Machine Graphics and Vision, Vol.5., No.3., pp. 483-494, , ISSN 1230-0535, 1996.

Trading Time for Space: An O(1) Average Time Algorithm for Point-in-Polygon Location Problem

Theoretical Fiction or Practical Usage?

Václav Skala
Department of Computer Science
University of West Bohemia
Univerzitní 22, Box 314
306 14 Plzeň
Czech Republic

Abstract

Algorithms for Point-in-polygon problem solution are very often used especially in computer graphics applications. The naive implementation has O(N) processing time complexity or O(lg N) complexity if a convex polygon is considered. A new algorithm of O(1) processing complexity was developed. The important feature of the algorithm is that preprocessing complexity is O(N) and memory requirements depend on geometrical properties of the given polygon. Usage of the algorithm is expected in applications where many points are tested whether resides in the given polygon or not. The presented approach can be considered as alternative to the parallel processing usage. Experimental results are included, too.

Keywords: Point-in-Polygon Algorithm, Data Structures, Algorithm Complexity, Geometry.

1. Introduction

Point-in-Polygon algorithms are very often used in many applications, especially within many computer graphics packages. In some applications a number of edges of given convex polygon can be very high. In some cases there is critical processing time even if number of edges is small because very high number of points are processed. Therefore it is necessary to use faster algorithms with O(N) or O(lg N) complexities, where N is a number of edges of the given polygon.

Nevertheless if N is small an algorithm with O(N) complexity is often used. Such a naive algorithm can be described by alg.1. Let us consider a convex polygon with the anticlockwise orientation and define following functions

$$F_i(x) = (a_i x + b_i y + c_i) \qquad i = 0, \ldots, N-1$$
$$G_i(x) = (u_i x + v_i y + w_i) \qquad i = 0, \ldots, N-2$$

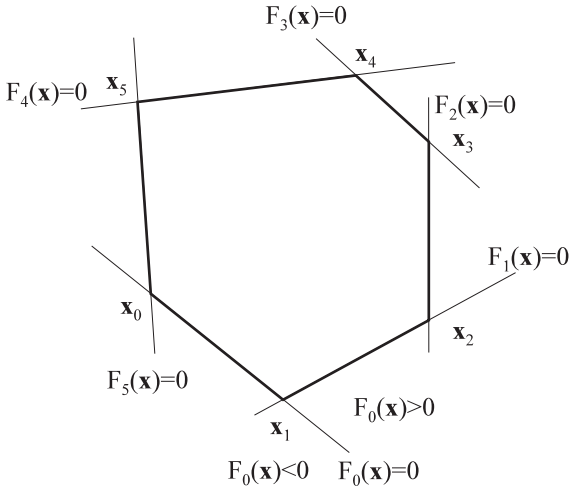
Machine Graphics and Vision, Vol.5., No.3., pp. 483-494, , ISSN 1230-0535, 1996.

where

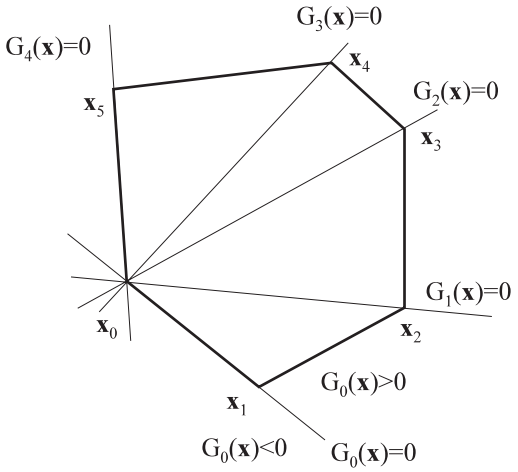
$F_i(x) = 0$ is an equation for the oriented line on which the line segment $x_i \; x_{i+1}$ lies, see fig.1 (+ means addition modulo N). If x lies inside of the polygon then $F_i(x) \geq 0 \;\; \forall \;\; i$,

$G_i(x) = 0$ is an equation for the oriented line on which the line segment $x_0 \; x_{i+1}$ lies see fig.2, i.e. $F_0(x) \equiv G_0(x)$ and lines orientations are shown on fig.1 and fig.2 (will be used in the following algorithms),

N is a number of edges of the given convex polygon.



