Motion Planning

Jyh-Ming Lien

Department of Computer Science
George Mason University

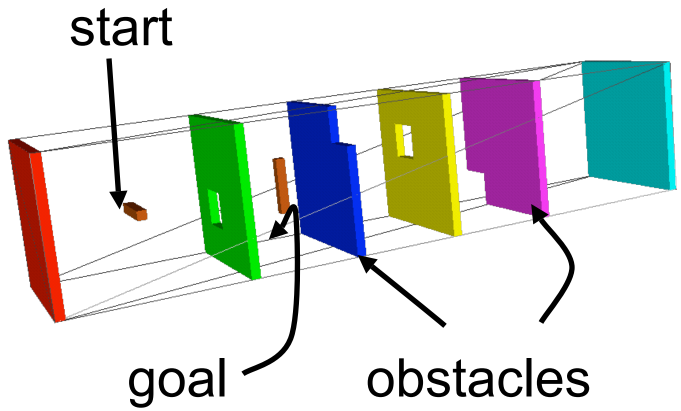
Based on many people's lecture notes

Seth Hutchinson at the University of Illinois at Urbana-Champaign, Leo Joskowicz at Hebrew University, Jean-Claude Latombe at Stanford University, Nancy Amato at Texas A&M University, Burchan Bayazit at Washington University in St. Louis

Motion Planning
in continuous spaces

(Basic) Motion Planning
(in a nutshell):

Given a *movable object*, find a *sequence of valid configurations* that moves the object from the start to the goal.



Main Steps In Motion Planning

Workspace

↓

Configuration space

↓

Discretization

↓

Search

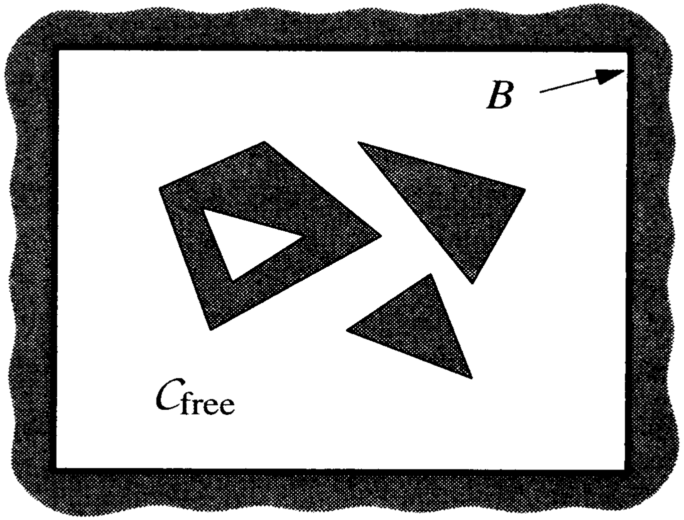
↓

Path or no solution

Classical Motion Planning

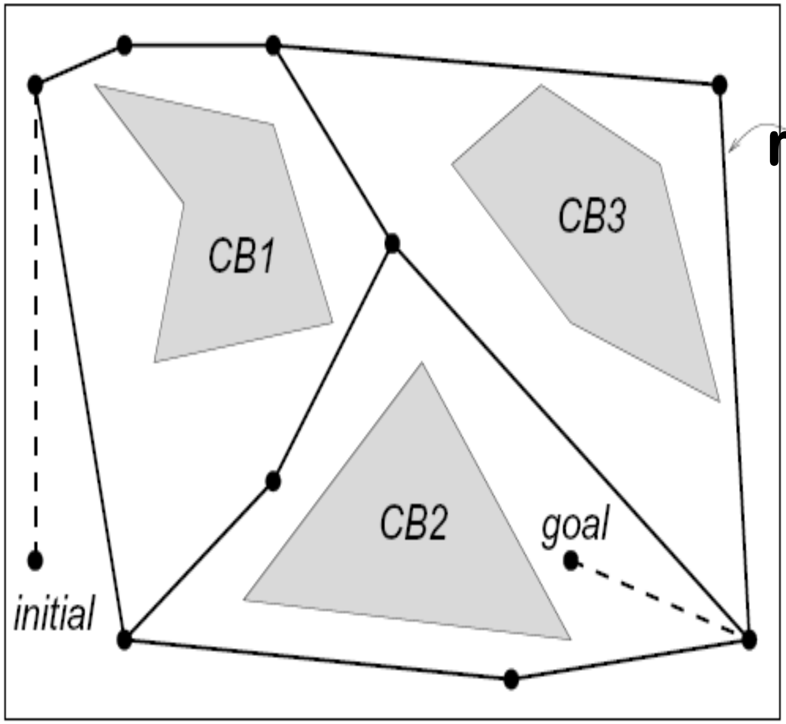
- Given a point robot and a workspace described by polygons

- **Roadmap methods**
  - Visibility graph
  - Cell decomposition
  - Retraction

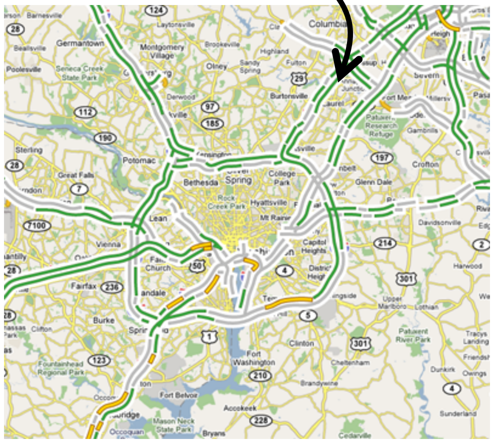


Roadmap Methods

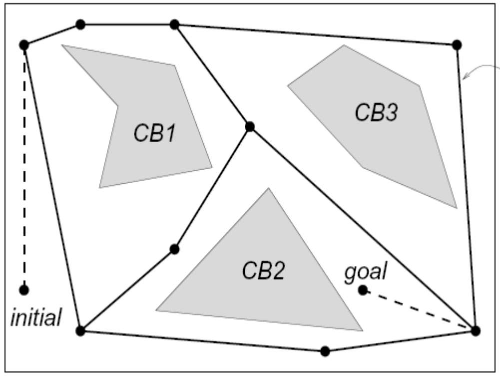
Capture the connectivity of C_{free} with a roadmap (graph or network) of one-dimensional curves



roadmap



Roadmap Methods



Path Planning with a Roadmap

Input: configurations q_{init} and q_{goal} , and B

Output: a path in C_{free} connecting q_{init} and q_{goal}

1. Build a roadmap in C_{free} (preprocessing)

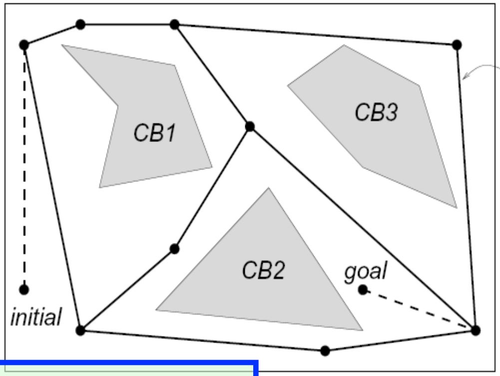
- **roadmap nodes are free configurations (or semi-free)**
- **two nodes connected by edge if can (easily) move between them**

2. Connect q_{init} and q_{goal} to roadmap nodes v_{init} and v_{goal}

3. Find a path in the roadmap between v_{init} and v_{goal}

- directly gives a path in C_{free}

Roadmap Methods



Path Planning with a Roadmap

Input: configurations q_{init} and q_{goal} , and B

Output: a path in C_{free} connecting q_{init} and q_{goal}

1. Build a roadmap in C_{free} (preprocessing)
   - roadmap nodes are free configurations (or semi-free)
   - two nodes connected by edge if can (easily) move between them

