CS164: Collision Detection, Motion Planning

[Slides: Latombe]

Leonidas Guibas
Computer Science Dept.
Stanford University

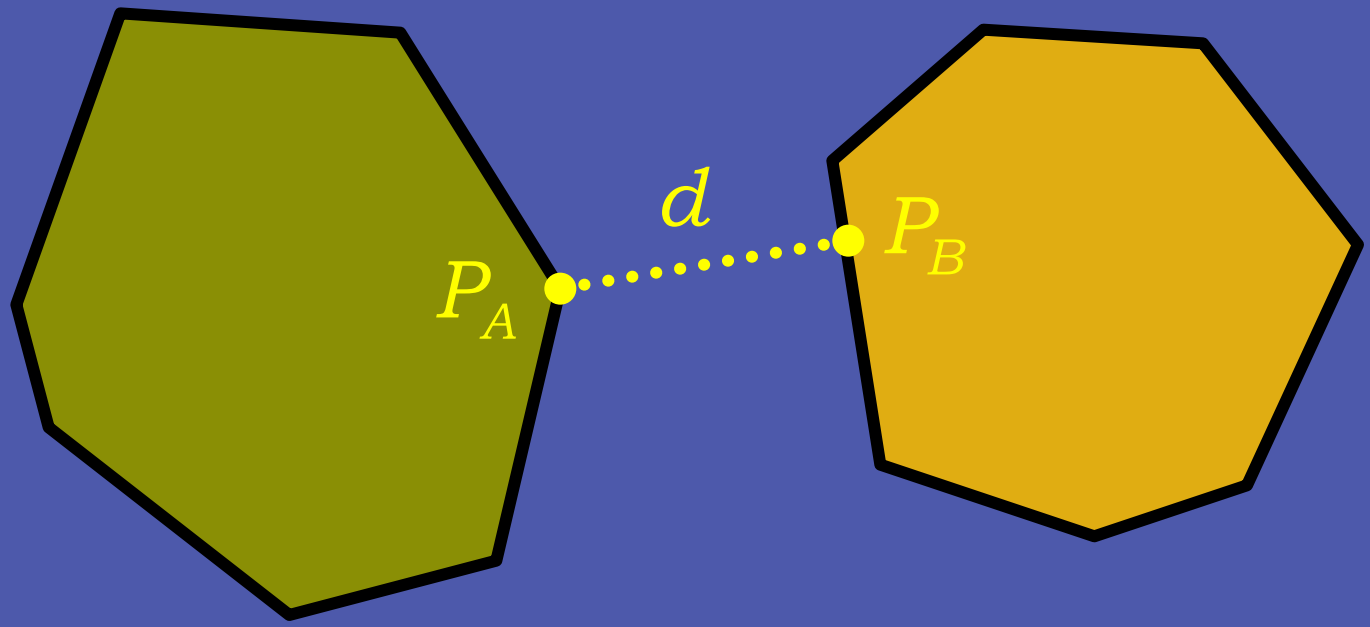
More Collision Detection

The Gilbert-Johnson-Keerthi (GJK) Algorithm

- Like Lin-Canny, GJK is an algorithm for maintaining the closest pair of points (and therefore the distance) between two moving convex objects
- First, a simplified version of the algorithm
  - One object is a point at the origin, other is static
  - Example illustrating algorithm
- The distance subalgorithm
- GJK for two static objects
  - One no longer necessarily a point at the origin
- GJK for moving objects

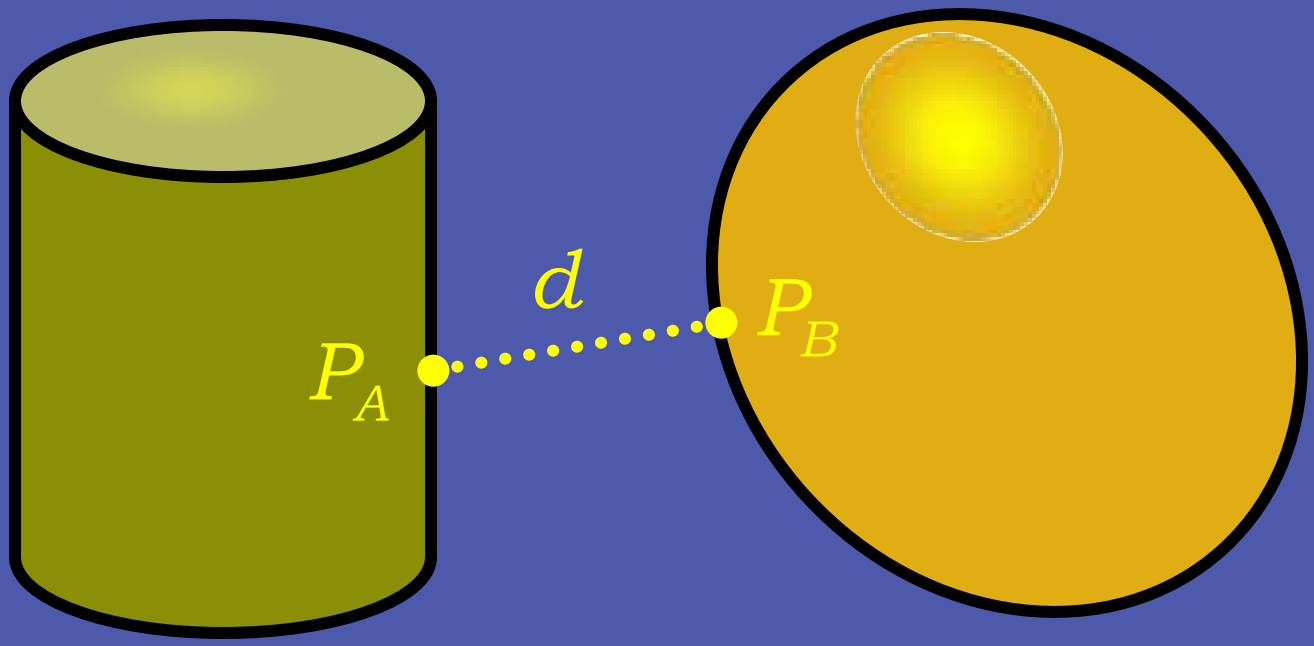
GJK solves proximity queries

- Given two convex polyhedra
  - GJK computes distance *d*
  - GJK can also return closest pair of points P_A, P_B

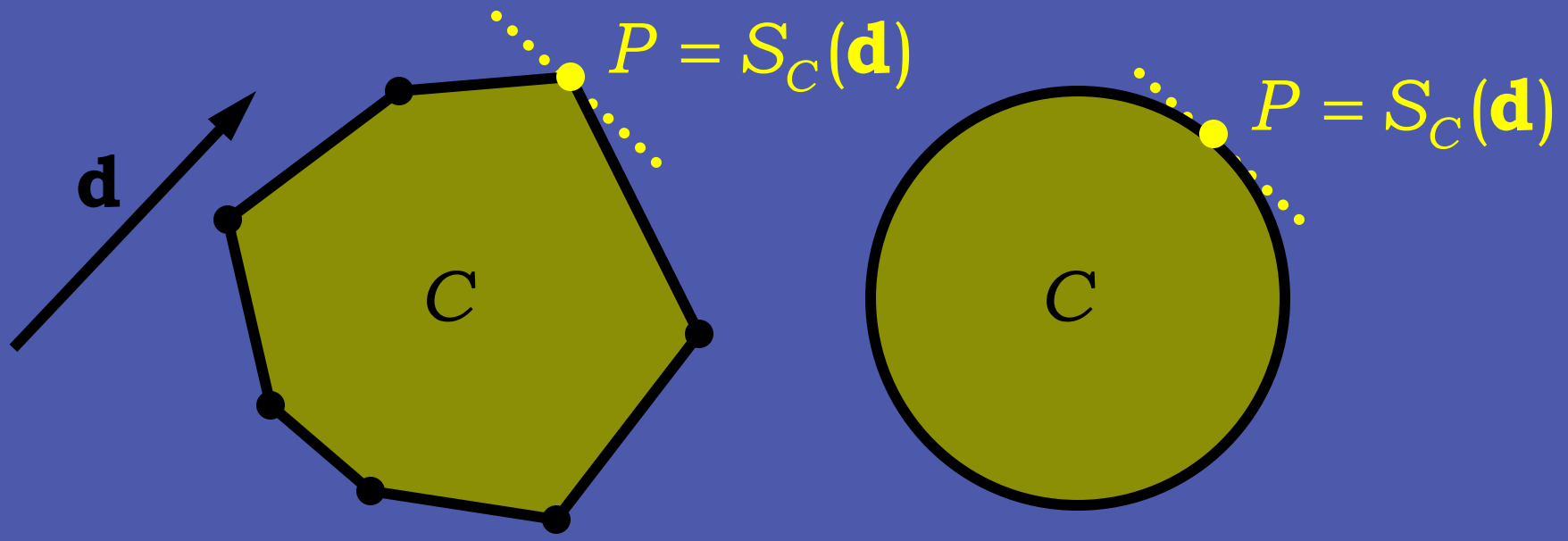


GJK solves proximity queries

- Generalized for arbitrary convex objects
  - As long as they can be described in terms of a *support mapping* function



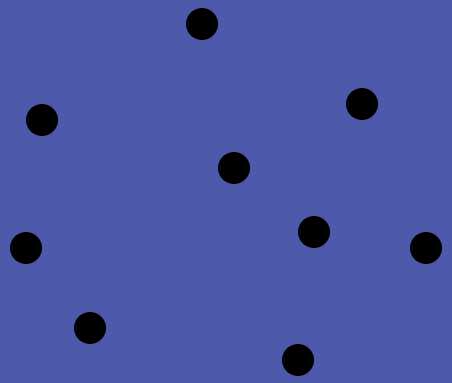
Terminology: Support Mapping Function



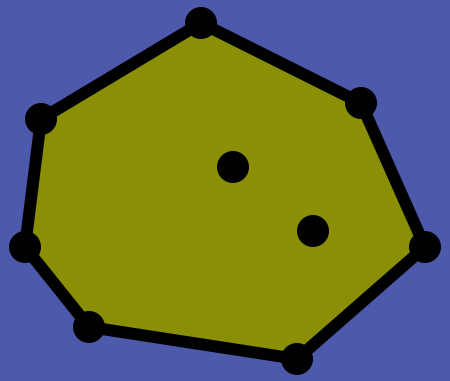
Supporting (or extreme) point P for direction $\mathbf{d}$

returned by support mapping function $S_C(\mathbf{d})$

Terminology



Point set *C*



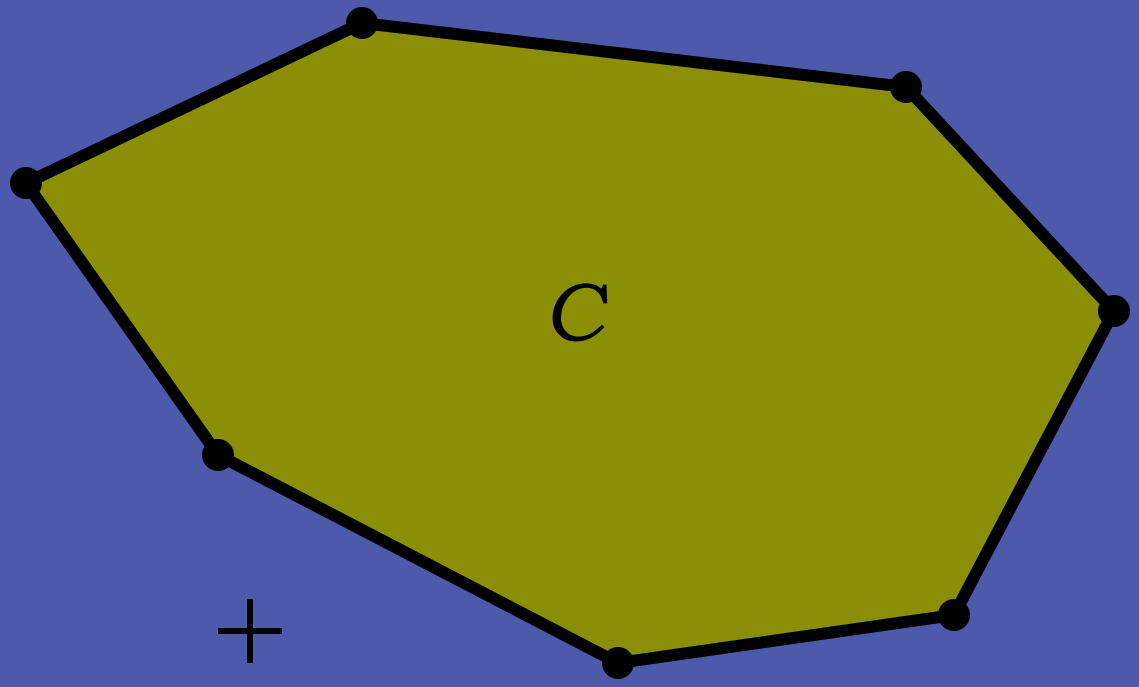
Convex hull, CH(*C*)

GJK Simplified: Point (0) vs. Body (C)

1. Initialize a simplex Q with d+1 points from C (in d dimensions)
2. Compute point P of minimum norm in CH(Q)
3. If P is the origin, exit; return 0
4. Reduce Q to the smallest subset Q' of Q, such that P in CH(Q')
5. Let $V=S_C(-P)$ be a supporting point in direction $-P$
6. If V no more extreme in direction $-P$ than P itself, exit; return $\|P\|$
7. Add V to Q. Go to step 2

GJK example 1(10)

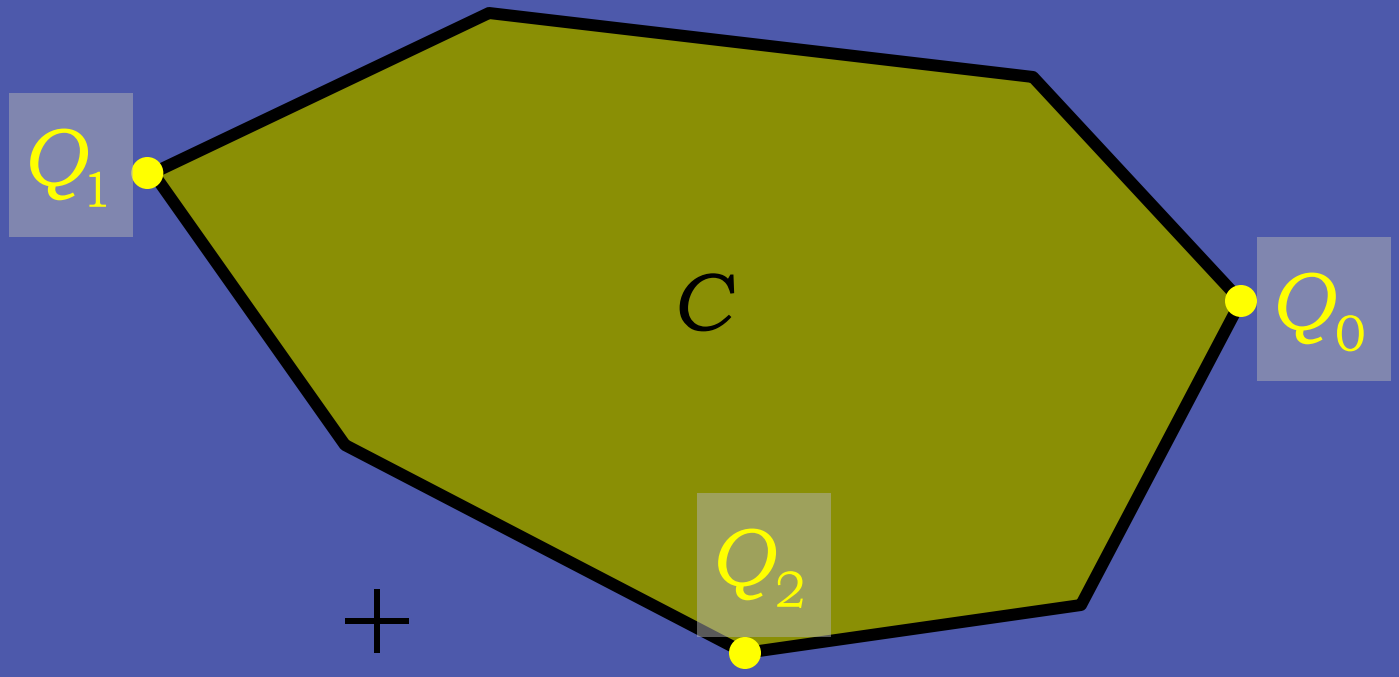
INPUT: Convex polyhedron *C* given as the convex hull of a set of points



GJK example 2(10)

