Efficient Angle Summation Algorithm for Point Inclusion Test and Its Robustness*

Stepan Yu. Gatilov
A.P. Ershov Institute of Informatics Systems,
Novosibirsk, Russia
Ledas Ltd., Novosibirsk, Russia
stgatilov@gmail.com

Abstract

A winding angle summation approach to the point-in-polygon problem is considered. The winding angle summation is backward stable and resistant to possible gaps between subsequent edges. A precomputed bounding volume hierarchy can be used to accelerate point queries for winding angle summation. The main theorem states that a BVH-accelerated point query takes no longer than $O\left(K \log \frac{n}{K}\right)$ time if all the bounding boxes are tight. Here, n is the number of edges in the polygon, and K is the scaled absolute winding angle (called the condition number). It is proved that the absolute winding angle never exceeds the absolute turning angle for a closed piecewise smooth regular curve. Hence, there is a non-trivial bound on query time complexity, independent of the point being tested.

Keywords: point-in-polygon, winding angle, bounding volume hierarchy
AMS subject classifications: 65D17, 65D18

1 Introduction

The point-in-polygon (PIP) problem is a basic problem of computational geometry. It asks whether a given point in the plane lies inside a given simple polygon. Sometimes, one of the three answers must be chosen: inside, outside, or on the boundary. In a classical setting, the polygon is defined by a sequence of its vertices, all the edges being line segments. In a generalized setting, the polygon is defined by a sequence of its edges, and some of the edges can be curvilinear.

The PIP problem and more general point location problems often arise in GIS and CAD applications. In CAD applications, the data usually is imprecise. Edges can be slightly offset, and there can be small gaps between subsequent edges of the polygon.

A good survey of simple solutions of the classical PIP is presented in [2]. The reliability of existing floating point implementations is studied in [12]. Perhaps the most popular approach is a ray intersection method. It states that any ray cast from

*Submitted: December 23, 2012; Revised: July 22, 2013; Accepted: August 17, 2013.

the point intersects the polygon in an odd number of points if and only if the point lies inside. This rule does not work properly in case the ray passes through a vertex[1].

A floating point implementation can produce incorrect results when the ray passes close to the point. For the classical PIP, this problem can be avoided by choosing axis-aligned directions and handling all the cases correctly. However, this solution does not work in a generalized setting. The second idea is to check several rays with random directions and use voting to determine the final result. This decreases dramatically the probability of incorrect result, but the possibility is never eliminated completely.

If several points are tested against a single polygon, the queries can be accelerated by precomputation. Bounding volume hierarchy (BVH) allows quick pruning of sets of edges known certainly not to intersect the ray. This speeds up the queries greatly, although upper bounds on the time complexity are not known[2].

Another well-known approach to the PIP problem is winding angle calculation. Here is the general idea. Put one end point of the vector onto the point being tested and trace the other end point along the polygon. After a single lap, count how many revolutions the vector performed. The winding number is zero if and only if the point lies outside the polygon.

The curse of the winding angle summation is its heavy use of inverse trigonometric functions. This is why winding angle summation is ofter considered to be the slowest and the worst solution to the PIP problem. As a compensation, the trivial winding angle summation test is backward stable. The stability of the winding angle summation algorithm is studied in the current paper.

An attempt to calculate the winding number without inverse trigonometric functions was made in [4], which suggests casting four axis-aligned rays and handling the intersection points cleverly. Hence, this modification looses the stability of winding angle summation and is very similar to the ray intersection method.

The current paper shows that the winding angle summation queries can be accelerated by a precomputed BVH. The idea is to replace a subsequence of edges with a single line segment on a proper condition. This allows quick pruning of subsequences of edges. The query takes no more than $O\left(K \log \frac{n}{K}\right)$ time, where K is the condition number of the problem. This bound is true only if all the bounding boxes in the BVH are tight.

Note that the classical point-in-polygon problem can be solved in optimal $O(\log N)$ time. Several optimal solutions to a more general point location problem have been proposed (see [1, 7, 11]). These algorithms are significantly more complex than the simple ones described above. Some of them can be generalized to curvilinear cases. Their stability analysis is not the purpose of the current paper. However, it is quite likely that they raise more stability concerns than the simple algorithms.

The paper is organized as follows. Section 2 contains general definitions and a trivial angle summation algorithm. Section 3 shows how to calculate winding angle along line segments and circular arcs. The error bounds due to floating point arithmetic are given for the line segment. Section 4 is dedicated to various stability questions. The condition number K is introduced. It is shown how edge perturbations affect the winding angle. The overall goal is to show backward stability of the algorithm. The BVH-accelerated algorithm with the complexity theorem is presented in Section 5. The application of the algorithm to NURBS curves[3] is described in Section 6.

[1] or touches a curvilinear edge

[2] more precisely, the author is not aware of such bounds

[3] NURBS curves is a class of spline curves widely used in computed-aided design software.

2 Angle Summation

A piecewise smooth regular simple contour γ defined on $[0,1]$ and a point P ($P \notin \gamma$) are given as inputs to the algorithm. The contour is given as a sequence of individual edges $\gamma_0, \gamma_1, \ldots, \gamma_{n-1}$ ($\gamma_i : [s_i, e_i] \to \mathbb{R}^2$). Each edge is smooth and belongs to some class of supported edges such as line segments, circular arcs, Bezier curves, etc.

We define the polar angle function $\varphi : [0,1] \to \mathbb{R}$ as follows: for any $t \in [0,1]$, the value $\varphi(t)$ is equal to the polar angle of the vector $(\gamma(t) - P)$. Since the polar angle is a multivalued function, we require the function $\varphi(t)$ to be continuous.

This function can also be defined nicely on a complex plane (points are treated as complex numbers):

$$\varphi(t) = \operatorname{Im} \ln (\gamma(t) - P) . \tag{1}$$

The difference of the polar angle φ along the curve γ is defined as

$$D^{\gamma}[\varphi] = \varphi(1) - \varphi(0) = \int_0^1 \dot{\varphi}(t)\, dt = \int_0^1 \frac{(\gamma(t) - P) \times \dot{\gamma}(t)}{\|\gamma(t) - P\| \|\dot{\gamma}(t)\|}\, dt.$$

This value is called winding angle along the curve γ.

We also will use the variation of polar angle along the curve γ, which is defined similarly to the difference,

$$V^{\gamma}[\varphi] = \int_0^1 |\dot{\varphi}(t)|\, dt,$$

which also can be called the absolute winding angle for obvious reasons.

The winding number is

$$W = \frac{1}{2\pi} D^{\gamma}[\varphi].$$

The winding number is an integer even if the contour is not simple. It follows from the argument principle that for a simple contour, the winding number must be either zero or one (or minus one for a clockwise contour orientation). Moreover, the winding number is zero if and only if the point lies outside the contour.

Hence, it is possible to use Algorithm 1.

Algorithm 1: Trivial winding angle summation

Input: A sequence of edges: $\gamma_0, \gamma_1, \ldots, \gamma_{n-1}$
for $i \in 0, \ldots, n-1$ **do** Calculate $D^{\gamma_i}[\varphi]$
Sum all the winding angles and scale by 2π to get W
if $W = 0$ **then return** Outside **else return** Inside

Since the precise winding number of the contour is known to be an integer, the calculated winding number can be rounded to the nearest integer before comparison. Note that the winding number can be calculated even for contours with self-intersections.

3 Edge Winding Angle Calculation

We assume that it is easy enough to calculate the winding angle for an edge. A line segment and a circular arc will be considered as examples.

3.1 Line Segment

Let γ be a directed line segment represented by coordinates of its endpoints. Then the winding angle can be calculated numerically by Algorithm 2.

Algorithm 2: `LineSegmentAngle`

Input: segment endpoints $S, E \in \mathbb{R}^2$, a point $P \in \mathbb{R}^2$
$U := S - P, \quad V := E - P$
return `atan2` $(U \times V,\, U \cdot V)$

Algorithm 2 uses the inverse trigonometric function `atan2`(y, x), which returns the polar angle in the range $[-\pi, \pi]$ for a nonzero vector with coordinates (x, y). Inverse trigonometric functions are very slow, explaining why the trivial winding angle calculation is considered to be the slowest point-in-polygon method. Another drawback is that the function `atan2`(y, x) is not present in the **IEEE 754** floating point standard.

We assume that there is a guaranteed upper bound $C\varepsilon$ on the relative error of the function `atan2`. Of course, the error is defined modulo 2π, meaning that jumps near $\pm\pi$ are possible. For instance, the Pentium architecture ensures that the error does not exceed 1 ulp ($= 2\varepsilon$) in round-to-nearest mode [5]. For more information about transcendental function evaluation, see [13].

