点。给定两个点$p_1$，$p_2$，**线段**$\overline{p_1p_2}$是$p_1$、$p_2$凸组合点的集合。称$p_1$，$p_2$为线段$\overline{p_1p_2}$的**端点**。有时，要考虑$p_1$，$p_2$的顺序，我们就称其为**有向线段**$\overrightarrow{p_1p_2}$。若$p_1$是原点$(0,0)$，我们把有向线段$\overrightarrow{p_1p_2}$视为向量$p_2$。

在本节中，将探索以下问题。

（1）给定两条有向线段$\overrightarrow{p_0p_1}$和$\overrightarrow{p_0p_2}$，$\overrightarrow{p_0p_1}$是否绕它们的公共端点p_0从$\overrightarrow{p_0p_2}$顺时针方向旋转而得？

（2）给定线段$\overline{p_1p_2}$和$\overline{p_2p_3}$，如果先沿$\overline{p_1p_2}$行进再沿$\overline{p_2p_3}$行进是否需要在点p_2处左转弯？

（3）线段$\overline{p_1p_2}$和$\overline{p_3p_4}$是否相交？

由于每个问题的输入规模都是$O(1)$，无疑均可在$O(1)$的时间内得到回答。此外，这个方法将仅使用加法、减法、乘法和比较运算。既不需要除法也不需要三角函数，这两种运算都很费时且容易产生舍入误差。例如，用“直接”方法判断两条线段是否相交——计算每条线段的直线方程$y=mx+b$（m为斜率，b为y轴上的截距），求出两直线的交点并检测该交点是否在两条线段中。这就需要使用除法，当两条线段几乎平行时，在实际的计算机中此问题对除法运算的精度是很敏感的。本节中的方法避免了除法，因此更加精确。

5.1.1　叉积及其应用

1. 叉积

叉积的计算是解决上述线段问题的方法之核心。考虑两个向量 p_1 和 p_2，如图 5-1(a)所示。**叉积** $p_1 \times p_2$ 可以解释为由点 (0,0)、p_1、p_2 和 $p_1+p_2=(x_1+x_2,\ y_1+y_2)$ 构成的平行四边形的带符号面积。一个等价的且更有用的定义是叉积是下列矩阵的行列式[①]：

$$\begin{aligned} p_1 \times p_2 &= \det\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -\,p_2 \times p_1 \end{aligned}$$

图 5-1(a)所示为向量 p_1 和 p_2 的叉积，是平行四边形的带符号面积。图 5-1(b)为浅阴影区域包含由向量 p 出发顺时针旋转的所有向量，而深阴影部分则包含从向量 p 出发反时针旋转的所有向量。

