Search and Motion Planning

Prof. Dr. Daniel Göhring

January 8, 2018

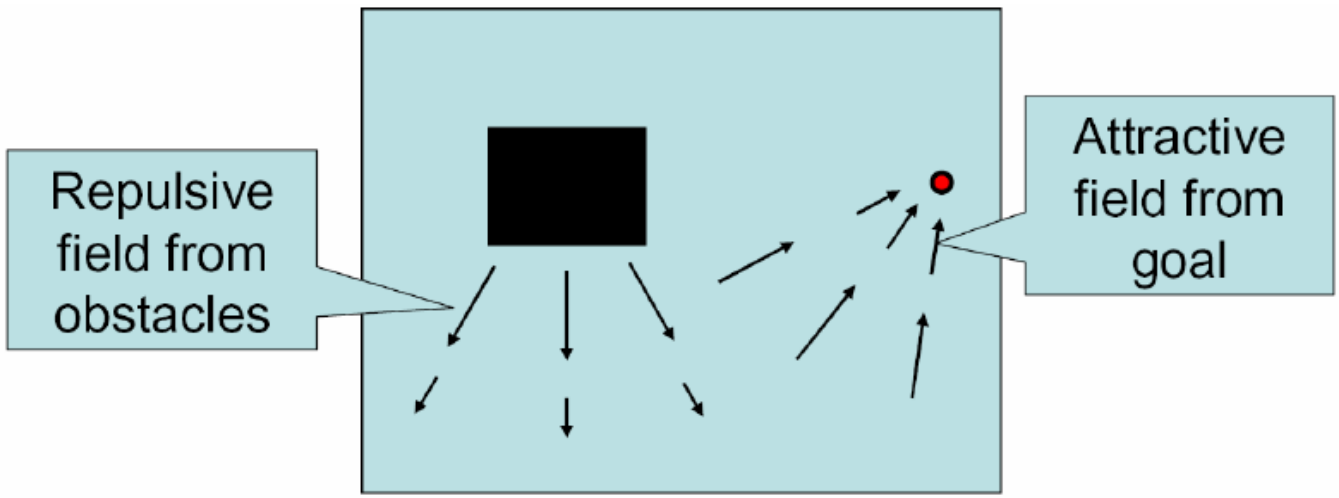
FOR INTERNAL USE ONLY

Table of contents

Topics

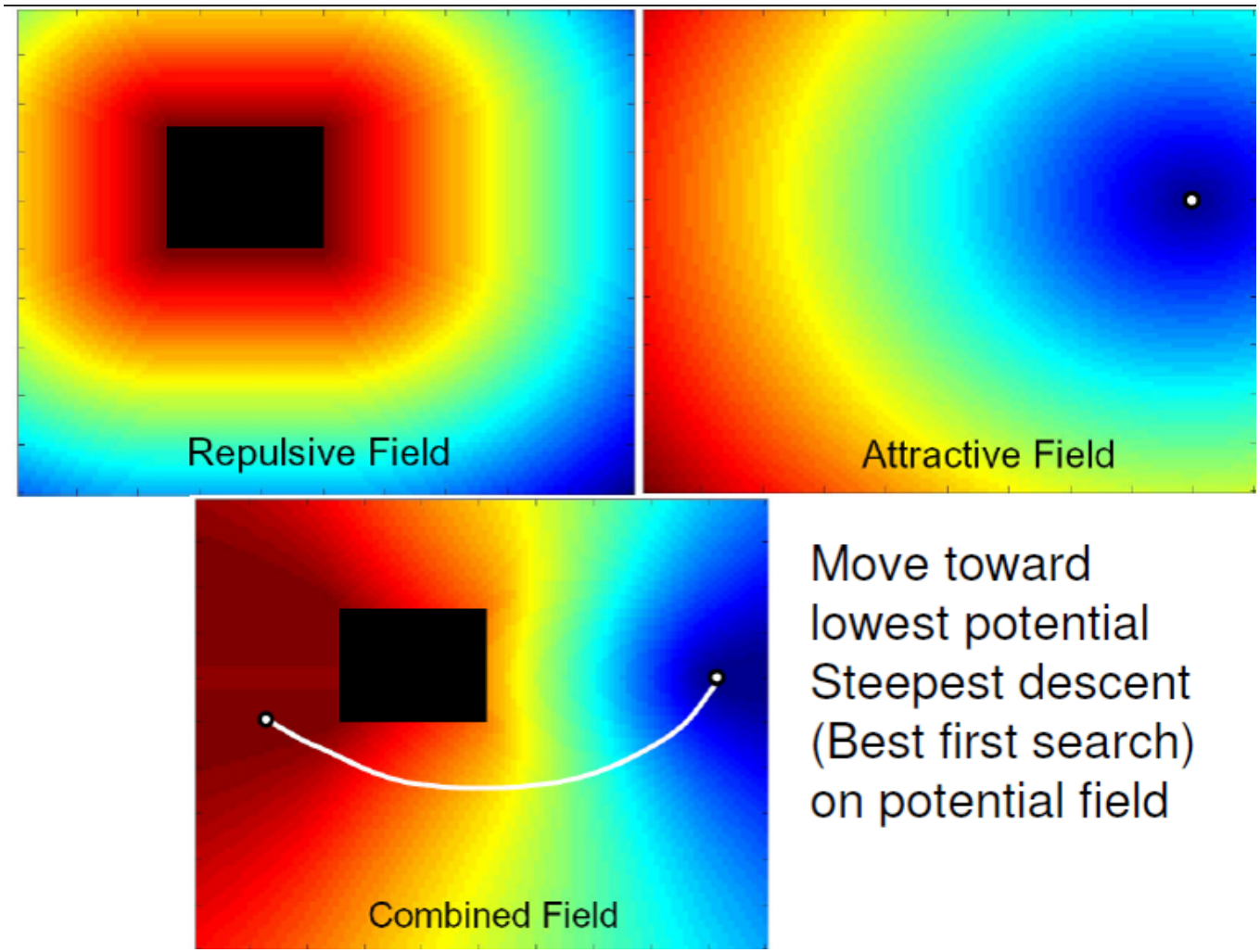
- Potential Fields
- Visibility Graphs
  - Roadmaps
  - Voronoi
- Approximate Cell Decomposition
- Probabilistic Roadmaps
- Rapidly Exploring Random Trees

Potential Fields



- Stay away from obstacles: Imagine that the obstacles are made of a material that generate a repulsive field
- Move closer to the goal: Imagine that the goal location is a particle that generates an attractive field

Potential Fields



Move toward lowest potential
Steepest descent (Best first search) on potential field

Potential Fields

$$U_g(\mathbf{q}) = d^2(\mathbf{q}, \mathbf{q}_{goal})$$

Distance to goal state

$$U_o(\mathbf{q}) = \frac{1}{d^2(\mathbf{q}, Obstacles)}$$

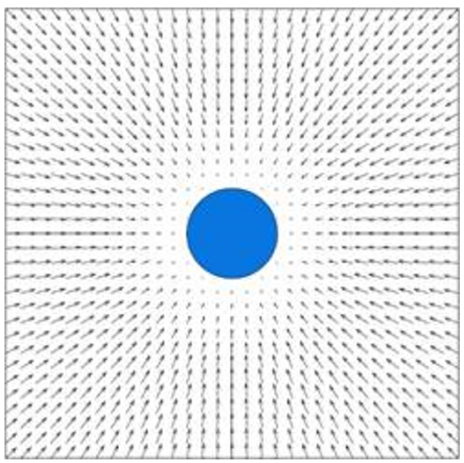
Distance to nearest obstacle point.
Note: Can be computed efficiently by using the *distance transform*

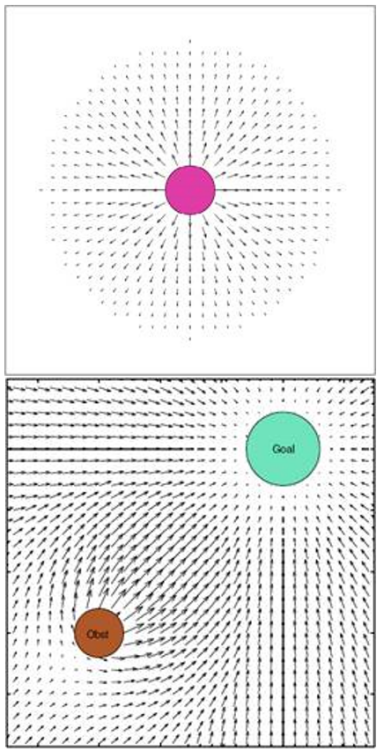
$$U(\mathbf{q}) = U_g(\mathbf{q}) + \lambda U_o(\mathbf{q})$$

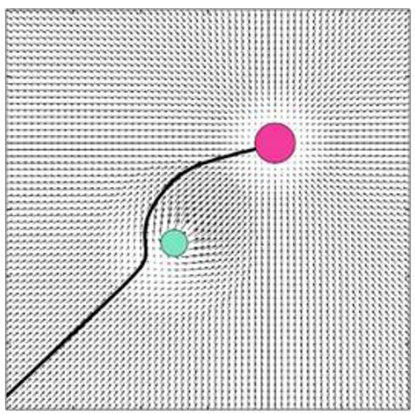
λ controls how far we stay from the obstacles

Potential Fields, Representation

- Closed Form as a math. formular,
- Gridbased





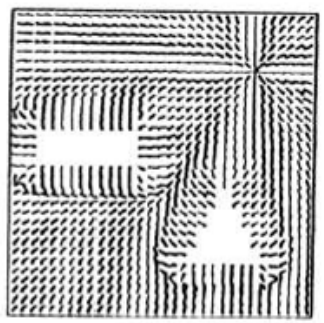


Example

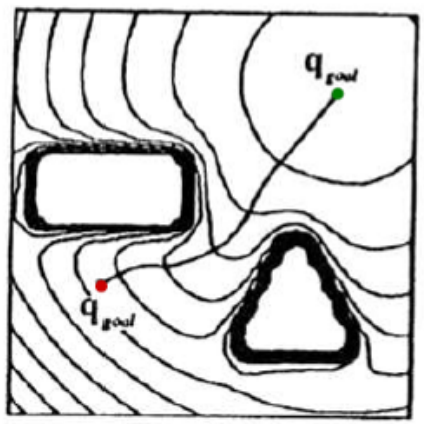
Potential field method

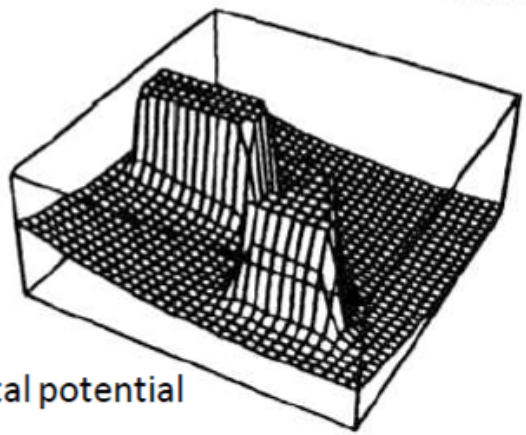
- After the total potential is obtained, generate force field (negative gradient)
- Let the sum of the forces control the robot.

Negative gradient



Equipotential contours

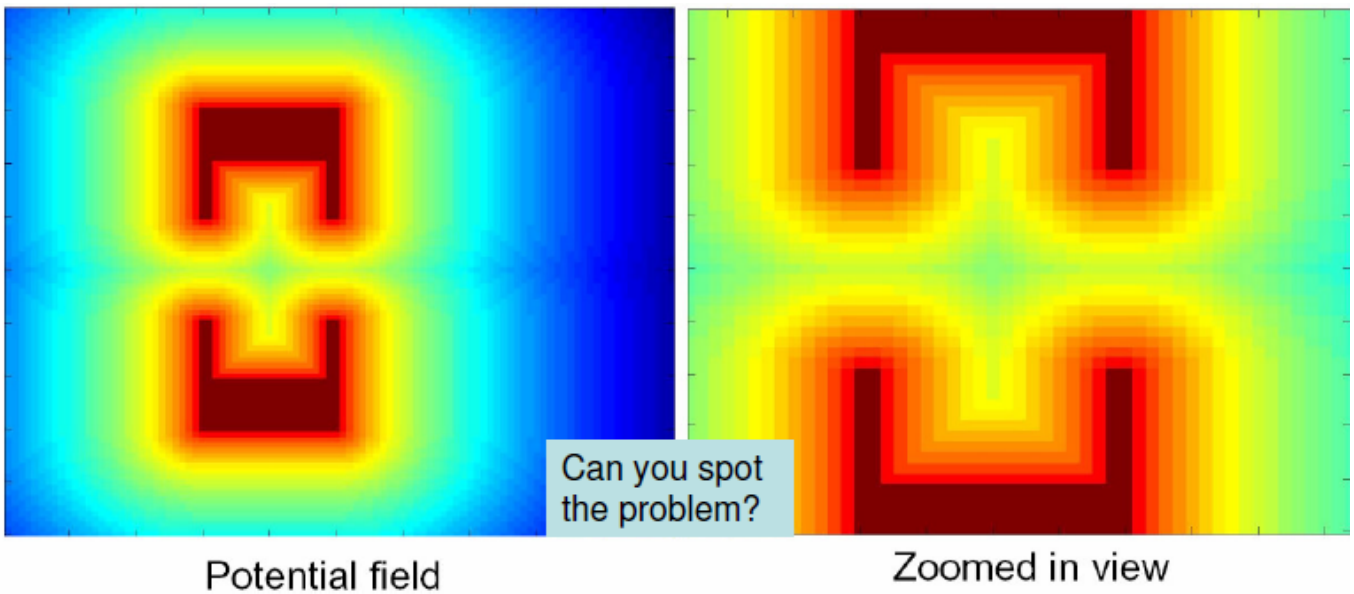




Total potential

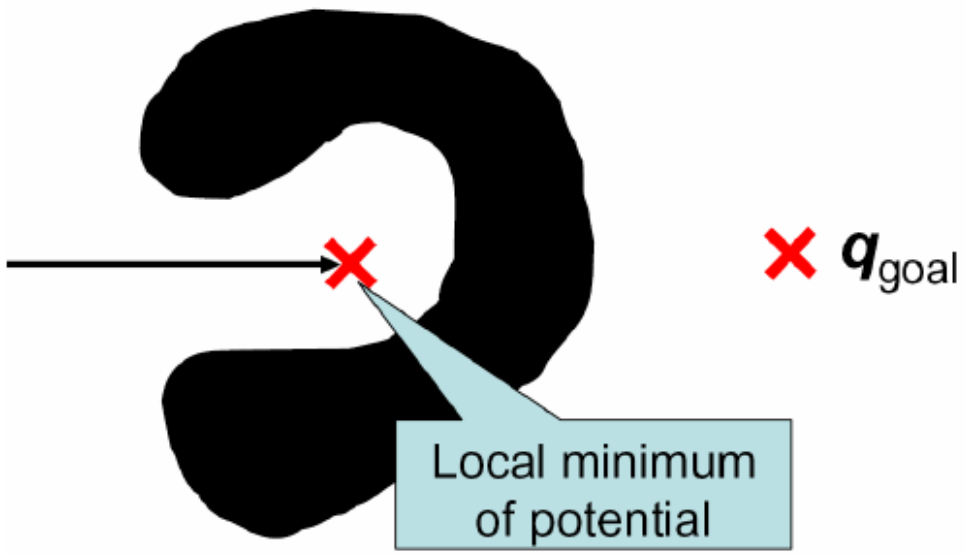
To a large extent, this is computable from sensor readings.

