On the Canonical k-vertex Polygon Spatial Retrieval Problem [*]

V. Bistiolas [†] S. Sioutas [‡] D. Sofotassios [§] K. Tsichlas [¶]

Abstract

The polygon retrieval problem on points is the problem of preprocessing a set of n points on the plane, so that given a polygon query, the subset of points lying inside it can be reported efficiently.

It is of great interest in areas such as Computer Graphics, CAD applications, Spatial Databases and GIS developing tasks. In this paper we study the problem of canonical k-vertex polygon queries on the plane. A canonical k-vertex polygon query always meets the following specific property: a point retrieval query can be transformed into a linear number (with respect to the number of vertices) of point retrievals for orthogonal objects such as rectangles and triangles (throughout this work we call a triangle orthogonal iff two of its edges are axis-parallel).

We present two new algorithms for this problem, both of which use $O(n \log^2 n)$ space. The time complexities for the queries is $O(k \log^3 n + A)$ for the first algorithm and $O(k \log n + A)$ for the second algorithm, where A denotes the size of the answer and k is the number of vertices.

The best previous solution for the general polygon retrieval problem uses $O(n^2)$ space and answers a query in $O(k \log n + A)$ time, where k is the number of vertices.

1 Introduction

Given a set S of n points on the plane, the problem of retrieving a subset $S' \in S$ that lie in the interior of a planar geometric object is of great interest in the areas of Computational Geometry, Spatial Databases, Computer Graphics, CAD and GIS applications. The efficiency of the solutions presented so far depends on the existence or not of orthogonality on the query object, which means that not all the line segments forming the query figure are vertical or horizontal.

A range tree for a example, is powerful enough to support windowing of points (i.e. the query object is an arbitrary axis-parallel rectangle) in $O(\log n + A)$ time using $O(n \log n)$ space. The problem becomes harder as the complexity of the query object increases (i.e. triangle, quadrilateral, arbitrary polygon), and there is no full orthogonality.

[*]This research was partially supported by the Hellenic Research Programme PYTHAGORAS (EPEAEK).

[†]Department of Computer Engineering and Informatics, University of Patras, P.O. BOX 26500, Greece; bistiolas@ceid.upatras.gr.

[‡]Department of Computer Engineering and Informatics, University of Patras, Greece and Digital Systems and Media Computing Laboratory (DSMC), Hellenic Open University (HOU); sioutas@eap.gr

[§]Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece; sofos@cti.gr

[¶]Computer Science Department, King's College, Strand, London WC2R 2LS, England; kostas@dcs.kcl.ac.uk

It is important to notice that the point retrieval problem for simple polygons is an interesting problem for many application areas. In medicine for example, the term ROI, which stands for Region Of Interest, is widely used by physisians in order to indicate a polygonal region of arbitrary complexity on their scene (i.e. Radiology Image). Many information systems developed so far support retrieval queries at such a region by first computing its bounding rectangle. Then, they report all the points inside the rectangle and finally these points are filtered so that only the points inside the region remain.

Willard [11], was the first to present a solution for the point retrieval problem for simple k-vertex polygons. It uses $O(n)$ space and the query time is $O(n^{0.77}+A)$. Edelsbruner and Welzl [8], reduced the query time to $O(n^{0.69}+A)$. A faster algorithm was presented by Edelsbruner, Kirkpatrick and Maurer [7] that uses $O(k\log n+A)$ query time and $O(n^7)$ space. Cole and Yap [5], presented a method using $O(n^2/\log n)$ space and $O(k\log n\log\log n+A)$ time. Finally, Paterson and Yao [9] have presented the best known solution (for a simple arbitrary polygon). This solution uses $O(n^2)$ space and answers a query in $O(k\log n+A)$ time.

In this work we consider the polygon retrieval problem on points for a subset of simple canonical polygons which always satisfy the following strict property: a retrieval query on a set of points can be transformed to a linear number (on the number of their vertices) of queries on orthogonal objects such as rectangles and triangles. We call a triangle orthogonal iff two of its edges are axis-parallel. We present two solutions with $O(n\log^2 n)$ space and $O(k\log^3 n+A)$,$O(k\log n+A)$ time complexities respectively.

This paper is organized as follows. In Section 2 we briefly introduce some preliminary data structures. In Section 3 we give the details of our algorithms. In Section 4 we present some special extensions for the general polygon retrieval problem based on algorithm of Paterson and Yao for which we shortly introduce the fundamental notions. Finally, some conclusions and further extensions concerning this problem are considered in Section 5.

