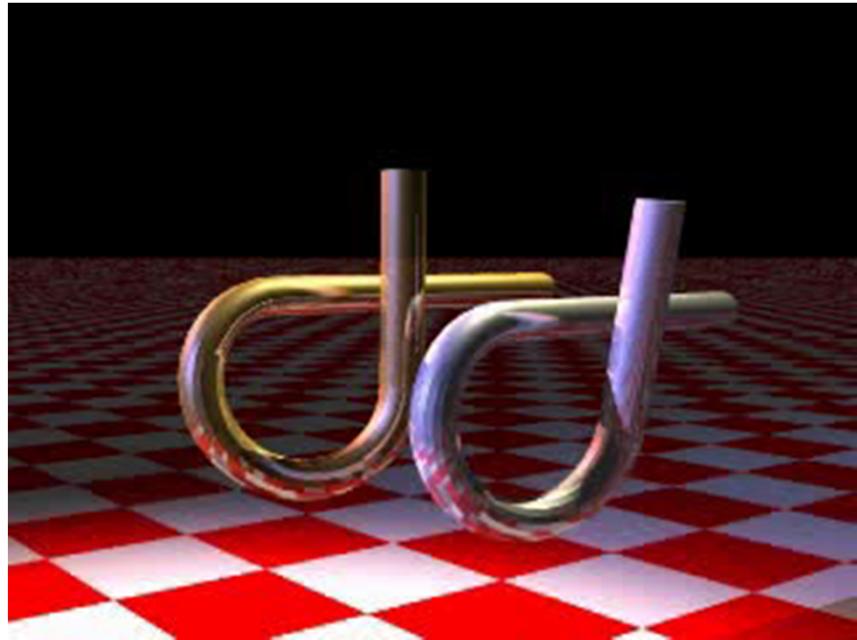


Path/Motion Planning

Movies/demos provided by James Kuffner and Howie Choset +
Examples from J.C. Latombe's and Steve LaValle's book
Excellent reference:
S. LaValle. Planning algorithms. Cambridge University Press. 2007.



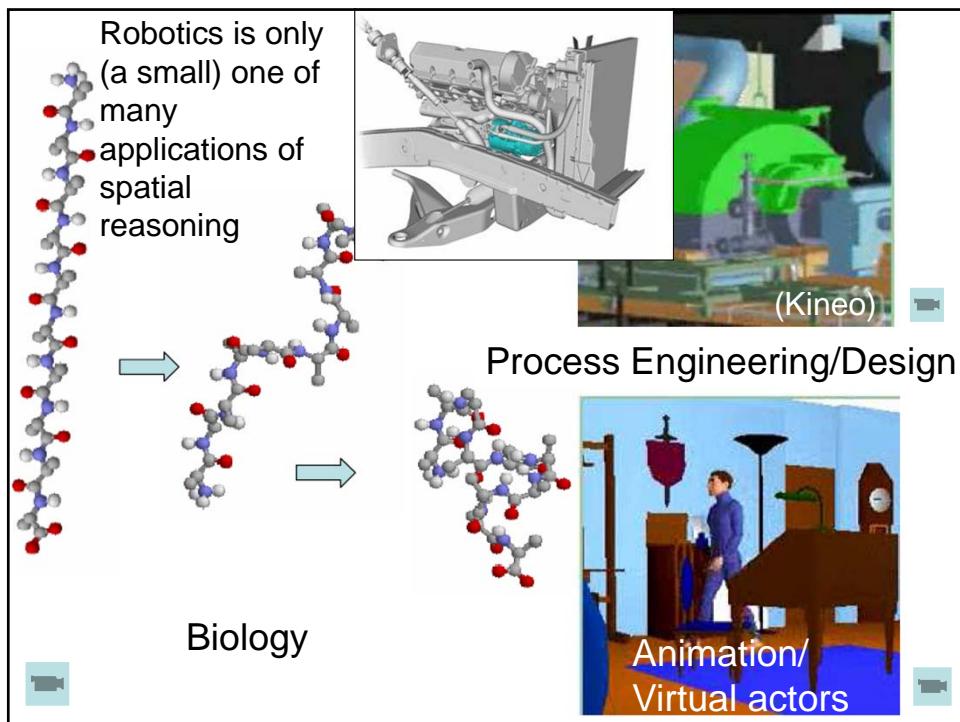


Example from James Kuffner



Path/Motion Planning

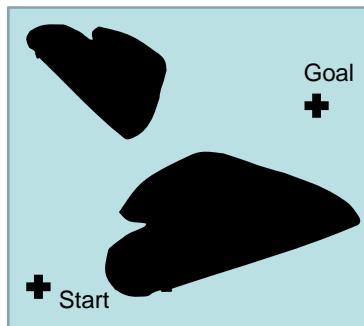
- Application of earlier search approaches (A*, stochastic search, etc.)
- Search in geometric structures
- ***Spatial reasoning***
- Challenges:
 - Continuous state space
 - Large dimensional space



Approach

- Convert the problem to a search problem through some space (e.g., using A*)
- What is the state space?
- How to represent it (continuous → discrete)?

Simple approach: State = position



Moving a point through space around obstacles

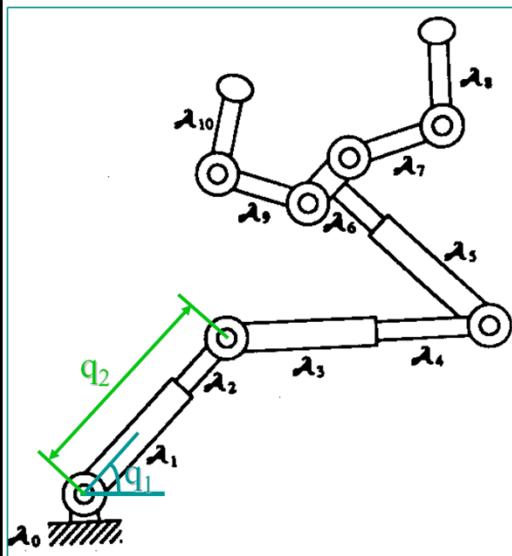
State space: (x, y)



Moving a piano through space around obstacles

State space: (x, y, θ)

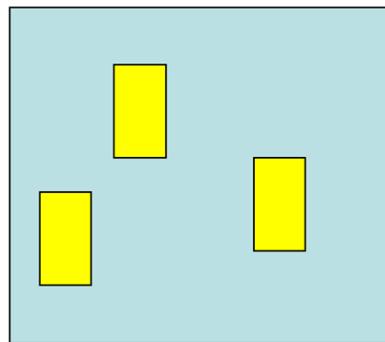
Degrees of Freedom



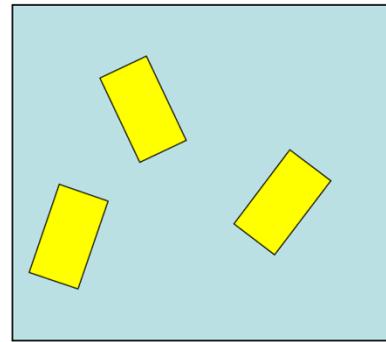
- The geometric configuration is defined by p degrees of freedom (DOF)
- Assuming p DOFs, the geometric configuration A is defined by p variables:

$$A(\mathbf{q}) \text{ with } \mathbf{q} = (q_1, \dots, q_p)$$

Examples

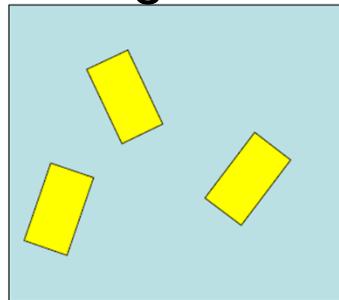


Allowed to move only
in x and y : 2DOF



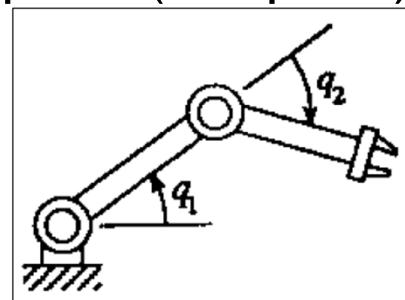
Allowed to move in x
and y and to rotate:
3DOF (x, y, θ)

Configuration Space (C-Space)



$$\mathbf{q} = (x, y, \theta)$$

$\mathcal{C} = \mathbb{R}^2 \times \text{set of 2-D rotations}$



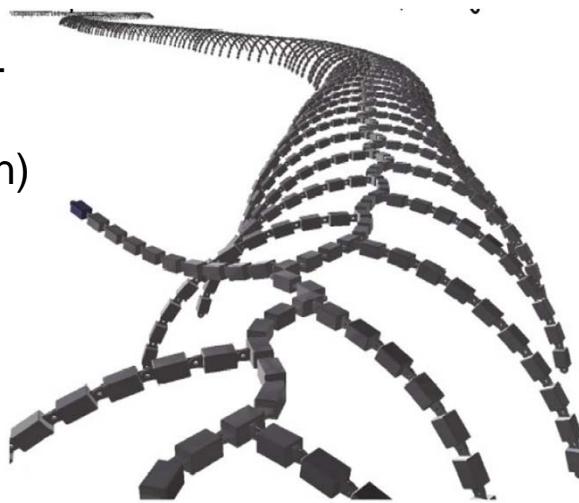
$$\mathbf{q} = (q_1, q_2)$$

$\mathcal{C} = \text{2-D rotations} \times \text{2-D rotations}$

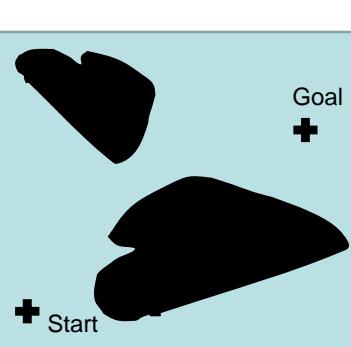
- Configuration space \mathcal{C} = set of values of \mathbf{q} corresponding to legal configurations
- Defines the set of possible parameters (the search space) and the set of allowed paths
- Assumptions:
 - We have defined a distance in C-space
 - We have defined a notion of “volume” in C-space (formally, a measure)

Large C-Space Dimension

Millipede-like robot
(S. Redon)



~13,000 DOFs !!!



