Computational Geometry

Prerequisites

- Graph Theory
- Shortest Path

Tools

This module discusses several algorithms that calculate various geometric properties, mostly based on only two operations described below: cross product and arctangent.

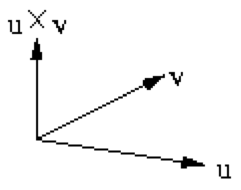
Cross Product

The cross product of *u* and *v* is written as *u x v*. Computationally, the *cross product* of two three-dimensional vectors *u* and *v* is the vector determinant of the following matrix (where **i**, **j**, and **k** are unit vectors in the *x*, *y*, and *z* directions respectively):

$$\begin{vmatrix} i & j & k \\ ux & uy & uz \\ vx & vy & vz \end{vmatrix}$$

That equation works out to:

$(u_y v_z - v_y u_z)\mathbf{i} + (u_z v_x - u_x v_z)\mathbf{j} + (u_x v_y - u_y v_x)\mathbf{k}$

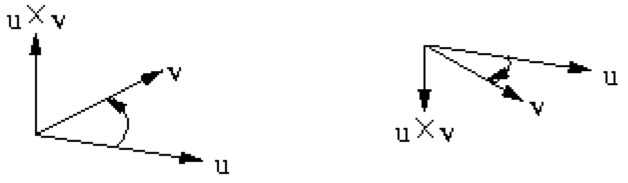


This definition can be used for vectors in two dimensions by using three-dimensional vectors with a z component of 0. The resulting vector will only have a z value.

The cross product has three properties:

- The *cross product* of two vectors is perpendicular to both vectors.
- The length of the cross product is equal to the product of:
    - the length of *u*,
    - the length of *v*, and
    - the sine of the angle between the vectors.

Of the two different directions that are perpendicular to both *u* and *v*, the direction the cross product points depends on whether *u* is ``to the right'' of *v* or ``to the left.''



Dot product

The *dot product* of two vectors *u* and *v* is a scalar written as $u \cdot v$. Computationally, it is defined in three dimensions as: $u_x v_x + u_y v_y + u_z v_z$

The dot product is actually equal to the product of:

- the length of *u*
- the length of *v*
- the cosine of the angle between *u* and *v*.

Presuming *u* and *v* are non-zero, if the dot product if negative, *u* and *v* make an angle greater than 90 degrees. If it is zero, then *u* and *v* are perpendicular. If *u cdot v* is positive, then the two vectors form an acute angle.

Arctangent

The *arctangent*function calculates the (an) angle whose tangent is its argument and generally returns a real number between -pi/2 and pi/2. An additional function in C, *atan2*, takes two arguments: a *DELTA y* value and a *DELTA x* value (in that order!). It determines the angle between the given vector and the positive *x* axis and returns a value between -pi and pi. This has the advantage of removing concerns about dividing by zero or writing code to repair angles in order to handle the negative *x* cases. The *atan2* function is almost always easier to use than the simpler *atan* function that takes only one argument.

Particular Debugging Problems

The main problem with geometric problems is that they spawn **a lot** of special cases. Be on the lookout for these special cases and **make sure your program works for all of them**.

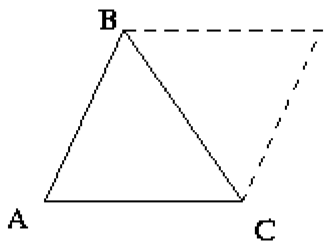
Floating point calculations also create a new set of problems. Floating point calculations are rarely precise, as the computer only maintains so many bits (digits) of accuracy: be aware of this. In particular, when checking if two values are equal, check to see if they are within some small tolerance of each other not precisely equal.

Geometric Algorithms

Here are some of snippets that can help you solve geometry problems.

Area of Triangle

To calculate the area of a triangle with vertices (a, b, c), pick a vertex (say a) and create a vector to the other two vertices (let *u* = b - a, and *v* = c - a). The area of the triangle (a, b, c) is one half the length of cross product *u x v*.



An alternative method to find the area of triangle is to use Hero's formula. If the lengths of the sides of a triangle are *a*, *b*, and *c*, let *s = (a+b+c)/2*. The area of the triangle is then

sqrt(s (s-a)*(s-b)*(s-c))* .

