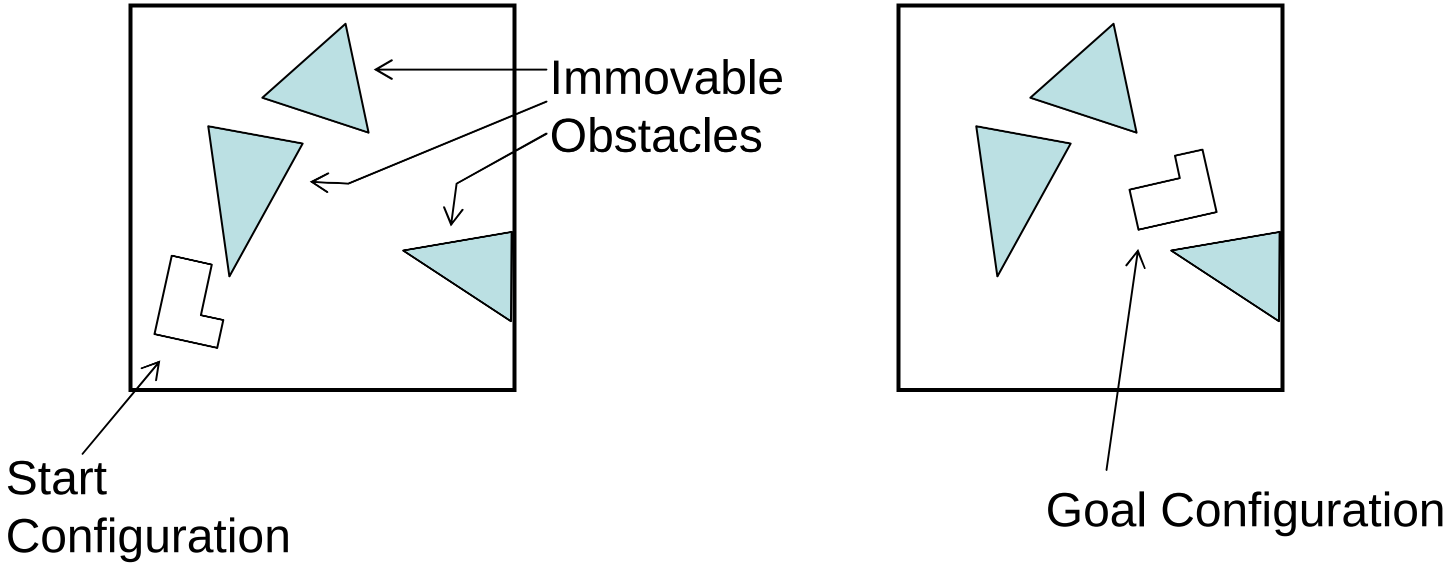
Robot Motion Planning

CSIS 4463

Spatial Reasoning



Can't we use our previous methods?

Discrete Search? – Not a discrete problem

CSP? – Not a natural CSP formulation

Probabilistic? – Nope.

Robots

For our purposes, a robot is:

A set of moving rigid objects called <u>LINKS</u> which are connected by <u>JOINTS</u>.

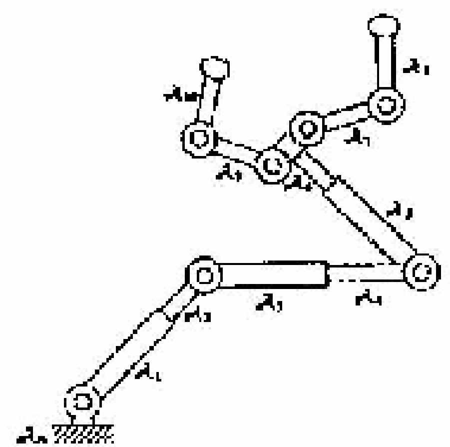


Fig. 11. Structure of the 10-DOF manipulator.

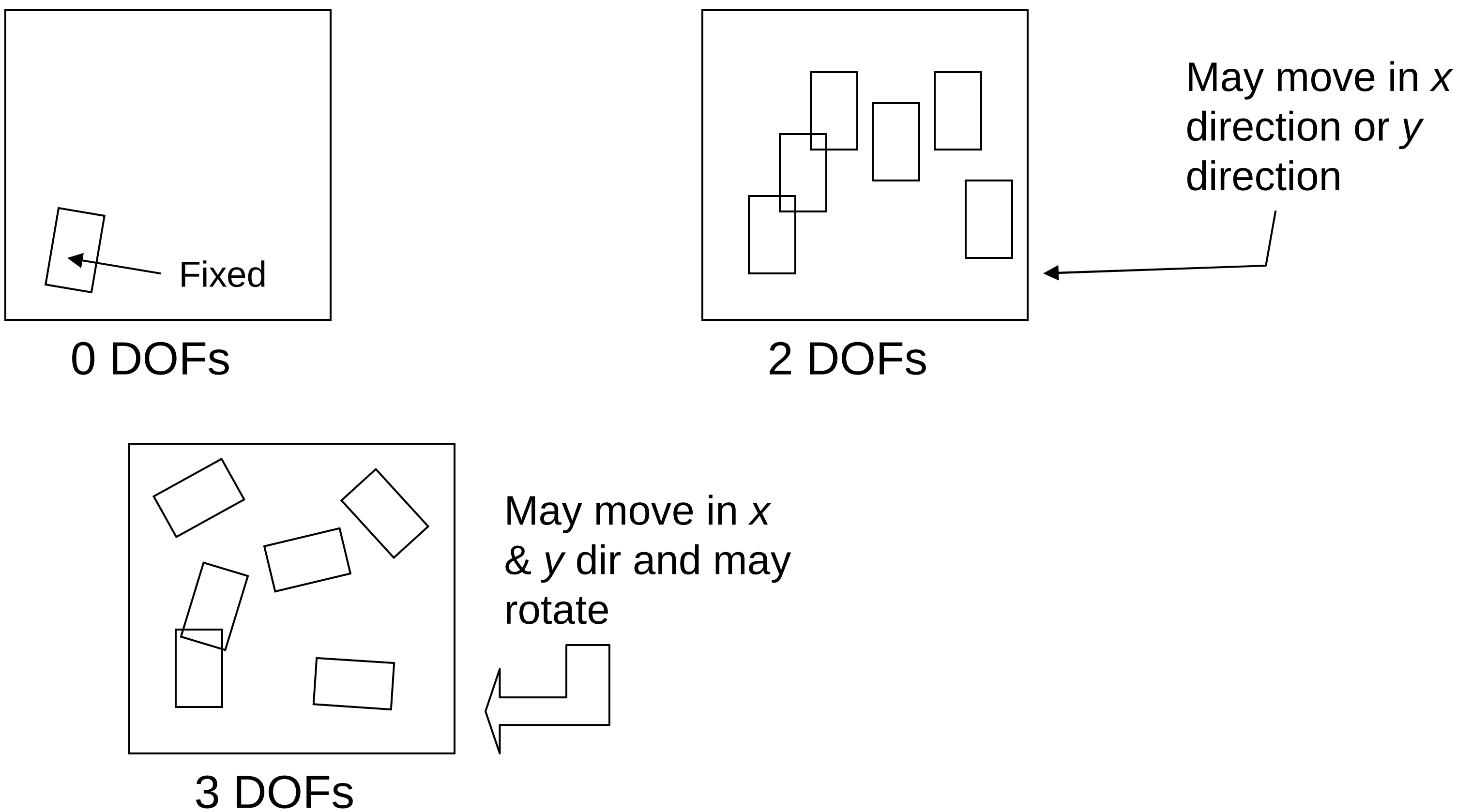
Typically, joints are REVOLUTE or PRISMATIC.