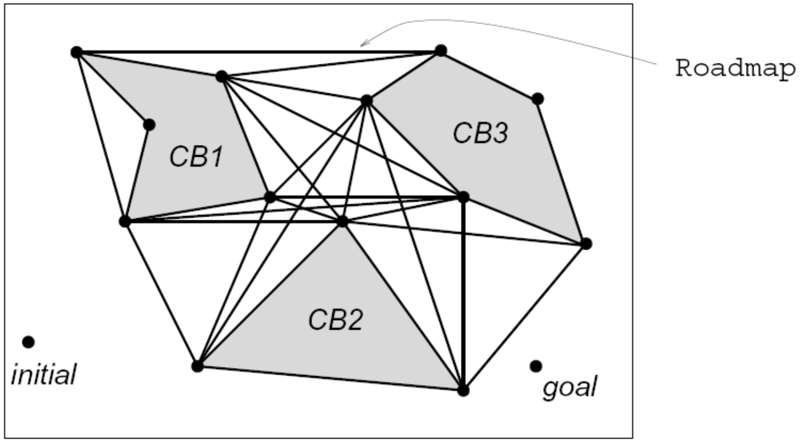
difficult part

2. Connect q_{init} and q_{goal} to roadmap nodes v_{init} and v_{goal}

3. Find a path in the roadmap between v_{init} and v_{goal}
   - directly gives a path in C_{free}

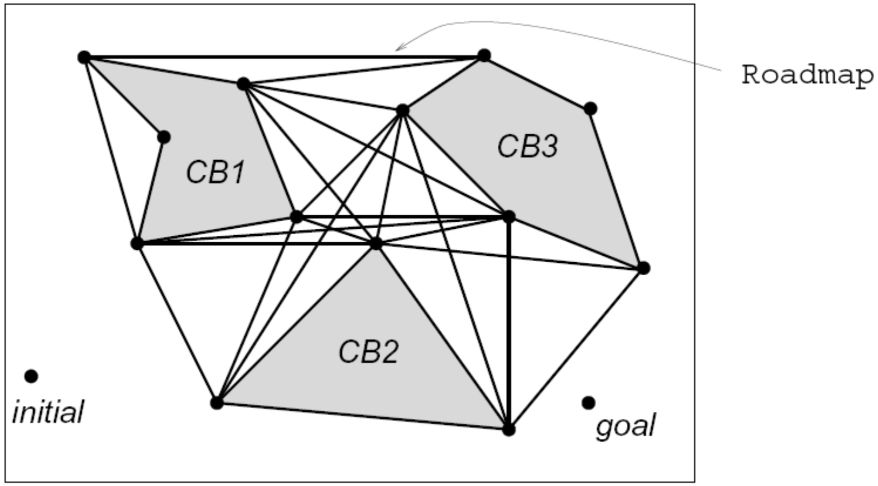
Visibility Graph

- A visibility graph of C-space for a given C-obstacle is an undirected graph G where
  - nodes in G correspond to vertices of C-obstacle
  - nodes connected by edge in G if
    - they are connected by an edge in C-obstacle, or
    - the straight line segment connecting them lies entirely in *Cfree*
  - (could add q_{init} and q_{goal} as roadmap nodes)



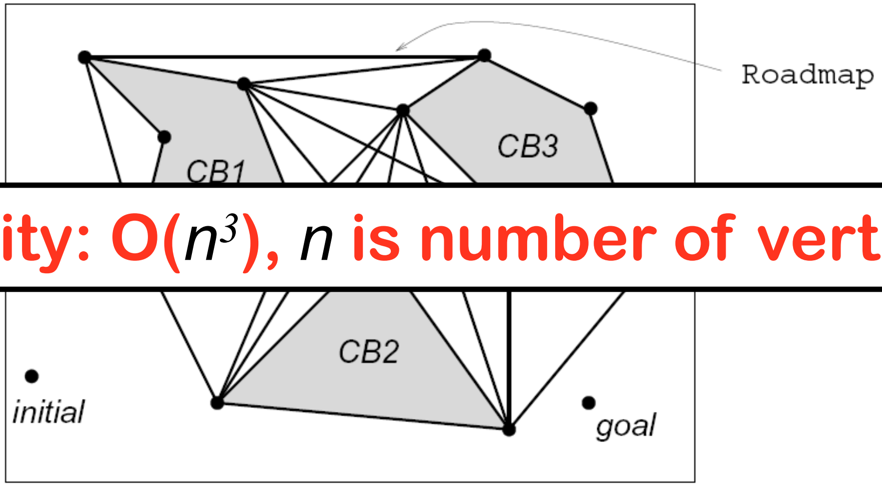
Visibility Graph

- Brute Force Algorithm
  - add all edges in C-obstacle to G
  - for each pair of vertices (x, y) of C-obstacle, add the edge (x, y) to G if the straight line segment connecting them lies entirely in cl(C-free)
    - test (x; y) for intersection with all O(n) edges of C-obstacle
    - O(n^2) pairs to test, each test takes O(n) time



Visibility Graph

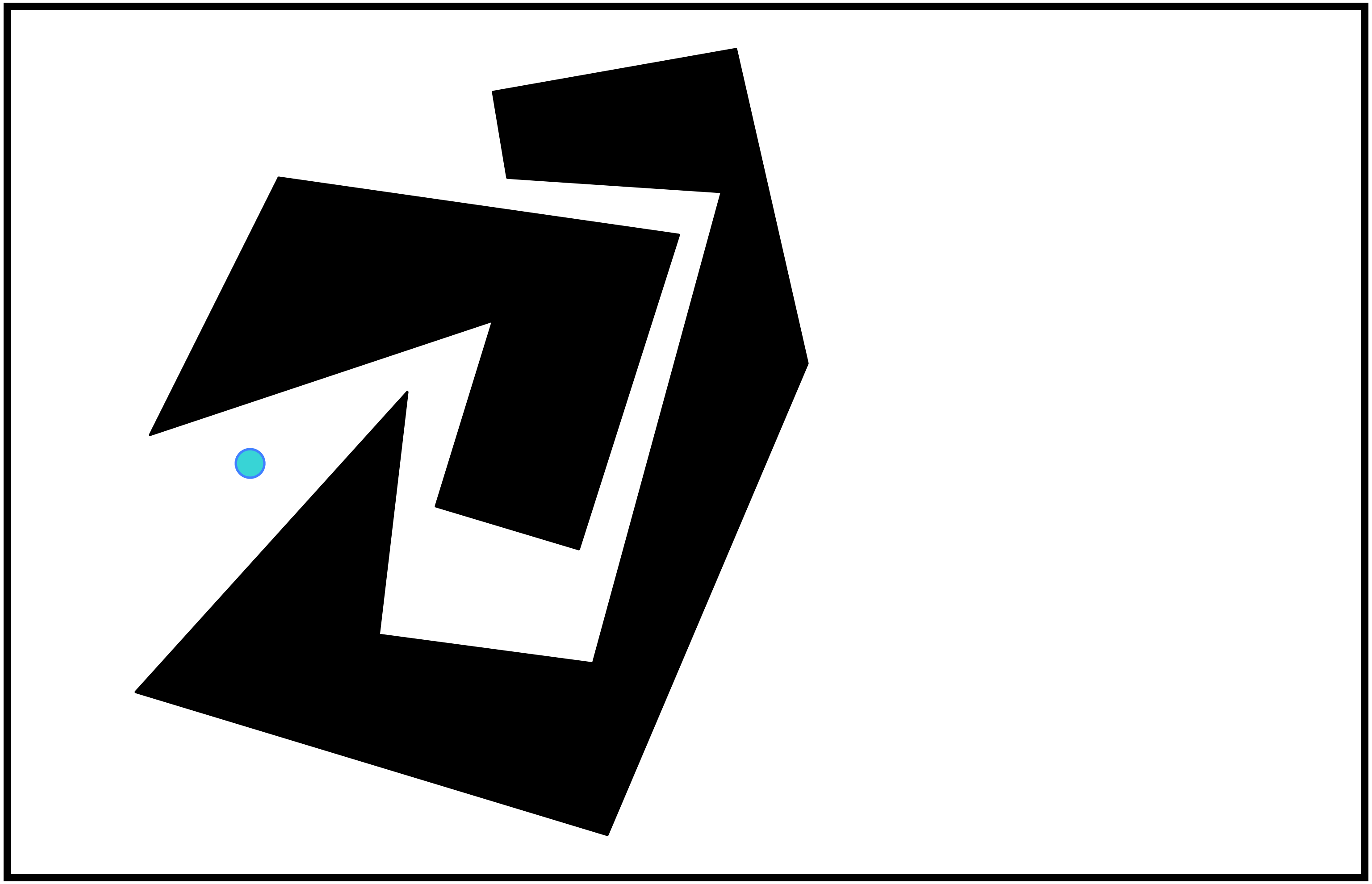
- Brute Force Algorithm
  - add all edges in C-obstacle to G
  - for each pair of vertices (x, y) of C-obstacle, add the edge (x, y) to G if the straight line segment connecting them lies entirely in cl(C-free)
    - test (x; y) for intersection with all O(n) edges of C-obstacle
    - O(n^2) pairs to test, each test takes O(n) time