Potential Fields: Limitations



- Completeness
- Problems in higher dimensions

Local Minimum Problem



- Potential fields in general exhibit local minima
- Special case: Navigation function
  - $U(q_{goal}) = 0$
  - For any q different from q_{goal}, there exists a neighbor q' such that $U(q') < U(q)$

Getting out of Local Minima I

Repeat

- If U(q) = 0 return Success
- If too many iterations return Failure
- Else:
  - Find neighbor q_n of q with smallest $U(q_n)$
  - If $U(q_n) < U(q)$ OR q_n has not yet been visited
    - Move to $q_n (q \leftarrow q_n)$
    - Remember q_n

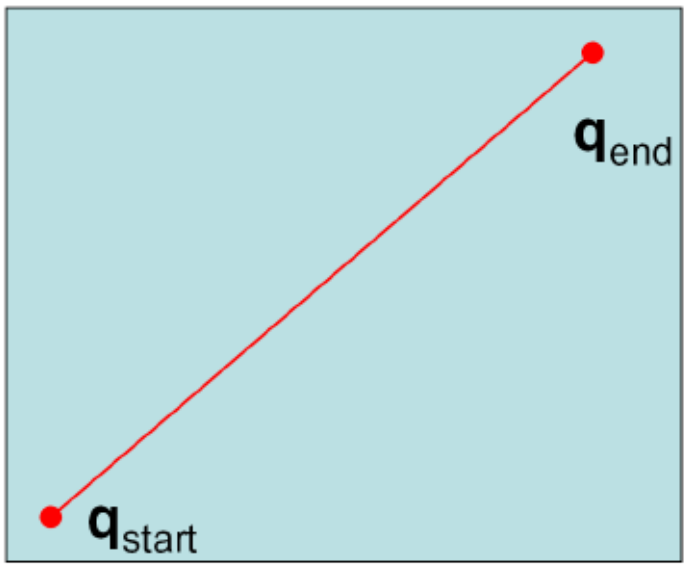
  May take a long time to explore the region around local minima

Getting out of Local Minima II

Repeat

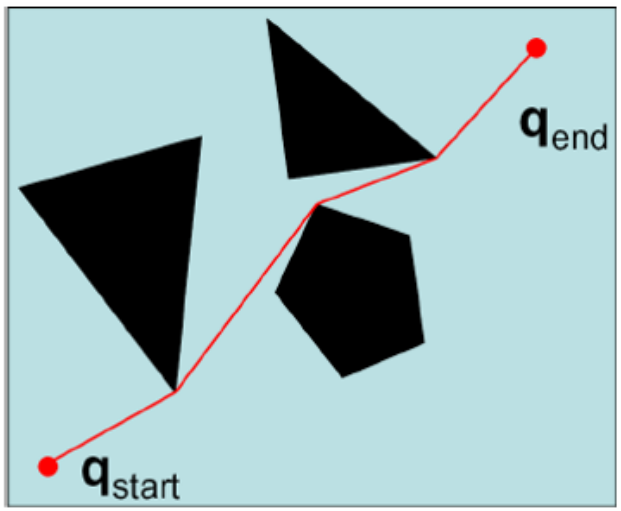
- If U(q) = 0 return Success
- If too many iterations return Failure
- Else:
  - Find neighbor q_n of q with smallest $U(q_n)$
  - If $U(q_n) < U(q)$
    - Move to $q_n (q \leftarrow q_n)$
  - Take a random walk for T steps starting at q_n
  - Set q to the configuration reached at the end of the random walk

Visibility Graphs



In the absence of obstacles, the best path is the straight line between $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$

Visibility Graphs



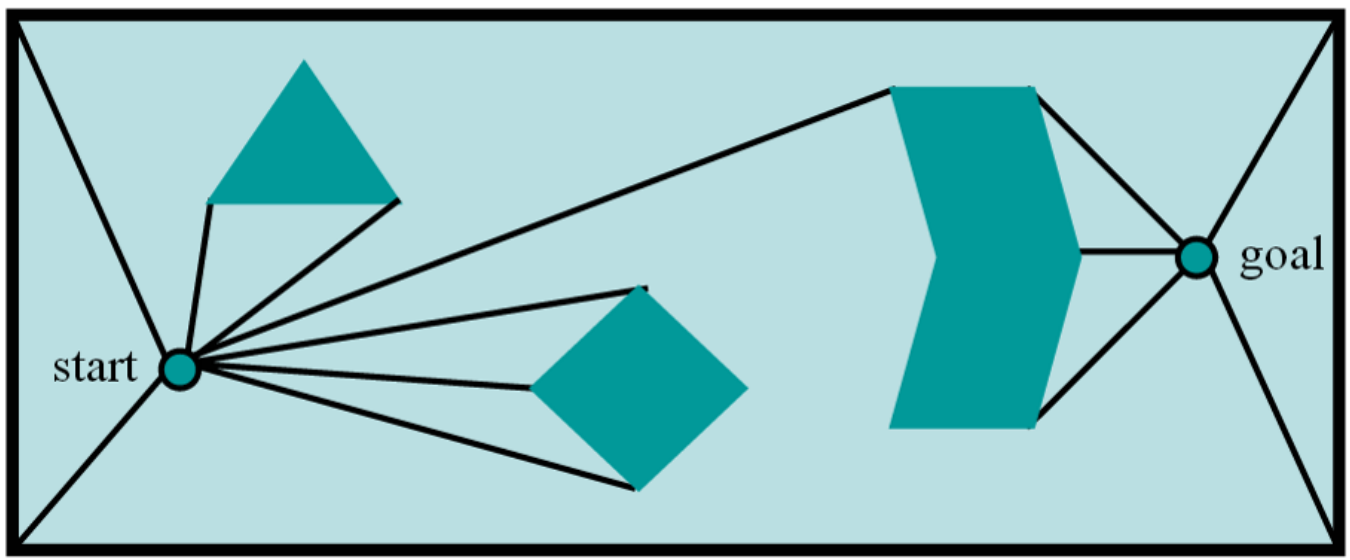
- Assuming polygonal obstacles: It looks like the shortest path is a sequence of straight lines joining the vertices of the obstacles.
- Is this always true?

Roadmap Approaches, Visbility Graphs

- Used to find Euclidean shortest paths among a set of polygonal obstacles in the plane
- Construct all of the line segments that connect vertices to one another (and that do not intersect the obstacles themselves)
- Formed by connecting all "visible" vertices, the start point and the end point, to each other
- For two points to be "visible" no obstacle can exist between them
- Paths exist on the perimeter of obstacles
- A graph is defined, converts the problem into graph search
- Dijkstra's algorithm, $O(N^2), N =$ the number of vertices in C-space

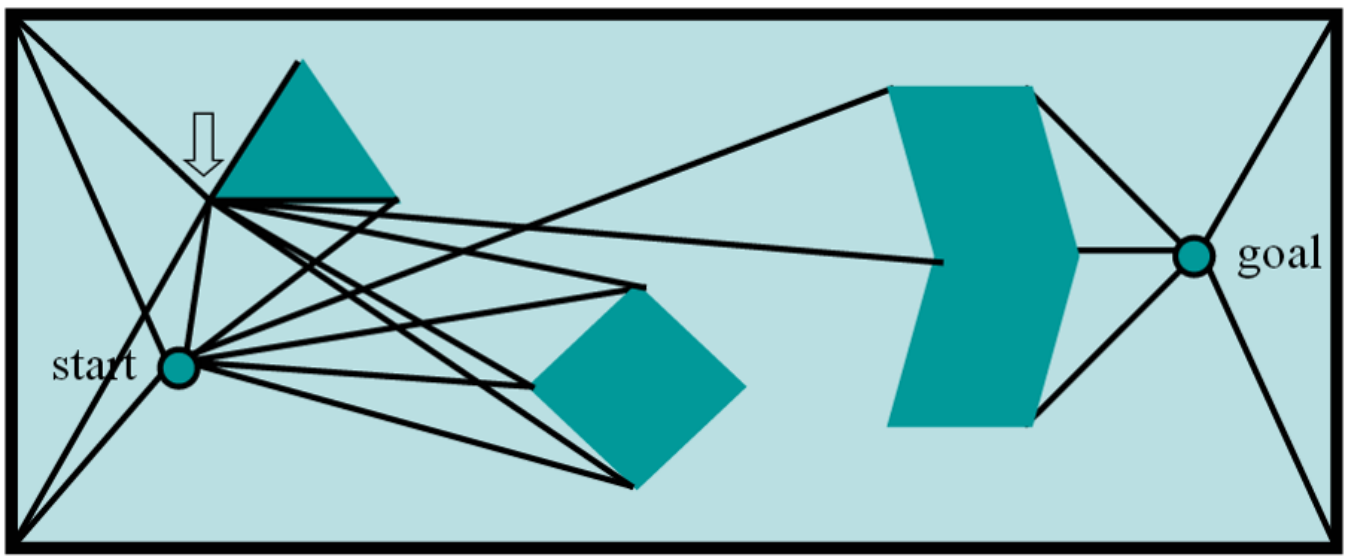
The Visibility Graph in Action

First, draw lines of sight from the start and goal to all "visible" vertices and corners of the world.



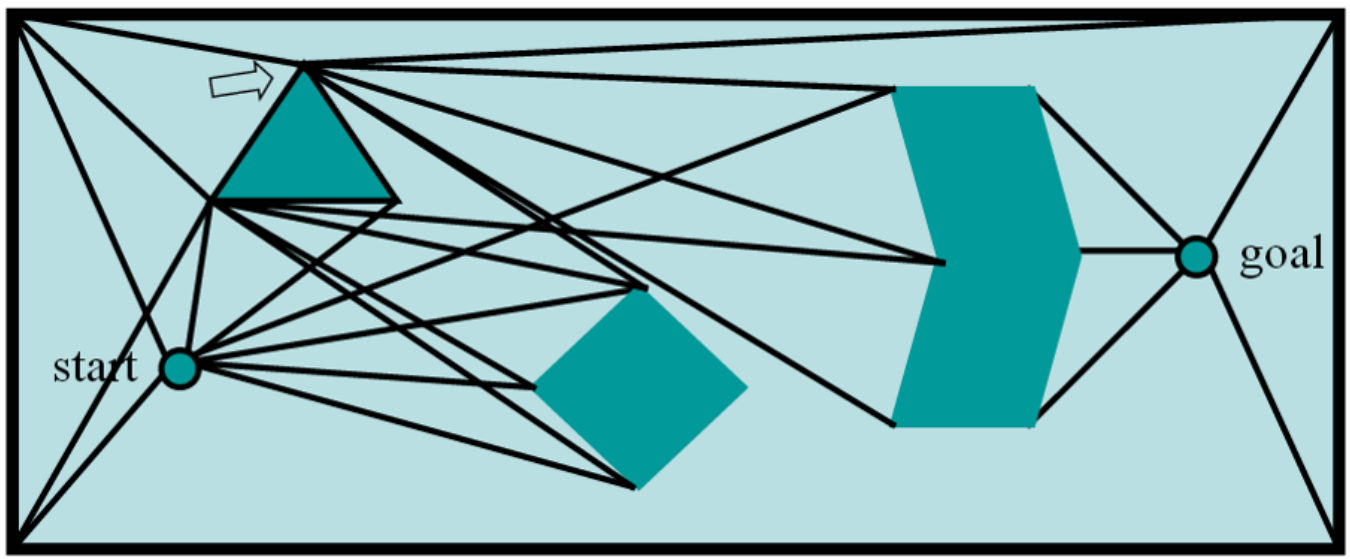
The Visibility Graph in Action

Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



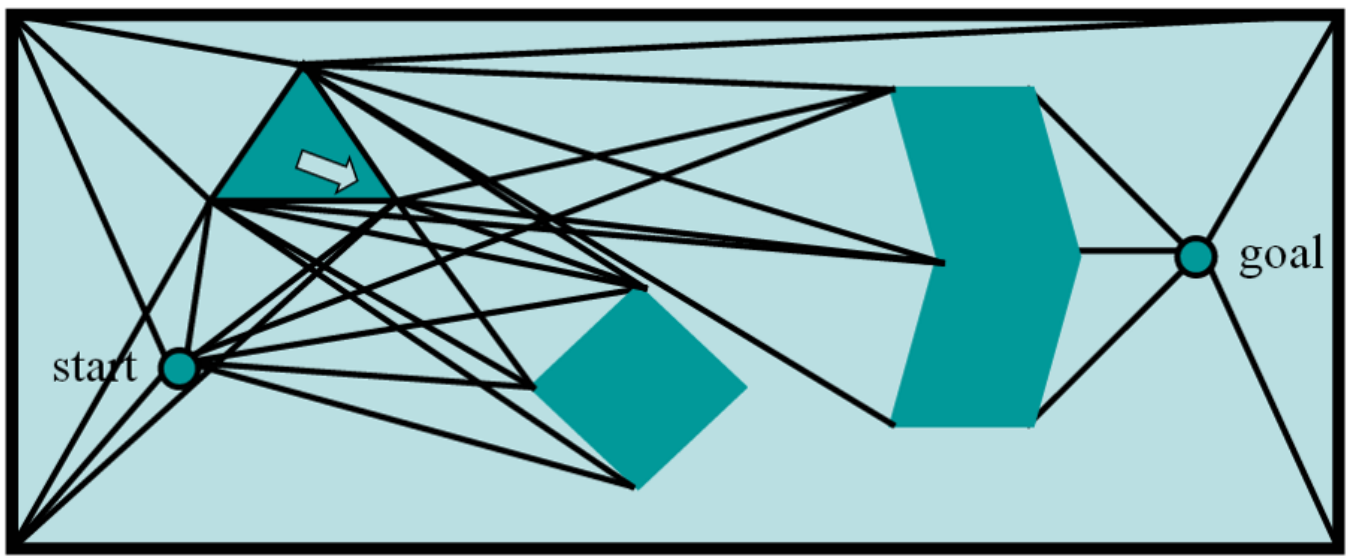
The Visibility Graph in Action

Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



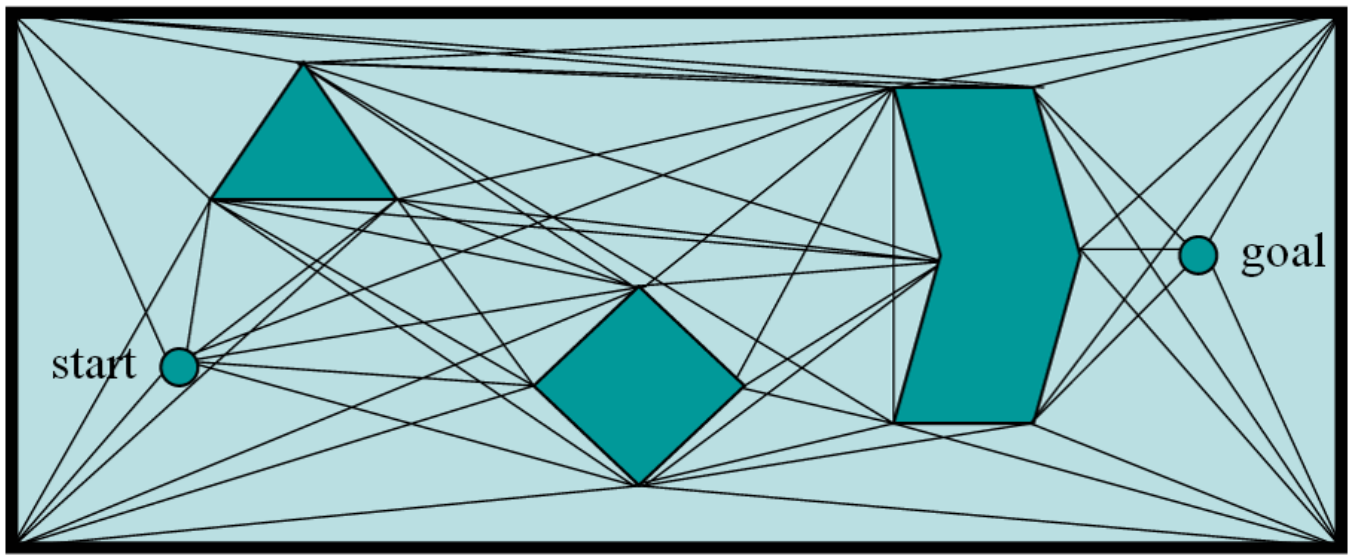
The Visibility Graph in Action

Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.

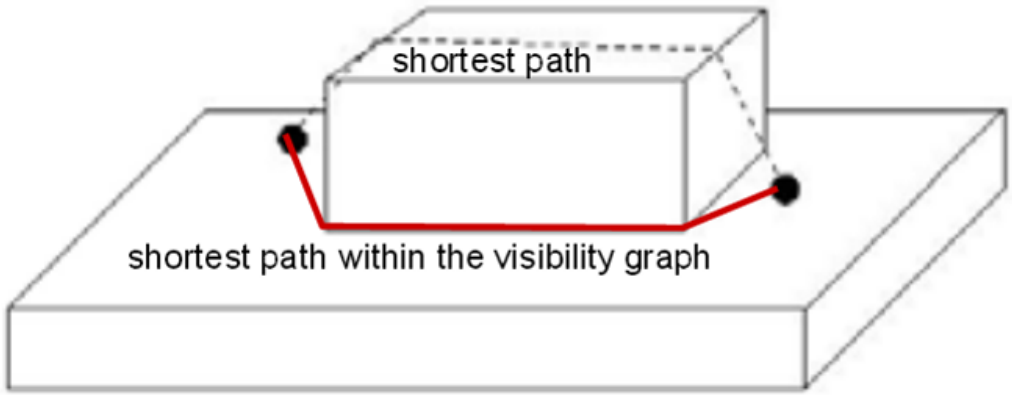


The Visibility Graph in Action

Repeat until done.

