Motion Planning

Jyh-Ming Lien

Department of Computer Science
George Mason University

Based on many people's lecture notes

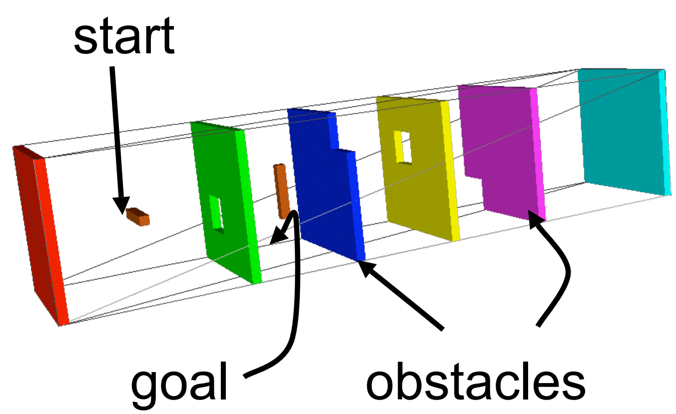
Seth Hutchinson at the University of Illinois at Urbana-Champaign, Leo Joskowicz at Hebrew University, Jean-Claude Latombe at Stanford University, Nancy Amato at Texas A&M University, Burchan Bayazit at Washington University in St. Louis

Motion Planning

in continuous spaces

(Basic) Motion Planning
(in a nutshell):

Given a *movable object*, find a *sequence of valid configurations* that moves the object from the start to the goal.



Main Steps In Motion Planning

Workspace

↓

Configuration space

↓

Discretization

↓

Search

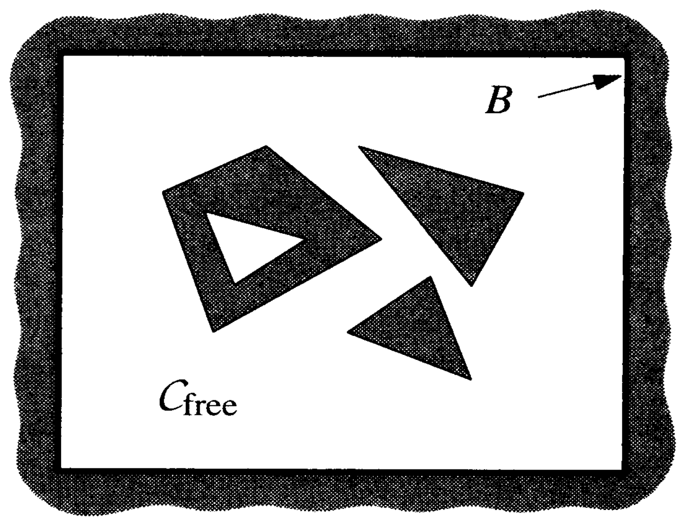
↓

Path or no solution

Classical Motion Planning

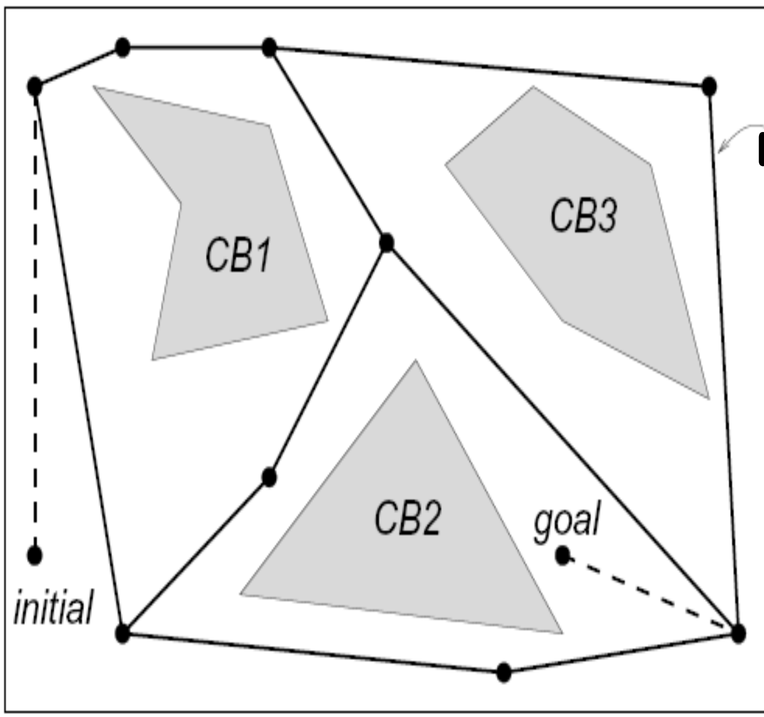
- Given a point robot and a workspace described by polygons

- **Roadmap methods**
  - Visibility graph
  - Cell decomposition
  - Retraction

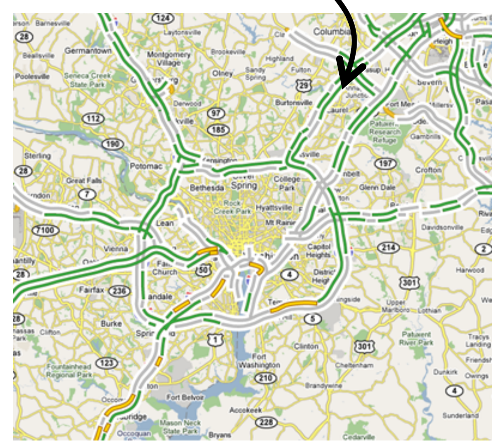


Roadmap Methods

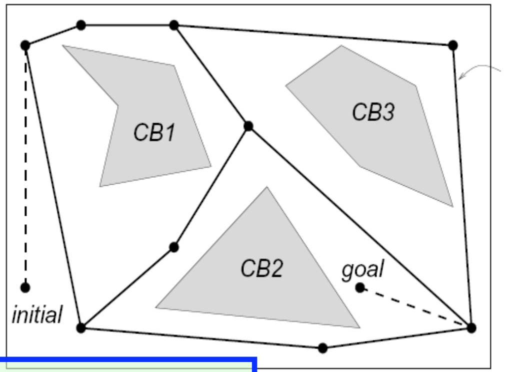
Capture the connectivity of C_{free} with a roadmap (graph or network) of one-dimensional curves



roadmap



Roadmap Methods



Path Planning with a Roadmap

Input: configurations q_{init} and q_{goal} , and B

Output: a path in C_{free} connecting q_{init} and q_{goal}

1. Build a roadmap in C_{free} (preprocessing)
   - roadmap nodes are free configurations (or semi-free)
   - two nodes connected by edge if can (easily) move between them

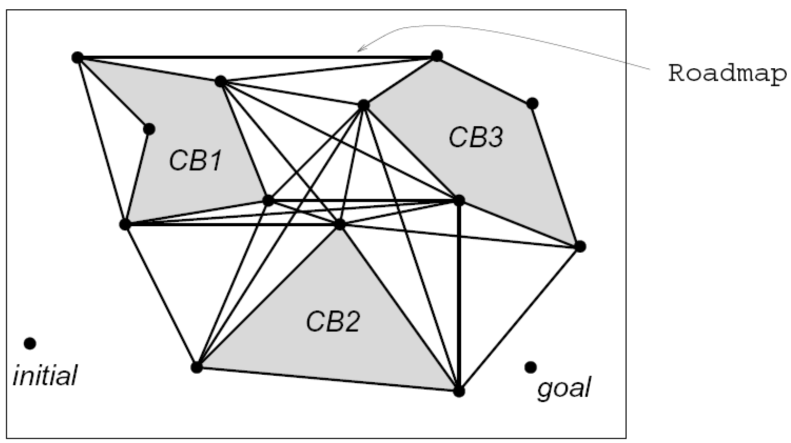
2. Connect q_{init} and q_{goal} to roadmap nodes v_{init} and v_{goal}

3. Find a path in the roadmap between v_{init} and v_{goal}
   - directly gives a path in C_{free}

difficult part

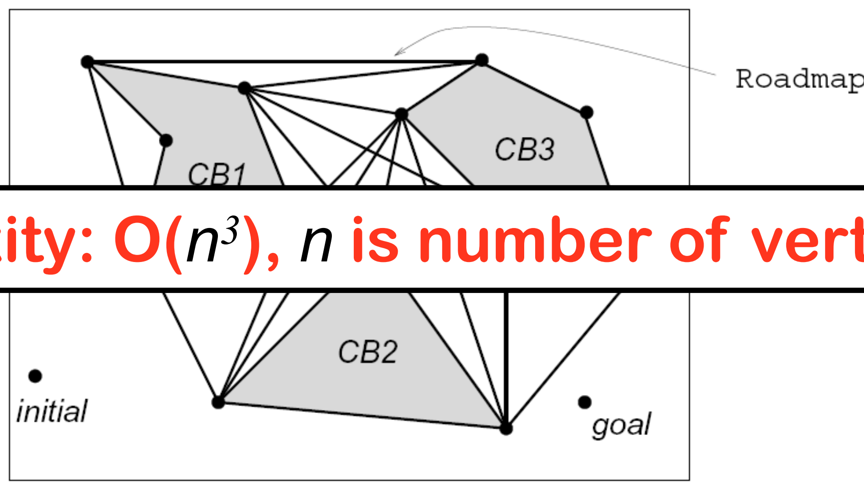
Visibility Graph

- A visibility graph of C-space for a given C-obstacle is an undirected graph G where
  - nodes in G correspond to vertices of C-obstacle
  - nodes connected by edge in G if
    - they are connected by an edge in C-obstacle, or
    - the straight line segment connecting them lies entirely in *Cfree*
  - (could add q_{init} and q_{goal} as roadmap nodes)



Visibility Graph

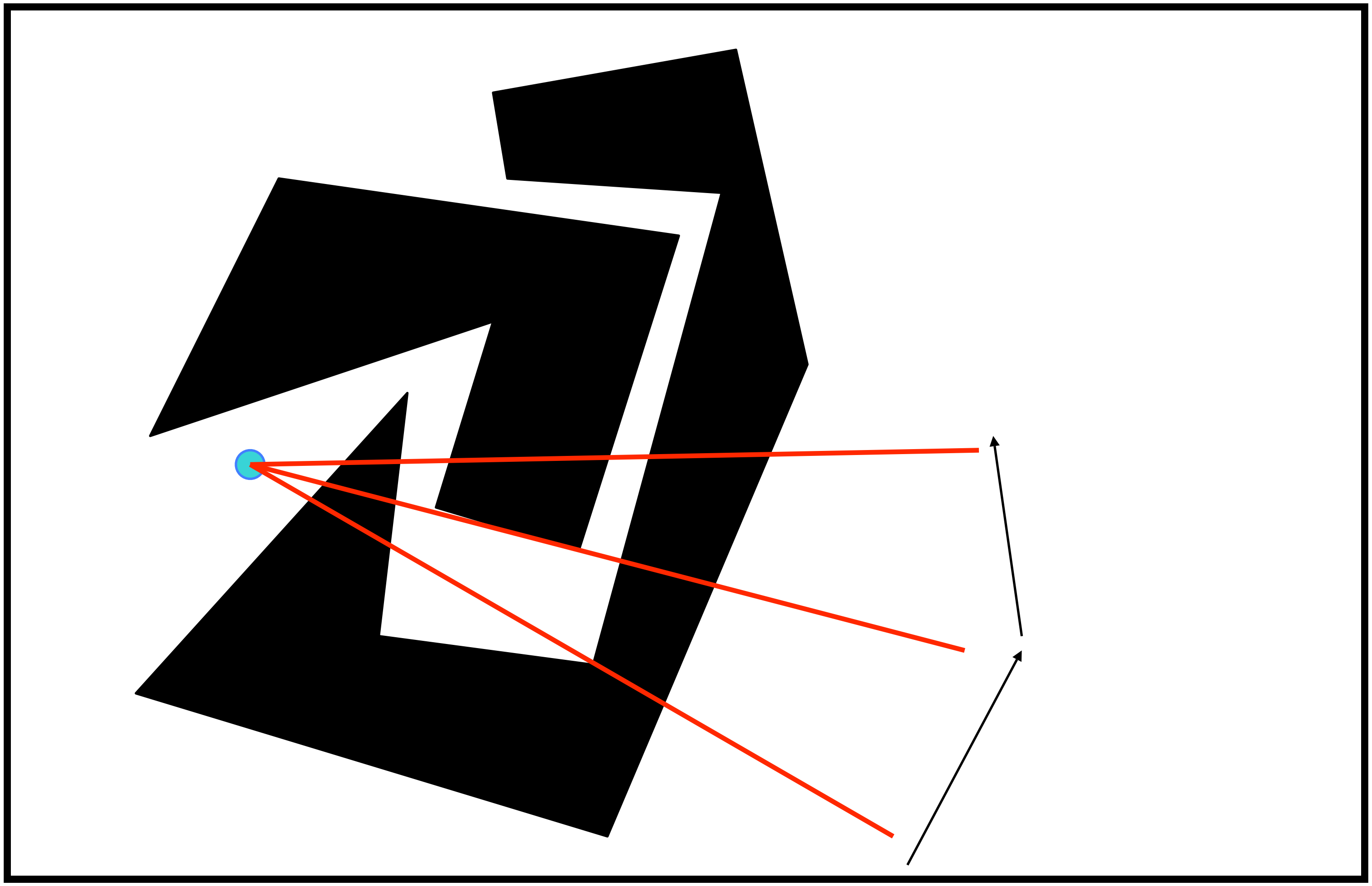
- Brute Force Algorithm
  - add all edges in C-obstacle to G
  - for each pair of vertices (x, y) of C-obstacle, add the edge (x, y) to G if the straight line segment connecting them lies entirely in cl(C-free)
    - test (x; y) for intersection with all O(n) edges of C-obstacle
    - O(n^2) pairs to test, each test takes O(n) time



Complexity: O(n^3), n is number of vertices in C-obstacle

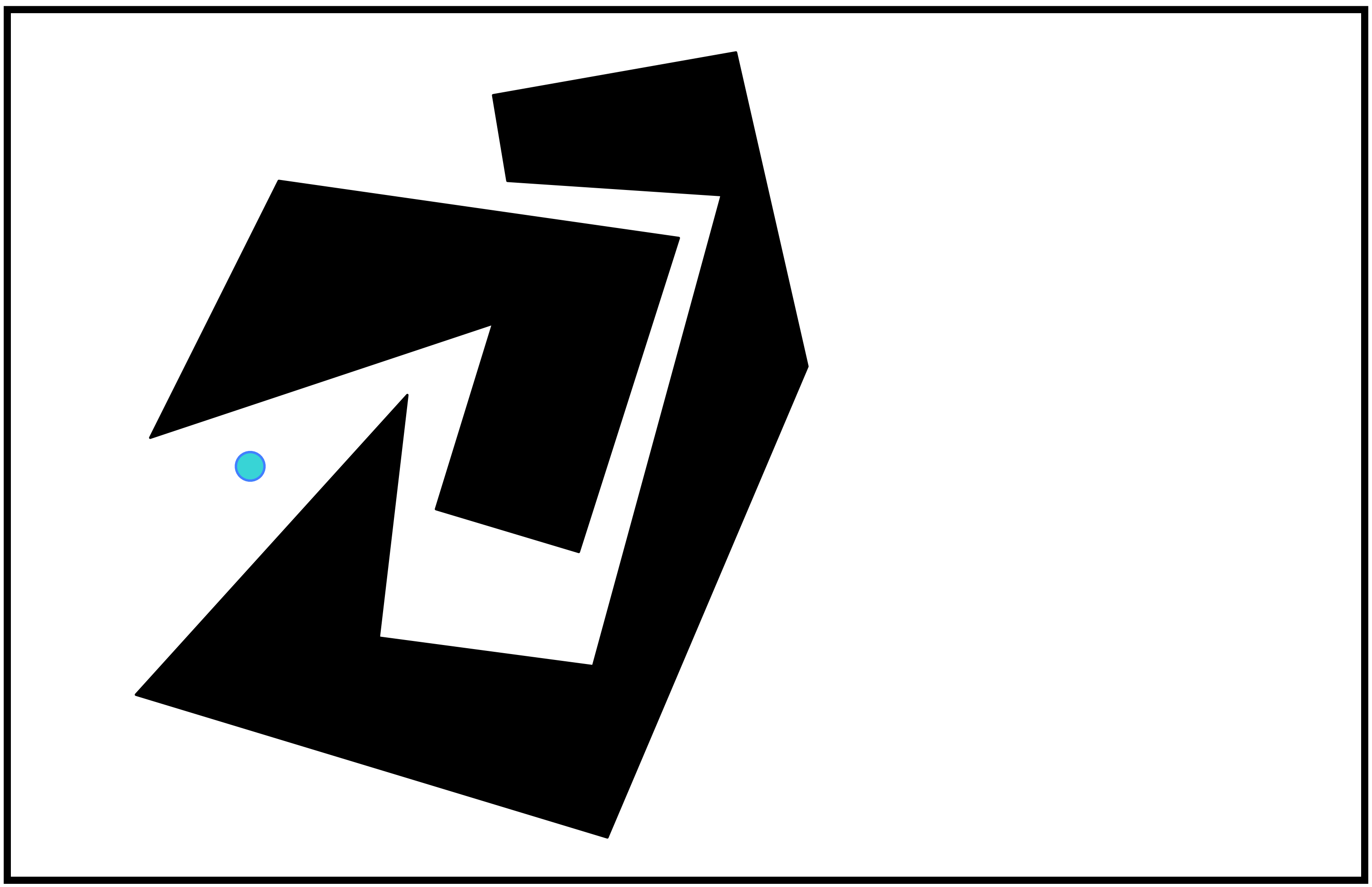
Visibility Graph

- A better algorithm?



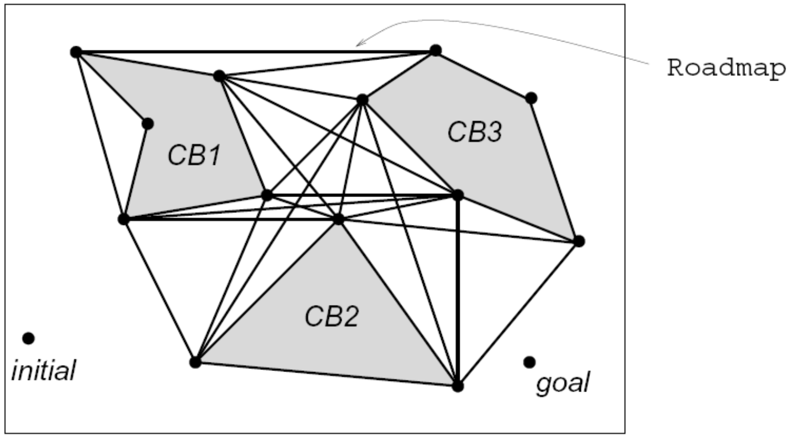
Visibility Graph

- An even better algorithm?



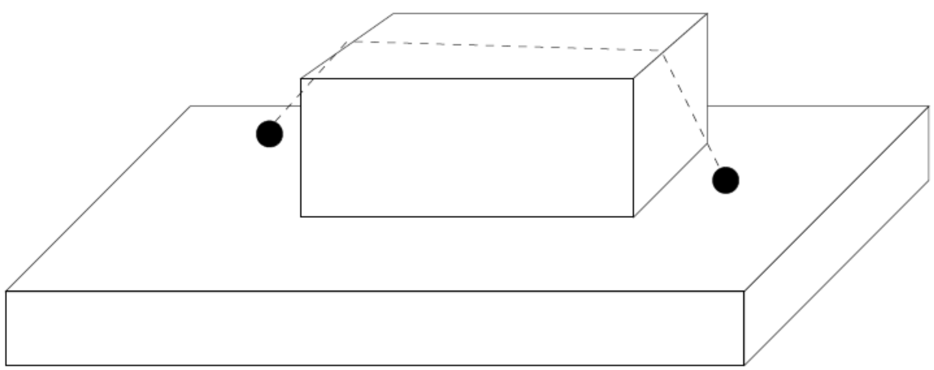
Visibility Graph

- **Visibility graphs (Good news)**
  - are conceptually simple
  - shortest paths (if query cannot see each other)
  - we have efficient algorithms if is polygonal
    - *$O(n^2)$,* where n is number of vertices of C-obstacle
    - *$O(k + n \log n)$,* where k is number of edges in G
  - we can make a 'reduced' visibility graph (don't need all edges)



Visibility Graph in 3-D

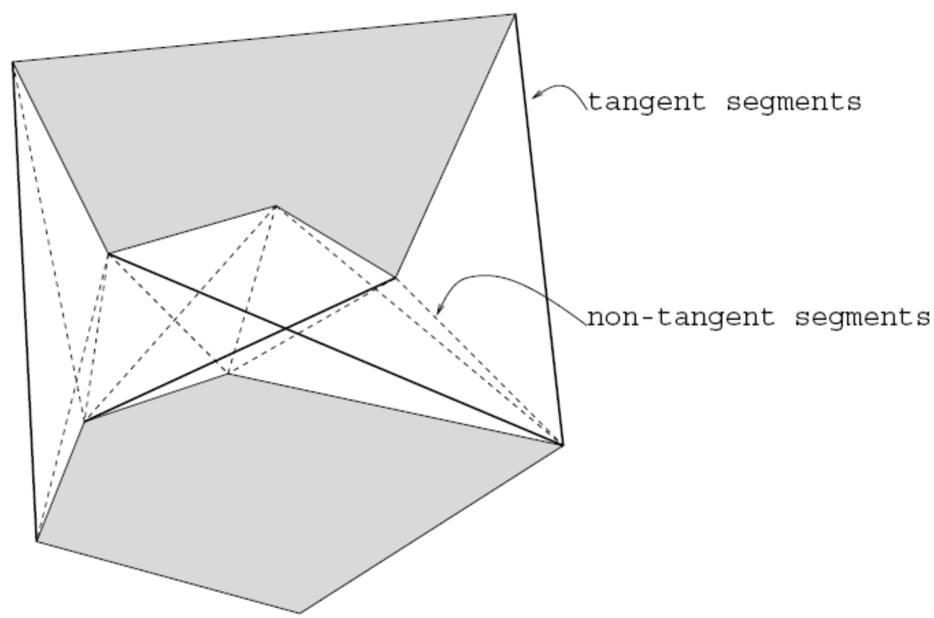
- Visibility graphs don't necessarily contain shortest paths in R^3
  - in fact finding shortest paths in R^3 is NP-hard [Canny 1988]
  - $(1 + \varepsilon^2)$ approximation algorithm [Papadimitriou 1985]



Bad news: really only suitable for two-dimensional C

Reduced Visibility Graph

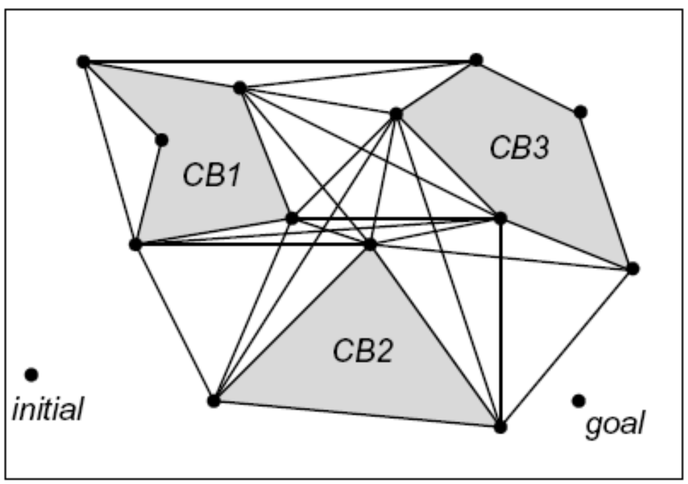
- we don't really need all the edges in the visibility graph (even if we want shortest paths)
- Definition: Let *L* be the line passing through an edge (x; y) in the visibility graph G. The segment (x; y) is a tangent segment *iff* L is tangent to C-obstacle at both x and y.



