
Path Planning for Point Robots

Sung-Eui Yoon
(윤성의)

Course URL:
<http://sglab.kaist.ac.kr/~sungeui/MPA>

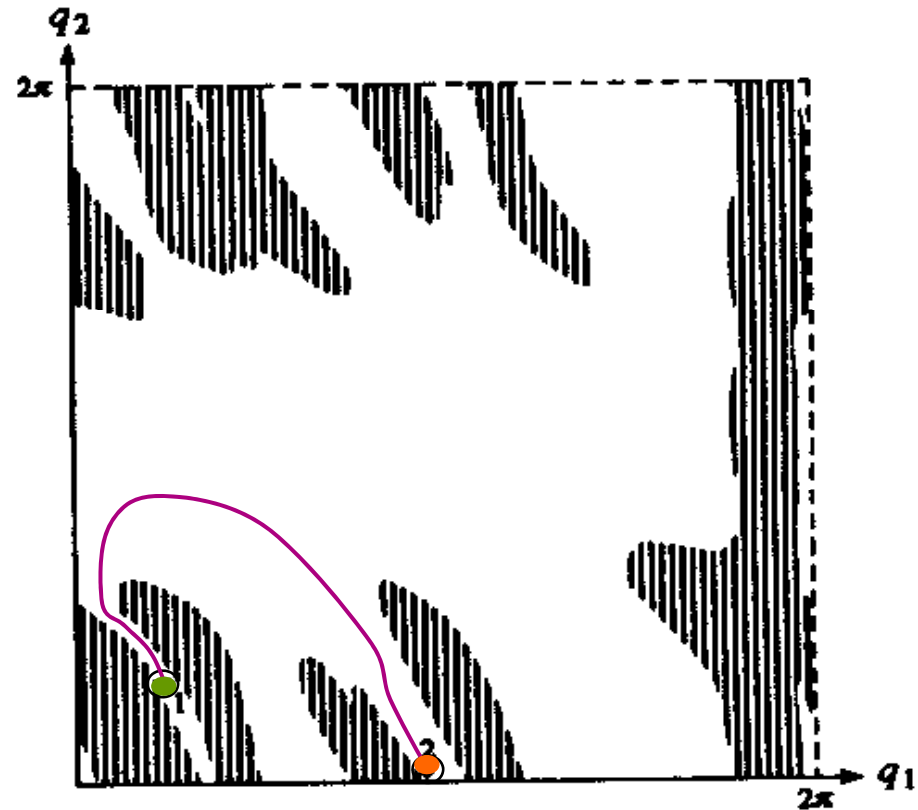
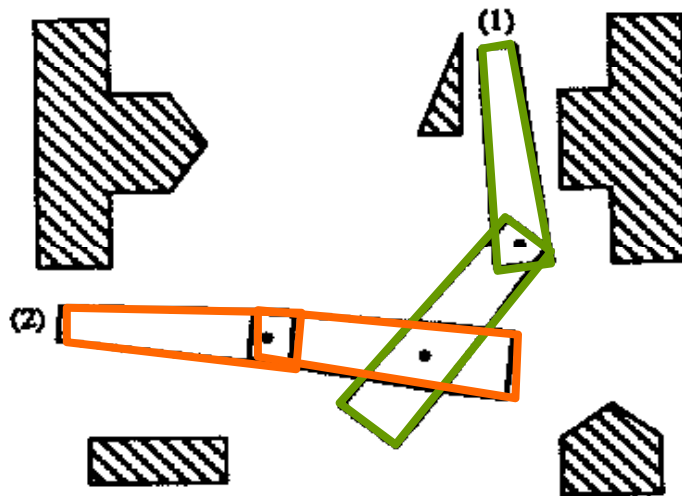
KAIST



Class Objectives

- **Motion planning framework**
- **Classic motion planning approaches**

Configuration Space: Tool to Map a Robot to a Point



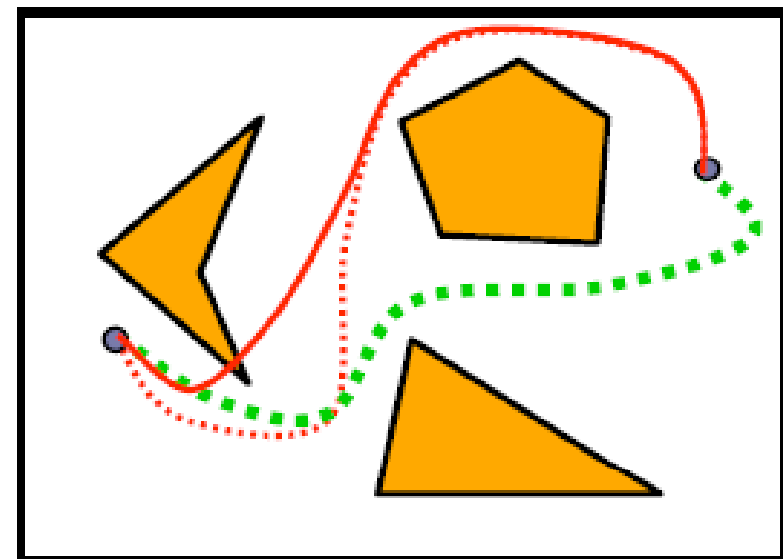
Problem

□ Input

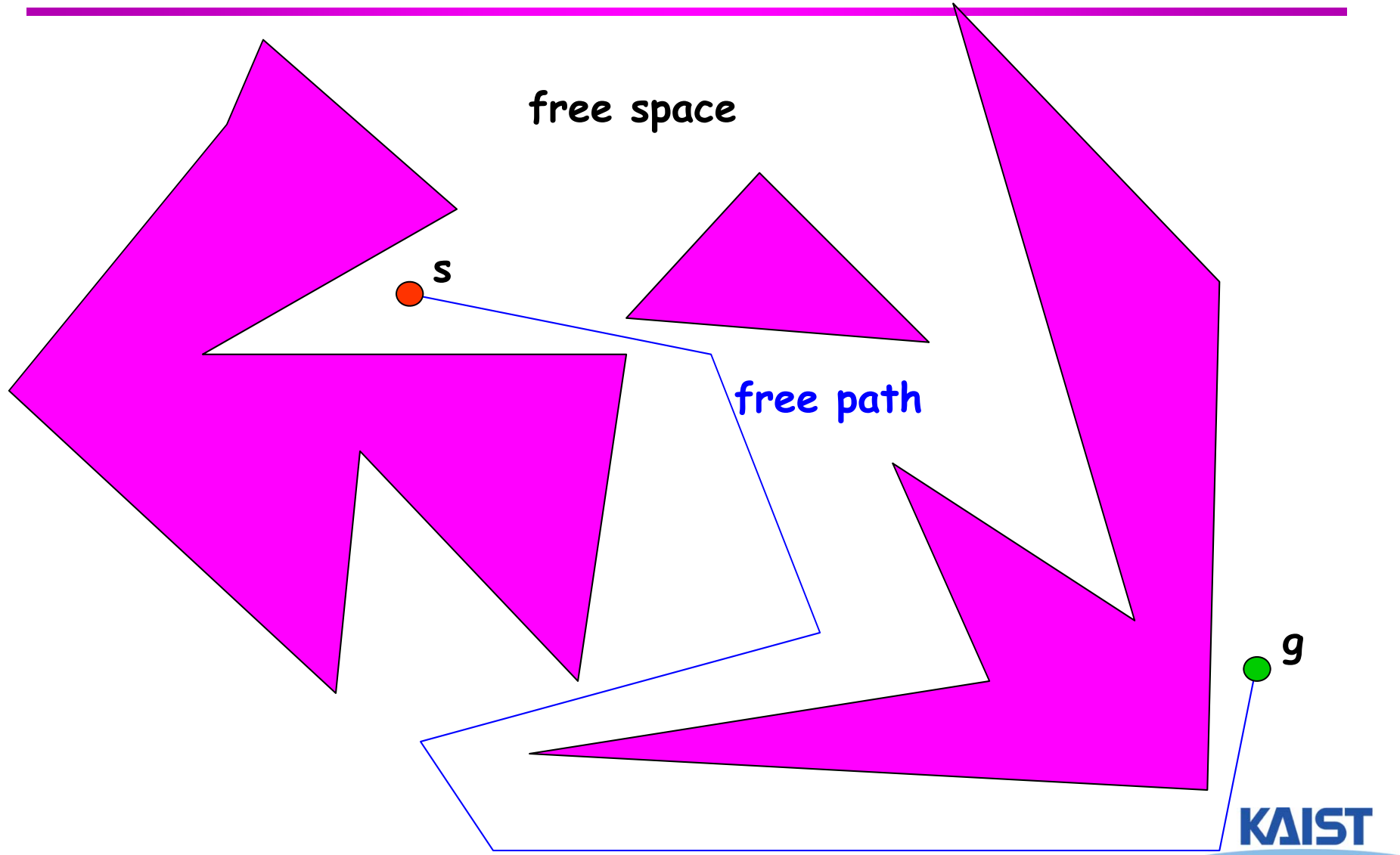
- Robot represented as a **point** in the **plane**
- Obstacles represented as polygons
- Initial and goal positions