1. Initialize the simplex set *Q* with up to *d*+1 points from *C* (in *d* dimensions)

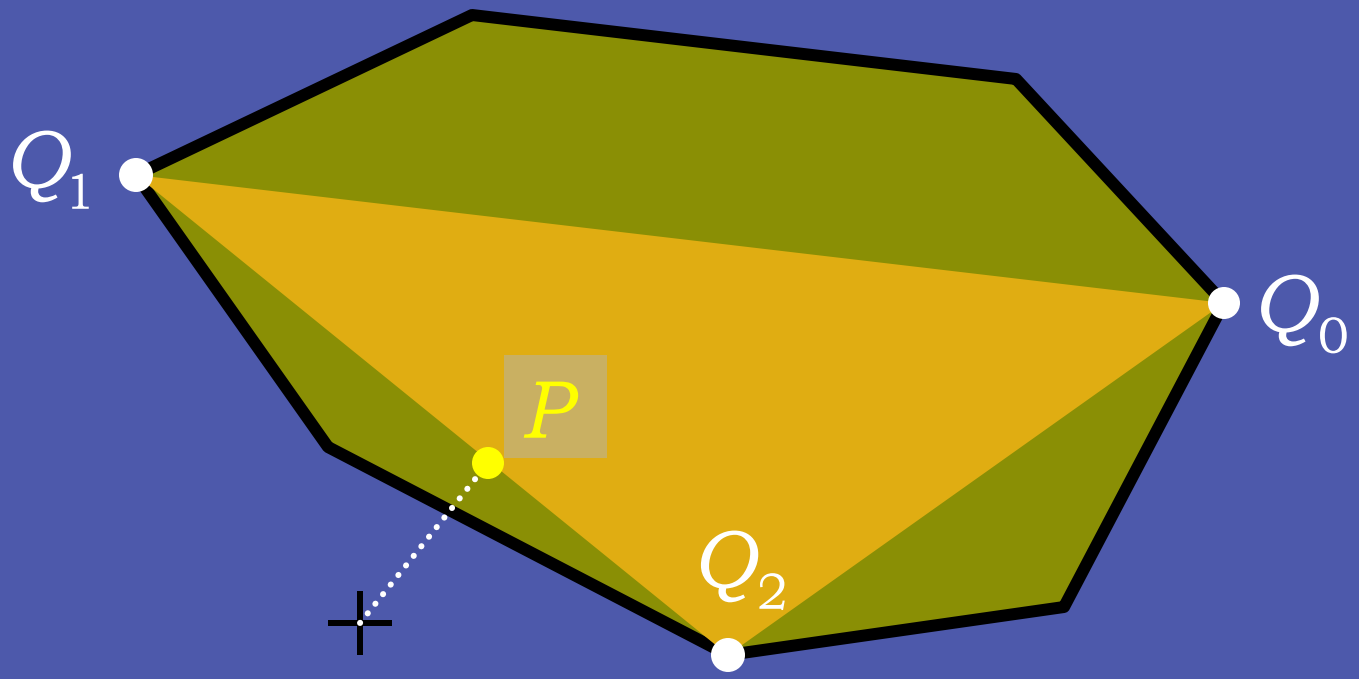
$$Q = \{Q_0, Q_1, Q_2\}$$



GJK example 3(10)

2. Compute point *P* of minimum norm in CH(*Q*)

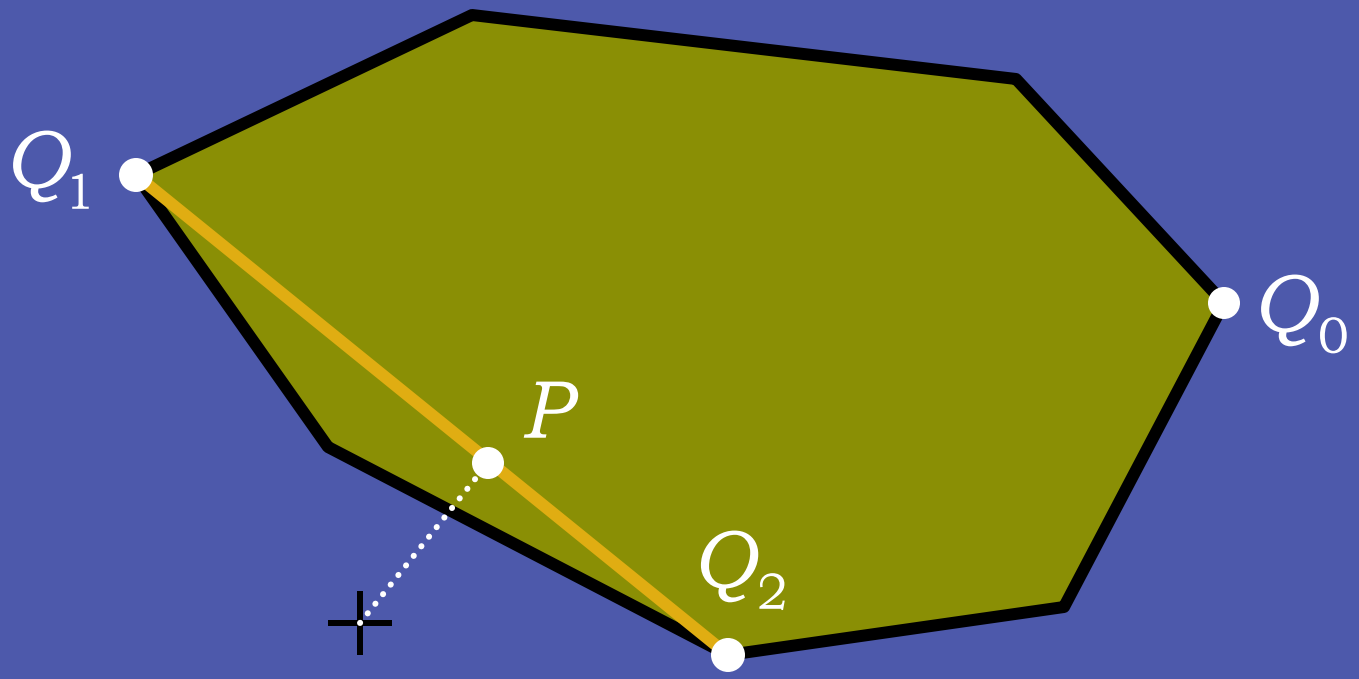
$$Q = \{Q_0, Q_1, Q_2\}$$



GJK example 4(10)

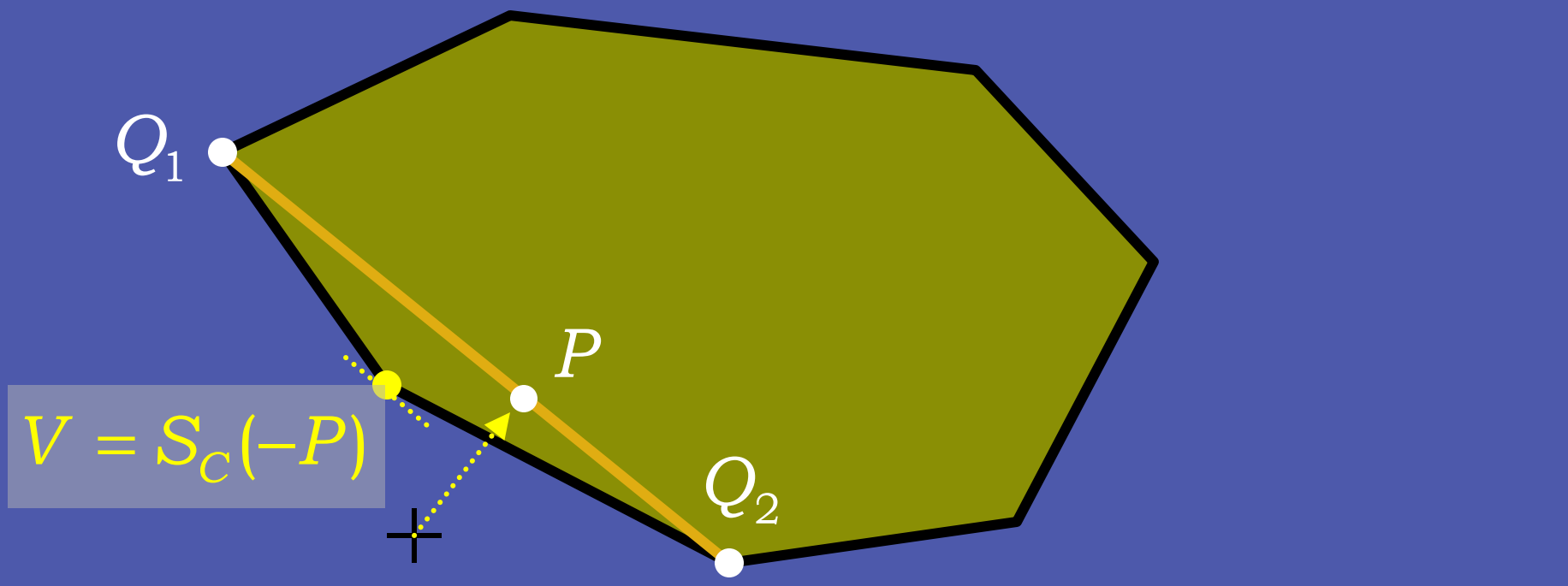
3. If *P* is the origin, exit; return 0
4. Reduce *Q* to the smallest subset *Q'* of *Q*, such that *P* in CH(*Q'*)

$$Q = \{Q_1, Q_2\}$$



GJK example 5(10)

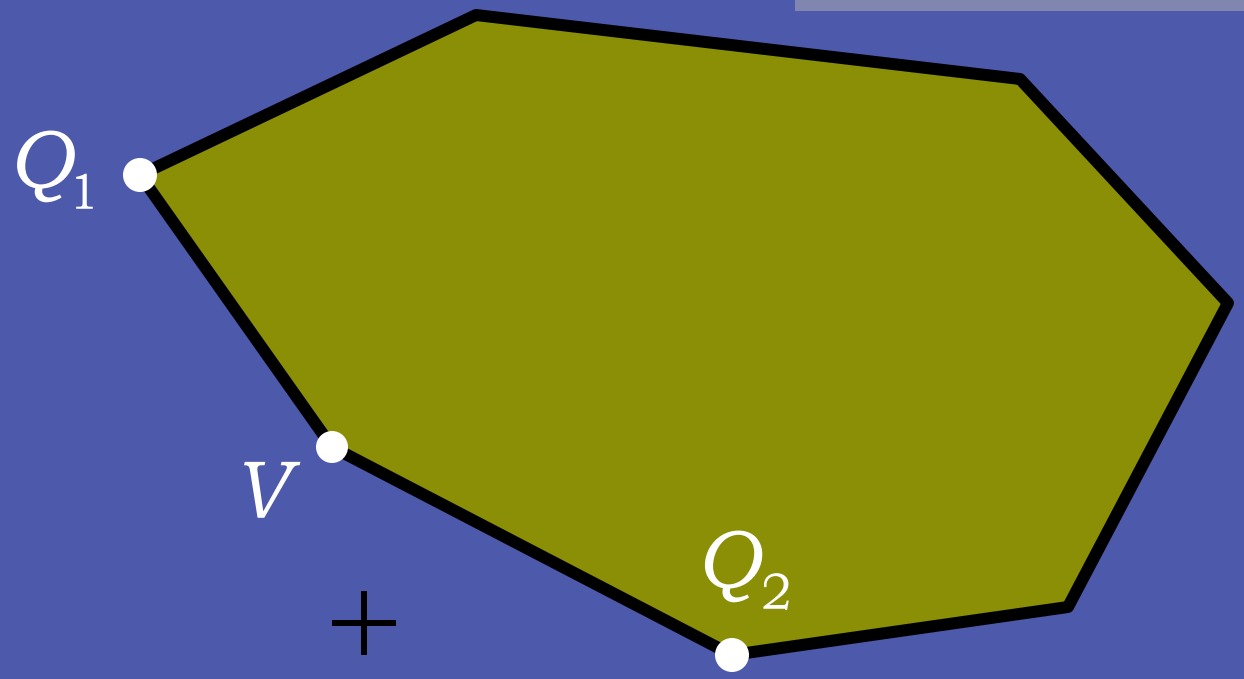
5. Let $V=S_C(-P)$ be a supporting point in direction $-P$



GJK example 6(10)

6. If *V* no more extreme in direction –*P* than *P* itself, exit; return ||*P*||
7. Add *V* to *Q*. Go to step 2

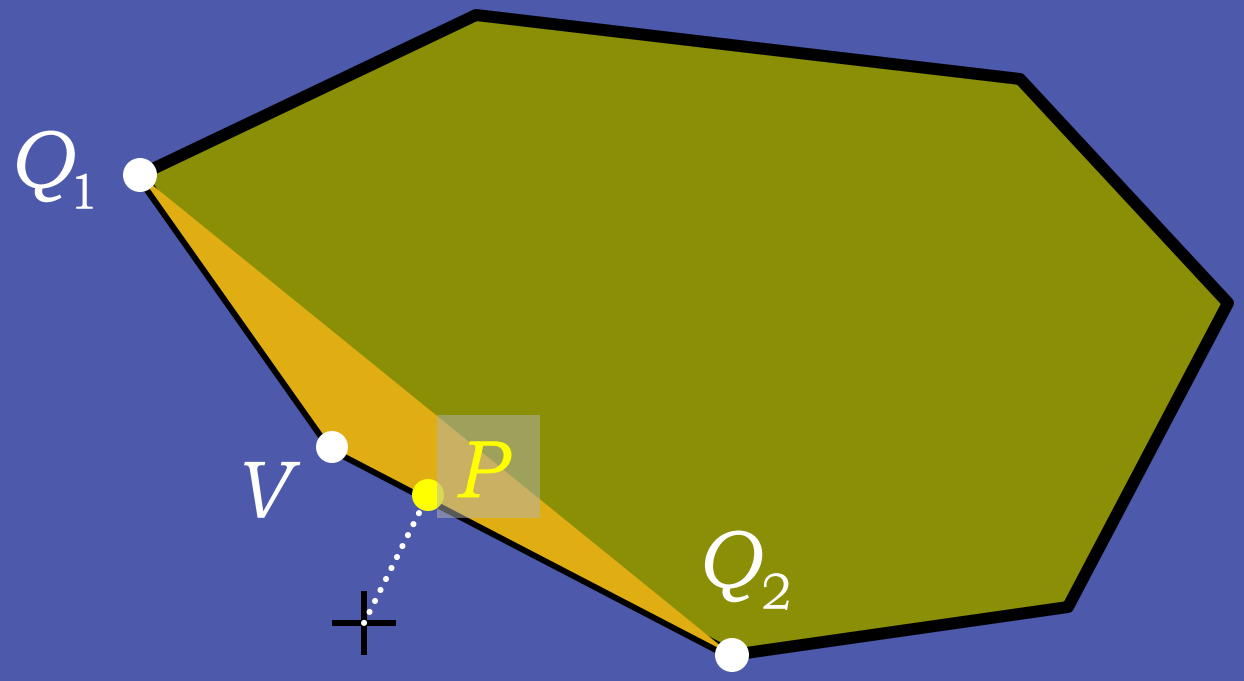
$$Q = \{Q_1, Q_2, V\}$$



GJK example 7(10)

2. Compute point *P* of minimum norm in CH(*Q*)

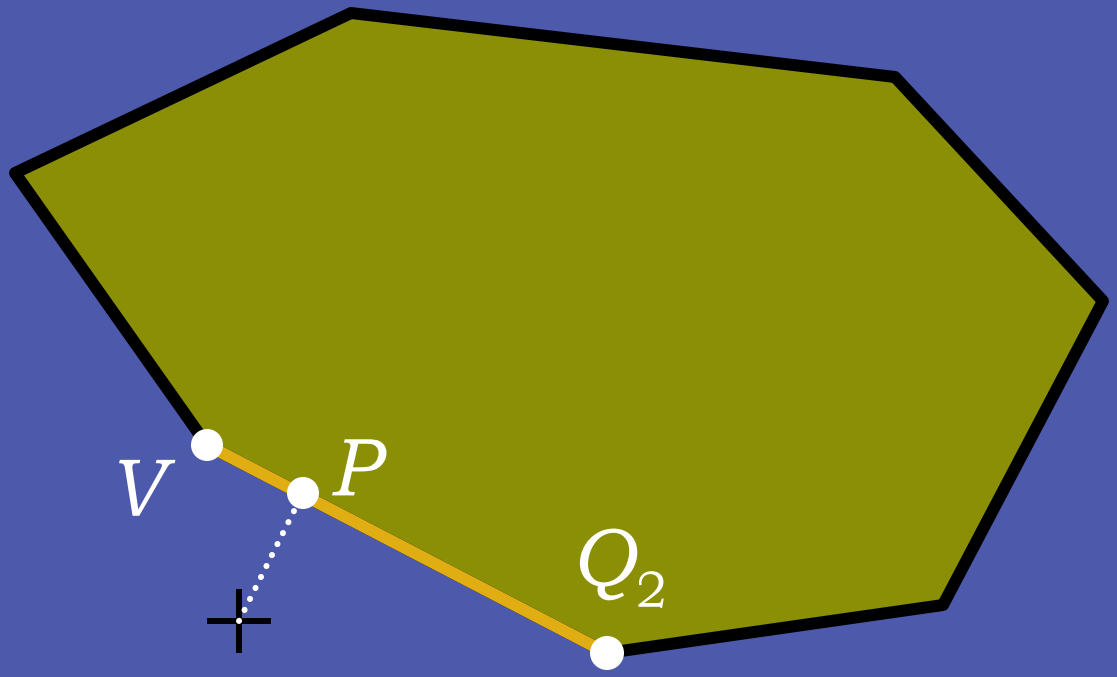
$$Q = \{Q_1, Q_2, V\}$$



GJK example 8(10)

