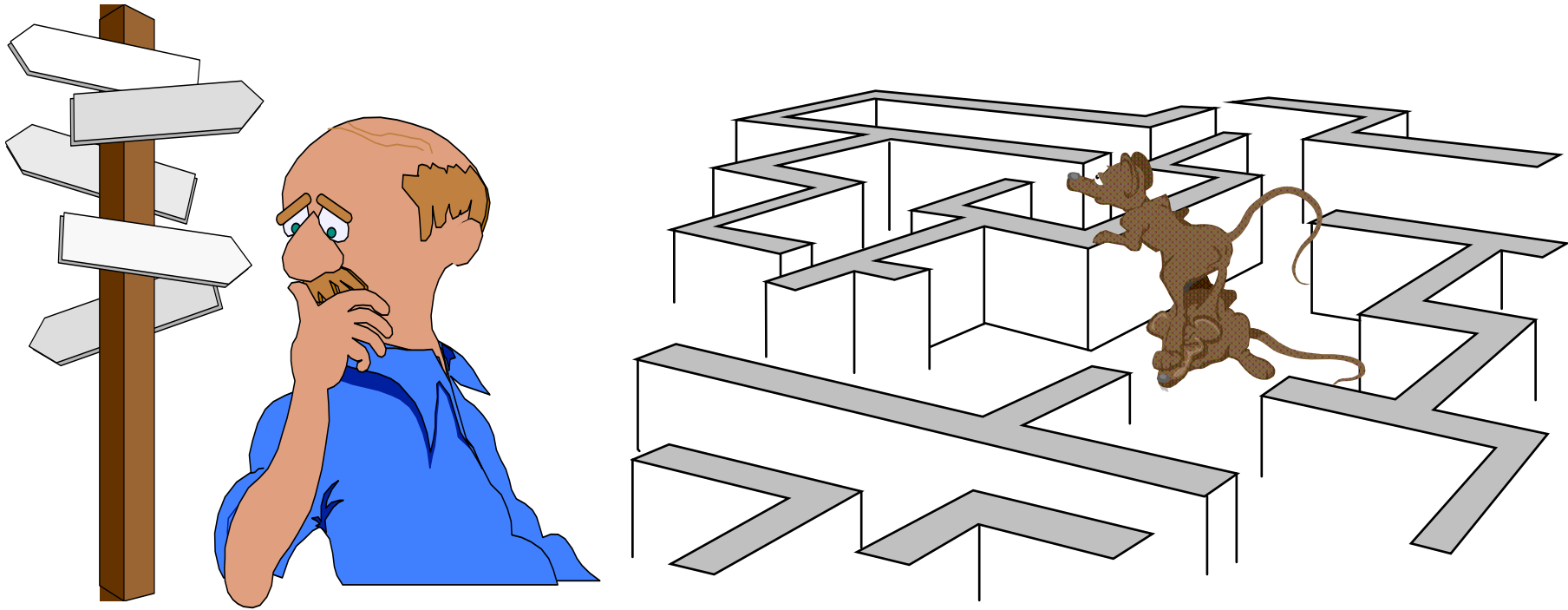


Motion Planning

Howie CHoset

What is Motion Planning?

- Determining where to go



Overview

- The Basics
 - Motion Planning Statement
 - The World and Robot
 - Configuration Space
 - Metrics
- Path Planning Algorithms
 - Start-Goal Methods
 - Map-Based Approaches
 - Cellular Decompositions
- Applications
 - Navigating Large Spaces
 - Coverage

The World consists of...

- Obstacles
 - Already occupied spaces of the world
 - In other words, robots can't go there
- Free Space
 - Unoccupied space within the world
 - Robots “might” be able to go here
 - To determine where a robot can go, we need to discuss what a *Configuration Space* is

Motion Planning Statement

If W denotes the robot's workspace,

And C_i denotes the i 'th obstacle,

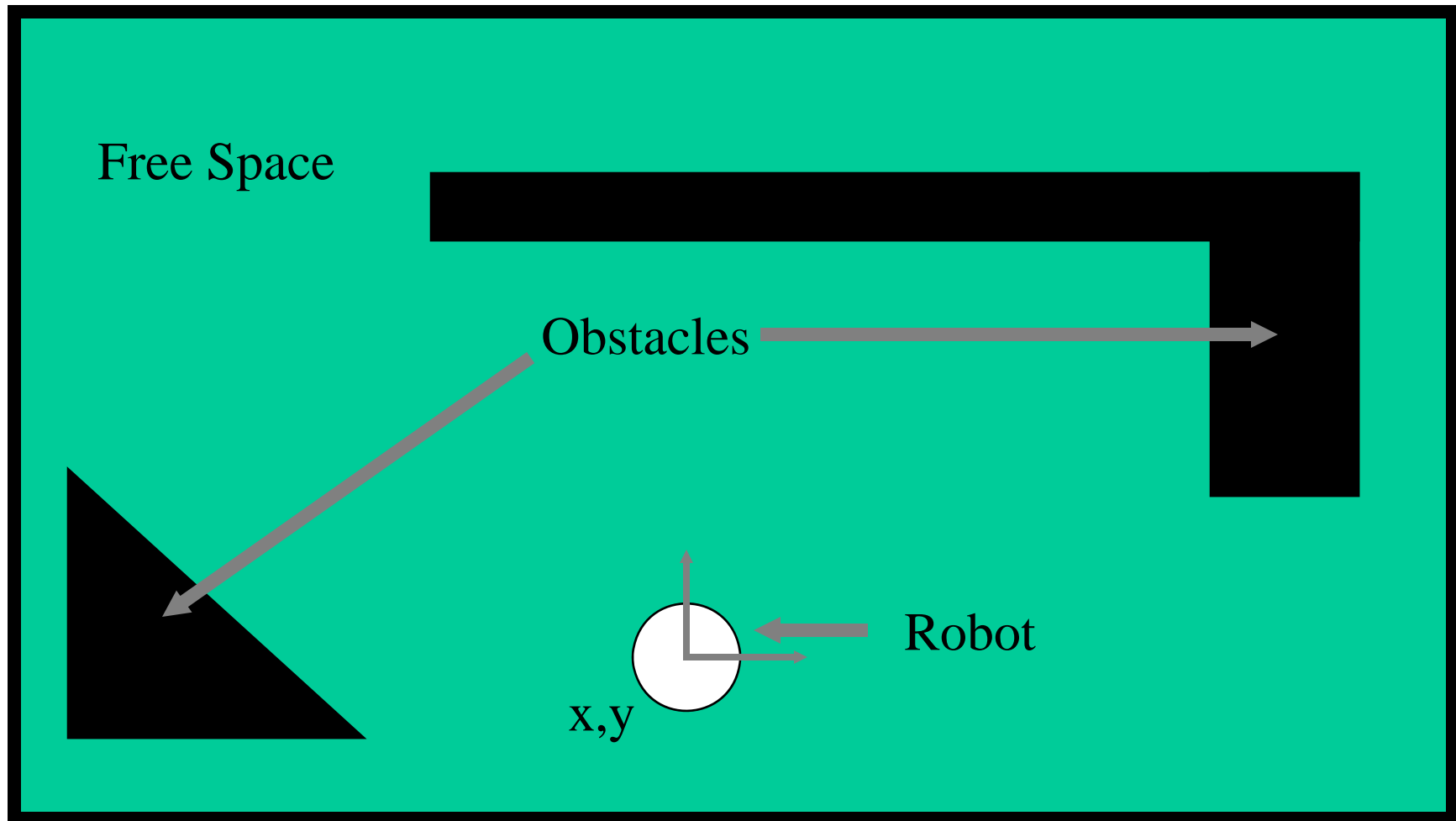
Then the robot's free space, FS , is defined as:

$$FS = W \setminus (\cup C_i)$$

And a path $c \in C^0$ is $c : [0,1] \rightarrow FS$

where $c(0)$ is q_{start} and $c(1)$ is q_{goal}

Example of a World (and Robot)



What is a good path?

Basics: Metrics

- There are many different ways to measure a path:
 - Time
 - Distance traveled
 - Expense
 - Distance from obstacles
 - Etc...

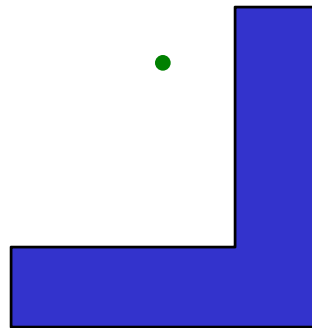


Bug 1

But some computing power!

- known direction to goal
- otherwise local sensing

walls/obstacles & **encoders**



"Bug 1" algorithm

- 1) head toward goal
- 2) if an obstacle is encountered, circumnavigate it *and* remember how close you get to the goal
- 3) return to that closest point (by wall-following) and continue

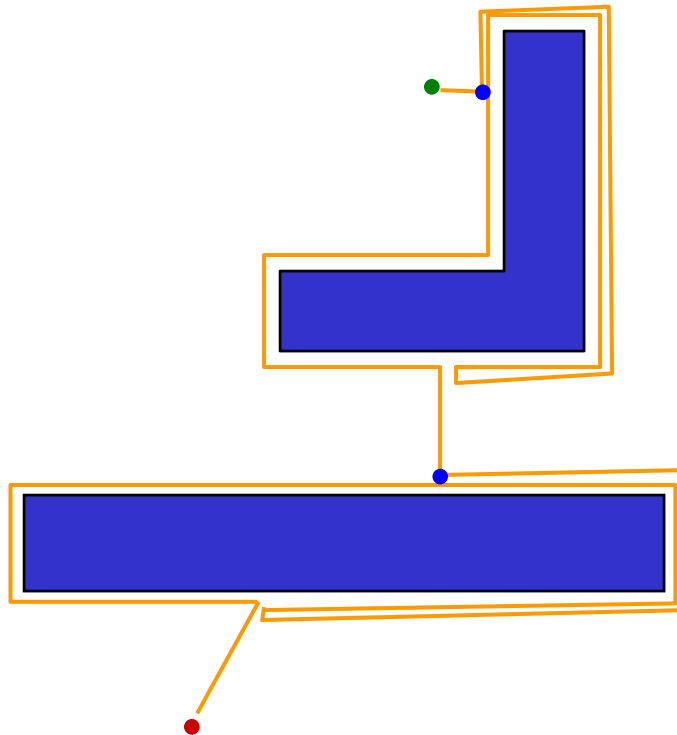


Bug 1

But some computing power!

- known direction to goal
- otherwise local sensing

walls/obstacles & **encoders**



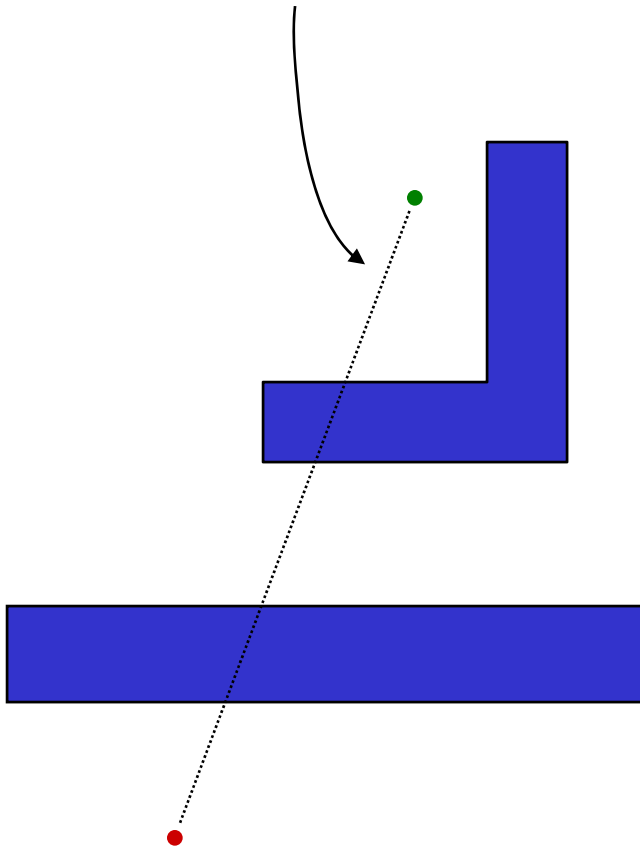
"Bug 1" algorithm

- 1) head toward goal
- 2) if an obstacle is encountered, circumnavigate it *and* remember how close you get to the goal
- 3) return to that closest point (by wall-following) and continue

Bug2

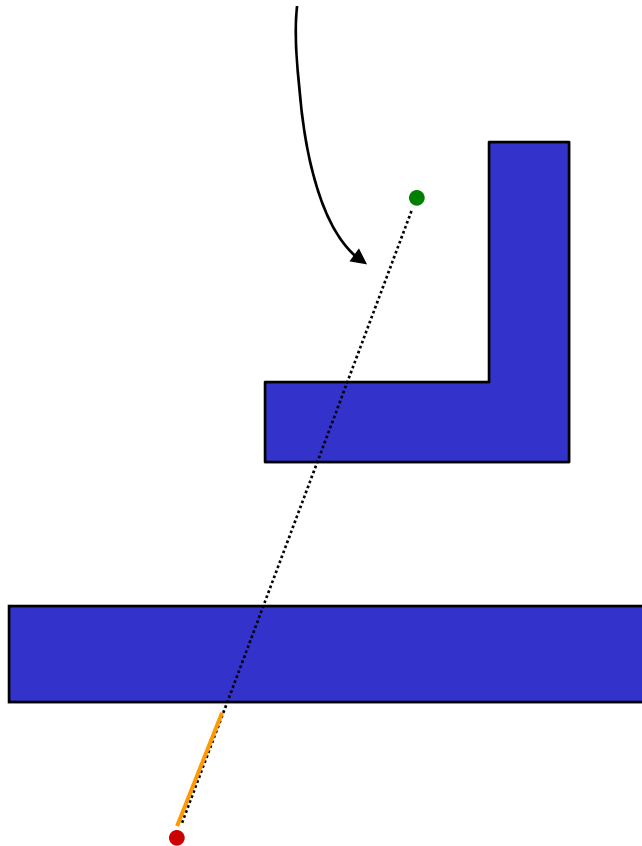
Call the line from the starting point to the goal the *m-line*

"Bug 2" Algorithm



A better bug?

Call the line from the starting point to the goal the *m-line*

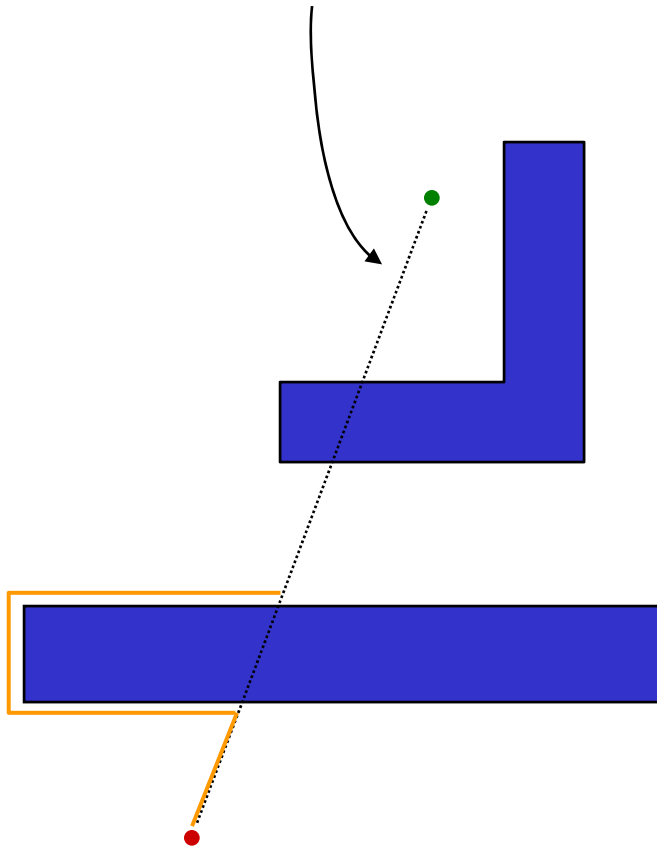


"Bug 2" Algorithm

1) head toward goal on the *m-line*

A better bug?

Call the line from the starting point to the goal the *m-line*

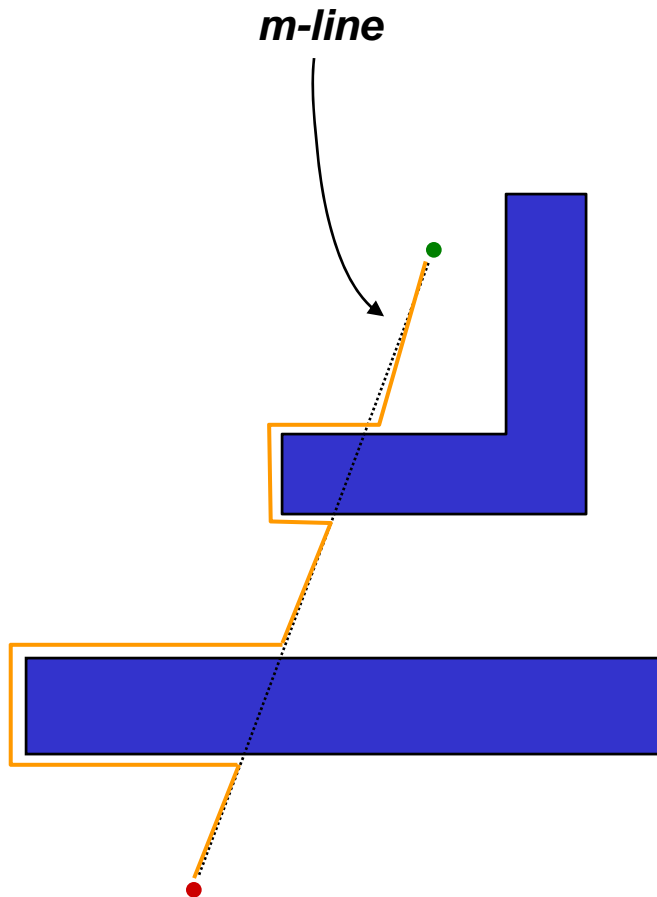


"Bug 2" Algorithm

- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again.

A better bug?

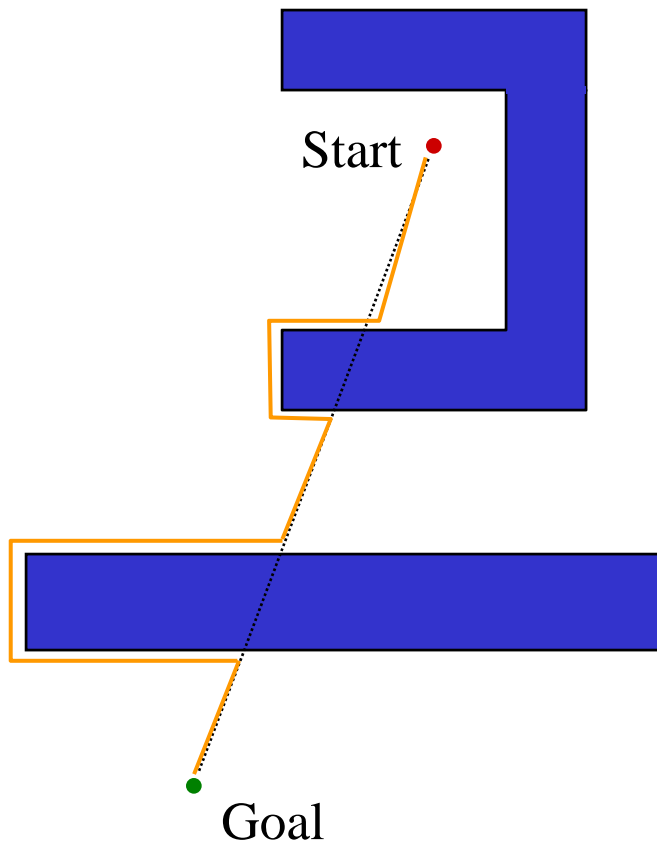
"Bug 2" Algorithm



- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again.
- 3) Leave the obstacle and continue toward the goal

A better bug?

"Bug 2" Algorithm

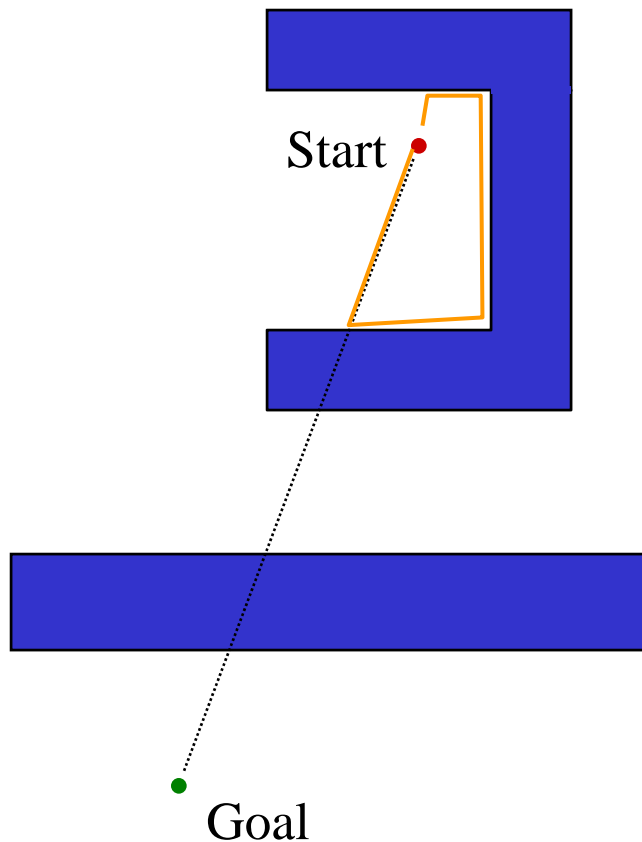


- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again.
- 3) Leave the obstacle and continue toward the goal

Better or worse than Bug1?

A better bug?

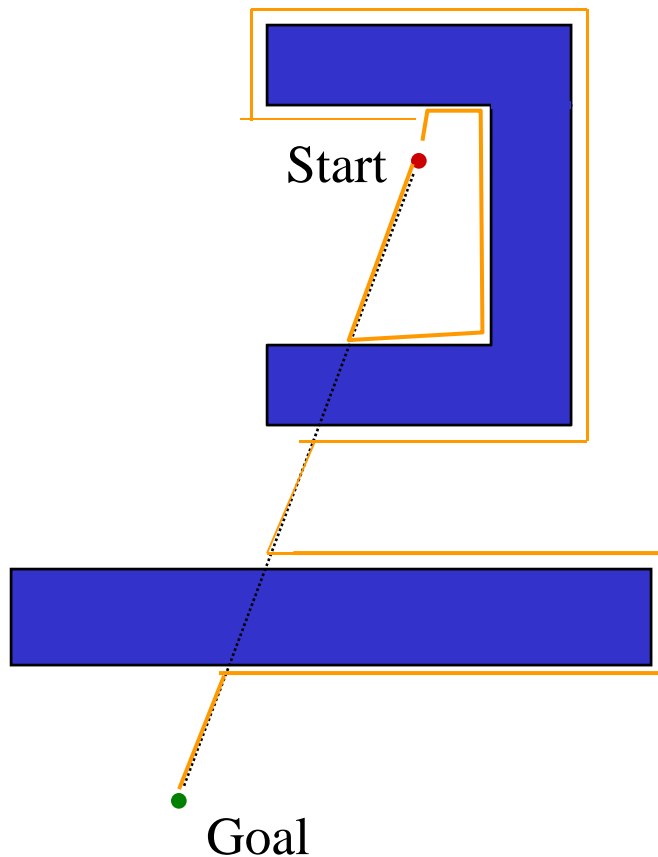
"Bug 2" Algorithm



- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again.
- 3) Leave the obstacle and continue toward the goal

A better bug?

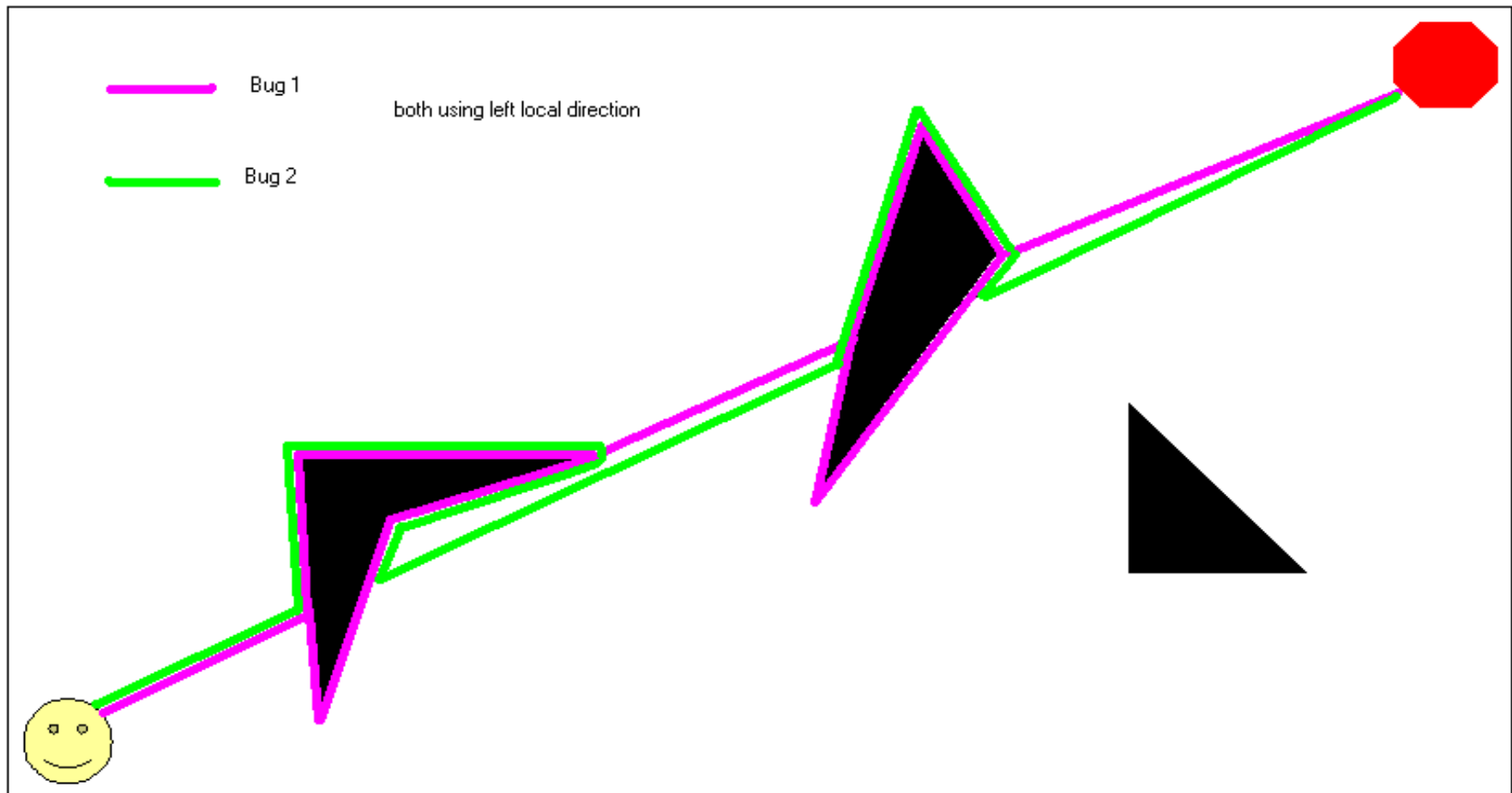
"Bug 2" Algorithm



- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again *closer to the goal*.
- 3) Leave the obstacle and continue toward the goal

Better or worse than Bug1?