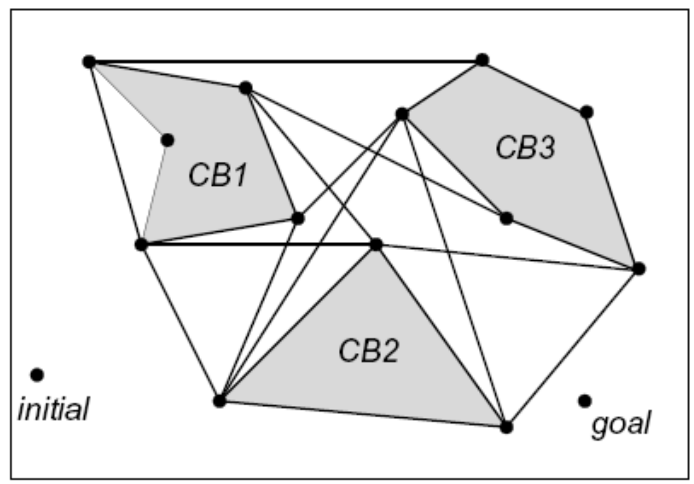
Reduced Visibility Graph

- It turns out we need only keep
  - convex vertices of C-obstacle
  - non-CB edges that are tangent segments

Visibility Graph

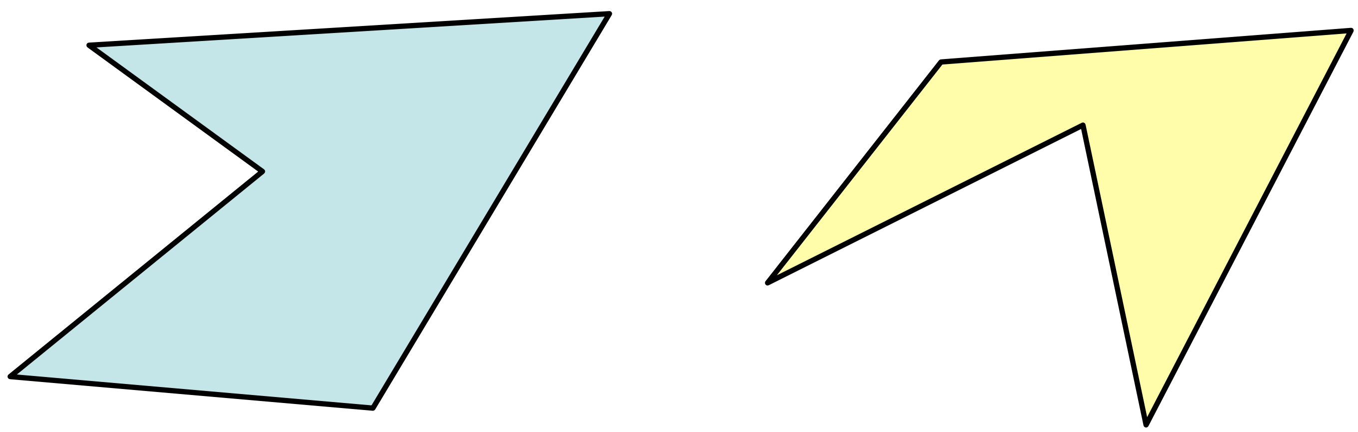


Reduced Visibility Graph



Reduced Visibility Graph

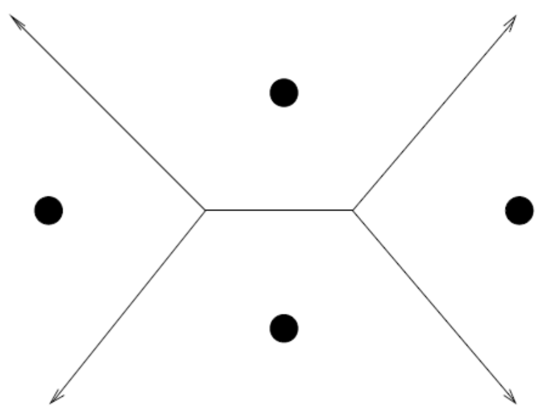
- Reduced visibility graphs are easier to build
  - construct convex hull of each C-obstacle piece eliminate non-convex vertices
  - construct pairwise tangents between each convex C-obstacle piece
- easy to construct tangents between two convex polygons
  - How?



Voronoi Diagram for Point Sets

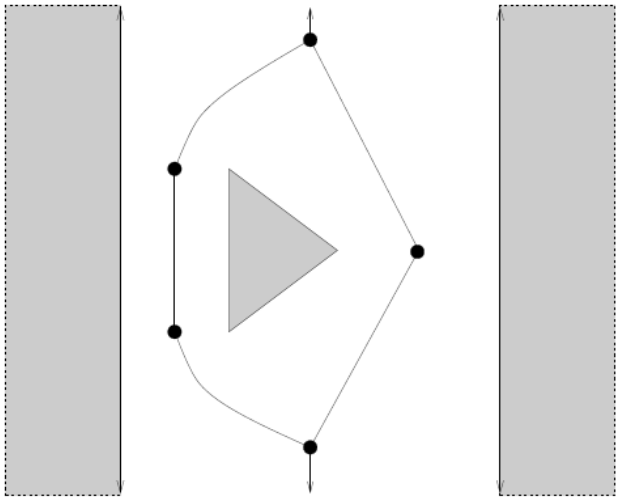
- Voronoi diagram of point set X consists of straight line segments, constructed by
  - computing lines bisecting each pair of points and their intersections
  - computing intersections of these lines
  - keeping segments with more than one nearest neighbor

- segments of Vor(X) have largest clearance from X and regions identify closest point of X



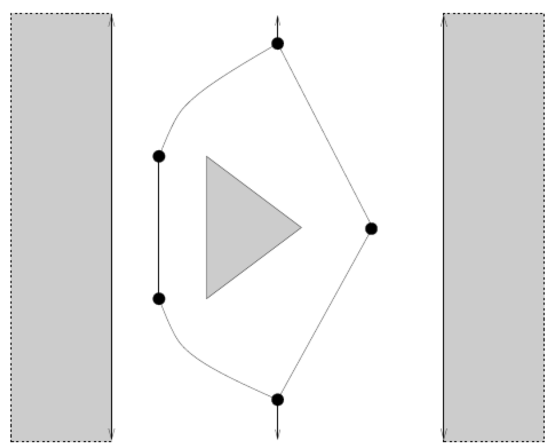
Voronoi Diagram for Point Sets

- When $C = R^2$ and polygonal C-obstacle, Vor(Cfree) consists of a finite collection of straight line segments and parabolic curve segments (called arcs)
  - straight arcs are defined by two vertices or two edges of C-obstacle, i.e., the set of points equally close to two points (or two line segments) is a line



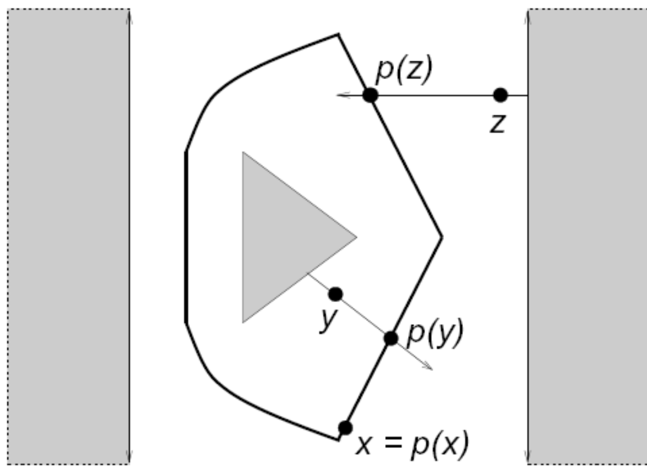
Voronoi Diagram for Point Sets

- Naive Method of Constucting V or(Cfree)
  - compute all arcs (for each vertex-vertex, edge-edge, and vertex-edge pair)
  - compute all intersection points (dividing arcs into segments)
  - keep segments which are closest only to the vertices/edges that



Retraction

- Retraction $\rho : C_{free} \rightarrow \mathrm{Vor}(C_{free})$



To find a path:

1. compute Vor(C_{free})

2. find paths from q_{init} and q_{goal} to $\rho(q_{init})$ and $\rho(q_{goal})$, respectively

3. search Vor(C_{free}) for a set of arcs connecting $\rho(q_{init})$ and $\rho(q_{goal})$

Cell Decomposition

- Idea: decompose C_{free} into a collection K of non-overlapping cells such that the union of all the cells exactly equals the free C-space

- Cell Characteristics:
  - geometry of cells should be simple so that it is easy to compute a path between any two configurations in a cell
  - it should be pretty easy to test the adjacency of two cells, i.e., whether they share a boundary
  - it should be pretty easy to find a path crossing the portion of the boundary shared by two adjacent cells

- Thus, cell boundaries correspond to 'criticalities' in C, i.e., something changes when a cell boundary is crossed. No such criticalities in C occur within a cell.

Cell Decomposition

- **Preprocessing:**
  - represent C_{free} as a collection of cells (connected regions of C_{free})
    - planning between configurations in the same cell should be 'easy'
  - build connectivity graph representing adjacency relations between cells
    - cells adjacent if can move directly between them
- **Query:**
  - locate cells k_{init} and k_{goal} containing start and goal configurations
  - search the connectivity graph for a 'channel' or sequence of adjacent cells connecting k_{init} and k_{goal}
  - find a path that is contained in the channel of cells

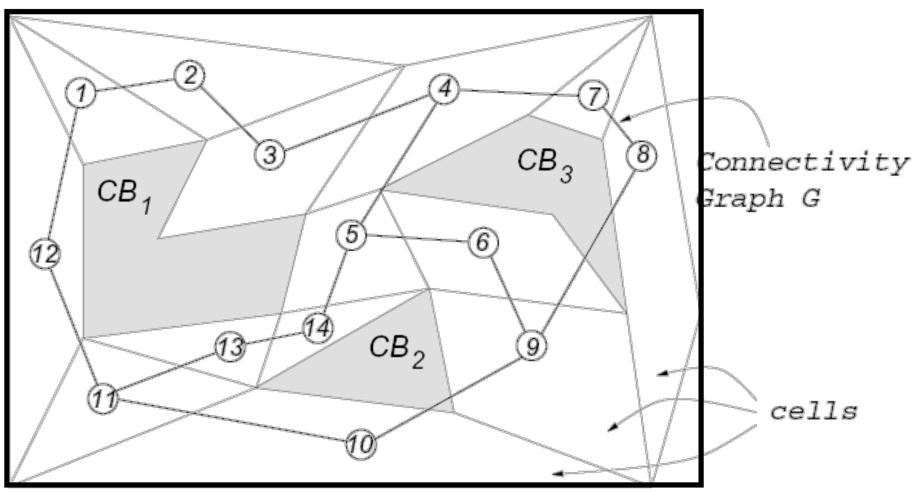
Difficult

- Two major variants of methods:
  - exact cell decomposition:
    - set of cells exactly covers C_{free}
    - complicated cells with irregular boundaries (contact constraints)
    - harder to compute
  - approximate cell decomposition:
    - set of cells approximately covers C_{free}
    - simpler cells with more regular boundaries

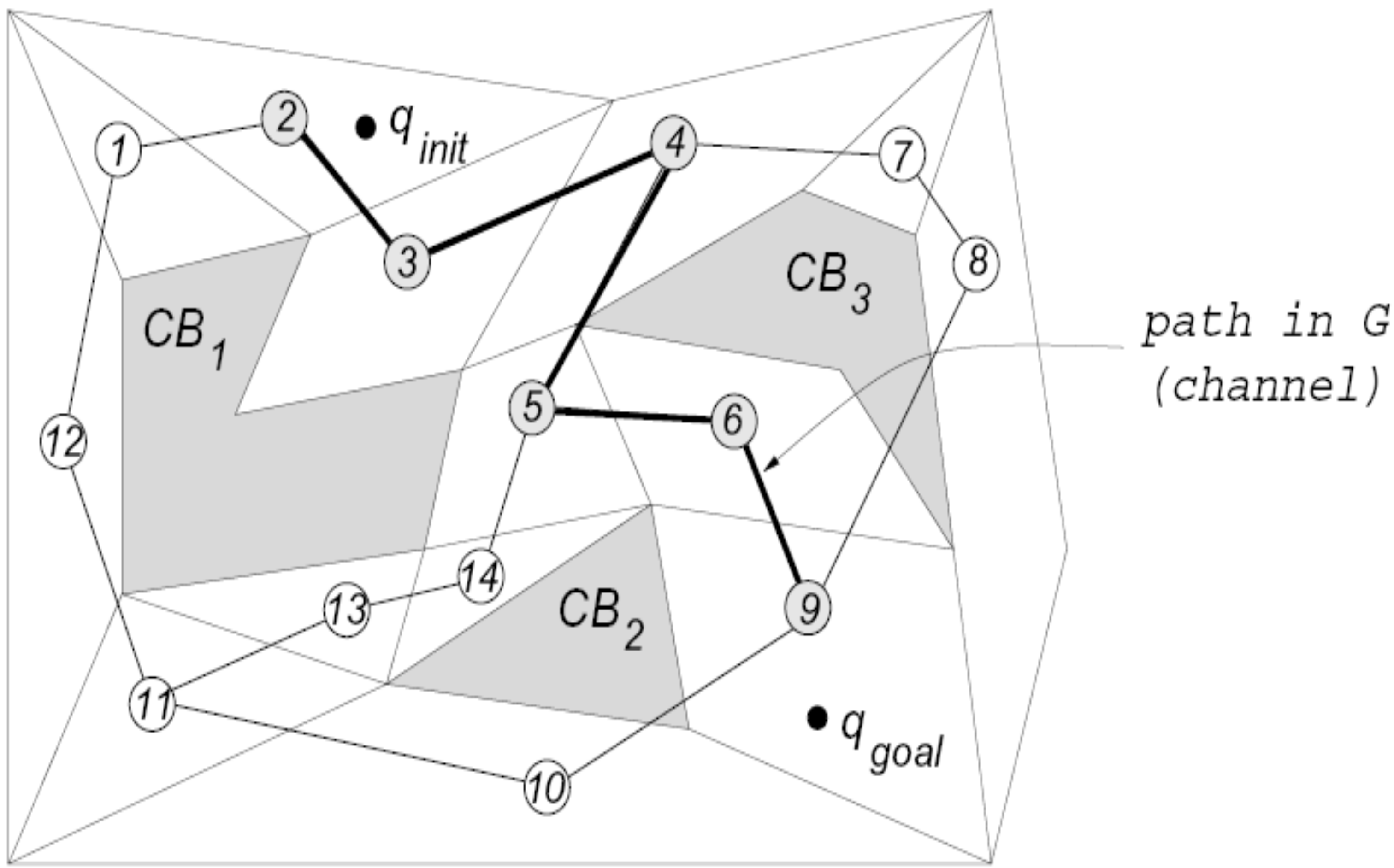
Convex Decomposition

- A convex polygonal decomposition K of C_{free} is a finite collection of convex polygons, called cells, such that the interiors of any two cells do not intersect and the union of all cells is C_{free}.
  - Two cells k and k′ ∈ K are adjacent iff k∩k′ is a line segment of non-zero length (i.e., not a single point)

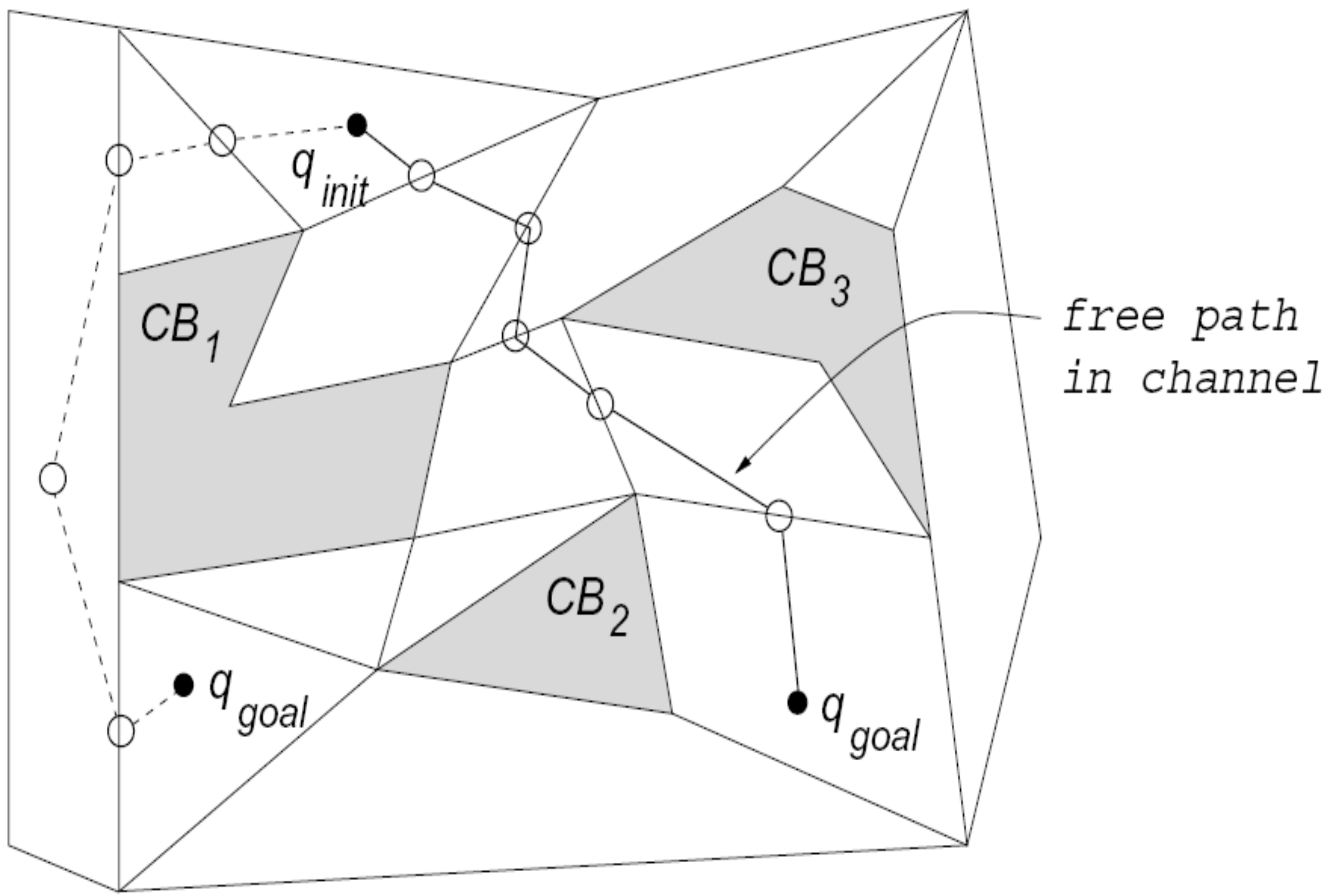
- The connectivity graph associated with a convex polygonal decomposition K of C_{free} is an undirected graph G where
  - nodes in G correspond to cells in K
  - nodes connected by edge in G iff corresponding cells adjacent in K



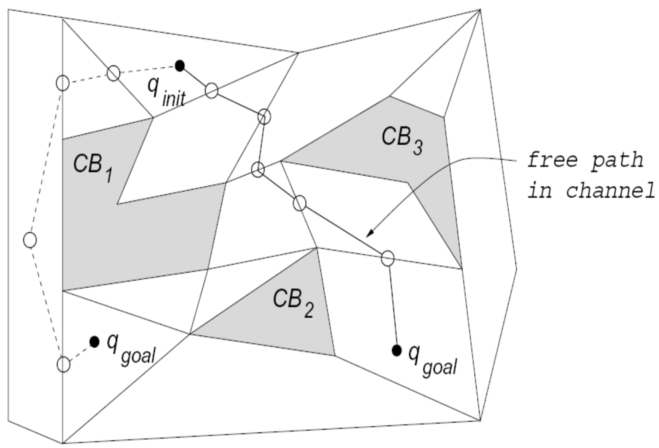
Convex Decomposition



Convex Decomposition



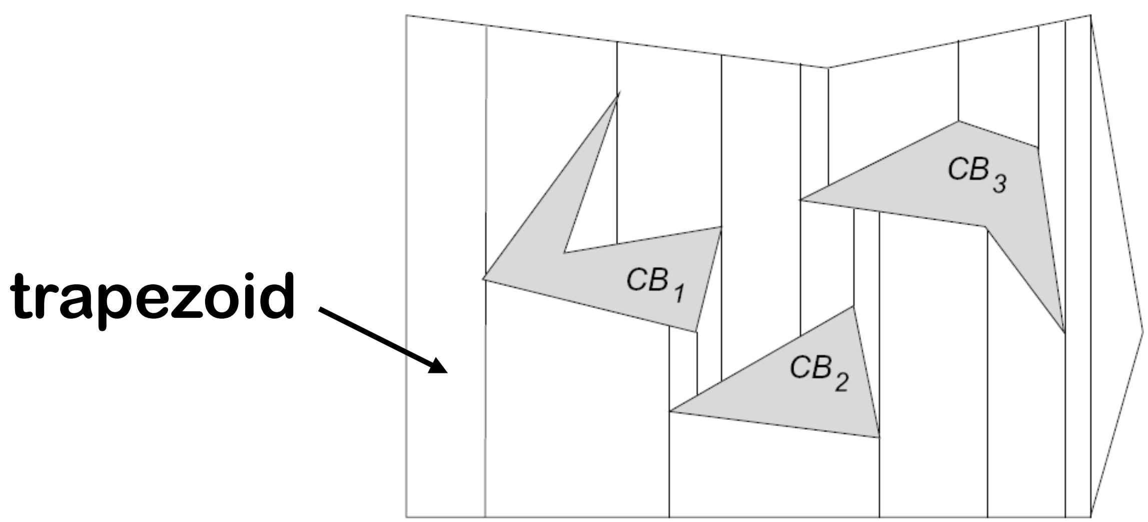
Convex Decomposition



Bad news: Computing convex decomposition is not easy nor can be done efficiently. In fact the problem is NP hard to generate minimum number of convex components for polygon with holes

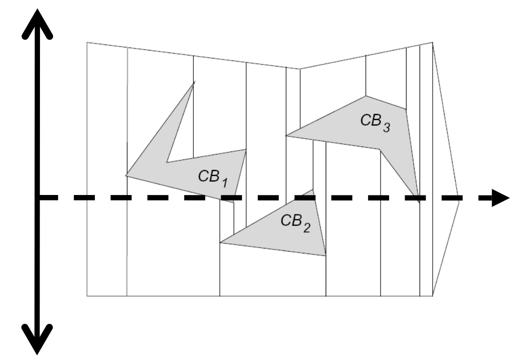
Trapezoidal Decomposition

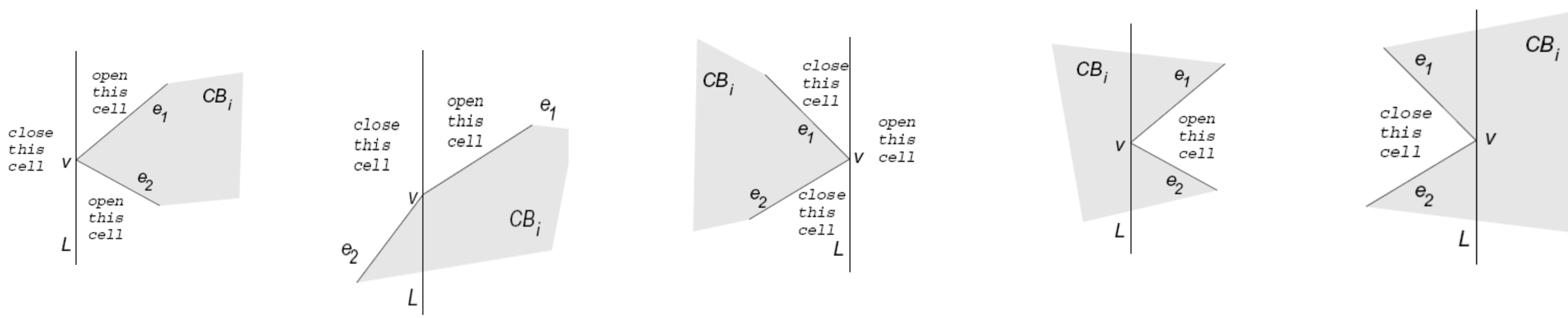
- Basic Idea: at every vertex of C-obstacle, extend a vertical line up and down in Cfree until it touches a C-obstacle or the boundary of Cfree



Trapezoidal Decomposition

- Sweep line algorithm
  - Add vertical lines as we sweep from left to right
  - Events need to be handled accordingly





trapezoidal decomposition can be built in O(n log n) time

Approx. Cell Decomposition

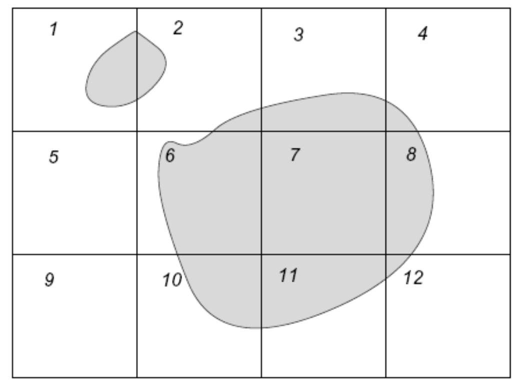
- Construct a collection of non-overlapping cells such that the union of all the cells approximately covers the free C-space!

