[CP-Algorithms](https://cp-algorithms.com/)[Page Authors](https://github.com/e-maxx-eng/e-maxx-eng/commits/master/src/geometry/basic-geometry.md)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | |  |  | | |  | |

**Basic Geometry**

**Table of Contents**

[Linear operations](https://cp-algorithms.com/geometry/basic-geometry.html#toc-tgt-0)

[Dot product](https://cp-algorithms.com/geometry/basic-geometry.html#toc-tgt-1)

[Definition](https://cp-algorithms.com/geometry/basic-geometry.html#toc-tgt-2)

[Properties](https://cp-algorithms.com/geometry/basic-geometry.html#toc-tgt-3)

[Cross product](https://cp-algorithms.com/geometry/basic-geometry.html#toc-tgt-4)

[Definition](https://cp-algorithms.com/geometry/basic-geometry.html#toc-tgt-5)

[Properties](https://cp-algorithms.com/geometry/basic-geometry.html#toc-tgt-6)

[Exercises](https://cp-algorithms.com/geometry/basic-geometry.html#toc-tgt-7)

[Line intersection](https://cp-algorithms.com/geometry/basic-geometry.html#toc-tgt-8)

[Planes intersection](https://cp-algorithms.com/geometry/basic-geometry.html#toc-tgt-9)

In this article we will consider basic operations on points in Euclidean space which maintains the foundation of the whole analytical geometry. We will consider for each point rr the vector r⃗ r→ directed from 00 to rr. Later we will not distinguish between rr and r⃗ r→ and use the term **point** as a synonym for **vector**.

**Linear operations**

Both 2D and 3D points maintain linear space, which means that for them sum of points and multiplication of point by some number are defined. Here are those basic implementations for 2D:

**struct point2d {**

**ftype x, y;**

**point2d() {}**

**point2d(ftype x, ftype y): x(x), y(y) {}**

**point2d operator+=(const point2d &t) {**

**x += t.x;**

**y += t.y;**

**return \*this;**

**}**

**point2d operator-=(const point2d &t) {**

**x -= t.x;**

**y -= t.y;**

**return \*this;**

**}**

**point2d operator\*=(ftype t) {**

**x \*= t;**

**y \*= t;**

**return \*this;**

**}**

**point2d operator+(const point2d &t) const {**

**return point2d(\*this) += t;**

**}**

**point2d operator-(const point2d &t) const {**

**return point2d(\*this) -= t;**

**}**

**point2d operator\*(ftype t) const {**

**return point2d(\*this) \*= t;**

**}**

**};**

**point2d operator\*(ftype a, point2d b) {**

**return b \* a;**

**}**

And 3D points:

**struct point3d {**

**ftype x, y, z;**

**point3d() {}**

**point3d(ftype x, ftype y, ftype z): x(x), y(y), z(z) {}**

**point3d operator+=(const point3d &t) {**

**x += t.x;**

**y += t.y;**

**z += t.z;**

**return \*this;**

**}**

**point3d operator-=(const point3d &t) {**

**x -= t.x;**

**y -= t.y;**

**z -= t.z;**

**return \*this;**

**}**

**point3d operator\*=(ftype t) {**

**x \*= t;**

**y \*= t;**

**z \*= t;**

**return \*this;**

**}**

**point3d operator+(const point3d &t) const {**

**return point3d(\*this) += t;**

**}**

**point3d operator-(const point3d &t) const {**

**return point3d(\*this) -= t;**

**}**

**point3d operator\*(ftype t) const {**

**return point3d(\*this) \*= t;**

**}**

**};**

**point3d operator\*(ftype a, point3d b) {**

**return b \* a;**

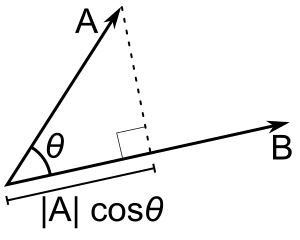
**}**

Here **ftype** is some type used for coordinates, usually **int**, **double** or **long long**.

**Dot product**

**Definition**

The dot (or scalar) product a⋅ba⋅b for vectors aa and bb can be defined in two identical ways. Geometrically it is product of the length of the first vector by the length of the projection of the second vector onto the first one. As you may see from the image below this projection is nothing but |a|cosθ|a|cos⁡θ where θθ is the angle between aa and bb. Thus a⋅b=|a|cosθ⋅|b|a⋅b=|a|cos⁡θ⋅|b|.