Such joints each give one DEGREE OF FREEDOM.

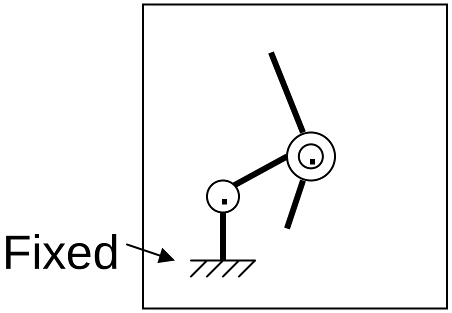
Given *p* DOFs, the configuration of the robot can be represented by *p* values $\boldsymbol{q} = (q_1\ q_2 \cdots q_p)$ where q_i is the angle or length of the *i*'th joint

Free-Flying Polygons

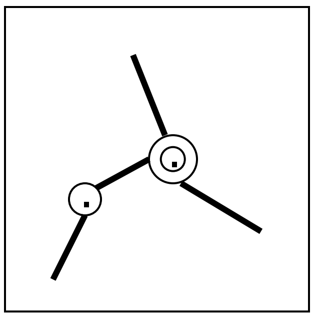
If part of the robot is fixed in the world, the joints are all the DOFs you're getting. But if the robot can be free-flying we get more DOFs.



Examples

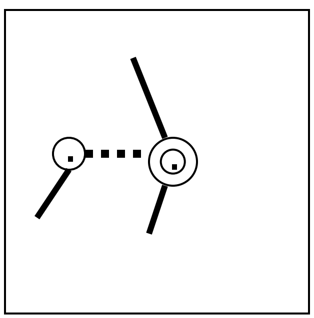


How many DOFs?



Free flying

How many DOFs?



Midline ■■■ must always be horizontal.

How many DOFs?

The configuration $\underset{\sim}{q}$ has one real valued entry per DOF.

Robot Motion Planning

An important, interesting, spatial reasoning problem.

- Let *A* be a robot with *p* degrees of freedom, living in a 2-D or 3-D world.
- Let *B* be a set of obstacles in this 2-D or 3-D world.
- Call a configuration LEGAL if it neither intersects any obstacles nor self-intersects.
- Given an initial configuration $\underset{\sim}{q}_{start}$ and a goal config $\underset{\sim}{q}_{goal}$, generate a continuous path of legal configurations between them, or report failure if no such path exists.

Configuration Space

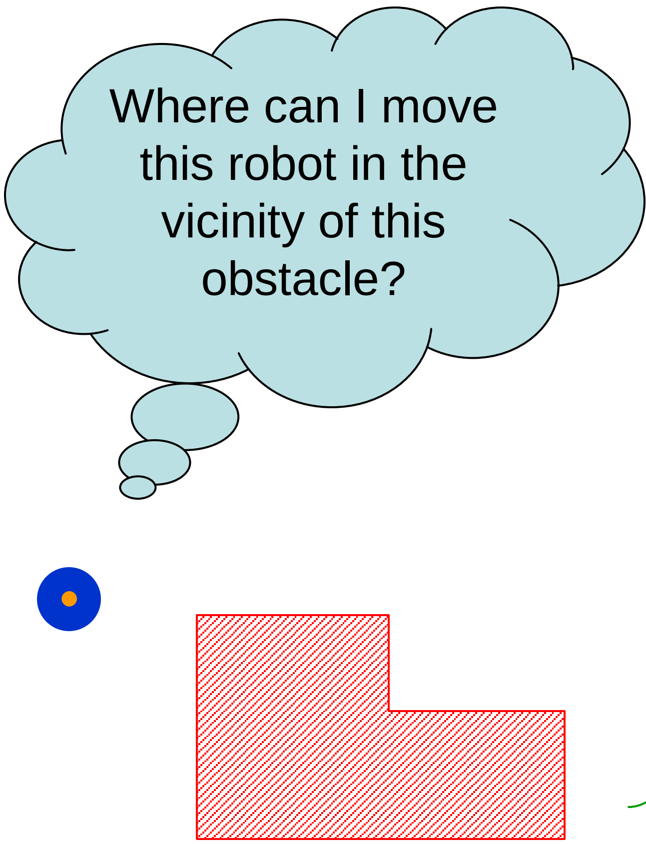
Is the set of legal configurations of the robot. It also defines the topology of continuous motions

For rigid-object robots (no joints) there exists a transformation to the robot and obstacles that turns the robot into a single point. The

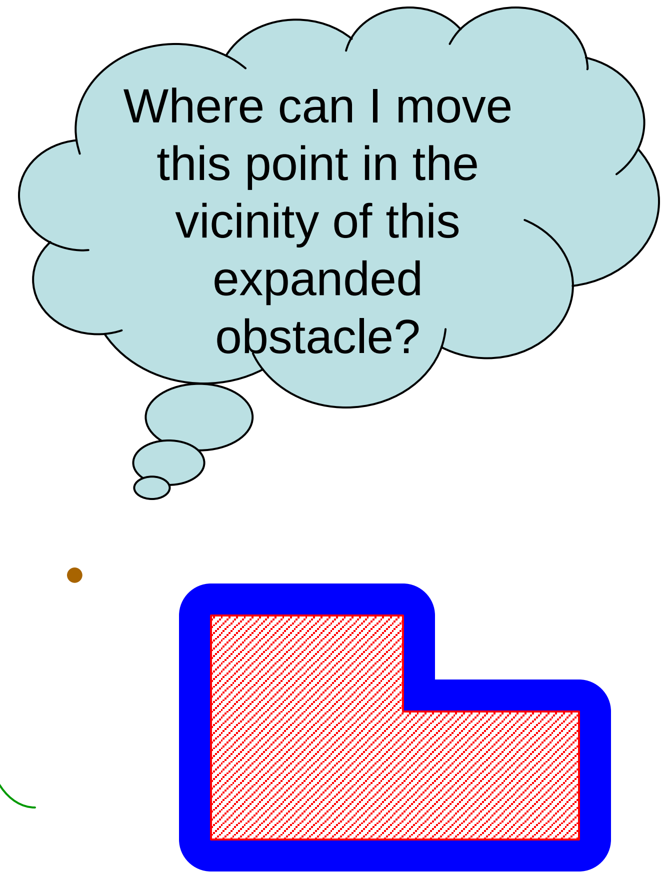
C-Space Transform

Configuration Space Transform Examples

2-D World
2 DOFs



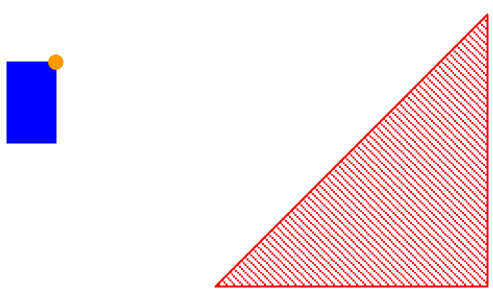
…is equivalent to…



Configuration Space Transform Examples

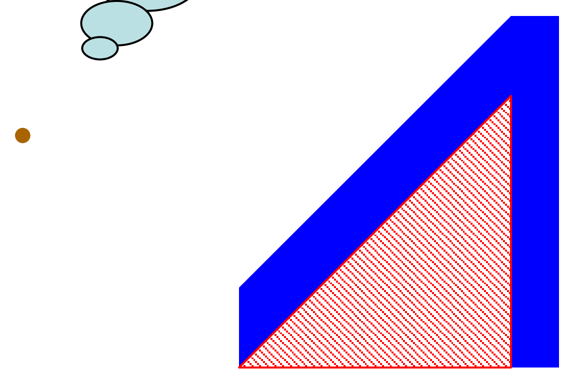
2-D World
2 DOFs

Where can I move this robot in the vicinity of this obstacle?



