

# Cross product

From Wikipedia, the free encyclopedia

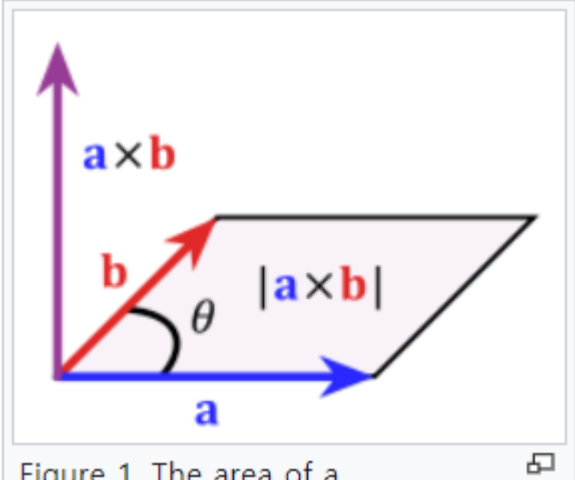
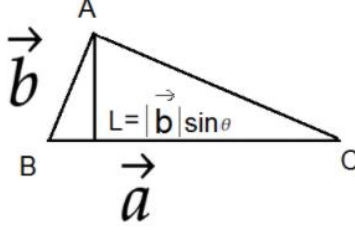


Figure 1. The area of a parallelogram as the magnitude of a cross product

$$\mathbf{a} \times \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \sin(\theta) \, \mathbf{n}$$



$$\begin{aligned} \text{area of } \triangle &= \frac{1}{2} \times |\vec{a}| \times |\vec{b}| \times \sin\theta \\ &= \frac{1}{2} \times [\vec{a} \times \vec{b}] \end{aligned}$$

## Example 5: Finding the Area of a Triangle Given Its Three Vertices

Find the area of a triangle  $ABC$ , where  $A(-8,-9)$ ,  $B(-7,-8)$ , and  $C(9,-2)$ .

## Answer

The magnitude of the cross product of two vectors is equal to the area of the parallelogram spanned by them. The area of the triangle  $ABC$  is equal to half the area of the parallelogram spanned by two vectors defined by its vertices:

$$\text{the area of } ABC = \frac{1}{2} \left\| \overrightarrow{AB} \times \overrightarrow{AC} \right\| = \frac{1}{2} \left\| \overrightarrow{BA} \times \overrightarrow{BC} \right\| = \frac{1}{2} \left\| \overrightarrow{CB} \times \overrightarrow{CA} \right\|.$$