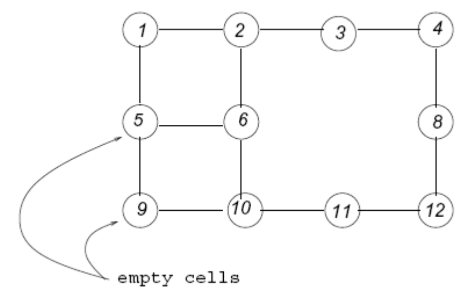
- Cell characteristics
  - Cell should have simple shape
  - Easy to test adjacency of two cells
  - Easy to find path across two adjacent cells

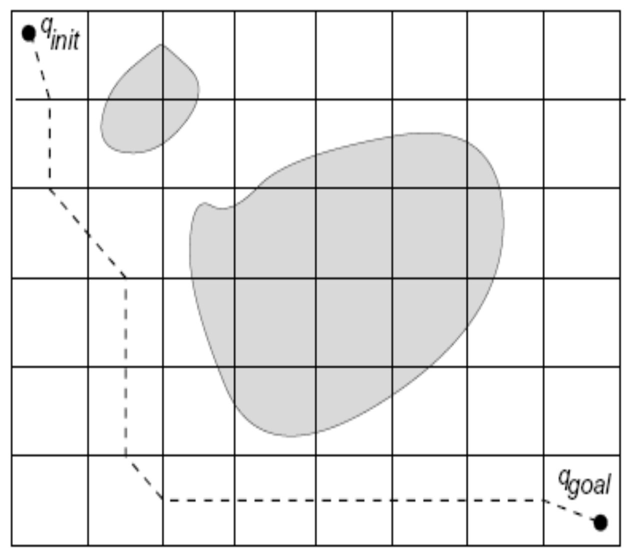
Approx. Cell Decomposition

- Each cell is
  - Empty
  - Full
  - Mixed

- Different resolution
  - Different roadmap

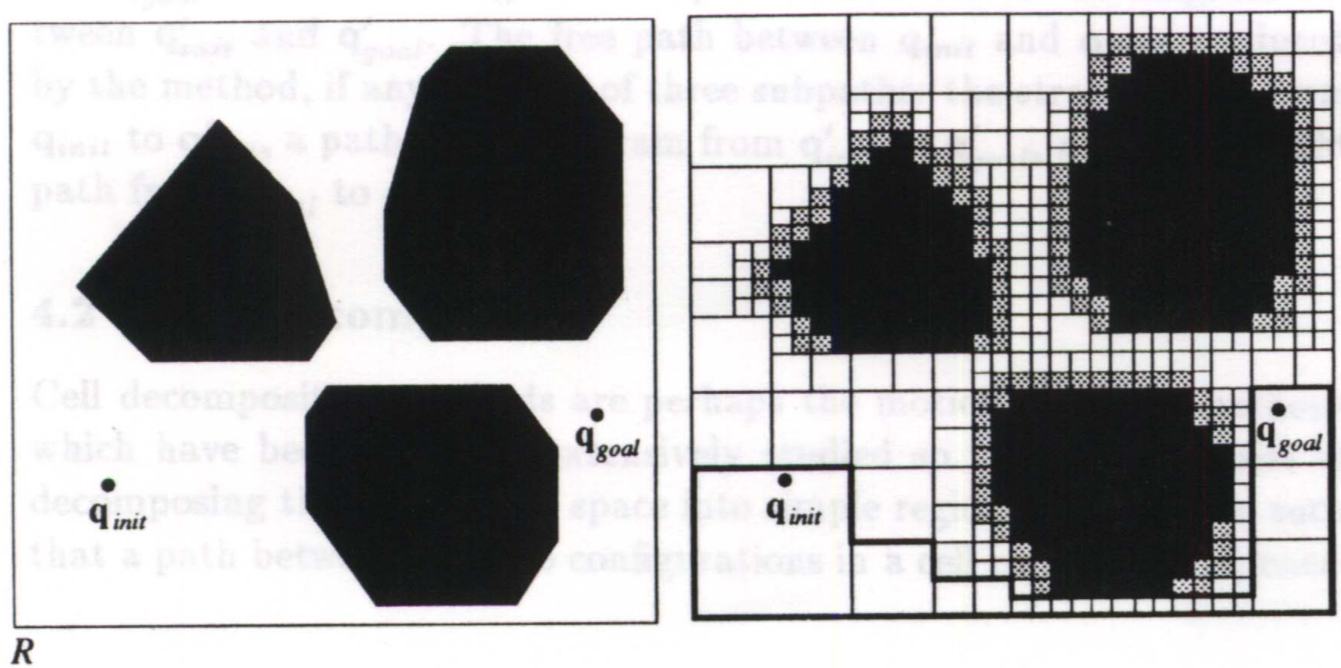






Approx. Cell Decomposition

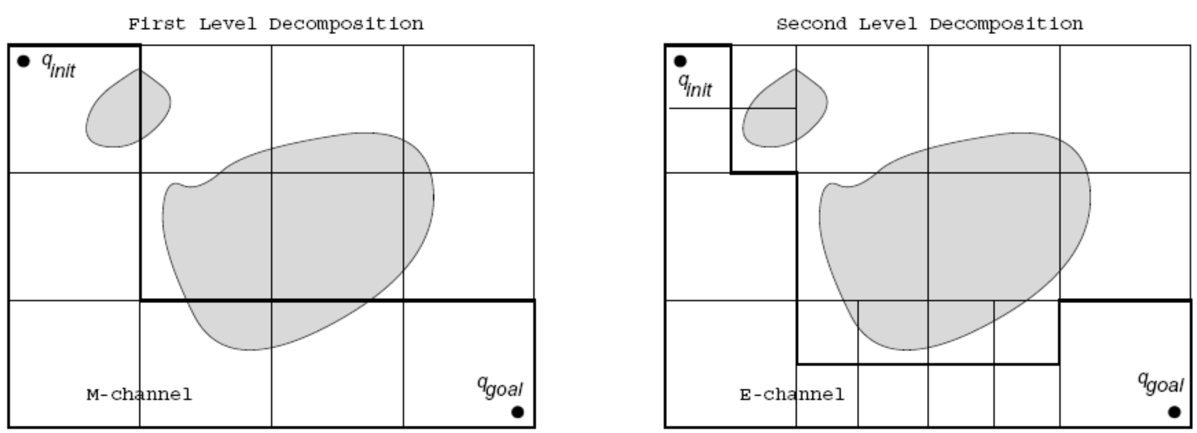
- Higher resolution around CBs



(a) (b)

Approx. Cell Decomposition

- Hierarchical approach
  - Find path using empty and mixed cells
  - Further decompose mixed cells into smaller cells



Approx. Cell Decomposition

- Advantages:
  - simple, uniform decomposition
  - easy implementation
  - adaptive
- Disadvantages:
  - large storage requirement
  - Lose completeness

- Bottom line 1: We sacrifice exactness for simplicity and efficiency

- Bottom line 2: Approx. cell decomposition methods are practically for lower dimension C, i.e., dof <5, b/c they generate too many cells, i.e. (N^d) cells in d dimension

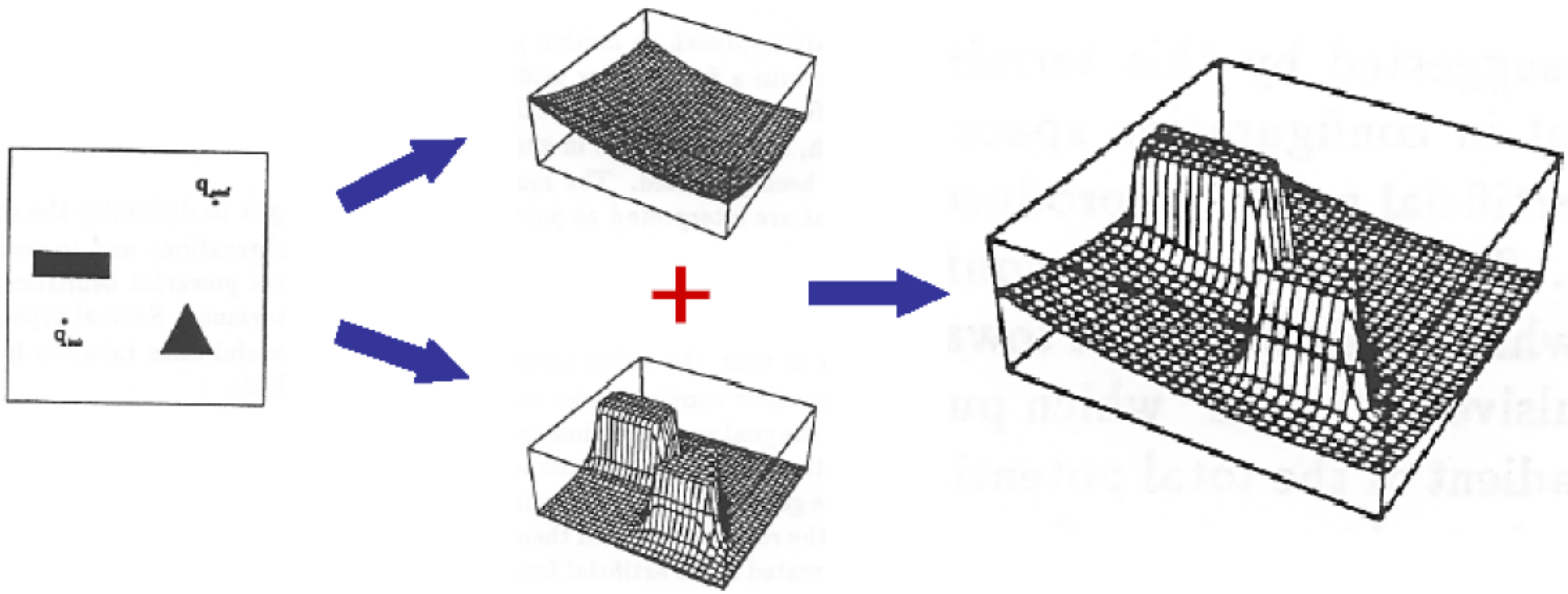
Potential Field Methods

- Approach initially proposed for real-time collision avoidance [Khatib, 86].
  - Hundreds of papers published on it

$$F_{Goal} = -k_p(x - x_{Goal})$$

$$F_{Obstacle} = \begin{cases} \eta\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2}\frac{\partial\rho}{\partial x} & \text{if } \rho \le \rho_0, \\ 0 & \text{if } \rho > \rho_0 \end{cases}$$

Potential Field Methods



Potential Field+Grid Search

- Superimpose a grid over C-space
- Each cell has a potential value
- Search from start to goal on the grid using best-first search or A* search

Potential Field Methods

- At each step move an increment in the direction that minimizes the energy
  - \+ Good heuristic for high DOF

– Can get trapped in local minima
  - use some probabilistic motion to escape

– Oscillations can also occur

General Motion Planning Problems

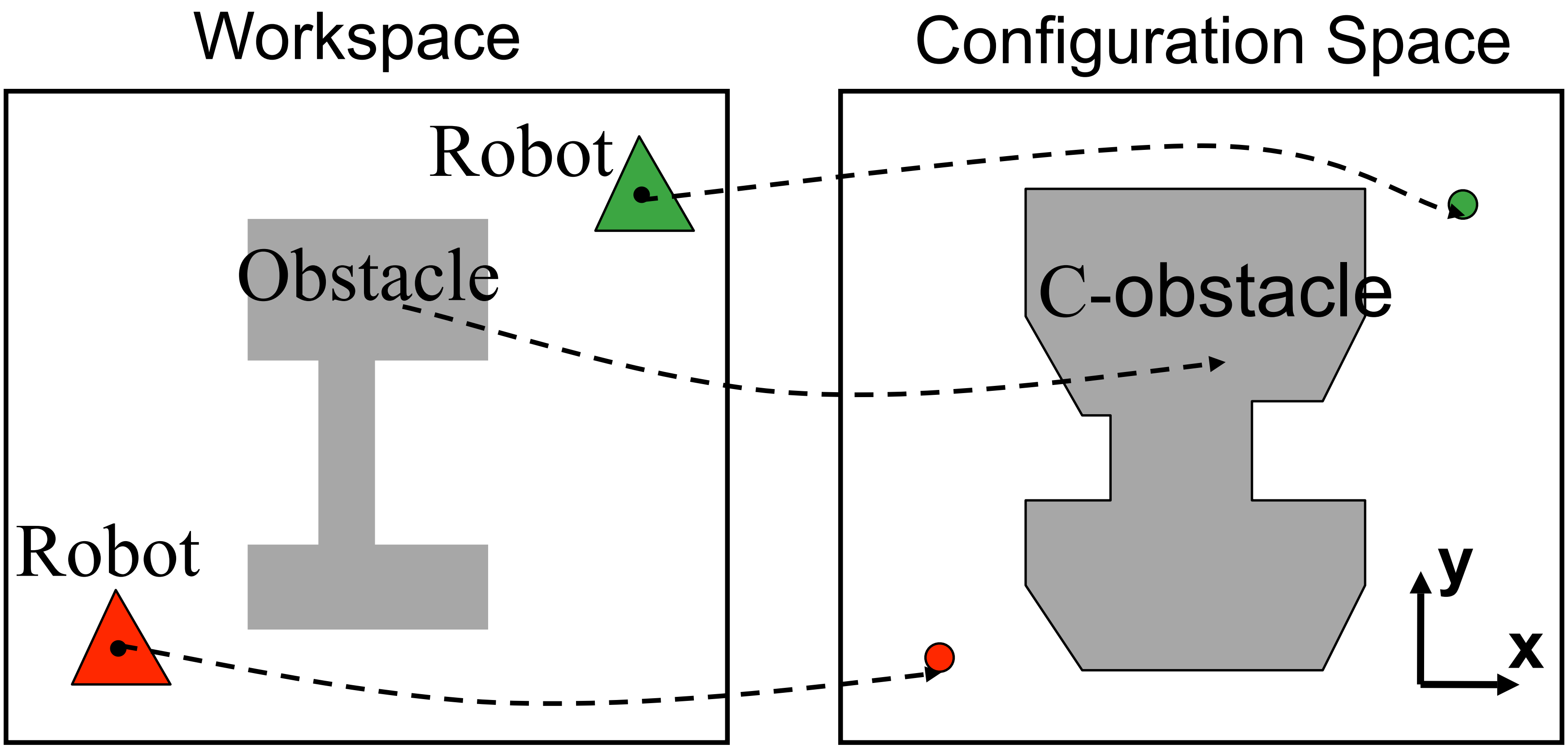
- Well, most robot is not a point and can have arbitrary shape
- What should we do if our robot is not a point?

Configuration Space

- Convert rigid robots, articulated robots, *etc.* into points

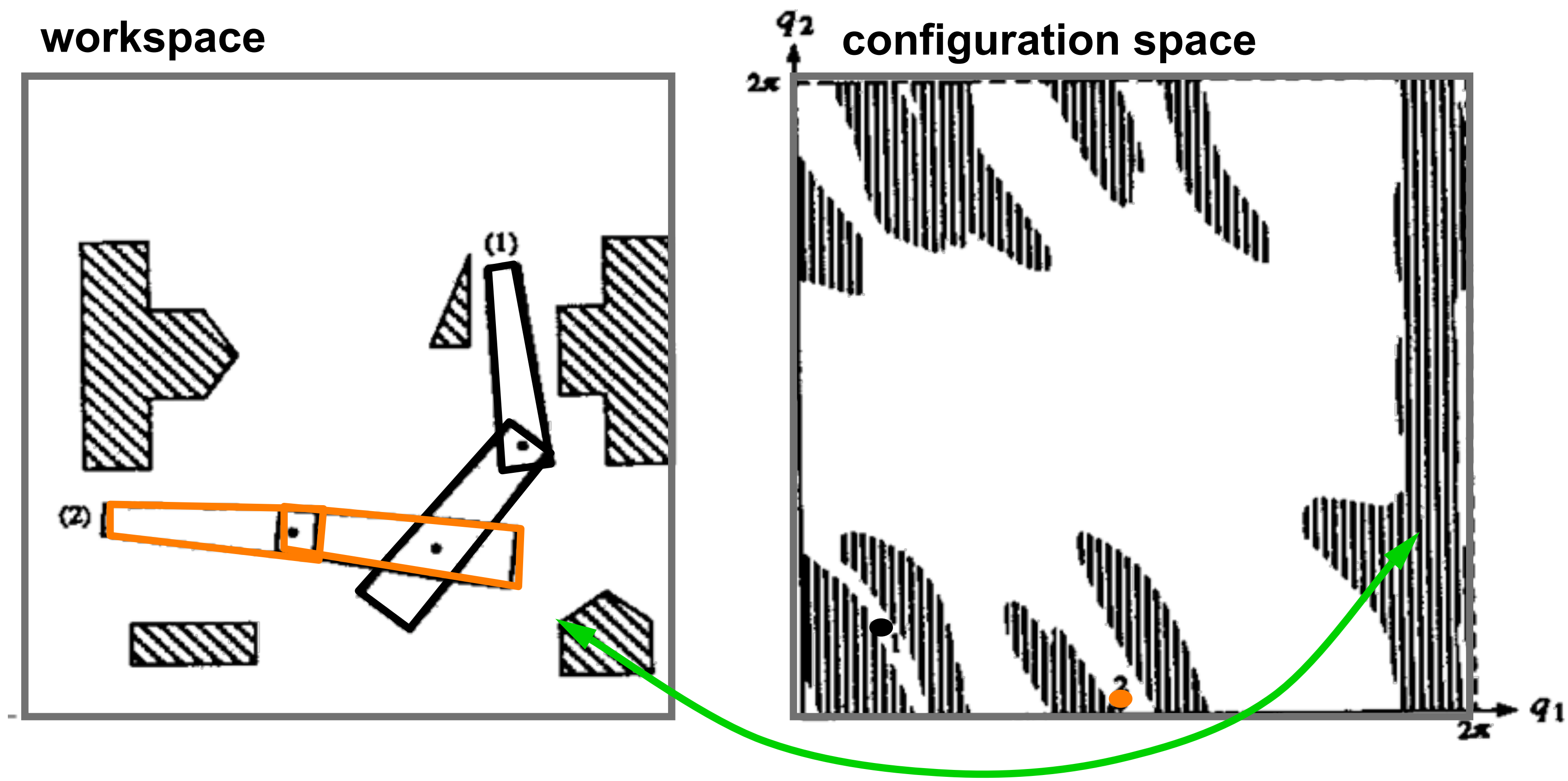
- Apply algorithms for moving points

Configuration Space



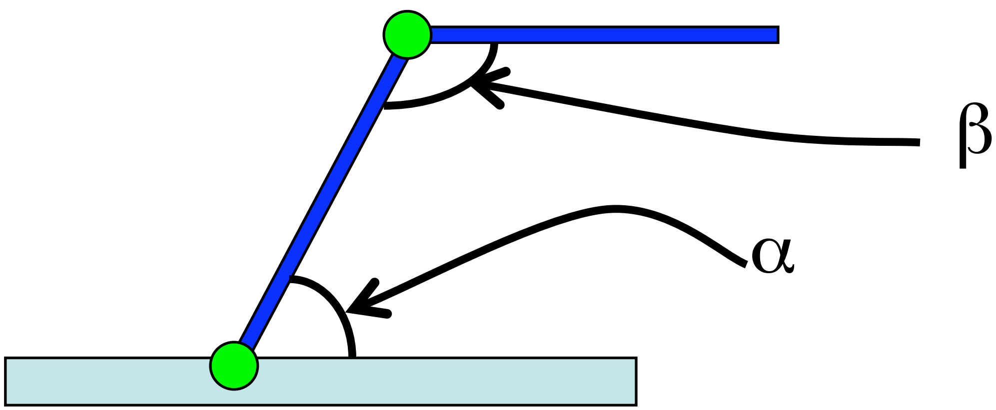
- C-obstacle is a polygon.