Assuming that the input data is exact, it is rather easy to derive an upper bound on the error of Algorithm 2. It is well known that the error of dot and cross products does not exceed $2\varepsilon(1 + O(\varepsilon))$ multiplied by the product of vector norms. Combine that with subtraction of P, and the errors of `atan2` arguments are[4]

$$x_{err} = \frac{(U \cdot V)_{err}}{\|U\|\|V\|} \le 4\varepsilon\,(1 + O(\varepsilon))\,, \quad y_{err} = \frac{(U \times V)_{err}}{\|U\|\|V\|} \le 4\varepsilon\,(1 + O(\varepsilon))\,.$$

The exact values of these two numbers are $x = \cos\triangle\varphi$ and $y = \sin\triangle\varphi$. Then the maximal deviation of angle $\triangle\varphi$ due to imprecise arguments does not exceed $4\sqrt{2}\varepsilon\,(1 + O(\varepsilon))$. The `atan2` function itself adds up to $C\pi\varepsilon$ error. Hence, the overall error of the angle is

$$\triangle\varphi_{err} \le \left(4\sqrt{2} + C\pi\right)\varepsilon \le q_{ang}\,\varepsilon,$$

where the constant q_{ang} is introduced for brevity. However, this error is modulo 2π. If any jump near $\pm\pi$ occurs, the winding number also jumps by one (which destroys the point-in-polygon test). Therefore, it is necessary to bound the true angle away from $\pm\pi$ by at least the error of computation. To achieve this, the distance from the point P to the line segment $[SE]$ should be bounded from below ($L = |SE|$) by

$$\operatorname{dist}(P, [SE]) > \frac{q_{ang}}{4}\varepsilon L. \tag{2}$$

The proved error bound shows that the winding angle calculation for line segment is very precise unless the point is very close to the segment. It is quite interesting that the error bound does not depend on the distance from the point to the end points. When the distance is very small (e.g., $O(\varepsilon)$), even 1 ulp perturbation in any input number can yield large error (e.g., $O(1)$).

Sometimes the function `acos` is used instead of `atan2` for angle calculation. This is a bad idea because the error for very small angles can be of the order of $\sqrt{\varepsilon}$.

[4]Dot and cross products are denoted as $u \cdot v$ and $u \times v$. V_{err} denotes the computational error of the value V.

3.2 Circular Arc

Let γ be a directed circular arc represented by the coordinates of its center C, its radius r, and the polar angles α and β of its start and end points. The sign of $\beta - \alpha$ determines the orientation of the arc (clockwise or counterclockwise), and its absolute value determines the length of the arc (less than 2π).

Algorithm 3: Winding angle calculation for a circular arc

Input: center $C \in \mathbb{R}^2$, radius $r \in \mathbb{R}$, angles $\alpha, \beta \in \mathbb{R}$, a point $P \in \mathbb{R}^2$
$S := (C_x + r\cos\alpha; C_y + r\sin\alpha)$
$E := (C_x + r\cos\beta; C_y + r\sin\beta)$
$\theta :=$ `LineSegmentAngle`(S, E, P)
if $\|C - P\| < r$ **then**
  if $\beta > \alpha$ **then** `NormalizePeriodic`$(\theta \to [0, 2\pi])$
  else `NormalizePeriodic`$(\theta \to [-2\pi, 0])$
end
return θ

Algorithm 3 uses the function `LineSegmentAngle`, which is implemented by Algorithm 2. The function `NormalizePeriodic` returns the angle, which is equivalent to its argument θ modulo 2π, but belonging to a specified interval. It is possible to perform error analysis of this algorithm, but it is too tedious to consider all the cases. One point is that the error depends on the distance from the point to the arc endpoints.

4 Stability

4.1 Condition Number

Definition 4.1. *Given a piecewise smooth regular contour γ and a point P not on it, the condition number of the winding angle summation problem (K) is defined as*[5]

$$K = K(\gamma, P) = \frac{1}{2\pi} V^{\gamma}[\varphi].$$

The condition number is important for both winding angle summation and ray intersection approaches to the point-in-polygon problem.

The following proposition defines K in terms of ray intersection.

Proposition 4.1. *Suppose that the derivative $\dot{\varphi}(t)$ of the polar angle function has a finite number of zeros. Let ξ be a uniform random variable with values in $[0, 2\pi)$. Cast a ray from the point P into the direction with polar angle ξ. Then the average number of points where it intersects the contour γ is equal to K.*

Proof. Since the number of zeros and breaks of $\dot{\varphi}(t)$ is finite, and the total variation of $\varphi(t)$ is finite, the curve γ can be divided into a finite number of pieces $p_i : [a_i, b_i] \to \mathbb{R}^2$ such that:

1. Each piece is continuously differentiable (no breaks).

[5] see Section 2 for the definition of φ

2. The polar angle function $\varphi(t)$ is strictly monotonic on each piece.
3. The total variation of $\varphi(t)$ on each piece is less than 2π.

The mathematical expectation is an integral of the number of intersection points over angles $[0, 2\pi)$ divided by 2π. The condition number is an integral of $|\dot\varphi(t)|$ over the parameter $t \in [0, 1]$ divided by 2π. Both integrals are additive, so it is sufficient to prove the required equality for all the pieces.

The ray intersects a piece p_i at exactly one point for $\xi \in [\varphi(a_i), \varphi(b_i)]$[6] and does not intersect it for other values of ξ. Hence, the average number of intersection points is $\frac{1}{2\pi}|\varphi(a_i) - \varphi(b_i)|$. On the other hand, the variation of φ is equal to the absolute value of the difference, so K has exactly the same value for the piece. □

Clearly, the proposition is also true for piecewise-analytical regular contours. Each analytical piece either has constant $\varphi(t)$ or has a finite number of points with $\dot\varphi(t) = 0$. The pieces with constant $\varphi(t)$ do not affect both integrals (there is a finite number of them).

4.2 Summation Error

Algorithm 1 computes the sum of many real numbers. When computing the sum of n real numbers a_i by the trivial algorithm, the error bound is (see [3] for details)

$$S_{err} \le n\varepsilon \sum_{i=0}^{n-1} |a_i| + O\left(\varepsilon^2\right). \tag{3}$$

The condition number of the summation problem, which shows sensitivity to input perturbations, is

$$\text{cond} = \frac{\sum |a_i|}{|\sum a_i|}.$$

In the case of winding angle summation, $a_i = D^{\gamma_i}[\varphi]$. The condition number cond described above is useless in this context because the true sum is often zero. Moreover, the calculated sum is rounded to the nearest multiple of 2π because the true sum is known to be such a multiple. That is why it is more beneficial to consider a number

$$K_1 = \frac{1}{2\pi} \sum_i |a_i|.$$

This number shows how much relative perturbations in a_i affect the winding number,

$$|\triangle W| \le K_1 \max_i \frac{|\triangle a_i|}{|a_i|}.$$

Proposition 4.2. *The numbers K_1 and K are related:*
1. *$K_1 \le K$.*
2. *As long as the contour γ is tesselated into smaller and smaller edges, the number K_1 converges to K.*

Proof. The first part follows from triangle inequality:

$$\begin{aligned} 2\pi K_1 &= \sum_i |D^{\gamma_i}[\varphi]| = \sum_i \left| \int_{\gamma_i} \dot\varphi(t)\, dt \right| \\ &\le \sum_i \int_{\gamma_i} |\dot\varphi(t)|\, dt = \sum_i V^{\gamma_i}[\varphi] = V^{\gamma}[\varphi] = 2\pi K. \end{aligned}$$

[6] interval endpoints may be reversed

Since $\dot\varphi(t)$ is continuous on every edge γ_i, according to the Mean Value Theorem,

$$|D^{\gamma_i}[\varphi]| = \left|\int_{\gamma_i} \dot\varphi(t)\,dt\right| = |\dot\varphi(t_i^*)|\ (e_i - s_i),$$

where $[s_i, e_i]$ is the domain of edge γ_i, and $t_i^* \in (s_i, e_i)$ is some parameter inside it. Now it is easy to see that $2\pi K_1$ is a Riemann sum of the integral $\int_\gamma |\dot\varphi(t)|\,dt$. Hence, if $\max(e_i - s_i) \to 0$, then $K_1 \to K$ due to definition of Riemann integral. □

Also, if all the edges are line segments, then $K_1 = K$.

4.3 Edge Perturbation Error

The algorithm is stable under edge perturbations. To show this, we prove that small perturbation in an edge results in a small perturbation in its winding angle.