Start-Goal Algorithm: Lumelsky Bug Algorithms



Lumelsky Bug Algorithms

- Unknown obstacles, known start and goal.
- Simple “bump” sensors, encoders.
- Choose arbitrary direction to turn (left/right) to make all turns, called “local direction”
- Motion is like an ant walking around:
 - In Bug 1 the robot goes all the way around each obstacle encountered, recording the point nearest the goal, then goes around again to leave the obstacle from that point
 - In Bug 2 the robot goes around each obstacle encountered until it can continue on its previous path toward the goal

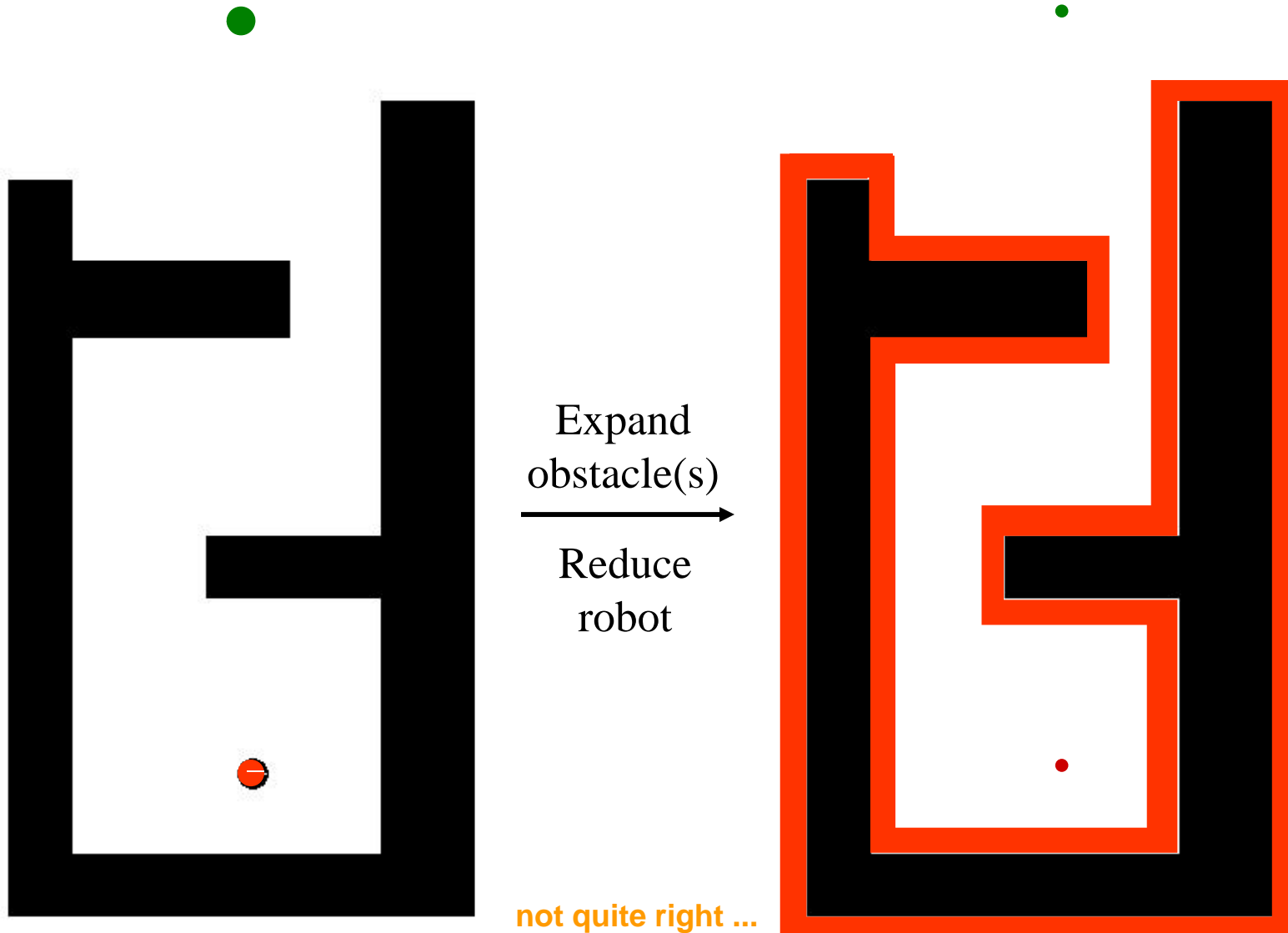
Assumptions?

Assumptions

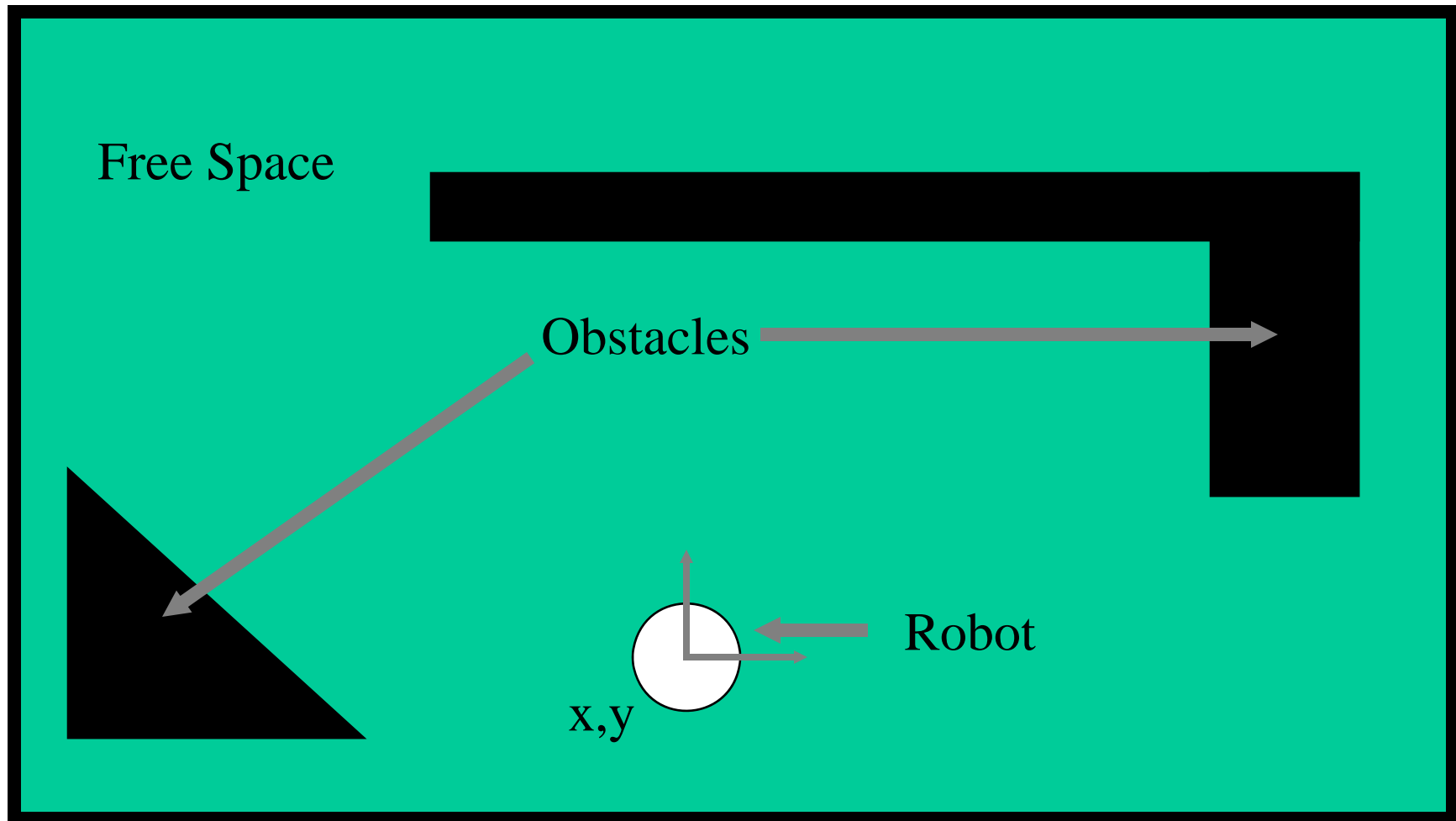
- Size of robot
- Perfect sensing
- Perfect control
- Localization (heading)

What else?

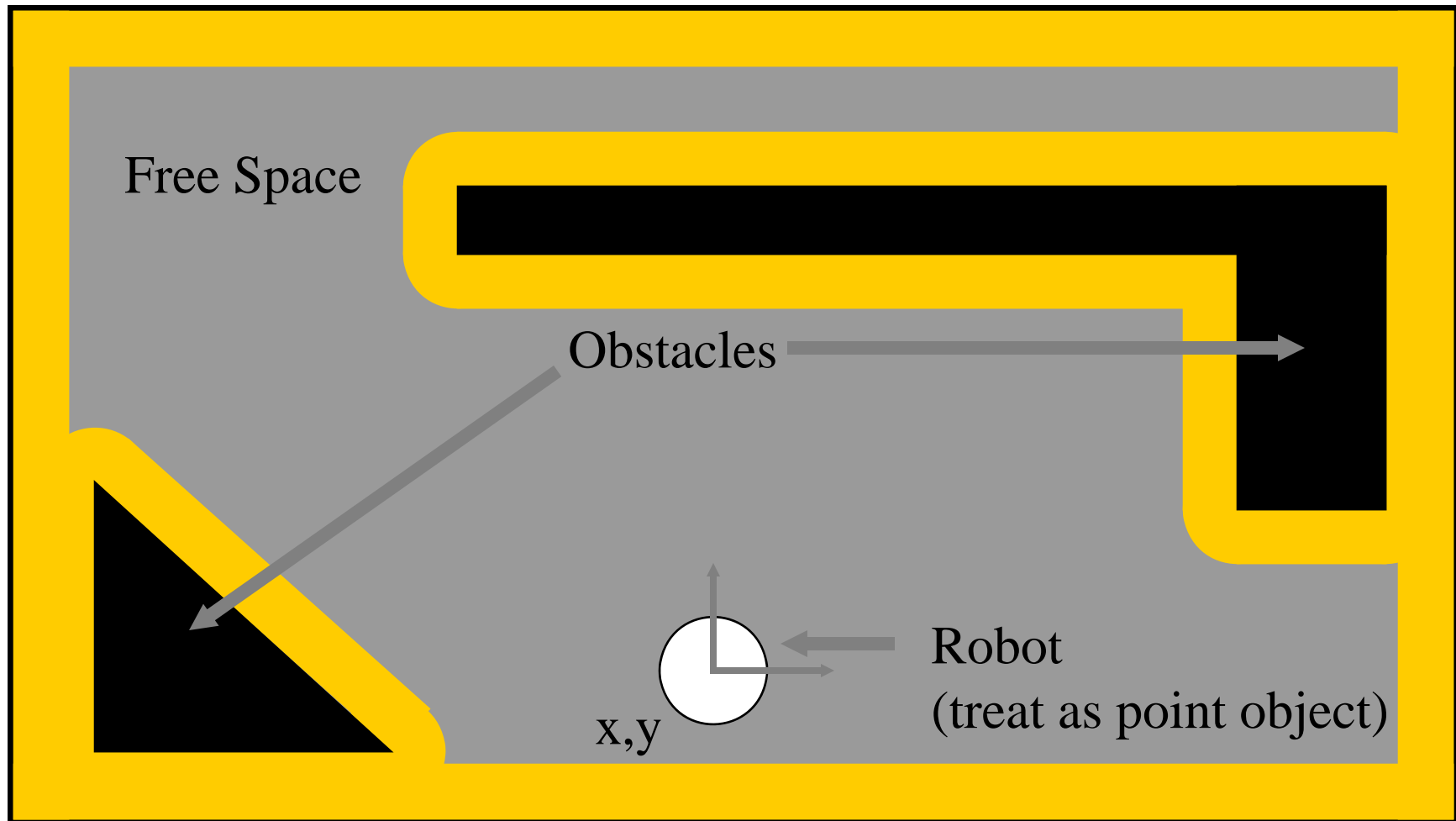
What is the position of the robot?



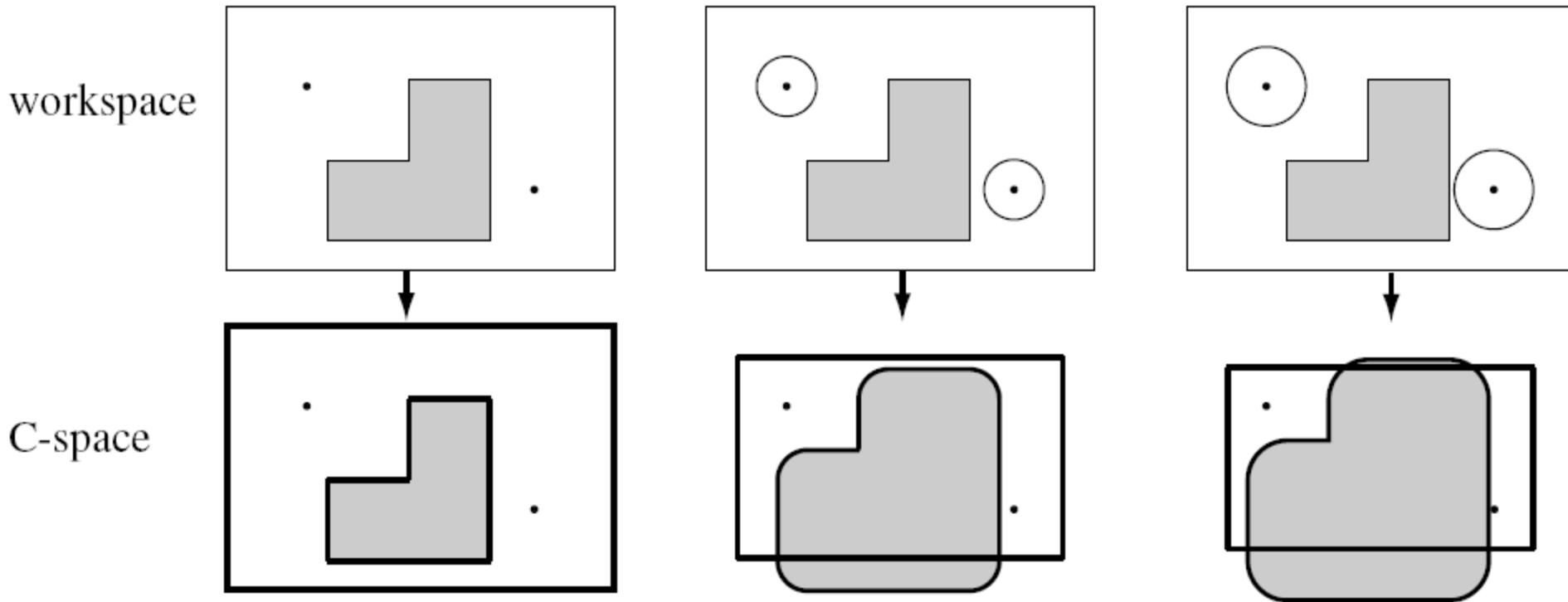
Example of a World (and Robot)



Configuration Space: Accommodate Robot Size

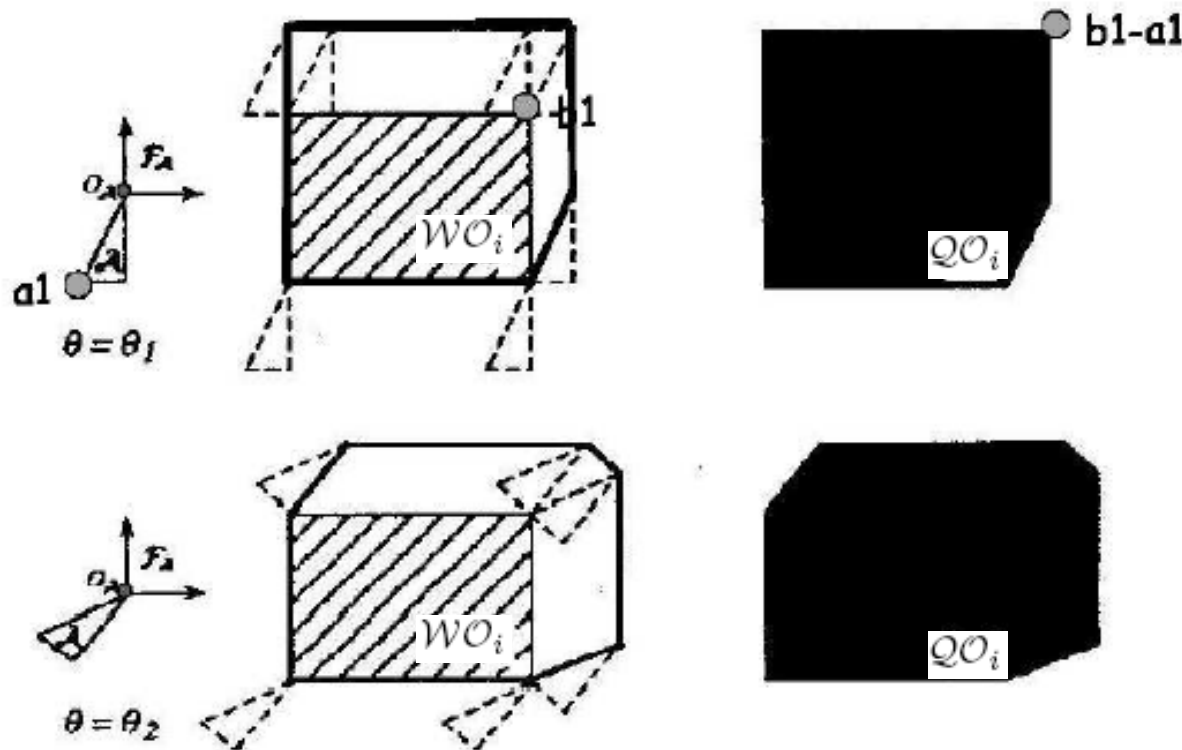


Trace Boundary of Workspace



Pick a reference point...

Translate-only, non-circularly



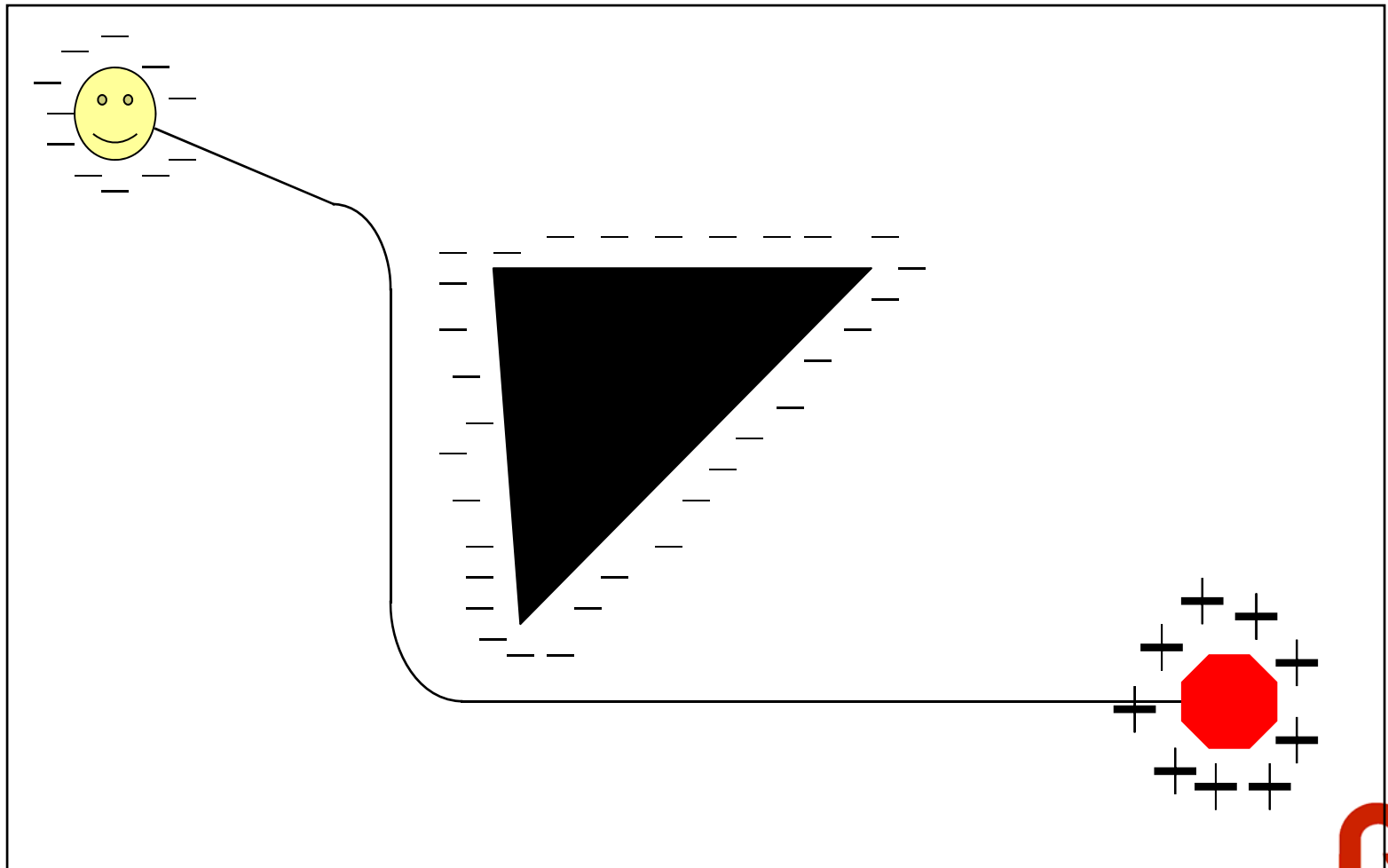
$$QO_i = \{q \in \mathcal{Q} \mid R(q) \cap WO_i \neq \emptyset\}.$$

Pick a reference point...

