Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

On Point-inclusion Test in Convex Polygons and Polyhedrons

Mahdi Imanparast*[1] and Mehdi Kazemi Torbaghan†[2]

[1]Department of Computer Science, University of Bojnord, Bojnord, Iran.
[2]Department of Mathematics, University of Bojnord, Bojnord, Iran.

ABSTRACT

A new algorithm for point-inclusion test in convex polygons is introduced. The proposed algorithm answers the point-inclusion test in convex polygons in $\mathcal{O}(\log n)$ time without any preprocessing and with $\mathcal{O}(n)$ space. The proposed algorithm is extended to do the point-inclusion test in convex polyhedrons in three dimensional space. This algorithm can solve the point-inclusion test in convex $3D$ polyhedrons in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n)$ preprocessing time and $\mathcal{O}(n)$ space.

ARTICLE INFO

Article history:
Received 02, November 2019
Received in revised form 14, April 2020
Accepted 10 May 2020
Available online 01, June 2020

Keyword: Point-in-polygon, Point-inclusion test, Convex polygons, Convex polyhedrons, Preprocessing time.

AMS subject Classification: 68U05, 68W40, 97R60

1 Introduction

Point-in-polygon is a basic operation in geographical information systems (GIS), and has applications in areas that deal with geometrical data, such as computer graphics, computer vision, and motion planning [7, 9]. This problem is defined as: given a polygon P and a query point q, determine whether point q is inside P or not. For instance, we can imagine whenever we click with a mouse on a shape in the screen, a point-in-polygon test is solved [13]. This operation is also called point-inclusion test, and used

*Corresponding author: M. Imanparast. Email: m.imanparast@ub.ac.ir
†m.kazemi@ub.ac.ir

in several tasks such as ray tracing and point location. If the number of points is too high or the given polygon is constant, a preprocessing is done to increase the efficiency and speed of the test. In ray tracing and other applications, initial polygons are in 3D. It is good to project polygon and query point into 2D for simplifying the computation by ignoring one component. The best component for ignoring usually is the component when it ignores a polygon with larger area is obtained [4]. This can be done by taking absolute of the polygon plane normal components, and finding the largest one. Doing this work for a polygon may be taken more memory but it increases the intersection test speed. However, when we do not have any guarantee that the vertices of the polygon are on a plane, the projection may be useful [4, 19].
A method for reducing the query time is to use a bounding rectangle of the polygon. If point q is outside of this bounding rectangle, q can not be in P and the problem will be solved. By assuming that the probability of the point q be in P is $\frac{1}{2}$, the expected running time can be reduced to the half. Since the defining coordinates of the bounding rectangle of a polygon is a common item that is stored in a vector-based GIS systems, this operation does not use more than four boolean operations and it can be done before each point-in-polygon algorithm. Although, point-in-polygon problem does not seem such hard, but when the polygon is concave and has many vertices, an efficient and reliable algorithm is necessary [7, 9]. Several algorithms are proposed for point-in-polygon problem, but, some of them have disadvantages in implementation or running time. The efficiently of an algorithm can be evaluated by: the required query time to respond the answer of a query, required storage for the data structure, and the preprocessing time which is used to arrange the data for searching.

1.1 Backgrounds

In this subsection, we will present surveys on point-in-polygon algorithms and their complexity. In the following, we assume that the polygon is simple polygon, namely, the edges of the polygon only have intersection in its vertices. There are some algorithms for this problem in simple polygons, but two of them are very popular. One is counting ray crossing and the other is winding number, which we will explain them below. One of the earliest algorithm for point-in-polygon problem is the ray crossings test or even-odd rule. Using this method, we can determine in $\mathcal{O}(n)$ time without preprocessing, whether a point is inside a simple polygon. The earliest presentation of this algorithm is relevant to Shimrat [16], though it has a bug in it, corrected by Hacker [6]. Draw a line from query point to the infinity (usually, a ray parallel to the axes, say $+x$, is used), and count the number of intersections of this ray with the edges of the polygon. If this number is odd, query point is inside. In spite of this idea seems simple; its implementation is not easy. Ray may be hit on a vertex, or the query point q may be located on the boundary of P. In these cases, we want result q is in P, because P is closed. This algorithm is workable for both convex and concave polygons and vector-based images. Another algorithm is the winding number or sum of angles method. Assume, we stay in point q and look at the point p that moves through the boundary of P. If q be inside P, then, sum