Complexity: O(n^3), n is number of vertices in C-obstacle

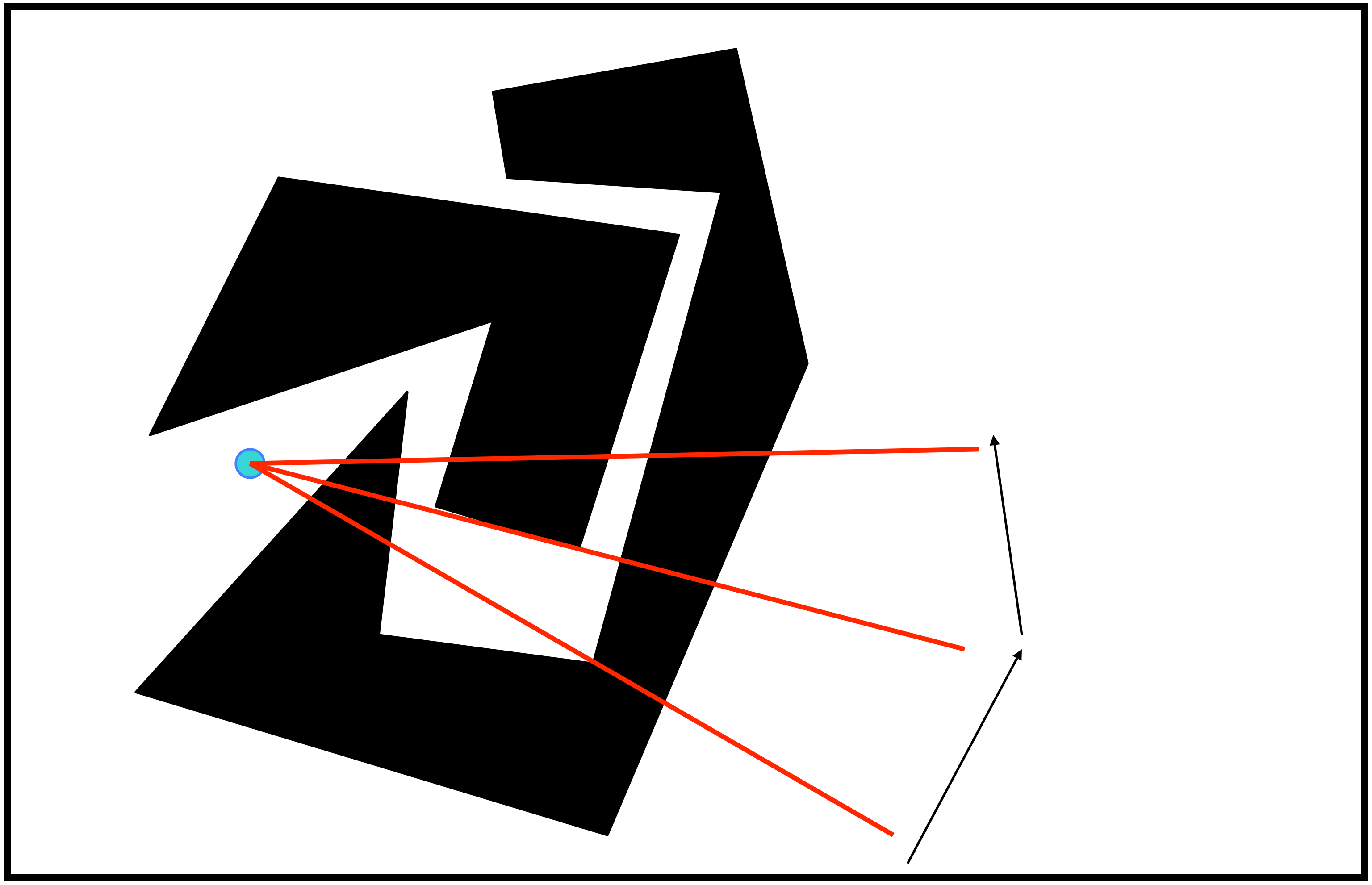
Visibility Graph

- A better algorithm?



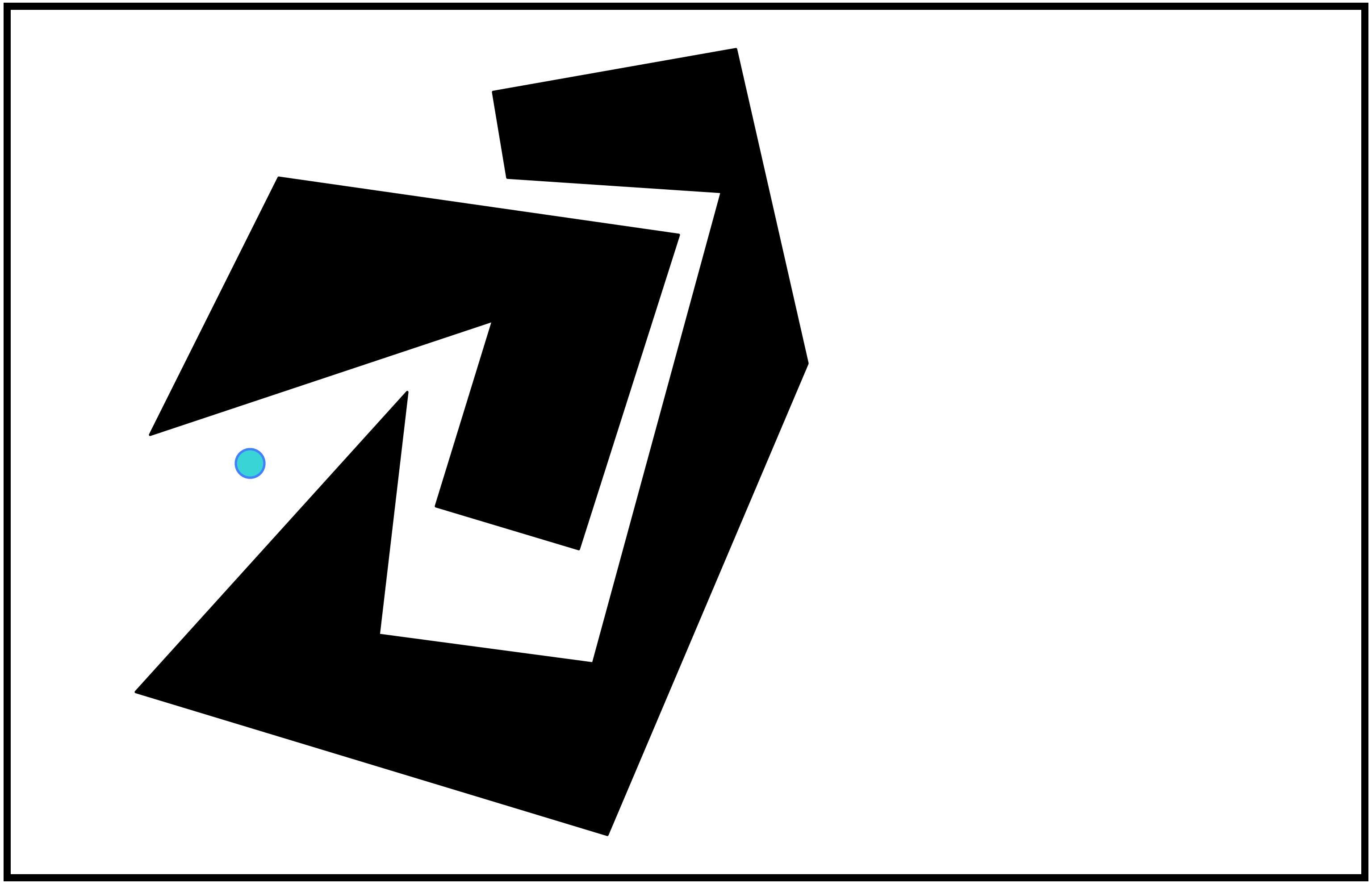
Visibility Graph

- A better algorithm?



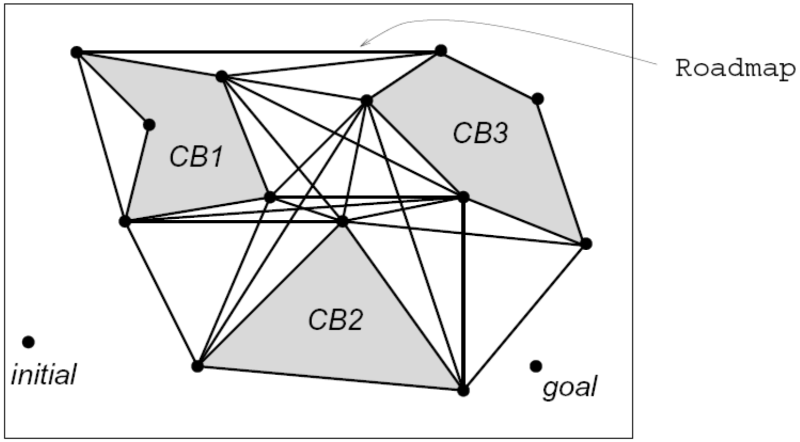
Visibility Graph

- An even better algorithm?