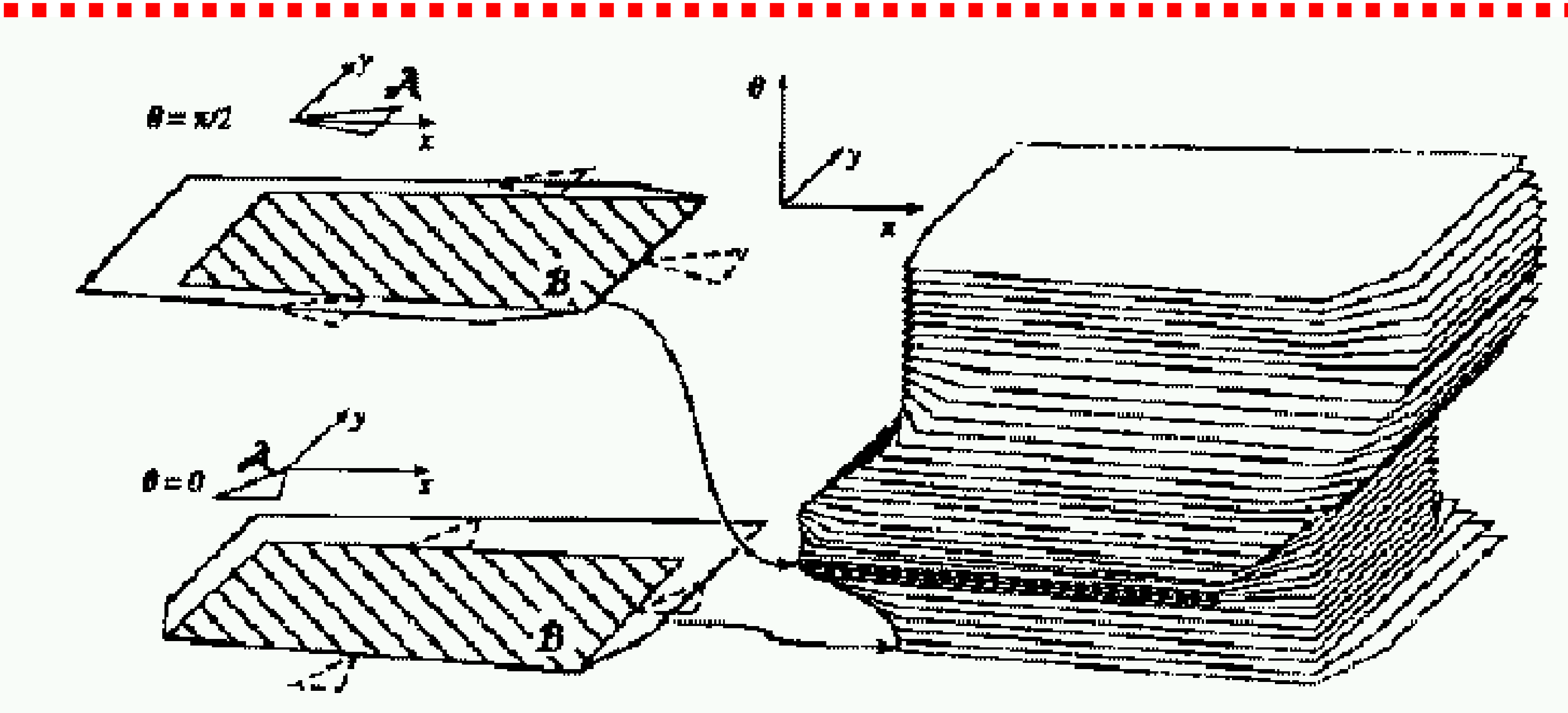
…is equivalent to…

Where can I move this point in the vicinity of this expanded obstacle?



Assuming you're not allowed to rotate

Configuration Space Transform Examples



We’ve turned the problem from “Twist and turn this 2-D polygon past this other 2-D polygon” into “Find a path for this point in 3-D space past this weird 3-D obstacle”.

Why’s this transform useful?

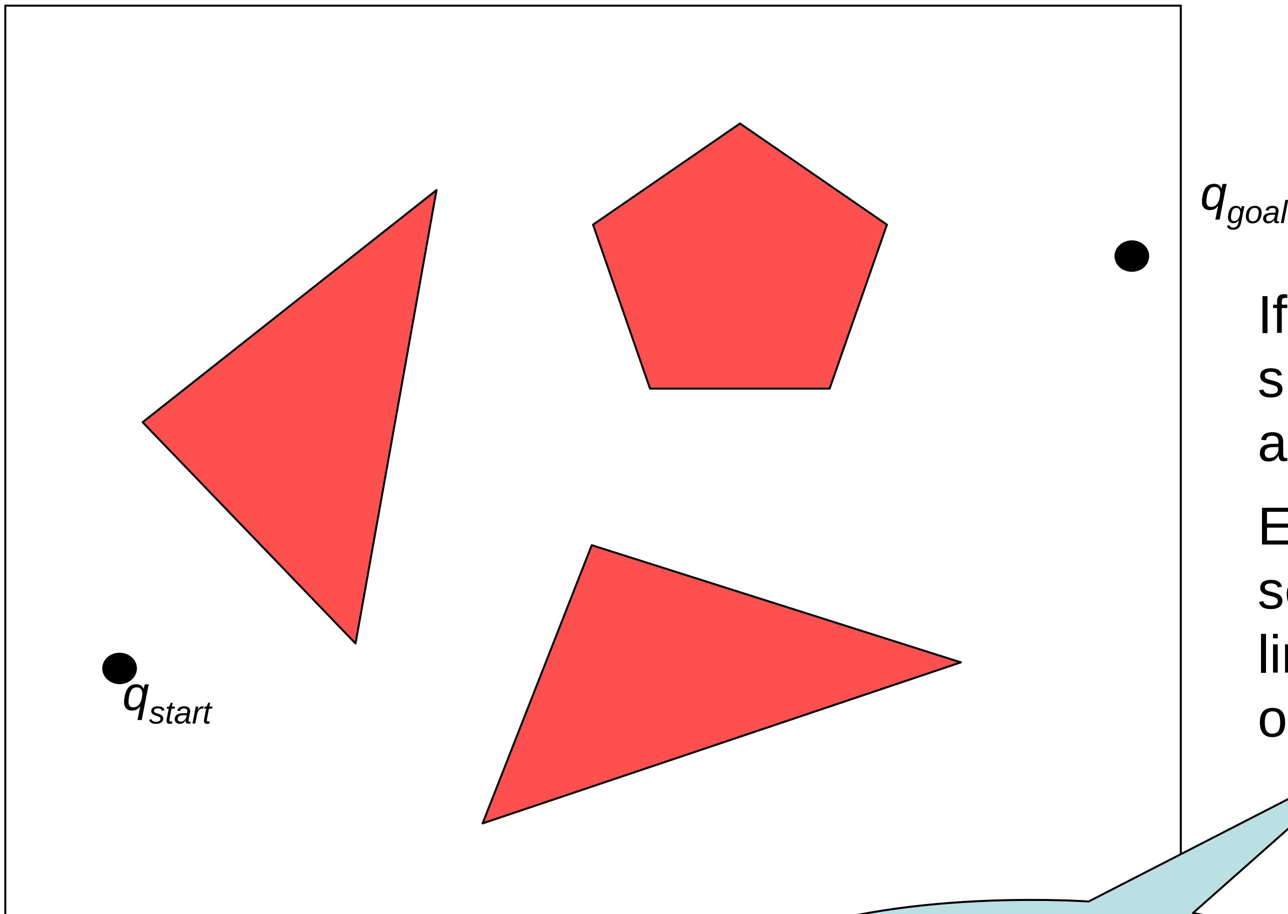
Because we can plan paths for points instead of polyhedra/polygons

Robot Motion Planning Research

…Has produced four kinds of algorithms. The first is the Visibility Graph.

