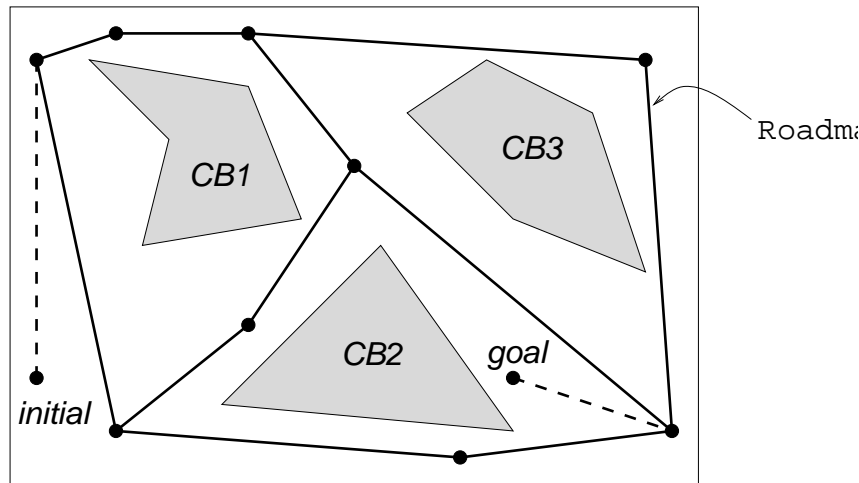


Roadmap Methods

Acknowledgement: Parts of these course notes are based on notes from courses given by Jean-Claude Latombe at Stanford University (and Chapter 4 in his text *Robot Motion Planning*, Kluwer, 1991), O. Burchan Bayazit at Washington University in St. Louis. Seth Hutchinson at the University of Illinois at Urbana-Champaign, and Leo Joskowicz at Hebrew University.

Roadmap Methods

Idea: capture the connectivity of \mathcal{C}_{free} with a **roadmap** (graph or network) of one-dimensional curves



PATH PLANNING WITH A ROADMAP

input: configurations \mathbf{q}_{init} and \mathbf{q}_{goal} , and \mathcal{B}

output: a path in \mathcal{C}_{free} connecting \mathbf{q}_{init} and \mathbf{q}_{goal}

1. build a roadmap in \mathcal{C}_{free} (preprocessing)
 - roadmap nodes are free configurations (or semi-free)
 - two nodes connected by edge if can (easily) move between them
2. connect \mathbf{q}_{init} and \mathbf{q}_{goal} to roadmap nodes v_{init} and v_{goal} (in same connected component)
3. find a path in the roadmap between v_{init} and v_{goal}
 - directly gives a path in \mathcal{C}_{free}

Hard part is building the roadmap

Building Roadmaps – The Hard Part

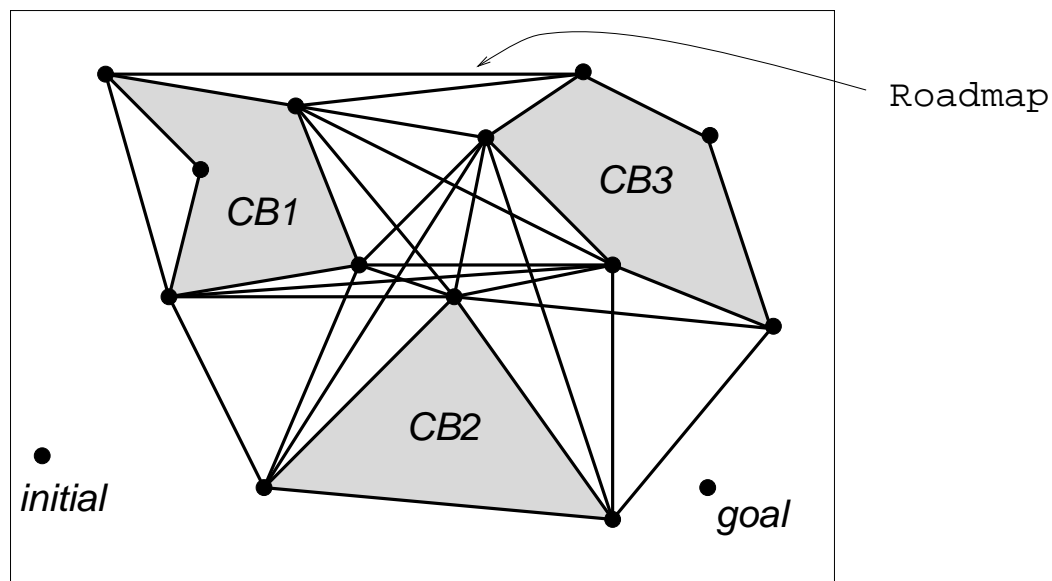
Deterministic Methods (Traditional Approaches)