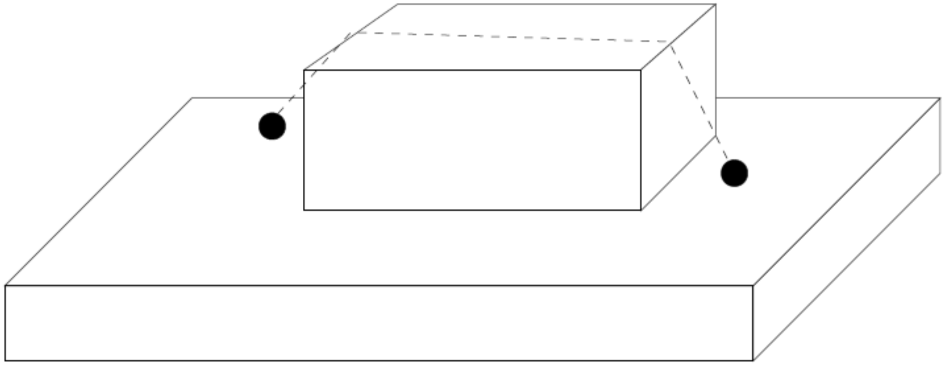
Visibility Graph

- **Visibility graphs (Good news)**
  - are conceptually simple
  - shortest paths (if the query cannot see each other)
  - we have efficient algorithms if WS is polygonal
    - *$O(n^2)$*, where n is number of vertices of C-obstacle
    - *$O(k + n \log n)$*, where k is number of edges in G
  - we can make a 'reduced' visibility graph (don't need all edges)



Visibility Graph in 3-D

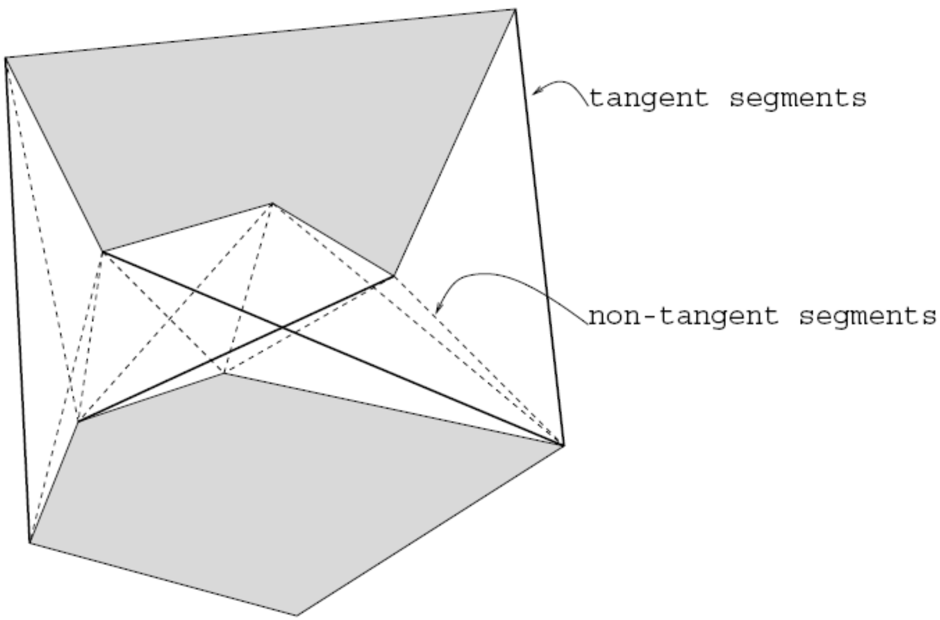
- Visibility graphs don't necessarily contain shortest paths in R^3
  - in fact finding shortest paths in R^3 is NP-hard [Canny 1988]
  - $(1 + \varepsilon^2)$ approximation algorithm [Papadimitriou 1985]



Bad news: really only suitable for two-dimensional C

Reduced Visibility Graph

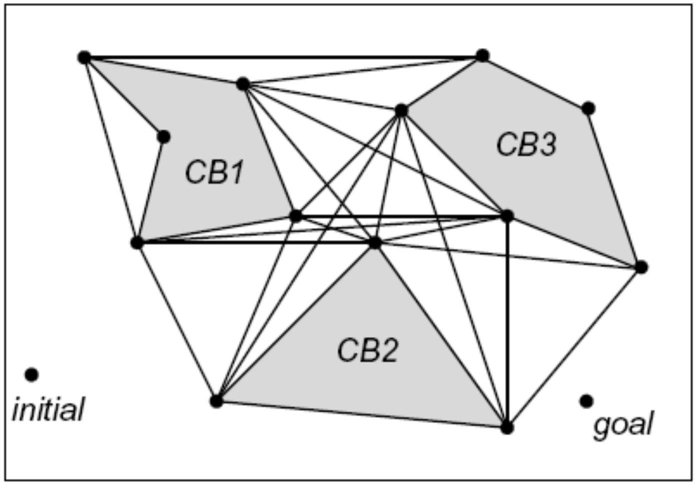
- we don't really need all the edges in the visibility graph (even if we want shortest paths)
- Definition: Let L be the line passing through an edge (x; y) in the visibility graph G. The segment (x; y) is a tangent segment *iff* L is tangent to C-obstacle at both x and y.



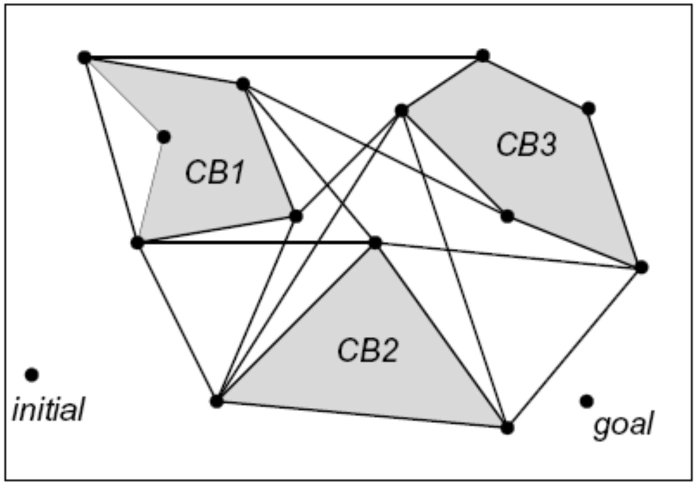
Reduced Visibility Graph

- It turns out we need only keep
  - convex vertices of C-obstacle
  - non-CB edges that are tangent segments

Visibility Graph

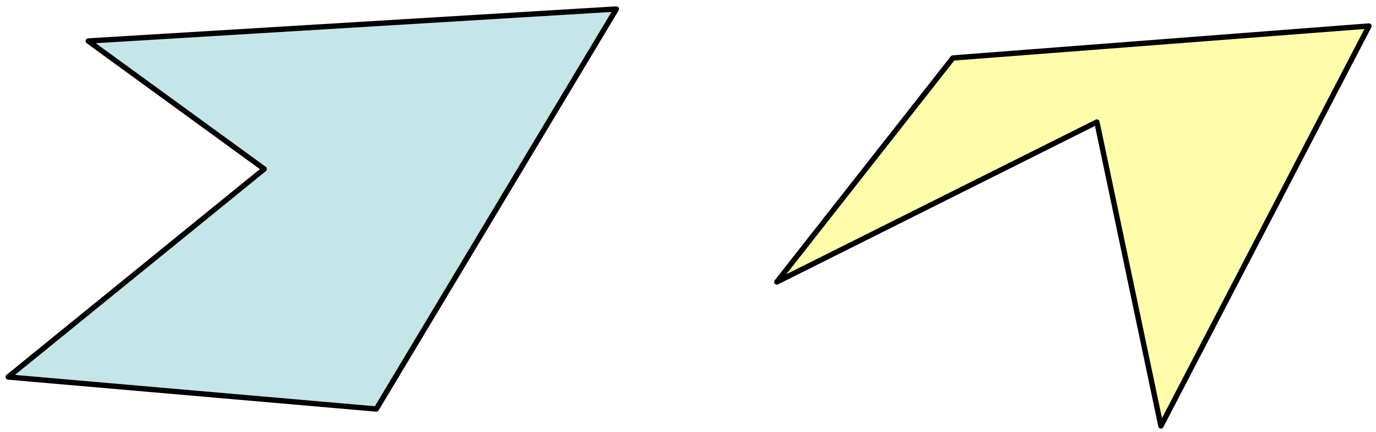


Reduced Visibility Graph



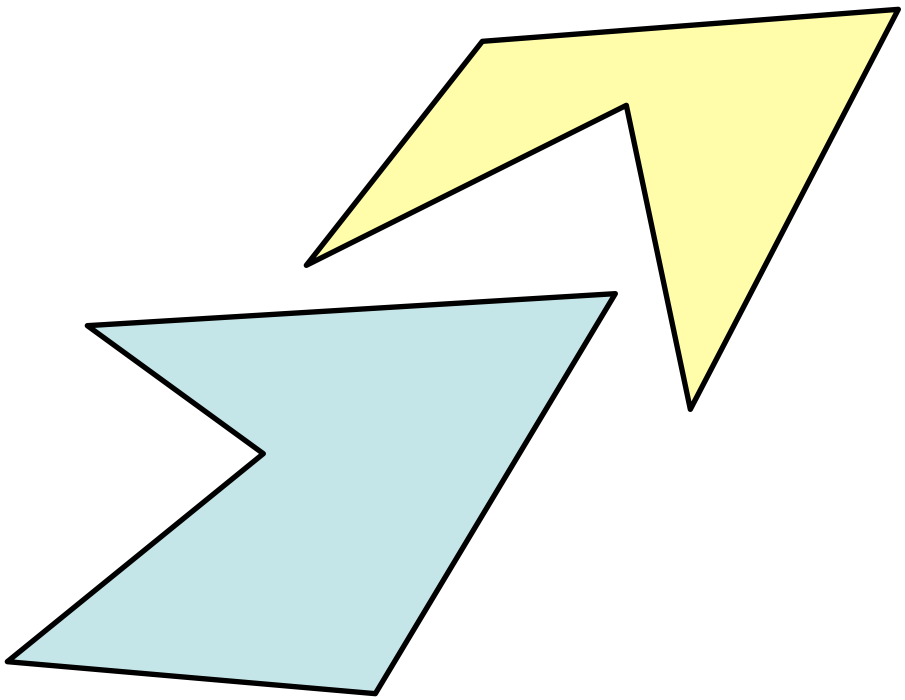
Reduced Visibility Graph

- Reduced visibility graphs are easier to build
  - construct convex hull of each C-obstacle piece eliminate non-convex vertices
  - construct pairwise tangents between each convex C-obstacle piece
- easy to construct tangents between two convex polygons
  - How?



