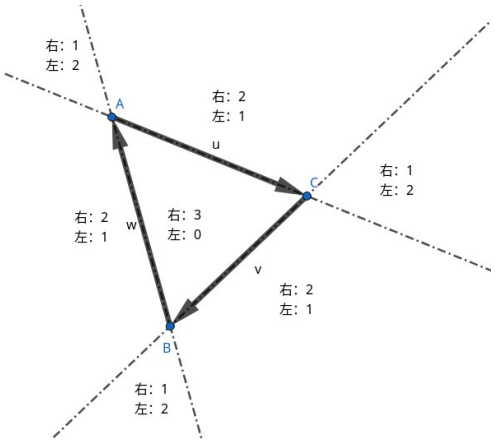
绘制多边形之旅

前言

多边形是常见的图形，可以使用多边形拟合各种各样的图形。但是，在计算机世界里，多边形的内与外并不是先决地存在的。为了在离散的像素世界绘画多边形(光栅化)需要使用一些算法手段。本文将介绍一些简单的判断方法，给大家提供参照。

简单多边形 — 三角形

三角形是边最少的多边形，因而具有一些好的性质。为寻找三角形如何把平面分为内外两部分，从而找到三角形内部的点具有的性质，不妨把三边延长，同时取三顶点依次连接的方向(不妨取顺时针方向)作为直线的方向。注意到，各个直线分别把整个平面分成了以它为界的两个部分，每半个部分都可以用方向直线的左或右描述。

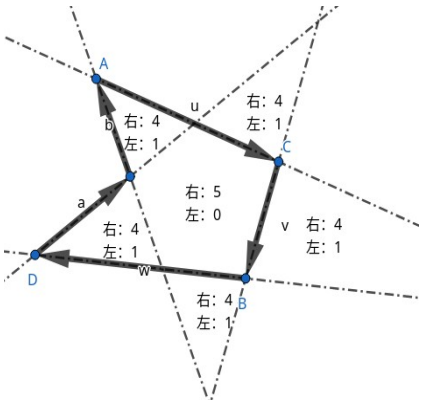


三角形的内部的点总是位于三条边的右侧，而不位于任何一边的左侧。这一性质使得我们可以使用外积(叉积)的正负性计算点是否位于三角形内。

```
if (NumberOfPoints() == 3){
        // 这个是三角形的判断方法，可以看作简化后的多边形判断方法
        bool A = ((point - getPoint(0)) X(getEdge(0)) > 0);
        bool B = ((point - getPoint(1)) X(getEdge(1)) > 0);
        bool C = ((point - getPoint(2)) X(getEdge(2)) > 0);
        return !((A != B) or (B != C));
}
```

c++伪代码(X 表示向量外积)

因为判断点之于三角形的位置如此简便，再加之三角形的许多其他性质，使这个算法十分广为人知。许多图形程序，特别是 3D 程序，都利用这样的三角形算法绘制图形(虽然主要是因为三个点确定一个面)；算法研究者也在寻求将边数大于四的多边形拆分为若干三角形的算法，以简化一些过程。



然而，三边以上的任意多边形并不能相似地简单推广为点位于所有边的同一侧(右图即为反例)。事实上，只有边的延长线不相交于除顶点外的点多边形(凸多边形)才可以推广判断。

然而，直接判断点和多边形的关系并非是不可能的。事实上，多边形矢量图的绘制，也常常使用不化简为多个三角形，而直接判断的方法，接下来，我将介绍一种直接判断点和多边形关系的方法。

一般多边形

假如你是一只在平面上上携带计数器漫游的蚂蚁。对于平面上的一个闭合图形，你若要判断你是否位于多边形内呢？你只需要记住你多少次经过边界就行了！如果你经过了偶数次边界，那么你就位于多边形之外；如果你经过了奇数次边界那么，你位于多边形之内。根据这一想法，有如下的简化简化过程。

如果我们看蚂蚁的轨迹上一点，记录他经过这一点时，所路过的所有点对应经过了奇数次还是偶数次。我们就可以判断路径上这点到底位于平面内还是平面外了。为了简化这个流程，我们令

待判断的点 S=(x_{dest}, y_{dest}).