2 Preliminary Data Structures

2.1 Half Plane Range Query

The half plane range query problem is the problem of reporting all the points in a set S of n points on the plane that lie on a given side of a query line L. This section makes a very short description of the method presented by Chazelle, Guibas, and Lee [4] that achieves optimal $O(\log n+A)$ query time and linear space using the notion of duality. The main steps of their algorithm are the following:

- Preprocessing

  1. Partition S into a set of convex layers:
     (a) Define S_i as the convex hull of all the points currently in S
     (b) Remove the vertices of S_i from S
     (c) Increment i, repeat the process

     The time cost is $O(n\log n)$ while the space complexity is $O(n)$, using a technique that computes convex hulls in a dynamic environment [1].

2. Augment the set of layers building vertical connections as follows: for each vertex w of layer S_i, keep a pointer to the two edges immediately above and below w. This clearly uses $O(n)$ extra space.
3. Using duality, the transformation of each vertex w into its corresponding line maps each layer into another convex polygon. The produced mapping is organized into a point location structure, occupying $O(n)$ space.

- Query Processing

1. Given a query line L transform it into its corresponding dual point P_L.
2. Apply a planar point location algorithm for the point P_L in the properly organized structure. This determines the innermost layer among the layers containing the point P_L. Thus, in the dual mapping it determines the innermost layer among the layers that L intersects. Call this layer *neighboring*. Using an optimal point location algorithm, this costs $O(\log n)$ time.
3. Using the pointers mentioned at step 3 of the preprocessing procedure, it is easy to report one vertex lying at the query half plane for each layer which encloses the neighboring one.
4. Traverse each layer from each of the vertices reported across the part of the layer inside the half plane. Report the vertices traversed.

Clearly, these steps lead to an algorithm for answering half plane range queries using $O(n)$ space and $O(\log n + A)$ query time. We use this method in order to answer orthogonal triangle range queries on points.

2.2 Priority Search Tree

In this subsection, we briefly review the priority search tree of McCreight [12]. Let S be a set of n points on the plane. We want to store them in a data structure, so that the points that lie inside a semi-infinite strip of the form $[a, b] \times (-\infty, c]$, can be found efficiently.

The priority search tree is a binary search tree over the x-coordinates of the points. The root of the tree contains the point p with the minimum y-coordinate. The left (resp. right) subtree is recursively defined for the set of points in $S - \{p\}$. The set $S - \{p\}$ is partitioned equally into the two subtrees of the root. As a result, it is easy to see, that a point is stored in a node on the search path from the root to the leaf containing its x-coordinate.

Initially, queries with ranges that are half-infinite in both x and y directions are considered (i.e. they are of the form $(-\infty, b] \times (-\infty, c]$). This special case is also known as *quadrant range search*. To answer a quadrant range query, we find the $O(\log n)$ nodes in the search path P_b for point b. Let L_b be the left children of these nodes that do not lie on the path (see Figure 1). In $O(\log n)$ time, the points of the nodes of $P_b \bigcup L_b$ that lie in the query-range can be determined. Then, for each node of L_b storing a point inside the range query, its two children are visited and checked whether their points lie in the range. This procedure continues recursively, as long as points in the query-range are found.

The correctness of the query algorithm is proved as follows. First, observe that nodes to the right of the search path, have points with x-coordinate larger than b and therefore lie outside the query-range. The points of P_b may have x-coordinate larger than b or they may have y-coordinate larger than c. In any case, they are not reported. The nodes of

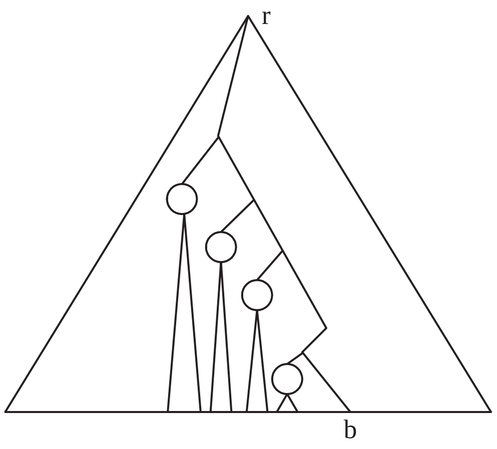

Figure 1: P_b: the search path, L_b: the nodes that are left sons of nodes on P_b and do not belong to the path.