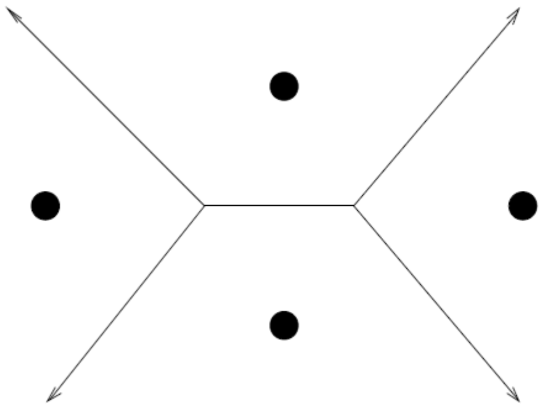
Reduced Visibility Graph

- Reduced visibility graph does not work if the convex hulls of two obstacles intersect



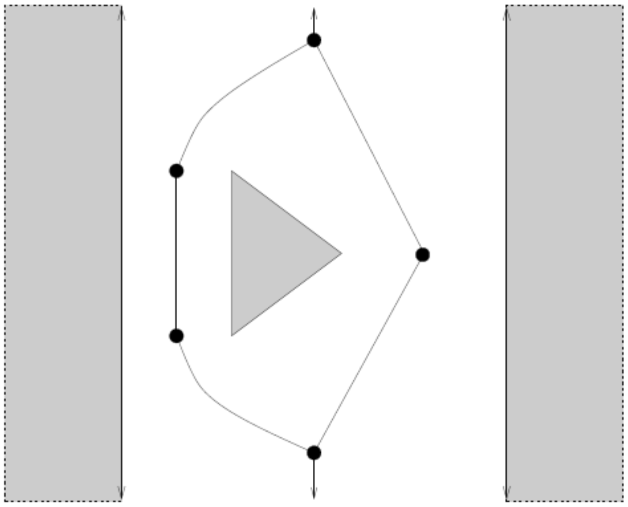
Voronoi Diagram for Point Sets

- 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



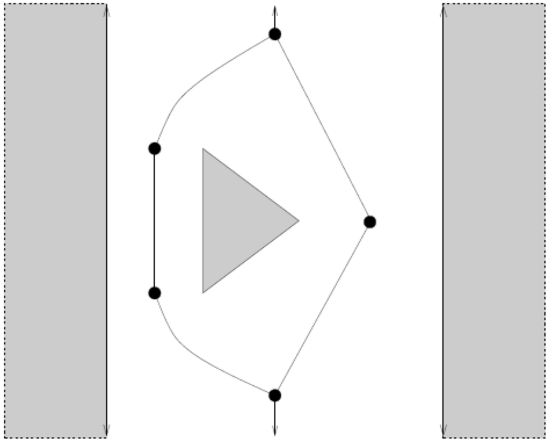
Voronoi Diagram for Point Sets

- When C = R^2 and polygonal C-obstacle, Vor(Cfree) 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 C-obstacle, i.e., the set of points equally close to two points (or two line segments) is a line



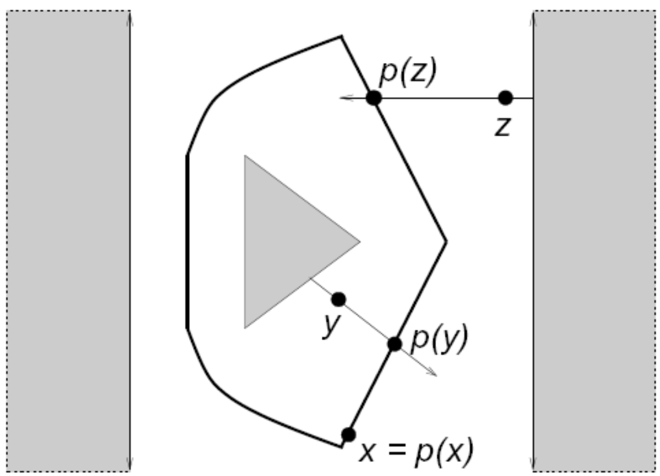
Voronoi Diagram for Point Sets

- Naive Method of Constucting V or(Cfree)
  - compute all arcs (for each vertex-vertex, edge-edge, and vertex-edge pair)
  - compute all intersection points (dividing arcs into segments)
  - keep segments which are closest only to the vertices/edges that



Retraction

- Retraction $\rho : C_{free} \rightarrow \mathrm{Vor}(C_{free})$



To find a path:

1. compute Vor(C_{free})

2. find paths from q_{init} and q_{goal} to $\rho(q_{init})$ and $\rho(q_{goal})$, respectively

3. search Vor(C_{free}) for a set of arcs connecting $\rho(q_{init})$ and $\rho(q_{goal})$

Cell Decomposition

- Idea: decompose C_{free} into a collection K of non-overlapping cells such that the union of all the cells exactly equals the free C-space

- Cell Characteristics:
  - geometry of cells should be simple so that it is easy to compute a path between any two configurations in a cell
  - it should be pretty easy to test the adjacency of two cells, i.e., whether they share a boundary
  - it should be pretty easy to find a path crossing the portion of the boundary shared by two adjacent cells

- Thus, cell boundaries correspond to 'criticalities' in C, i.e., something changes when a cell boundary is crossed. No such criticalities in C occur within a cell.

Cell Decomposition

Cell Decomposition

Difficult

Cell Decomposition

- **Preprocessing:**
  - represent C_{free} as a collection of cells (connected regions of C_{free})
    - planning between configurations in the same cell should be 'easy'
  - build connectivity graph representing adjacency relations between cells
    - cells adjacent if can move directly between them
- **Query:**
  - locate cells k_{init} and k_{goal} containing start and goal configurations
  - search the connectivity graph for a 'channel' or sequence of adjacent cells connecting k_{init} and k_{goal}
  - find a path that is contained in the channel of cells

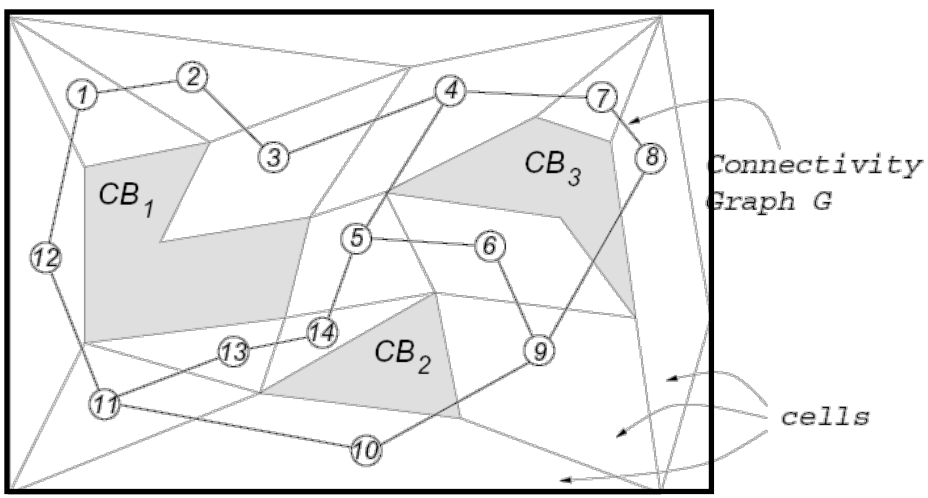
Difficult

- Two major variants of methods:
  - exact cell decomposition:
    - set of cells exactly covers C_{free}
    - complicated cells with irregular boundaries (contact constraints)
    - harder to compute
  - approximate cell decomposition:
    - set of cells approximately covers C_{free}
    - simpler cells with more regular boundaries