Moving a point through space around obstacles

State space: (x, y)

A valid path is when the point is never inside an obstacle



Moving a piano through space around obstacles

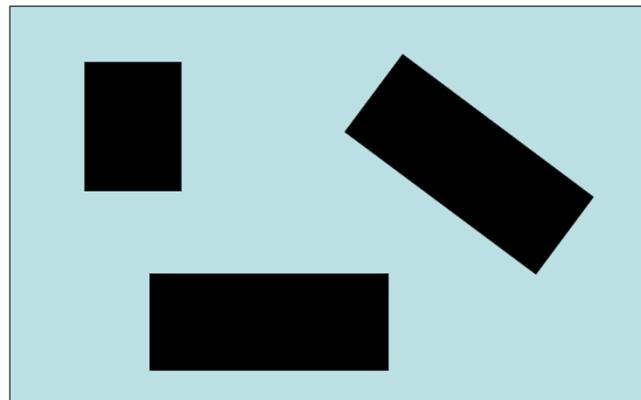
State space: (x, y, θ)

A valid path is when the *piano* never *intersect* the obstacles

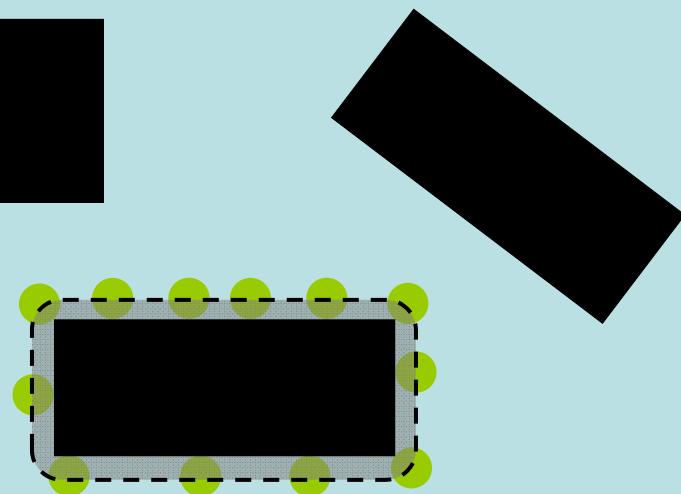
Sounds very expensive: We need to

1. Transform piano to its shape for each
2. Check for intersection with the obstacles

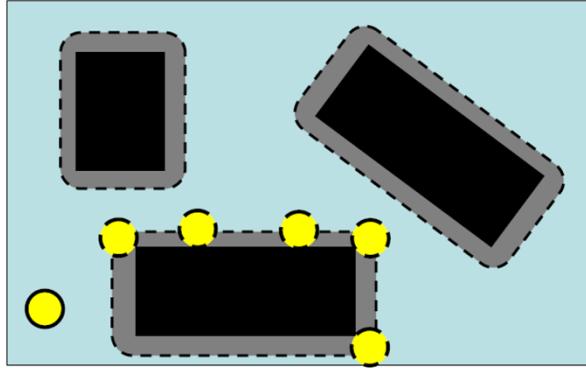
Free Space: Point



- $\mathcal{C}_{\text{free}} = \{\text{Set of parameters } \mathbf{q} \text{ for which } A(\mathbf{q}) \text{ does not intersect obstacles}\}$
- For a point robot in the 2-D plane: \mathbb{R}^2 minus the obstacle regions

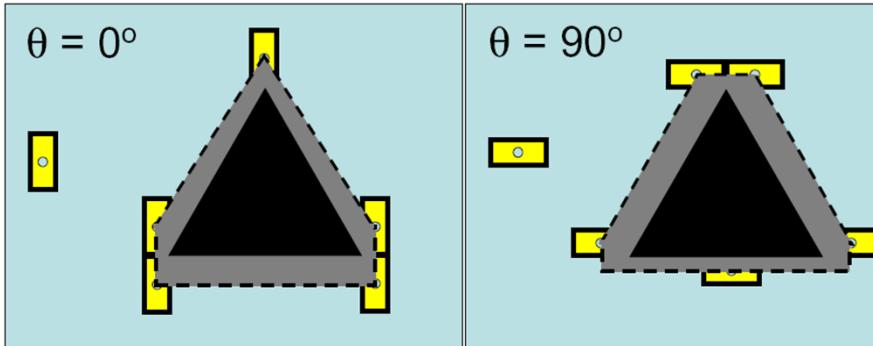


Free Space: Symmetric Robot



- We still have $\mathcal{C} = \mathbb{R}^2$ because orientation does not matter
- Reduce the problem to a point robot by expanding the obstacles by the radius of the robot

Free Space: Non-Symmetric Robot



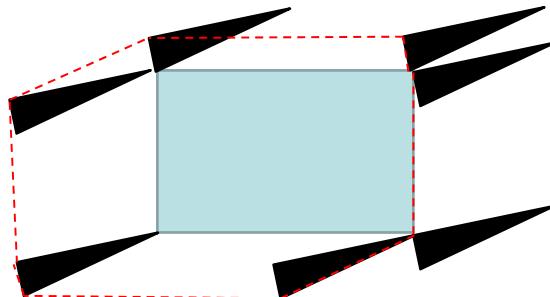
- The configuration space is now three-dimensional (x, y, θ)
- We need to apply a different obstacle expansion for each value of θ
- We still reduce the problem to a point robot by expanding the obstacles

Formal definition of the free space trick (simple case)

- Translation case: Minkowski difference

$$X \ominus Y = \{x - y; x \in X, y \in Y\}$$

$$C_{obs} = O \ominus A$$



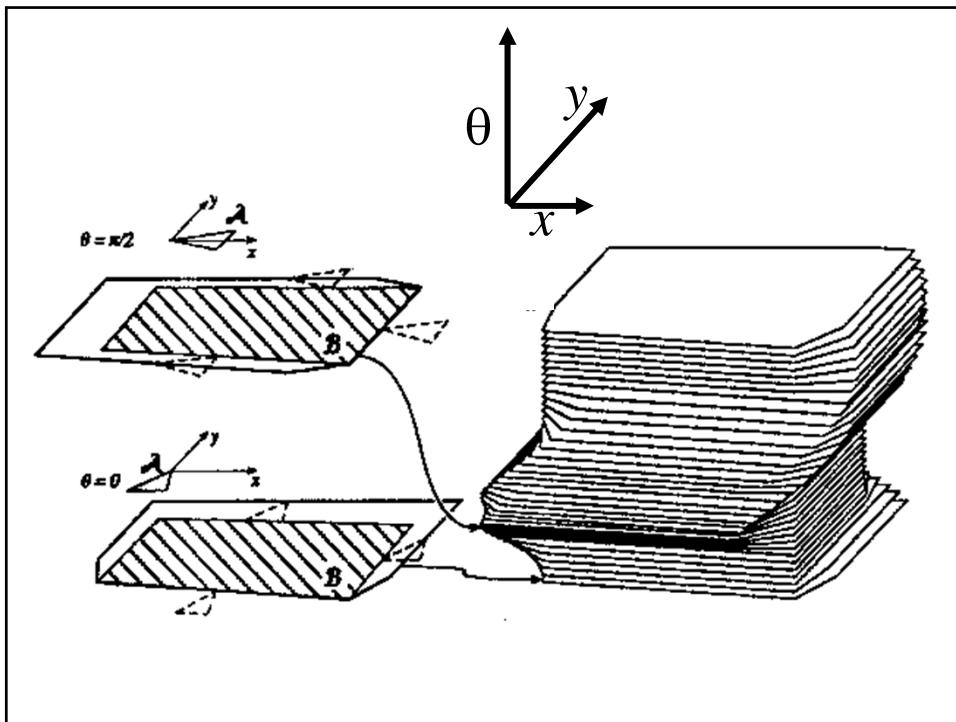
- All obstacles can be represented as unions of convex shapes
- Efficient algorithm for convex obstacles

- Property:

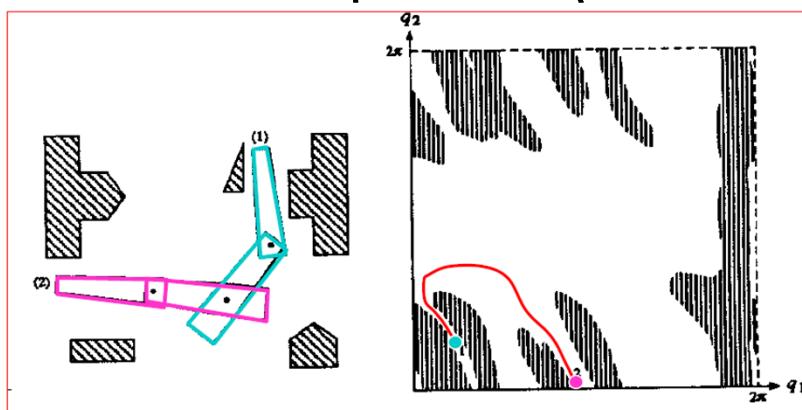
- Free path of object through C-O is equivalent to
- Free path of a point through

$$C_{free} = C - (O \ominus A)$$

We need to worry only about finding a path for a point



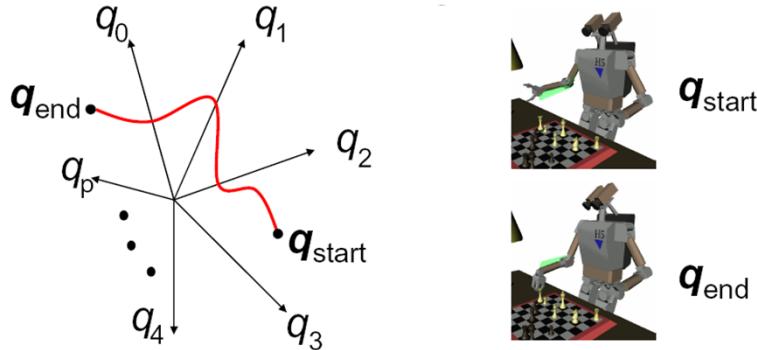
More Complex C-Spaces



- In all cases: The problem is reduced to finding the path of a **point** through configuration space by “expanding the obstacles”

Free space construct generalizes but is much more complex → Motivation for sampling techniques which do *not* require constructing C_{free} explicitly.