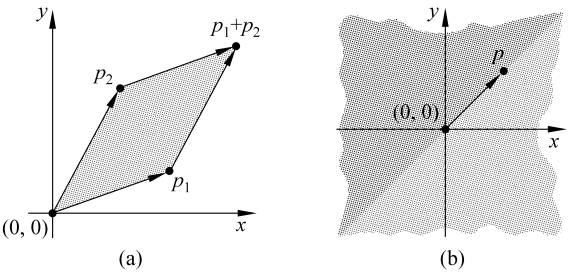

图 5-1　向量及其叉积

利用对向量叉积的定义，下列的定理是很有用的。

定理 5-1　假定向量 p_1 和 p_2 所张角介于 0～π 之间，则：

(1) 若 $p_1 \times p_2$ 为负当且仅当 p_1 是从 p_2 反时针方向旋转而得；

(2) 若 $p_1 \times p_2$ 为正当且仅当 p_1 是从 p_2 绕原点(0,0)顺时针方向旋转而得；

(3) 若 $p_1 \times p_2$ 为零，则产生边界条件，此时，两个向量**共线**，或方向一致或方向相反。

图 5-2 形象地说明了定理 5-1。

图 5-2(a)所示向量 p_1 是从 p_2 反时针方向旋转而得。平行四边形 $Op_1p_3p_2$ 的面积＝三角形 Op_1p_1' 面积＋梯形 $p_1p_3p_2'p_1'$－三角形 Op_2p_2'－梯形 $p_2'p_2p_3p_3'=x_1y_1/2+x_2(2y_1+y_2)/2-x_2y_2/2-x_1(2y_2+y_1)/2=x_2y_1-x_1y_2>0$。图 5-2(b)中向量 p_1 是从 p_2 反时针方向旋转而得。平行四边形 $Op_2p_3p_1$ 的面积＝梯形 $p_2'p_2p_3p_3'$ 面积＋梯形 $p_3'p_3p_1p_1'$－三角形 $p_2'p_2O$ 面积－三角形 $Op_1p_1'=-y_1(2x_2+x_2)/2+y_2(2x_1+x_2)/2-x_2y_2/2+x_1y_1/2=x_1y_2-x_2y_1>0$。图 5-2(c)向量 p_1、p_2 共线。$y_2/x_2=y_1/x_1 \Rightarrow x_2y_1-x_1y_2=0$。

利用定理 5-1，可直接判定有向线段 $\overrightarrow{p_0p_1}$ 是否从有向线段 $\overrightarrow{p_0p_2}$ 绕它们的公共端点 p_0 顺

① 事实上，叉积是一个三维概念。它是一个按“右手法则”既垂直于 p_1 又垂直于 p_2 的向量，其模长为 $|x_1y_2-x_2y_1|$。然而，本章的内容说明，将叉积视为值 $x_1y_2-x_2y_1$ 是方便的。

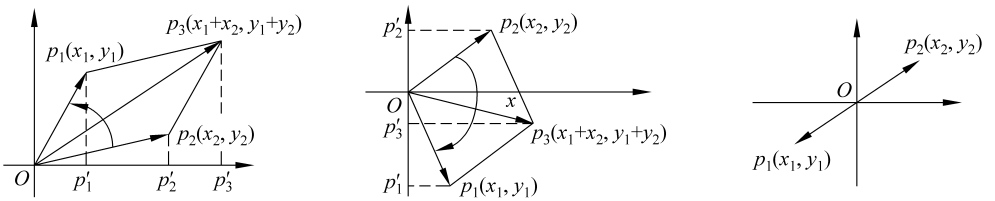

图 5-2

时针旋转而得。为此，只要将 p_0 转换为原点。即设 p_1-p_0 表示向量 $p_1'=(x_1', y_1')$，其中 $x_1'=x_1-x_0$ 及 $y_1'=y_1-y_0$。类似地，定义 p_2-p_0。然后计算叉积：

$$(p_1-p_0)\times(p_2-p_0)=(x_1-x_0)(y_2-y_0)-(x_2-x_0)(y_1-y_0)$$

如果此叉积为正，则 $\overrightarrow{p_0p_1}$ 是从有向线段 $\overrightarrow{p_0p_2}$ 绕它们的公共端点 p_0 顺时针旋转而得；若为负，则为逆时针方向。

2. 判断相继两直线段左转或右转

下一个问题是两条相继线段 $\overline{p_0p_1}$ 和 $\overline{p_1p_2}$ 是在点 p_1 处左转还是右转。等价地，要设法判定给定角 $\angle p_0p_1p_2$ 的转向。叉积使得我们可以不计算角而回答此问题。

```
DIRECTION(p0, p1, p2)                ▷计算 p0p1 , p1p2 的转向
1 return (p2-p0)×(p1-p0)
```

算法 5-1 计算相继线段 $\overline{p_0p_1}$、$\overline{p_1p_2}$ 的转向

如图 5-3 所示，直接检测有向线段 $\overrightarrow{p_0p_2}$ 是从有向线段 $\overrightarrow{p_0p_1}$ 是顺时针还是逆时针旋转而得。为此，计算叉积 $(p_2-p_0)\times(p_1-p_0)$。若此叉积为负，则 $\overrightarrow{p_0p_2}$ 是从有向线段 $\overrightarrow{p_0p_1}$ 逆时针旋转而得的，因此在 p_1 处左转。正的叉积值意味着顺时针并右转。叉积为 0 意味着 p_0、p_1 和 p_2 共线。