L_b and their descendants have points with x-coordinate smaller than b, so that only their y-coordinates need to be tested. The children of nodes of L_b with y-coordinate less than c must be considered. In particular, the reporting procedure proceeds recursively, as long as points inside the query range are found. If a point of a node u does not lie inside the query-range, then this point has y-coordinate larger than c. Therefore, all points in the subtree rooted at u lie outside the query-range and they are not reported. We can easily bound the query time by $O(\log n + t)$, since $O(\log n)$ time is needed to visit the nodes in $P_b \bigcup L_b$ and $O(t)$ time is necessary for the reporting procedure in their subtrees.

Finally, consider the general case where the query ranges are of the form $[a, b] \times (-\infty, c]$. The query algorithm finds the nodes in the two search paths P_a and P_b for a and b respectively. Let C be the set of nodes, consisting of all left children of the nodes in P_a and all right children of the nodes in P_b. The nodes of $P_a \bigcup P_b \bigcup C$ that have points inside the query range can be determined in $O(\log n)$ time. Then, the descendants of nodes of C are traversed recursively as long as their points lie in the query range. Note that the nodes of C and their descendants have x-coordinates inside the query-range and as a result only their y-coordinates need to be considered. The correctness of the algorithm follows by similar arguments as in the case of quadrant range queries, while the query time remains $O(\log n + t)$.

2.3 Modified Priority Search Tree

Let S be a set of n points on the plane with coordinates (x, y), where $x \in \{1, \ldots, M\}$ and $y \in \Re$. Without loss of generality we assume that all points are distinct. We will show how to store the points in a data structure, so that the t points in a query range of the form $(-\infty, b] \times (-\infty, c]$, can be found in $O(t)$ time. Our structure relies on the priority search tree, which we augment with list-structures similar to those in [13].

We store the points in a priority search tree T, of size $O(n)$ as described in the previous section. For convenience we will assume that the tree T is a complete binary tree (i.e. all its leaves have depth $\log n$). Note that if n is not a power of 2, then we may add some dummy leaves so that T becomes complete. We also use an array A of size M, which stores pointers to the leaves of T. Specifically, $A[i]$ contains a pointer to the leaf of T with maximum

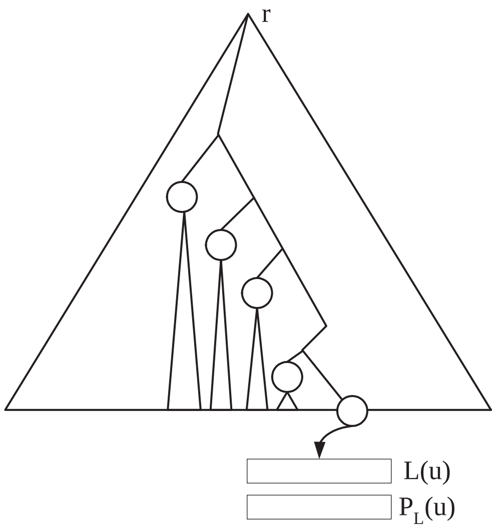

Figure 2: $L(u)$: it stores the points of the nodes of L_i. $P_L(u)$: it stores the points of the nodes of P_i which have x-coordinate smaller or equal to i.

x-coordinate smaller or equal to i. This array is used to determine in $O(1)$ time the leaf of the search path P_b for b. In each leaf u of the tree with x-coordinate i we store the lists $L(u)$ and $P_L(u)$. The list $L(u)$ stores the points of the nodes of L_i. The list $P_L(u)$ stores the points of the nodes of P_i which have x-coordinate smaller or equal to i. Both lists also contain pointers to the nodes of T that contain these points. Each list $L(u)$, $P_L(u)$, stores its nodes in increasing y-coordinate of their points (see Figure 2).

To answer a query of the form $(-\infty, b] \times (-\infty, c]$ we find in $O(1)$ time the leaf u of the search path P_b for b. Then, we traverse the list $P_L(u)$ and report its points until we find a point with y-coordinate greater than c. We traverse the list $L(u)$ in the same manner and find the nodes of L_b whose points have y-coordinate less than or equal to c. For each such node we report the respective point and then we continue the reporting procedure further in its subtree, as long as we find points inside the range.

The following theorem bounds the size and the query time of our structure.