The Configuration Space

- What it is
 - A set of “reachable” areas constructed from knowledge of both the robot and the world
- How to create it
 - First abstract the robot as a point object. Then, enlarge the obstacles to account for the robot’s footprint and degrees of freedom
 - In our example, the robot was circular, so we simply enlarged our obstacles by the robot’s radius (*note the curved vertices*)

Start-Goal Algorithm: Potential Functions



Attractive/Repulsive Potential Field

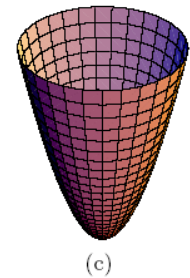
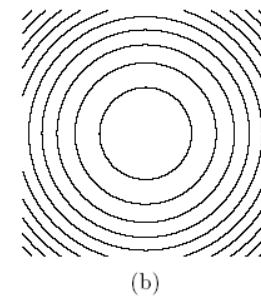
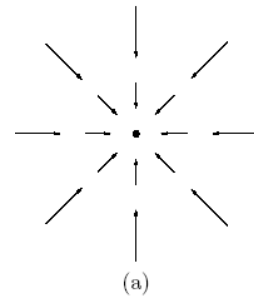
$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

- U_{att} is the “attractive” potential --- move to the goal
- U_{rep} is the “repulsive” potential --- avoid obstacles

Artificial Potential Field Methods: Attractive Potential

Quadratic Potential

$$U_{\text{att}}(q) = \frac{1}{2}\zeta d^2(q, q_{\text{goal}}),$$



$$\begin{aligned} F_{\text{att}}(q) &= \nabla U_{\text{att}}(q) = \nabla \left(\frac{1}{2}\zeta d^2(q, q_{\text{goal}}) \right), \\ &= \frac{1}{2}\zeta \nabla d^2(q, q_{\text{goal}}), \\ &= \zeta(q - q_{\text{goal}}), \end{aligned}$$

Distance

$$d : R^2 \times R^2 \rightarrow R$$

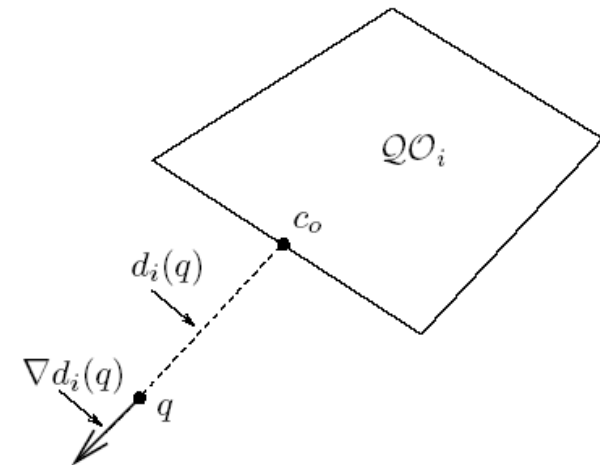
L1 Metric	(diamond)	$d(a, b) = a_x - b_x + a_y - b_y $
-----------	-----------	---------------------------------------

L2 Metric	(circle)	$d(a, b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$
-----------	----------	--

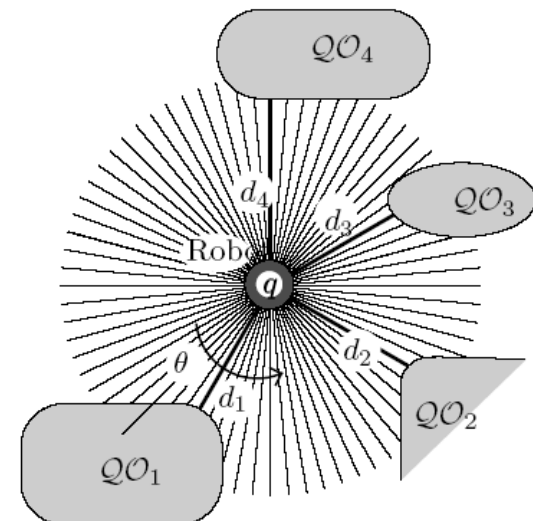
Distance to Obstacle(s)

$$d_i(q) = \min_{c \in \mathcal{QO}_i} d(q, c).$$

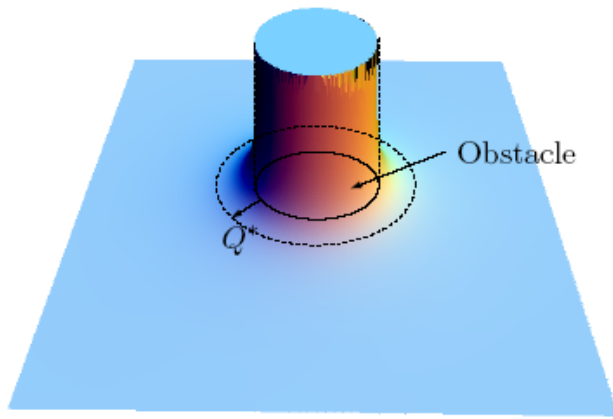
$$\nabla d_i(q) = \frac{q - c}{d(q, c)}$$



$$D(q) = \min d_i(q)$$



The Repulsive Potential

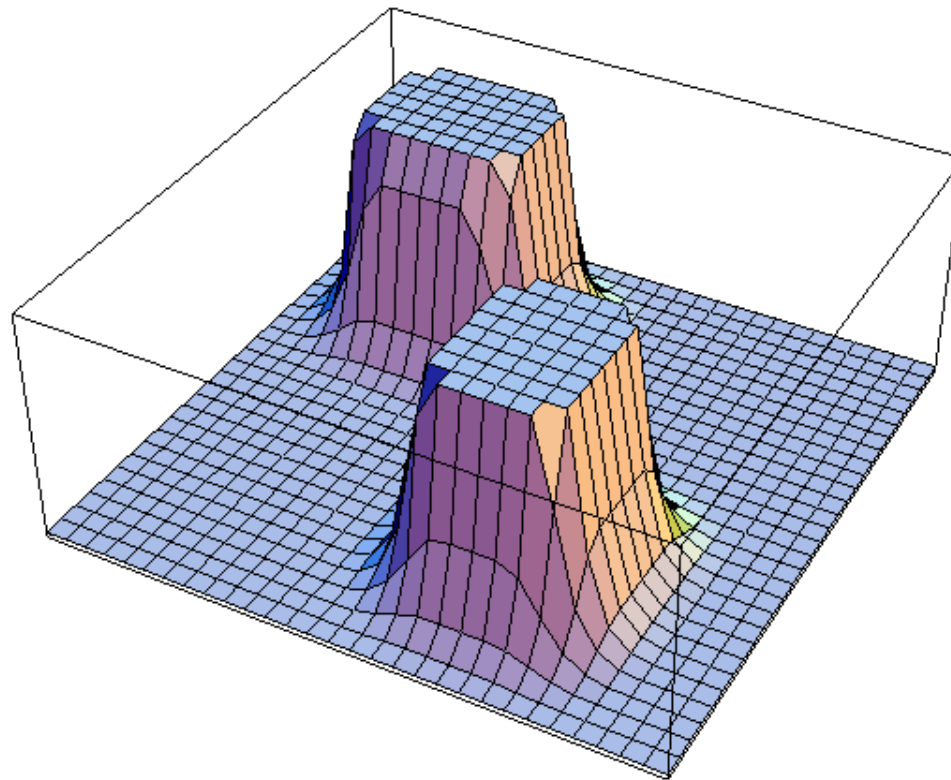


$$U_{\text{rep}}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{D(q)} - \frac{1}{Q^*}\right)^2, & D(q) \leq Q^*, \\ 0, & D(q) > Q^*, \end{cases}$$

whose gradient is

$$\nabla U_{\text{rep}}(q) = \begin{cases} \eta \left(\frac{1}{Q^*} - \frac{1}{D(q)} \right) \frac{1}{D^2(q)} \nabla D(q), & D(q) \leq Q^*, \\ 0, & D(q) > Q^*, \end{cases}$$

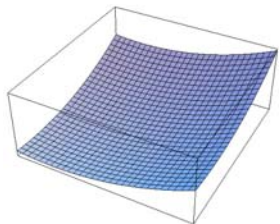
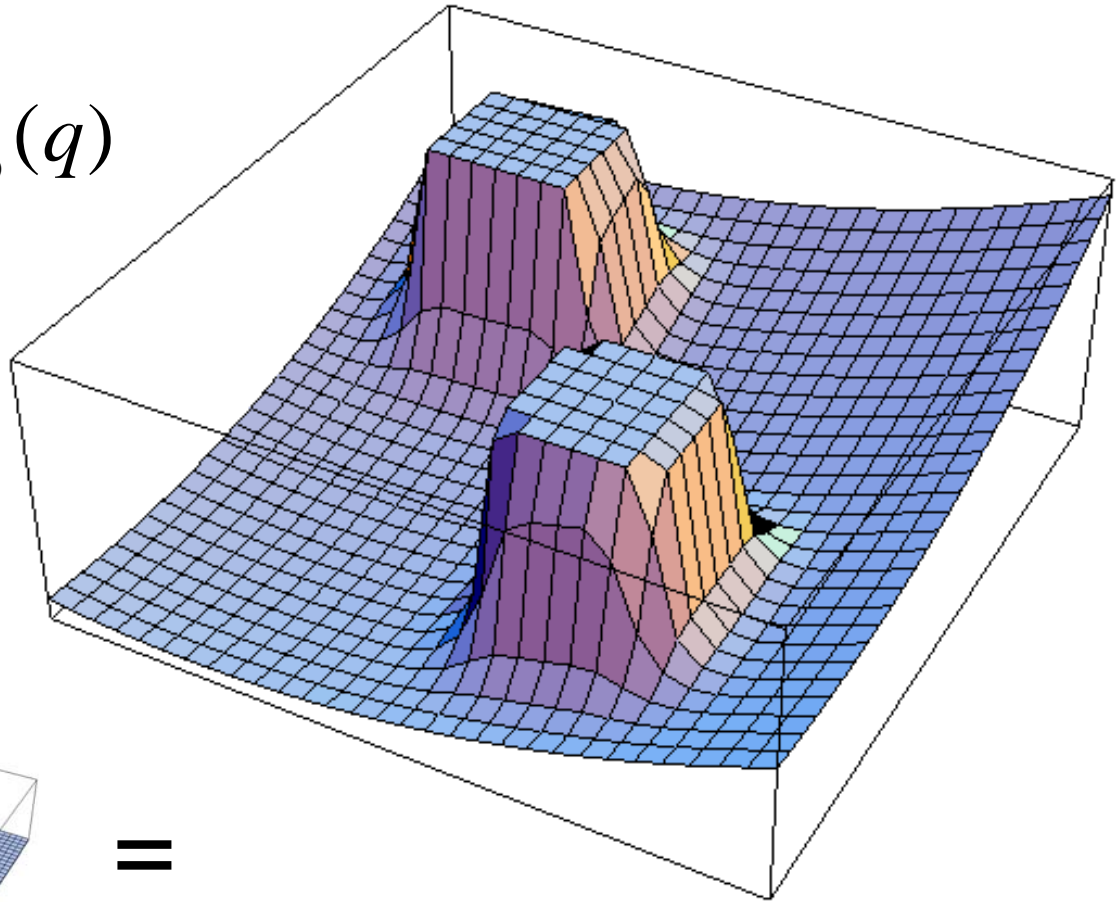
Repulsive Potential



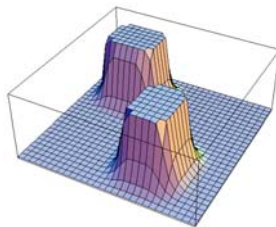
Total Potential Function

$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

$$F(q) = -\nabla U(q)$$

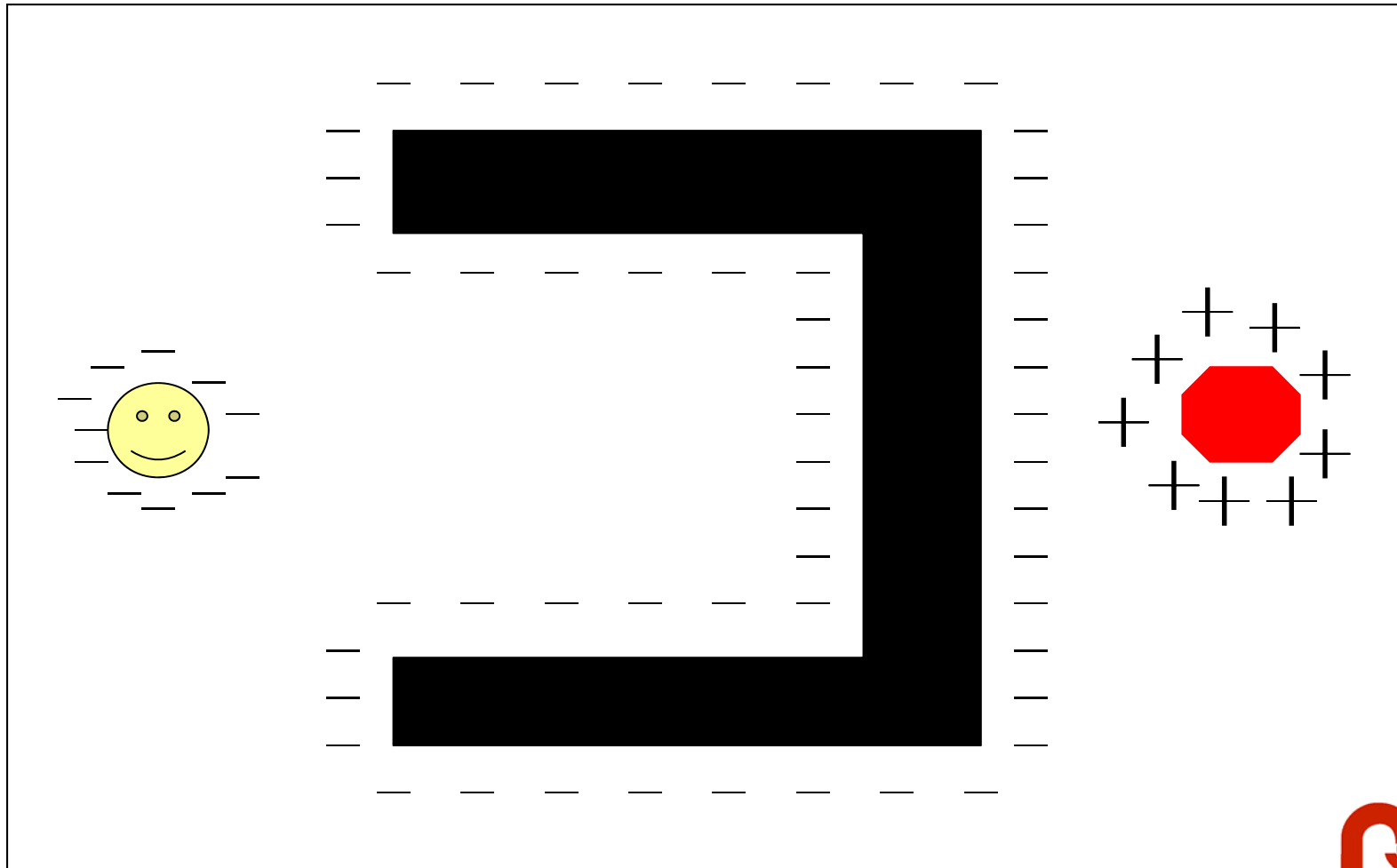


+



=

Local Minimum Problem with the Charge Analogy

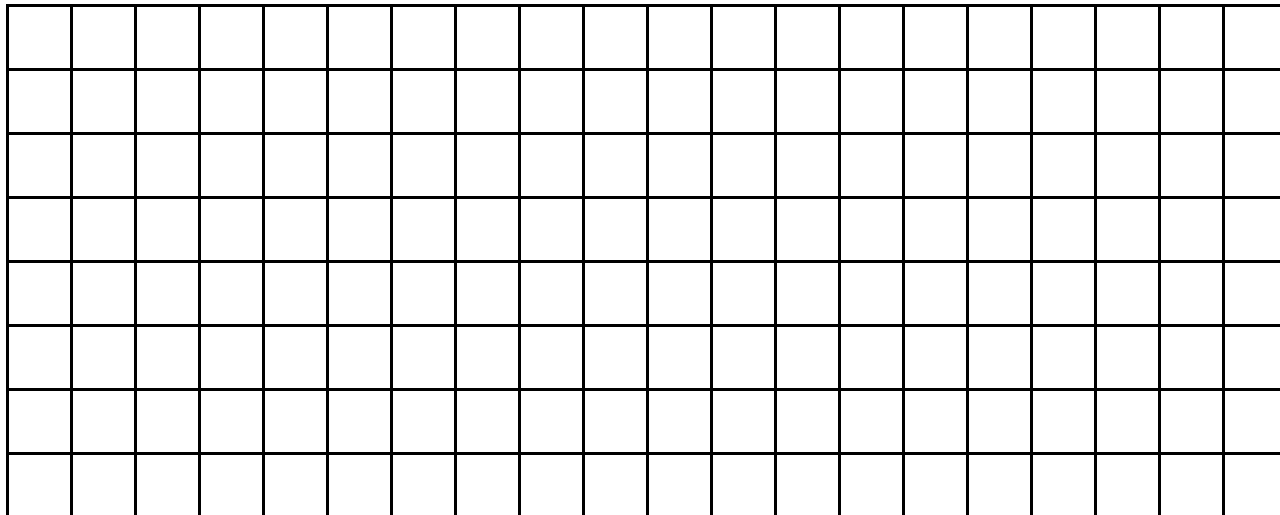


The Wavefront Planner

- A common algorithm used to determine the shortest paths between two points
 - In essence, a breadth first search of a graph
- For simplification, we'll present the world as a two-dimensional grid
- Setup:
 - Label free space with 0
 - Label start as START
 - Label the destination as 2

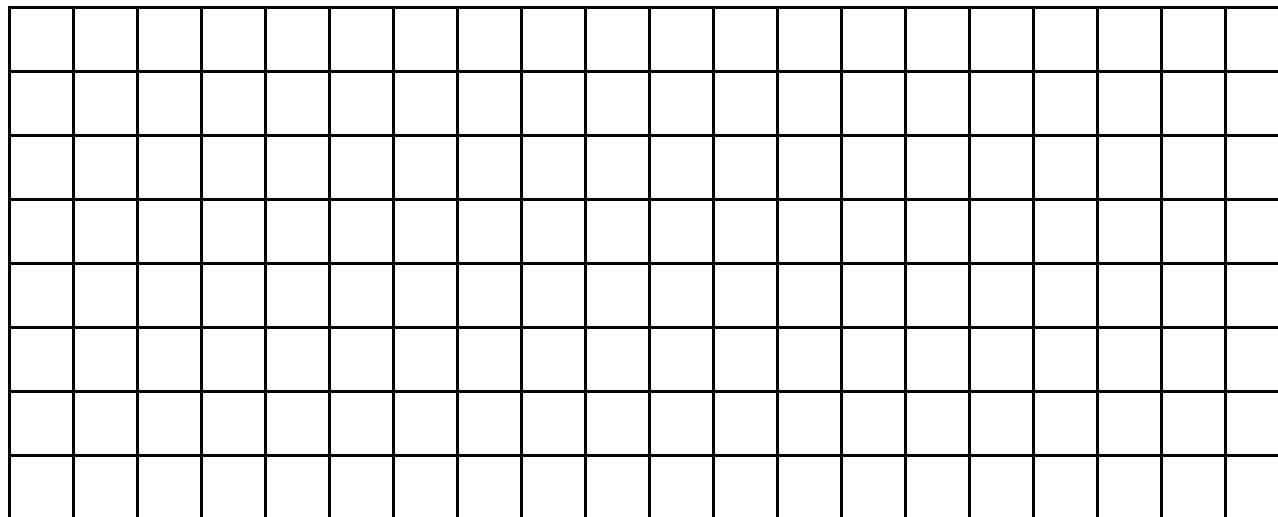
Representations

- World Representation
 - You could always use a large region and distances
 - However, a grid can be used for simplicity



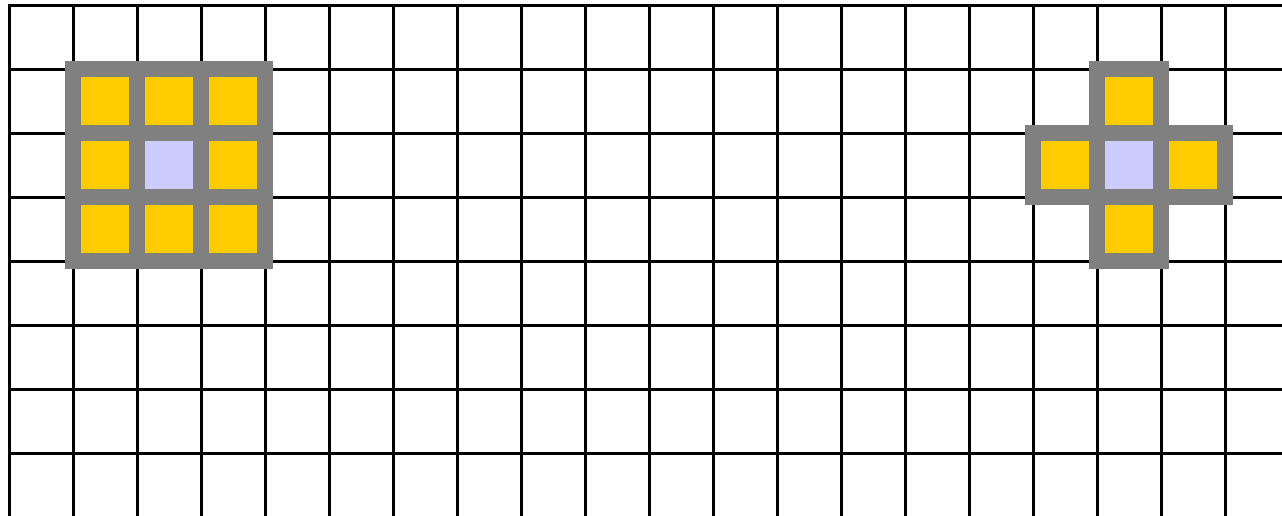
Representations: A Grid

- Distance is reduced to discrete steps
 - For simplicity, we'll assume distance is uniform
- Direction is now limited from one adjacent cell to another
 - Time to revisit Connectivity (Remember Vision?)



Representations: Connectivity

- 8-Point Connectivity
- 4-Point Connectivity
 - (approximation of the $L1$ metric)



The Wavefront Planner: Setup

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 1)

- Starting with the goal, set all adjacent cells with “0” to the current cell + 1
 - 4-Point Connectivity or 8-Point Connectivity?
 - Your Choice. We'll use 8-Point Connectivity in our example

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 2)

- Now repeat with the modified cells
 - This will be repeated until no 0's are adjacent to cells with values ≥ 2
 - 0's will only remain when regions are unreachable

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

The Wavefront in Action (Part 3)

- Repeat again...

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	0	5	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3
0	0	0	0	0	0	0	0	0	0	0	0	0	5	4	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

The Wavefront in Action (Part 4)

- And again...

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	6	6	6
3	0	0	0	0	1	1	1	1	1	1	1	1	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	6	5	4	4
1	0	0	0	0	0	0	0	0	0	0	0	6	5	4	3
0	0	0	0	0	0	0	0	0	0	0	0	6	5	4	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

The Wavefront in Action (Part 5)

- And again until...

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	7	7	7	7	7
4	0	0	0	0	1	1	1	1	1	1	1	1	6	6	6
3	0	0	0	0	1	1	1	1	1	1	1	1	5	5	5
2	0	0	0	0	0	0	0	0	0	0	7	6	5	4	4
1	0	0	0	0	0	0	0	0	0	0	7	6	5	4	3
0	0	0	0	0	0	0	0	0	0	0	7	6	5	4	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

The Wavefront in Action (Done)

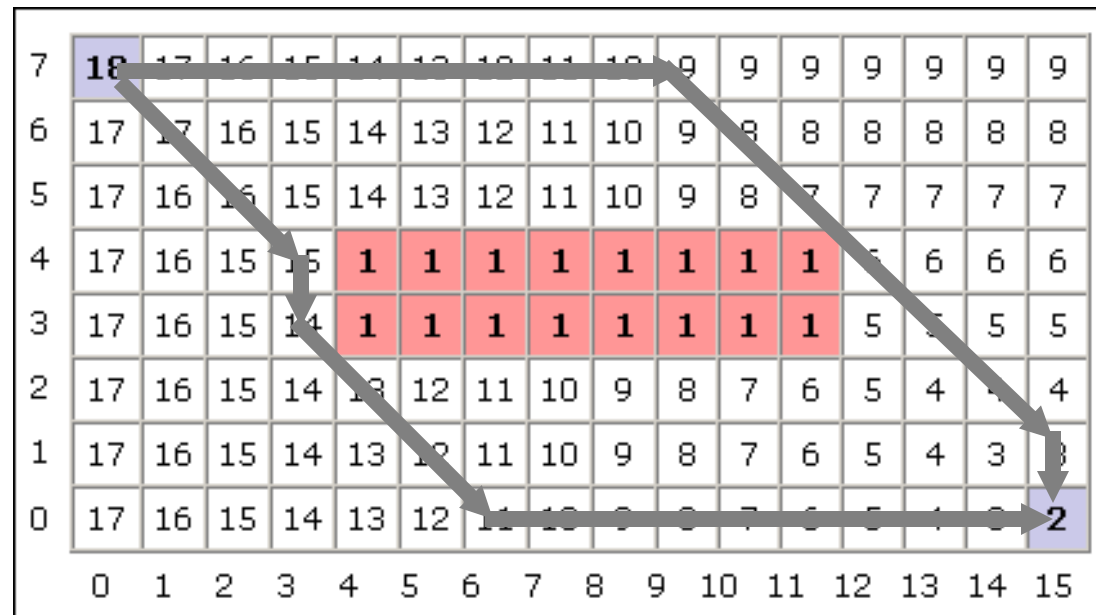
- You're done
 - Remember, 0's should only remain if unreachable regions exist

7	18	17	16	15	14	13	12	11	10	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7
4	17	16	15	15	1	1	1	1	1	1	1	1	6	6	6
3	17	16	15	14	1	1	1	1	1	1	1	1	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

The Wavefront, Now What?

- To find the shortest path, according to your metric, simply always move toward a cell with a lower number
 - The numbers generated by the Wavefront planner are roughly proportional to their distance from the goal

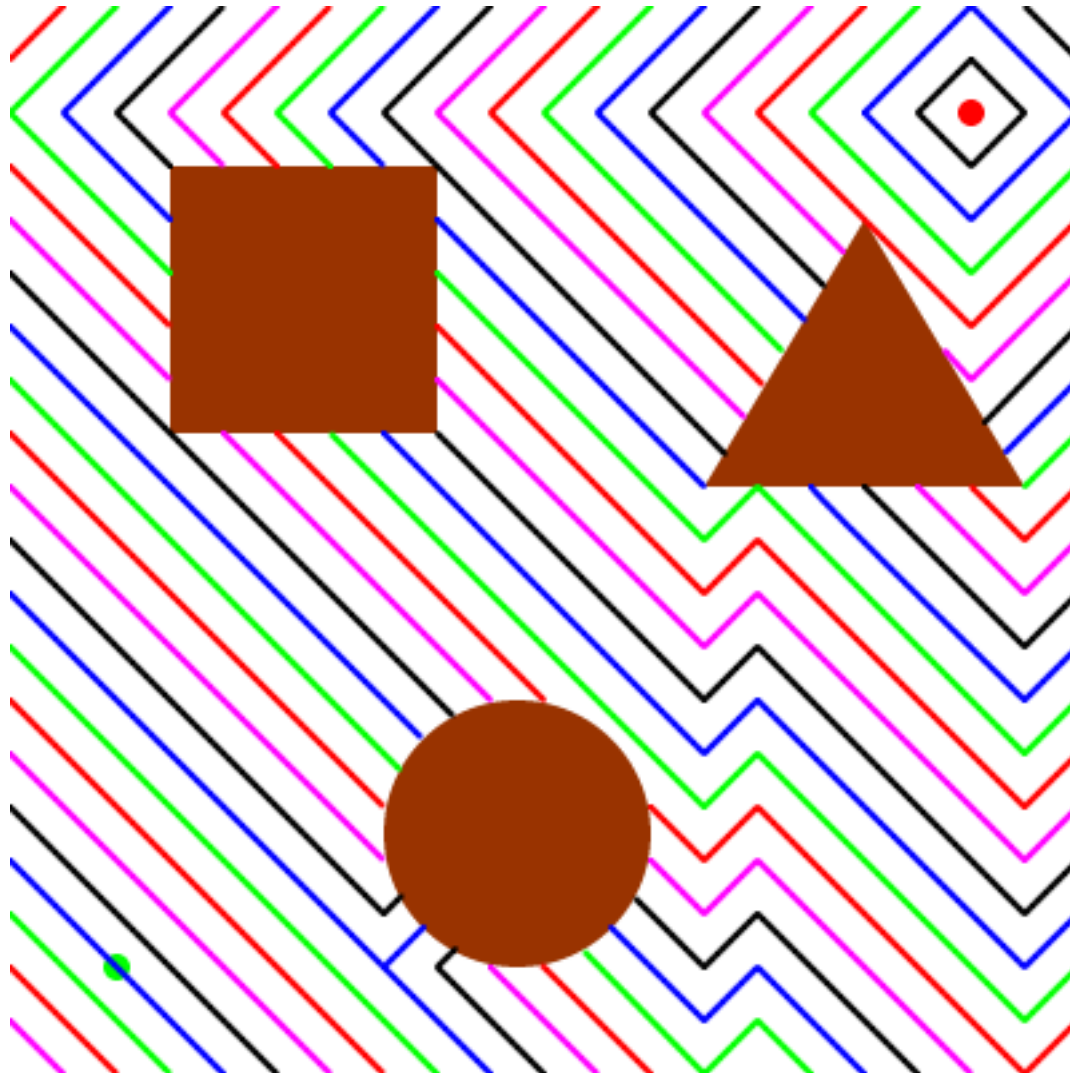
Two
possible
shortest
paths
shown



Wavefront (Overview)

- Divide the space into a grid.
- Number the squares starting at the start in either 4 or 8 point connectivity starting at the goal, increasing till you reach the start.
- Your path is defined by any uninterrupted sequence of decreasing numbers that lead to the goal.

