

# A Novel Point Inclusion Test for Convex Polygons Based on Voronoi Tessellations<sup>\*</sup>

Rahman Salim Zengin<sup>a,1,\*</sup>, Volkan Sezer<sup>a,2</sup>

<sup>a</sup>*Istanbul Technical University, Istanbul, Turkey*

---

## Abstract

The point inclusion tests for polygons, in other words the point in polygon (PIP) algorithms are fundamental tools for many scientific fields related to computational geometry and they have been studied for a long time. PIP algorithms get direct or indirect geometric definition of a polygonal entity and validate its containment of a given point. The PIP algorithms which are working directly on the geometric entities derive linear boundary definitions for the edges of the polygon. Moreover, almost all direct test methods rely on the two point form of the line equation to partition the space into half-spaces. Voronoi tessellations use an alternate approach for half-space partitioning. Instead of line equation, distance comparison between generator points is used to accomplish the same task. Voronoi tessellations consist of convex polygons which are defined between generator points. Therefore, Voronoi tessellations have become an inspiration for us to develop a new approach of PIP testing specialized for convex polygons. Essential equations to the conversion of a convex polygon to a voronoi polygon are derived along this paper. As a reference, a very standard convex PIP testing algorithm, *the sign of offset*, is selected for comparison. For generalization of the comparisons *the ray crossing* algorithm is used as another reference. All algorithms are implemented as vector and matrix operations without any branching. This enabled us to benefit from the CPU optimizations of the underlying linear algebra libraries. All algorithms are tested for three different polygon sizes and varying point batch sizes. Overall, our proposed algorithm has performed better with varying margin between 10% to 23% comparing to the reference methods.

**Keywords:** point inclusion test, point in polygon, convex polygon, voronoi tessellations

---

## 1. Introduction

Various point inclusion tests [1] are used in many applications[2], including planning for autonomous driving [3], geographical information systems [4, 5, 6], and computer graphics [7, 8]. Any improvement on the efficiency of point inclusion tests will provide a direct benefit to the mentioned areas.

When autonomous driving related planning applications are considered, planning is mostly done

in a 2D space. Collected real time sensor data, especially Lidar based point cloud data is mapped to the 2D space where the planning operation is performed. Collision check is one of the most critical components of motion planning. Several simplifications on collected data and vehicle representation is required to make it efficient. Modeling the vehicle as a circle or combination of several circles is one of the widely used techniques for collision check. Although this simplification works well for most of the situations, there is always an accuracy problem depending on the number of circles that are used to model the vehicle [3].

In order to make a more accurate collision check, vertical footprint of the car can be modeled as a convex polygon in a suitable manner by simply finding a convex hull bounding the car. In order to make a real time motion planning, efficient collision

---

<sup>\*</sup>This work was supported by the Turkish Scientific and Technological Research Council (TUBITAK) under project no. 118E809.

<sup>\*</sup>Corresponding author

Email addresses: [rszengin@itu.edu.tr](mailto:rszengin@itu.edu.tr) (Rahman Salim Zengin ), [sezerv@itu.edu.tr](mailto:sezerv@itu.edu.tr) (Volkan Sezer )

<sup>1</sup>ORCID: <https://orcid.org/0000-0002-3104-4677>

<sup>2</sup>ORCID: <https://orcid.org/0000-0001-9658-2153>

check algorithms that are able to test big batches of points against the convex polygon model of the car are needed. Even though there are simple and well known algorithms, we propose an alternative algorithm based on Voronoi approach to accomplish the same task.

Geographical information systems [4, 5, 6] are another field that relies on point inclusion tests. It is used to process large databases of cartographic data. Measurements taken in the field must be matched with the prior information related to the area. For that reason, point inclusion tests are run on big databases using the measurements.

Another field in which the point in polygon queries are being actively used is, computer graphics [7, 8]. A scene is composed of many many object models which are composed of polygons. For visualization on the screen, proper rasterization of the geometric data is needed. To match the pixels on the screen with the geometric data according to the defined camera model, polygons are mapped to the screen plane. Then membership of every screen pixel is determined via point inclusion testing, so that pixels can be painted properly. Geometric model of the object which is subject to point inclusion testing is mostly known priorly and not changing. Considering this fact, instead of using polygonal model of the object directly, a preprocessed and simpler to use equivalent definition of the object can be utilized.

Voronoi tessellations are simple, understandable and well known. They consist of convex polygons and there is a huge literature related to voronoi tessellations. Simplest point inclusion tests are based on line equations and point to line distance calculations. Conversion of a convex polygon to a voronoi polygon has the advantage of using only point to point distance calculations. Required equations for the conversion of a convex polygon to a voronoi polygon are derived step by step throughout this paper. These derivations are directly used for the implementation of the proposed algorithm.