Are Two Line Segments Parallel?

To check if two line segments are parallel, create vectors along each line segment and check to see if their cross product is (almost) zero.

Area of polygon

The area of a polygon with vertices (x_1, y_1), ..., (x_n, y_n) is equal to the determinant:

$$\frac{1}{2}\begin{vmatrix} x1 & x2 & ... & xn \\ & & & \\ y1 & y2 & ... & yn \end{vmatrix}$$

where the determinate is defined to be similar to the 2 by 2 determinant: $x_1 y_2 + x_2y_3 + ... + x_n y_1 - y_1 x_2 - y_2x_3 - ... - y_n x_1$

Distance from a point to a line

The distance from a point P to a line AB is given by the magnitude of the cross product. In particular, *d*(P,AB) = |(P - A) x (B - A)| / | B - A| .

To determine the distance from a point P to the plane defined by A, B, and C, let *n* = (B - A) x (C - A). The distance is then give by the following equation: *d*(P,ABC) = (P-A) · n / |n|.

Points on a line

A point is on a line if the distance from the point to the line is 0.

Points on the same side of line

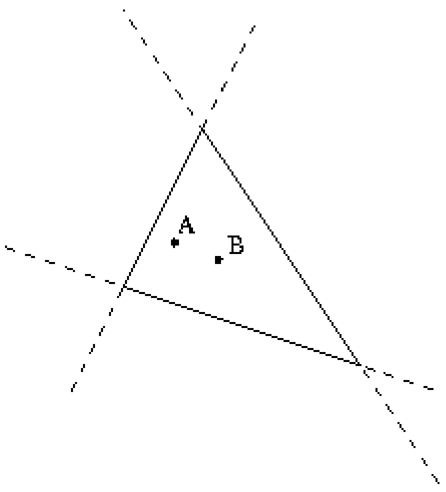
This notion only makes sense for two dimensions. To check if points C and D are on the same side of line AB, calculate the z component of *(B - A) x (C - A)* and *(B - A) x (D - A)*. If the *z* components have the same sign (i.e., their product is positive), then C and D are on the same side of the line AB.

Point on line segment

To calculate if a point C is on the line segment AB, check if C is on the line AB. If it is, then check if the length of AB is equal to the sum of the lengths of AC and CB.

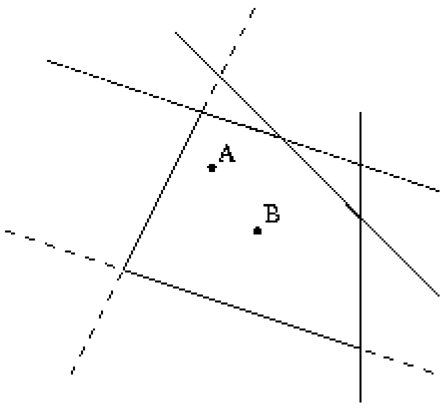
Point in triangle

To check if a point A is in a triangle, find another point B which is within the triangle (the average of the three vertices works well). Then, check if the point A is on the same side of the three lines defined by the edges of the triangle as B.



Point in convex polygon

The same trick works for a convex polygon:



Four (or more) points are coplanar

To determine if a collection of points is coplanar, select three points, A, B, and C. Now, if, for any other point D, (B - A) x (C - A)) · (D - A) = ~0, then the collection of points resides in some plane.

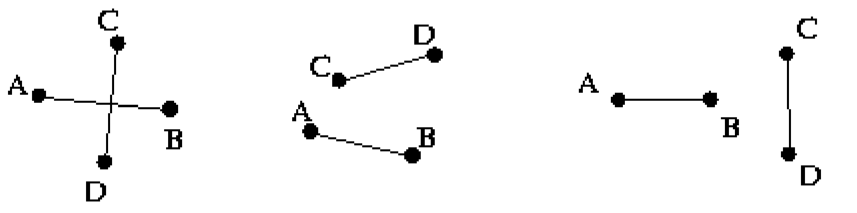
Two lines intersect

Two lines intersect if and only if they are not parallel in two dimensions.

In three dimensions, two lines AB and CD intersect if they are not parallel and A, B, C, and D are coplanar.

Two line segments intersect

In two dimensions, two line segments AB and CD intersect if and only if A and B are on opposite sides of the line CD and C and D are on opposite sides of line AB.