Theorem 1 *Given a set of n points on the plane with coordinates (x, y) such that $x \in \{1, \ldots, M\}$ and $y \in \Re$, we can store them in a data structure with $O(M + n \log n)$ space that supports quadrant range queries in $O(t)$ time, where t is the number of reported points.*

Proof: The query algorithm finds the t' points of nodes of $P_b \bigcup L_b$ that lie inside the query-range in $O(t')$ time by simple traversals of the lists $P_L(u)$, $L(u)$. The search in these subtrees takes $O(t)$ additional time for reporting t points in total. Therefore, the query algorithm needs $O(t)$ time. Each list $P_L(u)$, $L(u)$ stores the respective points in the nodes of the path from root to u, and points in the left children of nodes of this path. So, the size of each list is $O(\log n)$ and the space of T is $O(n \log n)$. The space of the whole structure is $O(M + n \log n)$ because of the size of the array A. $\Box$

2.4 Geometric Transformation Of Duality

In homogeneous coordinates, there exists a duality between the point (a, b, c) and the line $(ax + by + cz = 0)$ for all $(a, b, c) \neq (0, 0, 0)$. Furthermore, the following pairs are dual:

- points on a line $\longleftrightarrow$ lines throught a point
- line segment $\longleftrightarrow$ "double wedge" of lines
- set of lines intersecting a line segment $\longleftrightarrow$ set of points in a double wedge (see Figure 3)

A more formal definition of this powerfull tool can be found in [4].

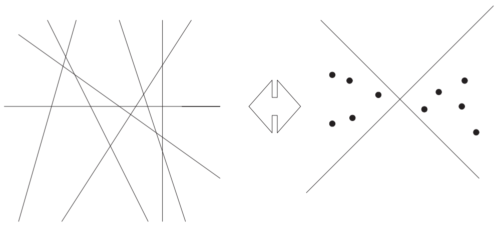

Figure 3: An Example of Dual Pair

3 Algorithms for Orthogonal Objects

3.1 Data Structures for Orthogonal Triangle Range Queries

An orthogonal triangle range query is the problem of determining all the points from a set S of n points on the plane lying inside an orthogonal triangle. Recall, a triangle is orthogonal iff two of its edges are axis-parallel. Let T be an orthogonal triangle defined by the point (x_q, y_q) and the line L_q that is not axis-parallel (see Figure 4).

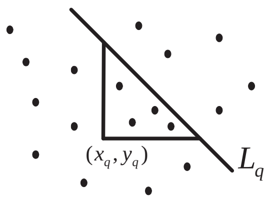

Figure 4: The query triangle.

A retrieval query for this problem can be supported efficiently by the following data structures:

1. **A 3-layered Tree:** Such a query is performed through the following steps:

   (a) In the tree storing the pointset S according to x-coordinates, traverse the path to x_q. All the points having x-coordinate in the range $[x_q, \infty)$ are stored at the subtrees on the nodes that are right sons of a node of the search path and do not belong to the path. There are at most $\lceil \log n \rceil$ such disjoint subtrees.

(b) For every such subtree traverse the path to y_q. By a similar argument as in the previous step, at most $\lceil \log n \rceil$ disjoint subtrees are located, storing points that have y-coordinate in the range $[y_q, \infty)$.

(c) For each subtree in Step 2, apply the half-plane range query to retrieve the points that lie on the side of line L_q towards the triangle.

The correctness of the above algorithm follows from the data structure used. Since we have to visit $O(\log n)$ subtrees in each step, the query time is $O(\log^3 n + A)$ while the space complexity is $O(n \log^2 n)$.

2. **A two layered tree:** We replace the first two layers of the previous structure by the Modified Priority Search Tree. In this case, a polygon retrieval query is performed according to the following steps:

   (a) In the Modified Priority Search Tree by scanning the appropriate lists we can find in $O(t)$ time a subset of t points with x-coordinate in the range $[x_q, \infty)$ and y-coordinate in the range $[y_q, \infty)$.

   (b) For each subset in Step 1 apply the half-plane range query to retrieve the points that lie on the side of line L_q towards the triangle.

   Since we access the appropriate quadrant subset of points in constant time (by the modified PST) the overall query time becomes $O(\log n + A)$. In addition, each leaf u stores two y-ordered lists ($P_L(u)$ and $L(u)$), both of size $O(\log n)$. The total number of sublists that start from the head of $L(u)$ or $P_L(u)$ is $O(\log n)$. Their total size is $1+2+3+...+O(\log n) = O(\log^2 n)$. In the pre-processing step we must build for each such subset of points (note that these points are y-ordered) the linear data structure of Chazelle for half-plane range query. The $O(n \log^2 n)$ space follows.

