**Algorithm Type**: Extreme Waypoint Search Algorithm

**Algorithm Description**:

It is often useful to find an extreme point of a 2D polygon. For example, the vertices with maximal or minimal x-coordinates or y-coordinates define a polygon's bounding box. More generally, one might want to find an extreme point in an arbitrary direction. For any set of n points, such as an n-vertex polygon, this is easily done in O(n) time by testing each point against the previously found extreme. But, for the special case of a convex polygon, an adaptation of binary searching can find the extreme point in only O(log-n) time. We describe this algorithm in detail for an arbitrary query direction vector u. Also, we show how this algorithm can be applied to compute the distance from a convex polygon to a line.

Note that the convex hull of a point set or polygon is precisely the collection of extreme points in all possible directions. Thus, if several extreme point queries are expected for an arbitrary polygon, it may make sense to first compute its convex hull, and then do queries on this hull in O(log-h) time, where h.le.n is the number of hull vertices. For 2D simple polygons, the convex hull can be found efficiently in O(n) time.

**Algorithm Pseudo-code**:

The straightforward brute-force way to find an extreme point is to test all points incrementally, and remember the current extreme for the points tested so far. Each new point considered only has to be compared to the current extreme one. This works for any set of n points, and is clearly an O(n) algorithm. The algorithm for finding the extreme maximum and minimum relative to u is simple, as shown in the following pseudo-code.

Input: W = {V0,V1,...,Vn-1}  is a set of n points  
       **u** = a direction vector

Put: max = min = 0  
for each point Vi in {V1,...,Vn-1}  (i=1,n-1)  
{  
    if (**u** · (Vi- Vmax) > 0) {  // Viis above the prior max  
        max = i;                // new max index = i  
        continue;               // a new max can't be a new min  
    }  
    if (**u** · (Vi- Vmin) < 0)    // Viis below the prior min  
        min = i;                // new min index = i  
}  
  
return max = index of max vertex

For an arbitrary set of points without structure, this is the best one can do. For a set of n points, this algorithm will perform (n-1) dot product comparisons to compute the maximum of a set.