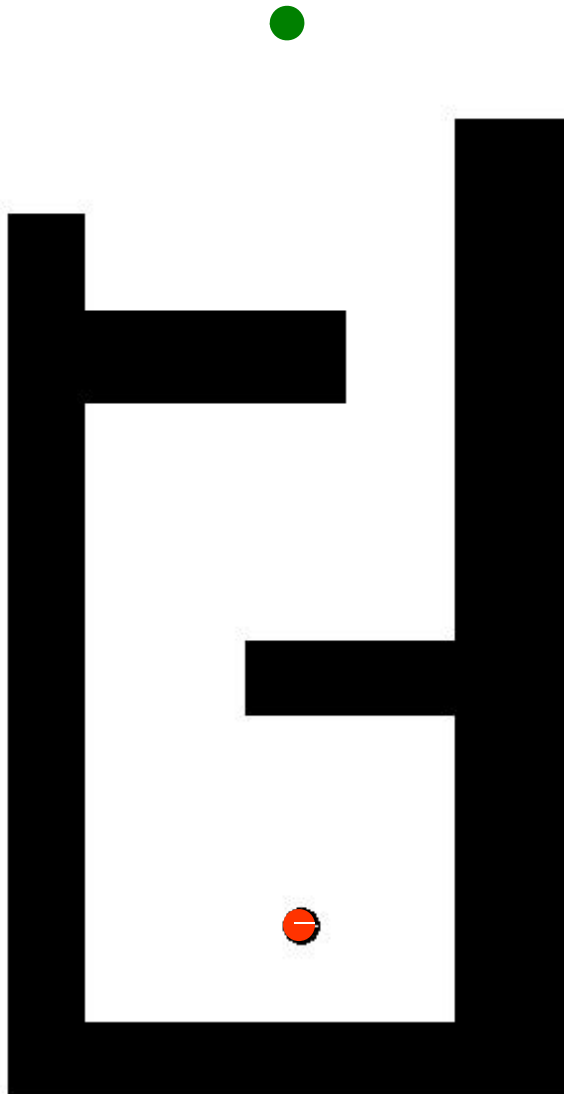
Wavefront Slide



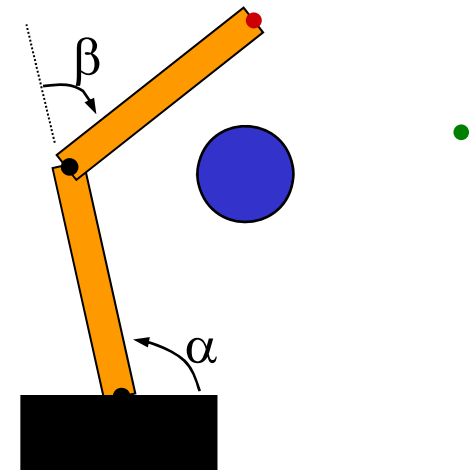
Return to Configuration Spaces

- Non-Euclidean
- Non-Planar

What if the robot is not a point?



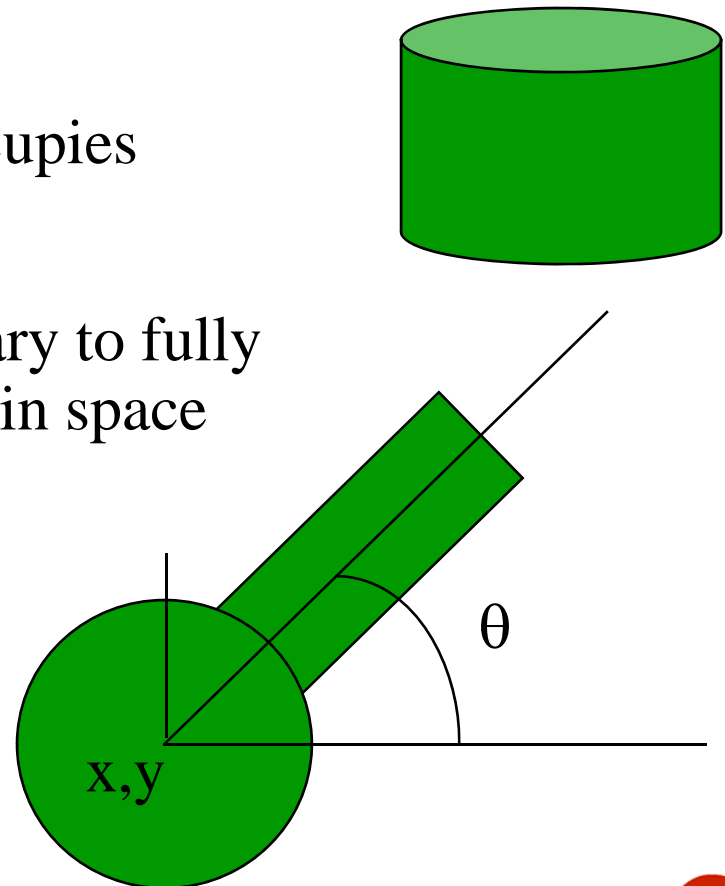
The Scout should probably not be modeled as a point...



Nor should robots with extended linkages that may contact obstacles...

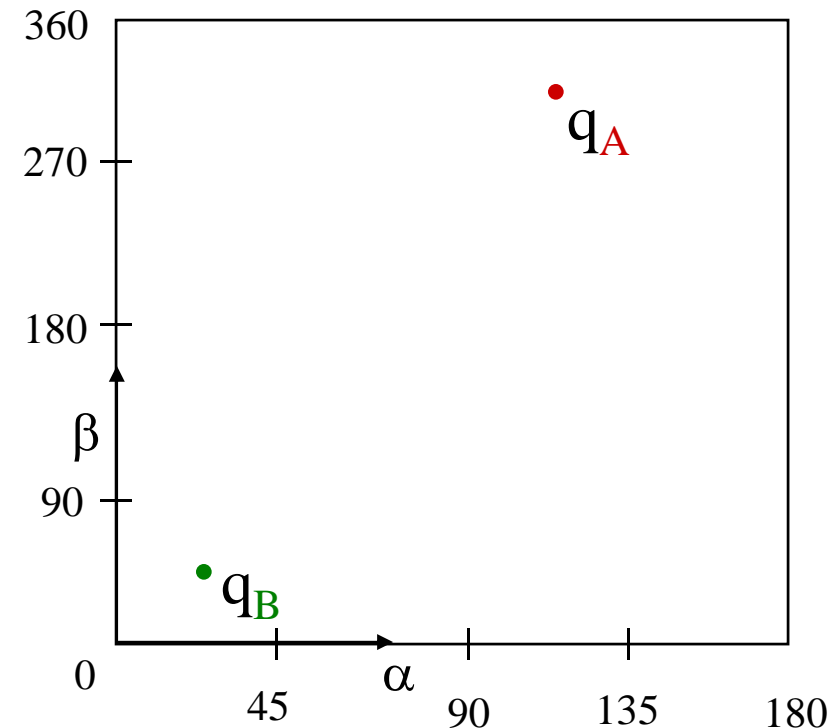
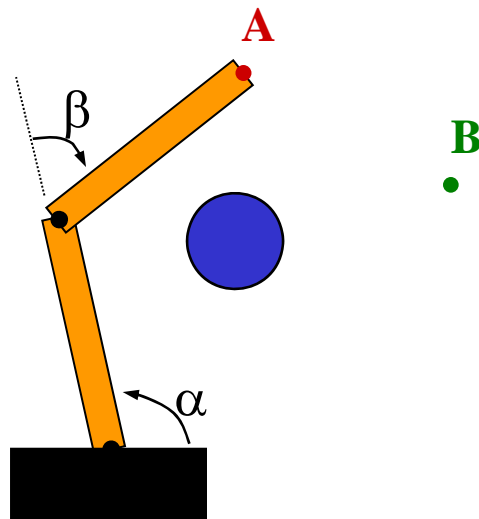
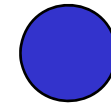
Configuration Space: the robot has...

- A Footprint
 - The amount of space a robot occupies
- Degrees of Freedom
 - The number of variables necessary to fully describe a robot's configuration in space
 - You'll cover this more in depth later
 - *fun with non-holonomic constraints, etc*



Configuration Space “Quiz”

Where do we put ?



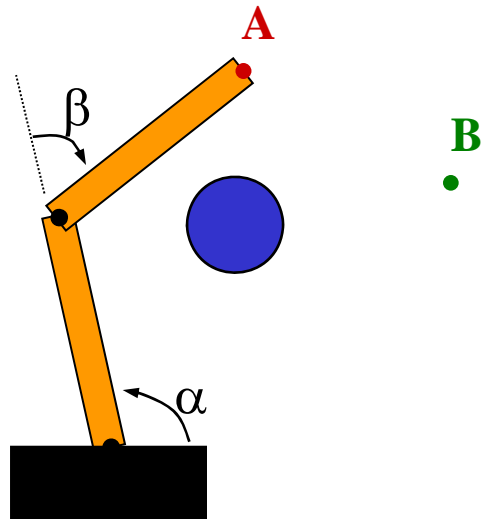
An obstacle in the robot's workspace

Torus

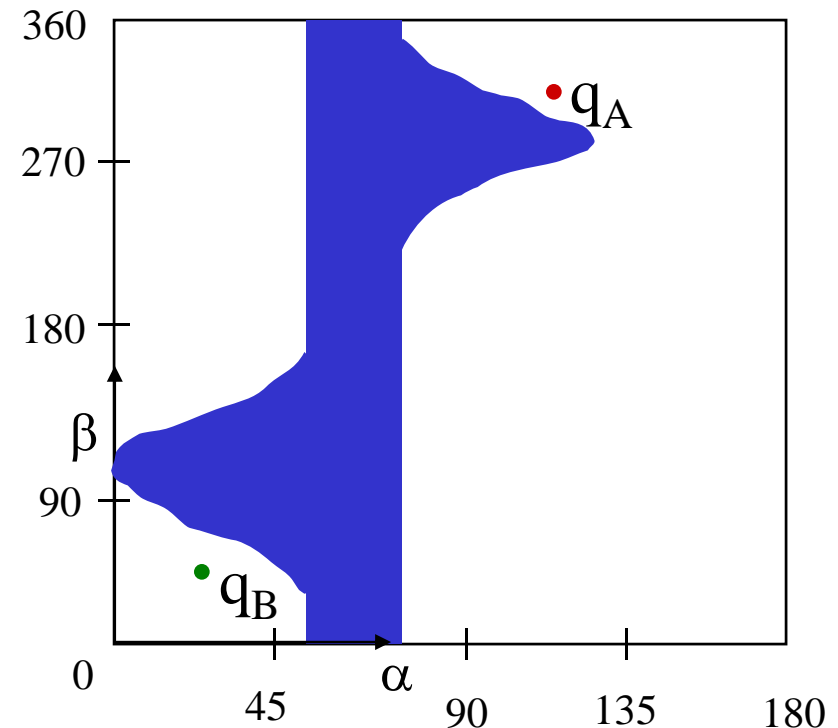
(wraps horizontally and vertically)

Configuration Space Obstacle

Reference configuration



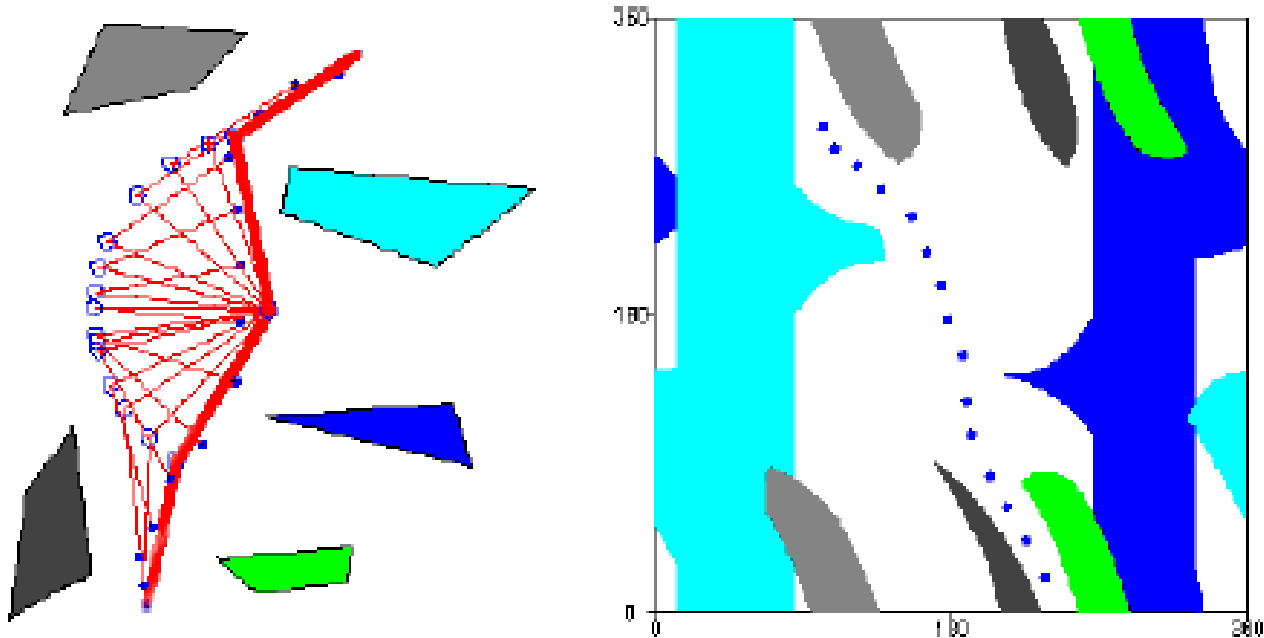
How do we get from **A** to **B** ?



An obstacle in the robot's workspace

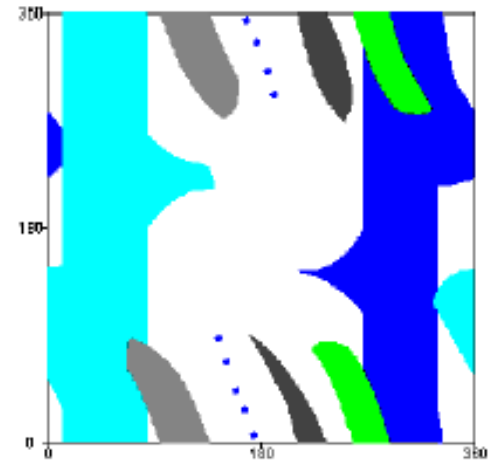
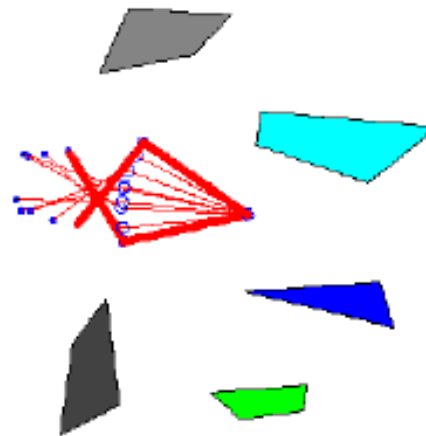
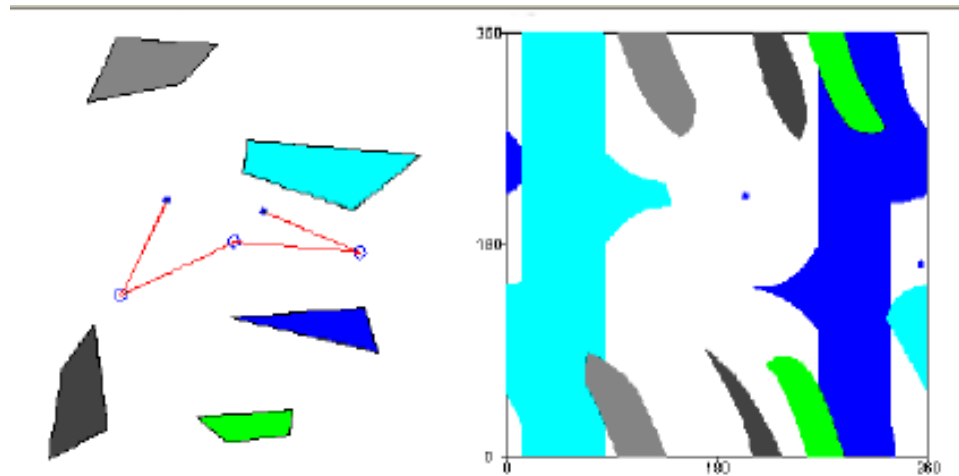
The C-space representation
of this obstacle...

Two Link Path



Thanks to Ken Goldberg

Two Link Path



More Example Configuration Spaces (contrasted with workspace)

- Free moving (no wheels) robot in plane:
 - workspace \mathbb{R}^2
 - configuration space \mathbb{R}^2
- 3-joint revolute arm in the plane
 - Workspace, a torus of outer radius $L1 + L2 + L3$
 - configuration space $T^3 = S^1 \times S^1 \times S^1$
- 2-joint revolute arm with a prismatic joint in the plane
 - workspace disc of radius $L1 + L2 + L3$
 - configuration space $T^2 \times \mathbb{R}$
- 3-joint revolute arm mounted on a mobile robot (holonomic)
 - workspace is a “sandwich” of radius $L1 + L2 + L3$
 - $\mathbb{R}^2 \times T^3$
- 3-joint revolute arm floating in space
 - workspace is \mathbb{R}^3
 - configuration space is $SE(3) \times T^3$

Map-Based Approaches: Roadmap Theory

- Properties of a roadmap:
 - Accessibility: there exists a collision-free path from the start to the road map
 - Departability: there exists a collision-free path from the roadmap to the goal.
 - Connectivity: there exists a collision-free path from the start to the goal (on the roadmap).



- a roadmap exists \Leftrightarrow a path exists
- Examples of Roadmaps
 - Generalized Voronoi Graph (GVG)
 - Visibility Graph

Two-Equidistant

- *Two-equidistant surface*

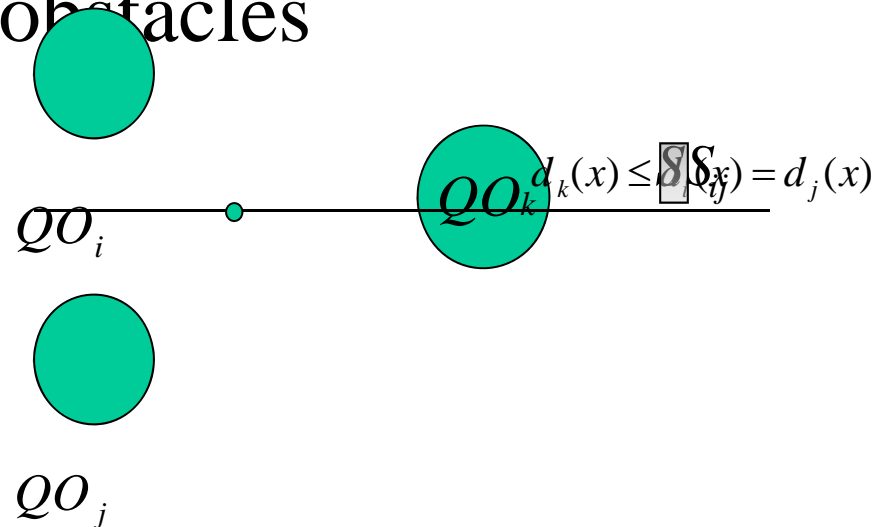
$$S_{ij} = \{x \in Q_{\text{free}} : d_i(x) - d_j(x) = 0\}$$

QO_i

QO_j

More Rigorous Definition

Going through obstacles

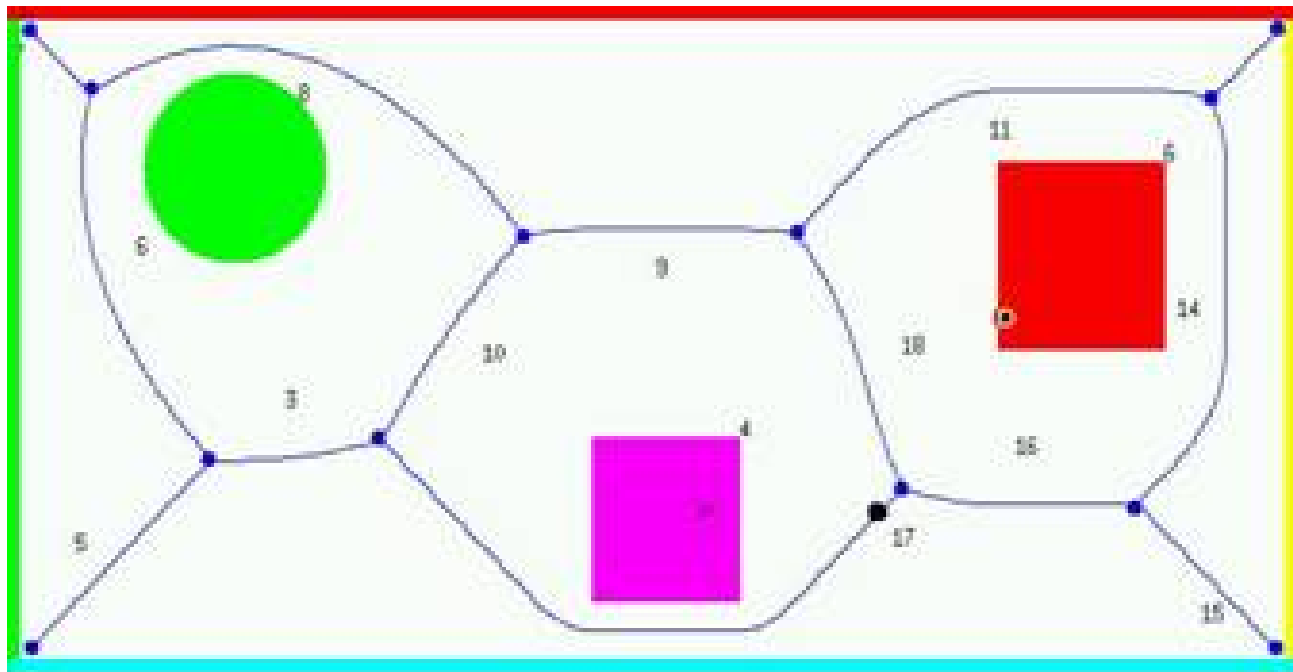


Two-equidistant face

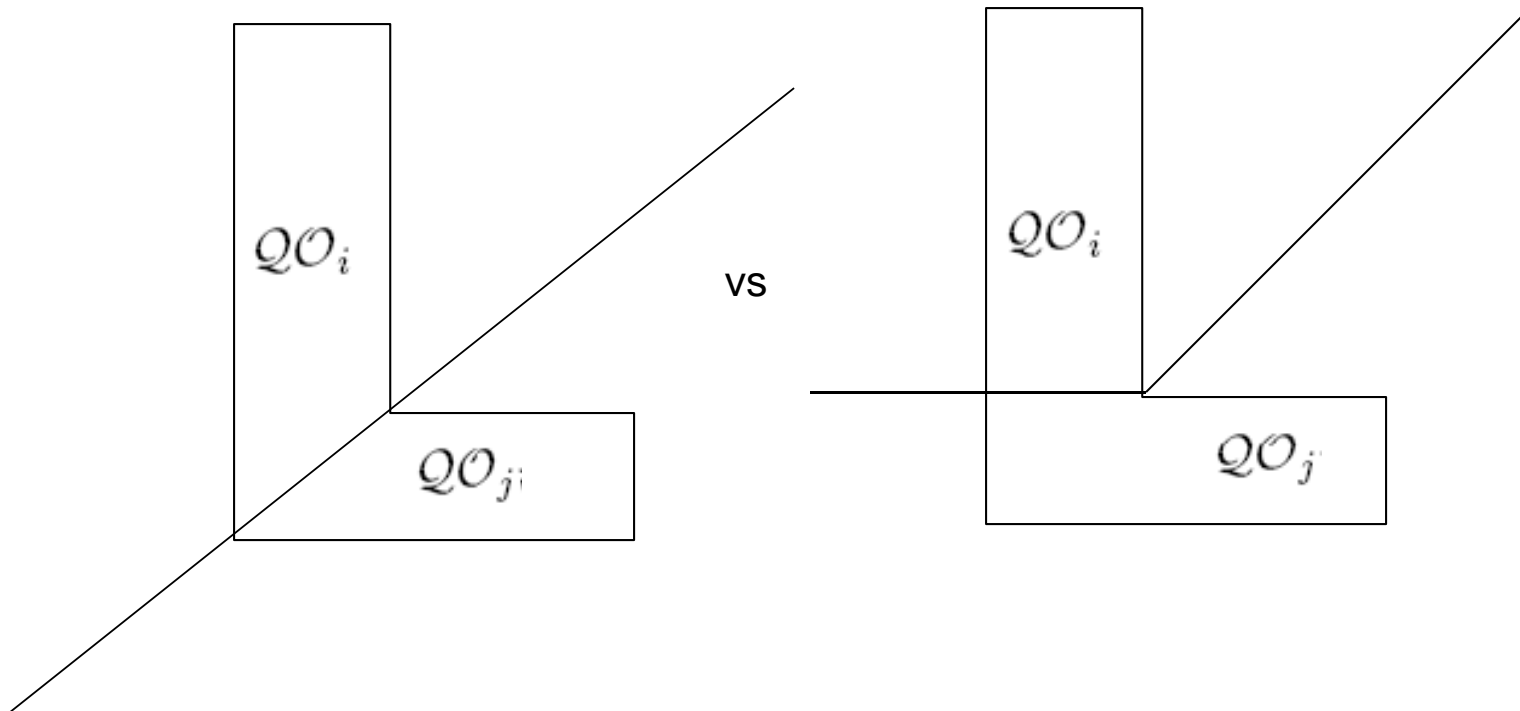
$$F_{ij} = \{x \in \mathbb{S}_{ij} : d_i(x) = d_j(x) \leq d_h(x), \forall h \neq i, j\}$$