3. If *P* is the origin, exit; return 0
4. Reduce *Q* to the smallest subset *Q'* of *Q*, such that *P* in CH(*Q'*)

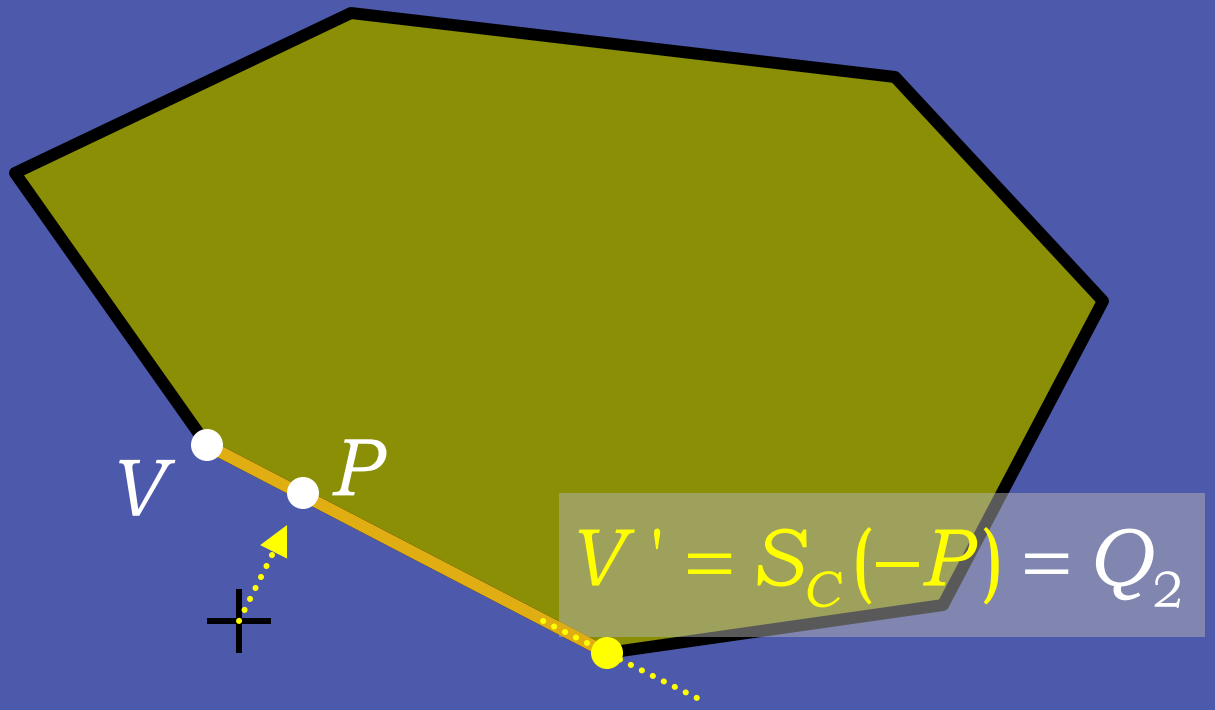
$$Q = \{Q_2, V\}$$



GJK example 9(10)

5. Let $V=S_C(-P)$ be a supporting point in direction $-P$

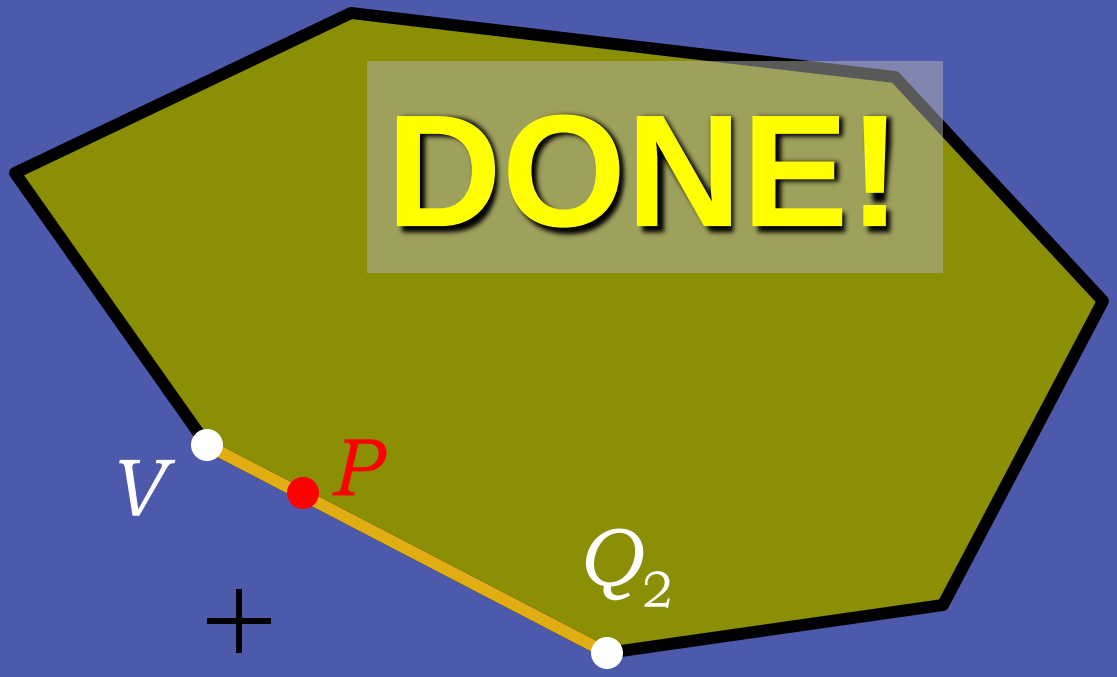
$$Q = \{Q_2, V\}$$



GJK example 10(10)

6. If *V* no more extreme in direction –*P* than *P* itself, exit; return ||*P*||

$$Q = \{Q_2, V\}$$



Distance subalgorithm 1(2)

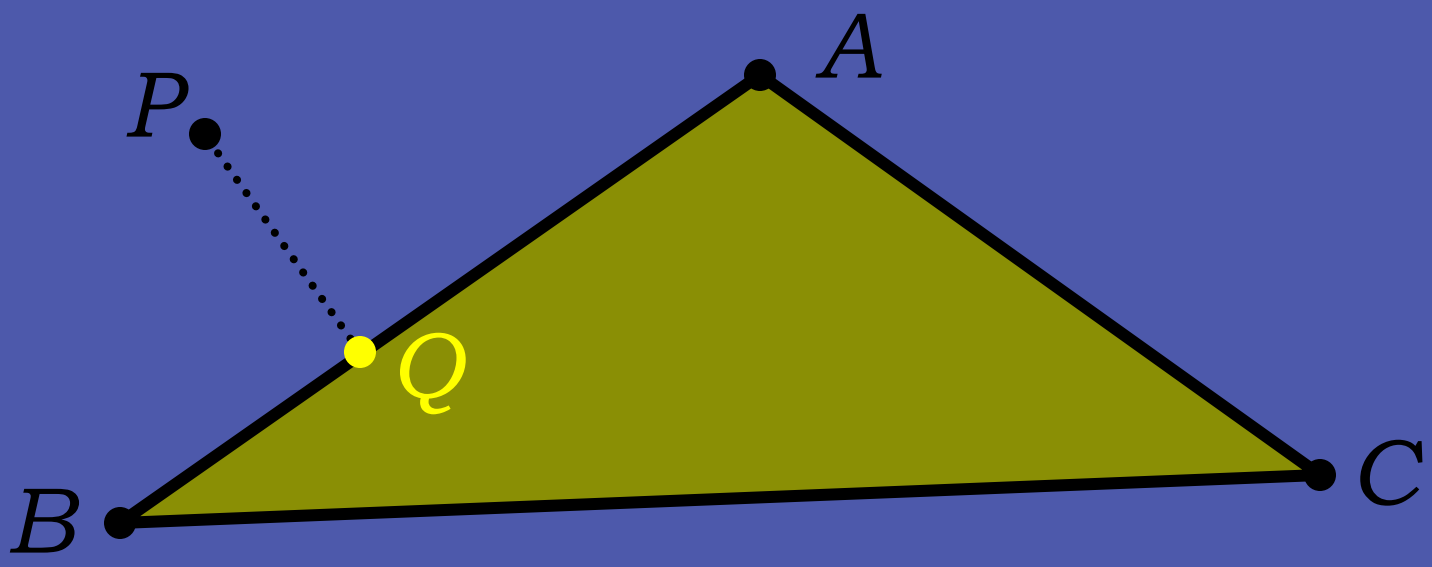
- Approach #1: Solve algebraically
  - Used in original GJK paper (1988)
  - Johnson's distance subalgorithm
    - Searches all simplex subsimplices
    - Solves system of linear equations for each subset
    - Recursive formulation
    - From era when math operations were expensive
    - Robustness problems

Distance subalgorithm 2(2)

- Approach #2: Solve geometrically
  - Mathematically equivalent
    - But more intuitive
    - Therefore easier to make robust
  - Use straightforward primitives:
    - **`ClosestPointOnEdgeToPoint()`**
    - **`ClosestPointOnTriangleToPoint()`**
    - **`ClosestPointOnTetrahedronToPoint()`**
  - Second function outlined here
    - The approach generalizes

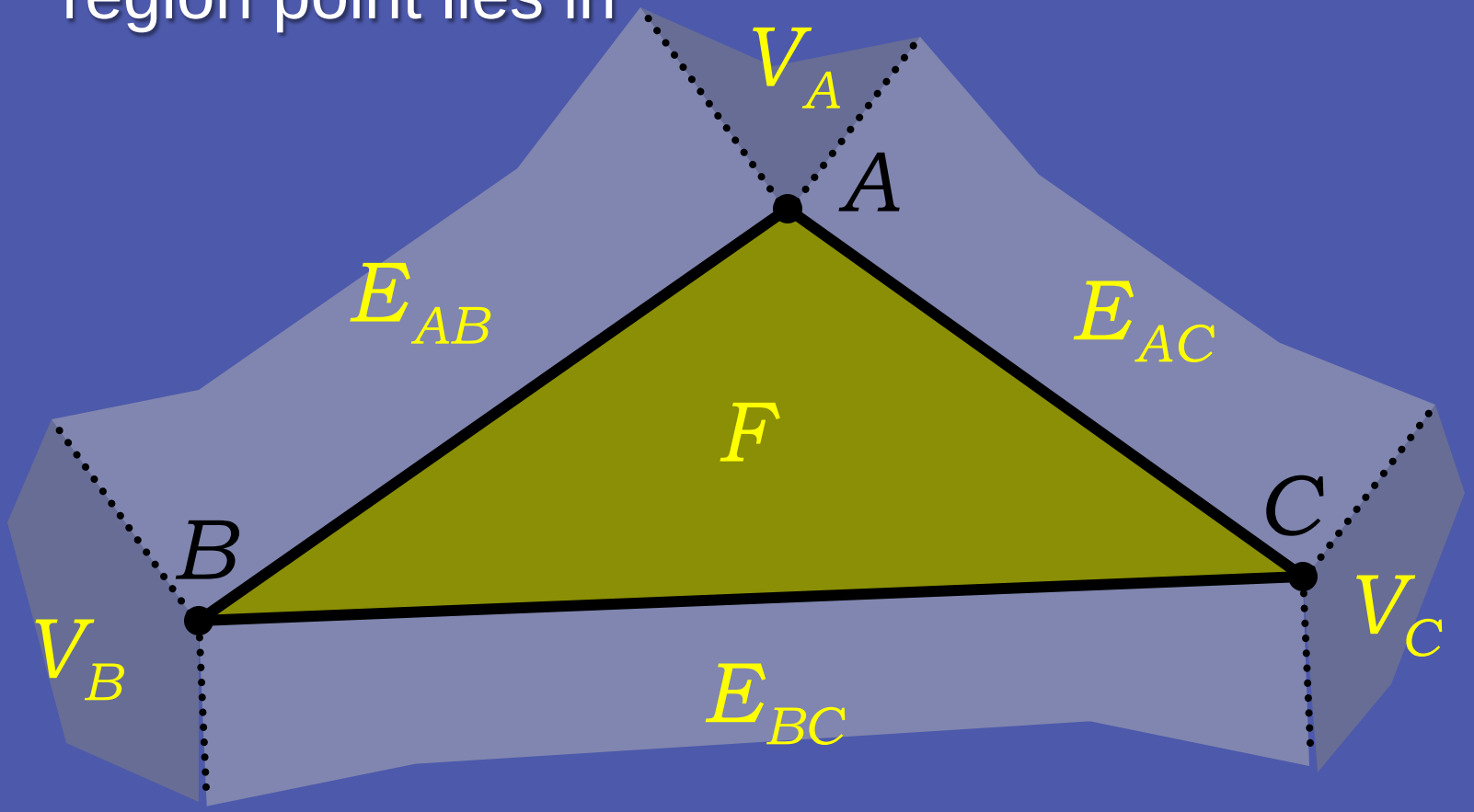
Closest point on triangle

- ClosestPointOnTriangleToPoint()
  - Finds point on triangle closest to a given point



Closest point on triangle (Lin Canny idea)

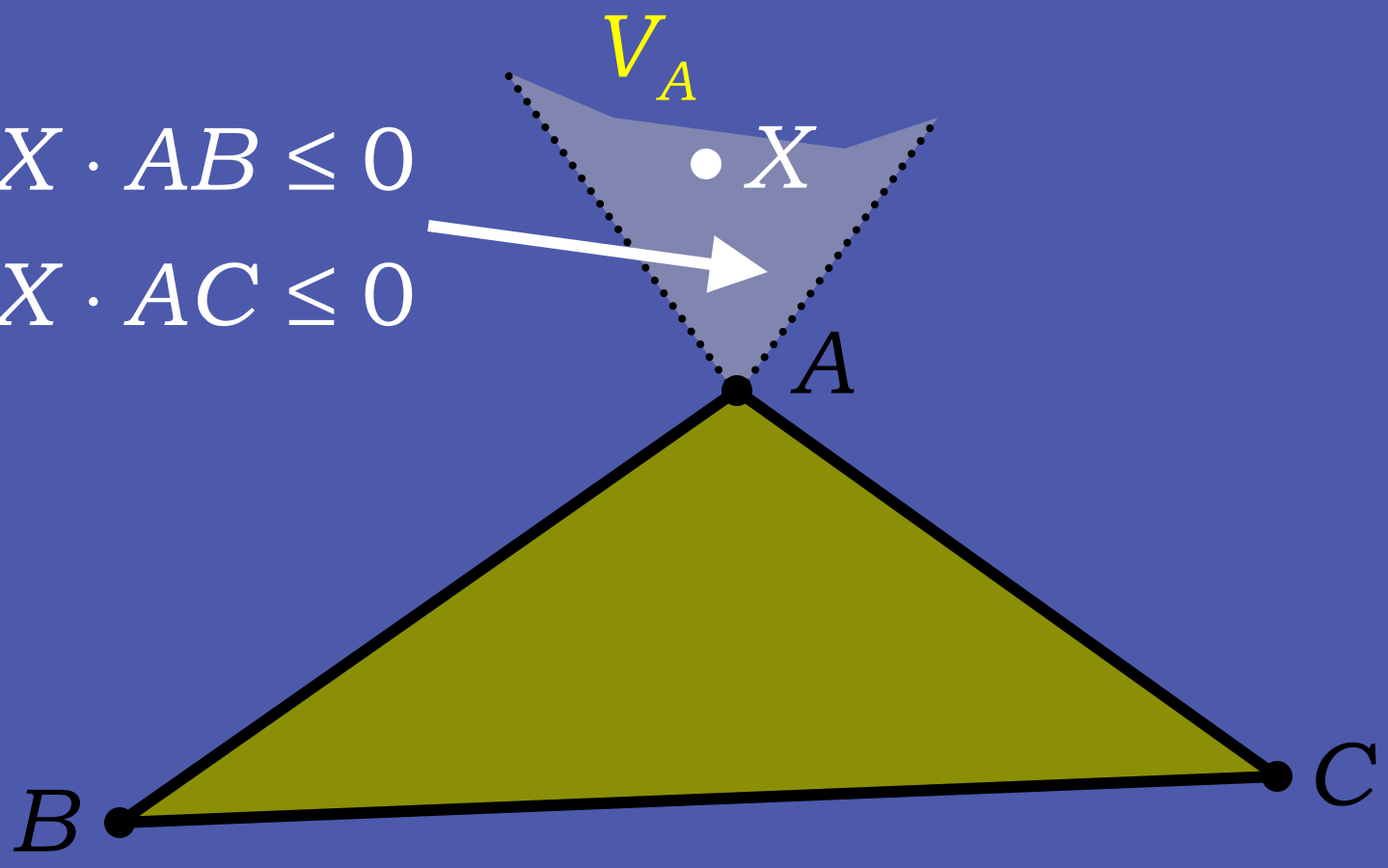
- Separate cases based on which feature Voronoi region point lies in