Note that both of the checks are necessary, as for the last case one of the checks returns true, while the other testifies to the fact that AB and CD do not intersect. In three dimensions, solve following system of equations, where *i* and *j* are the unknowns:

$$A_x + (B_x - A_x)\ i = C_x + (D_x - C_x)\ j$$
$$A_y + (B_y - A_y)\ i = C_y + (D_y - C_y)\ j$$
$$A_z + (B_z - A_z)\ i = C_z + (D_z - C_z)\ j$$

If this system has a solution (i, j), where $0 <= i <= 1$ and $0 <= j <= 1$, then the line segments intersect at: ($A_x + (B_x - A_x)i$, $A_y + (B_y - A_y)i$, $A_z + (B_z - A_z)\ i$.

Point of Intersection of Two Lines

For the lines AB and CD in two dimensions, the most straight-forward way to calculate the intersection of them is to solve the system of two equations and two unknowns:

$$A_x + (B_x - A_x)i = C_x + (D_x - C_x)\ j$$
$$A_y + (B_y - A_y)i = C_y + (D_y - C_y)\ j$$

The point of intersection is:

$$(A_x + (B_x - A_x)\ i,\ A_y + (B_y - A_y)\ i)$$

In three dimensions, solve the same system of equations as was used to check line intersection, and the point of intersection is:

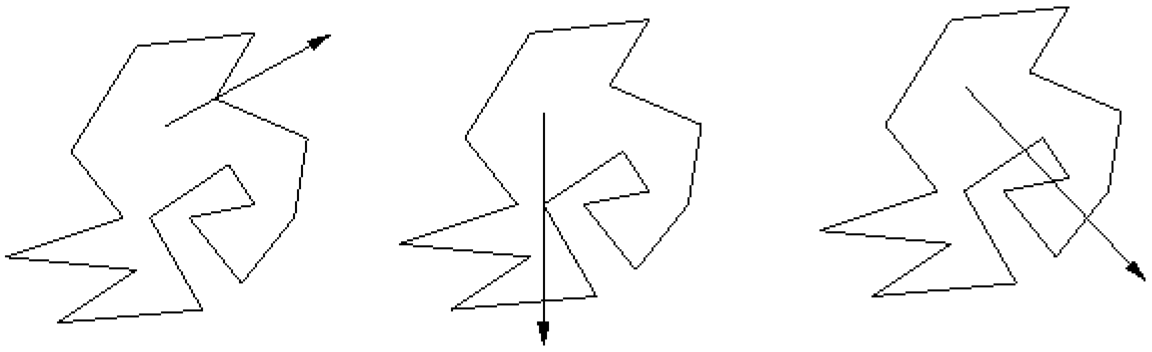
$$(A_x + (B_x - A_x)i,\ A_y + (B_y - A_y)i,\ A_z + (B_z - A_z)i)$$

Checking convexity of 2-dimensional polygon

To check the convexity of a 2-dimensional polygon, walk the polygon in clock-wise order. For each triplet of consecutive points (A, B, C), calculate the cross product (B - A) x (C - A). If the *z* component of each of these vectors is positive, the polygon is convex.

Point in non-convex polygon

To calculate if a point is within a nonconvex polygon, make a ray from that point in a random direction and count the number of times it intersects the polygon. If the ray intersects the polygon at a vertex or along an edge, pick a new direction. Otherwise, the point is within the polygon if and only if the ray intersects the polygon an odd number of times.



This method also extends to three dimensions (and higher), but the restriction on intersection is that it only intersects at faces and not at either a vertex or an edge.

Geometry Methodologies

Geometric problems introduce several different tricks that can be used to either reduce the run-time or approximate the solution.

Monte Carlo

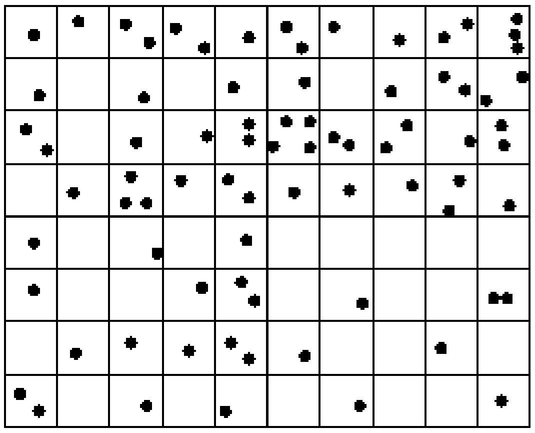
The first geometric trick is based on randomness. Instead of calculating the probability that something occurs, simulate a random event and calculate the fraction of times it occurs. If enough events are simulated, the difference between these two values becomes very small.