Configuration Space



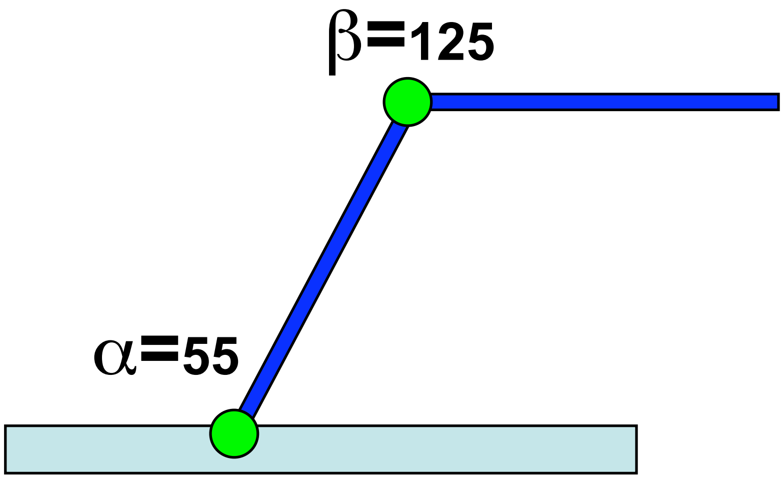
Workspace

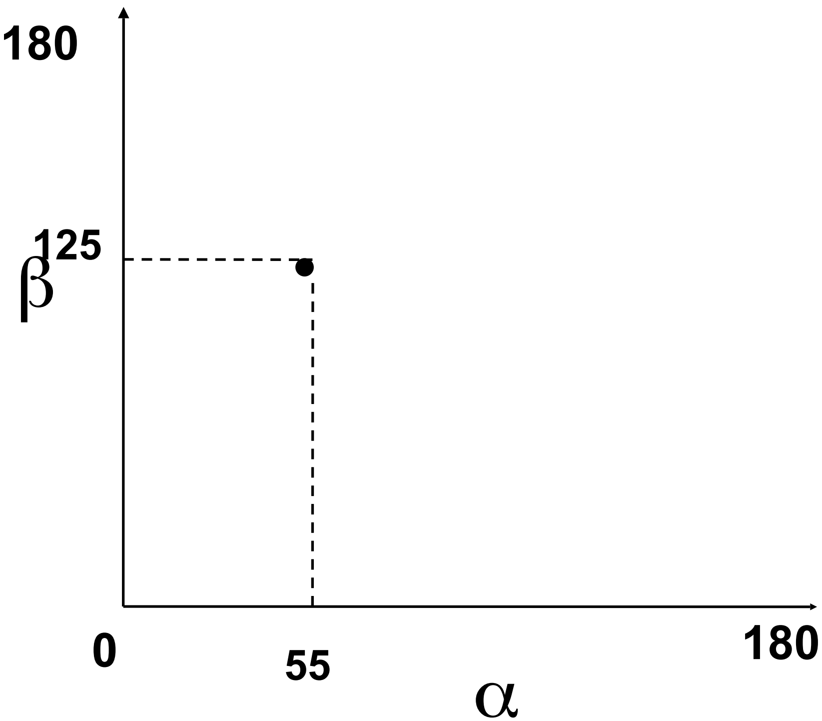
Degree of freedom (DOF)