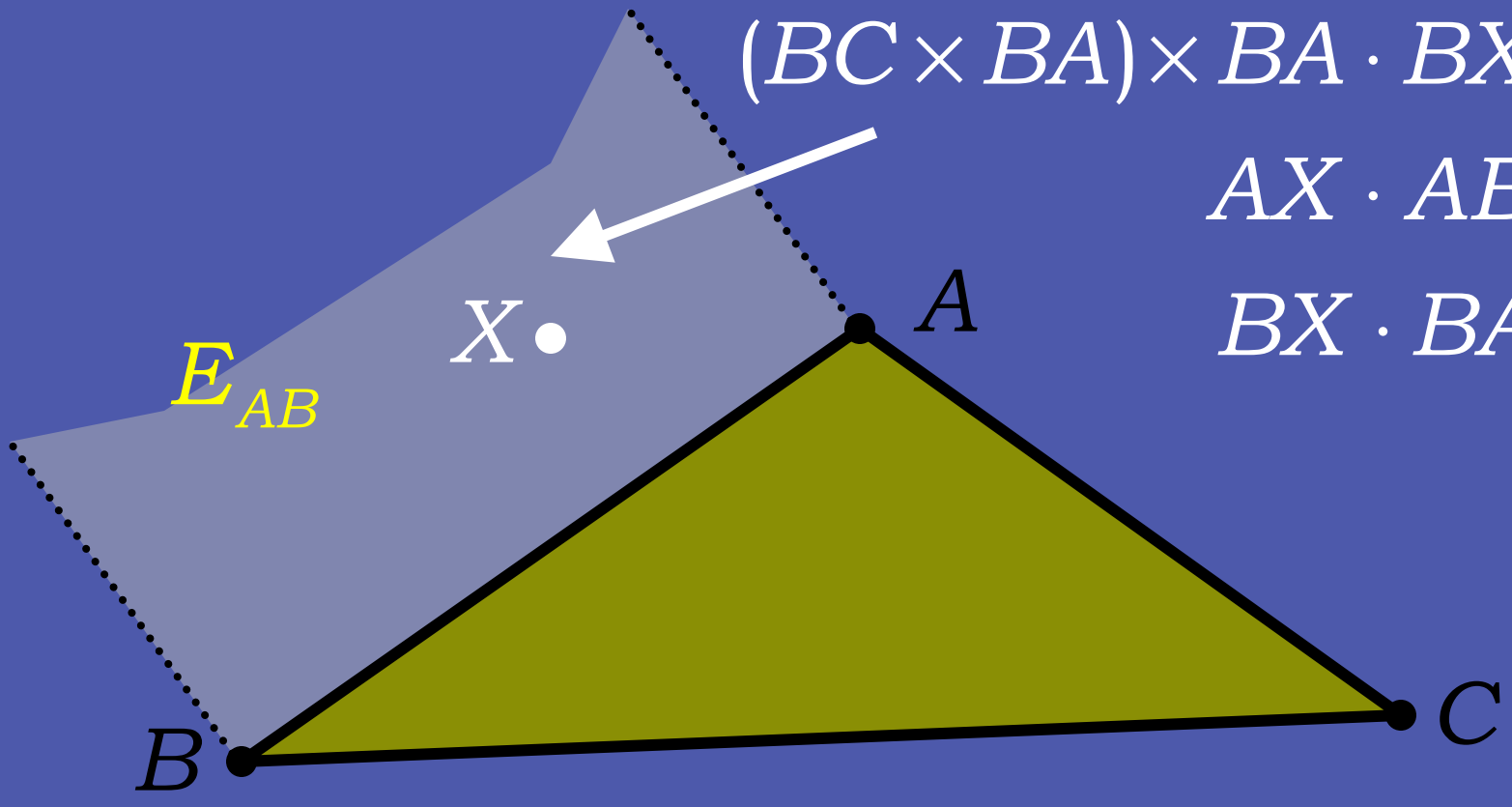
Closest point on triangle

$AX \cdot AB \leq 0$

$AX \cdot AC \leq 0$



Closest point on triangle



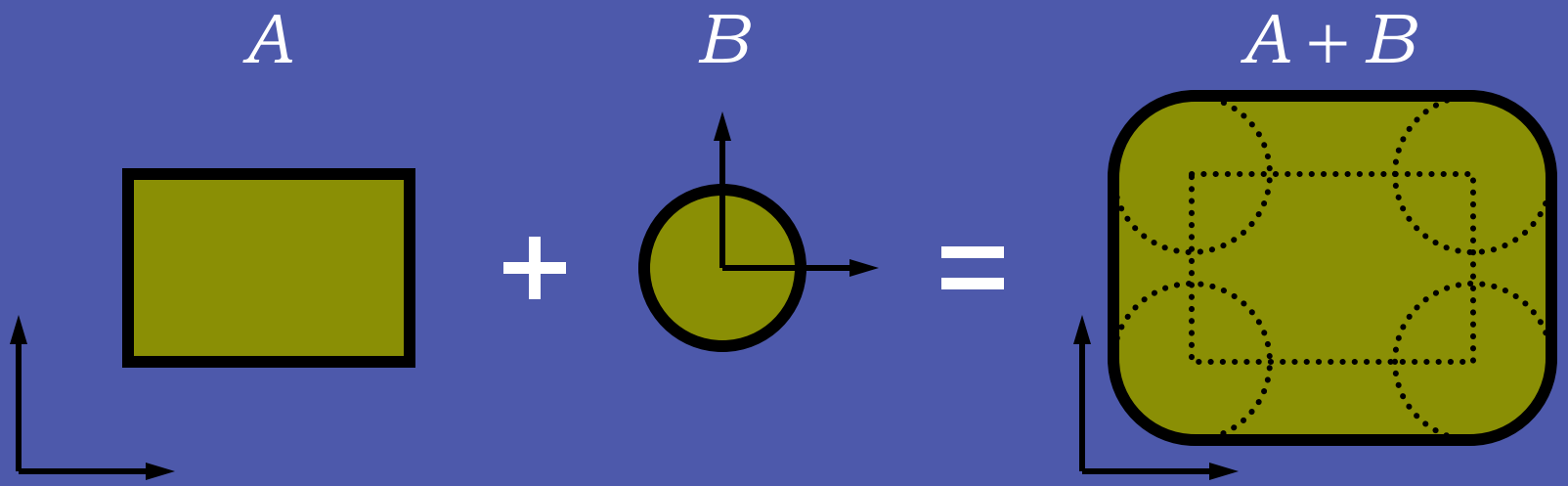
GJK for two bodies

- What about two polyhedra, *A* and *B*?
- Reduce problem into the one solved
  - No change to the algorithm!
  - Relies on the properties of the **Minkowski difference** of *A* and *B*

- Not enough time to go into full detail
  - Just a brief description

Minkowski sum & difference

- Minkowski sum
  - The sweeping of one convex object with another



- Defined as:
  - $A + B = \{\mathbf{a} + \mathbf{b} : \mathbf{a} \in A, \mathbf{b} \in B\}$

Minkowski sum & difference

- Minkowski difference, defined as:
  - $A - B = \{\mathbf{a} - \mathbf{b} : \mathbf{a} \in A, \mathbf{b} \in B\}$

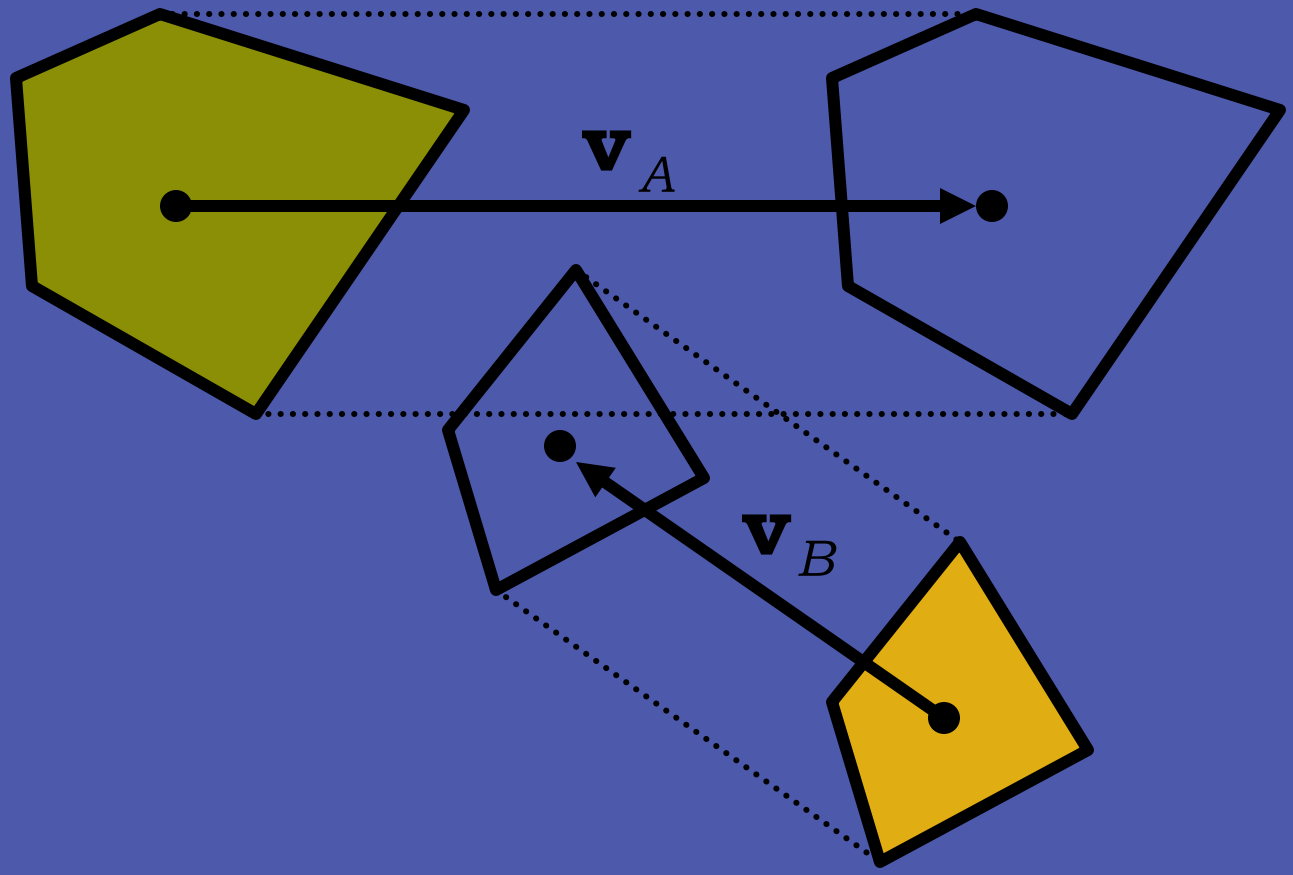
    $= A + (-B)$
- Can write distance between two objects as:
  - $\text{distance}(A, B) = \min\{\|\mathbf{a} - \mathbf{b}\| : \mathbf{a} \in A, \mathbf{b} \in B\}$

    $= \min\{\|\mathbf{c}\| : \mathbf{c} \in A - B\}$
- *A* and *B* intersect iff *A*–*B* contains the origin*!*
  - Distance between *A* and *B* given by point of minimum norm in *A*–*B!*

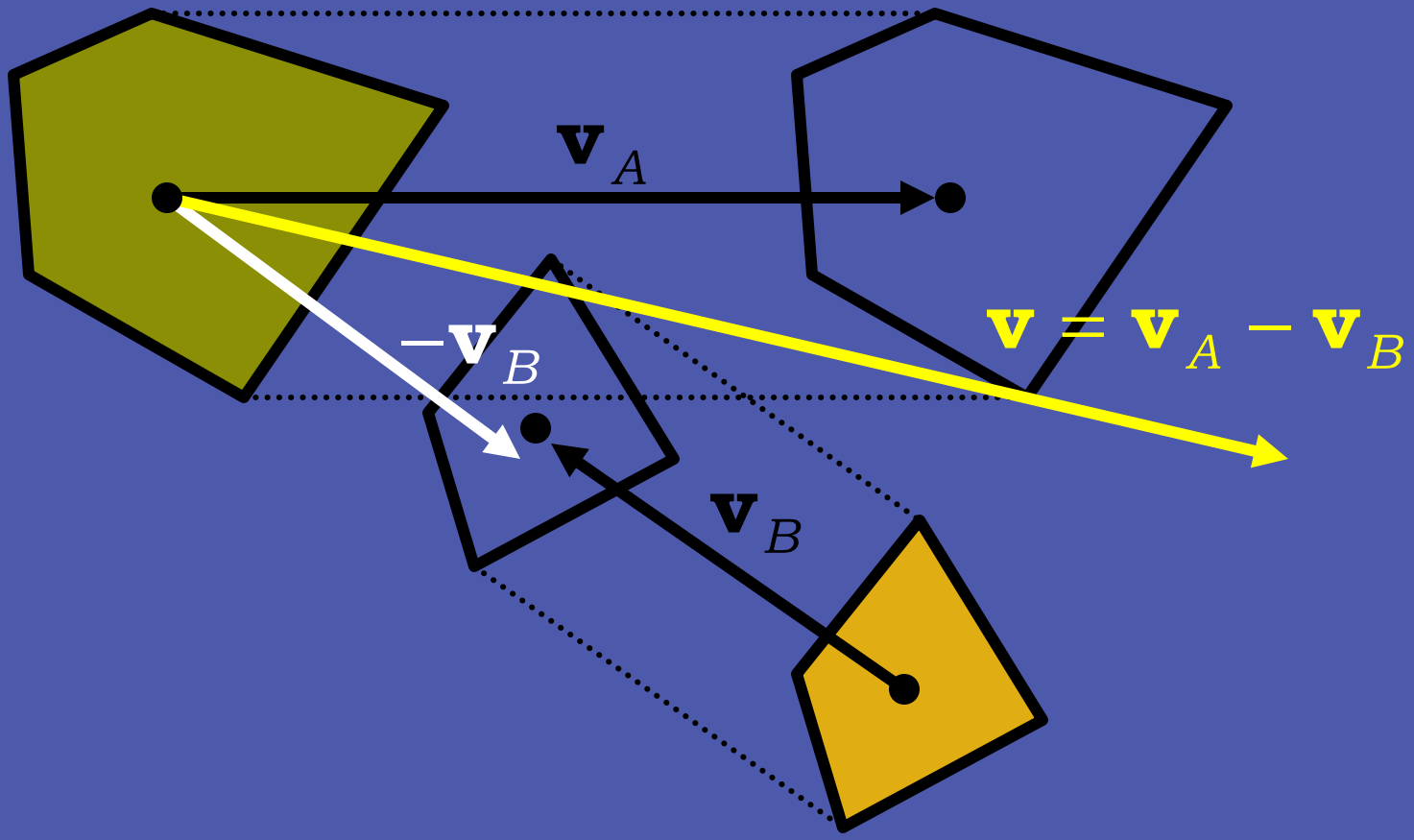
The generalization

- *A* and *B* intersect iff *A*–*B* contains the origin*!*
  - Distance between *A* and *B* given by point of minimum norm in *A*–*B!*
- So use previous procedure on *A*–*B!*
- Only change needed: computing $S_C(\mathbf{d}) = S_{A-B}(\mathbf{d})$
- Support mapping is separable, so can form it by computing support mapping for *A* and *B* separately*!*
  - $S_C(\mathbf{d}) = S_{A-B}(\mathbf{d}) = S_A(\mathbf{d}) - S_B(-\mathbf{d})$

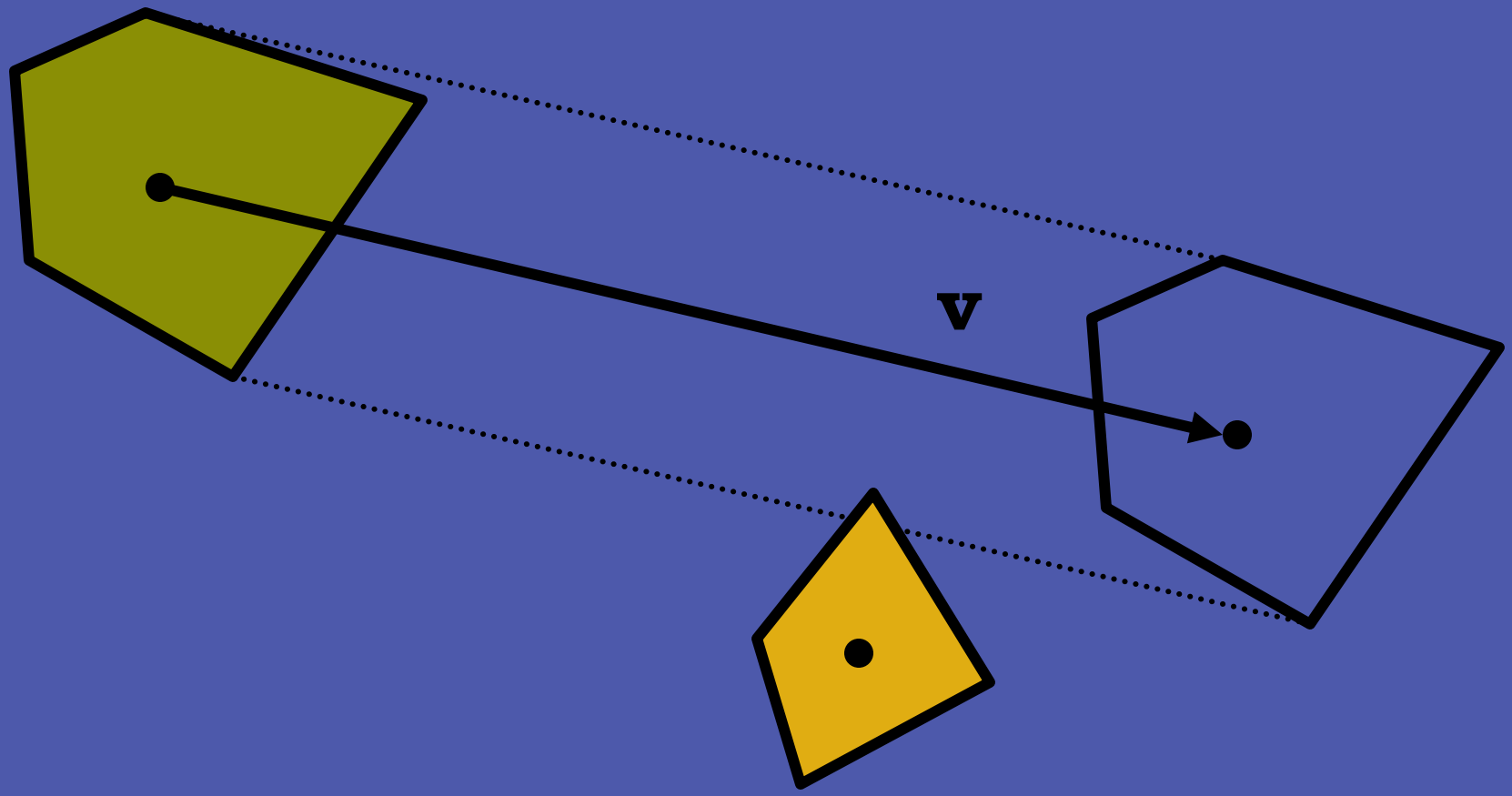
GJK for moving bodies (translation)