For completeness, we summarize two simple and well known point inclusion approaches, and then used these as a comparison to our proposed method. In order to compare the algorithms in an equal manner, all algorithms are implemented, using vector and matrix operations instead of simple loops, with the help of the related libraries. In this way, computations are handled more efficiently.

For the relatively smaller size of point batches such as 1024 points, due to the conversion over-

head our proposed algorithm gives the worst results. However, as processed batch size increases our algorithm takes the lead. Processing time of the new algorithm becomes 23% better for 5 edges, 13% better for 8 edges and 11% better for 11 edges.

The structure of the paper is as follows: In (Section 2) two reference algorithms are mentioned and the notation which is used throughout the paper is given. In (Section 3), conversion of a convex polygon to a voronoi polygon is described and required equations are derived. In (Section 4), the point inclusion testing procedure using the generators is described. In (Section 5), expected performance of our proposed algorithm is discussed. In (Section 6.1), for a certain generated test data our proposed algorithm is compared against the *sign of offset* algorithm to prove its correctness. In (Section 6.2), experimental setup is described, experimental results are shared and discussed.

## 2. Background

The use of line equations for linear boundaries are explained in this section via two standard point inclusion algorithms.

### 2.1. Ray crossing method

The *ray crossing* method [9, 1] is the golden standard of the point inclusion tests for polygons. It can be used for point inclusion testing of simple polygons. As can be seen on the (Figure 1) a ray directed to the  $+x$  direction is used to count crossings of the ray and the polygon. If the ray crosses the polygon edges in odd numbers, it is inside, otherwise it is outside.

All edges of the polygon are traversed and checked whether they are on the same  $y$  level of the point. If applicable, line equation in the two point form [10] is used to determine the half-plane the point is present. For a  $+x$  going ray it must be on the left half-plane of the line. If so, it is counted as a crossing.

The pseudocode of the branchless ray crossing implementation which is used for experiments is given in (Algorithm 1).

### 2.2. Sign of offset

The *sign of offset* [5, 1] method is the simplest point in polygon algorithm specialized for convex polygons. A point in a convex polygon given in (Figure 2). For subsequent vertices of the polygon,

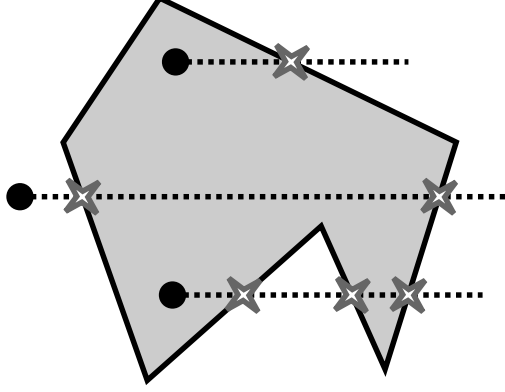


Figure 1: Ray crossing method

#### Function CrossingInclusion

**Data:**

//  $Q^i$ : Vertices,  $(2 \times n)$

$Q^i \leftarrow [q_1 \ \dots \ q_n]$

//  $X$ : Points,  $(2 \times m)$

$X \leftarrow [x_1 \ \dots \ x_m]$

**Result:**  $IsIn$ : Boolean,  $(m,)$

**begin**

//  $Q^j$ : Rolled vertices,  $(2 \times n)$

$Q^j \leftarrow [q_n, q_1 \ \dots \ q_{n-1}]$

// Edges in Y range,  $(n \times m)$

$InRange \leftarrow (Q_1^i > X_1) \oplus (Q_1^j > X_1)$

// LHS & RHS of the line equation,  $(n \times m)$

$LHS \leftarrow (X_1 - Q_1^j) \circ (Q_0^i - Q_0^j)$

$RHS \leftarrow (X_0 - Q_0^j) \circ (Q_1^i - Q_1^j)$

// Is edge going up,  $(n,)$

$Up \leftarrow Q_1^i > Q_1^j$

// Is point on the left,  $(n \times m)$

$OnLeft \leftarrow Up ? (LHS > RHS) :$

$(RHS > LHS)$

$Crossing \leftarrow InRange \wedge OnLeft$

$IsIn \leftarrow Mod_2(\sum_i Crossing_{ij}) = 0$

**end**

**Algorithm 1:** Ray crossing point inclusion test

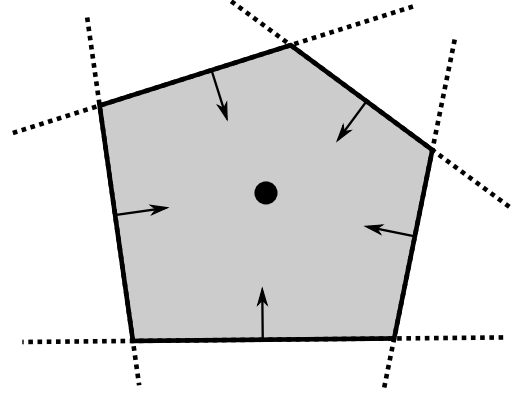


Figure 2: Sign of offset method

the offset of the point from the line passing through the edge is calculated using the two point form of line equation. If the offset of the point has the same sign for all edges of the polygon, the point is inside. Otherwise, if the two subsequent edges give different signs for the offset of the point, it is outside.