3.2 The Results

Every canonical k-vertex polygon can be decomposed into $O(k)$ orthogonal triangles because of the strict symmetry in the topology of the vertices. In Figure 5 such a decomposition is depicted.

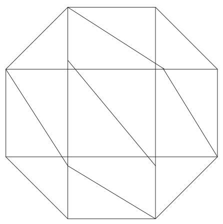

Figure 5: Orthogonal triangle decomposition of a canonical octagon

- ***Solution 1***: Querying 3-layered data structure $O(k)$ times, the $O(n \log^2 n)$ space and $O(k \log^3 n + A)$ time follows.
- ***Solution 2***: Querying the two layered data structure $O(k)$ times, the $O(n \log^2 n)$ space and $O(k \log n + A)$ time follows.

4 Some Thoughts on the General Polygon Retrieval Problem

4.1 The Algorithm of Paterson and Yao

This section sketches the basic ideas of the algorithm for polygon retrieval on points that Paterson and Yao [9] have presented.

This problem is to preprocess a set S of n points on the plane, so that for any query polygon P the subset of them lying inside it can be reported efficiently. We assume that the given polygon is convex with k vertices. If not, (P is non-convex) then a suitable decomposition into convex parts can be carried out and the algorithm can be applied to every such part. The key-idea of Paterson/Yao's algorithm is a further decomposition of every convex part into $O(k)$ simpler parts called *quads*. They solve the problem for each such quad separately, using the well-known geometric transformation of duality.

To set up the data structure, Paterson/Yao first sort the n points according to their polar angles at the origin and then store the ordered sequence in a leaf-oriented balanced binary search tree of depth $O(\log n)$. This structure answers the query: "determine the points having polar angle in the range $[a_1, a_2]$ by traversing the two paths to the leaves corresponding to a_1, a_2. The points stored as leaves at the subtrees of the nodes which lie between the two paths are exactly these points in the range $[a_1, a_2]$. For each subtree, the points stored at its leaves are organized further to a second level data structure as follows: by the duality correspondence, these points are mapped to a set of straight lines, and a point location structure is built for this planar subdivision [6]. Using this data structure a query of the form: "Report the points lying in a double wedge" is reduced to a query to the dual structure of the form: "report the lines that intersect a query line segment."

At this point, we are ready to give a step by step description of the polygon retrieval algorithm of Paterson and Yao:

1. Let P be the k-vertex convex query polygon. Separate P into simpler regions called *quads* by cutting the plane into *sectors*. The *sectors* are obtained by drawing semi-infinite lines from the *origin* to each vertex of P. Each *quad* has a particularly simple form (it is the intersection of a sector and a double wedge). (see Figure 6)

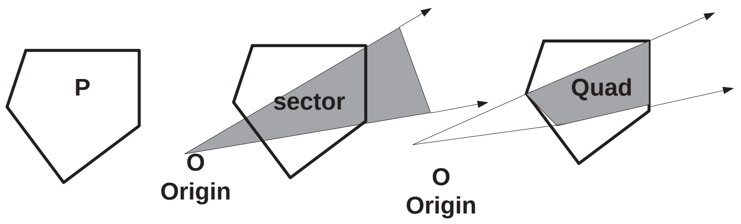

Figure 6: Sectors and Quads

Every k-gon is the disjoint union of $O(k)$ such *quads*.

2. Using the two-level data structure do the following for each quad with bounding rays ℓ_1, ℓ_2:

    - Determine the (at most) $2\lceil logn \rceil$ disjoint subtrees that descend from the two search paths to the angles of ℓ_1, ℓ_2 and lie between them. All the points between ℓ_1 and ℓ_2 are stored as leaves in these subtrees.

- for each such subtree in turn report the points inside the double wedge determining the quad using the dual point location structure.

The space used is $O(n^2)$ because of the point location structure and the query time is $O(k \log n + A)$ (there exist $O(k)$ quads and the time spent for every "quad-retrieval" is $O(\log^2 n + A_i)$ which can be improved using the fractional cascading technique [3] on the point location structures). It is worthwile to note that the above algorithm works even when P is unbounded. If P is non-convex, then the preliminary decomposition step into convex parts does not change the asymptotic space and time bounds.