Transform the problem…

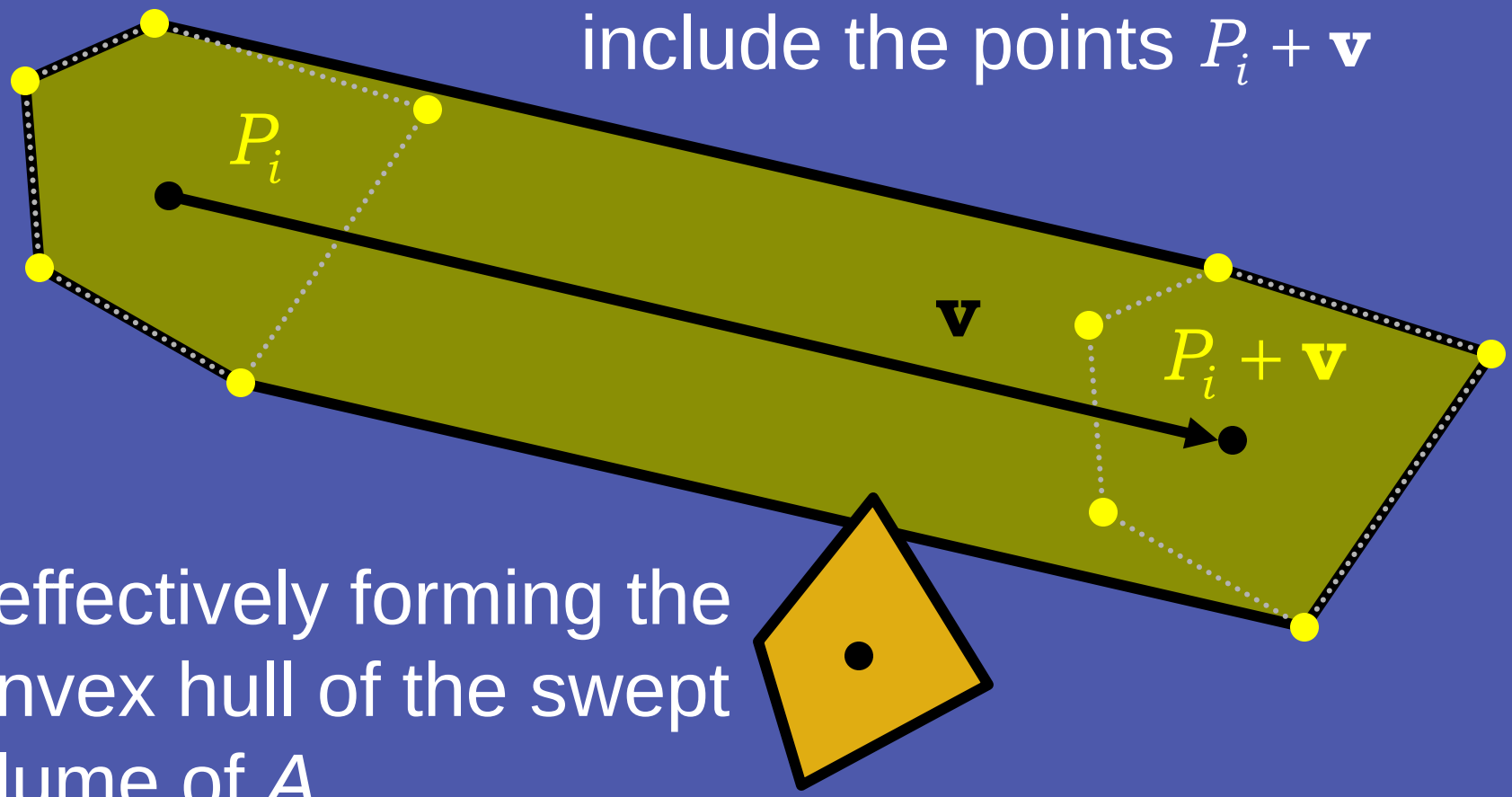


…into moving vs stationary



Alt #1: Point duplication

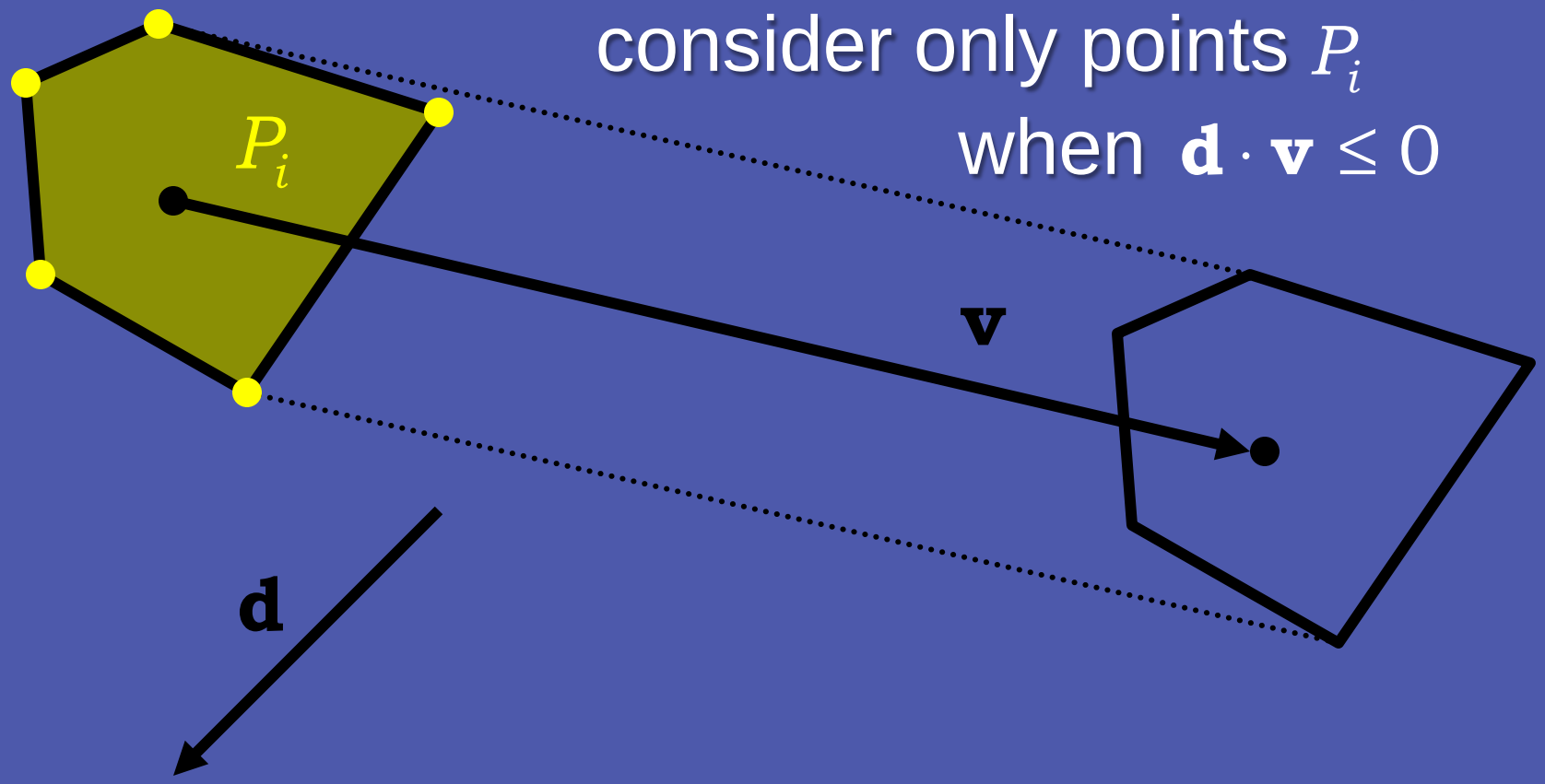
Let object *A* additionally include the points $P_i + \mathbf{v}$



…effectively forming the convex hull of the swept volume of *A*

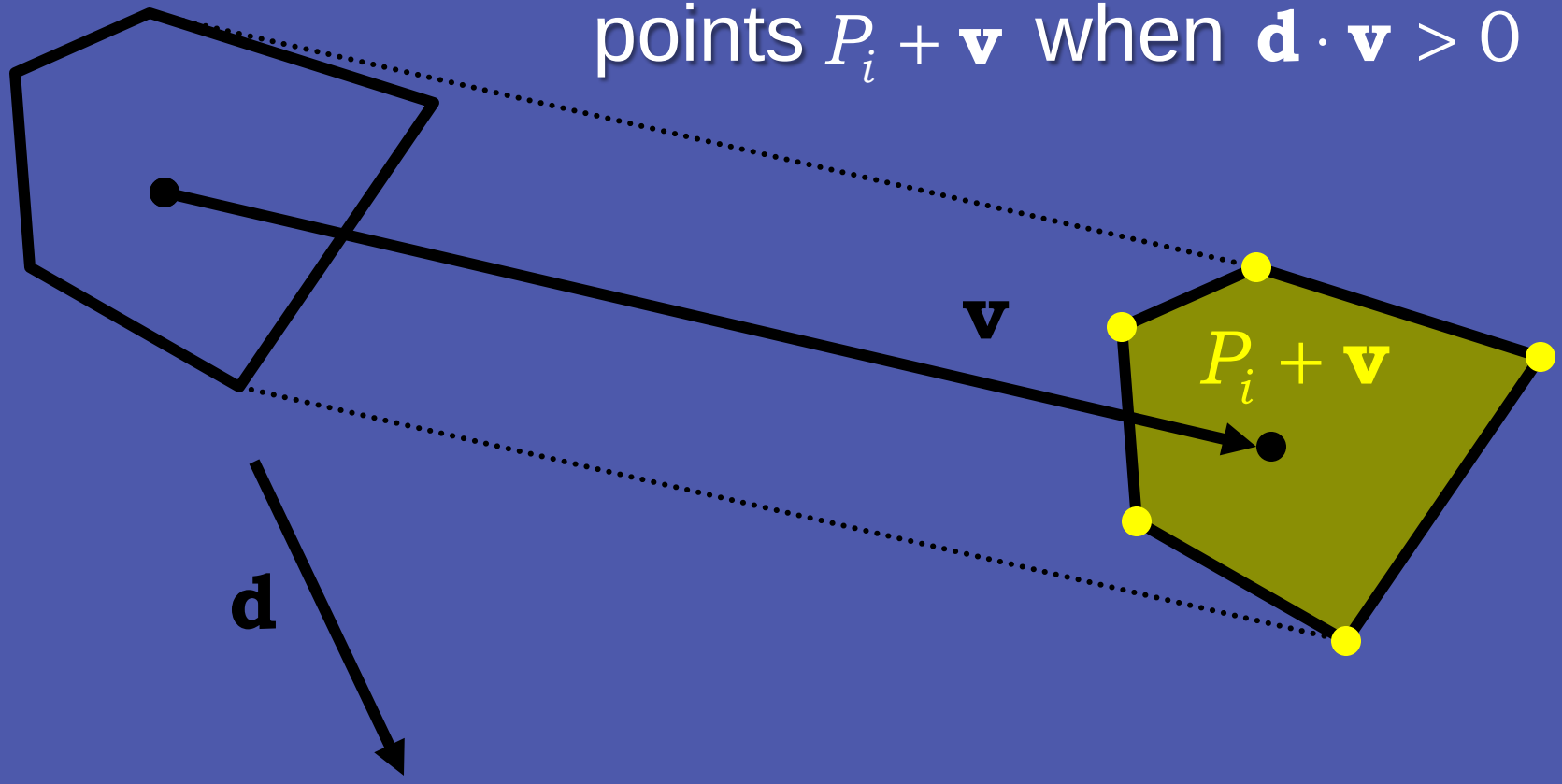
Alt #2: Support mapping

Modify support mapping to consider only points P_i when $\mathbf{d} \cdot \mathbf{v} \leq 0$



Alt #2: Support mapping

…and to consider only points $P_i + \mathbf{v}$ when $\mathbf{d} \cdot \mathbf{v} > 0$



GJK for moving objects

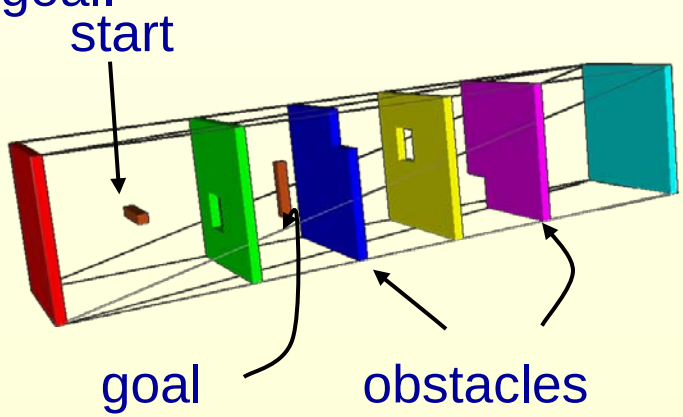
- Presented solution
  - Gives only Boolean interference detection result
- Interval halving over v gives time of collision
  - Using simplices from previous iteration to start next iteration speeds up processing drastically
- Overall, always starting with the simplices from the previous iteration makes GJK…
  - Incremental
  - Very fast

Motion Planning

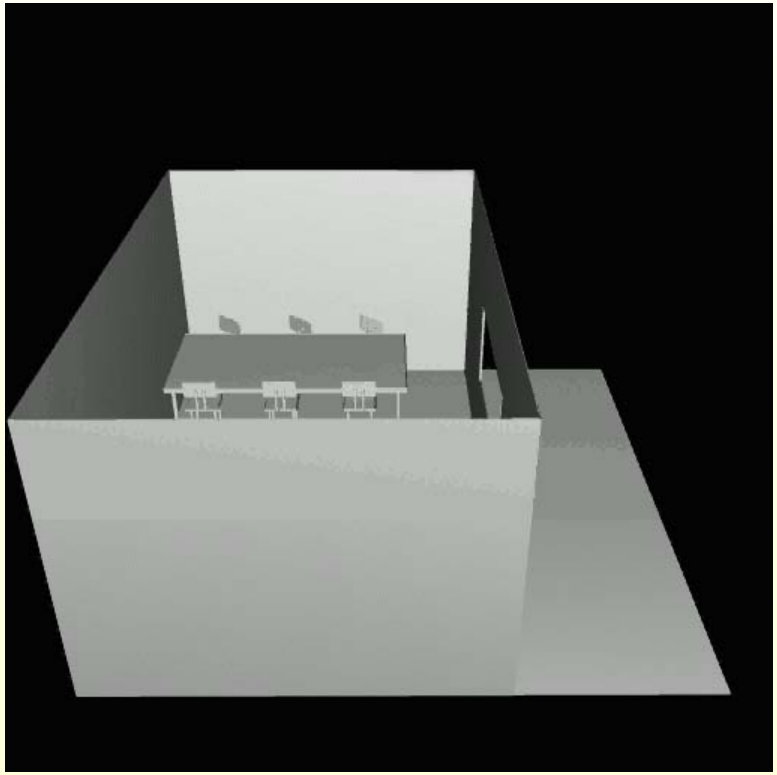
Motion Planning

(Basic) Motion Planning:

Given a *movable object* and a description of the *environment*, find a *sequence of collision-avoiding configurations* that moves it from the start to the goal.