Configuration Space

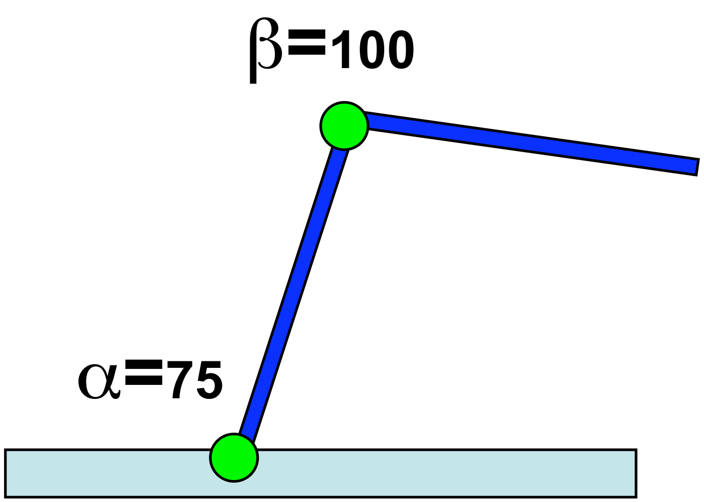
C-Space

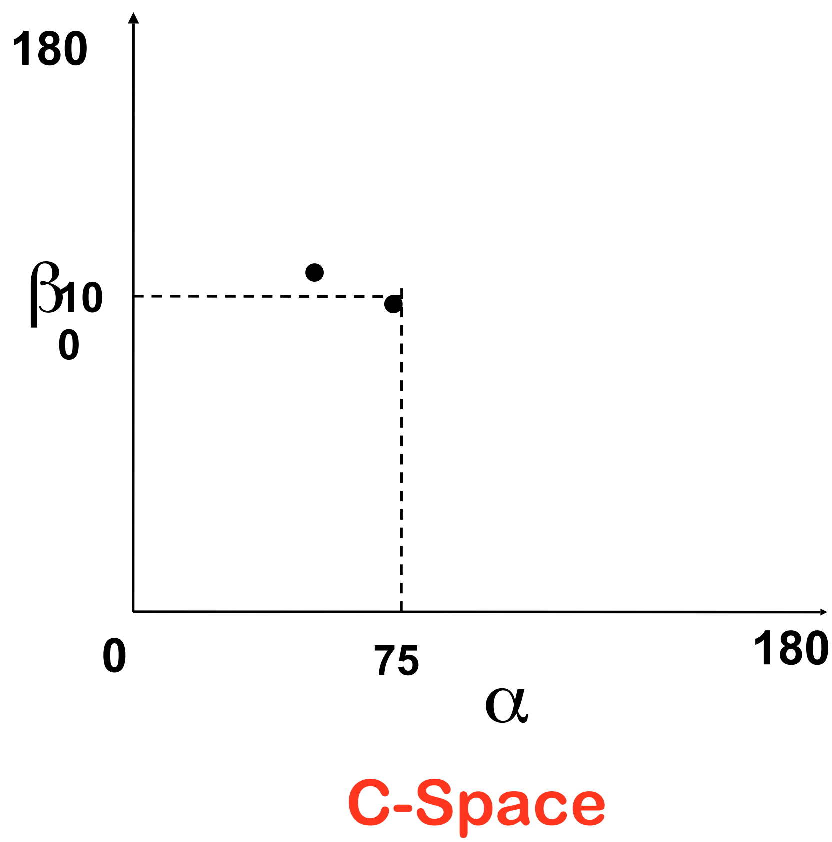




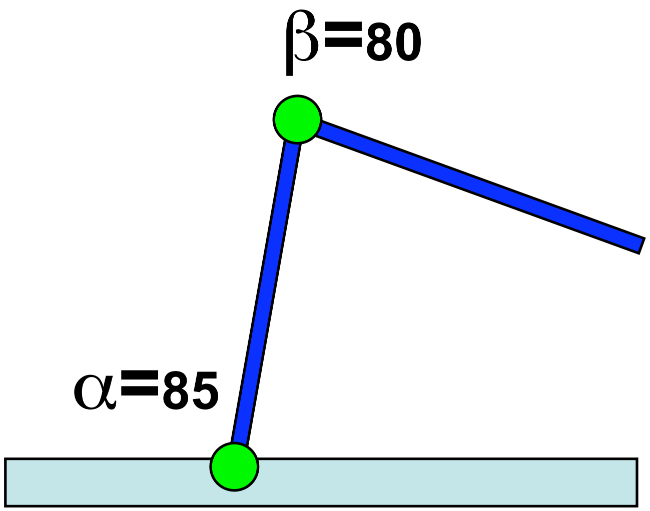
C-Space

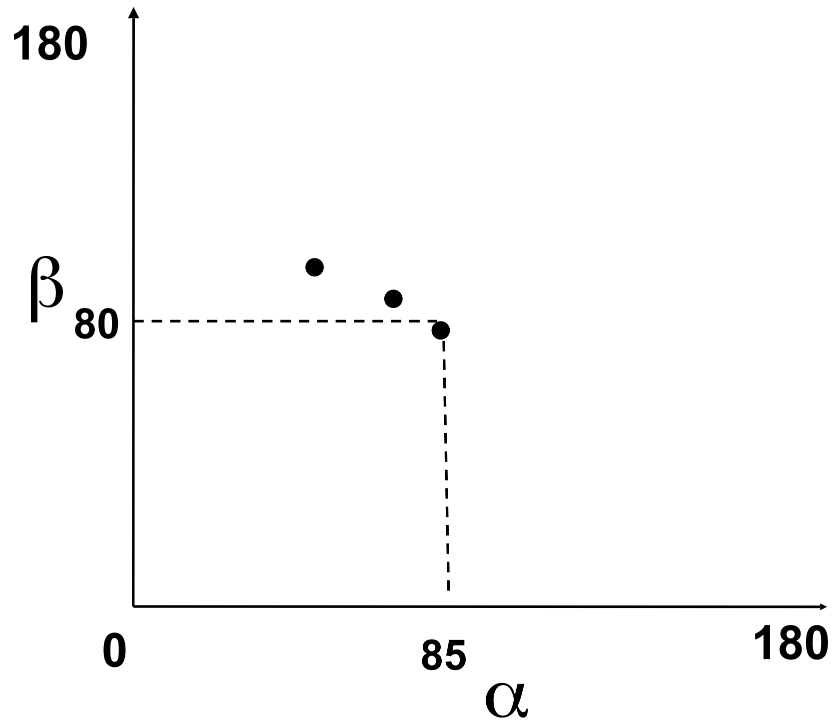
C-Space





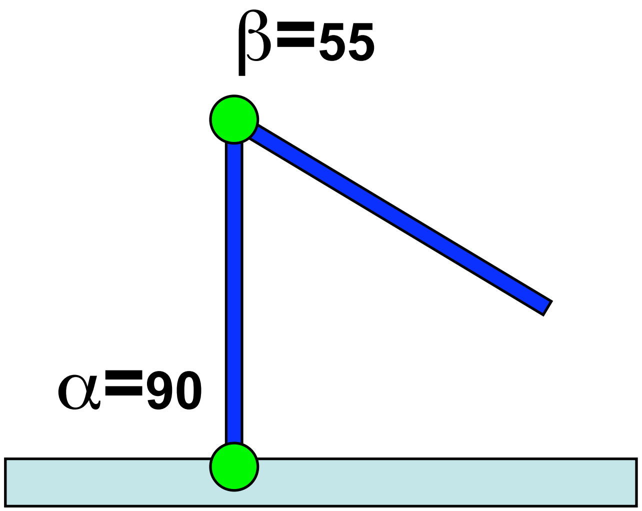
C-Space

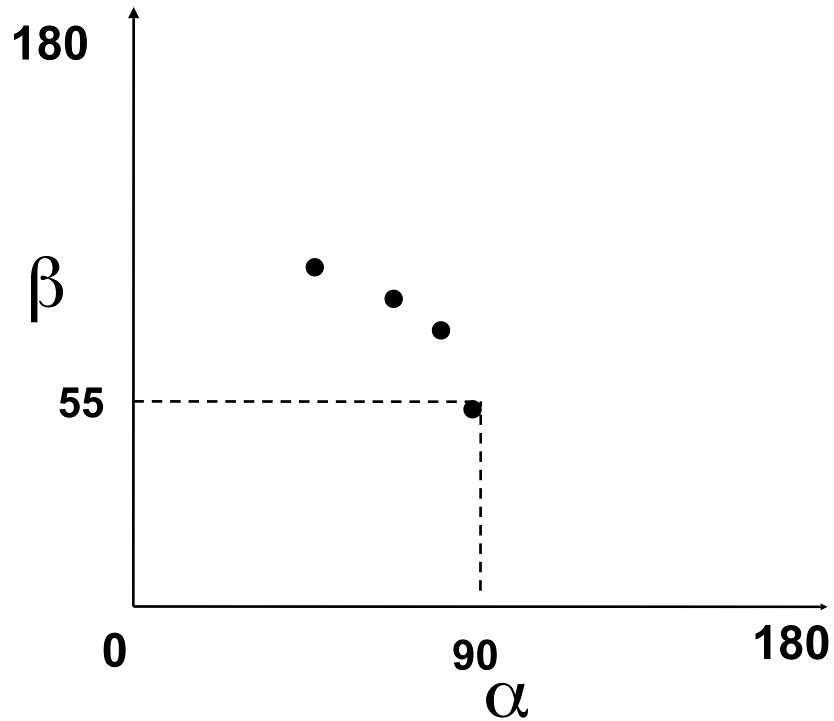




C-Space

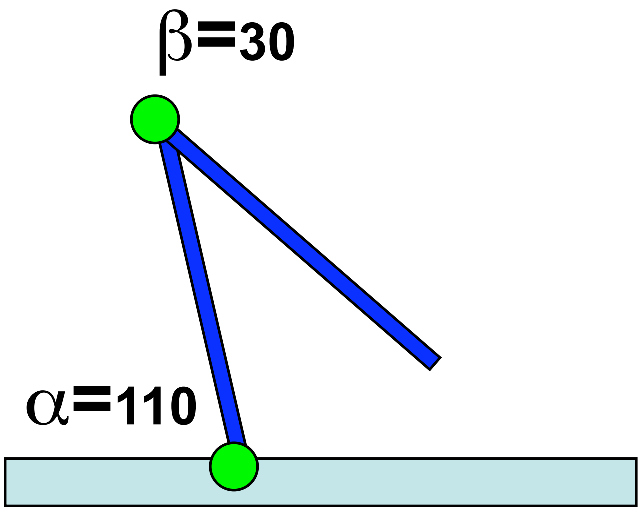
C-Space

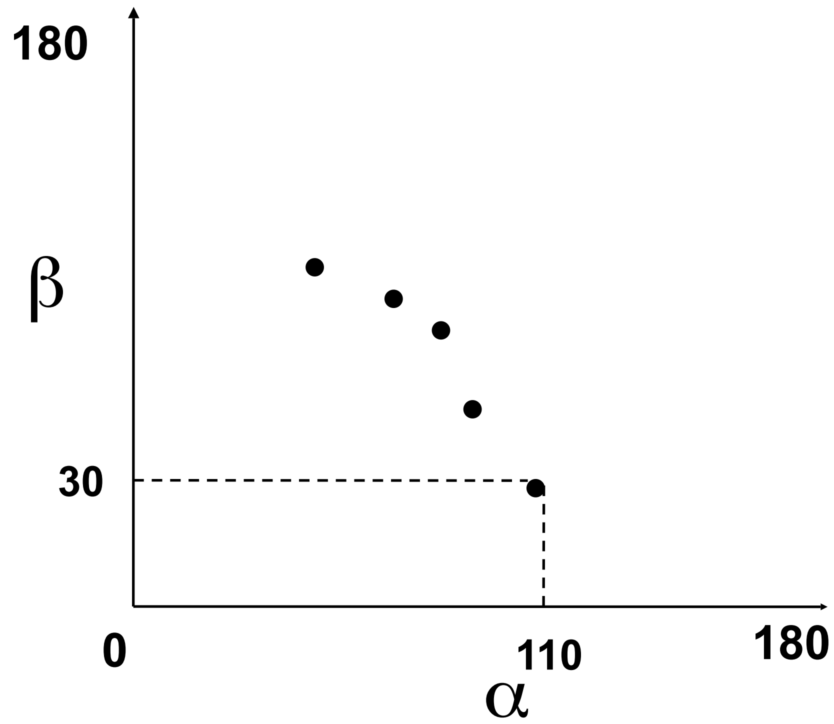




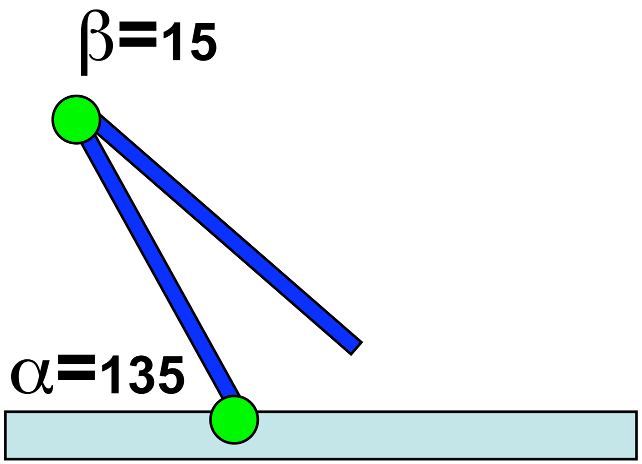
C-Space

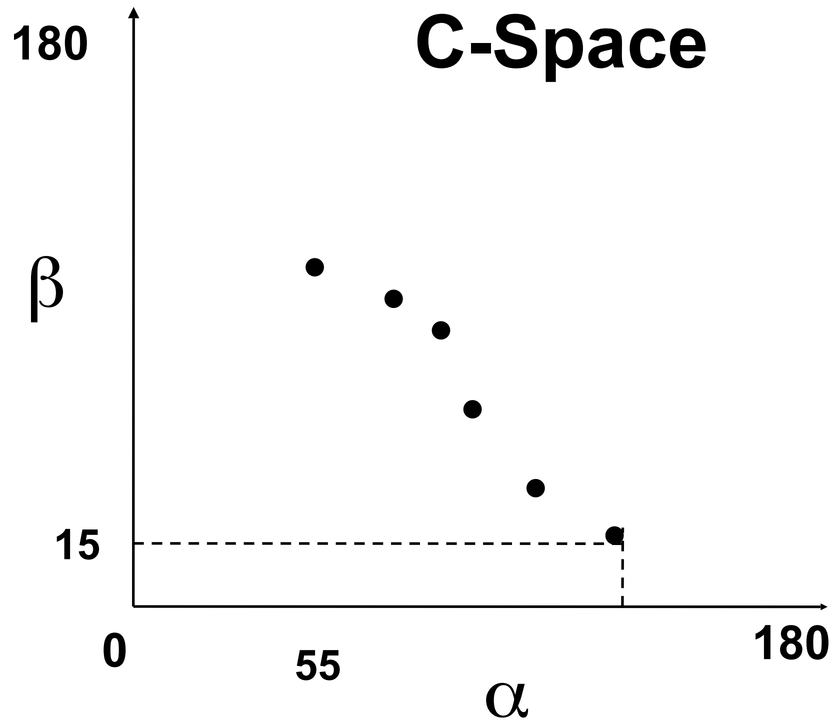
C-Space





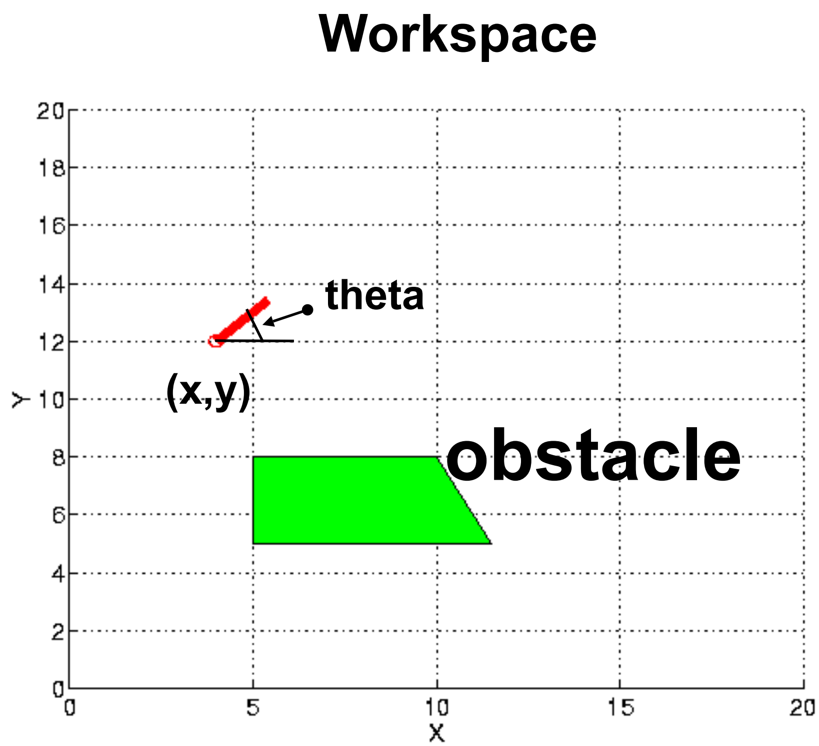
C-Space



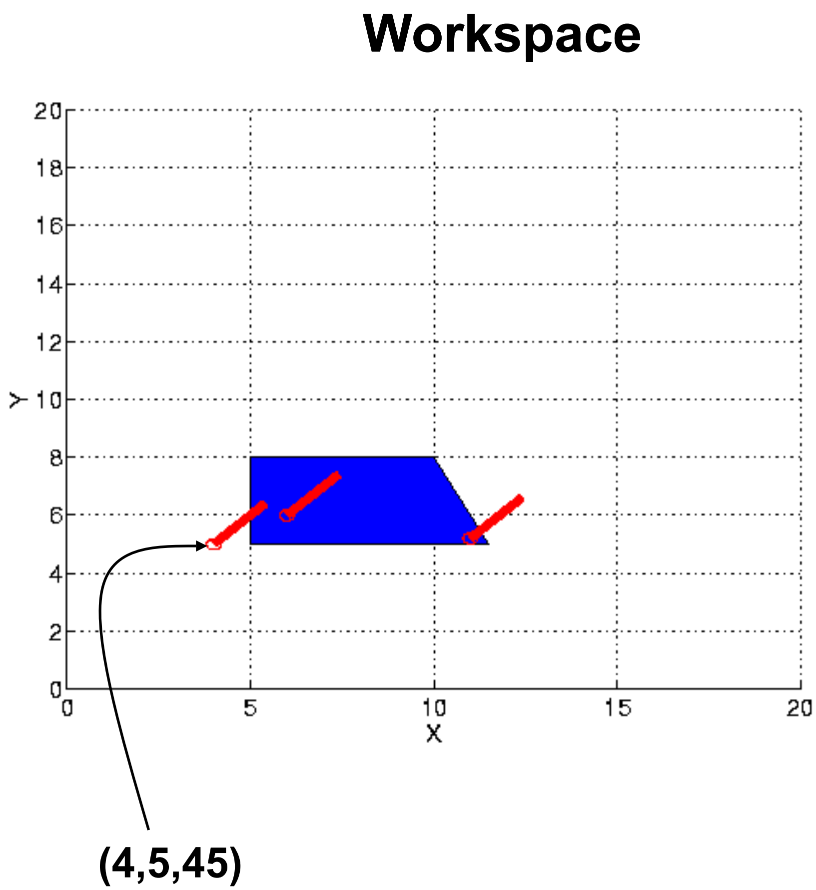


C-Space

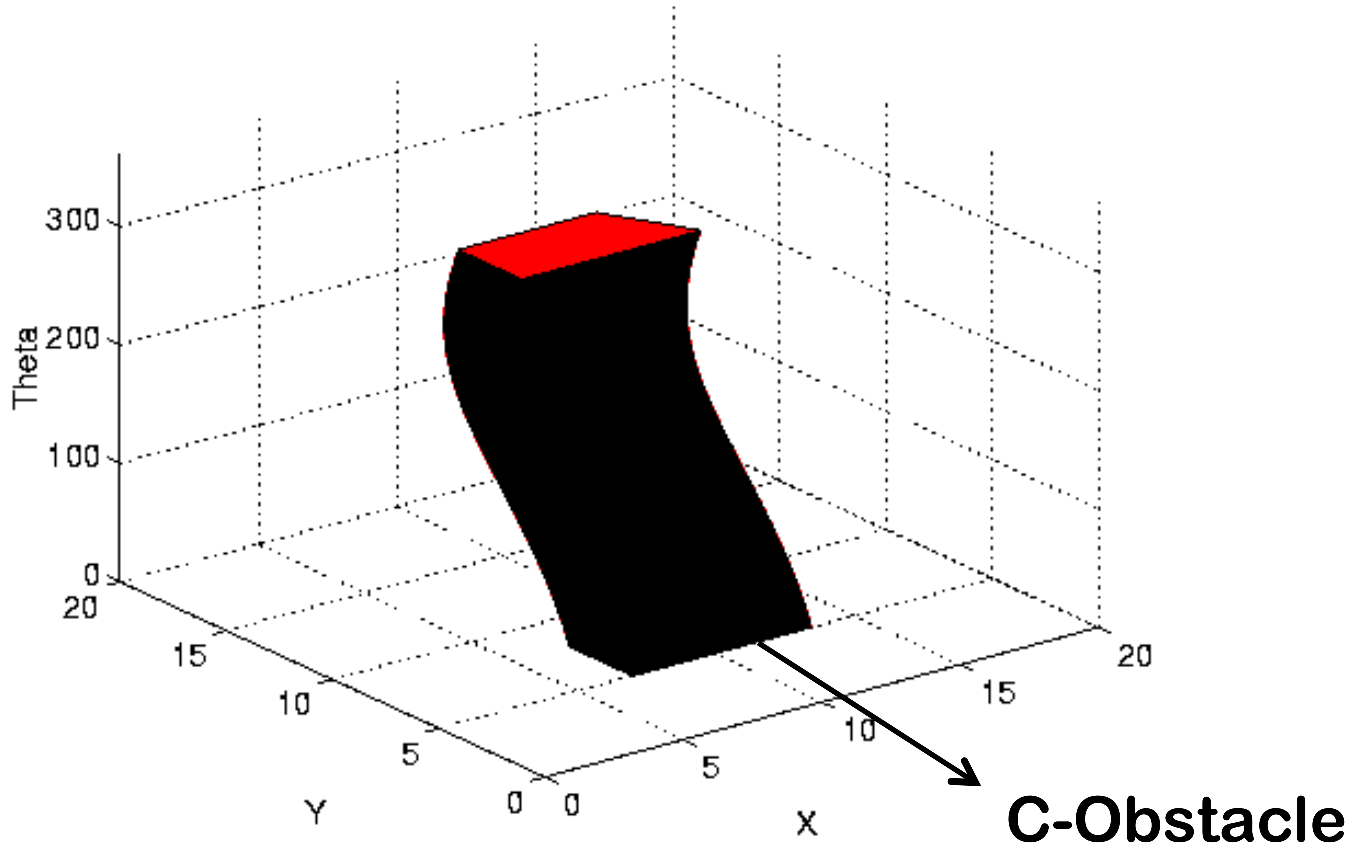
Workspace Obstacle



Configuration (x,y,theta)



C-Space Obstacle

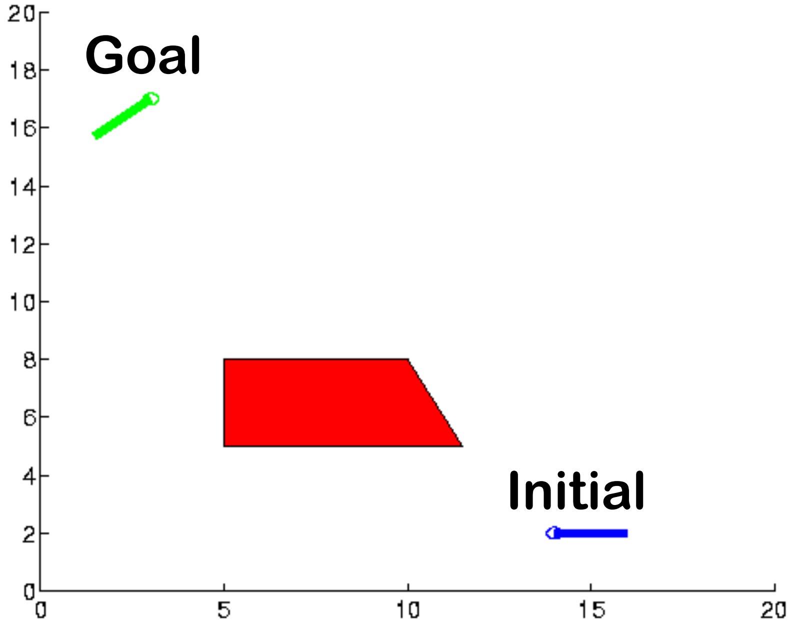


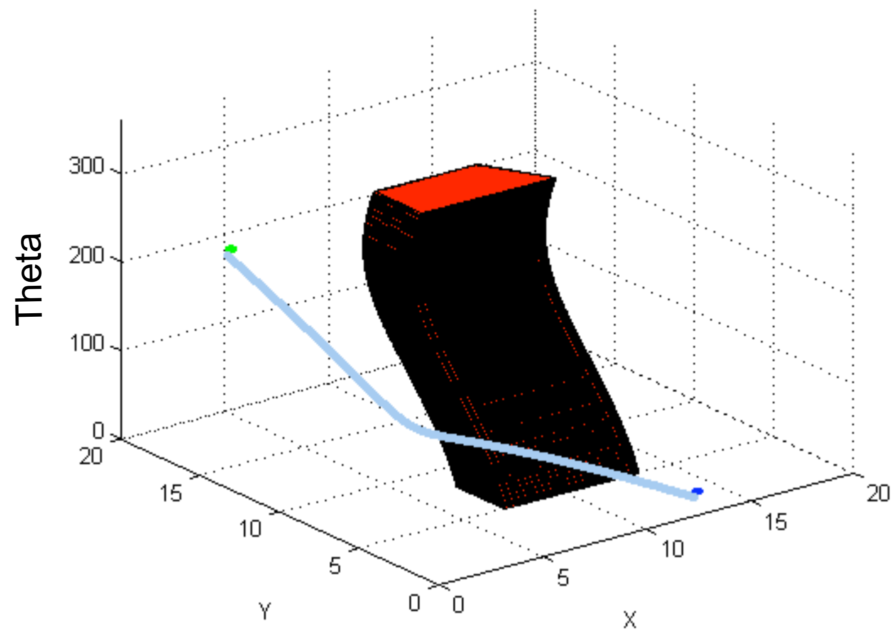
Really look like this???

Finding a Path

Find a path in workspace for a robot

Find a path in C-space for a point

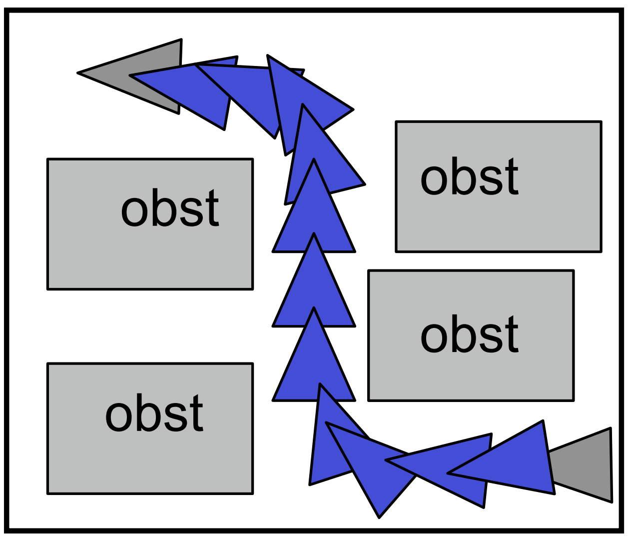




Motion Planning in C-space

Simple workspace obstacle transformed Into complicated C-obstacle!!

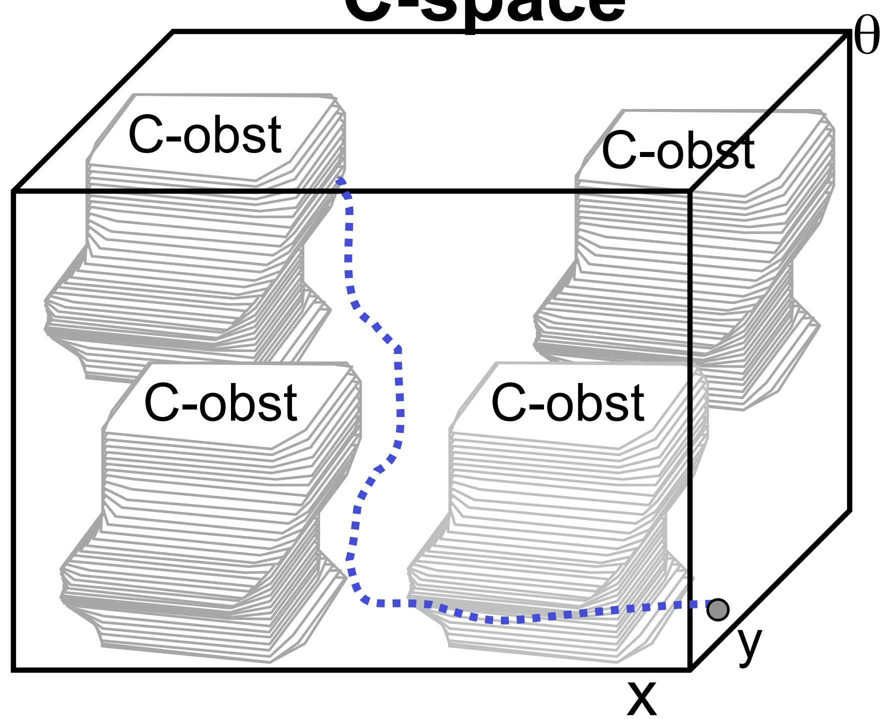
Workspace



robot

Path is swept volume

C-space

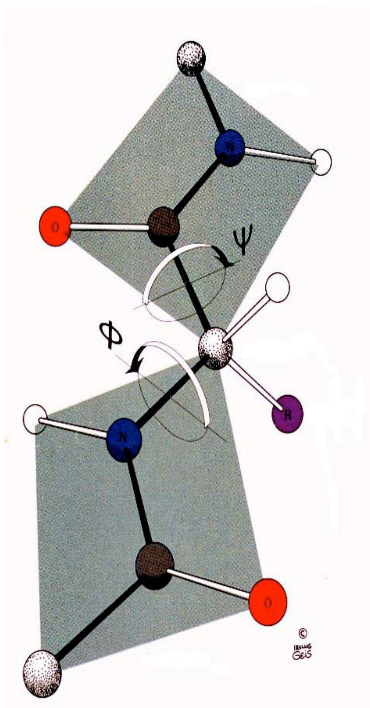


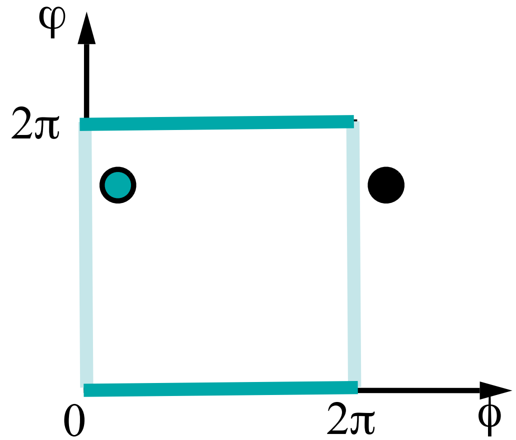
robot

Path is 1D curve

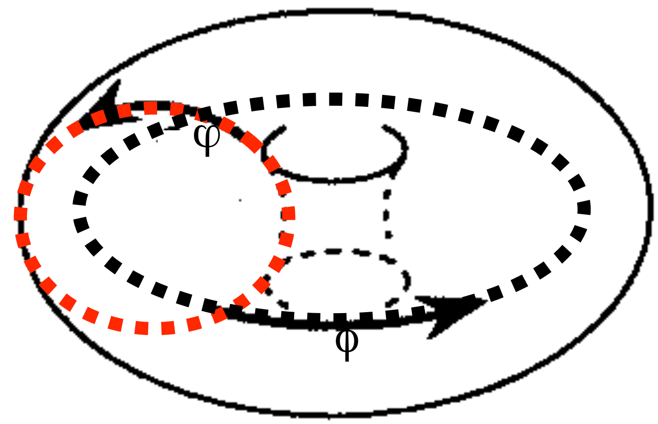
Topology of the configuration pace

- The topology of *C* is usually **not** that of a Cartesian space R^n.



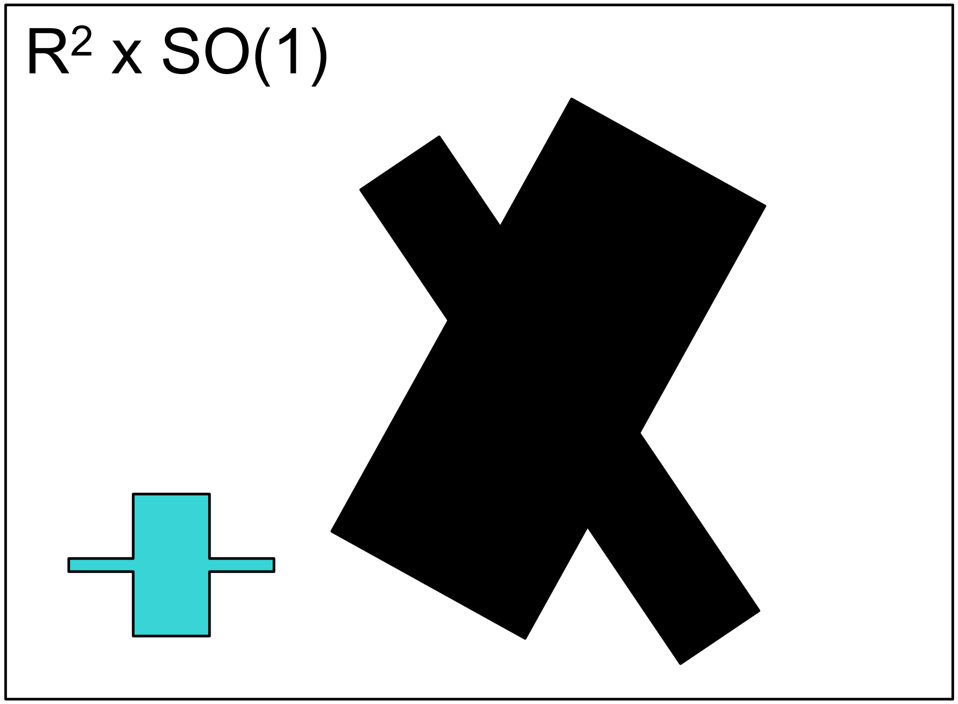


$C = S^1 \times S^1$

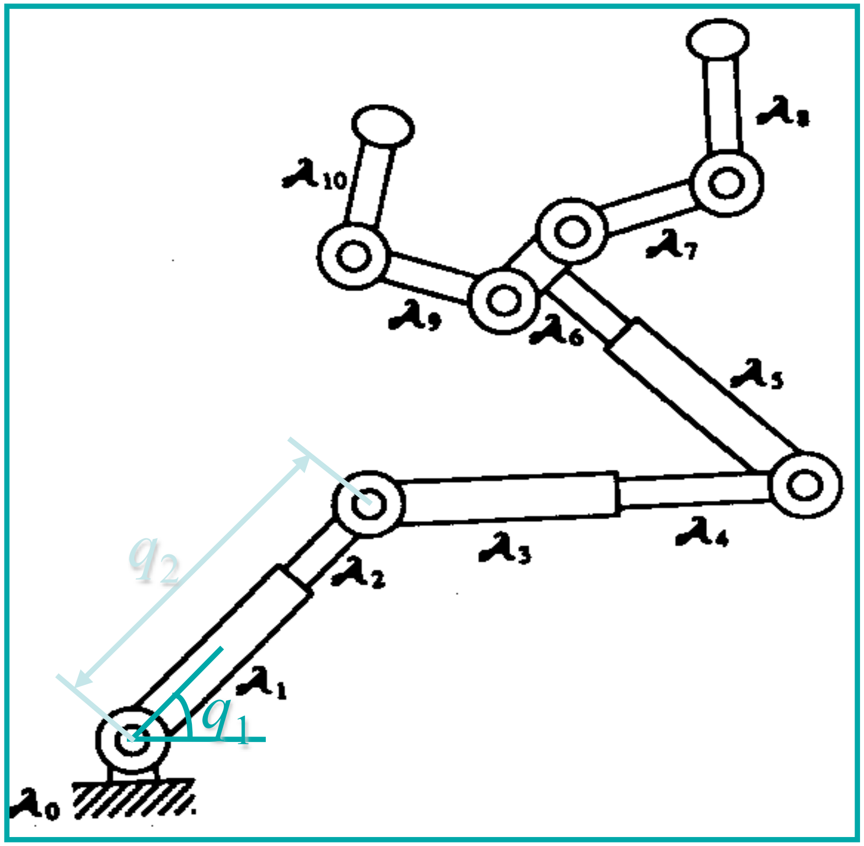


Example: rigid robot in 2-D workspace

- dim of configuration space = **???**
- Topology**???**



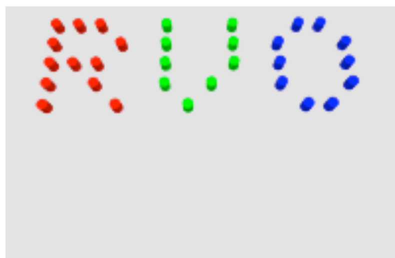
Example: articulated robot



An articulated object is a set of rigid bodies connected at the joints.

- Number of dofs?
- What is the topology?

Example: Multiple robots

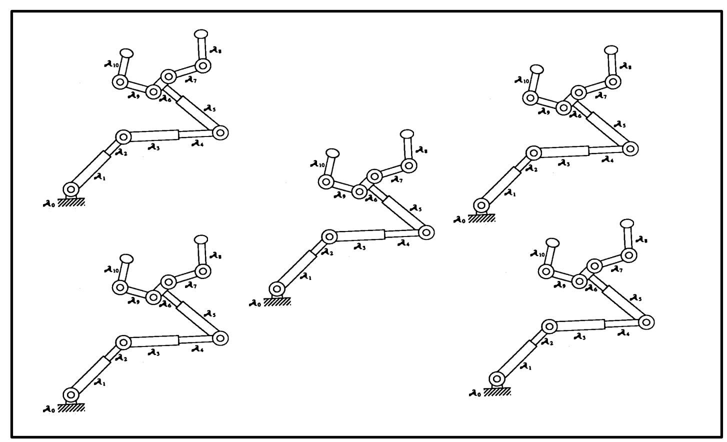


ROV, GAMMA group, UNC

- Given n robots in 2-D
- What are the possible representations?

J.J. Kuffner et al.

- What is the number of dofs?



5 articulated robots

Metric in configuration space

- A **metric** or **distance** function d in a configuration space C is a function

$$d : (q, q') \in C^2 ® \; d(q, q') \geq 0$$

such that

  - $d(q, q') = 0$ if and only if $q = q'$,
  - $d(q, q') = d(q', q)$,
  - $d(q, q') \leq d(q, q'') + d(q'', q')$.

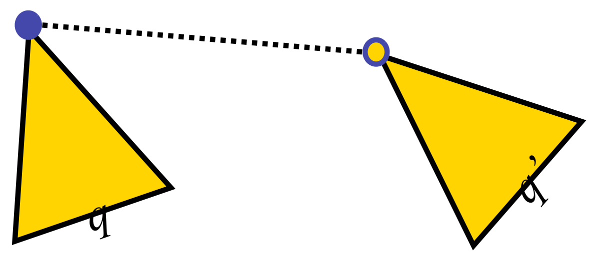
aka. Triangle inequality

Example

- Robot A and a point x on A
- $x(q)$: position of x in the workspace when A is at configuration q
- A distance d in C is defined by

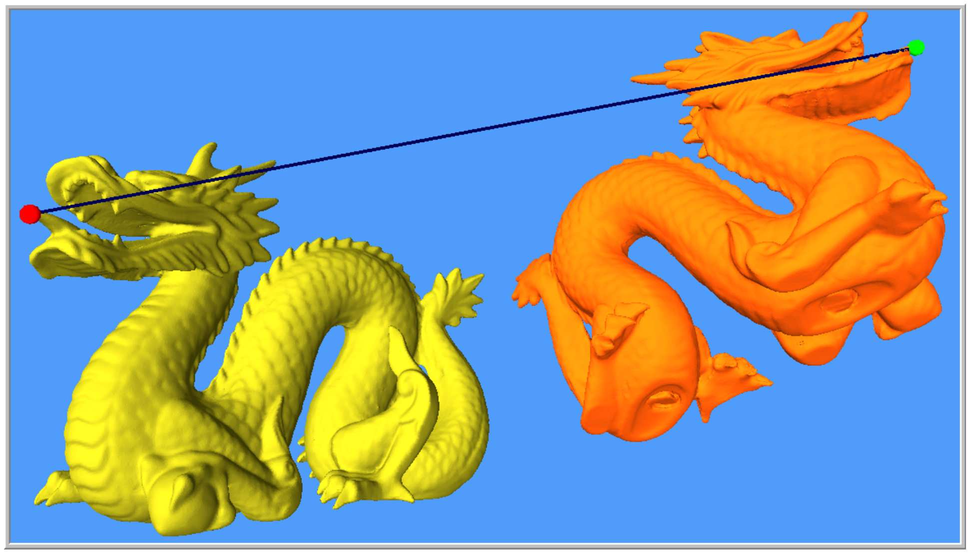
$$d(q, q') = \max_{x \in A} \| x(q) - x(q') \|$$

where $||x - y||$ denotes the Euclidean distance between points x and y in the workspace.



