Visibility graph for off-road pathfinding

Denis Kozub

1. Introduction

The problem of off-road routing can be simplified to pathfinding on a plane with polygonal and polylinear obstacles. In this paper only polylinear obstacles are discussed. To solve this problem, a structure called *visibility graph* is built. Visibility graph is called *reduced* if its edges only include *supporting lines* from a point to a polygon. It can be proved that a reduced visibility graph can always provide an optimal route on a map with polygonal obstacles.

2. Problem statement

Given a set of **h** polygons with $\mathbf{n}=\sum_{i=1}^{h}\mathbf{n_i}$ vertices the goal is to build a reduced visibility graph on a given area with **O(nh log n)** time complexity. It is motivated by the desire to build routes in real-time, without the user waiting for computation. Existing algorithms [1] [2] [3] are either slower or too hard to implement or build a non-reduced visibility graph. The described algorithm achieves the set complexity while building a correct reduced visibility graph.

3. Dynamic graph

An important feature of the algorithm is the ability to find vertices of the visibility graph, incident to a given point without building the graph itself. This would speed up the pathfinding algorithm, since in most cases heuristic algorithms do not discover all nodes of the graph. This feature is inherent to naive and sweeping line algorithms, but cannot be achieved by faster algorithms, which require triangulation.

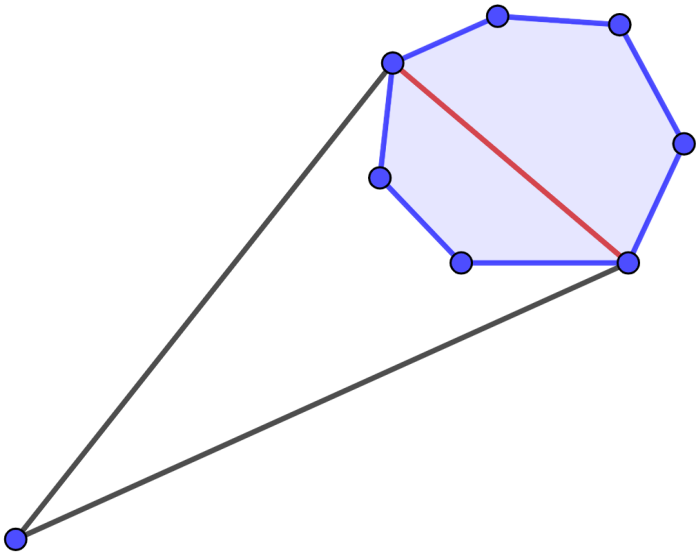
4. Precomputing

Precomputing includes building a convex hull for each polygon using Chan algorithm [4] with time complexity $\sum_{i=1}^{h} \mathbf{O(n_i \log h_i)}$ and finding polar angles to each vertex or a polygon from one of the vertices of a polygon for each polygon, which takes **O(n)** time.

5. Algorithm explanation

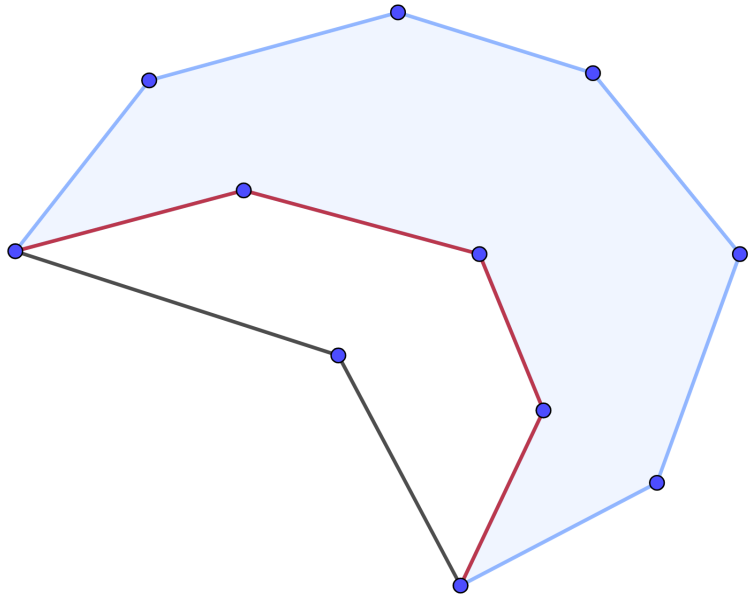
The visibility graph algorithm for each point can be divided into two parts: finding supporting lines to each polygon and building a visibility graph for these lines. Time complexity for both parts should not exceed **O(h log n)**.

Step 1: Given a point, a polygon with $\mathbf{n_i}$ vertices and its convex hull (to be precise, from now on "convex hull" stands for its closure), define whether a point is strictly inside the convex hull. This is called a localization problem [5] and can be solved in $\mathbf{O(\log n_i)}$.