The pseudocode of the implemented branchless variant of the algorithm is given in (Algorithm 2).

#### Function SignOfOffsetInclusion

**Data:**

//  $Q^i$ : Vertices,  $(2 \times n)$

$Q^i \leftarrow [q_1 \ \dots \ q_n]$

//  $X$ : Points,  $(2 \times m)$

$X \leftarrow [x_1 \ \dots \ x_m]$

**Result:**  $IsIn$ : Boolean,  $(m,)$

**begin**

//  $Q^j$ : Rolled vertices,  $(2 \times n)$

$Q^j \leftarrow [q_n, q_1 \ \dots \ q_{n-1}]$

// LHS & RHS of the line equation,  $(n \times m)$

$LHS \leftarrow (X_1 - Q_1^j) \circ (Q_0^i - Q_0^j)$

$RHS \leftarrow (X_0 - Q_0^j) \circ (Q_1^i - Q_1^j)$

// Sign test,  $(n \times m)$

$D \leftarrow LHS < RHS$

// Are all same sign

$IsIn \leftarrow Mod_n(\sum_i D_{ij}) = 0$

**end**

**Algorithm 2:** Sign of offset point inclusion test

### 2.3. Notation

For simplicity and clearance, definitions related to voronoi tessellations [11] are slightly modified

and adapted.

Throughout this paper, only 2-dimensional euclidean space,  $R^2$  is considered. Boldface denotes a vector, such as  $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)^T$ . Superscript  $T$  denotes transpose as usual. For a polygon which has  $n$  vertices, vertices of the polygon are denoted with additional indexes, such as  $\mathbf{q}_i, \mathbf{q}_j$ , where  $i, j = \{1, \dots, n\}$  and  $i \neq j$  where edges considered. The set of vertices of the voronoi polygon is  $Q = \{\mathbf{q}_1, \dots, \mathbf{q}_n\}$ .

A voronoi polygon is a convex region defined by an inner generator point and some outer generator points such as,

$$V(\mathbf{p}_0) = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{p}_0\| \leq \|\mathbf{x} - \mathbf{p}_k\| \quad \forall k \in \{1, \dots, n\}\} \quad (1)$$

where  $V(\mathbf{p}_0)$  denotes voronoi polygon related to the generator point  $\mathbf{p}_0$ .

A generator point  $\mathbf{p}_k$  belongs to the set of generator points  $P$  of the voronoi polygon. Inner generator point is always indexed as  $\mathbf{p}_0$  independent of the polygonal edge count  $n$ . For every edge of the voronoi polygon there is an outer generator point so that the set of generator points is  $P = \{\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n\}$ .

Edges are equidistant set of points between inner generator and outer generators. Precisely,

$$e_k = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{p}_0\| = \|\mathbf{x} - \mathbf{p}_k\|\} \quad (2)$$

where  $k \in \{1, \dots, n\}$ . The set of edges of the  $V(\mathbf{p}_0)$  can be denoted as  $E = \{e_1, \dots, e_n\}$ .

The whole set of edges of the voronoi polygon is called boundary and it is denoted related to the inner generator point of the region encircled as  $\partial V(\mathbf{p}_0)$ . Although a voronoi graph has multiple polygonal regions, throughout this study, we are only interested in defining a single voronoi polygon.

For two subsequent vertices  $\mathbf{q}_i, \mathbf{q}_j$  of a polygon, two point form of the line equation [10] can be written as

$$(x_2 - q_{i2})(q_{j1} - q_{i1}) = (x_1 - q_{i1})(q_{j2} - q_{i2}) \quad (3)$$

where the point,  $\mathbf{x} = (x_1, x_2)^T$ .

### 3. Conversion of convex polygons to voronoi polygons

Vertices of a convex polygon ( $\mathbf{q}_i$  on Figure 3) can be taken as the vertices of a voronoi polygon, and

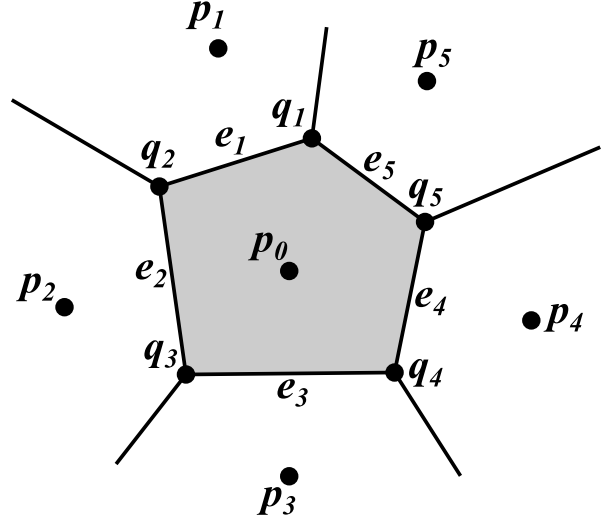


Figure 3: Generators, vertices and edges of a voronoi polygon

edges of a convex polygon ( $e_k$  on Figure 3) can be taken as the boundary of a voronoi polygon.