Unfortunately, the $O(n^2)$ space bound is the best possible if we directly use the dual point location structures. This means that in order to reduce the space requirements of the later algorithm, we have to find an algorithm supporting "quad retrieval" without using the duality on arbitrary sets of $O(n)$ points which lies to $O(n^2)$ space subdivisions. The next section demonstrates such an algorithm working efficiently when P is a canonical k-vertex polygon.

4.2 An Extended Approach

In this section we present an extended approach for the general polygon retrieval problem. The proposed solution is based on the algorithm of Paterson/Yao for answering "quad retrieval" queries, which was described in the previous section. The key-idea is the reduction of a quad retrieval query to a constant $O(1)$ number of orthogonal triangles retrieval queries. This is achieved by drawing parallel lines (vertical or horizontal) from the vertices of the quad so that the lines stay inside the quad and intersect the rays from the origin. To do this, a suitable decomposition of each quad is needed. In the general case of an arbitrary (non canonical) k-vertex polygon, such a decomposition is not always possible. For example, the quad in Figure 7(a) can be decomposed into $O(1)$ orthogonal triangles while the quad in Figure 7(b) cannot be decomposed.

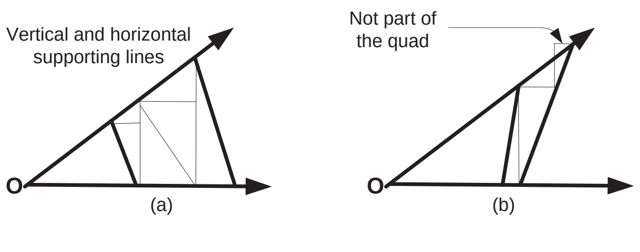

Figure 7: Decomposition of a quad

This of course depends on various geometric characteristics of the quad. If the polygon is k-vertex canonical, we can always directly carried out a decomposition into $O(k)$ orthogonal triangles because of the strict symmetry in the topology of the vertices. Arbitrary k-vertex polygons satisfy such a decomposition only in special cases.

5 Conclusions

This work presents efficient methods for answering retrieval queries on points for canonical polygons. We manage to design efficient algorithms based on the special structure of canonical polygons. In fact, as in many other problems, our algorithms are efficient because we managed to reduce the problem from these polygons to simple orthonormal objects like rectangles and orthonormal triangles.

One deficiency or our algorithms is that they are static. One interesting line of research would be the design of dynamic algorithms. It still remains an open problem whether an $O(n \log^{O(1)} n)$ space and $O(\log^{O(1)} n + A)$ time algorithm exists for the general problem.

References

[1] Chazelle B., *Optimal algorithms for computing depths and layers*, Brown University, Technical Report, CS-83-13, March 1983.

[2] Chazelle B., Dobkin D., *Decomposing a polygon into its convex parts*, Proc. 11^{th} ACM Symp. on Theory on Comp.,1979, pp.38-45.

[3] Chazelle B.,Guibas L., *Fractional Cascading: a data structuring technique with Geometric Applications*, 12^{th} ICALP 1985, pp. 90-100.

[4] Chazelle B.,Guibas L., Lee D.L., *The power of Geometric Duality*,Proc. of 24^{th} IEEE Annual Symposium on Foundations of Computer Science, 1983, pp.217-225.

[5] Cole R., Yap C., *Geometric Retrieval Problems*, Proc. 24^{th} IEEE Annual Symposium on Foundations of Computer Science 1983, pp. 112-121.

[6] Edelsbruner H., Guibas L.,Stolfi J., *Optimal point location in a monotone subdivision*, Technical Report 2, DEC systems Research Center, 1984.

[7] Edelsbruner H., Kirkpatrick D., Maurer H., *Polygonal Intersection Searching*, Information Processing Letters, 14, 1982, pp. 74-79.

[8] Edelsbruner H., Welzl E., *Halfplanar range search in linear space and $O(n^{0.695})$ query time*, Inform. Proc. Letters 23, 1986, pp. 289-293.

[9] Paterson M.S., Yao F.F., *Point Retrieval for Polygons*, journal of Algorithms, 1986, pp.441-447.

[10] Welzl E. *Partition trees for counting and other range searching problems*, Proc. 4th ACM Symp. on Computational Geometry, 1988, pp. 23-33.

[11] Willard D., *Polygon Rertieval*, SIAM J. Computing, 14,1982,pp. 149-165.

[12] E. M. McCreight, *Priority search trees*, SIAM J. Comput. 14 (1985), pp. 257-276.

[13] H. Overmars, *Efficient data structures for range searching on a grid*, J. Algorithms 9 (1988), pp. 254-275.