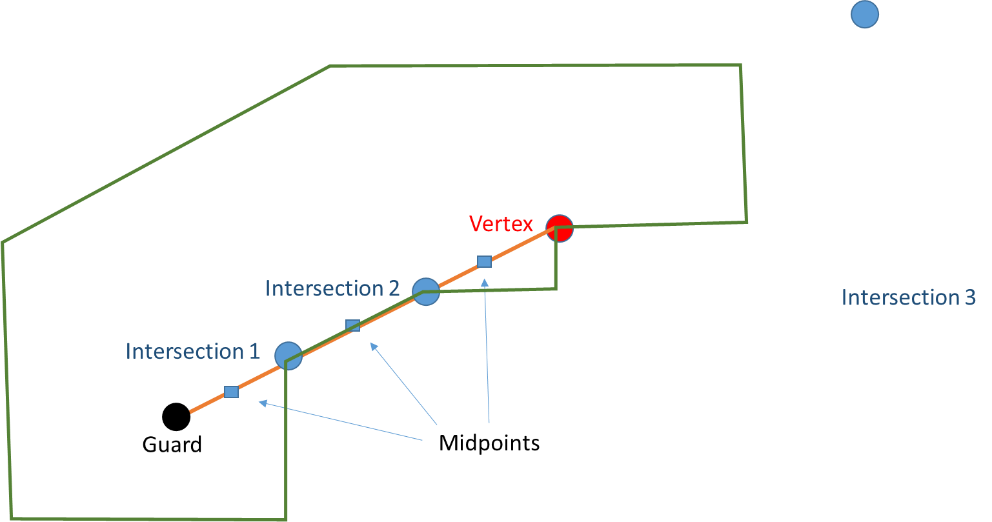
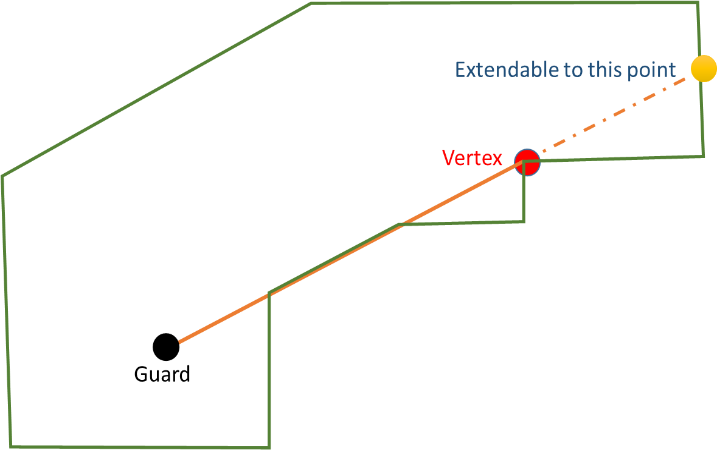
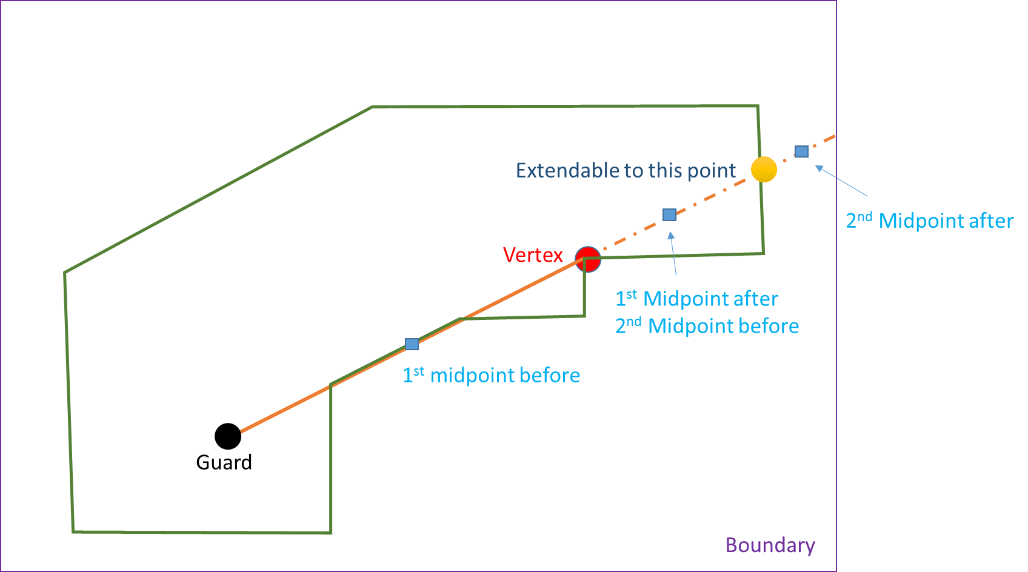
**Finding visibility polygons for each guard.**

