# 2D LineSegment-Polygon-Intersection

# Part 1. Convex Polygon (Comprehensive Experiment)

## Introduction

Totally 9 different line segment to convex polygon testing algorithms are implemented and compared with each other to see the performance.

These algorithms are categorized into 4 groups:

1. Brute Forces approaches, which are simply based on 2D line segment to line segment test and can also be used for non-convex polygon.
2. Separating-Axis-Test approaches, with the similar principle of SAT-based polygon-polygon test described in *Schneider’s Geometric Tools for Computer Graphics* [1].
3. Binary-Search approaches. We implement *O’Rourke’s* solution [2] that divides the polygon into two chains by finding extreme points, and provide **another improved algorithm which is our main work** (section 2.3.2).
4. Other methods, like LIFO (Last-In-First-Out) that is borrowed from *Eric’s Fast Ray-Convex Polyhedron Intersection*[3].

A test program is provided to do the performance comparisons among those algorithms. Vertices number of the polygon is an important factor, and also the Hit-Ratio, like *Marta’s* code [4]. The convex polygons are randomly generated by the strategy in *Eric’s Point-In-Polygon* [5] test code. Besides, we use one of the brute force methods to generate “true” results.

The following sections will describe the algorithms and also the test program in detail.

## Algorithms

### Brute Forces approaches

There’re 2 algorithms:

1. Basic solution.

This method takes 2 steps:

1. Test whether any of the 2 endpoints of the line segment is inside the polygon. If true, stop; else, goto next. Crossings algorithm is used for Point-In-Polygon test.
2. Loop the edge of the polygon and test to the line segment until intersection is found. Antonio’s method is used for 2 line segments test.
3. Optimized solution.

This method is almost the same to the last one except adding separation tests along x and y axes to line segments test for early rejection. From the test results later on, this improvement is very useful.

Actually, the 2 methods above can also be used for non-convex polygons.

### 2.2 Separating-Axis-Test approaches

SAT algorithm is one of the classic approaches to solve (convex) polygon-polygon intersection [1]. The normal vectors of polygon’s edges are used as the separation axes. The line segment (p0, p1) can be considered as degenerate polygon, and its normal vector is one of the separation axes. The SAT can be applied as follows:

1. Simple SAT.

For each separation axis, calculate the projection intervals of the polygon and the line segment, and test whether the two overlap or not (Figure. 1a).

Projection interval of the polygon is got by projecting **all** the vertices to the separations axis and finding the min and max. So, the algorithm is overall O(N\*N) (N is the vertices number of the polygon).

1. Optimized SAT.

Part 1: Assuming that the vertices of the polygon are ordered CCW, the normal vector n(i) of each edge e(i) points outwards and makes the whole polygon behind e(i) (Figure. 1b). So, the projection interval of the polygon on n is [u, 0](u < 0) if translating the origin to one vertex of e(i). If the projection of p0 or p1 is less than 0, discard this axis and goto the next one; if both are greater than 0, separation is true and stop processing. Since it is no need to care the exact value of u, projection calculation of the polygon’s vertices is discarded, and the complexity is O(N).

Part 2: For normal vector n of (p0, p1), loop the vertices of the polygon until finding the projection interval strides 0 (translating the origin to p0 or p1). If the polygon is completely on one side of the line segment, separation is true and stop processing. The complexity is O(N) in the worst case.

This method is greatly improved compared with the simple one, and the overall complexity is O(N).

1. Finding Extreme Points.

The only change to the previous one is, for Part 2, binary search is applied to finding minimum and maximum extreme points of the convex polygon [6]. If the minimum extreme is on the positive side of (p0, p1) or maximum extreme is on the negative side of (p0, p1), separation is true. The complexity is O(logN).

This algorithm only improves one part of the previous one, and the overall complexity is still O(N)(= O(N) + O(logN)).