图 5-3 所示为使用叉积判断相继线段 $\overline{p_0p_1}$、$\overline{p_1p_2}$ 在 p_1 处的转向。检测有向线段 $\overrightarrow{p_0p_2}$ 是从 $\overrightarrow{p_0p_1}$ 顺时针还是逆时针旋转而得。图 5-3(a)若是逆时针，则在该点处左转。图 5-3(b)若是顺时针，则为右转。

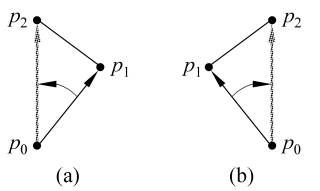

图 5-3 叉积及转向

3. 判断两条线段是否相交

为确定两条线段是否相交，检测每条线段是否跨越包含另一条线段的直线。线段 $\overline{p_1p_2}$ 的端点 p_1 位于一条直线的一边，而端点 p_2 位于另一边，则该线段**跨越**此直线。若 p_1 或 p_2 位于直线上则发生边界情形。两条线段相交当且仅当下列两个条件至少发生一个。

(1) 每一条线段跨越包含另一条线段的直线。

(2) 一条线段的一个端点位于另一条线段上。

下列的过程实现了这一思想。若两条线段相交，SEGMENTS-INTERSECT 返回 TRUE，若不相交，返回 FALSE。它调用子过程 DIRECTION，该过程利用上述的叉积方法计算相对方位，还要调用子过程 IN-BOX，该过程确定一个点是否落于以已知线段为对角线的矩形内。

```
SEGMENTS-INTERSECT(p1, p2, p3, p4)          ▷检测线段p1p2与p3p4是否相交
1  d1←DIRECTION(p3, p4, p1)                 ▷计算p4p3到p1p3的转向
2  d2←DIRECTION(p3, p4, p2)                 ▷计算p4p3到p2p3的转向
3  d3←DIRECTION(p1, p2, p3)                 ▷计算p2p1到p3p1的转向
4  d4←DIRECTION(p1, p2, p4)                 ▷计算p2p1到p4p1的转向
5  if ((d1 * d2<0) and (d3 * d4<0))
6        then return TRUE
7  elseif d1=0 and IN-BOX(p3, p4, p1)
8        then return TRUE
9  elseif d2=0 and IN-BOX(p3, p4, p2)
10       then return TRUE
11 elseif d3=0 and IN-BOX(p1, p2, p3)
12       then return TRUE
13 elseif d4=0 and IN-BOX(p1, p2, p4)
14       then return TRUE
15 else return FALSE

IN-BOX(pi, pj, pk)                          ▷检测点pk是否处于以pipj为对角线的矩形内
1  if min(xi, xj)≤xk≤max(xi, xj) and min(yi, yj)≤yk≤max(yi, yj)
2       then return TRUE
3       else return FALSE
```

算法 5-2　检测两条线段是否相交的算法

图 5-4 所示为过程 SEGMENTS-INTERSECT 中的各种情形。图 5-4(a)为线段$\overline{p_1p_2}$和$\overline{p_3p_4}$相互跨越对方所在直线。由于$\overline{p_3p_4}$跨越包含$\overline{p_1p_2}$的直线，叉积$(p_3-p_1)\times(p_2-p_1)$和$(p_4-p_1)\times(p_2-p_1)$的符号相反。由于$\overline{p_1p_2}$跨越包含$\overline{p_3p_4}$的直线，叉积$(p_1-p_3)\times(p_4-p_3)$和$(p_2-p_3)\times(p_4-p_3)$的符号相反。图 5-4(b)为线段$\overline{p_3p_4}$跨越包含$\overline{p_1p_2}$的直线，但$\overline{p_1p_2}$并不跨越包含$\overline{p_3p_4}$的直线。叉积$(p_1-p_3)\times(p_4-p_3)$和$(p_2-p_3)\times(p_4-p_3)$的符号相同。图 5-4(c)为点 p_3与 p_1和 p_2共线且介于两者之间。图 5-4(d)为点 p_3与$\overline{p_1p_2}$共线，但它并不介于 p_1和 p_2之间。两条线段不相交。