□ Output

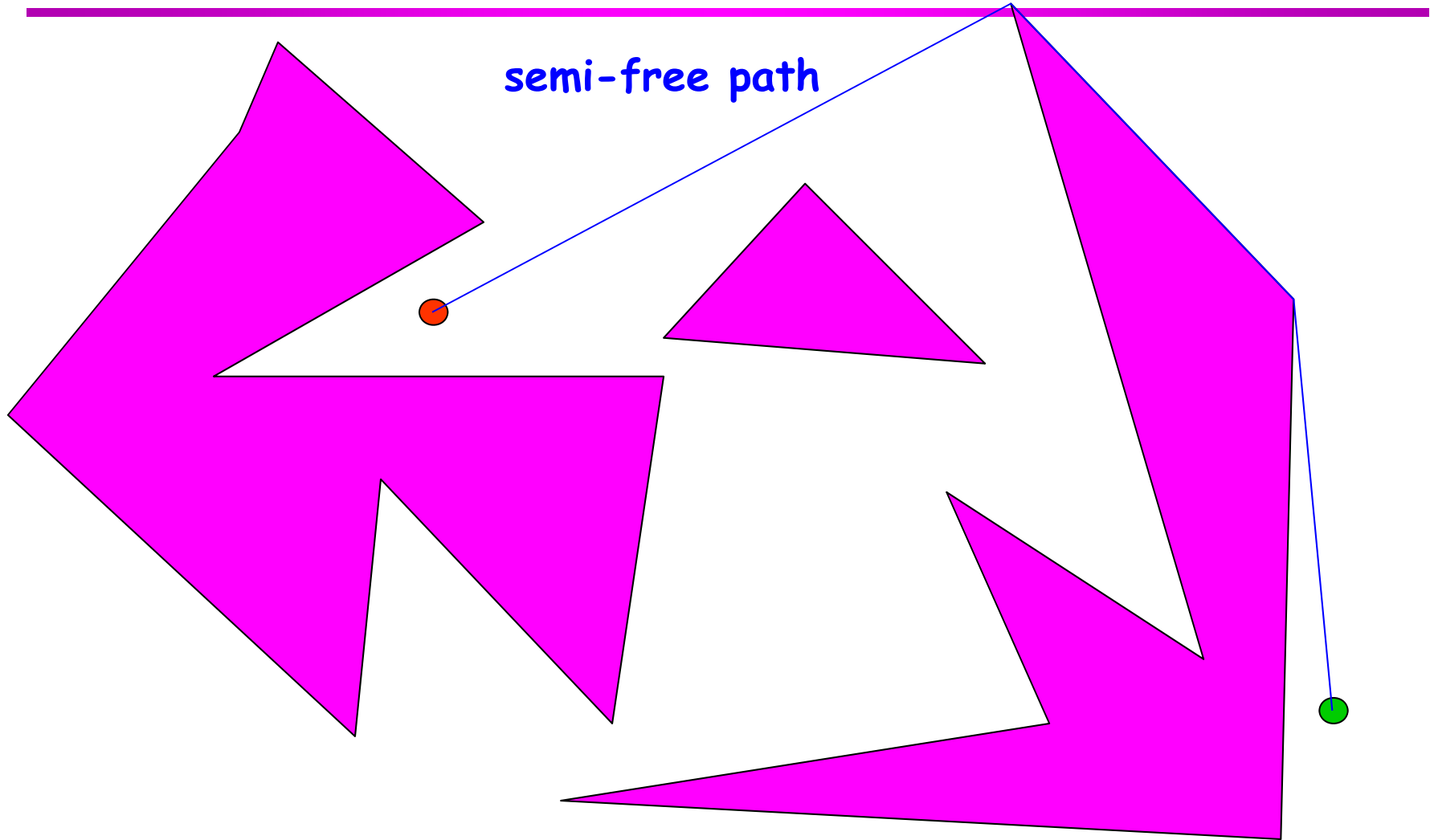
A collision-free path between the initial and goal positions



Problem



Problem



Types of Path Constraints

- **Local** constraints:
lie in free space
- **Differential** constraints:
have bounded curvature
- **Global** constraints:
have minimal length

Motion-Planning Framework

Continuous representation

(configuration space formulation)



Discretization

(random sampling, processing critical geometric events)

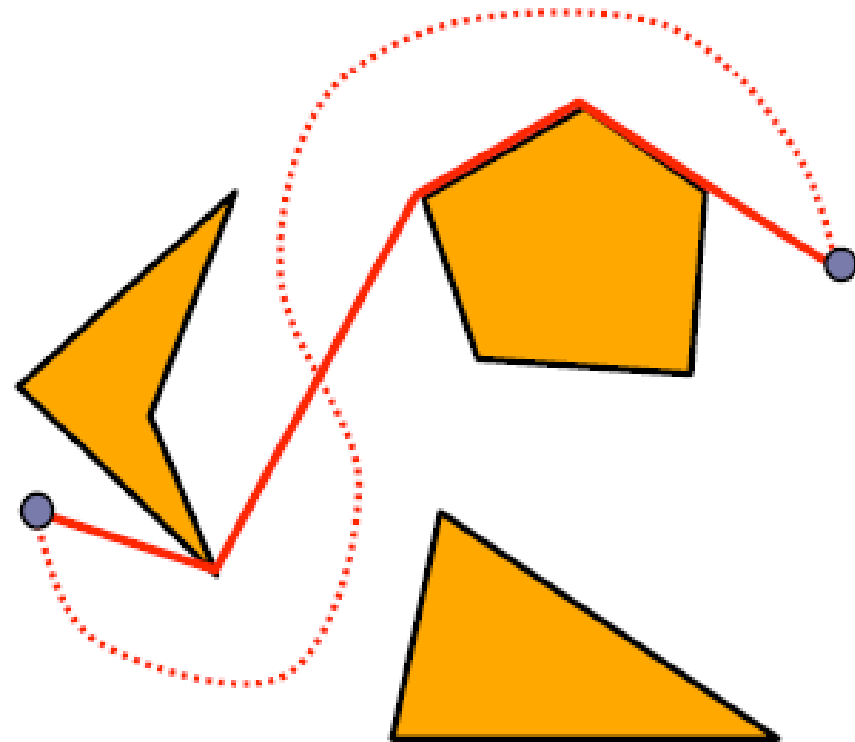


Graph searching

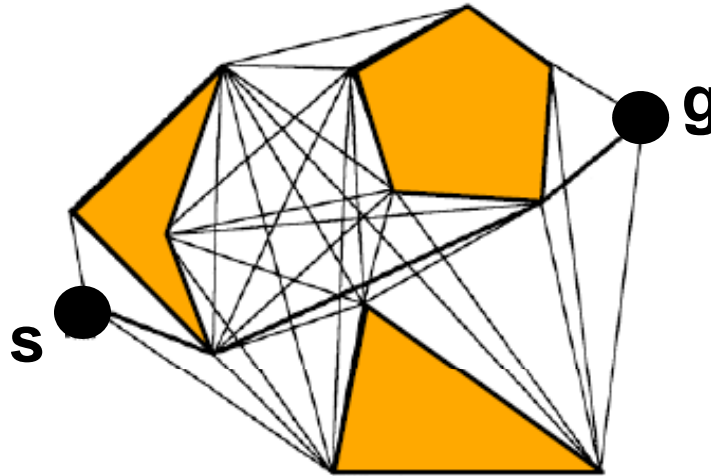
(blind, best-first, A*)

Visibility graph method

- **Observation:** If there is a collision-free path between two points, then there is a polygonal path that bends only at the obstacles vertices.
- **Why?**
Any collision-free path can be transformed into a polygonal path that bends only at the obstacle vertices.
- A **polygonal path** is a piecewise linear curve.

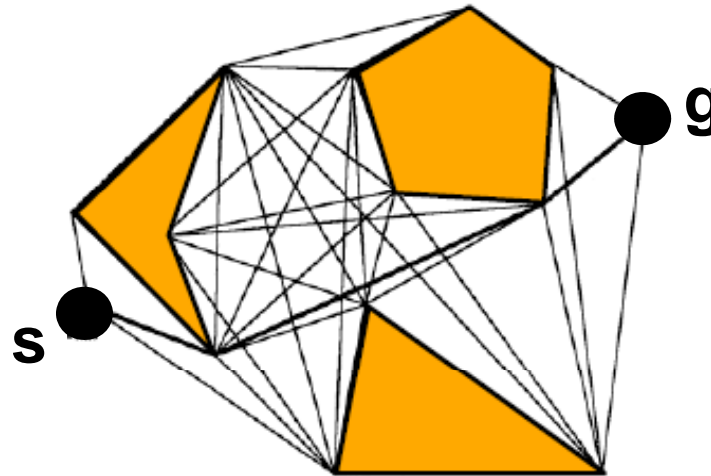


Visibility Graph



- A **visibility graph** is a graph such that
 - Nodes: s , g , or obstacle vertices
 - Edges: An edge exists between nodes u and v if the line segment between u and v is an obstacle edge or it does not intersect the obstacles

Visibility Graph



- A **visibility graph**
 - Introduced in the late 60s
 - Can produce shortest paths in 2-D configuration spaces

Simple Algorithm

- **Input:** s , q , polygonal obstacles
 - **Output:** visibility graph G
- 1: **for** every pair of nodes u, v
 - 2: **if** segment (u, v) is an obstacle edge **then**
 - 3: insert edge (u, v) into G ;
 - 4: **else**
 - 5: **for** every obstacle edge e
 - 6: **if** segment (u, v) intersects e
 - 7: go to (1);
 - 8: insert edge (u, v) into G ;
 - 9: Search a path with G using A^*

Computation Efficiency

```
1: for every pair of nodes  $u, v$   $O(n^2)$ 
2:   if segment  $(u, v)$  is an obstacle edge then  $O(n)$ 
3:     insert edge  $(u, v)$  into  $G$ ;
4:   else
5:     for every obstacle edge  $e$   $O(n)$ 
6:       if segment  $(u, v)$  intersects  $e$ 
7:         go to (1);
8:     insert edge  $(u, v)$  into  $G$ ;
```

- **Simple algorithm: $O(n^3)$ time**
- **More efficient algorithms**
 - Rotational sweep $O(n^2 \log n)$ time, etc.
- **$O(n^2)$ space**

Motion-Planning Framework

Continuous representation
(configuration space formulation)



Discretization
(random sampling, processing critical geometric events)

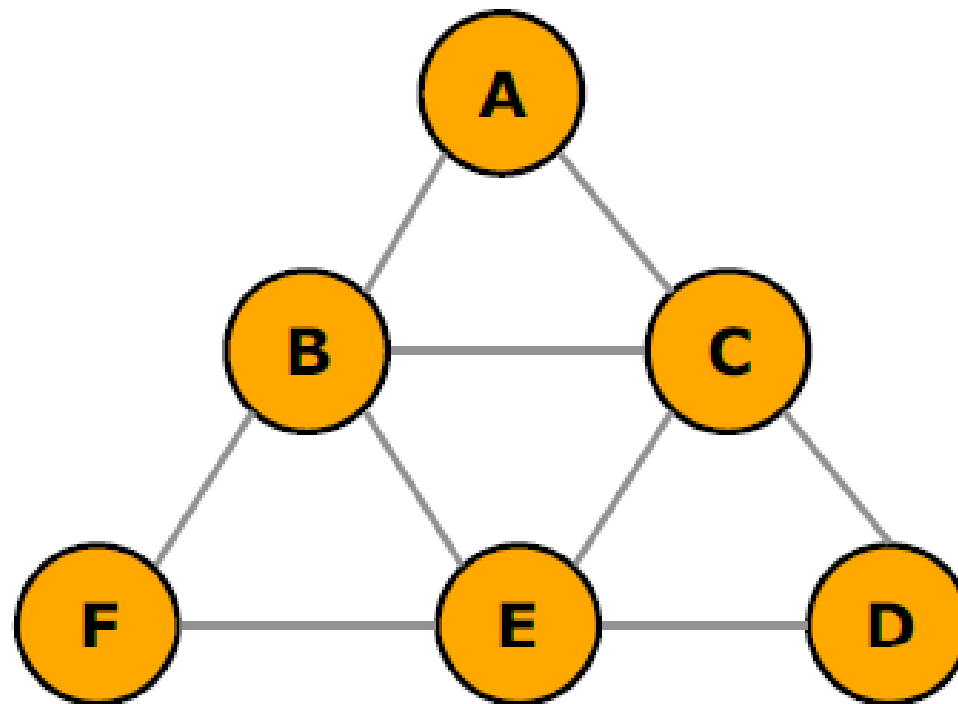


Graph searching
(blind, best-first, A*)