Because determination of generator points ( $\mathbf{p}_k$  on Figure 3) is only constrained by  $\partial V(\mathbf{p}_0)$ , it is free to choose any internal point as  $\mathbf{p}_0$ . But to distribute outer generators homogeneously and to preserve symmetry, when the polygon is symmetric, and to have a guaranteed point inside, the centroid of the polygon is used as the inner point. The centroid of a convex object always lies inside, because any line passing through the centroid can only cross the edges of the convex polygon at two points. Then, outer generator points can be found accordingly.

As can be seen on (Figure 3) placement of generator points determines not only  $\partial V(\mathbf{p}_0)$  but also the edges going to the infinity between outer generator points. However, our problem is only constrained on  $\partial V(\mathbf{p}_0)$ .

Having  $n$  edges or lines passing through  $n$  couples of vertices constrained on  $(n + 1)$  generator points, gives us freedom of choosing one of the generator points. Although setting any of the generator points sets all the others, setting the inner generator is more reasonable, because all the edges are defined depending upon it.

The centroid of a polygon [12] can be calculated as follows:

Let  $Q$  be a cyclically ordered set of polygon vertices and  $\mathbf{q}_i, \mathbf{q}_j$  are subsequent vertices accordingly.

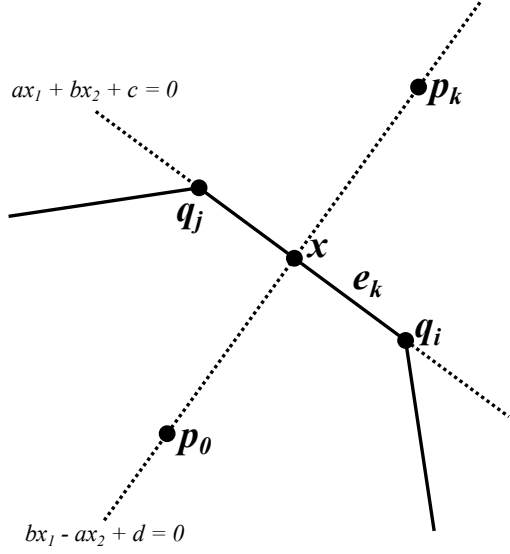


Figure 4: Finding outer generator of an edge

Summation over  $Q$ ,

$$A = \frac{1}{2} \sum_Q \det[\mathbf{q}_i \mathbf{q}_j] \quad (4)$$

$$\mathbf{p}_0 = \frac{1}{6A} \sum_Q (\mathbf{q}_i + \mathbf{q}_j) \det[\mathbf{q}_i \mathbf{q}_j] \quad (5)$$

gives the area (4) and centroid (5) of the polygon. The pseudocode of the centroid calculation is given in (Algorithm 3).

**Function** CalculateCentroid

**Data:**

//  $Q^i$ : Vertices,  $(2 \times n)$

$Q^i \leftarrow [\mathbf{q}_1 \ \cdots \ \mathbf{q}_n]$

**Result:**  $\mu$ : Centroid,  $(2,)$

**begin**

//  $Q^j$ : Rolled vertices,  $(2 \times n)$

$Q^j \leftarrow [\mathbf{q}_n, \mathbf{q}_1 \ \cdots \ \mathbf{q}_{n-1}]$

//  $A$ : Partial areas,  $(n,)$

$A \leftarrow Q_0^j \circ Q_1^i - Q_0^i \circ Q_1^j$

$a \leftarrow \frac{1}{2} \sum A$  //  $a$ : Area,  $(1,)$

$\mu \leftarrow ((Q^i + Q^j)A)/(6a)$

**end**

**Algorithm 3:** Calculation of the centroid

Standard form equation of the line passing through an edge can be derived from two point form equation.  $\mathbf{q}_i$  and  $\mathbf{q}_j$  are two vertices of the edge  $e_k$ .  $\mathbf{x}$  is a point of the edge. As defined in (2)  $\mathbf{p}_0$  and

$\mathbf{p}_k$  are two points, equidistant to the  $e_k$ . The line passing through  $\mathbf{p}_0$  and  $\mathbf{p}_k$  is perpendicular to (3).

