CS 164 & CS 266: Computational Geometry

Lecture 5

Visibility and shortest paths

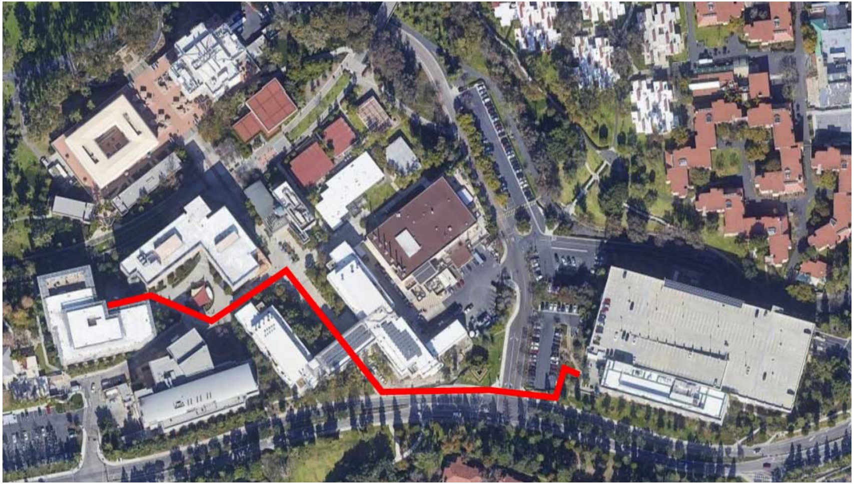
David Eppstein
University of California, Irvine

Fall Quarter, 2025

Shortest paths

Geometric shortest paths

Input: polygonal obstacles, two points s and t outside obstacles

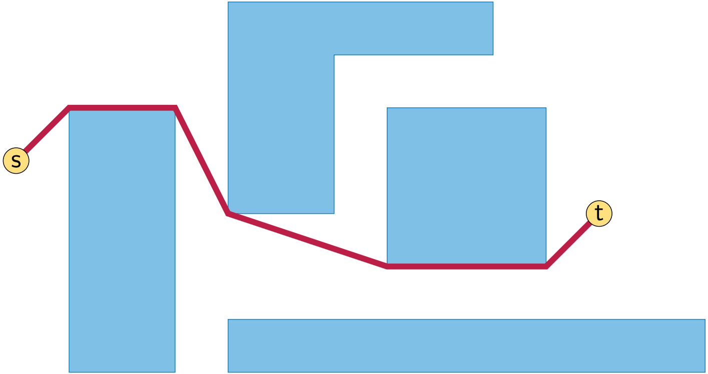


Output: shortest path from s to t

Difficulty: There are infinitely many possible paths!

Simplifying observations

Shortest path can only turn at vertices of obstacles



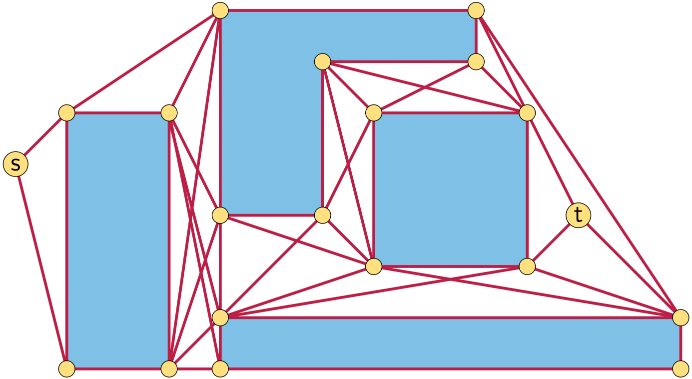
Between any two vertices (also including s and t as vertices)
it follows a straight line segment

Why: Any other path could be shortcut by going more straight

Conversion to a graph shortest path problem

Vertices: s, t, and corners of obstacles

Edges: Pairs of vertices that can see each other (no obstacle or vertex between them), weighted by Euclidean distance



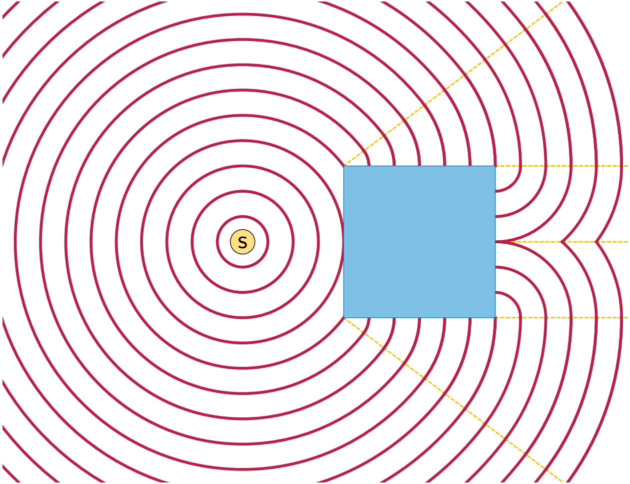
“Visibility graph”