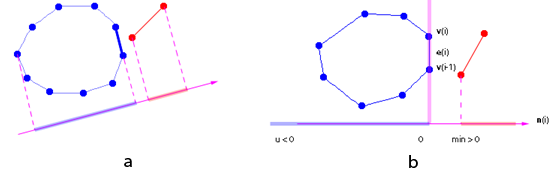


Figure 1. Separating-Axis-Test for LineSegment-Polygon intersection. (a. Simple approach to compute projection intervals; b. optimized approach)

### 2.3 Binary-Search approaches.

It is obviously that the line segment can have at most 2 intersections with the convex polygon. So, we can firstly find the 2 edges of the polygon which probably intersect the line segment (p0, p1), and then do line segment to line segment test.

1. *O’Rourke’s* solution [2].

This algorithm proceeds in 2 steps:

1. Divide the convex polygon into 2 chains which are monotone (one increasing and the other decreasing) along the normal vector n of (p0, p1). The two chains are connected at the minimum and maximum extreme points which are found by means of binary search the same to the method above (Figure 2a). The complexity of this step is O(logN).
2. For each chain, since it is monotone along n, apply binary search to find the edge (v0, v1) that may have chance to intersect (p0, p1) (v0, v1 will be on different sides of the infinite line of (p0, p1). See Figure 2b). Finally, test whether (v0, v1) intersects (p0, p1) (Figure 2b). The complexity is also O(logN).

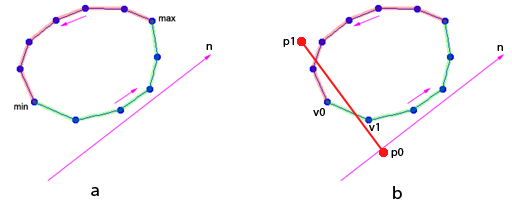


Figure 2. O’Rourke’s binary search solution to LineSegment-Polygon test. (a. 2 chains monotone along n and connected by minimum and maximum extreme points; b. Binary search in the increasing chain to find edge (v0, v1) that is likely to intersect (p0, p1))

1. **Improved approach**.

The basic strategy of this optimized algorithm is almost the same to the O’Rourke’s original one, that is to find 2 chains, and each chain has at most one intersection with the line segment L, so binary partition method can be taken on each chain to search for the intersection. The key difference between the two is: O’Rourke’s algorithm divides the whole polygon into 2 chains connected by the minimum and maximum extreme points along the normal vector n of L. So, each chain is guaranteed to be monotone along **n** and can have at most one intersection with L. However, the new algorithm does NOT require the chain to be monotone along **n**, but ONLY ensure that the chain has at most one intersection with L. Then, binary search can still be applied to check the intersection. E.g., on the convex polygon in Figure 3.1, Start and End are at different sides of L, so there’ll one and only one intersection between chain [Start, End] (the green one) and the infinite line along L. Therefore, binary search can be applied even though the chain is not monotone with n.

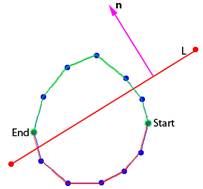


Figure 3.1. Non-monotone chains can also be used for binary search.

To find the 2 chains, O’Rourke’s original “binary search to find extreme point on convex polygon” is improved to achieve Early Escape. The following cases show the core of the algorithm:

Pre-condition: Let v0 on the negative side of L (if v0 is found on the positive side, let n = -n );

Case 1 (figure 3.2a):

Step1: Start binary search to find the maximum extremum along n on the convex polygon; StartVertex = EndVertex = v0;

Step2: MiddleVertex = v4; v4 is on the positive side of L, so stop (early escape) and get the 2 chains: [v0, …, v4] and [v4, … v0] (counter clockwise).

Case 2 (figure 3.2b):

Step1: Start binary search to find the maximum extremum along n on the convex polygon; StartVertex = EndVertex = v0;

Step2: MiddleVertex = v4; v4 is also on the negative side and is not the maximum extremum, so halve the chain by letting EndVertex = v4 (ensure new chain contains the maximum extremum); Continue binary search;

Step3: MiddleVertex = v2; v2 is on the positive side, so stop (early escape) and get the 2 chains: [v0,…, v2] and [v2, …, v4] (counter clockwise).

Case3 (figure 3.2c ):

Step1: Same as Case 2;

Step2: Same as Case2;

Step3: MiddleVertex = v2; v2 is still on the negative side but v2 is already the maximum extremum, so stop and return false since the polygon has no chance to intersect with L.

