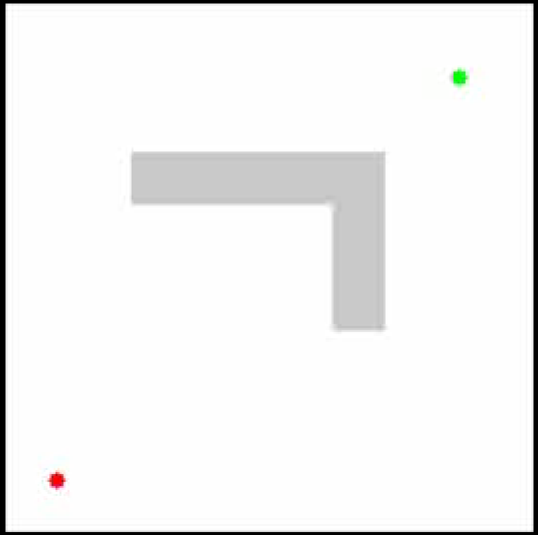
Motion Planning I – Point Robots

Last time…

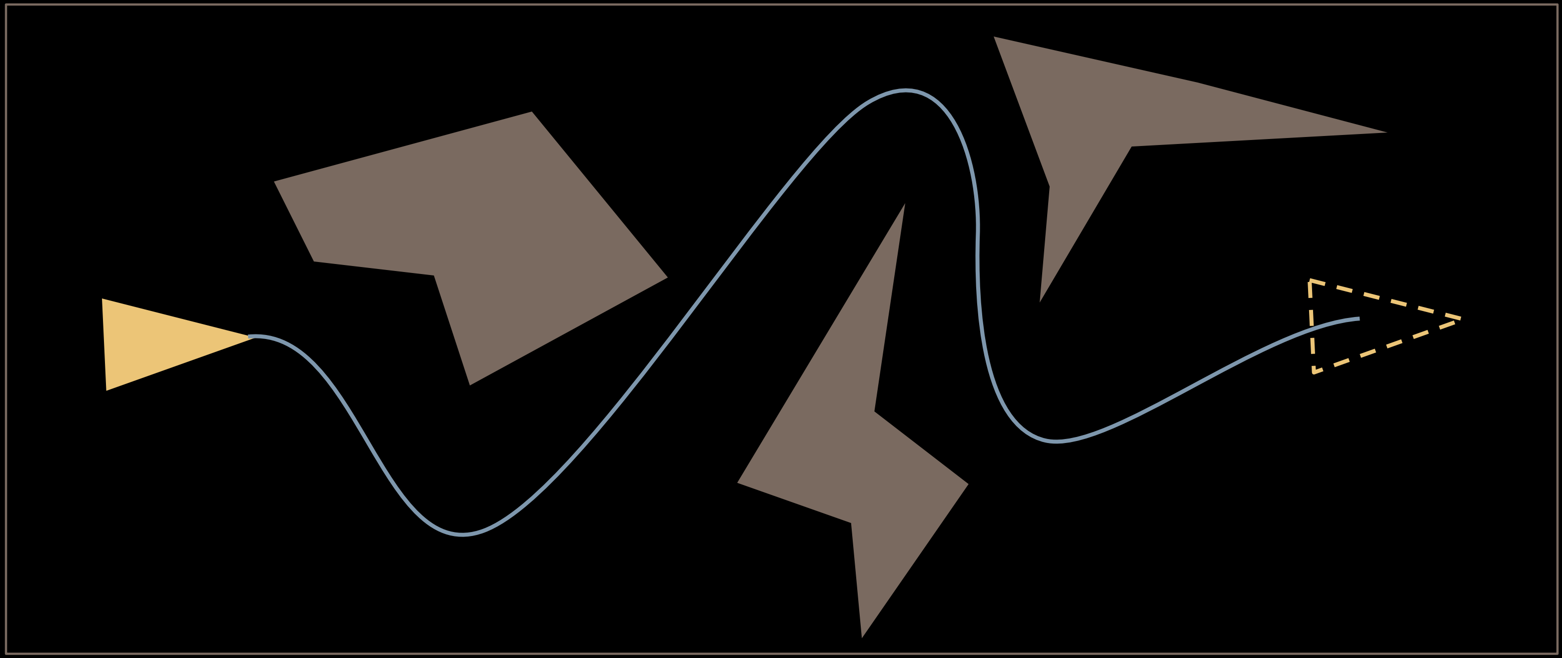
- We saw how to search for a path in a graph



- Today we will frame the problem of searching for a path for a robot
- We'll use some graph search methods to solve it

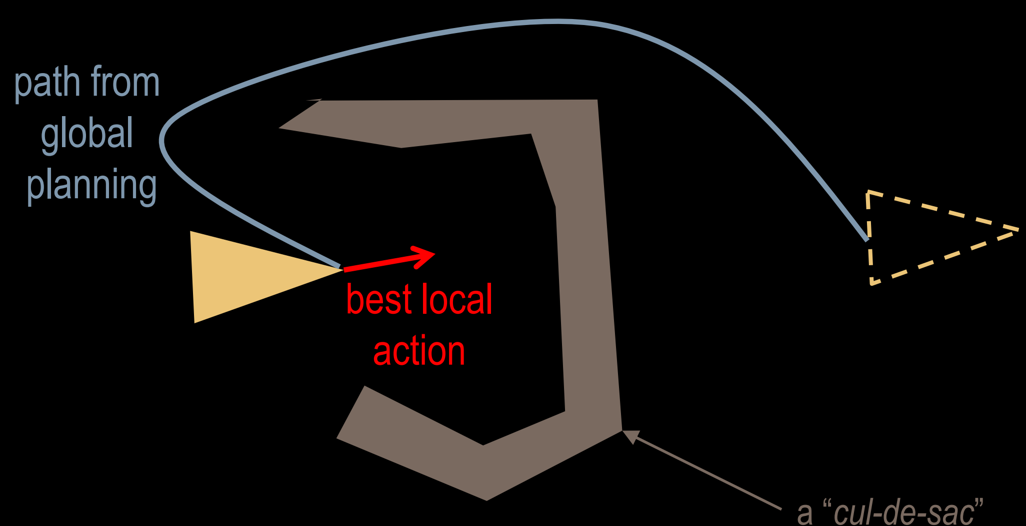
What is motion planning?

- The automatic generation of motion

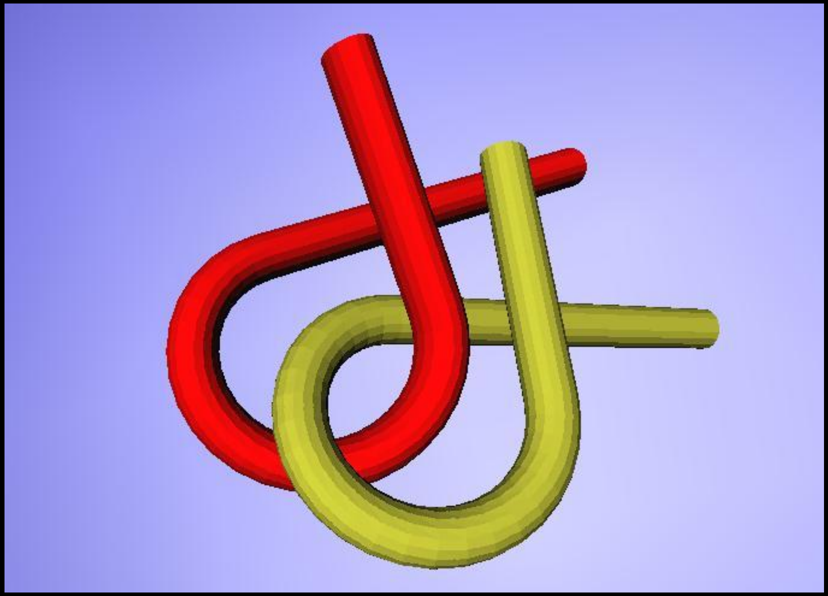


Why Motion Planning Instead of Obstacle Avoidance?

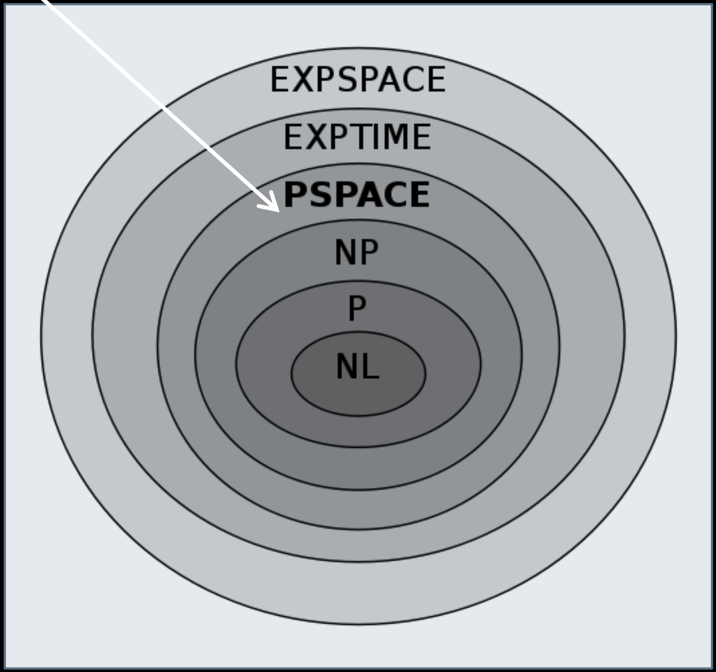
- Path planning
  - low-frequency, time-intensive search method for global finding of a (optimal) path to a goal
- Obstacle avoidance (aka "local navigation")
  - fast, reactive method with local time and space horizon
- Distinction: Global vs. local reasoning



Is motion planning hard?



