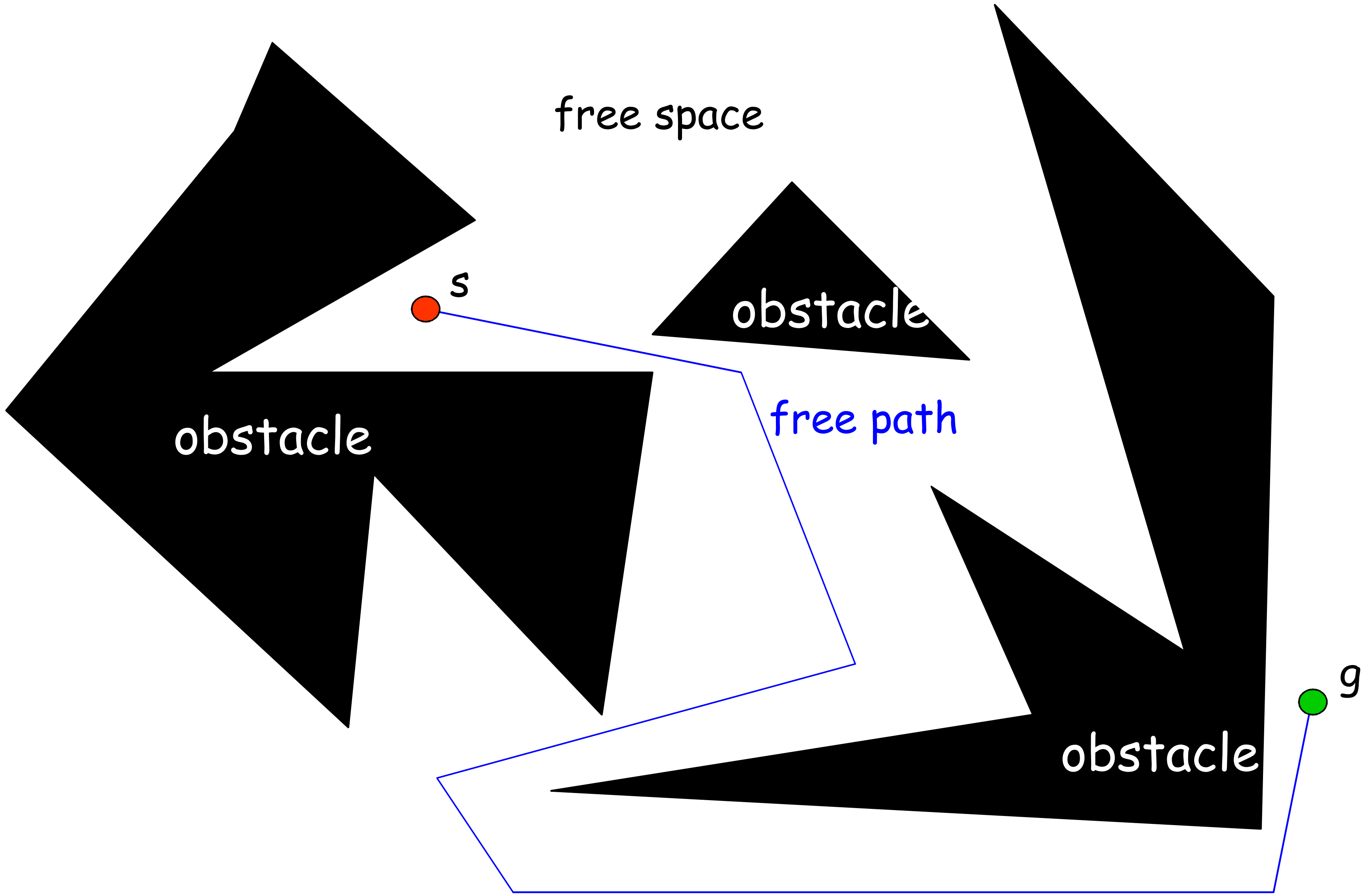
Planning

Planning

- plan = sequence of actions
  - to get from current state to goal state
- planning = find plan
  - using search algorithms
- often *domain* specific
  - i.e., using knowledge
    about the underlying problem structure
- example: path/motion-planning

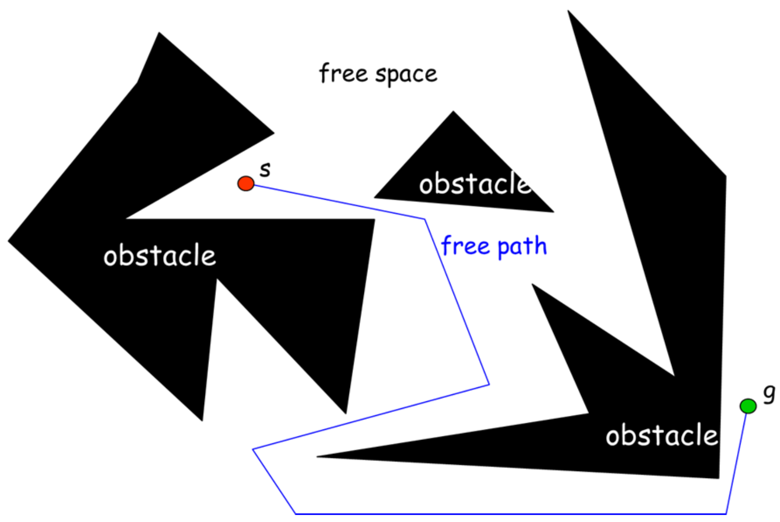
Path/Motion Planning

The Path Planning Problem



The Path Planning Problem

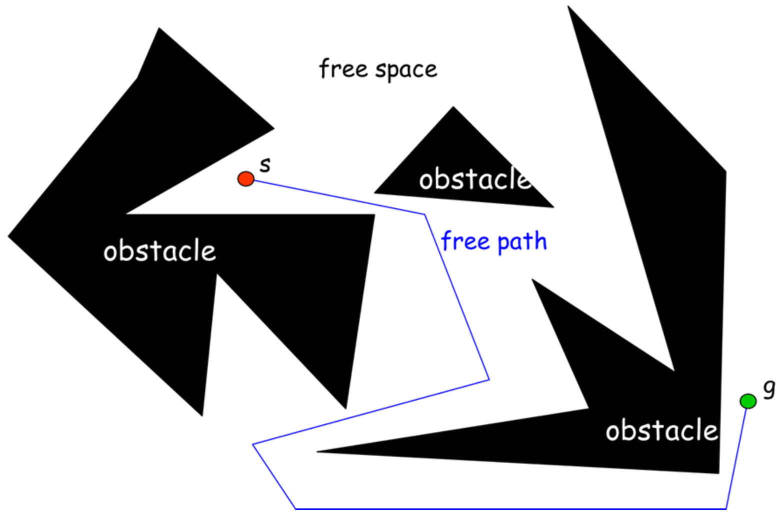
- given
  - map with free space and obstacles
  - start *s* and goal *g* points (or poses)
- find path, i.e.,
  - sequence of points p_i
  - through free space
  - connecting *s* and *g*



The Path Planning Problem

find path as sequence of points p_i

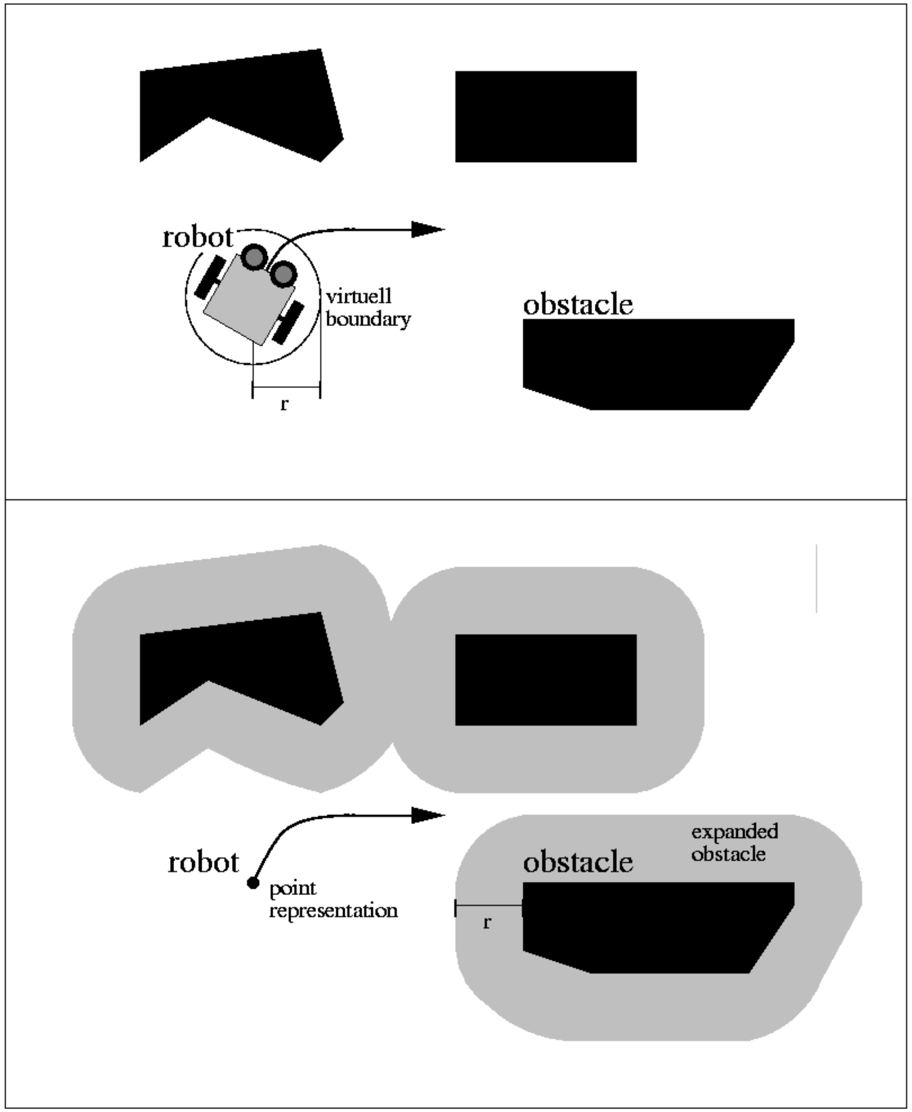
- assuming canonical actions
- to get from p_i to p_{i+1}