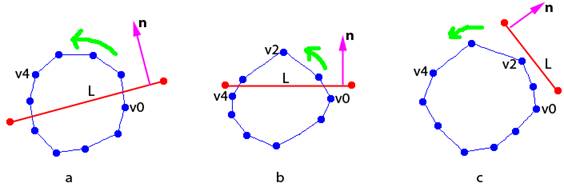


Figure 3.2. 3 main cases during finding 2 chains in improved algorithm.

### 2.4 Other algorithms

There’re 2 uncategorized algorithms:

1. Integrate Point-In-Polygon and Extremes.

2 steps:

1. Do Point-In-Polygon test for p0 and p1, if any one inside, return (Figure 4a). For convex polygon, optimized crossings algorithm can be used which stops until finding 2 (possible) crossings.
2. If p0, p1 are both outside, find the 2 extreme points (min, max) of the polygon along the normal vector, and test whether (min, max) intersects (p0, p1) (Figure 4b).

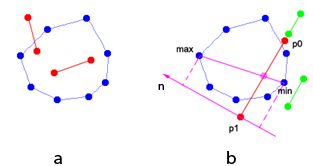


Figure 4. Point-In-Polygon and Extremes integration algorithm. (a. Intersection is true when point in polygon; b. When p0, p1 are both outside, it will intersect the polygon **if and only if** it intersect (min, max), the line segment of the 2 extreme points. The red segment intersects the polygon and the green ones fail.)

1. Last-In-First-Out (LIFO) approach.

This algorithm takes the same strategy as that in *Eric’s Fast Ray-Convex Polyhedron Intersection*[3]. It calculates intersection between the line segment and the infinite line of polygon’s edge, update the maximum entering value (last in) or the minimum leaving value (first out). If find leaving is before entering, stop processing and report no intersection.

## Test Program

During one time execution, the test program calculates the total time consumed by each algorithm for pn \* sn times tests, where pn and sn represent the number of polygon and line segment respectively. Vertices count and hit ratio are the two key factors for the performance contest among algorithms, and a pair of (c, r) will be used in one time execution.

We use regular polygon for the test. pn regular polygons (with its center at the origin) of c vertices are generated and each polygon rotates a random angle round the origin, just like that in *Eric’s Point-In-Polygon* [5] code. For each polygon, generate sn line segments with (r \* sn) ones intersect the polygon (this is learning from *Marta’s* code [4] and one of the brute force method is used for the basic algorithm). The endpoints of the line segment are all randomly generated within 1.2 times extended bounding box of the polygon. After generating the polygons and line segments, run each algorithm and report the total time it takes.

### 3.1 Files overview

1. Data structures (Math.h/cpp).
2. LineSegment-Polygon test algorithms (Algorithm.h/cpp):
3. Brute Force:

SegmentPolygonIntersection\_BruteForce();

SegmentPolygonIntersection\_OptimizedBruteForce();

1. SAT:

SegmentPolygonIntersection\_SimpleSAT();

SegmentPolygonIntersection\_OptimizedSAT();

SegmentPolygonIntersection\_ExtremeSAT();

1. Binary Search:

SegmentPolygonIntersection\_BinarySearchByExtremes();

SegmentPolygonIntersection\_BinarySearch();

1. Others:

SegmentPolygonIntersection\_ExtremePlusPointInside();

SegmentPolygonIntersection\_LIFO();

1. Test program (Untility.h/cpp)
2. GeneratePolygonSegments(): Generate polygons and line segments;
3. RunSegmentPolygonAlgorithms(): Run LineSegment-Polygon algorithms and output the time used;
4. Main program (Test.cpp).

Figure 5 shows the execution result of the main test program. Given vertex count, hit ratio, polygon number and line segment number (and regular polygon radius), run all algorithms and get the total time for each of them.

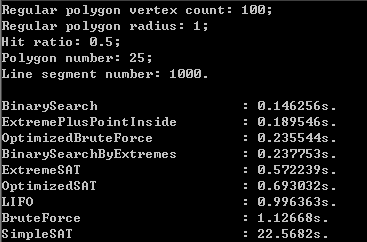


Figure 5. Program execution result.