* Get 1. Polygon object containing attributes its vertices and lines attributes, 2. Guard positions
* Create array to store visibility polygons for every guard.
* Iterate through the guards and for each guard find the reachable vertices and then the points that are reachable beyond them if there happen to be any:
* Find the vertices that are visible/reachable:

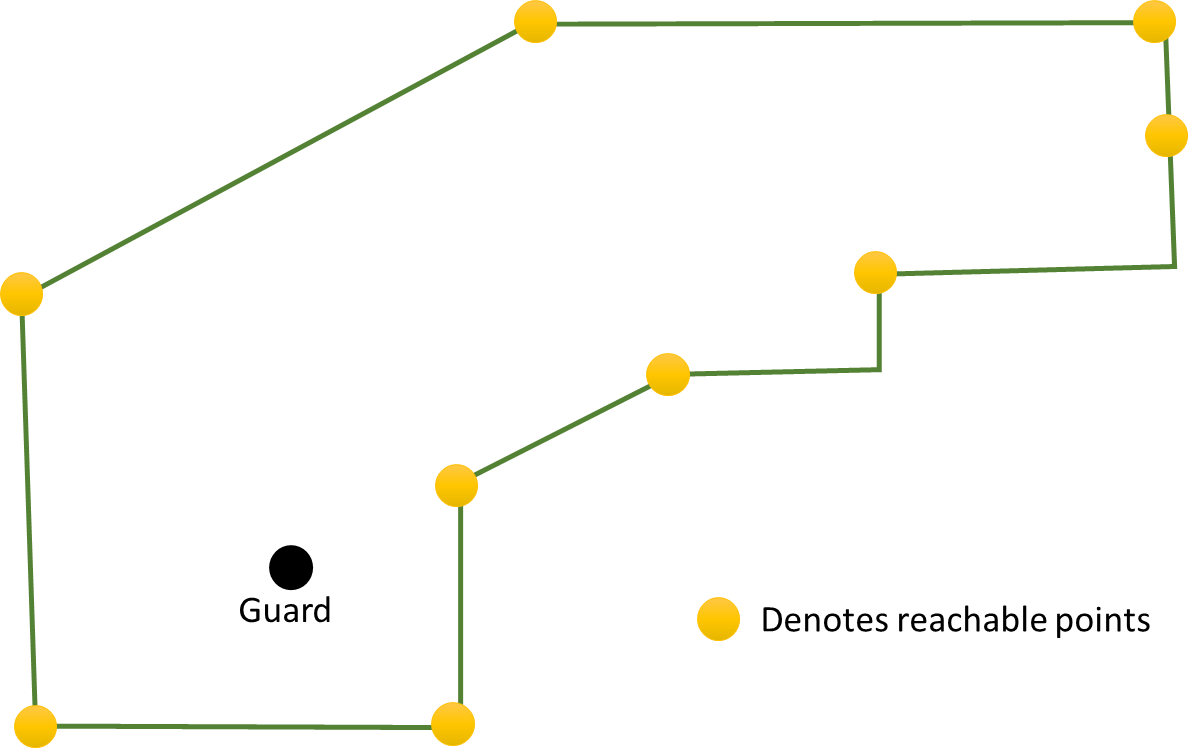
1. For each vertex, create a line segment from the guard to the vertex
2. Find all the intersection points on this line segment with any line segment from the polygon
3. Sort these intersection points by the distance they are away from the guard – in ascending order.
4. Iterate through these intersection points in the sorted order and see if the consecutive midpoints lie in the polygon – if any of them do not, the vertex is not visible. Otherwise, the vertex is visible and is pushed to the guard’s list of reachable vertices.

\*Orange line is the guard to vertex line segment created in step 1.