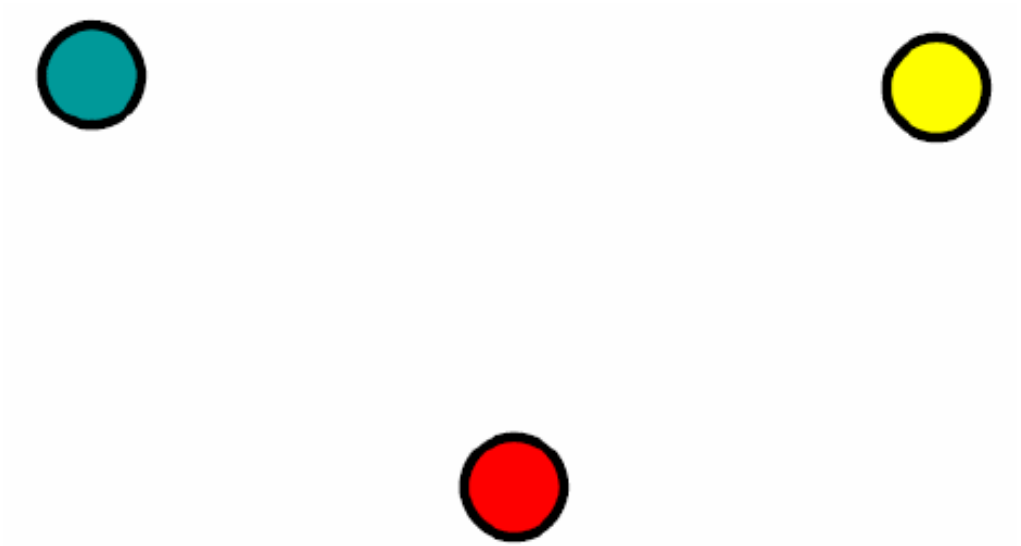


Optimality



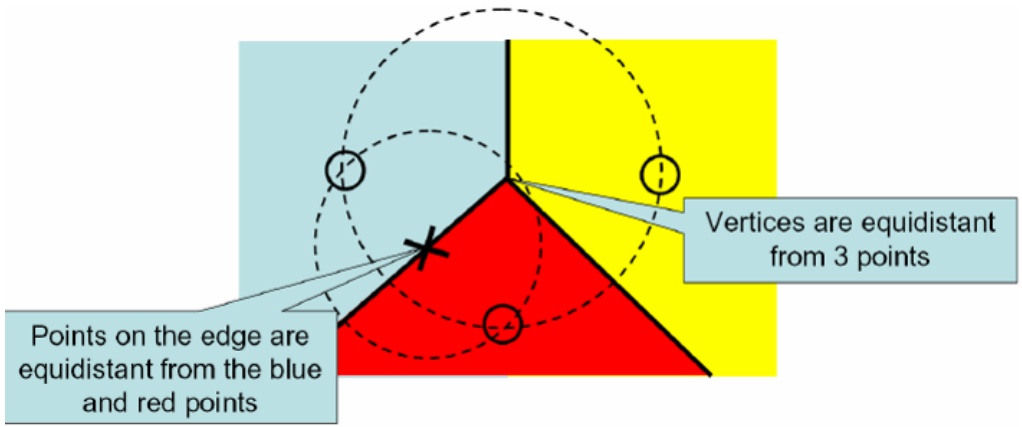
- Visibility graphs do not preserve their optimality in higher dimensions
- In addition, the paths they find are "semi-free," i.e. in contact with obstacles
- No clearance, any execution error will lead to a collision
- Need other types of roadmaps for safer paths

Voronoi Diagrams



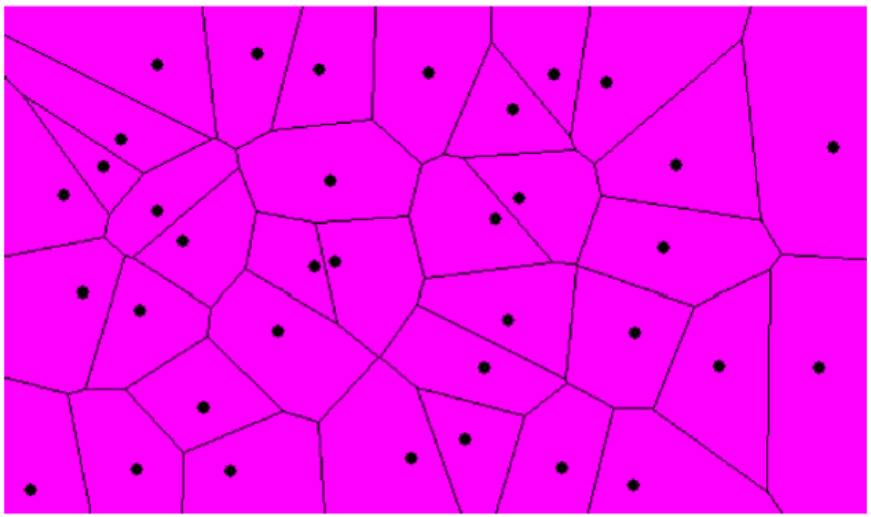
- Given a set of data points in the plane:
  - Color the entire plane such that the color of any point in the plane is the same as the color of its nearest neighbor

Voronoi Diagrams



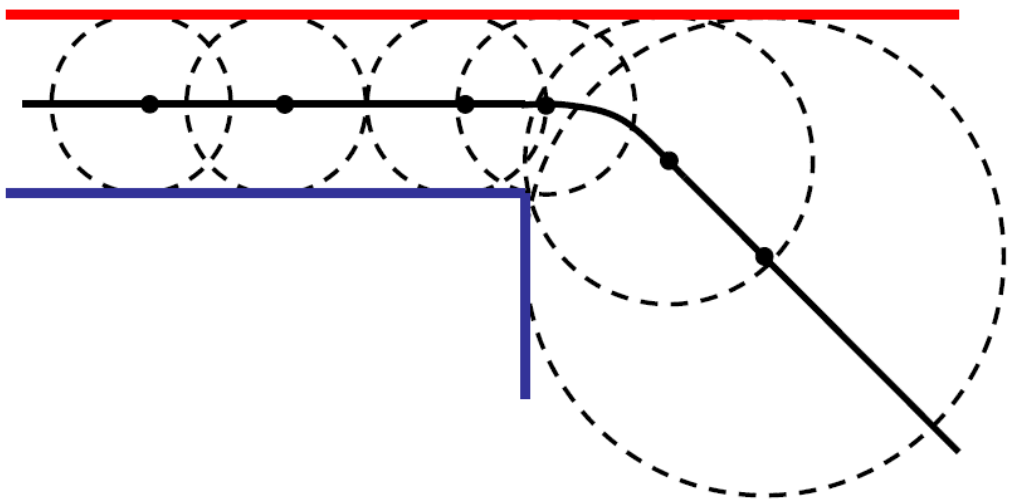
- Voronoi diagram = The set of line segments separating the regions corresponding to different colors
  - Line segment = points equidistant from 2 data points
  - Vertices = points equidistant from > 2 data points

Voronoi Diagrams

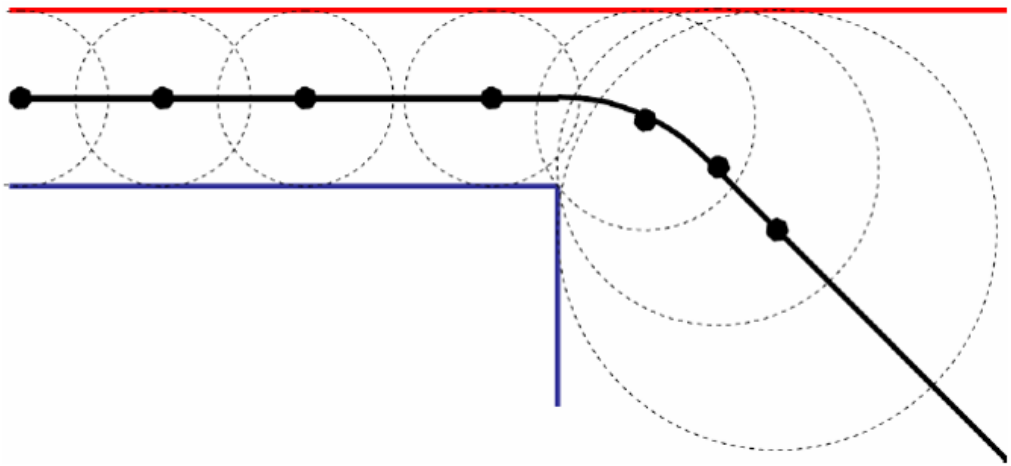


- Complexity (in the plane):
- O(N log N) time
- O(N) space

Voronoi Diagram

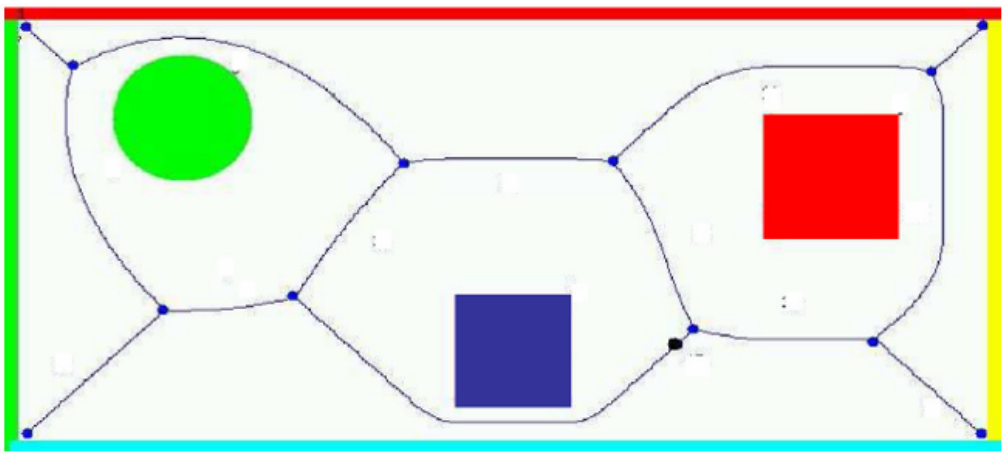


Voronoi Diagrams: Beyond Points



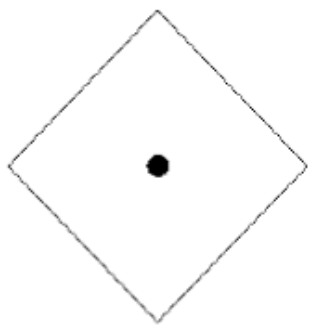
- Edges are combinations of straight line segments and segments of quadratic curves
- Straight edges: Points equidistant from 2 lines
- Curved edges: Points equidistant from one corner and one line

Voronoi Diagrams (Polygons)



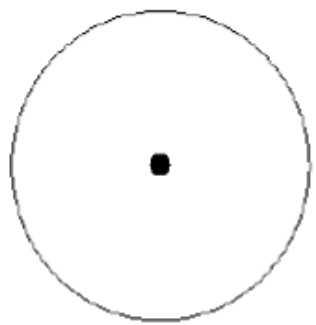
Voronoi Diagrams: Metric

- Many ways to measure distance; two are:
  - L_1 metric
    - (x,y) : |x| + |y| = const
  - L_2 metric
    - (x,y) : $x^2 + y^2$ = const



$\{(x,y) : |x| + |y| = const\}$

L1

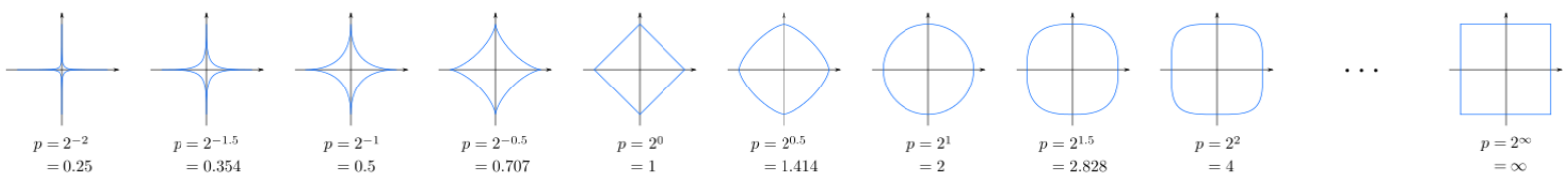


$\{(x,y) : x^2 + y^2 = const\}$

L2

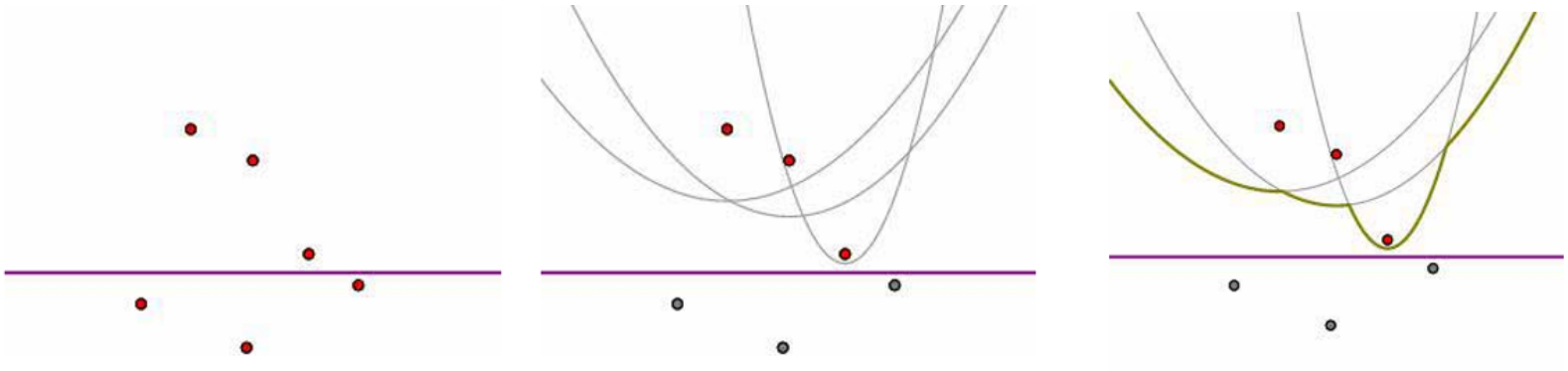
Minkowski Metric

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

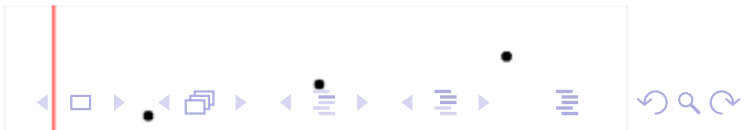


Voronoi Diagram: Construction (Roughly)

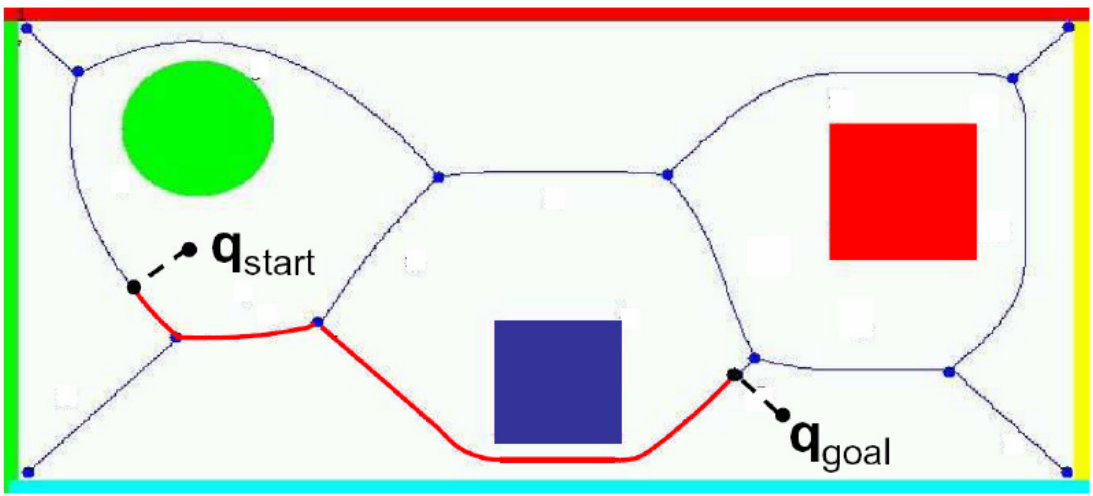
beachline



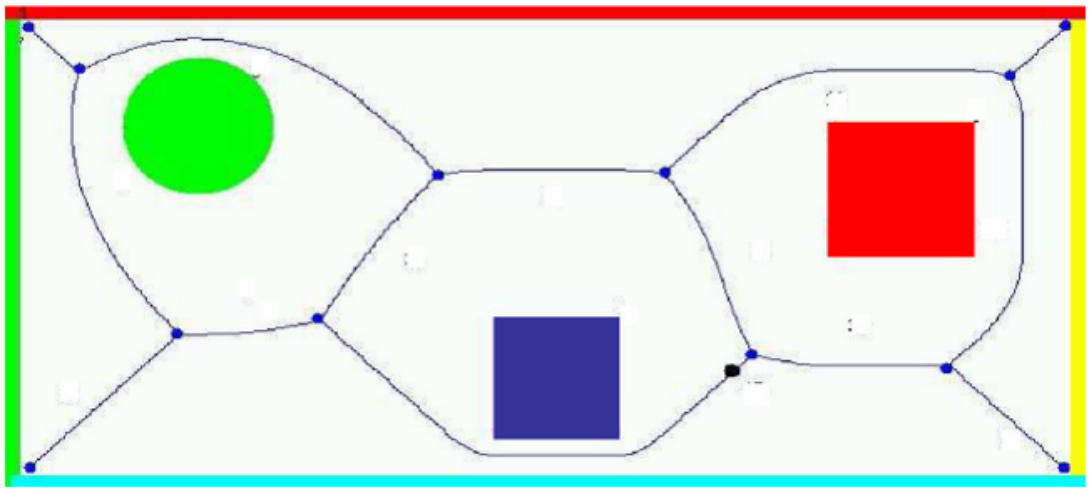
- For a Voronoi-Diagram in 2D: there are many options, e.g.: Fortune's Algorithm (a form of Sweepline Algorithm)
- Using O(n log n) time and
- O(n) space
- Find intersections of 2 parabolas (breakpoints) and intersection points of three parabolas while sweeping down