Representing the Mobile System

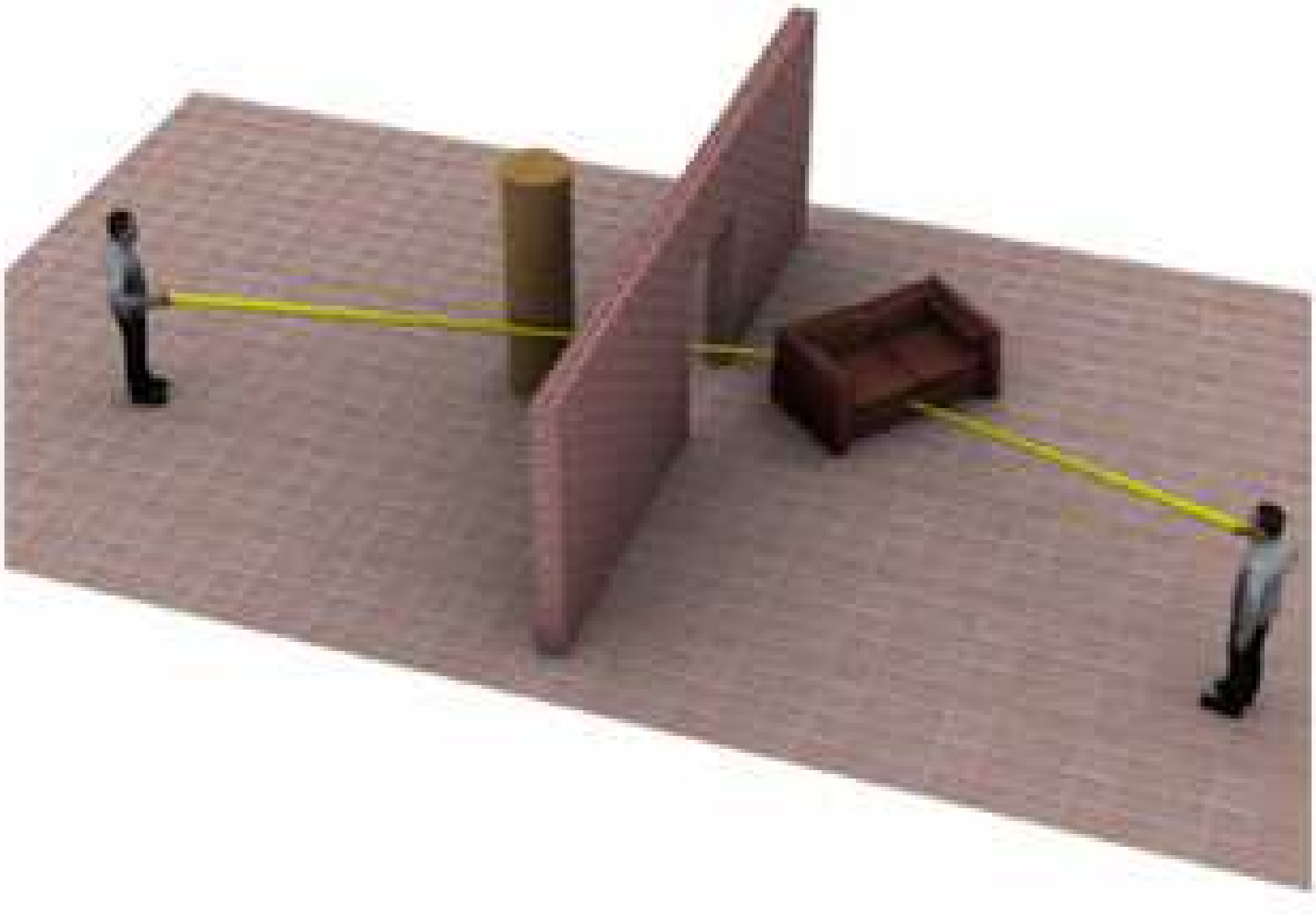
- treat system as point
- increase obstacles by e.g. system radius

aka obstacle growing



Motion Planning

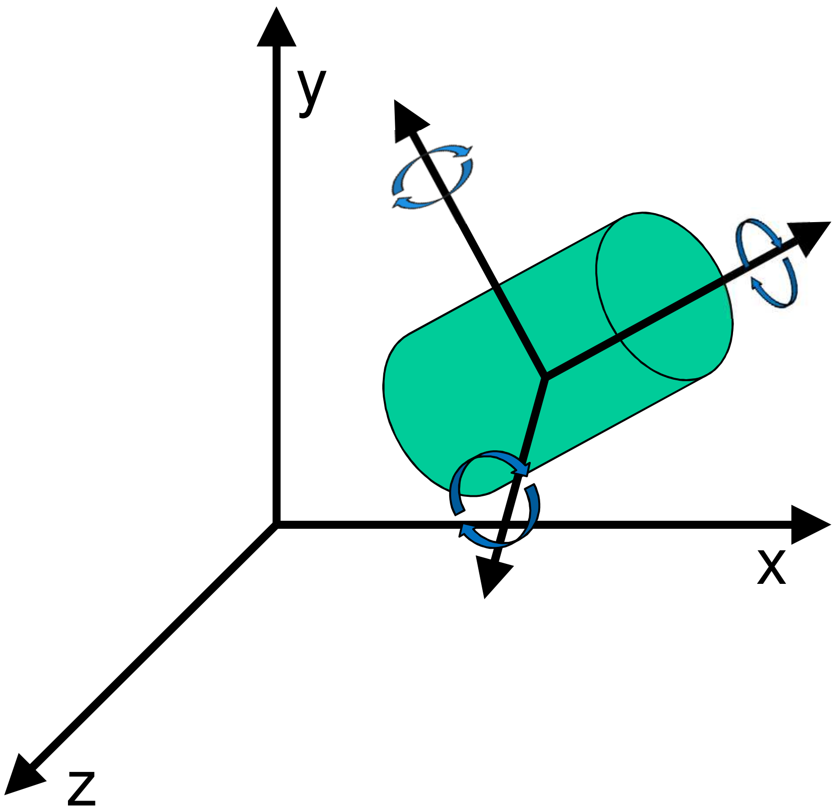
things can get more complicated...



Rigid Body in 3D

- not only point ***x*** for position
- but also orientation

=> ***pose*** (position & orientation)

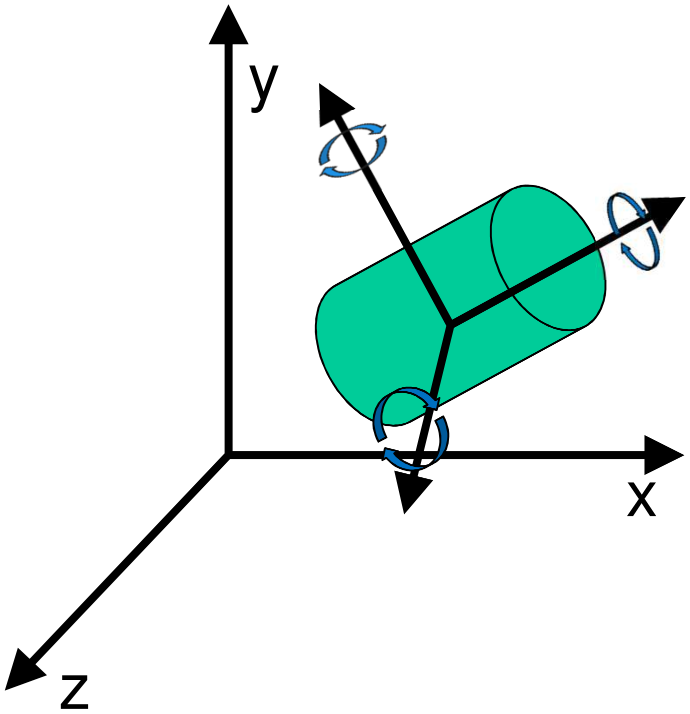


e.g.: via Roll, Pitch, Yaw

Rigid Body in 3D

Degree of Freedom, DoF:

= number of independent motion variables

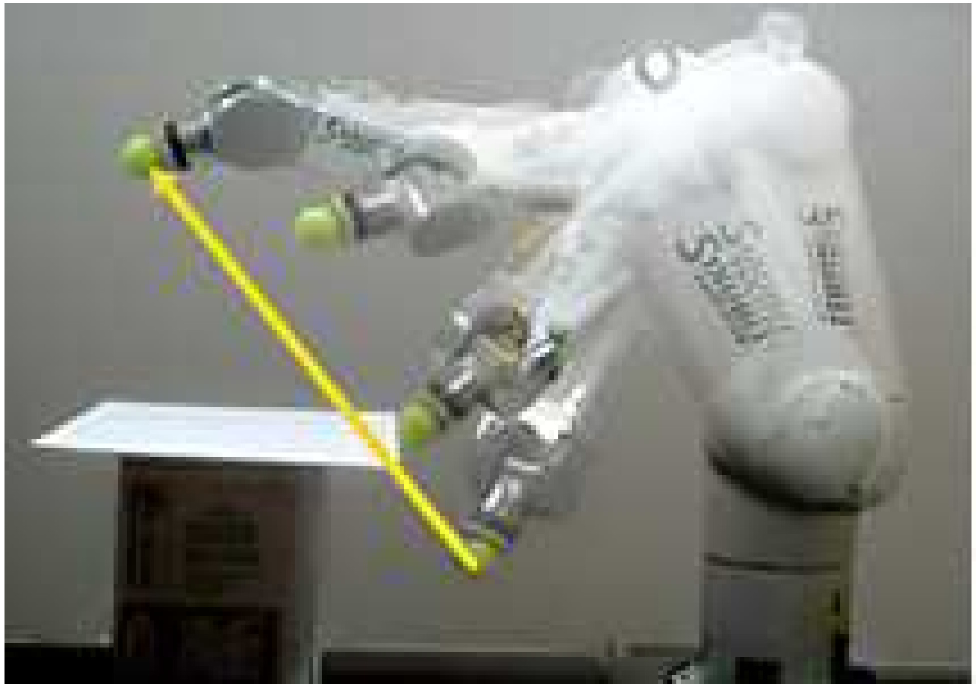


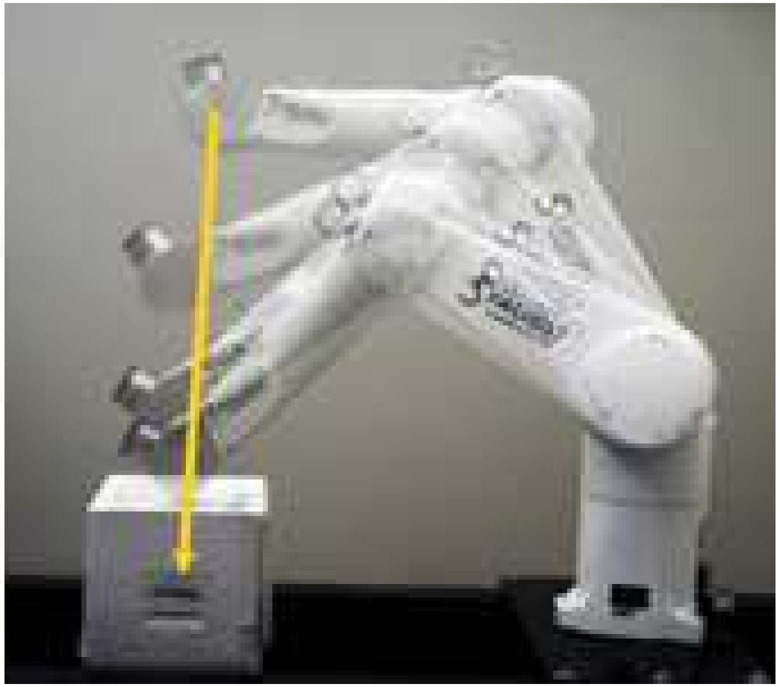
rigid body **pose** in 3D: 6 DoF

- 3 DoF translation (position)
- 3 DoF rotation (orientation)

Configuration Space

Motion Planning





e.g., interested in collision-free motion of robot hand

- system consists of moving parts
- not an option to treat hand/system as a point
- and just use obstacle growing

Space Representation for Motion Planning

- work space (WS)
  - physical space
  - with start & goal(s)
  - physical obstacles and free space
- configuration space (CS)
  - space spanned by system's DoFs
  - configuration = vector in CS
  - i.e., values for each DoF
  - e.g., angles for robot arm joints

Space Representation for Motion Planning

- configuration space (CS)
- forbidden space (in CS)
  – aka obstacle space in CS
  – configurations that lead to collisions (or otherwise undesirable states)
- free space (in CS)
  – CS minus forbidden space
  – i.e., all configurations that do not lead to collisions

Space Representation for Motion Planning

- configuration space (CS)
- forbidden/obstacle space (in CS)
- free space (in CS)

motion planning:

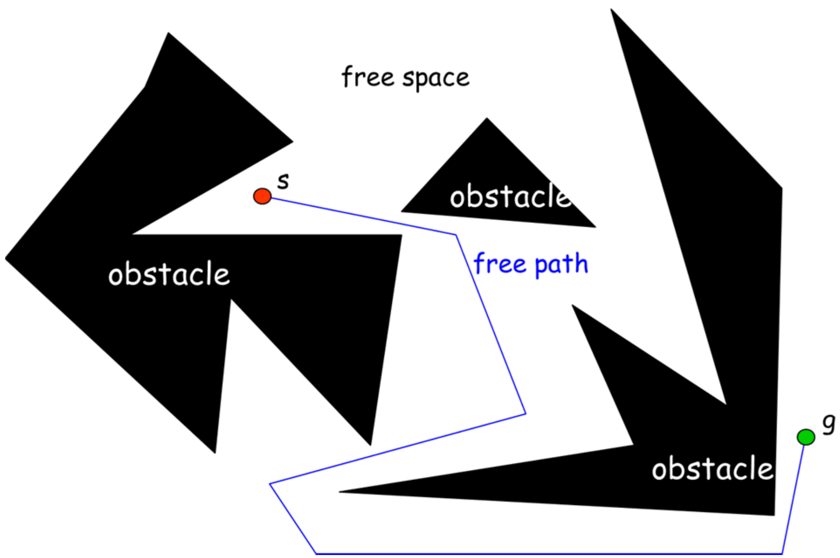
find collision free path

= path through free space (in CS)