Solving  $\mathbf{x}$  for two equations gives

$$\mathbf{x} = \frac{\begin{pmatrix} b_k^2 & -a_k b_k \\ -a_k b_k & a_k^2 \end{pmatrix} \mathbf{p}_0 - c_k \begin{pmatrix} a_k \\ b_k \end{pmatrix}}{a_k^2 + b_k^2} \quad (6)$$

where

$$a_k = q_{i2} - q_{j2}$$

$$b_k = q_{j1} - q_{i1}$$

$$c_k = -(a_k q_{i1} + b_k q_{i2})$$

$\mathbf{p}_0$  and  $\mathbf{p}_k$  are equidistant to  $\mathbf{x}$ . Writing this equation and leaving  $\mathbf{p}_k$  alone on the left hand side gives  $\mathbf{p}_k$  as

$$\begin{aligned} \mathbf{p}_k - \mathbf{x} &= \mathbf{x} - \mathbf{p}_0 \\ \Rightarrow \mathbf{p}_k &= 2\mathbf{x} - \mathbf{p}_0 \end{aligned} \quad (7)$$

By substituting  $\mathbf{x}$  into (7), outer generator points can be found as

$$\mathbf{p}_k = \frac{\begin{pmatrix} b_k^2 - a_k^2 & -2a_k b_k \\ -2a_k b_k & a_k^2 - b_k^2 \end{pmatrix} \mathbf{p}_0 - 2c_k \begin{pmatrix} a_k \\ b_k \end{pmatrix}}{a_k^2 + b_k^2} \quad (8)$$

The generator calculation procedure is given in (Algorithm 4).

#### 4. Point inclusion test via generator points

After the set of generators  $P$  has been found, the point inclusion test is simply testing the condition provided in (1).

The ordinary distance metric for the definition of the voronoi polygon is *eucledian distance* or equivalently *L2 norm*. To test the inclusion of a random point, its distances to all generators are calculated. If it is closest to the generator  $\mathbf{p}_0$  it is inside of the polygon. Otherwise it is outside of the polygon.

Ordinarily, calculating the *L2 norm* of a vector (9) takes squaring, summing and then square rooting of the vector components.

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2} \quad (9)$$

However squaring of both sides of (1) does not change the order of distances because squaring is a monotonic operation.

$$V(p_0) = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{p}_0\|_2^2 \leq \|\mathbf{x} - \mathbf{p}_k\|_2^2\} \quad (10)$$

#### Function CalculateGenerators

**Data:**  
//  $Q^i$ : Vertices,  $(2 \times n)$   
 $Q^i \leftarrow [q_1 \ \cdots \ q_n]$   
**Result:**  $P$ : Generators,  $(2 \times (n + 1))$   
**begin**  
     $P_{,0} \leftarrow \text{CalculateCentroid}(Q^i)$   
    //  $Q^j$ : Rolled vertices,  $(2 \times n)$   
     $Q^j \leftarrow [q_n, q_1 \ \cdots \ q_{n-1}]$   
     $a \leftarrow Q_1^i - Q_1^j$  //  $(n,)$   
     $b \leftarrow Q_0^j - Q_0^i$  //  $(n,)$   
     $c = -(a \circ Q_0^i + b \circ Q_1^i)$  //  $(n,)$   
    //  $W$ :  $(2 \times 2 \times n)$   
     $W \leftarrow \begin{bmatrix} b^2 - a^2 & -2ab \\ -2ab & a^2 - b^2 \end{bmatrix}$   
     $d \leftarrow \sum_j W_{ijk} P_{j0}$  //  $(2 \times n)$   
     $e \leftarrow -2c \circ \begin{bmatrix} a \\ b \end{bmatrix}$  //  $(2 \times n)$   
     $P_{,1:(n+1)} \leftarrow (d + e) \odot (a^2 + b^2)$   
**end**

**Algorithm 4:** Calculation of generators

#### Function VoronoiInclusion

**Data:**  
//  $Q$ : Vertices,  $(2 \times n)$   
 $Q \leftarrow [q_1 \ \cdots \ q_n]$   
//  $X$ : Points,  $(2 \times m)$   
 $X \leftarrow [x_1 \ \cdots \ x_m]$   
**Result:**  $IsIn$ : Boolean,  $(m,)$   
**begin**  
    //  $P$ : Generators,  $(2 \times (n + 1))$   
     $P \leftarrow \text{CalculateGenerators}(Q)$   
    //  $\Delta$ : Differences,  $(2 \times (n + 1) \times m)$   
     $\Delta \leftarrow X - P$   
    //  $M$ : Metrics,  $((n + 1) \times m)$   
     $M_{jk} \leftarrow \sum_i \Delta_{ijk} \Delta_{ijk}$   
     $IsIn \leftarrow M_1 \leq M_k, \forall k \in \{2, \dots, (n + 1)\}$   
**end**

**Algorithm 5:** Voronoi point inclusion test

The square root and the square vanish, when these are applied together. Then equation (10) becomes

$$V(p_0) = \{x \mid (x - p_0)^T(x - p_0) \leq (x - p_k)^T(x - p_k)\} \quad (11)$$

The derived simplification (11) is an alternate way of distance comparison. It improves the performance of comparisons and preserves the order of distances.