Definition of $F_i(x)$ functions
Figure 1



Definition of $G_i(x)$ functions
Figure 2

```
procedure PRECOMPUTE; { O(N) preprocessing time }
begin for i := 0 to N - 1 do
      COMPUTE_COEFF(x_i, x_{i+1}, a_i, b_i, c_i);
      { compute coefficients for F_i (x) }
end { PRECOMPUTE };

function POINT_IN_POLYGON (x: point): boolean;
{ algorithm with O(N) processing time }
begin POINT_IN_POLYGON := false;
      for i := 0 to N - 1 do
        if F_i(x) < 0 then EXIT; { exit from procedure }
        { point x must lie on the same side for all edges }
      POINT_IN_POLYGON := true; { point INSIDE of the polygon }
end { POINT_IN_POLYGON };
```

Algorithm 1

2. Algorithms with O(lg N) complexities

It has been proved that algorithms with processing complexity O(lg N) exist, see [PRE85a]. The original algorithm is based on a presumption that a special point x_A exists so that it is inside of the given polygon, see fig.3.

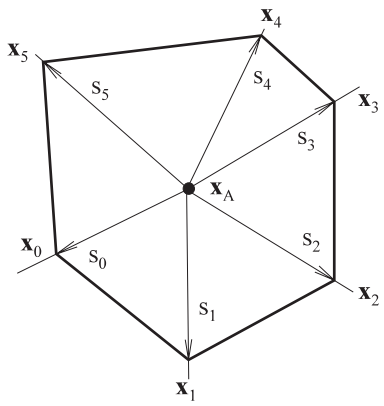


Figure 3

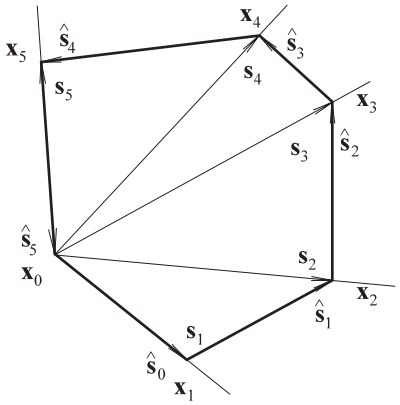


Figure 4

The answer if a given point lies inside the given convex polygon can be given in O(lg N) time using binary search over indices i, j asking whether the given point x lies inside wedge $x_i x_A x_j$ using O(N) space. O(N) preprocessing time is needed for finding any convenient point x_A, e.g. a centroid can be taken.

Let us define

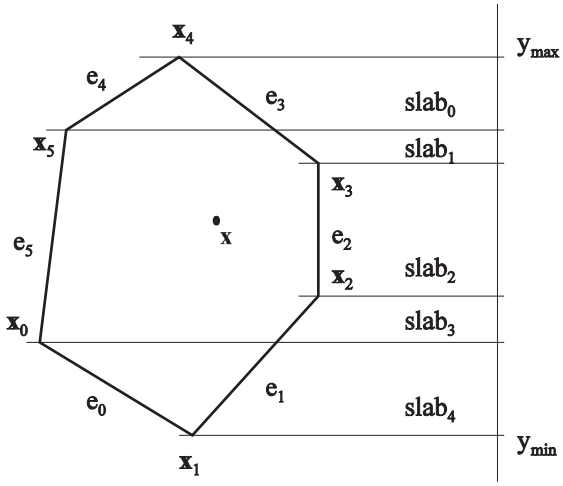
$$s = x - x_A, \qquad s_i = x_i - x_A, \qquad \hat{s}_{i+1} = x_{i+1} - x_i$$

Then the determination whether the point x is inside of the wedge $x_i x_A x_j$ can be made by using condition

$$[s \times s_i]_z < 0 \text{ xor } [s \times s_j]_z > 0$$

where $[a \times b]_z$ means z-coordinate of the cross product of a and b usually for the first step $i = 0$ and $j = N - 1$.