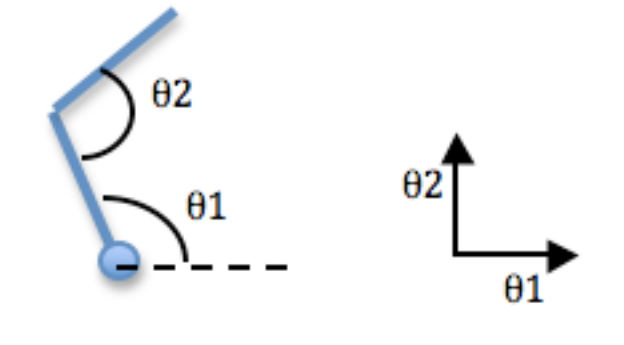


The Piano Movers Problem



37

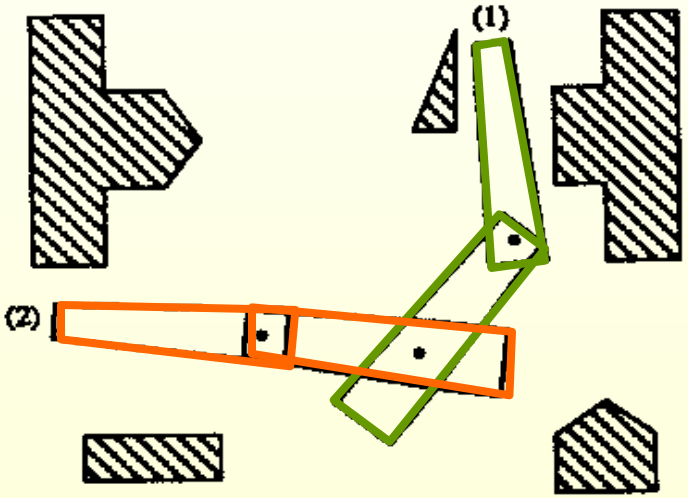
Configuration Space



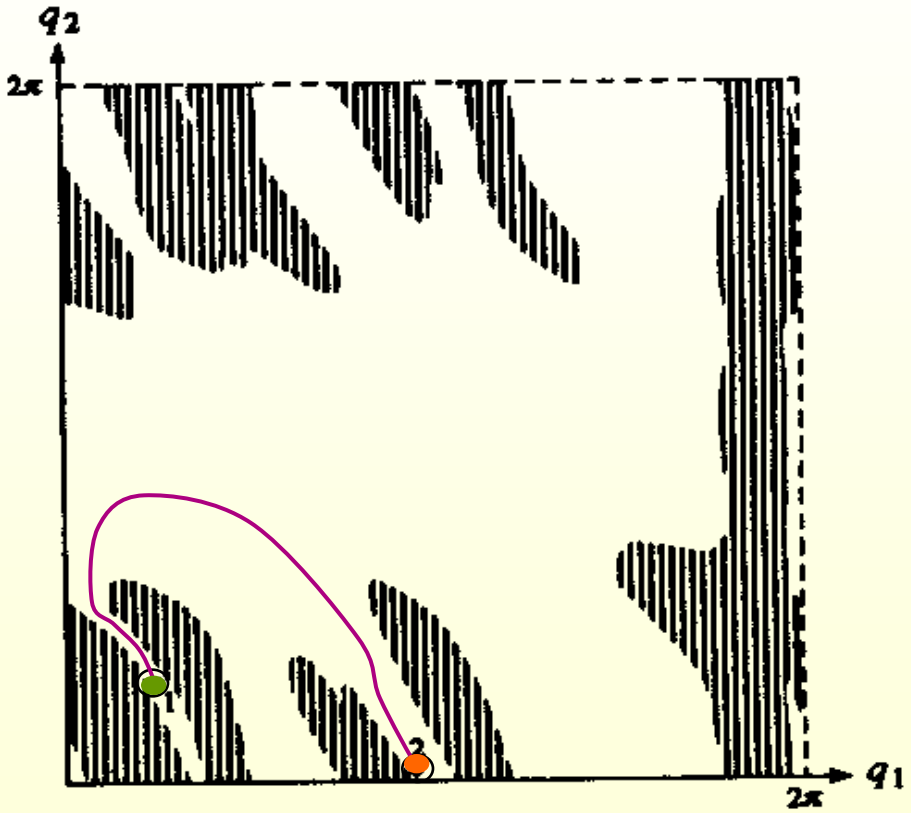
http://www-inst.eecs.berkeley.edu/~cs188/fa08/demos/robot.html

Configuration Space: Tool to Map a Robot to a Point

2-parameter robot



Real Space

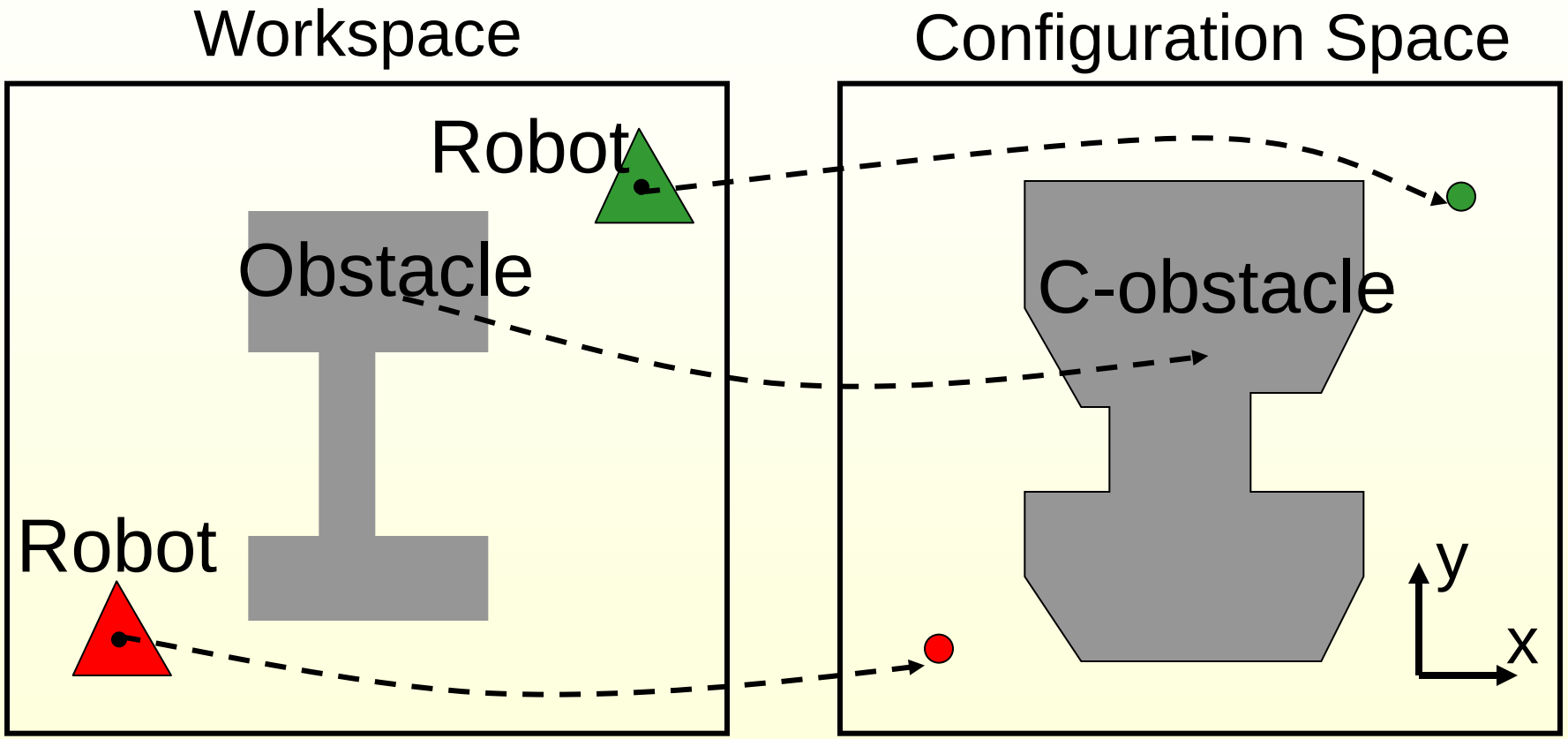


Configuration Space

Problem – Configuration Space of aTranslating Robot

- **Input:**
  - Polygonal moving object translating in 2-D workspace
  - Polygonal obstacles
- **Output**: configuration space obstacles represented as polygons

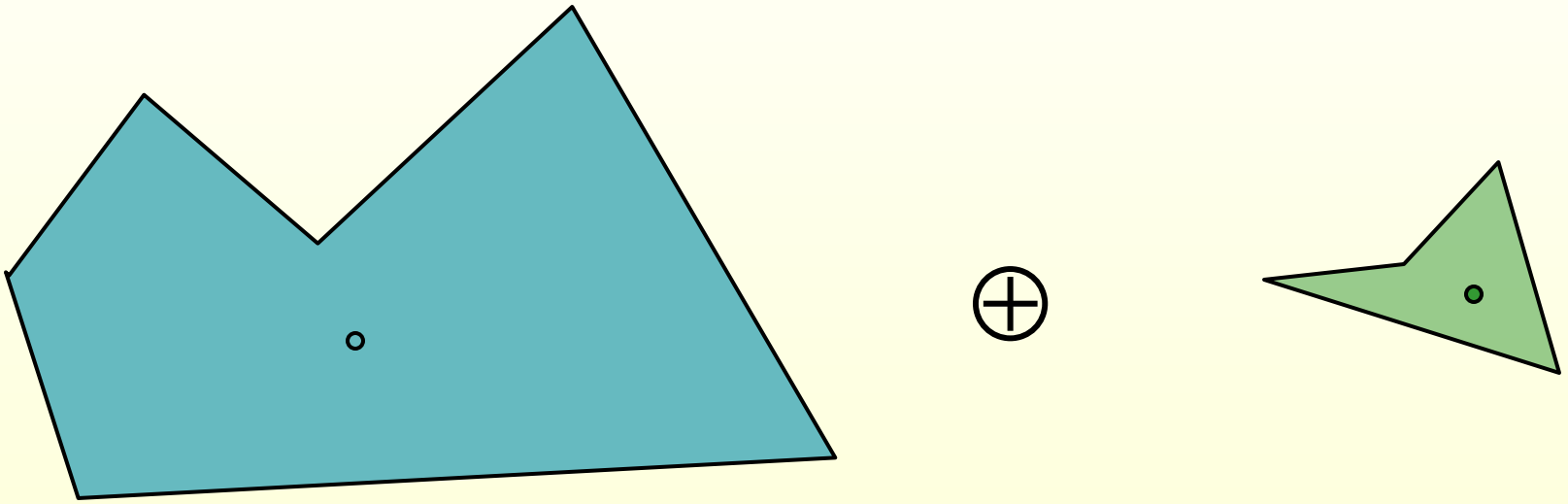
Configuration Space of a Translating Robot



- C-obstacle is a polygon.

Minkowski Sum

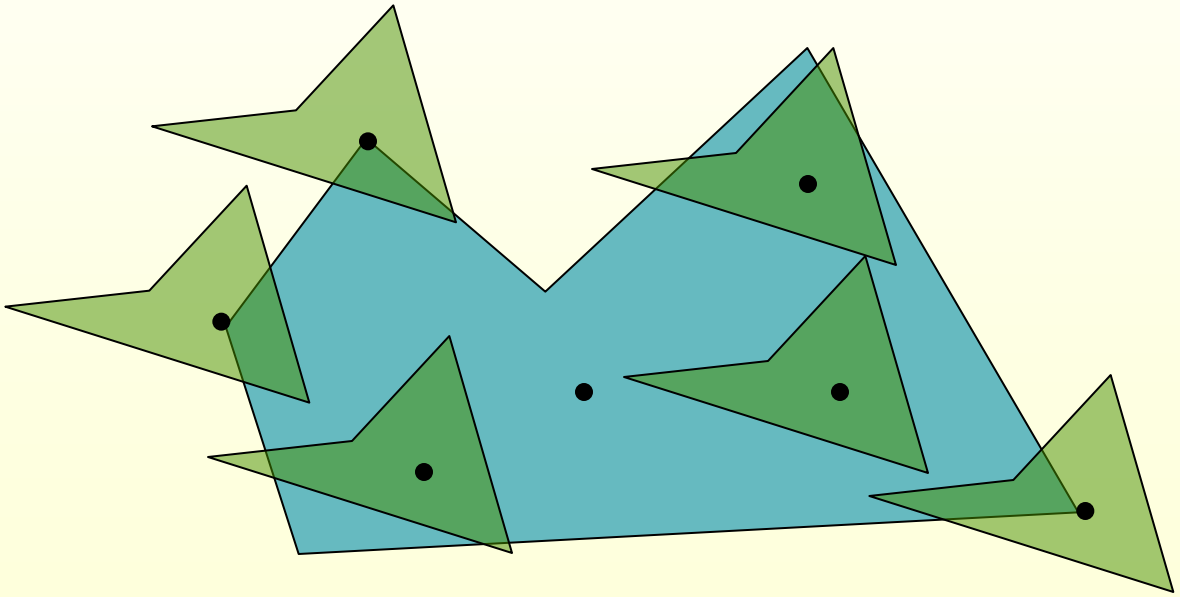
$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$



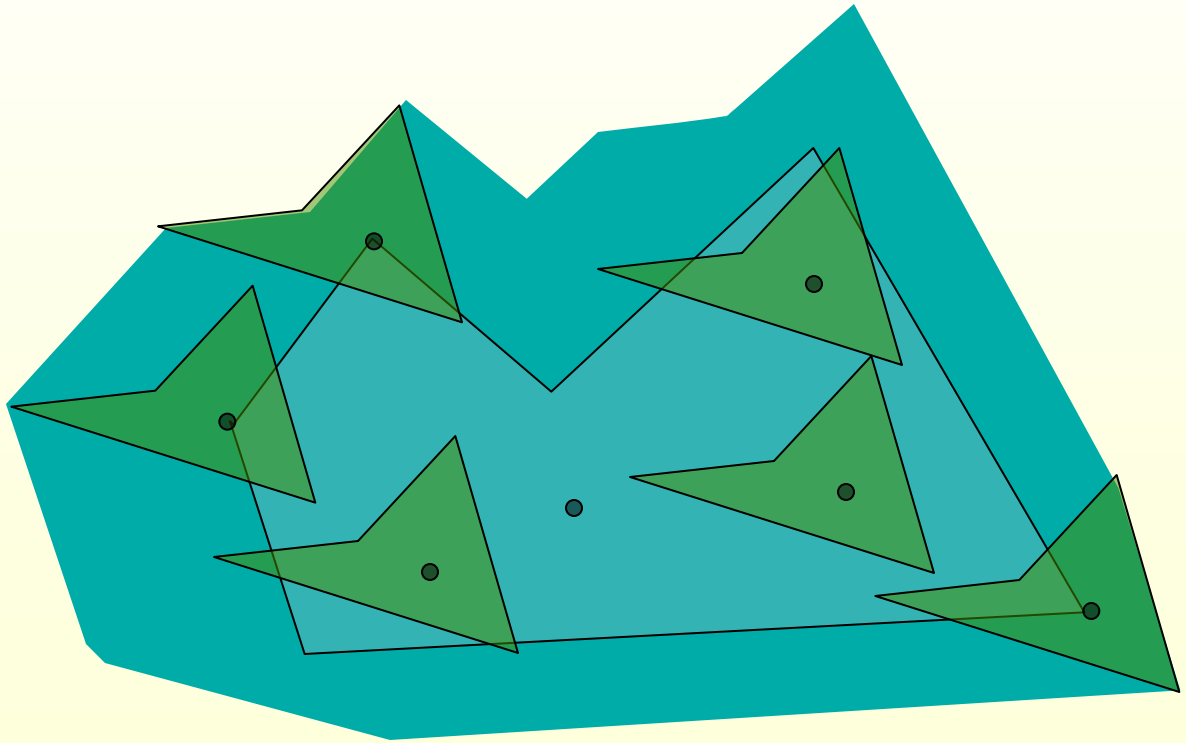
A ⊕ B

Minkowski Sum

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$



Minkowski Sum



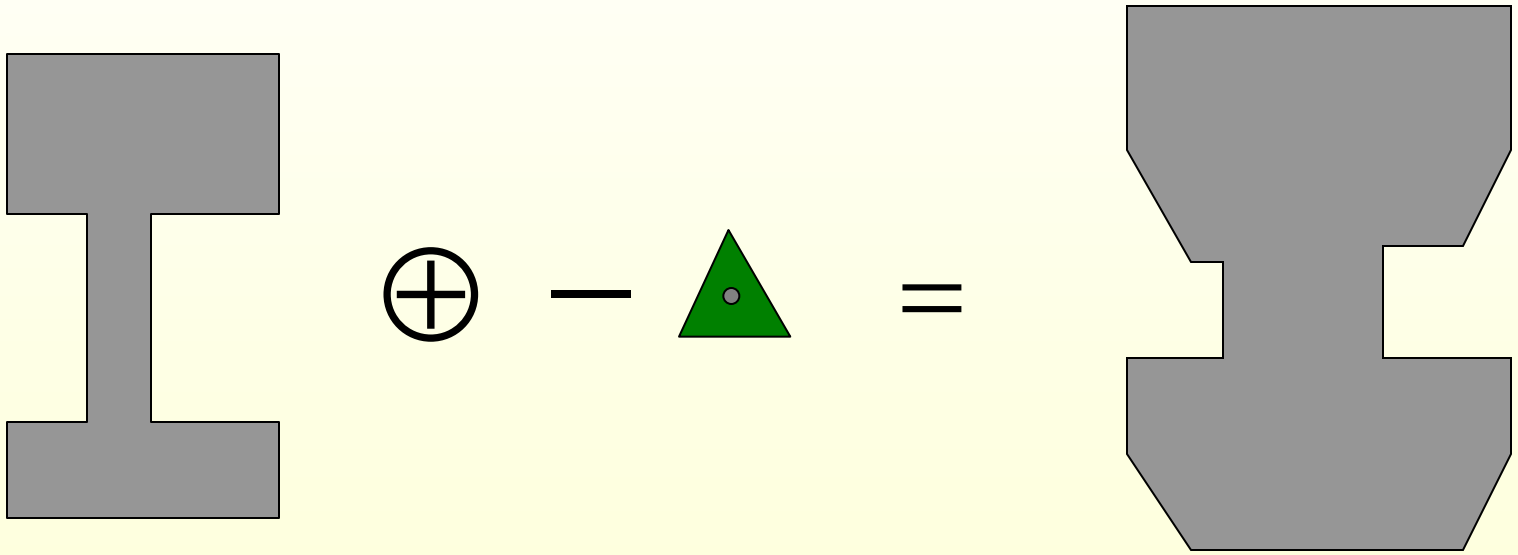
Minkowski Sum

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$

Configuration Space Obstacle

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

Classic result by Lozano-Perez and Wesley 1979



Obstacle
O

Robot
$\mathcal{R}_{46}$

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

Properties of Minkowski Sum

- Minkowski sum of boundary of P and boundary of Q is a subset of the boundary of

$$P \oplus Q$$

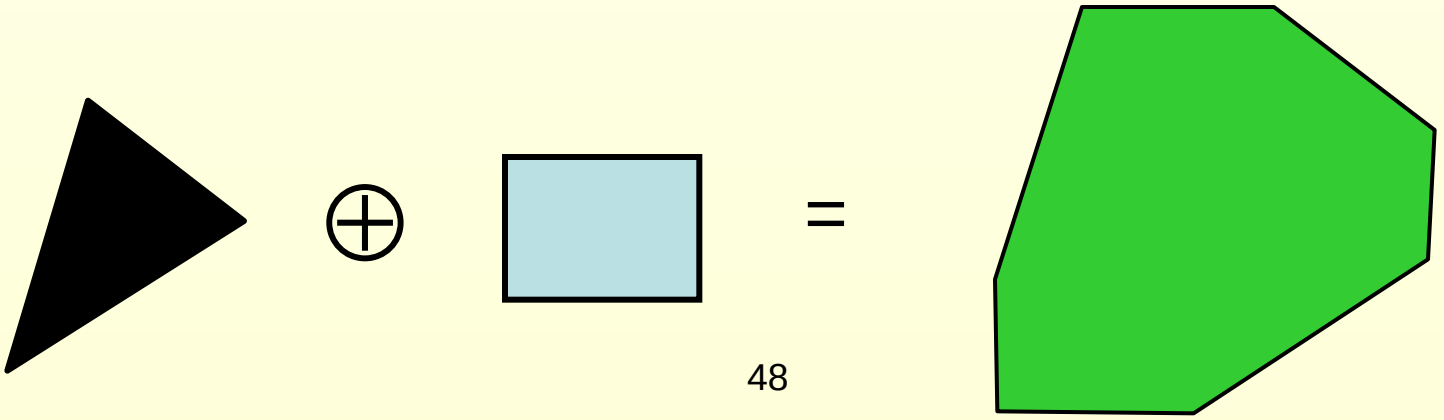
- Minkowski of two convex sets is convex