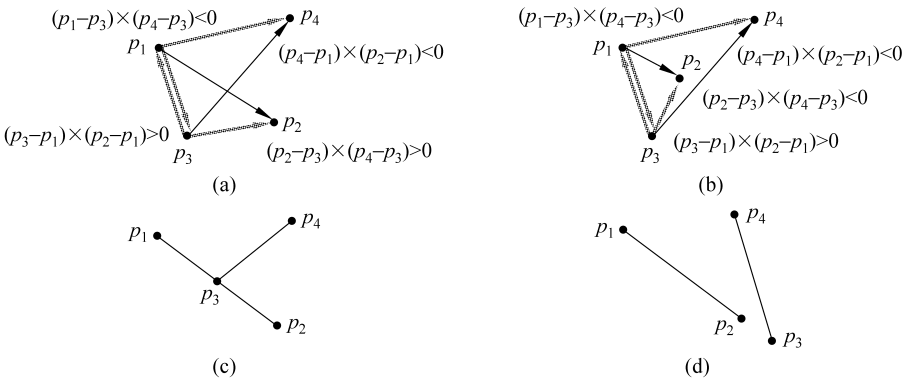

图 5-4　平面上两条线段的相交的各种情形

SEGMENTS-INTERSECT 运行如下。第 1～4 行计算每一个端点 p_i 关于其他线段的相对方位 d_i。若所有的相对方位非零，则不难确定两条线段 $\overline{p_1p_2}$ 和 $\overline{p_3p_4}$ 是否相交。若有向线段 $\overrightarrow{p_3p_1}$ 和 $\overrightarrow{p_3p_2}$ 具有相对于 $\overrightarrow{p_3p_4}$ 的正的方位，线段 $\overline{p_1p_2}$ 跨越包含线段 $\overline{p_3p_4}$ 的直线。此时，d_1 和 d_2 的符号相反。相似地，若 d_3 和 d_4 的符号相反，则 $\overline{p_3p_4}$ 跨越包含 $\overline{p_1p_2}$ 的直线。若第 5 行的检测为真，则两线段相互跨越，且 SEGMENTS-INTERSECT 返回 TRUE。图 5-4(a)展示了此情形。否则，两条线段并不相互跨越对方所在的直线，但可能会发生边界情形。若所有的相对方位均非零，没有边界情形发生。若第 7～13 行的所有对 0 的测试都失败，则 SEGMENTS-INTERSECT 在第 15 行返回 FALSE。图 5-4(b)展示了这一情形。

任一相对方位 d_k 为 0，则发生边界情形。此时，知道 p_k 与另一条线 $\overline{p_ip_j}$ 段共线。过程 IN-BOX(p_i，p_j，p_k)返回 p_k 是否介于线段 $\overline{p_ip_j}$ 的两个端点之间；该过程假定 p_k 与 $\overline{p_ip_j}$ 共线。图 5-4(c)和图 5-4(d)展示了共线点的情形。在图 5-4(c)中，p_3 在上 $\overline{p_1p_2}$，所以 SEGMENTS-INTERSECT 在第 12 行返回 TRUE。在图 5-4(d)中，没有端点在另一条线段上，所以 SEGMENTS-INTERSECT 在第 15 行返回 FALSE。

5.1.2 向量的极角

在一些几何算法中常用到极角的概念。把向量 p_1-p_0 与 x 正半轴的夹角称为点 p_1 关于点 p_0 的**极角**。例如，(3,5)关于(2,4)的极角是向量(1,1)与 x 正半轴的夹角 45°或 $\pi/4$ 弧度。(3,3)关于(4,2)的极角是向量(−1,1)与 x 正半轴的夹角 135°或 $3\pi/4$ 弧度。