Finding paths using visibility graphs

- Construct the visibility graph (the rest of this lecture)
- Use Dijkstra's algorithm for shortest paths in graphs with positive edge weights
- Exact comparisons of sums of square roots are problematic (we don't know how to do them efficiently) but if you just want a nearly-shortest path, floating point is good enough
- Can take time $\Theta(n^2)$ because visibility graphs can be big

Faster in two dimensions

"Continuous Dijkstra" (simulate waves expanding from start)
can get $O(n \log n)$ [Hershberger and Suri 1999]



More next time!

Three dimensions

Finding exact shortest paths is NP-hard

[Canny and Reif 1987]

Shortest paths can be approximated by placing many points along boundary edges, constructing visibility graph, and running a graph shortest path algorithm

[Papadimitriou 1985]

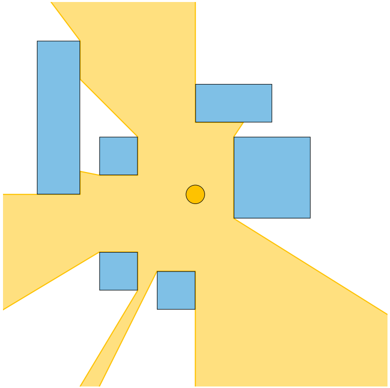
Computing visibility by radial scan

Problem formulation

Input can be represented as n non-crossing line segments (sides of polygonal obstacles)

For a given point of view, we may want to output:

- Set of segment endpoints it can see
- Entire visibility polygon: how far it can see in all directions



The main idea

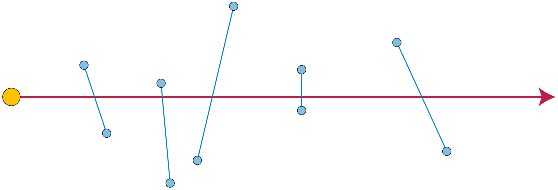
Like plane sweep, but sweep a ray circularly around a point to find what can be seen from the point

[McGarry 2004]

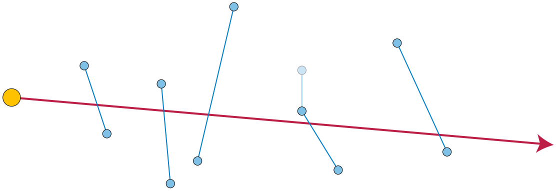
$O(n \log n)$ time per point, $O(n^2 \log n)$ to repeat around all vertices

Main steps of algorithm

Draw ray in positive x-direction from viewpoint,
find sorted order of segments crossed by the ray



Sweep clockwise around viewpoint maintaining order



Data structures

The sweep algorithms we already saw maintain a binary search tree of things that cross the sweep line, ordered by height

Instead, we need a priority queue (so we can find the closest to the viewpoint), ordered by distance from the viewpoint

As in previous sweeps, we need to be able to compare the order of pairs of items, but we don't want to calculate exact coordinates of crossing points (because it changes infinitely often, while the order only changes at discrete events)

A binary heap will work — it only needs comparisons

Pseudocode

Check which obstacle segments cross the horizontal ray and sort them by distance from the viewpoint

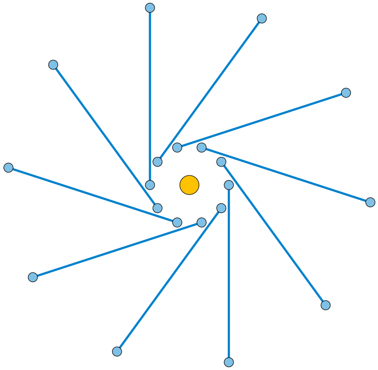
Initialize a priority queue of crossings using the sorted order

Initialize visibility polygon with an edge on the closest obstacle

For each segment endpoint, sorted clockwise around viewpoint:

- If it is first endpoint of its segment, add new segment to priority queue, and if it is the new closest segment, connect it into the visibility polygon
- If it is the last endpoint of its segment, remove from priority queue, and if it was the closest segment, connect visibility polygon to the new closest segment

