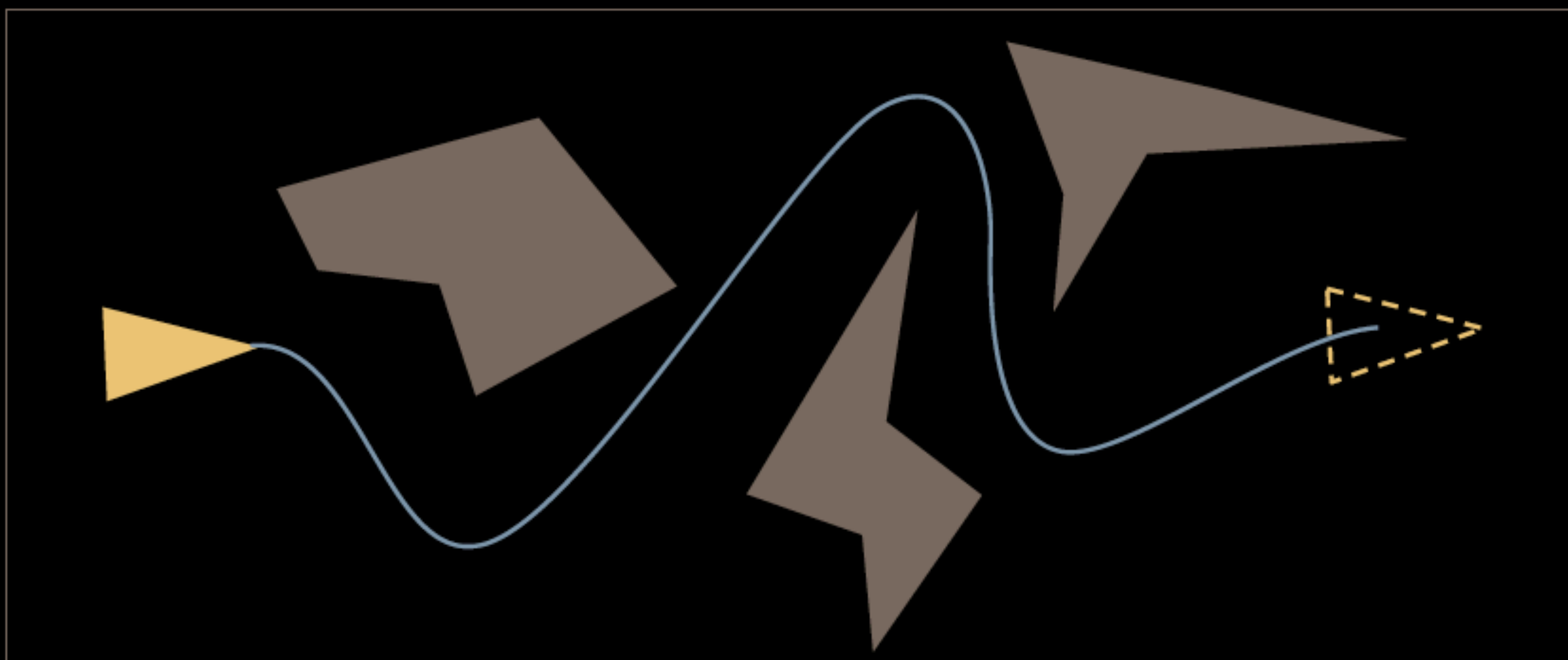
Motion Planning

D.A. Forsyth

(with slides sampled from various sources)

What is motion planning?

- The automatic generation of motion
  - Path + velocity and acceleration along the path



Li slides

Basic Problem Statement

- Motion planning in robotics
  - Automatically compute a path for an object/robot that does not collide with obstacles.

Why is this not just optimization?

- Find minimum cost set of controls that
  - take me from A to B
  - do not involve
    - collision
    - unnecessary extreme control inputs
    - unnecessary extreme behaviors

These will have to deal with collisions, etc.

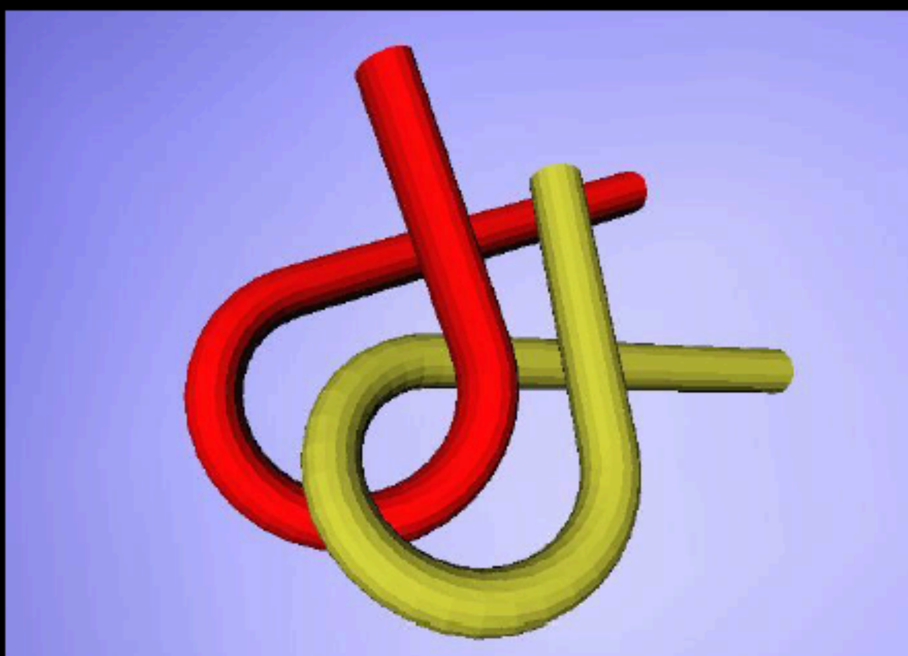
$$\text{minimize } f(\mathbf{x}) \tag{1a}$$

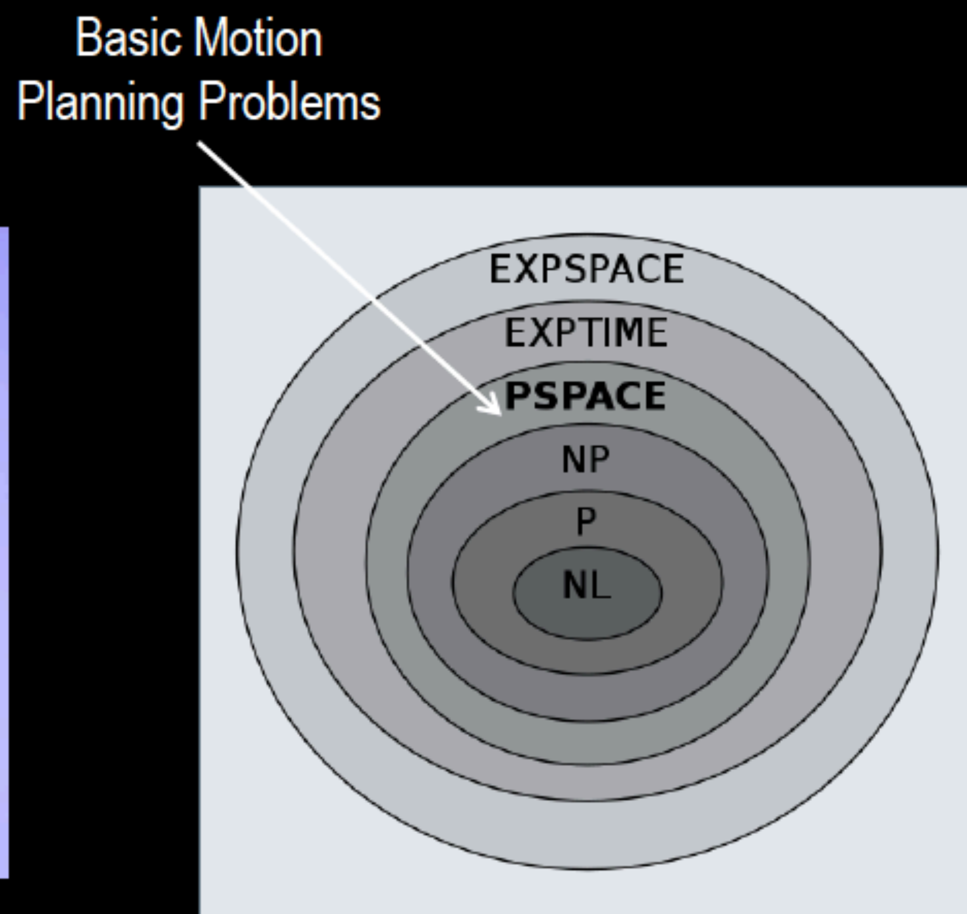
$$\text{subject to} \tag{1b}$$

$$g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \ldots, n_{ineq} \tag{1c}$$