Space Representation for Motion Planning

important note:

- if moving system is a point
  - e.g., “simple” mobile system & obstacle growing
- then configuration space = work space
  - aka path planning

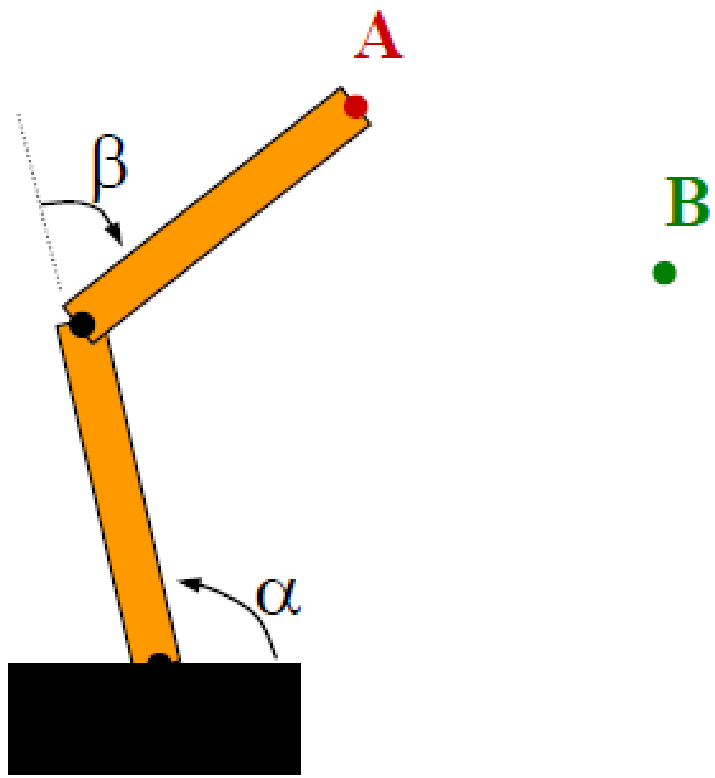


Example Motion Planning

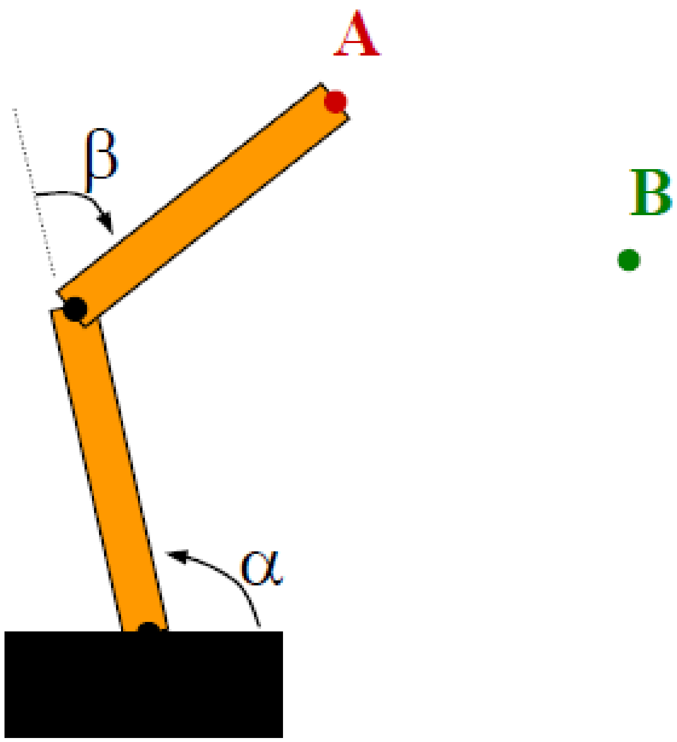
2D robot arm

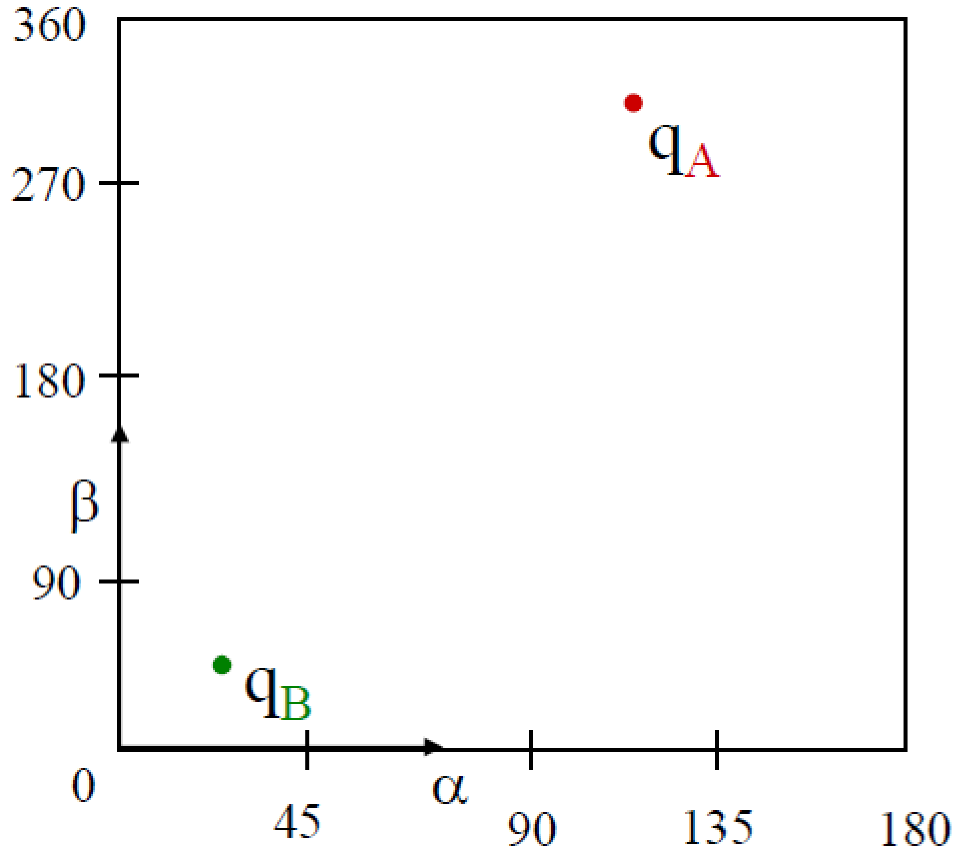
- with 2 DoF
- rotational joints α, β

configuration space?