Voronoi Diagram

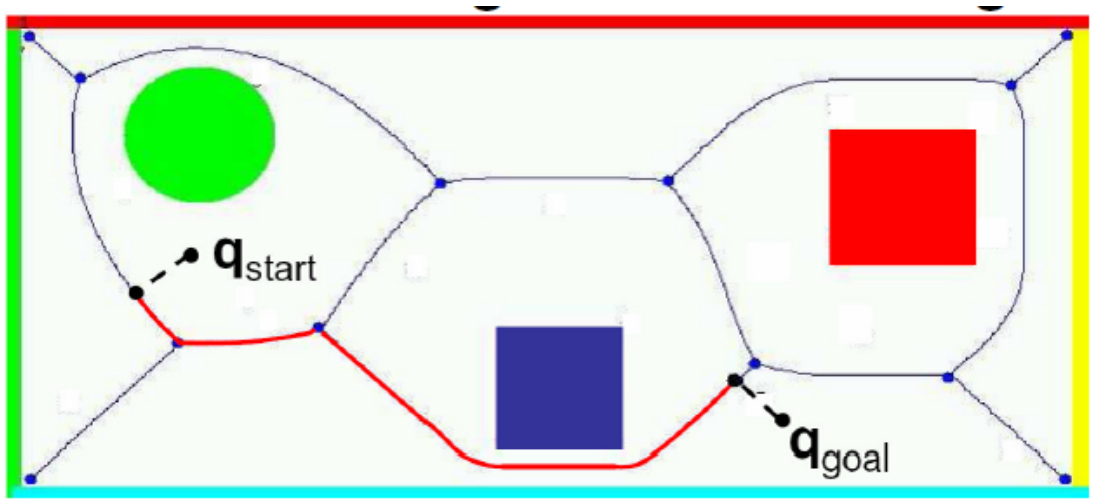


Voronoi Diagram - Polygons



- Key property: The points on the edges of the Voronoi diagram are the furthest from the obstacles
- Idea: Construct a path between q_{start} and q_{goal} by following edges on the Voronoi diagram
- Use the Voronoi diagram as a roadmap graph instead of the visibility graph

Voronoi Diagram Planning



- Find the point q^*_{start} of the Voronoi diagram closest to q_{start}
- Find the point q^*_{goal} of the Voronoi diagram closest to q_{goal}
- Compute shortest path from q^*_{start} to q^*_{goal} on the Voronoi diagram

Voronoi: Weaknesses

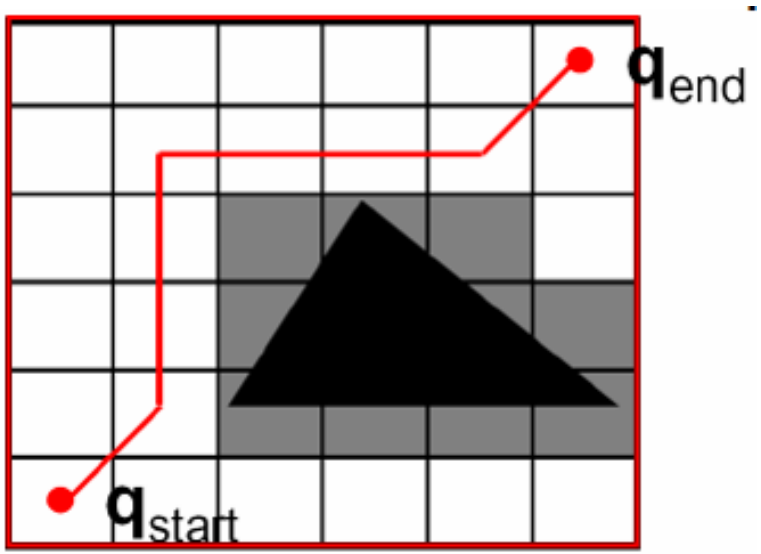
- Difficult to compute in higher dimensions or nonpolygonal worlds
- Approximate algorithms exist
- Use of Voronoi is not necessarily the best heuristic ("stay away from obstacles") can lead to paths that are much too conservative
- Can be unstable $\rightarrow$ small changes in obstacle configuration can lead to large changes in the diagram

Approximate Cell Decomposition

Idea

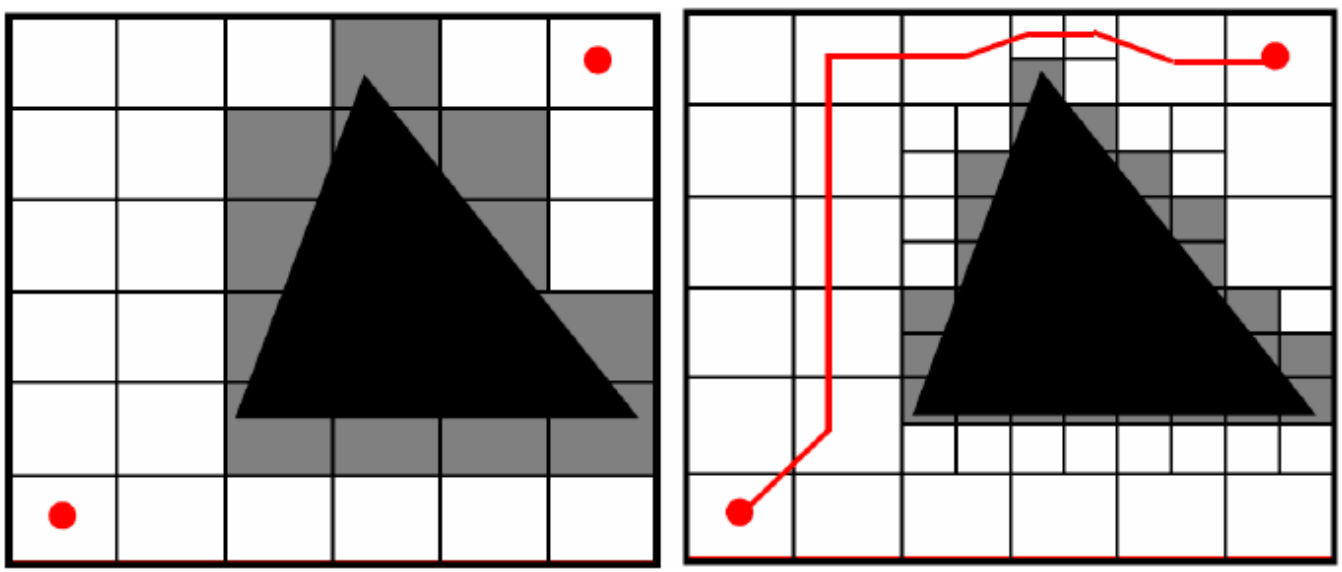
Decompose the space into cells so that any path inside a cell is obstacle free.

Approximate Cell Decomposition

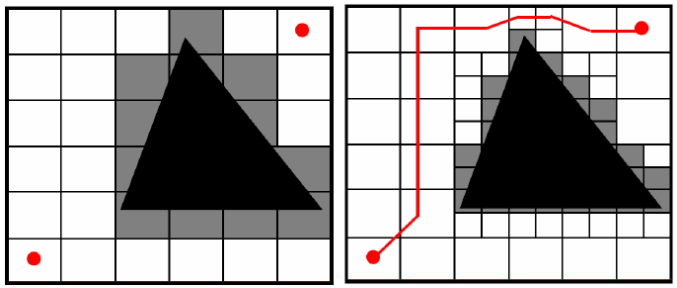


- Define a discrete grid in C-Space
- Mark any cell of the grid that intersects Cobs as blocked
- Find path through remaining cells by using (for example) A* (e.g., use Euclidean distance as heuristic)
- Cannot be complete as described so far. Why?

Approximate Cell Decomposition

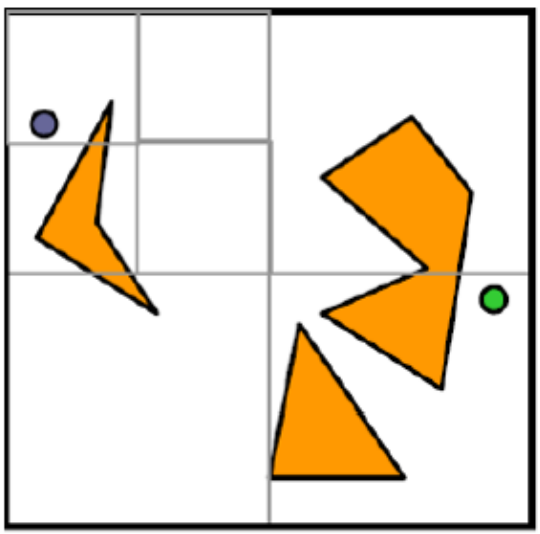


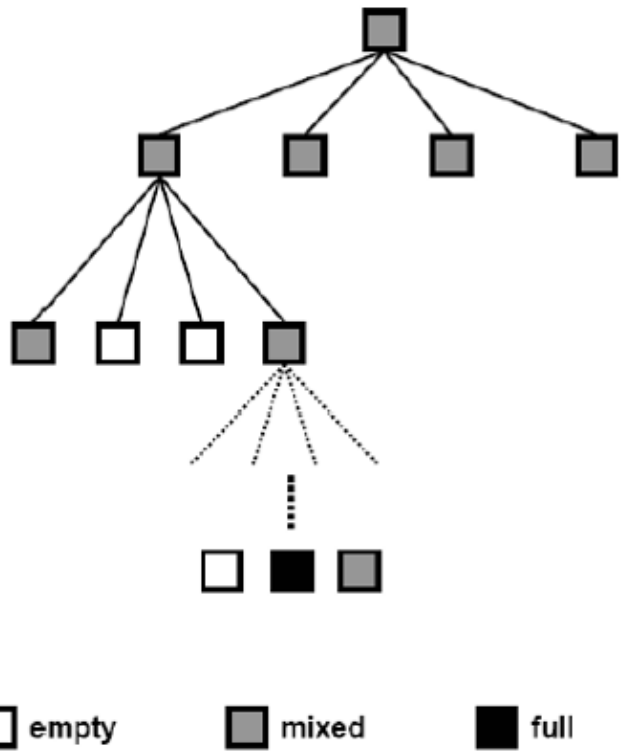
Approximate Cell Decomposition



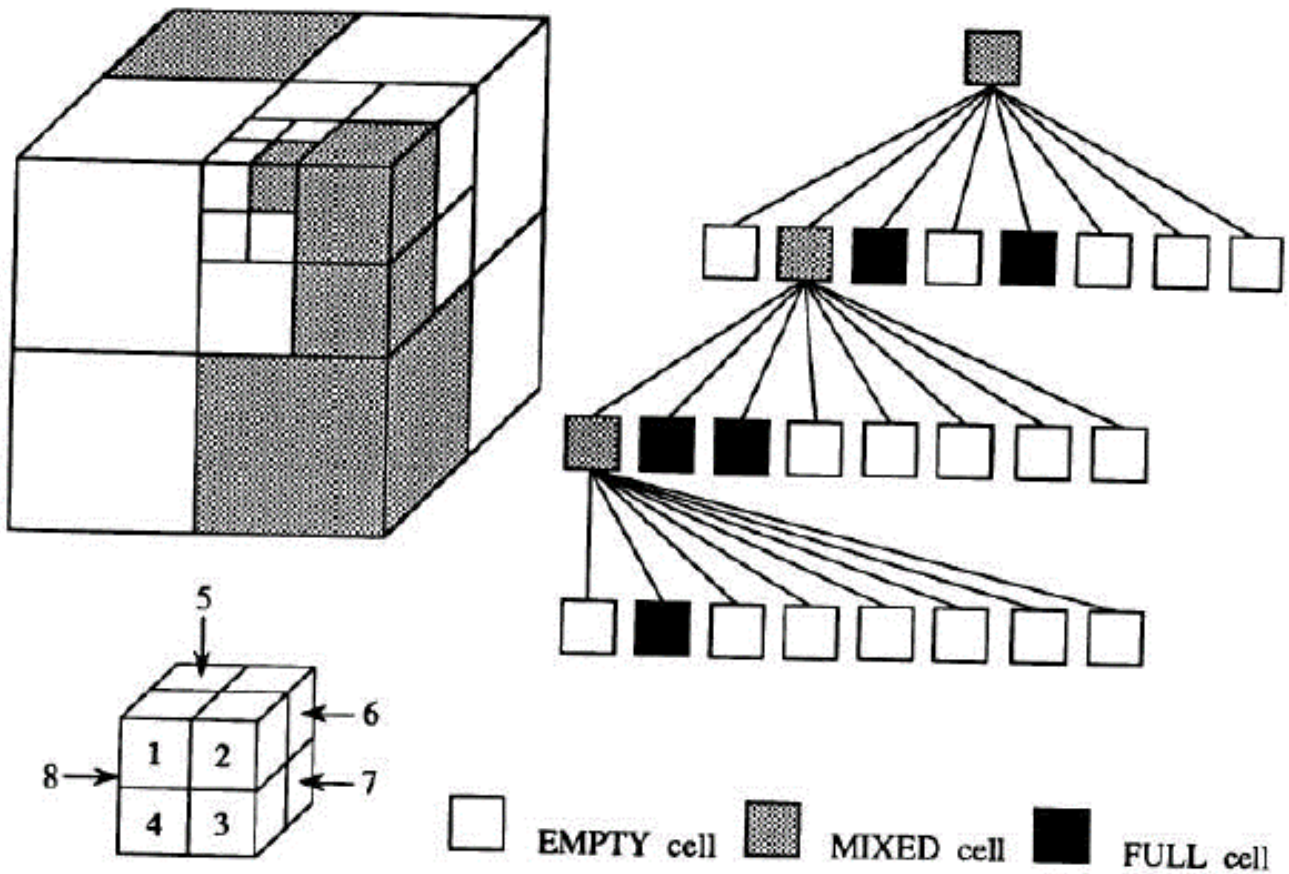
- Cannot find a path in this case even though one exists
- Solution:
- Distinguish between
  - Cells that are entirely contained in C_{obs} (FULL) and
  - Cells that partially intersect C_{obs} (MIXED)
- Try to find a path using the current set of cells
- If no path found:
  - Subdivide the MIXED cells and try again with the new set of cells