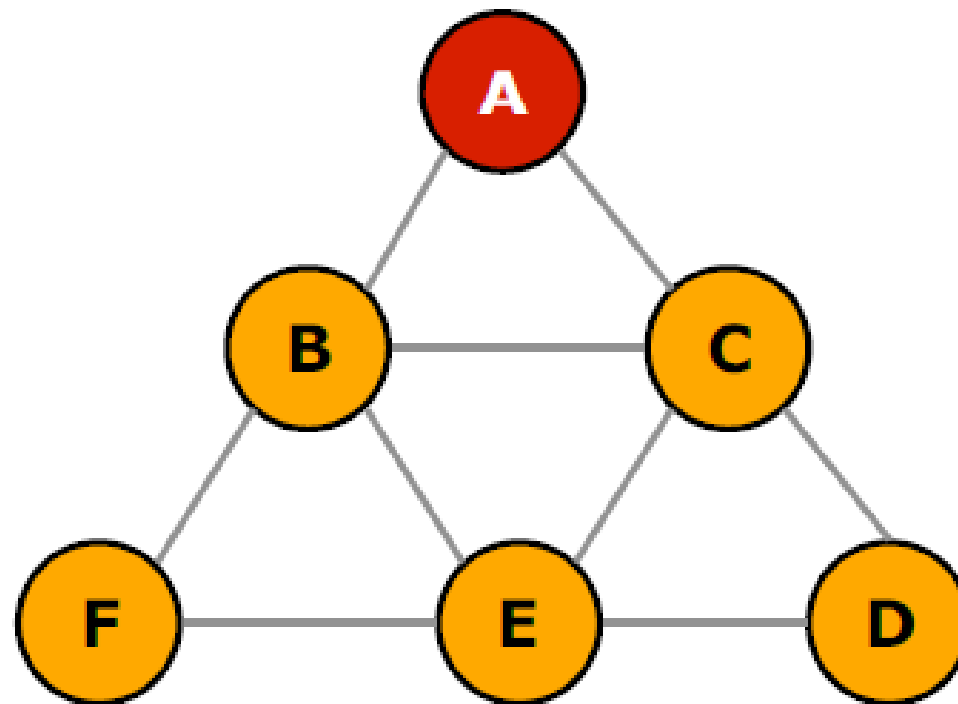
Graph Search Algorithms

- Breadth, depth-first, best-first
- Dijkstra's algorithm
- A*

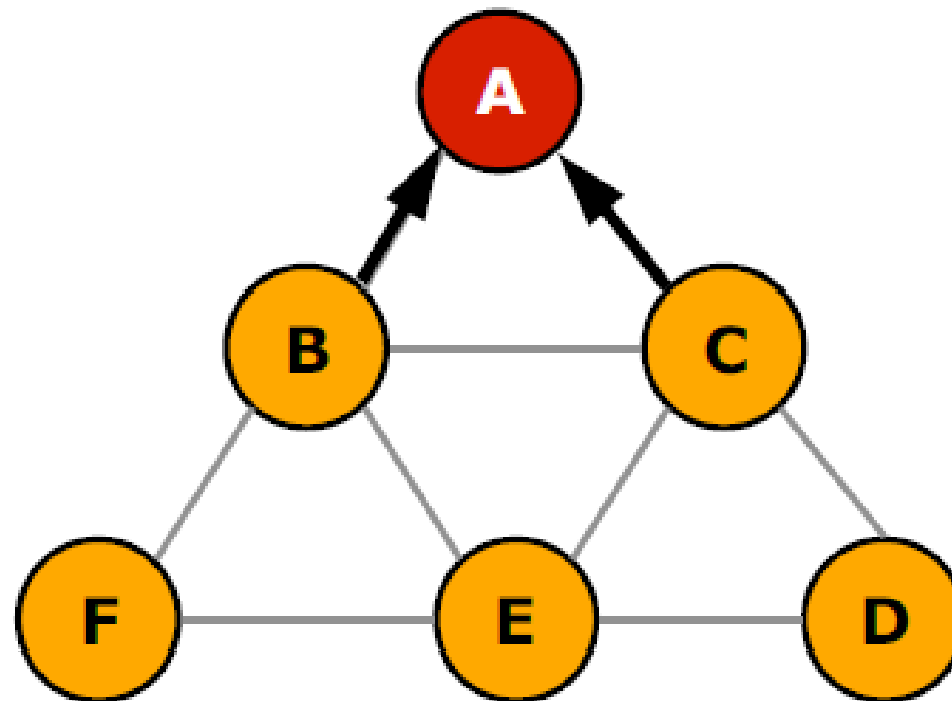
Breadth-first search



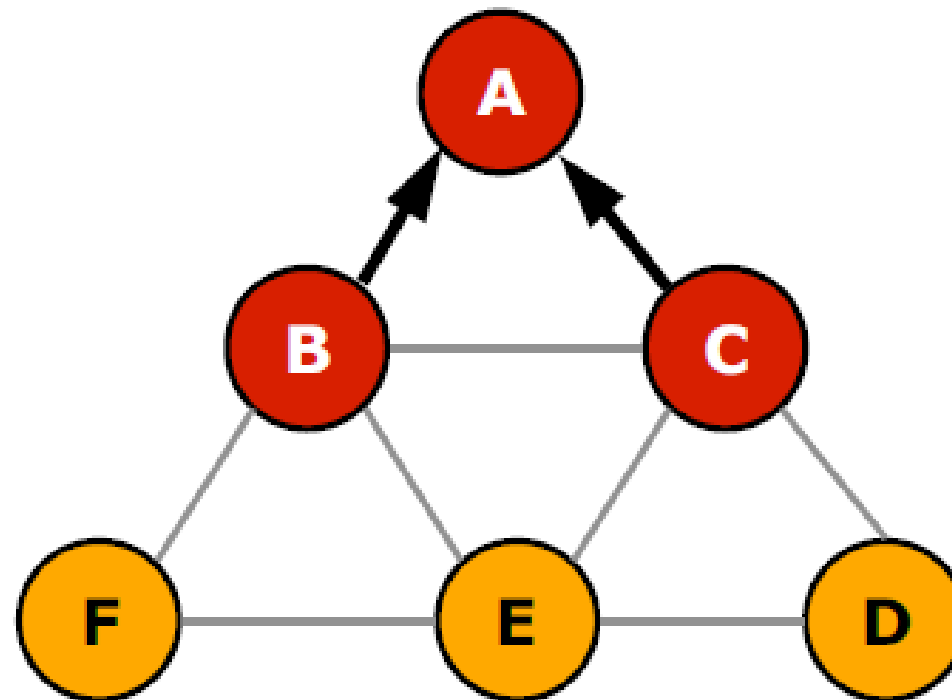
Breadth-first search



Breadth-first search



Breadth-first search



Dijkstra's Shortest Path Algorithm

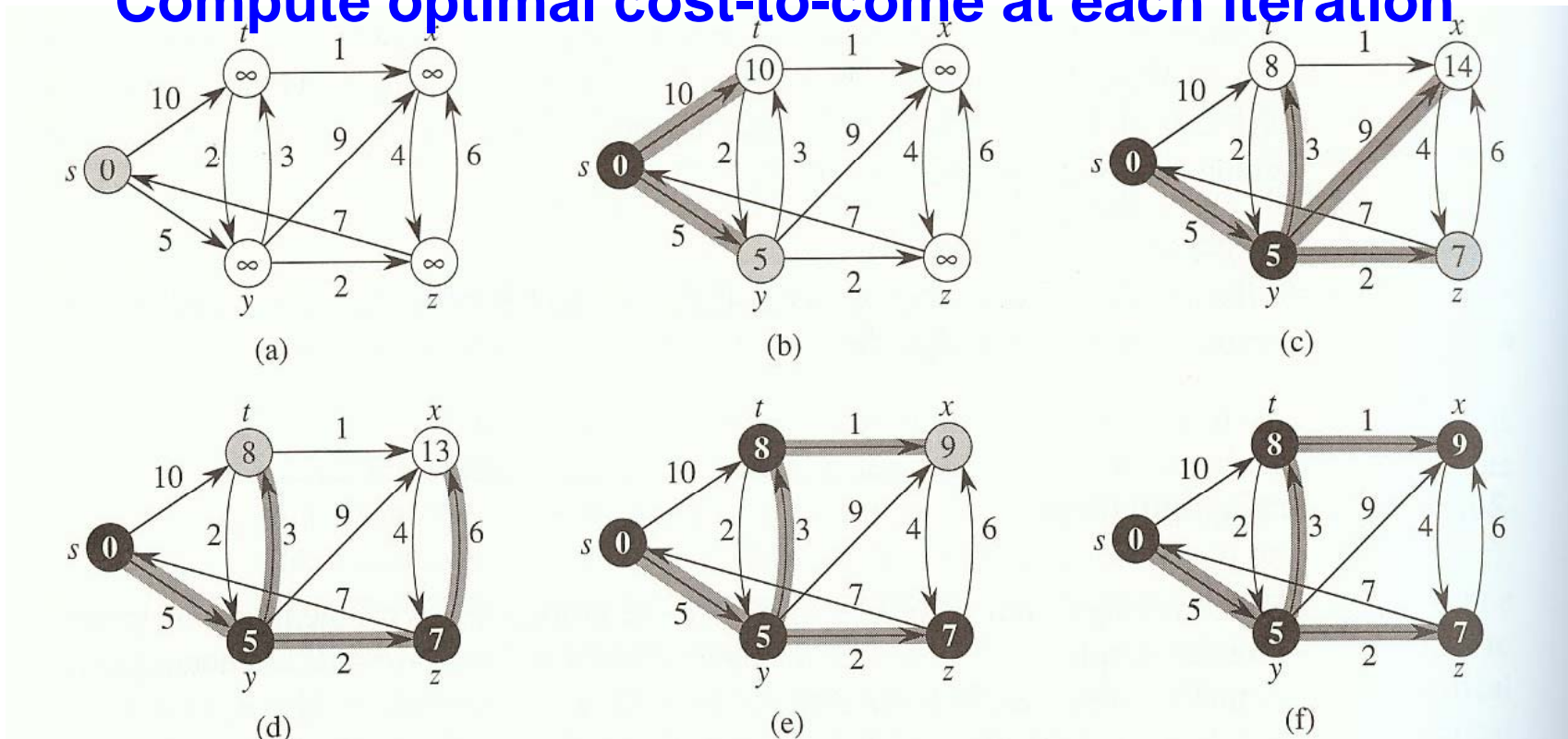
- Given a (non-negative) weighted graph, two vertices, s and g :
 - Find a path of minimum total weight between them
 - Also, find minimum paths to other vertices
 - Has $O(|V| \lg|V| + |E|)$

