CSCE 452/752 Fall 2025

15. Motion planning

TEXAS A&M UNIVERSITY

AARRG

Introduction

Motion planning algorithms generate continuous paths through the robot's **configuration space**.

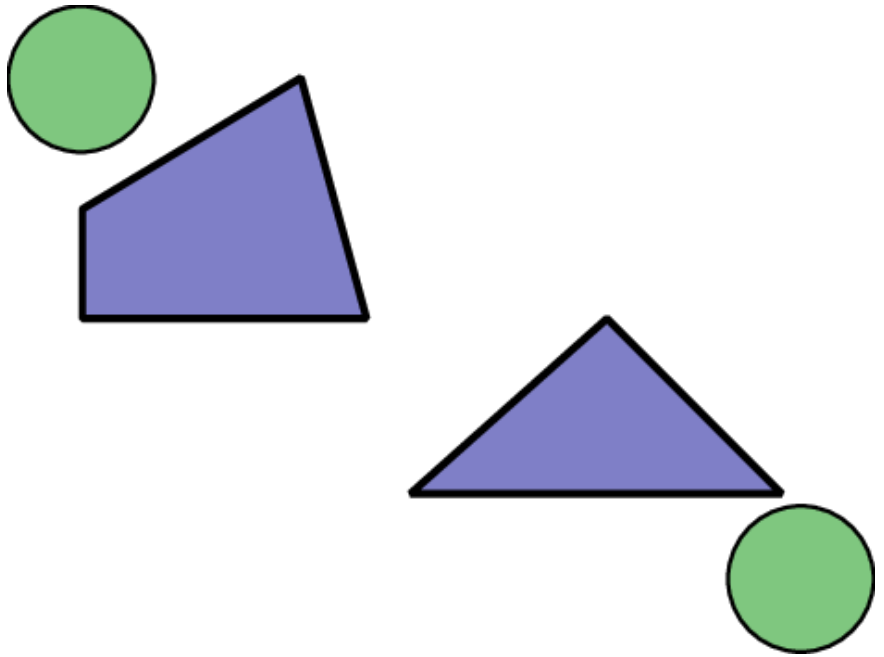
Overview:

- The configuration space allows us to treat the robot as a point.
- For a typical motion planning problem, it is impractical to attempt to represent the complete obstacle boundary explicitly.
- Instead, use algorithms based on **sampling**. Two well-known sampling based algorithms are PRM and RRT.

Configuration Space

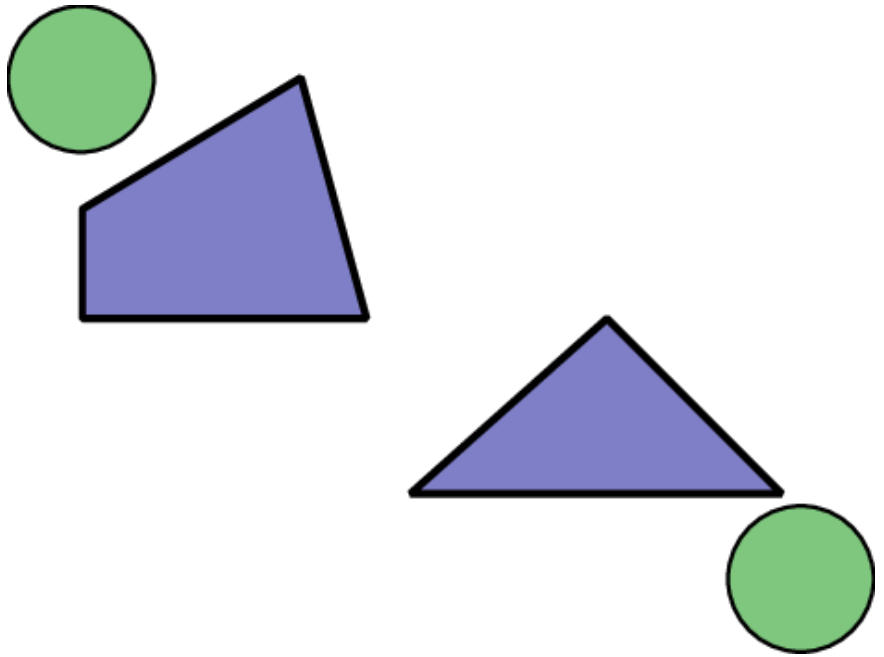
Motivation

Suppose we want to plan the motions for a circular robot amongst some known obstacles.



Intuition

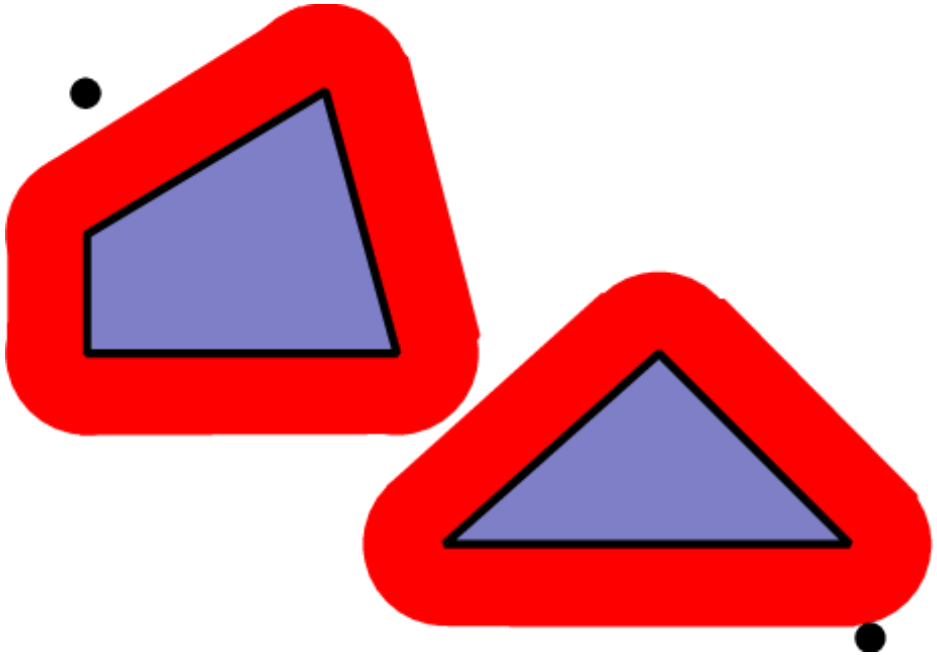
Informally, we want to shrink the robot down to a point and expand the obstacles by the same amount.



If we keep the center point out of the “expanded” obstacles, then the entire robot will stay out of the real workspace obstacles.