Visibility Graph

Suppose someone gives you a CSPACE with polygonal obstacles

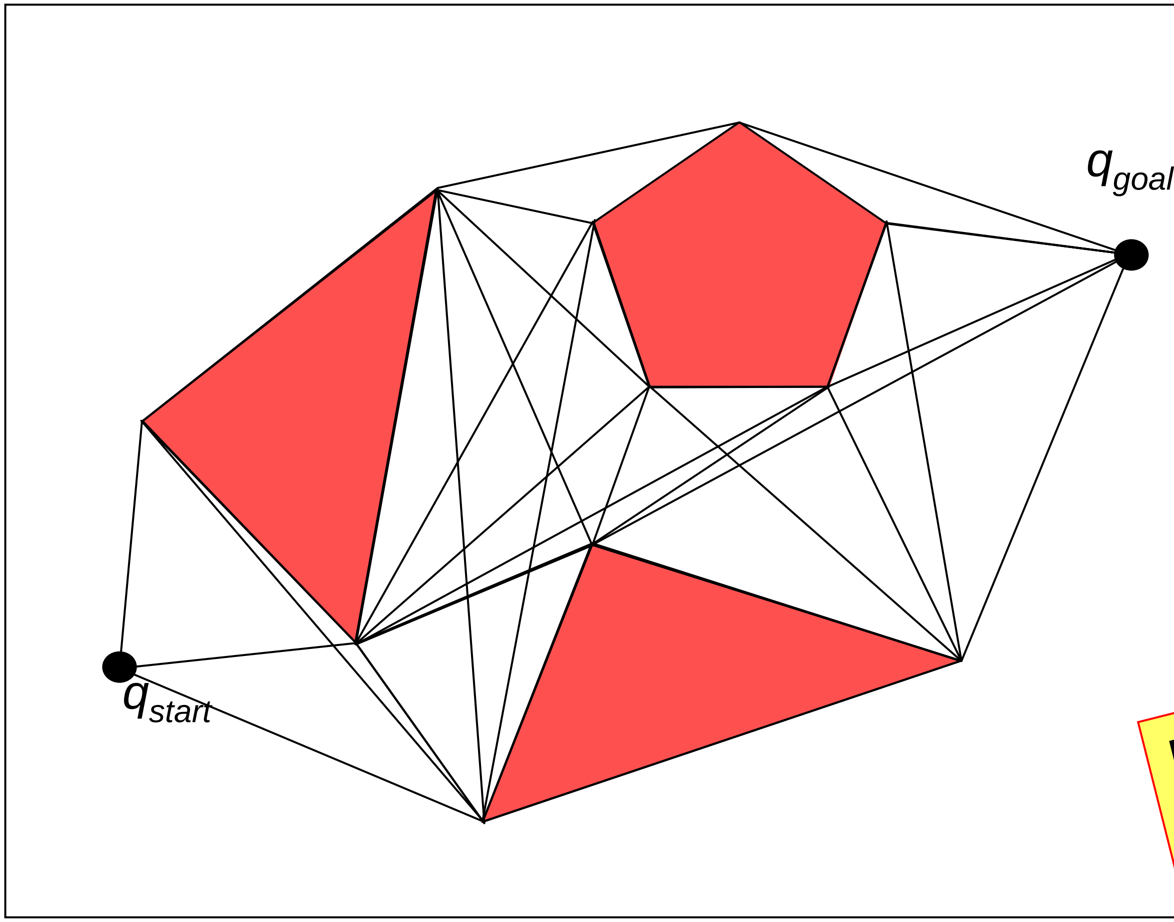


If there were no blocks, shortest path would be a straight line.

Else it must be a sequence of straight lines "shaving" corners of obstacles.

Obvious, but very awkward to prove

Visibility Graph Algorithm



1. Find all non-blocked lines between polygon vertices, start and goal.
2. Search the graph of these lines for the shortest path. (Guess best search algorithm?)

If there are n vertices, the easy algorithm is $O(n^3)$. Slightly tougher $O(n^2 \log n)$. $O(n^2)$ in theory.

Visibility Graph Method

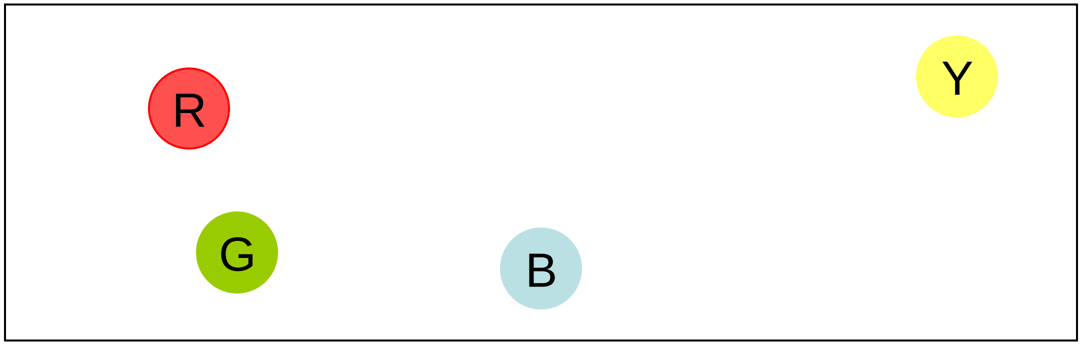
COMPLAINT

- Visibility graph method finds the shortest path.
- But it does so by skirting along and close to obstacles.
- Any error in control, or model of obstacle locations, and Bang! Screech!!

Who cares about optimality?

Perhaps we want to get a non-stupid path that steers as far from the obstacles as it can.

Voronoi Diagrams



Someone gives you some dots. Each dot is a different color.