$$h_i(\mathbf{x}) = 0, \quad i = 1, 2, \ldots, n_{eq} \tag{1d}$$

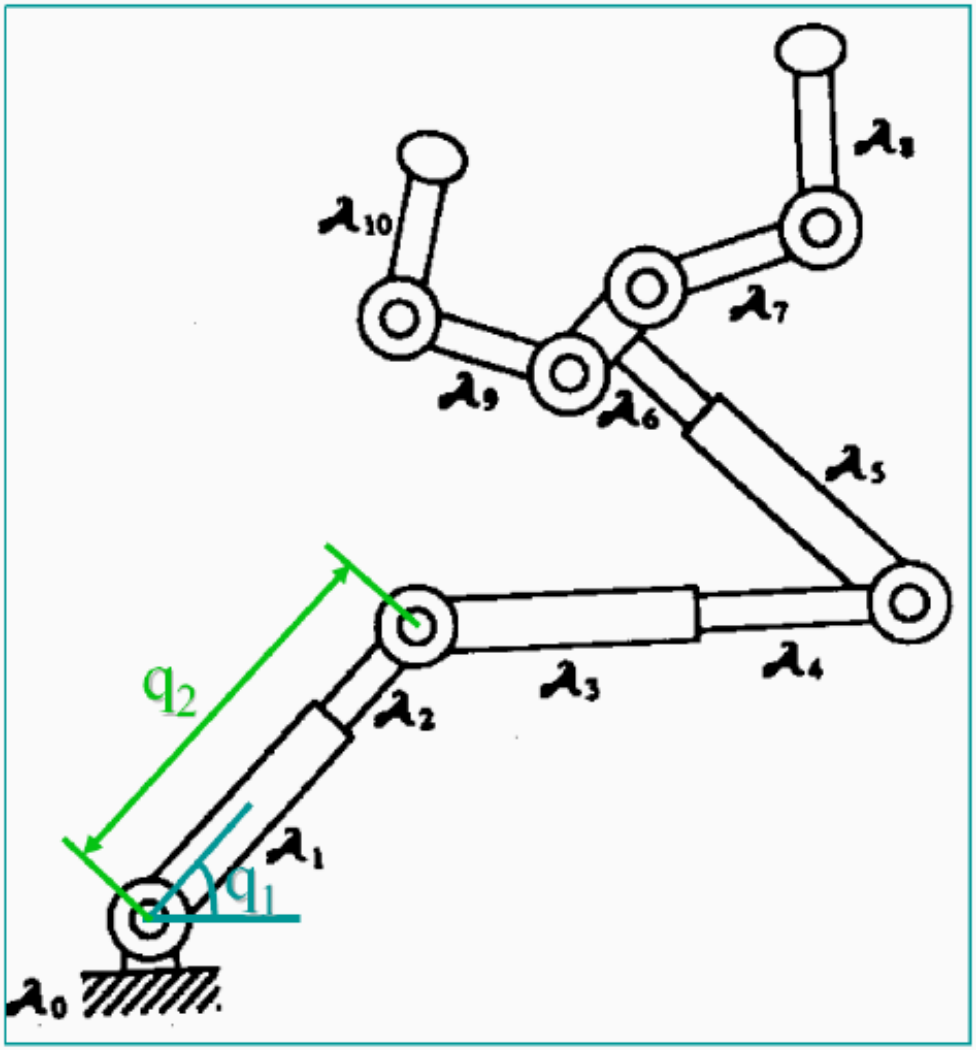
Is motion planning hard?





Li slides

Degrees of Freedom



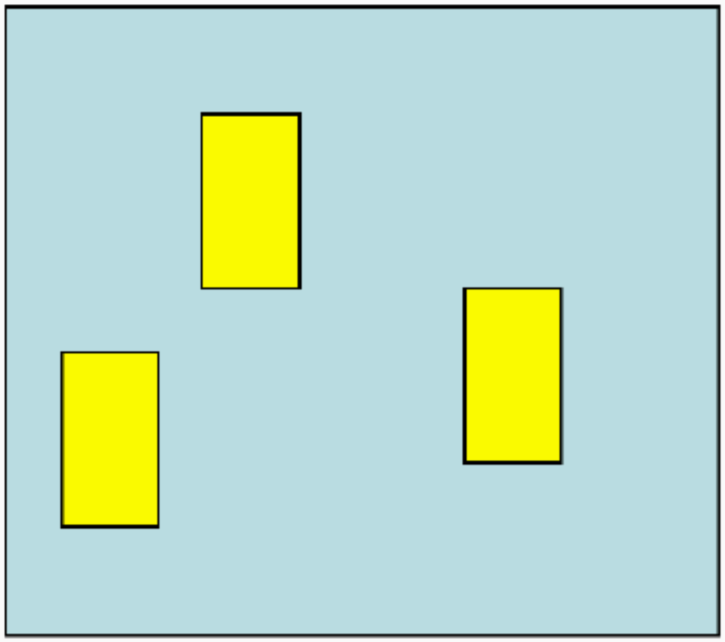
- The geometric configuration of a robot is defined by *p* degrees of freedom (DOF)
- Assuming *p* DOFs, the geometric configuration *A* of a robot is defined by *p* variables:

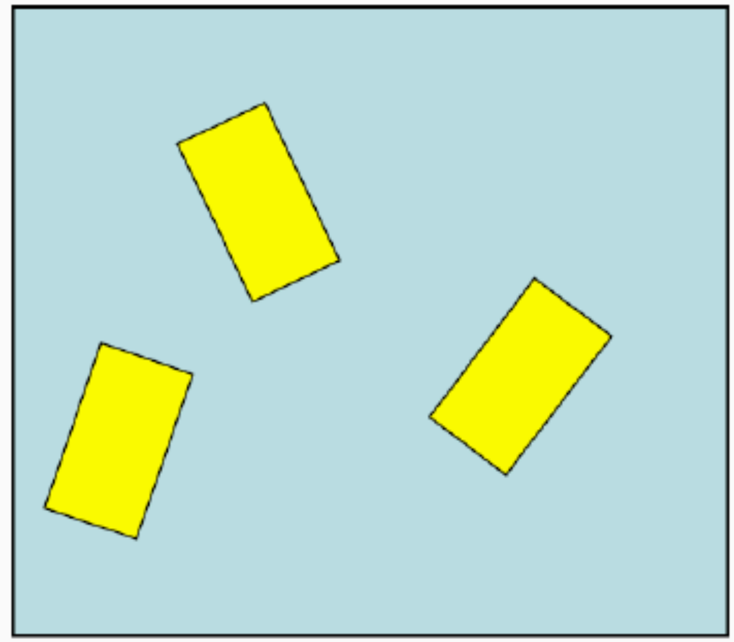
$A(\boldsymbol{q})$ with $\boldsymbol{q} = (q_1,\ldots,q_p)$

- Examples:
  - Prismatic (translational) DOF: q_i is the amount of translation in some direction
  - Rotational DOF: q_i is the amount of rotation about some axis

Our car has 3

Examples

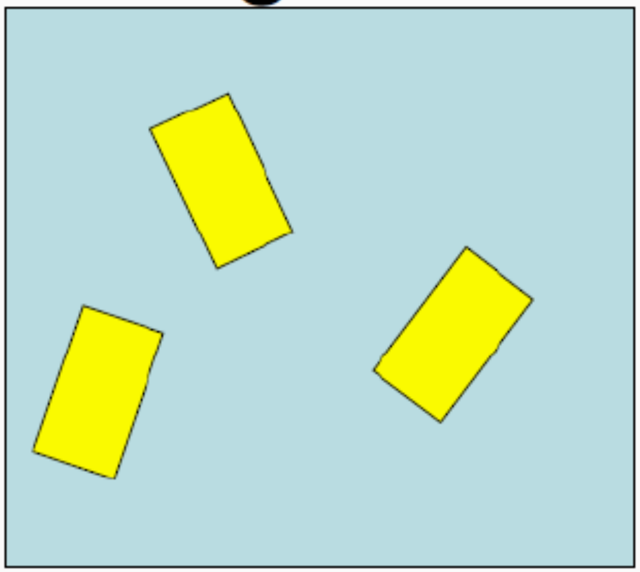




Allowed to move only in *x* and *y*: 2DOF

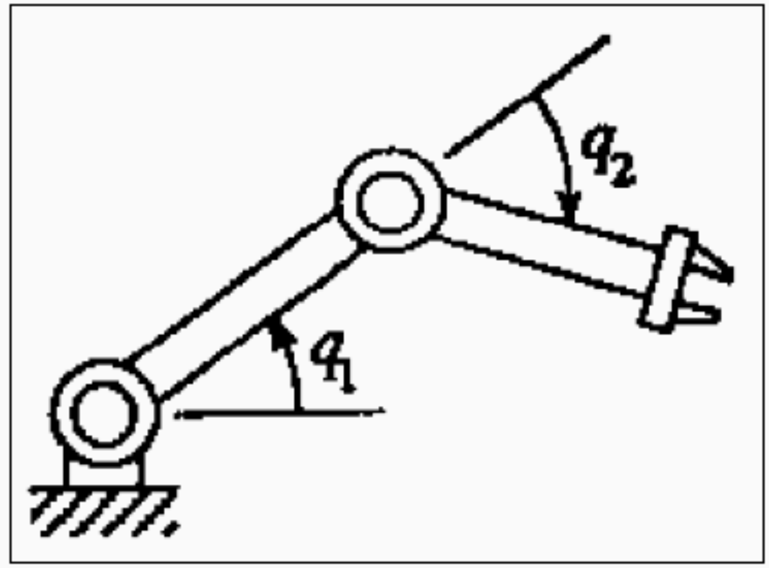
Allowed to move in *x* and y and to rotate: 3DOF (x,y,θ)