Quadtree Decomposition

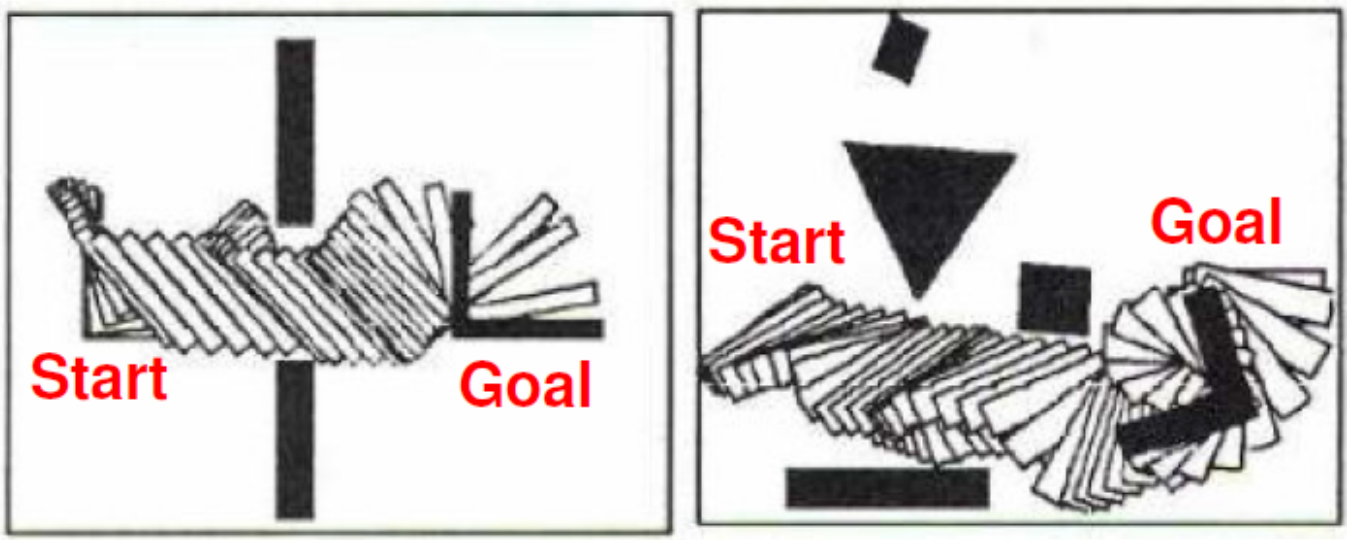




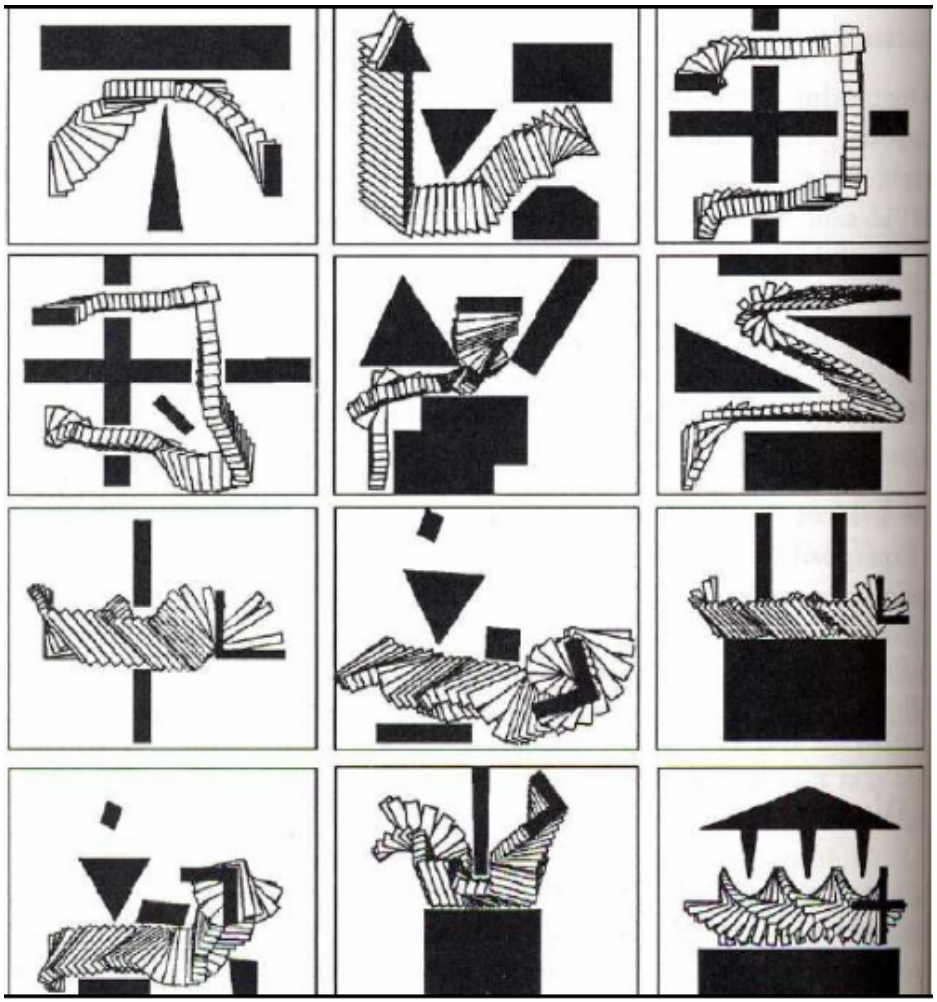
Octree Decomposition



Example



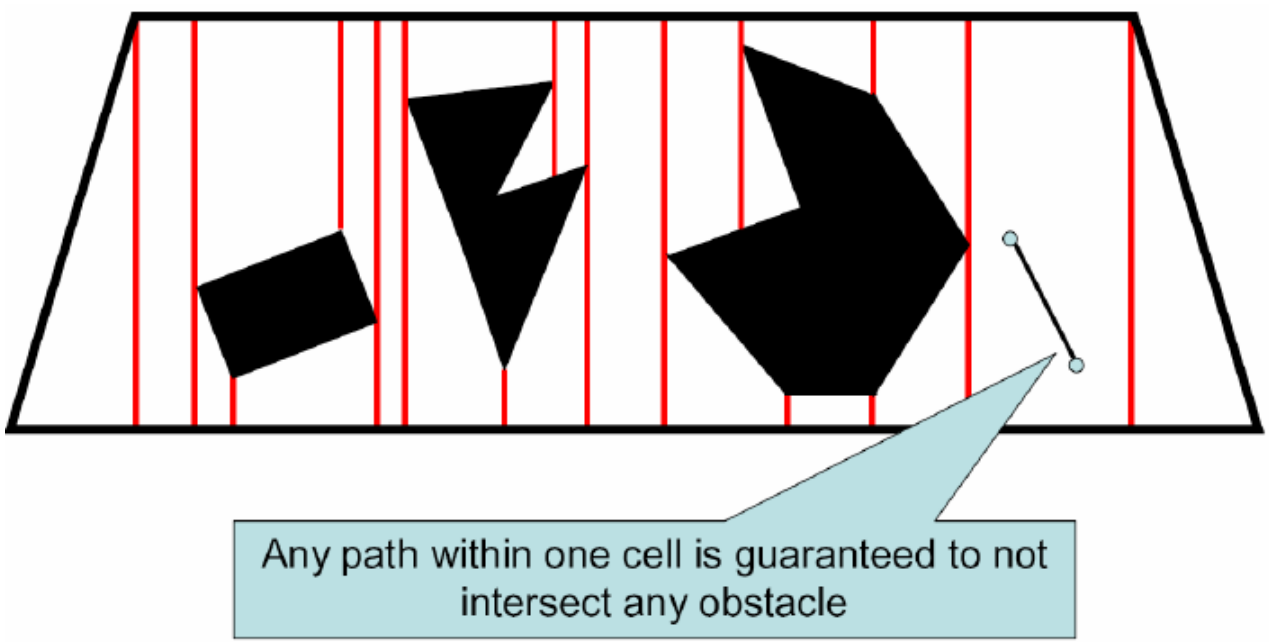
Example



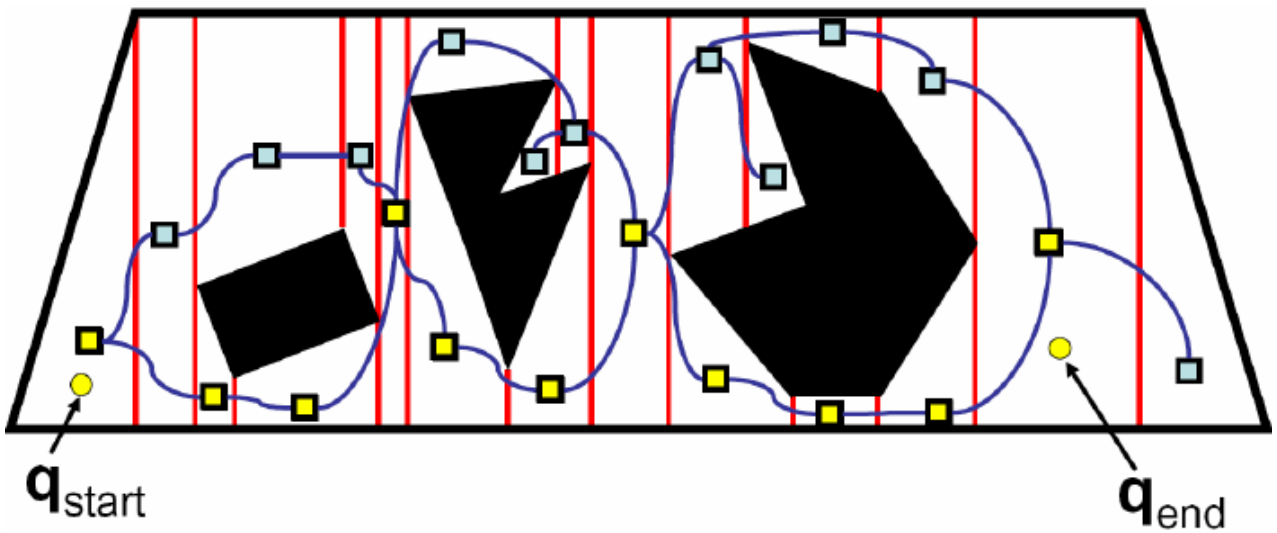
Approximate Cell Decomposition: Limitations

- Good:
  - Limited assumptions on obstacle configuration
  - Approach used in practice
  - Find obvious solutions quickly
- Bad:
  - No clear notion of optimality ("best" path)
  - Trade-off completeness/computation
  - Still difficult to use in high dimensions

Exact Cell Decomposition

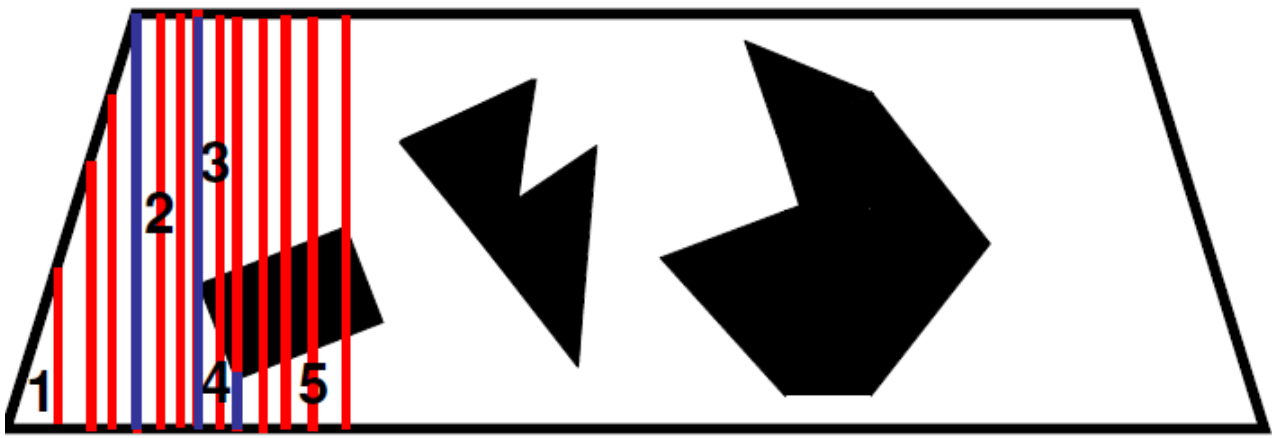


Exact Cell Decomposition



- The graph of cells defines a roadmap
- The graph can be used to find a path between any two configurations

Exact Cell Decomposition

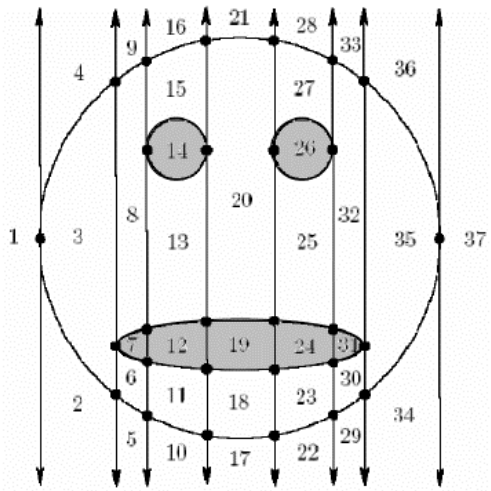


- Critical event: Create new cell
- Critical event: Split cell

Plane Sweep Algorithm

- Initialize current list of cells to empty
- Order the vertices of Cobs along the x direction
- For every vertex:
  - Construct the plane at the corresponding x location
  - Depending on the type of event:
    - Split a current cell into 2 new cells OR
    - Merge two of the current cells
  - Create a new cell

Exact Cell Decomposition



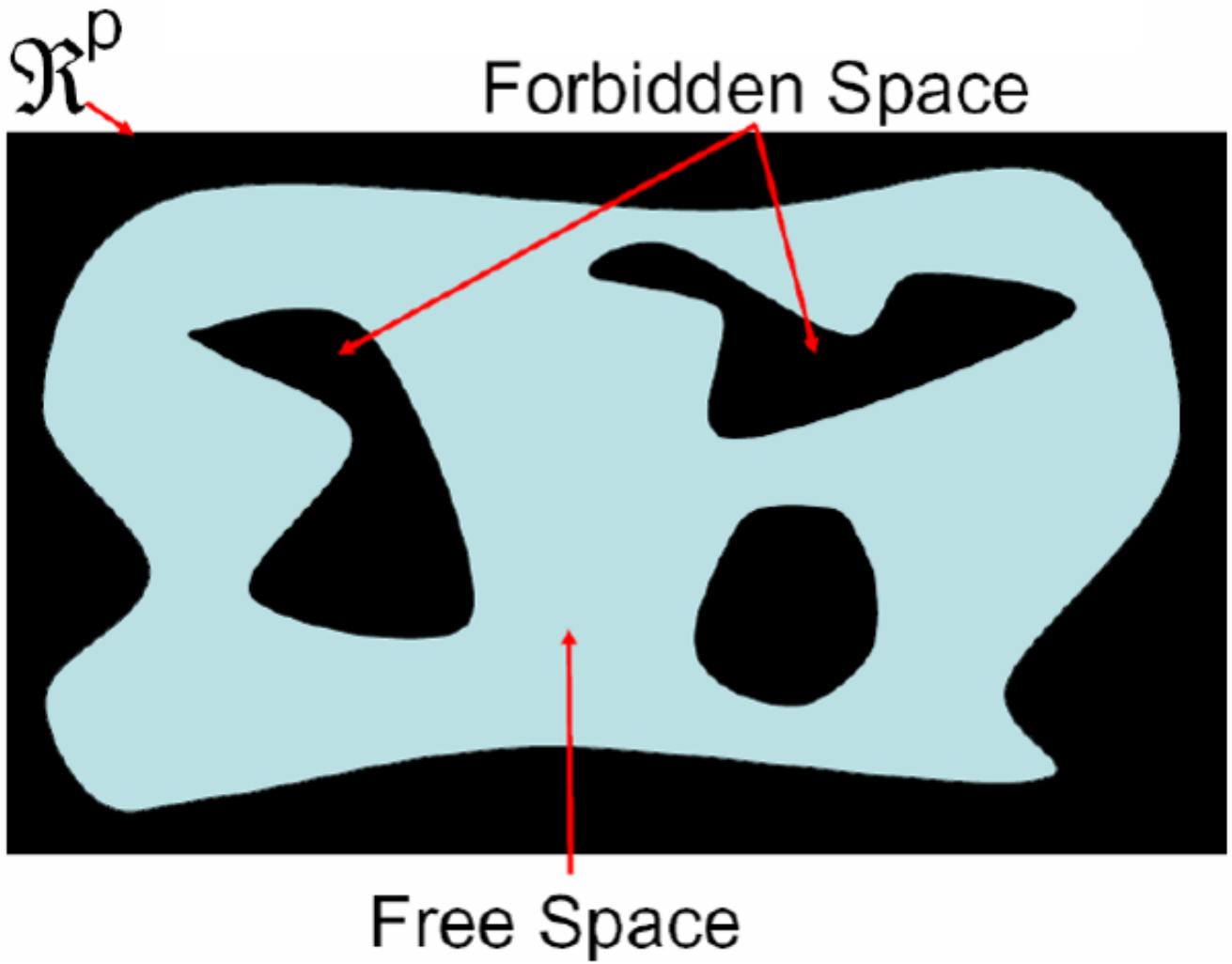
- A version of exact cell decomposition can be extended to higher dimensions and non-polygonal boundaries ("cylindrical cell decomposition")
- Provides exact solution completeness
- Expensive and difficult to implement in higher dimensions

Sampling Techniques (Probabilistic Roadmaps) [5]