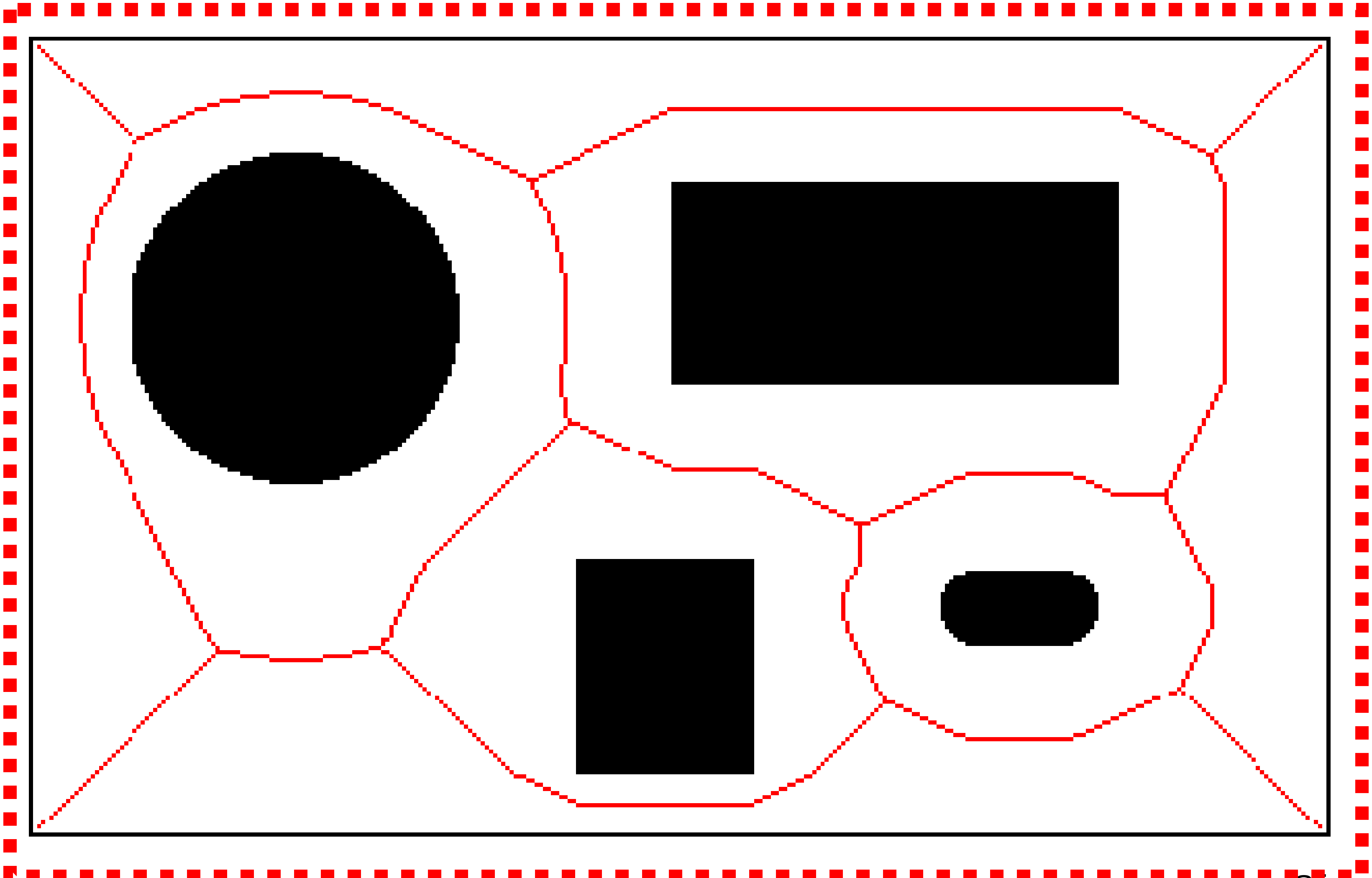
You color in the whole of 2-D space according to this rule:

> "The color of any given point equals the color of the nearest dot."

The borders between your different regions are a VORNOI DIAGRAM.

> For *n* point in 2-D space the exact Voronoi diagram can be computed in time O(*n* log *n*).

Voronoi Diagram from Polygons instead of Points



Voronoi Diagram Methods for C-Space Motion Planning

- Compute the Voronoi Diagram of C-space.
- Compute shortest straightline path from start to any point on Voronoi Diagram.
- Compute shortest straightline path from goal to any point on Voronoi Diagram.
- Compute shortest path from start to goal along Voronoi Diagram.

Voronoi Diagrams

COMPLAINT

- Assumes polygons, and very complex above 2-D.

  Answer: very nifty approximate algorithms (see Howie Choset's work http://voronoi.sbp.ri.cmu.edu/~choset)

- This "use Voronoi to keep clear of obstacles" is just a heuristic. And can be made to look stupid:

Can you see how?

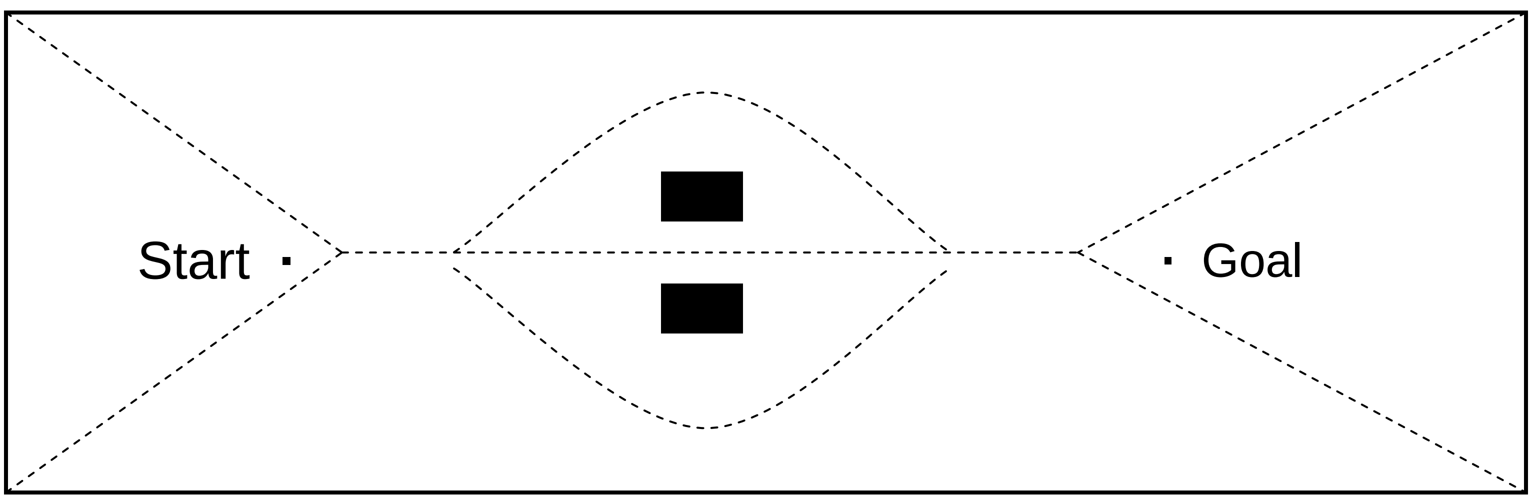
Voronoi Diagrams

COMPLAINT

- Assumes polygons, and very complex above 2-D.

  Answer: very nifty approximate algorithms (see Howie Choset's work http://voronoi.sbp.ri.cmu.edu/~choset)