Definition 4.2. *Consider a piecewise smooth regular curve $\gamma : [s, e] \to \mathbb{R}^2$. A piecewise smooth regular curve γ_δ is its δ-perturbation if*

$$\forall t \in [s; e] \quad \|\gamma_\delta(t) - \gamma(t)\| < \delta.$$

Proposition 4.3. *Let γ_δ be a δ-perturbation of a curve γ with $\delta < \operatorname{dist}(P, \gamma) = d$. Then*

$$|D^{\gamma_\delta}[\varphi] - D^{\gamma}[\varphi]| \le 2 \arcsin \frac{\delta}{d}.$$

Proof. We denote the curve γ thickened by δ as G:

$$G = B(\gamma, \delta) = \left\{Q \in \mathbb{R}^2 \mid \operatorname{dist}(Q, \gamma) < \delta\right\}.$$

The curves γ and γ_δ are inside G, and the point P is outside (with positive distance). Due to (1), the winding angle can be defined as a complex integral

$$D^{\gamma}[\varphi] = \int_s^e \dot\varphi(t)\,dt = \operatorname{Im} \int_s^e \frac{\dot\gamma(t)\,dt}{\gamma(t) - P} = \operatorname{Im} \int_\gamma \frac{dz}{z - P}.$$

The curves γ and γ_δ are homotophic in G by the trivial linear homotopy $\Gamma(\theta, t) = (1 - \theta)\gamma(t) + \theta\gamma_\delta(t)$. Let $\gamma(s) = A, \gamma(e) = B, \gamma_\delta(s) = C$, and $\gamma_\delta(e) = D$. Construct a curve $\widehat{\gamma_\delta} = \gamma_{AC} \cup \gamma_\delta \cup \gamma_{DB}$, where γ_{AC} and γ_{DB} are line segments $[AC]$ and $[DB]$, respectively. These line segments are homotopic to the endpoints of the curve γ (which are A and B). Hence, the curves $\widehat{\gamma_\delta}$ and γ are homotopic and have same endpoints.

The function $1/(z - P)$ is holomorphic in G. By the Cauchy theorem, its integrals along the curves γ and $\widehat{\gamma_\delta}$ are equal,

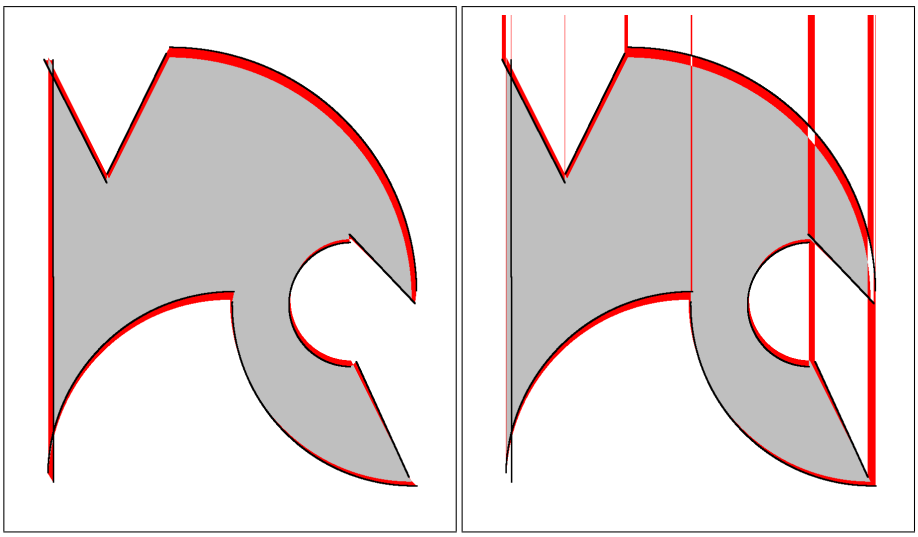
$$\operatorname{Im} \int_{\widehat{\gamma_\delta}} \frac{dz}{z - P} = \operatorname{Im} \int_\gamma \frac{dz}{z - P}.$$

Then it is easy to see that

$$\begin{aligned} |D^{\gamma}[\varphi] - D^{\gamma_\delta}[\varphi]| &= |D^{\gamma_{AC}}[\varphi] + D^{\gamma_{DB}}[\varphi]| \\ &\le \arcsin \frac{\delta}{\|P - A\|} + \arcsin \frac{\delta}{\|P - B\|} \\ &\le 2 \arcsin \frac{\delta}{\operatorname{dist}(P, \gamma)}, \end{aligned}$$

where the last inequalities are obvious by geometric means. □

Note that the edge perturbation can change the endpoints of the edge. Hence, the winding angle summation algorithm is resistant to gaps between subsequent edges as long as these gaps are small enough. Figure 1 shows how winding angle summation and ray intersection behave on a perturbed contour.



(a) winding angle summation **(b)** ray intersection

Figure 1: Behavior of the methods on a perturbed contour. Perturbed edges are drawn in black; misclassified points are indicated in red.

4.4 Backward Stability

Suppose that the error of the calculated winding angle along an edge is small enough if the point lies far enough from it,

$$\operatorname{dist}(P,\gamma_i) = d > \theta M\varepsilon \quad \Longrightarrow \quad D^{\gamma_i}[\varphi]_{err} < \alpha\varepsilon + \beta\frac{M\varepsilon}{d}. \tag{4}$$

Let us call such a class of edges "precise". Here M is the scale of the contour (e.g., maximal absolute coordinate). For instance, the class of line segments is "precise" with $\alpha = q_{ang}, \beta = 0, \theta = \frac{1}{4}q_{ang}$ (see Section 3.1).

Theorem 4.1. *Assume that the class of edges is "precise". Suppose that the edges are perturbed with magnitude not greater than δ prior to the winding angle calculation. Then Algorithm 1 produces correct result provided that*

$$n\varepsilon < \frac{1}{3} \quad \textit{and} \quad \operatorname{dist}(P,\gamma) = d > \max\left\{2\delta,\ \delta + \theta M\varepsilon,\ n\frac{4\delta + 3\beta M\varepsilon}{\pi - n\varepsilon(2\pi K + 2\alpha)}\right\},$$

where the denominator of the last fraction is required to be positive.

Proof. The error in winding angle comes from three sources. First of all, the edges are perturbed. The error due to perturbation does not exceed (see Proposition 4.3)

$$A = n \cdot 2 \arcsin \frac{\delta}{d}.$$

After the edges are perturbed, the winding angle is calculated for them. As the edges are still in the "precise" class (see (4)), the error does not exceed[7]

$$B = n \cdot \left(\alpha\varepsilon + \beta \frac{M\varepsilon}{d - \delta} \right).$$

Finally, the winding angles for individual edges are summed altogether. The summation error is bounded by (see (3))[8]

$$C = n\varepsilon \left(2\pi K + (A + B) \right).$$

The algorithm works correctly if the overall error is less than π, i.e.

$$2\pi K n\varepsilon + (A + B)(1 + n\varepsilon) < \pi.$$

Using $\arcsin(\frac{\delta}{d}) < \frac{3}{2}\frac{\delta}{d}$ and $\frac{1}{d-\delta} < \frac{2}{d}$, the condition can be strengthened:

$$2\pi K n\varepsilon + \left(3n\frac{\delta}{d} + \alpha n\varepsilon + 2n\beta\frac{M\varepsilon}{d} \right)(1 + n\varepsilon) < \pi.$$

Since $1 + n\varepsilon < \frac{4}{3}$, the stronger condition is

$$\frac{4n\delta}{d} + \frac{3n\beta M\varepsilon}{d} < \pi - 2\pi K n\varepsilon - 2\alpha n\varepsilon.$$

If the right hand side is positive, then it is equivalent to

$$d > \frac{4n\delta + 3n\beta M\varepsilon}{\pi - 2\pi K n\varepsilon - 2\alpha n\varepsilon},$$

which is exactly the condition from the theorem statement. □

Theorem 4.1 gives some idea about how bad the contour can be so that the winding angle algorithm still works correctly. Also, it shows that the algorithm is backward stable in the usual sense. The calculated result is always exact for a slightly modified input; the point may be moved by at most D^* (where D^* is the lower bound on the distance from the theorem).

5 Acceleration

The acceleration of the winding angle summation requires precomputation of the bounding volume hierarchy (otherwise $O(n)$ is the complexity of the fastest algorithm possible). A rooted ordered tree T is built during the precomputation phase. The set of its nodes is denoted by $V(T)$ and the set of its leaves by $L(T)$.

Each leaf of the tree T corresponds to a single edge of the contour; a leaf $v \in L(T)$ corresponds to the edge γ_v. For an internal node $v \in V(T) \setminus L(T)$, the sequence of

[7] The distance d may have decreased by at most δ.

[8] The true sum of absolute values is $2\pi K$, but it may have increased by $A + B$.

curves corresponding to the leaves in a v-subtree is denoted by γ_v. For the algorithm to be correct, the curve γ_v must be a continuous part of the whole contour γ, i.e.

$$\gamma_v = \gamma|_{[s_{\alpha_v}, e_{\beta_v}]} ,$$

where α_v and β_v are the leftmost and the rightmost leaves in the v-subtree. Later, we will use the notation $s_v = s_{\alpha_v}$, $e_v = e_{\beta_v}$.

