A New Quick Point Location Algorithm

José Poveda, Michael Gould, Arlindo Oliveira*

Departamento de Lenguajes y Sistemas Informáticos
Universitat Jaume I, Castellón, Spain
albalade@uji.es, gould@uji.es

*Departamento de Engenharia Informática. Instituto Superior Técnico
Universidade Técnica de Lisboa, Lisboa, Portugal
aml@inesc-id.pt

Abstract. We present a new quick algorithm for the solution of the well-known point location problem and for the more specific problem of point-in-polygon determination. Previous approaches to this problem are presented in the first sections of this paper. In the remainder of the paper, we present a new quick location algorithm based on a quaternary partition of the space, as well as its associated cost and data structures.

1 Introduction

In this paper we present a new quick algorithm for the general solution of the point location problem[1] and the specific solution of the point-in-polygon problem [1] on which we will focus our attention. Some of the most efficient solutions for the point-in-polygon problem reduce the solution to that of other fundamental problems in computational geometry, such as computing the triangulation of a polygon or computing a trapezoidal partition of a polygon to solve, then, in an efficient way, the point-location problem for that trapezoidal partition. Two different methods for solving the point-in-polygon problem have become popular: counting ray-crossings and computing "winding" numbers. Both algorithms lead to solutions with a less-than-attractive cost of O(n), however the first one is significantly better than the second [2]. An implementation comparison by Haines [3] shows the second to be more than twenty times slower.

Methods of polygon triangulation include greedy algorithms [2], convex hull differences by Tor and Middleditch [4] and horizontal decompositions [5]. Regarding solutions for the point-location problem, there is a long history in computational geometry. Early results are surveyed by Preparata and Shamos [1]. Of all the methods suggested for the problem, four basically different approaches lead to optimal O(log(n)) search time, and O(n) storage solutions. These are the *chain method* by Edelsbrunner et al. [6], which is based on segment trees and fractional cascading, the

[1] The well-known point-location problem could be more appropriately called point-inclusion problem.

triangulation refinement method by Kirkpatrick [7], and the *randomised incremental method* by Mulmuley [8]. Recent research has focused on dynamic point location, where the subdivision can be modified by adding and deleting edges. A survey of dynamic point location is given by Chiang and Tamassia [9]. Extension beyond two dimensions of the point-location problem is still an open research topic.

2 Point Location Problems in Simple Cases

Several algorithms are available to determine whether a point lies within a polygon or not. One of the more universal algorithms is counting ray crossings, which is suitable for any simple polygons with or without holes. The basic idea of the counting ray crossings algorithm is as follows:

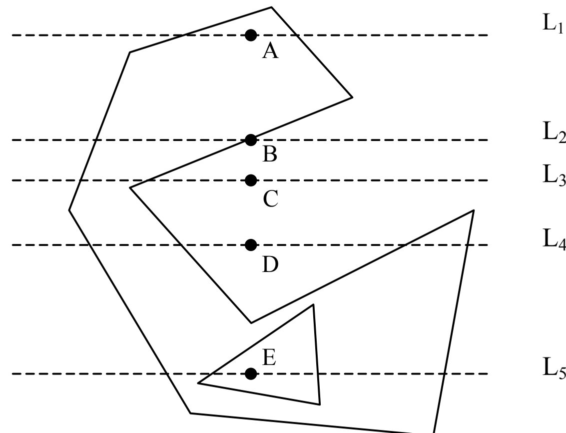

Fig. 1. Ray-crossings algorithm

Let L be the scan-line crossing the given point, Lcross is the number of left-edge ray crossings and Rcross is the number of right-edge ray crossings. First, we need to determine whether the given point is a vertex of the polygon or not; if not, we calculate Lcross and Rcross. If the parities of Lcross and Rcross are different (Fig.1, case B) then this point lies on the edge of the polygon. Otherwise, if Lcross (or Rcross) is an odd number, (Fig.1 case A) the point must lie inside the polygon. If Lcross (or Rcross) is an even number (Fig 1, cases C, D, E) the point must lie outside the polygon.

When the polygon is degraded to a triangle, the above-mentioned problem can be solved by the following algorithm shown in Fig. 2. We assume the three vertices A, B and C are oriented counterclockwise. If point P lies inside ABC, then it must lie to the left of all three vectors AB, BC and CA. Otherwise, p must lie outside ABC. This algorithm may also be applied to any convex polygon, in principle without holes, if the vertices of the polygon are ordered counterclockwise or clockwise.