### 3.2 Experimental results

To compare the performances of the 9 algorithms, we use regular polygons with its vertex count ranging from small to large (3, 5, 10, 20, 50, 100, 200, 500) and hit ratio from low to high (0.0, 0.2, 0.5, 0.8, 1.0). Currently, the experiments are all done on one laptop with Intel Core2 T5600 1.83GHz. Experimental results are shown in the tables below (red, green and blue represent the first, the second and the third most efficient algorithm).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Regular polygon vertex count: 3**;  Regular polygon radius: 1;  Polygon number: 25;  Line segment number: 1000. | | | | | |
| **Hit Ratio :** | **0.0** | **0.2** | **0.5** | **0.8** | **1.0** |
| BruteForce | 0.121643s. | 0.110953s. | 0.0928984s. | 0.0776392s. | 0.0656223s. |
| OptimizedBruteForce | 0.0903039s. | 0.0869552s. | 0.0783555s. | 0.0741812s. | 0.0678435s. |
| SimpleSAT | 0.093546s. | 0.107284s. | 0.122677s. | 0.139626s. | 0.150048s. |
| OptimizedSAT | 0.0763195s. | 0.080446s. | 0.0831301s. | 0.0888968s. | 0.0906705s. |
| ExtremeSAT | 0.104532s. | 0.110098s. | 0.115502s. | 0.123315s. | 0.127541s. |
| BinarySearchByExtremes | 0.113976s. | 0.116872s. | 0.119802s. | 0.123356s. | 0.125594s. |
| BinarySearch | 0.0917812s. | 0.0940167s. | 0.0960463s. | 0.0983321s. | 0.101689s. |
| ExtremePlusPointInside | 0.123757s. | 0.116276s. | 0.100047s. | 0.0881461s. | 0.0777906s. |
| LIFO | 0.07925s. | 0.086599s. | 0.0866264s. | 0.0914524s. | 0.0947995s. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Regular polygon vertex count: 5**;  Regular polygon radius: 1;  Polygon number: 25;  Line segment number: 1000. | | | | | |
| **Hit Ratio :** | **0.0** | **0.2** | **0.5** | **0.8** | **1.0** |
| BruteForce | 0.16289s. | 0.14371s. | 0.116128s. | 0.0856606s. | 0.0669124s. |
| OptimizedBruteForce | 0.0947271s. | 0.0883894s. | 0.0820121s. | 0.0729149s. | 0.0662961s. |
| SimpleSAT | 0.111003s. | 0.138824s. | 0.182786s. | 0.226348s. | 0.254996s. |
| OptimizedSAT | 0.0834743s. | 0.0875863s. | 0.0956708s. | 0.102685s. | 0.107543s. |
| ExtremeSAT | 0.111808s. | 0.119086s. | 0.132502s. | 0.144452s. | 0.152096s. |
| BinarySearchByExtremes | 0.123519s. | 0.1282s. | 0.134289s. | 0.140863s. | 0.144506s. |
| BinarySearch | 0.0963812s. | 0.0978912s. | 0.102268s. | 0.106029s. | 0.108743s. |
| ExtremePlusPointInside | 0.141217s. | 0.124171s. | 0.105835s. | 0.0872731s. | 0.0741835s. |
| LIFO | 0.0961265s. | 0.0995501s. | 0.108123s. | 0.11674s. | 0.121189s. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Regular polygon vertex count: 10**;  Regular polygon radius: 1;  Polygon number: 25;  Line segment number: 1000. | | | | | |
| **Hit Ratio :** | **0.0** | **0.2** | **0.5** | **0.8** | **1.0** |
| BruteForce | 0.26367s. | 0.226712s. | 0.168626s. | 0.110258s. | 0.0734996s. |