An axis-aligned bounding box B_v of the curve γ_v is stored in each node $v \in V(T)$,

$$\gamma_v \subseteq B_v.$$

Bounding boxes B_v are not required to be minimal, but the algorithm runs faster for tighter boxes. Strictly speaking, the boxes are considered to be closed sets in all the proofs.

The algorithm works correctly for any structure of the tree T. However, a balanced binary tree is a great choice for both its efficiency and simplicity. Hence, all other variants are ignored; from this point, T is assumed to be a balanced binary tree. The left and the right sons of a node v are denoted by $\mathcal{L}(v)$ and $\mathcal{R}(v)$, respectively.

5.1 Precomputation

Algorithm 4: `Precompute`

Input: A sequence of edges: $\gamma_0, \gamma_1, \ldots, \gamma_{n-1}$
Output: A tree T with precomputed values B_v, α_v, β_v for all $v \in V(T)$

Build a balanced binary tree T with n leaves.
Put the edges $\gamma_0, \ldots, \gamma_{n-1}$ into correspondence with the leaves of T (in correct order).
for $v \in L(T)$ **do**
    $B_v :=$ `EdgeBoundingBox`(γ_v)
    $\alpha_v, \beta_v := v$
end
for $v \in V(T) \setminus L(T)$ *ordered by height decreasing* **do**
    $B_v :=$ `SupBox`$(B_{\mathcal{L}(v)}, B_{\mathcal{R}(v)})$
    $\alpha_v := \alpha_{\mathcal{L}(v)}$, $\beta_v := \beta_{\mathcal{R}(v)}$
end

Algorithm 4 builds a tree T with all the desired values. The function `EdgeBoundingBox` calculates some enclosing axis-aligned box for a given input edge, and the function `SupBox` finds the minimal enclosing box for a pair of boxes.

5.2 Query

A query is handled by recursive traversal of the tree T with the following pruning. If the point P lies outside of the box B_v, then the winding angle along the curve γ_v is computed immediately. This angle is equal to the winding angle of the line segment with the same end points.

Algorithm 5: `NodeAngle`

Input: A node $v \in V(T)$, a point $P \notin \gamma_v$
Output: Winding angle along the curve γ_v around P

```
1 if P ∉ B_v then
      return LineSegmentAngle (γ_v(s_v), γ_v(e_v), P)
  end
2 if v ∈ L(T) then
      return EdgeAngle (γ_v[s_v ... e_v], P)
  end
3 return NodeAngle (L(v), P) + NodeAngle (R(v), P)
```

Algorithm 6: `Query`

Input: A point $P \notin \gamma$
Output: Winding angle around P along the curve γ

```
// Launch traversal of the tree T from its root r
return NodeAngle (r, P)
```

Proposition 5.1. *Algorithm 5 returns $D^{\gamma_v}[\varphi]$ for an arbitrary node $v \in V(T)$*[9].

Proof. Proved by induction over the tree T.

Three cases are possible for a node $v \in V(T)$ as labeled on the left of the algorithm pseudocode. In case 2, the function `EdgeAngle` explicitly calculates the winding angle along the curve γ_v. In case 3, induction hypothesis and continuity of the curve γ_v are used. If $l = \mathcal{R}(v)$ and $r = \mathcal{L}(v)$ are the sons of the internal node v,

$$D^{\gamma_l}[\varphi] + D^{\gamma_r}[\varphi] = D^{\gamma_v}[\varphi],$$

The only nontrivial case is 1 (see Figure 2).

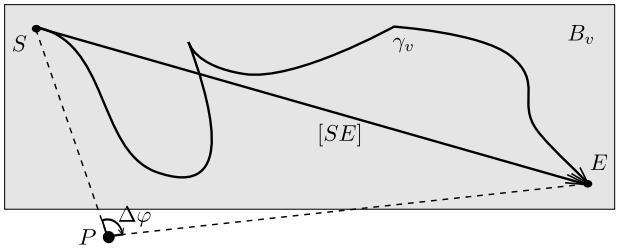


Figure 2: Curve γ_v is equivalent to segment $[SE]$.

To prove it, we express the winding angle in terms of a complex integral[10]. The integrand $1/(z-P)$ is holomorphic in B_v because $P \notin B_v$ (and $\operatorname{dist}(P, B_v) > 0$). The curve γ_v is trivially homotopic to the line segment $[SE]$ in the domain B_v (S is the

[9] when executed in exact arithmetic
[10] as in the proof of Proposition 4.3

starting point of γ_v, and E is its end point). Both curves are piecewise smooth regular, so by the Cauchy theorem, the winding angles of γ_v and $[SE]$ are equal. Obviously, $P \notin [SE]$, so the function `LineSegmentAngle` precisely calculates the winding angle of $[SE]$. □

By setting $v = r$ (root), the following is proved:

Corollary 5.1. *Algorithm 6 returns $D^\gamma[\varphi]$.*

It should be noted that, unlike the trivial angle summation, the accelerated algorithm contains a minor instability. Checking $P \notin B_v$ in the case 1 of Algorithm 5 is not sufficient. The point P can be arbitrarily close to the line segment $[SE]$, which can lead to a failure of the `LineSegmentAngle` function due to finite precision arithmetic.

To solve the problem, a practical implementation must slightly enlarge the box B_v prior to the check. An acceptable tolerance value can be derived from (2). Alternatively, one can use interval arithmetic inside the function `LineSegmentAngle` and check afterwards that the width of the returned interval is less than π.

5.3 Complexity Theorem

The algorithm uses several basic functions which are implemented differently depending on the class of edges. We introduce time complexities $\mathcal{T}_{ea}$ and $\mathcal{T}_{bb}$, such that

1. The function `EdgeAngle` takes $O(\mathcal{T}_{ea})$ time, and
2. The function `EdgeBoundingBox` takes $O(\mathcal{T}_{bb})$ time.

It is assumed that these functions are not free (i.e., they take at least constant time).

Proposition 5.2. *The precomputation of Algorithm 4 requires $O(n)$ additional memory and takes $O(n\mathcal{T}_{bb})$ time.*

Proof. T is a binary tree, so $|V(T)| = 2|L(T)| - 1 = O(n)$. The rest is obvious. □

Theorem 5.1.

1. *Assume that the function* `EdgeBoundingBox` *computes a tight (i.e., inclusion minimal) bounding box for all the edges.*
2. *Define $\hat{K}$ as*[11]
$$\hat{K} = \min(4K, n).$$

Then the time complexity of the query (Algorithm 6) is

$$\begin{cases} O(\hat{K} \log \frac{n}{\hat{K}} \ + \ \hat{K}\mathcal{T}_{ea}), & \hat{K} \geq 1 \\ O(1), & \hat{K} < 1. \end{cases}$$

Several lemmas are necessary for the proof.

Lemma 5.1. *Let B_v be a tight bounding box for a curve γ_v ($v \in V(T)$). Then if $P \in B_v$, the total variation of the polar angle along γ_v can be bounded from below:*

$$V^{\gamma_v}[\varphi] \geq \frac{\pi}{2}.$$

[11] see Definition 4.1 for the condition number K

Proof. Since the bounding box B_v is minimal by inclusion, each of its sides contains at least one point of γ_v. Denote four such points (one on each side) by A_i and the vertices of the box by Q_i $(i = 1 \dots 4)$. The points can be ordered cyclically by polar angle around the center of the box: $Q_1, A_1, Q_2, A_2, Q_3, A_3, Q_4, A_4$. None of the points A_i coincides with P because $P \notin \gamma_v$.

To prove the lemma, it is enough to show that there are points A_i and A_j such that

$$\angle A_i P A_j \geq \frac{\pi}{2}. \tag{5}$$

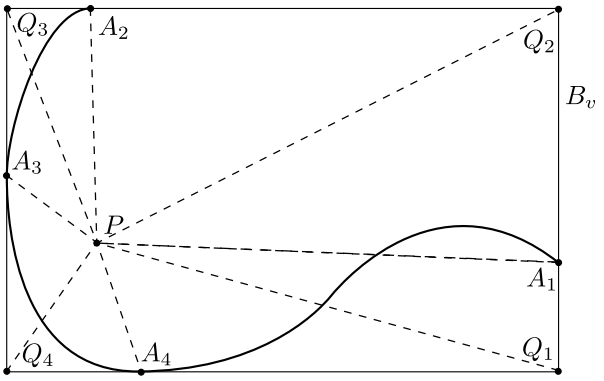


Figure 3: The curve γ_v and its point on the box boundary.

First, we consider a singular case: the point P coincides with a vertex Q_i. Then the points A_{i-1} and A_i on the adjacent sides can be chosen: so that $\angle A_{i-1} P A_i = \frac{\pi}{2}$, and the inequality (5) is true.