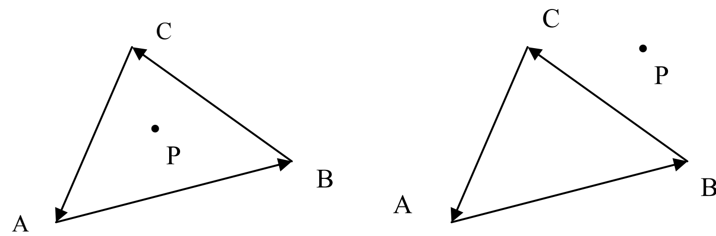

Fig. 2. Conditions for the location of P. (a) P lies to the left of all vectors AB, BC an CD. (b) P lies to the right of vector BC

An equivalent algorithm is based on the inner product among the vectors defined by the viewpoint of the query point and the normal vectors of the edges. This algorithm can be used for convex polygons, and extended for the degenerate case of convex polygons with holes in its interior. The main criterion of this algorithm is based on the analysis of the sign of the inner products among the normal vectors of the edges of the polygon, with the unitary vector whose direction is determined by the query point and any point of the edge (normally the middle point of the edge is taken).

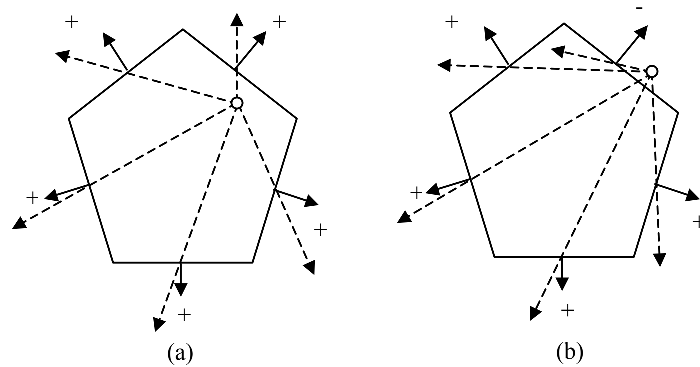

Fig. 3. Inner product as a criterion to determine the point-in-polygon problem for the case of convex polygons

If all the inner products are positive, the query point lies in the interior of the polygon, whereas if all the products are positive except one that is null, the point *could* lie on the edge whose associated product is null. If all the products are positive except two that are null, the point lies in the vertex associated with the edges whose inner products are null. In any other case the point is outside the polygon.

3 A Quick Point Location Algorithm

The method we present here is based on a recursive quaternary partition of the space by means of a quadtree data structure. In essence it is not very different from the methods presented previously, though it does have some important characteristics. On one hand the following method uses a "Bounding Quad-tree Volume, (BQV)" in seeking to obtain O(log n) to solve the point location problem as the best methods do, but on the other hand it tries to resolve commonly encountered problems with some adaptive characteristics to the particular problem.

Given P as a partition of the space, as for example a polygon, the objective of the BQV method is to define a set of two-dimensional intervals, V_i, E_i, I_i, O_i in order to define a partition Q of all the space, in such a way that for any query point, we return an interval of the partition Q in which the query point is located. If the query point is in a box V_i or E_i we can calculate with just a simple operation--in fact an inner product--the location of the point regarding the whole polygon P and for points located in an interval I_i or O_i we resolve directly with a simple comparison. Additionally, in order to define an optimum method we should be able to index this set of intervals with at least O(log n).

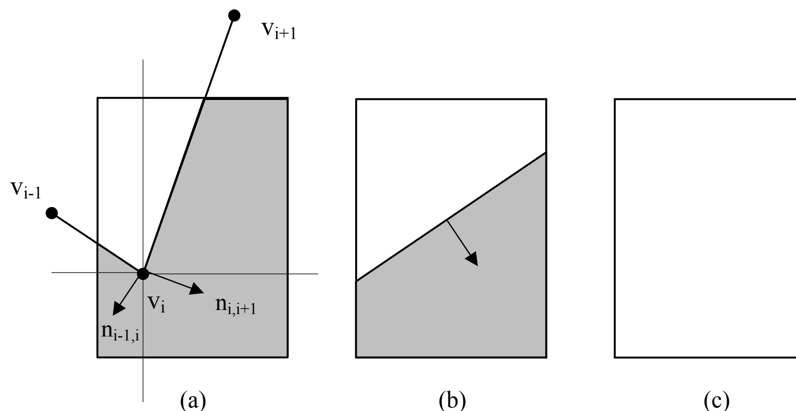

Fig. 4. The different types of intervals defined to cover the space; (a) interval V_i, just with a vertex of P, (b) interval E_i, just with an edge of P, (c) interval I_i, wholly inside the polygon, or interval O_i, wholly outside the polygon