General Voronoi Diagram

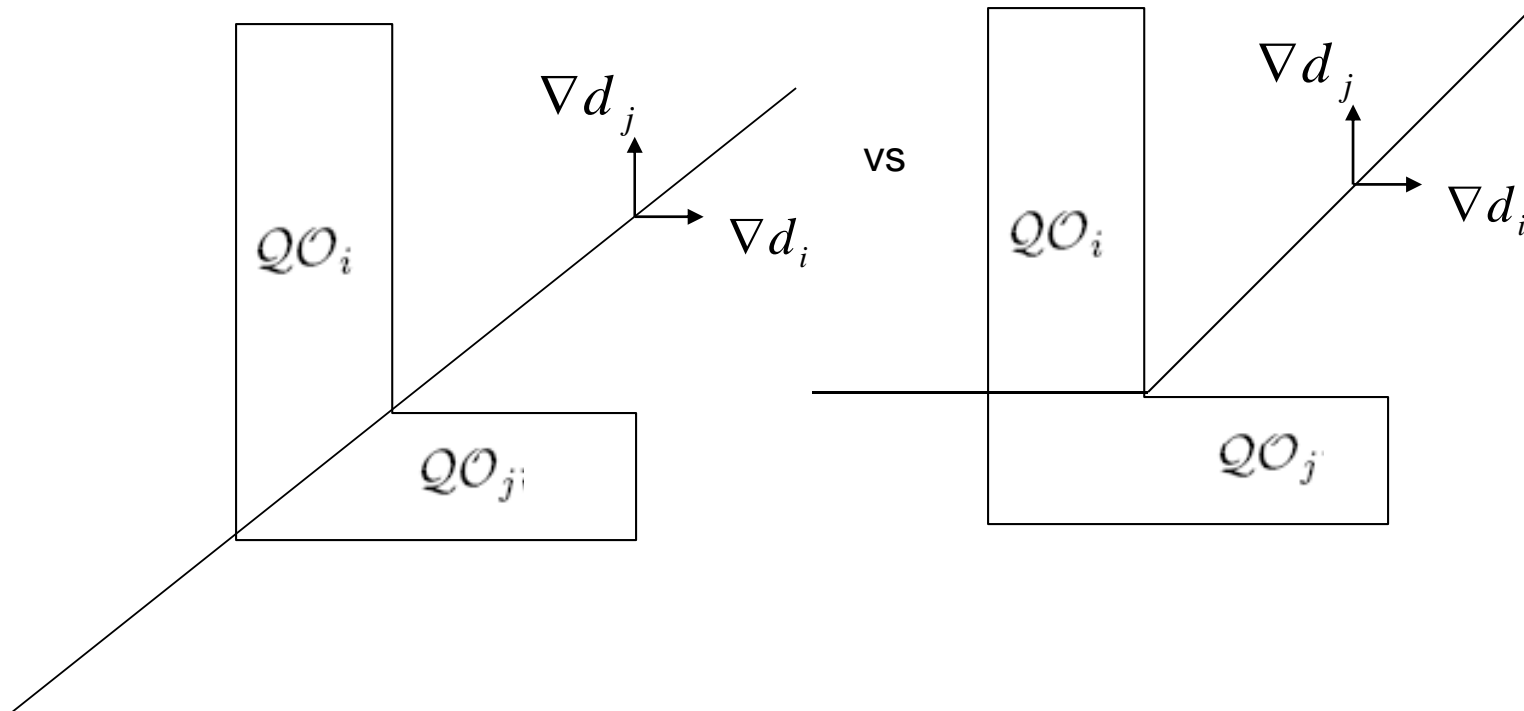
$$\text{GVD} = \bigcup_{i=1}^{n-1} \bigcup_{j=i+1}^n F_{ij}$$



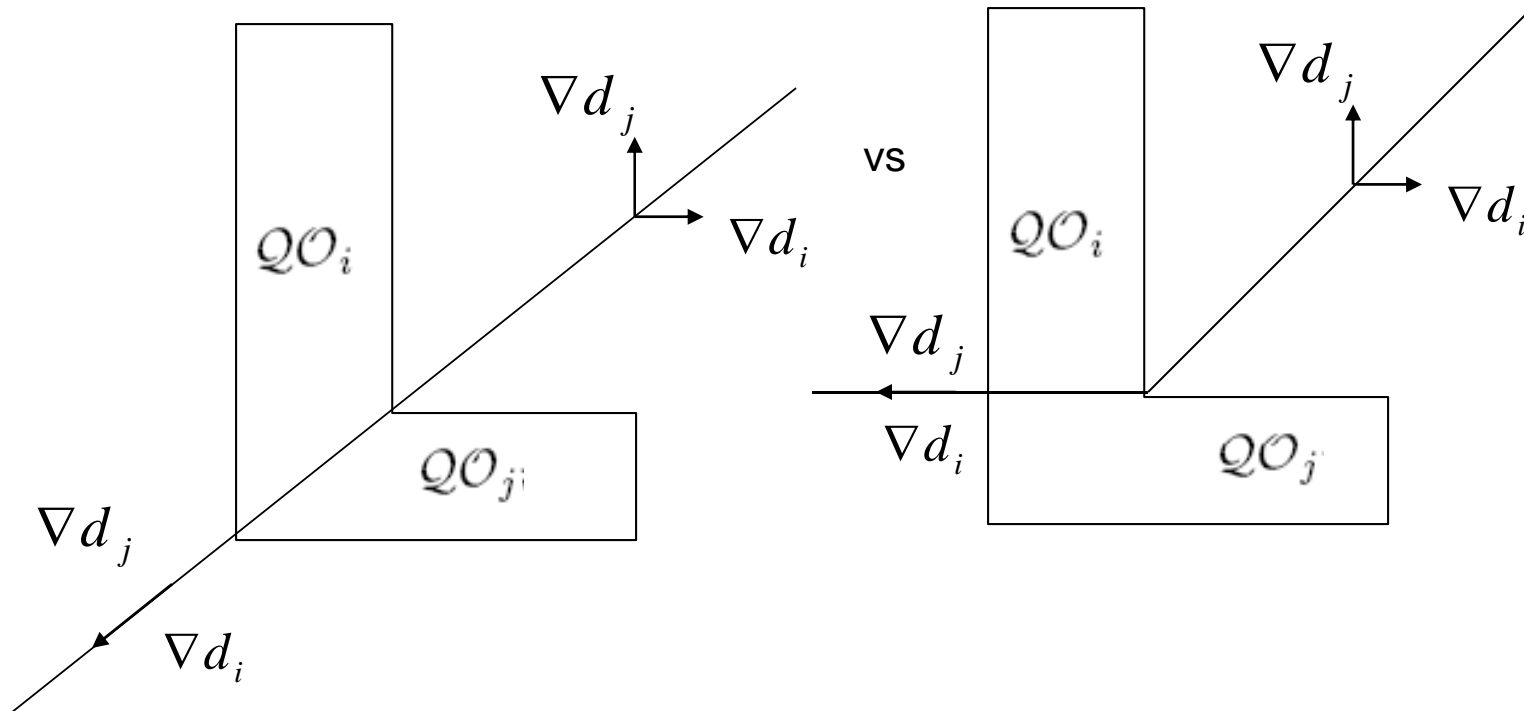
What about concave obstacles?



What about concave obstacles?



What about concave obstacles?



Two-Equidistant

- *Two-equidistant surface*

$$S_{ij} = \{x \in Q_{\text{free}} : d_i(x) - d_j(x) = 0\}$$

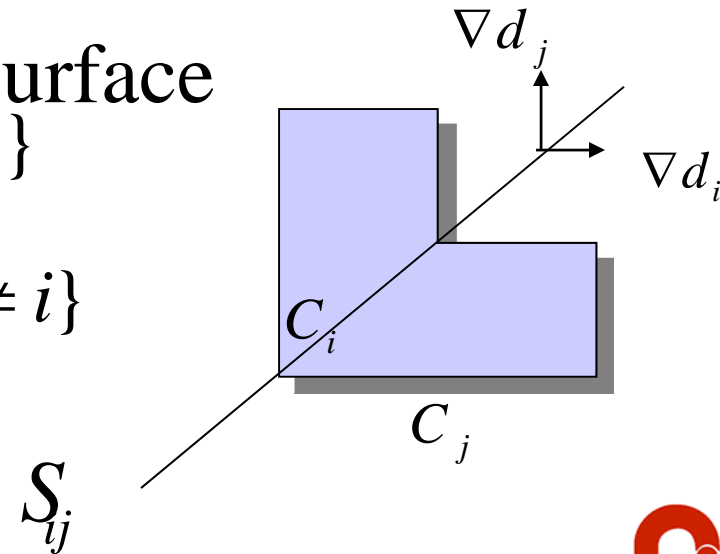
Two-equidistant surjective surface

$$SS_{ij} = \{x \in S_{ij} : \nabla d_i(x) \neq \nabla d_j(x)\}$$

$$F_{ij} = \{x \in SS_{ij} : d_i(x) \leq d_h(x), \forall h \neq i\}$$

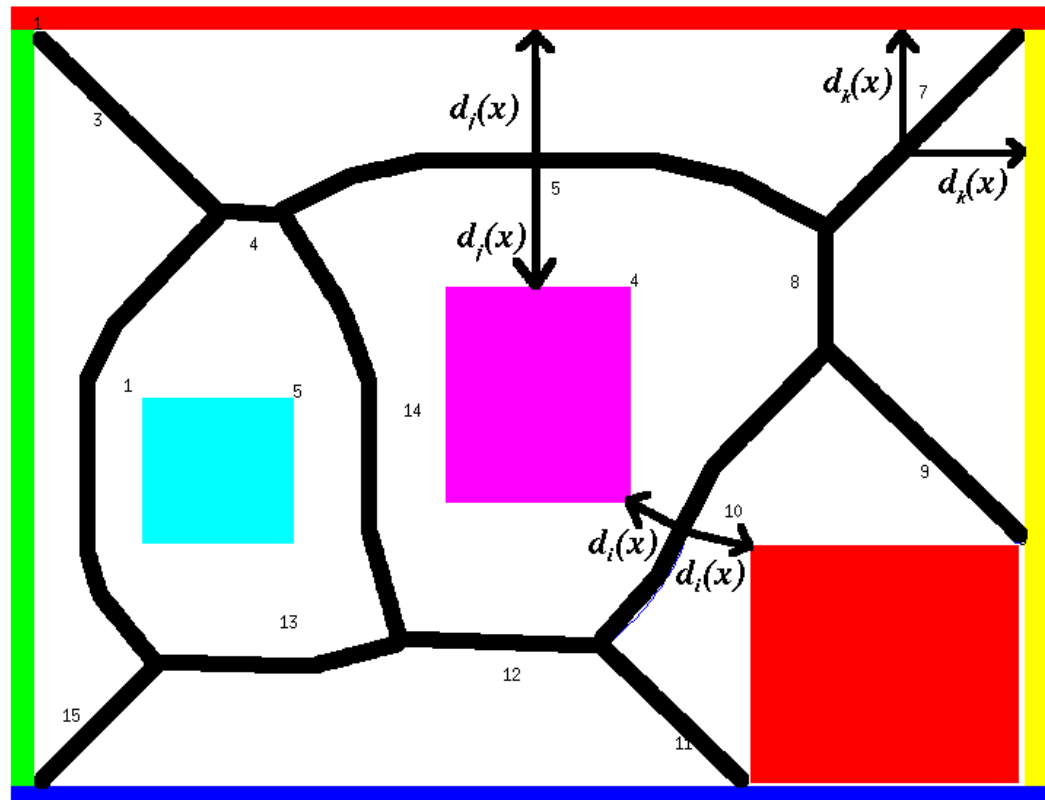
$$\text{GVD} = \bigcup_{i=1}^{n-1} \bigcup_{j=i+1}^n F_{ij}$$

Two-equidistant Face



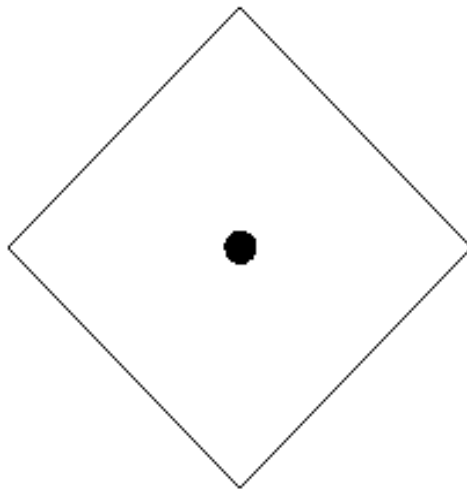
Roadmap: GVG

- A GVG is formed by paths equidistant from the two closest objects
- *Remember “spokes”, start and goal*



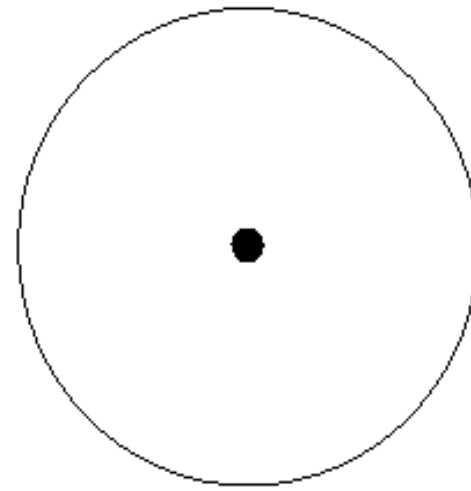
- This generates a very safe roadmap which avoids obstacles as much as possible

Voronoi Diagram: Metrics



$$\{(x,y) : |x| + |y| = \text{const}\}$$

L1

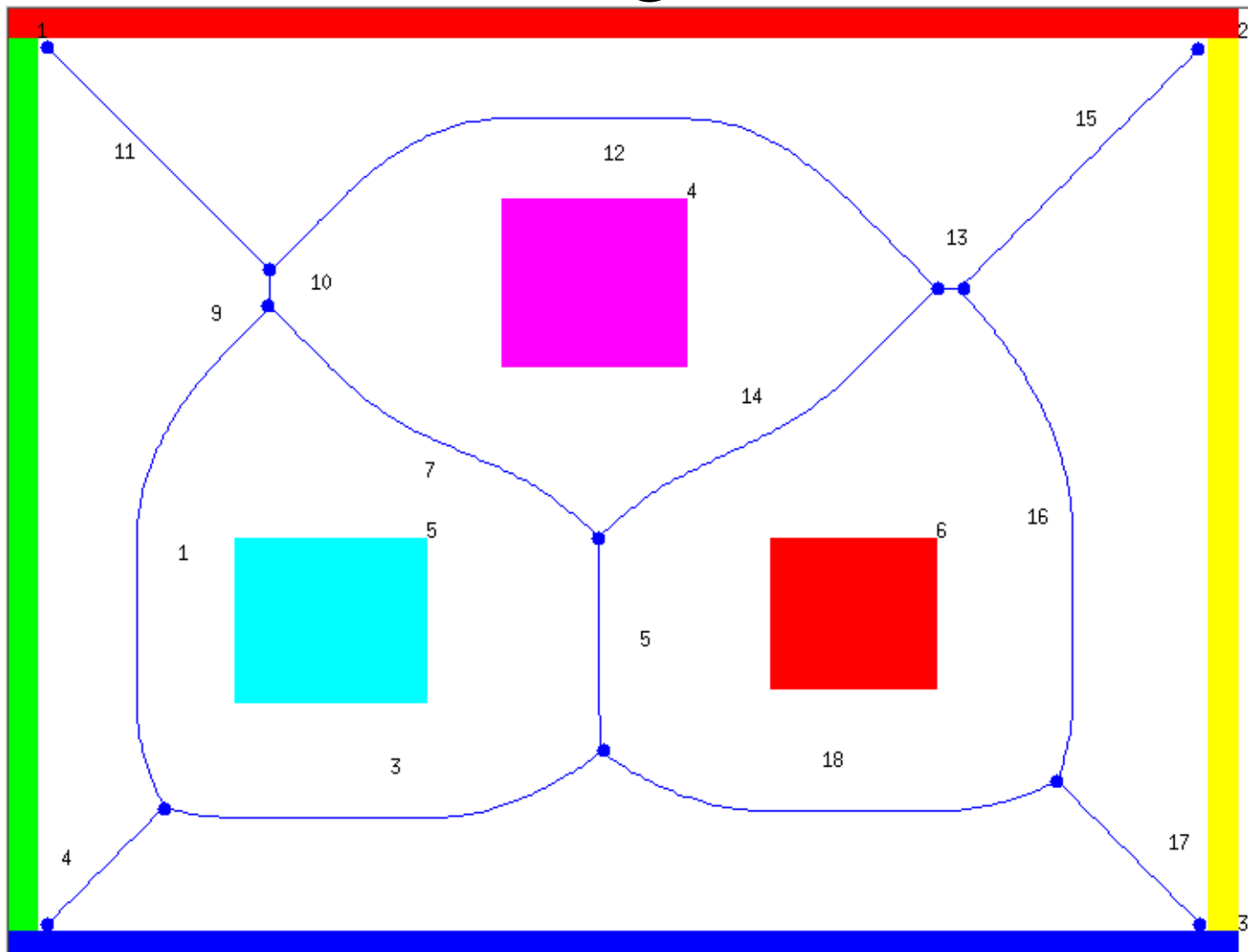


$$\{(x,y) : x^2 + y^2 = \text{const}\}$$

L2

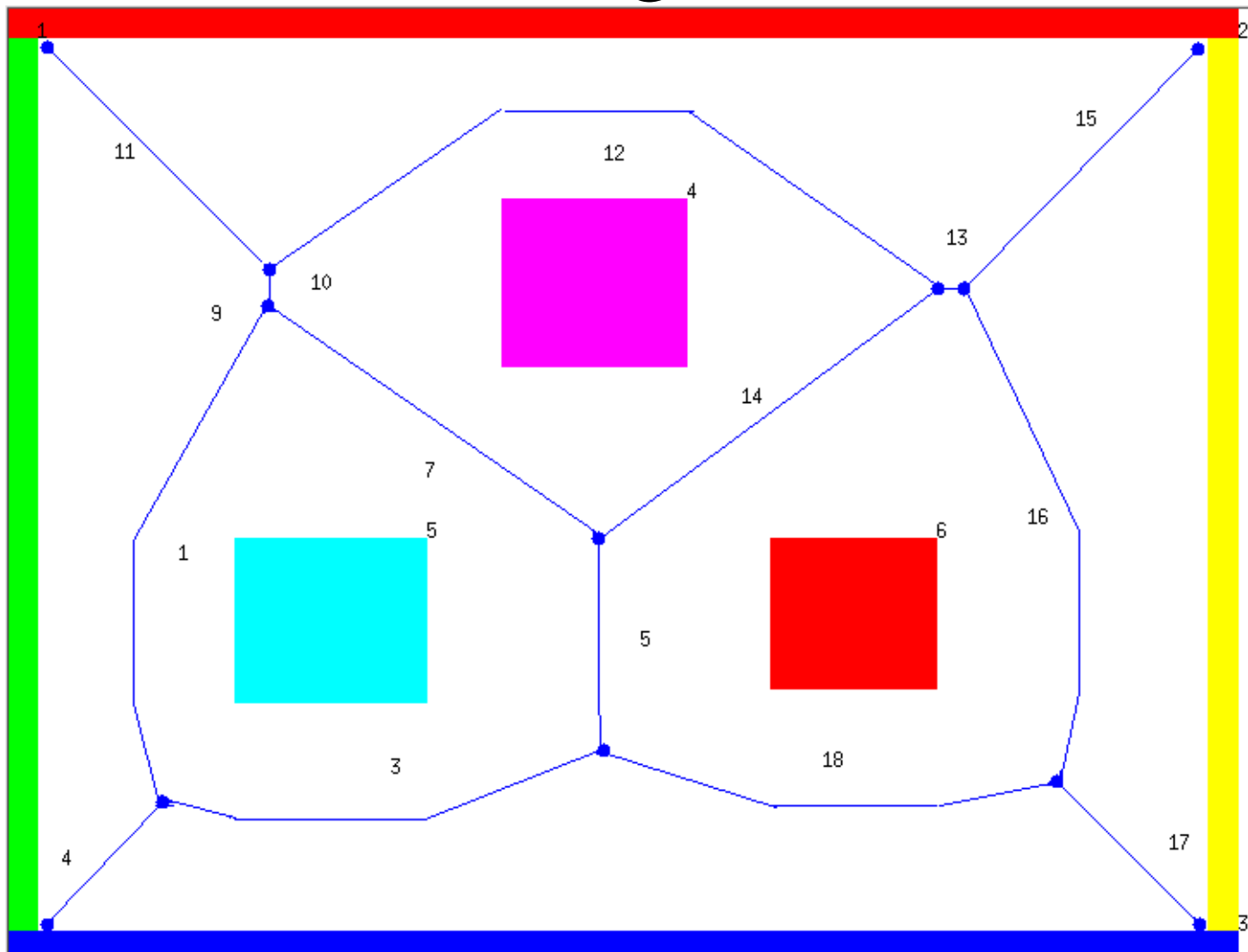
Voronoi Diagram (L2)

Note the
curved
edges



Voronoi Diagram (L1)

Note the
lack of
curved
edges

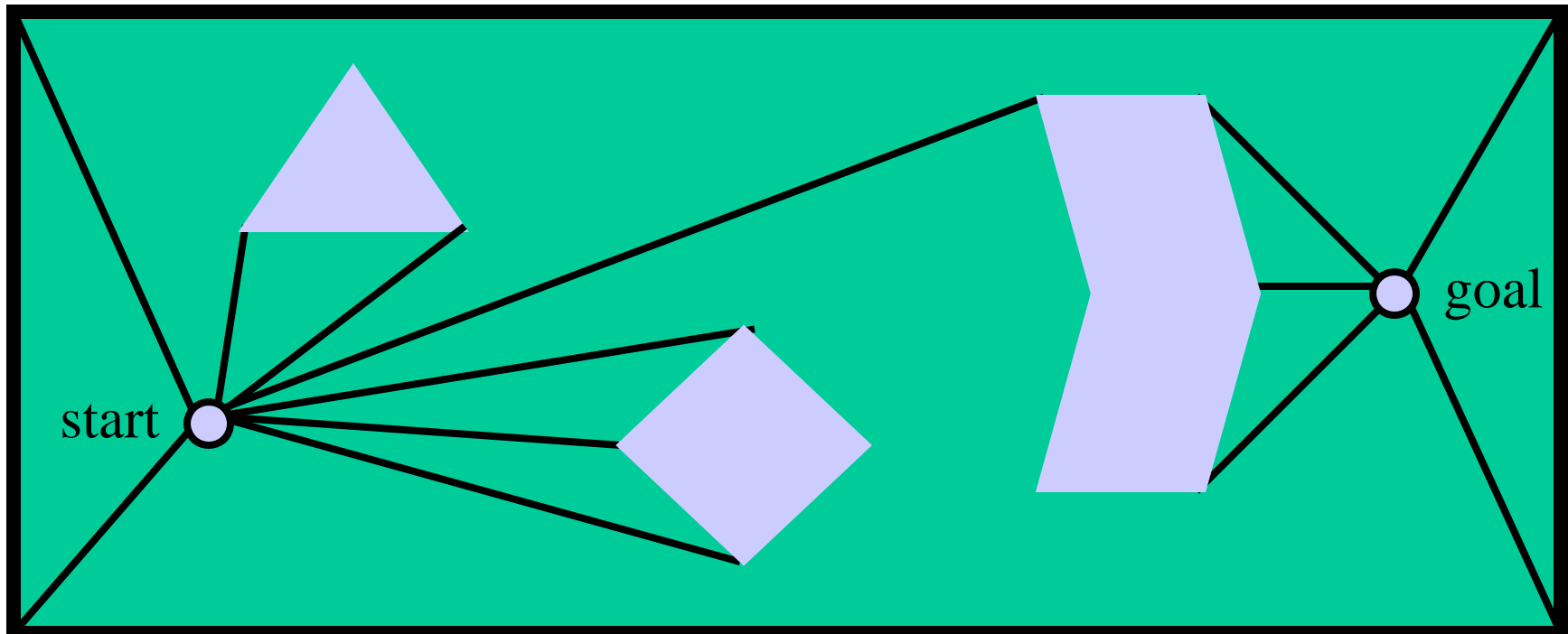


Roadmap: Visibility Graph

- Formed by connecting all “visible” vertices, the start point and the end point, to each other
- For two points to be “visible” no obstacle can exist between them
 - Paths exist on the perimeter of obstacles
- In our example, this produces the shortest path with respect to the L2 metric. However, the close proximity of paths to obstacles makes it dangerous

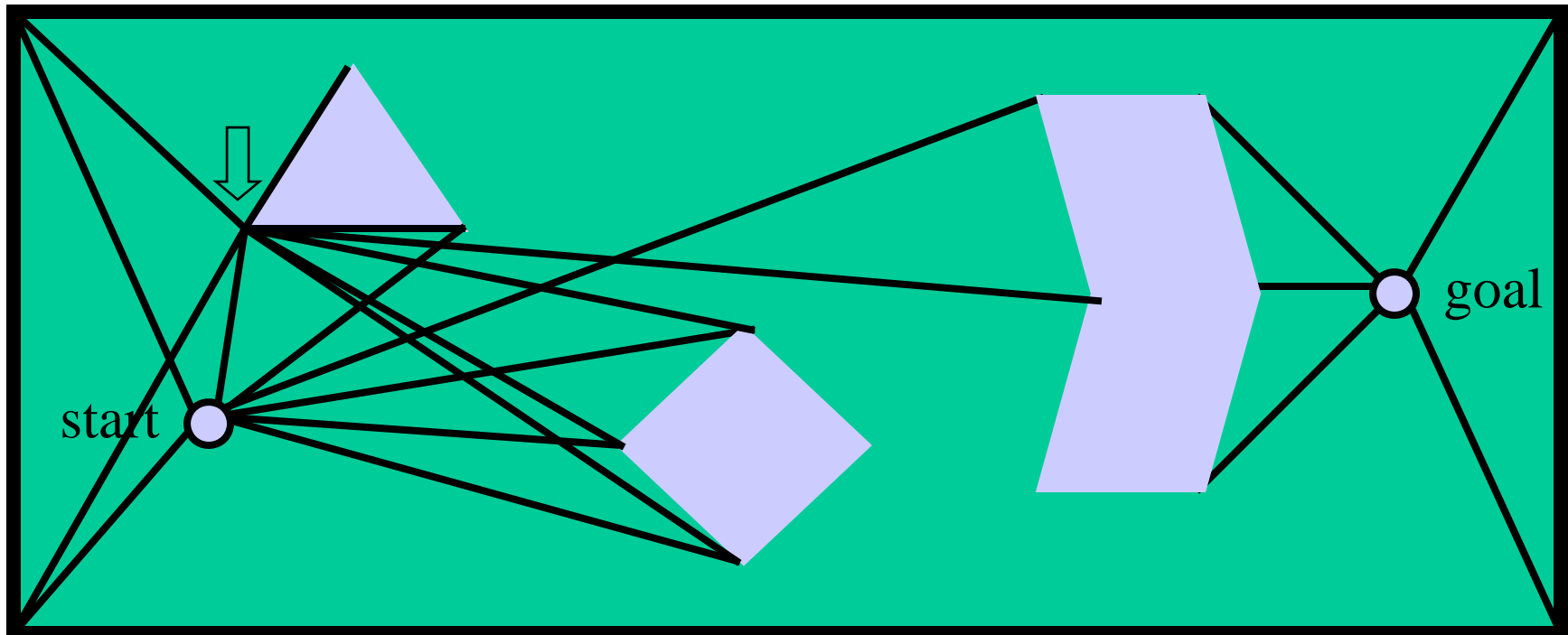
The Visibility Graph in Action (Part 1)

- First, draw lines of sight from the start and goal to all “visible” vertices and corners of the world.