It is easy to see that for $P \in B_v$ and arbitrary i, the following is true:

$$\angle Q_{i-1} P Q_i + \angle Q_i P Q_{i+1} \geq \frac{\pi}{2}. \tag{6}$$

In the general case, the existence of the sought pair of points is proved by contradiction. Suppose that the converse is true for each i and j: $\angle A_i P A_j < \frac{\pi}{2}$. Then all the four points A_1, A_2, A_3, A_4 are seen from P under an angle less than $\frac{\pi}{2}$. The vertices Q_k separate the consecutive points A_{k-1} and A_k. Hence, for some k, we have three vertices Q_{k-1}, Q_k, Q_{k+1} which are seen from P under an angle less than $\frac{\pi}{2}$. But this contradicts the proved inequality (6).

Thus the inequality (5) must be true for some i and j. □

It is worth noting that if the function `EdgeBoundingBox` returns only tight axis-aligned bounding boxes, then B_v is also a tight bounding box of the curve γ_v for each node v.

Lemma 5.2. *Consider a balanced binary rooted tree T with n leaves and $X \subseteq V(T)$, which is an arbitrary subset of size $k \geq 1$ of its nodes. Denote the set of nodes on the path from the root to a node v by $P(v)$. Then*

$$\left| \bigcup_{v \in X} P(v) \right| = O\left(k \log \frac{n}{k} + k\right).$$

This lemma was stated and briefly proved in [6].

Now we return to the proof of Theorem 5.1.

Proof. Consider all the nodes v of the tree T for which the function `NodeAngle` is called. Denote the subtree consisting of these nodes by T_0. For any node $v \in V(T_0)$, it is clear from the definition of Algorithm 5 that

1. v is an internal node of T_0 (i.e. $v \in V(T_0) \setminus L(T_0)$) $\iff$ case 3 works for v.
2. v is a leaf of T_0 (i.e., $v \in L(T_0)$) $\iff$ case 1 or case 2 works for v.
3. v is a leaf of T (i.e., $v \in L(T)$) $\Longleftarrow$ case 2 works for v (i.e., $v \in V_{ea}$).

The cases are marked in Algorithm 5. V_{ea} is the set of vertices for which case 2 works.

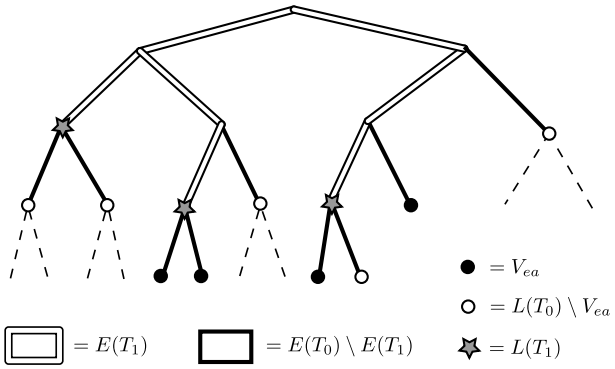


Figure 4: Example of the trees T, T_0, T_1 in the proof of Theorem 5.1.

We obtain the tree T_1 by removing all the leaves from the tree T_0. In other words, T_1 is a subtree of all the internal nodes of T_0. For internal nodes of T_0, case 3 of the algorithm works. Hence $P \in B_v$, and due to the Lemma 5.1

$$V^{\gamma_v}[\varphi] \geq \frac{\pi}{2}.$$

For any two incomparable nodes u and v of T, the corresponding curves γ_u and γ_v (parts of γ) do not overlap. In particular, the leaves of T_1 are incomparable, so the upper bound holds:

$$2\pi K = V^{\gamma}[\varphi] \geq \sum_{v \in L(T_1)} V^{\gamma_v}[\varphi] \geq \frac{\pi}{2}|L(T_1)|.$$

Then the number of leaves of the tree T_1 is bounded by

$$m = |L(T_1)| \leq 4K.$$

Also, $|L(T_1)| \leq |L(T)| = n$, since T_1 is a subtree of T. Hence, we have

$$m \leq \min(4K, n) = \hat{K}. \tag{7}$$

If $\hat{K} < 1$, then $|L(T_1)| = m = 0$, meaning that the tree T_1 is empty, and the tree T_0 consists only of the root node. Case 1 of the algorithm works for the root node

and takes $O(1)$ time. That proves the theorem for the case $\hat{K} < 1$. Hence, it can be assumed for the rest of the proof that T_1 is nonempty.

It is easy to see that the paths from all the leaves of T_1 to its root cover the whole tree T_1, i.e.,

$$\bigcup_{v \in L(T_1)} P(v) = V(T_1).$$

Hence, Lemma 5.2 can be applied to bound the size of T_1,

$$|V(T_1)| = O\left(m \log \frac{n}{m} + m\right).$$

The node sets $V(T_0)$ and $V(T_1)$ differ exactly by the set $L(T_0)$. Each leaf $v \in L(T_0)$ has exactly one father $f \in V(T_1)$. At the same time, each node $f \in V(T_1)$ can have no more than two sons $v \in L(T_0)$, so

$$\begin{aligned} |L(T_0)| \le 2|V(T_1)| \implies |V(T_0)| &= |V(T_1)| + |L(T_0)| \\ &\le 3|V(T_1)| = O\left(m \log \frac{n}{m} + m\right). \end{aligned} \tag{8}$$

The overall time complexity consists of the time to handle all the nodes $v \in V(T_0)$. It takes $O(\mathcal{T}_{ea})$ time to handle a node in case 2 of the algorithm and $O(1)$ in the other cases. The overall number of nodes of T_0 is already bounded. Denote the set of nodes for which the case 2 works as V_{ea}. To complete the proof, the size of this set has to be bounded.

Consider a node $v \in V_{ea} \subseteq (L(T) \cap L(T_0))$. Its father f belongs to $V(T_1)$. Denote the set of all such fathers by F. Clearly,

$$|V_{ea}| \le 2|F|.$$

If H is the height of the tree T, then the height of v is either H or $H-1$ (recall that T is balanced). Then the height of f is either $H-1$ or $H-2$. Since the height of T_1 is $H-1$, the node $f \in F$ is either its leaf or a father of two its leaves. In the second case, one of the sons cannot belong to F itself (because f is a father of some $v \in L(T)$). This means that an injection from F to $L(T_1)$ can be constructed, so $|F| \le |L(T_1)|$. Finally, the bound is established,

$$|V_{ea}| \le 2|F| \le 2|L(T_1)| = O(m). \tag{9}$$

Merge the bounds (8), (9), and (7) together. The overall time complexity of Algorithm 6 does not exceed

$$\begin{aligned} |V(T_0)| + |V_{ea}|\mathcal{T}_{ea} &= O\left(m \log \frac{n}{m} + m + m\mathcal{T}_{ea}\right) \\ &= O\left(\hat{K} \log \frac{n}{\hat{K}} + \hat{K}\mathcal{T}_{ea}\right). \end{aligned}$$

□

5.4 Complexity Corollaries

The complexity theorem just proved has some nice corollaries. We assume in this section that all the bounding boxes are tight.

Corollary 5.2. *The time complexity of Algorithm 6 does not exceed $O(n\mathcal{T}_{ea})$. Thus, the accelerated algorithm is not slower asymptotically that the trivial angle summation.*

Proof. Follows from $\hat{K} \le n$ and from the theorem. □

Corollary 5.3. *Let D be a star-shaped domain (around the point P) bounded by contour γ. Then Algorithm 6 works in $O(\mathcal{T}_{ea} + \log n)$ time.*

Proof. Since the curve γ bounds some domain,

$$D^\gamma[\varphi] = \pm 2\pi.$$

The domain is star-shaped around P, so the polar angle function $\varphi(t)$ is monotonic. Then its variation is equal to the absolute value of its difference:

$$K = \frac{1}{2\pi} V^\gamma[\varphi] = \frac{1}{2\pi} \left| D^\gamma[\varphi] \right| = 1.$$

Hence, $\hat{K} = O(1)$, and the theorem proves the rest. □

Corollary 5.4. *Let the curve γ bound a convex domain D. Then the time complexity of Algorithm 6 does not exceed $O(\mathcal{T}_{ea} + \log n)$.*

Proof. If the point P belongs to the domain D, then D is star-shaped around it, and this corollary follows from Corollary 5.3.

If the point lies outside, then the curve γ can be broken into two parts so that the polar angle function $\varphi(t)$ is monotonic on each of the parts. Due to the convexity of D and P lying outside of it, the curve γ is seen from P under angle less than π.

Then the total variation is less than 2π, hence $K < 1$. The proof is finished by applying the theorem. □

The time complexity bound in Theorem 5.1 greatly depends on the condition number K. As the condition number decreases, the time complexity also decreases. Intuitively, the condition number K can be often small in practice. Several upper bounds on the condition number (proved below) confirm that.

Proposition 5.3. *Let $D = \operatorname{dist}(P, \gamma)$ and L be the length of γ. Then,*

$$K \le \frac{1}{2\pi} \int_\gamma \frac{dl}{|z - P|} \le \frac{L}{2\pi D},$$

where z is the point moving through the curve γ, and dl is the length of the curve element.