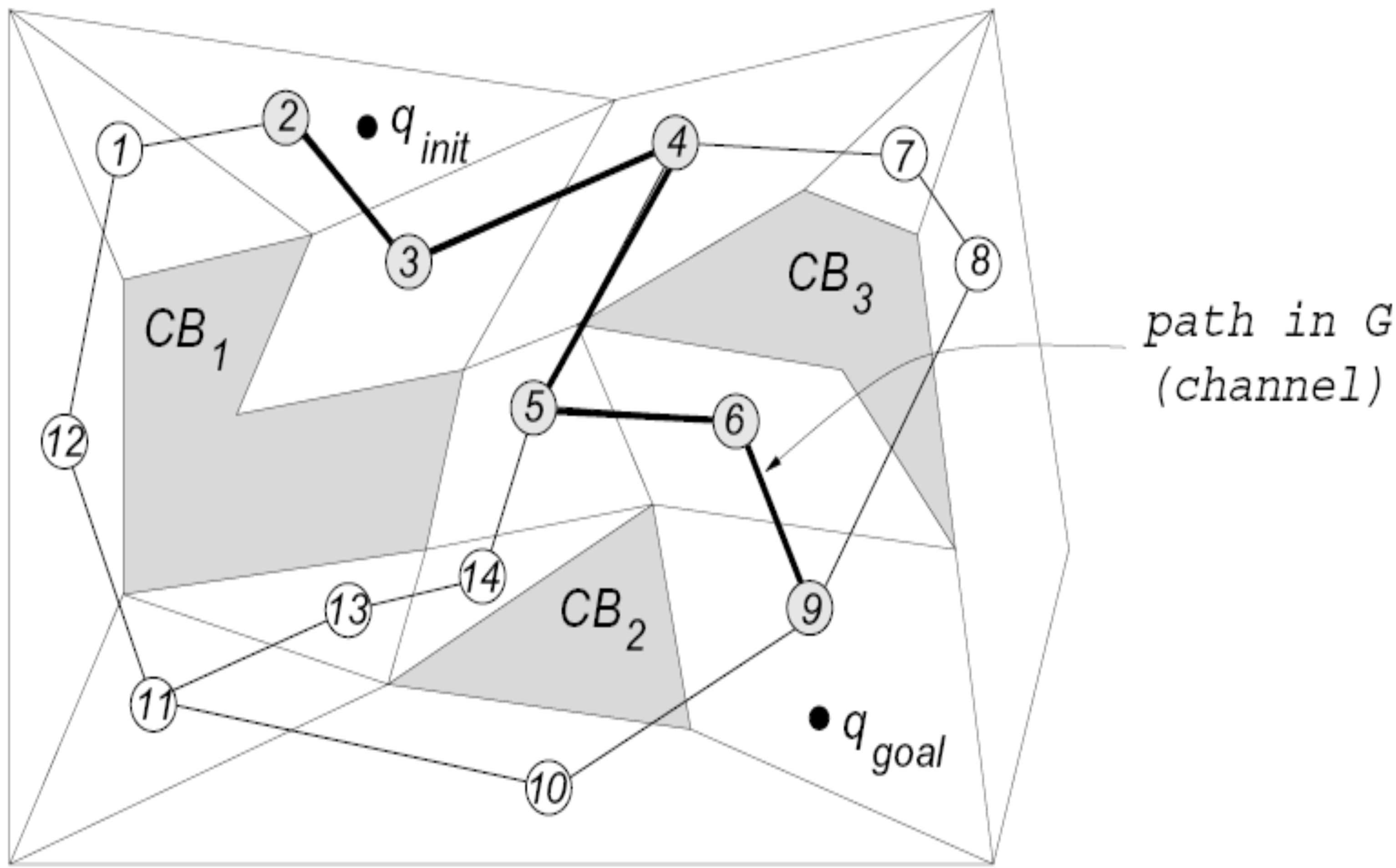
Convex Decomposition

- A convex polygonal decomposition K of C_{free} is a finite collection of convex polygons, called cells, such that the interiors of any two cells do not intersect and the union of all cells is C_{free}.
  – Two cells k and k′ ∈ K are adjacent iff k∩k′ is a line segment of non-zero length (i.e., not a single point)

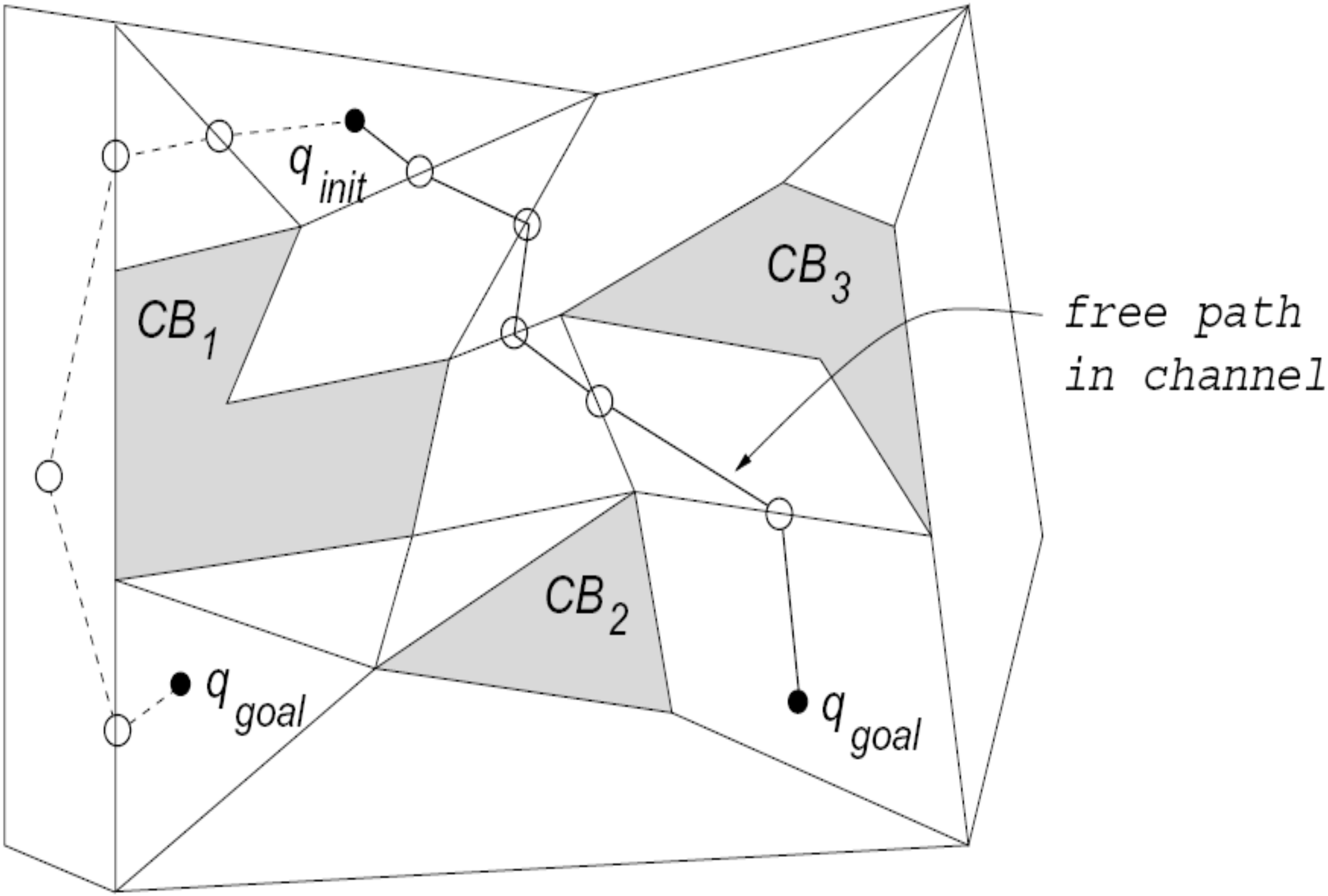
- The connectivity graph associated with a convex polygonal decomposition K of C_{free} is an undirected graph G where
  – nodes in G correspond to cells in K
  – nodes connected by edge in G iff corresponding cells adjacent in K



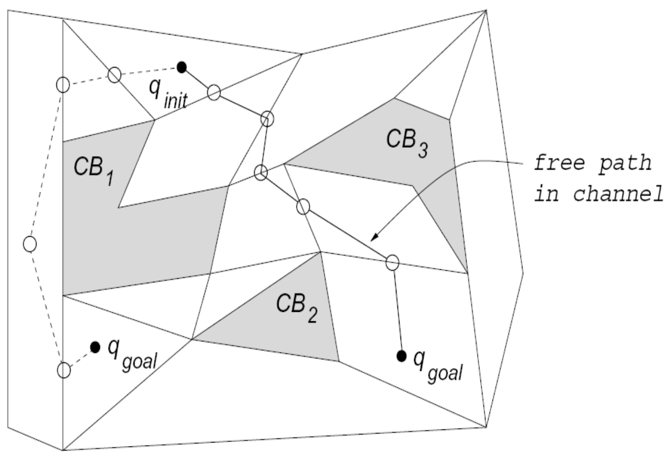
Convex Decomposition



Convex Decomposition



Convex Decomposition

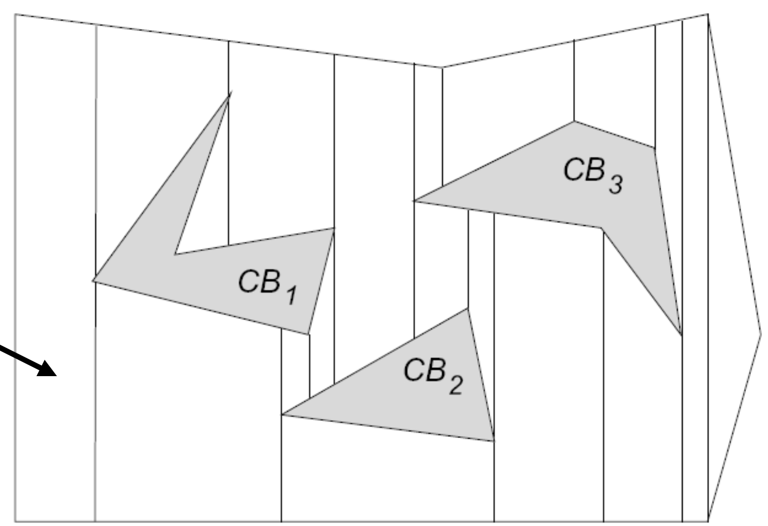


Bad news: Computing convex decomposition is not easy nor can be done efficiently. In fact the problem is NP hard to generate minimum number of convex components for polygon with holes