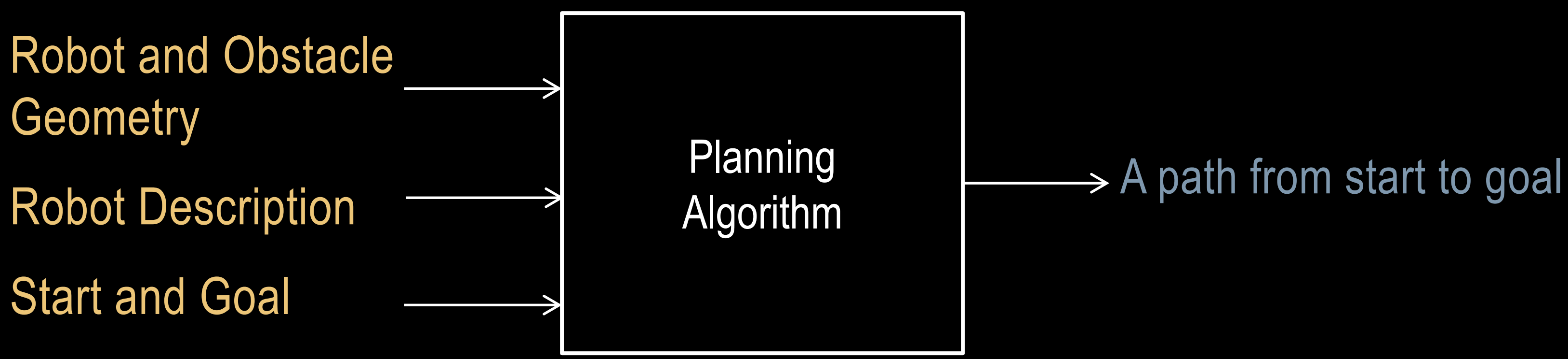
Basic Motion
Planning Problems



Complexity

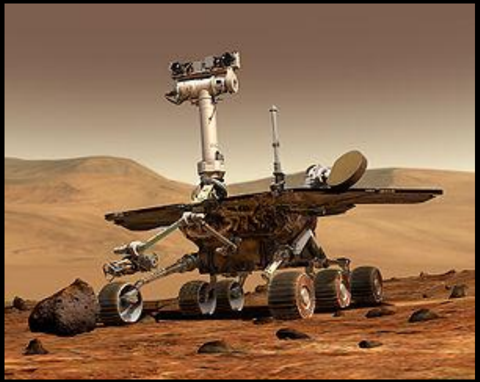
Basic Problem Statement

- *Automatically compute a path for an object/robot that does not collide with obstacles.*

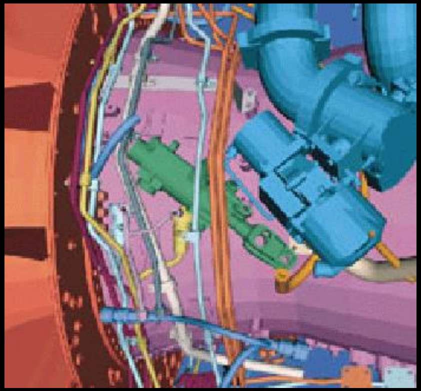


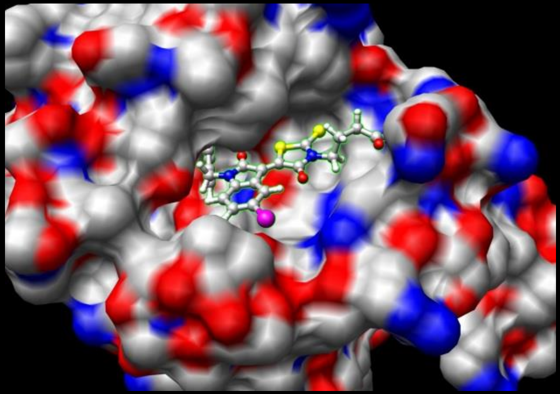
What can motion planning do?