Examples

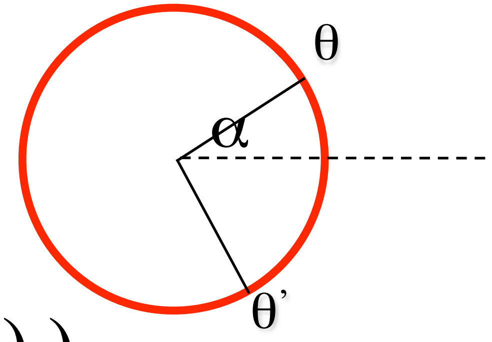
- Maximum distance between the object in two configurations



C-Dist, Zhang et al. SPM 2007

Examples in $R^2 \times S^1$

- Consider $R^2 \times S^1$
  - $q = (x, y, \theta)$, $q' = (x', y', \theta')$ with $\theta, \theta' \in [0, 2\pi)$
    $\alpha = \min\{ |\theta - \theta'|, 2\pi - |\theta - \theta'| \}$



- $d(q, q') = \mathrm{sqrt}((x-x')^2 + (y-y')^2 + \alpha^2))$

- $d(q, q') = \mathrm{sqrt}((x-x')^2 + (y-y')^2 + (\alpha r)^2)$, where r is the maximal distance between a point on the robot and the reference point

Break

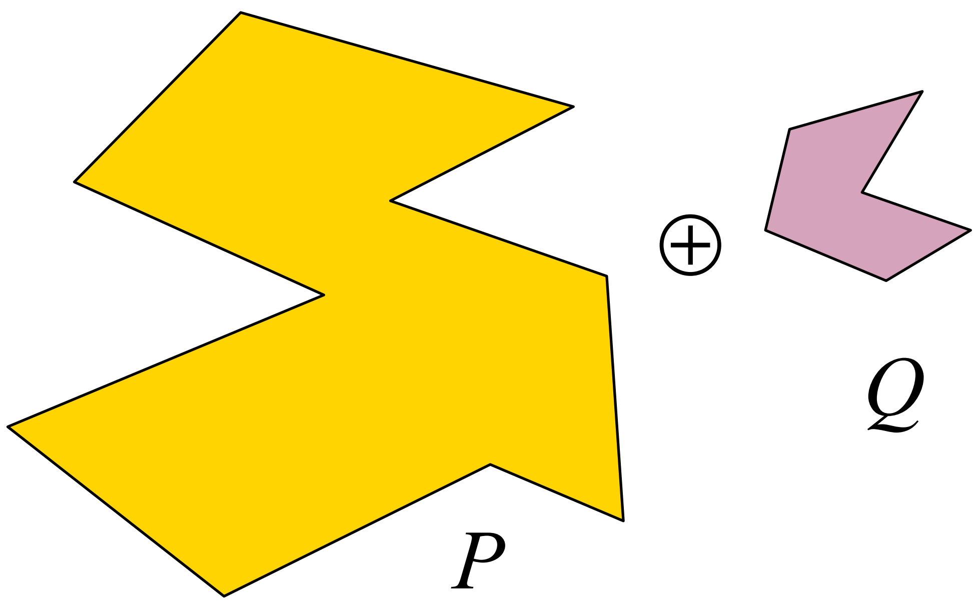
- 10 min break

Convert Workspace to C-Space

- How?

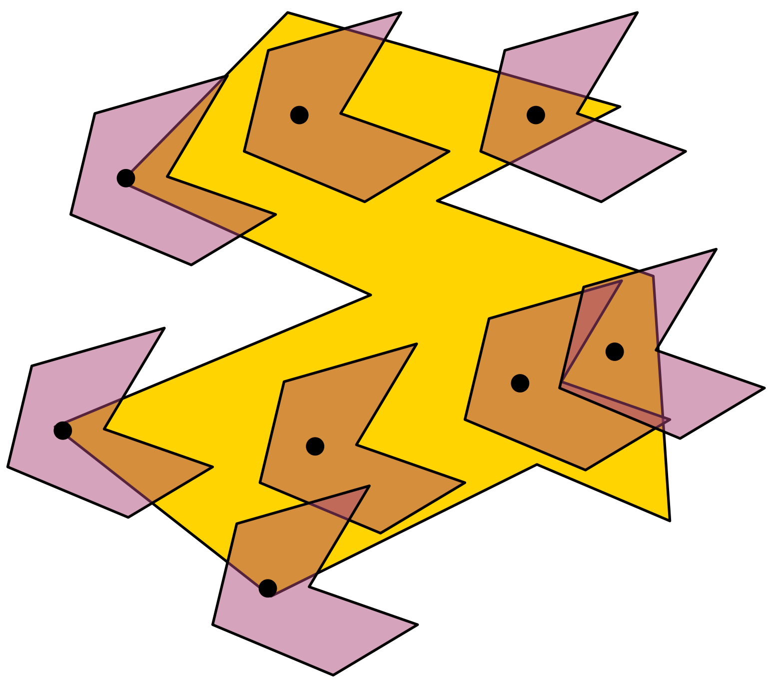
Minkowski Sum

- Minkowski sum
  - $P \oplus Q = \{p+q \mid p \in P, q \in Q\}$



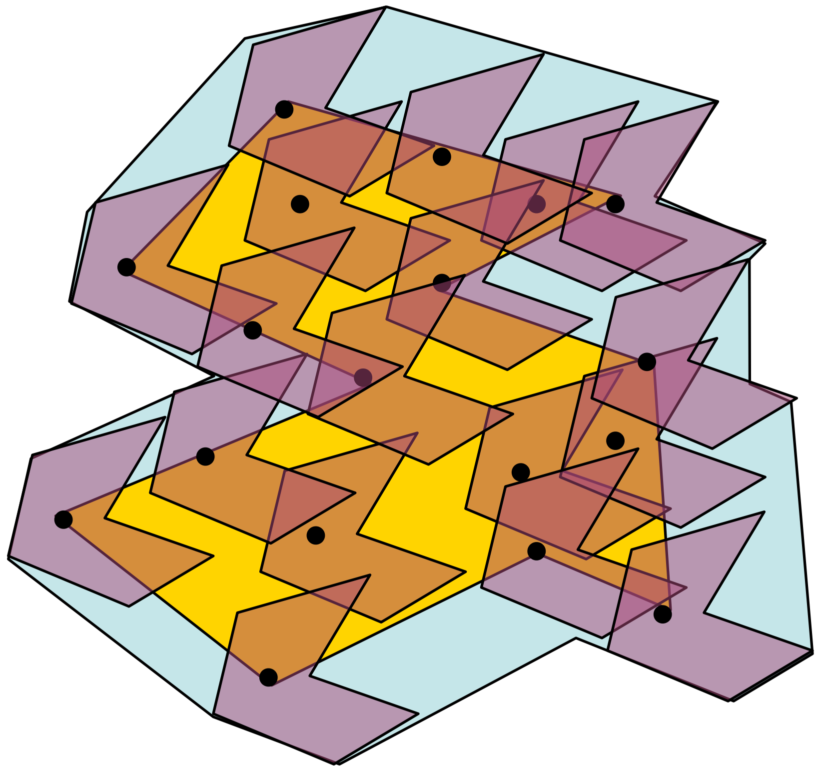
Minkowski Sum

- Minkowski sum
  - $P \oplus Q = \{p+q \mid p \in P,\ q \in Q\}$



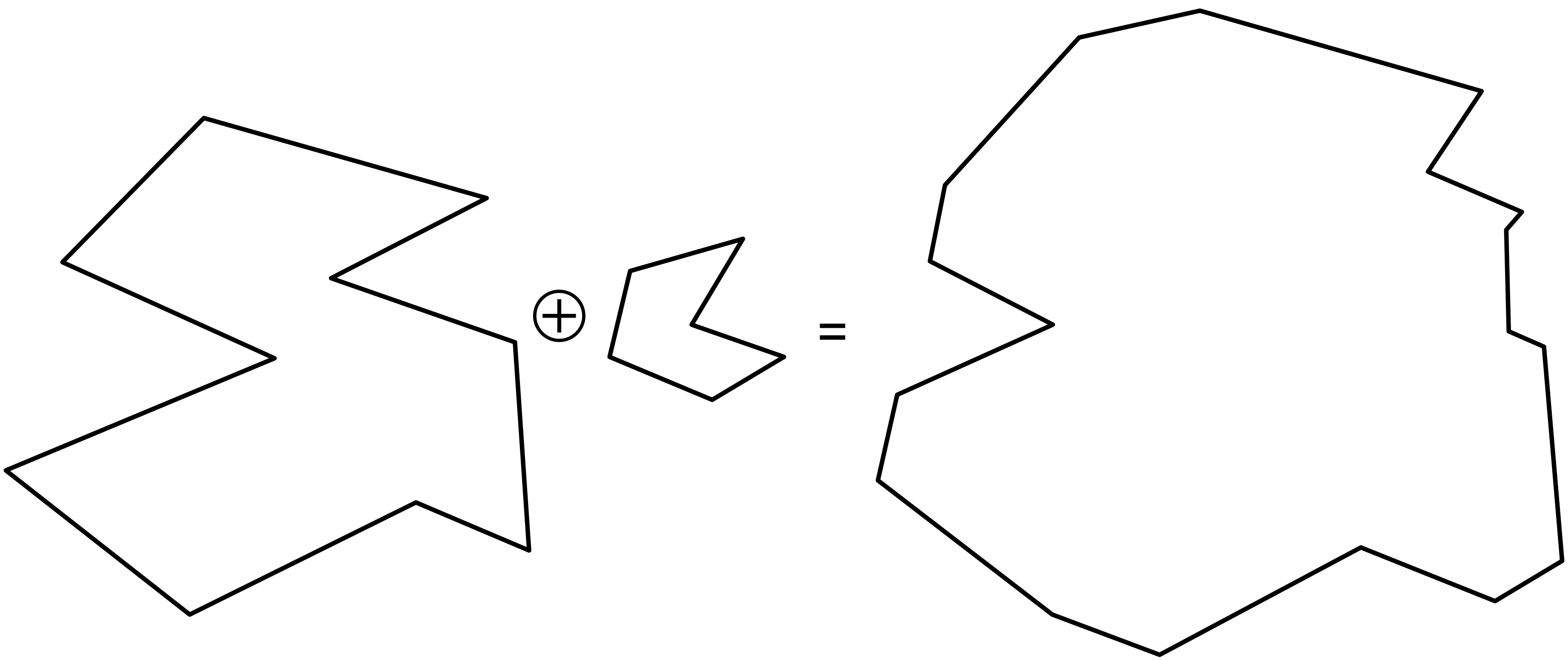
Minkowski Sum

- Minkowski sum
  - $P \oplus Q = \{p+q \mid p \in P,\ q \in Q\}$



Minkowski Sum

- Minkowski sum
  - $P \oplus Q = \{p+q \mid p \in P, q \in Q\}$



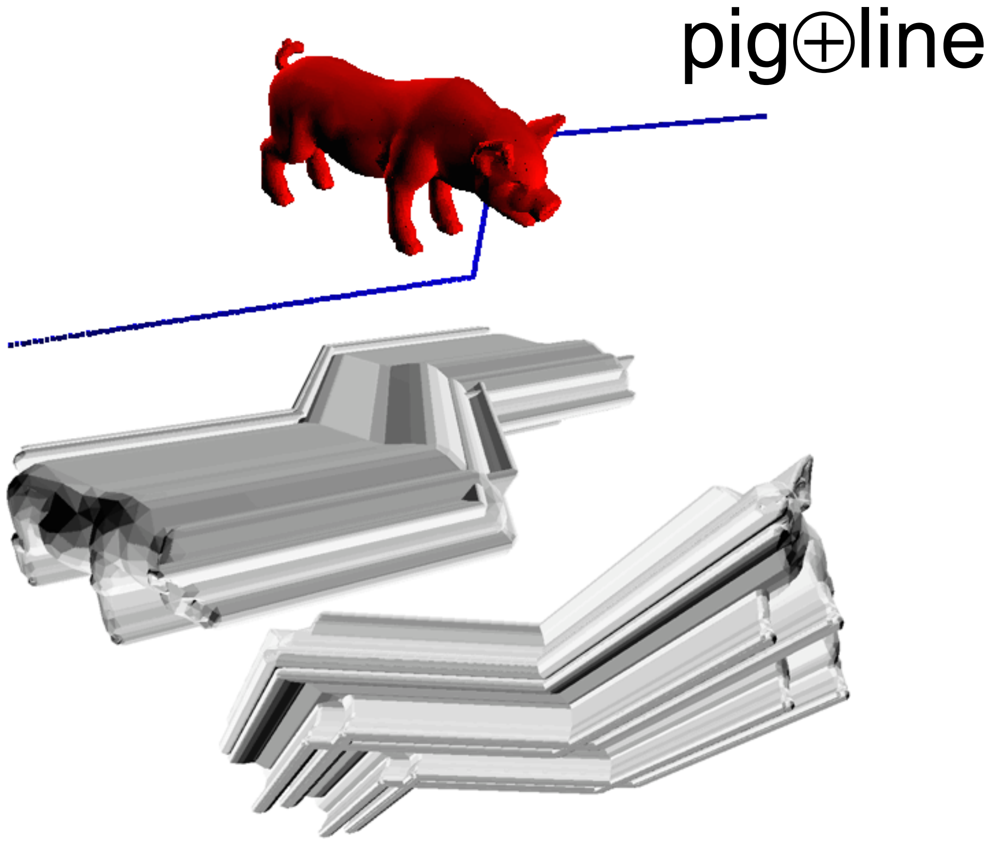
Applications

- Dilation/Offset

kids⊕cube

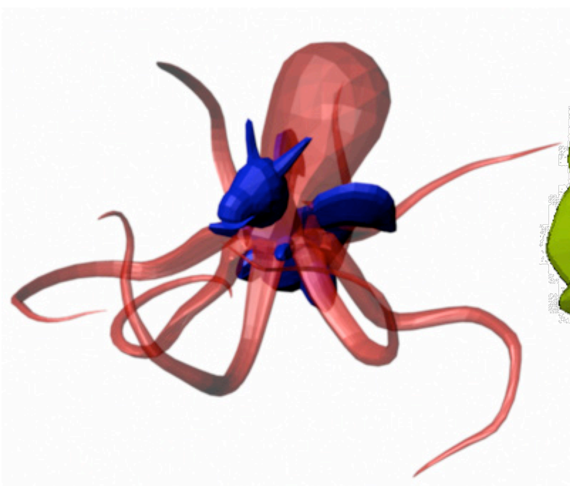
Applications

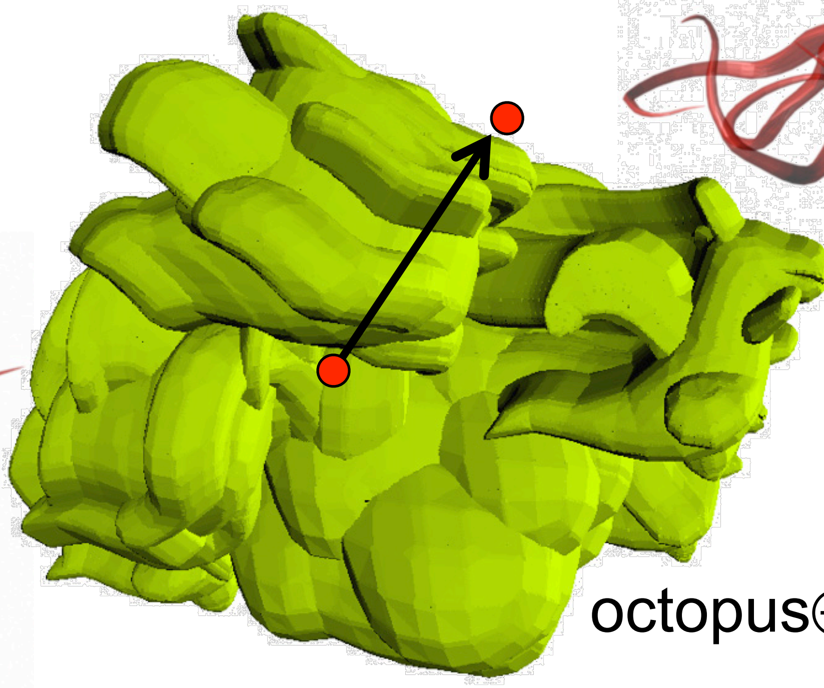
- Sweep volume

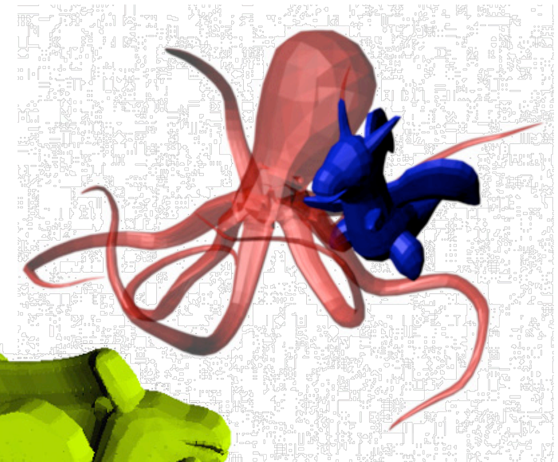


Applications

- Penetration depth estimation



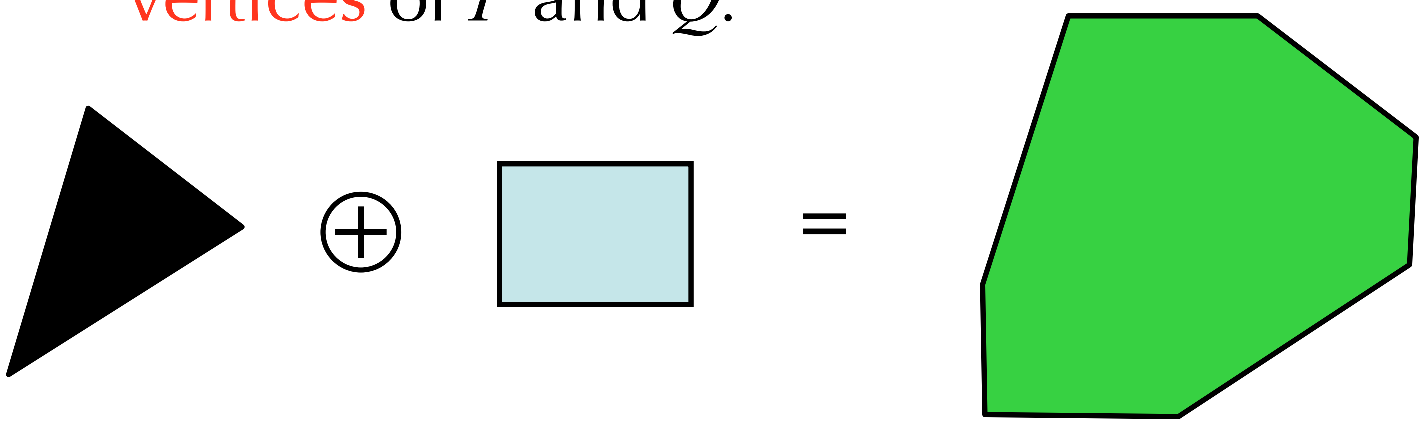




octopus⊕(-dragon)

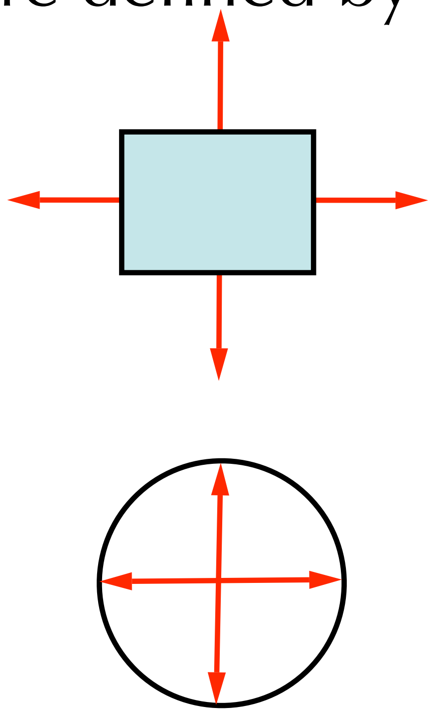
Minkowski sum of convex polygons

- The Minkowski sum of two convex polygons P and Q of m and n vertices respectively is a convex polygon $P + Q$ of $m + n$ vertices.
  - The vertices of $P + Q$ are the "sums" of vertices of P and Q.