Configuration Space (C-Space)



$\boldsymbol{q} = (x, y, \theta)$

$\mathcal{C} = \Re^2$ x set of 2-D rotations



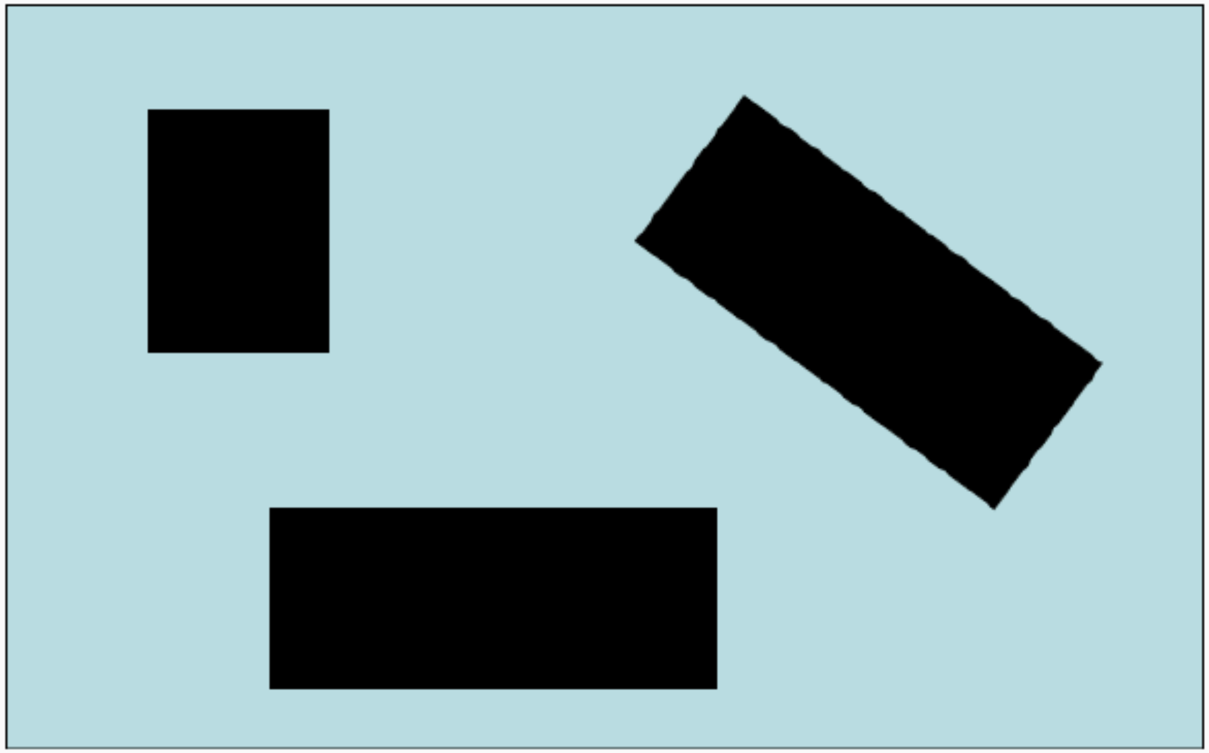
$\boldsymbol{q} = (q_1, q_2)$

$\mathcal{C} =$ 2-D rotations x 2-D rotations

- Configuration space $\mathcal{C}$ = set of values of $\boldsymbol{q}$ corresponding to legal configurations of the robot
- Defines the set of possible parameters (the search space) and the set of allowed paths

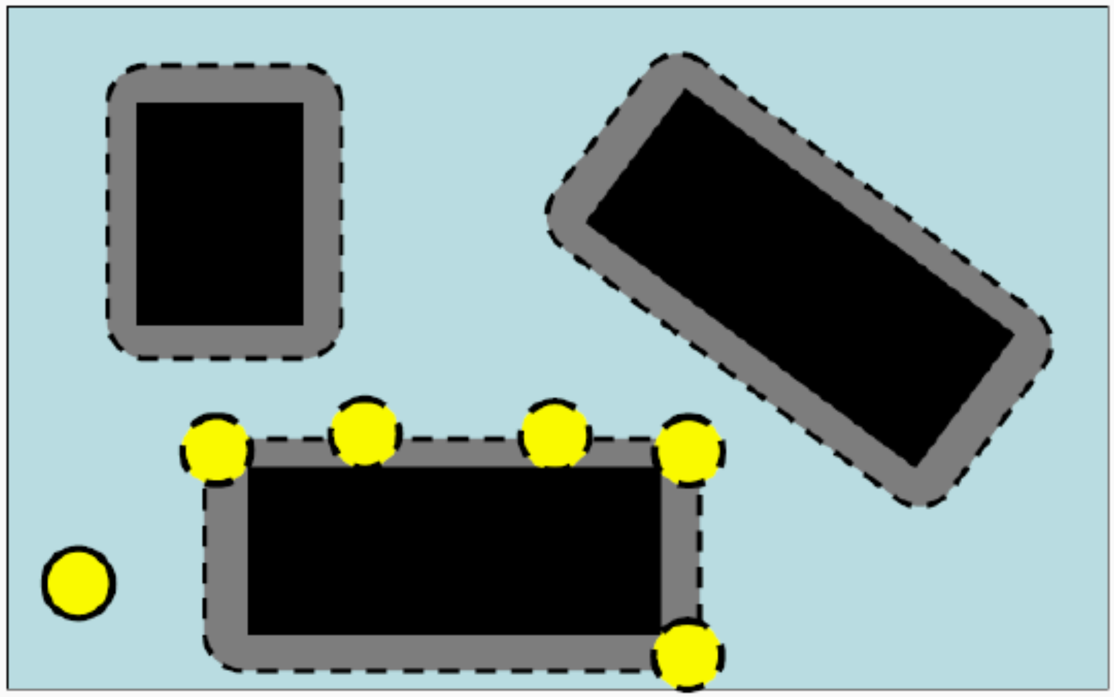
Choset slides

Free Space: Point Robot



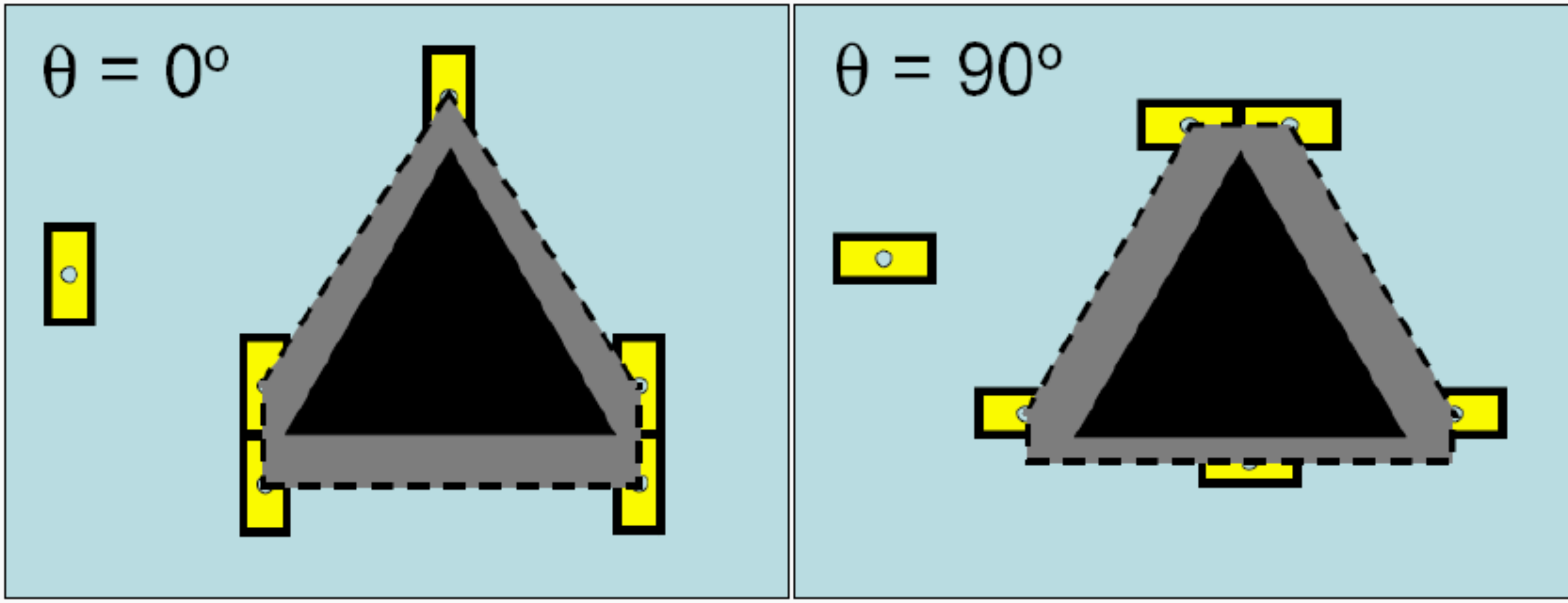
- $\mathcal{C}_{free}$ = {Set of parameters ***q*** for which *A*(***q***) does not intersect obstacles}
- For a point robot in the 2-D plane: R^2 minus the obstacle regions