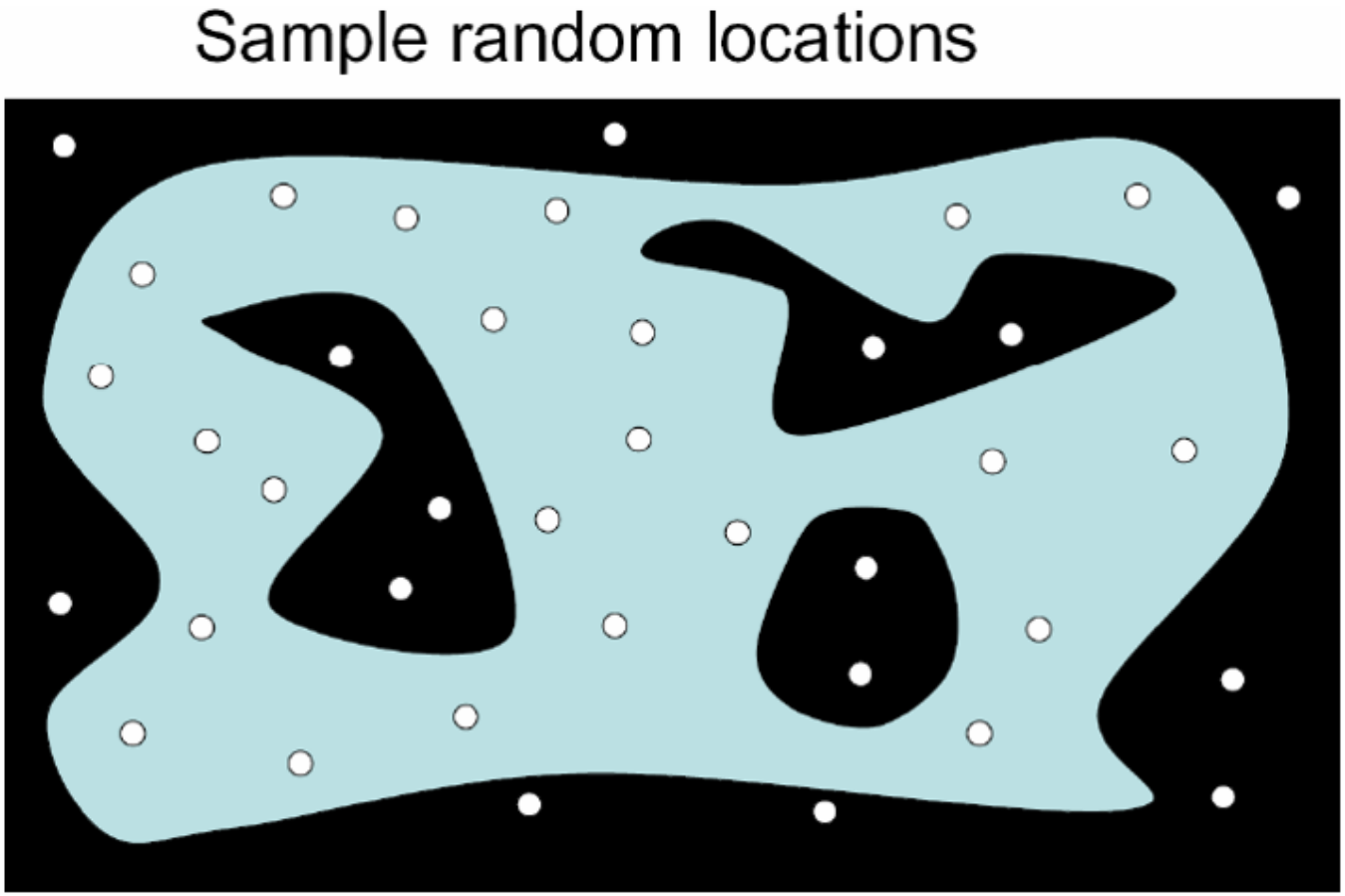
Idea

Completely describing and optimally exploring the C-space is too hard in high dimension and it is not necessary $\rightarrow$ Limit ourselves to finding a "good" sampling of the C-space

Sampling Techniques (Probabilistic Roadmaps)

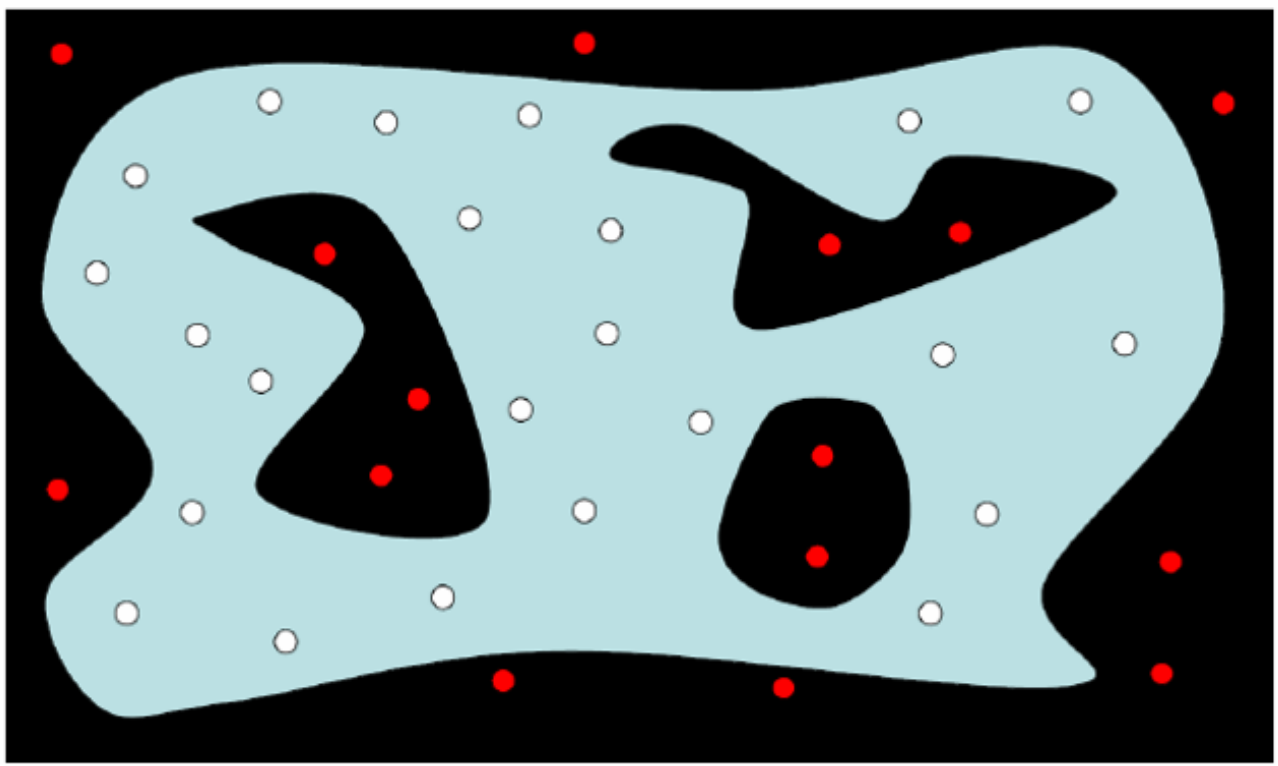


Sampling Techniques



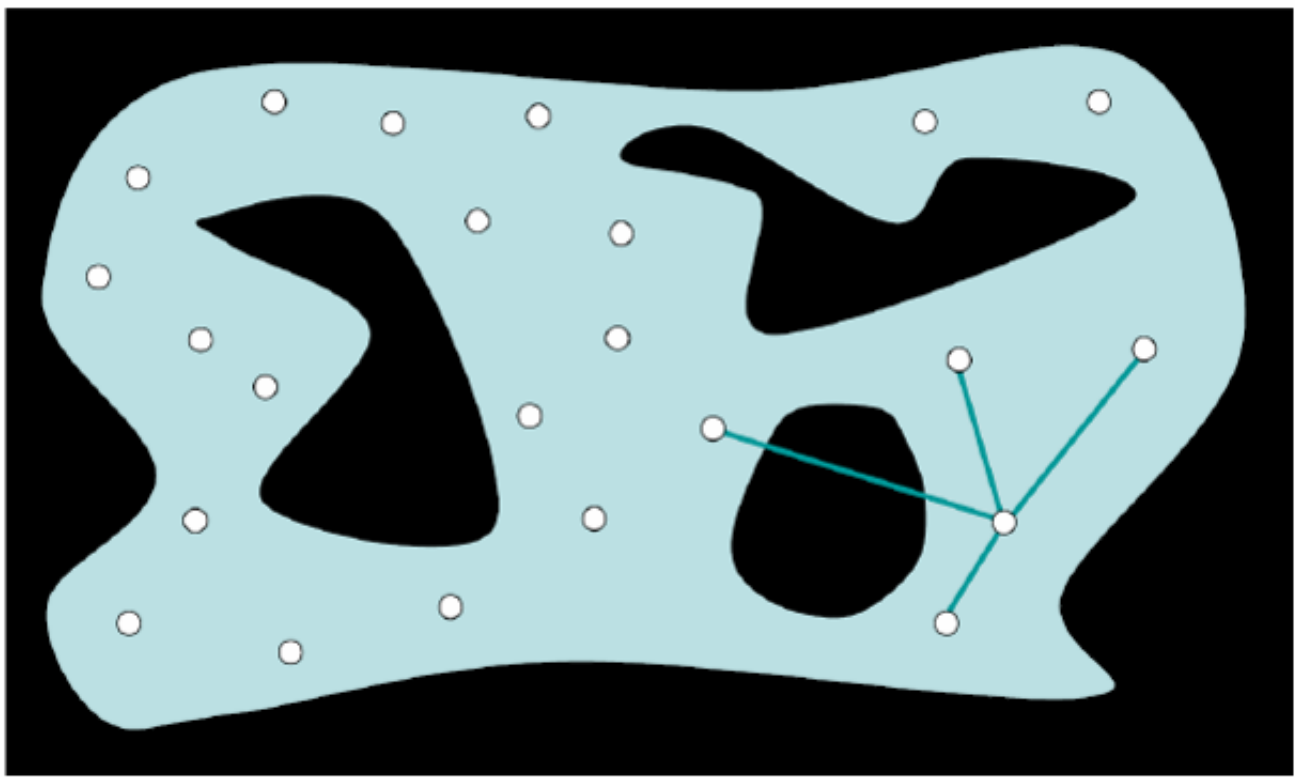
Sampling Techniques

Remove the samples in the forbidden regions



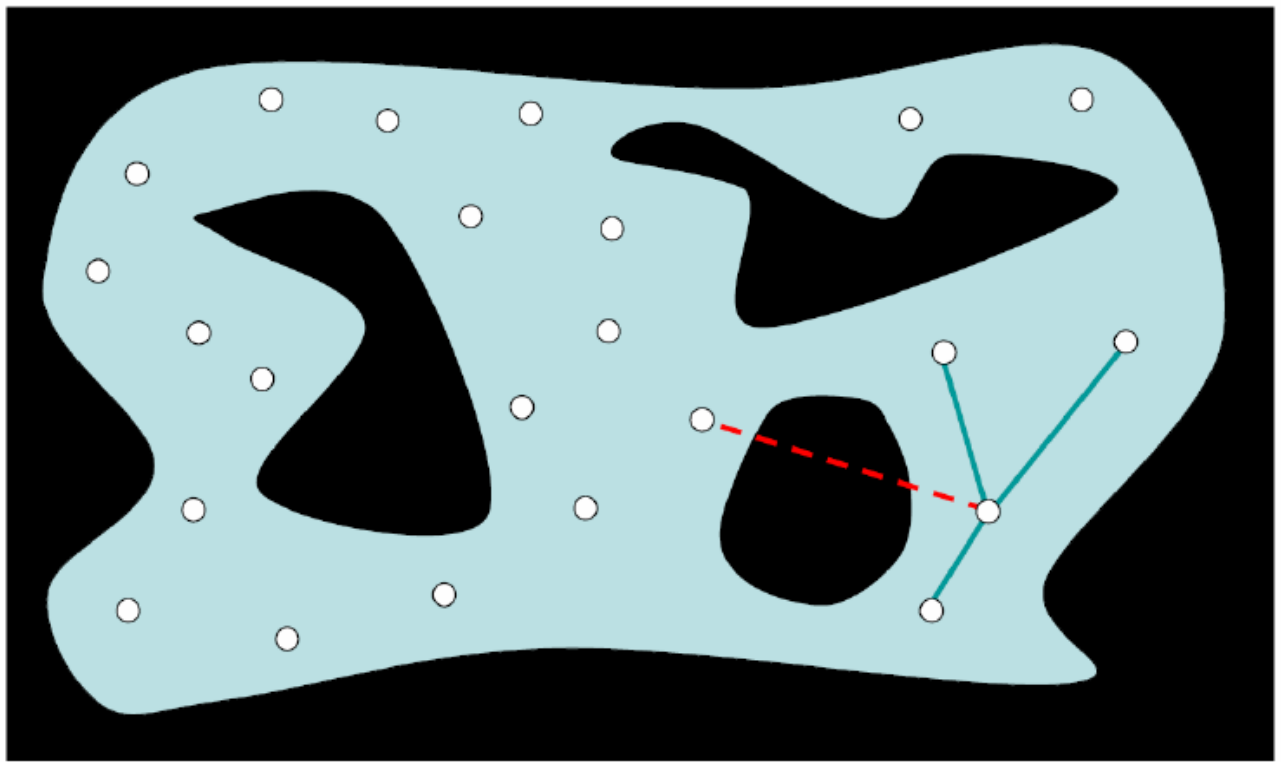
Sampling Techniques

Link each sample to its K nearest neighbors



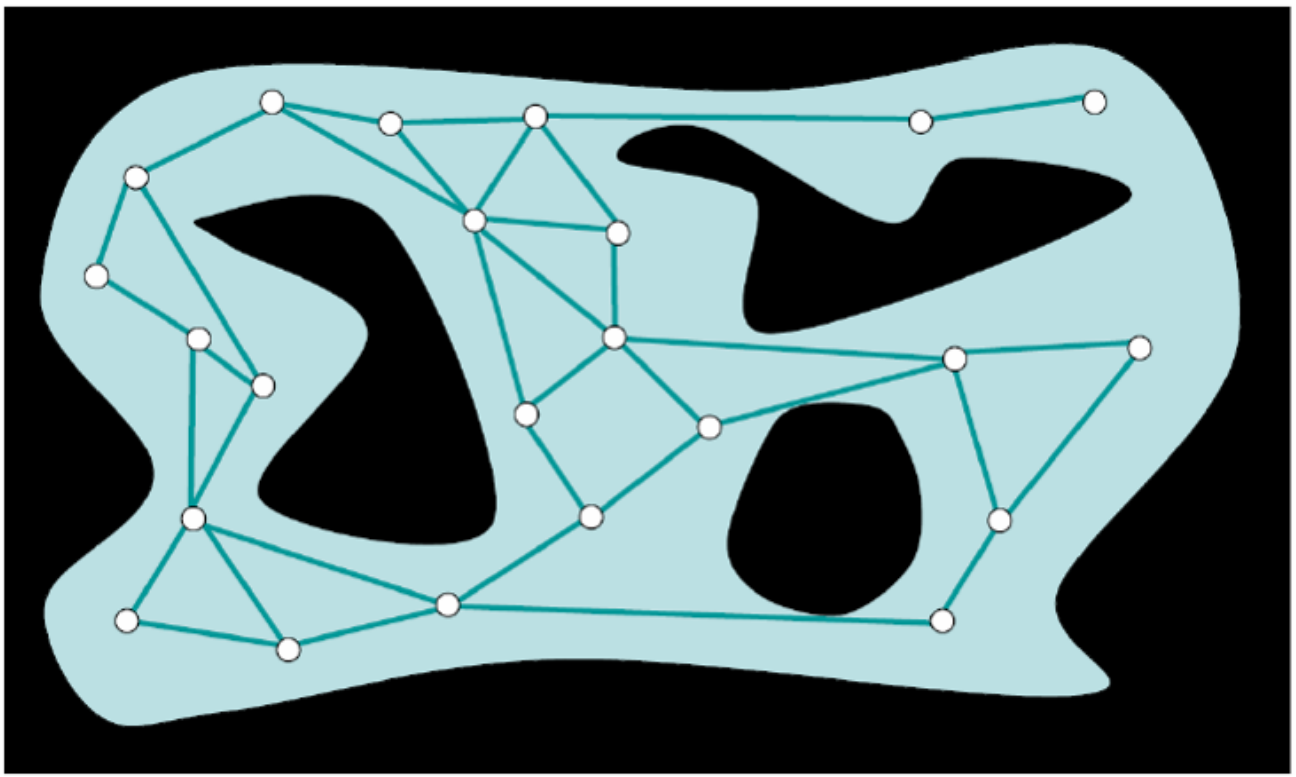
Sampling Techniques

Remove the links that cross forbidden regions



Sampling Techniques

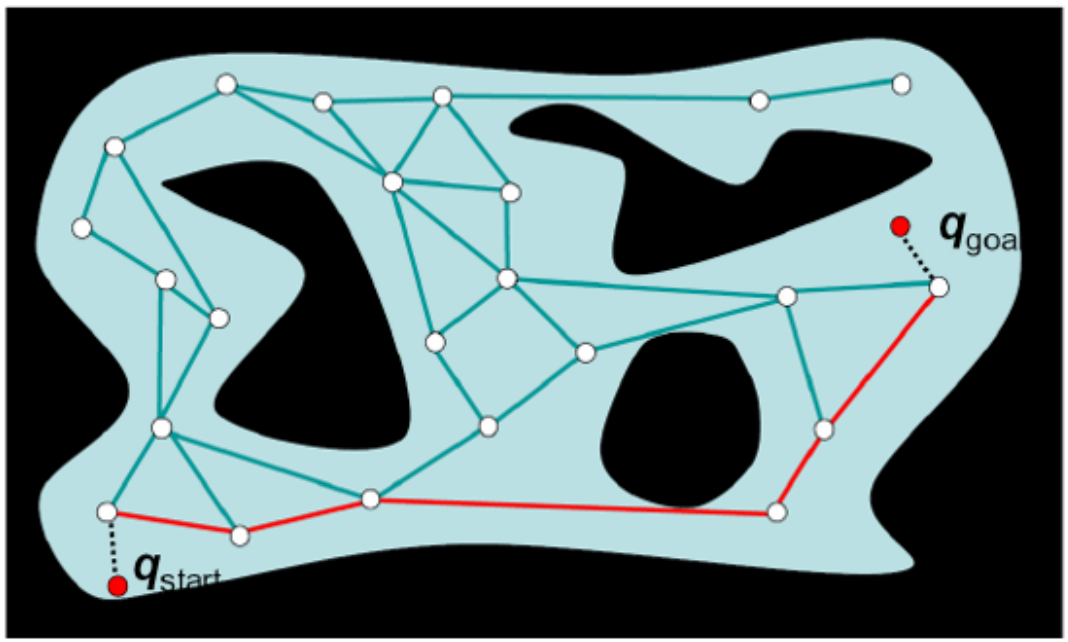
Remove the links that cross forbidden regions