This can be helpful to determine something like the area of a figure. Instead of calculating the area directly, determine a bounding box, and throw ``darts'' at the box, and estimate what the probability of hitting the figure is. If this is calculated accurately enough, this can give a good estimate of the actual area.

The problem with this method is to get a good relative error (error divided by the actual value) requires a large number of successful events. If the probability of the event occurring is very small, the method does not yield good results.

Partitioning

Partitioning is a method to improve the speed of a geometric algorithm. This entails dividing the plane up into sections (usually by a grid but sometimes into radial sections or some other method), and bucketing the objects into appropriate section(s). When looking for objects within some figure, only those sections which have a non-zero intersection with that figure need to be examined, thereby greatly reducing the cost of the algorithm. This is helpful to determine the set of objects within some distance of a given point (the figure is a circle) or to check for intersections (the figure is a line).



Graph Problems

Sometimes what may look like a geometric problem is really a graph problem. Just because the input is points in the plane does not mean it's a geometric algorithm.

Example Problems

Point Moving

Given a set of line segments in the plane, and two points A and B, is it possible to move from A to B without crossing any of the segments?

The line segments partition the plane into regions. Determine these regions, and see if A and B reside in the same region.

Bicycle Routing

Given a collection of non-intersecting buildings along with start and end locations, find the shortest path from A to B that doesn't go through any buildings.

Analysis: This is really a graph problem. The nodes are the start and end locations, along with the vertices of the buildings. There are edges between any two nodes such that the line segment between them does not intersect any buildings, with weight equal to the length of the length of the line segments. Once that graph has been calculated, the problem is shortest path.

Maximizing Line Intersections

Given a collection of segments in the plane, find the greatest number of segments which can by intersected by drawing a single line.

Analysis: With a little bit of thought, it is clear that the line segment must pass through two of the vertices of the collection of line segments. Thus, try all pairs of vertices, and calculate the crossing for each. Combining this with partitioning gives an algorithm that runs fairly quickly.

Polygon Classification

Given a collection of segments defining a polygon, determine if it is simple (no two non-consecutive line segments intersect) and convex.

Computational Geometry

计算几何

译 By SuperBrother

目录

知识准备

图论

最短路

操作

这个模块讨论几个计算某些几何问题的算法，这些算法大多基于下列两个操作：叉积和反正切。

叉积

u 和 v 的叉积被表示成 u x v。两个**三维的向量** u,v 的叉积是下列行列式（**i,j,k 为 x,y,z 轴上单位向量**）:

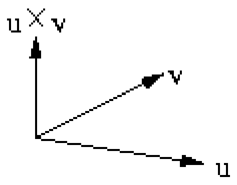
| i j k |

| Ux Uy Uz |

| Vx Vy Vz |

即:

(Uy*Vz-Vy*Uz)i + (Uz*Vx-Ux*Vz)j + (Ux*Vy-Uy*Vx)k



z 分量为 0 时，上面式子就化为 2 维的两向量叉积了。结果只有 z 分量。

即：

| Ux Uy |

| Vx Vy |

Ux*Vy-Uy*Vx

（注意！二维的叉积较为猥琐。。其实叉积最早就是定义在三维空间内的，当 z=0 时的特殊情形才是二维向量积。“二维向量”表达式求出的是一个超出平面的向量。。这个向量的方向我们不关心，但**有时又要利用它**。。。（例如求多边形面积）故上式不甚严格）