The dot product holds some notable properties:

1. a⋅b=b⋅aa⋅b=b⋅a
2. (α⋅a)⋅b=α⋅(a⋅b)(α⋅a)⋅b=α⋅(a⋅b)
3. (a+b)⋅c=a⋅c+b⋅c(a+b)⋅c=a⋅c+b⋅c

I.e. it is a commutative function which is linear with respect to both arguments. Let's denote the unit vectors as

ex=⎛⎝⎜100⎞⎠⎟,ey=⎛⎝⎜010⎞⎠⎟,ez=⎛⎝⎜001⎞⎠⎟.ex=(100),ey=(010),ez=(001).

With this notation we can write the vector r=(x;y;z)r=(x;y;z) as r=x⋅ex+y⋅ey+z⋅ezr=x⋅ex+y⋅ey+z⋅ez. And since for unit vectors

ex⋅ex=ey⋅ey=ez⋅ez=1,ex⋅ey=ey⋅ez=ez⋅ex=0ex⋅ex=ey⋅ey=ez⋅ez=1,ex⋅ey=ey⋅ez=ez⋅ex=0

we can see that in terms of coordinates for a=(x1;y1;z1)a=(x1;y1;z1) and b=(x2;y2;z2)b=(x2;y2;z2) holds

a⋅b=(x1⋅ex+y1⋅ey+z1⋅ez)⋅(x2⋅ex+y2⋅ey+z2⋅ez)=x1x2+y1y2+z1z2a⋅b=(x1⋅ex+y1⋅ey+z1⋅ez)⋅(x2⋅ex+y2⋅ey+z2⋅ez)=x1x2+y1y2+z1z2

That is also the algebraic definition of the dot product. From this we can write functions which calculate it.

**ftype dot(point2d a, point2d b) {**

**return a.x \* b.x + a.y \* b.y;**

**}**

**ftype dot(point3d a, point3d b) {**

**return a.x \* b.x + a.y \* b.y + a.z \* b.z;**

**}**

When solving problems one should use algebraic definition to calculate dot products, but keep in mind geometric definition and properties to use it.

**Properties**

We can define many geometrical properties via the dot product. For example

1. Norm of aa (squared length): |a|2=a⋅a|a|2=a⋅a
2. Length of aa: |a|=a⋅a−−−−√|a|=a⋅a
3. Projection of aa onto bb: a⋅b|b|a⋅b|b|
4. Angle between vectors: arccos(a⋅b|a|⋅|b|)arccos⁡(a⋅b|a|⋅|b|)
5. From the previous point we may see that the dot product is positive if the angle between them is acute, negative if it is obtuse and it equals zero if they are orthogonal, i.e. they form a right angle.

Note that all these functions do not depend on the number of dimensions, hence they will be the same for the 2D and 3D case:

**ftype norm(point2d a) {**

**return dot(a, a);**

**}**

**double abs(point2d a) {**

**return sqrt(norm(a));**

**}**

**double proj(point2d a, point2d b) {**

**return dot(a, b) / abs(b);**

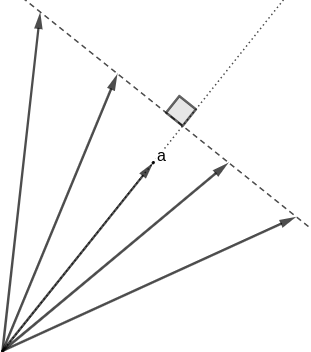
**}**

**double angle(point2d a, point2d b) {**

**return acos(dot(a, b) / abs(a) / abs(b));**

**}**

To see the next important property we should take a look at the set of points rr for which r⋅a=Cr⋅a=C for some fixed constant CC. You can see that this set of points is exactly the set of points for which the projection onto aa is the point C⋅a|a|C⋅a|a| and they form a hyperplane orthogonal to aa. You can see the vector aa alongside with several such vectors having same dot product with it in 2D on the picture below:

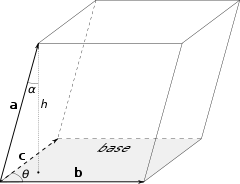


In 3D such vectors will rather form a plane than a line. Note that this result allows us to define a line in 2D as r⋅n=Cr⋅n=C or (r−r0)⋅n=0(r−r0)⋅n=0 where nn is vector orthogonal to the line and r0r0 is any vector already present on the line and C=r0⋅nC=r0⋅n. In the same manner a plane can be defined in 3D.