| OptimizedBruteForce | 0.107361s. | 0.100371s. | 0.0878424s. | 0.0752056s. | 0.0686428s. |
| SimpleSAT | 0.136011s. | 0.242959s. | 0.398326s. | 0.557078s. | 0.672847s. |
| OptimizedSAT | 0.104628s. | 0.113777s. | 0.127205s. | 0.141576s. | 0.150886s. |
| ExtremeSAT | 0.126709s. | 0.143398s. | 0.166729s. | 0.188786s. | 0.20623s. |
| BinarySearchByExtremes | 0.142095s. | 0.147394s. | 0.155511s. | 0.163665s. | 0.176298s. |
| BinarySearch | 0.113004s. | 0.1098s. | 0.112417s. | 0.114335s. | 0.118326s. |
| ExtremePlusPointInside | 0.158163s. | 0.141863s. | 0.117789s. | 0.091192s. | 0.075016s. |
| LIFO | 0.127514s. | 0.13832s. | 0.156012s. | 0.174637s. | 0.190666s. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Regular polygon vertex count: 20**;  Regular polygon radius: 1;  Polygon number: 25;  Line segment number: 1000. | | | | | |
| **Hit Ratio :** | **0.0** | **0.2** | **0.5** | **0.8** | **1.0** |
| BruteForce | 0.466942s. | 0.387853s. | 0.275974s. | 0.159161s. | 0.0837875s. |
| OptimizedBruteForce | 0.133676s. | 0.120413s. | 0.103126s. | 0.0849083s. | 0.0739181s. |
| SimpleSAT | 0.234333s. | 0.609501s. | 1.16774s. | 1.72959s. | 2.10541s. |
| OptimizedSAT | 0.143579s. | 0.161055s. | 0.189649s. | 0.21683s. | 0.235277s. |
| ExtremeSAT | 0.14922s. | 0.177314s. | 0.221016s. | 0.264698s. | 0.294848s. |
| BinarySearchByExtremes | 0.163859s. | 0.168675s. | 0.179398s. | 0.18935s. | 0.197245s. |
| BinarySearch | 0.120652s. | 0.120539s. | 0.122417s. | 0.122946s. | 0.125206s. |
| ExtremePlusPointInside | 0.183319s. | 0.160762s. | 0.130335s. | 0.0985081s. | 0.0802795s. |
| LIFO | 0.182436s. | 0.208446s. | 0.248301s. | 0.289062s. | 0.315612s. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Regular polygon vertex count: 50**;  Regular polygon radius: 1;  Polygon number: 25;  Line segment number: 1000. | | | | | |
| **Hit Ratio :** | **0.0** | **0.2** | **0.5** | **0.8** | **1.0** |
| BruteForce | 1.06586s. | 0.876601s. | 0.595721s. | 0.309784s. | 0.122283s. |
| OptimizedBruteForce | 0.21365s. | 0.189388s. | 0.153532s. | 0.116642s. | 0.0922039s. |
| SimpleSAT | 0.671185s. | 2.79606s. | 5.98951s. | 9.19374s. | 11.3157s. |
| OptimizedSAT | 0.269843s. | 0.311622s. | 0.378207s. | 0.443571s. | 0.489739s. |
| ExtremeSAT | 0.184105s. | 0.252797s. | 0.360351s. | 0.462351s. | 0.533637s. |
| BinarySearchByExtremes | 0.188067s. | 0.196144s. | 0.210204s. | 0.222514s. | 0.231869s. |
| BinarySearch | 0.13496s. | 0.134402s. | 0.134381s. | 0.133266s. | 0.133214s. |
| ExtremePlusPointInside | 0.222306s. | 0.195315s. | 0.155859s. | 0.11804s. | 0.0936421s. |
| LIFO | 0.344396s. | 0.417532s. | 0.527684s. | 0.635512s. | 0.709466s. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Regular polygon vertex count: 100**;  Regular polygon radius: 1;  Polygon number: 25;  Line segment number: 1000. | | | | | |