Dijkstra's Shortest Path Algorithm

- A set S
 - Contains vertices whose final shortest-path cost has been determined
- **DIJKSTRA** (G, s)
 1. Initialize-Single-Source (G, s)
 2. $S \leftarrow \text{empty}$
 3. Queue \leftarrow Vertices of G
 4. **While** Queue is not empty
 5. **Do** $u \leftarrow$ min-cost from Queue
 6. $S \leftarrow$ union of S and $\{u\}$
 7. **for** each vertex v in Adj [u]
 8. **do** RELAX (u, v)

Dijkstra's Shortest Path Algorithm

Compute optimal cost-to-come at each iteration



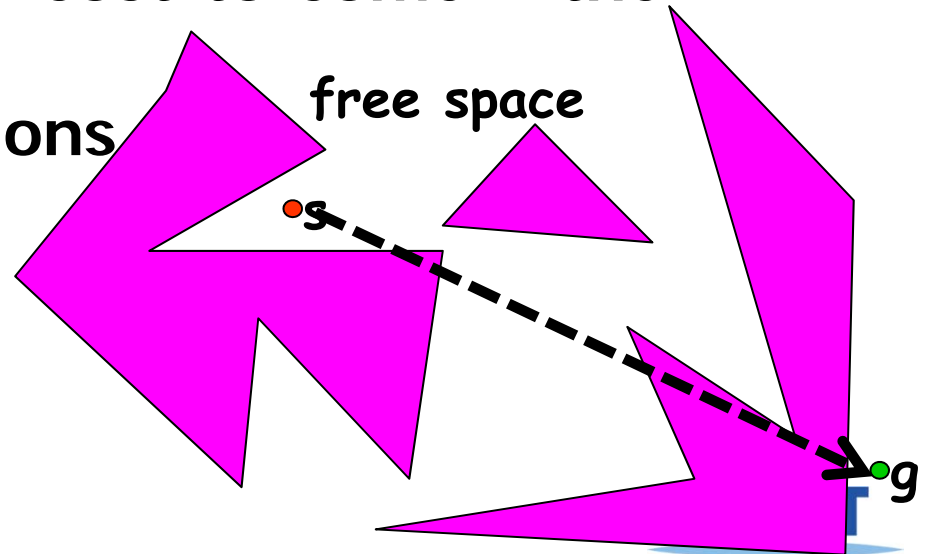
Black vertices are in the set.

White vertices are in the queue.

Shaded one is chosen for relaxation.

A* Search Algorithm

- An extension of Dijkstra's algorithm based on a heuristic estimate
 - Conservatively estimate the cost-to-go from a vertex to the goal
 - The estimate should not be greater than the optimal cost-to-go
 - Sort vertices based on "cost-to-come + the estimated cost-to-go"
 - Can find optimal solutions with fewer steps



Best-First Search

- **Pick a next node based on an estimate of the optimal cost-to-go cost**
 - **Greedily finds solutions that look good**
 - **Solutions may not be optimal**
 - **Can find solutions quite fast, but can be also very slow**

Framework

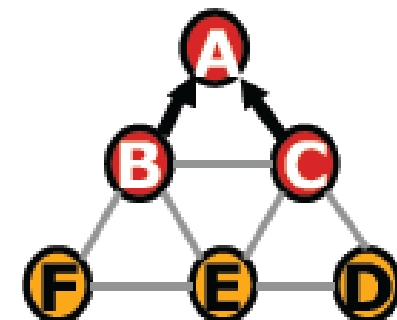
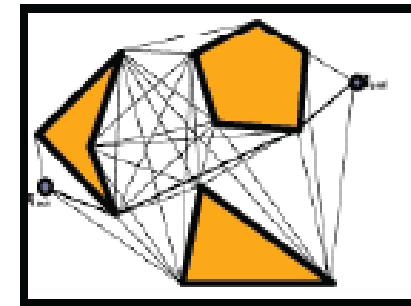
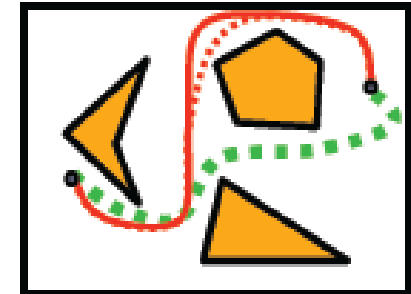
continuous representation



discretization
construct visibility graph



graph searching
breadth-first search



Computational Efficiency

- Running time $O(n^3)$
 - Compute the visibility graph
 - Search the graph
- Space $O(n^2)$

- Can we do better?

Classic Path Planning Approaches

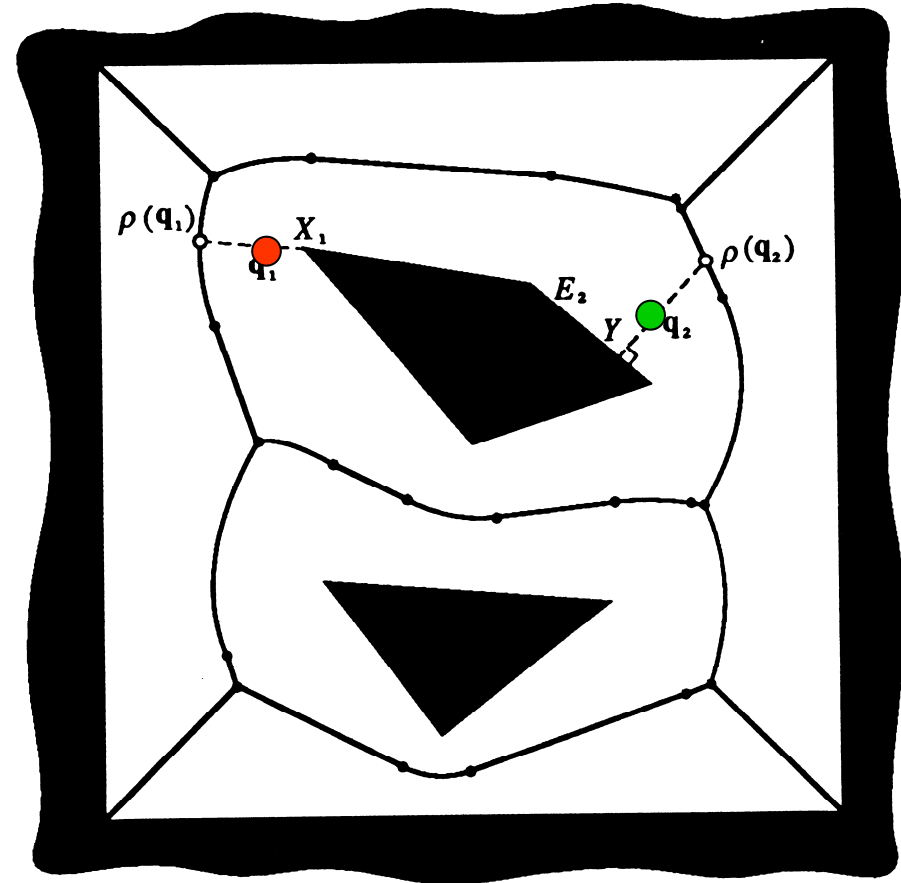
- **Roadmap**
 - Represent the connectivity of the free space by a network of 1-D curves
- **Cell decomposition**
 - Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells
- **Potential field**
 - Define a function over the free space that has a global minimum at the goal configuration and follow its steepest descent

Classic Path Planning Approaches

- **Roadmap**
 - Represent the connectivity of the free space by a network of 1-D curves
- **Cell decomposition**
 - Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells
- **Potential field**
 - Define a function over the free space that has a global minimum at the goal configuration and follow its steepest descent

Roadmap Methods

- Visibility Graph
 - Shakey project, SRI [Nilsson 69]
- Voronoi diagram
 - Introduced by computational geometry researchers
 - Generate paths that maximize clearance
 - $O(n \log n)$ time and $O(n)$ space



Other Roadmap Methods

- **Visibility graph**
- **Voronoi diagram**
- **Silhouette**
 - **First complete general method that applies to spaces of any dimension and is singly exponential in # of dimensions [Canny, 87]**
- **Probabilistic roadmaps**