叉积有 3 个特点：

两个向量的叉积是一个与这两个向量同时垂直的向量。

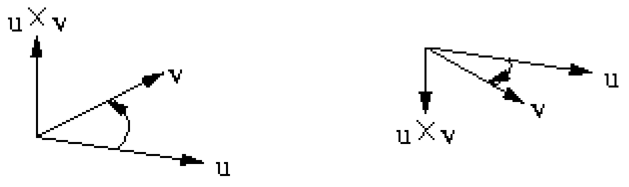
叉积的大小等于下面 3 个量的乘积:

u 的大小

v 的大小

u,v 夹角的正弦。

当然与 u,v 同时垂直的向量有两个方向，叉积的方向取决于 u 在 v 的右边还是在 v 的左边。



点积

两个向量 u,v 的点积是一个标量，用 u · v 表示。在三维空间中它被定义为：uxvx + uyvy + uzvz。

点积的值由以下三个值确定：

u 的大小

v 的大小

u,v 夹角的余弦。

在 u,v 非零的前提下，点积如果为负，则 u,v 形成的角大于 90 度；如果为零，那么 u,v 垂直；如果为正，那么 u,v 形成的角为锐角。

反正切

反正切函数对于一个给定的正切值，返回一个在-pi/2 到 pi/2 之间的角（即-90 度至+90 度）。C 中另外有一个函数 atan2，给出 y 分量和 x 分量（注意顺序！），计算向量与 x 正半轴的夹角，在-pi 到 pi 之间。它的优点就是不需担心被 0 除，也不需为了处理 x 为负的情况而写代码修改角。atan2 函数几乎比普通的 atan 函数更简单，因为只需调用一次。

全面考虑问题

这些几何问题大多都产生**很多**特殊情况。注意这些特殊情况并且要**保证自己的程序能处理所有的情况**。

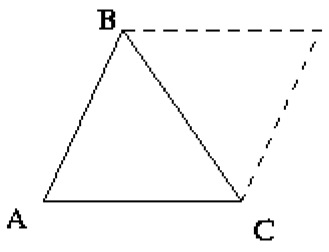
浮点运算也会带来新问题。浮点运算很难精确，因为注意：计算机只能计算一定精度。特别地，当判断两个值是否相等时，要判断两个值的差在一个很小的范围内，而不是直接相等。

一些几何算法

这里是一些能帮助你计算几何问题的东西。

三角形面积

为了计算由点(A,B,C)构成的三角形的面积，先选取一个顶点（例如 A），再向剩余两个顶点作向量（令 u=b-a, v=c-a）。三角形(A,B,C)的面积即为 u,v 叉积长度的一半。



另一个求三角形面积的变通方法就是用海伦公式。如果三角形三边长为 a,b,c，令 s=(a+b+c)/2，那么三角形面积就是:

```
sqrt(s*(s-a)*(s-b)*(s-c))
```

两条线段平行吗？

为了判断两条线段是否平行，分别沿两条线段建立向量，判断叉积是否（几乎为）零。

多边形面积

由点(x1,y1)...(xn,yn)组成的多边形的面积等于下列行列式的值：

```
1    | x1  x2 . . .  xn |
---   |                 |
 2    | y1  y2 . . .  yn |
```

也等于下面的式子的值:

x1y2 + x2y3 + ... + xny1 - y1x2 - y2x3 - ... - ynx1

就是选取原点作标准 连接原点和个顶点 并且两两个地计算叉积并加起来

点到直线的距离

点 P 到直线 AB 的距离也可以由叉积给出，准确的说,d(P,AB) = |(P - A) x (B - A)| / | B - A| 。

为了点 P 到由点 A,B 和 C 确定的平面的距离，令 n = (B - A) x (C - A)，那么 d(P,ABC) = (P-A) · n / |n|。

点在直线上

如果点到直线的距离是 0，那么点在直线上。

点都在直线的同侧

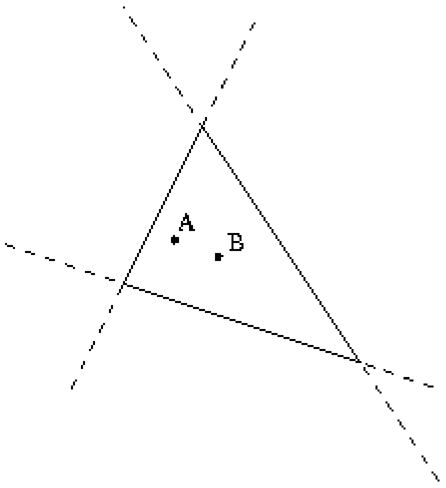
只讲两个点的情况。如果要确定点 C 和 D 是否在直线 AB 同侧，计算(B - A) x (C - A)和(B - A) x (D - A)的 z 分量。如果同号（或如果积为正），那么点 C,D 在直线 AB 同侧。