Path/Motion Planning Problem



- A = system with p degrees of freedom in 2-D or 3-D
- CB = Set of obstacles
- A configuration \mathbf{q} is legal if it does not cause to intersect the obstacles
- Given start and goal configurations ($\mathbf{q}_{\text{start}}$ and \mathbf{q}_{goal}), find a continuous sequence of legal configurations from $\mathbf{q}_{\text{start}}$ to \mathbf{q}_{goal} .
- Report failure if no path is found

Any Formal Guarantees? Generic Piano Movers Problem



- Formal Result (but not terribly useful for practical algorithms):
 - p : Dimension of \mathcal{C}
 - m : Number of polynomials describing $\mathcal{C}_{\text{free}}$
 - d : Max degree of the polynomials
- A path (if it exists) can be found in time ***exponential*** in p and ***polynomial*** in m and d

[From J. Canny. "The Complexity of Robot Motion Planning". MIT Ph.D. Dissertation, 1987]

Completeness

- Important definition:
- An algorithm is complete if:
 - If a path exists, it finds it in *finite* time
 - If a path does not exist, it returns in *finite* time
- Sound if:
 - Guaranteed to never cross an obstacle
- Less important:
 - Optimal if guaranteed to find the shortest path (if it exists)

Approaches

- Cell decomposition
- Roadmaps
- Sampling Techniques
(RRT, DRT, PRM,...)
- On-line algorithms
 D^* , ARA*,...

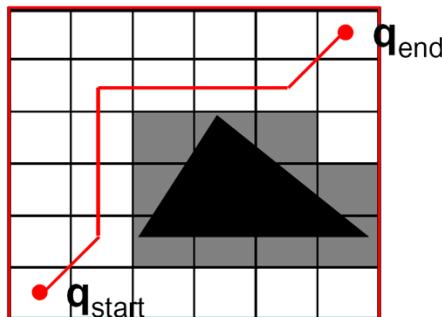
In all cases: Reduce the intractable problem in continuous C-space to a tractable problem in a discrete space → Use all of the techniques we know (A^* , stochastic search, etc.)

Approaches

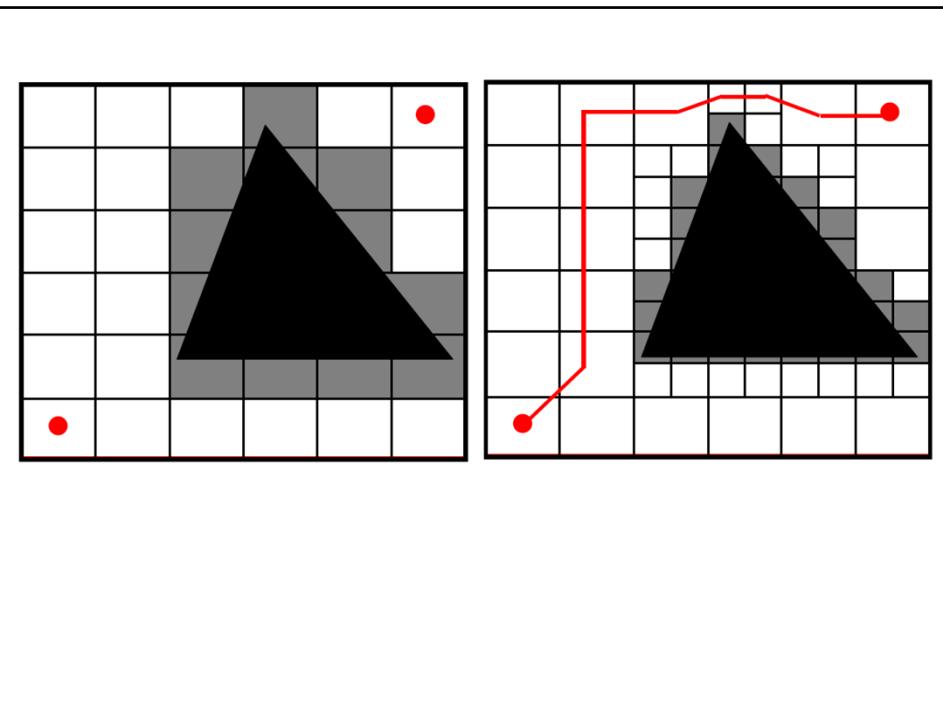
- Cell decomposition ←
- Roadmaps
- Sampling Techniques
(RRT, DRT, PRM,...)
- On-line algorithms
 D^* , ARA*,...

Decompose the space into cells so that any path inside a cell is obstacle free

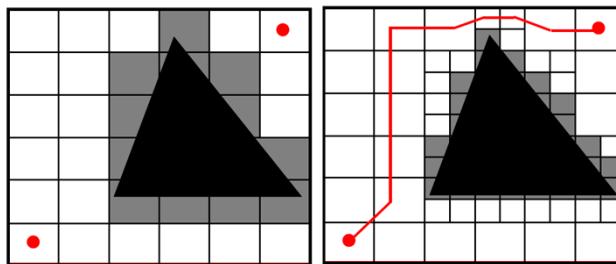
Approximate Cell Decomposition



- Define a discrete grid in C-Space
- Mark any cell of the grid that intersects \mathcal{C}_{obs} as blocked
- Find path through remaining cells by using (for example) A^* (e.g., use Euclidean distance as heuristic)
- Cannot be *complete* as described so far. Why?
- Is it optimal?

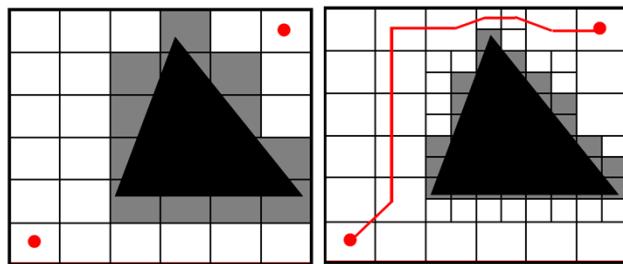


Approximate Cell Decomposition

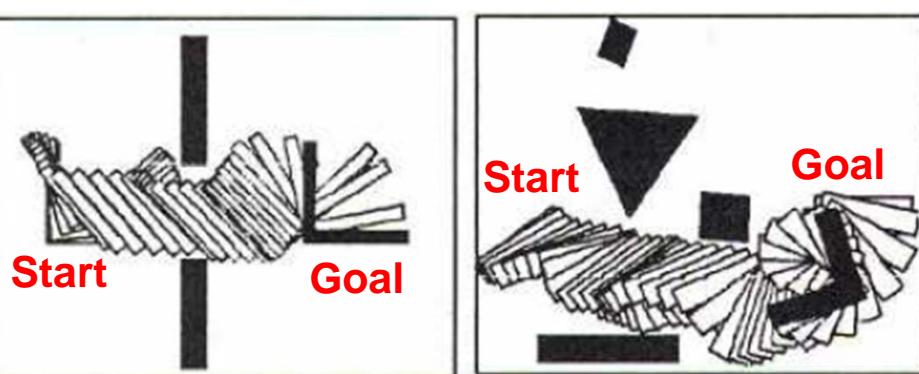


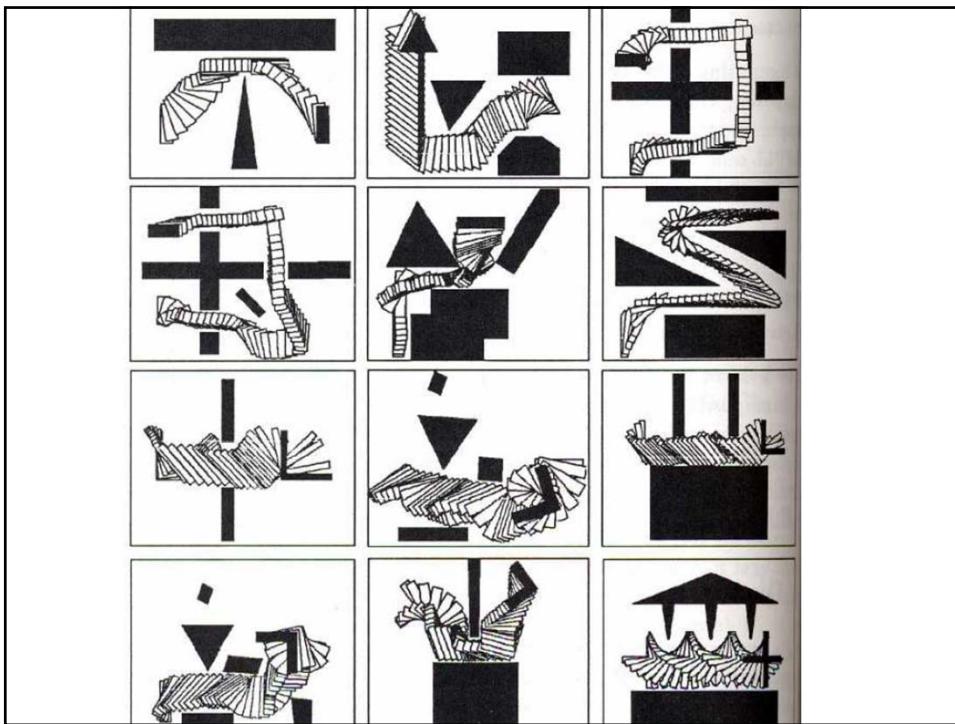
- Cannot find a path in this case even though one exists
- Solution:
- Distinguish between
 - Cells that are entirely contained in \mathcal{T}_{obs} (*FULL*) and
 - Cells that partially intersect \mathcal{T}_{obs} (*MIXED*)
- Try to find a path using the current set of cells
- If no path found:
 - Subdivide the *MIXED* cells and try again with the new set of cells

Is it complete now?