* For every reachable vertex, check if you can extend the ray from the guard beyond the reachable vertex to an edge on the polygon.
  1. Find the gradient of the line between the guard and the vertex
  2. Use this to create a line from the vertex to a point beyond the boundary of the polygon.
  3. Find the intersection points on the line
  4. Sort these intersection points in ascending order
* Have a list with the intersection points in sorted order with the guard and the vertex then inserted as the head of the list.
* Iterate through the list to see if the midpoint of the ith and i+1th index and i+1th and i+2th index both lie in the polygon. If either of them do not, then the point at the i+1th index is the furthest the ray can be extended to. Otherwise continue to the next iteration of the for-loop.

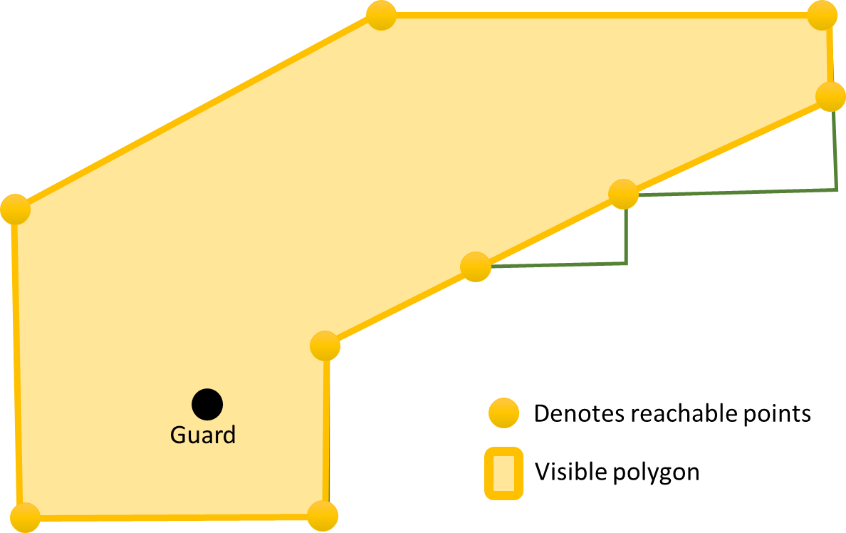
For instance when considering the vertex highlighted in red we find that the midpoint before and after - both lie in the polygon - so we continue on to the next iteration (i.e. the next intersection point – in the diagram highlighted as yellow). Here, we find that the midpoint before lies inside the polygon however, the midpoint that immediately follows does not, so hence this point would have to be the furthest point the ray can be extended to.

* Now that we have the furthest point it can be extended to. Check if the vertex that has been extended is already in the list of reachable points. If it isn’t, then this point is pushed to the list of points in the visible polygon.
* Check also if the extended point (that was just found) - is already in the list of reachable points. If it is not, this point is also pushed to the list of points in the visible polygon.
* We do this to avoid having duplicate points in the list of reachable points.

\*We now have all the extendable points in an unsorted array:

* + Sort the extended points into an array:
* Iterate through all of the lines of the polygon in (anti-clockwise) order and for each line:
  1. Find the points in the unsorted extended points array that lie on the line
  2. Sort them in ascending order according to the distance they are away from the start of the line.
  3. Push into the sorted array.
* Remove any duplicate points from the sorted array.
  + Push sorted array into array consisting of visible polygons for every guard.

\*We now have an array of the visibility polygons of each guard.



**Part 1: Finding minimal set of guards (positions) for a polygon**

1. Find the

* Using

1. Find a refutation

* Iterate
  + - For line in

**Part 2: Finding a refutation point**

1. Find the vertices of the non-visible areas of the polygon (given all the visible polygons for each guard)

* Using the JavaScript GPC library we compute the difference between the whole polygon and each visible polygon. And this gives us the vertices of the non-visible areas of the polygon.

1. Find a refutation point within a non-visible area

* Iterate through the non-visible areas to find a refutation point
  + - For each line in the non-visible area find the midpoint of the line, call this line x.
    - Iterate through each vertex in the non-visible area and create a line to that vertex from point x. Find the midpoint of this line and see if it falls within the polygon using the ray casting algorithm.
    - If a point is found to lie in the non-visible area, return this point.