点在线段上

为了求出点 C 是否在线段 AB 上，先判断点 A 是否在直线 AB 上，再判断线段 AB 的长度是否等于线段 AC 长度与线段 BC 长度之和。

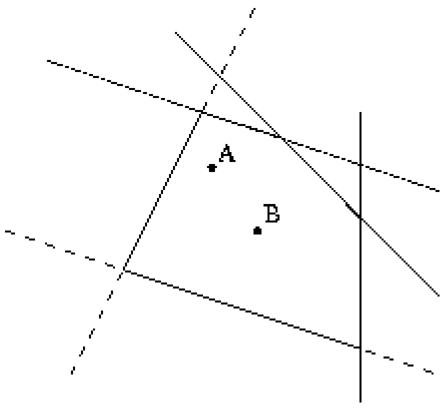
点在三角形内

要确定点 A 是否在三角形内，首先选择一个三角形内部的点 B(重心就很不错)。接下来，判断点 A,B 是否都在三边所在的三条直线的同侧。



点在凸多边形内

方法同上



四点（或更多）共面

如果要确定一组点是否共面，任选 3 个点。如果对于任意点 D，有(B - A) x (C - A)) · (D - A) = ~0，那么这些点共面。

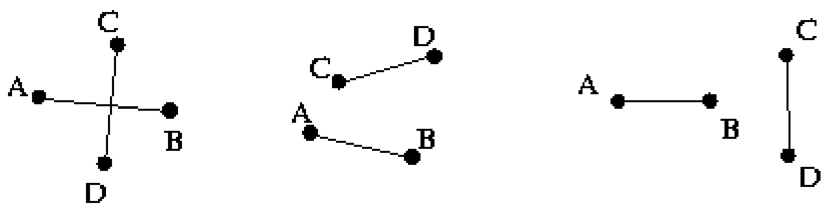
两条直线相交

平面内两条直线相交当且仅当直线不平行。

空间内，两直线 AB,CD 相交则 AB,CD 不平行，且点 A,B,C,D 共面。

两条线段相交

平面内,两条线段相交当且仅当 A,B 在线段 CD 异侧且 C,D 在线段 AB 异侧。



注意两个判断都是必须的，例如第三种情况第一个判断为 true，但第二个判断说明线段 AB,CD 不相交。在空间中，计算下面方程组，其中 i,j 未知：

Ax + (Bx - Ax) i = Cx + (Dx - Cx) j

Ay + (By - Ay) i = Cy + (Dy - Cy) j

Az + (Bz - Az) i = Cz + (Dz - Cz) j

如果方程组有解(i,j)满足 0<=i<=1，0<=j<=1，那么两线段相交于点(Ax + (Bx - Ax)i, Ay + (By - Ay)i, Az + (Bz - Az) i)

两直线的交点

在平面内的两条直线 AB,CD，求交点最直接的方法就是解下列的二元二次方程组：

Ax + (Bx - Ax)i = Cx + (Dx - Cx) j

Ay + (By - Ay)i = Cy + (Dy - Cy) j

交点是:

(Ax + (Bx - Ax) i, Ay + (By - Ay) i)

空间内，解同样的方程组来判断交点，交点是：

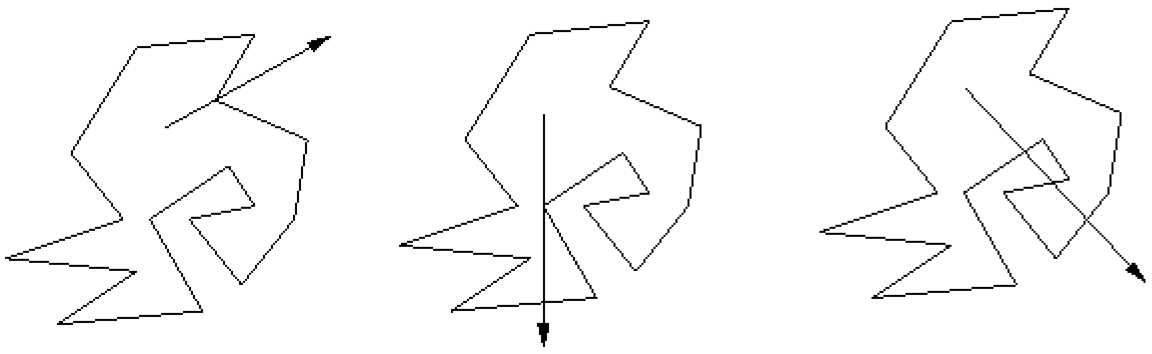
(Ax + (Bx - Ax)i, Ay + (By - Ay)i, Az + (Bz - Az)i)