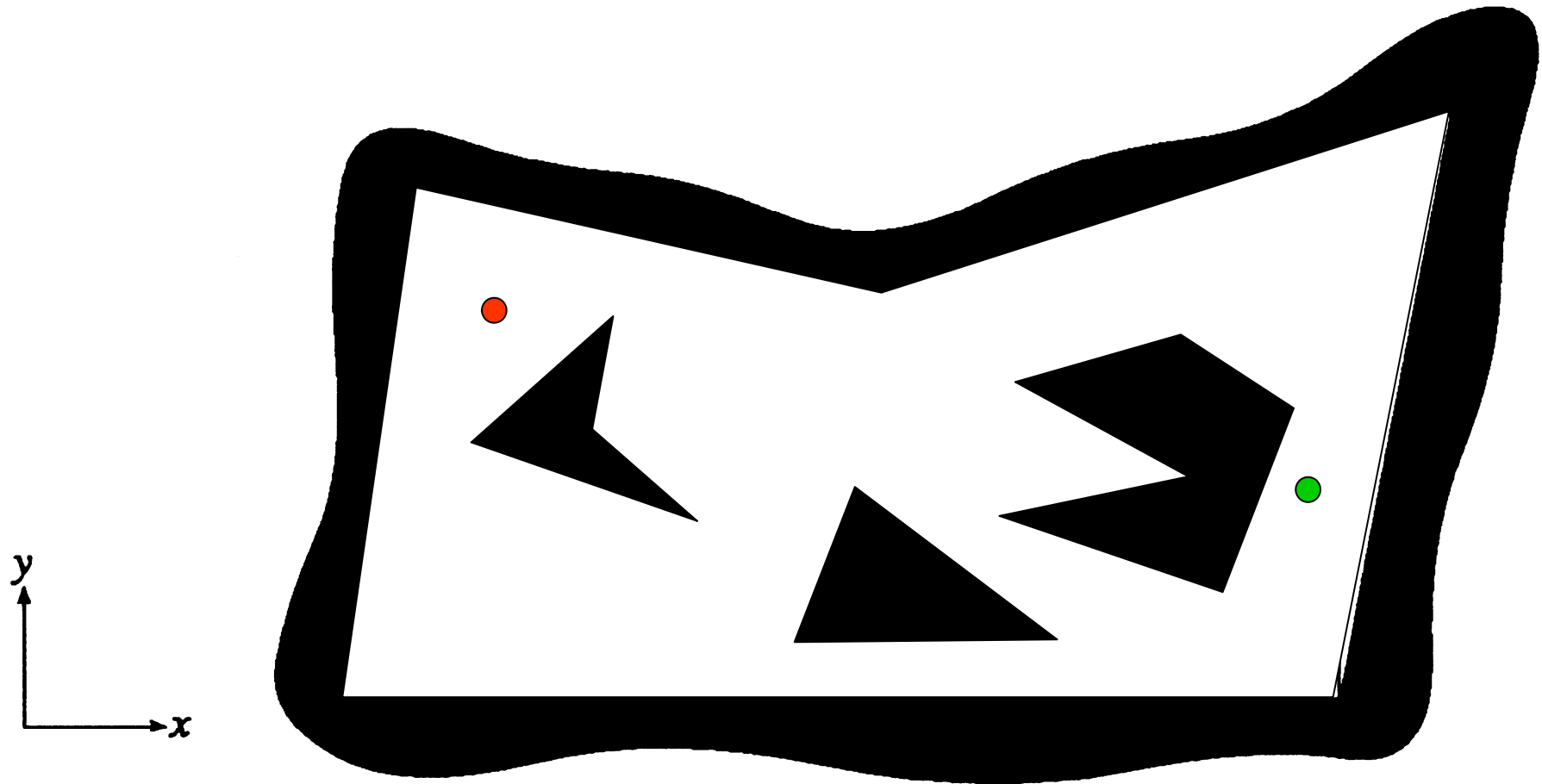
Classic Path Planning Approaches

- **Roadmap**
 - Represent the connectivity of the free space by a network of 1-D curves
- **Cell decomposition**
 - Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells
- **Potential field**
 - Define a function over the free space that has a global minimum at the goal configuration and follow its steepest descent

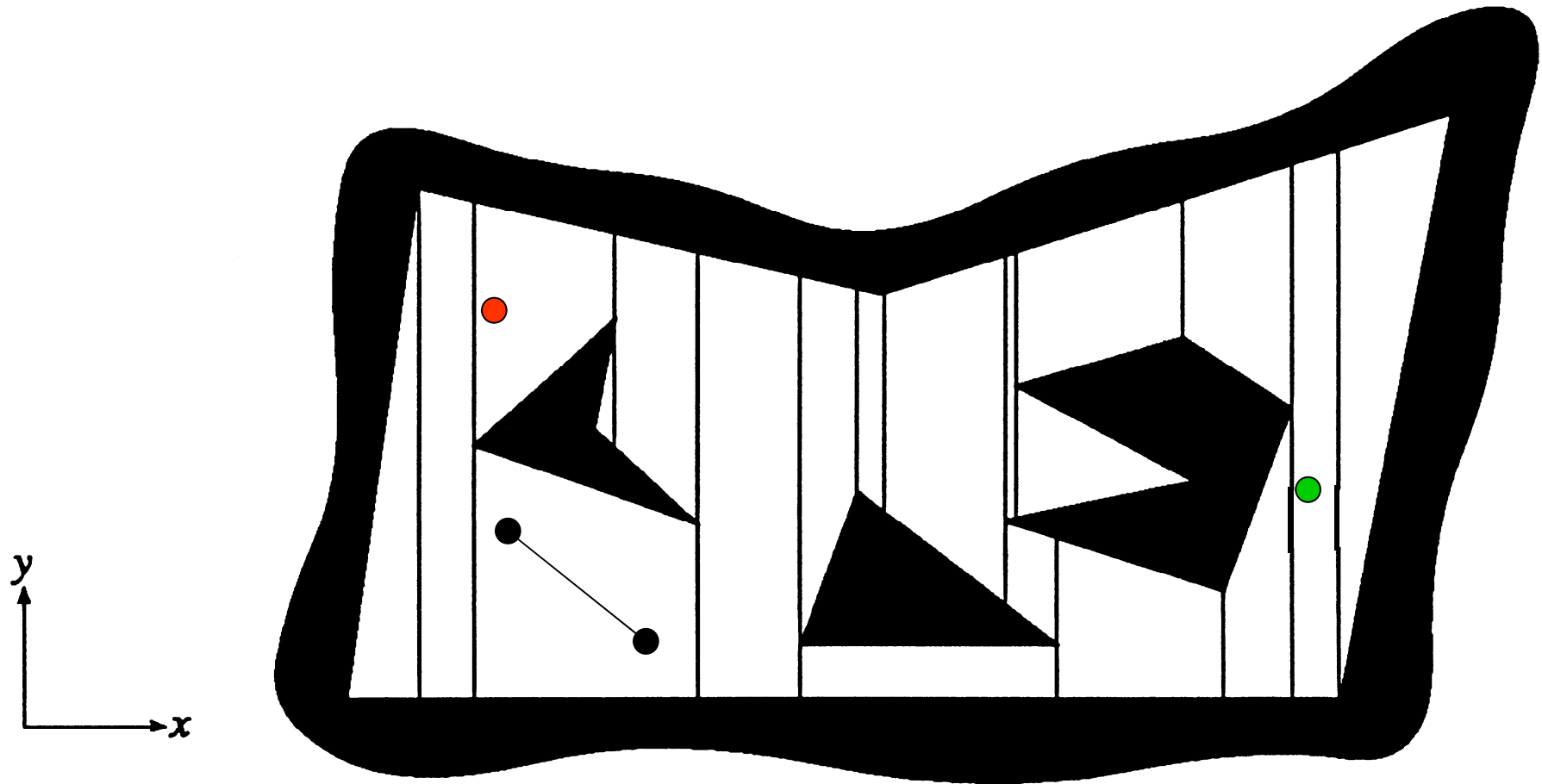
Cell-Decomposition Methods

- **Two classes of methods:**
 - Exact and approximate cell decompositions
- **Exact cell decomposition**
 - The free space F is represented by a collection of non-overlapping cells whose union is exactly F
 - Example: trapezoidal decomposition