A non-complication



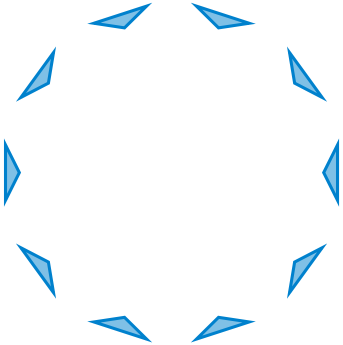
The obstacles may not have a linear ordering consistent with pairwise closeness comparisons

But we only order the subset of obstacles crossing the sweep ray, and this subset always has a consistent order

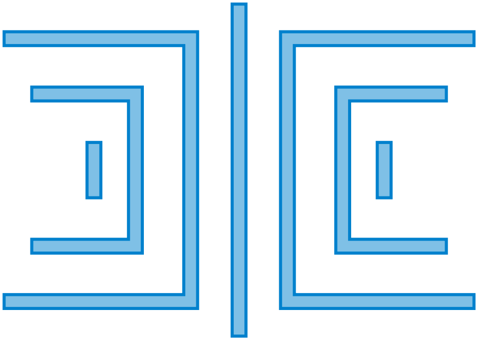
Output-sensitive visibility graph

Output sensitive analysis

Instead of analyzing the running time only in terms of the input size n (number of obstacle segments), also use output size k (number of visibility graph edges)



In the worst case everything can see everything else and $k = \Theta(n^2)$

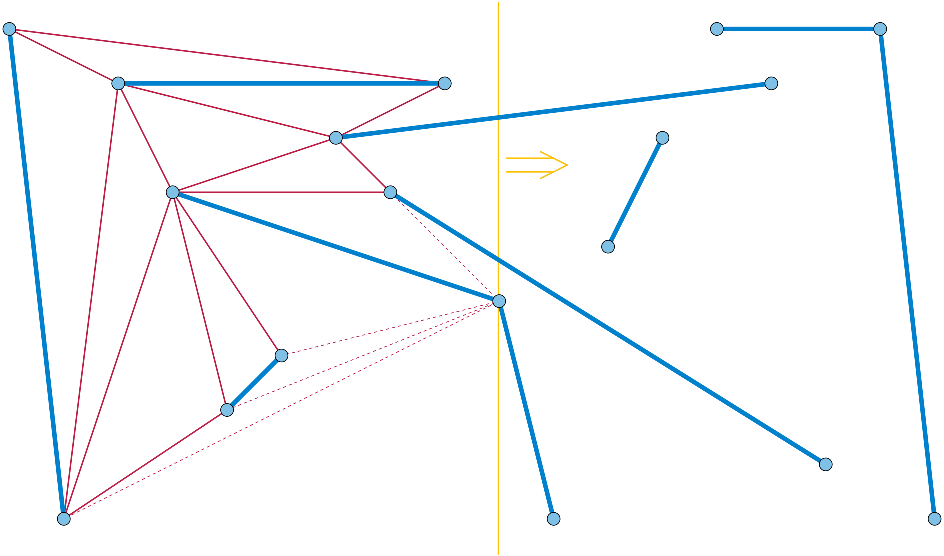


When obstacles block most lines of sight, $k \ll n^2$ and $O(k)$ is much faster than $O(n^2)$

Graduate reading included this for convex hulls: $O(n \log h)$ instead of $O(n \log n)$

Plane sweep triangulation (again)

In left-to-right order, connect each vertex
to everything to the left that it can still see

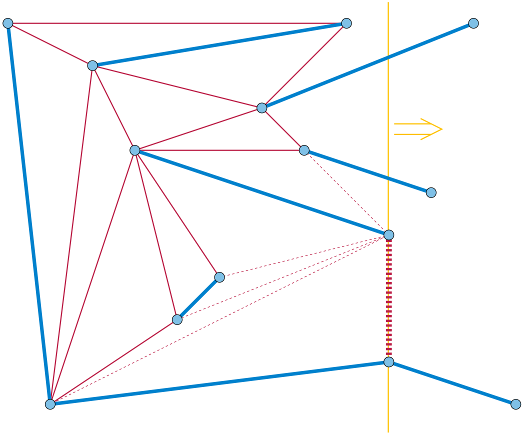


Ties in sweep order

If two vertices are tied in the sweep order (they have same x-coordinate) and visible to each other

Triangulation cannot add an edge blocking their visibility

So when it sweeps over them it will connect them by an edge



Horizontal sweep will find vertical visibility edges!