Free Space: Symmetric Robot



- We still have $\mathcal{C} = R^2$ because orientation does not matter
- Reduce the problem to a point robot by expanding the obstacles by the radius of the robot

Free Space: Non-Symmetric Robot



- The configuration space is now three-dimensional (x, y, θ)
- We need to apply a different obstacle expansion for each value of θ
- We still reduce the problem to a point robot by expanding the obstacles

Any Formal Guarantees? Generic Piano Movers Problem

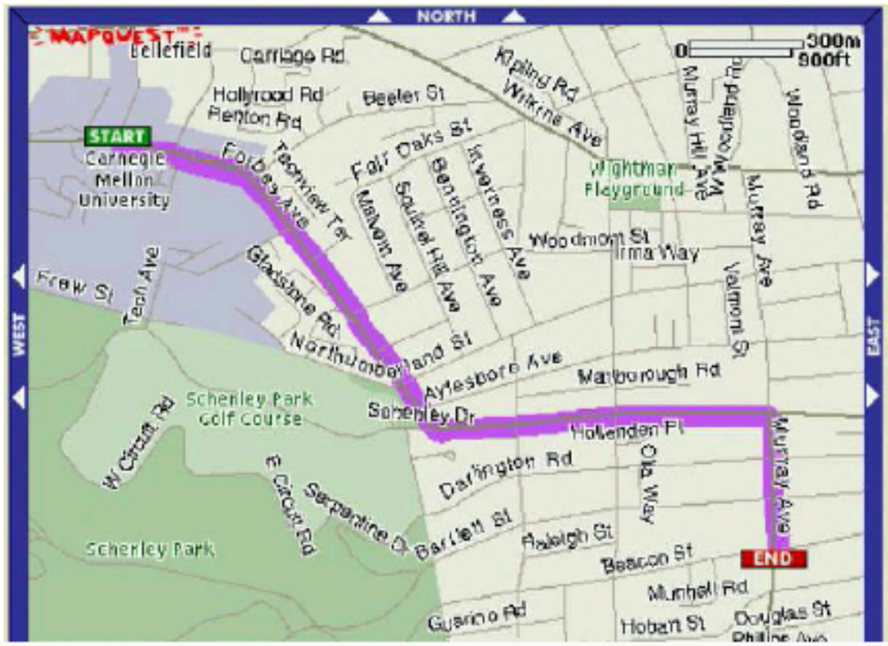
- Formal Result (but not terribly useful for practical algorithms):
  - *p*: Dimension of $\mathcal{C}$
  - *m*: Number of polynomials describing $\mathcal{C}_{free}$
  - *d*: Max degree of the polynomials
- A path (if it exists) can be found in time *exponential* in *p* and *polynomial* in *m* and *d*

[From J. Canny. "The Complexity of Robot Motion Planning Plans". MIT Ph.D. Dissertation. 1987]

Observation

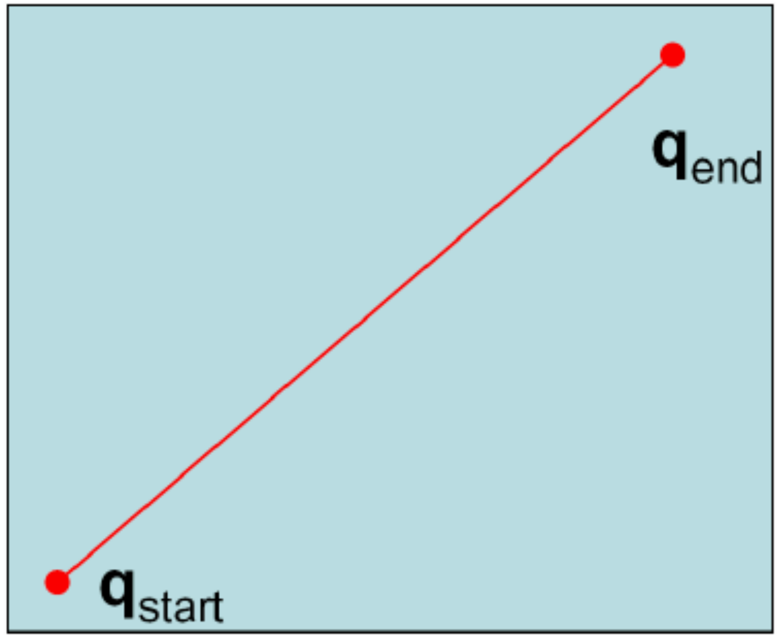
- Generally, searching a graph is pretty straightforward
    - Dijkstra, A*, etc - know how to do this
- Strategy
    - get a graph we can search

Roadmaps



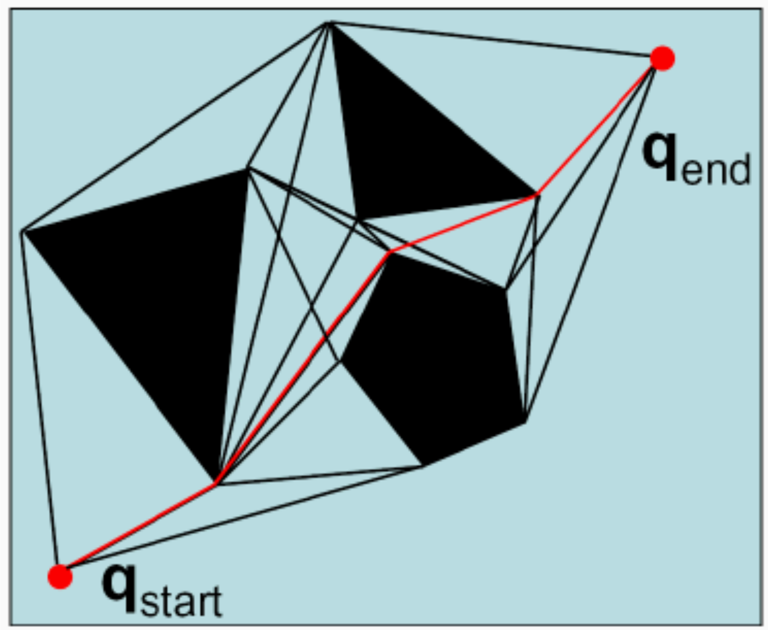
- General idea:
  - Avoid searching the entire space
  - Pre-compute a (hopefully small) graph (the roadmap) such that staying on the "roads" is guaranteed to avoid the obstacles
  - Find a path between $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ by using the roadmap

Visibility Graphs



In the absence of obstacles, the best path is the straight line between q_{start} and q_{goal}