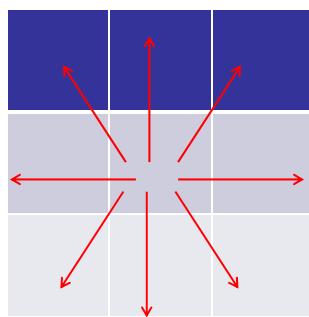
- An algorithm is *resolution complete* when:
 - If a path exists, it finds it in *finite* time
 - ~~If a path does not exist, it returns in *finite* time~~



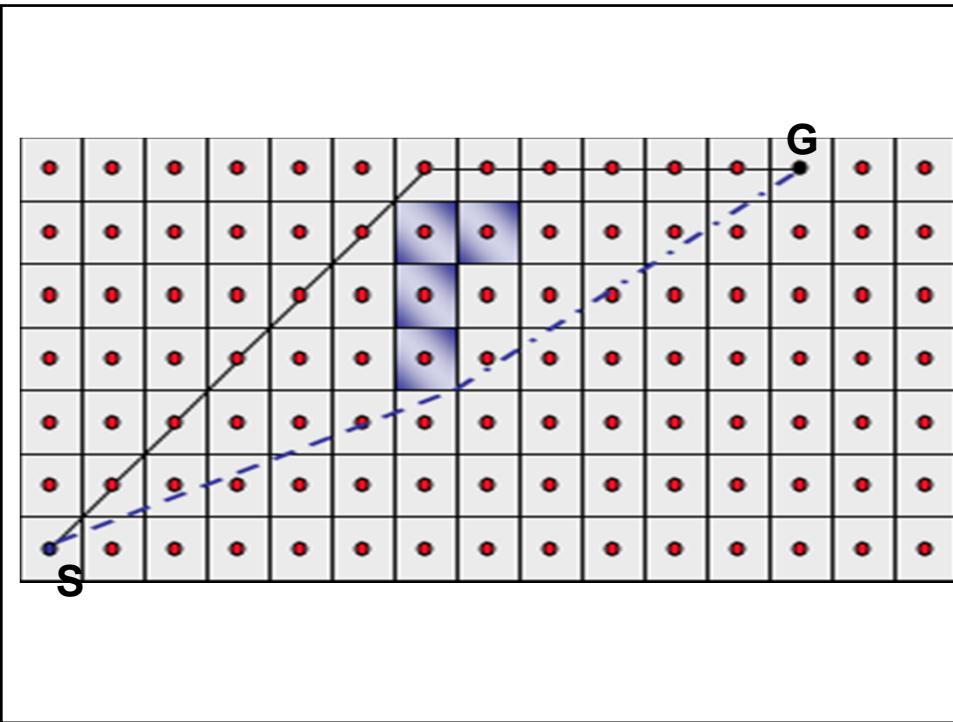


Optimality issues

- We took care of completeness
- How about optimality? Why is it not optimal?

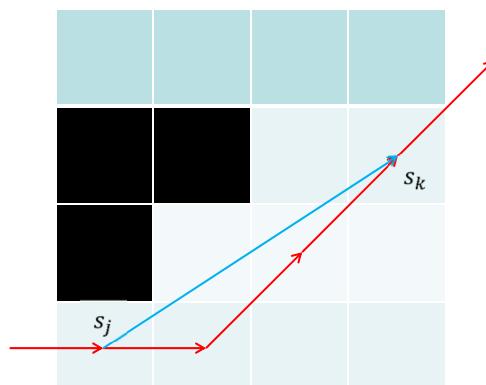


- To improve we'll need the notion of visibility: s is visible by s' iff the line between s and s' does not intersect obstacles
- Consecutive states on a sound path are visible from each other



Solution I

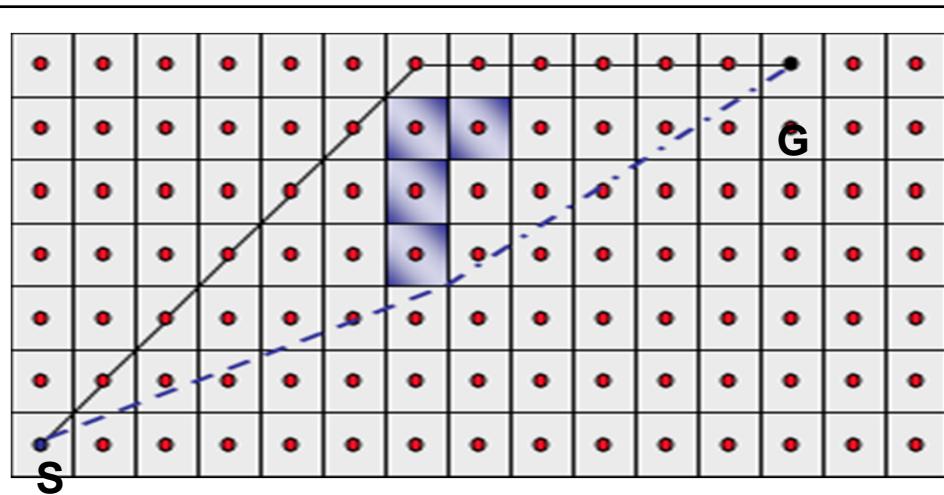
- Allow connection to further states than the neighbors on the grid
- Key observation:
 - If $(s_{start}, s_1, \dots, s_{goal})$ is a valid path
 - If s_j is *visible* from s_k
 - Then $(s_{start}, s_1, \dots, s_{j-1}, s_j, s_k, s_{k+1}, s_{goal})$ is a valid path



Solution I

- A* post-processing (A* smoothing)
- Iterate starting at s_{goal}
- If parent(parent(current state)) is visible from current state
 - Delete parent(current state)
- Else
 - current state \leftarrow parent(current state)

Example: A. Botea, M. Muller, J. Schaeffer. Near optimal hierarchical path-finding. Journal of game development. 2004.



- Cannot be smoothed!
- Can we do something different *while* searching

Solution II

- Allow parents that are non-neighbors in the grid (but visible) to be used *during search*



Standard A*

$$g(s') = g(s) + c(s, s')$$

Insert s' with estimate

$$g(s') = g(s) + c(s, s') + h(s')$$

Theta*

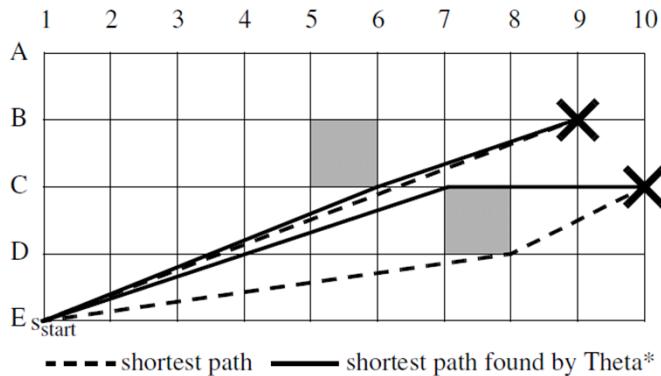
IF $\text{parent}(s)$ is *visible* from s'

$$g(s') = g(\text{parent}(s)) + c(\text{parent}(s), s') + h(s')$$

Nash, Daniel, Koenig, Felner. Theta*: Any-Angle Path Planning on Grids. AAAI 2007.

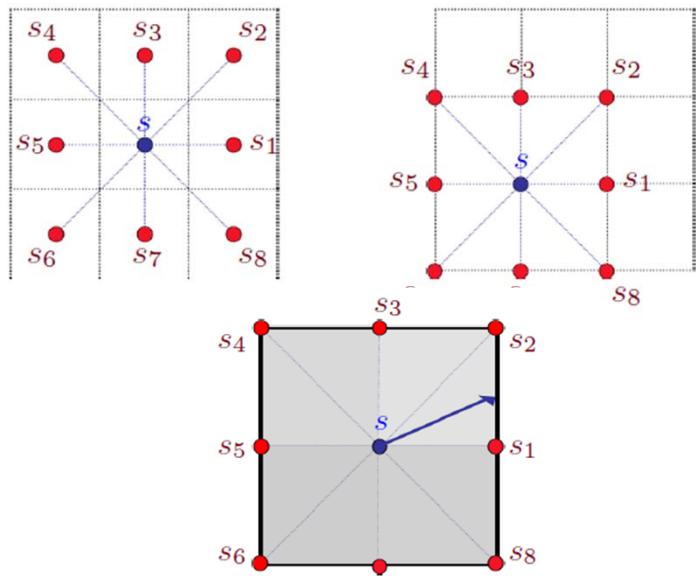
Solution II

- Why does it work? Why does it give a lower cost path?
- Note: This approximates searching through the entire visibility graph of the grid nodes (too expensive to be practical)



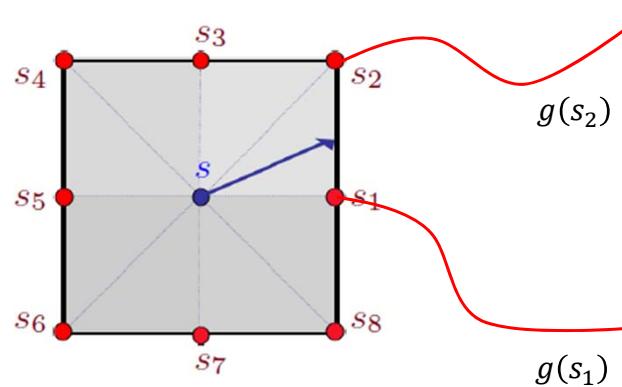
Nash, Daniel, Koenig, Felner. Theta*: Any-Angle Path Planning on Grids. AAAI 2007.