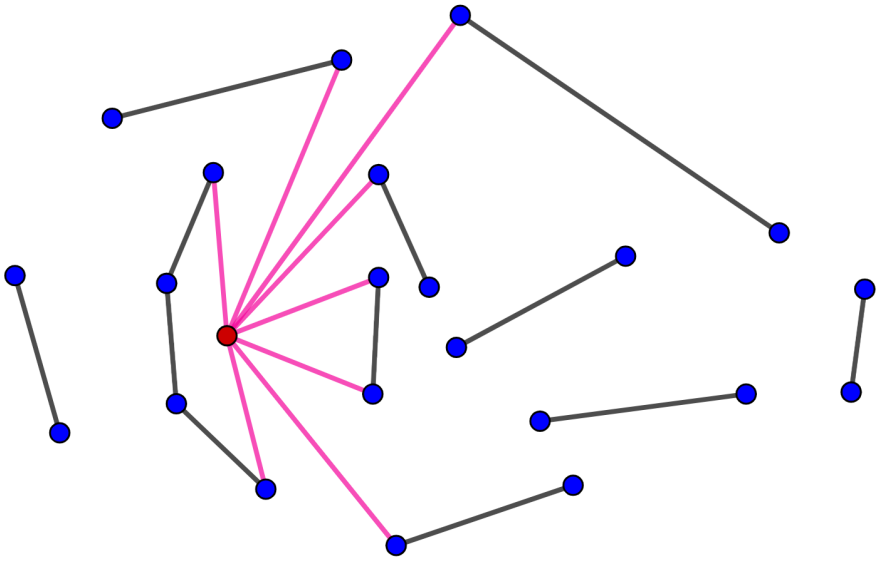


Step 2: If a point is outside the convex hull, find a pair of supporting lines to this polygon in $\mathbf{O(\log n_i)}$. The algorithm was invented by Preparata in 1979 as a key part of his famous real-time convex hull algorithm. However, it seems to be a little slow, so a new, much more elegant algorithm was invented (see

appendix). Having built a pair of supporting lines, we now have a segment formed by them, which can be used to represent the whole polygon. Indeed, it preserves all its visibility properties (see picture above).



Step 3: If a point is strictly inside the convex hull, finding supporting lines to a non-convex polygon takes $\mathbf{O(n^2)}$ time (brute force). However, $\mathbf{O(\log n_i)}$ time complexity is achieved almost in every case due to polygon configuration on the plane. Having built a pair of supporting lines, we now have a polyline formed by them, which can be used to represent the whole polygon. Once again, it preserves all its visibility properties (see picture above).



Step 4: Having built segments and polylines representing all polygons, the last thing is to build a visibility graph for all of them (see picture above). The total number of segments (considering the ones forming the polylines will be very close to **h** due to the reason discussed in step 3. Therefore, even a basic sweeping line algorithm [3] will achieve suitable **O(h log h)**.

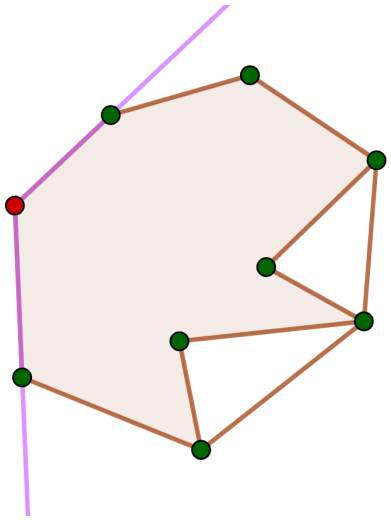
6. Complexity evaluation

Considering that step 3 is almost never computed, for a given point and a given polygon, part 1 of the algorithm (steps 1..3) takes $\mathbf{O(\log n_i)}$ time. For all polygons time complexity is $\sum_{i=1}^{h} \mathbf{c * \log n_i} = \mathbf{O(h \log n).}$ Part 2 of the algorithm (step 4) does not increase this complexity. Therefore, for each of **n** points total time is **O(nh log n).**

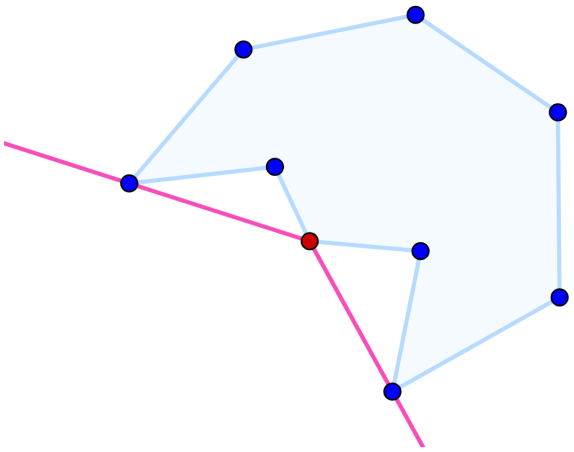
7. Restriction angles

Since each point is a part of a polygon, it is impossible to estimate its own polygon with a segment. Therefore, a concept of restriction angles is used. There are 2 variants:

1) A point is a part of its polygon's convex hull. In this case the 2 neighbour convex hull points define a restriction angle.