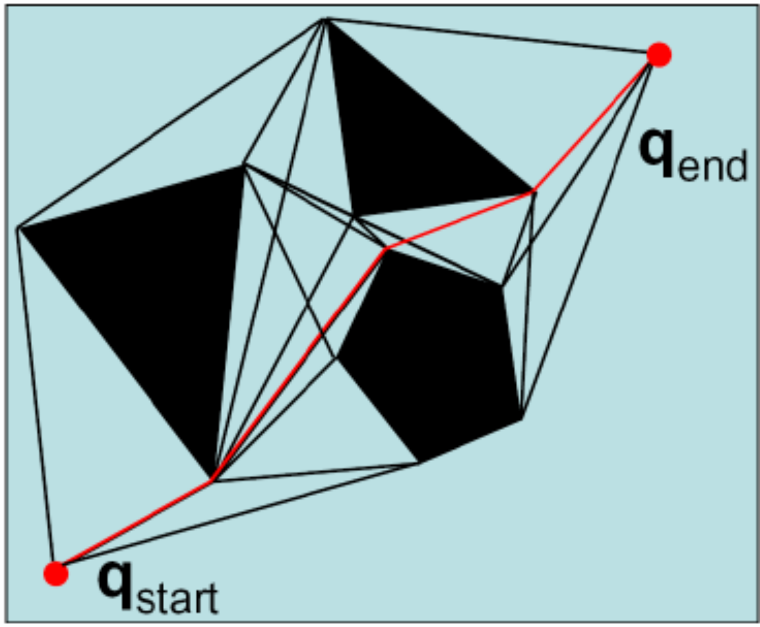
Visibility Graphs



- Visibility graph G = set of unblocked lines between vertices of the obstacles + $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$
- A node P is linked to a node P' if P' is visible from P
- Solution = Shortest path in the visibility graph

Issues

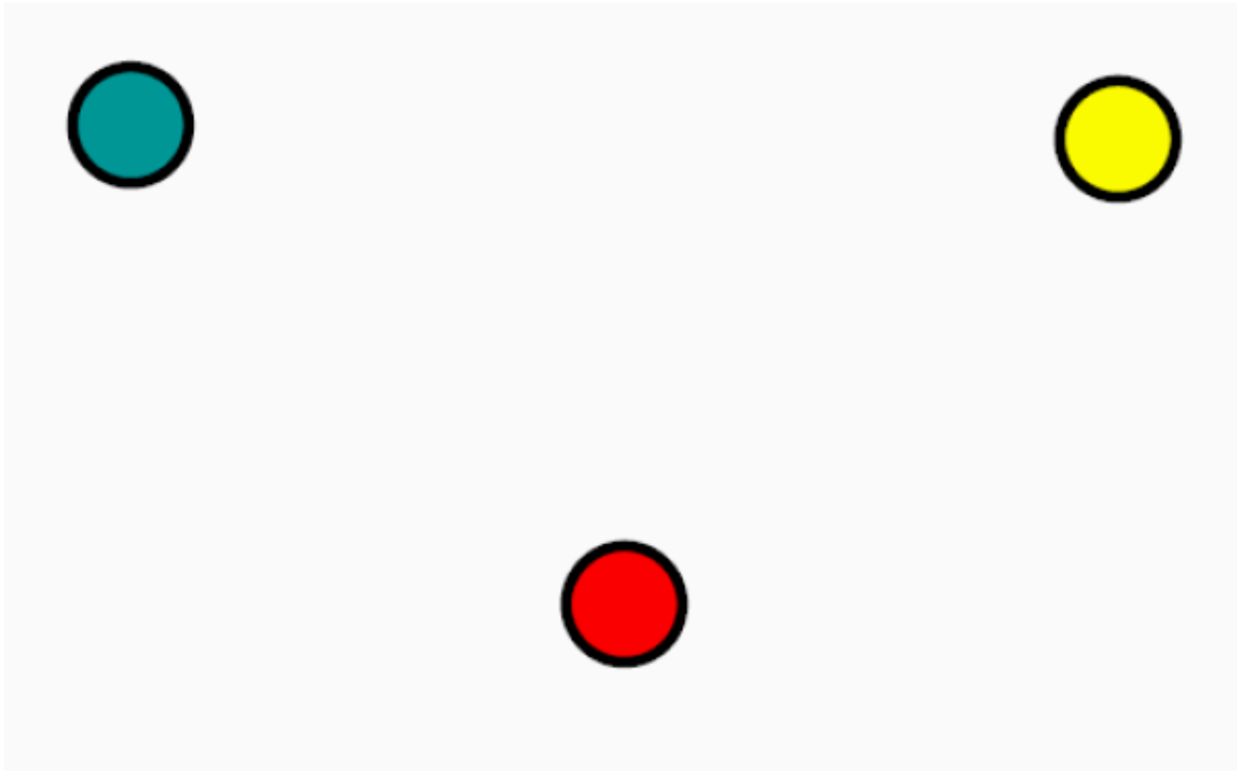
- Constructing
  - Relatively straightforward with a sweep algorithm
    - Variant (visibility complex) root cause of early computer games
      - Wolfenstein 3D, Doom II, etc
- What if configuration space is not 2D
  - You can still construct, MUCH harder
- MANY locally optimal paths
  - topology of free space clearly involved



Visibility Graphs: Weaknesses

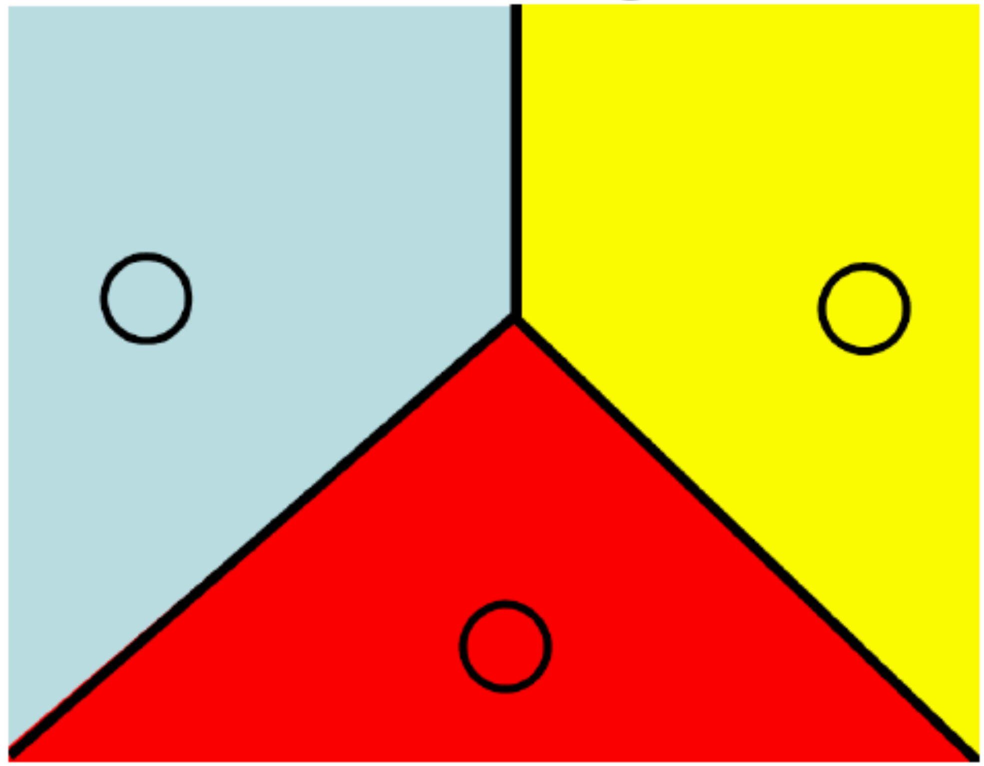
- Shortest path but:
  - Tries to stay as close as possible to obstacles
  - Any execution error will lead to a collision
  - Complicated in >> 2 dimensions
- We may not care about strict optimality so long as we find a safe path. Staying away from obstacles is more important than finding the shortest path
- Need to define other types of “roadmaps”