Although the sizes of the intervals shown in Fig. 4 have been represented with the same size, this factor depends on the partition of the space created by the quadtree. Note also, that for the case of polygons we define and order the vertices of the polygon, for the sake of simplicity of implementation. For the case of traversing polygons it is easier to use an array as a storage data structure instead of typical data structures for graphs in the case of a general partition.

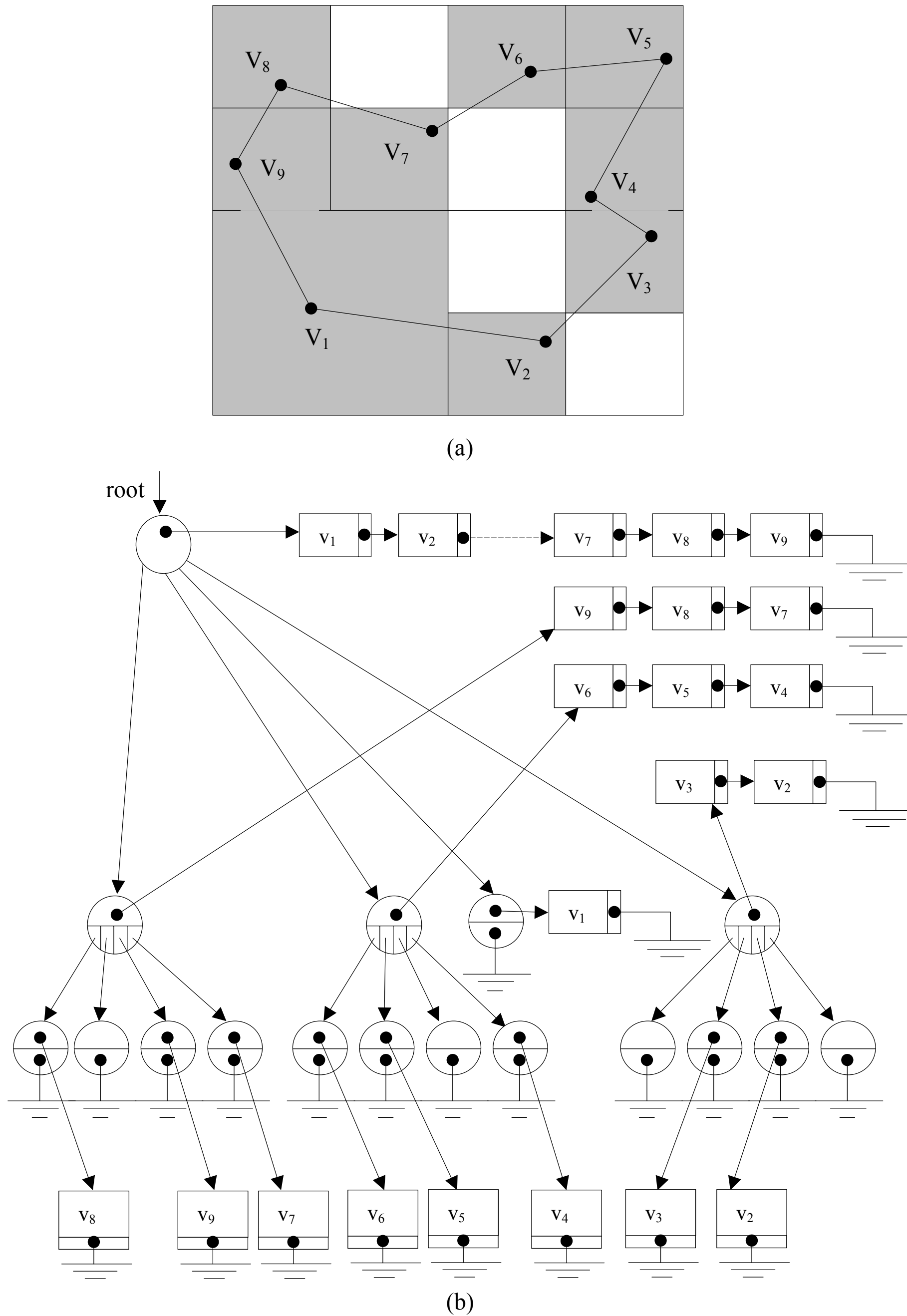

Fig. 5. Creation of a quaternary partition with identification of intervals V_i. (a) Each Vi interval identifies in the creation of the quad-tree is marked as V_i interval, in the figure shadowed intervals. (b) Intermediate data structure for the creation of the quaternary tree

Given a polygon P, the first step is to create a recursive quaternary partition of the space for the identification of the V_i intervals. To do that we create a quadtree associated to the space in which we embed the polygon with the simple condition that any node must have associated either one vertex or zero. In the creation of the quadtree, every node of the tree has a list associated with the points inside of its associated area as shown in figure 5. Once the tree has been created these lists are not useful and can be deleted.

For the identification of the intervals E_i, once the V_i intervals have been identified we proceed to calculate the intervals E_i between each two consecutive V_i intervals as shown in Fig. 6. The property we will use for making this calculus is twofold: on one hand we will calculate the intersection point between the intervals and the edge defined by two consecutive vertices and, on the other hand, we will use neighbourhood properties between intervals.