- Automatically generate motion



- Automatically validate





Applications: Mobile Robots

Roomba iCreate

Mars Rovers

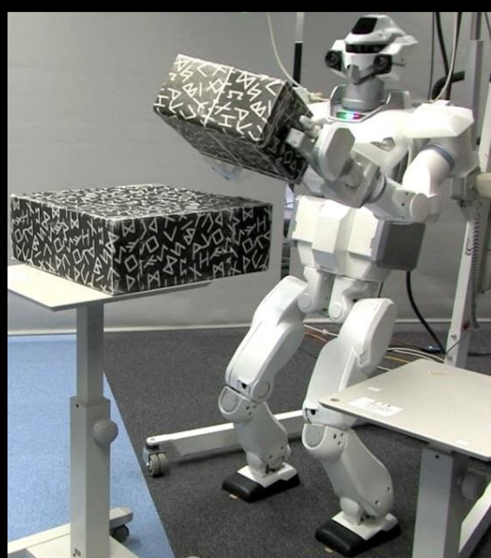
DARPA Urban Challenge

Google Self-Driving Car

Applications: Robotic Manipulation

Factory Automation

Personal Robots

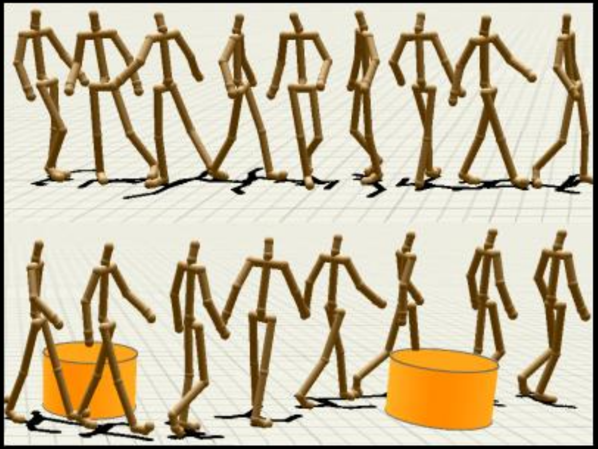


Humanoid Robots

Personal Robots

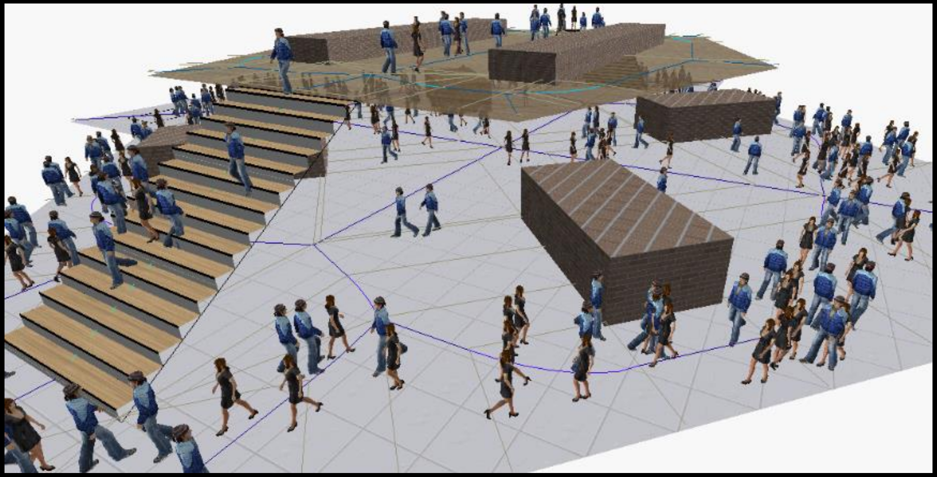
Applications: Computer Games/Graphics

Path Finding in Games



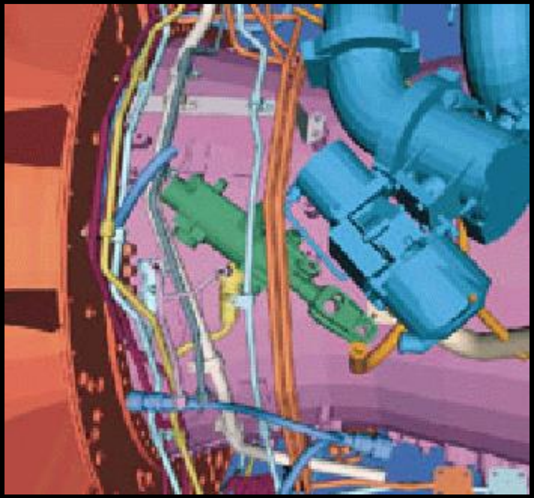
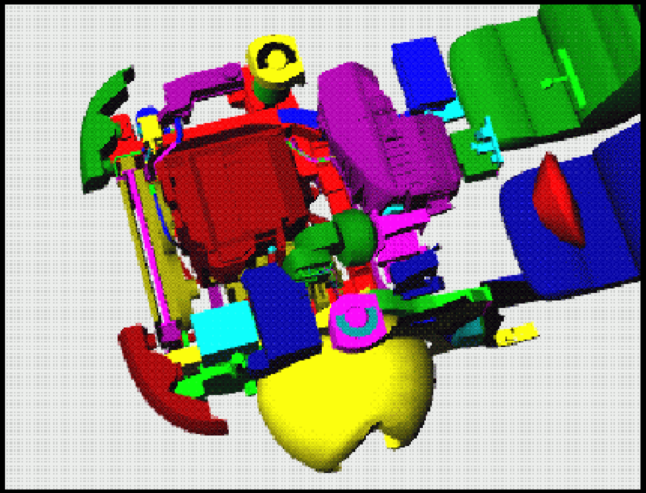
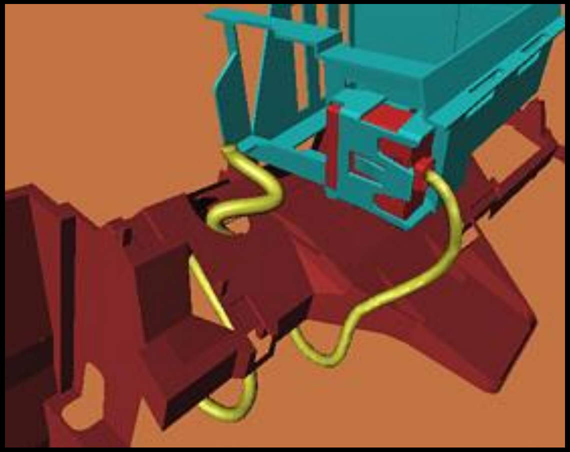
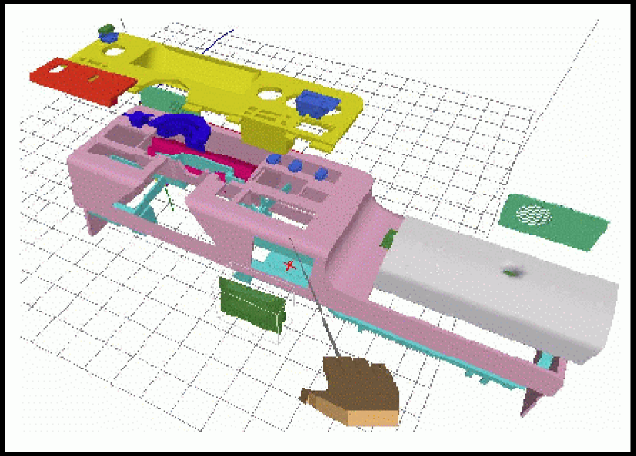
Character Animation

Retargeting Motion Capture

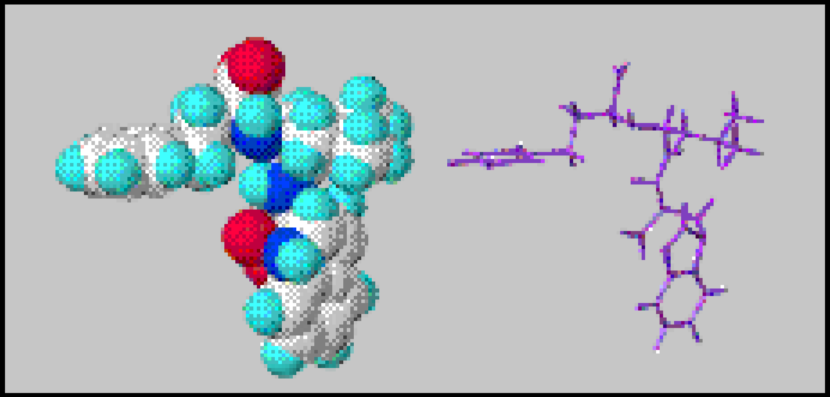


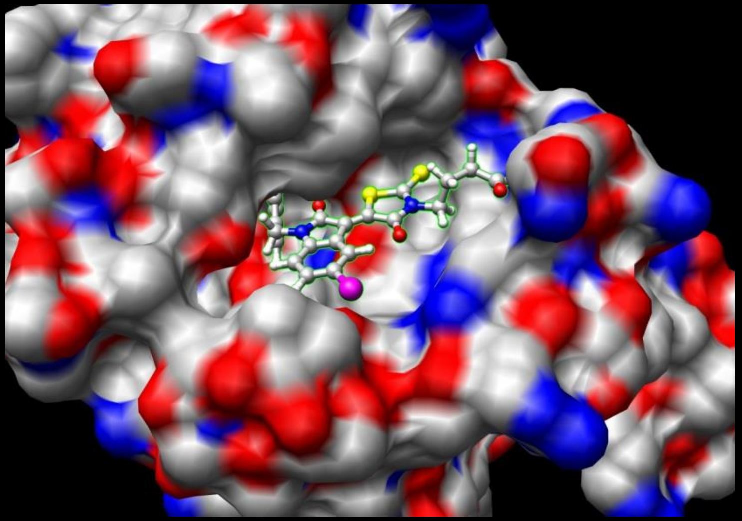
Animation of Crowds

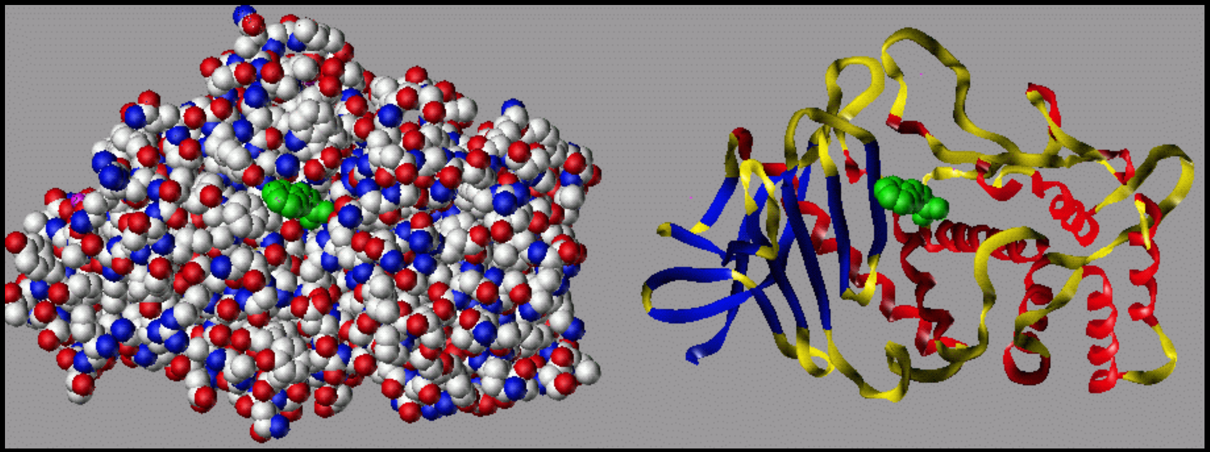
Applications: Assembly Planning

Applications: Computational Biology

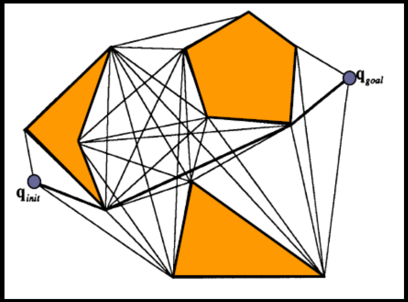




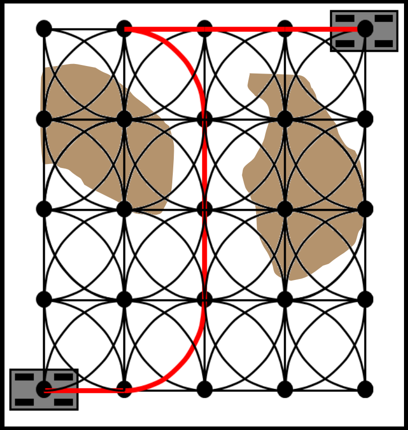


Approaches

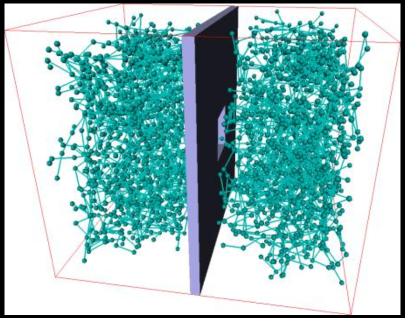
- Exact algorithms
  - Either find a solution or prove none exists
  - Very computationally expensive
  - Unsuitable for high-dimensional spaces



- Discrete Search
  - Divide space into a grid, use A* to search
  - Good for vehicle planning
  - Unsuitable for high-dimensional spaces



- Sampling-based Planning
  - Sample the C-space, construct path from samples
  - Good for high-dimensional spaces
  - Weak completeness and optimality guarantees



What matters?