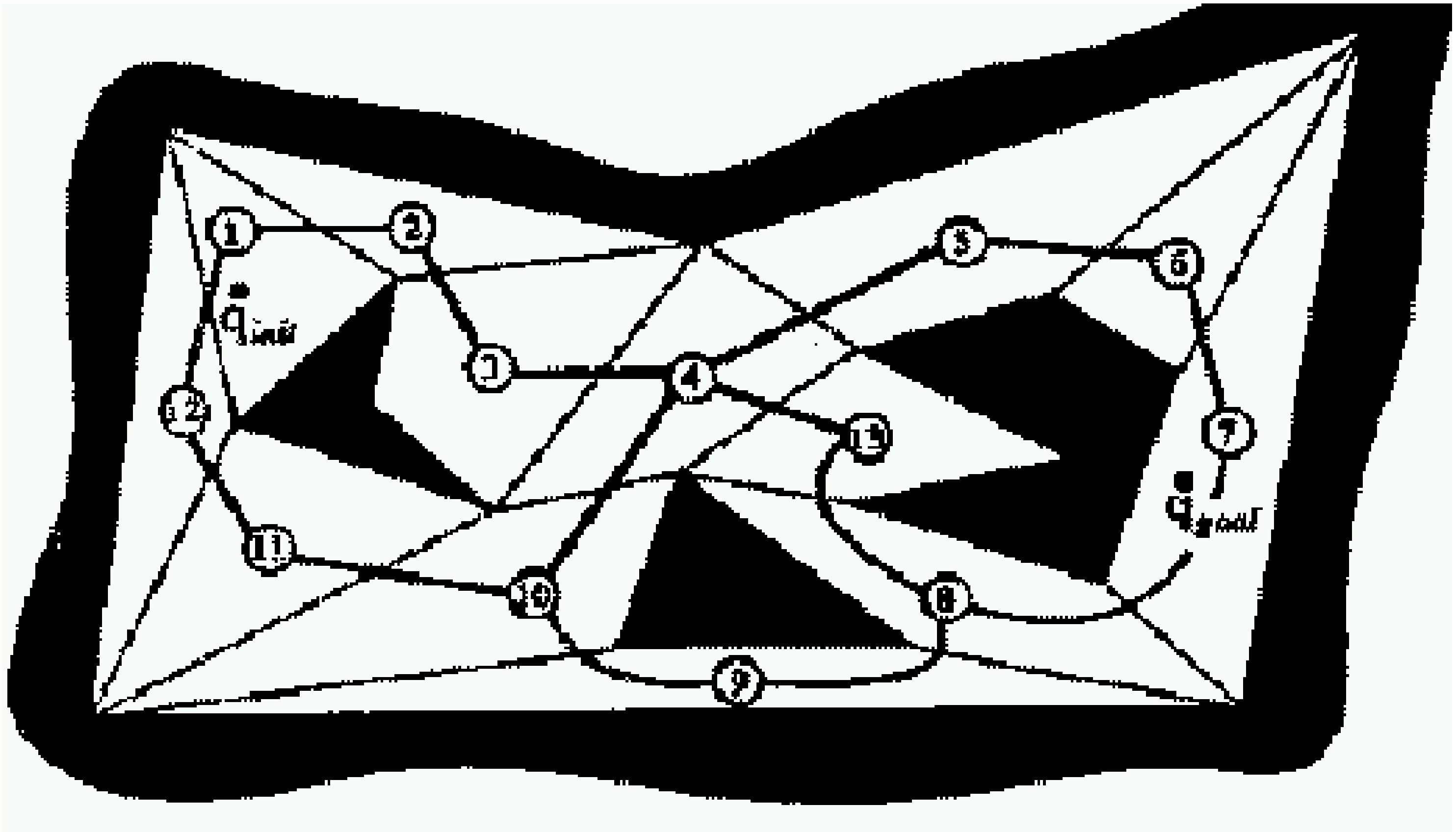
- This "use Voronoi to keep clear of obstacles" is just a heuristic. And can be made to look stupid:



Cell Decomposition Methods

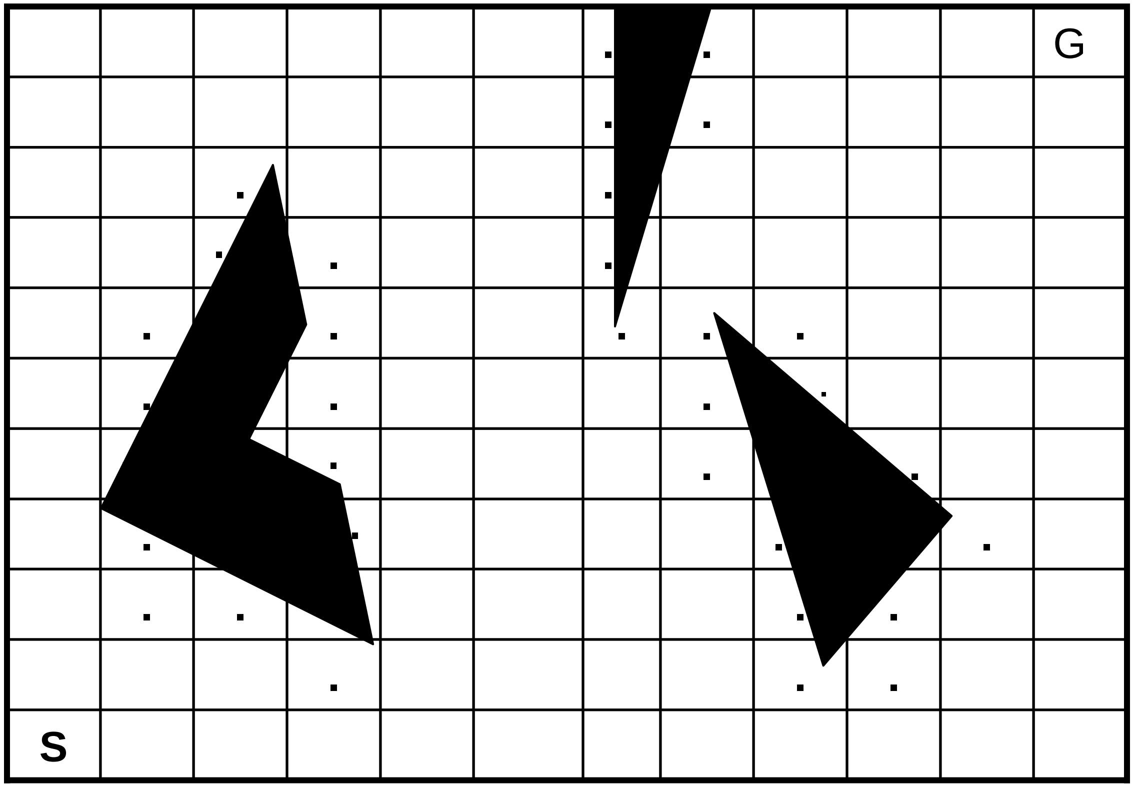
Cell Decomp Method One: Exact Decomp

- Break free space into convex exact polygons.



…But this is also impractical above 2-D or with non-polygons.