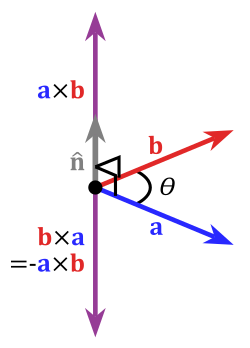
**Cross product**

**Definition**

Assume you have three vectors aa, bb and cc in 3D space joined in a parallelepiped as in the picture below:



How would you calculate its volume? From school we know that we should multiply the area of the base with the height, which is projection of aa onto direction orthogonal to base. That means that if we define b×cb×c as the vector which is orthogonal to both bb and ccand which length is equal to the area of the parallelogram formed by bb and cc then |a⋅(b×c)||a⋅(b×c)| will be equal to the volume of the parallelepiped. For integrity we will say that b×cb×c will be always directed in such way that the rotation from the vector bb to the vector ccfrom the point of b×cb×c is always counter-clockwise (see the picture below).



This defines the cross (or vector) product b×cb×c of the vectors bb and cc and the triple product a⋅(b×c)a⋅(b×c) of the vectors aa, bb and cc.

Some notable properties of cross and triple products:

1. a×b=−b×aa×b=−b×a
2. (α⋅a)×b=α⋅(a×b)(α⋅a)×b=α⋅(a×b)
3. For any bb and cc there is exactly one vector rr such that a⋅(b×c)=a⋅ra⋅(b×c)=a⋅r for any vector aa.   
   Indeed if there are two such vectors r1r1 and r2r2 then a⋅(r1−r2)=0a⋅(r1−r2)=0 for all vectors aa which is possible only when r1=r2r1=r2.
4. a⋅(b×c)=b⋅(c×a)=−a⋅(c×b)a⋅(b×c)=b⋅(c×a)=−a⋅(c×b)
5. (a+b)×c=a×c+b×c(a+b)×c=a×c+b×c. Indeed for all vectors rr the chain of equations holds:

r⋅((a+b)×c)=(a+b)⋅(c×r)=a⋅(c×r)+b⋅(c×r)=r⋅(a×c)+r⋅(b×c)=r⋅(a×c+b×c)r⋅((a+b)×c)=(a+b)⋅(c×r)=a⋅(c×r)+b⋅(c×r)=r⋅(a×c)+r⋅(b×c)=r⋅(a×c+b×c)

Which proves (a+b)×c=a×c+b×c(a+b)×c=a×c+b×c due to point 3.

1. |a×b|=|a|⋅|b|sinθ|a×b|=|a|⋅|b|sin⁡θ where θθ is angle between aa and bb, since |a×b||a×b| equals to the area of the parallelogram formed by aaand bb.

Given all this and that the following equation holds for the unit vectors

ex×ex=ey×ey=ez×ez=0,ex×ey=ez, ey×ez=ex, ez×ex=eyex×ex=ey×ey=ez×ez=0,ex×ey=ez, ey×ez=ex, ez×ex=ey

we can calculate the cross product of a=(x1;y1;z1)a=(x1;y1;z1) and b=(x2;y2;z2)b=(x2;y2;z2) in coordinate form:

a×b=(x1⋅ex+y1⋅ey+z1⋅ez)×(x2⋅ex+y2⋅ey+z2⋅ez)=a×b=(x1⋅ex+y1⋅ey+z1⋅ez)×(x2⋅ex+y2⋅ey+z2⋅ez)=

(y1z2−z1y2)ex+(z1x2−x1z2)ey+(x1y2−y1x2)(y1z2−z1y2)ex+(z1x2−x1z2)ey+(x1y2−y1x2)

Which also can be written in the more elegant form:

a×b=∣∣∣∣∣exx1x2eyy1y2ezz1z2∣∣∣∣∣, a⋅(b×c)=∣∣∣∣x1x2x3y1y2y3z1z2z3∣∣∣∣a×b=|exeyezx1y1z1x2y2z2|, a⋅(b×c)=|x1y1z1x2y2z2x3y3z3|

Here |⋅||⋅| stands for the determinant of a matrix.

Some kind of cross product (namely the pseudo-scalar product) can also be implemented in the 2D case. If we would like to calculate the area of parallelogram formed by vectors aa and bb we would compute |ez⋅(a×b)|=|x1y2−y1x2||ez⋅(a×b)|=|x1y2−y1x2|. Another way to obtain the same result is to multiply |a||a| (base of parallelogram) with the height, which is the projection of vector bb onto vector aa rotated by 90∘90∘which in turn is aˆ=(−y1;x1)a^=(−y1;x1). That is, to calculate |aˆ⋅b|=|x1y2−y1x2||a^⋅b|=|x1y2−y1x2|.