of the angles is 2π, otherwise it is 0. This means that we use sum of angles between q and each two adjacent vertices of P. The angles may be positive, negative or zero. This algorithm is workable for both convex and concave polygons. Although, sum of angles is a pleasing algorithm but its dependency to floating point and trigonometric computation make it very slow [8, 13, 18]. There are other methods such as swath method [15], grid cell method [1, 20], and triangle fans methods (based on decomposition to triangles) for convex [2] and general polygons [3, 5]. All of these algorithms have also $\mathcal{O}(n)$ running time. For more details on complexity of these algorithms the reader is referred to [7, 9, 8]. Since in this paper, we focus on the algorithm for convex polygons. In the following, we review the algorithms available for inclusion in convex polygons. Sum of area method is an algorithm that is used only for convex polygons. This algorithm connects a line from query point q to vertices of the polygon and splits it to the triangles. If the sum of areas of triangles be equal to the area of the polygon, then q is inside. Clearly, this method needs $\mathcal{O}(n)$ time. Sign of the offset method [17] assumes that the vertices of the polygon P are listed in counterclockwise (CCW) order. If the distance between query point q and the edge V_iV_{i+1} has the same sign as the distance between V_{i+2} and edge V_iV_{i+1} for all edge of P, then point q is inside. This means that the point q is at the same side of any two consecutive edges of P. This algorithm is only valid for convex polygons. Orientation method [12] is another strategy which tests the point against each edge in turn. It uses an approach same as sign of offset method and only applies for convex polygons. If the point q be on the left of all edge of the polygon, then q is inside. Left-on test takes $\mathcal{O}(1)$ time, so, the algorithm needs $\mathcal{O}(n)$ time in overall. This method is better than the sum of area method, because it needs fewer memory and computation. This algorithm uses less additional storage in comparison with other methods, and it can be coded easily [7]. Preparata and Shamos [14] discussed a method which is called wedge method. The polygon is preprocessed by adding a central point c. For point c in the convex polygon, n wedge is formed with the lines that passed through c and the vertices of the polygon. Similar to the swath method, wedge which includes the point q is found in $\mathcal{O}(\log n)$ time with a binary search tree structure. Given the wedge, we need only compare q to the unique edge of P that cuts it, and we will find whether q is internal to P. See Figure 1. However, this algorithm has various limitations and tends to bog down when actually coded due to expensive operations. It is slow for polygons with few edges, because the initial cost is high, but the binary search makes it fast when the number of edges is high [7]. It can be seen, this is an algorithm that found in $\mathcal{O}(\log n)$ query time, whether a point q is inside a convex polygon or not. But it uses $\mathcal{O}(n)$ preprocessing time to create the wedges. So, it is an $\mathcal{O}(n)$-time algorithm in general. Finally, O'Rourk [13] showed that using a modified version of binary search one can find the extreme points of a convex polygon in $\mathcal{O}(\log n)$ time. In fact, he showed how the intersection of a line with a convex polygon can be solved by applying the extreme finding algorithm. Now, suppose we draw a horizontal line passing through query point q. In this case, by finding two intersection points of the horizontal line passing through query point q with a convex polygon sides, it is possible to determine whether point q is inside or outside the convex polygon. Thus, this algorithm that we call as stabbing method, can solve the problem