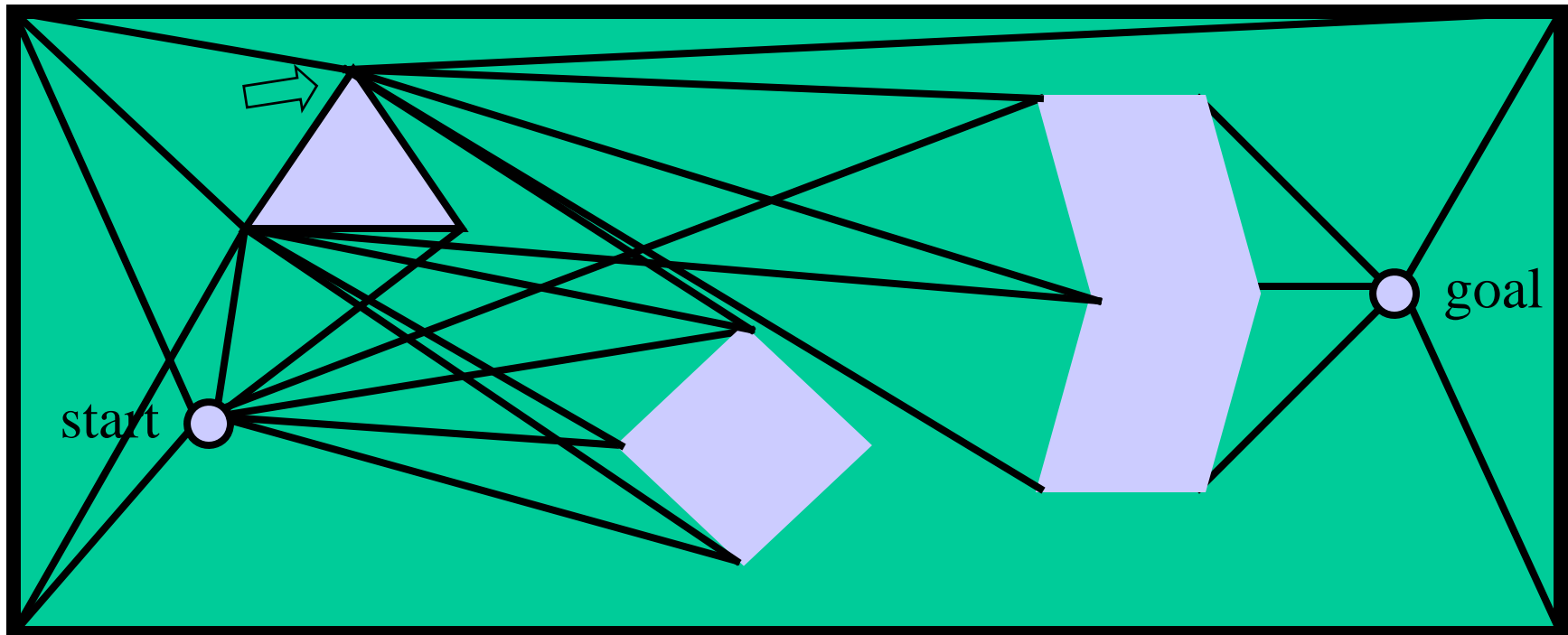
The Visibility Graph in Action (Part 2)

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



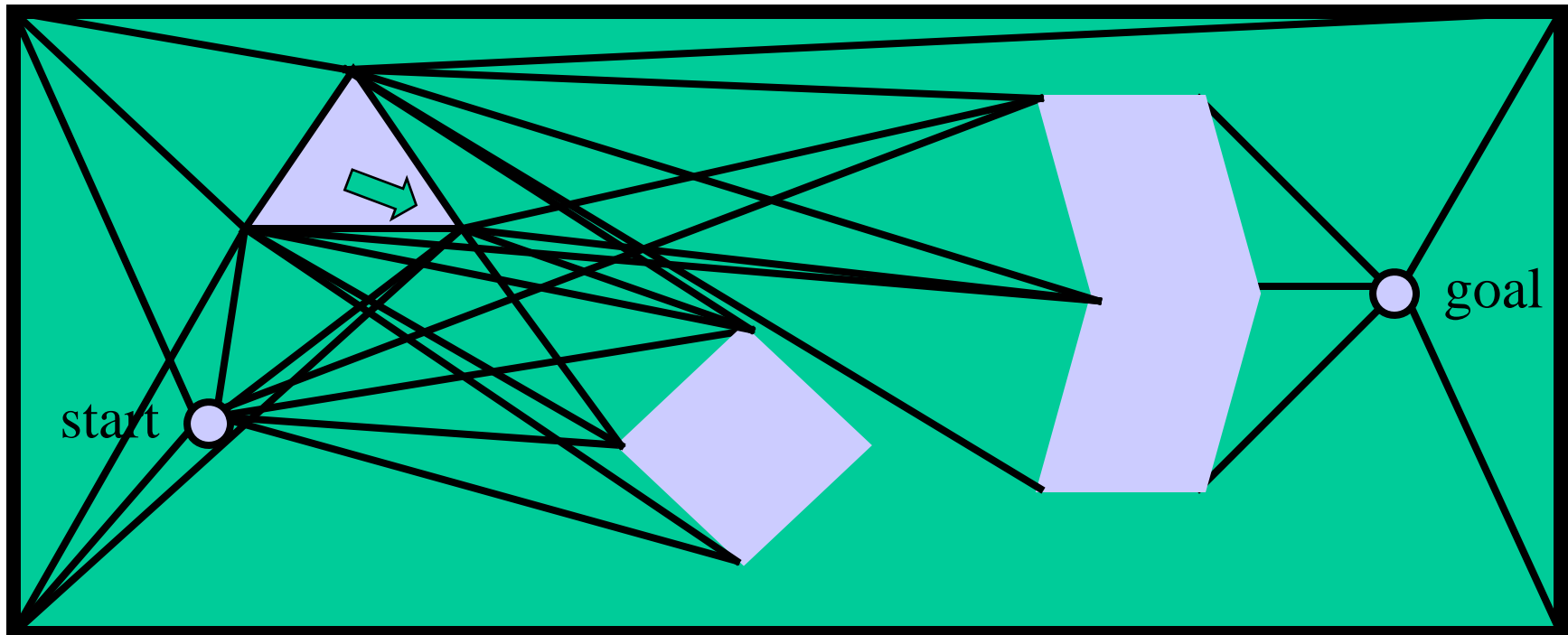
The Visibility Graph in Action (Part 3)

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



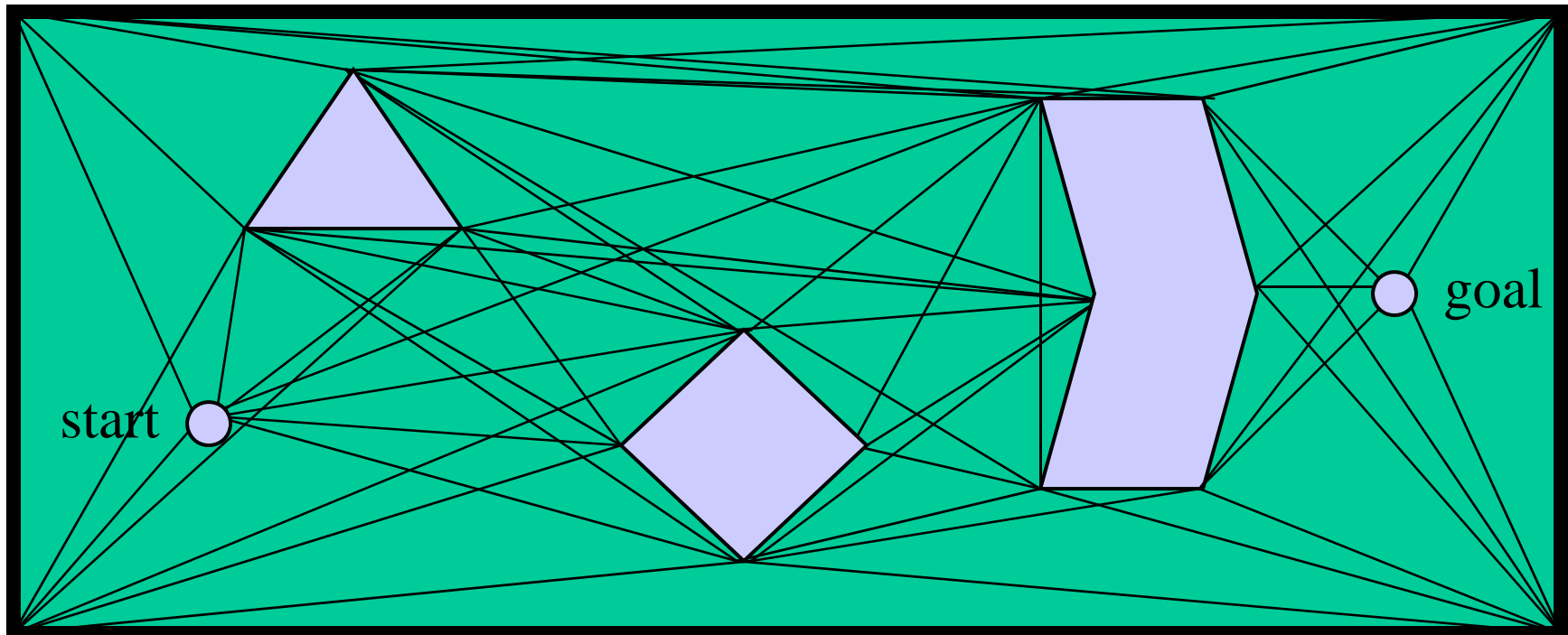
The Visibility Graph in Action (Part 4)

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



The Visibility Graph (Done)

- Repeat until you're done.

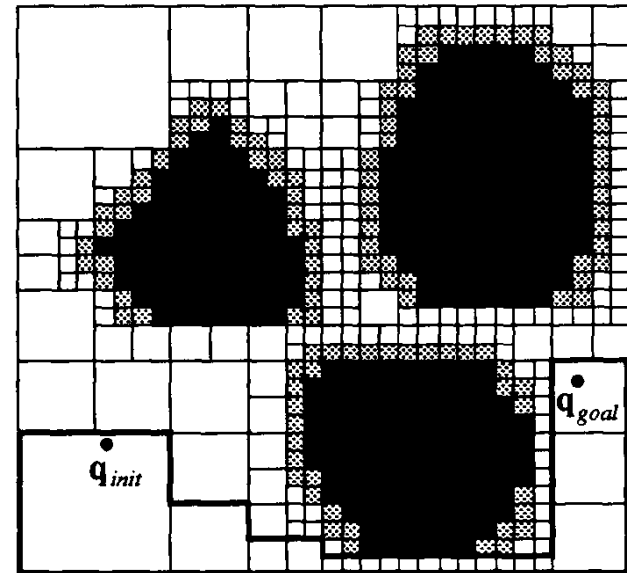
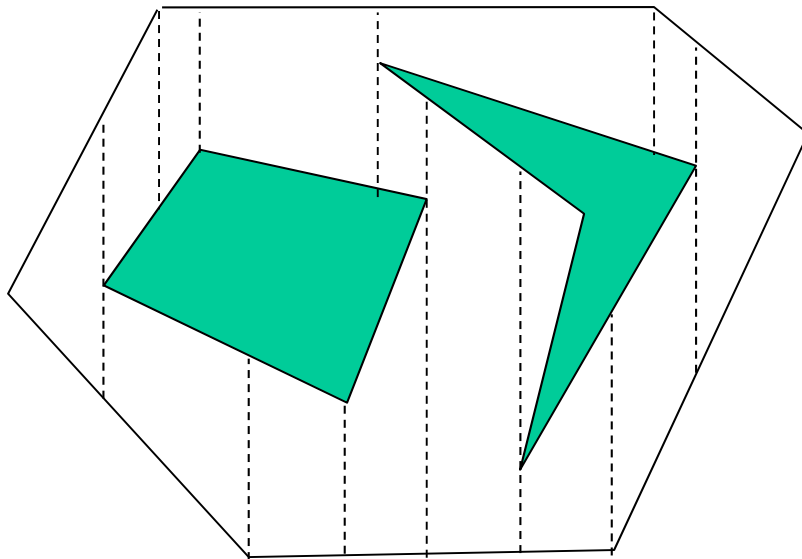


Visibility Graph Overview

- Start with a map of the world, draw lines of sight from the start and goal to every “corner” of the world and vertex of the obstacles, not cutting through any obstacles.
- Draw lines of sight from every vertex of every obstacle like above. Lines along edges of obstacles are lines of sight too, since they don’t pass through the obstacles.
- If the map was in Configuration space, each line potentially represents part of a path from the start to the goal.

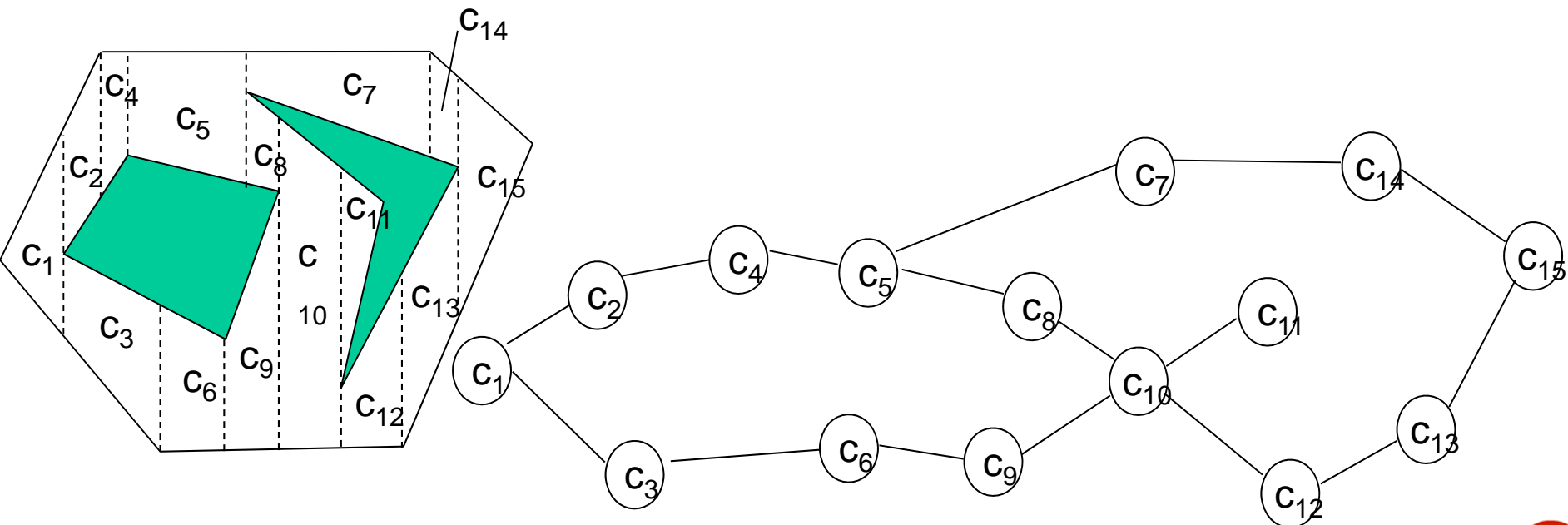
Exact Cell vs. Approximate Cell

- Cell: simple region



Adjacency Graph

- Node correspond to a cell
- Edge connects nodes of adjacent cells
- Two cells are *adjacent* if they share a common boundary



Set Notation

Some set notation

- Interior of A ($\text{int}(A)$) is the largest open subset of A
- Closure of A ($\text{cl}(A)$) is the smallest closed set that contains A
- Complement of A (\bar{A}) is everything not in A .
- Boundary of A (∂A) is the closure of A take away its interior.

Examples

Examples

- $\text{int}[0, 1] = (0, 1), \text{int}(0, 1) = (0, 1)$
- $\text{cl}[0, 1] = [0, 1], \text{cl}(0, 1) = [0, 1]$
- $\bar{[0, 1]} = (-\infty, 0) \cup (1, \infty)$
- $\partial[0, 1] = \partial(0, 1) = \{0, 1\}$

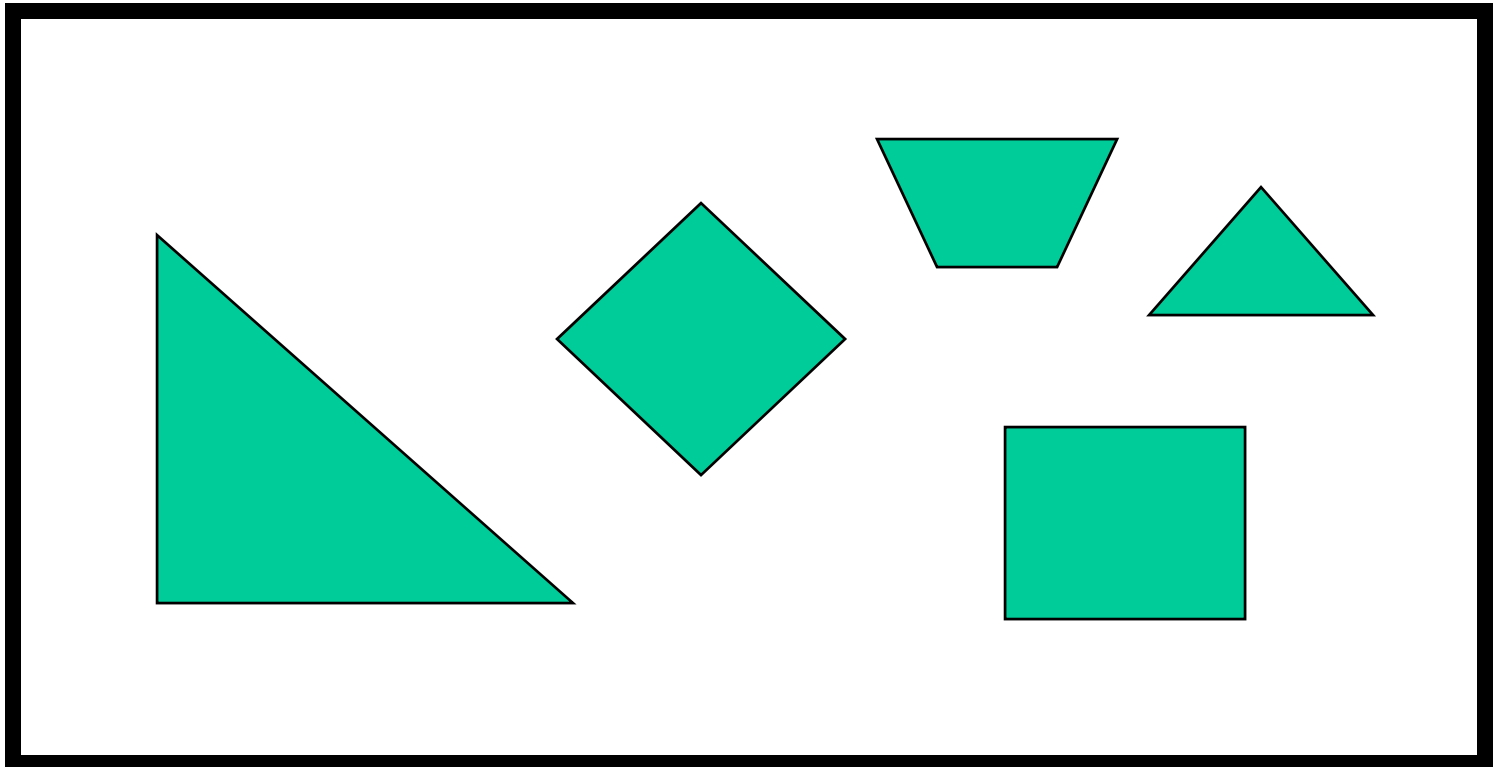
Definition

Exact Cellular Decomposition (as opposed to approximate)

- ν_i is a cell
- $\text{int}(\nu_i) \cap \text{int}(\nu_j) = \emptyset$ if and only if $i \neq j$
- $Fs \cap (\text{cl}(\nu_i) \cap \text{cl}(\nu_j)) \neq \emptyset$ if ν_i and ν_j are adjacent cells
- $Fs = \cup_i (\nu_i)$

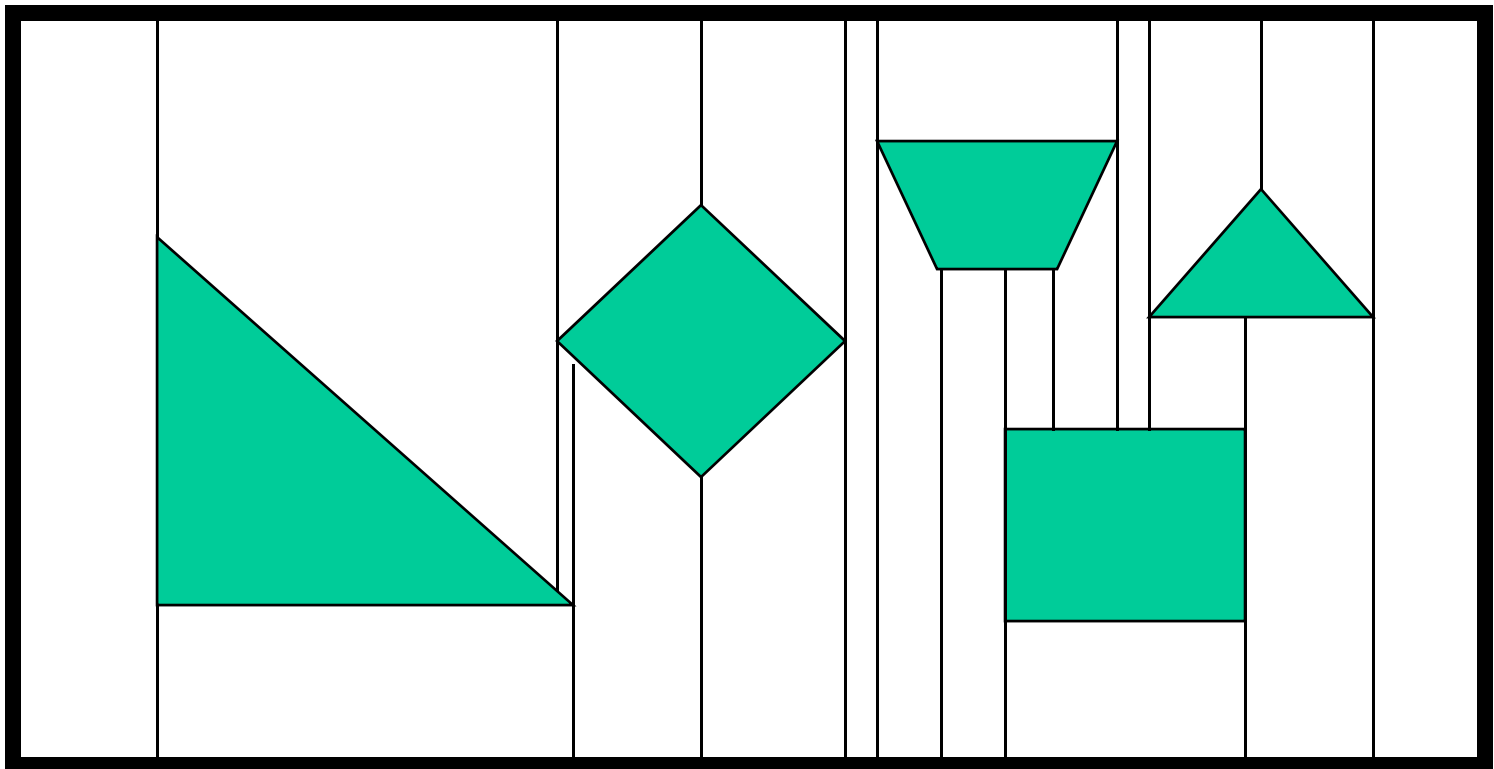
Cell Decompositions: Trapezoidal Decomposition