Solution III



- Idea: Allow crossing cell edges

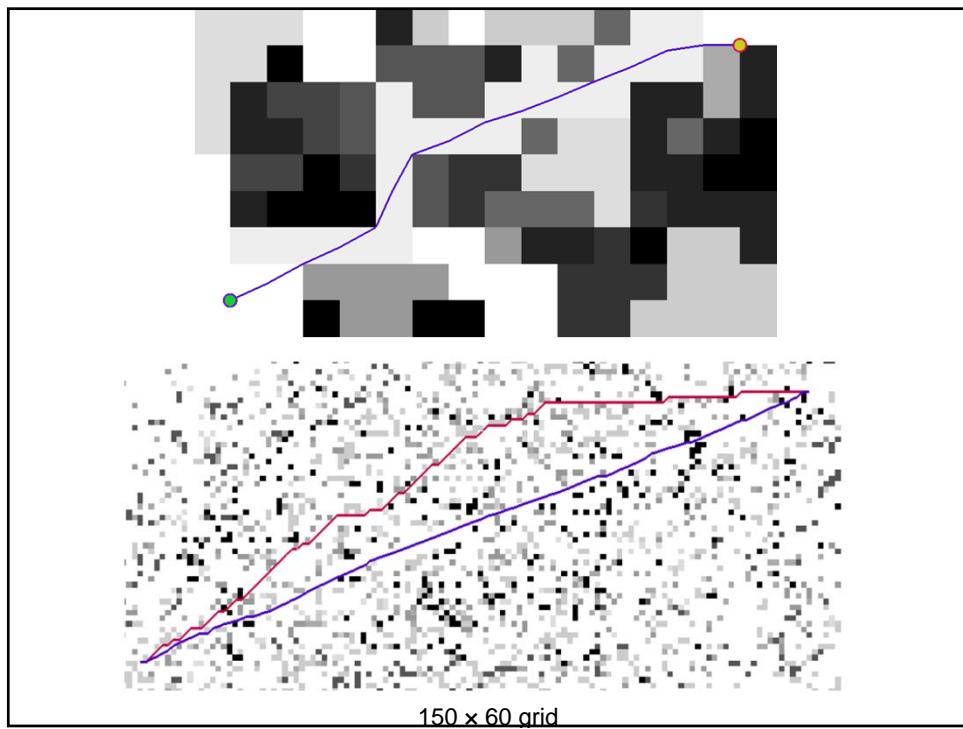
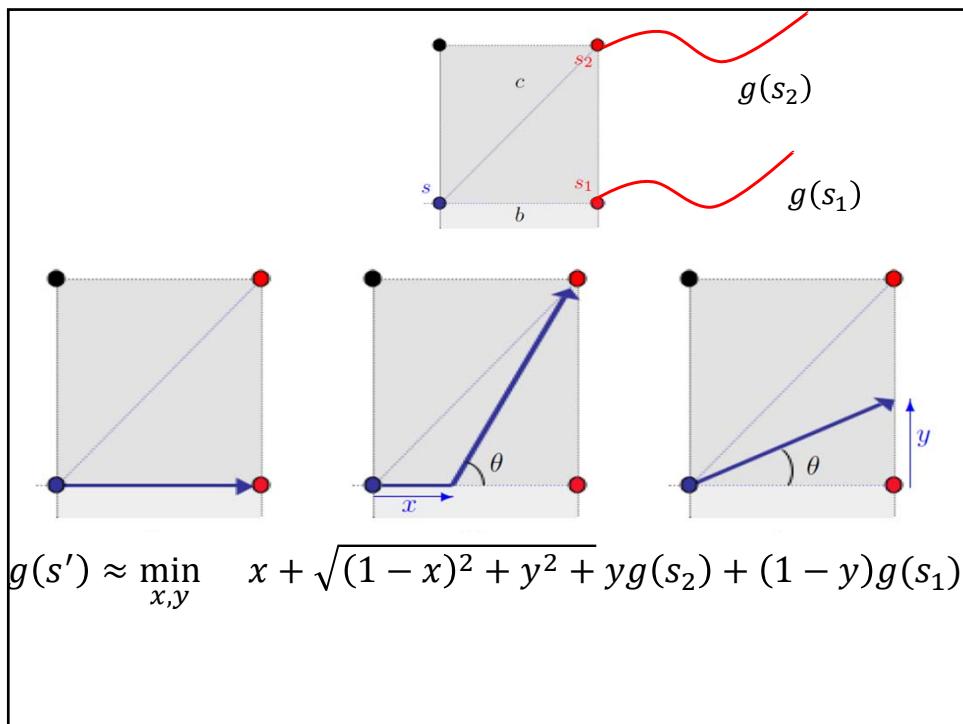
Solution III



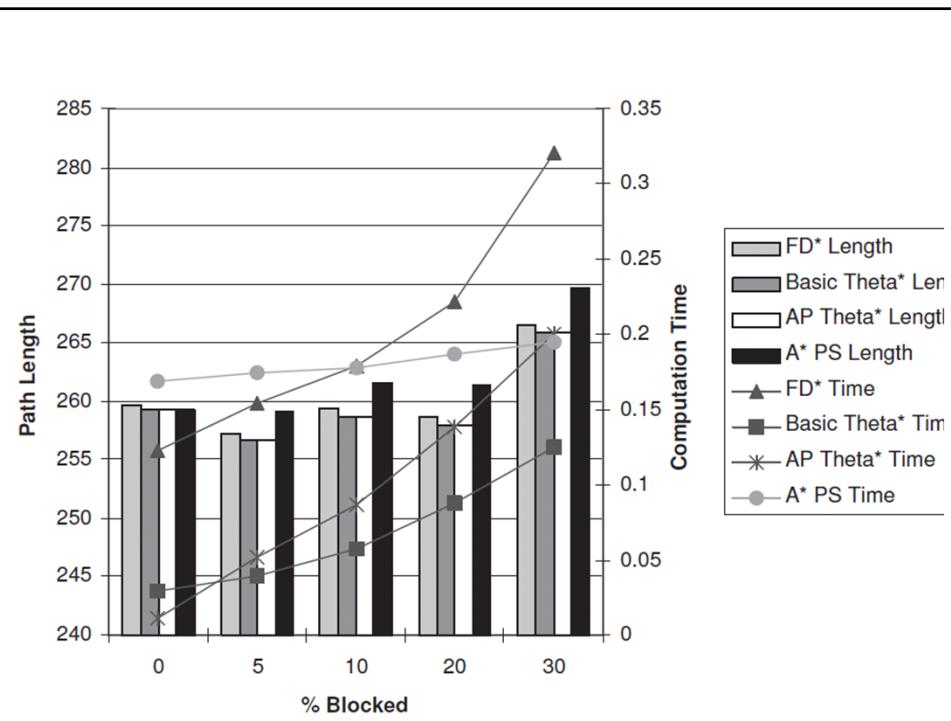
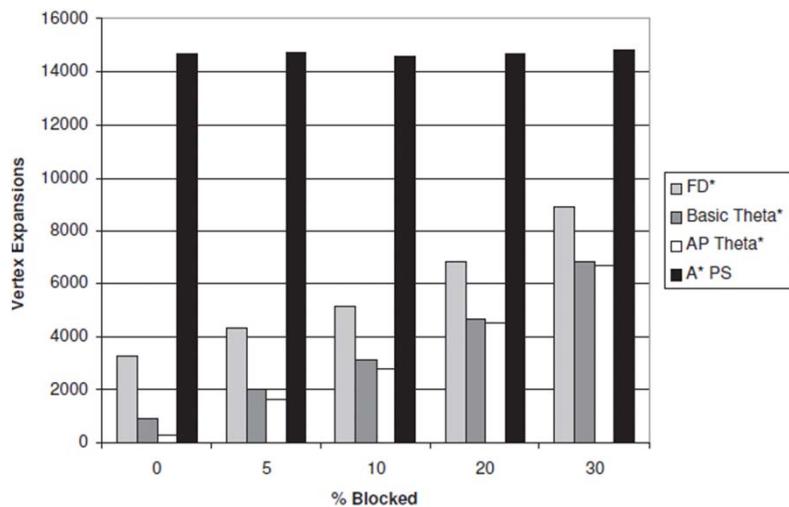
$$g(s) = \min_{s'}(g(s') + c(s, s'))$$

Intractable: need to search over all of the s'

Approximation: $g(s') \approx yg(s_2) + (1 - y)g(s_1)$



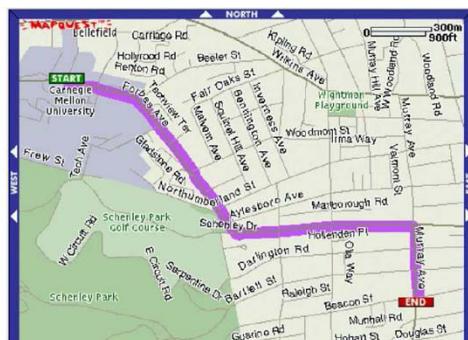
Does it make a difference?



Approaches

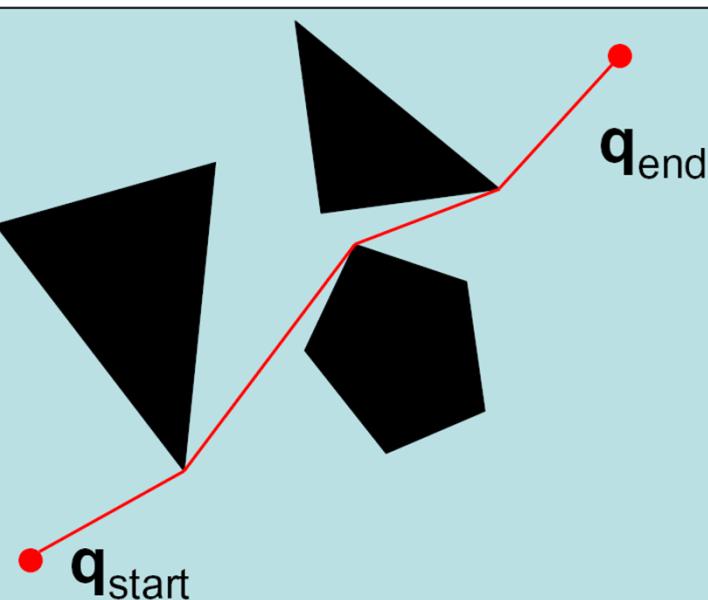
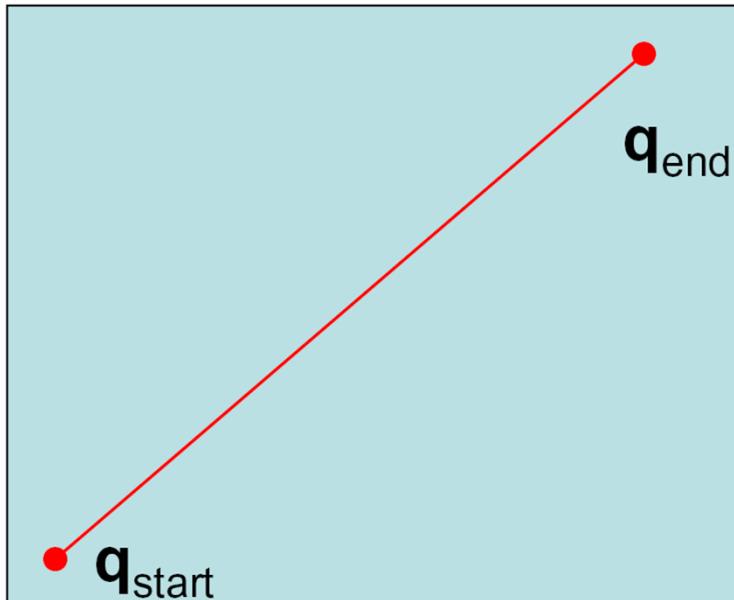
- Cell decomposition
- Roadmaps
- Sampling Techniques
(RRT, DRT, PRM,...)
- On-line algorithms
 D^* , ARA*,...