Minkowski sum of convex polygons

- The Minkowski sum of two convex polygons P and Q of m and n vertices respectively

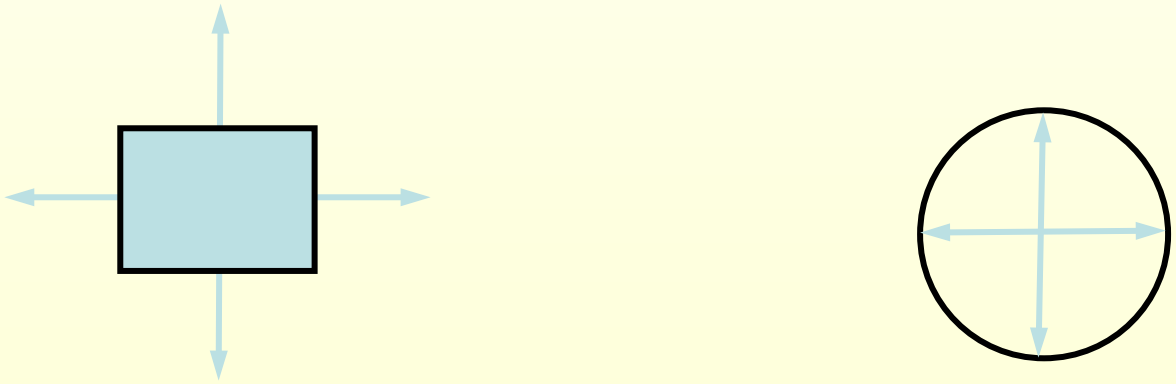
  is a convex polygon $P \oplus Q$ of $m + n$ vertices.

  - The vertices of $P \oplus Q$ are the “sums” of vertices of P and Q.



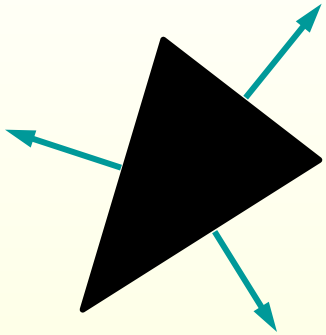
Gauss Map

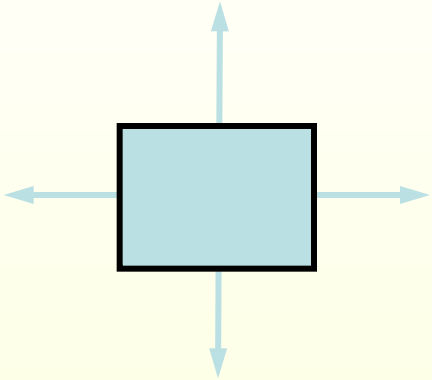
- Gauss map of a convex polygon
  - Edge → point on the circle defined by the normal
  - Vertex → arc defined by its adjacent edges

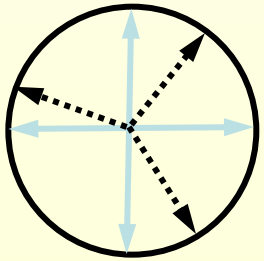


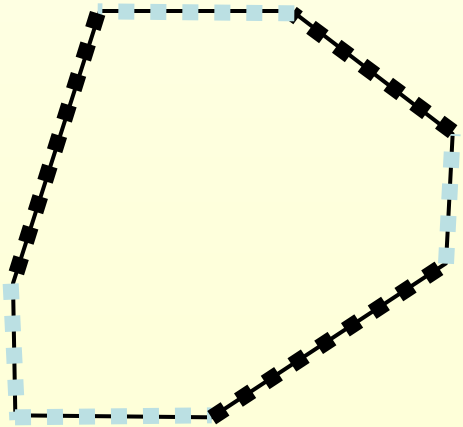
Gauss Map Property of Minkowski Sum

- p+q belongs to the boundary of Minkowski sum only if the Gauss map of p and q overlap.









Computational efficiency

- Running time $O(n+m)$
- Space $O(n+m)$

Minkowski Sum of Non-convex Polygons

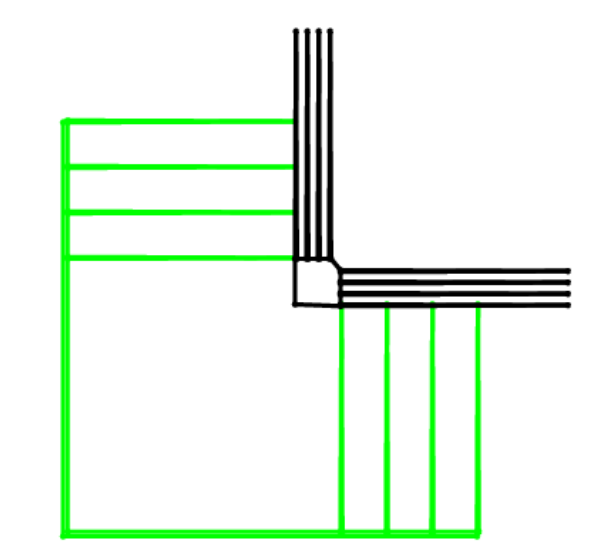
- Decompose into convex polygons (*e.g.*, triangles or trapezoids),
- Compute the Minkowski sums, and
- Take the union

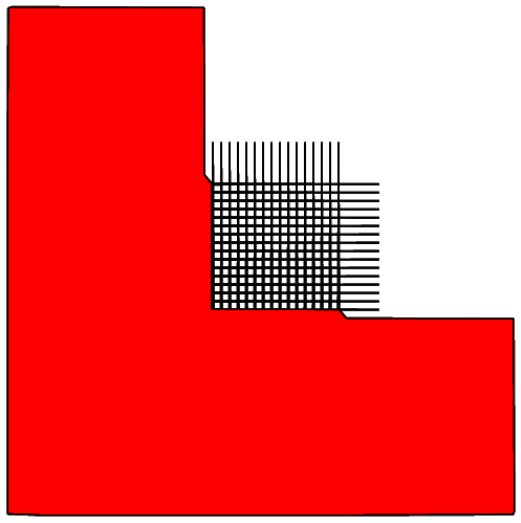
- Complexity of Minkowski sum $O(n^2m^2)$

Worst case example

- $O(n^2m^2)$ complexity

2D example
Agarwal et al. 02





3D Minkowski Sum

- Convex case
  - $O(nm)$ complexity
  - Many methods known for computing Minkowski sum in this case
- Convex hull method
  - Compute sums of all pairs of vertices of *P* and *Q*
  - Compute their convex hull
  - O(mn log(mn)) complexity
- More efficient methods are known [Guibas and Seidel 1987]

3D Minkowski Sum

- Non-convex case
  - $O(n^3m^3)$ complexity
- Computationally challenging
- Common approach resorts to convex decomposition

3D Minkowski Sum Computation

- Two objects *P* and *Q* with m and n convex pieces respectively
  - Compute mn pairwise Minkowski sums between all pairs of convex pieces
  - Compute the union of the pairwise Minkowski sums
- Main bottleneck
  - Union computation
  - mn is typically large (tens of thousands)
  - Union of mn pairwise Minkowski sums has a complexity close to $O(m^3n^3)$
- No practical algorithms known for exact Minkowski sum computation

Minkowski Sum Approximation

- Accurate and efficient approximate algorithm [Varadhan and Manocha 2004]

- Provides certain “geometric” and “topological” guarantees on the approximation
  - Approximation is “close” to the boundary of the Minkowski sum
  - It has the same number of connected components and genus as the exact Minkowski sum

Brake Hub
(4,736 tris)

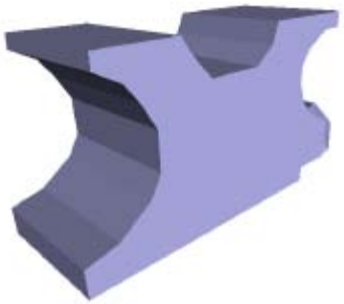
⊕

Rod
(24 tris)

=

Union of
1,777 primitives

Anvil
(144 tris)

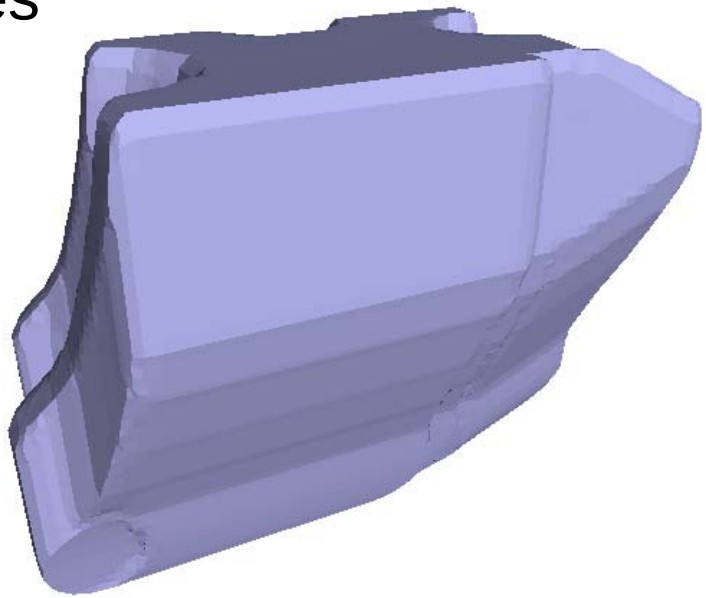


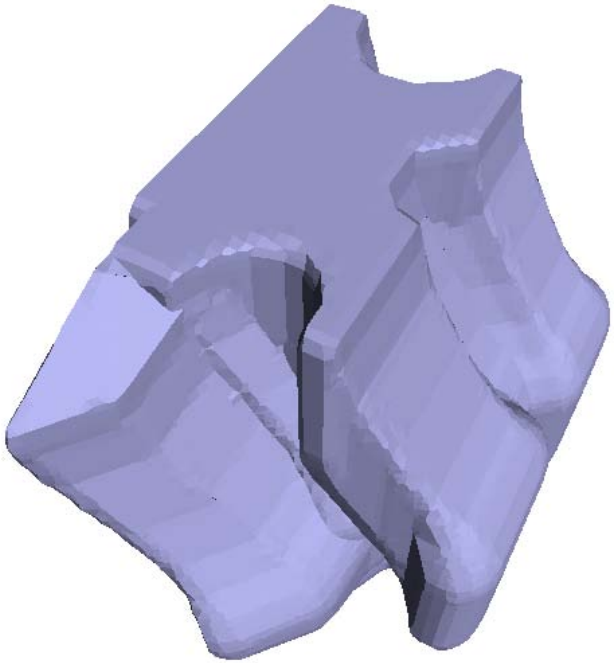
⊕

Spoon
(336 tris)

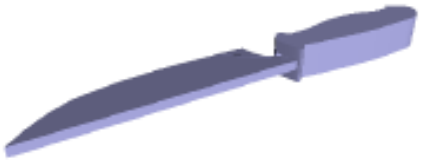
=

Union of
4,446 primitives





Knife
(516 tris)

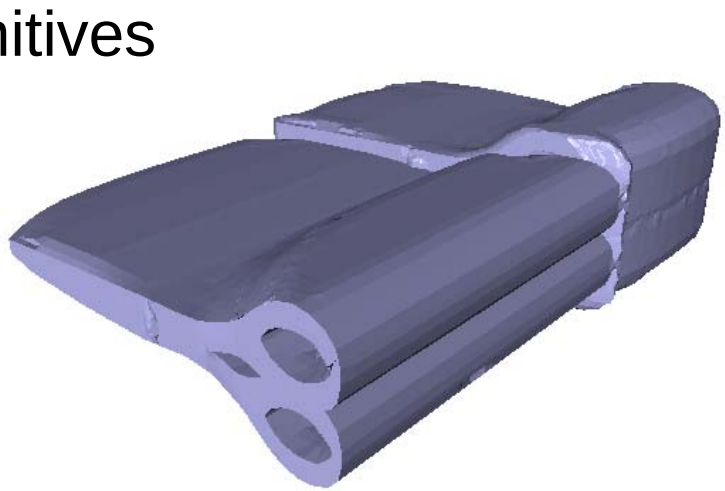


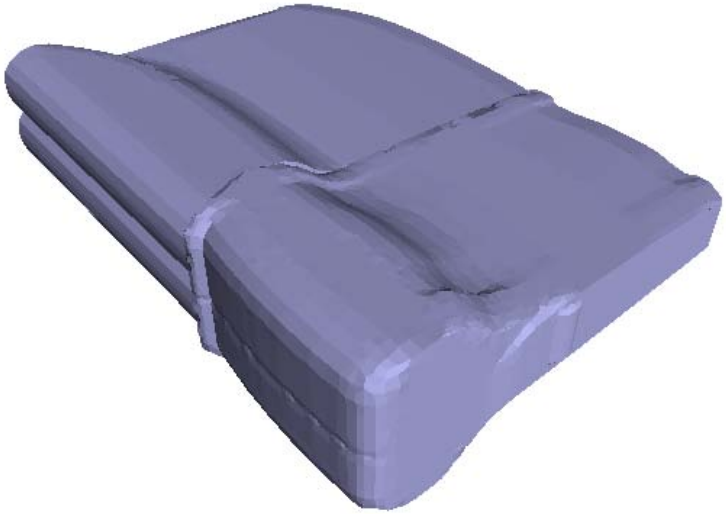
⊕

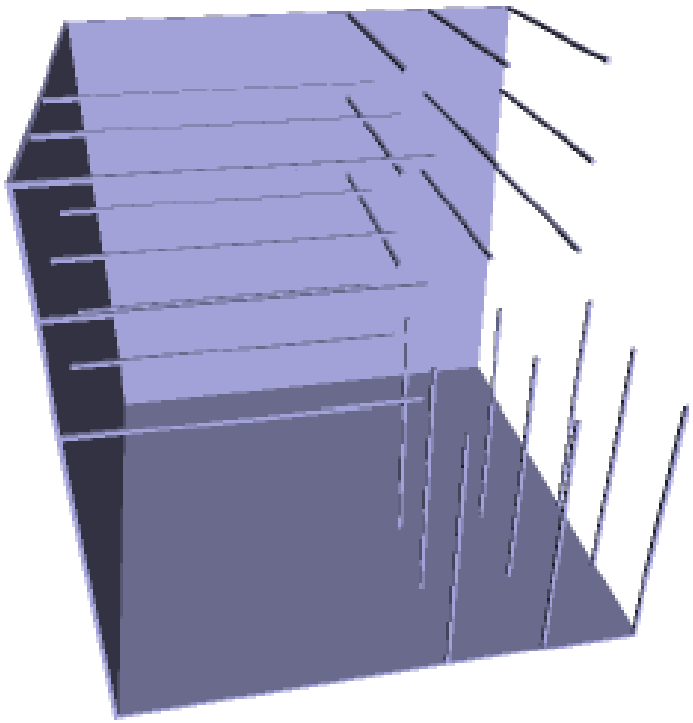
Scissors
(636 tris)