A faster solution can be made if we set point x_A equal to x_0. In this case, see fig.4, the algorithm will be faster as we need only O(1) preprocessing complexity. For determination whether a point x is inside of the given polygon a similar approach can be used as in previous algorithm. More effective processing can be obtained if vectors s_i and $\hat{s}_i$ are precomputed. It means that we have got a little bit faster algorithm with O(N) preprocessing and O(lg N) processing complexities. But we still need to compute z-coordinate of a cross product two times for each step. It is possible to use separation functions $G_i(x)$ instead of using z-coordinate of the cross product. Algorithm with such modification is faster because it uses as much precomputed values as possible and can be described by alg.2.



y'	AEL	
y_4	4	3
y_3	5	3
y_5	5	2
y_2	5	1
y_0	0	1

indexes of associated edges for each slab

Figure 5

```
procedure PRECOMPUTE; { O(N) preprocessing time }
begin for i := 0 to N - 1 do
      begin COMPUTE_COEFF(x_i, x_{i+1}, a_i, b_i, c_i);
         { compute coefficients for F_i(x) }
         if i ≠ (N − 1) then
             COMPUTE_COEFF( x_0, x_{i+1}, u_i, v_i, w_i);
             { compute coefficients for G_i(x) }
     end { for };
end { PRECOMPUTE };

function POINT_IN_POLYGON (x:  point):  boolean;
begin { O(lg N) processing time }
      POINT_IN_POLYGON := false;
      i := 0; j := N-1;
      ξ := G_i(x) ≥ 0;   η := G_{j−1}(x) ≥ 0;
      if (not ξ) or η then EXIT;
      { point outside of the wedge x5x0x1 ing fig.2.  }
      while (j - i) > 1 do
      begin
         k := (i + j) / 2;   { implemented as shift to the right }
         ζ := G_k(x) ≥ 0;
         if ζ then i := k else begin j := k;
      end { while };
      POINT_IN_POLYGON := F_i(x) ≥ 0;
```

```
      { check whether the point x is inside of the triangle x_i x_0 x_j }
end { POINT_IN_ POLYGON };
```

Algorithm 2

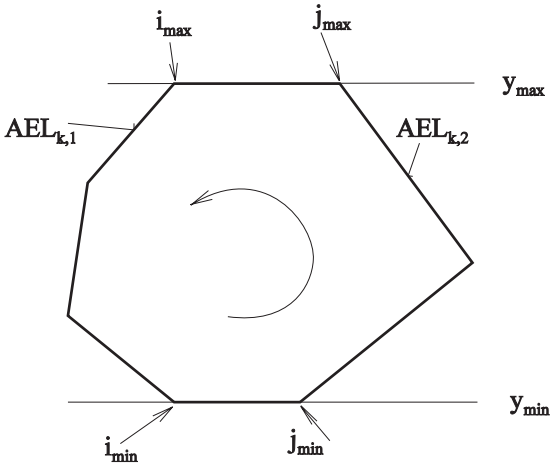


Figure 6

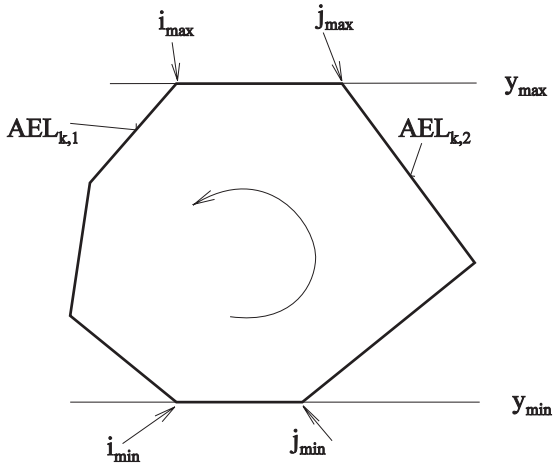


Figure 7

The second approach [SAL78a] used slab technique, see fig.5. In this case we have to sort all given vertices x_i according to their y-coordinate and associate a pair of indices of active edges with each slab in the AEL (Active Edge List). In this case we get O(N) preprocessing time complexity but processing time is still O(lg N) as we have to search for slab in which the given point x lies using binary search, see alg.3. If we find a relevant slab ($slab_2$ in fig.5) we have only to just evaluate whether