Intuition

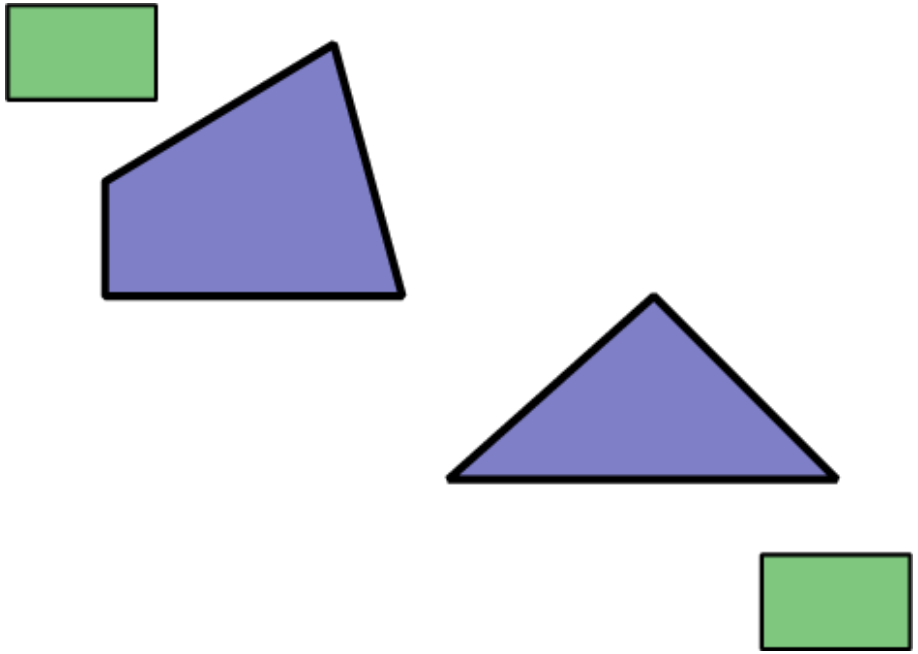
Informally, we want to shrink the robot down to a point and expand the obstacles by the same amount.



If we keep the center point out of the “expanded” obstacles, then the entire robot will stay out of the real workspace obstacles.

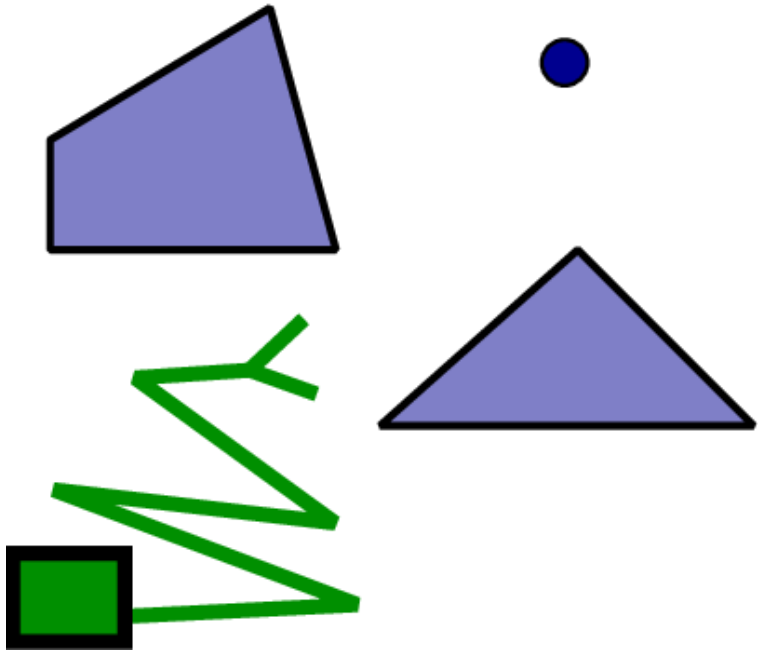
...but this intuition can only take us so far.

What if the robot is not a circle?



...but this intuition can only take us so far.

What if we want to plan motions for a multi-link manipulator to grasp a distant object?



Definition

It looks like we'll need to think about this more carefully.

> The **configuration space** (C-space) $\mathcal{C}$ of a system contains one point for each combination of values for the robot's position, orientation, and internal joint positions.

Comments:

- The configuration space is a special kind of state space.
- The configuration space should be a **topological space**. Informally, this means that it is meaningful to talk about the “neighborhood” of a configuration.
- Many algorithms require the configuration space to be a **metric space**, which means that there is some reasonable definition of **distance** between pairs of configurations.

C-spaces for our examples

To find the configuration space for a given system, think about what parameters are needed to describe the robot's situation.

- Circular robot: We need just the x and y coordinates, so

$$\mathcal{C} = \mathbb{R} \times \mathbb{R}.$$

- Rectangular robot: Orientation is important, so

$$\mathcal{C} = \mathbb{R} \times \mathbb{R} \times S^1 = SE(2).$$

- Robotic arm: Five revolute joints, so

$$\mathcal{C} = S^1 \times S^1 \times S^1 \times S^1 \times S^1.$$

Common C-spaces

There are some C-spaces that occur in many different contexts:

- $SO(2)$ ("special orthogonal group"): A robot that can rotate (but not translate) in the plane.
- $SE(2)$ ("special Euclidean group"): A robot that can translate and rotate in the plane.
- $SO(3)$: A robot that can rotate in 3-space.
- $SE(3)$: A robot that can translate and rotate in 3-space.

(How many dimensions do each of these C-spaces have?)

Free space and obstacle space

The configuration space itself just describes where the robot is. It doesn't take the obstacles into account.

Obstacles:

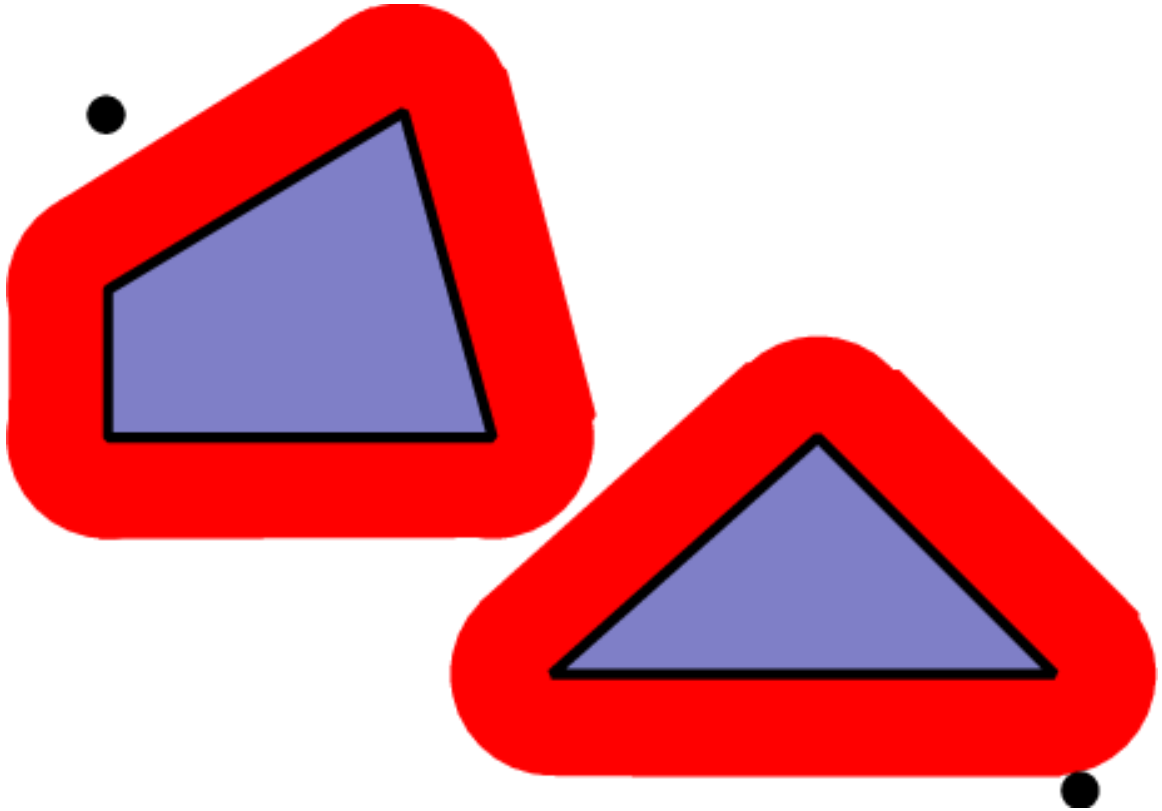
- An **obstacle configuration** is a configuration in which the robot is in collision with something in the environment, including possibly itself.
- The set of obstacle configurations is denoted $\mathcal{C}_{\text{obst}}$.

Free space:

- Everything else is a **free configuration**.
- The set of free configurations is $\mathcal{C}_{\text{free}} = \mathcal{C} - \mathcal{C}_{\text{obst}}$.

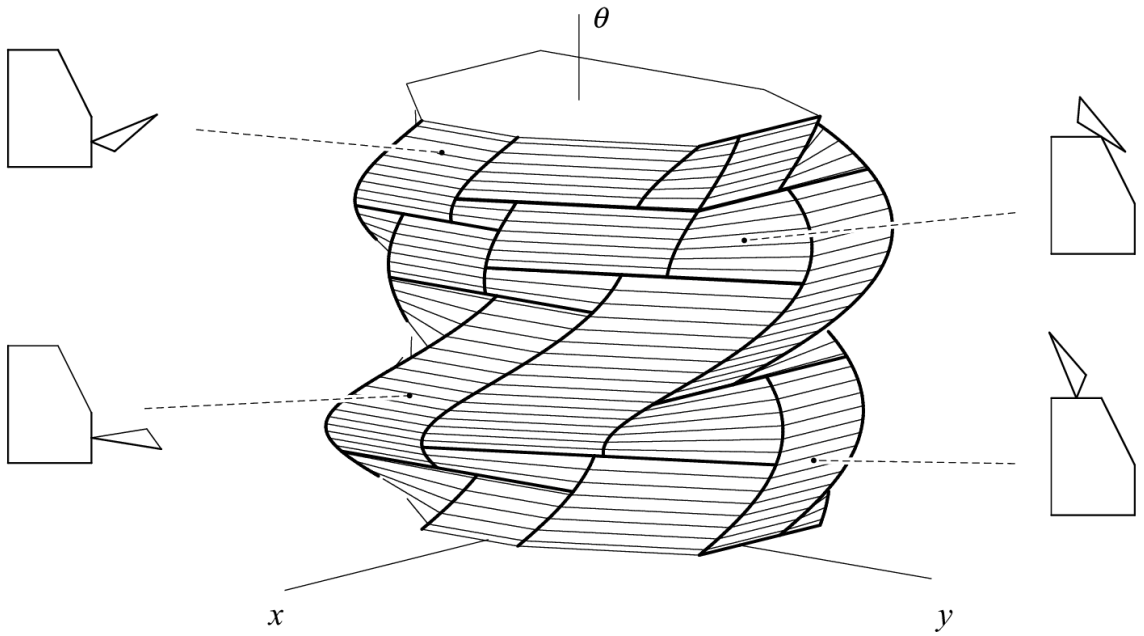
Back to our examples

For the disc robot:



Back to our examples

For the rectangle robot:

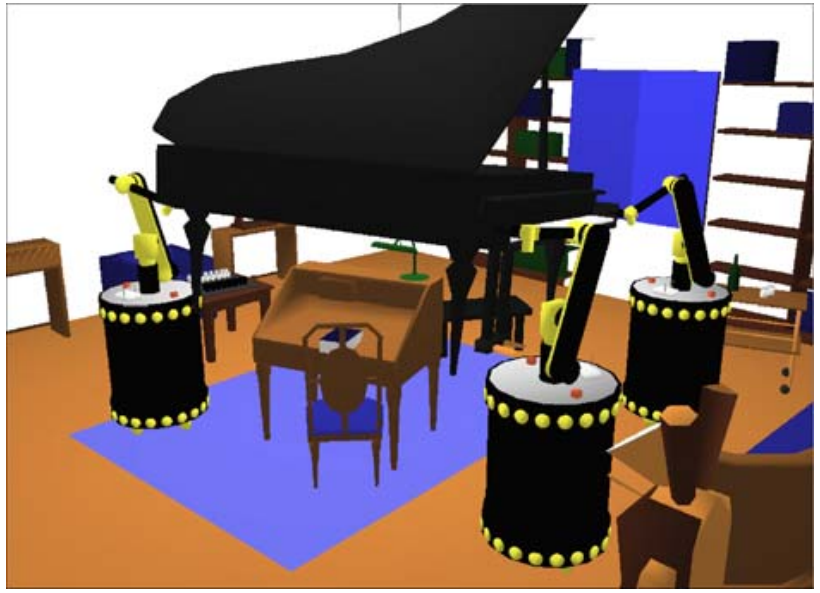


Motion planning algorithms

The problem

The **motion planning problem** has these inputs:

- A description of $\mathcal{C}_{\text{free}}$.
- A start configuration.
- A goal configuration.



This is sometimes called the **piano movers' problem**.

Combinatorial approaches

Good news: Combinatorial algorithms exist to solve this problem.

Bad news: These algorithms are:

- Difficult to understand.
- Difficult to implement.
- Exponential time.

More bad news: This problem is PSPACE-complete. (PSPACE-complete is at least as bad as NP-complete, and likely even worse.) So we shouldn't expect to ever come up with an efficient algorithm.