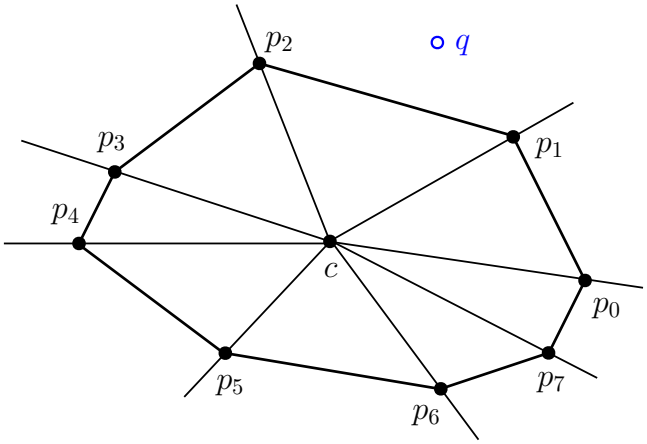

Figure 1: Division into wedges. By binary search we find the wedge that q lies in it. By comparing q against edge p_1p_2, we find that it is external.

Table 1: A summary on the complexity of point-in-polygon algorithms in convex polygons, where n is the number of the polygon vertices.

Algorithm	**Preprocess time**	**Query time**	**Storage**	**Polygon type**
Sum of area	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	convex
Sign of offset	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	convex
Orientation	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	convex
Wedge	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	convex
Stabbing	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	convex
Ours	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	convex

of point-inclusion test in a convex polygon in $\mathcal{O}(\log n)$ time by considering an $\mathcal{O}(\log n)$ preprocessing time to find the extreme points. Table 1 shows a summary of the complexity of point-in-polygon algorithms. The last row indicates on our proposed algorithm. In the next section, we present a new algorithm for the point-inclusion test problem in a convex polygon. The main advantages of the new algorithm over the previous algorithms are that it does not need to be preprocessed and can easily be expanded to higher dimensions and polyhedrons.

2 The proposed algorithm

In this section, we propose an $\mathcal{O}(\log n)$-time algorithm for convex polygon inclusion test without any preprocessing time. Let P be a convex polygon with n vertices $V_0, V_1, ..., V_n =$

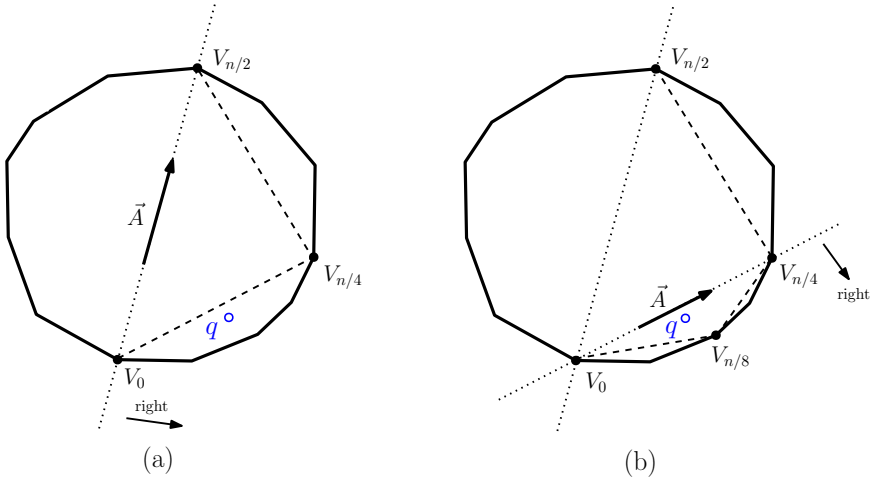

Figure 2: (a) Query point q is in the right side of the vector $V_0V_{n/2}$ (b) The proposed algorithm after two iterations detects that q is in triangle $\triangle V_0V_{n/8}V_{n/4}$ and so, it is in P.

V_0 in counterclockwise (CCW) order that are stored in an array. Note that, storing the vertices of a polygon in CCW order is a common way in most graphical library softwares. Let QR denotes the query range, and the initial QR be $[V_0, V_1, ..., V_{n-1}, V_n]$. First, we assume that $a = V_0$ and $b = V_{n/2}$, and compute a vector:

$$\vec{A} = \vec{ab} = V_{n/2} - V_0. \tag{1}$$

The line containing this vector splits vertices of the convex polygon into two sets. Moreover, this line partitions the plane into left half-plane ($\mathcal{LHP}$) and right half-plane ($\mathcal{RHP}$). We can determine in $\mathcal{O}(1)$ time, whether the query point q is in the right or left half-plane.