2) A point is not a part of its polygon's convex hull. In this case the 2 supporting points define a restriction angle.



For each point restriction angle is checked while building a visibility graph at step 4. The visibility graph vertices, which are inside the restriction angle, will not be included.

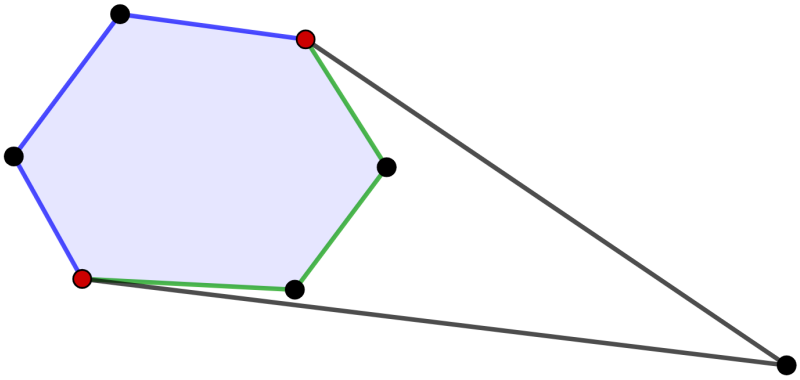
8. Approximation techniques

In order to speed up computations, Ramer–Douglas–Peucker approximation algorithm is used to simplify polygons. Since computation time should not depend much on size of the area, hierarchical approach is used. It involves removing small polygons from consideration and altering RDP parameters according to the size of the area to keep the amount of graph edges in the same range.

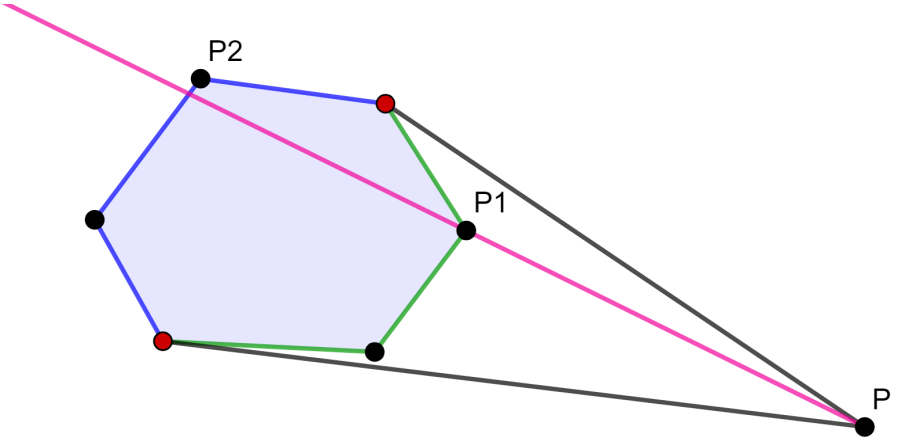
Appendix

Finding supporting lines from a point to a convex polygon on a plane. **O(log n)** algorithm by Denis Kozub. The algorithm works with the property of a convex polygon in respect to a point: semi-planes forming the polygon are

sorted by the fact of containing the point. Therefore 2 subsets are formed (segments, formed by semi-planes, containing the point, are drawn in blue, segments, formed by semi-planes, not containing the point, are drawn in green). Points dividing them (marked in red) are supporting points from a given point, which are to be found.



To find supporting points using binary search, 2 elements, each one of them belonging to one of the subsets respectively, are required. One of them (P1) can be found by picking a random vertex of a polygon. The second one (P2) can be found as one of the vertices, forming a segment, which is crossed by line (P, P1).



This can be done similarly to locating a point in a convex polygon [5] in **O(log n)** time. Having found P1 and P2, binary search can be applied to find supporting points in **O(log n)** time as well. A working implementation can be found in geometry/supporting_convex.py.

References

1. Danny Z. Chen and Haitao Wang, “A NEW ALGORITHM FOR COMPUTING VISIBILITY GRAPHS OF POLYGONAL OBSTACLES IN THE PLANE”

2. Subir Kumar Ghosh & David M. Mounts, “AN OUTPUT-SENSITIVE ALGORITHM FOR COMPUTING VISIBILITY GRAPHS”

3. M. de Berg, O. Cheong, M. Kreveld, M. Overmars «Computational Geometry» // Springer, pp. 326-331. 2008.

4. Timothy M. Chan. "Optimal output-sensitive convex hull algorithms in two and three dimensions". Discrete and Computational Geometry, Vol. 16, pp.361–368. 1996.

5. Препарата Ф., Шеймос М. Раздел 2.2: Задачи локализации точек. // Вычислительная геометрия: введение. — Москва: Мир, 1989.

6. F.P. Preparata. “An Optimal Real-Time Algorithm for Planar Convex Hulls”, 1979