What's the problem?

The complexity of the problem comes from the fact that the C-space obstacles can be difficult to compute and represent.

- The configuration space may have high dimension.
- The obstacle boundaries may be complicated.
- Accurately describing the obstacle boundaries may require an absurd amount of memory.

Collision detection

Instead of representing $\mathcal{C}_{\text{obst}}$ explicitly, efficient algorithms are built around **collision detection**.

> Given a configuration, determine whether or not it's an obstacle configuration.

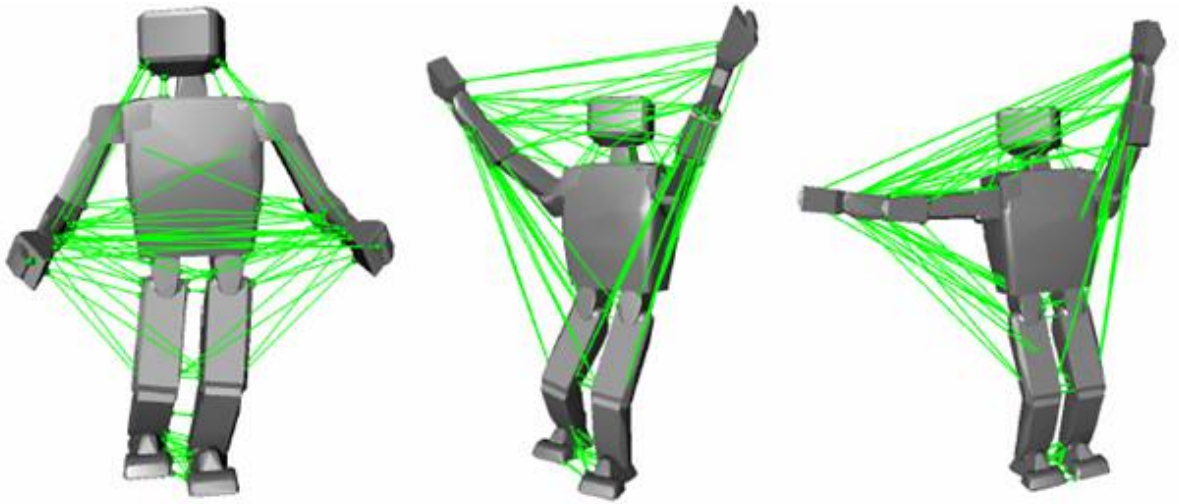
Or, more generally:

> Given a short path in configuration space, determine whether or not any configuration along that path is an obstacle configuration.

Collision detection overview

To perform collision tests, one can transform (that is, translate, rotate, and adjust joints) the geometric model of the robot, then test whether this transformed version intersects any obstacles.

We'll treat collision detection basically as a black box, but there are efficient geometric algorithms for this, especially if we are willing to accept an approximation.



Sampling based motion planning

Using collision detection queries, we can design algorithms that work well for many common types of instances.

The best existing algorithms of this type are **sampling-based** algorithms.

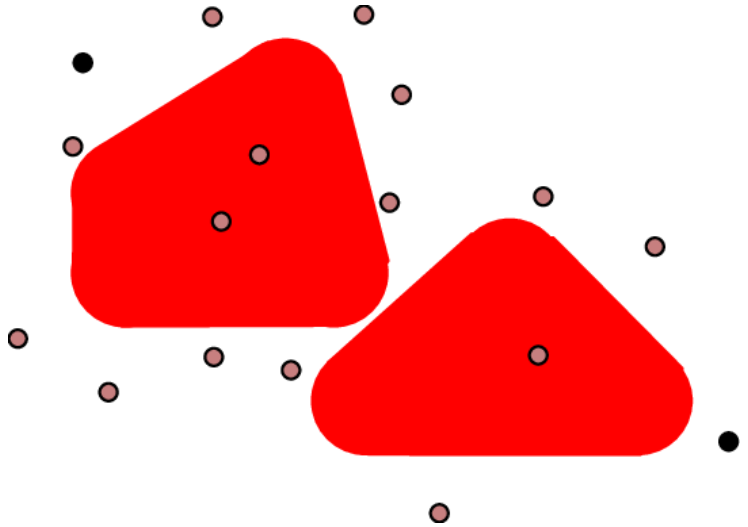
- The algorithm chooses a sequence of (random) samples that are well-distributed throughout the C-space.
- Using these samples as a guide, the algorithm constructs a graph in $\mathcal{C}_{\text{free}}$, attempting to connect the initial and goal configurations.

The differences between sampling based algorithms are in the details of how the samples are used, and in what kind of graph is constructed.

Probabilistic roadmaps

The **probabilistic roadmap** (PRM) builds an arbitrary graph.

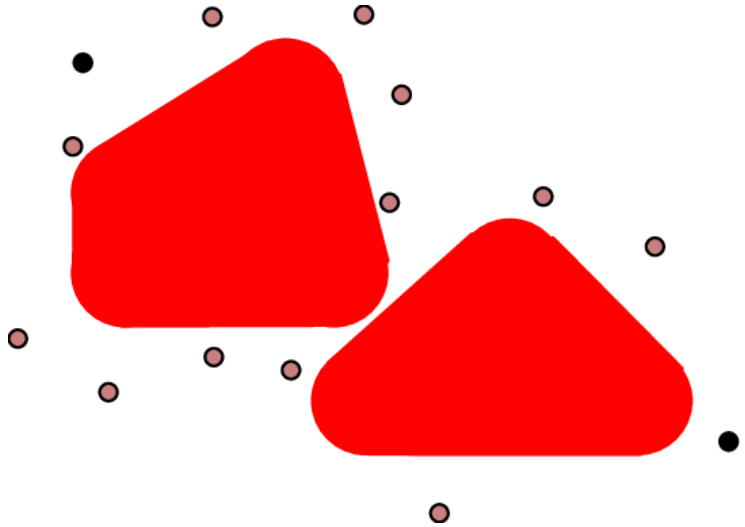
- Use a collection of samples, plus the initial and final configurations, as nodes in the graph.
- Discard samples that are in obstacles.
- Attempt to connect pairs of nodes that are within the **connection distance** of each other. Use the collision detector to determine whether a connection can be made.



Probabilistic roadmaps

The **probabilistic roadmap** (PRM) builds an arbitrary graph.