Trapezoidal Decomposition

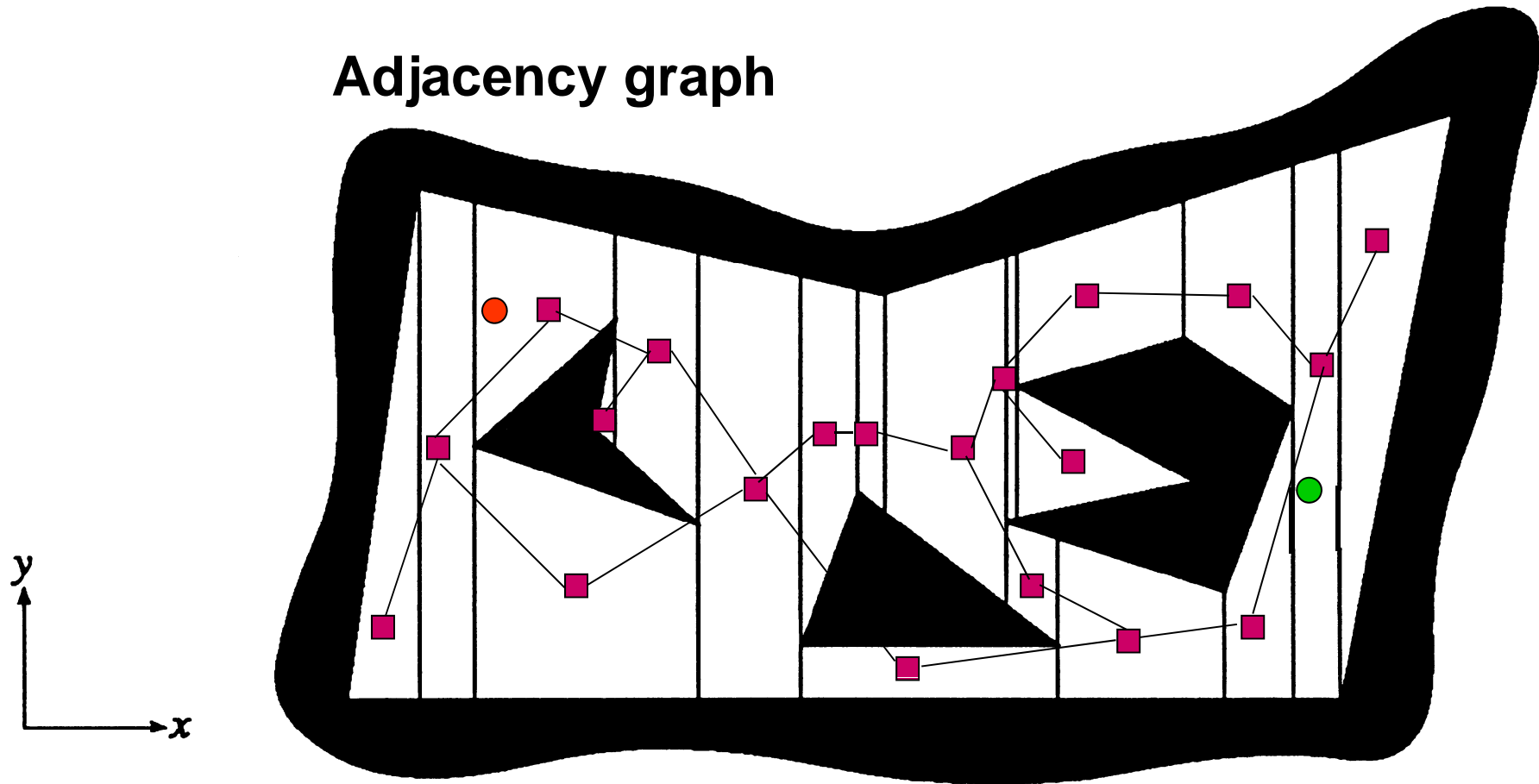


Trapezoidal Decomposition

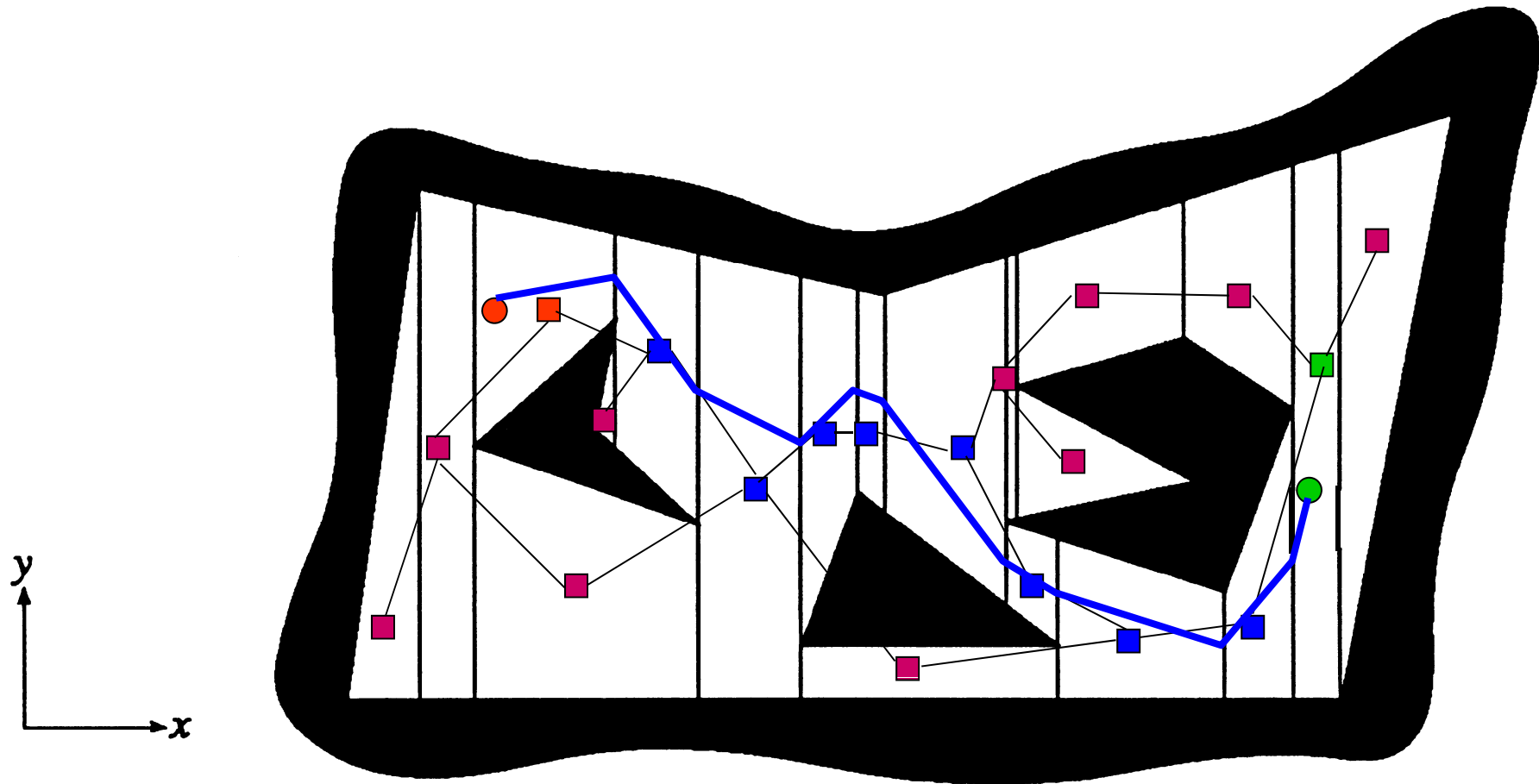


Trapezoidal Decomposition

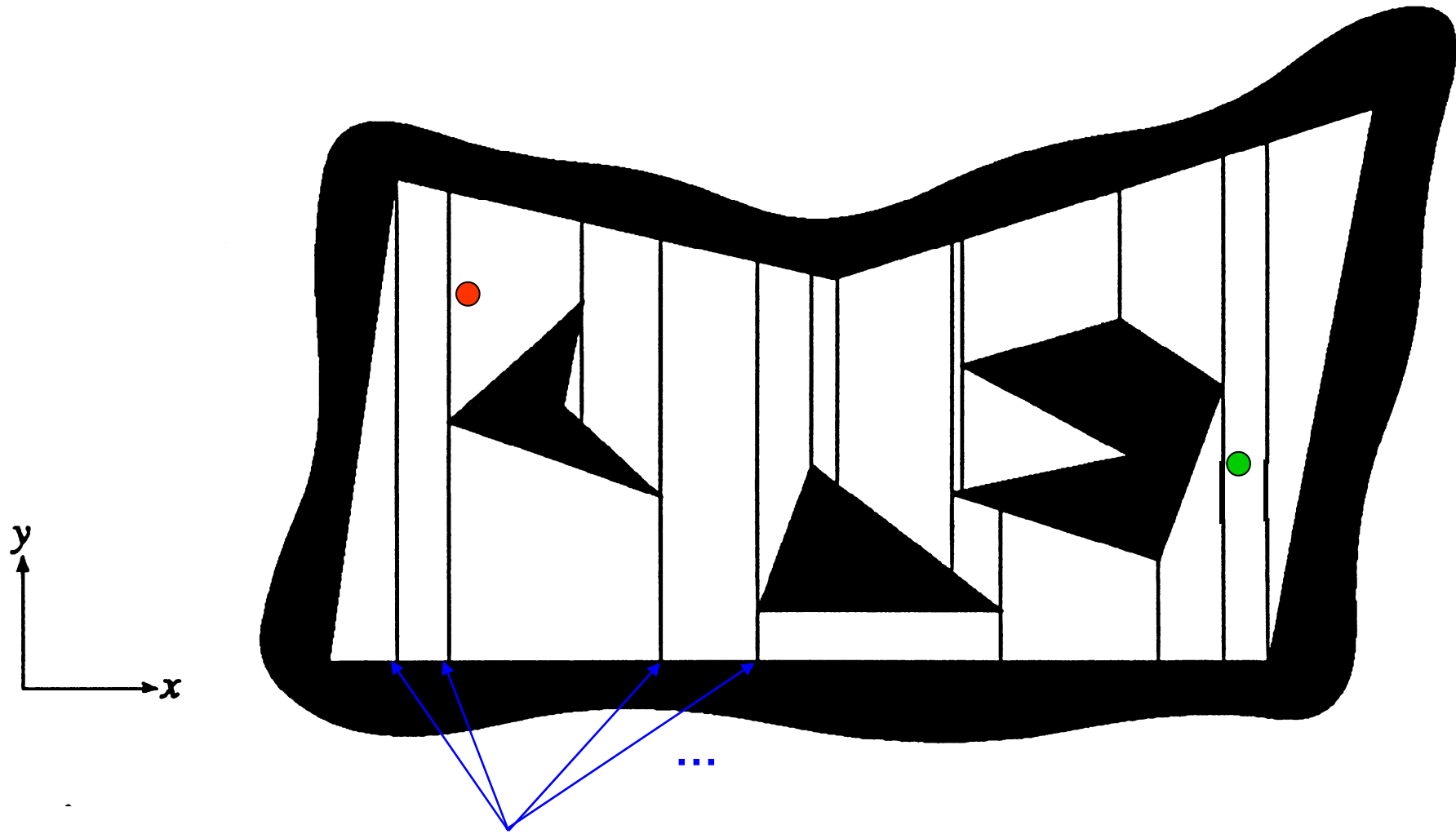
Adjacency graph



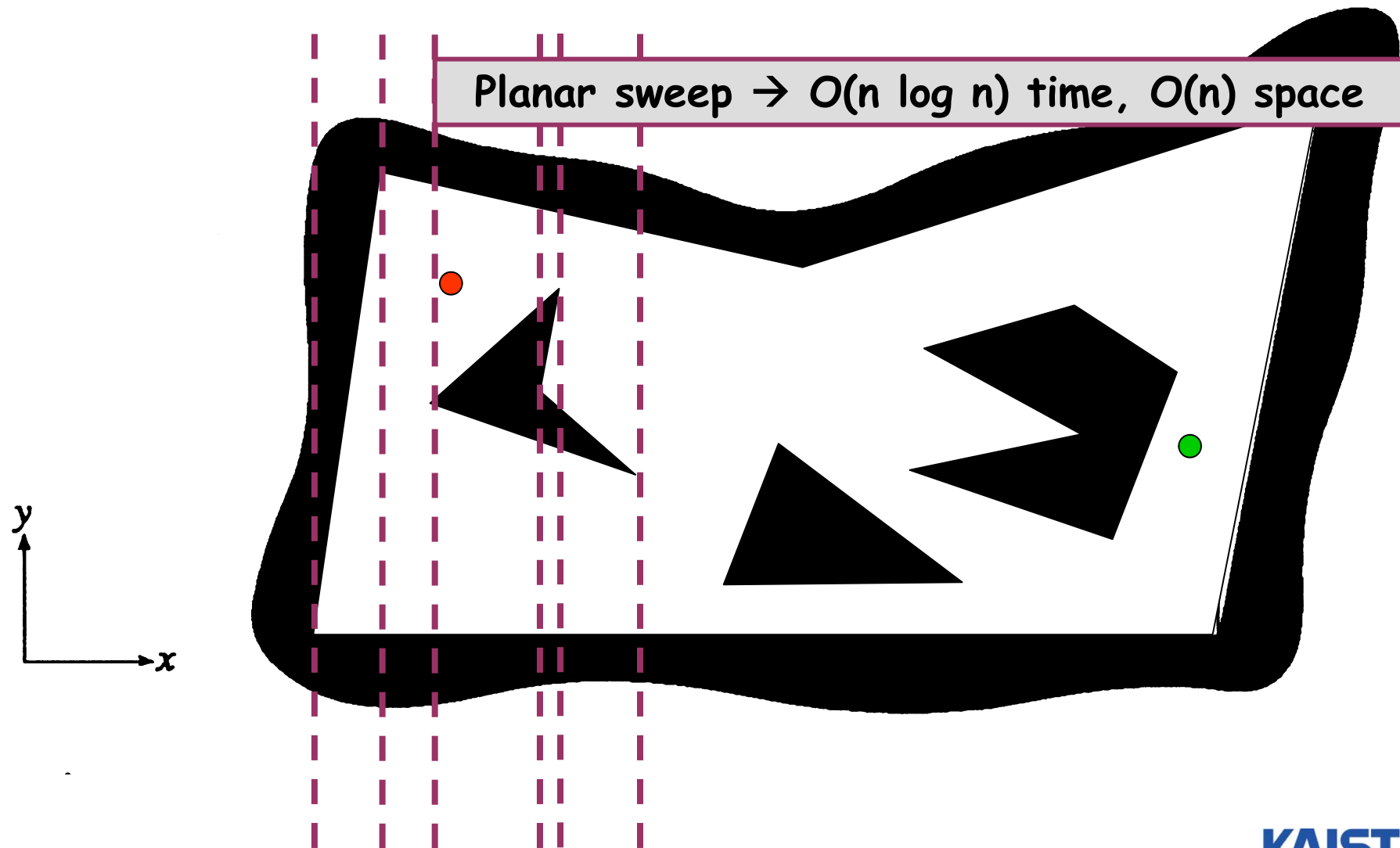
Trapezoidal Decomposition



Trapezoidal Decomposition



Trapezoidal Decomposition

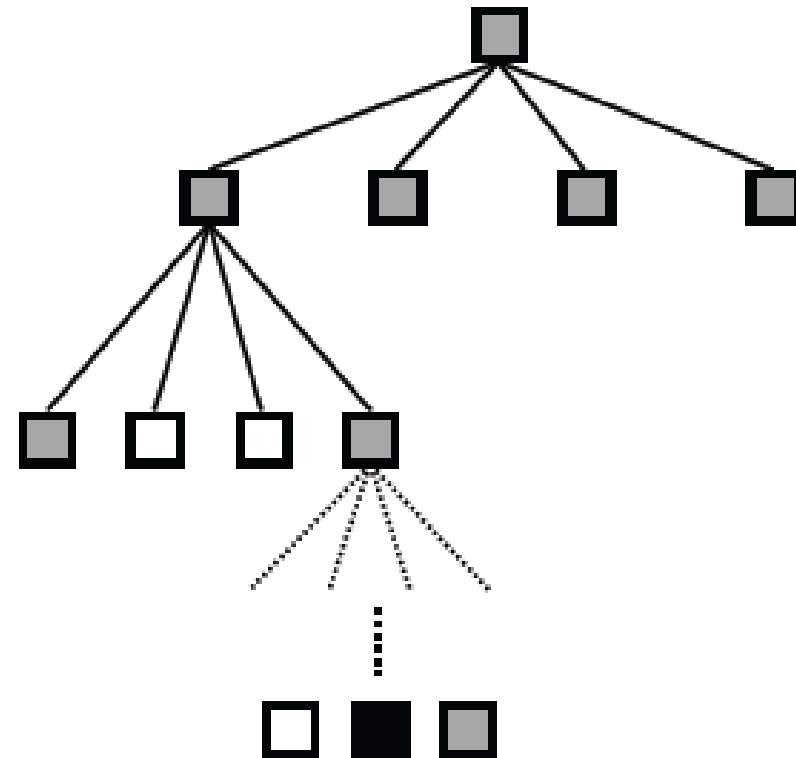
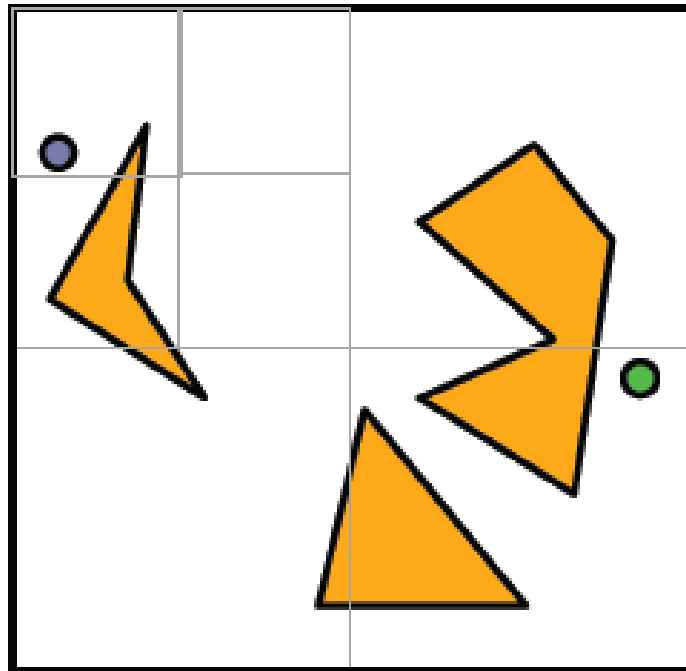


Cell-Decomposition Methods

- **Two classes of methods:**
 - Exact and approximate cell decompositions

- **Approximate cell decomposition**