Proof. Recall that polar angle function can be expressed in terms of a complex function (see (1)). Then its derivative is (points are treated as complex numbers)

$$\dot{\varphi}(t) = \operatorname{Im}\left(\frac{\dot{\gamma}(t)}{\gamma(t) - P} \right).$$

This representation trivially yields the bound

$$|\dot{\varphi}(t)|\, dt \le \frac{|\dot{\gamma}(t)|\, dt}{|\gamma(t) - P|} = \frac{dl}{|z - P|} \le \frac{dl}{D}.$$

Integrate over the curve γ to get the necessary inequality on K. □

The curve γ is assumed to be piecewise smooth regular from the very beginning.

Define a direction angle function $\alpha(t)$ equal to the polar angle of the tangent vector $\dot\gamma(t)$ for any $t \in [0,1]$. The polar angle is defined modulo 2π. To resolve the ambiguity in $\alpha(t)$, choose it in such a way that it is continuous on each individual edge, and its jumps between edges are minimal ($\pm\pi$ jumps can be resolved either way). For any such choice of $\alpha(t)$, the total variation is the same.

It is also possible to express the total variation of $\alpha(t)$ as

$$2\pi Q = V^\gamma[\alpha] = \sum_{i=0}^{n-1} V^{\gamma_v}[\alpha] + \sum_{i=0}^{n-1} (\dot\gamma(s_i + 0) \wedge \dot\gamma(e_{i-1} - 0)),$$

where $(u \wedge v)$ denotes the smallest angle between arbitrary vectors u and v. Strictly speaking, Q can even be infinite.

The value Q is remarkable because 1) it depends only on the curve γ (i.e., it does not depend on the point P), 2) intuitively, it should be small, especially for smooth curves, and 3) it is an upper bound for the condition number for closed curves (see below).

Theorem 5.2. *Let γ be a piecewise smooth regular closed curve. Then*

$$2\pi V^\gamma[\varphi] = K \le Q = 2\pi V^\gamma[\alpha].$$

See Appendix A for the proof.

5.5 Results

The accelerated algorithm described here was applied to several artificial point-in-polygon problems. The summary is presented in Table 1.

test name	n	Q	K	E	τ	$\frac{\tau}{E}$	$\frac{\tau}{K}$
CircleUniform:Center	65536	1.00	1.00	56.00	7	0.125	1.750
CircleUniform:Border	65536	1.00	1.00	56.00	33	0.589	8.250
CircleSkewed:Center	65536	1.00	1.00	56.00	63	1.125	15.750
Star:Center	256	71.73	1.00	24.00	7	0.292	1.750
Star:Worst	256	71.73	6.53	85.99	35	0.407	1.341
Koch:Center	49152	10923.00	4.97	224.18	10	0.045	0.503
Koch:Border	49152	10923.00	8.32	350.30	33	0.094	0.992
HandDrawn:Corner	179	22.44	3.10	47.79	12	0.251	0.967
HandDrawn:Center	179	22.44	4.67	60.94	28	0.459	1.498
HandDrawn:Worst	179	22.44	4.75	61.46	40	0.651	2.106

Table 1: Accelerated algorithm: some numbers

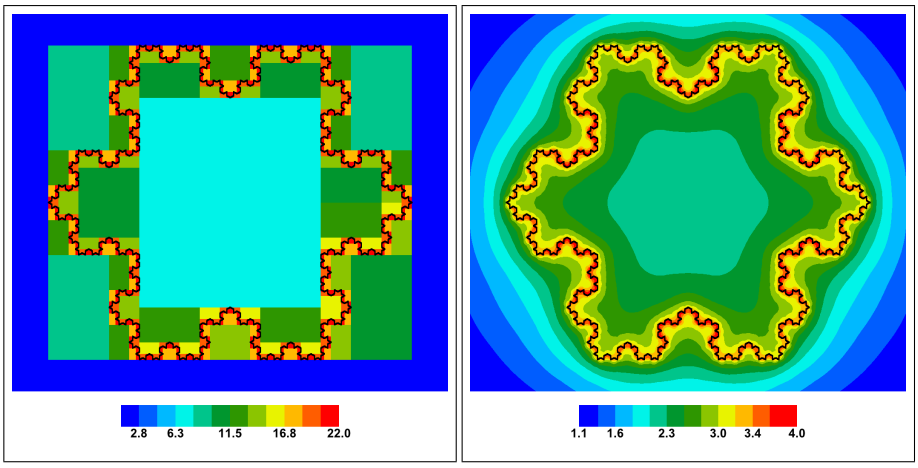
Here, $\tau = V(T_0)$ is the number of times the function `NodeAngle` was called (which is proportional to overall time complexity). $E = K \log_2 \frac{n}{K}$ denotes the upper bound on the number of `NodeAngle` calls according to Theorem 5.1[12]. K is the condition number (see Definition 4.1), Q is the scaled absolute turning angle (see Theorem 5.2).

[12] up to a constant factor

The number of edges is denoted by n. The last two columns show the estimation for constant factors in the asymptotic complexities $O\left(K \log \frac{n}{K}\right)$ (from the theorem) and $O(K)$ (supposed).

Five different polygons are used. "CircleUniform" is a circle uniformly tesselated into edges. "CircleSkewed" is also a circle, but tesselated in a special way: two edges are so large that their bounding boxes contain the center. "Star" is a star-shaped polygon, "Koch" is a finite generation of the Koch snowflake fractal. "HandDrawn" is some average sized contour drawn by hand. The second part of the test name indicates the point being tested. "Worst" is a point with maximal time complexity, "Border" is a point very close to the polygon edges, "Corner" is a point near the upper-left corner of the polygon, and "Center" is obvious.

Figures 5, 6, and 7 show how the time complexity and condition number depend on the point being tested. The contours used are the same as in Table 1, although the number of edges is sometimes smaller.



(a) time complexity τ **(b)** condition number K

Figure 5: "Koch" polygon ($n = 1024$)

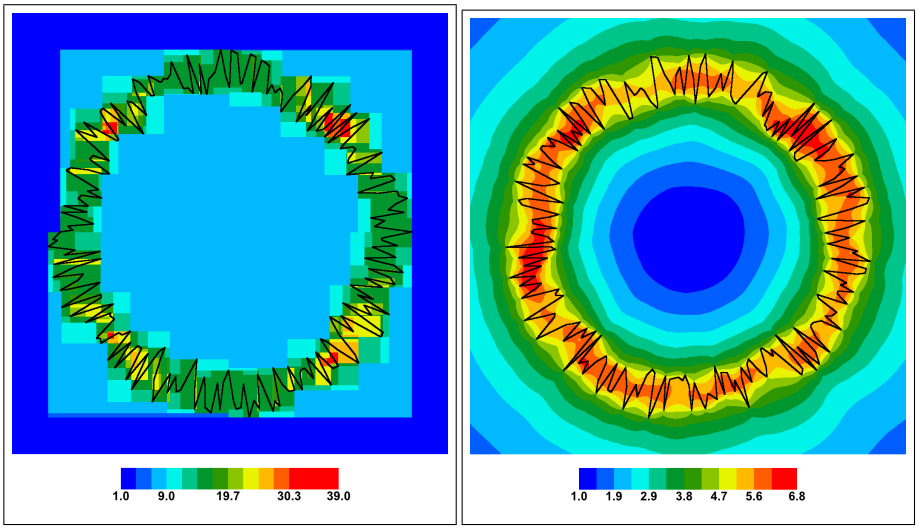
6 Application to NURBS Curves

NURBS curves form a class of parametric curves which includes arbitrary piecewise-polynomial splines and also some rational ones. For an introduction to NURBS curves, see the classical book [10].

NURBS curves have several properties which permit easy application of the accelerated algorithm:

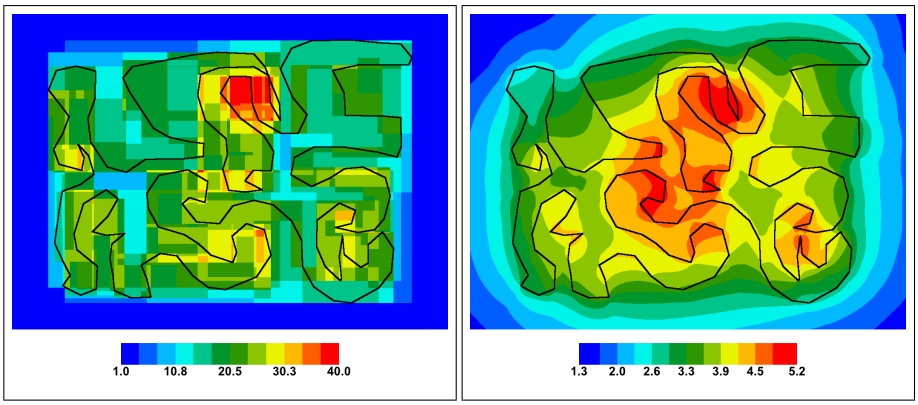
1. The bounding box can be efficiently (over)estimated from control points of the curve (due to the convex hull property).
2. A curve can be efficiently split into two curves at any given parameter value (by a knot insertion algorithm).
3. Its endpoints are equal to the first and the last of the control points[13].