- Motion planning algorithms are judged on
  - Completeness
  - Optimality
  - Speed (AKA efficiency)
  - Generality
- These vary in importance depending on the application

What matters: Completeness

- Will the algorithm solve all solvable problems?
- Will the algorithm return no solution for unsolvable problems?
- What if the algorithm is probabilistic?

- For what application(s) is completeness very important?
- For what application(s) is completeness not important?

What matters: Optimality

- Will the algorithm generate the shortest path?
- Will the algorithm generate the least-cost path (for an arbitrary cost function)?
- Do we need optimality or is feasibility enough?

- For what application(s) is optimality very important?
- For what application(s) is optimality not important?

What matters: Speed (AKA Efficiency)

- How long does it take to generate a path for **real-world** problems?
- How does the run-time scale with dimensionality of the problem and complexity of models?
- Is there a quality vs. computation time tradeoff?

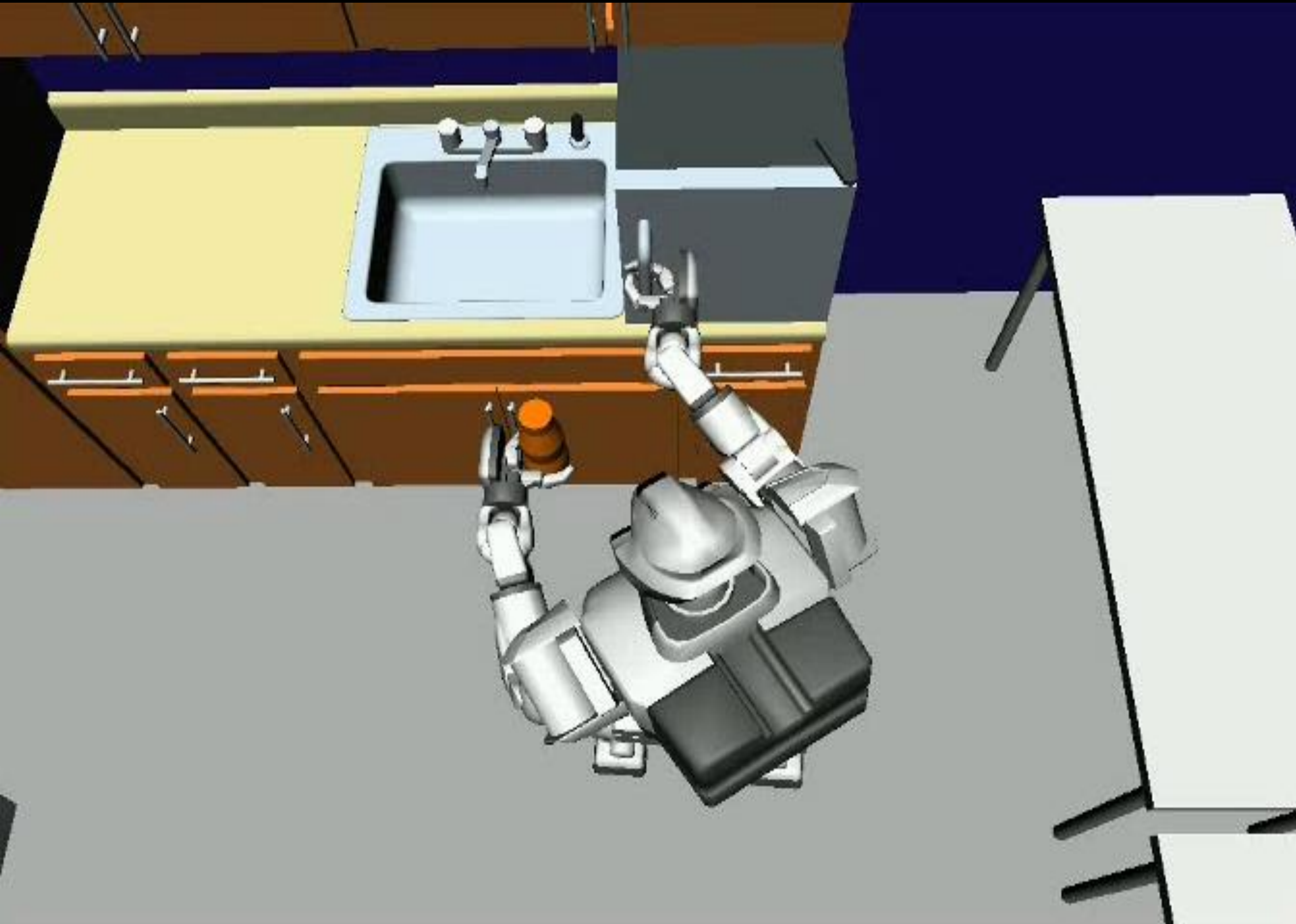
- For what application(s) is speed very important?
- For what application(s) is speed not important?

What matters: Generality

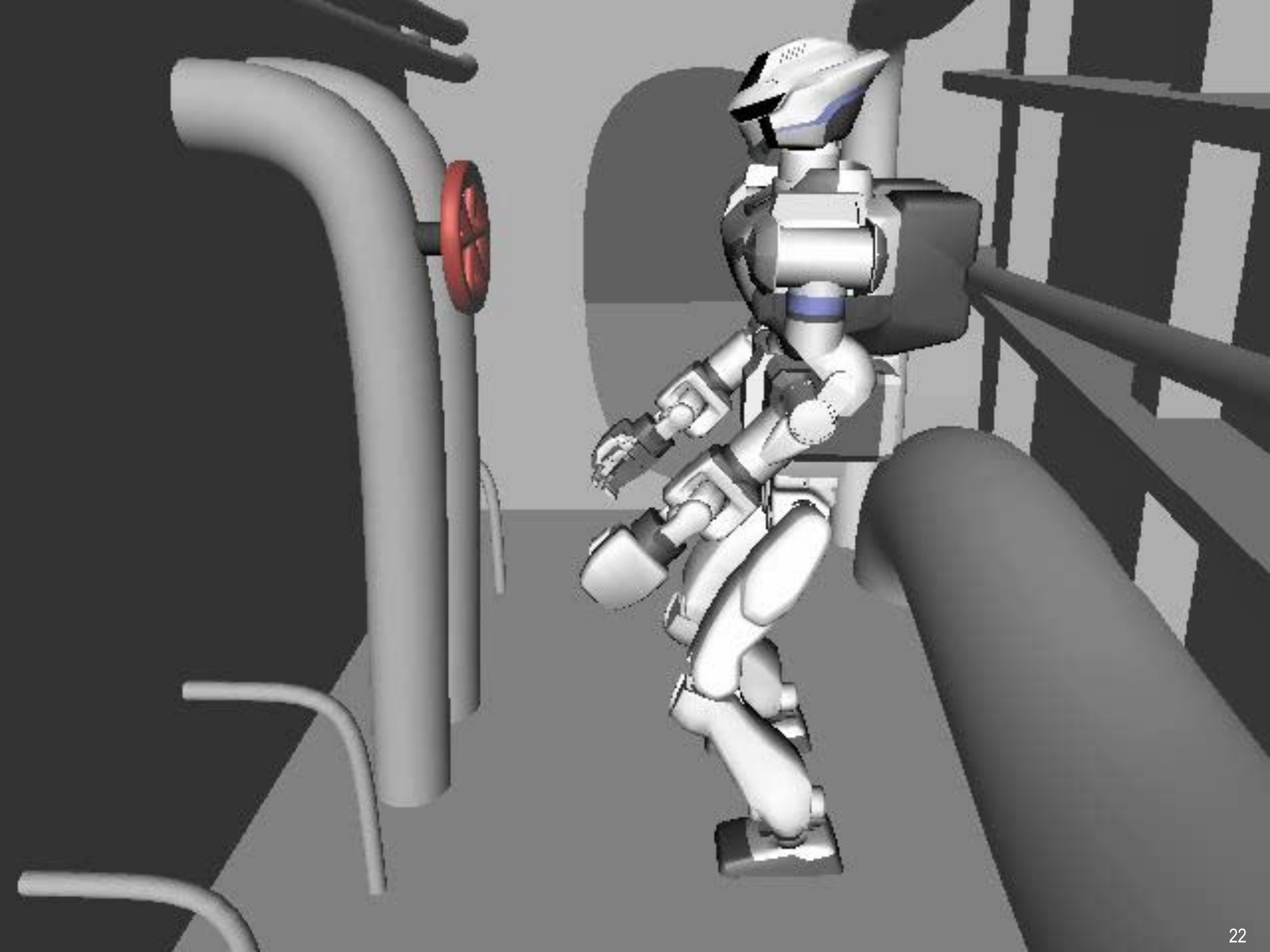
- Generality is the vaguest criterion, but often the most important
- What types of problems can it solve?
- What types of problems can't it solve?