 - The free space F is represented by a collection of non-overlapping cells whose union is contained in F
 - Cells usually have simple, regular shapes (e.g., rectangles and squares)
 - Facilitates hierarchical space decomposition

Quadtree decomposition

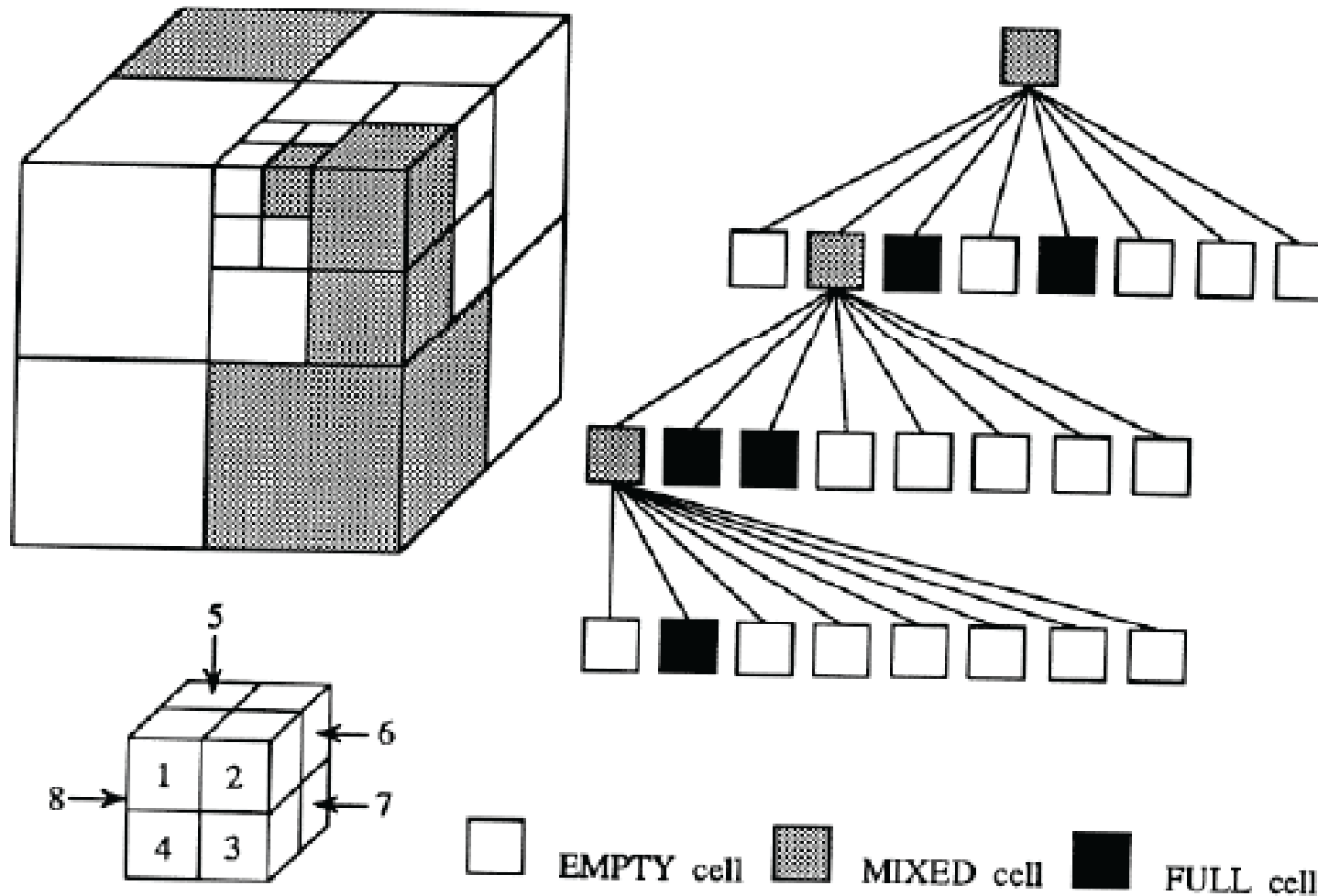


□ empty

■ mixed

■ full

Octree decomposition



Sketch of Algorithm

1. Decompose the free space F into cells
2. Search for a sequence of **mixed** or **free** cells that connect that initial and goal positions
3. Further decompose the mixed
4. Repeat 2 and 3 until a sequence of **free** cells is found