The resulting graph is a *probabilistic roadmap (PRM)*

Sampling Techniques

Link the start and goal to the PRM and search using A*



Sampling Techniques

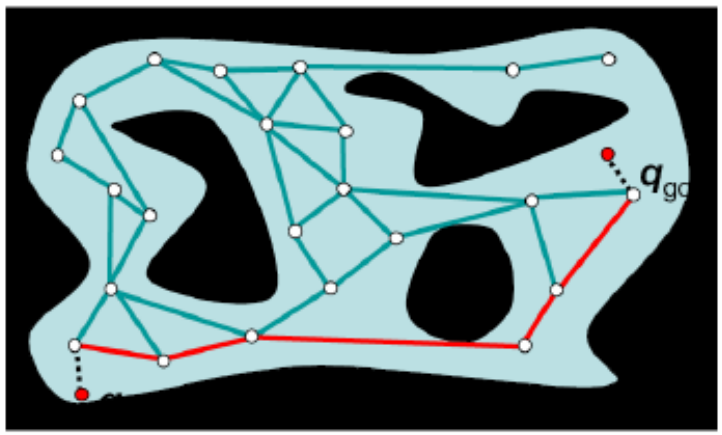
Continuous Space

↓

Discretization

↓

A* Search

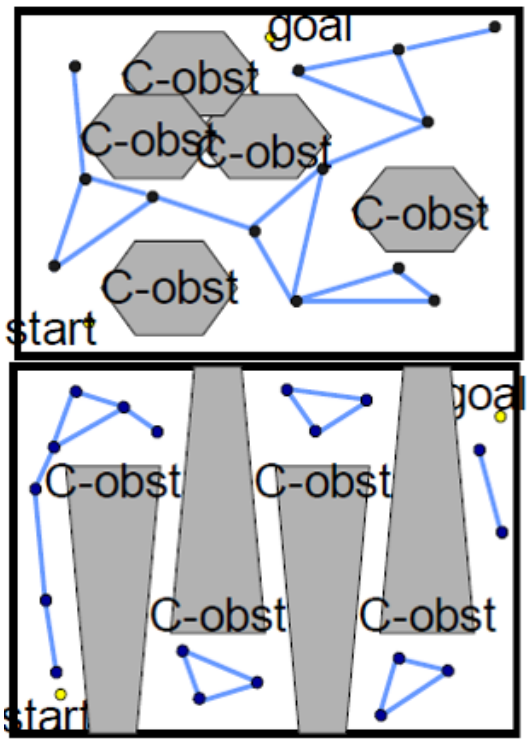


- "Good" sampling strategies are important:
  - Uniform sampling
  - Sample more near points with few neighbors
  - Sample more close to the obstacles
  - Use pre-computed sequence of samples

Sampling Techniques

- Remarkably, we can find a solution by using relatively few randomly sampled points
- In most problems, a relatively small number of samples is sufficient to cover most of the feasible space with probability 1
- For a large class of problems:
  - Probability for finding a path converges to 1 (with the number of samples)
- But cannot detect that a path does not exist

Probabilistic Roadmaps, Summary



PRMs: The Good News

1. PRMs are probabilistically complete
2. PRMs apply easily to high-dimensional C-space
3. PRMs support fast queries w/ enough preprocessing

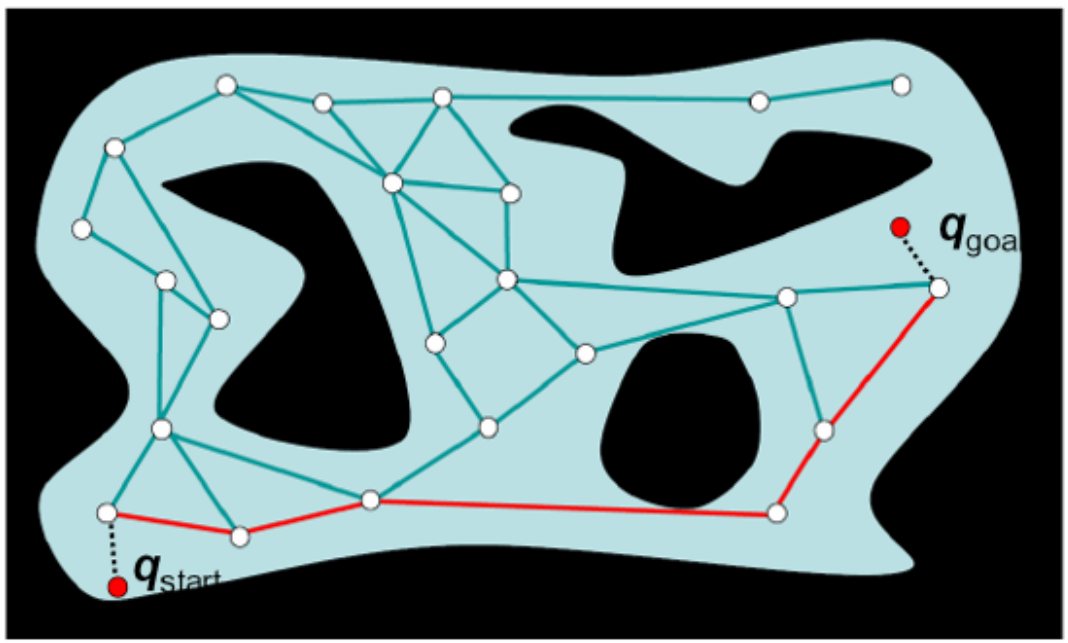
Many success stories where PRMs solve previously unsolved problems

PRMs: The Bad News

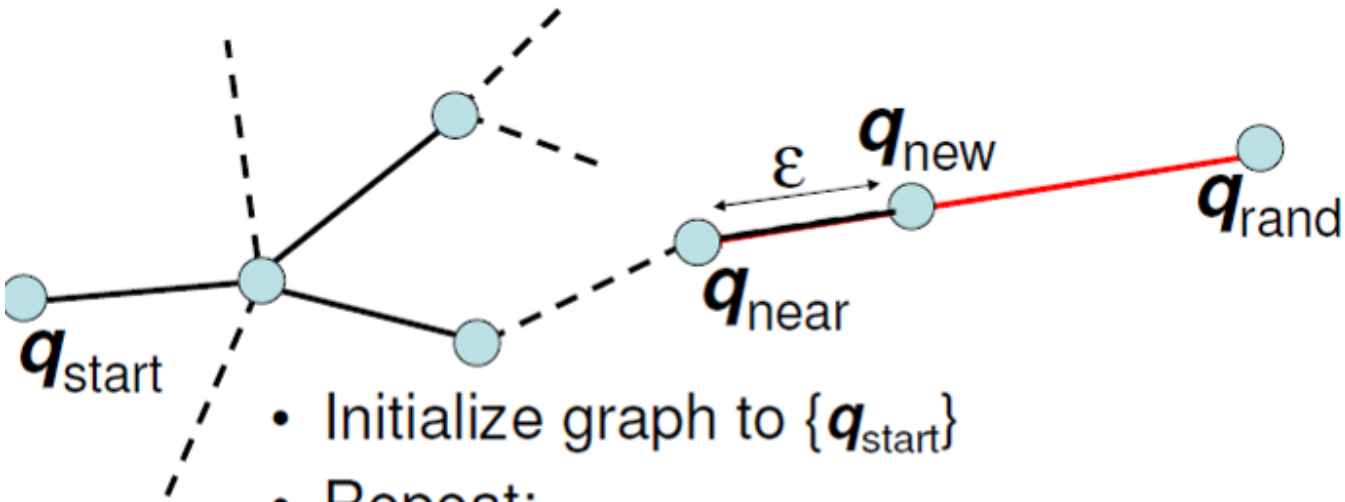
1. PRMs don't work as well for some problems:
– unlikely to sample nodes in narrow passages
– hard to sample/connect nodes on constraint surfaces

Probabilistic Roadmaps, Revisited

Link the start and goal to the PRM and search using A*



Even More Radical than PRMs: Rapidly Exploring Random Trees (RRT)



- Initialize graph to $\{q_{start}\}$
- Repeat:
  - Select random new sample q_{target}
  - Find closest node q_{near} to q_{target}
  - Create edge (q_{near}, q_{new}) if no collisions

Rapidly-Exploring Random Trees (RRT)

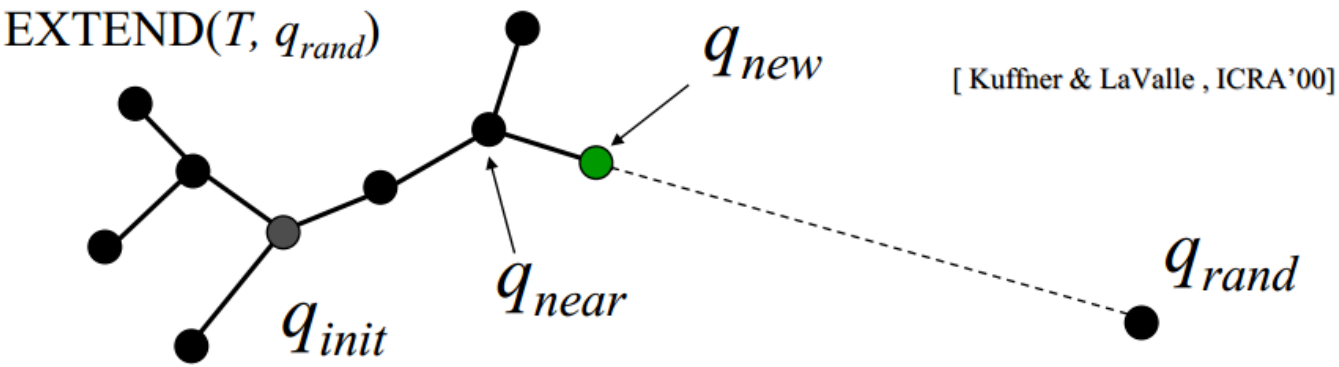
- Tree is rooted in q_{init}
- add K vertices in K loops:
  - Create random sample
  - Find closest vertex in tree (closest to sample)
  - Create "new" node in between sample and closest vertex
  - Add this "new" vertex to tree and the edge between new and closest vertex
- An RRT can be intuitively considered as a Monte-Carlo way of biasing search into largest Voronoi regions

Rapidly-Exploring Random Trees (RRT)

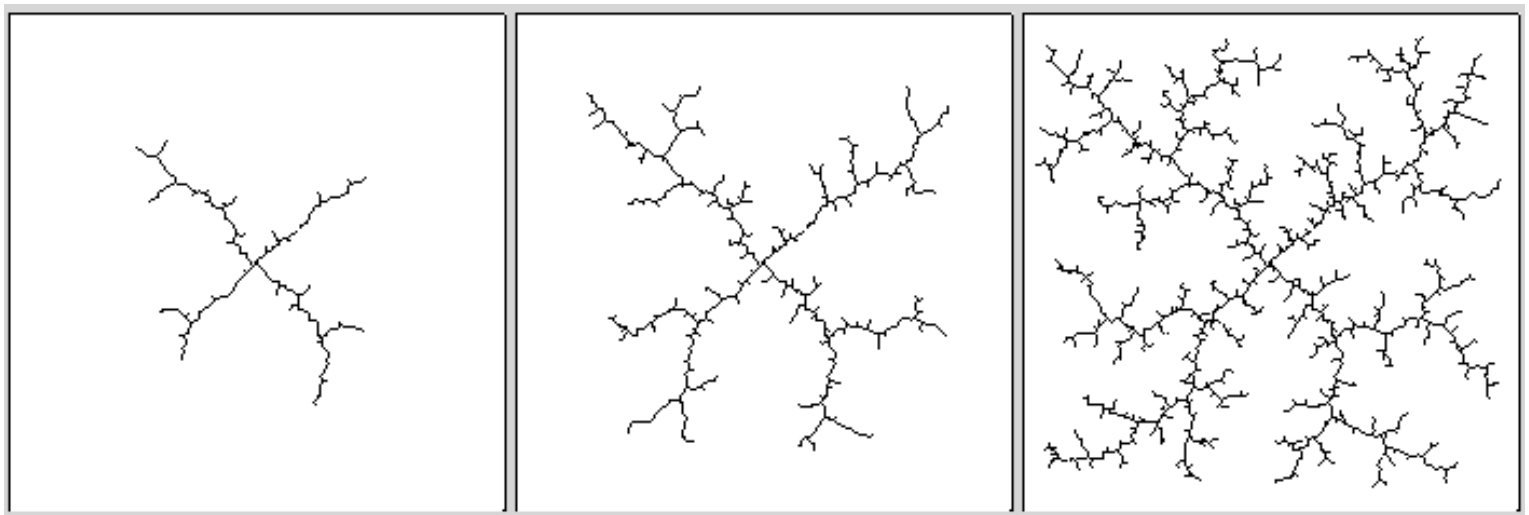
BUILD_RRT(q_{init}, K, Δq)

1 G.init(q_{init});
2 **for** $k = 1$ **to** K
3 $q_{rand} \leftarrow$ RAND_CONF();
4 $q_{near} \leftarrow$ NEAREST_VERTEX(q_{rand}, G);
5 $q_{new} \leftarrow$ NEW_CONF(q_{near}, Δq);
6 G.add_vertex(q_{new});
7 G.add_edge(q_{near}, q_{new});
8 Return G

Rapidly-Exploring Random Trees (RRT)