算法中常需要比较两个向量极角的大小。数学中，要计算每个向量与 x 正半轴夹角的正切值 $\frac{y-y_0}{x-x_0}$，然后利用反正切函数 arctan 以及向量所在的象限计算出极角。下面考虑一种不使用三角函数和反三角函数，而比较向量间极角大小的方法。不失一般性，设非零向量 $p(x,y)$ 的模 $|p|$(p 到原点 O 的距离 $\sqrt{x^2+y^2}$)为 1，(如果 $|p|\neq 1$，可按下述方法进行**规格化**：令 $p'=p/|p|=\left(\frac{x}{\sqrt{x^2+y^2}},\frac{y}{\sqrt{x^2+y^2}}\right)$，则 p' 与 p 方向一致，且 $|p'|=1$。)记 p 与 x 正半轴的夹角为 α。设 $p_0(1,0)$ 为 x 轴上的单位向量，记为 (x_0,y_0)，其与 x 轴正向夹角 $\beta=0$。由于 p、p_0 都是规格化了的向量，所以 $x=\cos\alpha$，$y=\sin\alpha$，$x_0=\cos\beta$，$y_0=\sin\beta$。考虑叉积

$$\begin{aligned} p_0\times p &= yx_0-xy_0\\ &= \sin\alpha\cos\beta-\cos\alpha\sin\beta\\ &= \sin(\alpha-\beta)\\ &= \sin\alpha \end{aligned}$$

在数学中，单位圆上的弧长 α 较小时($0<\alpha<\pi/2$)，$\sin\alpha\approx\alpha$。当 p 位于第Ⅰ象限时，如图 5-5(a)所示，就用 $y=\sin\alpha$ 替代 p 的极角 α。当 p 位于第Ⅱ象限时，如图 5-5(b)所示，$\sin\alpha=\sin(\pi/2+\alpha')$。此时，用 $\pi/2-x$ 替代 p 的极角 α。当 p 位于第Ⅲ象限时，如图 5-5(c)所示，$\sin\alpha=\sin(\pi+\alpha')$。此时，用 $\pi-y$ 替代 p 的极角 α。当 p 位于第Ⅳ象限时，如图 5-5(d)所示，$\sin\alpha=\sin(3\pi/2+\alpha')$。此时，用 $3\pi/2+x$ 替代 p 的极角 α。

图 5-5 所示为规格化向量 p 与 $p_0=(1,0)$ 的叉积与极角的关系。图 5-5(a)中，p 位于

第Ⅰ象限，$p_0 \times p = \sin\alpha$。图 5-5(b)中，$p$ 位于第Ⅱ象限，$p_0 \times p = \sin\alpha = \sin(\pi/2 + \alpha')$。图 5-5(c)中，$p$ 位于第Ⅲ象限，$p_0 \times p = \sin\alpha = \sin(\pi + \alpha')$。图 5-5(d)中，$p$ 位于第Ⅳ象限，$p_0 \times p = \sin\alpha = \sin(3\pi/2 + \alpha')$。

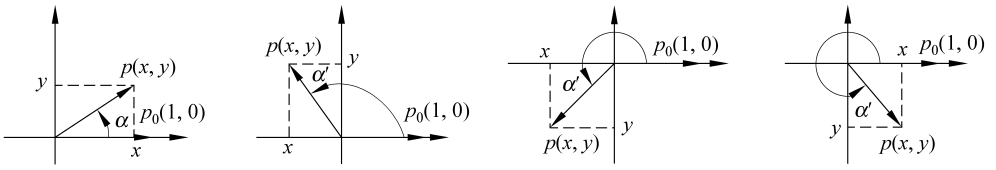

图 5-5　叉积与极角的关系

这样，无须真正计算出 p 的极角，却可以对两个规格化后的向量比较极角的大小。把上述替代极角的值称为向量的伪极角。下列过程描述了计算向量 p 相对于 p_0 的伪极角(即向量 $p - p_0$ 的伪极角)算法。

```
PSUDO-POLAR-ANGLE(p, p0)
1 p1(x1, y1) = p − p0
2 将 p1 规格化
3 if x1 ≥ 0 且 y1 ≥ 0              ▷第Ⅰ象限
4      then return y1
5 if x1 < 0 且 y1 ≥ 0              ▷第Ⅱ象限
6      then return π/2 − x1
5 if x1 < 0 且 y1 < 0              ▷第Ⅲ象限
7      then return π − y1
8 return 3π/2 + x1                 ▷第Ⅳ象限
```