Classic Path Planning Approaches

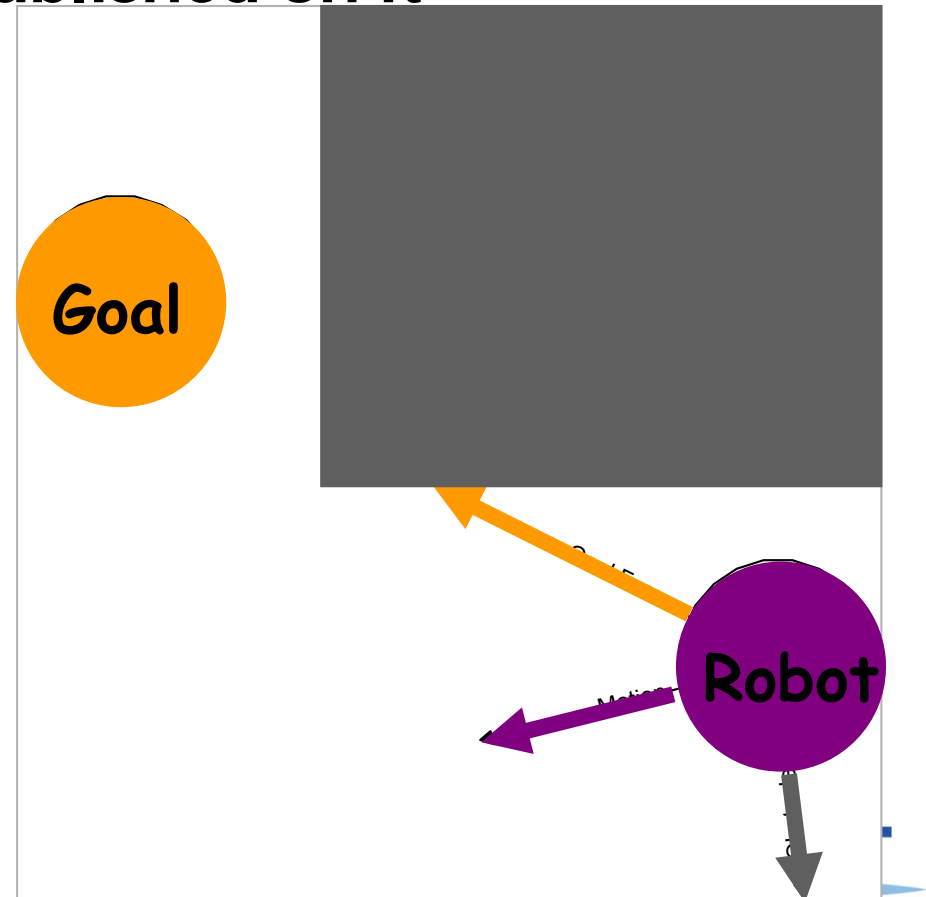
- **Roadmap**
 - Represent the connectivity of the free space by a network of 1-D curves
- **Cell decomposition**
 - Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells
- **Potential field**
 - Define a function over the free space that has a global minimum at the goal configuration and follow its steepest descent

Potential Field Methods

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

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

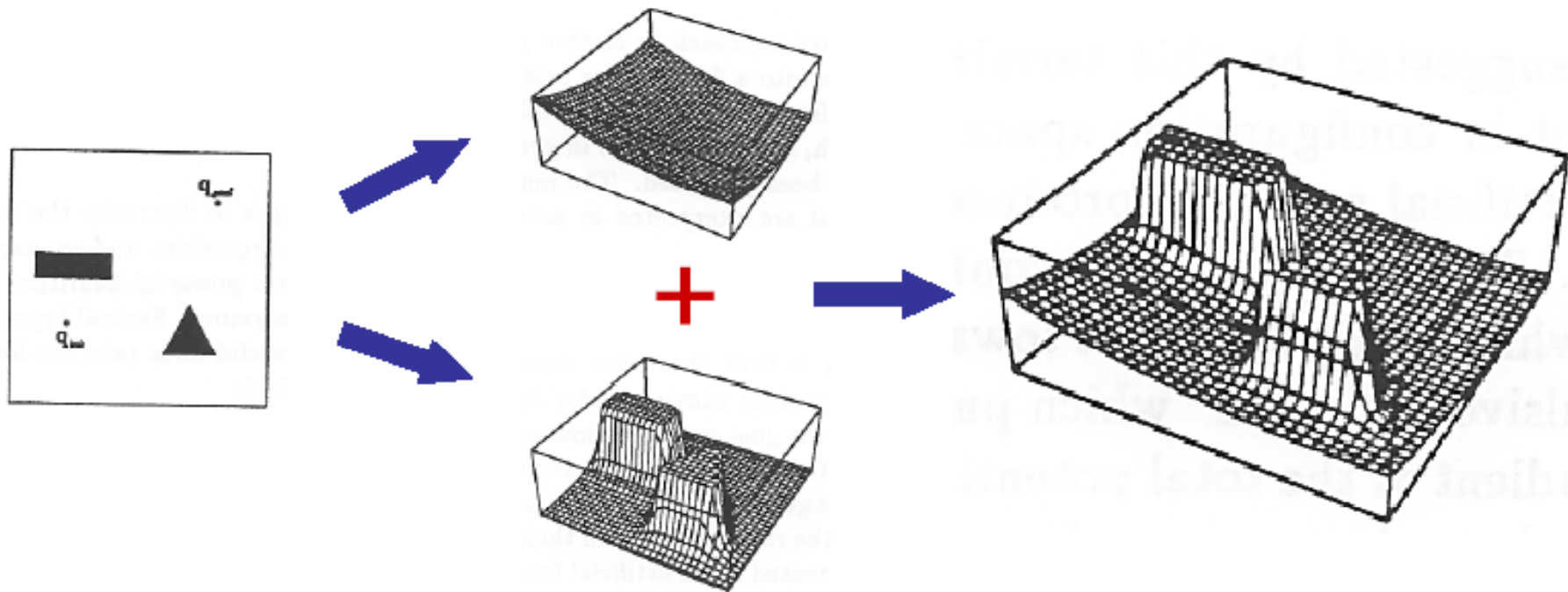
$$F_{Obstacle} = \begin{cases} \eta \left(\frac{1}{\rho} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2} \frac{\partial \rho}{\partial x} & \text{if } \rho \leq \rho_0, \\ 0 & \text{if } \rho > \rho_0 \end{cases}$$



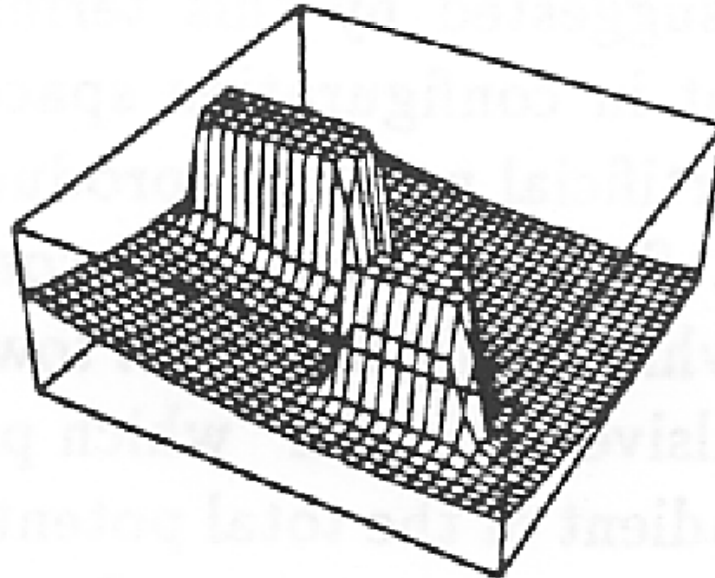
Potential Field

- A scalar function over the free space
- To navigate the robot applies a force proportional to the negated gradient of the potential field
- A navigation function is an ideal potential field that
 - Has global minimum at the goal
 - Has no local minima
 - Grows to infinity near obstacles
 - Is smooth

Attractive and Repulsive fields



Local Minima



- **What can we do?**
 - **Escape from local minima by taking random walks**
 - **Build an ideal potential field that does not have local minima**

Sketch of Algorithm

- Place a regular grid G over the configuration space
- Compute the potential field over G
- Search G using a best-first algorithm with potential field as the heuristic function

Question

- Can such an ideal potential field be constructed efficiently in general?

Completeness

- A **complete** motion planner always returns a solution when one exists and indicates that no such solution exists otherwise
 - Is the visibility algorithm complete? Yes
 - How about the exact cell decomposition algorithm and the potential field algorithm?

Class Objectives were:

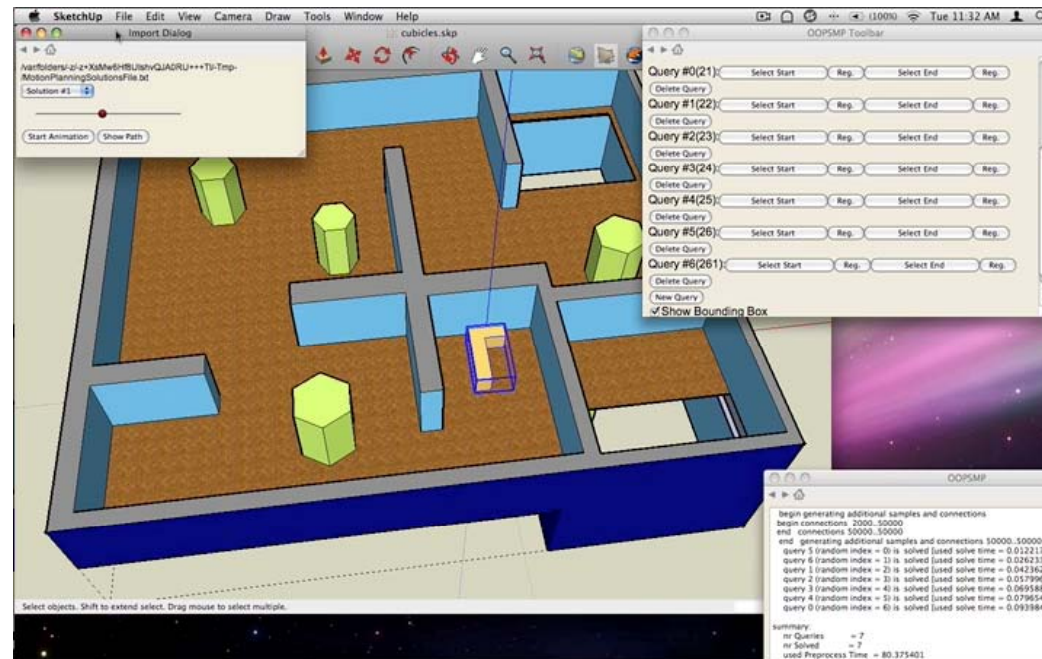
- Motion planning framework
- Classic motion planning approaches

Homework for Every Class

- **Go over the next lecture slides**
- **Come up with one question on what we have discussed today and submit at the beginning of the next class**
 - **0 for no questions**
 - **2 for typical questions**
 - **3 for questions with thoughts**
 - **4 for questions that surprised me**

Homework

- Install [Open Motion Planning Library \(OMPL\)](#)
- Create a scene and a robot
- Find a collision-free path and visualize the path



Homework

- Deadline: 11:59pm, Sep.-27 (Tue.)
- Delivery: send an email to TA (bluekdct@gmail.com) that contains:
 - An image that shows a scene with a robot, with a computed path

Conf. Deadline

- **ICRA**
 - Sep., 2011
- **IROS**
 - Mar., 2012



Next Time....

- Configuration spaces