Then, we make a triangle by two vertices V_0, $V_{n/2}$ and another middle vertex between these two vertices. At the first iteration, if q be in right half-plane, the index of the middle vertex is $\lceil (0 + n/2)/2 \rceil = n/4$, and else, it is $\lceil (n/2 + n)/2 \rceil = 3n/4$. If the query point q be in this formed triangle, we stop the algorithm and else we update QR and vector $\vec{A}$ and follow this recursive process until we find the query point q in a triangle or finish the search, since it satisfies the stop conditions. Figure 2 illustrates an example.
A special case may be occur when the point q lies on the line that contains vector $\vec{A}$. But this problem has a simple solution, we should only check whether q is in segment $\vec{ab}$ or not. This can be done easily in $\mathcal{O}(1)$ time. Now, we can present this algorithm in more details as follows:

Algorithm 1: Point in Convex Polygon (P, q)

Input: A convex polygon P defined by its vertices $V_0, V_1, ..., V_n$ and $V_0 = V_n$ in CCW

order and a query point q.
Output: `YES`, if q be inside P, and `NO` otherwise.

1: $F \leftarrow False$;
2: $a \leftarrow 0$;
3: $b \leftarrow n/2$;
4: $c \leftarrow n$;
5: $\vec{A} \leftarrow V_b - V_a$;
6: **while** $(!F)$
7: **if** $q \in Line(\vec{A})$ **AND** $q \in Segment(\vec{ab})$ **then**
8: $F \leftarrow True$; **break**;
9: **if** $q \in Line(\vec{A})$ **AND** $q \notin Segment(\vec{ab})$ **then**
10: **break**;
11: **if** $q \in \mathcal{RHP}(\vec{A})$ **then**
12: $a \leftarrow a, b \leftarrow \lceil (a+b)/2 \rceil, c \leftarrow b$;
13: **else**
14: $a \leftarrow b, b \leftarrow \lceil (b+c)/2 \rceil, c \leftarrow c$;
15: **if** $(a+b)/2 = a$ **OR** $(b+c)/2 = b$ **then**// q is outside of P
16: **break**;
17: **if** $q \in \triangle abc$ **then**
18: $F \leftarrow True$;
19: **end while**
20: **if** $F = True$ **then**
21: output '`YES`';
22: **else**
23: output '`NO`';

In the following, we explain how the proposed algorithm solves the point-in-polygon problem in convex polygons in $\mathcal{O}(\log n)$ time. Let the initial QR be $[V_0, V_1, ..., V_{n-1}, V_n]$. Consider running the proposed algorithm on two convex polygons which are shown in Figure 3. For the convex polygon in Figure 3 (a), after three time iterations of the **while** loop, in line 15 of the algorithm, we reach in a situation that does not exist any triangle. This means that, we have three vertices V_a, V_b and V_c that they do not form a triangle, and they degenerate to a line. Hence, the algorithm has finished and reported that the point q is outside of the polygon. Similarly, for the case query point q be inside a convex polygon (Figure 3 (b)), in the final iteration of the **while** loop, we reach to a triangle which includes the query point q, and the algorithm returns YES. So, in each iteration of the **while** loop, this algorithm reduces QR to $1/2QR$ and follows the search in this new range. In fact, this is a geometric paradigm of the binary search. Therefore, if $T(n)$ be the running time of the algorithm, we have: $T(n) = T(n/2) + \mathcal{O}(1)$. This equation solves to $\mathcal{O}(\log n)$. Thus, the Algorithm 1 can report if a query point q is inside a convex polygon or not, in $\mathcal{O}(\log n)$ time at the worst case. It can also report the output in $\mathcal{O}(1)$ comparison in the best case, when the point q is inside the first formed triangle. In addition, Algorithm 1 only uses $\mathcal{O}(n)$ storage for storing polygon vertices and $\mathcal{O}(1)$ additional variables in the process of the algorithm.