| **Hit Ratio :** | **0.0** | **0.2** | **0.5** | **0.8** | **1.0** |
| BruteForce | 2.06604s. | 1.69196s. | 1.11992s. | 0.556522s. | 0.182707s. |
| OptimizedBruteForce | 0.34636s. | 0.30118s. | 0.232905s. | 0.167818s. | 0.123617s. |
| SimpleSAT | 1.9433s. | 10.1306s. | 22.4615s. | 34.7381s. | 42.9872s. |
| OptimizedSAT | 0.473889s. | 0.559138s. | 0.690157s. | 0.820093s. | 0.909169s. |
| ExtremeSAT | 0.229843s. | 0.365986s. | 0.570676s. | 0.771323s. | 0.910231s. |
| BinarySearchByExtremes | 0.208919s. | 0.218426s. | 0.234117s. | 0.249676s. | 0.259982s. |
| BinarySearch | 0.147612s. | 0.145654s. | 0.144692s. | 0.142659s. | 0.140968s. |
| ExtremePlusPointInside | 0.263134s. | 0.233217s. | 0.185272s. | 0.13977s. | 0.110265s. |
| LIFO | 0.616095s. | 0.763835s. | 0.987047s. | 1.21468s. | 1.35618s. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Regular polygon vertex count: 200**;  Regular polygon radius: 1;  Polygon number: 25;  Line segment number: 1000. | | | | | |
| **Hit Ratio :** | **0.0** | **0.2** | **0.5** | **0.8** | **1.0** |
| BruteForce | 4.07408s. | 3.30609s. | 2.18579s. | 1.06042s. | 0.304563s. |
| OptimizedBruteForce | 0.608991s. | 0.523862s. | 0.397052s. | 0.270962s. | 0.183101s. |
| SimpleSAT | 6.50202s. | 38.5572s. | 87.0886s. | 135.495s. | 167.941s. |
| OptimizedSAT | 0.885784s. | 1.05443s. | 1.31863s. | 1.57525s. | 1.75071s. |
| ExtremeSAT | 0.306201s. | 0.573823s. | 0.972973s. | 1.3788s. | 1.64866s. |
| BinarySearchByExtremes | 0.231256s. | 0.242208s. | 0.260109s. | 0.27697s. | 0.290816s. |
| BinarySearch | 0.160886s. | 0.158418s. | 0.155855s. | 0.152231s. | 0.150208s. |
| ExtremePlusPointInside | 0.330157s. | 0.291609s. | 0.236763s. | 0.180063s. | 0.142744s. |
| LIFO | 1.16408s. | 1.45209s. | 1.90979s. | 2.36056s. | 2.65657s. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Regular polygon vertex count: 500**;  Regular polygon radius: 1;  Polygon number: 25;  Line segment number: 1000. | | | | | |
| **Hit Ratio :** | **0.0** | **0.2** | **0.5** | **0.8** | **1.0** |
| BruteForce | 10.0524s. | 8.17592s. | 5.36704s. | 2.55466s. | 0.682019s. |
| OptimizedBruteForce | 1.39726s. | 1.19051s. | 0.887702s. | 0.575355s. | 0.370982s. |
| SimpleSAT | 33.1144s. | 235.087s. | 534.313s. | 835.007s. | 1034.66s. |
| OptimizedSAT | 2.10076s. | 2.53743s. | 3.19322s. | 3.83521s. | 4.27294s. |
| ExtremeSAT | 0.485934s. | 1.16874s. | 2.1617s. | 3.15782s. | 3.81582s. |
| BinarySearchByExtremes | 0.257636s. | 0.272701s. | 0.29298s. | 0.313211s. | 0.328198s. |
| BinarySearch | 0.176758s. | 0.173088s. | 0.168737s. | 0.163641s. | 0.160269s. |
| ExtremePlusPointInside | 0.489966s. | 0.441871s. | 0.36292s. | 0.290154s. | 0.241438s. |
| LIFO | 2.75089s. | 3.53137s. | 4.67963s. | 5.80999s. | 6.56319s. |