- need to *build* representation of \mathcal{C}_{free}
 - these methods are *complete*
 - most methods apply only to low-dimensional C-space
 - Canny's *Silhouette Method* was a break through in motion planning
 - applies to general case
 - it is only singly exponential in m , the dimension of \mathcal{C} , i.e., $O(2^m)$ versus $O(2^{2^m})$ for Collins Decomposition exact approximation method
 - complicated and slow except for special cases (low dimensional and polygonal \mathcal{C})
 - (Chapter 4 in Latombe)
-

Probabilistic Methods (More recent, now method of choice)

- *don't build* representation of \mathcal{C}_{free}
- these methods are *not complete*
- apply to any C-space (low or high dimensional)
- usually easy and fast
- (recent results – papers)

Roadmap: Visibility Graph of C-space



A **visibility graph** of C-space for a given \mathcal{CB} is an undirected graph G where

- nodes in G correspond to vertices of \mathcal{CB}
- nodes connected by edge in G if
 - they are connected by an edge in \mathcal{CB} , or
 - the straight line segment connecting them lies entirely in \mathcal{C}_{free}
- (could add \mathbf{q}_{init} and \mathbf{q}_{goal} as roadmap nodes)

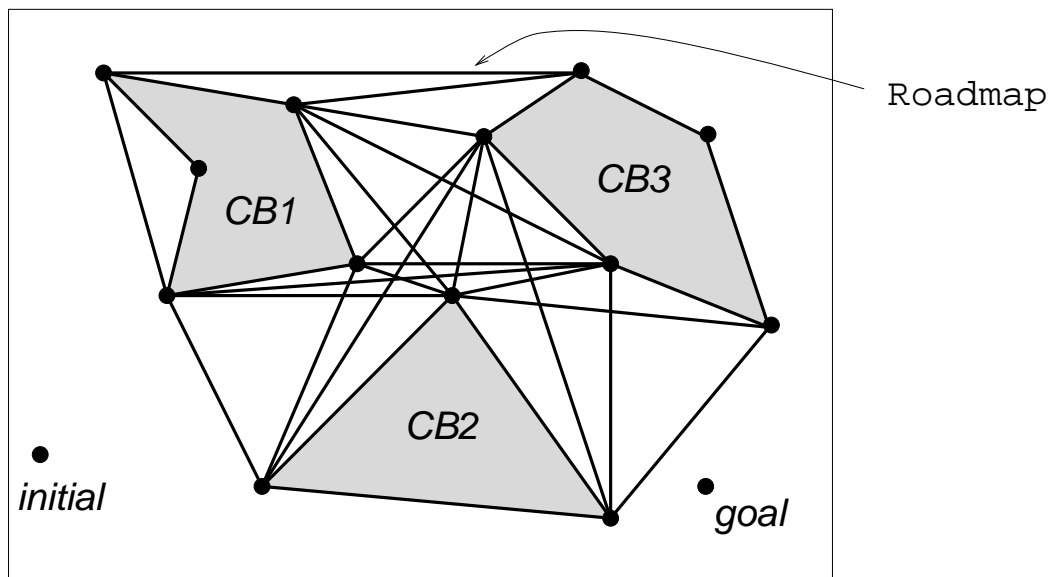
Visibility graphs ...