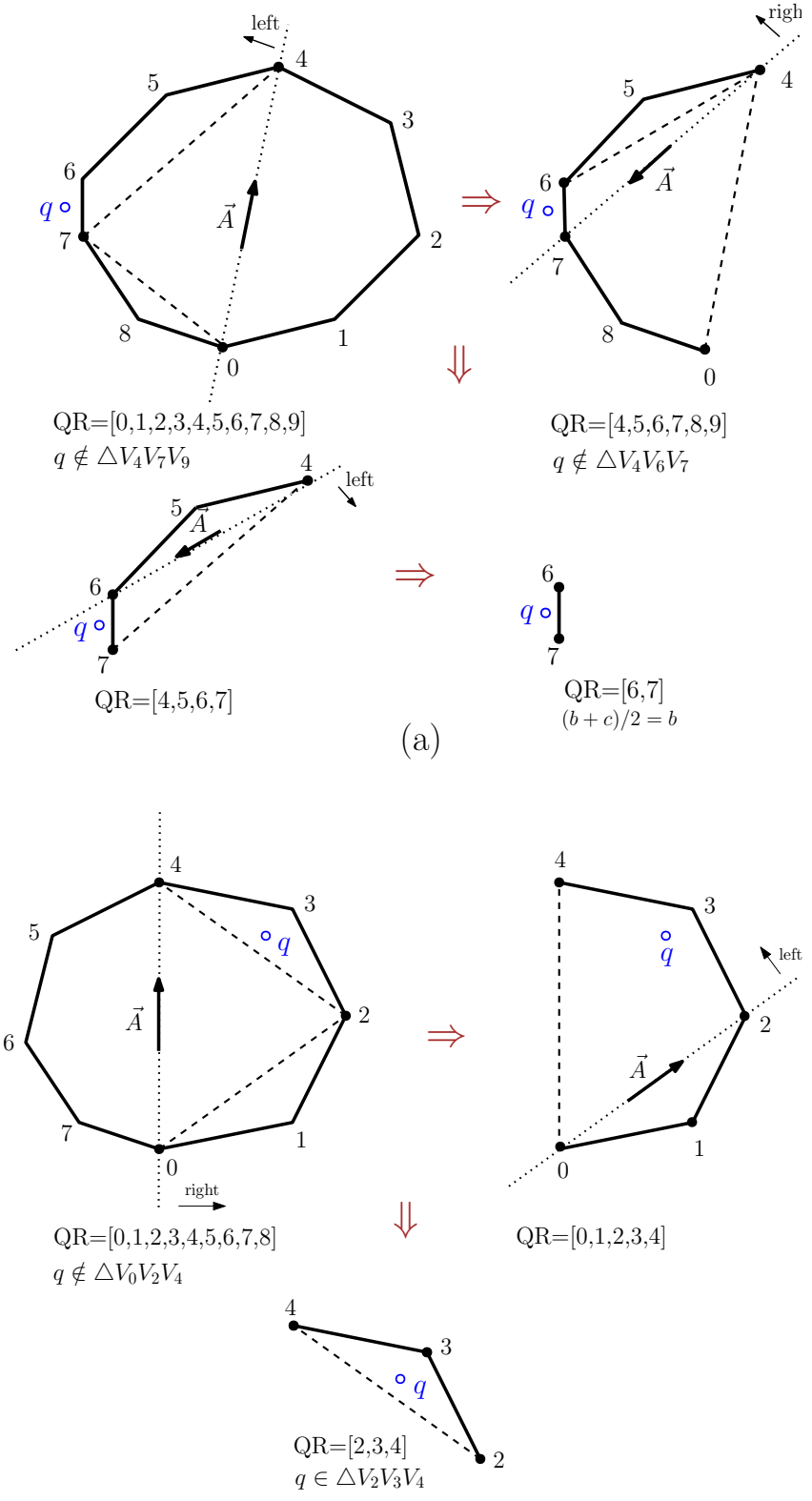

Figure 3: (a) Point q is outside of P, because the last test shows that: $(b+c)/2 = b$. This means that, there is not any triangle which contains q. (b) After three iterations of the **while** loop, Algorithm 1 finds that q is in the triangle $\triangle V_2V_3V_4$ and so, it is in P.