If we will take the sign into consideration then the area will be positive if the rotation from aa to bb (i.e. from the view of the point of ezez) is performed counter-clockwise and negative otherwise. That defines the pseudo-scalar product. Note that it also equals |a|⋅|b|sinθ|a|⋅|b|sin⁡θwhere θθ is angle from aa to bb count counter-clockwise (and negative if rotation is clockwise).

Let's implement all this stuff!

**point3d cross(point3d a, point3d b) {**

**return point3d(a.y \* b.z - a.z \* b.y,**

**a.z \* b.x - a.x \* b.z,**

**a.x \* b.y - a.y \* b.x);**

**}**

**ftype triple(point3d a, point3d b, point3d c) {**

**return dot(a, cross(b, c));**

**}**

**ftype cross(point2d a, point2d b) {**

**return a.x \* b.y - a.y \* b.x;**

**}**

**Properties**

As for the cross product, it equals to the zero vector iff the vectors aa and bb are collinear. The same thing holds for the triple product, it is equal to zero iff the vectors aa, bb and cc are coplanar.

From this we can obtain universal equations defining lines and planes. A line can be defined via its direction vector dd and an initial point r0r0 or by two points aa and bb. It is defined as (r−r0)×d=0(r−r0)×d=0 or as (r−a)×(b−a)=0(r−a)×(b−a)=0. As for planes, it can be defined by three points aa, bb and cc as (r−a)⋅((b−a)×(c−a))=0(r−a)⋅((b−a)×(c−a))=0 or by initial point r0r0 and two direction vectors lying in this plane d1d1and d2d2: (r−r0)⋅(d1×d2)=0(r−r0)⋅(d1×d2)=0.

In 2D the pseudo-scalar product also may be used to check the orientation between two vectors because it is positive if the rotation from the first to the second vector is clockwise and negative otherwise. And, of course, it can be used to calculate areas of polygons, which is described in a different article. A triple product can be used for the same purpose in 3D space.

**Exercises**

**Line intersection**

There are many possible ways to define a line in 2D and you shouldn't hesitate to combine them. For example we have two lines and we want to find their intersection points. We can say that all points from first line can be parameterized as r=a1+t⋅d1r=a1+t⋅d1 where a1a1 is initial point, d1d1 is direction and tt is some real parameter. As for second line all its points must satisfy (r−a2)×d2=0(r−a2)×d2=0. From this we can easily find parameter tt:

(a1+t⋅d1−a2)×d2=0⇒t=(a2−a1)×d2d1×d2(a1+t⋅d1−a2)×d2=0⇒t=(a2−a1)×d2d1×d2

Let's implement function to intersect two lines.

**point2d intersect(point2d a1, point2d d1, point2d a2, point2d d2) {**

**return a1 + cross(a2 - a1, d2) / cross(d1, d2) \* d1;**

**}**

**Planes intersection**

However sometimes it might be hard to use some geometric insights. For example, you're given three planes defined by initial points aiaiand directions didi and you want to find their intersection point. You may note that you just have to solve the system of equations:

⎧⎩⎨r⋅n1=a1⋅n1,r⋅n2=a2⋅n2,r⋅n3=a3⋅n3{r⋅n1=a1⋅n1,r⋅n2=a2⋅n2,r⋅n3=a3⋅n3

Instead of thinking on geometric approach, you can work out an algebraic one which can be obtained immediately. For example, given that you already implemented a point class, it will be easy for you to solve this system using Cramer's rule because the triple product is simply the determinant of the matrix obtained from the vectors being its columns:

**point3d intersect(point3d a1, point3d n1, point3d a2, point3d n2, point3d a3, point3d n3) {**

**point3d x(n1.x, n2.x, n3.x);**

**point3d y(n1.y, n2.y, n3.y);**

**point3d z(n1.z, n2.z, n3.z);**

**point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));**

**return point3d(triple(d, y, z),**

**triple(x, d, z),**

**triple(x, y, d)) / triple(n1, n2, n3);**

**}**

Now you may try to find out approaches for common geometric operations yourself to get used to all this stuff.

(c) 2014-2018 translation by <http://github.com/e-maxx-eng>