Example

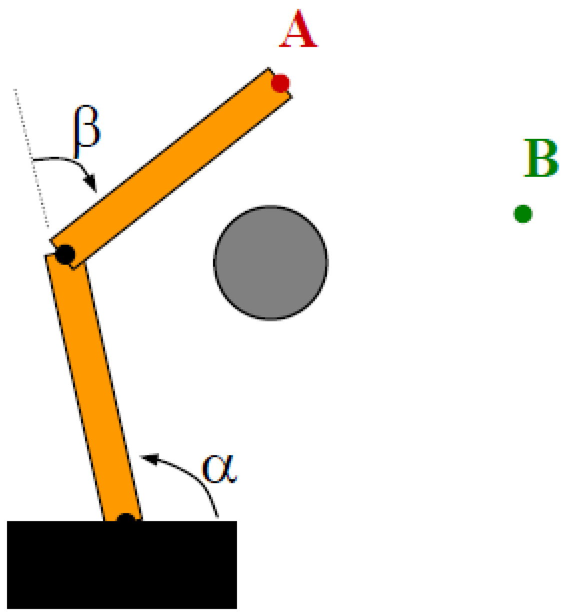


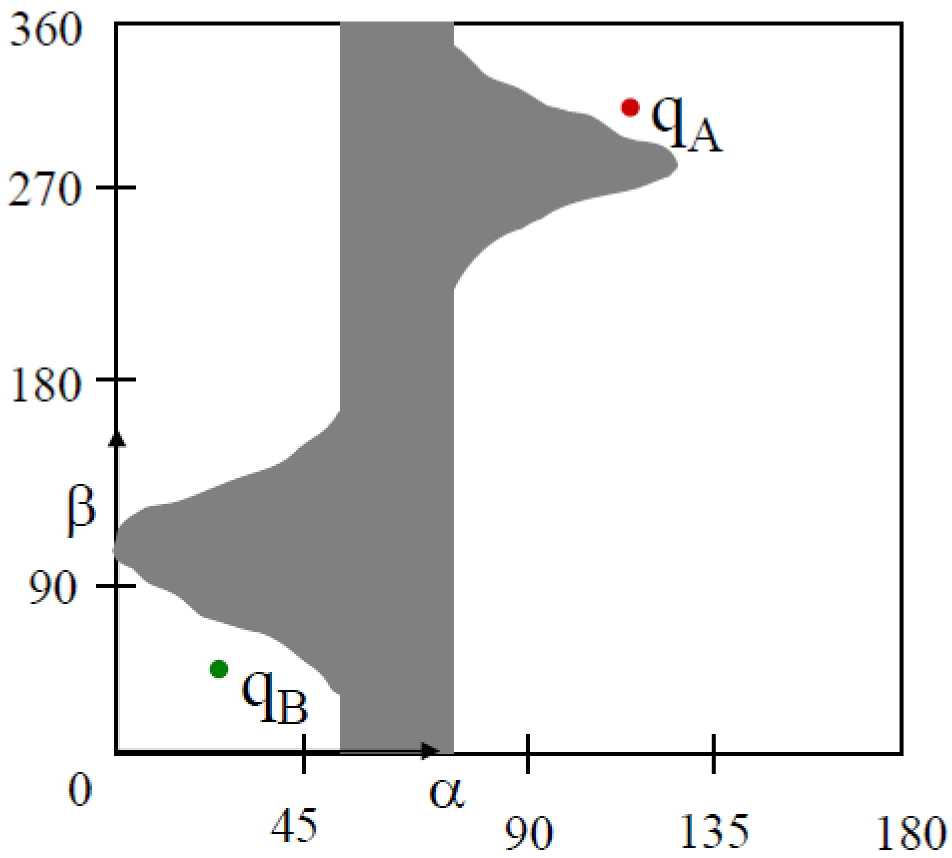


suppose α, β are not constrained, i.e., can „freely rotate around“

⇒ CS is a torus

Example

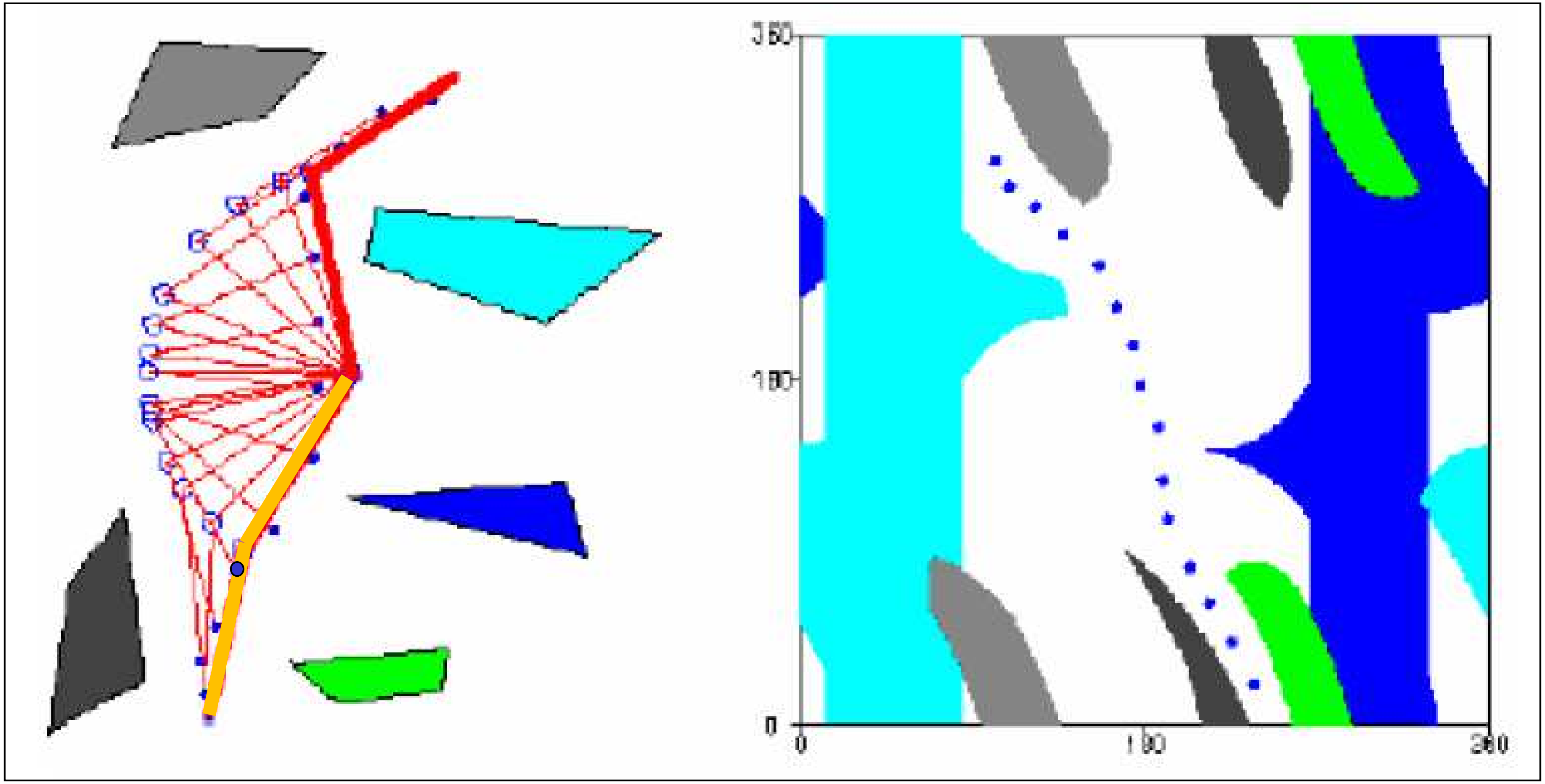




an obstacle in the CS

Further Example

2 links and 2 joints, 5 obstacles



start config.

goal config.

intermediate configurations (during execution of the plan)

● position of the “hand” in physical, respectively configuration space

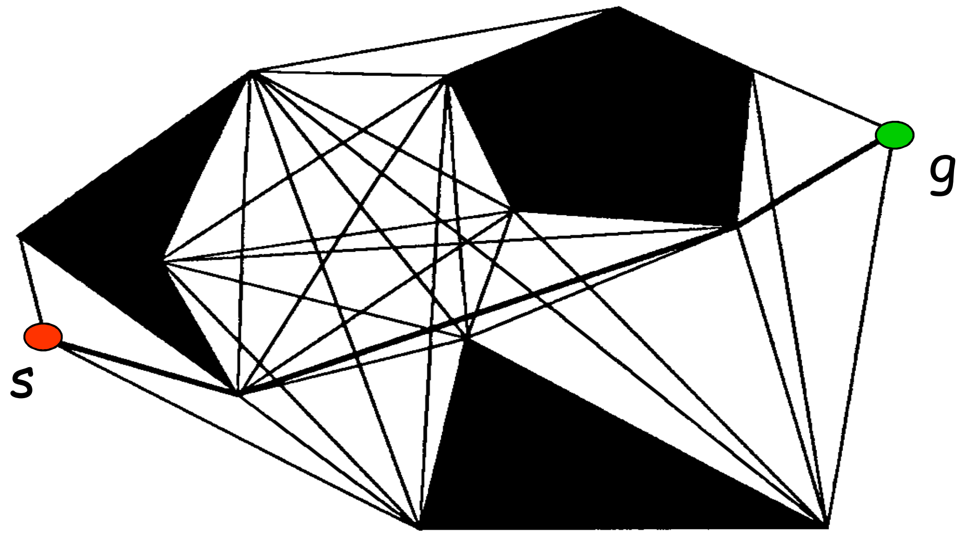
Free Space Representations

Representation Approaches

- Roadmap
  - represent connectivity of free space by a network
- Cell decomposition
  - decompose free space into simple cells
  - connectivity = 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

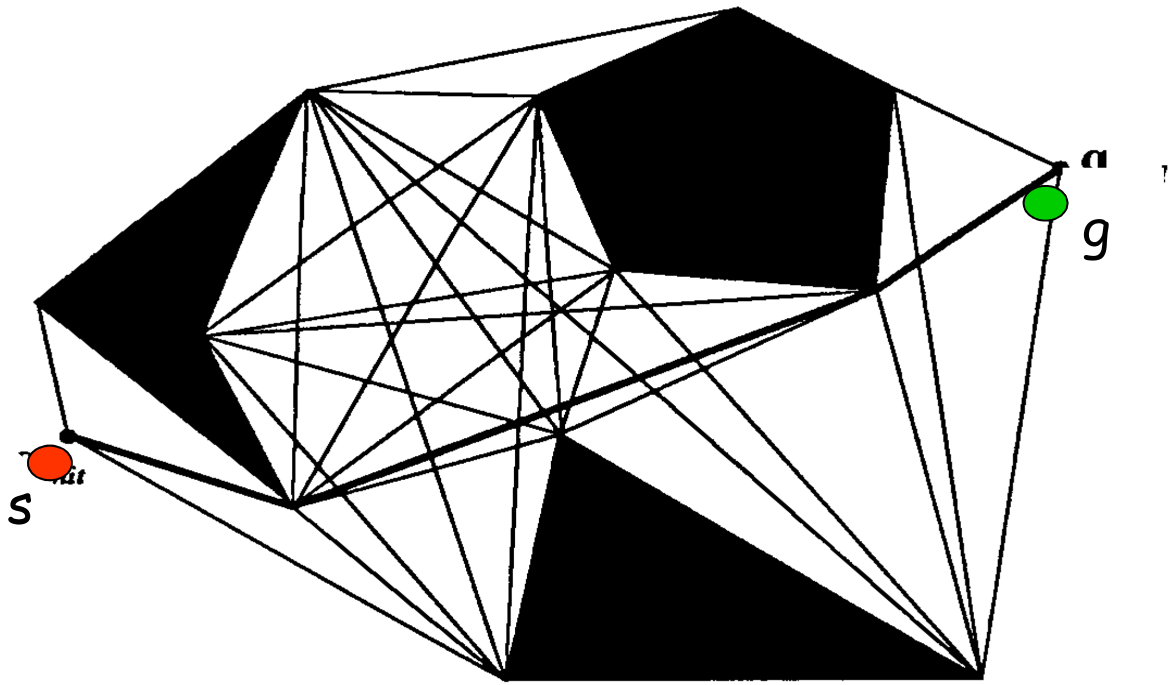
Roadmap: Visibility Graph

- Shakey (late 60s)
- polygons for obstacles
  - vertices, aka (path) nodes, and (obstacle) edges
- connect obstacle corners
  - and start *s* and goal *g*
  - with free space edges aka (path) segments



Roadmap: Visibility Graph

- given roadmap (here visibility graph)
- find path: e.g., A*



Simple Algorithm

1. add all obstacles vertices in VG, plus start and goal
2. For every pair of nodes u, v in VG
3. If segment(u,v) is an obstacle edge then
4. insert (u,v) into VG
5. else
6. for every obstacle edge e
7. if segment(u,v) intersects e
8. then goto 2
9. insert (u,v) into VG

Complexity

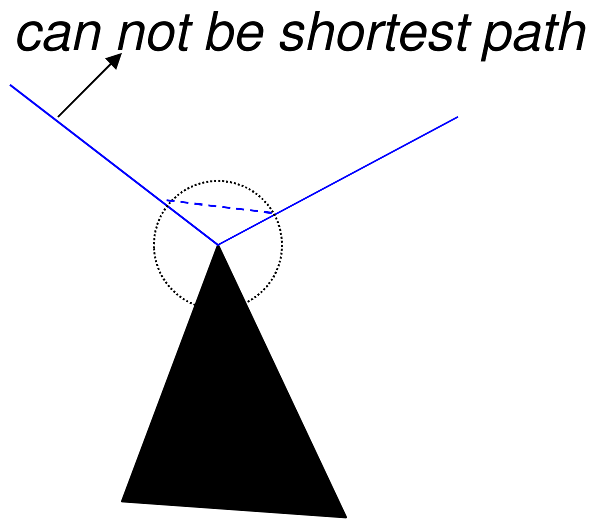
Space: $O(n^2)$

Time:

- Simple algorithm: $O(n^3)$ time
- Rotational sweep: $O(n^2 \log n)$
- Optimal algorithm: $O(n^2)$