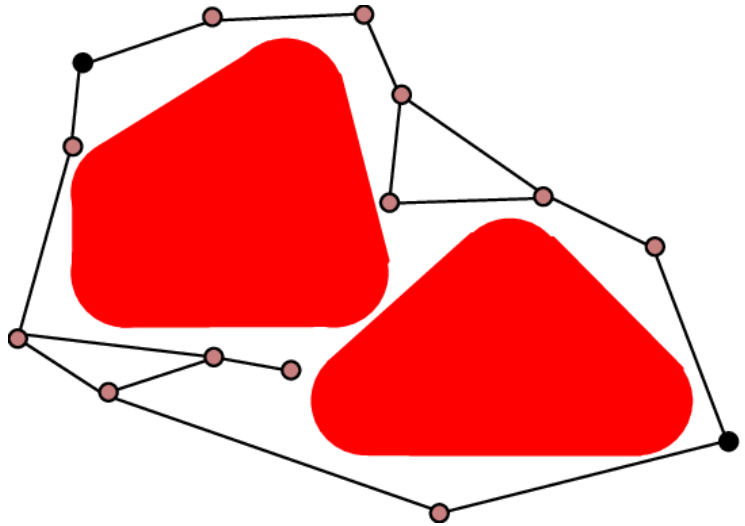
- Use a collection of samples, plus the initial and final configurations, as nodes in the graph.
- Discard samples that are in obstacles.
- Attempt to connect pairs of nodes that are within the **connection distance** of each other. Use the collision detector to determine whether a connection can be made.



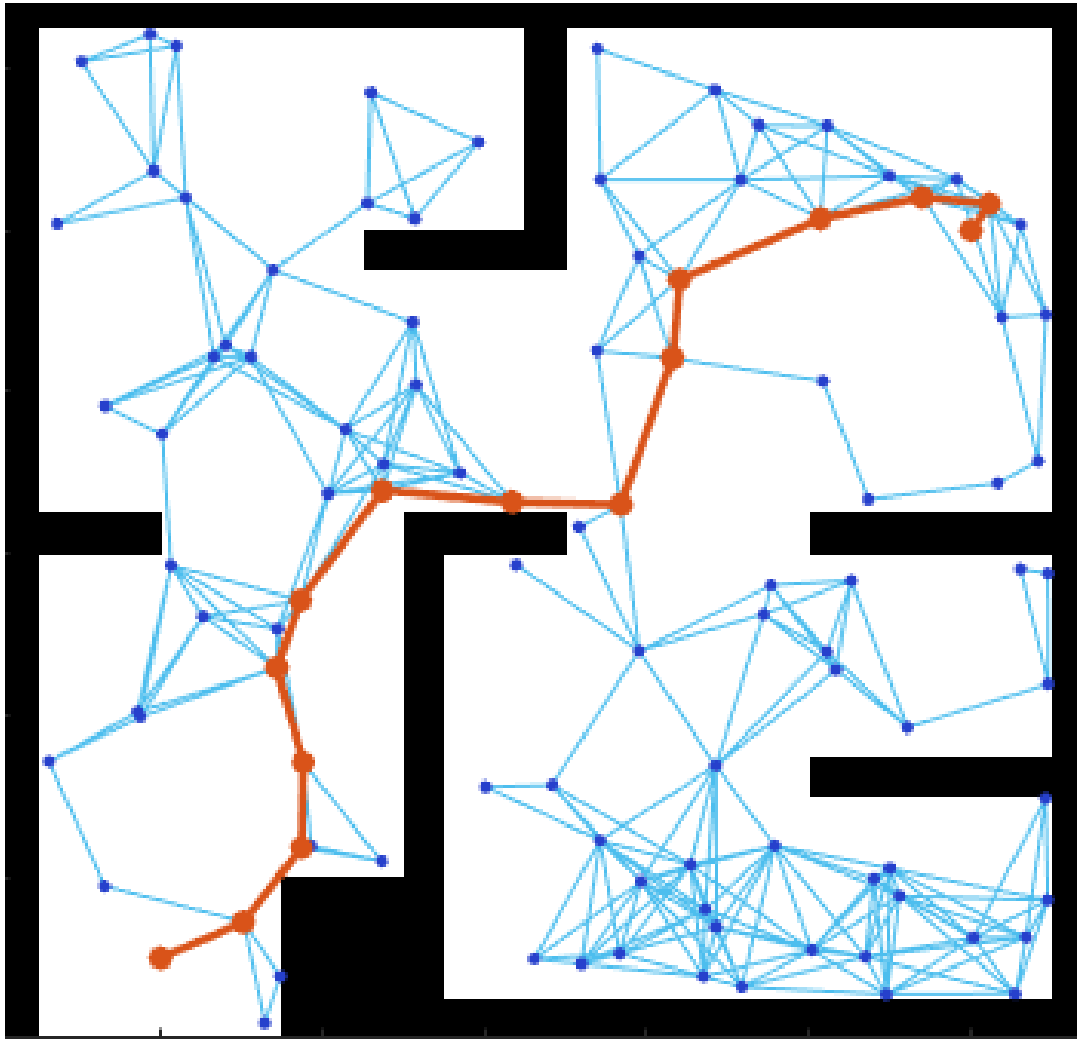
Probabilistic roadmaps

The **probabilistic roadmap** (PRM) builds an arbitrary graph.

- Use a collection of samples, plus the initial and final configurations, as nodes in the graph.
- Discard samples that are in obstacles.
- Attempt to connect pairs of nodes that are within the **connection distance** of each other. Use the collision detector to determine whether a connection can be made.



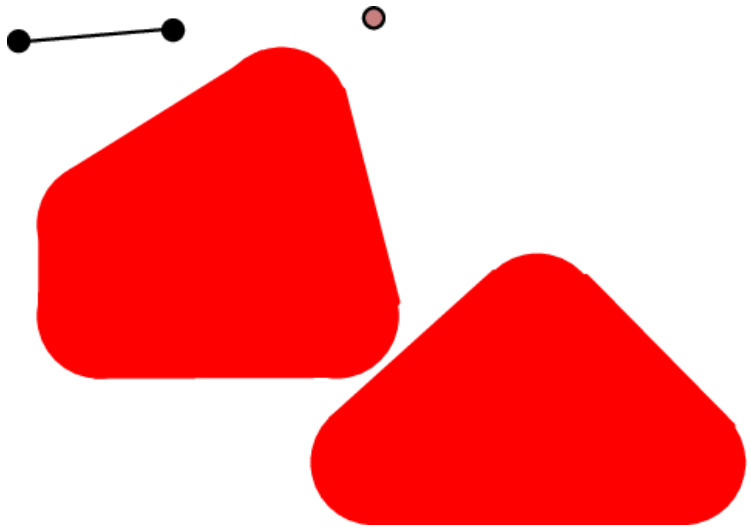
PRM example



Rapidly-exploring random trees

The **rapidly-exploring random tree** (RRT) builds a tree.