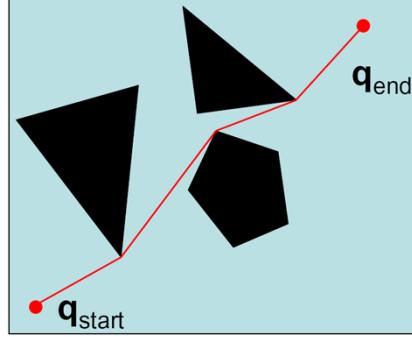
Roadmaps



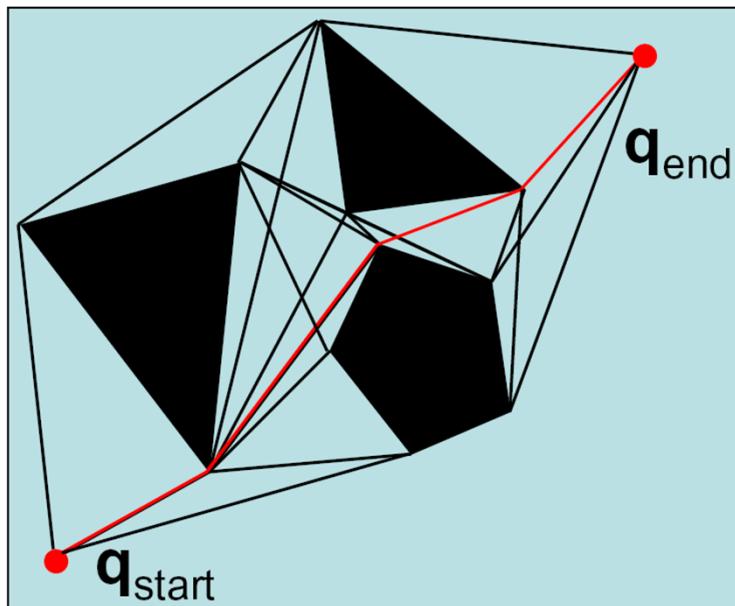
- General idea:
 - Avoid searching the entire space
 - Pre-compute a (hopefully small) graph (the roadmap) such that staying on the “roads” is guaranteed to avoid the obstacles
 - Find a path between q_{start} and q_{goal} by using the roadmap

First “obvious” approach (but not practical)

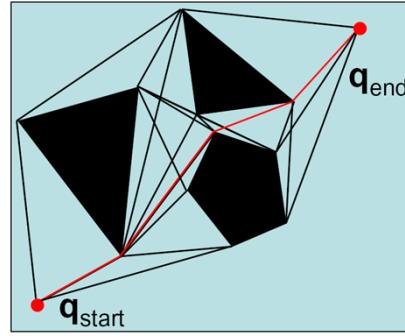




- Assuming polygonal obstacles: It looks like the shortest path is a sequence of straight lines joining the vertices of the obstacles.
- This is always true → Idea:
 - Link the vertices into a graph
 - Search (e.g., A*) through that graph

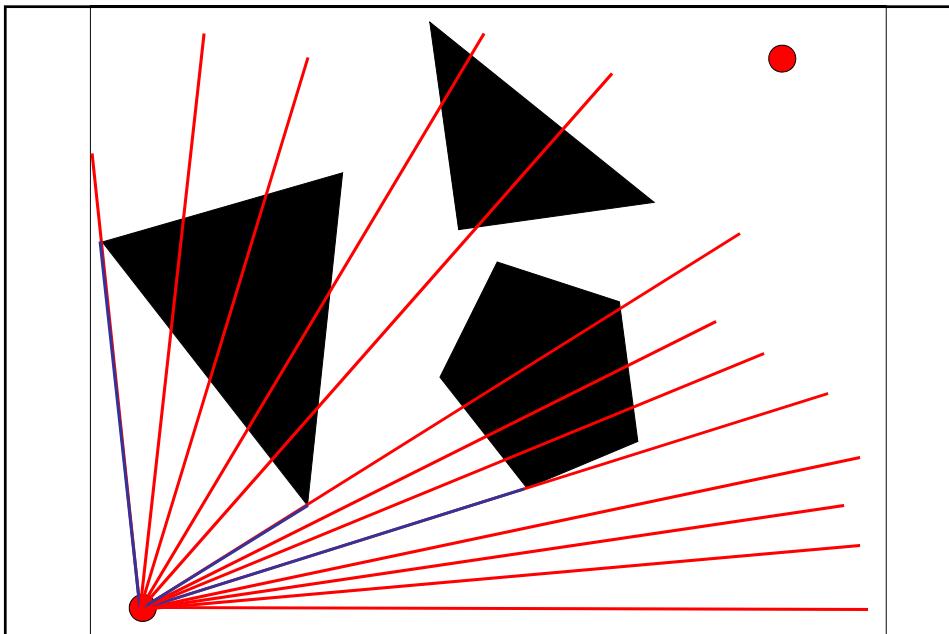


Visibility Graphs (Lozano-Perez et al.)

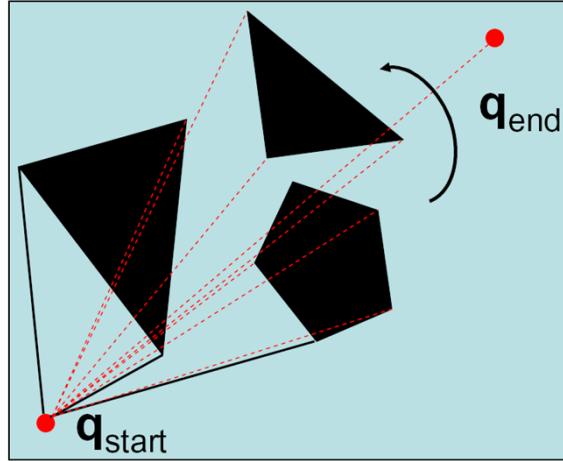


- Visibility graph G = set of unblocked lines between vertices of the obstacles + q_{start} and q_{goal}
- A node P is linked to a node P' if P' is visible from P
- Solution = Shortest path in the visibility graph

Note important concept for later: *visibility*

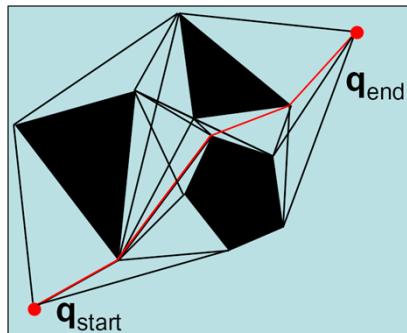


Construction: Sweep Algorithm



- Sweep a line originating at each vertex
- Record those lines that end at visible vertices

Complexity

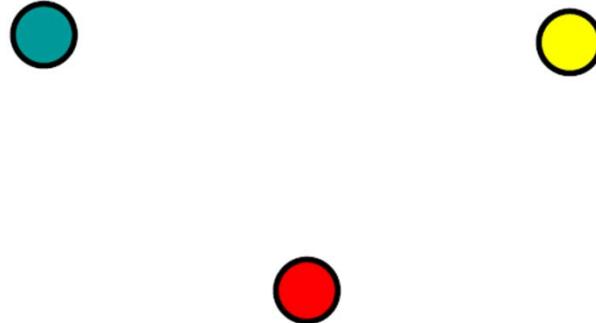


- N = total number of vertices of the obstacle polygons
- Naïve: $O(N^3)$
- Sweep: $O(N^2 \log N)$
- Optimal: $O(N^2)$

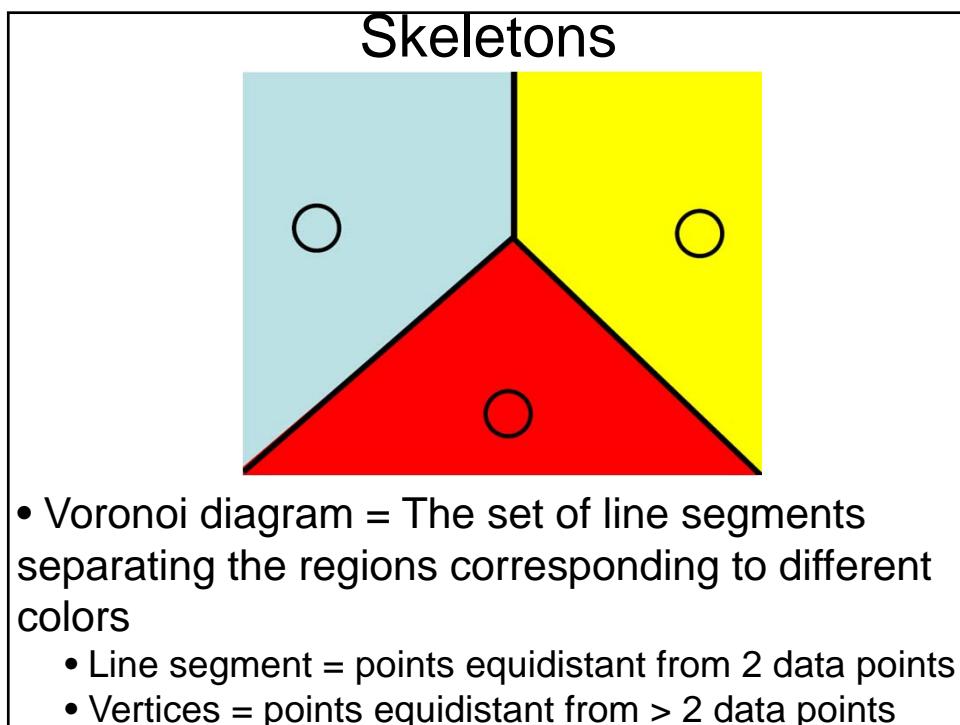
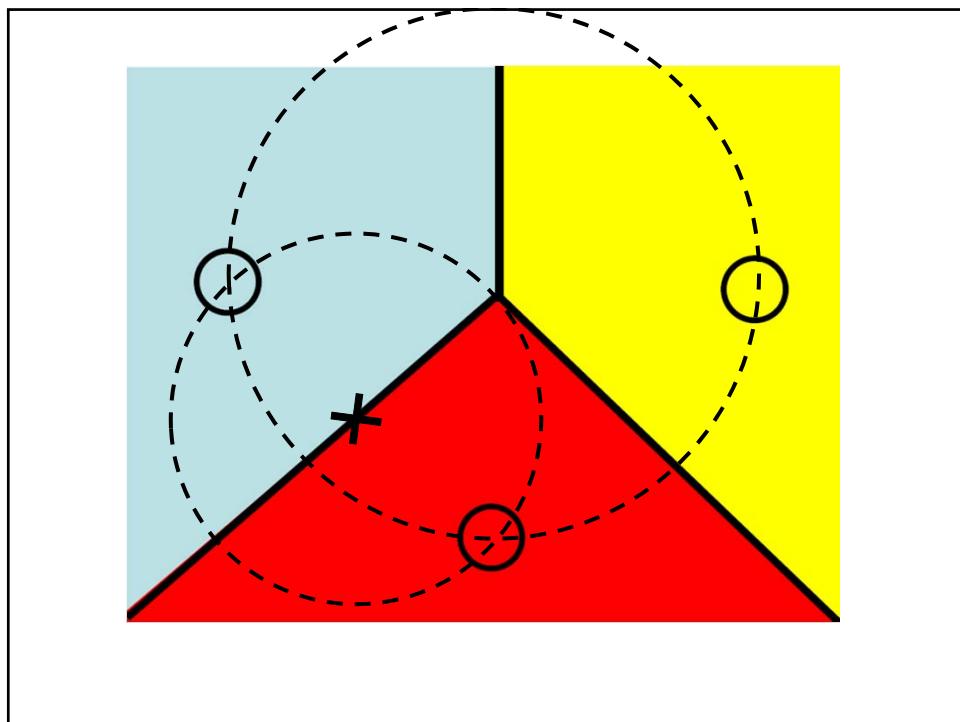
Why not practical?

