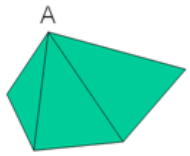
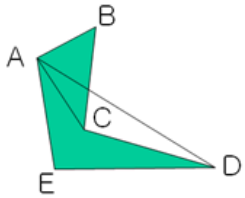
# 计算任意多边形的面积

对于凸多边形，很容易计算，如下图，以多边形的某一点为顶点，将其划分成几个三角形，计算这些三角形的面积，然后加起来即可。已知三角形顶点坐标，三角形面积可以利用向量的叉乘来计算。




对于凹多边形，如果还是按照上述方法划分成三角形，如下图，多边形的面积 = S_ABC + S_ACD + S_ADE，这个面积明显超过多边形的面积。

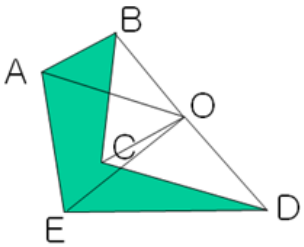



我们根据二维向量叉乘求三角形ABC面积时，利用的是

$$|\overrightarrow{AB} \times \overrightarrow{AC}| = |AB| * |AC| * sin(A)$$
$$S_{ABC} = 0.5 * |AB| * |AC| * sin(A)$$
$$即 S_{ABC} = 0.5 * |\overrightarrow{AB} \times \overrightarrow{AC}|$$

这样求出来的面积都是正数，但是向量叉乘是有方向的，即 $\overrightarrow{AB} \times \overrightarrow{AC}$ 是有正负的，如果把上面第三个公式中的绝对值符号去掉，即 $S_{ABC} = 0.5 * (\overrightarrow{AB} \times \overrightarrow{AC})$ ，那么面积也是有正负的。反应在上面第二个图中，S = S_ABC + S_ACD + S_ADE，如果S_ABC和S_ADE是正的，那么S_ACD是负的，这样加起来刚好就是多边形的面积。对于凸多边形，所有三角形的面积都是同正或者同负。

如果我们不以多边形的某一点为顶点来划分三角形而是以任意一点，如下图，这个方法也是成立的：S = S_OAB + S_OBC + S_OCD + S_ODE + S_OEA。计算的时候，当我们取O点为原点时，可以简化计算。 本文地址




当O点为原点时，根据向量的叉积计算公式，各个三角形的面积计算如下：

S_OAB = 0.5*(A_x*B_y - A_y*B_x) 【(A_x，A_y)为A点的坐标】

S_OBC = 0.5*(B_x*C_y - B_y*C_x)

S_OCD = 0.5*(C_x*D_y - C_y*D_x)

S_ODE = 0.5*(D_x*E_y - D_y*E_x)

S_OEA = 0.5*(E_x*A_y - E_y*A_x)

代码如下

```
struct Point2d
{
    double x;
```

```
    double y;
    Point2d(double xx, double yy): x(xx), y(yy){}
};

//计算任意多边形的面积，顶点按照顺时针或者逆时针方向排列
double ComputePolygonArea(const vector<Point2d> &points)
{
    int point_num = points.size();
    if(point_num < 3)return 0.0;
    double s = 0;
    for(int i = 0; i < point_num; ++i)
        s += points[i].x * points[(i+1)%point_num].y - points[i].y * points[(i+1)%point_num].x;
    return fabs(s/2.0);
}
```

该算法还可以优化一下，对上面的式子合并一下同类项

S = S_OAB + S_OBC + S_OCD + S_ODE + S_OEA =

0.5*(A_y*(E_x-B_x) + B_y*(A_x-C_x) + C_y*(B_x-D_x) + D_y*(C_x-E_x) + E_y*(D_x-A_x))

这样减少了乘法的次数，代码如下：

```
struct Point2d
{
    double x;
    double y;
    Point2d(double xx, double yy): x(xx), y(yy){}
};

//计算任意多边形的面积，顶点按照顺时针或者逆时针方向排列
double ComputePolygonArea(const vector<Point2d> &points)
{
    int point_num = points.size();
    if(point_num < 3)return 0.0;
    double s = points[0].y * (points[point_num-1].x - points[1].x);
    for(int i = 1; i < point_num; ++i)
        s += points[i].y * (points[i-1].x - points[(i+1)%point_num].x);
    return fabs(s/2.0);
}
```