- are conceptually simple
- we have efficient algorithms if \mathcal{CB} is polygonal
 - $O(n^2)$, where n is number of vertices of \mathcal{CB}
 - $O(k + n \log n)$, where k is number of edges in G
- we can make a 'reduced' visibility graph (don't need all edges)
- really only suitable for two-dimensional \mathcal{C}

Visibility Graphs (cont'd)

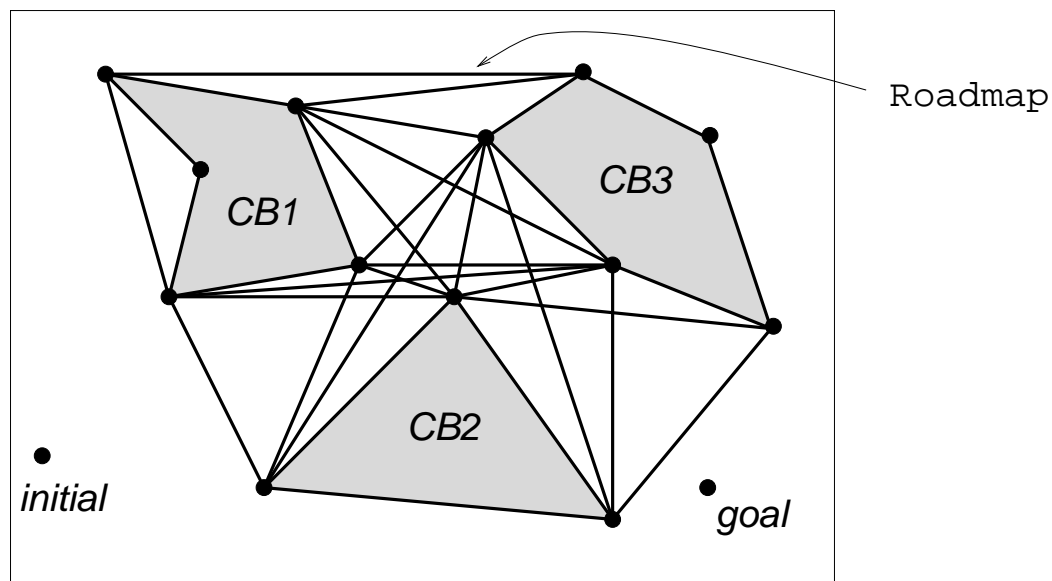
Fact: If $\mathcal{C} = \mathbb{R}^2$ and \mathcal{CB} is polygonal, then there exists a semi-free path connecting \mathbf{q}_{init} and \mathbf{q}_{goal} iff there exists a simple polygonal path lying in $cl(\mathcal{C}_{free})$ whose endpoints are \mathbf{q}_{init} and \mathbf{q}_{goal} and whose interior points are vertices of \mathcal{CB} . (Proposition 1, p. 155 in [Latombe]).

Note: the internal portion of this polygonal path, if it exists, will be contained in the visibility graph, i.e., this is a complete planning method.



Fact: If $\mathcal{C} = \mathbb{R}^2$ and \mathcal{CB} is polygonal, then the the shortest path between \mathbf{q}_{init} and \mathbf{q}_{goal} can be will be contained in the visibility graph (when \mathbf{q}_{init} and \mathbf{q}_{goal} are vertices of G)

Constructing Visibility Graphs



Brute Force Construction of Visibility Graph G

1. add all edges in \mathcal{CB} to G
2. for each pair of vertices (x, y) of \mathcal{CB} , add the edge (x, y) to G if the straight line segment connecting them lies entirely in $cl(\mathcal{C}_{free})$
 - test (x, y) for intersection with all $O(n)$ edges of \mathcal{CB}
 - $O(n^2)$ pairs to test, each test takes $O(n)$ time

Complexity: $O(n^3)$, n is number of vertices in \mathcal{CB}

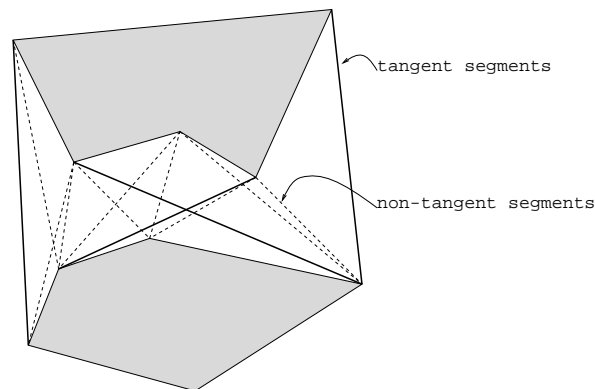
More efficient algorithms are known . . .