Voronoi Diagrams



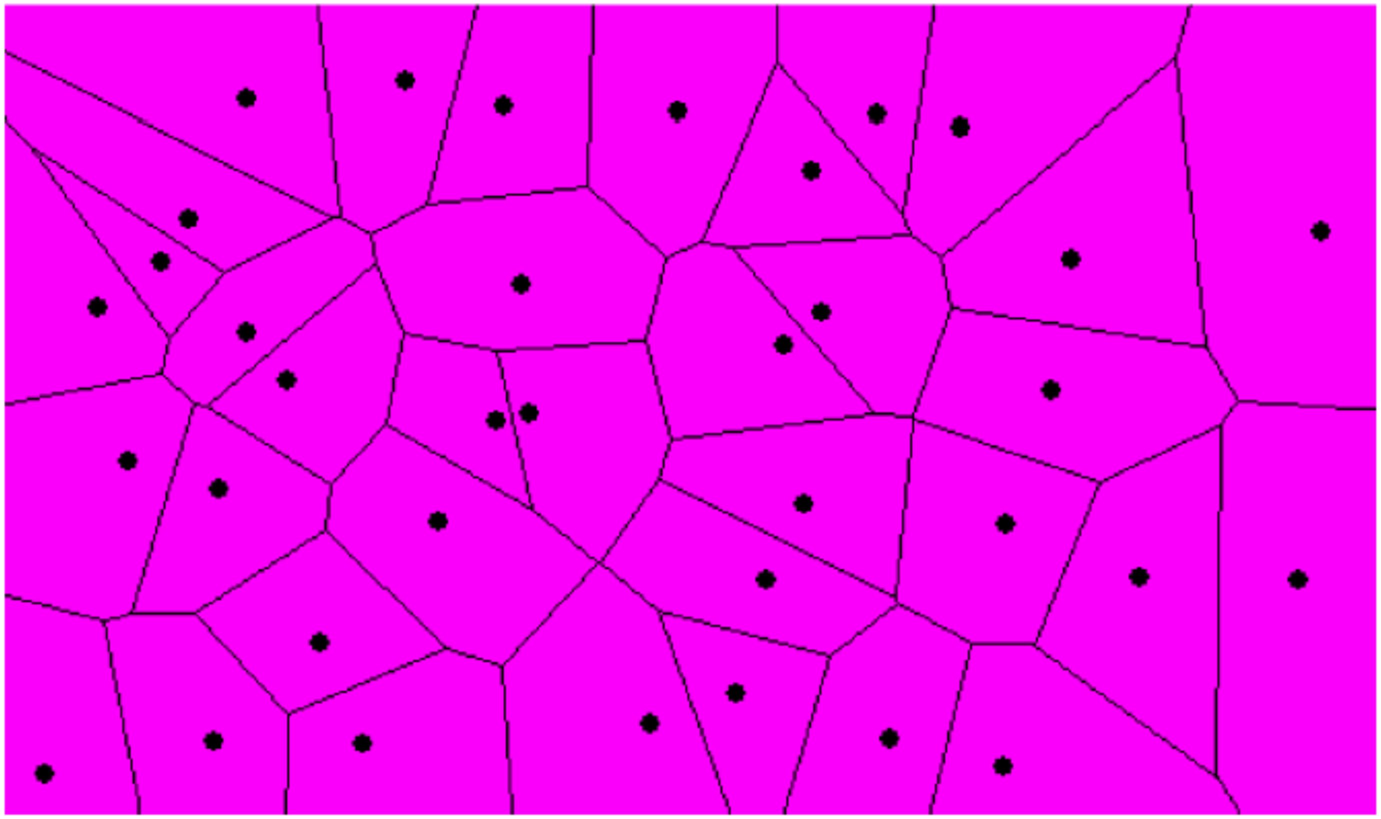
- Given a set of data points in the plane:
  - Color the entire plane such that the color of any point in the plane is the same as the color of its nearest neighbor

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

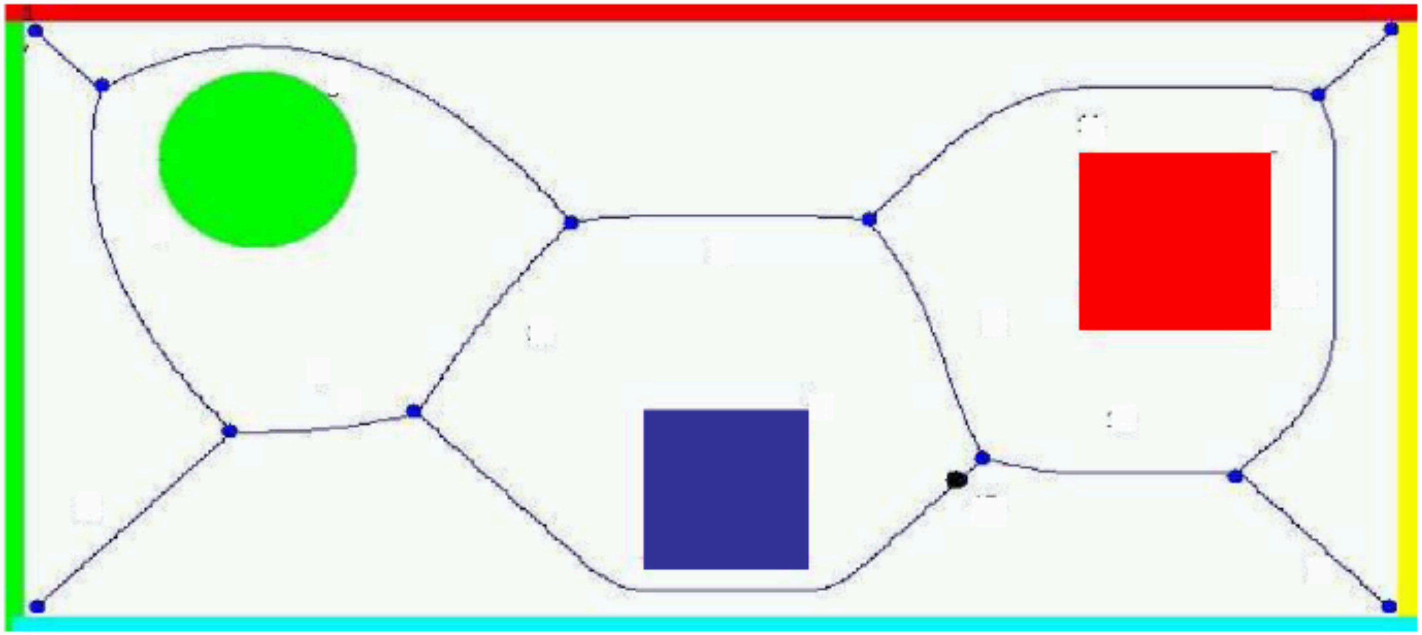
Voronoi Diagrams



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

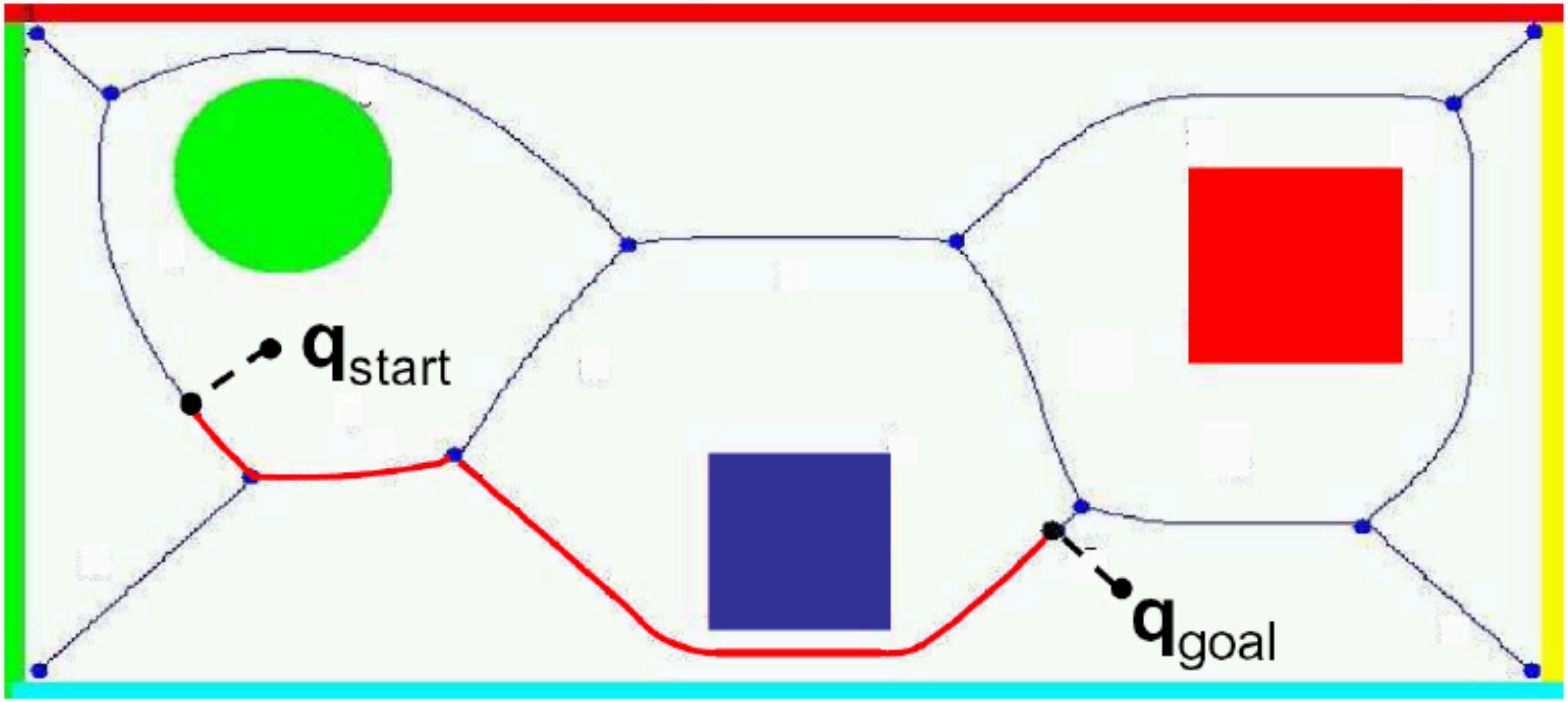
(See for example http://www.cs.cornell.edu/Info/People/chew/Delaunay.html for an interactive demo)