Trapezoidal Decomposition

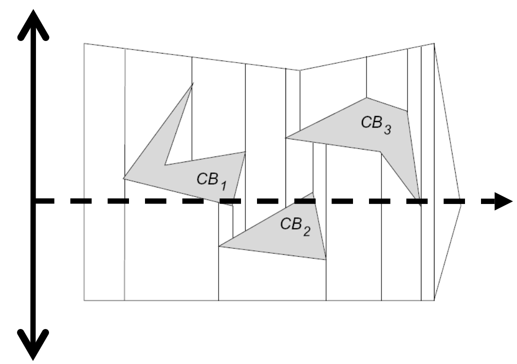
- Basic Idea: at every vertex of C-obstacle, extend a vertical line up and down in Cfree until it touches a C-obstacle or the boundary of Cfree

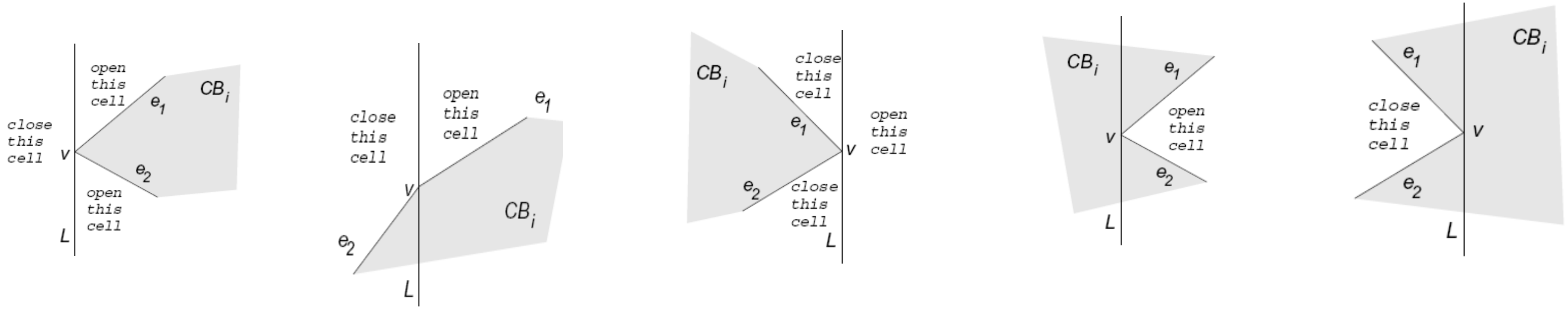
trapezoid



Trapezoidal Decomposition

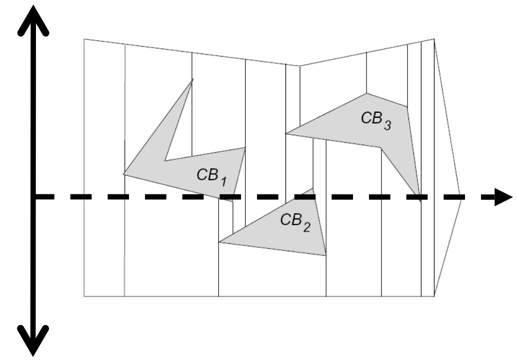
- Sweep line algorithm
  - Add vertical lines as we sweep from left to right
  - Events need to be handled accordingly

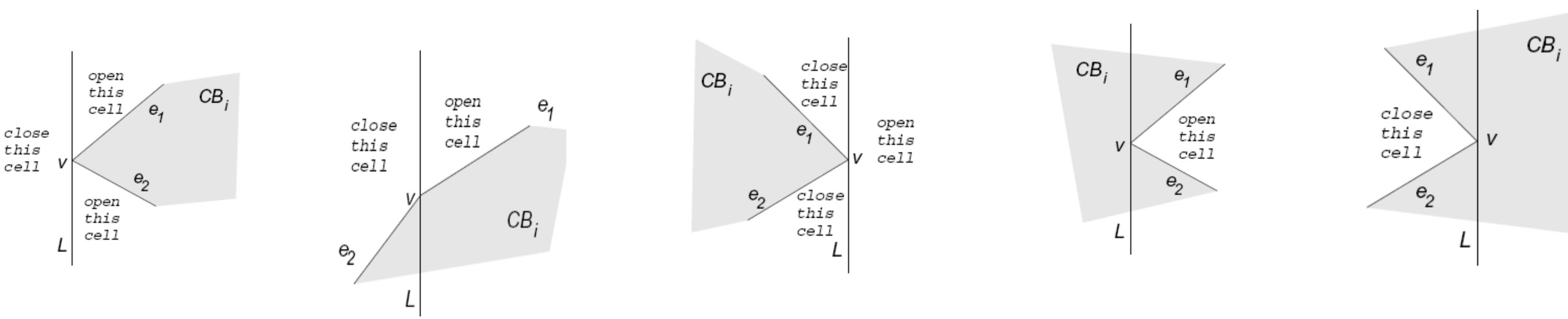




Trapezoidal Decomposition

- Sweep line algorithm
  - Add vertical lines as we sweep from left to right
  - Events need to be handled accordingly





trapezoidal decomposition can be built in O(n log n) time

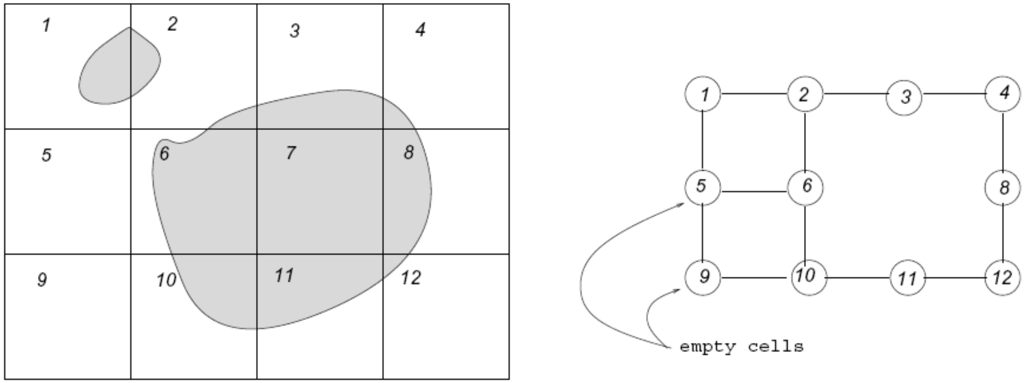
Approx. Cell Decomposition

- Construct a collection of non-overlapping cells such that the union of all the cells approximately covers the free C-space!

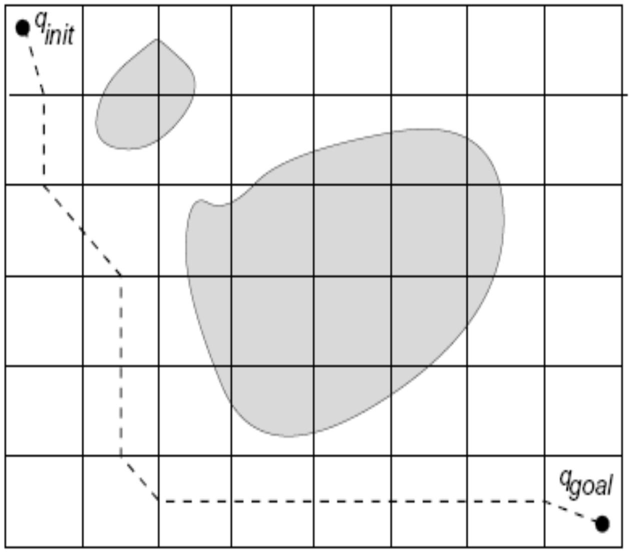
- Cell characteristics
  - Cell should have simple shape
  - Easy to test adjacency of two cells
  - Easy to find path across two adjacent cells