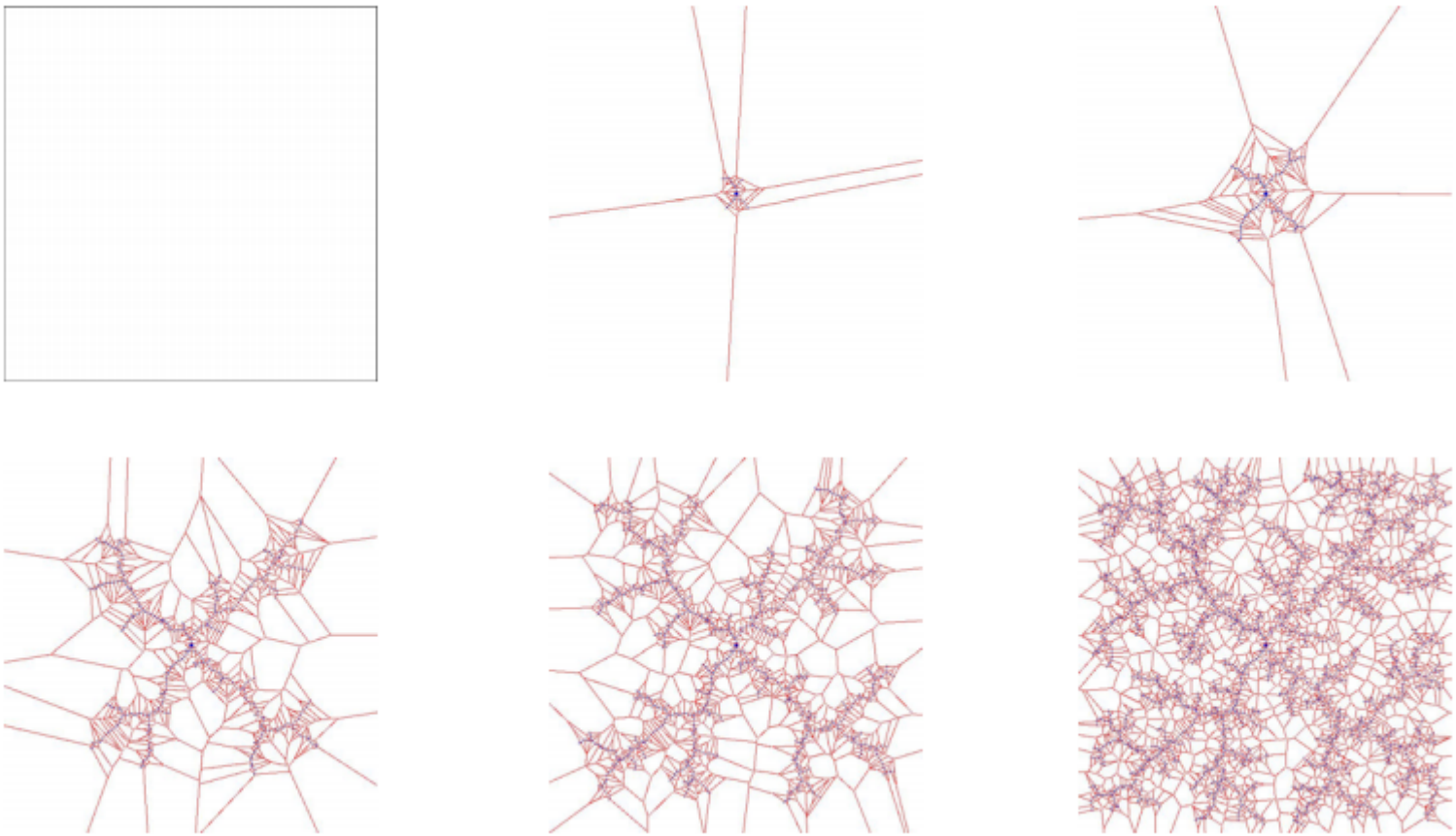


Example



- Euclidean metric
- $C = [0, 100][0, 100]$
- $q_{init} = (50, 50)$
- $\triangle q = 1$

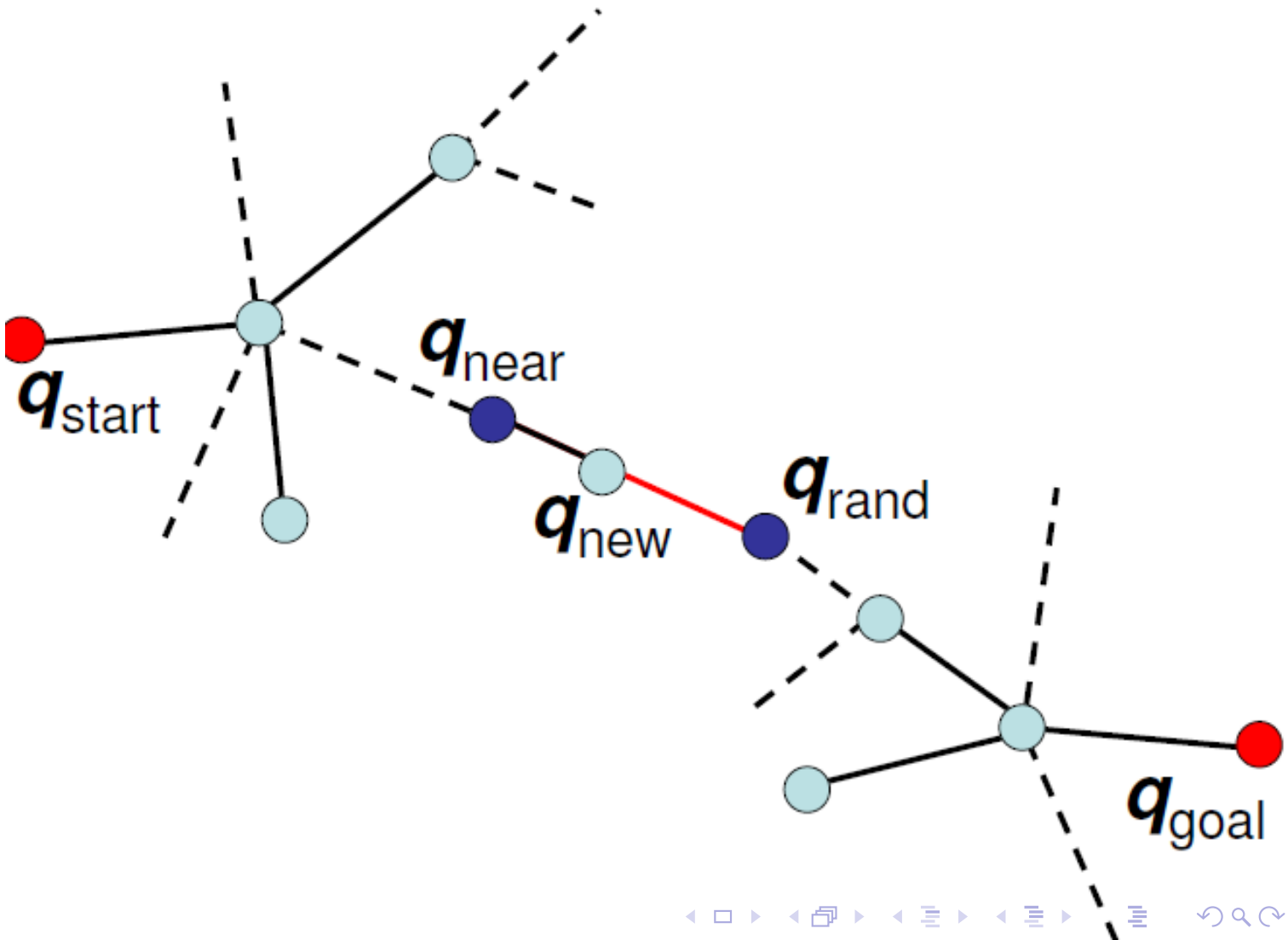
RRTs and Bias towards large Voronoi Regions



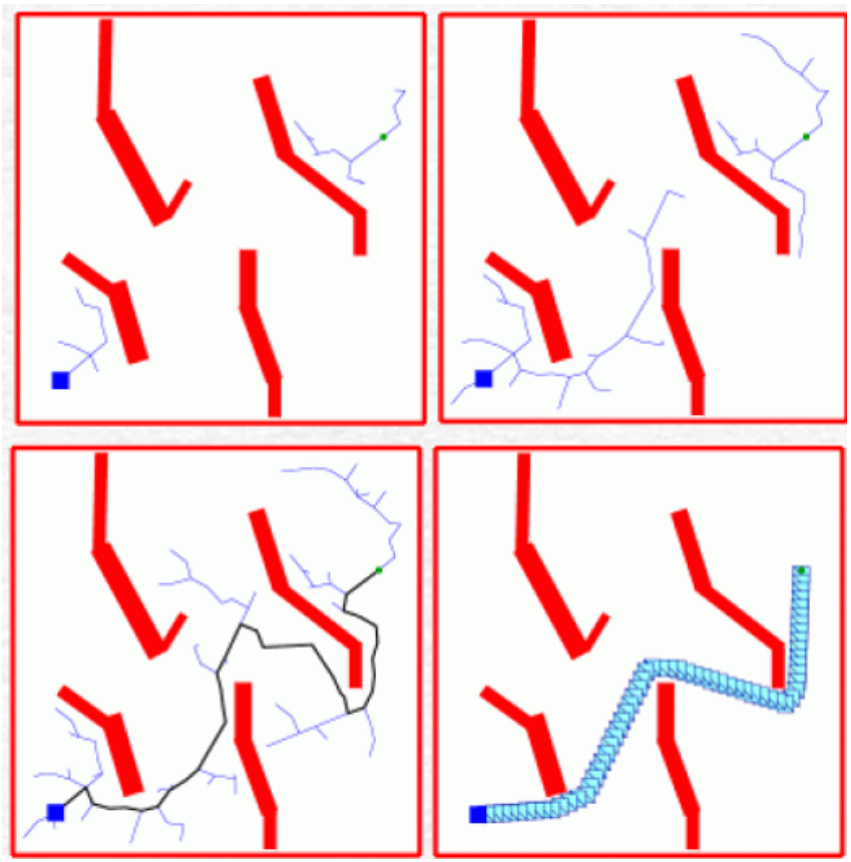
RRT and Goal Position

- Usually the sampling is not just random, with a certain factor the goal position is sampled more often.
- E.g. Uniform samling: 50%, goal sampling, 50%
- Step size is the third parameter to set

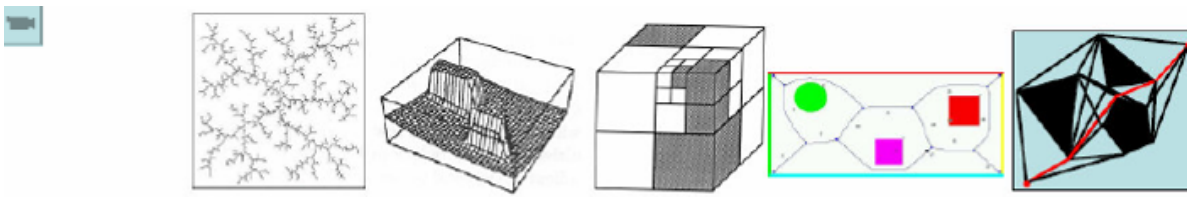
Bidirectional RRT



RRTs and Obstacles



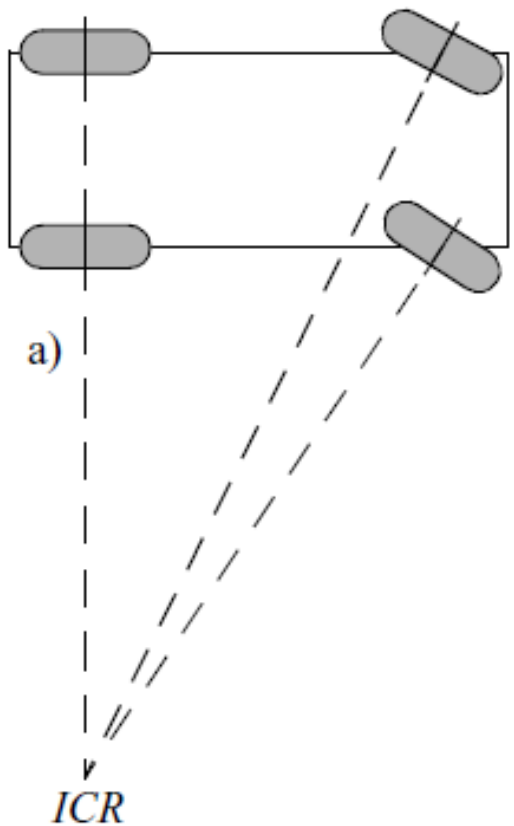
Overview of Planning Algorithms

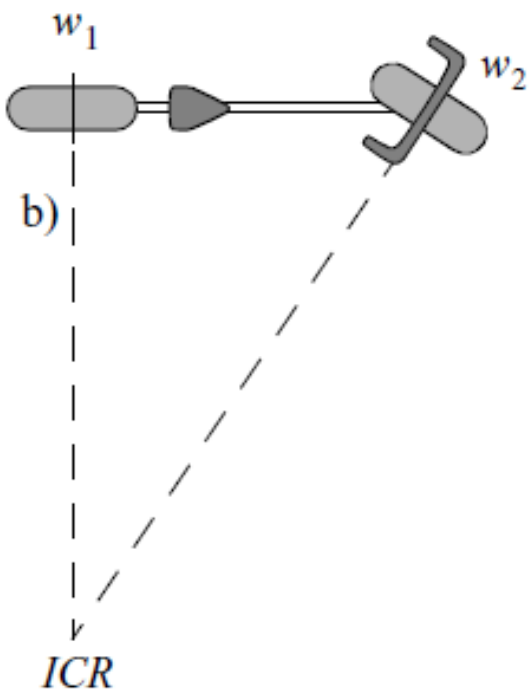


	Sampling	Potential Fields	Approx. Cell Decomposition	Voronoi	Visibility
Practical in ~2-D or 3-D	Y	Y	Y	Y	Y
Practical in >> 2-D or 3-D	Y	Y (using randomized version)	??	N	N
Fast	Y	Y	Y	In low dim.	In 2-D
Online Extensions	Y	Y	??	??	N
Complete?	Probabilistically complete	Probabilistically-resolution complete	Resolution-Complete	Y	Y

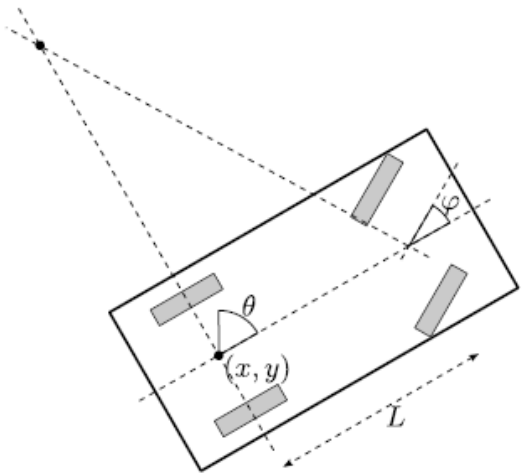
Applications for RRTs

How can plans from RRTs lead to a trajectory for a car?





Car Example



$$\dot{x} = v\ \cos\theta$$
$$\dot{y} = v\ \sin\theta$$
$$\dot{\theta} = (v/L)\ \tan\varphi$$
$$|\varphi| < \Phi$$

State $q = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$ **Controls** $u = \begin{pmatrix} v \\ \varphi \end{pmatrix}$

Sytem equation $\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v\ \cos\theta \\ v\ \sin\theta \\ (v/L)\ \tan\varphi \end{pmatrix}$

Dubins Curves [5]

- Dubins car: constant velocity, and steer $\varphi \in [-\Phi, \Phi]$
- Neglecting obstacles, there are only **six** types of trajectories that connect any configuration $\in \mathbb{R}^2 \times \mathbb{S}^1$:

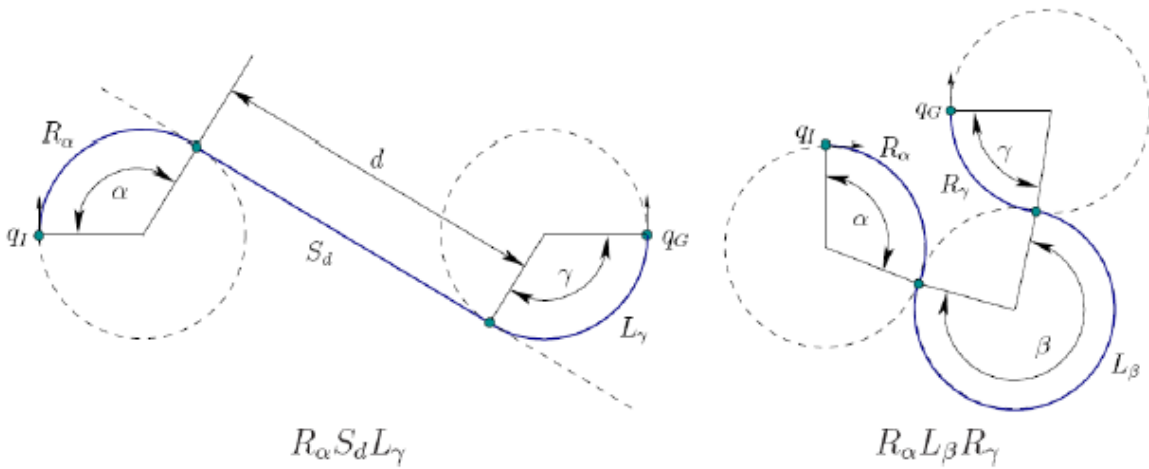
$$\{LRL, RLR, LSL, LSR, RSL, RSR\}$$

- annotating durations of each phase:

$$\{L_\alpha R_\beta L_\gamma, , R_\alpha L_\beta R_\gamma, L_\alpha S_d L_\gamma, L_\alpha S_d R_\gamma, R_\alpha S_d L_\gamma, R_\alpha S_d R_\gamma\}$$

with $\alpha \in [0, 2\pi), \beta \in (\pi, 2\pi), d \geq 0$

Dubins Curves



$\rightarrow$ By testing all six types of trajectories for (q_1, q_2) we can define a Dubins metric for the RRT – and use the Dubins curves as controls!

Literature

- [1] J.C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, 1991.
- [2] S. LaValle, Planning Algorithms. 2006. http://msl.cs.uiuc.edu/planning/
- [3] Likhachev, ARA*, CMU
- [4] Choset, Motion Planning, CMU,
- [5] Toussaint, Lecture Notes Robotics, 2011
- [6] Dr. John (Jizhong) Xiao, City College New York