



WIKIPEDIA
The Free Encyclopedia

[Main page](#)
[Contents](#)
[Featured content](#)
[Current events](#)
[Random article](#)
[Donate to Wikipedia](#)
[Wikipedia store](#)

Interaction

[Help](#)
[About Wikipedia](#)
[Community portal](#)
[Recent changes](#)
[Contact page](#)

Tools


[What links here](#)
[Related changes](#)
[Upload file](#)
[Special pages](#)
[Permanent link](#)
[Page information](#)
[Wikidata item](#)
[Cite this page](#)

Print/export

[Create a book](#)
[Download as PDF](#)
[Printable version](#)

Languages

[Català](#)
[Deutsch](#)
[فارسی](#)
[Русский](#)
[Српски / srpski](#)
[Українська](#)
[Tiếng Việt](#)

 [Edit links](#)

[Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#) [View history](#)

Point in polygon

From Wikipedia, the free encyclopedia

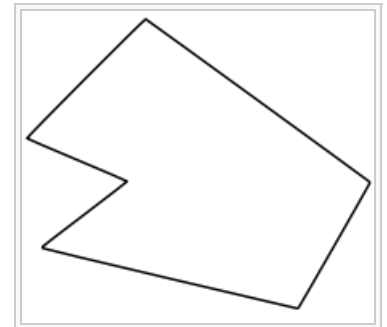
In [computational geometry](#), the **point-in-polygon (PIP)** problem asks whether a given point in the plane lies inside, outside, or on the boundary of a [polygon](#). It is a special case of [point location](#) problems and finds applications in areas that deal with processing geometrical data, such as [computer graphics](#), [computer vision](#), [geographical information systems](#) (GIS), [motion planning](#), and [CAD](#).

An early description of the problem in computer graphics shows two common approaches (ray casting and angle summation) in use as early as 1974.^[1]

An attempt of computer graphics veterans to trace the history of the problem and some tricks for its solution can be found in an issue of the *Ray Tracing News*.^[2]

Contents [hide]

- [1 Ray casting algorithm](#)
- [2 Winding number algorithm](#)
- [3 Comparison](#)
- [4 Point in polygon queries](#)
 - [4.1 Special cases](#)
- [5 References](#)
- [6 See also](#)



An example of a simple polygon

Ray casting algorithm [\[edit\]](#)



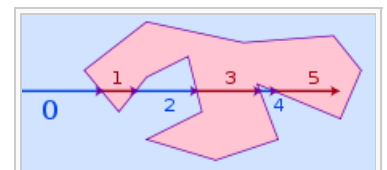
This section **does not cite any references or sources**. Please help improve this section by [adding citations to reliable sources](#). Unsourced material may be challenged and [removed](#). *(June 2013)*

One simple way of finding whether the point is inside or outside a [simple polygon](#) is to test how many times a [ray](#), starting from the point and going in any fixed direction, intersects the edges of the polygon. If the point is on the outside of the polygon the ray will intersect its edge an [even number](#) of times. If the point is on the inside of the polygon then it will intersect the edge an [odd number](#) of times. Unfortunately, this method won't work if the point is *on* the edge of the polygon.

This algorithm is sometimes also known as the **crossing number algorithm** or the **even-odd rule algorithm**, and is known as early as 1962.^[3] The algorithm is based on a simple observation that if a point moves along a ray from infinity to the probe point and if it crosses the boundary of a polygon, possibly several times, then it alternately goes from the outside to inside, then from the inside to the outside, etc. As a result, after every two "border crossings" the moving point goes outside. This observation may be mathematically proved using the [Jordan curve theorem](#).

If implemented on a computer with [finite precision arithmetics](#), the results may be incorrect if the point lies very close to that boundary, because of rounding errors. This is not normally a concern, as speed is much more important than complete accuracy in most applications of computer graphics. However, for a formally correct [computer program](#), one would have to introduce a [numerical tolerance](#) ϵ and test in line whether P (the Point) lies within ϵ of L (the Line), in which case the algorithm should stop and report " P lies very close to the boundary."

Most implementations of the ray casting algorithm consecutively check intersections of a ray with all sides of the



The number of intersections for a ray passing from the exterior of the polygon to any point; if odd, it shows that the point lies inside the polygon. If it is even, the point lies outside the polygon; this test also works in three dimensions.

polygon in turn. In this case the following problem must be addressed. If the ray passes exactly through a [vertex](#) of a polygon, then it will intersect 2 segments at their endpoints. While it is OK for the case of the topmost vertex in the example or the vertex between crossing 4 and 5, the case of the rightmost vertex (in the example) requires that we count one intersection for the algorithm to work correctly. A similar problem arises with horizontal segments that happen to fall on the ray. The issue is solved as follows: If the intersection point is a vertex of a tested polygon side, then the intersection counts only if the second vertex of the side lies below the ray. This is effectively equivalent to considering vertices *on* the ray as lying slightly *above* the ray.