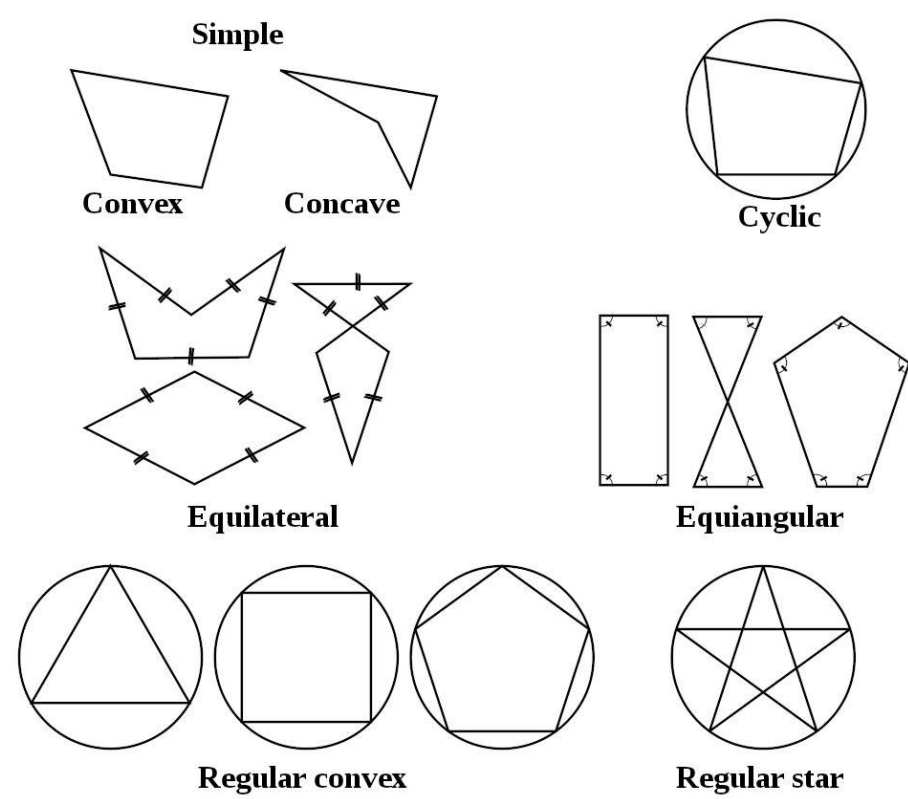
# Polygon

From Wikipedia, the free encyclopedia

## Convexity and non-convexity

Polygons may be characterized by their convexity or type of non-convexity:

- Convex**: ⊗<sup>ⓘ</sup> any line drawn through the polygon (and not tangent to an edge or corner) meets its boundary exactly twice. As a consequence, all its interior angles are less than 180°. Equivalently, any line segment with endpoints on the boundary passes through only interior points between its endpoints.
- Non-convex: a line may be found which meets its boundary more than twice. Equivalently, there exists a line segment between two boundary points that passes outside the polygon.
- Simple**: the boundary of the polygon does not cross itself. All convex polygons are simple.
- Concave**: Non-convex and simple. There is at least one interior angle greater than 180°.



## Equality and symmetry

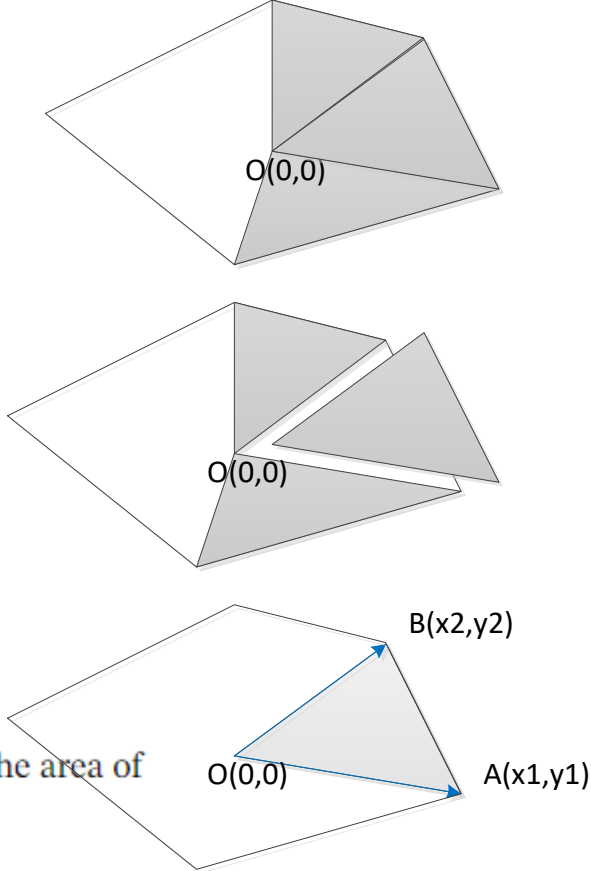
- Regular**: the polygon is both *isogonal* and *isotoxal*. Equivalently, it is both *cyclic* and *equilateral*, or both *equilateral* and *equiangular*. A non-convex regular polygon is called a *regular star polygon*.
- Isogonal** or **vertex-transitive**: all corners lie within the same **symmetry orbit**. The polygon is also cyclic and equiangular.
- Isotoxal** or **edge-transitive**: all sides lie within the same **symmetry orbit**. The polygon is also equilateral and tangential.
- Cyclic**: all corners lie on a single circle, called the **circumcircle**.
- Equilateral**: all edges are of the same length. The polygon need not be convex.
- Equiangular**: all corner angles are equal.
- Tangential**: all sides are tangent to an **inscribed circle**.

## Area

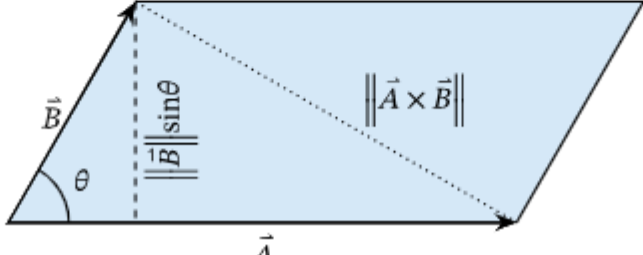
In this section, the vertices of the polygon under consideration are taken to be  $(x_0,y_0),(x_1,y_1),\ldots,(x_{n-1},y_{n-1})$  in order. For convenience in some formulas, the notation  $(x_n,y_n)=(x_0,y_0)$  will also be used.