- $O(n^2 \log n)$ with simple sweepline algorithm
- can be improved to $O(n^2)$
- *output-sensitive* $O(k + n \log n)$, where k is number of edges in G

Reduced Visibility Graphs

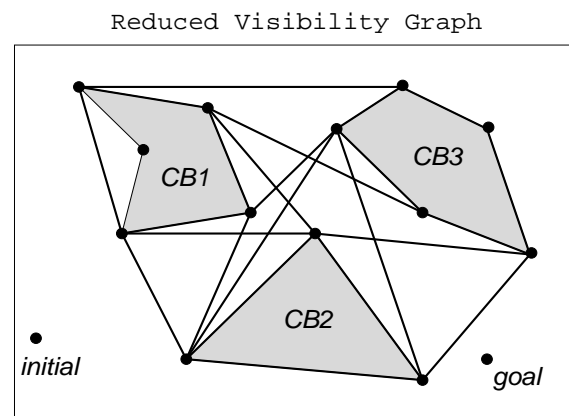
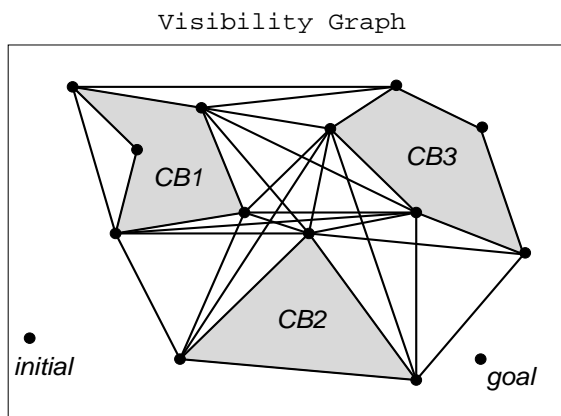
Idea: we don't really need all the edges in the visibility graph (even if we want shortest paths)

Definition: Let L be the line passing through an edge (x, y) in the visibility graph G . The segment (x, y) is a **tangent segment** iff L is tangent to \mathcal{CB} at both x and y .



It turns out we need only keep

- convex vertices of \mathcal{CB}
- non- \mathcal{CB} edges that are tangent segments



Building Reduced Visibility Graphs

Reduced visibility graphs are easier to build

1. construct convex hull of each \mathcal{CB} piece
 - eliminate non-convex vertices
2. construct pairwise tangents between each convex \mathcal{CB} piece
 - easy to construct tangents between two convex polygons

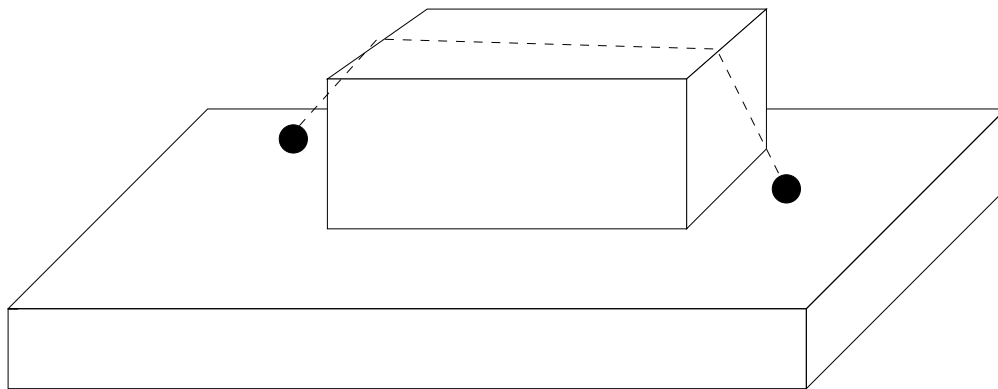
Complexity: $O(n + c^2 \log n)$, c is number of \mathcal{CB} pieces

- convex hulls in $O(n_i \log n_i)$ time (or $O(n_i)$ if given polygon vertices in cyclic connection order)
 - pairwise tangents in $O(\log n_i + \log n_j)$ time
-

Note: visibility graphs *don't* necessarily contain shortest paths in \mathbb{R}^3

– in fact finding shortest paths in \mathbb{R}^3 is NP-hard [Canny 1988]

– $(1 + \epsilon)$ approximation algorithm [Papadimitriou 1985]