- Shortest path but:
 - Tries to stay as close as possible to obstacles
 - Any execution error will lead to a collision
 - Complicated in >> 2 dimensions
- We may not care about strict optimality so long as we find a safe path. Staying away from obstacles is more important than finding the shortest path
- Need to define other types of “roadmaps”

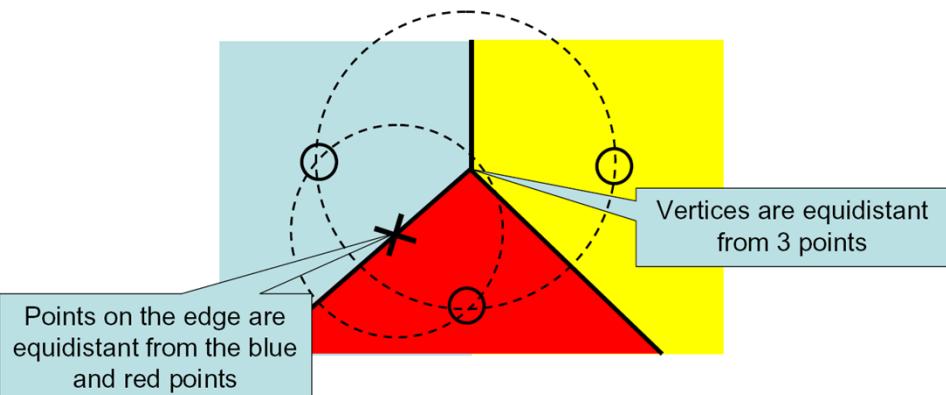
Skeletons



- Given a set of data points in the plane:
 - Color the entire plane such that the color of any point in the plane is the same as the color of its nearest neighbor

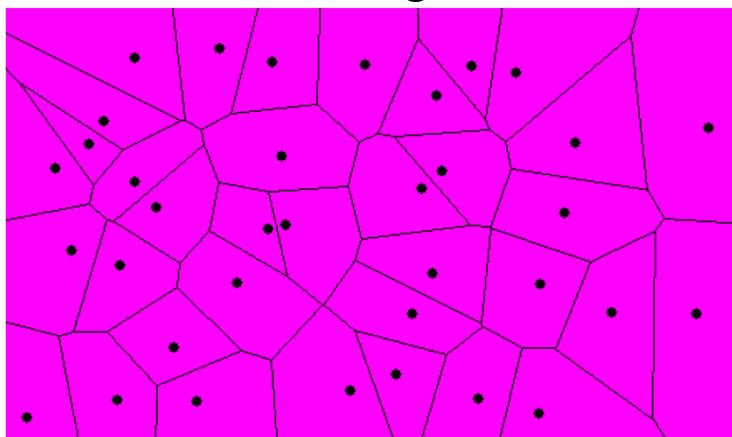


Skeletons



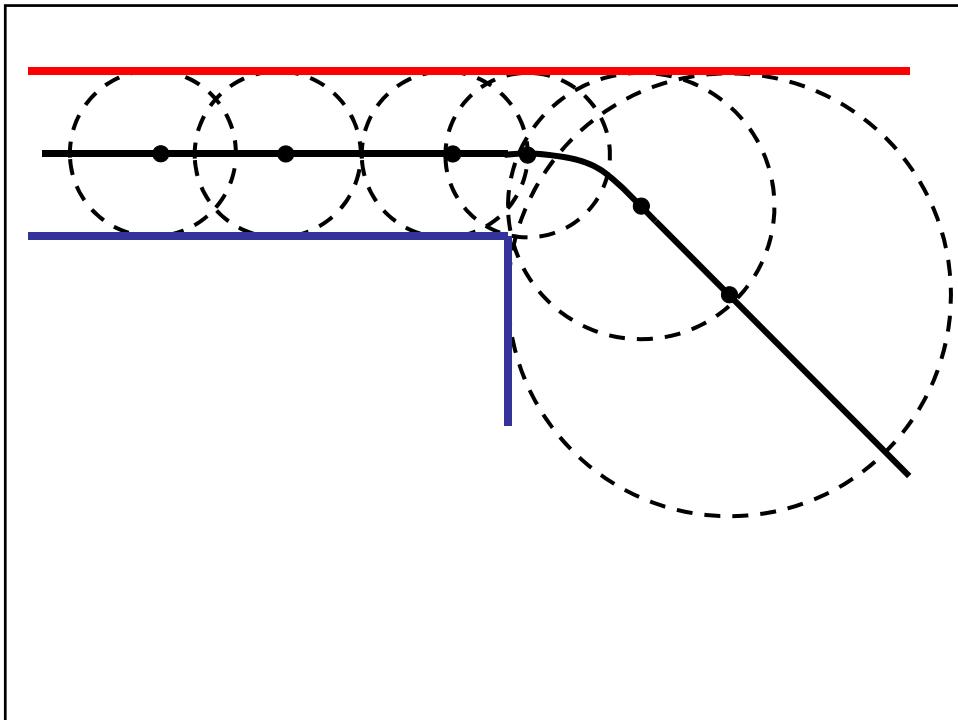
- Voronoi diagram = The set of line segments separating the regions corresponding to different colors
 - Line segment = points equidistant from 2 data points
 - Vertices = points equidistant from > 2 data points

Voronoi Diagrams

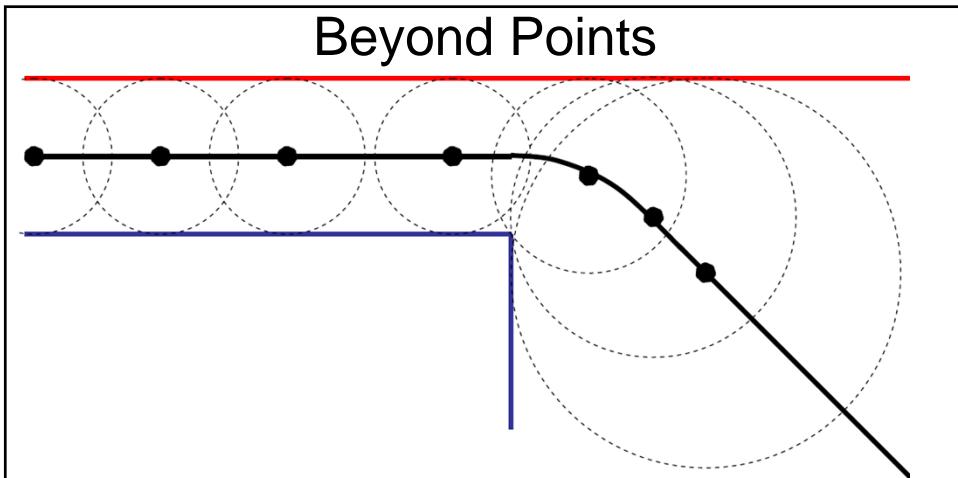


- Complexity (in the plane):
- $O(N \log N)$ time
- $O(N)$ space

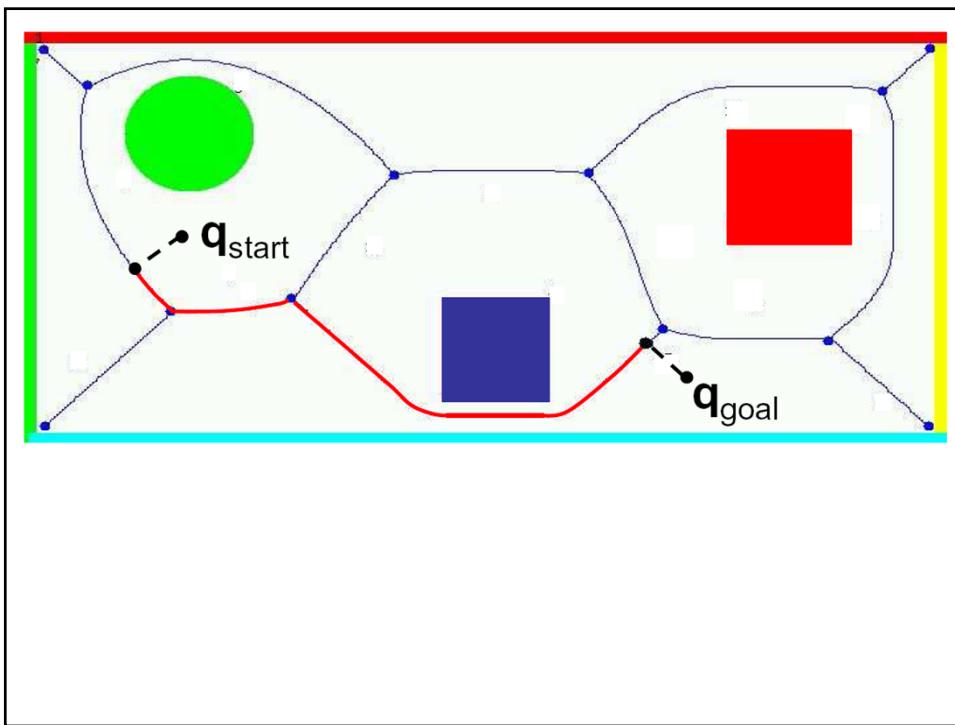
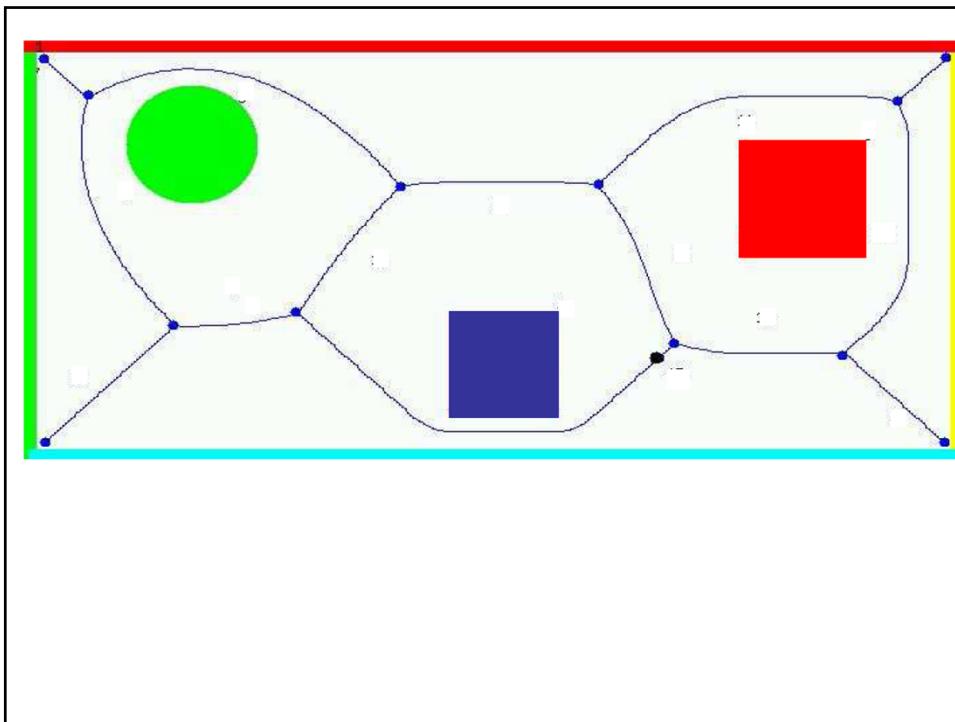
(See for example <http://www.cs.cornell.edu/Info/People/chew/Delaunay.html> for an interactive demo)