$$(F_2(x) \geq 0)\ and\ (F_5(x) \geq 0)$$

This algorithm is a little faster because for finding a slab only two comparison are needed instead of cross products computation. There are some small complications caused by horizontal edges at the top and bottom of the given convex polygon. Those edges must be left out; for details see [SAL78a].There are special cases, fig.6, that must be handled carefully. Fig.7 shows the usual situation during construction of the AEL list, described by alg.3.

```
procedure PRECOMPUTE; { O(N) preprocessing time }
begin { algorithm uses slabs technique }
      y_min := +∞;    y_max := −∞;
      for i := 0 to N - 1 do
      begin { find minimal and maximal valus }
         if y_i < y_min then begin y_min := y_i; i_min := i; end;
         if y_i > y_max then begin y_max := y_i; i_max := i; end;
      end;
```

```
      { AEL_{i,k} is the active edge list for the i-th slab; k=1,2 }
      Δ := y_max − y_min;
      { find all slabs - horizontal edges must be removed }
      j_max := i_max;    j_min := i_min;
      { remove upper horizontal edge }
      if y_{(i_max+1) mod N} = y_max    { for top vertices }
            then i_max := (i_max + 1) mod N
            else if y_{(j_max−1) mod N} = y_max
                  then j_max := (j_max − 1) mod N;
      if y_{(i_min−1) mod N} = y_min { for bottom vertices }
            then i_min := (i_min − 1) mod N
            else if y_{(j_min+1) mod N} = y_min
                  then j_min := (j_min + 1) mod N;
      i0 := i_max;     j0 := j_max;     k := 0;
      i := (i_max + 1) mod N; j := (j_max − 1) mod N;
      y'_k := y_{i_max};
      AEL_{k,1} := i_max;     AEL_{k,2} := j;
      while not ((i = i_min) and (j = j_min)) do
      begin
       k := k + 1;
       if y_i > y_j then
         begin y'_k := y_i;     AEL_{k,1} := i0;     AEL_{k,2} := j;
             i0 := i; i := (i + 1) mod N;
         end
       else if y_i < y_jthen
         begin y'_k := y_j;     AEL_{k,1} := i0;     AEL_{k,2} := j;
             j0 := j; j := (j − 1) mod N;
         end
       else
         begin y'_k := y_j;     AEL_{k,1} := i0;     AEL_{k,2} := j;
             i0 := i; i := (i  + 1) mod N;
             j0 := j; j := (j − 1) mod N;
         end
      end { while };
end { PRECOMPUTE };

function POINT_IN_POLYGON (x:  point):  boolean;
begin { O(log N) processing time }
      i := 0; j := N - 1;     POINT_IN_POLYGON := falce;
      if y ∉< y_min, y_max > then EXIT;
      { the point x is outside of all slabs, see fig.5 }
      while (j − i) > 1 do
      begin k := (i + j)/2;    { implemented as shift to the right }
```

```
        if y > y'_k then j := k  else i  := k;
      end { while };
      { now test only two edges of the i-th slab from AEL }
      POINT_IN_POLYGON := (F_AEL_i,1(x) ≥ 0) and (F_AEL_i,2(x) ≥ 0);
end { POINT_IN_POLYGON }
```