Once again, the case of the ray passing through a vertex may pose numerical problems in [finite precision arithmetics](#): for two sides adjacent to the same vertex the straightforward computation of the intersection with a ray may not give the vertex in both cases. If the polygon is specified by its vertices, then this problem is eliminated by checking the y-coordinates of the ray and the ends of the tested polygon side before actual computation of the intersection. In other cases, when polygon sides are computed from other types of data, other tricks must be applied for the [numerical robustness](#) of the algorithm.

Winding number algorithm [\[edit\]](#)

Another algorithm is to compute the given point's [winding number](#) with respect to the polygon. If the winding number is non-zero, the point lies inside the polygon. One way to compute the winding number is to sum up the [angles subtended](#) by each side of the polygon.^[4] However, this involves costly [inverse trigonometric functions](#), which generally makes this algorithm slower than the ray casting algorithm. Luckily, these inverse trigonometric functions do not need to be computed. Since the result, the sum of all angles, can add up to 0 or 2π (or multiples of 2π) only, it is sufficient to track through which quadrants the polygon winds,^[5] as it turns around the test point, which makes the winding number algorithm comparable in speed to counting the boundary crossings.

There is a significant speed-up (known since 2001) of the winding number algorithm. It uses signed crossings, based on whether each crossing is left-to-right or right-to-left. Details and C++ code are given at ^[6]. Angles are not used, and no trigonometry is involved. The code is as fast as the simple boundary crossing algorithm. Further, it gives the correct answer for nonsimple polygons, whereas the boundary crossing algorithm fails in this case.

Comparison [\[edit\]](#)

For [simple polygons](#), both algorithms will always give the same results for all points. However, for [complex polygons](#), the algorithms may give different results for points in the regions where the polygon intersects itself, where the polygon does not have a clearly defined inside and outside. In this case, the former algorithm is called the [even-odd rule](#). One solution is to transform (complex) polygons in simpler, but even-odd-equivalent ones before the intersection check.^[7] This, however, is computationally expensive. It is better to use the fast "signed crossings" winding number algorithm which gives the correct result, even when the polygon overlaps itself. The point is then inside the polygon whenever the winding number is nonzero.

Point in polygon queries [\[edit\]](#)

The point in polygon problem may be considered in the general repeated [geometric query](#) setting: given a single polygon and a sequence of query points, quickly find the answers for each query point. Clearly, any of the general approaches for planar [point location](#) may be used. Simpler solutions are available for some special polygons.

Special cases [\[edit\]](#)

 This section requires [expansion](#).
(August 2013)

Simpler algorithms are possible for [monotone polygons](#), [star-shaped polygons](#), [convex polygons](#) and [triangles](#).

The triangle case can be solved easily by use of a [barycentric coordinate system](#), [parametric equation](#) or [dot product](#).^[8] The dot product method extends naturally to any convex polygon.

References [\[edit\]](#)

- [↑] [Ivan Sutherland](#) et al., "A Characterization of Ten Hidden-Surface Algorithms" 1974, *ACM Computing Surveys* vol. 6 no. 1.
- [↑] ["Point in Polygon, One More Time..."](#) [↗](#), *Ray Tracing News*, vol. 3 no. 4, October 1, 1990.
- [↑] [Shimrat, M.](#), "Algorithm 112: Position of point relative to polygon" 1962, *Communications of the ACM* Volume 5

Issue 8, Aug. 1962

4. [^] Hornmann, K.; Agathos, A. (2001). "The point in polygon problem for arbitrary polygons". *Computational Geometry* **20** (3): 131. doi:[10.1016/S0925-7721\(01\)00012-8](https://doi.org/10.1016/S0925-7721(01)00012-8)[↗].
5. [^] Weiler, Kevin (1994), "An Incremental Angle Point in Polygon Test", in Heckbert, Paul S., *Graphics Gems IV*, San Diego, CA, USA: Academic Press Professional, Inc., pp. 16–23, ISBN 0-12-336155-9.
6. [^] Sunday, Dan (2001), *Inclusion of a Point in a Polygon*, http://geomalgorithms.com/a03-_inclusion.html[↗].
7. [^] Michael Galetzka, Patrick Glauner (2012), *A correct even-odd algorithm for the point-in-polygon (PIP) problem for complex polygons*, [arXiv:1207.3502](https://arxiv.org/abs/1207.3502)[↗]
8. [^] [Accurate point in triangle test](#)[↗] "...the most famous methods to solve it"

See also ^[edit]

- [Java Topology Suite \(JTS\)](#)
- Discussion: <http://www.ics.uci.edu/~eppstein/161/960307.html>[↗]
- Winding number versus crossing number methods: http://geomalgorithms.com/a03-_inclusion.html[↗]

Categories: [Geometric algorithms](#)

This page was last modified on 28 August 2015, at 18:38.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