Reduced Visibility Graph

- aka Generalized Visibility Graph
- aka Tangent(ial) Graph

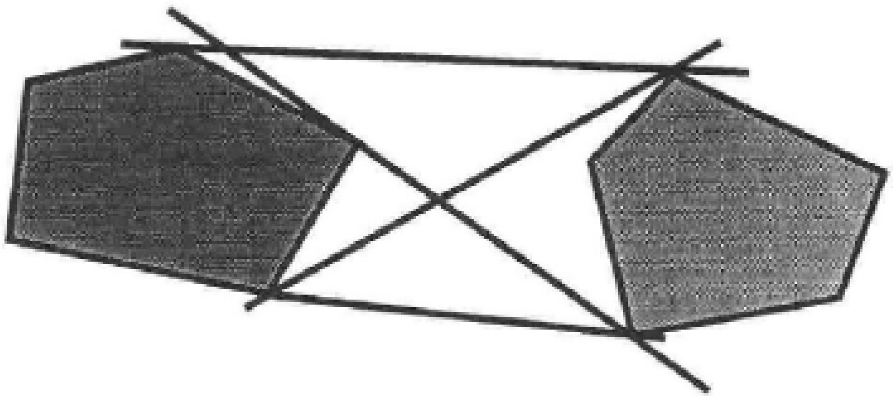


Eliminate
- concave obstacle vertices
- and non-tangent segments

Reduced Visibility Graph

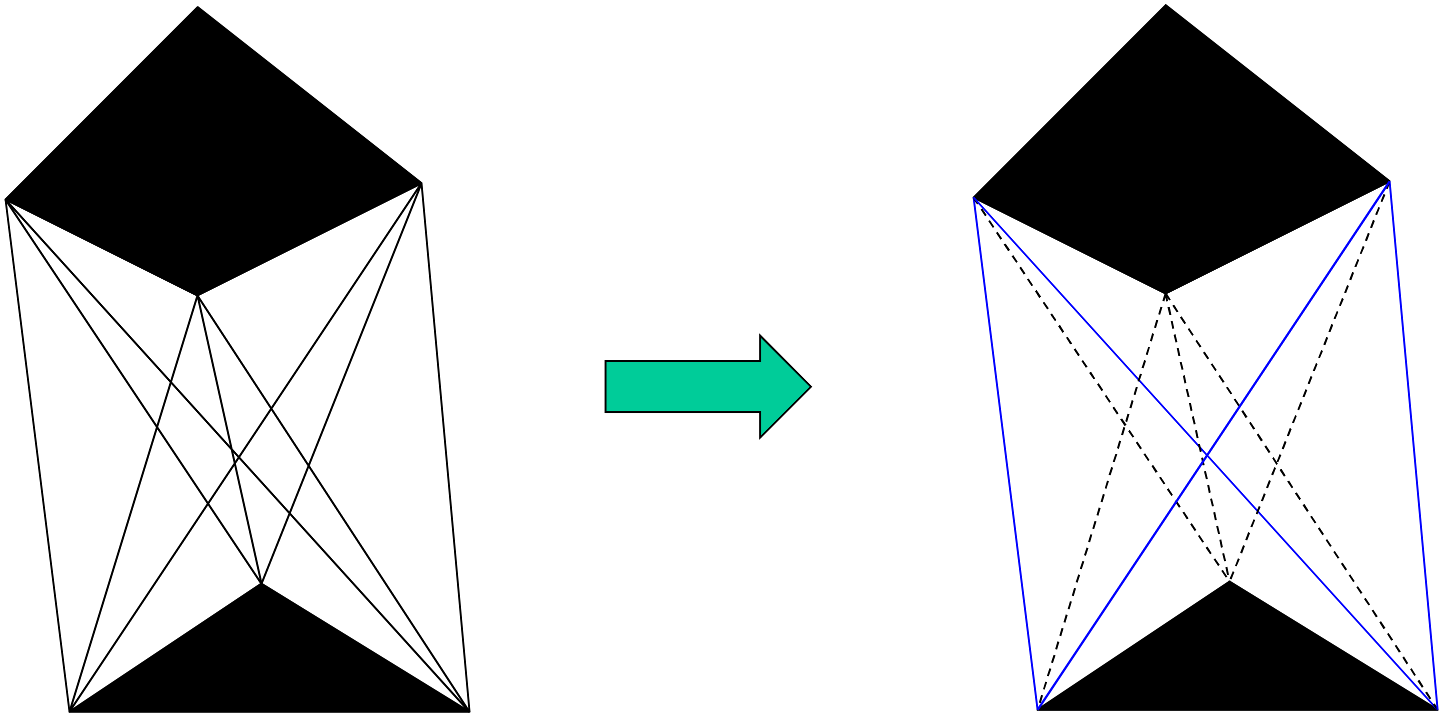
tangent segment = both its vertices are tangent points

Def.: if a line *L* contacts obstacle vertex *p* but does not intersect any internal point in a small neighboring obstacle region of *p*, then *L* is tangent to *p* and *p* is a tangent point



two separated polygons => 4 tangents

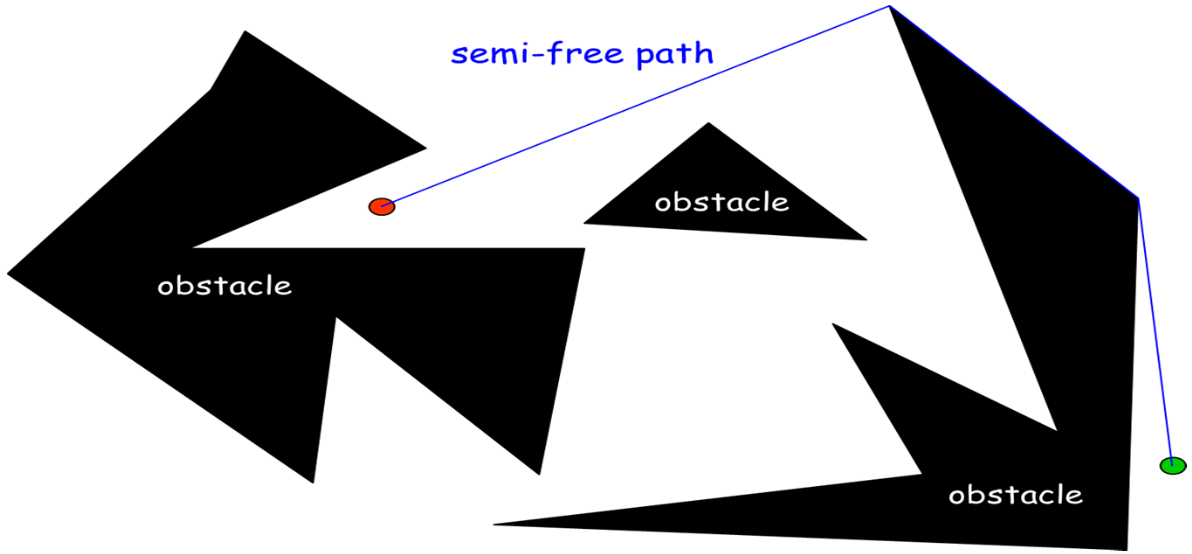
Reduced Visibility Graph



tangent segments

Problem with Visibility Graph

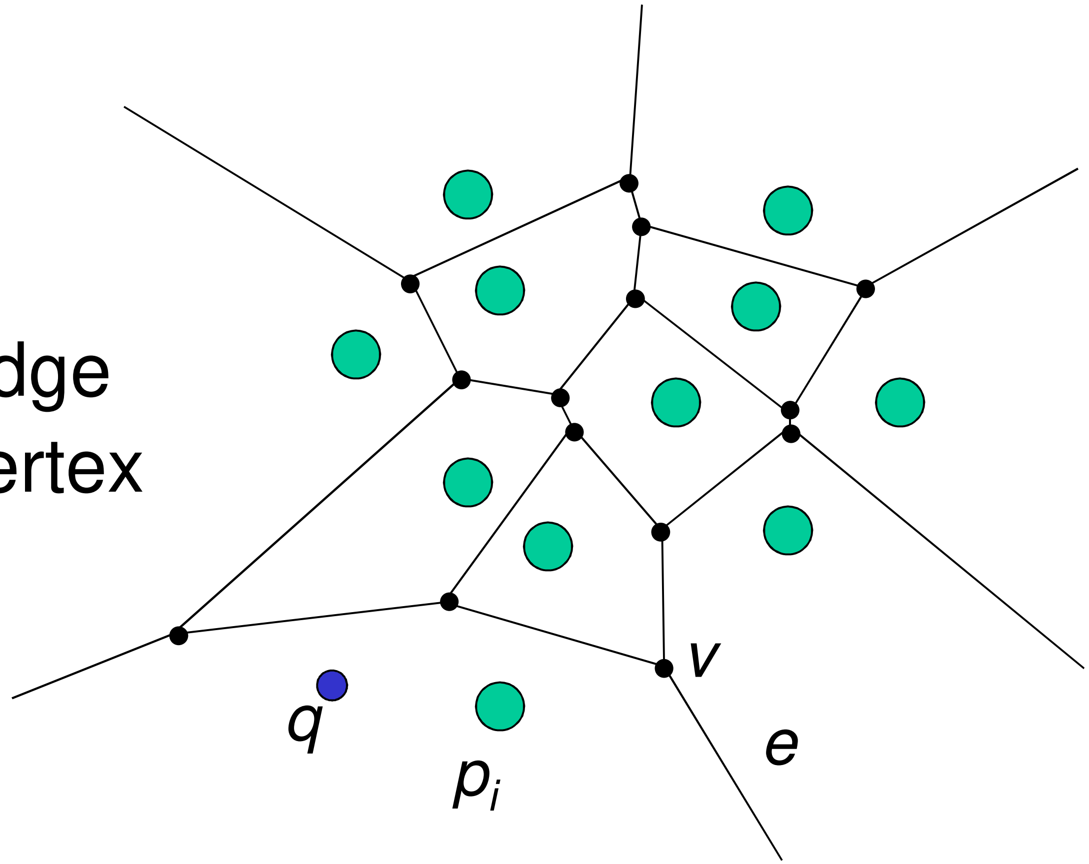
- can lead along obstacles
- aka semi-free path



how to get more, resp. max clearance?

Voronoi Diagram

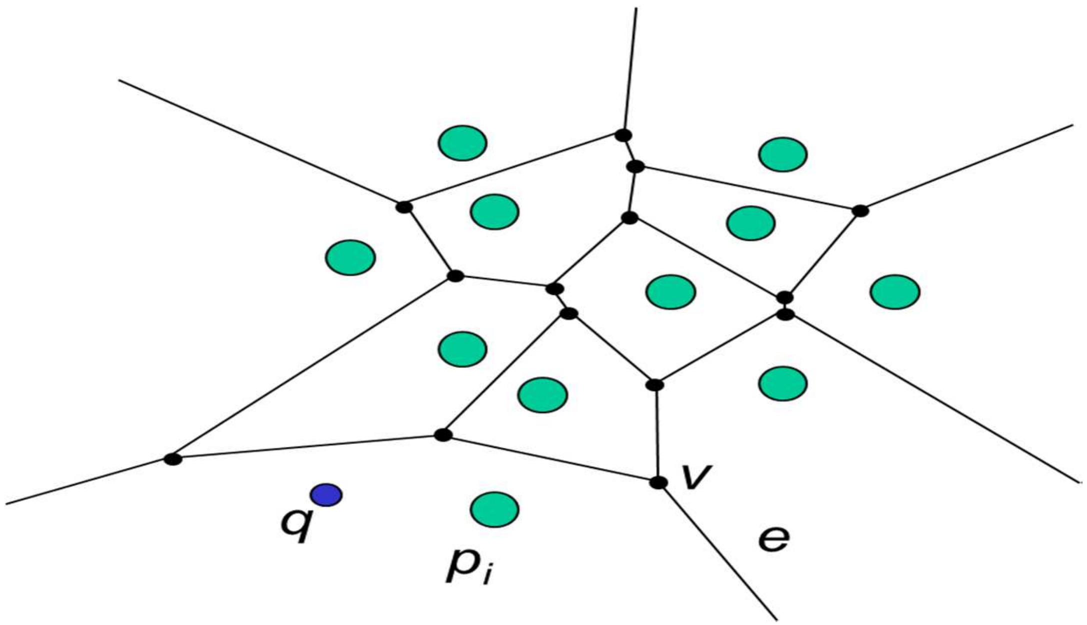
p_i : site points
q : free point
e : Voronoi edge
v : Voronoi vertex



- cells = areas with min distances to a vertex
- i.e., edges are equistant points between vertices

Definition of Voronoi Diagram

- Let P be a set of n distinct points (aka sites) in the plane
- The Voronoi diagram of P is the subdivision of the plane into n cells, one for each point
- A point q lies in the cell corresponding to a point $p_i \in P$ *iff* for each $p_i \in P, j \neq i$: Distance(q, p_i) < Distance(q, p_j)

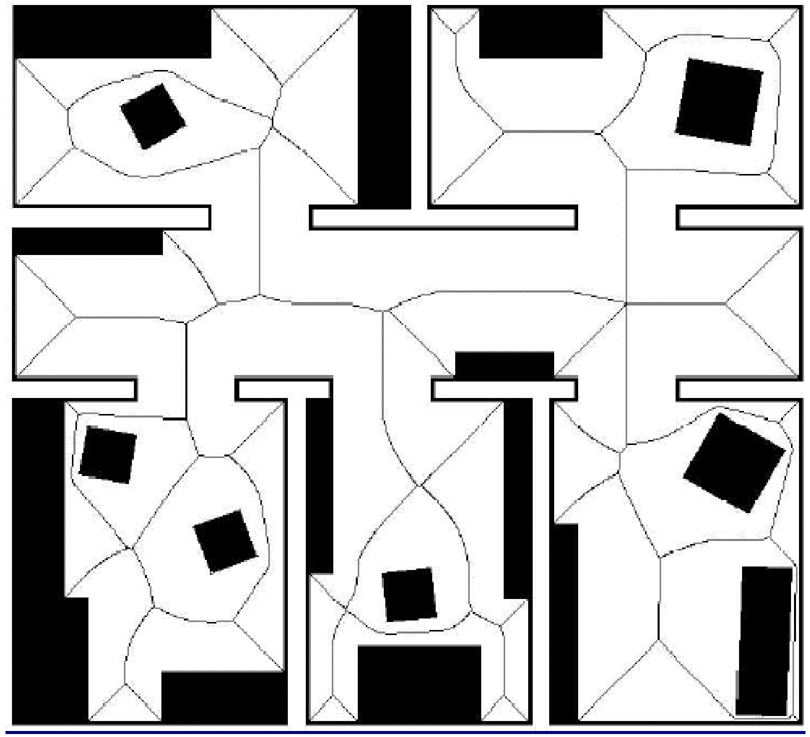


Generalized Voronoi Diagram (GVD)

instead of points

given set of polygons (obstacles)

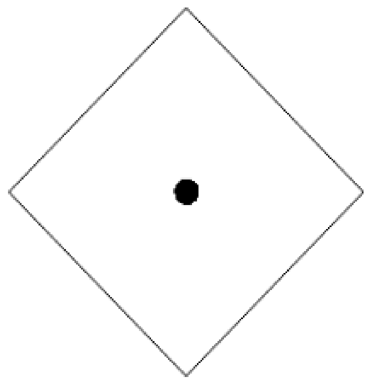
- GVD edges equidistant from the two closest obstacles
- Time: O(n log n)
- Space: O(n)



Voronoi Diagram: Metrics

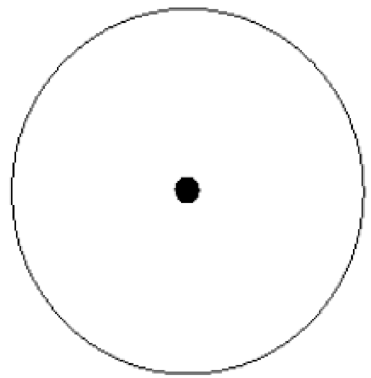
different options for distance, e.g.