## 4. References

1. P.J.Schneider, D.H.Eberly, “*Geometric Tools for Computer Graphics*”, Chapter 7.7.2.
2. O’Rourke, “*Computational Geometry in C*”, Cambridge University Press, Section 7.9.1, P271~272, 1998.
3. Eric Haines, "*Fast Ray-Convex Polyhedron Intersection*", Graphics Gems II, 1994.
4. Marta Löfstedt and Tomas Akenine-Möller, "An Evaluation Framework for Ray-Triangle Intersection Algorithms", Journal of graphics tools, 2004.
5. <http://www.erichaines.com/ptinpoly/>
6. O’Rourke, “*Computational Geometry in C*”, Cambridge University Press, Section 7.9, P269~271, 1998.

# Appendix A Low Hit-Ratio Experiments

All 9 algorithms are adapted to fit the precondition that 2 endpoints of the line segment are outside the convex polygon (some algorithms remain the same, like SAT-based ones). Below shows the experimental results in case of very low hit ratio.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Hit ratio: 0.01**;  Regular polygon radius: 1;  Polygon number: 100;  Line segment number: 10000. | | | | | |
| **Vertex count:** | **3** | **5** | **10** | **20** | **50** |
| BruteForce | 4.30268s. | 5.73424s. | 9.3062s. | 16.4852s. | 37.9874s. |
| OptimizedBruteForce | 3.26898s. | 3.26107s. | 3.62062s. | 4.44713s. | 6.91059s. |
| SimpleSAT | 3.74526s. | 4.34375s. | 5.39189s. | 9.40679s. | 28.7836s. |
| OptimizedSAT | 3.01204s. | 3.19377s. | 3.90356s. | 5.37994s. | 9.91069s. |
| ExtremeSAT | 4.05572s. | 4.29115s. | 4.81738s. | 5.59694s. | 7.01755s. |
| BinarySearchByExtremes | 4.28052s. | 4.61091s. | 5.21784s. | 5.97364s. | 6.94913s. |
| BinarySearch | 3.36419s. | 3.51365s. | 3.89066s. | 4.35844s. | 4.89046s. |
| ExtremePlusPointInside | 4.42859s. | 4.80664s. | 5.43994s. | 6.17468s. | 7.05753s. |
| LIFO | 3.19644s. | 3.67574s. | 4.80158s. | 6.94147s. | 13.0349s. |

The table below shows comparison between Axis and AQTime.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Regular polygon vertex count: 20;  Regular polygon radius: 1;  Polygon number: 100;  Line segment number: 10000. | | | | |
| Experiments below are done without running AQTime. | | | | |
| **Experiment :** | **1** | **2** | **3** | **4** |
| BruteForce | 16.4575s. | 16.3777s. | 16.3969s. |  |
| OptimizedBruteForce | 4.43147s. | 4.37403s. | 4.37933s. |  |
| SimpleSAT | 9.46294s. | 9.39374s. | 9.44068s. |  |
| OptimizedSAT | 5.48725s. | 5.43333s. | 5.44465s. |  |
| ExtremeSAT | 5.62675s. | 5.56971s. | 5.60965s. |  |
| BinarySearchByExtremes | 6.09795s. | 6.05882s. | 6.08648s. |  |
| BinarySearch | 4.38231s. | 4.32216s. | 4.33865s. |  |
| ExtremePlusPointInside | 6.17853s. | 6.12433s. | 6.16183s. |  |
| LIFO | 7.00966s. | 6.96349s. | 6.98068s. |  |
| Experiments below are done while running AQTime. | | | | |
| **Experiment :** | **1** | | **2** | |
| **Axis** | **AQTime** | **Axis** | **AQTime** |
| BruteForce | 17.3049s. | 14.6027s. | 17.3108s. | 1.460436s. |
| OptimizedBruteForce | 5.3027s. | 2.62383s. | 5.29741s. | 2.61663s. |
| SimpleSAT | 10.2288s. | 7.51396s. | 10.2653s. | 7.55150s. |
| OptimizedSAT | 6.386s. | 3.69886s. | 6.38957s. | 3.70194s. |
| ExtremeSAT | 6.58711s. | 3.86651s. | 6.57807s. | 3.85470s. |
| BinarySearchByExtremes | 7.09141s. | 4.37580s. | 7.06709s. |  |
| BinarySearch | 5.28968s. | 2.59436s. | 5.27969s. | 2.58098s. |
| ExtremePlusPointInside | 7.05685s. | 4.33323s. | 7.04529s. | 4.31613s. |
| LIFO | 7.89514s. | 5.20938s. | 7.89035s. | 5.20155s. |