判断平面内多边形的凹凸性

要判断平面内一多边形是否为凸，沿着顺时针方向扫一遍。对于每三个点(A,B,C)，计算叉积(B - A) x (C - A)。如果叉积的 z 分量均为负，多边形则为凸多边形。

点在凹多边形内

要确定点是否在凹多边形内，任选一点作射线，计算相交次数。如果与多边形相交于一点或一边，则换一个方向，否则，点在多边形内当且仅当射线与点相交次数为奇数。



这个方法也可以扩展到三位（或更高），但此时应相交于面（再判断），不是在点上或边上。

几何方法

这些几何方法介绍了一些技巧，可以用来优化时间和求得更精确的解。

蒙特卡洛方法(Monte Carlo)

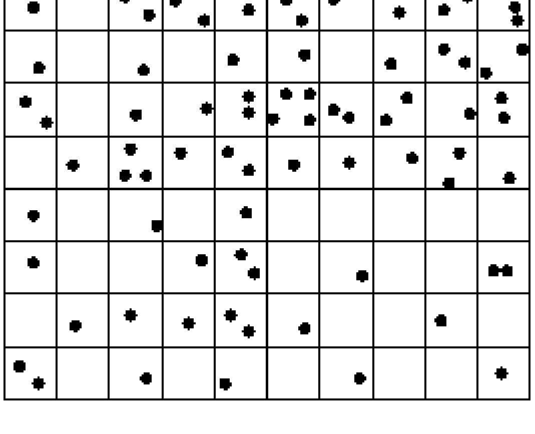
第一个方法是建立在随机化之上的。它不去直接计算某件事的概率，而是以一个随机事件来模拟，求得事件发生的频率。如果次数足够，频率和概率的差别会很小。

这对于确定某些东西来说是很有用的，例如计算某个图形的面积。它不去直接计算面积，而是建立一个已知大小的区域，每次向该区域扔一个“飞镖”，并计算击中区域内图形的频率。如果计算足够精确，就可以得到实际面积的一个足够好的近似值。

这个方法要得到一个足够小的相对误差（误差于实际值之商），需要大量成功发生的事件。如果发生的概率很小，结果就不好了。

分割技术

分割技术可用来优化时间。这需要把平面分割成小块（通常是分成小格，但有时也会用角度或其它方法），并将元素放入合适的区域中。当检查图案的某部分时，只有有该图案的格子会被检查到。所以，时间可以大大地优化，比如，要确定到定点的距离小于某个值的元素（此时图案是个圆），或求是否相交（此时图案为直线）。



转化为图

有时看起来像几何问题的问题实际上却是一个图的问题。因为输入是平面内的点，并不代表问题是几何问题。

例子

Point Moving

给定一组线段和点 A,B，能否在不穿过线段的情况下从 A 移动到 B?

线段将平面分割成不同部分,检查 A,B 是否在一个部分。

Bicycle Routing

给出有起点和终点的一组建筑物，找出从建筑物 A 到 B 不穿过任何建筑物的最短路线。

题解：图论问题。以建筑物的起点和重点为点，两点之间在不径直穿过任何建筑物时连一条边，边权为两点之间的距离。这样就把原问题转化成了最短路问题。

Maximizing Line Intersections

给出平面内一组点，找到能被一条直线能相交到的最多线段。

题解：很显然，直线必须经过两个交点。这样，枚举每对交点，计算此时直线的交点数。可用分割法优化。

Polygon Classification

给出一组直线所确定的多边形，判断多边形是否是简单的（没有任意两条不连续的线段相交）和凸的.