- For what application(s) is generality very important?
- For what application(s) is generality not important?


w = 4.98kg



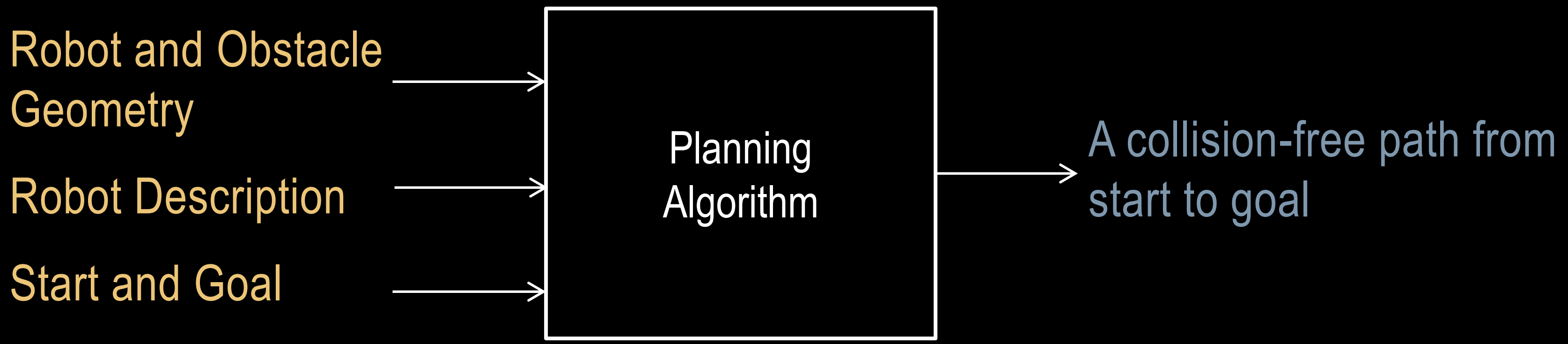

x3



Path Planning for Point Robots

Basic Problem Statement

- *Automatically compute a path for an object/robot that does not collide with obstacles.*



- Start simple:
  - The robot is a point that can move freely
  - The environment is 2D with polygonal obstacles

Methods

- Visibility graph
- Cell decomposition
- Potential fields

Framework

continuous representation

(configuration space formulation)

↓

discretization

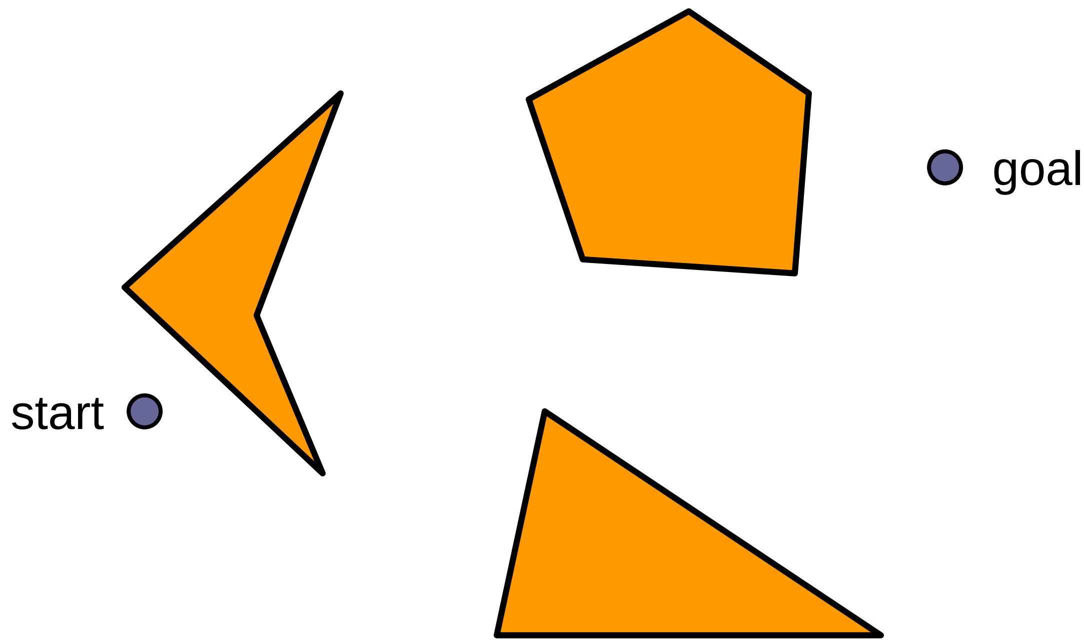
(random sampling, processing critical geometric events)

↓

graph searching

(breadth-first, best-first, A*)

Continuous Representation



Framework

continuous representation

↓

discretization

(random sampling, processing critical geometric events)

↓

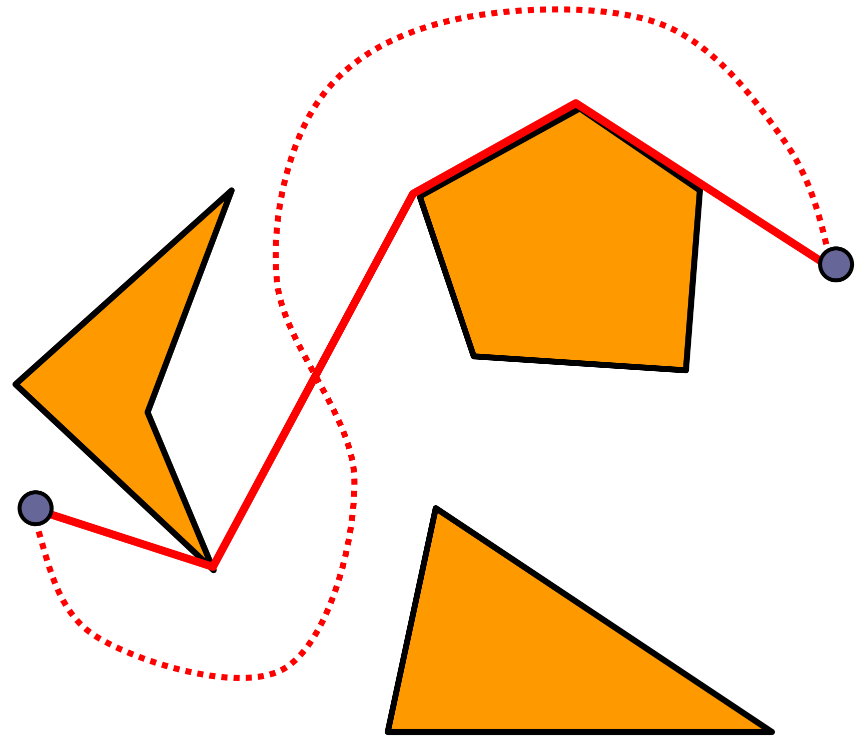
graph searching

(breadth-first, best-first, A*)

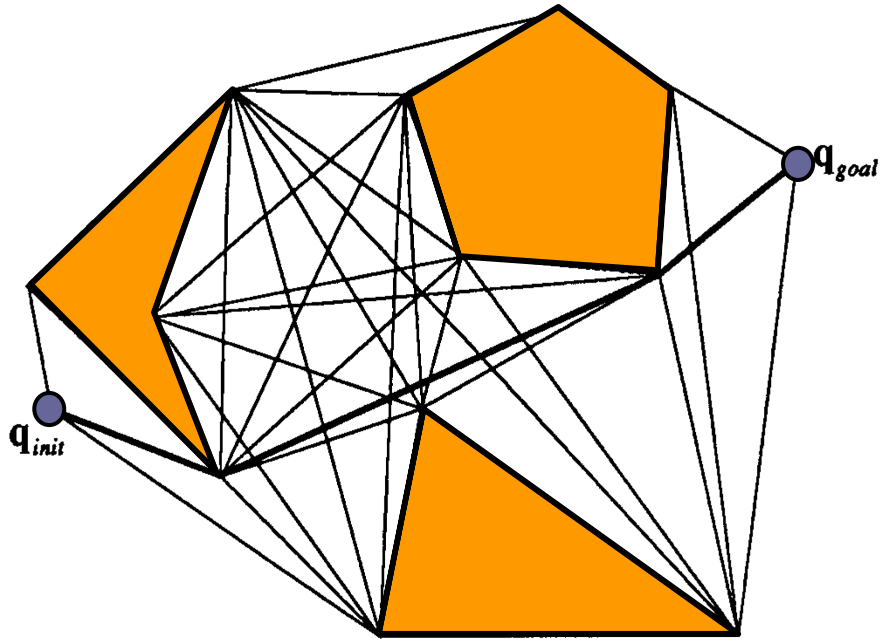
Visibility graph method

- **Observation**: If there is a collision-free path between two points, then there is a piece-wise linear path that bends only at the obstacles vertices.