Main idea of output-sensitive visibility

Construct a plane-sweep triangulation for a horizontal sweep

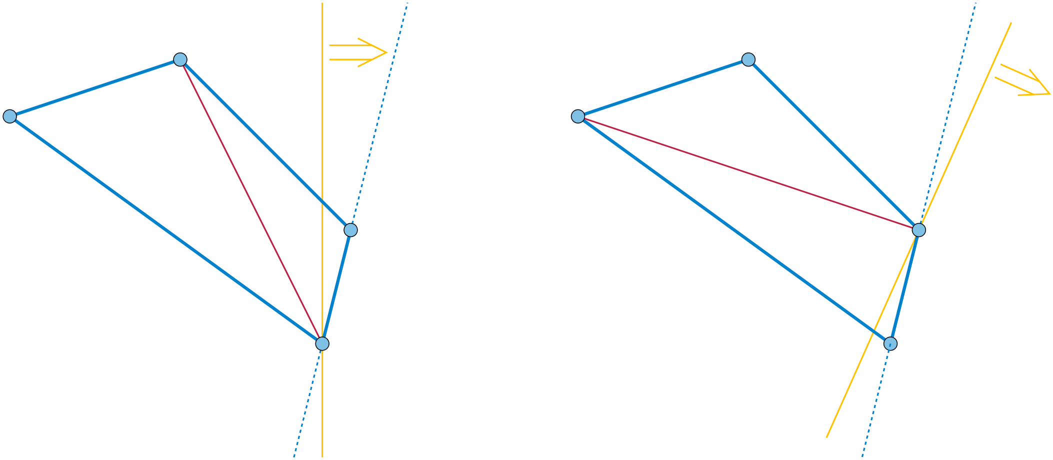
Continuously rotate the sweep direction and update the triangulation to match the new sweep direction

Every visibility edge will appear somewhere during the rotation!

(When the sweep direction becomes perpendicular to the edge)

How rotation of sweep line can change triangulation

Flip: Replace diagonal of convex quadrilateral by other diagonal



Happens when sweep line rotates through right side of quadrilateral

In 360° rotation of sweep line, each visibility edge is flipped in and out twice (once when each endpoint is rightmost)

Pseudocode

Construct plane-sweep triangulation

Initialize priority queue of flips (pairs of triangles in current triangulation that form convex quadrilaterals, prioritized by slope of right edge)

Repeat until we have done a 360° rotation:

- Use the priority queue to find the flip whose right edge xy has the closest slope to the current sweep line
- Update the sweep line slope
- While xy is the right edge of a flippable quadrilateral, flip it
- Update the priority queue to remove flips whose two triangles are no longer in the triangulation, and to add new flips formed by new pairs of triangles

Analysis

$O(n \log n)$ time to construct initial triangulation

2 flips per visibility edge $\Rightarrow 2k$ total $O(\log n)$ time per flip

Total time $O((n+k)\log n)$ [Overmars and Welzl 1988]

Slightly faster $O(k + n\log n)$ known [Ghosh and Mount 1991]

(Also uses plane-sweep triangulation in a more complicated way)

References and image credits I

John F. Canny and John H. Reif. New lower bound techniques for robot motion planning problems. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science (FOCS '87)*, pages 49–60. IEEE Computer Society, 1987. doi: 10.1109/SFCS.1987.42.

Subir Kumar Ghosh and David M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991. doi: 10.1137/0220055.

John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999. doi: 10.1137/S0097539795289604.

Justin McGarry. Aerographer's Mate Airman Steven L. Davidson reviews the AN-SPA radar screen that is used to read the heights of clouds within the radar's radius aboard USS Enterprise. Public domain (CC0) image, May 2 2004. URL https://commons.wikimedia.org/wiki/File:US_Navy_040502-N-7408M-006_Aerographer%27s_Mate_Airman_Steven_L._Davidson,_of_Rockhill,_S.C.,_reviews_the_AN-SPA_radar_screen_that_is_used_to_read_the_heights_of_clouds.jpg.

References and image credits II

Mark H. Overmars and Emo Welzl. New Methods for Computing Visibility Graphs. In Herbert Edelsbrunner, editor, *Proceedings of the Fourth Annual Symposium on Computational Geometry (SoCG '88)*, pages 164–171. ACM, 1988. doi: 10.1145/73393.73410.

Christos H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Information Processing Letters*, 20(5):259–263, 1985. doi: 10.1016/0020-0190(85)90029-8.