- A way to divide the world into smaller regions
- Assume a polygonal world



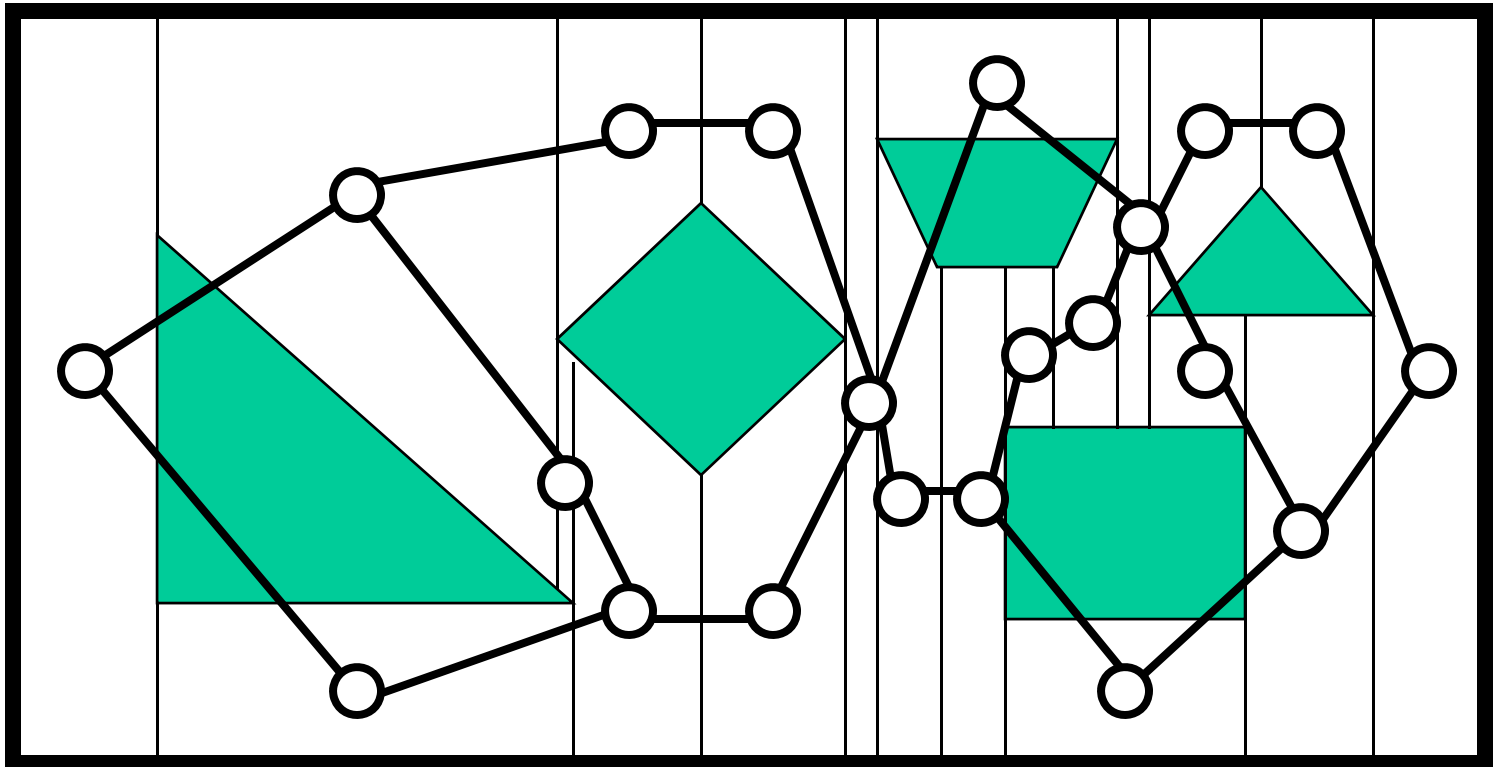
Cell Decompositions: Trapezoidal Decomposition

- Simply draw a vertical line from each vertex until you hit an obstacle. This reduces the world to a union of trapezoid-shaped cells



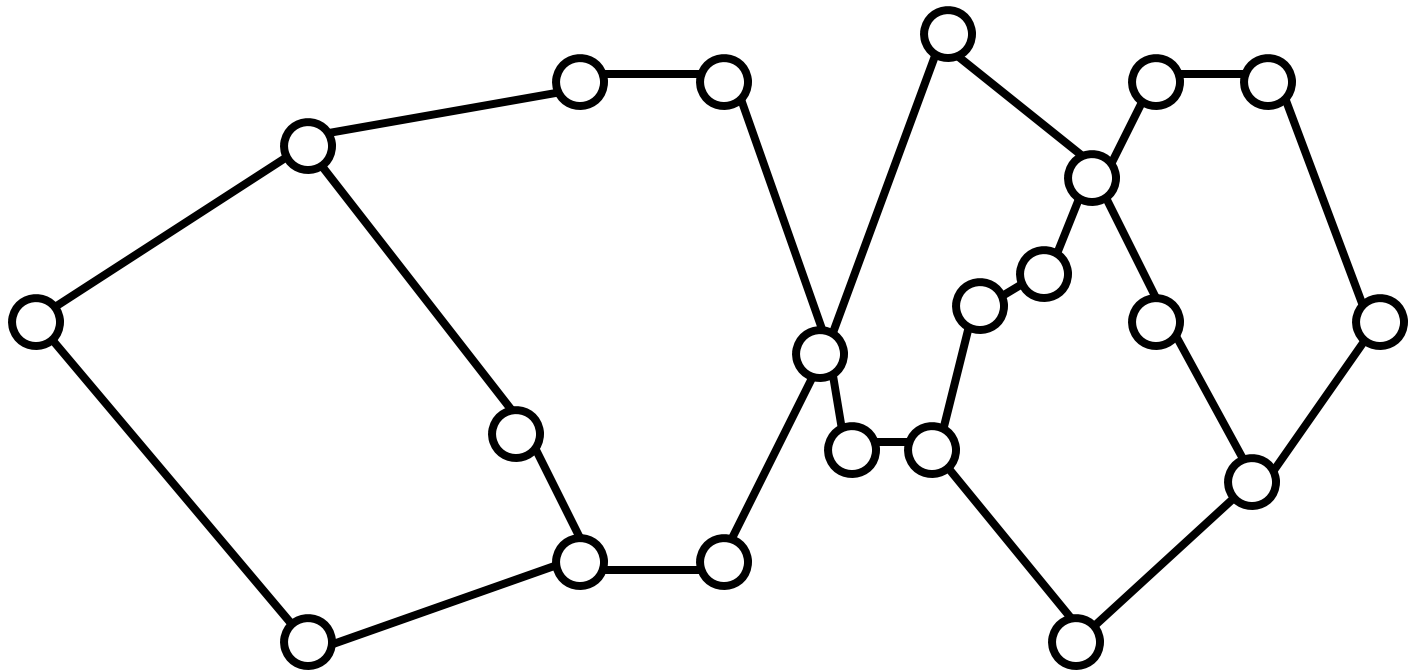
Applications: Coverage

- By reducing the world to cells, we've essentially abstracted the world to a graph.



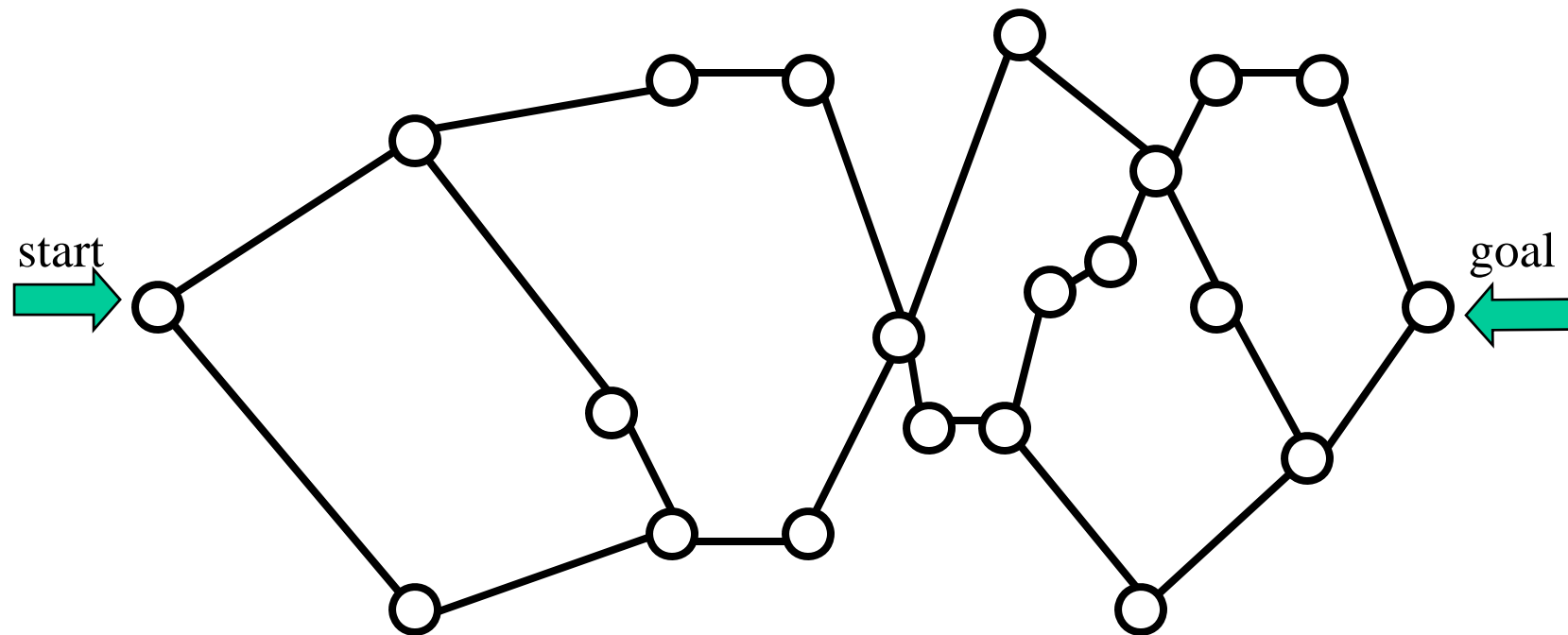
Find a path

- By reducing the world to cells, we've essentially abstracted the world to a graph.



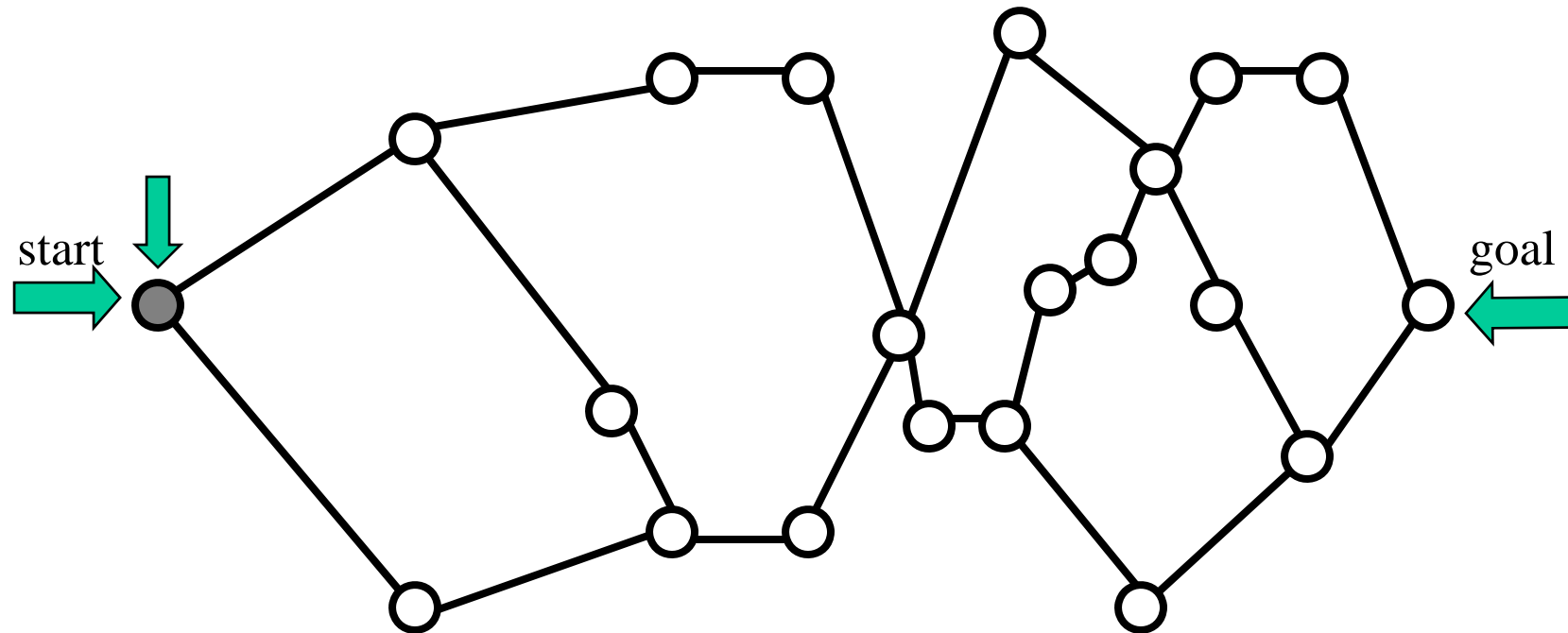
Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



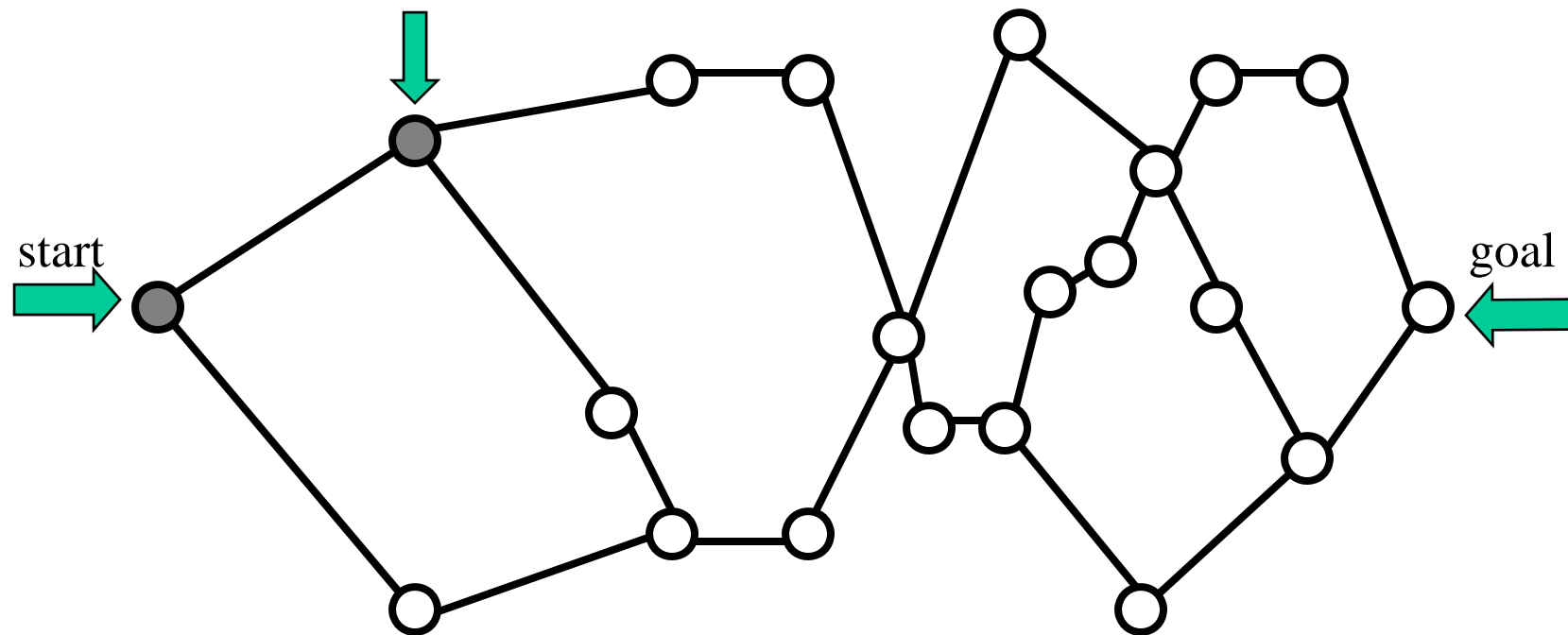
Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



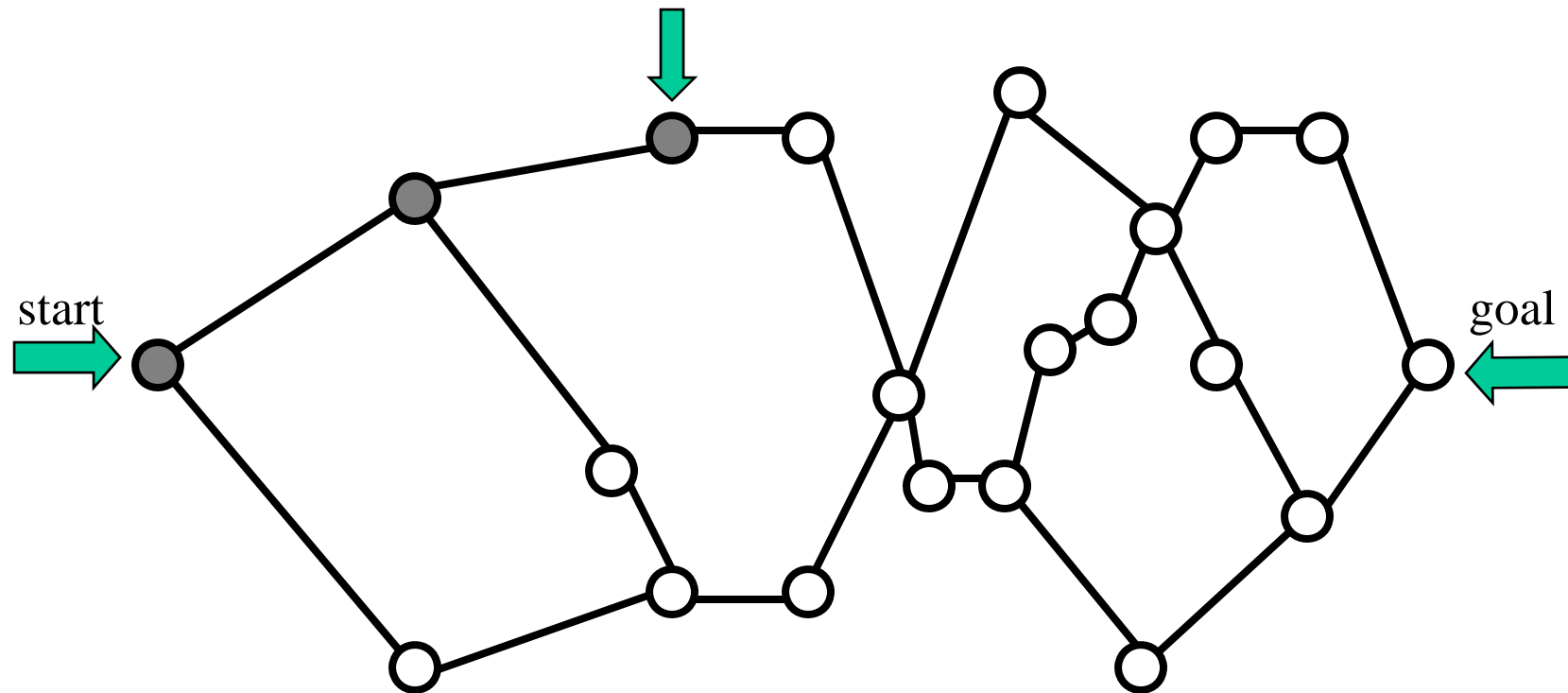
Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



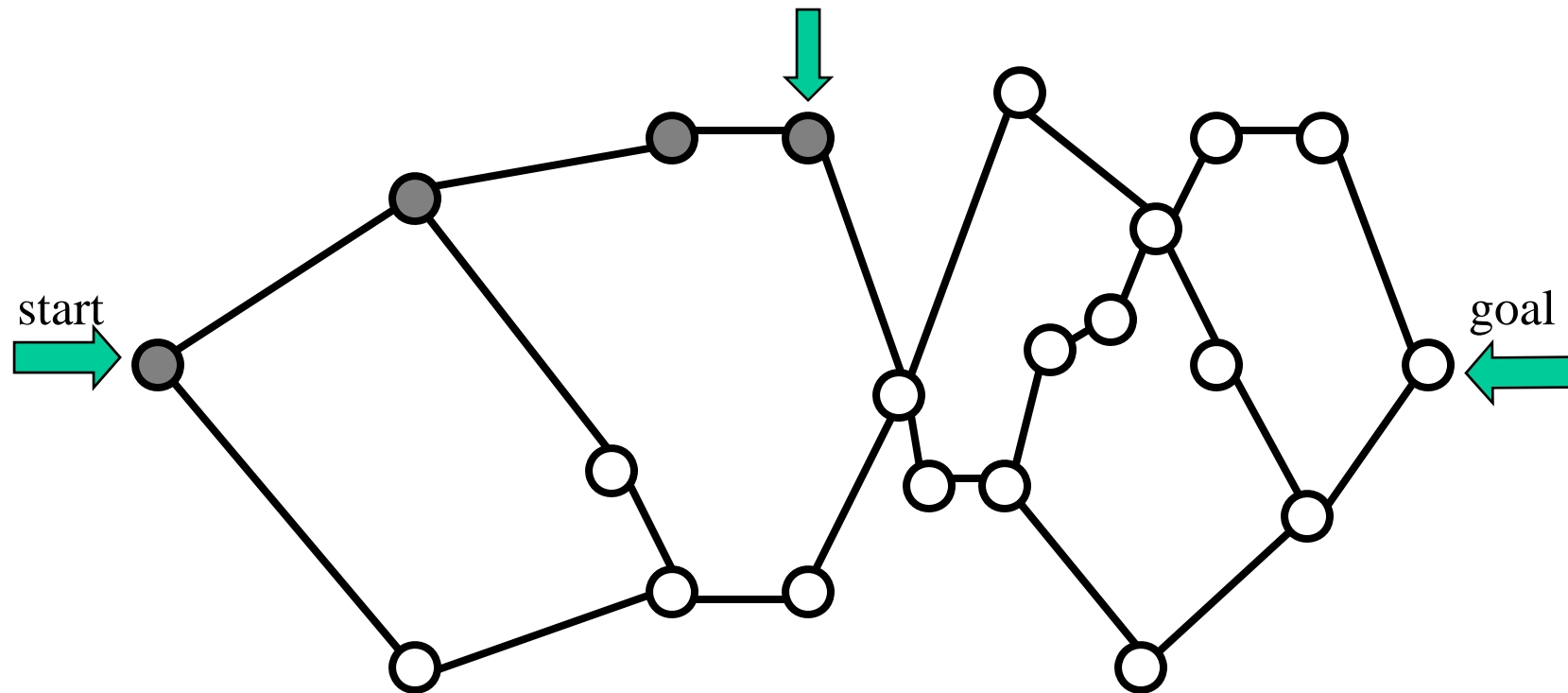
Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



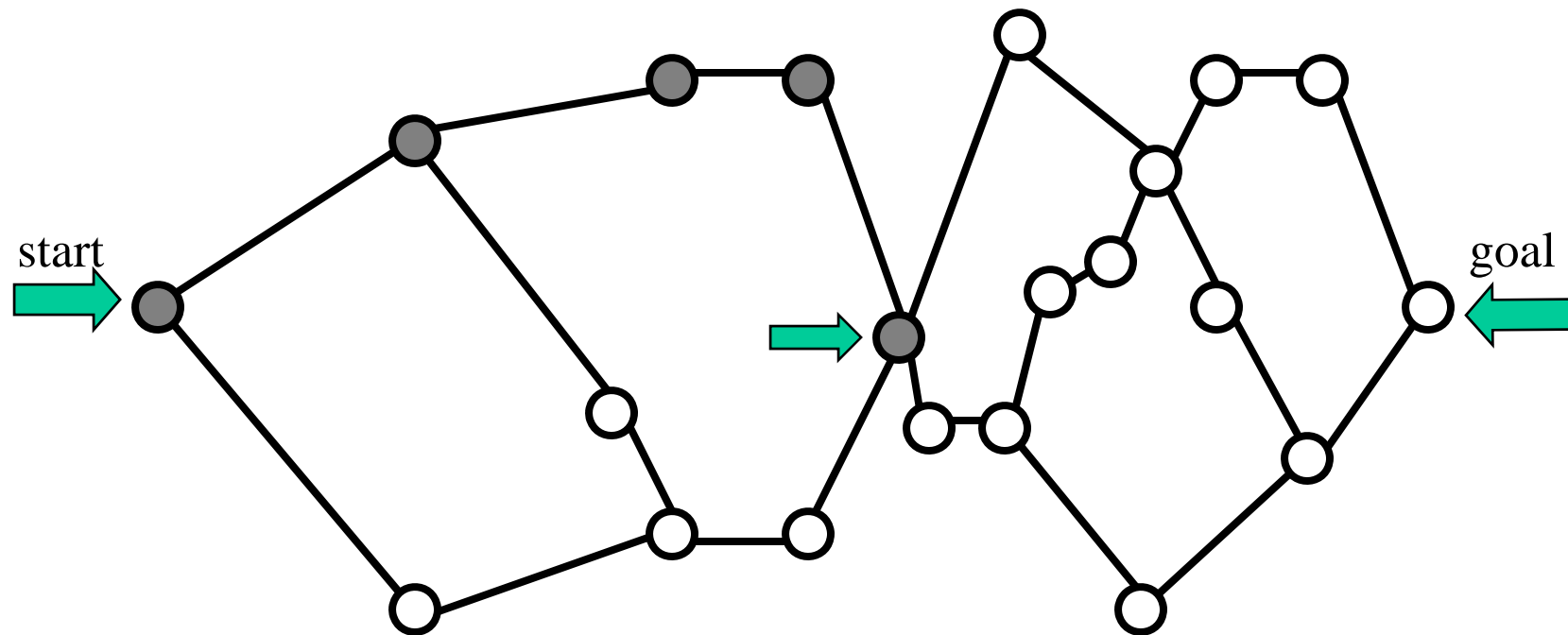
Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