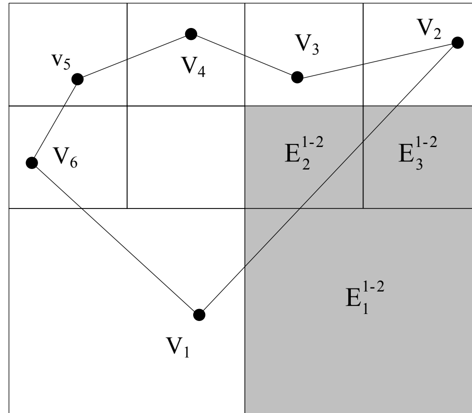

Fig. 6. Identification of E_i intervals of the space partition for a polygon P

For the calculus of the successive intersection points among edges and the intervals we need to calculate previously the angle B defined between the edge $e_{i\text{-}j}$ and the x axis. Note that B is an invariant for each E_i interval. After obtaining each intersection point we identify the neighbour to that common intersection point as the following E_i interval. The process continues until reaching the interval V_{i+1}.

For the calculus of the intersection points we start from a V_i interval. For this calculus we will define a reference system centred on the same vertex as shown in Fig. 7. For knowing at which edge will be the intersection point we will need to calculate the angles A_i as shown in Fig. 7.

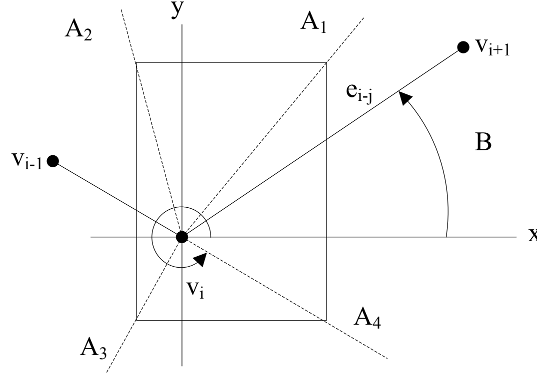

Fig. 7. Space and interval V_i are divided in four areas by the angles A_1, A_2, A_3 and A_4

In order to identify the intersection point of the edge V_i-V_{i+1} with the sides of the interval V_i, we firstly identify the intersection side by comparing the angle B with the angles A_i. The angles A_1, A_2, A_3, A_4 and B are given by the following formulas, in which we have taken into account the negative sense of the axis *y* as it is usual in graphics libraries.