算法 5-3　计算向量 $p - p_0$ 的伪极角

5.1.3　程序实现

1. 平面中点的数据表示

实现平面上点的数据类型如下所示。

```
1  #define epsilon 1e−10
2  #define PI 3.1415926
3  typedef struct{                                  /* 平面点 */
4      double x,y;                                  /* 横坐标与纵坐标 */
5  }Point;
6  double dist(Point *a,Point *b){                  /* 两点间距离 */
7      return sqrt(pow(a->x−b->x,2)+pow(a->y−b->y,2));
8  }
9  Point sub(Point *a,Point *b){                    /* 两点向量差 */
10     Point c={a->x−b->x,a->y−b->y};
11 return c;
```

```
12 }
13 double crossProduct(Point *a,Point *b){          /*叉积*/
14     return a->x*b->y-a->y*b->x;
15 }
```

程序 5-1 平面点数据类型及常规维护函数

程序 5-1 的说明如下。

(1) 第 3～5 行定义了表示平面上点的结构体类型 Point。Point 类型的数据具有 2 个 **double** 型的数据属性：点的横坐标 x 和纵坐标 y。

(2) 第 6～8 行定义的函数 dist 计算由参数 a、b 指引的两个 Point 点之间的距离。第 9～12 行定义的函数 sub 计算由参数 a、b 指引的两个 Point 点按坐标差计算所得的点(向量)。

(3) 第 13～15 行定义的函数 crossProduct 计算由参数 a、b 指引的两个 Point 点(向量)的叉积。

所有这些函数的定义代码都十分简单，读者不难阅读理解。Point 类型那个的定义及各函数的原型声明存储于文件夹 geometry 的头文件 point.h，函数的定义存储于同一文件夹的源文件 point.c 中。

2. 利用叉积检测线段性质

利用上述实现的平面上点的数据类型及向量的叉积计算函数，来实现算法 5-1 描述的 DERICTION 过程和算法 5-2 描述的 SEGMENTS-INTERSECT 过程。

```
1  int direction(Point *p0,Point *p1,Point *p2){   /*计算向量 p2p0、p1p0夹角方向*/
2      Point p=sub(p2,p0),q=sub(p1,p0);
3      double d=crossProduct(&p,&q);
4      if(d>0.0)                                   /*顺时针*/
5          return 1;
6      if(d<0.0)                                   /*逆时针*/
7          return-1;
8      return 0;                                   /*共线*/
9  }
10 int inBox(Point *pi,Point *pj,Point *pk){     /*检测 pk 落于以 pi、pj 为对角线的矩形内*/
11     double x1=pi->x<pj->x?pi->x:pj->x,
12         x2=pi->x>pj->x?pi->x:pj->x,
13         y1=pi->y<pj->y?pi->y:pj->y,
14         y2=pi->y>pj->y?pi->y:pj->y;
15         return x1<=pk->x&&pk->x<=x2&&y1<=pk->y&&pk->y<=y2;
16 }
17 int segmentsIntersect(Point *p1,Point *p2,Point *p3,Point *p4){
18     int d1=direction(p3,p4,p1),
19         d2=direction(p3,p4,p2),
20         d3=direction(p1,p2,p3),
21         d4=direction(p1,p2,p4);
```

```
22      if(d1==0&&inBox(p3,p4,p1))
23          return 1;
24      if(d1==0&&inBox(p3,p4,p1))
25          return 1;
26      if(d2==0&&inBox(p3,p4,p2))
27          return 1;
28      if(d3==0&&inBox(p1,p2,p3))
29          return 1;
30      if(d4==0&&inBox(p1,p2,p4))
31          return 1;
32      return 0;
33  }
```

程序 5-2　实现算法 5-1 和算法 5-2 的 C 函数

对程序 5-2 的说明如下。

(1) 第 1～9 行定义的函数 direction 实现算法 5-1 的 DIRECTION 过程，计算由指针参数 p0、p1、p2 指引的三个点构成的两条连续线段$\overline{p_0p_1}$和$\overline{p_1p_2}$的转向。第 2 行调用程序 5-1 中定义的函数 sub，计算向量 p_2-p_0 和 p_1-p_0，分别赋予 p 和 q。第 3 行调用函数 crossProduct，计算叉积$(p_2-p_0)\times(p_1-p_0)$并赋予 d。为便于使用，函数并不直接返回 d，而是根据 d 的符号返回 1 或 0 或－1。