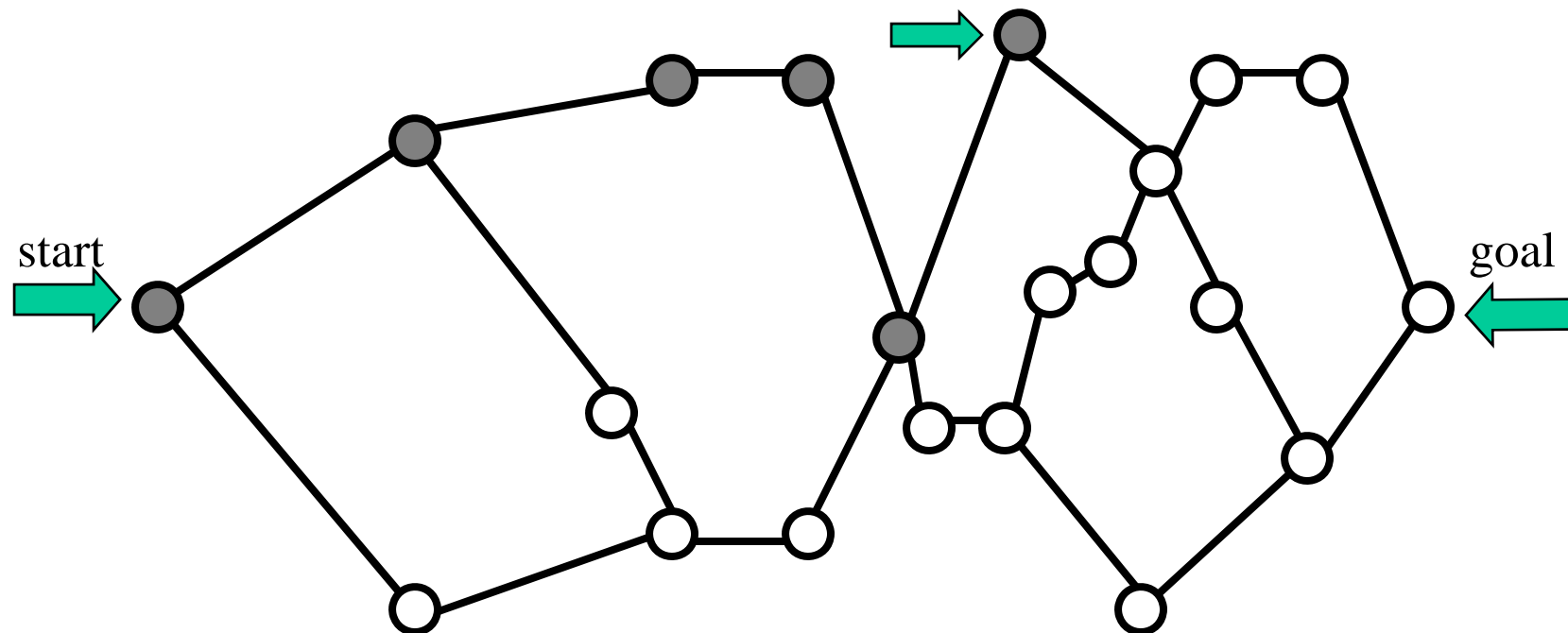
Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



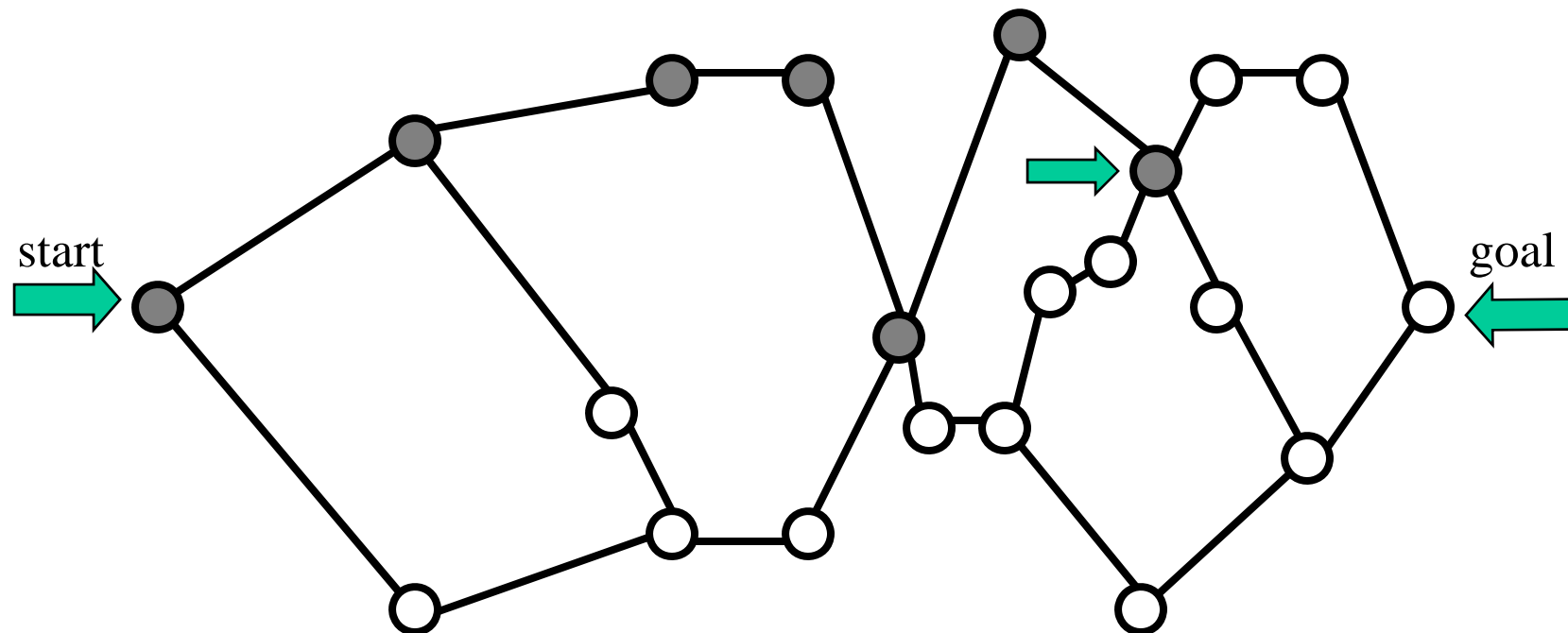
Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



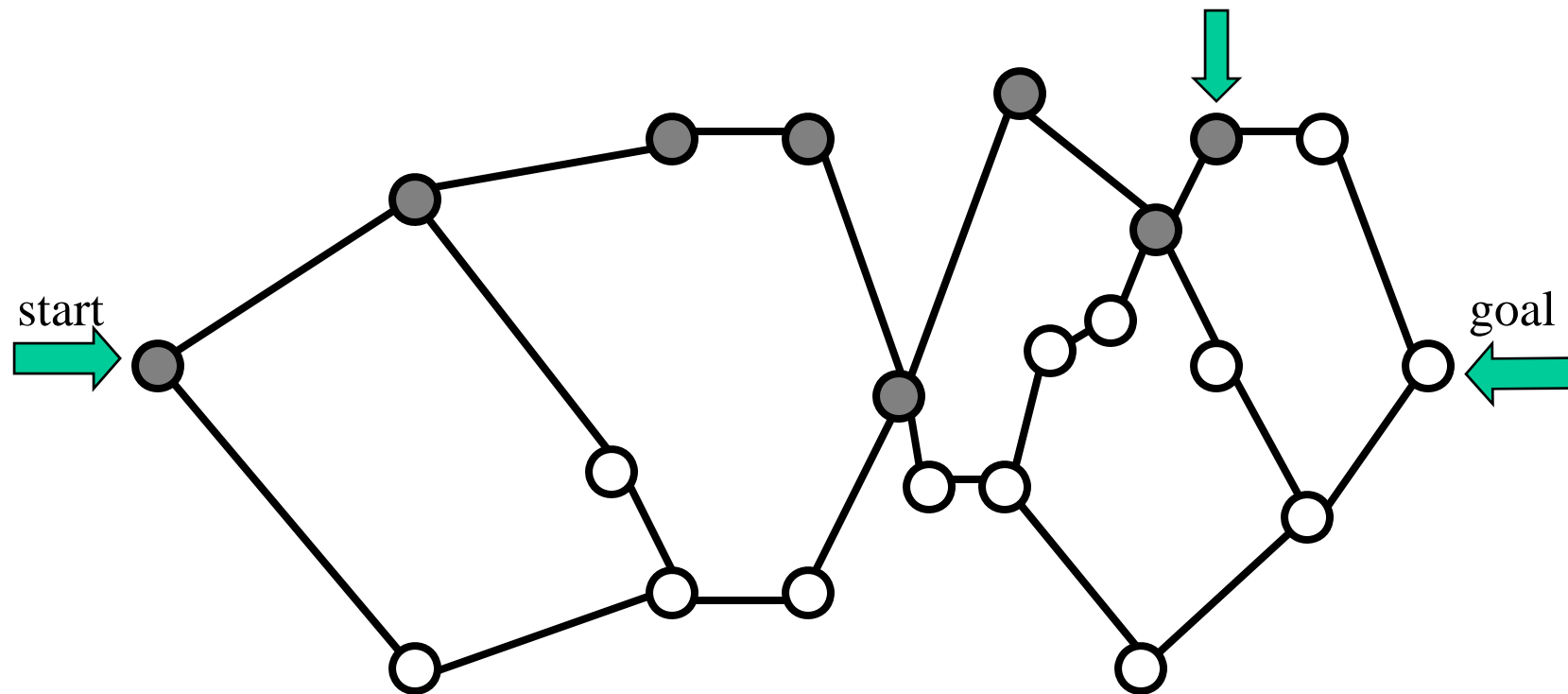
Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



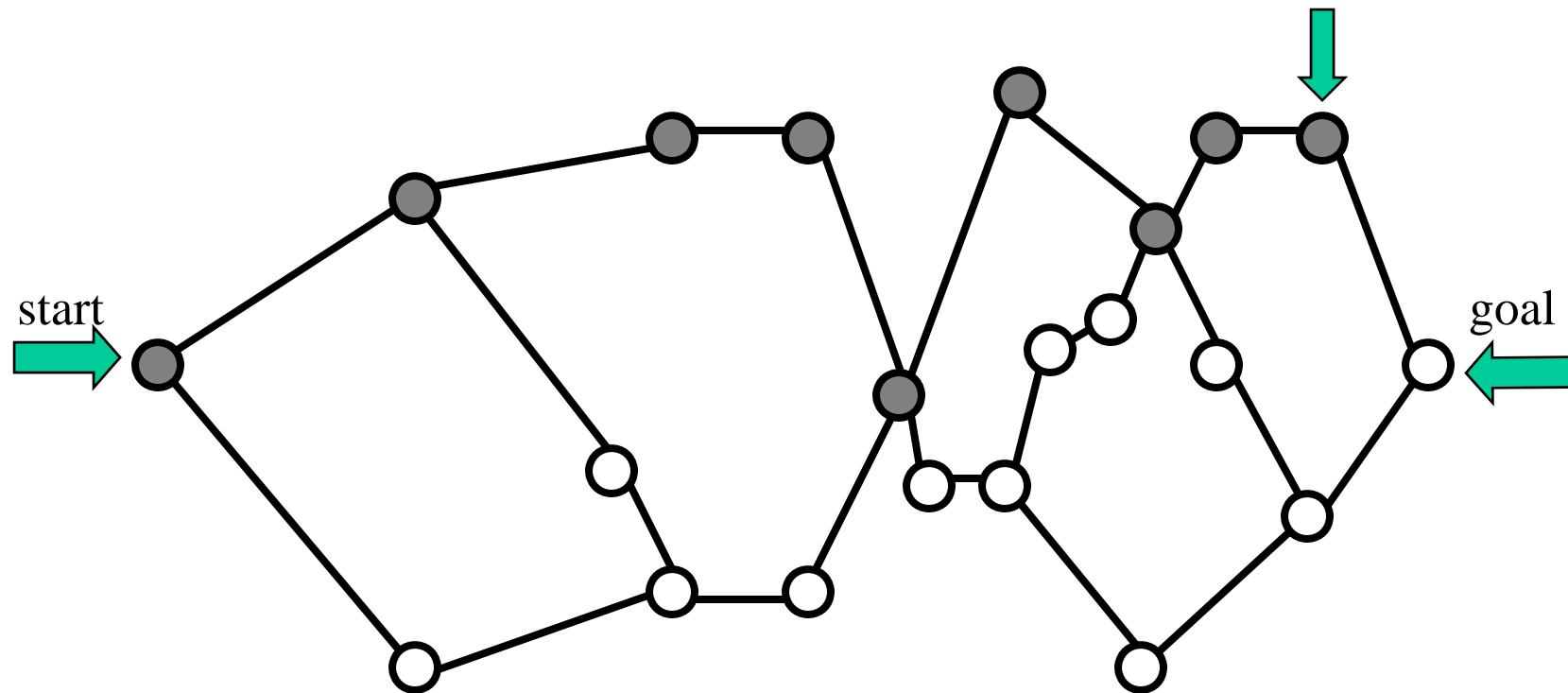
Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



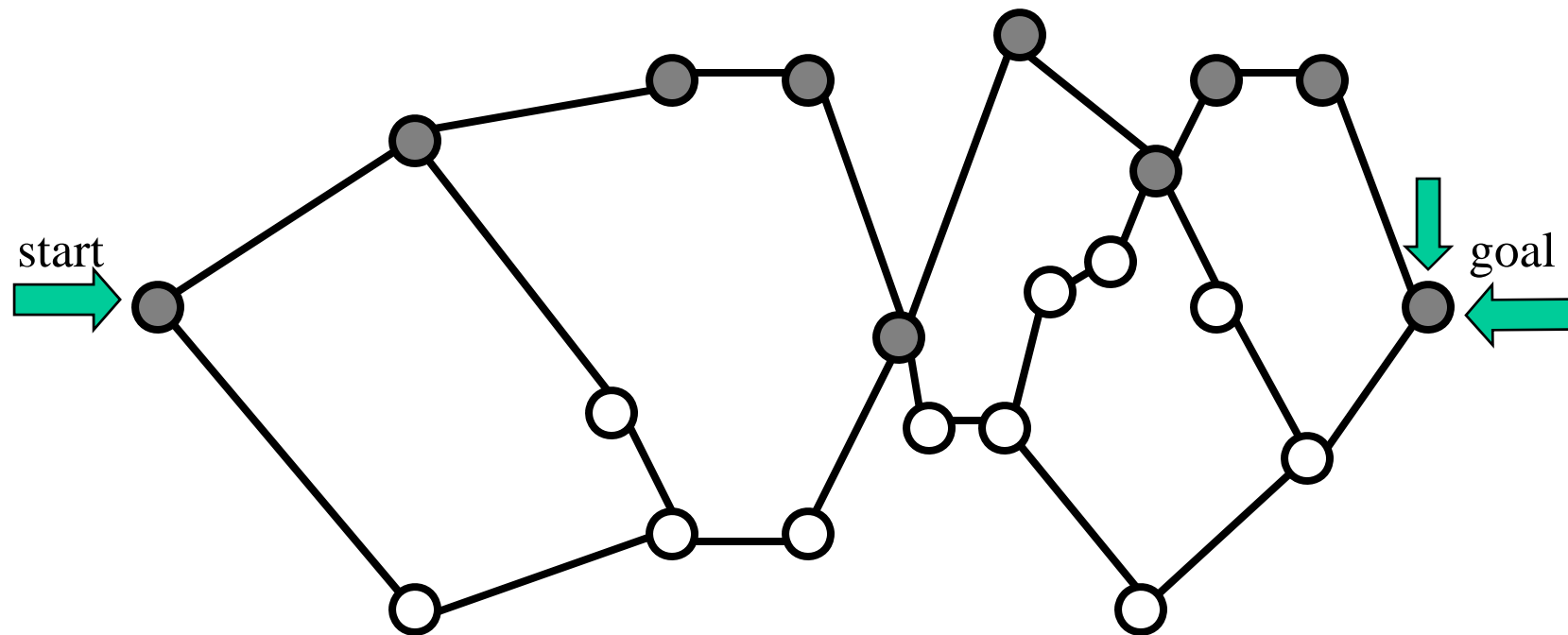
Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



Find a path

- With an adjacency graph, a path from start to goal can be found by simple traversal



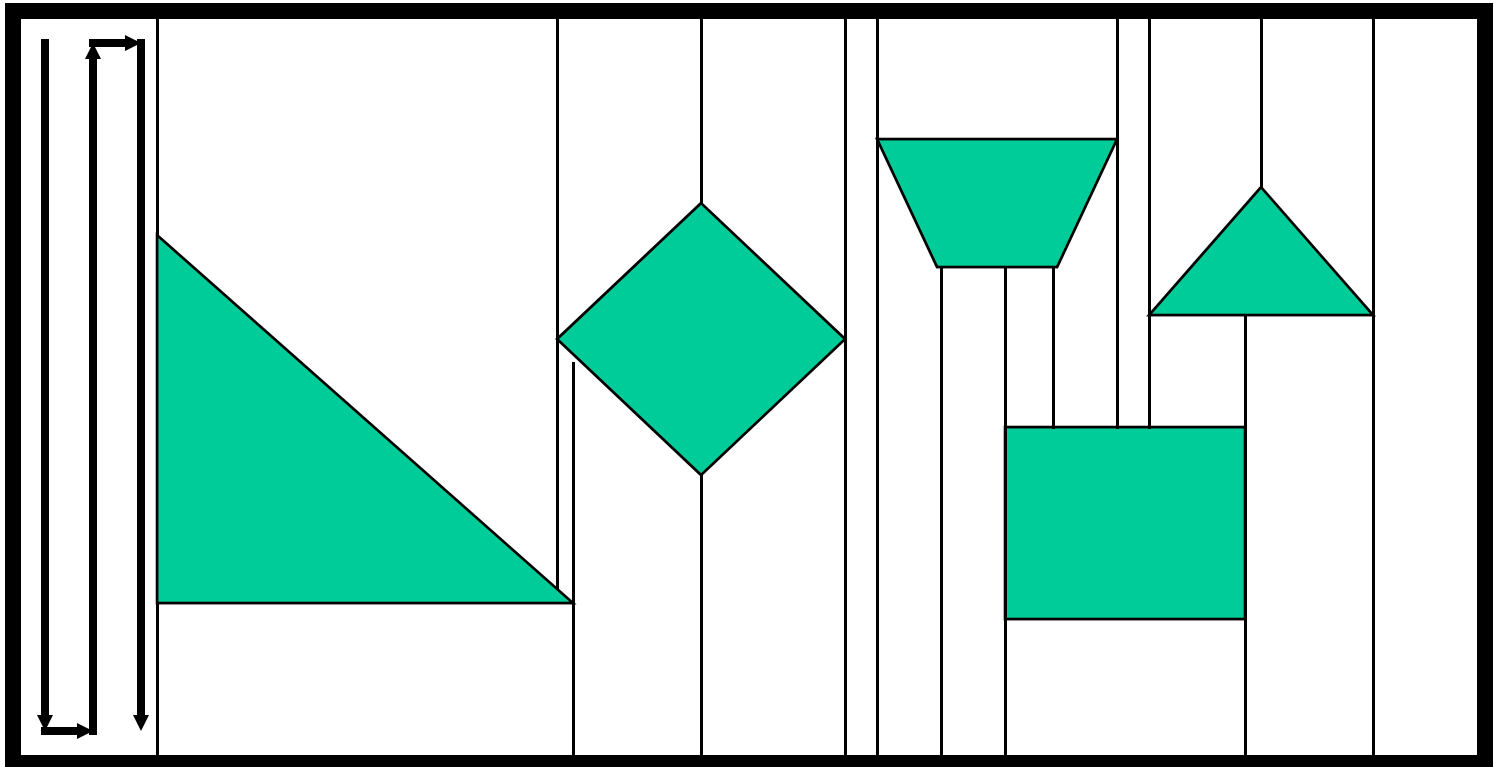
Connect Midpoints of Traps

Applications: Coverage

- First, a distinction between sensor and detector must be made
- *Sensor*: Senses obstacles
- *Detector*: What actually does the coverage
- We'll be observing the simple case of having an omniscient sensor and having the detector's footprint equal to the robot's footprint

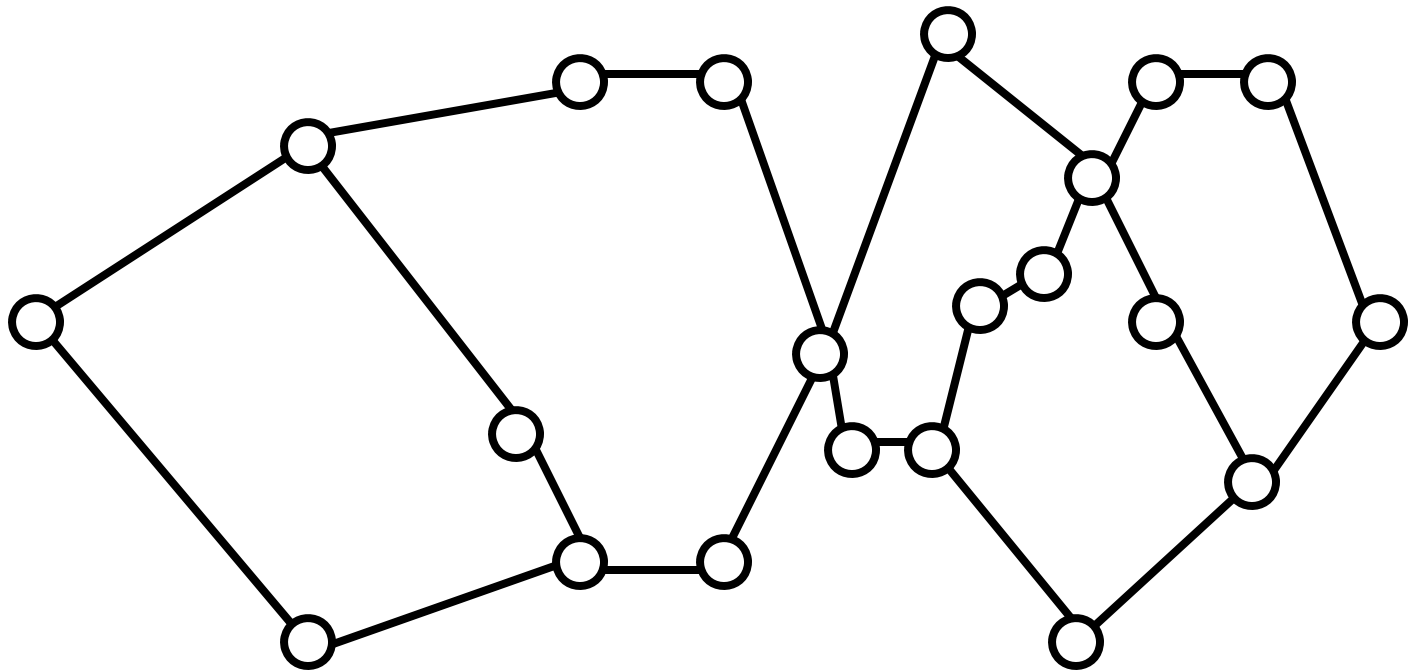
Cell Decompositions: Trapezoidal Decomposition

- How is this useful? Well, trapezoids can easily be covered with simple back-and-forth sweeping motions. If we cover all the trapezoids, we can effectively cover the entire “reachable” world.

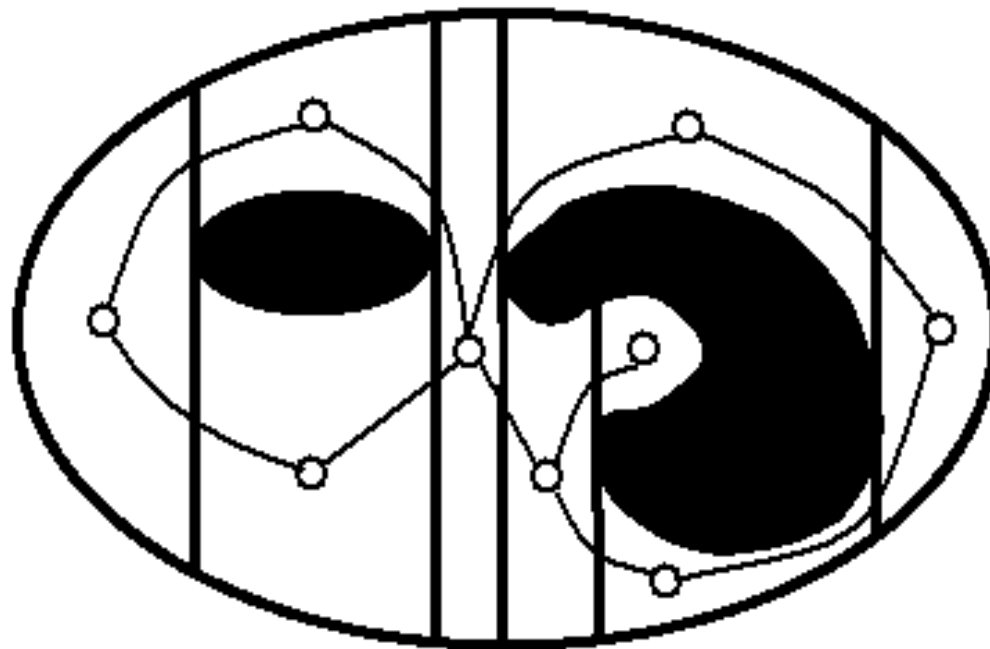


Applications: Coverage

- Simply visit all the nodes, performing a sweeping motion in each, and you're done.



Boustrophedon Decomposition



Conclusion: Complete Overview

- ☑ • The Basics
 - Motion Planning Statement
 - The World and Robot
 - Configuration Space
 - Metrics
- ☑ • Path Planning Algorithms
 - Start-Goal Methods
 - Lumelsky Bug Algorithms
 - Potential Charge Functions
 - The Wavefront Planner
 - Map-Based Approaches
 - Generalized Voronoi Graphs
 - Visibility Graphs
 - Cellular Decompositions => Coverage
- ☑ • *Done with Motion Planning!*