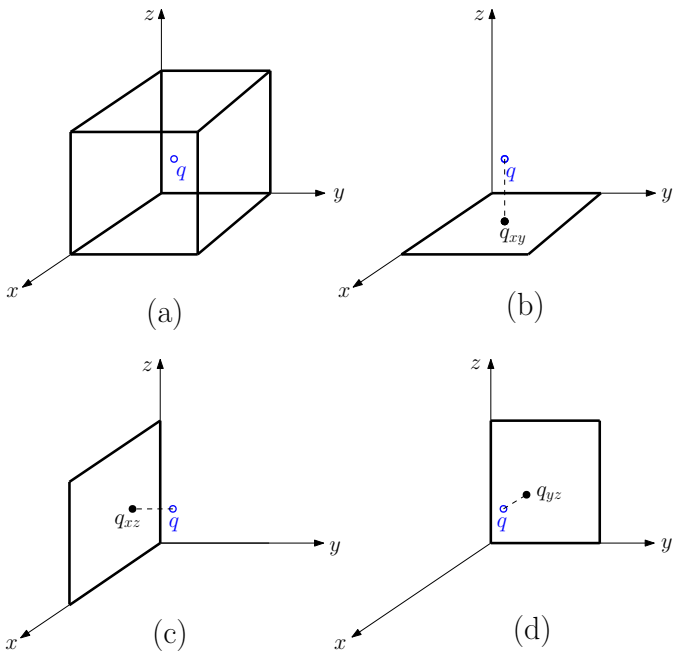

Figure 4: (a) Query point q is inside the cube P. (b) Project of the query point q and the polyhedron P on plane xy. (b) Project of the query point q and the polyhedron P on plane xz.(b) Project of the query point q and the polyhedron P on plane yz.

2.1 Inclusion test in 3D polyhedrons

Determining whether a point is inside a polyhedron has many application such as collision detection. This problem can be solved in a convex/ non-convex polyhedron similar to the two-dimensional one by generalizing the two methods of winding number and counting ray crossing [13], where they also need $\mathcal{O}(n)$ time.
In the following, we show that our proposed algorithm for convex polygons can also be extended to an $\mathcal{O}(\log n)$-time algorithm with $\mathcal{O}(n)$ preprocessing time for convex polyhedrons. The main step of the algorithm is to first project the polyhedron P and the query point q on three planes xy, xz and yz. In this case, if the point q is inside the polyhedron P, the projection of that point must be simultaneously in the projection of the polyhedron P on three planes xy, xz and yz. See Figure 4. After these three projection, we can do three point-inclusion test for the projection of the query point and the projection of the polyhedron on three planes xy, xz and yz, by using the Algorithm 1. Details of the proposed algorithm for point-inclusion test in a three dimensional polyhedron are presented below.

Algorithm 2: Point in Convex Polyhedron (P, q)

Input: A convex polyhedron P and a query point q.

Output: `YES`, if q be inside P, and `NO` otherwise.
1: $[P_{xy}, q_{xy}] \leftarrow$ Project of polyhedron P and query point q on plane xy;
2: $[P_{xz}, q_{xz}] \leftarrow$ Project of polyhedron P and query point q on plane xz;
3: $[P_{yz}, q_{yz}] \leftarrow$ Project of polyhedron P and query point q on plane yz;
4: **if** $((q_{xy} \in P_{xy}) = True)$ **AND** $((q_{xz} \in P_{xz}) = True)$ **AND** $((q_{yz} \in P_{yz}) = True)$ **then**
5: output '`YES`';
6: **else**
7: output '`NO`';