(2) 第 10～16 行定义的函数 inBox 实现算法 5-2 中 IN-BOX 过程，检验由指针参数 pk 指引的点 pk 是否落在由指针参数 pi、pj 指引的点连成的线段$\overline{p_ip_j}$作为对角线的矩形内。其中，第 11～14 行计算 $\min(x_i,x_j)$、$\max(x_i,x_j)$、$\min(y_i,y_j)$和 $\max(y_i,y_j)$分别赋予 x1、x2、y1、y2。第 15 行检测条件 $\min(x_i,x_j)\leqslant x_k\leqslant\max(x_i,x_j)$且 $\min(y_i,y_j)\leqslant y_k\leqslant\max(y_i,y_j)$，并返回检测结果(条件成立返回 1 否则返回 0)。

(3) 第 17～33 行定义的函数 segmentsIntersect，实现算法 5-2 中的 SEGMENTS-INTERSECT 过程，检测由指针参数 p1、p2、p3 和 p4 指引的点构成的线段$\overline{p_1p_2}$与$\overline{p_3p_4}$是否相交。程序代码结构与算法的伪代码结构十分接近，读者可对比阅读，此处不再赘述。

3. 向量的伪极角

```
1  double pabs(Point *a){                          /*计算向量的模*/
2      Point o={0.0,0.0};                          /*原点*/
3      return dist(a,&o);
4  }
5  void normalize(Point *a){                       /*向量正规化(模长为1)*/
6      double r=pabs(a);
7      if(r>=epsilon)                              /*非0向量*/
8          a->x/=r,a->y/=r;
9  }
10 double psudoPolarAngle(Point *a,Point *b){      /*计算向量a关于向量b的极角*/
11     Point p1=sub(a,b);
12         normalize(&p1);
```

```
13      if(pabs(&p1)<epsilon)
14          return 2 * PI;
15      if(p1.y>=0.0&&p1.x>=0.0)              /* p 位于第Ⅰ象限 */
16          return p1.y;
17      if(p1.y>=0.0&&p1.x<0.0)               /* p 位于第Ⅱ象限 */
18          return PI/2-p1.x;
19      if(p1.y<0.0&&p1.x<0.0)                /* p 位于第Ⅲ象限 */
20          return PI-p1.y;
21      return 3 * PI/2+p1.x;                 /* p 位于第Ⅳ象限 */
22  }
```

程序 5-3 实现算法 5-3 计算向量 a 关于向量 b 的极角的 PSUDO-POLAR-ANGLE 的函数

对程序 5-3 的说明如下。

(1) 第 1～4 行定义的函数 pabs 计算由指针参数 a 所指引的点到原点的距离——原点到该点形成的向量的模。

(2) 第 5～9 行定义的函数 normalize 对指针参数 a 指引的点构成的向量做规格化操作——使得其模为 1。

(3) 第 10～22 行定义的函数 psudoPolarAngle 实现算法 5-3 的 PSUDO-POLAR-ANGLE 过程,计算由指针参数 a,b 指引的点构成的向量伪极角。程序代码结构与算法的伪代码结构十分接近,读者可对照阅读。

程序 5-2 和程序 5-3 中定义的各函数均存储于 geometry 文件夹的源文件 point.c,各函数的原型声明存储于同一文件夹的头文件中。

5.2 判断是否存在线段相交

判断线段是否相交的问题可如下形式化。

输入:线段集合 $S=\{s_1,s_2,\cdots,s_n\}$。

输出:如果 S 中存在两条线段相交,输出布尔值 TRUE,否则输出 FALSE。

本节介绍一个判断线段集合中任意两条线段是否相交的算法。该算法用到一个叫做“扫描”的技术,该技术对很多计算几何算法都适用。该算法的运行时间是 $O(n\lg n)$,其中 n 是所给的线段数。算法仅判断是否有线段相交,并不输出任何具体的相交情形。