- L_1: Manhattan
- L_2: Euclidean



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

L1

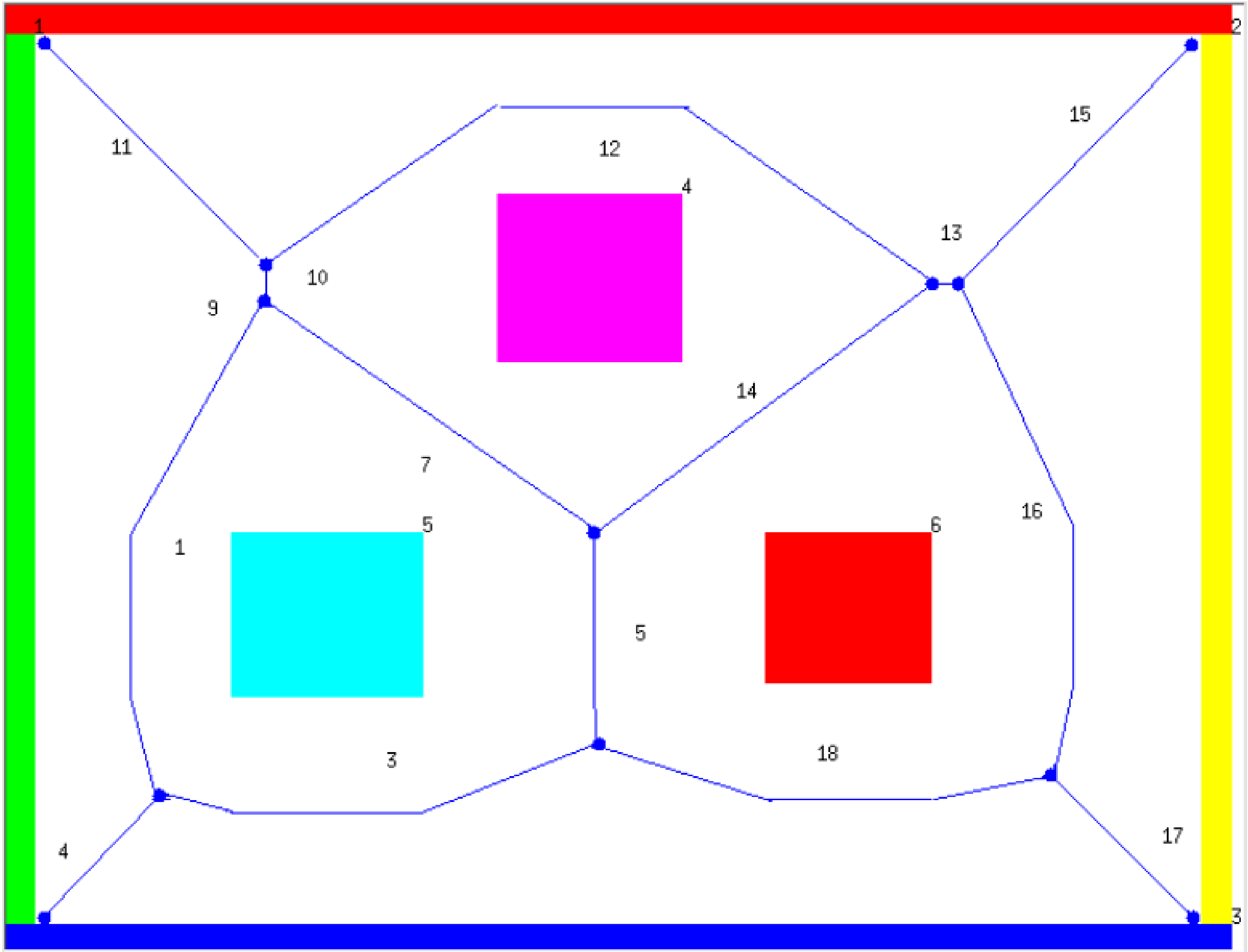


$\{(x,y) : x^2 + y^2 = \text{const}\}$

L2

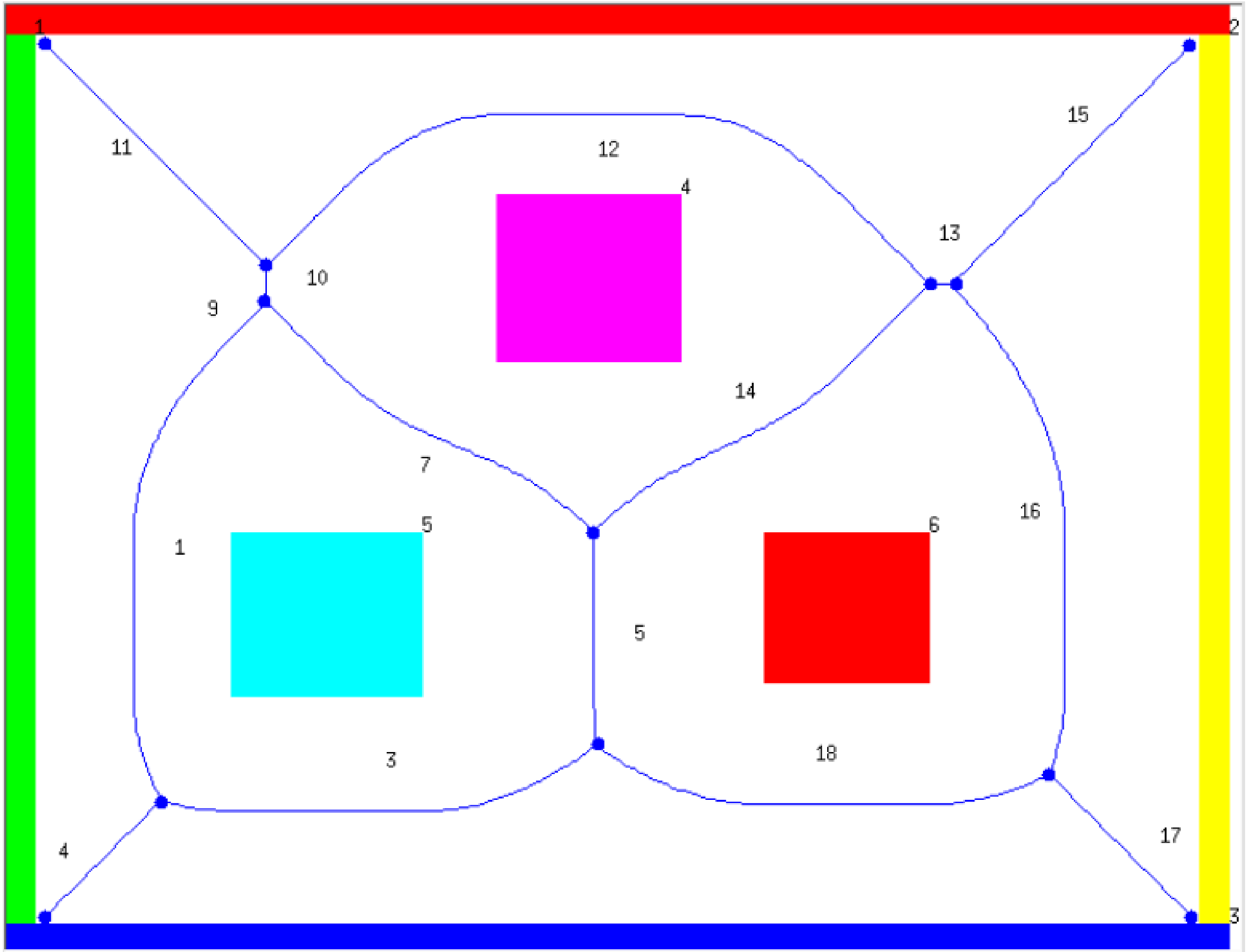
Voronoi Diagram: Metrics

- L_1
- note the straight lines



Voronoi Diagram: Metrics

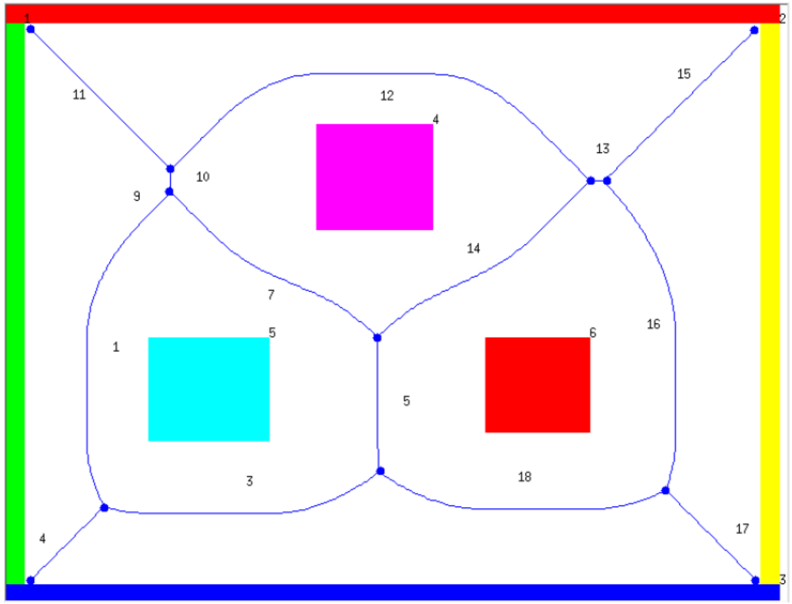
- L_2
- note the curved lines



Voronoi Diagram

polygonal obstacles
and Euclidean metric

set of points equidistant to
- 2 vertices = line
- 2 edges = line
- 1 vertex and 1 edge = parabola



Time: O(n log(n))
(both for point sets and polygon representation)