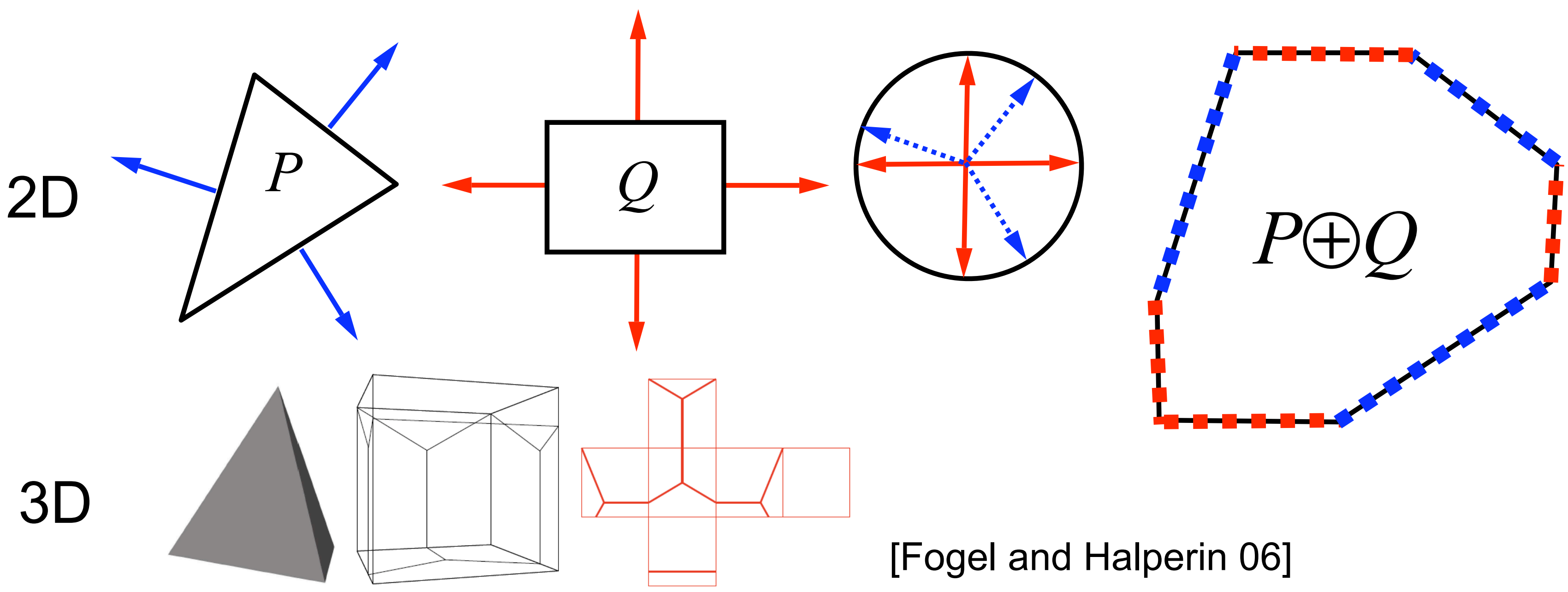
Gauss Map

- Gauss map of a convex polygon
  - Edge → point on the circle defined by the normal
  - Vertex → arc defined by its adjacent edges



Compute Minkowski Sum

- Convex object
  - Use Gaussian map
  - Compute convex hull of Point-based Minkowski sum (slower)

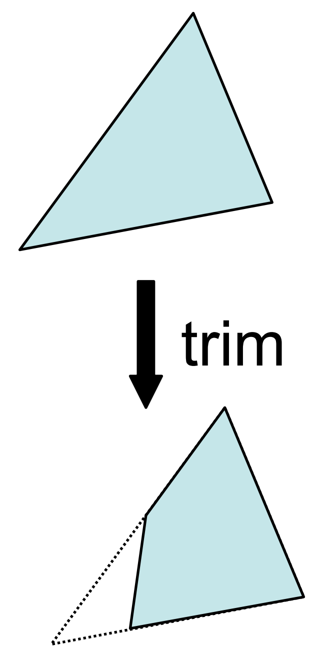


Complexity

- Convex-convex MK-sum
  - $O(n+m)$
- Convex-Nonconvex MK-sum
  - $O(nm)$
- Nonconvex-Nonconvex MK-sum
  - $O(n^2m^2)$

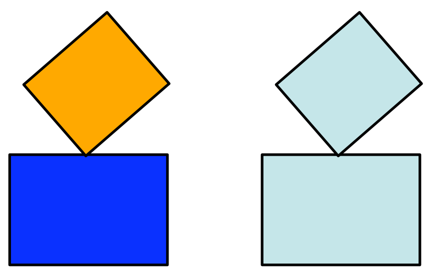
Compute 3D Minkowski Sum

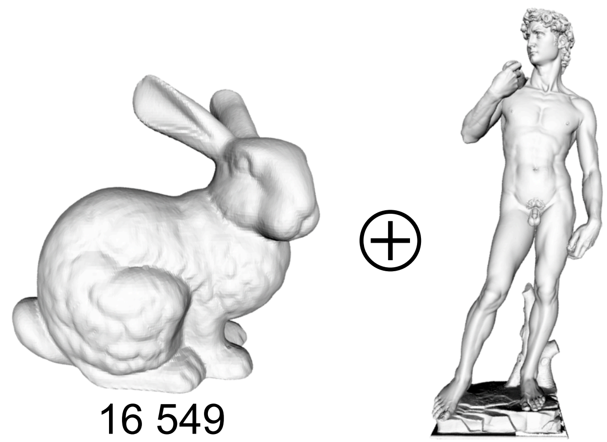
- Non-convex object
  - **Divide-n-conquer** [Evans et al. 92, Varadhan,Manocha 04]
    - Decomposition, e.g., *convex decomposition*
    - Pairwise Minkowski sums
    - Union of pairwise sums
  - **Trimming**
    - Compute a superset of Minkowski sum boundary [Kaul and Rossignac 91, Ghosh 93]
    - Trimming the superset
  - Several others…



Compute Minkowski Sum

- Difficulties in implementing **Divide-n-conquer**
  - Degenerate cases in union
  - Errors accumulate





16 549 convex components

85 132 convex components

>1.4 billion pairwise Minkowski sums

Problem

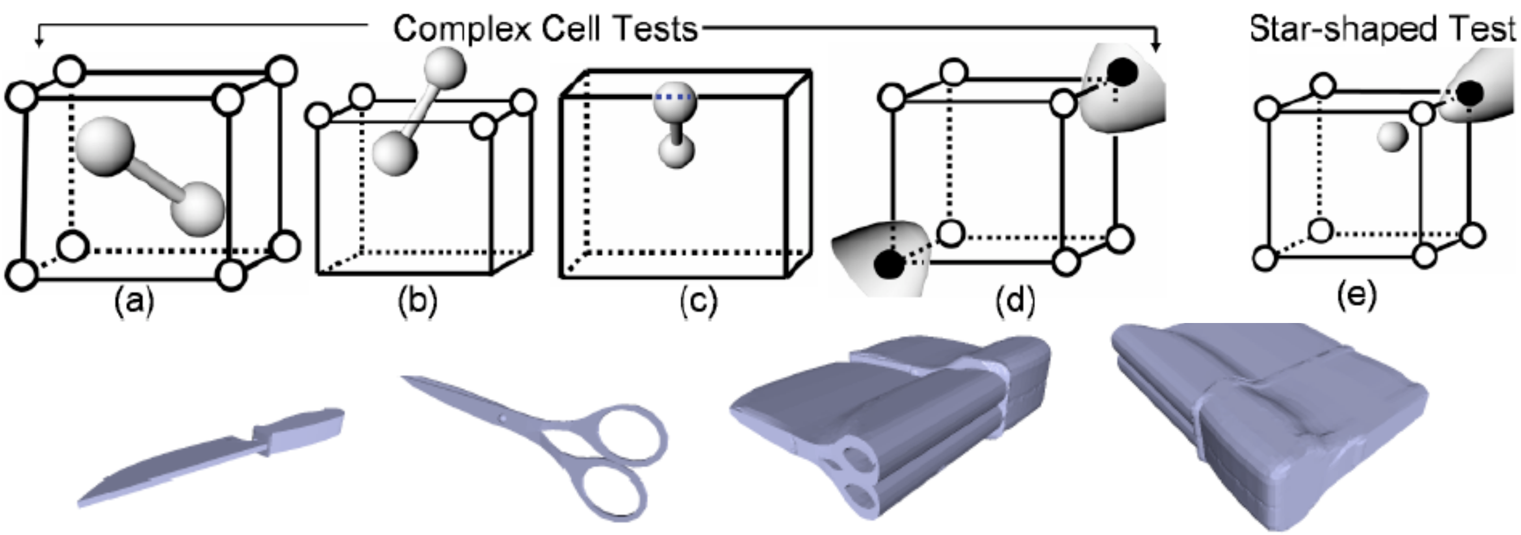
- Despite the importance of Minkowski sum, no practical implementation for 3D models can be found in public domain

- Why?

- What can we do about it?

Compute Minkowski Sum

- Approximate approach [Varadhan,Manocha PG04]
  - Avoid the union step
  - Using marching cube

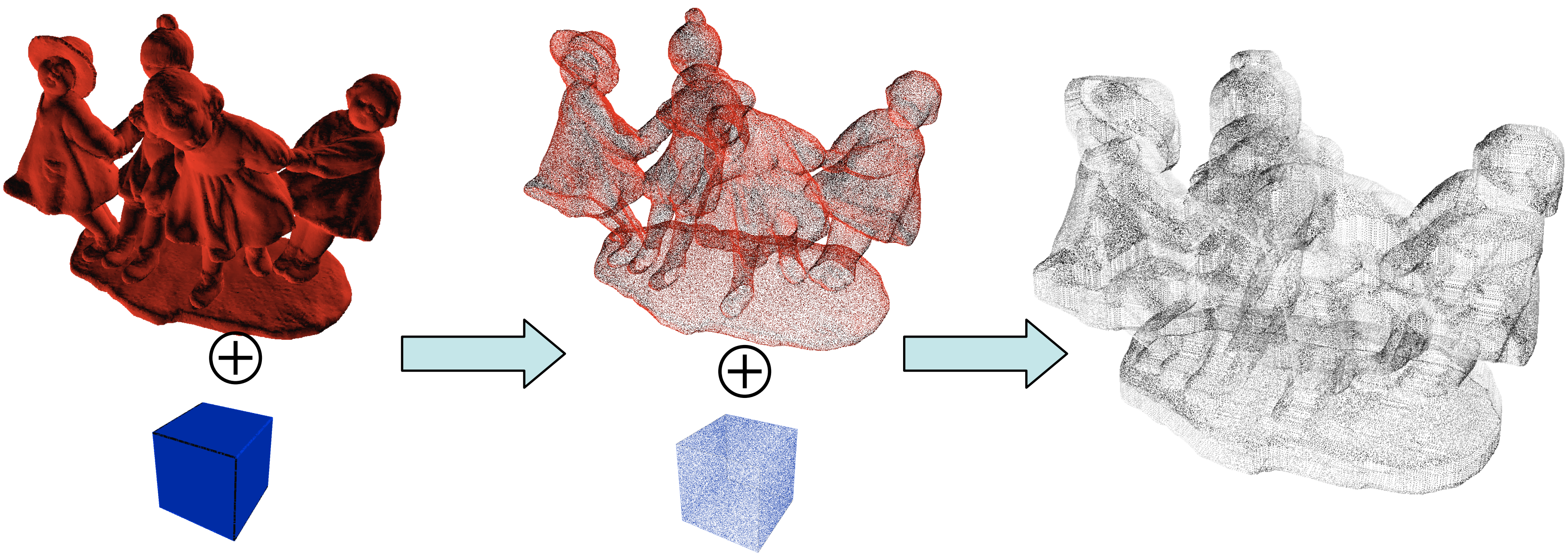


Images from [Varadhan,Manocha PG04]

Point-Based Minkowski Sum

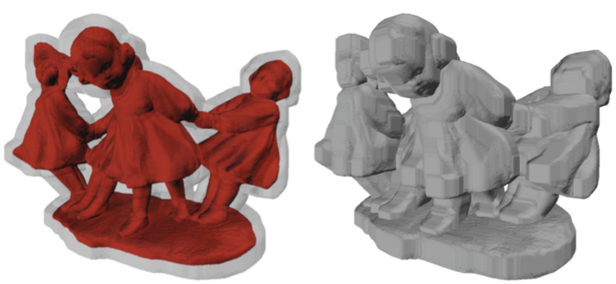
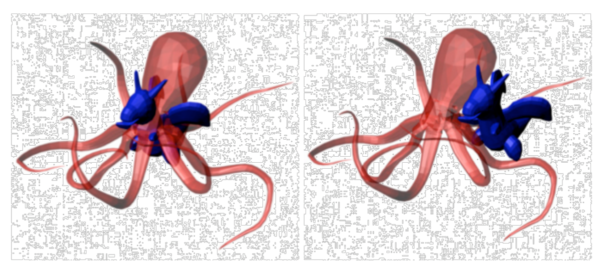
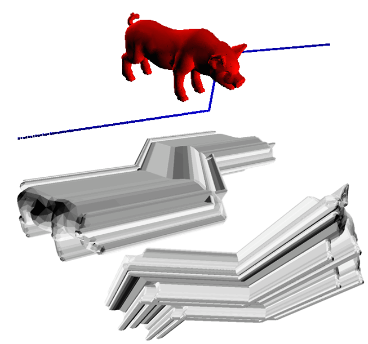
Lien, PG 2007

- Represent Minkowski sum boundary using points only
  - Sample points from the surface of P and Q
  - Compute the Minkowski sum of the points
  - Extract boundary points



Point-Based Minkowski Sum

- Benefits
  - We give up exactness to gain simplicity and efficiency
    - Avoid convex decomposition
    - Avoid computing unions
  - Still provide similar functionality as “mesh-based” Minkowski sum --- e.g., offset, sweep, penetration estimation, …

Point-Based Minkowski Sum

- Let S_P and S_Q be points sampled from δP and δQ
- Let $S_+ = S_P \oplus S_Q$
- Let $S = \delta(P \oplus Q) \cap S_+$
- Require: S is a d-cover of $\delta(P \oplus Q)$
  - i.e., any point of $\delta(P \oplus Q)$ has a point in S within distance d

We need two sub-routines:

1. A method to create S_P and S_Q so S is a d-cover of $\delta(P \oplus Q)$
2. A method to S = Filter(S_+)

Sample Points

- **Goal**: create S_P and S_Q so that S is a d-cover of $\delta(P \oplus Q)$

Theorem

If S_P and S_Q are d-cover of P and Q, then $S_+ = S_P \oplus S_Q$ must contain a d-cover of $\delta(P \oplus Q)$

Facets of Minkowski sum can only come from

- Facets of P
- Facets of Q
- Facets created by one edge of P and one edge of Q

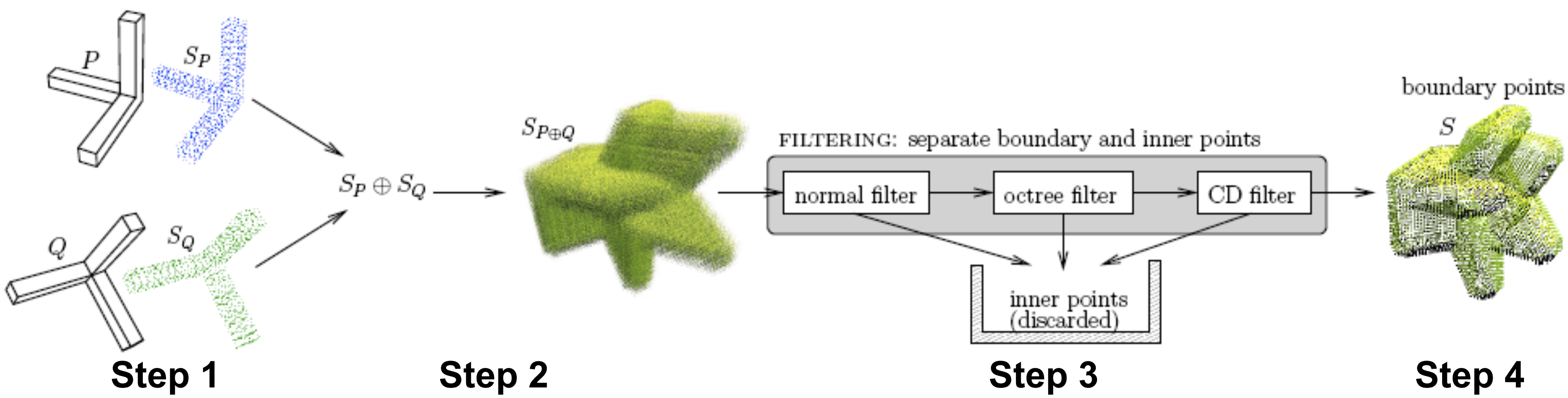
Extract Boundary Points

- **Goal**: $S = \delta(P \oplus Q) \cap S_+$

- We propose 3 filters
  - **Collision detection filter**
    - Slow but complete
  - **Normal filter** (Based on ideas in [Kaul and Rossignac 91])
    - Fast but incomplete
  - **Octree filter**
    - Fast but incomplete
  - These filters can be combined

Put It All Together

1. Sample S_P and S_Q as d-cover from ∂P and ∂Q
2. Compute $S_+ = S_P \oplus S_Q$
3. $S =$ filter(S_+)
    1. Normal filter
    2. Octree filter
    3. Collision Detection (CD) filter
4. S is a d-cover of $\partial(P \oplus Q)$