Algorithm 3

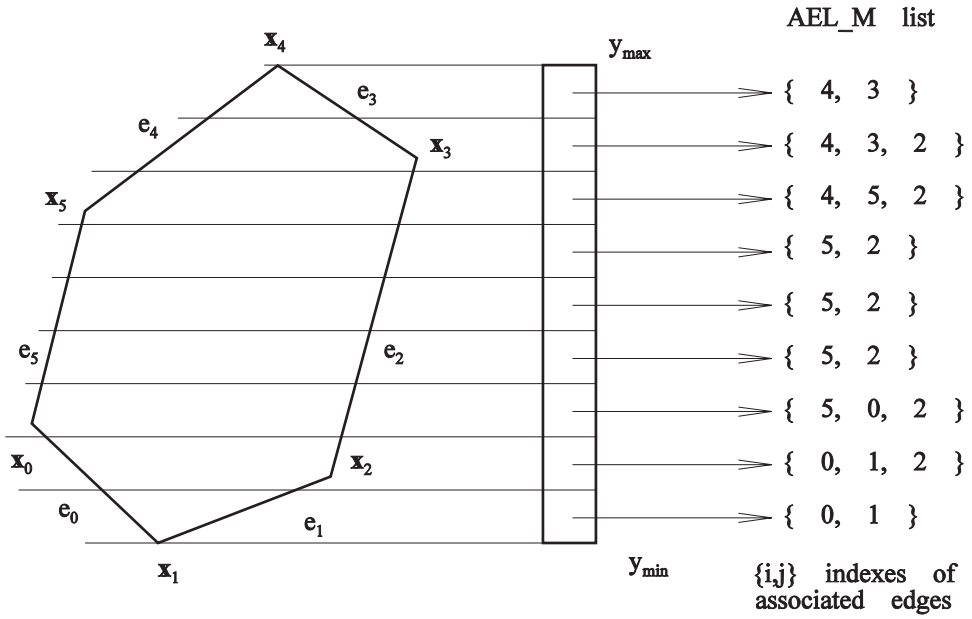
3. Principle of the proposed method

To be able to significantly decrease the processing time complexity it is necessary to find a method that gives a wedge or a slab index directly from y-coordinate with O(1) complexity.

Let us assume a situation shown in fig.8 where slabs are not determined according to y-coordinates of vertices of the given polygon but as "thin regular slices". It can be seen that if slices are "thit enough" there will be only two edges associated with each slab (if singular cases are not considered).

Let us define

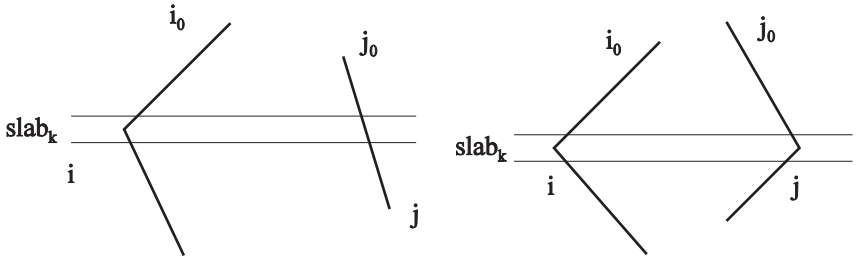
$$\delta = \min_{\substack{\forall_{i,j} \\ i \neq j}} \{ | y_i - y_j | \} \qquad \Delta = \max_{\forall_{i,j}} \{ | y_i - y_j | \}$$



Regular slab definition

Figure 8

Machine Graphics and Vision, Vol.5., No.3., pp. 483-494, , ISSN 1230-0535, 1996.



(a) Simple singularity (b) Singularity with δ=0

Possible cases when constructing AEL_M list (modified AEL)

Figure 9

Generally, the compexity of δ and Δ determination is O(N lg N), but because vertices define a convex polygon with known order of vertices the complexity is only O(N), see alg.4.

If slabs are regular with thickness smaller than δ then there will be usually 2 edges associated with each slab. If a vertex lies inside of a slab there will be three edges associated with that slab, see fig.9.a. Some special cases exist when $\delta = 0$, see fig.9.b. In those cases four edges are associated with the slab. In case of regular slabs it is possible to compute the index i of the relevant slab as

$$i = \frac{y_{max} - y}{\Delta} M$$

where M is a number of slabs, i.e.

$$M = \frac{y_{max} - y_{min}}{\delta} = \frac{\Delta}{\delta}$$

After substitution, the index of the slab for the given y-coordinate is determined as

$$i = (y_{max} - y)/\delta$$

Of course, it is necessary to evaluate condition

$$F_k(\mathsf{x}) \geq 0 \qquad \forall k$$

where k are indices of edges associated with the selected slab, i.e.