[13] true only for clamped NURBS curves



(a) time complexity τ **(b)** condition number K

Figure 6: "Star" polygon ($n = 256$)



(a) time complexity τ **(b)** condition number K

Figure 7: "HandDrawn" polygon ($n = 179$)

Suppose that a NURBS curve γ is implicitly divided into 2^H pieces uniformly by a parameter, where H is some large number. The accelerated algorithm can be launched by treating each piece as a separate edge of the contour. However, it requires expensive precomputation and much additional memory to construct a BVH tree from the bottom up. Instead of precomputing the BVH tree, it can be calculated on the fly during its traversal from top to bottom.

Since there is no reliable implementation of the `EdgeAngle` function, we remove the case 2 from Algorithm 6 and subdivide the pieces until case 1 works for all of them. As a result, the algorithm sets $H = \infty$. Some other stopping criteria can be added, for instance, terminate the algorithm with an error, if the size of B_v is too small.

Algorithm 7: `NURBSAngle`

Input: NURBS curve $\gamma : [s, e] \to \mathbb{R}^2$, a point $P \notin \gamma$
Output: winding angle $D^\gamma[\varphi]$
$B :=$ `NURBSBoundingBox`(γ)
if $P \notin B$ **then**
  return `LineSegmentAngle` $(\gamma(s), \gamma(e), P)$
end
if $|B| < \delta$ **then**
  terminate("Distance smaller than δ")
end
$\gamma_L, \gamma_R :=$ `SplitNURBSCurve` $(\gamma, \frac{s+e}{2})$
return `NURBSAngle` (γ_L, P) + `NURBSAngle` (γ_R, P)

The function `SplitNURBSCurve` splits a curve at a given parameter into two pieces. The function `LineSegmentAngle` was implemented by Algorithm 2. Algorithm 7 is not new; Klein [8] contains exactly the same algorithm. A method based on NURBS subdivision for ray intersection approach appeared earlier in [9].

7 Conclusion

The winding angle summation method has been studied in detail. It appears that backward stability is its strongest attribute. The key property that makes backward stability possible is the stability under edge perturbations. Some error and correctness bounds for the trivial algorithm are given.

The introduced condition number K is important for the winding angle method. This number is not dependent on the tesselation of the contour and also is invariant under rotations. It is understood to be important also for the randomized ray intersection method.

The winding angle calculation can be accelerated easily by precomputing a bounding volume hierarchy. Unlike the trivial algorithm, the accelerated algorithm requires choosing a single tolerance value for a practical implementation. It is explained how this value can be chosen.

The main result of the paper is the complexity theorem for the BVH-accelerated algorithm, which bounds the time complexity asymptotically depending on the condition number K. This theorem is true only if all the bounding boxes are tight[14]. In

[14] and, strictly speaking, only in exact arithmetic

some simple cases, the optimal $O(\log n)$ time complexity is achieved. In particular, this it true if the polygon is star-shaped around the point being tested or convex.

It is interesting to compare time complexity of the BVH-accelerated winding angle summation with that of the ray intersection algorithm. Suppose that ray intersection shoots a ray in random direction which intersects a contour in k points. Then it is necessary to trace paths from the root to at least k leaves in the BVH. These paths consist of $O(k \log \frac{N}{k})$ nodes in the worst case (although this number can be smaller for some sets of leaves). Now recall that condition number K is exactly the average of the number of intersection points k. This non-strict reasoning suggests that BVH-accelerated ray intersection algorithm has similar or higher asymptotic complexity. However, it does not mean that winding angle summation is really faster than ray intersection in the BVH-accelerated setting. It is still very likely that the inverse trigonometric functions take most of the time in the angle summation.

Several upper bounds on the condition number K have been proved. The most fascinating one is related to the scaled absolute turning angle. Just like the condition number K, it does not depend on the tesselation, but additionally it does not depend on the point being tested.

The winding angle summation approach is still very easy to implement, like ray intersection. An additional desirable feature is the ability to check that the calculated winding number is close to an integer. If the winding number is relatively far from integer, it is possible to print a diagnostic message saying that the contour may be incorrect. The winding angle approach is used in the Russian Geometric Kernel, which is being developed[15].

There are several directions for future work. First of all, the BVH-accelerated algorithm seems to often work in $O(K)$ time without the logarithmic factor (see Table 1). Perhaps this holds when the tesselation of the contour into edges is uniform, in some sense. Second, the winding angle summation is resistant to errors in winding angle calculation if they are not too high. Then it is possible to construct a valid PIP algorithm that calculates winding angles for curvilinear edges by adaptive numerical integration. Of course, the integration error must be kept under control, so the curves cannot be pure black-boxes. There must be some way to extract information about global behavior of the curve (interval analysis should be helpful here).

A Absolute Turning Angle Inequality

The purpose of this appendix is to prove Theorem 5.2. The polar angle function $\varphi(t)$ and the direction angle function were defined in Section 2 and Section 5.4. It is assumed for simplicity that the point P is located at the origin O.

Define the function $d(t)$ as

$$d(t) = \frac{\alpha(t) - \varphi(t)}{\pi}.$$

Lemma A.1. *Let γ be a smooth regular curve, $P \notin \gamma$. Then*

1. $\dot{\varphi}(t) = 0 \iff d(t) \in \mathbb{Z}$,
2. $\dot{\varphi}(t) > 0 \iff \exists z \in \mathbb{Z} \quad (2z < d(t) < 2z + 1)$, *and*

[15]Russian Geometric Kernel is a geometric modeling kernel being developed in Russia according to the governmental program "National Technological Base": "Developing Russian Licensable Software – a Mathematical Kernel for 3D-Modeling as a Basis of Computer Systems for Computer-Aided Design of Complex Engineering Products".

3. $\dot\varphi(t) < 0 \iff \exists z \in \mathbb{Z} \quad (2z+1 < d(t) < 2z+2)$.

Proof. Consider how transformation to polar coordinates affects derivatives ($P = O$):

$$\begin{pmatrix}\dot x\\ \dot y\end{pmatrix} = \begin{pmatrix}\cos\varphi & -r\sin\varphi\\ \sin\varphi & r\cos\varphi\end{pmatrix}\begin{pmatrix}\dot r\\ \dot\varphi\end{pmatrix}$$

$$\begin{pmatrix}\dot r\\ \dot\varphi\end{pmatrix} = \begin{pmatrix}\cos\varphi & \sin\varphi\\ -\frac{1}{r}\sin\varphi & \frac{1}{r}\cos\varphi\end{pmatrix}\begin{pmatrix}\dot x\\ \dot y\end{pmatrix}.$$

Simplifying $\dot\varphi(t)$, we get

$$\begin{aligned}\dot\varphi &= \frac{1}{r}(\dot y\cos\varphi - \dot x\sin\varphi)\\ &= \frac{\|\dot\gamma\|}{r}(\sin\alpha\cos\varphi - \cos\alpha\sin\varphi)\\ &= \frac{\|\dot\gamma\|}{r}\sin(\alpha-\varphi) = \frac{\|\dot\gamma\|}{r}\sin(\pi d).\end{aligned}$$

The curve γ is regular, so the sign of $\dot\varphi$ is equal to the sign of $\sin(\pi d)$. This is precisely what the statement says. □

First, the theorem is proved for a subset of smooth curves. Let us call $t \in [0,1]$ a critical point if and only if $\dot\varphi(t) = 0$.

Theorem A.1. *Let γ be a smooth curve, $P \notin \gamma$. Assume that γ has finite number of critical points. Then $K \le Q$.*

Proof. Without loss of generality, we assume that

1. critical point parameters are $0 = \tau_0 < \tau_1 < \tau_2 < \ldots < \tau_{k-1} < \tau_k = 1$;
2. $\varphi(0) = \alpha(0) = 0$, $\varphi(1) = 2\pi W_\varphi$, $\alpha(1) = 2\pi W_\alpha$ ($W_\varphi, W_\alpha \in \mathbb{Z}$).

It is always possible to transform the curve γ into such a curve by rotations and by shifting and reversing the parameter space. The only exceptional case when it is not possible is when γ has no critical points ($k = 0$), which will be handled later.