- Why?
  Any collision-free path can be transformed into a piece-wise linear path that bends only at the obstacle vertices.



What is a visibility graph?



- A **visibility graph** is a graph such that
  - Nodes: q_{init}, q_{goal}, or an obstacle vertex.
  - 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.

A simple algorithm for building visibility graphs

Input: q_{init}, q_{goal}, 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.
```

Computational 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
  - Optimal algorithm $O(n^2)$ time
- $O(n^2)$ space

Framework

continuous representation

(configuration space formulation)

↓

discretization

(random sampling, processing critical geometric events)

↓

graph searching

(breadth-first, best-first, A*)

Which method from last lecture should we use?

Framework

continuous representation

↓

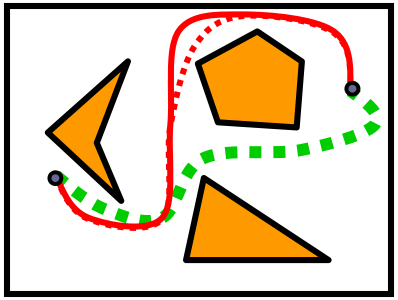
discretization

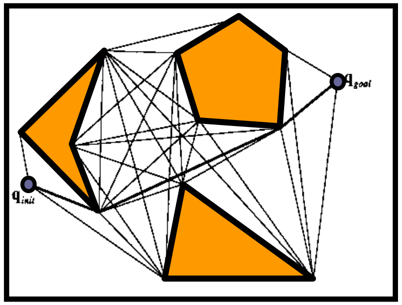
construct visibility graph

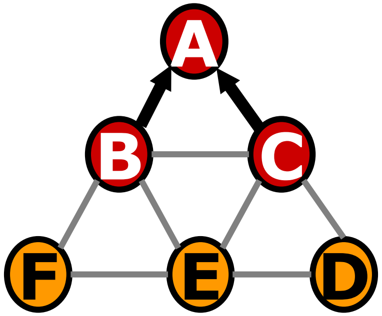
↓

graph searching

A*







Computational efficiency

- Running time $O(n^3)$
  - Compute the visibility graph
  - Search the graph
  - An optimal $O(n^2)$ time algorithm exists.
- Space $O(n^2)$
- **Can we do better?**

Break

Classic path planning approaches

- **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 potential function over the free space that has a global minimum at the goal and follow the steepest descent of the potential function

Classic path planning approaches

- **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 potential function over the free space that has a global minimum at the goal and follow the steepest descent of the potential function