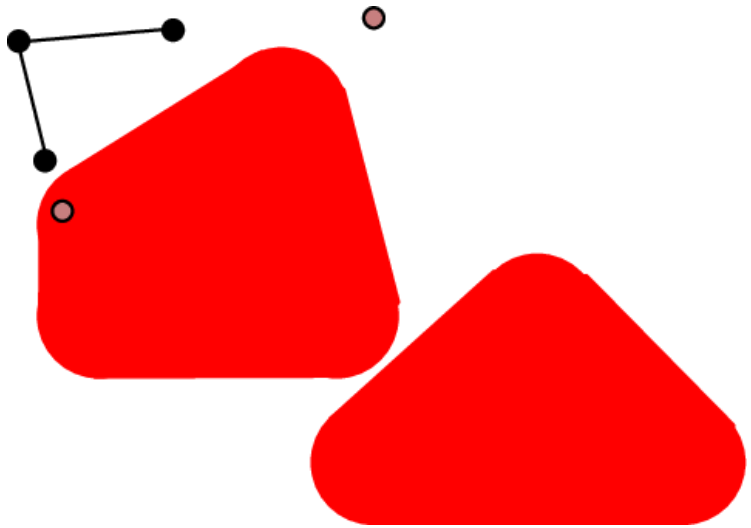
- Begin with the start configuration.
- Choose a random sample.
- Find the nearest configuration in the tree to this sample.
- Extend the tree from this configuration toward the sample.
- Repeat.



Rapidly-exploring random trees

The **rapidly-exploring random tree** (RRT) builds a tree.

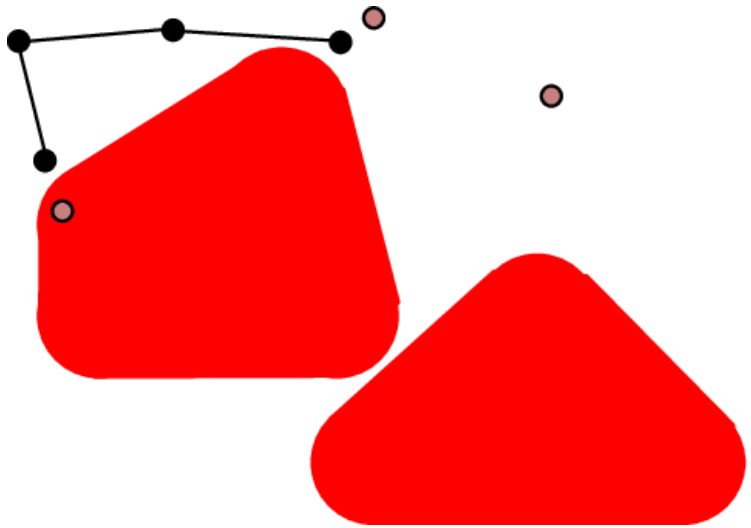
- Begin with the start configuration.
- Choose a random sample.
- Find the nearest configuration in the tree to this sample.
- Extend the tree from this configuration toward the sample.
- Repeat.



Rapidly-exploring random trees

The **rapidly-exploring random tree** (RRT) builds a tree.

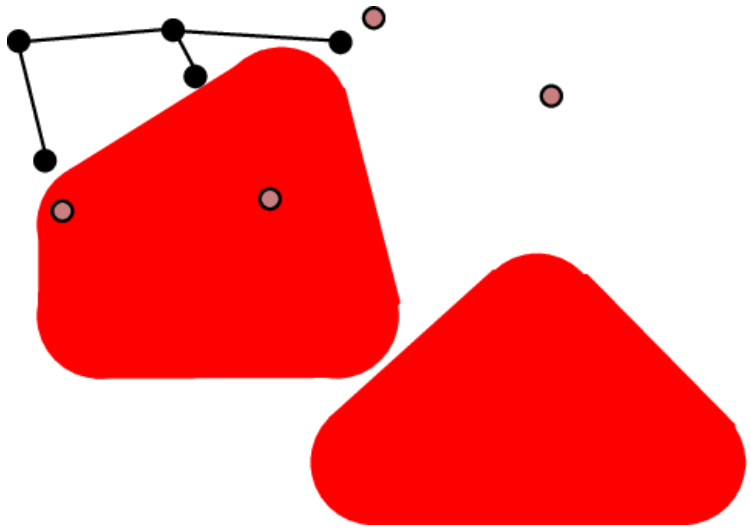
- Begin with the start configuration.
- Choose a random sample.
- Find the nearest configuration in the tree to this sample.
- Extend the tree from this configuration toward the sample.
- Repeat.



Rapidly-exploring random trees

The **rapidly-exploring random tree** (RRT) builds a tree.

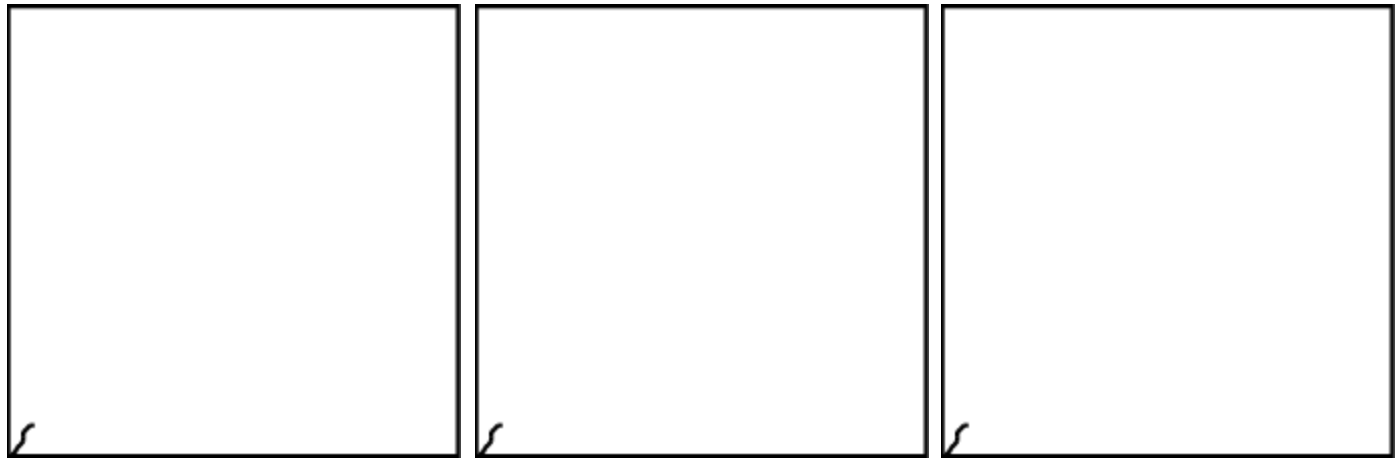
- Begin with the start configuration.
- Choose a random sample.
- Find the nearest configuration in the tree to this sample.
- Extend the tree from this configuration toward the sample.
- Repeat.