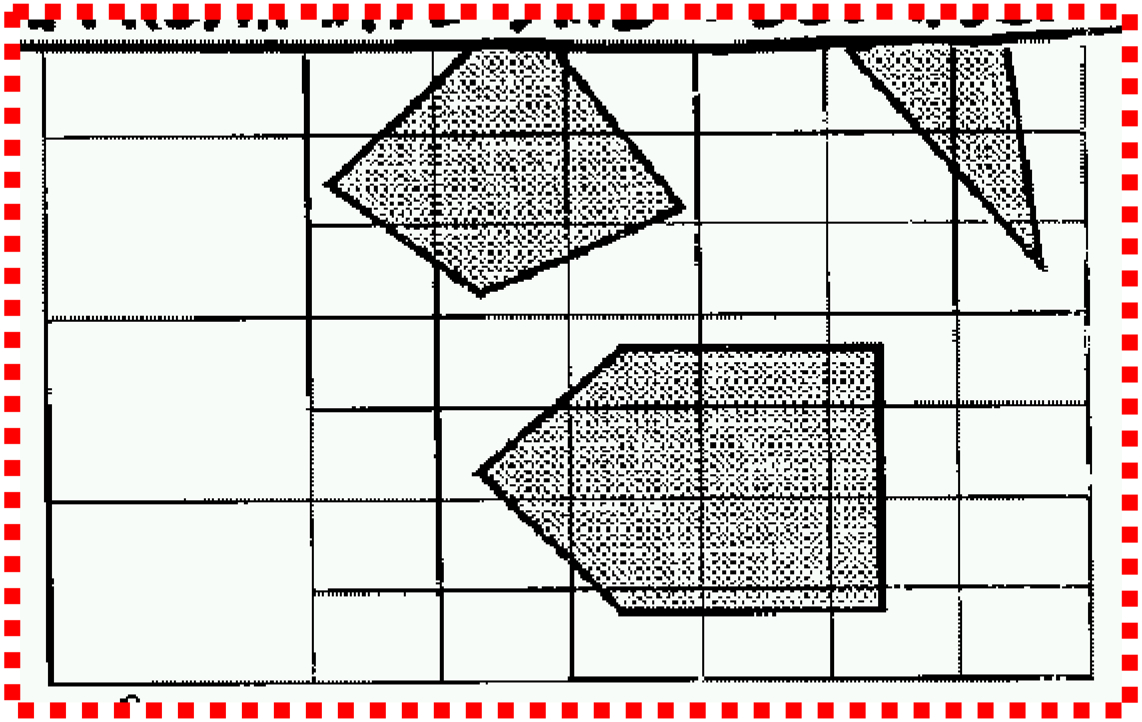
Approximate Cell Decomposition



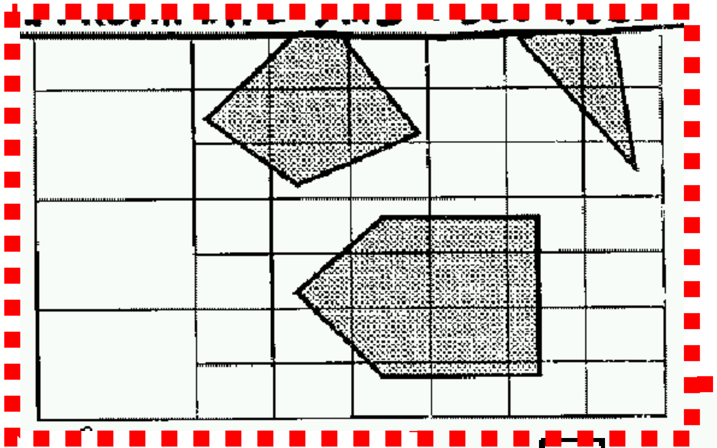
- Lay down a grid
- Avoid any cell which intersects an obstacle
- Plan shortest path through other cells (e.g. with A*)

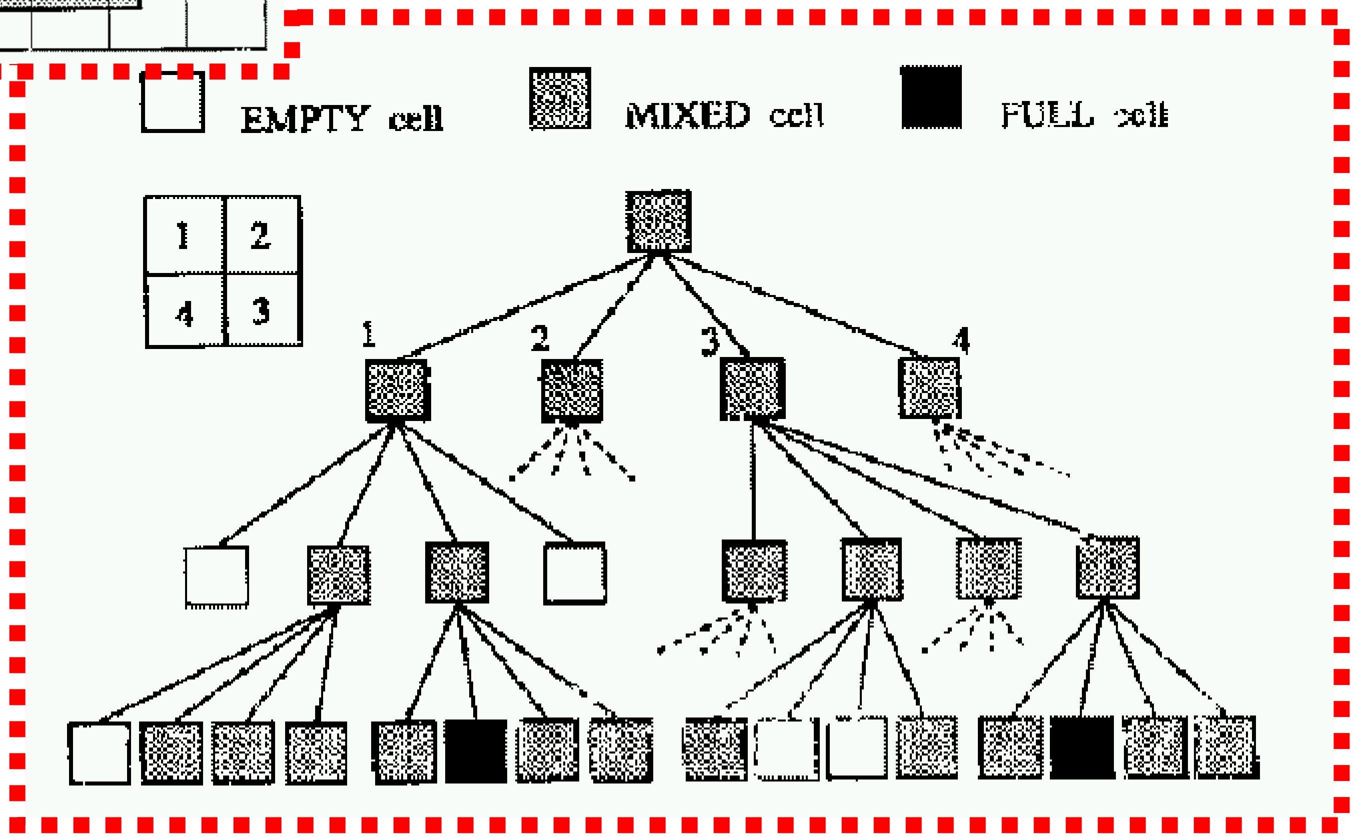
If no path exists, double the resolution and try again. Keep trying!!

Variable Resolution “Approximate and Decompose”



Variable Resolution “Approximate and Decompose”





Approximate Cell Decomposition

COMPLAINTS

- ❑ Not so many complaints. This is actually used in practical systems.

But

- o Not exact (no notion of "best" path)
- o Not complete: doesn't know if problem actually unsolvable
- o Still hopeless above a small number of dimensions?

Potential Methods

Define a function $u\left(\underset{\sim}{q}\right)$

$u:\text{Configurations} \rightarrow \Re$

Such that

$u \rightarrow$ huge as you move towards an obstacle

$u \rightarrow$ small as you move towards the goal

Write $d_g\left(\underset{\sim}{q}\right) =$ distance from $\underset{\sim}{q}$ to $\underset{\sim}{q}$ goal

$d_i\left(\underset{\sim}{q}\right) =$ distance from $\underset{\sim}{q}$ to nearest obstacle