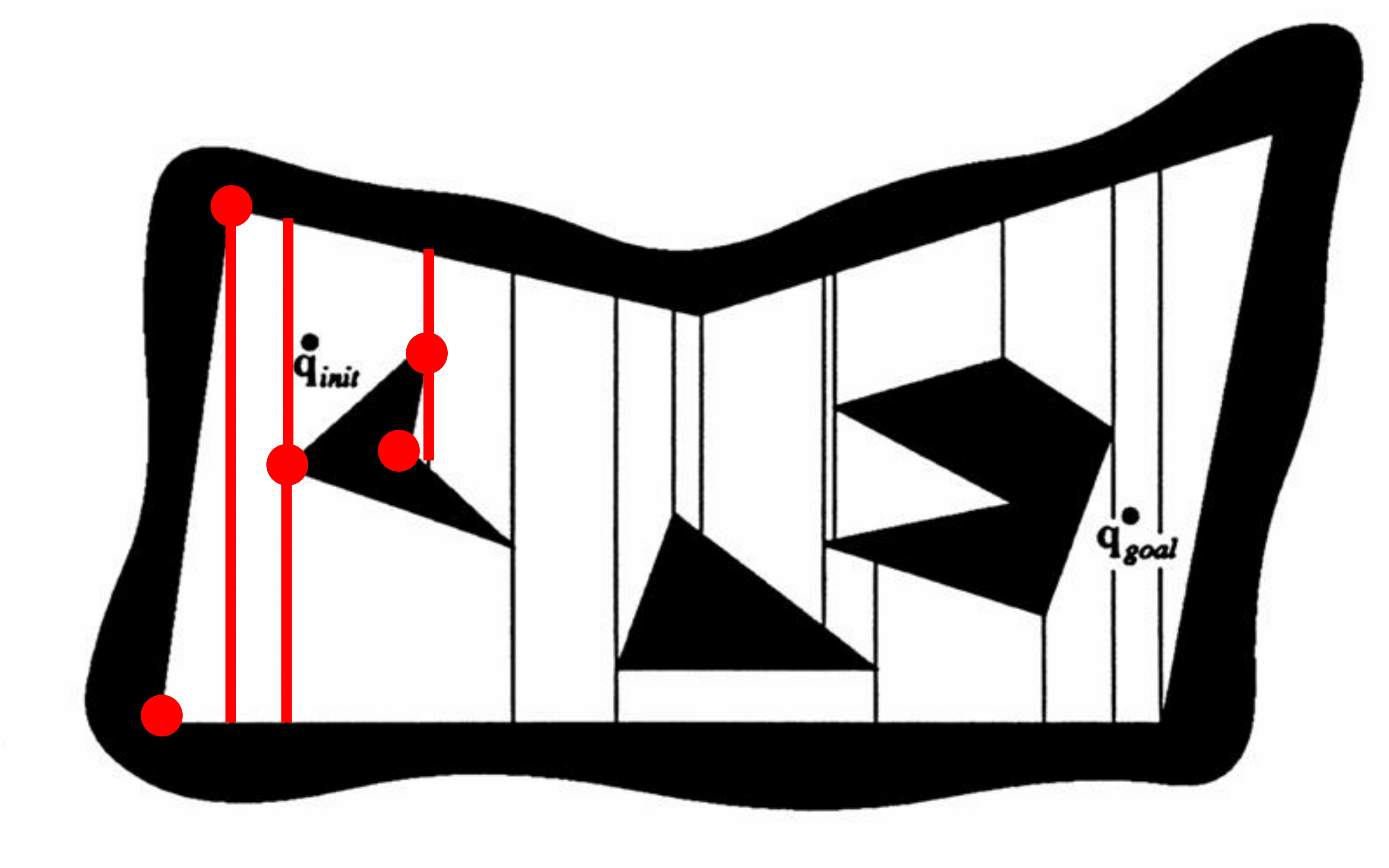
Cell-decomposition methods

- **Exact cell decomposition**

  The free space F is represented by a collection of non-overlapping simple cells whose union **is exactly** F.

  - Examples of cells: trapezoids, triangles

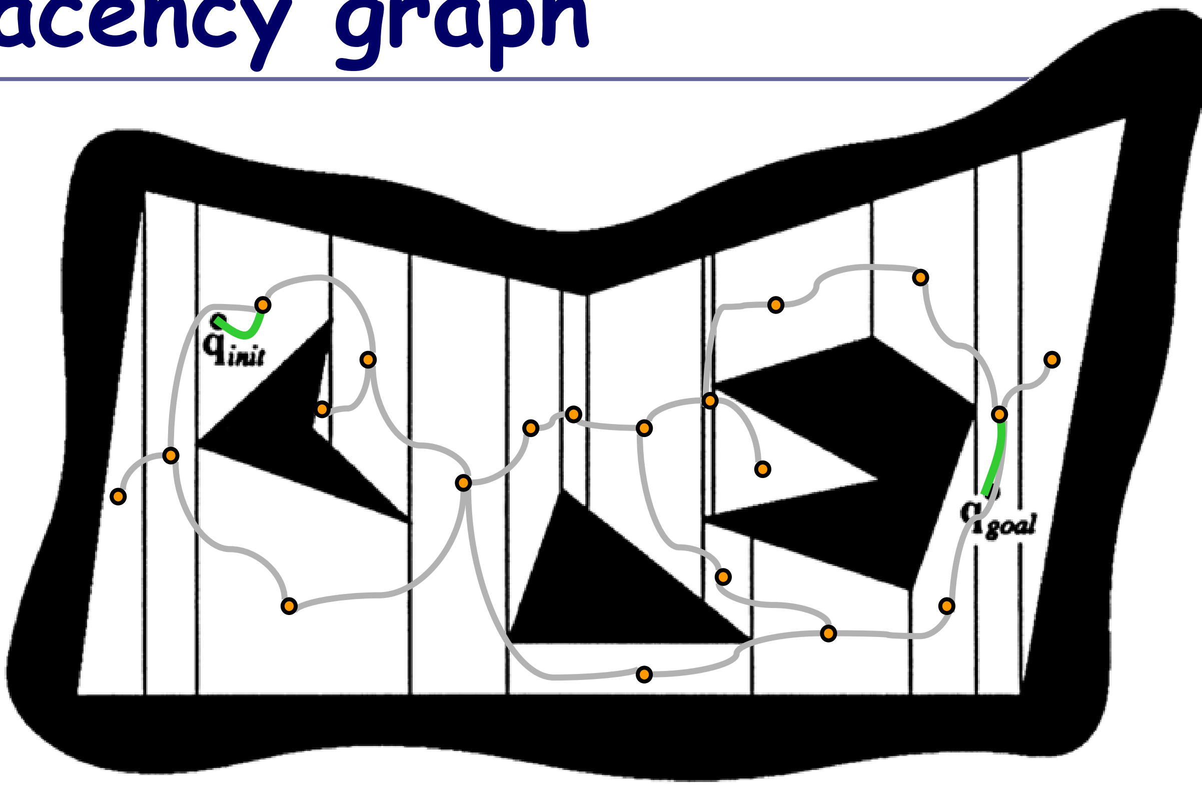
Trapezoidal decomposition



Computational efficiency

- Running time $O(n \log n)$ by planar sweep
- Space $O(n)$
- Mostly for 2-D environments

Adjacency graph



- **Nodes**: cells
- **Edges**: There is an edge between every pair of nodes whose corresponding cells are adjacent.
- A sequence of edges can be converted into a continuous path
  - This is easy to do when cells are convex. Why?

Framework

continuous representation

↓

discretization

construct an adjacency graph of the cells

↓

graph searching

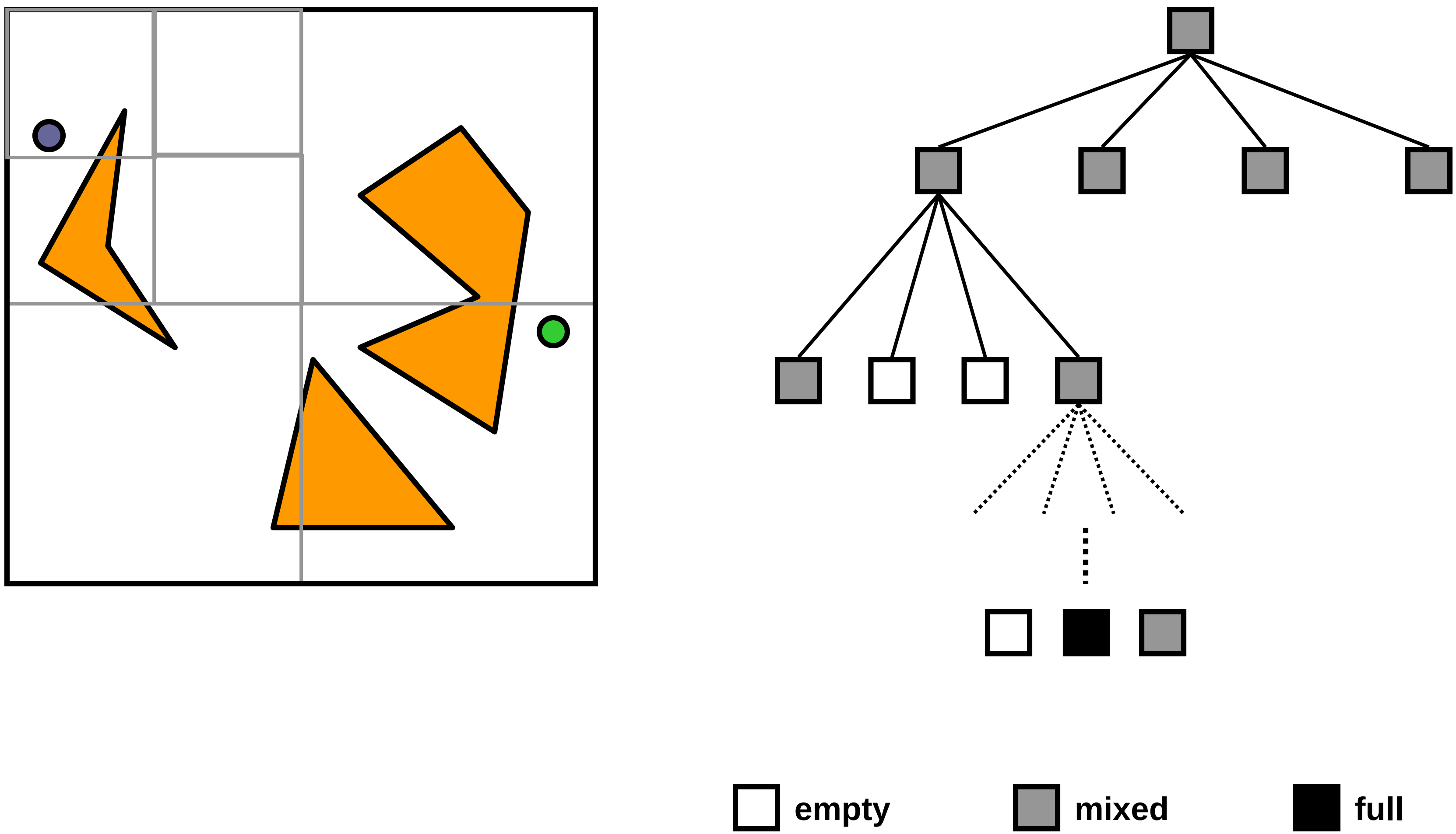
search the adjacency graph

Cell-decomposition methods

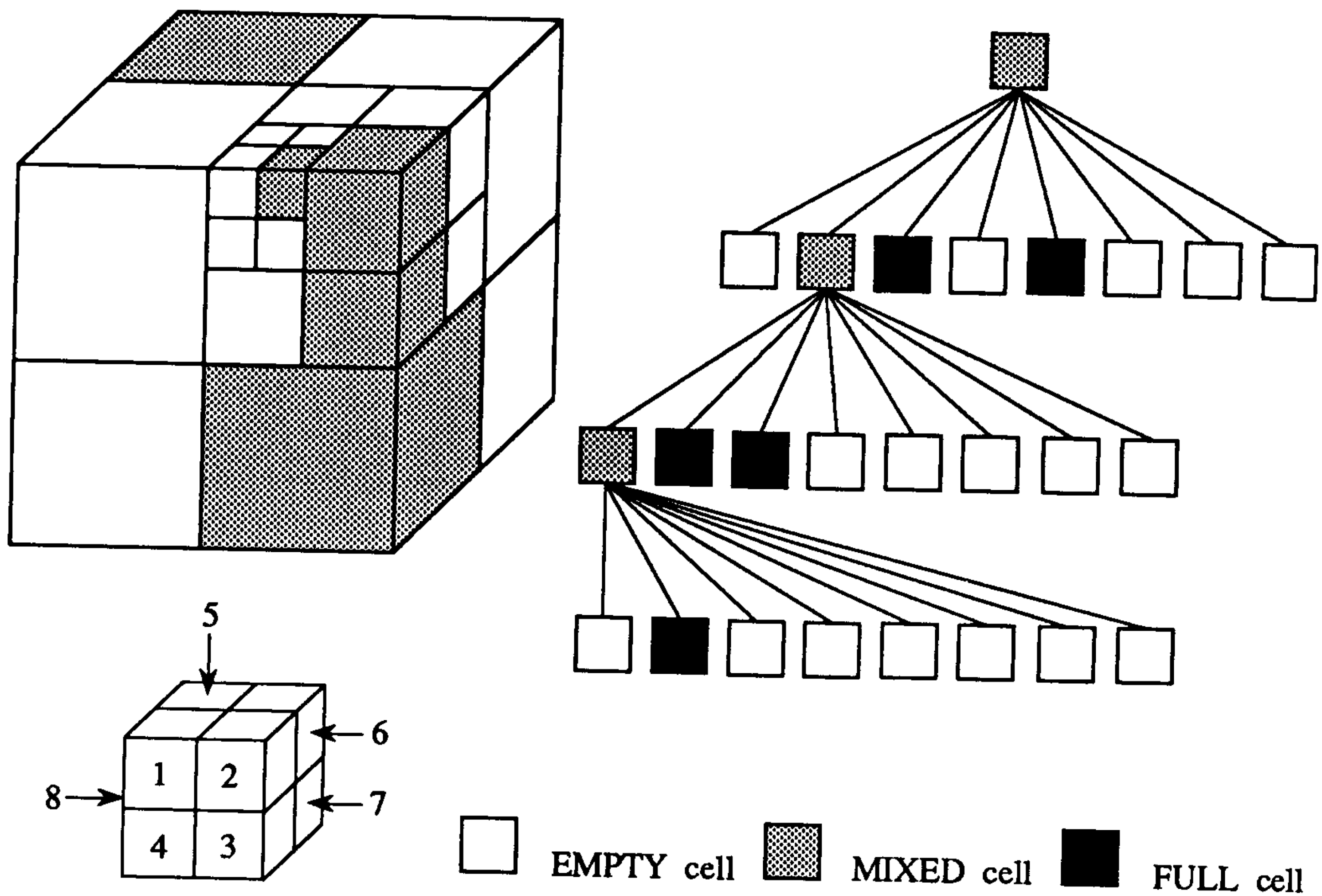
- Exact cell decomposition
- **Approximate cell decomposition**

  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, squares.
  - Facilitate hierarchical space decomposition

Quadtree decomposition



Octree decomposition



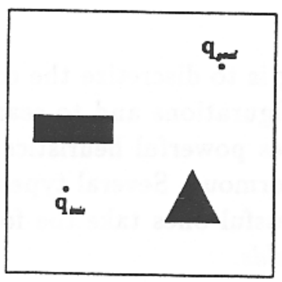
Sketch of the algorithm

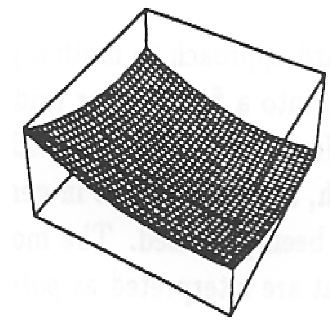
1. Decompose the free space F into cells.
2. Search for a sequence of **mixed** or **free** cells that connect the initial and goal positions.
3. Further decompose the mixed.
4. Repeat (2) and (3) until a sequence of **free** cells is found.

Classic path planning approaches