The pseudocode of the proposed point inclusion testing algorithm is given in (Algorithm 5)

## 5. Algorithm analysis

The calculation of polygon centroid takes  $O(n)$  time when it is done sequentially. Similarly, outer generator point calculations have time complexity of  $O(n)$ . But considering Single Instruction Multiple Data (SIMD) capabilities of modern CPUs, for small sizes of  $n$  computations will be optimized to be done with time complexity of  $O(1)$ .

For  $n$  vertices and  $m$  points;  $(n + 1)m$  distance calculations are done. Then using distances to the inner centroid as a reference, the number of distance comparisons to be made is  $nm$ . Conversion related computations are done initially and are independent of the batch size of processed points. As the batch size  $m$  of processed points increases, the conversion cost becomes less effective on the overall computational cost.

In practice, for determination of the status of a point, doing all computations and comparisons is not always needed. If the point under test is found to be closer to an outer generator, this breaks the  $\forall$  condition of (1). An early break opportunity arises here for a sequential implementation of the algorithm.

## 6. Experimental results and discussion

### 6.1. Correctness

To test correctness of the proposed point inclusion algorithm, random test points are sampled (Figure 5) around the polygon. The set of generators for the tested polygon are also plotted.

Inclusion test results of *the sign of offset* algorithm are used as the known ground truth reference. For the same test set, both algorithms gave the same results. The correctness of the proposed

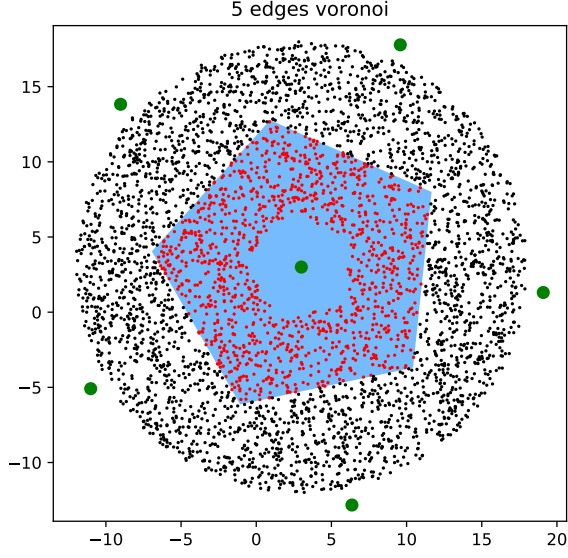


Figure 5: Correctness test of the proposed algorithm

algorithm is proved via this testing procedure. The correctness of the algorithm can be seen in (Figure 5) by looking at different coloring of the dots inside and outside.

## 6.2. Performance

In order to make a fair comparison, calculations are performed for all vertices, edges or generators etc. every time for each relevant method. Thus, experimental results reflect theoretical complexity analyses well.

The CPU used for the experimentation is Intel(R) Core(TM) i7-7700 running at 3.60GHz frequency. The system has 32G of RAM.

For ease of reproducibility, all implementations are done using Python [13] and related libraries [14, 15]. The source code [16] to reproduce the results is shared.

(Table 1) provides information about the processing time of each method for various types of convex polygons, using different number of test points. Per point processing times are calculated via dividing batch processing time into number of points in that batch. Best methods for each test scenario are shown bold. The difference column in the table is calculated between the best two results in a row according to the formula (12) below:

$$Difference\% = \frac{2ndBest - 1stBest}{2ndBest} \times 100 \quad (12)$$

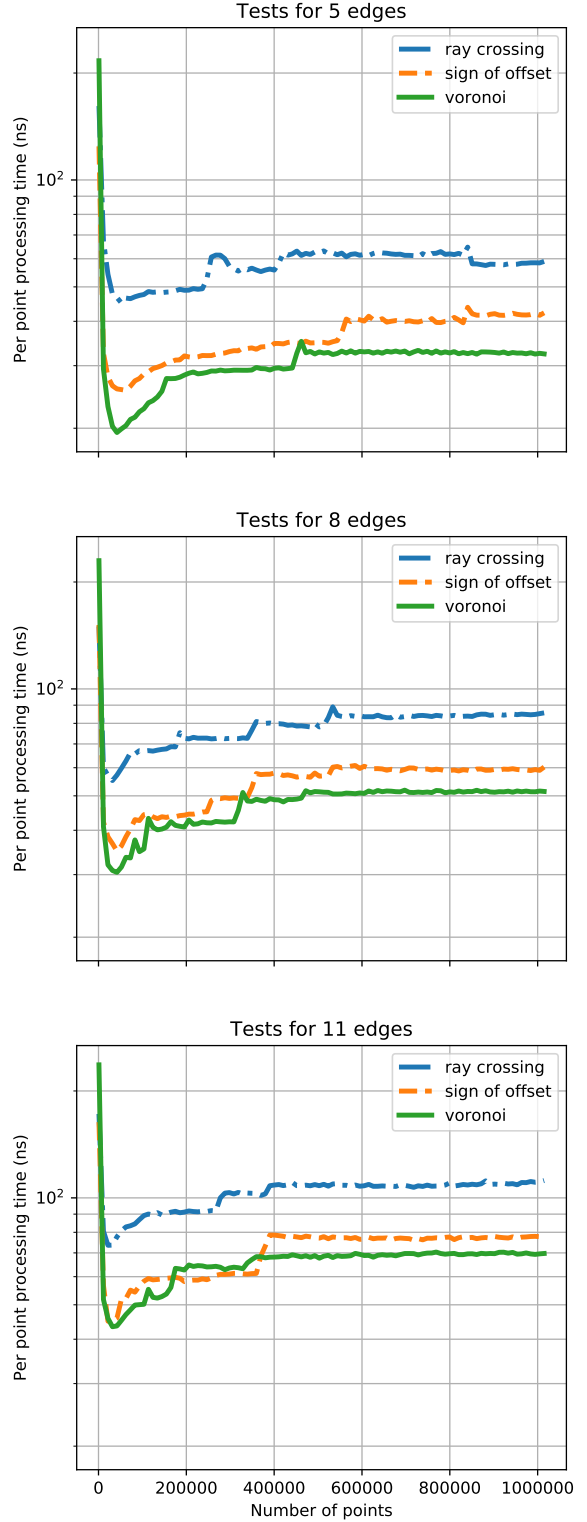


Figure 6: Tests for 5, 8, and 11 edges

Table 1: Per point processing times (ns)

Edges	Points	Algorithms			Diff %
		Ray Crossing	Sign of Offset	Voronoi	
5	1024	161.49	<b>124.34</b>	216.60	23.0
	164864	48.42	30.74	<b>27.58</b>	10.3
	338944	55.83	33.53	<b>29.12</b>	13.2
	513024	63.14	34.89	<b>32.55</b>	6.7
	676864	61.73	40.19	<b>32.99</b>	17.9
	850944	58.04	41.99	<b>32.68</b>	22.2
	1014784	58.98	42.26	<b>32.38</b>	23.4
8	1024	<b>149.98</b>	151.41	229.63	0.9
	164864	68.78	43.83	<b>42.29</b>	3.5
	338944	72.57	49.52	<b>48.35</b>	2.4
	513024	78.85	56.93	<b>51.19</b>	10.1
	676864	84.64	59.14	<b>51.37</b>	13.1
	850944	83.64	58.55	<b>51.38</b>	12.2
	1014784	85.63	60.25	<b>51.44</b>	14.6
11	1024	172.33	<b>163.43</b>	236.87	5.2
	164864	91.16	59.50	<b>55.95</b>	6.0
	338944	103.19	<b>61.00</b>	65.61	7.0
	513024	109.21	77.78	<b>68.53</b>	11.9
	676864	107.67	76.85	<b>68.69</b>	10.6
	850944	109.23	77.41	<b>69.60</b>	10.1
	1014784	111.89	78.48	<b>69.74</b>	11.1

Ray crossing is a more general algorithm which can test the inclusion of points for simple polygons. It is provided as one of the references here since it is the golden standard of point inclusion tests for benchmarking the point in polygon algorithms. As it is a more general and complicated approach, it is slower than the algorithms specialized for convex polygons.

On the other hand the sign of offset algorithm, which is specialized for convex polygons, gives best results for small problem sizes. Conversely, the proposed Voronoi based method is the slowest one for relatively smaller problem sizes since it has a constant conversion cost. On the other hand, with increasing problem size, per point cost of conversion decreases for the proposed method.

According to the numerical results given in the (Table 1), for small sizes of point batches such as 1024 points, due to the conversion overhead our proposed algorithm gives the worst results. However, as processed batch size increases our proposed algorithm takes the lead. Processing time of our algorithm becomes 23% better for 5 edges, 13% better for 8 edges and 11% better for 11 edges.

As it is illustrated in (Figure 6), the proposed algorithm gives best results for most of the batch sizes for all the three polygon sizes.