Voronoi Diagrams (Polygons)



- Key property: The points on the edges of the Voronoi diagram are the *furthest* from the obstacles
- Idea: Construct a path between $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$ by following edges on the Voronoi diagram
- (Use the Voronoi diagram as a roadmap graph instead of the visibility graph)

Voronoi Diagrams: Planning

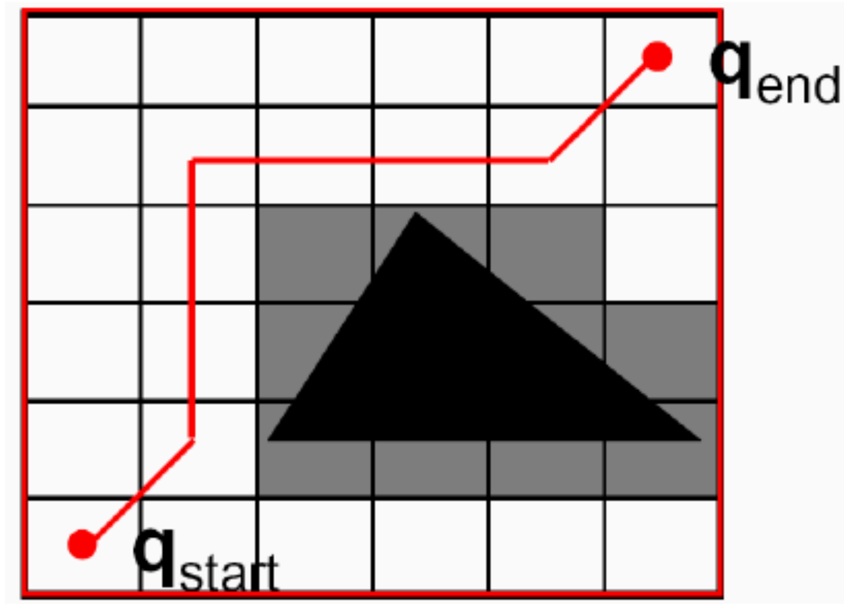


- Find the point $\mathbf{q}^*_{start}$ of the Voronoi diagram closest to $\mathbf{q}_{start}$
- Find the point $\mathbf{q}^*_{goal}$ of the Voronoi diagram closest to $\mathbf{q}_{goal}$
- Compute shortest path from $\mathbf{q}^*_{start}$ to $\mathbf{q}^*_{goal}$ on the Voronoi diagram

Voronoi: Weaknesses

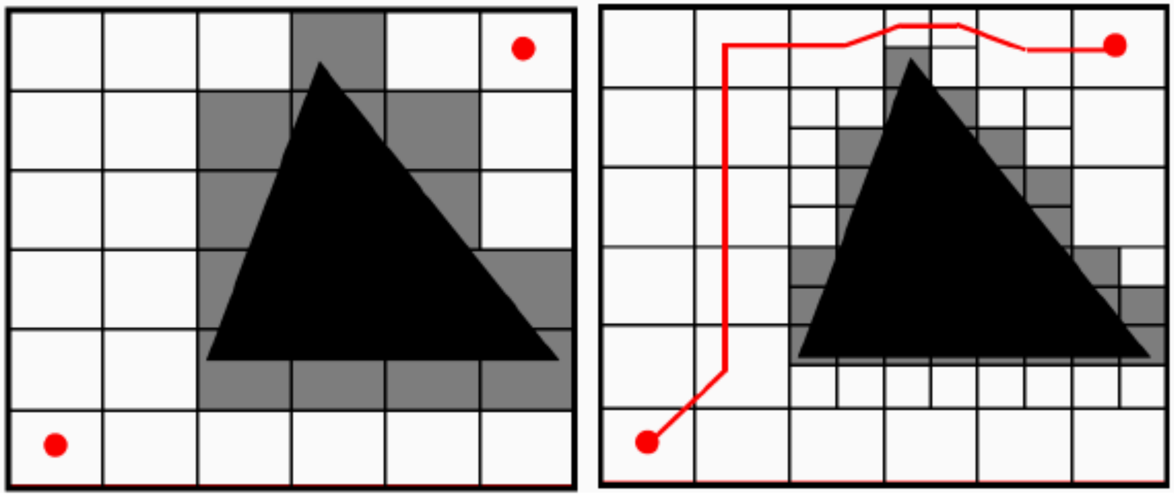
- Difficult to compute in higher dimensions or nonpolygonal worlds
- Approximate algorithms exist
- Use of Voronoi is not necessarily the best heuristic (“stay away from obstacles”) Can lead to paths that are much too conservative
- Can be unstable → Small changes in obstacle configuration can lead to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

Approximate Cell Decomposition



- Define a discrete grid in C-Space
- Mark any cell of the grid that intersects $\mathcal{C}_{obs}$ 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

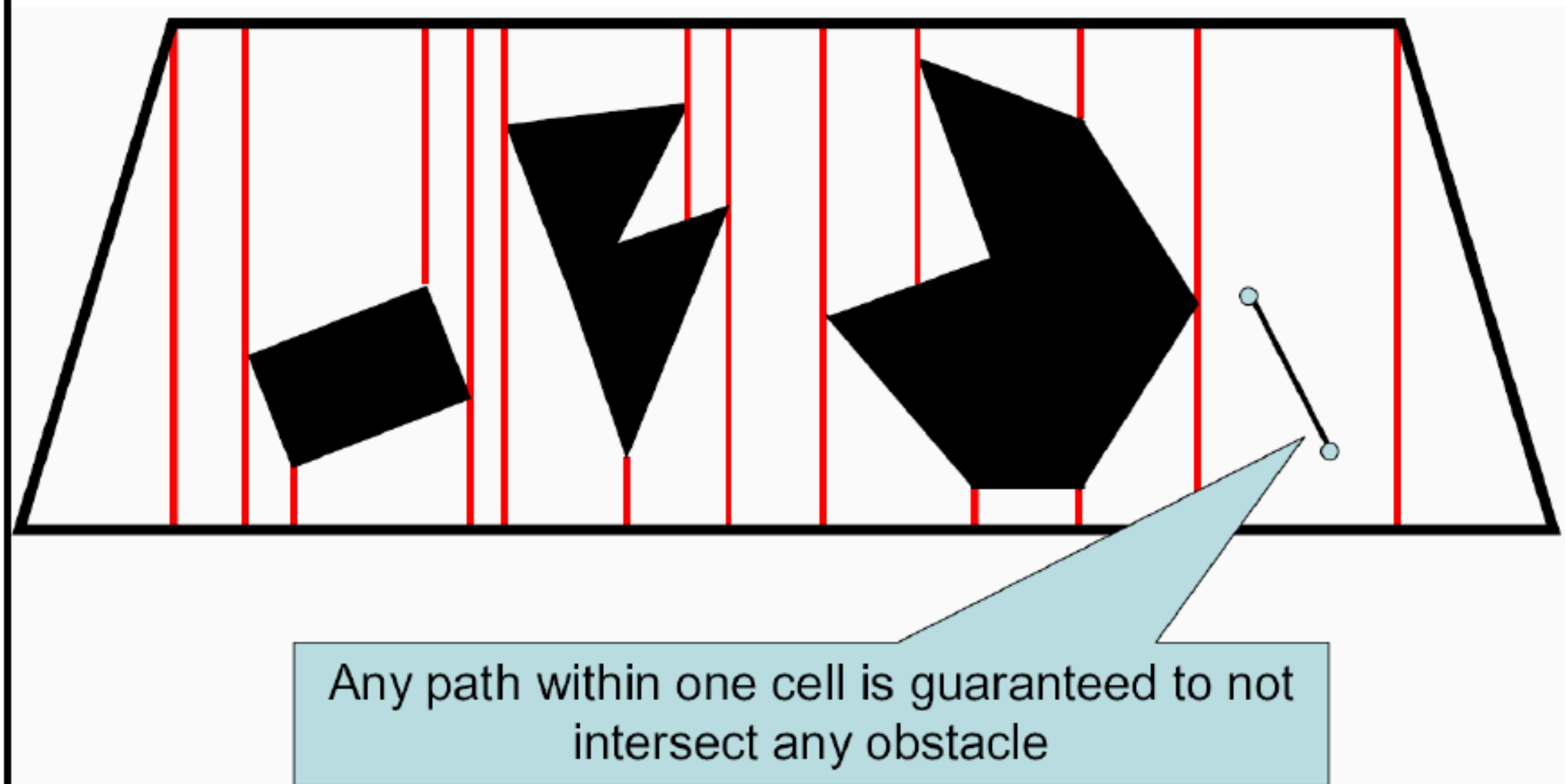


- Cannot find a path in this case even though one exists
- Solution:
- Distinguish between
  - Cells that are entirely contained in $\mathcal{C}_{obs}$ (*FULL*) and
  - Cells that partially intersect $\mathcal{C}_{obs}$ (*MIXED*)
- Try to find a path using the current set of cells
- If no path found: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