$k \in \{AEL_{i,1}, AEL_{i,2},\ resp. AEL_{i,3}, AEL_{i,4}$ if a slab contains a vertex $\}$

```
procedure PRECOMPUTE_MODIF; { O(N) preprocessing time }
begin  { algorithm for using regular slabs technique }
     PRECOMPUTE; { see alg.3 }
     { values y', Δ and AEL_k - determined by PRECOMPUTE procedure }
     δ = +∞;  { valute δ - a minimal defference in y-coordinates }
```

```
      for i := 1 to N - 1 do
         if δ > y'_i − y'_{i−1} then δ := y'_i − y'_{i−1};
      j := 0; i := 0; M := int(Δ/δ) + 1;
      while j ≤ k do      { compute Modified AEL - AEL_M }
      begin y := y_max − δ ⋆ i;
            AEL_M_{i,1} := AEL_{j,1}; AEL_M_{i,2} := AEL_{j,2};
            AEL_M_{i,3} := −1; AEL_M_{i,4} := −1; { element not used }
         if (y − δ) < y'_{j+1} then
         begin q := 2;
            if AEL_{j,1} ≠ AEL_{j+1,1} then
            begin AEL_M_{i,q} := AEL_{j+1,1}; q := q + 1; end;
            if AEL_{j,2} ≠ AEL_{j+1,2} then AEL_M_{i,q} := AEL_{j+1,2};
            j := j + 1;
         end { if };
         i := i + 1;   { i ≤ M, i.e. i-th slab is computed }
      end { while };
      δ_inv := 1/δ;
end { PRECOMPUTE_MODIF };


function POINT_IN_POLYGON (x: point): boolean;
begin { O(1) processing time }
      i := 0;  j := N - 1;     POINT_IN_POLYGON := false;
      if y ∉< y_min, y_max > then EXIT;
      { point outside of the polygon;  above or below }
      { determine a relevant regular slab index i }
      i := (y_max − y) ⋆ δ_inv;
      { now test only two edges from Active Edge List AEL }
      { F (x) should be implemented as a macro }
      if F_{AEL_M_{i,1}}(x) < 0 then EXIT;     { EXIT from procedure }
      if F_{AEL_M_{i,2}}(x) < 0 then EXIT;
      if AEL_M_{i,3} ≠ −1 then
{⋆} begin if F_{AEL_M_{i,3}}(x) < 0 then EXIT; { element not used }
{⋆}       if AEL_M_{i,4} ≠ −1 then
                  if F_{AEL_M_{i,4}}(x) < 0 then EXIT;
{⋆} end;
      POINT_IN_POLYGON := true;   { the point x is inside }
end { POINT_IN_POLYGON };
```

Algorithm 4

It can be seen that the proposed algorithm, see alg.4, has the following complexities

– for preprocessing time

O(N) for determination of δ, Δ,

O(M) for finding edges associated with slabs,

– for memory requirements

O(M) for storing associated edges for each slab

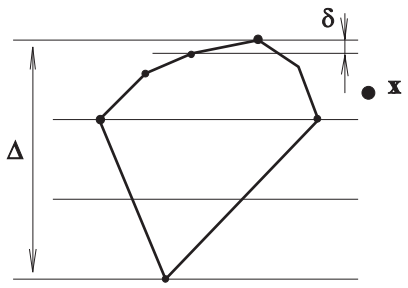
– for processing time

O(1) for index of a slab computation and determination whether the given point is inside of the given polygon.

It is obvious that preprocessing time complexity and memory requirements depend also on geometrical properties of the given polygon. If the ratio

$$M = \frac{\Delta}{\delta}$$

is very high it is possible to limit number of slabs. In this case we obtain more than four associated edges for a selected slab that might result into an algorithm with O(N) processing complexity, see fig.10.



If the point x is in that slab the algorithm complexity is O(N)

The influence of limited number of slabs to algorithm complexity

Figure 10

Let us assume that

$$\delta = 10^{-3} \qquad \text{and} \qquad \Delta = 10^{3}$$

then we would need

$$M = 10^{6}$$

slabs for polygon representation! On the other hand if we limit number of slabs to M =3 we can get for the third slab N - 2 edges to test, i.e. the algorithm would have complexity O(N) in this case. Nevertheless we can decide according to M which algorithm should be used, i.e. with O(lg N) or O(N) complexities according to convex polygon shape.