Abrupt changes of timings in (Figure 6) are result of changing memory transfer characteristics of three different algorithms. As problem size increases, temporary storage is shifted from the CPU cache to the system memory. As stepping up happens first with our proposed algorithm, this is a sign of higher temporary storage usage. And the same shift happens with the *ray crossing* following our algorithm and lastly the *sign of offset* algorithm because it has least temporary storage requirements.

## 7. Conclusion

A systematic approach to convert a convex polygon to a voronoi polygon is developed throughout this work. As a meaningful internal generator point selection scheme, centroid calculation of a polygon is chosen. The equations related to the centroid calculation are given consecutively. After that, the equations required to calculate outer generators in relation to the inner generator and the vertices of the convex polygon, are derived.

In order to demonstrate the advantages of our proposed algorithm, it is implemented as only vector and matrix operations without including any program branching. Reference algorithms are also implemented in a similar way. Certain tests are carried out to show that our proposed algorithm not only works properly but also it has a clear performance advantage over the reference algorithms for large size of point batches.

Conversion of a convex polygon to a voronoi polygon takes constant time according to the point batch size. It depends only on the number of edges of the polygon. If the geometry is known to be constant prior to the use, voronoi equivalent of the convex polygon can be calculated in advance. Both polygon vertices and generator points can be stored together in a database with only about  $2\times$  increase of the storage capacity needed.

Testing vertices of a convex polygon against generators of another one, collision detection between convex polygons can be accomplished. Performing this testing in a mutually inclusive way between two convex polygons reduces errors.

The purpose of this paper is showing applicability of generator points based boundary definition and determination to the point inclusion testing.



Besides, a novel algorithm is matched with prior knowledge in the field. Thus, the methods developed here can be extended to nonconvex polygons. The generator based boundary checks can be applied to prior point in polygon algorithms.

## Acknowledgements

This work was supported by the Turkish Scientific and Technological Research Council (TUBITAK) under project no. 118E809.

## References

- [1] E. Haines, Point in Polygon Strategies, in: Graphics Gems, Elsevier, 1994, pp. 24–46. doi:10.1016/B978-0-12-336156-1.50013-6.
- [2] D. Alciatore, R. Miranda, A winding number and point-in-polygon algorithm, Glaxo Virtual Anatomy Project Research Report, Department of Mechanical Engineering, Colorado State University (1995).
- [3] J. Ziegler, C. Stiller, Fast collision checking for intelligent vehicle motion planning, in: 2010 IEEE Intelligent Vehicles Symposium, 2010, pp. 518–522. doi:10.1109/IVS.2010.5547976.
- [4] T. K. Peucker, N. Chrisman, Cartographic Data Structures, The American Cartographer 2 (1) (1975) 55–69. doi:10.1559/152304075784447289.
- [5] S. Nordbeck, B. Rystedt, Computer cartography point-in-polygon programs, BIT Numerical Mathematics 7 (1) (1967) 39–64. doi:10.1007/BF01934125.
- [6] P. A. Longley, M. F. Goodchild, D. J. Maguire, D. W. Rhind (Eds.), Geographical Information Systems: Principles, Techniques, Management and Applications, 2nd Edition, Wiley, Hoboken, N.J, 2005.
- [7] E. Haines, FAST RAY – CONVEX POLYHEDRON INTERSECTION, Graphics Gems II (1991) 247–250 doi:10.1016/B978-0-08-050754-5.50053-0.
- [8] A. S. Glassner, An Introduction to Ray Tracing, Elsevier, 1989.
- [9] M. Shimrat, Algorithm 112: Position of point relative to polygon, Communications of the ACM 5 (8) (1962) 434. doi:10.1145/368637.368653.
- [10] E. W. Weisstein, "Two-Point Form." From MathWorld – A Wolfram Web Resource, <https://mathworld.wolfram.com/Two-PointForm.html>.
- [11] A. Okabe, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, 2nd Edition, Wiley Series in Probability and Statistics, Wiley, Chichester ; New York, 2000.
- [12] G. Bashein, P. R. Detmer, Centroid of a Polygon, in: Graphics Gems, Elsevier, 1994, pp. 3–6. doi:10.1016/B978-0-12-336156-1.50010-0.
- [13] G. van Rossum, Python tutorial, technical report CS-R9526, centrum voor wiskunde en informatica (CWI), amsterdam." (1995).
- [14] S. van der Walt, S. C. Colbert, G. Varoquaux, The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science Engineering 13 (2) (2011) 22–30. doi:10.1109/MCSE.2011.37.
- [15] J. D. Hunter, Matplotlib: A 2D Graphics Environment, Computing in Science Engineering 9 (3) (2007) 90–95. doi:10.1109/MCSE.2007.55.
- [16] R. S. Zengin, V. Sezer, Volimpro/voronoi\_pip: REL: V1.0, Zenodo (Jul. 2020). doi:10.5281/zenodo.3966996.