多边形的节点是有序点集 E =$(P_1, P_2, P_3, \ldots, P_n)$.

从而就有令 E 中每个点的横坐标最小值为x_{min}.

可以让这个蚂蚁从(x_{min}, y_{dest})的位置开始运动，连续的横向运动到 S (x_{dest}, y_{dest})。然后统计到 S 点时蚂蚁经过了几次边界就可以判断 S 点之于多边型的内外了。

P 到 S 连续运动的过程中若干次经过边界

- 对于经过边界的次数:
  当其为奇数,
  - 说明点在多边形内。

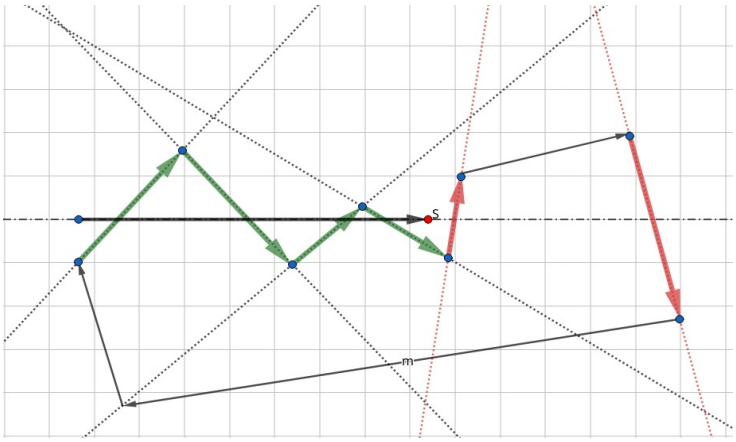
  当其为偶数,
  - 说明点在多边形外。

  判断完毕。

然而为了求解经过边界的次数无需让蚂蚁真正的动起来，我们只是要想知道蚂蚁有没有经过这一点，所以需要知道的是蚂蚁已经经过的路经(静态)经过的边的数量，这个过程可以分为两个步骤。

1. 判断边是否与蚂蚁运动轨迹相交:

   蚂蚁的运动轨迹所在的直线将平面分成两个部分。现顺次遍历这一个多边形的每一个节点。当节点和上一个节点 不同时位于平面被分成的一个部分的时候，这个边就有可能是蚂蚁曾经路过或将要路过的边。



2. 判断与路径相交的边否是曾经蚂蚁路过的:

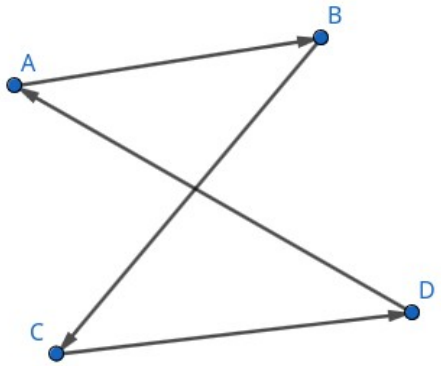
   待检测点和出发点如果位于此边的同一侧，那么蚂蚁未曾经过此边。

```
if (NumberOfPoints() > 3){
    int intersects = 0;
    for (size_t i = 0; i < NumberOfPoints(); i++)
    {
        if ((getPoint(i).y > point.y)
             != ((getPoint(i).y + getPoint(i).y) > point.y)){
            if ((point - getPoint(i)) X getEdge(i) >= 0
                !=( Vector2D( outline.x, point.y)
                   - getPoint(i)) X getEdge(i) > 0){
                intersects++;
            }
        }
    }
    if (intersects % 2 == 1){
        return true;
    }
}
```

c++伪代码(X 表示向量外积)

尾声:

我可能没有组织好语言，很多算法上的内容也许并不是最优解；一些多边形形式、几何证明也没涉及到，比如两个边相交的复杂多边形(如右图所示，虽然经常被认为不是有效的多边形)。但是我希望我的分享能够启发大家思考关于多边形和计算机几何的一些思考，如果有更好的想法或者更严谨的说法，欢迎大家批判指正。



感谢阅读