Path-Planning Approaches

- Roadmap
- **Cell decomposition**
- Potential field

Cell-Decomposition Methods

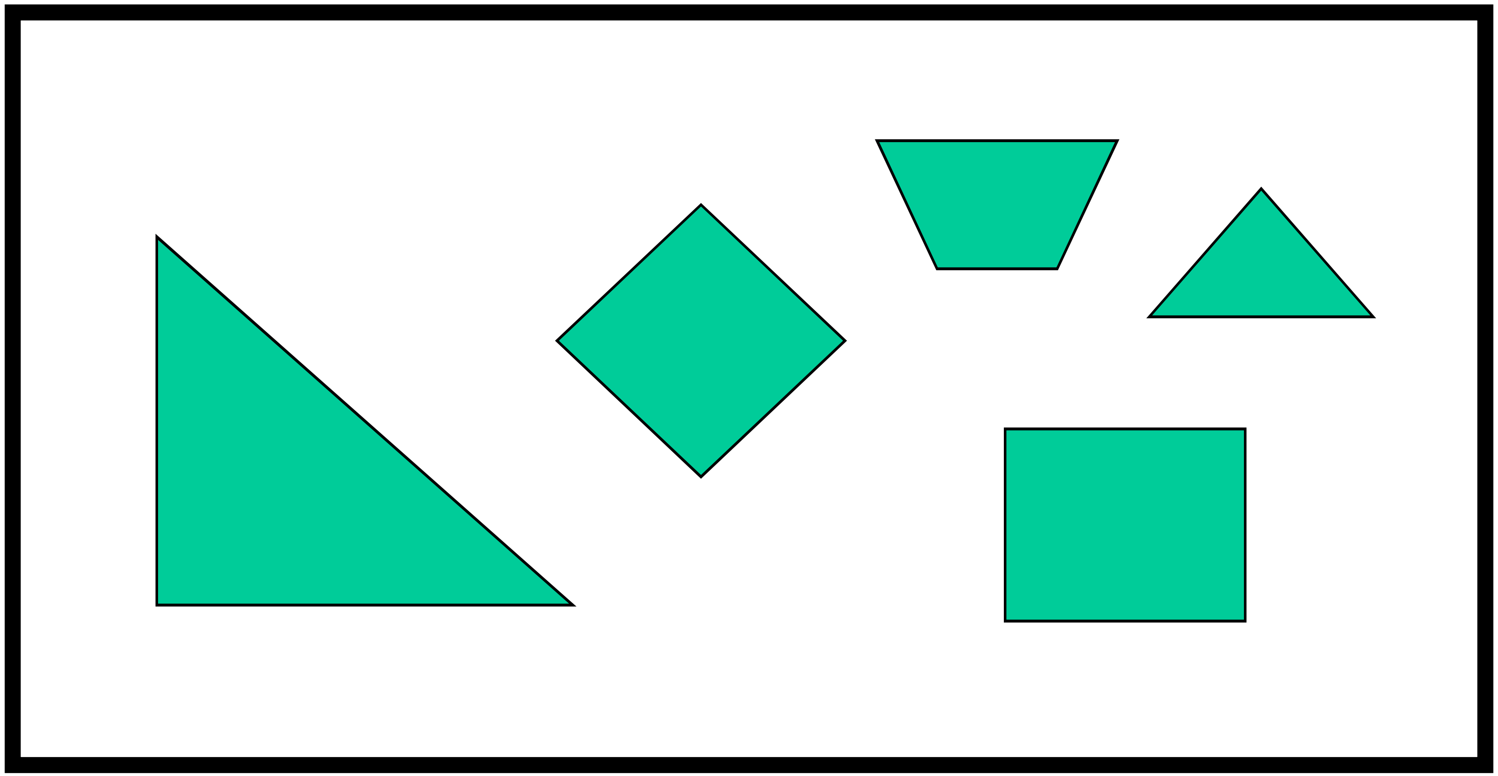
free space F represented
by non-overlapping cells

two classes:

- exact
  – union of cells is exactly F
- approximate
  – union of cells is contained in F

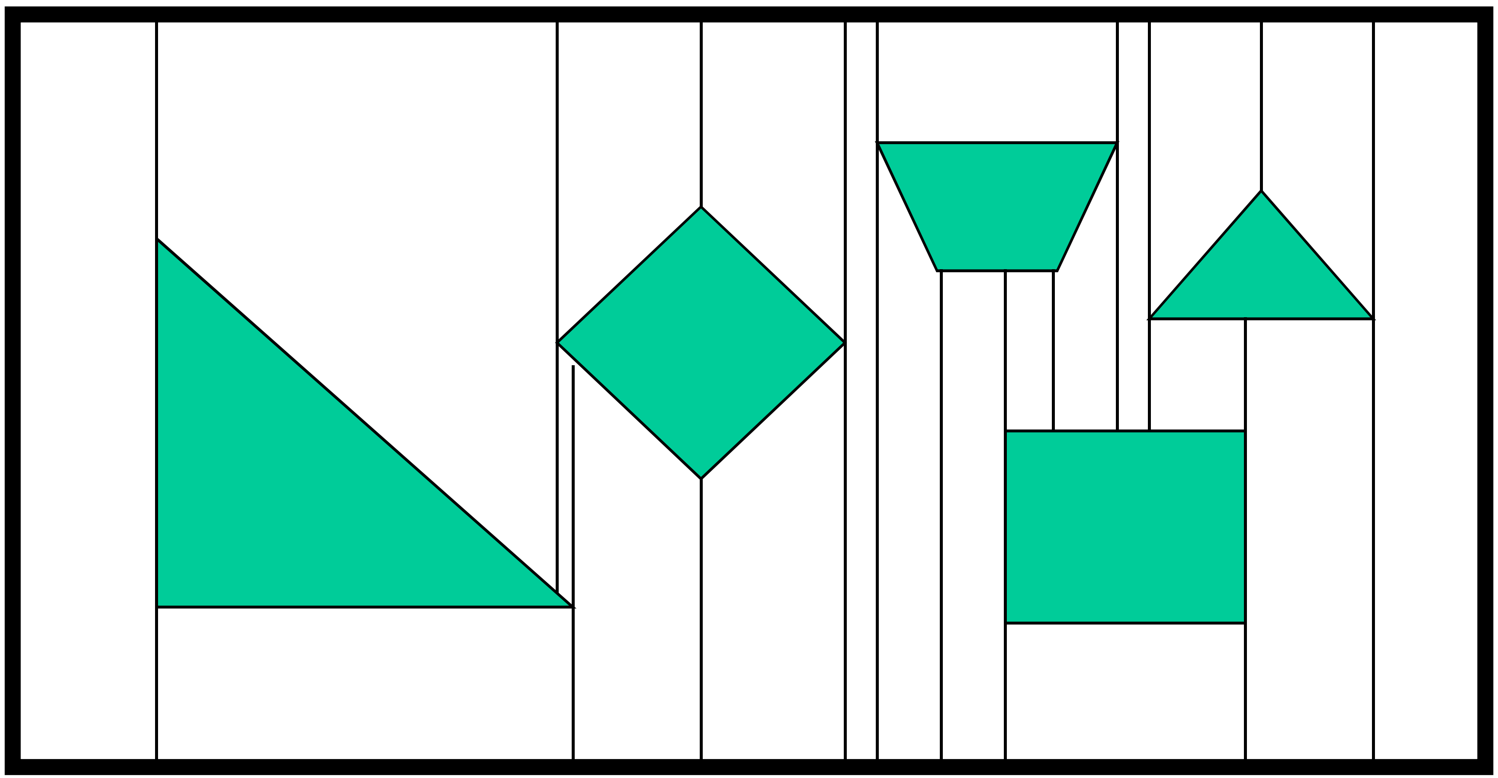
Exact: e.g. Trapezoidal Decomposition

obstacles = polygons



Trapezoidal Decomposition

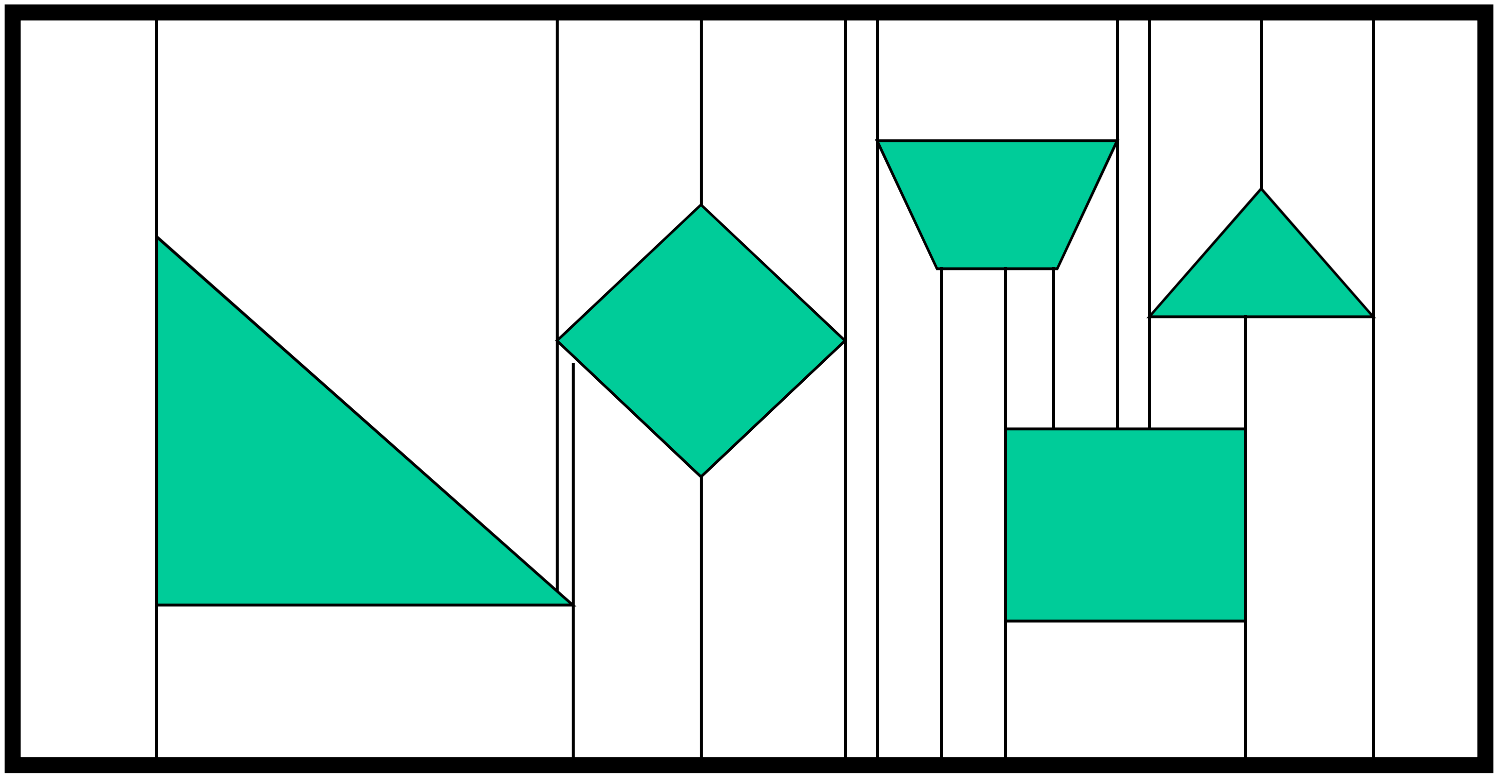
extend vertical line at every obstacle vertex
until it touches an other obstacle



Trapezoidal Decomposition

easy and efficient to compute with a sweep line alg.