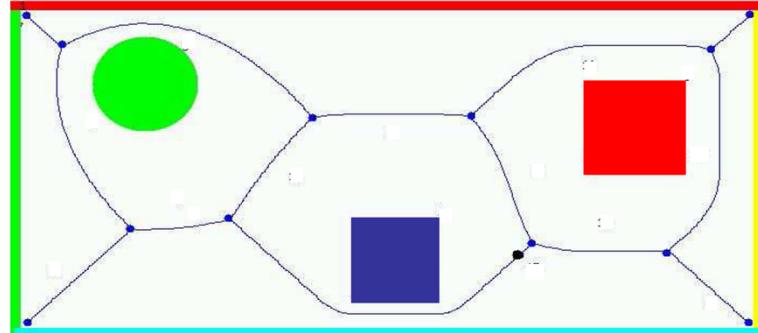
Beyond Points



- Edges are combinations of straight line segments and segments of quadratic curves
- Straight edges: Points equidistant from 2 lines
- Curved edges: Points equidistant from one corner and one line

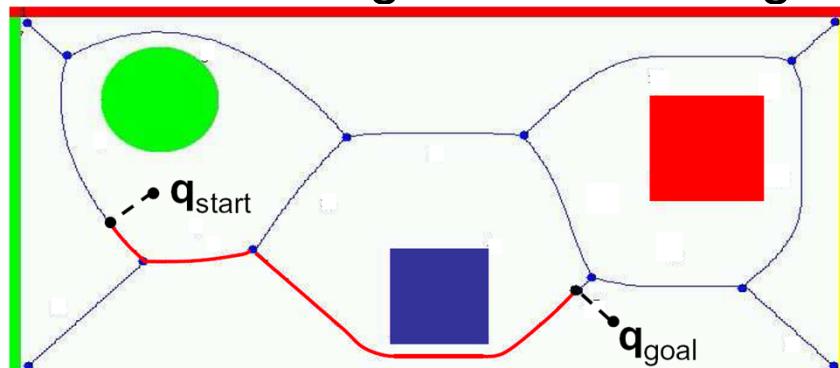


Voronoi Diagrams (Polygons)



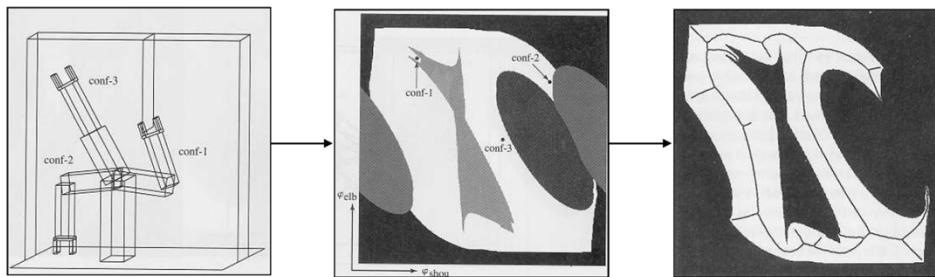
- Key property: The points on the edges of the skeleton are the *furthest* from the obstacles
- Idea: Construct a path between q_{start} and q_{goal} by following edges on the skeleton
- (Use the skeleton as a roadmap)

Voronoi Diagrams: Planning



- Find the point q^*_{start} of the graph closest to q_{start}
- Find the point q^*_{goal} of the graph closest to q_{goal}
- Compute shortest path from q^*_{start} to q^*_{goal} on the graph

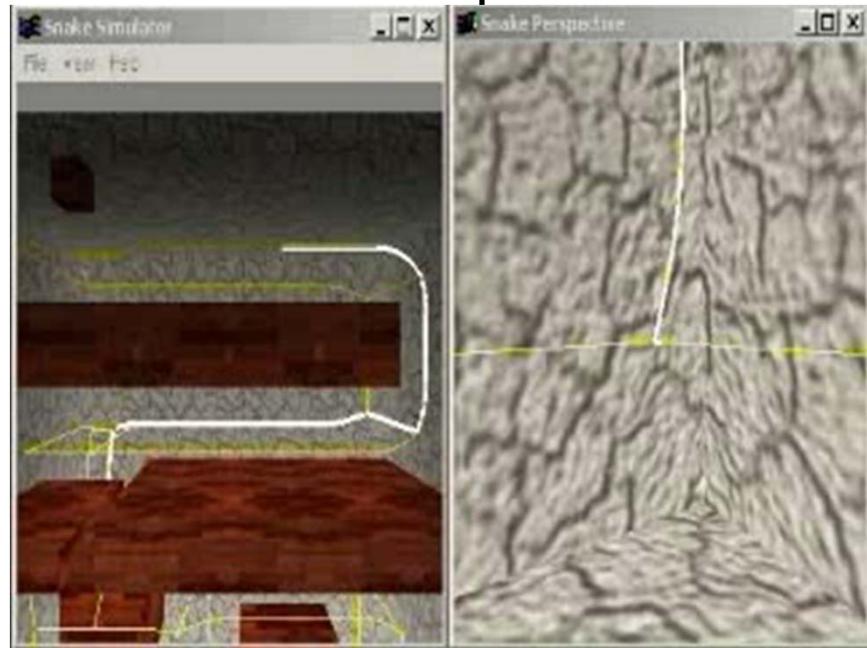
Example

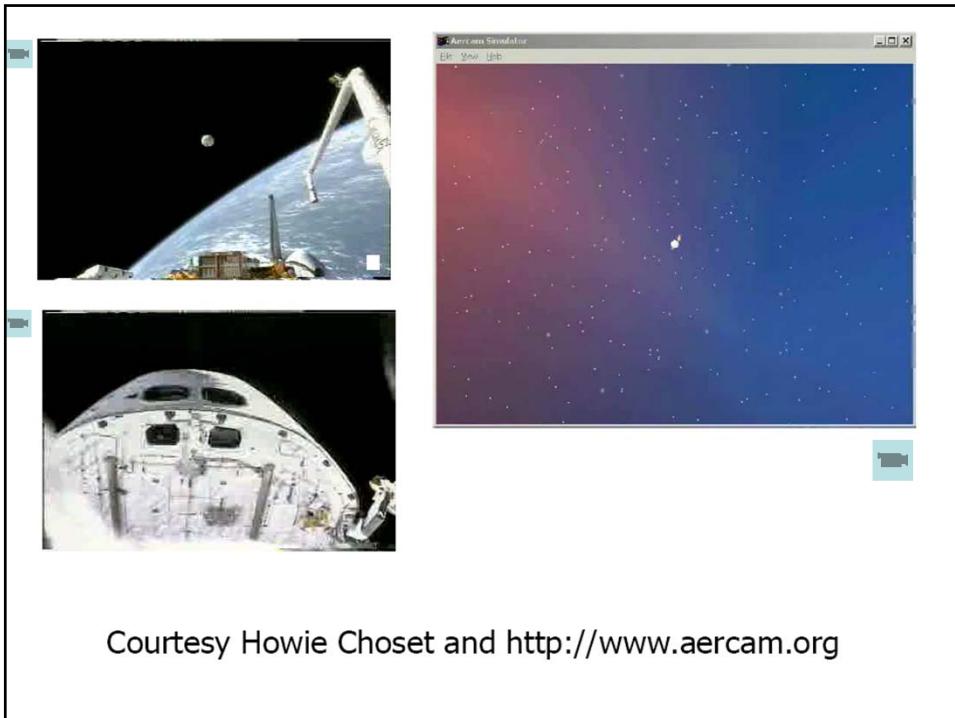


the example above is borrowed from "AI: A Modern Approach" by S. Russell & P. Norvig

- Complete? Optimal?

Example





Weaknesses

- Difficult to compute in higher dimensions or non-polygonal worlds
- Approximate algorithms exist
- Use of skeleton is not necessarily the best heuristic (“stay away from obstacles”) Can lead to paths that are much too conservative
- Can be unstable → Small changes in obstacle configuration can lead to large changes in the diagram

Approximate Cell Decomposition: Limitations

- Good:
 - Limited assumptions on obstacle configuration
 - Approach used in practice
 - Find obvious solutions quickly
- Bad:
 - No clear notion of optimality (“best” path)
 - Trade-off completeness/computation
 - Still difficult to use in high dimensions (need to compute C_{free} explicitly!)