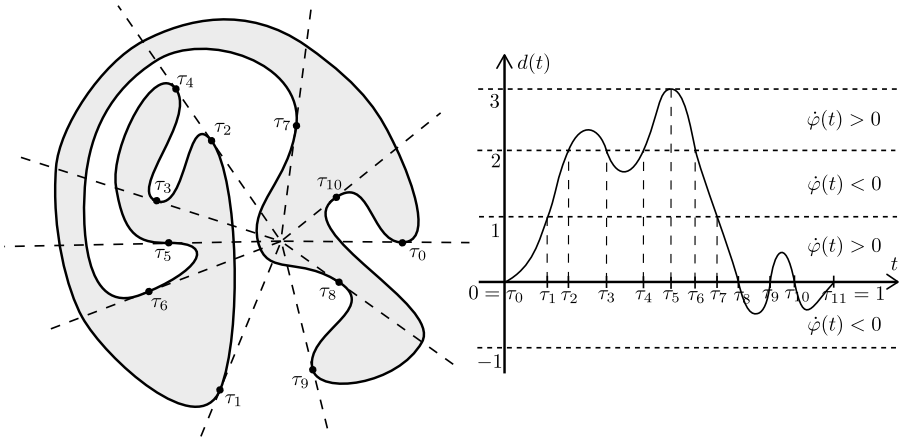


Figure 8: An example of a contour with its critical points and the function $d(t)$.

Lemma A.1 helps us understand the behavior of the function $d(t)$. Let $q_i = d(\tau_i)$. The lemma implies that $q_i \in \mathbb{Z}$. Hence, at any critical point $t = \tau_i$, the angles $\varphi(t)$ and $\alpha(t)$ differ by a multiple of π. This means that the position vector and the tangent vector lie on a common line. In the case of even $d(\tau_i)$, the vectors are codirectional, and otherwise they have opposite directions.

The lemma shows how $d(t)$ changes on the interval of monotonicity (τ_i, τ_{i+1}). The function $d(t)$ cannot take integer values on this interval because $\dot\varphi(t) \neq 0$. Hence, either $d(t) \in (q_i - 1, q_i)$ or $d(t) \in (q_i, q_i + 1)$, for all $t \in (\tau_i, \tau_{i+1})$. Hence, $q_{i+1} - q_i$ is one of $-1, 0, 1$.

Now it is possible to compare the total variations of the functions $\varphi(t)$ and $\alpha(t)$ over (τ_i, τ_{i+1}). Since φ is monotonic, its total variation is simple to calculate exactly:

$$V_{\tau_i}^{\tau_{i+1}}[\varphi] = \int_{\tau_i}^{\tau_{i+1}} |\dot\varphi(t)|\,dt = \left|\int_{\tau_i}^{\tau_{i+1}} \dot\varphi(t)\,dt\right| = |\varphi(\tau_{i+1}) - \varphi(\tau_i)| = \delta\varphi_i.$$

It suffices to bound the total variation of $\alpha(t)$ from below,

$$V_{\tau_i}^{\tau_{i+1}}[\alpha] \geq |\alpha(\tau_{i+1}) - \alpha(\tau_i)| = \delta\alpha_i.$$

Then we express $\alpha(t)$ in terms of $\varphi(t)$ and $d(t)$,

$$\delta\alpha_i = |\varphi(\tau_{i+1}) - \varphi(\tau_i) + \pi(q_{i+1} - q_i)|. \tag{10}$$

It is necessary to classify all the monotonicity intervals into three types:

1. Assume $q_i = q_{i+1}$.
   It immediately implies
   $$\delta\alpha_i = \delta\varphi_i.$$
2. Assume $q_i \neq q_{i+1}$, and q_i is even.
   We want to show that $q_{i+1} - q_i$ and $\varphi(\tau_{i+1}) - \varphi(\tau_i)$ have the same sign.
   Consider the case of $\varphi(t)$ increasing on (τ_i, τ_{i+1}). Then $\dot\varphi(t) > 0$ on the interval; hence, $d(t) \in (2z, 2z + 1)$ due to Lemma A.1. Since q_i is even and q_{i+1} is odd, q_i must be less than q_{i+1}: $q_{i+1} = q_i + 1$.
   The case of decreasing $\varphi(t)$ is similar: $\dot\varphi(t) < 0$, so $d(t) \in (2z + 1, 2z + 2)$. Since q_i is even, it must be greater than q_{i+1}: $q_{i+1} = q_i - 1$.
   The absolute value can be transformed because the signs are the same,
   $$\delta\alpha_i = |\varphi(\tau_{i+1}) - \varphi(\tau_i)| + \pi|q_{i+1} - q_i| = \delta\varphi_i + \pi.$$
3. Assume $q_i \neq q_{i+1}$ and q_i is odd.
   It is possible to show that $q_{i+1} - q_i$ and $\varphi(\tau_{i+1}) - \varphi(\tau_i)$ have opposite signs in such a case, though it is not necessary. The triangle inequality yields
   $$\delta\alpha_i \geq |\varphi(\tau_{i+1}) - \varphi(\tau_i)| - \pi|q_{i+1} - q_i| = \delta\varphi_i - \pi.$$

The inequality for the overall variations is

$$\begin{aligned} V[\alpha] &= \sum_{i=0}^{k-1} V_{\tau_i}^{\tau_{i+1}}[\alpha] \geq \sum_{i=0}^{k-1} \delta\alpha_i \\ &\geq \sum_{i=0}^{k-1} \delta\varphi_i + \pi(\#_{even} - \#_{odd}) \\ &= \sum_{i=0}^{k-1} \delta\varphi_i = \sum_{i=0}^{k-1} V_{\tau_i}^{\tau_{i+1}}[\varphi] = V[\varphi], \end{aligned}$$

where $\#_{even}$ and $\#_{odd}$ are the numbers of intervals of the types 2 and 3, respectively. It is easy to see that these two types alternate with each other because the parity of q_i changes after any of them. Also, the overall number of intervals of these two types must be even because $q_k - q_0 = d(1) - d(0) = 2(W_\alpha - W_\varphi)$ is even. Hence, the numbers $\#_{even}$ and $\#_{odd}$ must be equal.

This finishes the proof for the general case $k \geq 1$.

Now we consider the case $k = 0$ (no critical points). In this case, $\dot{\varphi}(t) \neq 0$ everywhere, so $d(t)$ is not integer-valued everywhere due to Lemma A.1. Then $d(1) = d(0)$ because the curve γ is closed. The formulas similar to (10) can be produced:

$$V[\alpha] \geq |\alpha(1) - \alpha(0)| = |(\varphi(1) - \varphi(0)) + \pi(d(1) - d(0))| = V[\varphi]. \qquad \square$$

It is possible to approximate an arbitrary smooth regular contour with a series of piecewise-polynomial smooth contours, each with finite number of critical points. If done correctly, the theorem is generalized to the smooth regular case.

If the contour is smooth regular everywhere except for one break with the angle different from π, then it can be approximated by a series of smooth regular contours by substituting an elliptic arc of diminishing size for the neighbourhood of the break. After such simple break is handled, the theorem can be extended to breaks with angle π and to any finite number of breaks.

Strict proofs of these steps are highly technical, and it is not worth to include them in this paper.

Acknowledgements

The author wishes to thank the referee for comments, which helped to improve readability of this paper.

References

[1] Herbert Edelsbrunner, Lionidas J Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.

[2] Eric Haines. Point in polygon strategies. In *Graphics gems IV*, pages 24–46. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[3] Nicholas J. Higham. The accuracy of floating point summation. *SIAM J. Sci. Comput*, 14:783–799, 1993.

[4] Kai Hormann and Alexander Agathos. The point in polygon problem for arbitrary polygons. *Comput. Geom. Theory Appl.*, 20(3):131–144, 2001.

[5] *Pentium Family Users Manual*, 1994. Appendix G.

[6] Witold Lipski Jr. and Franco P. Preparata. Finding the contour of a union of iso-oriented rectangles. *J. Algorithms*, 1(3):235–246, 1980.

[7] David Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12:28–35, 1983.

[8] Fritz Klein. A new approach to point membership classification in B-rep solids. In *Proceedings of the 13th IMA International Conference on Mathematics of Surfaces XIII*, pages 235–250, Berlin, Heidelberg, 2009. Springer-Verlag.

[9] Tomoyuki Nishita, Thomas W. Sederberg, and Masanori Kakimoto. Ray tracing trimmed rational surface patches. *SIGGRAPH Comput. Graph.*, 24(4):337–345, 1990.

[10] Les Piegl and Wayne Tiller. *The NURBS Book (2nd ed.)*. Springer-Verlag, New York, NY, USA, 1997.

[11] Neil Sarnak and Robert E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986.

[12] Stefan Schirra. How reliable are practical point-in-polygon strategies? In *Proceedings of the 16th Annual European Symposium on Algorithms*, ESA '08, pages 744–755, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] Shane Story, Ping Tak, and Peter Tang. New algorithms for improved transcendental functions on IA-64. In *IEEE Symposium on Computer Arithmetic*, pages 4–11, 1999.