Approx. Cell Decomposition

- Each cell is
  - Empty
  - Full
  - Mixed

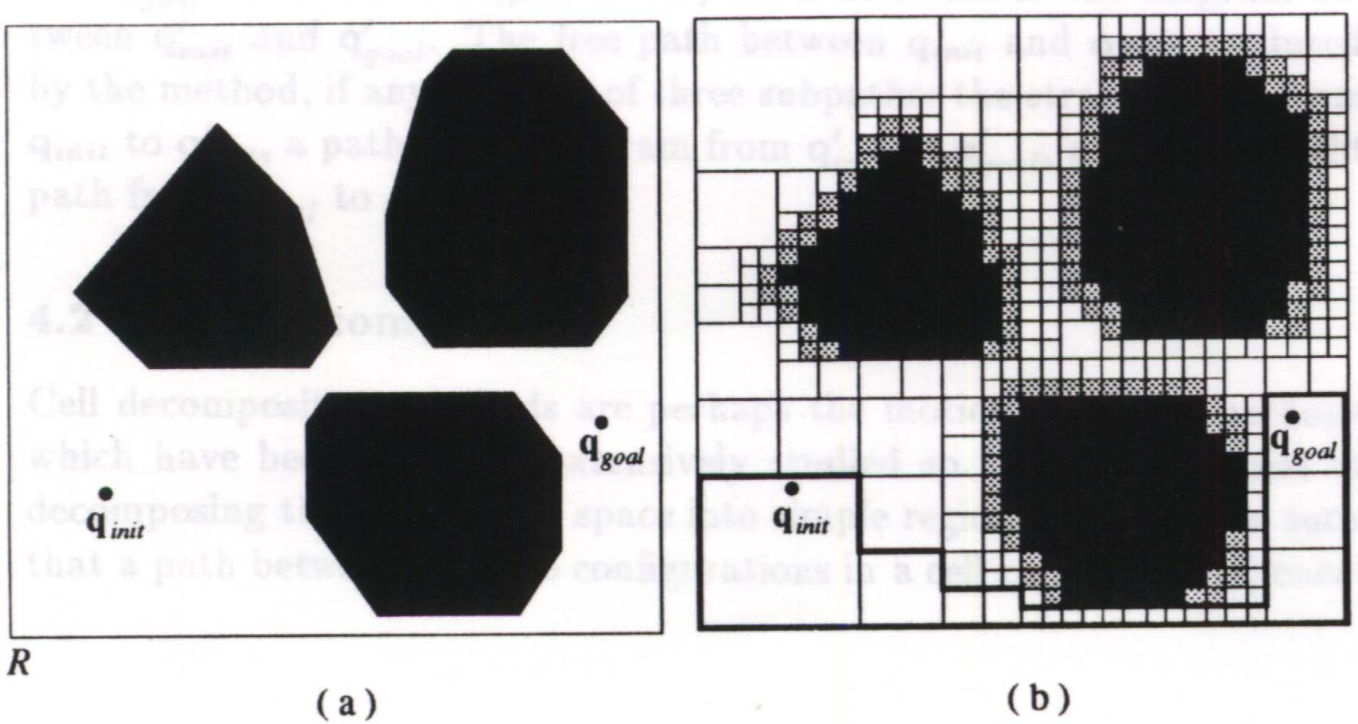


- Different resolution
  - Different roadmap



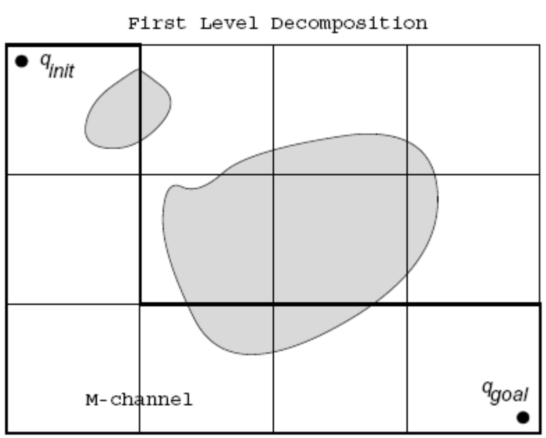
Approx. Cell Decomposition

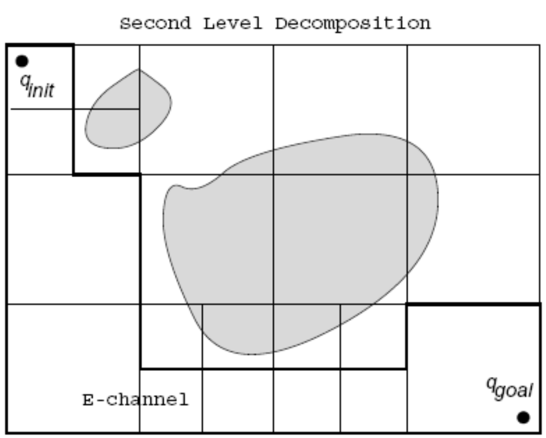
- Higher resolution around CBs



Approx. Cell Decomposition

- Hierarchical approach
  - Find path using empty and mixed cells
  - Further decompose mixed cells into smaller cells





Approx. Cell Decomposition

- Advantages:
  - simple, uniform decomposition
  - easy implementation
  - adaptive
- Disadvantages:
  - large storage requirement
  - Lose completeness

- Bottom line 1: We sacrifice exactness for simplicity and efficiency

- Bottom line 2: Approx. cell decomposition methods are practically for lower dimension C, i.e., dof <5, b/c they generate too many cells, i.e. (N^d) cells in d dimension

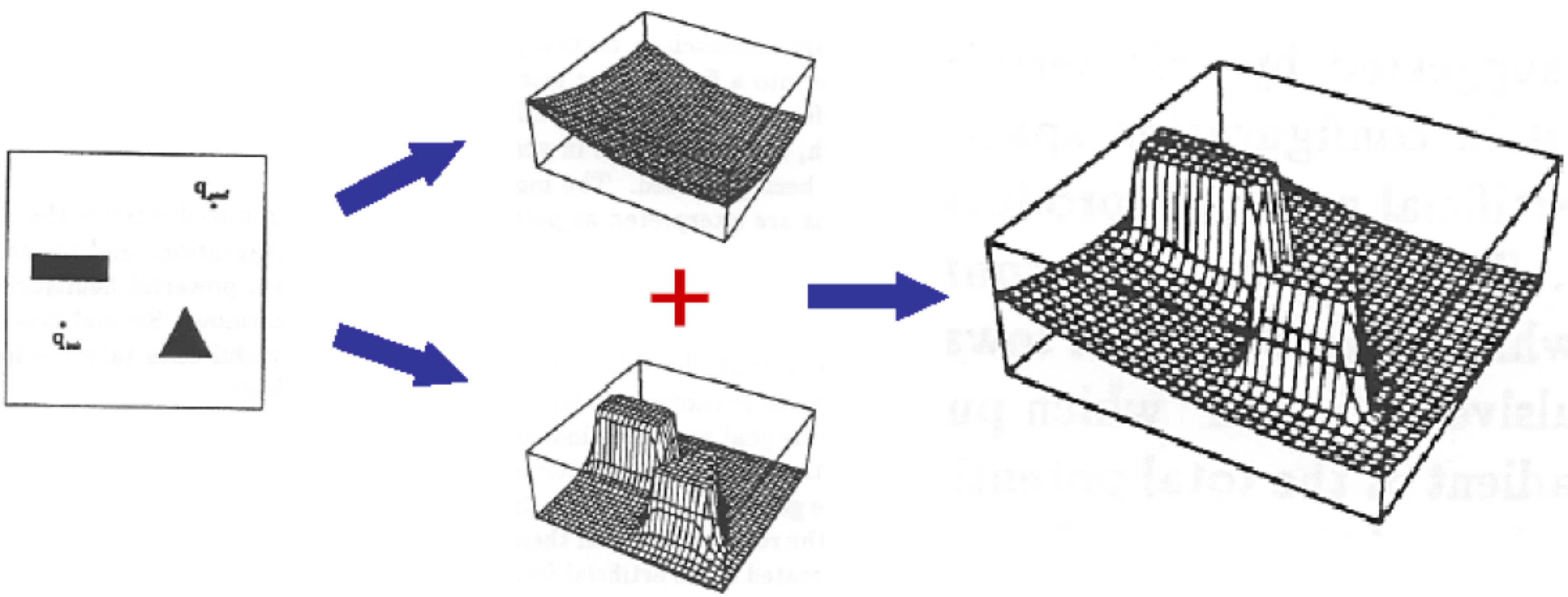
Potential Field Methods

- Approach initially proposed for real-time collision avoidance [Khatib, 86].
  - Hundreds of papers published on it

$$F_{Goal} = -k_p(x - x_{Goal})$$

$$F_{Obstacle} = \begin{cases} \eta \left(\frac{1}{\rho} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2} \frac{\partial \rho}{\partial x} & \text{if } \rho \leq \rho_0, \\ 0 & \text{if } \rho > \rho_0 \end{cases}$$

Potential Field Methods



Potential Field+Grid Search

- Superimpose a grid over C-space
- Each cell has a potential value
- Search from start to goal on the grid using best-first search or A* search

Potential Field Methods

- At each step move an increment in the direction that minimizes the energy
  - + Good heuristic for high DOF

– Can get trapped in local minima
  - use some probabilistic motion to escape

– Oscillations can also occur