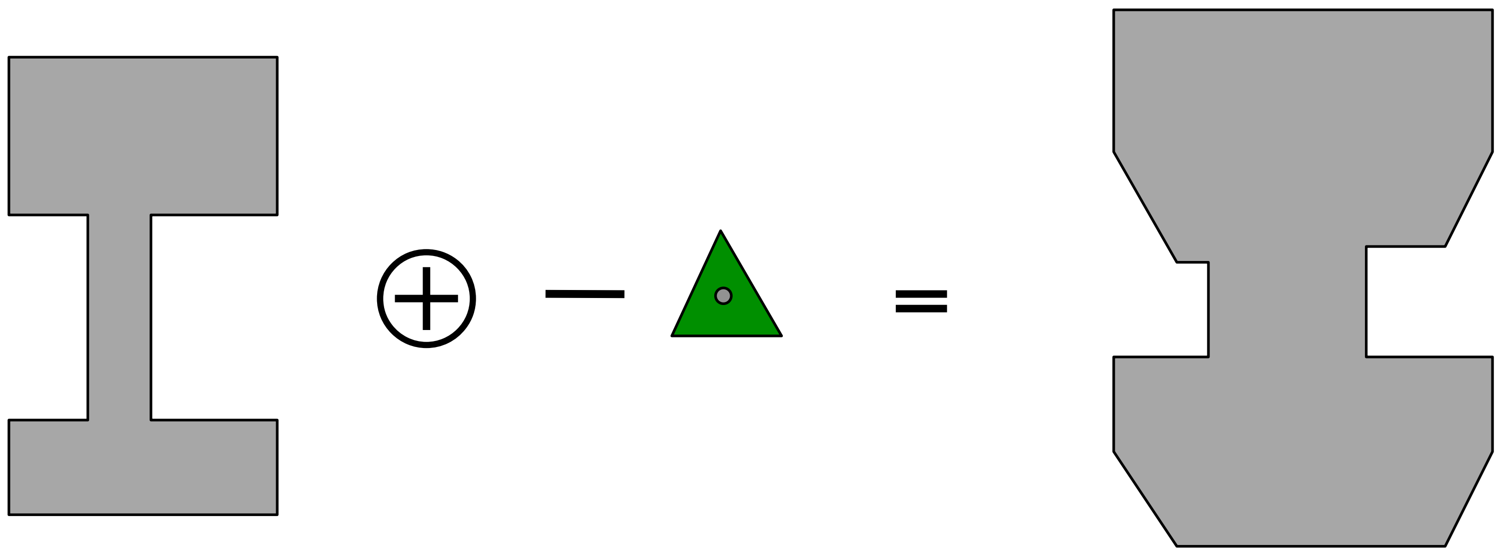
Back to Motion Planning

Minkowski sum allows us to solve problems with translational robots

C-Space Obstacle

C-obstacle is $\mathcal{O} \oplus -\mathcal{R}$

Classic result by Lozano-Perez and Wesley 1979

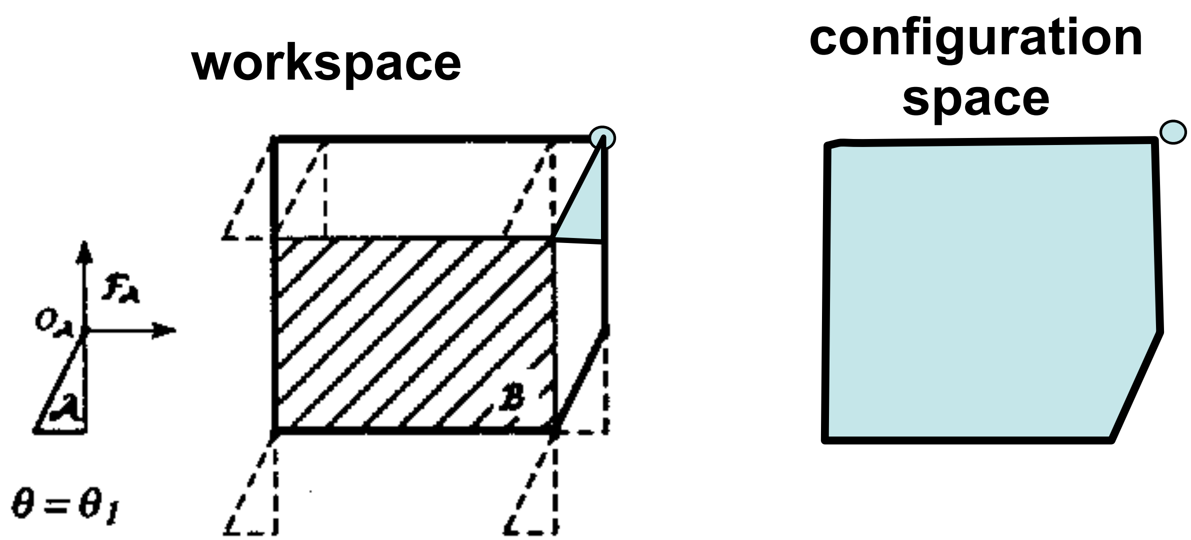


Obstacle $\mathcal{O}$

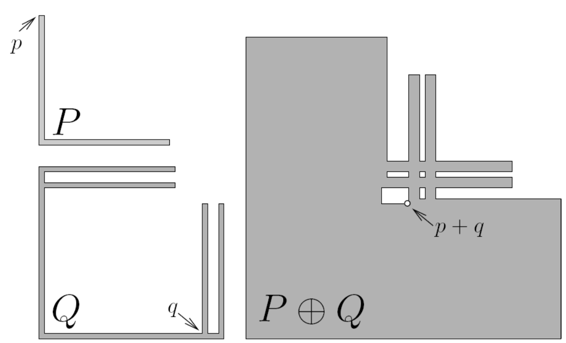
Robot $\mathcal{R}$

C-obstacle $\mathcal{O} \oplus -\mathcal{R}$

Polygonal robot translating in 2-D workspace

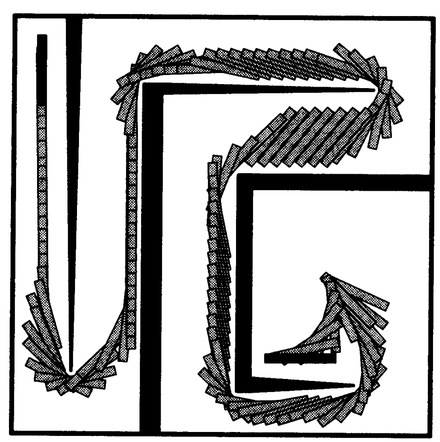


The complexity of the Minkowski sum is $O(n^2)$ in 2D

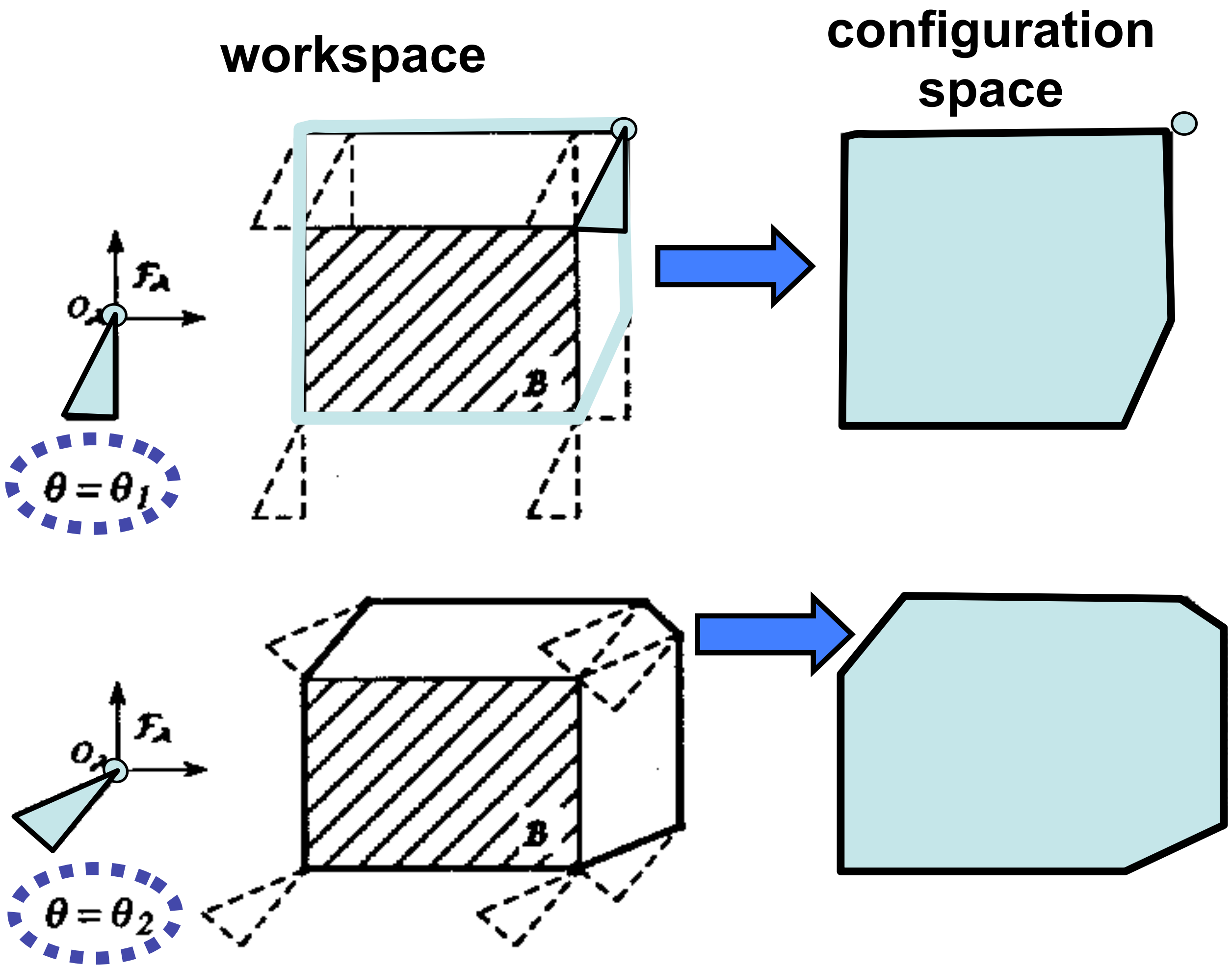


Robot with Rotations

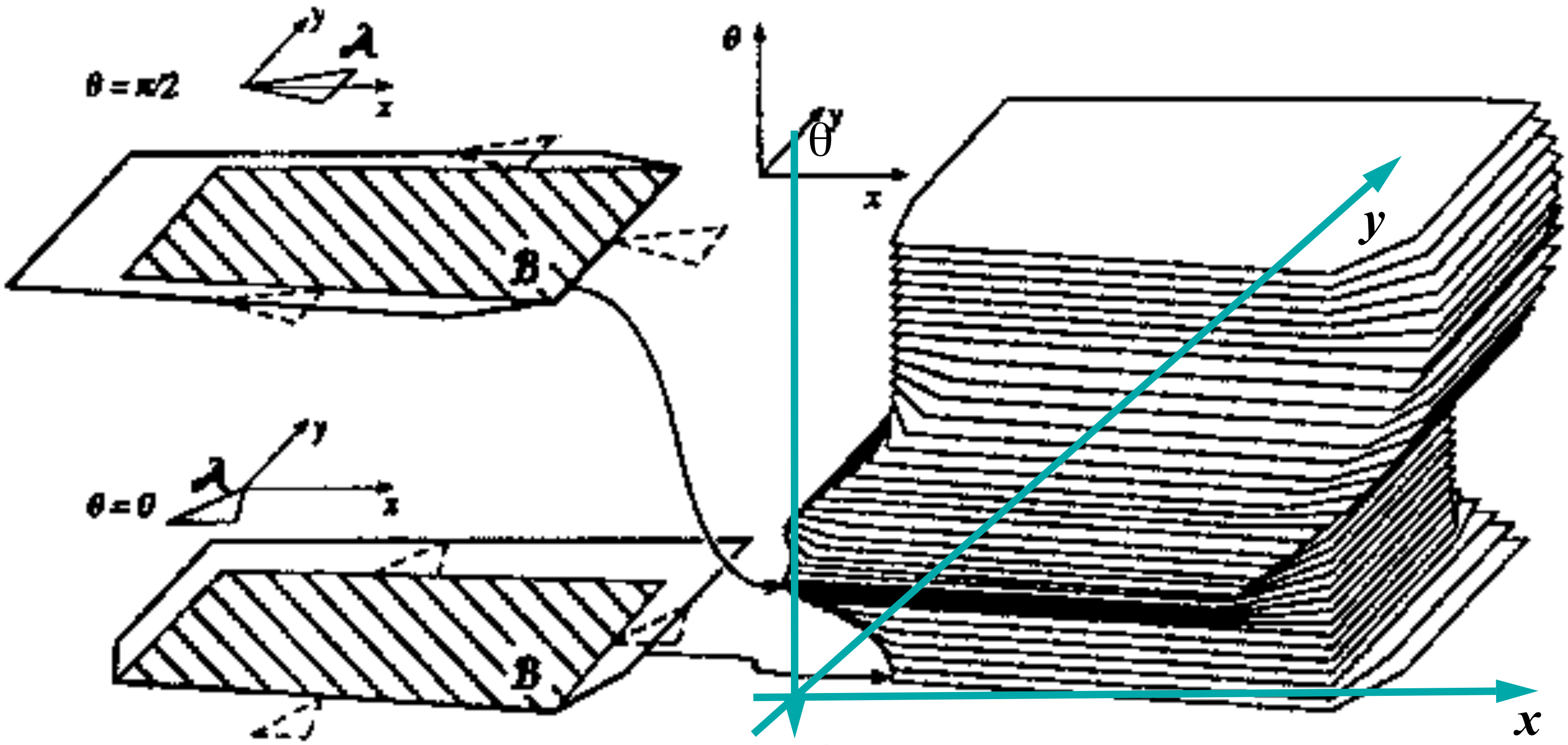
- If a robot is allowed rotation in addition to translation in 2D then it has 3 DOF
- The configuration space is 3D: (x,y,φ) where φ is in the range [0:360)



Polygonal robot translating & rotating in 2-D workspace

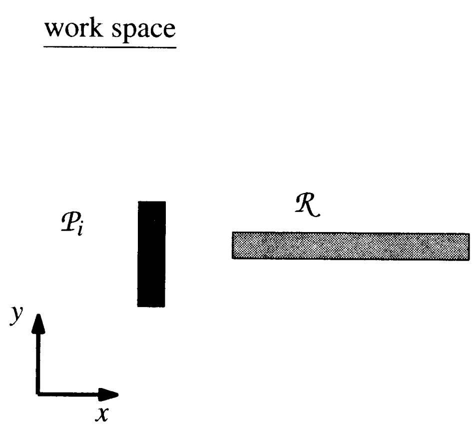


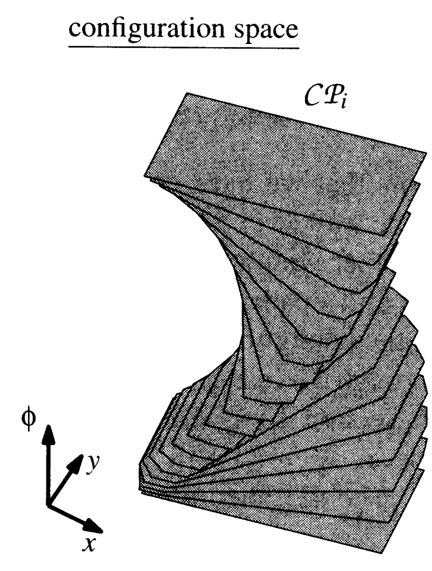
Polygonal robot translating & rotating in 2-D workspace



Mapping to C-Space

- The obstacles map to “twisted pillars” in C-Space
- They are no longer polygonal but are composed of curved faces and edges



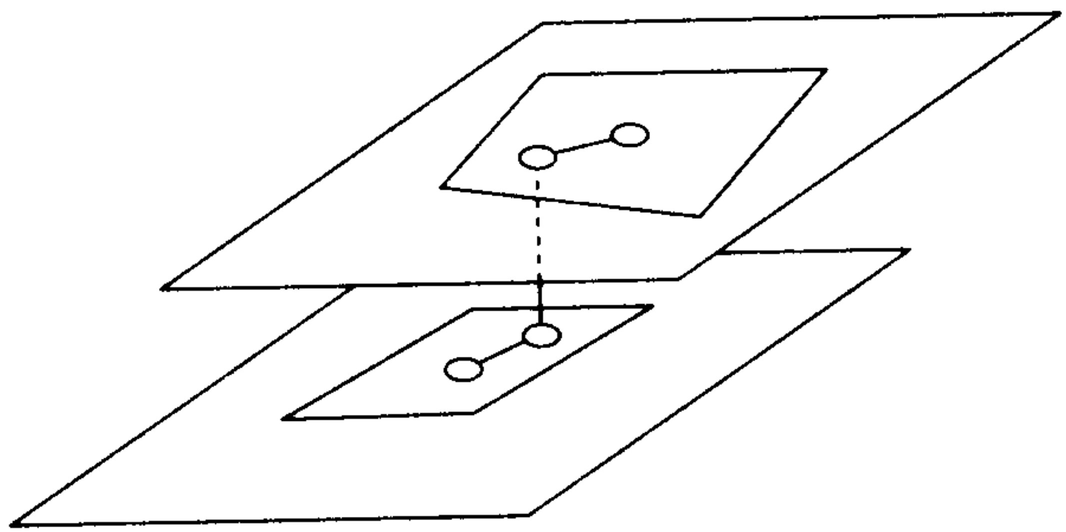


Computing Free Space

- Exact cell decomposition in 3D is really hard
- Compute z: a finite number of slices for discrete angular values
- Each slice will be the representation of the free space for a purely translational problem
- Robot will either move within a slice (translating) or between slices (rotating)

Computing the Road Map

- Each slice has a road map like before
- But how do we move between slices?



Moving Between Slices

- To find graph edges between two slices:

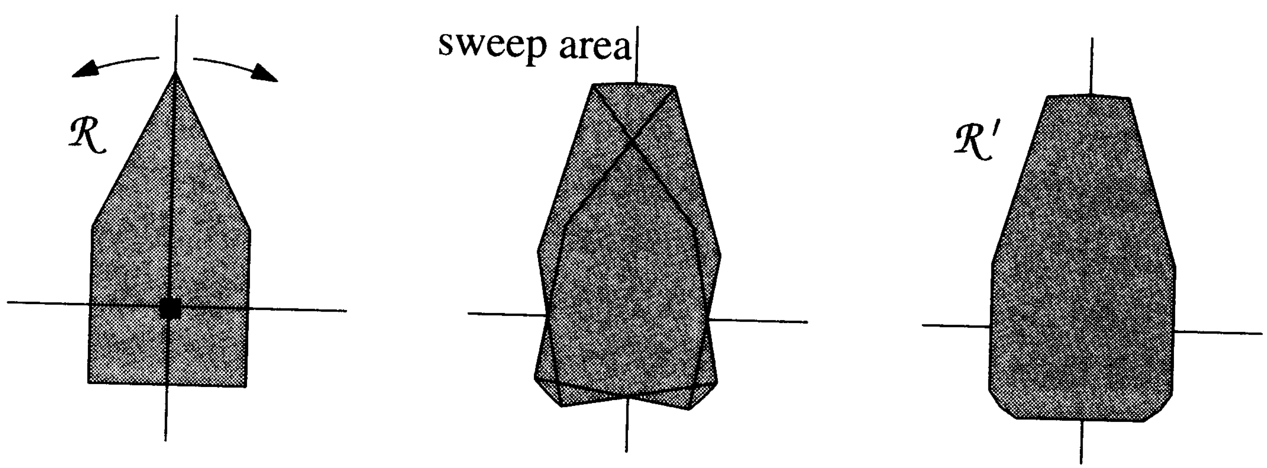
1. compute the overlay of the trapezoidal maps of the two slices to get all pairs of trapezoids that intersect (one trapezoid from each slice)
2. for each pair
3. find a point (x,y) in their intersection and make one new vertex in each slice at this (x,y)
4. connect the two new vertices
5. connect the each of the two new vertices to the vertex at the center of their respective trapezoids

Slice Problems (Aliasing)

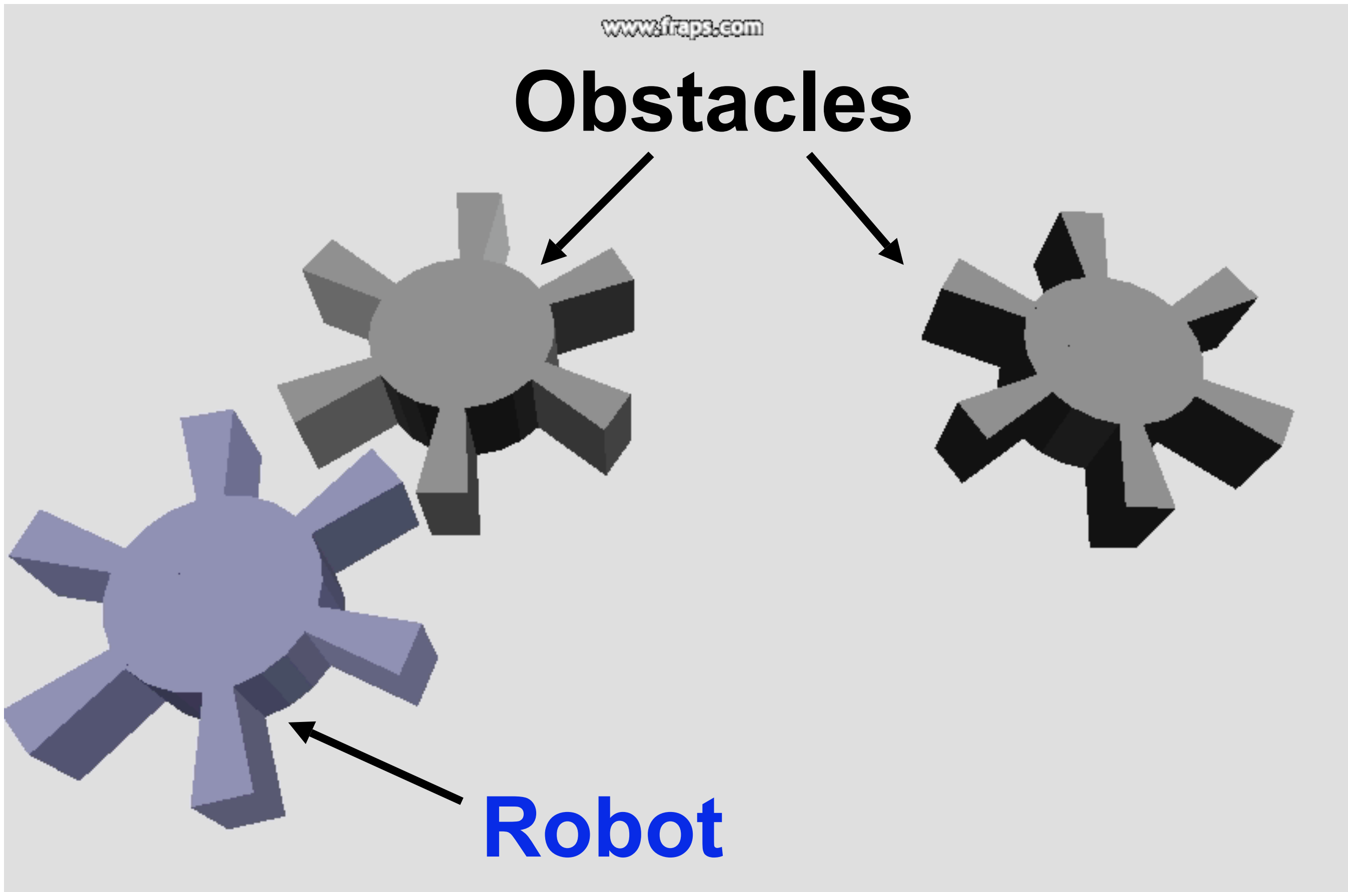
- Start and/or goal position may be in the free space whereas the start/goal position in the nearest slice may not
- May have an undetected collision when moving between slices
- Increasing the number of slices reduces problems but does not solve them

Dealing with the Problems

- Enlarge the robot by sweeping out some additional area (180^o/z) in each direction
- Introduces yet another way to incorrectly determine that there is no path

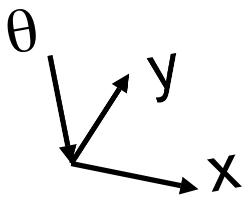


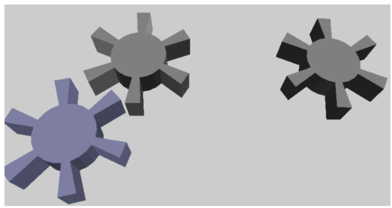
2D Translation and Rotation

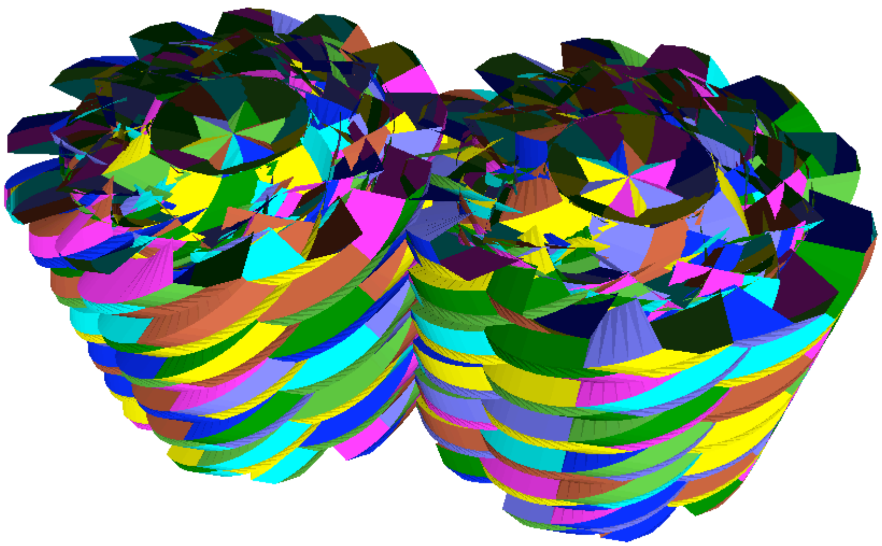


From Gokul and Varadhan at UNC

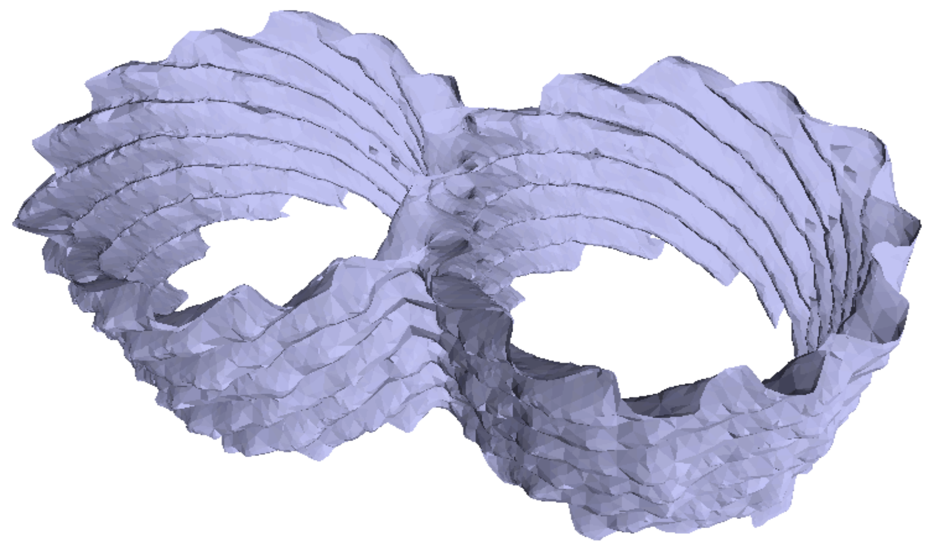
Free Space Approximation







3,929
contact surfaces

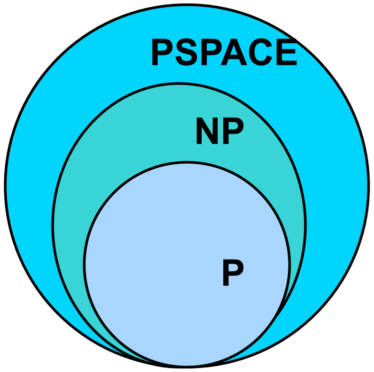


Free space boundary
approximation

Summary

- **Deterministic Roadmap Methods**
  - **Visibility graph (2D)**
  - **Retraction approach (2, 3D)**
  - **Exact cell decomposition (2&3D)**
    - **convex decomposition**
    - **trapezoidal decomposition**
  - **Approximate decomposition (2,3,4 D)**

The Complexity of



General motion planning problem is
PSPACE-hard [Reif 79, Hopcroft et al. 84 & 86]
PSPACE-complete [Canny 87]

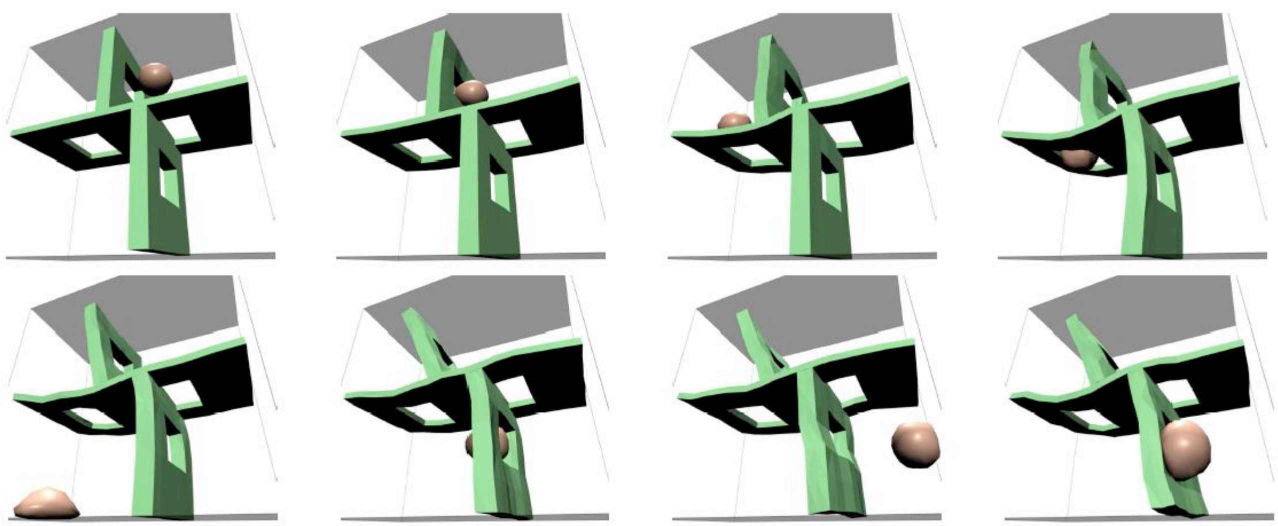
The best deterministic algorithm known has running time that is exponential in the dimension of the robot's C-space [Canny 86]

• C-space has high dimension - 6D for rigid body in 3-space
• simple obstacles have complex C-obstacles ⇒ impractical to compute explicit representation of freespace for more than 4 or 5 dof

So … attention has turned to *randomized algorithms*

Hard Motion Planning Problems

- What if we can to consider other kinematic constraints or the dynamics of the robot (moving object)?
- The problem is even harder



Next Week

- Probabilistic Roadmap Methods
  - A set of methods that can solve practical motion planning problems, including those with kinematic or dynamic constraints