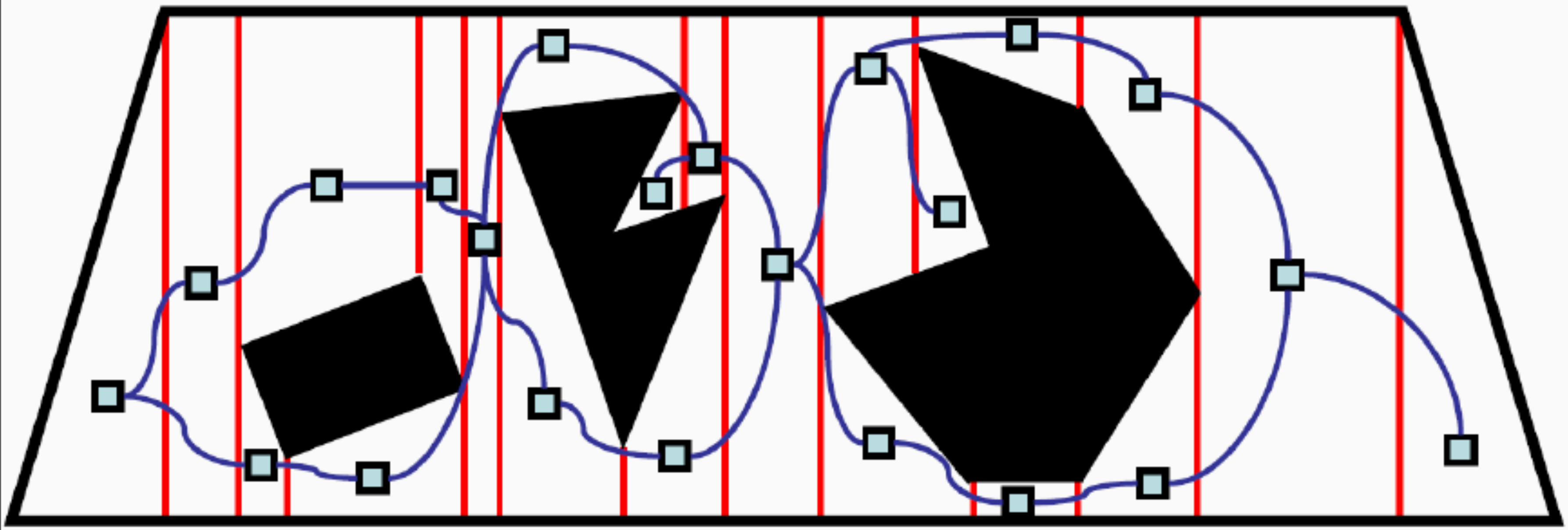
Approximate Cell Decomposition: Limitations

- 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

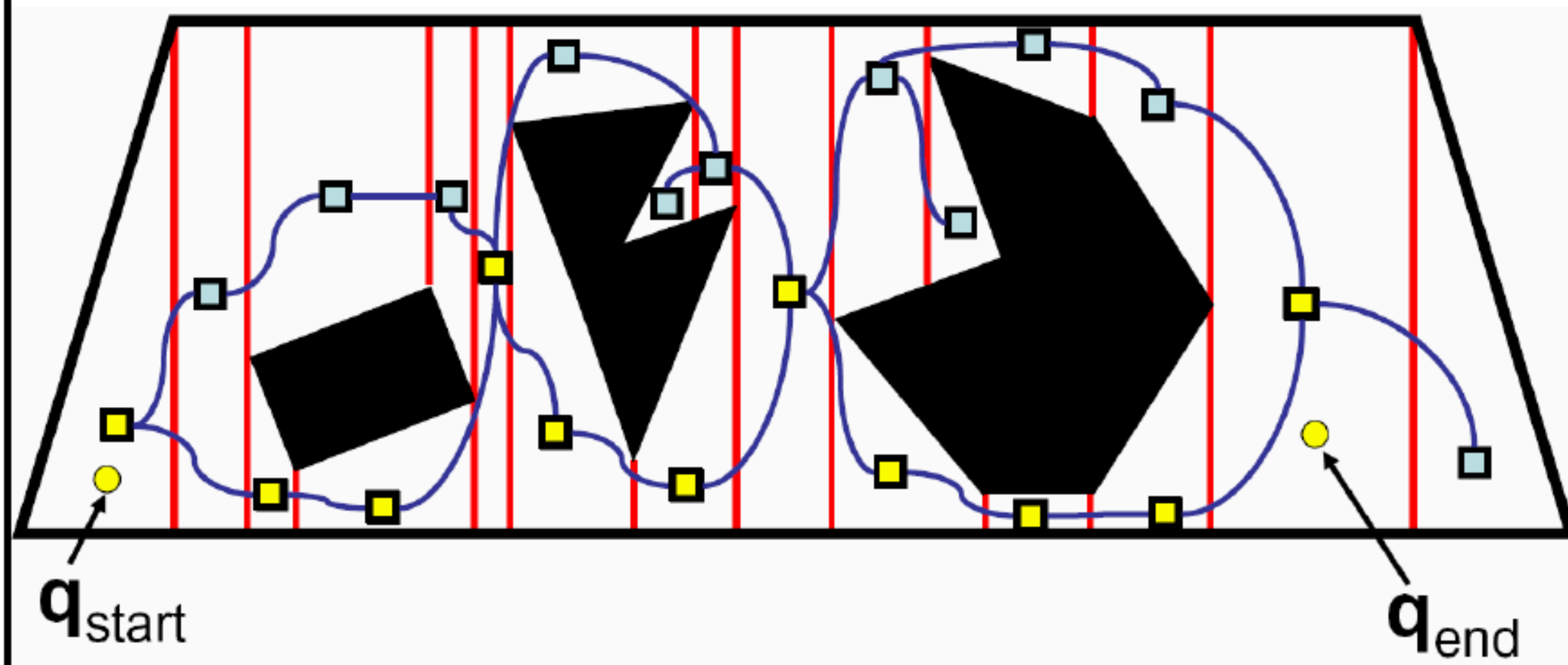


Exact Cell Decomposition



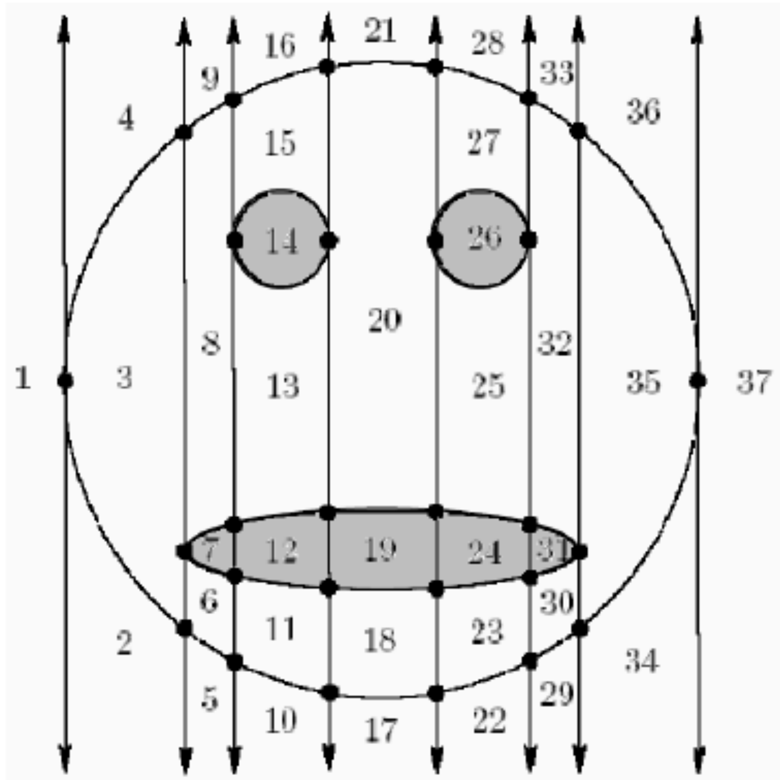
- The graph of cells defines a roadmap

Exact Cell Decomposition



- The graph can be used to find a path between any two configurations

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