在**扫描**过程中,一条假想的**扫描线**从左向右扫过给定的几何对象集合。扫描线的运动方向是 x 轴,把它视为时间轴。通过将几何对象置于动态数据结构内,并利用对象间的关系,扫描提供了一种对几何对象的排序方法。本节的线段相交算法按从左到右的顺序考虑所有线段的端点并在每遇到一个端点时检测相交性。

为简化描述确定 n 条线段是否有相交情形的算法并证明其正确性,要做两个前提假设。首先,假设输入的线段没有垂直的。其次,假设没有三条线段相交于一点。

5.2.1　算法描述与分析

1. 线段排序

由于假设没有竖直的输入线段，任何输入线段与给定的垂直扫描线至多有一个交点。因此可以按线段与扫描线上的交点的纵坐标对线段排序。

更准确地说，考虑两条线段 s_1 和 s_2。如果在横坐标 x 处的垂直扫描线与两者均相交，则说此两线段在 x 处可比。若 s_1、s_2 在 x 处可比，且 s_1 与扫描线的交点高于 s_2 与扫描线的交点，则称在 x 处 s_1 在 s_2 之上，记为 $s_1 >_x s_2$。例如，在图 5-6(a)中，有关系 $a >_r c$，$a >_t b$，$b >_t c$，$a >_t c$，以及 $b >_u c$。线段 d 与其他线段不可比。

对任意给定的 x，关系“$>_x$”是所有在 x 处与扫描线相交的线段的一个全序关系。在不同的 x 处，线段进入的序和离开的序可能是不同的。当线段的左端点遇到扫描线时进入序，而当右端点遇到扫描线时离开序。

当扫描线通过两条线段的交点时会发生什么呢？如图 5-6(b)所示，它们的位置在全序中是相反的。扫描线 v 和 w 分处于线段 e 和 f 的交点的两边，有 $e >_v f$ 和 $f >_w e$。由于假定了没有三条线段共点，必有某扫描线 x 使得相交线段 e 和 f 在全序 $>_x$ 中是**挨着**的。任何通过图 5-6(b)的阴影区域的扫描线，如 z、e 和 f 在此全序中是紧挨着的。

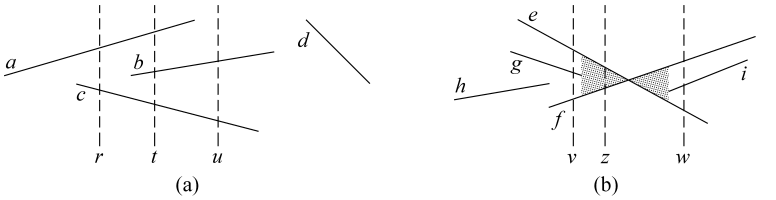

图 5-6　线段关于垂直扫描线的序

图 5-6 所示为线段关于垂直扫描线的序。图 5-6(a)中有 $a >_r c$，$a >_t b$，$b >_t c$，$a >_t c$，以及 $b >_u c$。线段 d 与其他线段不可比。图 5-6(b)中，当线段 e 和 f 相交时，它们的序是颠倒的：有 $e >_v f$ 和 $f >_w e$。任何通过阴影区域的扫描线(如 z)总使得 e 和 f 在全序中是紧挨着的。

2. 移动扫描线

扫描算法通常处理两个数据集合。

(1) **扫描线状态**给出与扫描线相交的对象间的关系。

(2) **事件点进度表**是一组按从左向右顺序横坐标序列，定义了扫描线的暂停位置，称这样的暂停位置为**事件点**。只有在事件点处才发生扫描线状态的改变。

在这个算法的事件点进度表中，每一个线段的端点为一个事件点。将各线段的端点按横坐标递增排序，从左向右逐一处理(若两个以上端点**共竖线**，即它们有共同的 x 坐标，按左端点先于右端点，对所有的共竖线左端点，按小 y 坐标先于大 y 坐标来打破僵局)。当遇到一条线段的左端点时，将其插入到扫描线状态，而当遇到该线段的右端点时，从扫描线状