RRT details

Some comments:

- The random samples “pull” the tree into the largest unexplored regions. Tree nodes that are the closest to large unexplored regions are the most likely to have children added. This is called **Voronoi bias**.

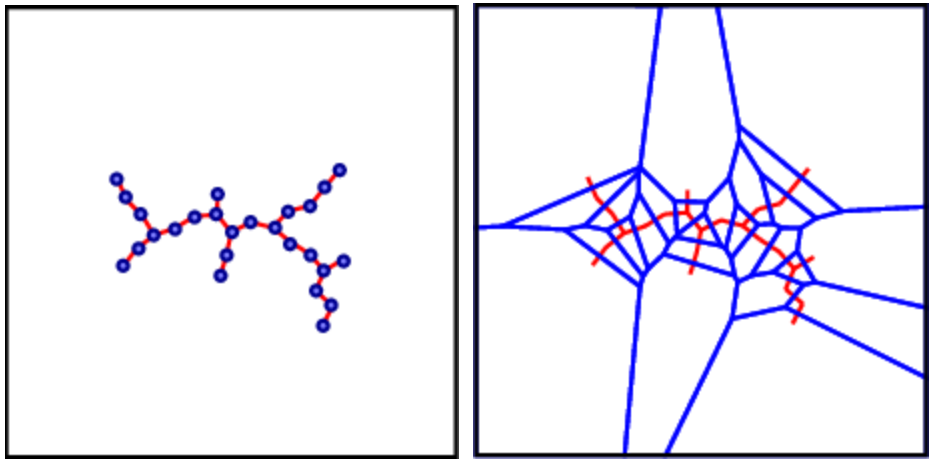


- In practice, one uses (at least) two trees and alternates between extending toward random samples and attempting to connect the two trees. Thus, it becomes a **bidirectional search**.

RRT details

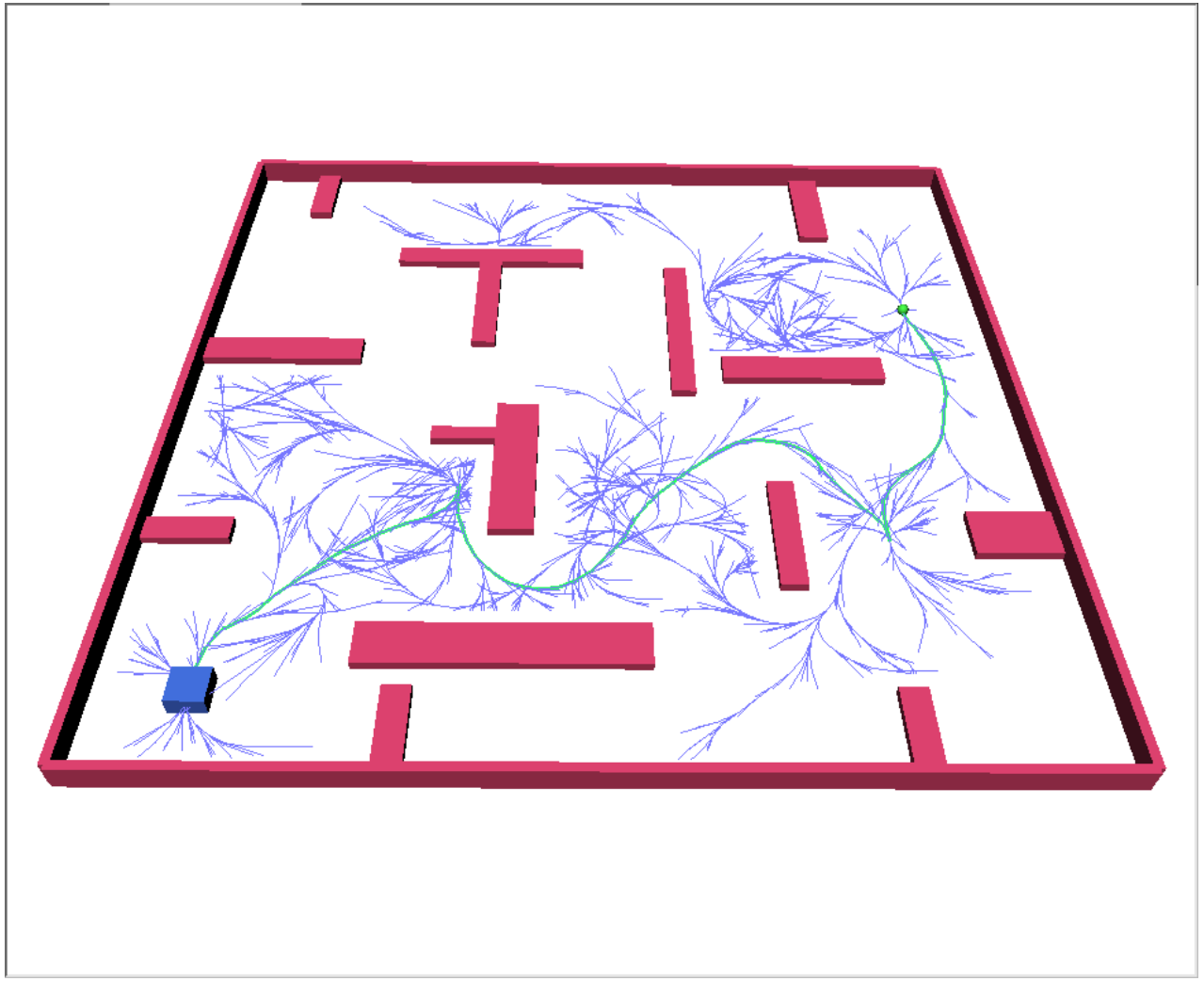
Some comments:

- The random samples “pull” the tree into the largest unexplored regions. Tree nodes that are the closest to large unexplored regions are the most likely to have children added. This is called **Voronoi bias**.



- In practice, one uses (at least) two trees and alternates between extending toward random samples and attempting to connect the two trees. Thus, it becomes a **bidirectional search**.

RRT example



Comparison between algorithms

PRMs are most useful when:

- Many motion planning instances need to be solved in the same C-space. (multiple query)

RRTs are most useful when:

- Only a single instance needs to be solved. (single query)
- The system has motion constraints that make it difficult to create direct, straight connections.

Completeness

If a solution exists, can we guarantee that the PRM or RRT algorithms will find it? **No!** Success depends on the sequence of samples we select.