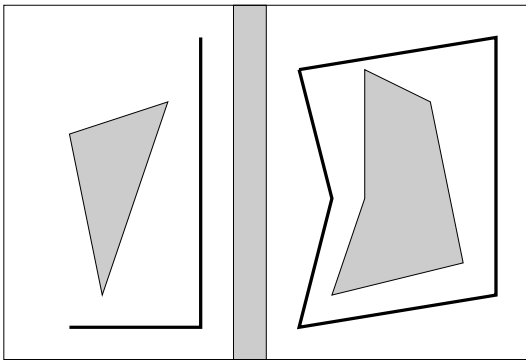
Final Note: Visibility graphs are not really suitable if the dimension of \mathcal{C} is greater than two ...

Roadmaps: the Retraction Approach

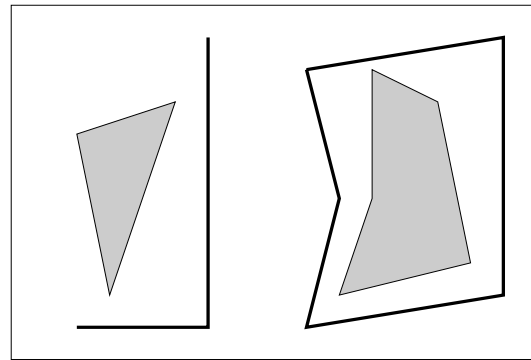
Basic Idea: 'retract' \mathcal{C}_{free} onto a 1-dimensional subset of itself (the roadmap).

The Retraction Approach

- a map $\rho : \mathcal{C}_{free} \rightarrow R$, $R \subset \mathcal{C}_{free}$, is a **retraction** iff it is continuous and its restriction to R is the identity map (i.e., $\rho(R) = R$)
 - thus, $\rho(x) \in R$ for all $x \in \mathcal{C}_{free}$ and $\rho(y) = y$ for all $y \in R$



a retraction



not a retraction
(not continuous)

- a retraction $\rho : \mathcal{C}_{free} \rightarrow R$ is **connectivity preserving** iff for all $x \in \mathcal{C}_{free}$, x and $\rho(x)$ belong to the *same* connected component of \mathcal{C}_{free}

Fact: *There exists a free path from \mathbf{q}_{init} to \mathbf{q}_{goal} iff there exists a path in R between $\rho(\mathbf{q}_{init})$ and $\rho(\mathbf{q}_{goal})$.*

This is why retractions make good roadmaps.

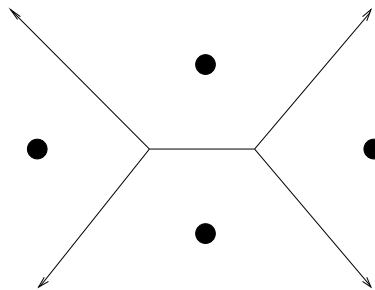
Retraction Example: Generalized Voronoi Diagrams

Assumption: $\mathcal{C} = \mathbb{R}^2$ and \mathcal{C}_{free} is a polygonal region (with boundary)

Definitions

- $\beta = \delta(\mathcal{C}_{free})$ (the boundary of \mathcal{C}_{free})
 - $clearance(\mathbf{q}) = \min(\|\mathbf{q} - \mathbf{p}\| \mid \mathbf{p} \in \beta)$, for $\mathbf{q} \in \mathcal{C}_{free}$
 - $near(\mathbf{q}) = \{\mathbf{p} \in \beta \mid \|\mathbf{q} - \mathbf{p}\| = clearance(\mathbf{q})\}$
i.e., $near(\mathbf{q})$ is set of boundary points of \mathcal{C}_{free} minimizing distance to \mathbf{q}
 - *Generalized Voronoi Diagram* $Vor(\mathcal{C}_{free}) = \{\mathbf{q} \in \mathcal{C}_{free} \mid |near(\mathbf{q})| > 1\}$
i.e., points in \mathcal{C}_{free} with at least *two* nearest neighbors in \mathcal{C}_{free} 's boundary
-

Example: **Voronoi Diagram for point sets** (original)