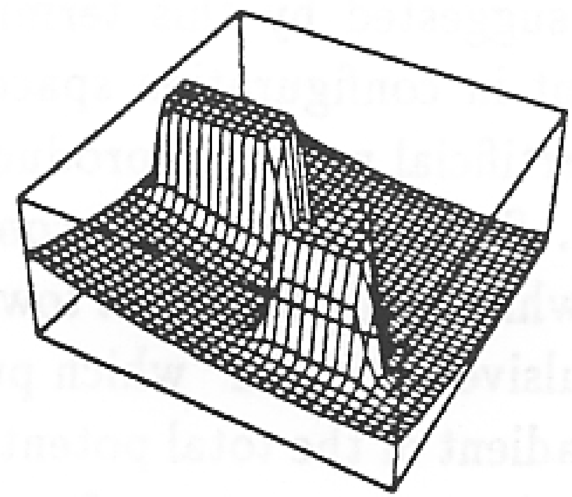
- **Cell decomposition**
  Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells
- **Potential field**
  Define a potential function over the free space that has a global minimum at the goal and follow the steepest descent of the potential function

Algorithm in pictures

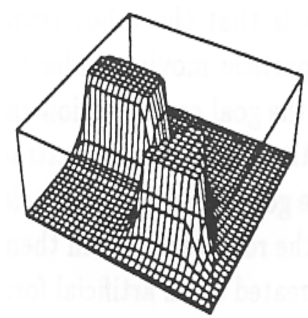




$$\phi_{\text{att}} = \frac{1}{2} k_{\text{att}} (x - x_{\text{goal}})^2$$



+



$$\phi_{\text{rep}} = \begin{cases} \dfrac{1}{2} k_{\text{rep}} \left(\dfrac{1}{\rho} - \dfrac{1}{\rho_0} \right)^2 & \text{if } \rho \le \rho_0, \\ 0 & \text{if } \rho > \rho_0 \end{cases}$$

Attractive & repulsive fields

$$F_{\text{att}} = -\nabla \phi_{\text{att}} = -k_{\text{att}}(x - x_{\text{goal}})$$

$$F_{\text{rep}} = -\nabla \phi_{\text{rep}} = \begin{cases} k_{\text{rep}} \left(\dfrac{1}{\rho} - \dfrac{1}{\rho_0} \right) \dfrac{1}{\rho^2} \dfrac{\partial \rho}{\partial x} & \text{if } \rho \leq \rho_0, \\ 0 & \text{if } \rho > \rho_0 \end{cases}$$

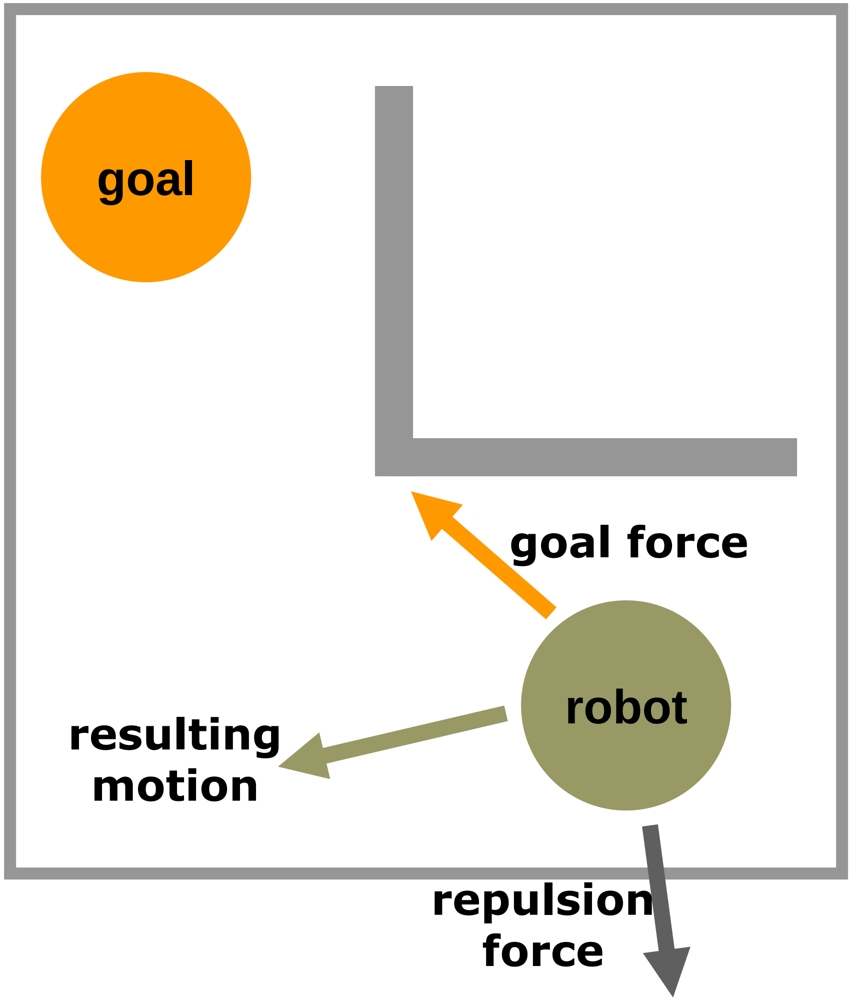
k_{att}, k_{rep} : positive scaling factors

x : position of the robot

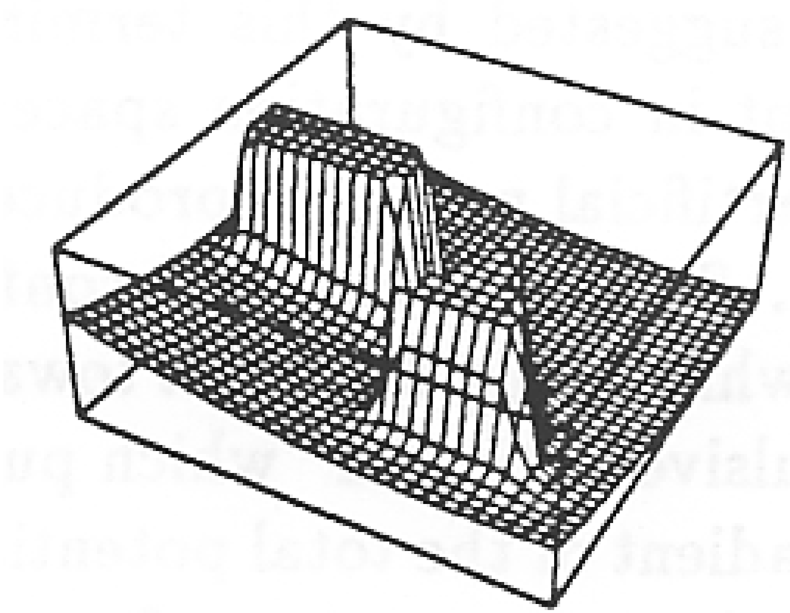
ρ : distance to the obstacle

ρ_0 : distance of influence

[Khatib, 1986]



Local minima



- What can we do?
  - Escape from local minima by taking random walks
  - Build an ideal potential field – navigation function – that does not have local minima
    - Computationally expensive 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 graph algorithm complete?
  - Is the exact cell decomposition algorithm complete?
  - Is the potential field algorithm complete?

Homework

- Read LaVallе Ch. 4.0-4.3