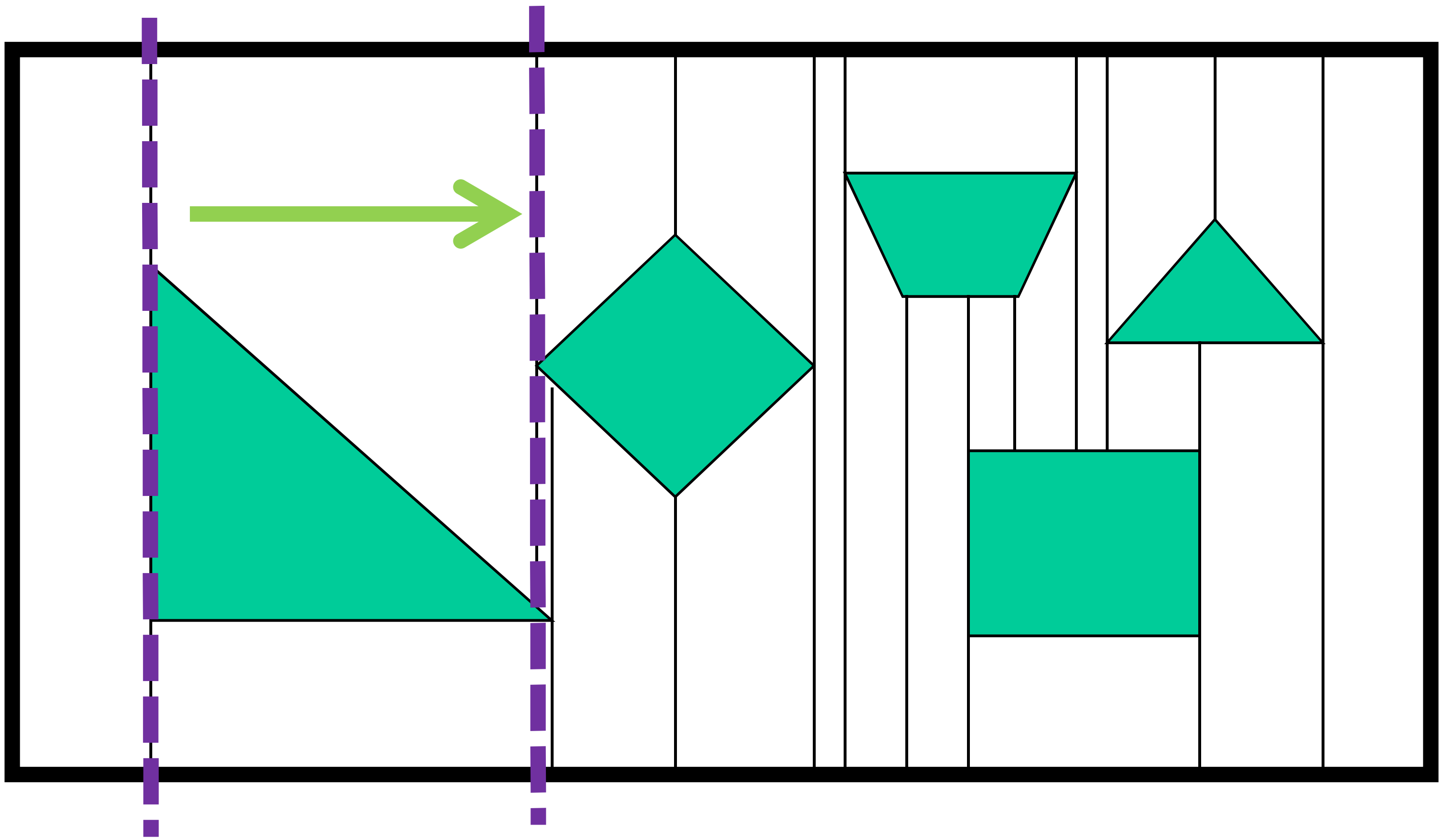
O(n log n) time, O(n) space



Trapezoidal Decomposition

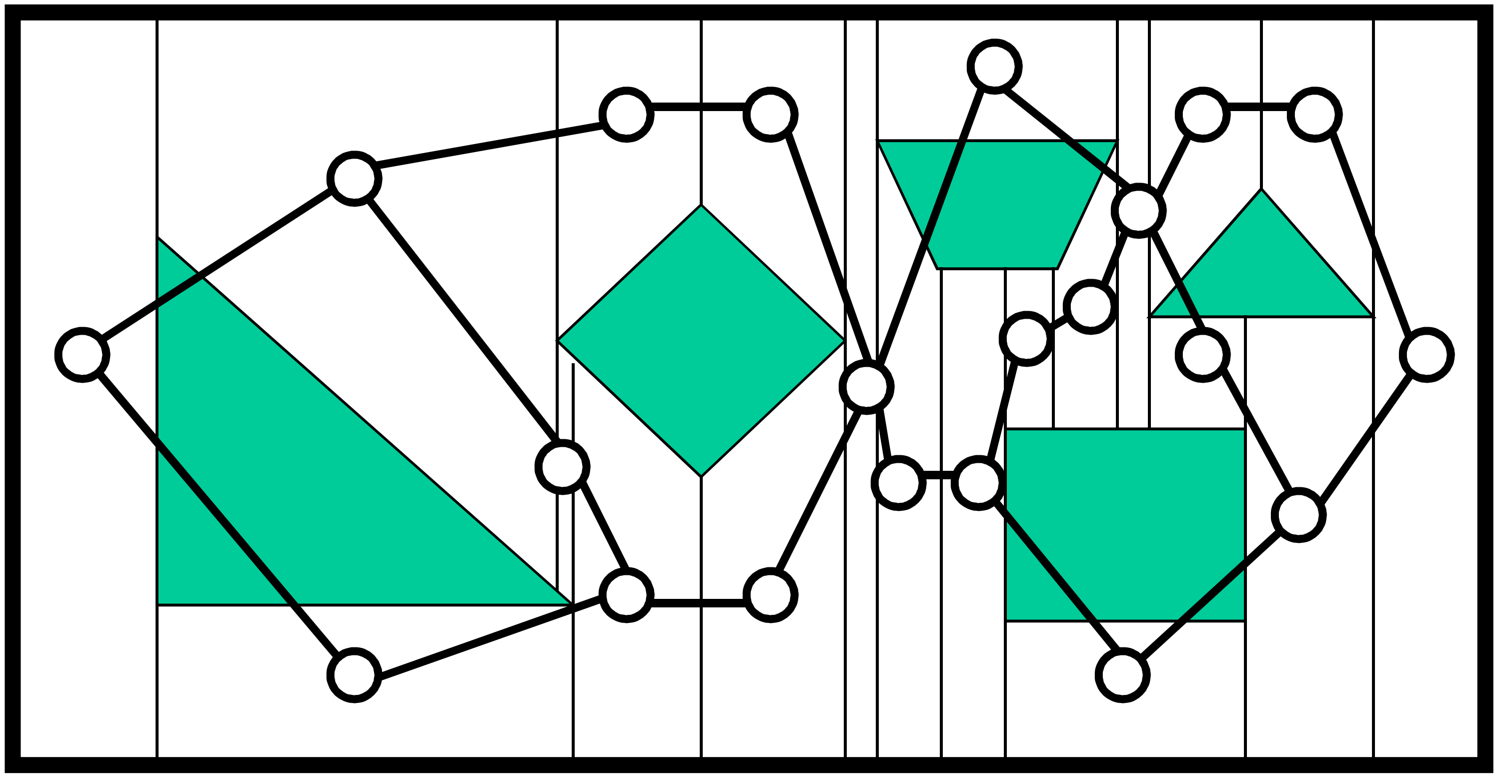
planar sweep line:

sort by x-coordinates, iterate over them



Trapezoidal Decomposition

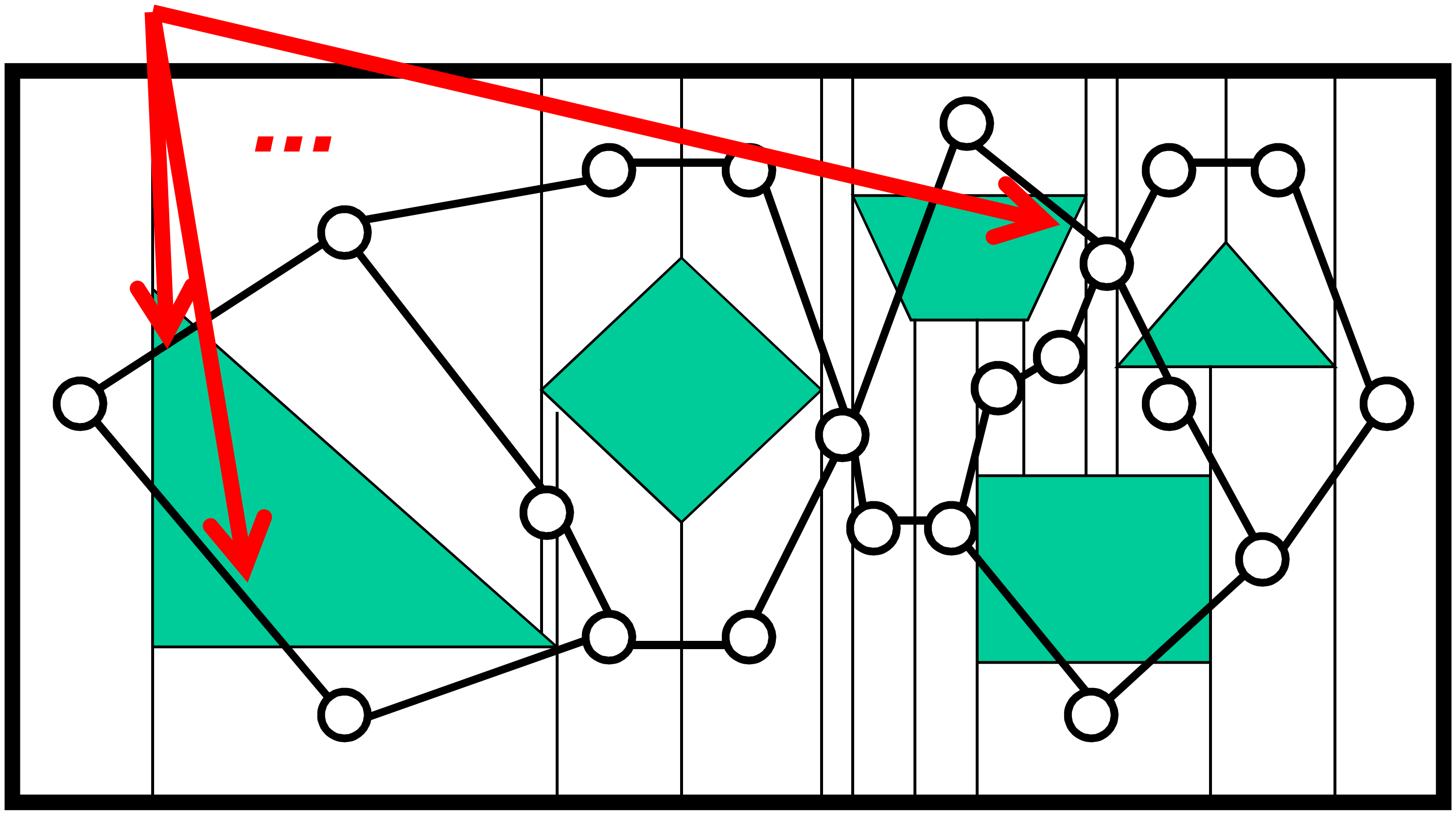
- can use centers of the trapezoids
- as vertices in a search graph



Trapezoidal Decomposition

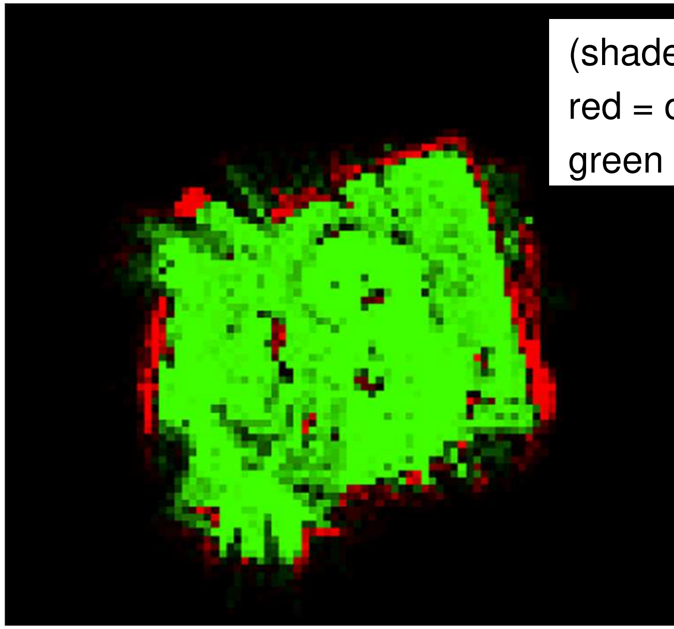
but this is not a proper roadmap!!!

⇒ need for local obstacle avoidance



Approximate: e.g. Regular Grid

- dominant form of (2D) map representation
- decomposition of space into regular cells (array)
  - occupied or free
  - potentially probabilistic
- graph (roadmap)
  - centers of free cells
  - adjacency = edge
  - 4 vs 8 connection



(shades of)
red = occupied
green = free