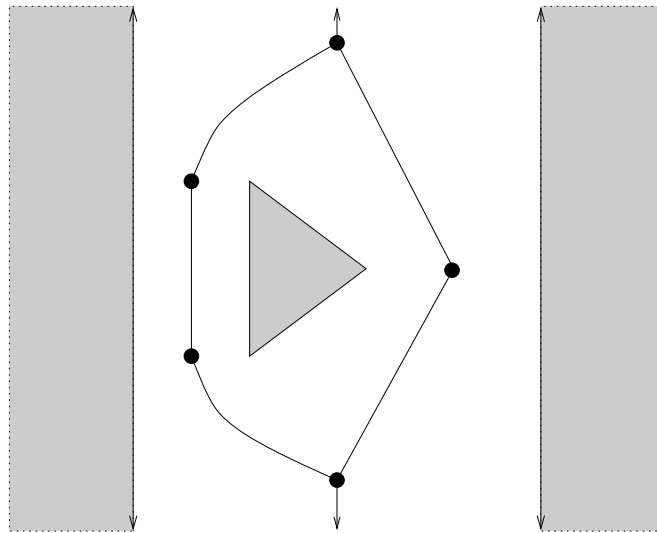


- Voronoi diagram of point set X consists of straight line segments
- constructed by
 - computing lines bisecting each pair of points and their intersections
 - computing intersections of these lines
 - keeping segments with more than one nearest neighbor
- segments of $Vor(X)$ have largest clearance from X and regions identify closest point of X

Generalized Voronoi Diagrams

When $\mathcal{C} = \mathbb{R}^2$ and polygonal \mathcal{CB} , $Vor(\mathcal{C}_{free})$ consists of a finite collection of straight line segments and *parabolic curve segments* (called **arcs**)

- **straight arcs** are defined by two vertices or two edges of \mathcal{CB} , i.e., the set of points equally close to two points (or two line segments) is a line
- **parabolic arcs** are defined by one vertex and one edge of \mathcal{CB} , i.e., the set of points equally close to a point and a line is a parabola



Naive Method of Constructing $Vor(\mathcal{C}_{free})$

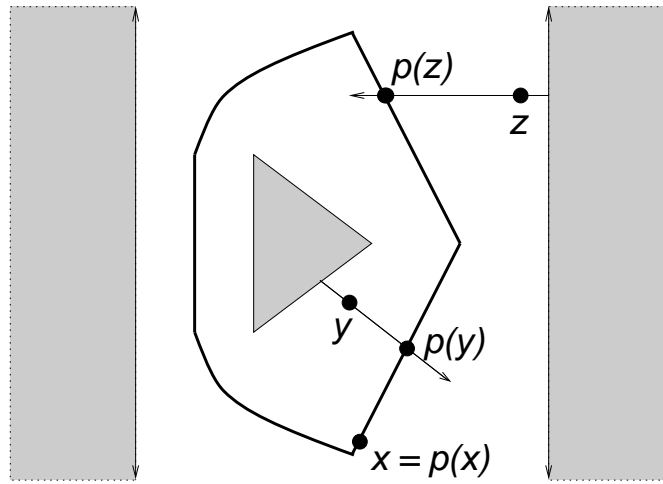
1. compute all arcs (for each vertex-vertex, edge-edge, and vertex-edge pair)
2. compute all intersection points (dividing arcs into segments)
3. keep segments which are closest only to the vertices/edges that defined them

Defining ρ for Generalized Voronoi Diagrams

To use $Vor(\mathcal{C}_{free})$ as our roadmap R , we need to define the retraction

$$\rho : \mathcal{C}_{free} \rightarrow Vor(\mathcal{C}_{free})$$

Case 1: $\mathbf{q} \in Vor(\mathcal{C}_{free})$: $\rho(\mathbf{q}) = \mathbf{q}$



Case 2: $\mathbf{q} \notin Vor(\mathcal{C}_{free})$

- let \mathbf{p} be the closest point of the boundary of \mathcal{C}_{free} to \mathbf{q}
- let L be the ray from \mathbf{p} passing through \mathbf{q} (L follows the steepest ascent of the $clearance()$ function from \mathbf{p})
- define $\rho(\mathbf{q})$ to be the intersection of L with $Vor(\mathcal{C}_{free})$