	$\Delta y > 0$	$\Delta y < 0$
$\Delta x > 0$	$B = 2\pi - \text{atan}\dfrac{\lvert\Delta y\rvert}{\lvert\Delta x\rvert}$	$B = \text{atan}\dfrac{\lvert\Delta y\rvert}{\lvert\Delta x\rvert}$
$\Delta x < 0$	$B = \pi - \text{atan}\dfrac{\lvert\Delta y\rvert}{\lvert\Delta x\rvert}$	$B = \pi + \text{atan}\dfrac{\lvert\Delta y\rvert}{\lvert\Delta x\rvert}$

Table 1. Calculus of the angle B

Defining Δx and Δy as the differences between the coordinate points of the vertices V_i, V_{i+1}, given by

$$\Delta x = v_{1+1}.x - v_i.x$$
$$\Delta y = v_{1+1}.y - v_i.y$$

once we have identified the side of the interval in which the intersection point c lies, we are prepared for the particular calculus of the intersection point c. Taking into consideration the angles A_i, we identify for each side of the interval V_i a positive part A_i^+ as well as a negative one A_i^-, as indicated in Figure 8.

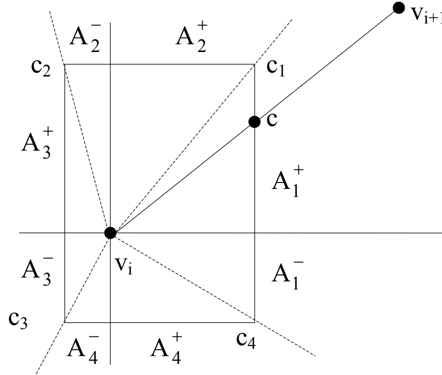

Fig. 8. Splitting of the sides of the interval V_i

Once identified the side in which the intersection point lies, we explicitly obtain the coordinates of the intersection point c. For example, if B is lower than A_1 and greater or equal to 0 --which means that c lies over A_1^+ as shown in Fig. 8-- we calculate the coordinates of the intersection point c; c.x and c.y, as follows[2]

$$c.x = c_1.x$$
$$c.y = v_i - \tan(B)\,(c_1.x - v_i.x)$$

After obtaining the intersection point c_i, the following step is to find the interval E_i neighbour to V_i on the side of the intersection point c_i. Assuming δ is the length of the minimum side of any interval of the space partition Q that we have created with the quadtree structure, then we can ensure that the point q defined in Table 2 is inside of the neighbour E_i. This fact allows us to locate the neighbour interval E_i by simply traversing the quadtree in order to locate the interval in which that point q is contained.

Side	q.x	q.y	Side	q.x	q.y
A_1^+	c.x+d/2	c.y	A_3^+	c.x-d/2	c.y
A_1^-	c.x+d/2	c.y	A_3^-	c.x-d/2	c.y
A_2^+	c.x	c.y-d/2	A_4^+	c.x	c.y+d/2
A_2^-	c.x	c.y-d/2	A_4^-	c.x	c.y+d/2

Table 2. Calculus of a point q, inner to the interval E_i neighbour to V_i

[2] The detailed calculus as well as an implementation of the more relevant algorithms can be found in [13]

If the interval E_i found, is V_{i+1} the calculus process for the intervals E_i that cover the edge V_iV_{i+1} has finished, and there is no any interval E_i between the vertices V_i and V_{i+1}, which is the case of vertices V_5V_6 or V_6V_1 in Fig. 6. If the interval found is not a vertex V_i we mark this interval in the quadtree as interval E_i and we proceed in a similar way to calculate the neighbour of this new interval, which shares an intersection point. For the calculus of neighbour intervals from an interval E_i we consider in each E_i interval, two angles A_1 and A_2, and four possible sides in which the intersection point can lie (see Fig. 9). The reference system now, as is shown in Fig. 9, is centred on the same intersection point.