Since projecting the query point and polyhedron can be done at a constant time, and storing convex hull of the polygons which are projected on three planes in CW or CCW order takes $\mathcal{O}(n)$ time according to the proposed algorithm due to Melkman [11]. Therefore, we need $\mathcal{O}(n)$ preprocessing time, for projecting the polyhedron and storing three new convex polygons. According to $\mathcal{O}(\log n)$ time, we need to check the inclusion test for projected query point and polygon on each plane by Algorithm 1. The total time of the algorithm will be $\mathcal{O}(\log n)$ with $\mathcal{O}(n)$ preprocessing time.
Note that the proposed algorithm is much more efficient than an $\mathcal{O}(n)$ time algorithm for polyhedrons. Because, if the structure of a three-dimensional polyhedron in memory is fixed, we only need one time to apply the $\mathcal{O}(n)$ preprocessing time for projecting the polyhedron on three planes. After that, every time we need to answer a query on that polyhedron, it only need $\mathcal{O}(\log n)$ time, while in an algorithm with $\mathcal{O}(n)$ time, the algorithm requires $\mathcal{O}(n)$ time for answering to each query.

3 Conclusion

In this paper, we have presented a simple algorithm for point-inclusion test problem in convex polygons. The proposed algorithm is an $\mathcal{O}(\log n)$-time algorithm for convex polygon inclusion test. The proposed algorithm does not use any preprocessing or complicated data structure, and only needs $\mathcal{O}(n)$ storage space. We also extend our point-inclusion test algorithm to an $\mathcal{O}(\log n)$-time algorithm with $\mathcal{O}(n)$ preprocessing time for convex polyhedrons in three dimensional space.

References

[1] Antonio, F., Faster Line Segment Intersection. Graphics Gems III (David Kirk, ed.), Academic Press, (1992), 199–202.

[2] Badouel, D., An Efficient Ray-Polygon Intersection. Graphics Gems I, Academic Press, (1990), 390–393.

[3] Berlin, J. E. P., Efficiency Considerations in Image Synthesis. SIGGRAPH '85 course notes, 11, (1985).

[4] Glassner, A. S. (editor), An Introduction to Ray Tracing. Academic Press, (1989).

[5] Green, C., Worley, S., Simple, Fast Triangle Intersection. Ray Tracing News, 6(1), (1993).

[6] Hacker, R., Certification of Algorithm 112, Position of Point Relative to Polygon. Communications of the ACM, 5(12), (1962), 606.

[7] Haines, E., Point in Polygon Strategies. Graphics Gems IV (Paul Heckbert,ed.), Academic Press, (1994), 24–46.

[8] Hormann, K., Agathos, A., The point in polygon problem for arbitrary polygons. Computational Geometry: Theory and Applications, 20(3), (2001), 131–144.

[9] Huang, C. W., Shih, T. Y., On the complexity of point-in-polygon algorithms. Computers and Geosciences, 23(1), (1997), 9–18.

[10] Jimenez, J. J., Feito, F. R., Segura, R. J., Robust and Optimized Algorithms for the Point-in-Polygon Inclusion Test without Pre-processing. Computer Graphics Forum, 28(8), (2009), 2264–2274.

[11] Melkman, A., On-line construction of the convex hull of a simple polygon, Information Processing Letters, 25, (1987), 11–12

[12] Nordbeck, S., Rystedt, B., Computer cartography point-in-polygon programs. BIT Numerical Mathematics, 7(1), (1967), 39–64.

[13] O'Rourk, J., Computational Geometry in C. 2nd Edition, Cambridge University Press, (1998), 239–242.

[14] Preparata, F. P., Shamos, M. I., Computational Geometry: an Introduction. New York, Springer-Verlag, (1985), 41–67

[15] Salomon, K. B., An efficient point-in-polygon algorithm. Computers and Geosciences, 4(2), (1978), 173–178.

[16] Shimrat, M., Algorithm 112, Position of Point Relative to Polygon. Communications of the ACM, 5(8), (1962), 434.

[17] Taylor, G., Point in polygon test. Survey Review, 32, (1994), 479–484.

[18] Weiler, K., An incremental angle point in polygon test. Graphics Gems IV (Paul Heckbert, ed.), Academic Press, (1994), 16–23.

[19] Woo, A., Ray tracing polygons using spatial subdivision. In proceedings of Graphics Interface'92, (1992), 184–191.

[20] Zalik, B., Kolingerova, I., A cell-based point-in-polygon algorithm suitable for large sets of points. Computer and Geosciences, 23(1), (1997), 109–118.