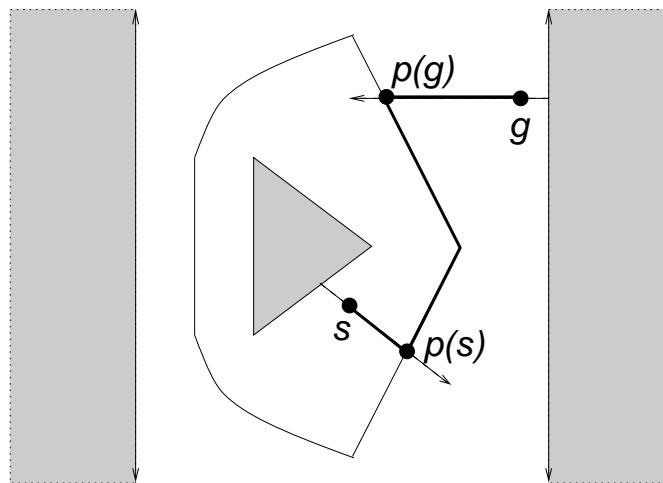
As desired, ρ :

1. preserves connectivity (\mathbf{q} and $\rho(\mathbf{q})$ in same connected region of \mathcal{C}_{free})
2. is continuous (not difficult to show)

Path Planning using Generalized Voronoi Diagrams

To find a path:

1. compute $Vor(\mathcal{C}_{free})$
2. find paths from \mathbf{q}_{init} and \mathbf{q}_{goal} to $\rho(\mathbf{q}_{init})$ and $\rho(\mathbf{q}_{goal})$, respectively
3. search $Vor(\mathcal{C}_{free})$ for a set of arcs connecting $\rho(\mathbf{q}_{init})$ and $\rho(\mathbf{q}_{goal})$



Finding a path from \mathbf{q}_{init} (or \mathbf{q}_{goal}) to $Vor(\mathcal{C}_{free})$

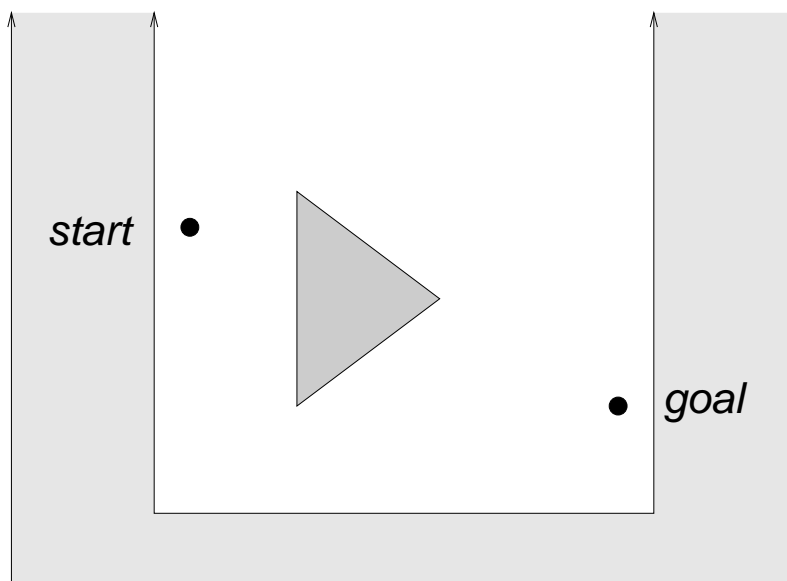
Fact: the line segment from \mathbf{q}_{init} to $\rho(\mathbf{q}_{init})$ lies entirely in \mathcal{C}_{free} (follows from the definition of ρ)

Thus, we can simply use this segment as our path!

Summary

- $Vor(\mathcal{C}_{free})$ has size $O(n)$, n is the number of vertices of \mathcal{CB}
- $Vor(\mathcal{C}_{free})$ can be computed in optimal $O(n \log n)$ time
- *only* works for $\mathcal{C} = \mathbb{R}^2$, but 'retraction-like' methods for other \mathcal{C} 's
- again, only practical for low dimensional \mathcal{C} 's ...

Mini HW #3: Generalized Voronoi Diagram Exercise



1. Draw the Generalized Voronoi Diagram for the figure above.
2. Label each arc in your diagram according to the edges and vertices that define it.
3. Show how to connect the start and goal configurations to the diagram.