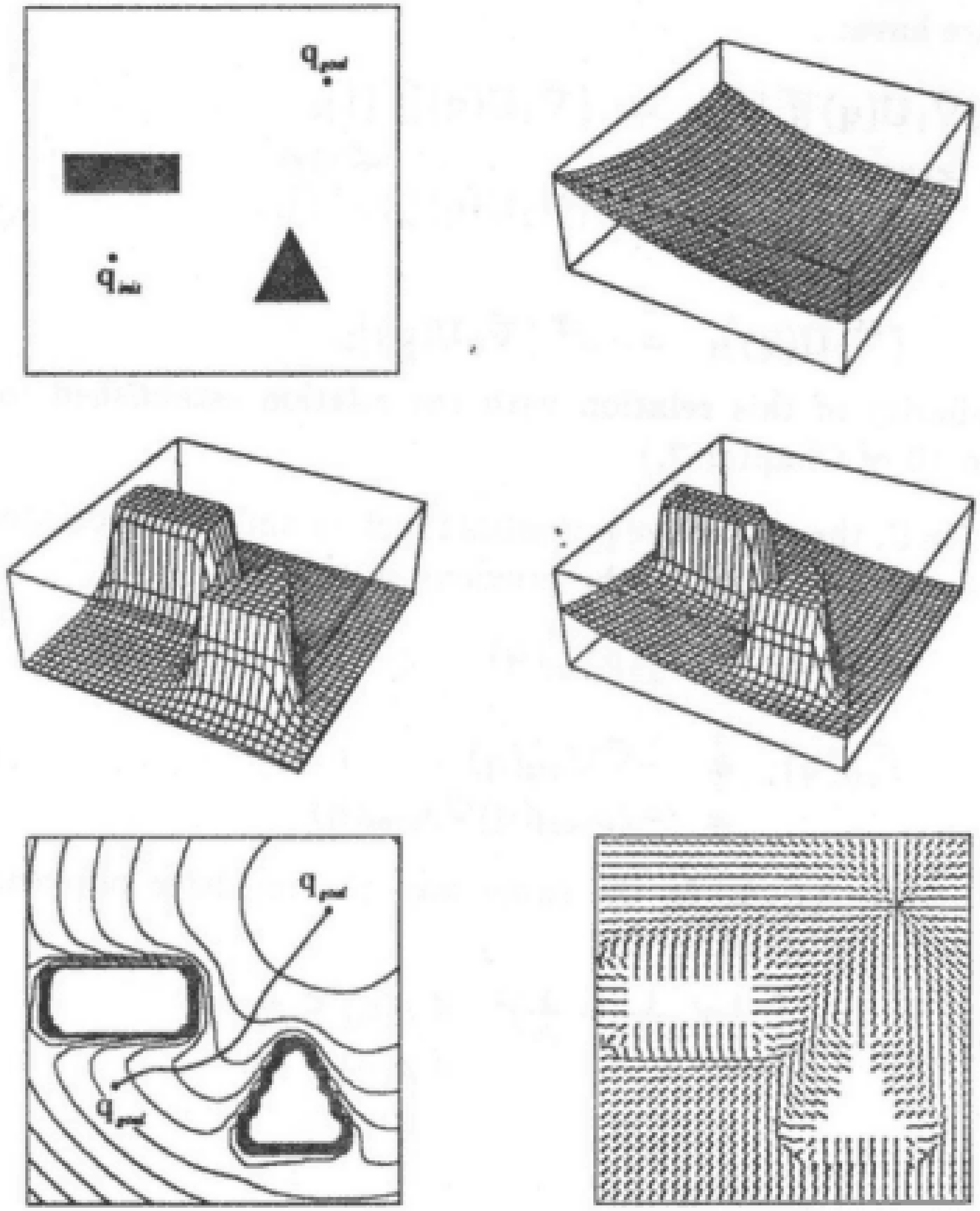
---

One definition of u: $u(q) = d_i(q) - d_g(q)$

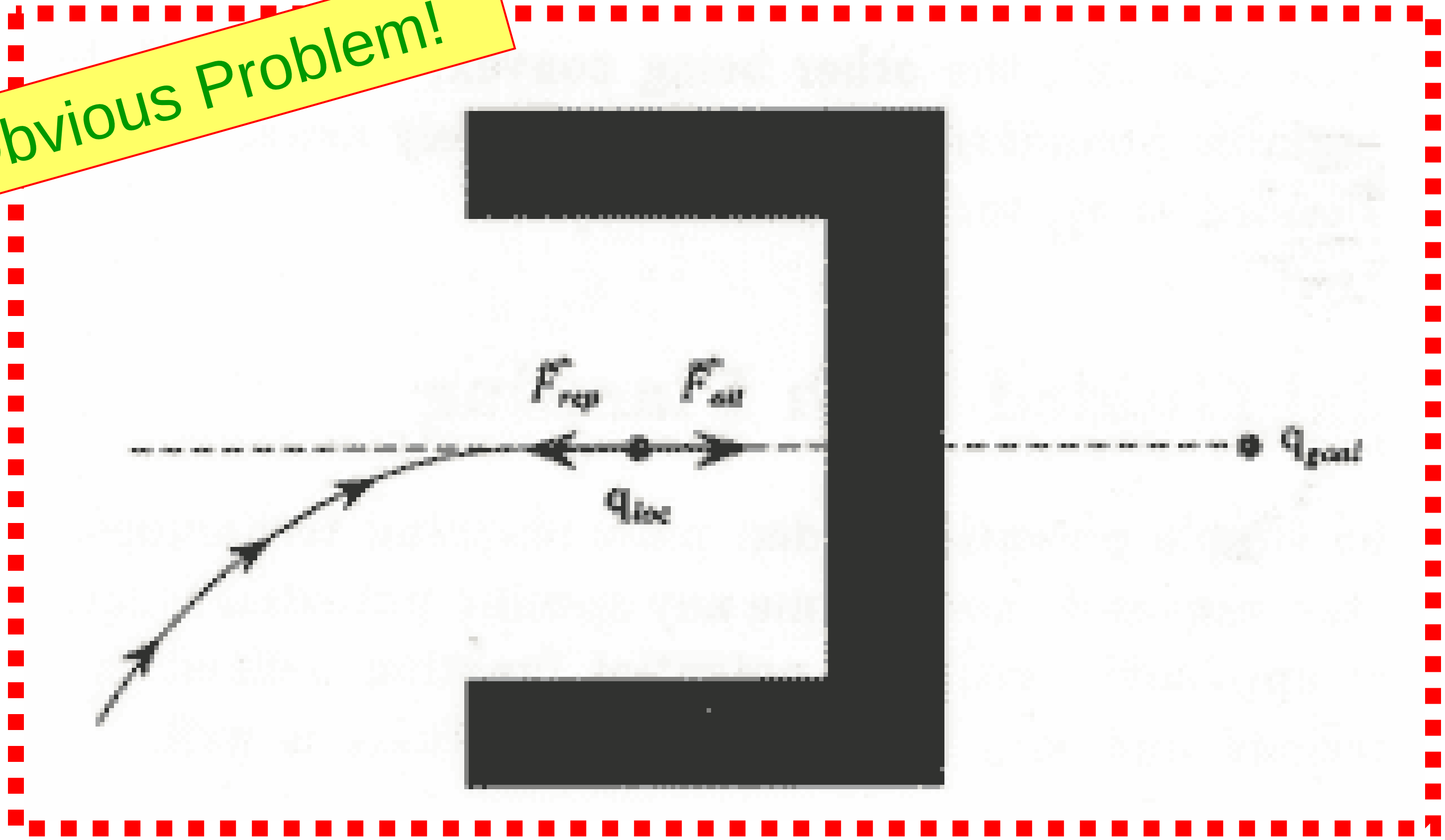
Preferred definition: $u(q) = \frac{1}{2}\sum\left(d_g(q)\right)^2 + \frac{1}{2}\eta\frac{1}{d_i(q)^2}$

SIMPLE MOTION PLANNER: Steepest Descent on *u*

Potential Field Example



Spot the Obvious Problem!



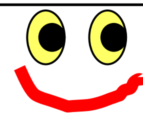
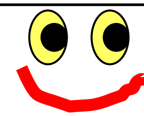
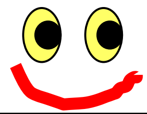
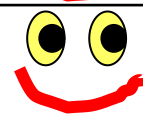
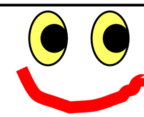
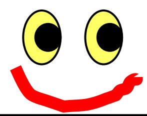
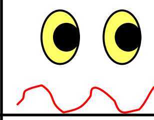
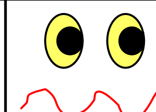
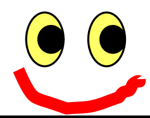
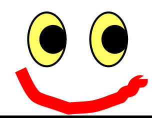
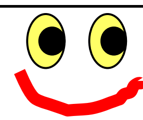
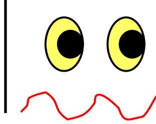
Solution I:

Use special local-minimum-free potential fields (Laplace equations can do this) – But very expensive to compute

Solution II:

When at a local minimum start doing some searching
- example soon

Comparison

	Potential Fields	Approx Cell Decomp	Voronoi	Visibility
Practical above 2 or 3 D?				
Practical above 8 D?				
Fast to Compute?				In 2-d
Usable Online?		 ?	 ?	
Gives Optimal?				In 2-d
Spots Impossibilities?				
Easy to Implement?			 ?	