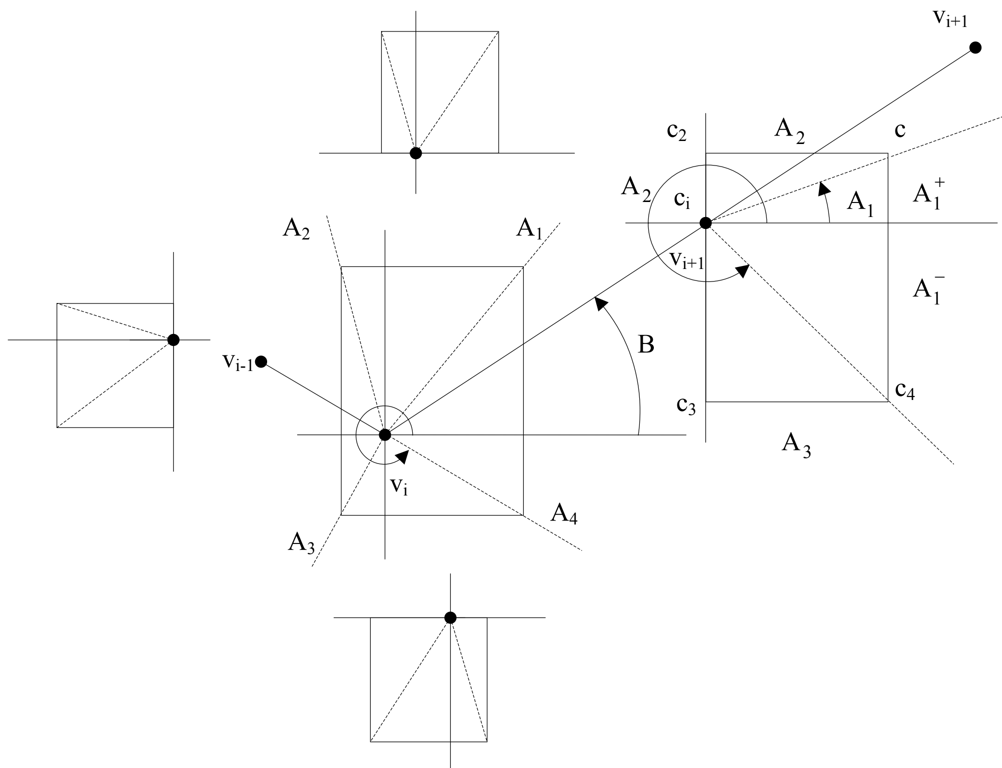

Fig. 9 . Definition of reference systems for the calculus of E_i intervals

At the beginning of the process, previous to the identification of the E_i intervals, in the identification process of V_i all the intervals of the spatial partition are created and labelled as outer intervals to the polygon if they do not correspond to a V_i interval, which in practice means to do nothing.

Once the process for the identification of the V_i intervals and E_i intervals has completed, for the labelling of inner intervals to the polygon P we must define what is inside and what is outside of P, by defining an inner point of the polygon and after that by means of a seed algorithm we will be able to fill the inner space to the polygon labelling all the inner intervals. After the process defined what we have, as result, is a

partition of the space in which we can find four types of intervals; V_i if the interval contains only one and only one vertex of the polygon, E_i if the interval is cut by any edge of the polygon, I_i if the interval is wholly inside the polygon and finally O_i if the interval is wholly outside the polygon.

4 Computational Cost

The cost of BQV algorithm is determined by the access to the interval in which the query point is located, plus a constant cost for each interval that in principle we ignore. As the intervals are indexed with a quaternary tree, in the average case in which we suppose a uniform spatial distribution of the vertex of the polygon over the initial space, the average cost of BQV method corresponds to the average height of the tree, log(m), where m is the number of intervals, always greater for a polygon but in the same order than the number of vertices for a graph.

However, surprisingly we find that there is no a direct relationship between the number of vertices of the polygon and the computational cost required to solve by the BQV method. This fact has to do with the adaptability presented by the BQV method. Let us calculate the worst case of the cost of the BQV method and we shall see that it corresponds to the depth of the tree. Then we will compare for the average case, the cost for a regular polygon inside a circle of a given radius.

The depth of a quadtree for a set of points P in the plane is as maximum $\log(D/d)+3/2$, where d is the minimum distance between any two points of P and D is the length of the initial square that contains P.

In a quadtree the length of the side associated to each node is half of the length of the parent. In general can be said that the length of the associated square to a node of depth i, will be $D/2^i$. On the other hand, the maximum distance between two inner points to a square is as maximum the distance of the diagonal, given by

$$D\sqrt{2}/2^i$$

for the area associated to the node of depth i. Take into consideration that an inner node of the quadtree has as minimum two points in its interior and the minimum distance between two point is d, at level i the inner nodes must satisfy

$$D\sqrt{2}/2^i \geq d$$

which implies

$$i \leq \log\left(D\sqrt{2}/d\right) = \log(D/d) + 1/2$$

and from the fact that the depth of a quadtree is exactly the maximum of the inner nodes plus one, we conclude that as maximum the depth of the quadtree will be