If the polygon is non-self-intersecting (that is, **simple**), the signed **area** is

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \quad \text{where } x_n = x_0 \text{ and } y_n = y_0,$$



- The cross product is *distributive*:  $(\vec{A} + \vec{B}) \times \vec{C} = \vec{A} \times \vec{C} + \vec{B} \times \vec{C}$ .
- The cross product is *anticommutative*:  $\vec{A} \times \vec{B} = -\vec{B} \times \vec{A}$ .
- The cross product of two collinear vectors is zero, and so  $\vec{A} \times \vec{A} = 0$ .
- The area of the parallelogram spanned by  $\vec{A}$  and  $\vec{B}$  is given by  $\|\vec{A} \times \vec{B}\|$ . It follows that the area of the triangle with  $\vec{A}$  and  $\vec{B}$  defining two of its sides is given by  $\frac{1}{2} \|\vec{A} \times \vec{B}\|$ .



## Centroid

Using the same convention for vertex coordinates as in the previous section, the coordinates of the centroid of a solid simple polygon are

$$C_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i),$$

$$C_y = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i).$$

In these formulas, the signed value of area  $A$  must be used.

For **triangles** ( $n=3$ ), the centroids of the vertices and of the solid shape are the same, but, in general, this is not true for  $n>3$ . The **centroid** of the vertex set of a polygon with  $n$  vertices has the coordinates

$$c_x = \frac{1}{n} \sum_{i=0}^{n-1} x_i,$$

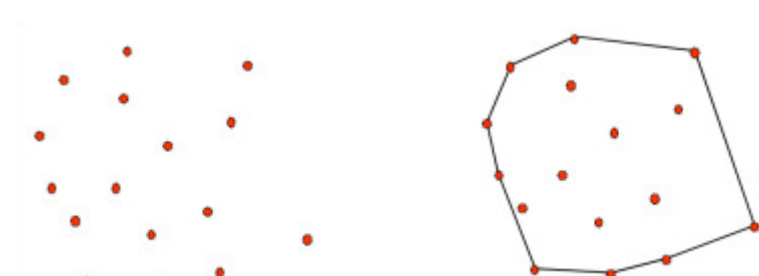
$$c_y = \frac{1}{n} \sum_{i=0}^{n-1} y_i.$$

```
//std::vector<KVector2> g_vertices;
KVector2 geoCenter = KVector2::zero;
for (auto v : g_vertices)
{
    geoCenter += v;
}
geoCenter.x /= g_vertices.size();
geoCenter.y /= g_vertices.size();
for (auto& v : g_vertices)
{
    v -= geoCenter;
}
```

# Convex hull

From Wikipedia, the free encyclopedia

In **geometry**, the **convex hull** or **convex envelope** or **convex closure** of a shape is the **smallest convex set that contains it**. The convex hull may be defined either as the intersection of all convex sets containing a given subset of a **Euclidean space**, or equivalently as the set of all **convex combinations** of points in the subset. For a **bounded** subset of the plane, the convex hull may be visualized as the shape enclosed by a rubber band stretched around the subset.



## Graham scan

From Wikipedia, the free encyclopedia

**Graham's scan** is a method of finding the **convex hull** of a finite set of points in the plane with **time complexity**  $O(n \log n)$ . It is named after **Ronald Graham**, who published the original algorithm in 1972.<sup>[1]</sup> The algorithm finds all vertices of the convex hull ordered along its boundary. It uses a **stack** to detect and remove concavities in the boundary efficiently.

1.

2.

3.

```
let points be the list of points
let stack = empty_stack()

find the lowest y-coordinate and leftmost point, called P0
sort points by polar angle with P0, if several points have the same polar angle then only keep the farthest

for point in points:
    # pop the last point from the stack if we turn clockwise to reach this point
    while count stack > 1 and ccw(next_to_top(stack), top(stack), point) <= 0:
        pop stack
    push point to stack
end
```