4. Experimental results

For experiments regular and irregular convex polygons were used and 5000 points were randomly generated outside and inside of the polygon. Results obtained in experiments are shown in tab. 1 – 4 and numbers are related to the processing time in seconds. All experiments were made on PC 486/50MHz. It is necessary to point out that if δ is taken as δ/q $(q > 1)$ the proposed algorithm is faster, see tab. 5. It is caused by the fact that if δ is taken smaller then it is not necessary to test the third or fourth edges, i.e. lines in alg.4 marked by $\{\star\}$ are not executed. Of course, it is necessary to see that decreasing δ value causes additional memory requirements.

N	3	4	5	10	20	50	100
Alg_1	1.04	1.32	1.65	3.18	6.15	15.22	30.32
Alg_3	0.99	0.99	1.10	1.05	1.26	1.42	1.53
Alg_4	1.15	1.10	1.32	1.31	1.27	1.10	1.10

Point inside of a regular polygon

Table 1

N	3	4	5	10	20	50	100
Alg_1	0.66	0.88	0.88	1.37	2.31	4.01	7.58
Alg_3	0.33	0.28	0.27	0.28	0.38	0.38	0.43
Alg_4	0.11	0.16	0.17	0.11	0.11	0.11	0.11

Point outside of a regular polygon

Table 2

N	3	4	5	10	20	50	100
Alg_1	0.77	1.33	1.64	3.19	6.15	9.17	12.24
Alg_3	0.89	0.99	0.94	1.20	1.27	1.32	1.43
Alg_4	1.16	1.16	1.10	1.16	1.15	1.10	1.10

Point inside of an irregular polygon

Table 3

N	3	4	5	10	20	50	100
Alg_1	0.61	0.82	0.93	1.87	3.57	4.45	5.93
Alg_3	0.38	0.44	0.49	0.61	0.66	0.66	0.66
Alg_4	0.11	0.11	0.11	0.11	0.10	0.11	0.11

Point outside of an irregular polygon

Table 4

q	1	2	4	10
speed-up	1.0	1.3	1.4	1.45

Choice of q to the proposed algorithm speed up

Table 5

5. Conclusion

The new modifications of Point-in-Polygon algorithm was developed for convex polygon case. The algorithm claims the preprocessing complexity O(N) and the processing complexity O(1), see tab.1. – 4. If number of slabs is limited, algorithm complexity depends on geometrical properties of the given polygon. The proposed O(1) algorithm is convenient for cases when many points are tested against a polygon that is constant. Such problems are often solved in geoscience programs in which a number of edges can be higher than 50 and number of tested points can reach 10^3. It is possible to show that test Point-in-convex polygon is k dual to the test whether a line intersects the convex polygon, i.e. dual to the line clipping problem. There is a hope that similar approach can be taken for developing a new line clipping algorithm with O(1) complexity. The presented approach shows also a possibility how some problems can be speed up without using parallel processing.

Acknowledgments

The author would like to express his thanks to Ms. I. Kolingerova for critical comments, Mr. P. Lederbuch for implementation and testing algorithms and to all who contributed to this work, especially to students of Computer Graphics courses at the University of West Bohemia in Plzen who stimulated this work and for many suggestions they proposed.

5. References

[CHE87a] Chen, M., Townsesnd, P.: Efficient and Consistent Algorithms for the Constainment of Points in Polygon and Polyhedra, EUROGRAPHICS'87 Conference Proceedings, 1987.

[HUB93a] Hubl, J.: A Point-by-point Clipping Algorithm, Technical Report, Muchen, 1993.

[LAN84a] Lane, J., Magedson, B., Rarick, M.: An Efficient Point in Polyhedron Algorithm, Computer Vision, Graphics and Image Processing, Vol. 26, pp. 118 – 125, 1984.

[PRE85a] Preperata, P. F., Shamos, M. I.: Computational Geometry, An Introduction, Springer Verlag, 1985.

[SAL78a] Salomon, K.: An Efficient Point-in polygon Algorithm, Computers & Geosciences, Vol. 4, pp. 173 – 178, 1978.

[SKA96a] Skala, V.: Line Clipping in E_2 with O(1) Processing Complexity, Accepted for publication in Computers & Graphics, Pergmon Press, Vol.20, No.4, 1996