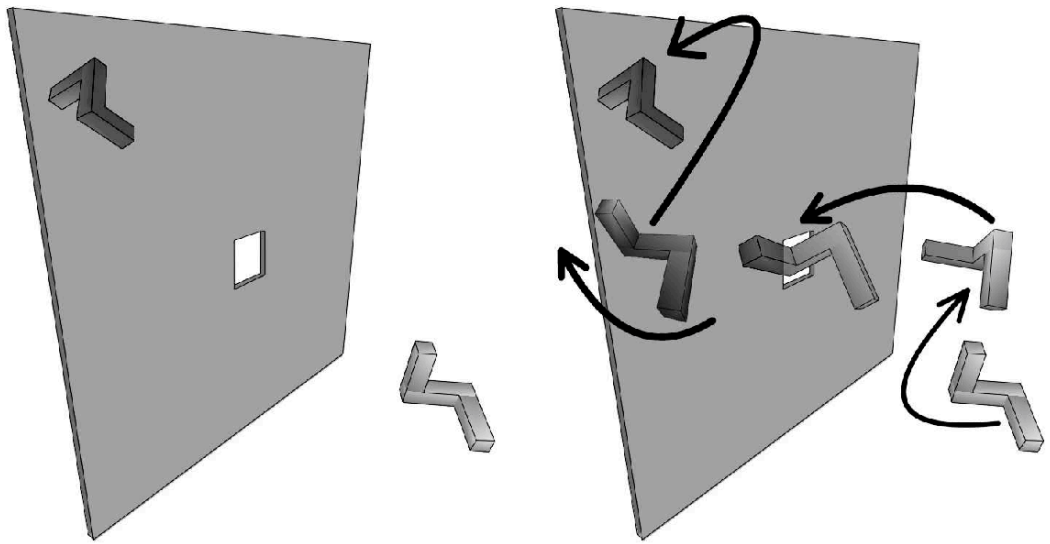
However, we can use a weaker notion of completeness:

A motion planner is **probabilistically complete** if the probability of failure goes to zero as the number of samples increases.

$$\lim_{n \to \infty} P(\text{failure}) = 0$$

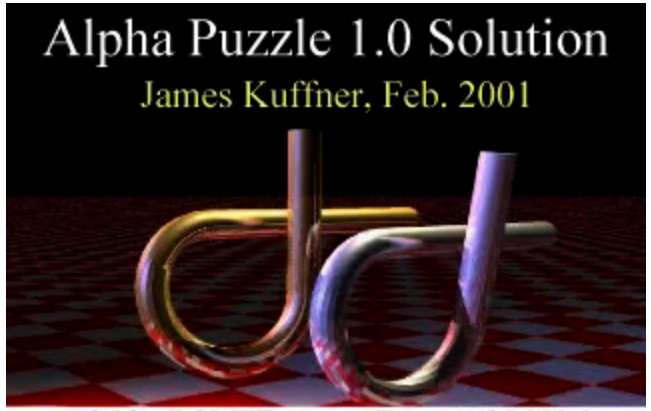
Both the PRM and the RRT methods are probabilistically complete.

When is motion planning hard? Narrow corridors



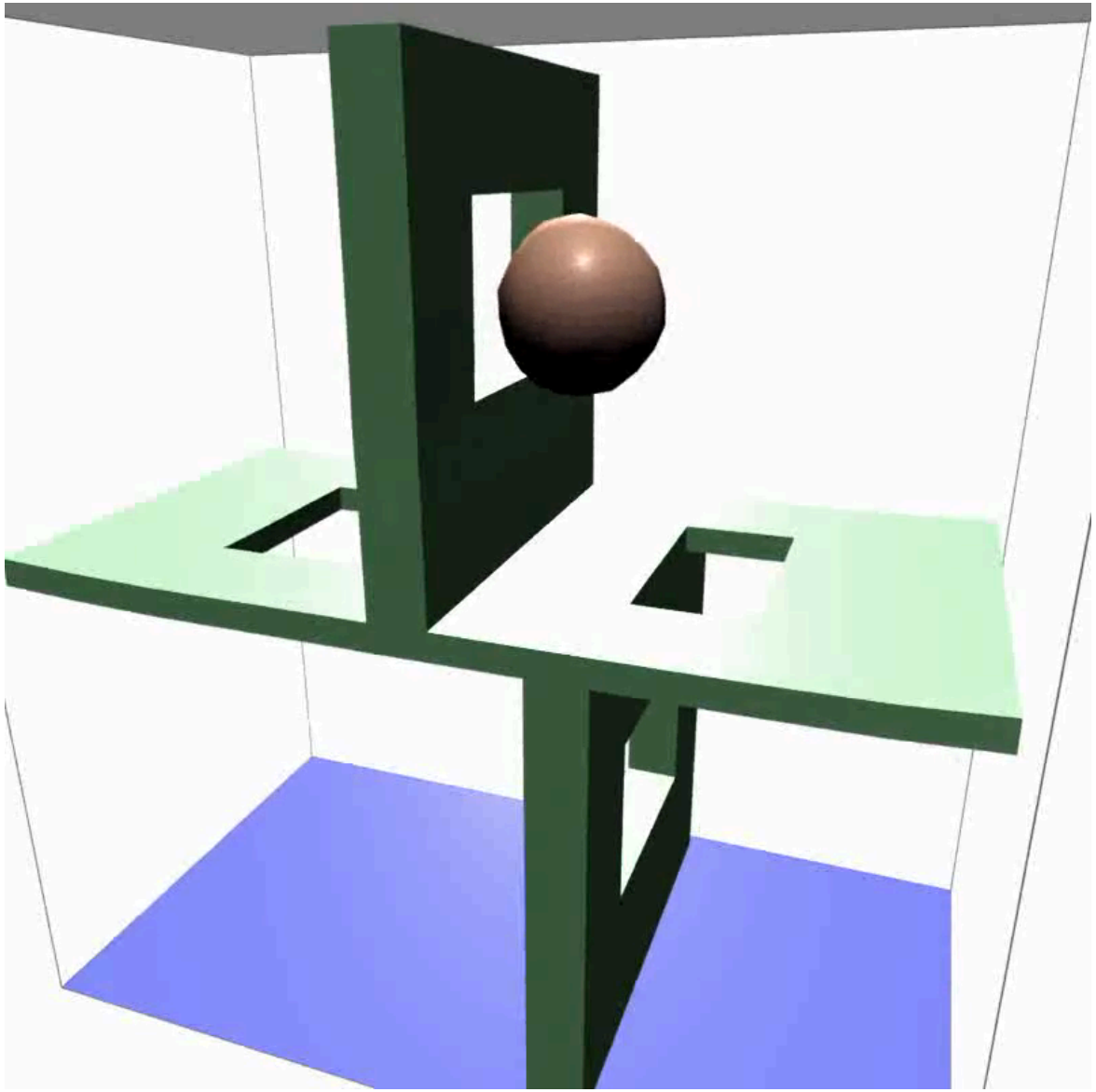
(a) (b)

Example: Alpha puzzle



model by DSMFT group, Texas A&M Univ.
original model by Boris Yamrom, GE

Example: Deformable objects



Example: “Wreckless” driving

Example: Grasping