$$\log(D/d) + 1/2 + 1 = \log(D/d) + 3/2$$

and therefore the cost of the BQV algorithm, in the worst case, will be as maximum

$$\log(D/d) + 3/2$$

From the previous relationship we observe that if d tends toward zero the maximum cost tends to infinity, the more relevant cost on which we must focus attention is the average cost, that can be calculated as follows

$$\overline{c} = \sum_i p(Q_i)\, c(Q_i)$$

In general it can be proven that for a closed polygon with a number n of vertices this cost is finite and corresponds to a small value. So, for example for an regular polygon built inside a circle of radius 300 pixels, the average cost with a number of vertices over 10000, is on the order of just three comparisons.

5 Conclusions

We have presented a new algorithm for the solution of the well-known point location problem and for the more specific problem of point-in-polygon determination. Far from the O(log n) of other algorithms presented in the literature, we have obtained an extraordinary cost that does not depend directly of the number of vertices of the polygon for the average case.

The worst case of the BQV method depends on the factor D/d in which the minimum distance between the vertices of the polygon is crucial, though for the average case can be demonstrated that the worst cases do not contribute relevantly. Although the focus of this paper has been centred on polygons, for simplicity reasons, the BQV method can be extended in the same way to any spatial partition.

Regarding the storage and the building cost, the costs obtained are higher than the optimum O(n) obtained for other algorithms for particular cases seen in early sections of this paper, but are lower than the $O(n^2)$ of other algorithms as for example the slab method. The BQV method does not offer an implementation as simple as other particular algorithms seen in this paper, but the use of the BQV algorithm could be suitable to solve the point-in-polygon problem in cases of high frequency queries and typical real-time constraint situations.

6 Acknowledgements

This work was partially supported by European Union projects Esprit 25029 (GeoIndex) and IST-2001-37724 (ACE-GIS). We would like to thank in particular Dr. Antonio Rito da Silva from INESC-ID, Universidade Técnica de Lisboa for the unconditional support given during the development of this work.

7 References

1. Preparata, Franco and Shamos, Michael. “Computational geometry, an introduction”, Springer-Verlag 1985.
2. O’Rourke, Joseph. “Computational geometry in C”, Cambridge University Press 2001.
3. Haines, Eric.. “Point in polygon strategies”. In Paul Heckbert, editor, Graphics Gems IV, pages 24-46. Academic Press, Boston, 1994.
4. Tor, S. B: and Middleditch, A. E.. “Convex decomposition of simple polygons”. ACM Trans. on Graphics, 3(4):244-265, 1984.
5. Seidel, R.. “A simple and fast incremental randomised algorithm for computing trapezoidal decomposition and for triangulating polygons”, Comput. Geom. Theory Appl. 1991.
6. Edelsbrunner, H., Guibas, L. J. and Stolfi, J., “Optimal point location in a monotone subdivision”, SIAM J. Comput.1986.
7. Kirkpatrick, D. G., “Optimal search in planar subdivisions”, SIAM J. Comput., 1983.
8. Mulmuley, K. and Schwarzkopf, O.. “ Randomized algorithms”, in J. E. Goodman and J. O’Rourke, eds., Handbook of Discrete and Computational Geometry, CRC Press LLC, Boca Raton, 1997.
9. Chiang, Yi-Jen, Tamassia, Roberto. “Dynamization of the trapezoid method for planar point location”. Seventh annual symposium on computational geometry. ACM Press, New York, NY, USA, 1991.
10. de Berg, M., van Kreveld M., Overmars, M., Schwarzkopf, O.. “Computational Geometry, Algorithms and applications”, Springer-Verlag 2000, Second Edition.
11. Dobkin, D. P. and Lipton, R. J. “Multidimensional searching problems”, SIAM J. Comput., 1976.
12. Kirkpatrick, D. G., Klawe, M. M. and Tarjan R. E. “Polygon triangulation in O(n log log n) time with simple data structures”, In Proc. 6th Annu. ACM Sympos. Comput. Geom., 1990.
13. Poveda, J.. “Contributions to the Generation, Visualisation and Storage of Digital Terrain Models: New Algorithms and Spatial Data Structures”, PhD. dissertation, Universitat Jaume I, Castellon Spain. 2004.