=

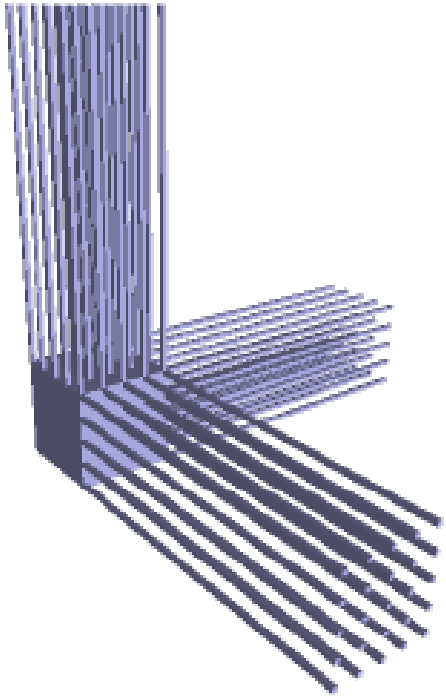
Union of
63,790 primitives



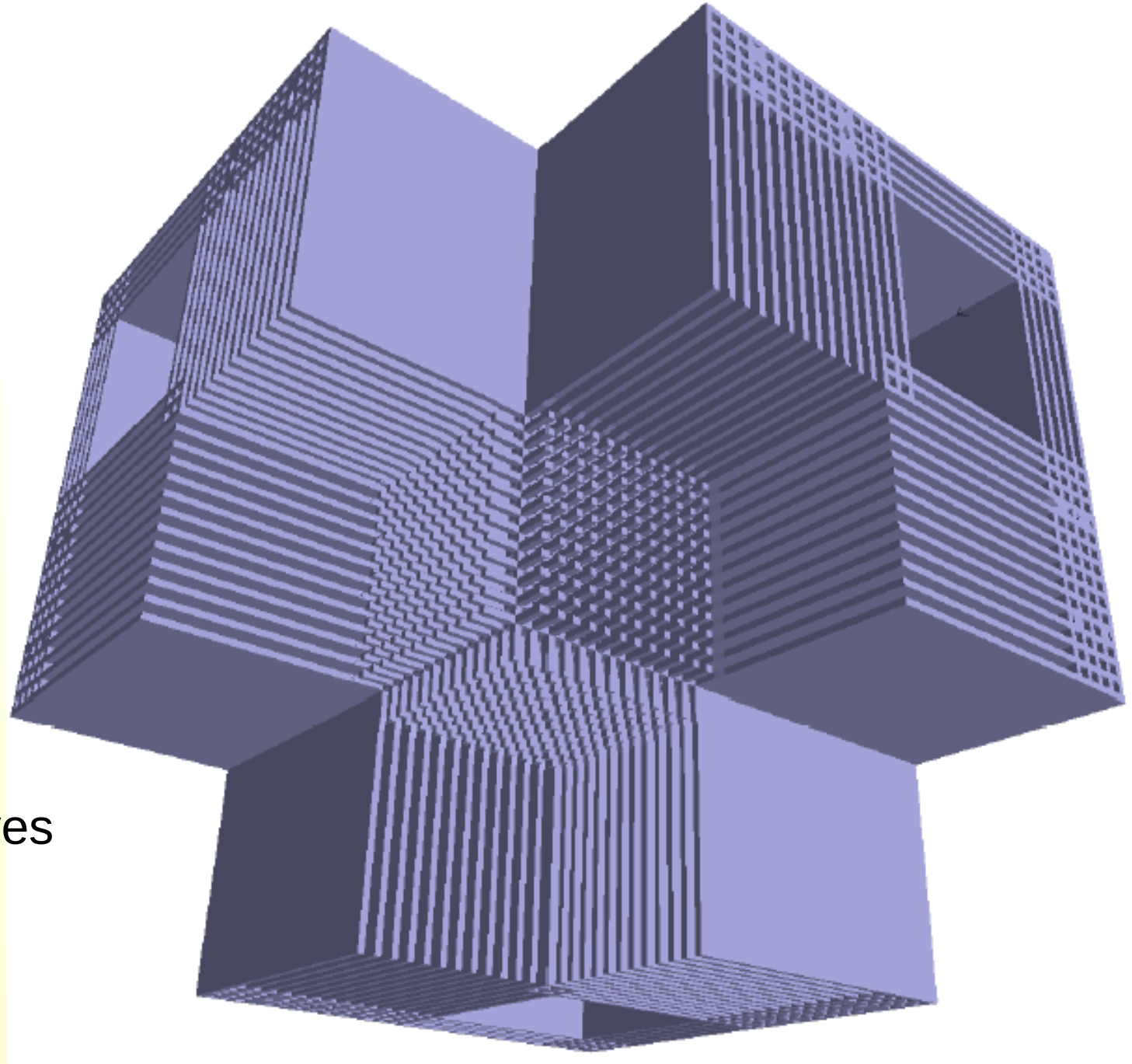




444 tris



1,134 tris



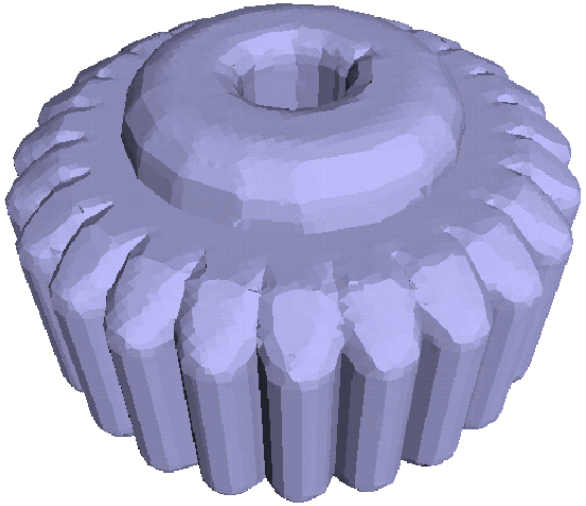
Union of
66,667 primitives

Offsetting

Cup
(1,000 tris)

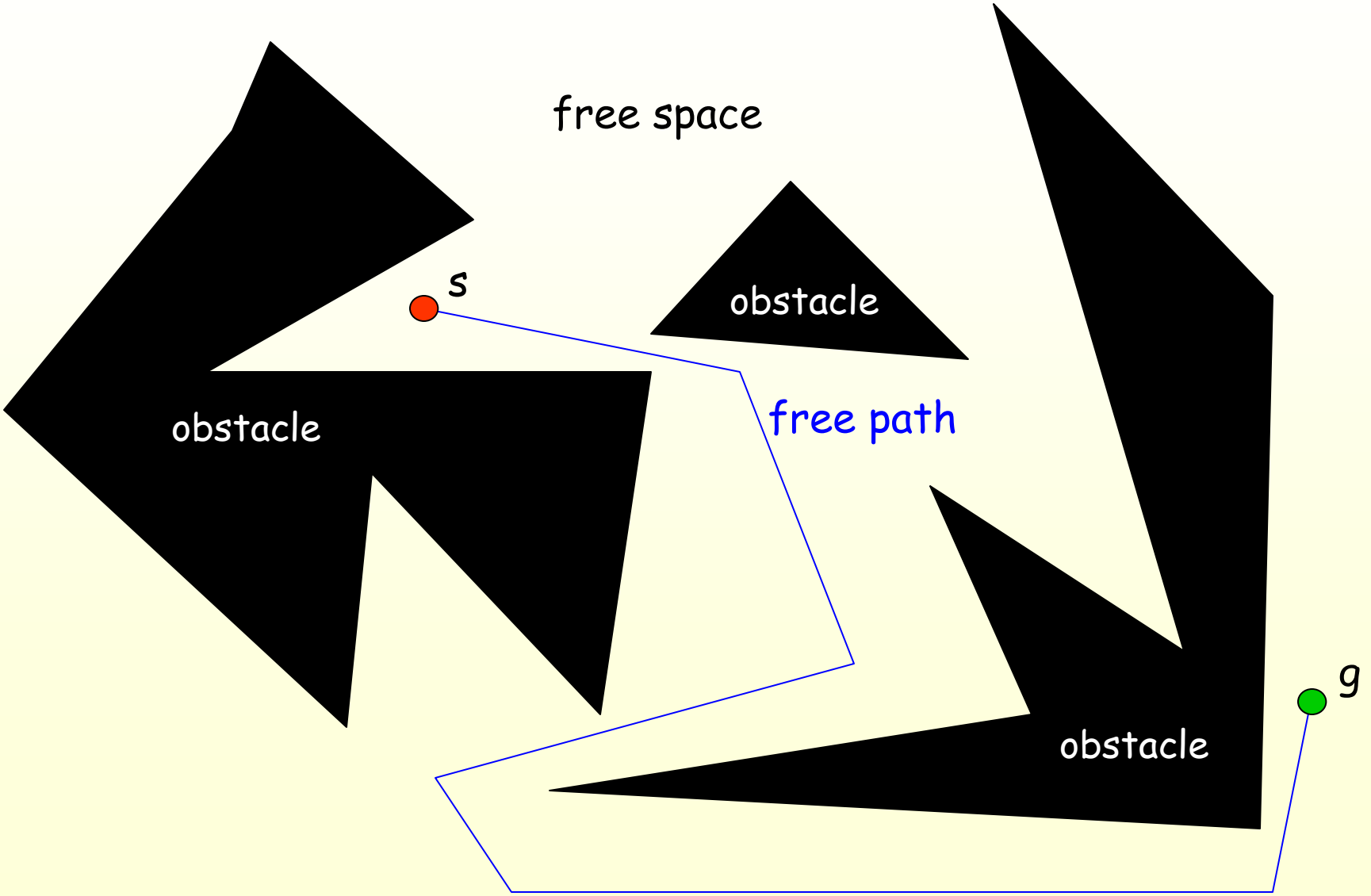
Cup Offset

Gear
2,382 tris)

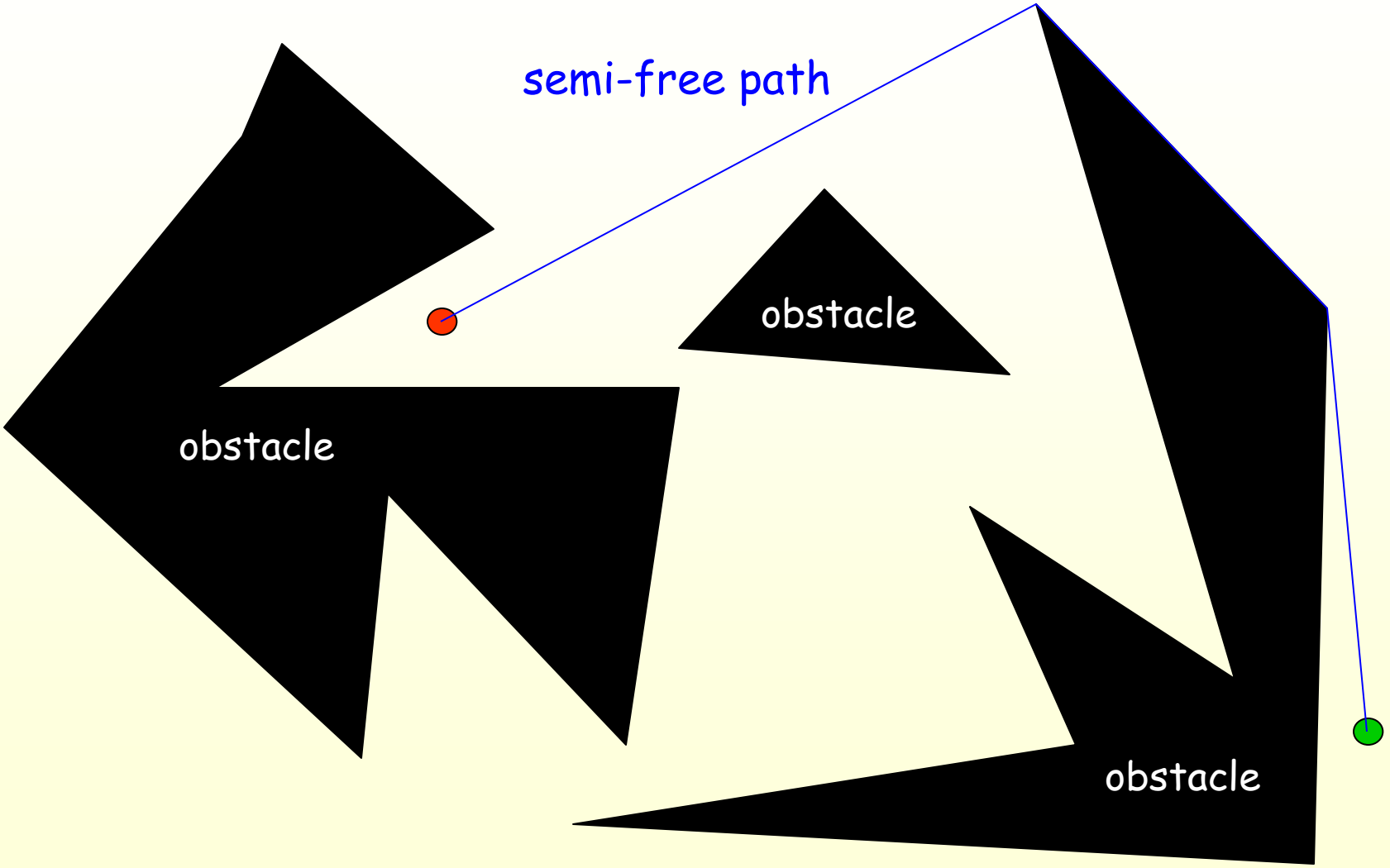


Gear Offset

Motion Planning Problem



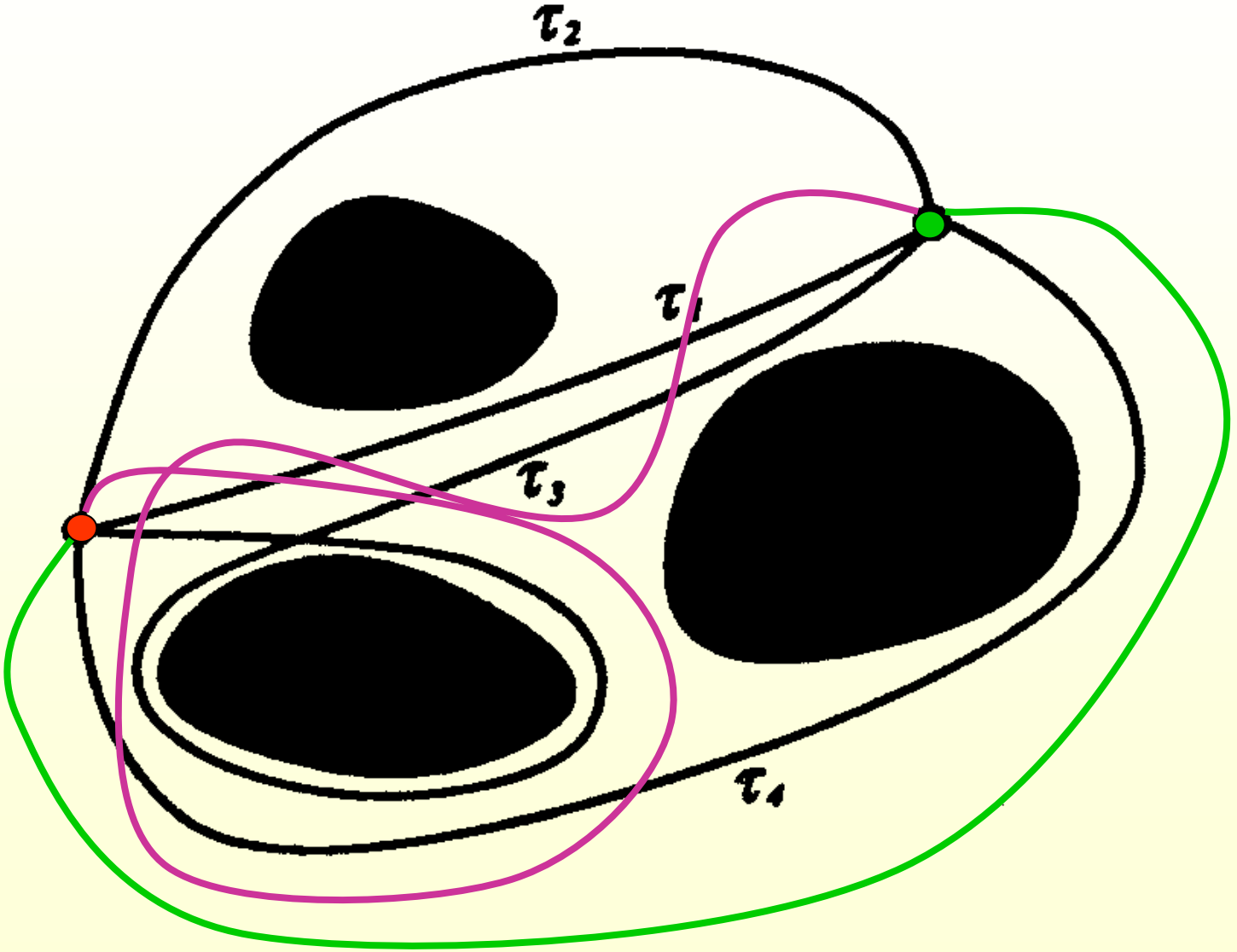
Problem



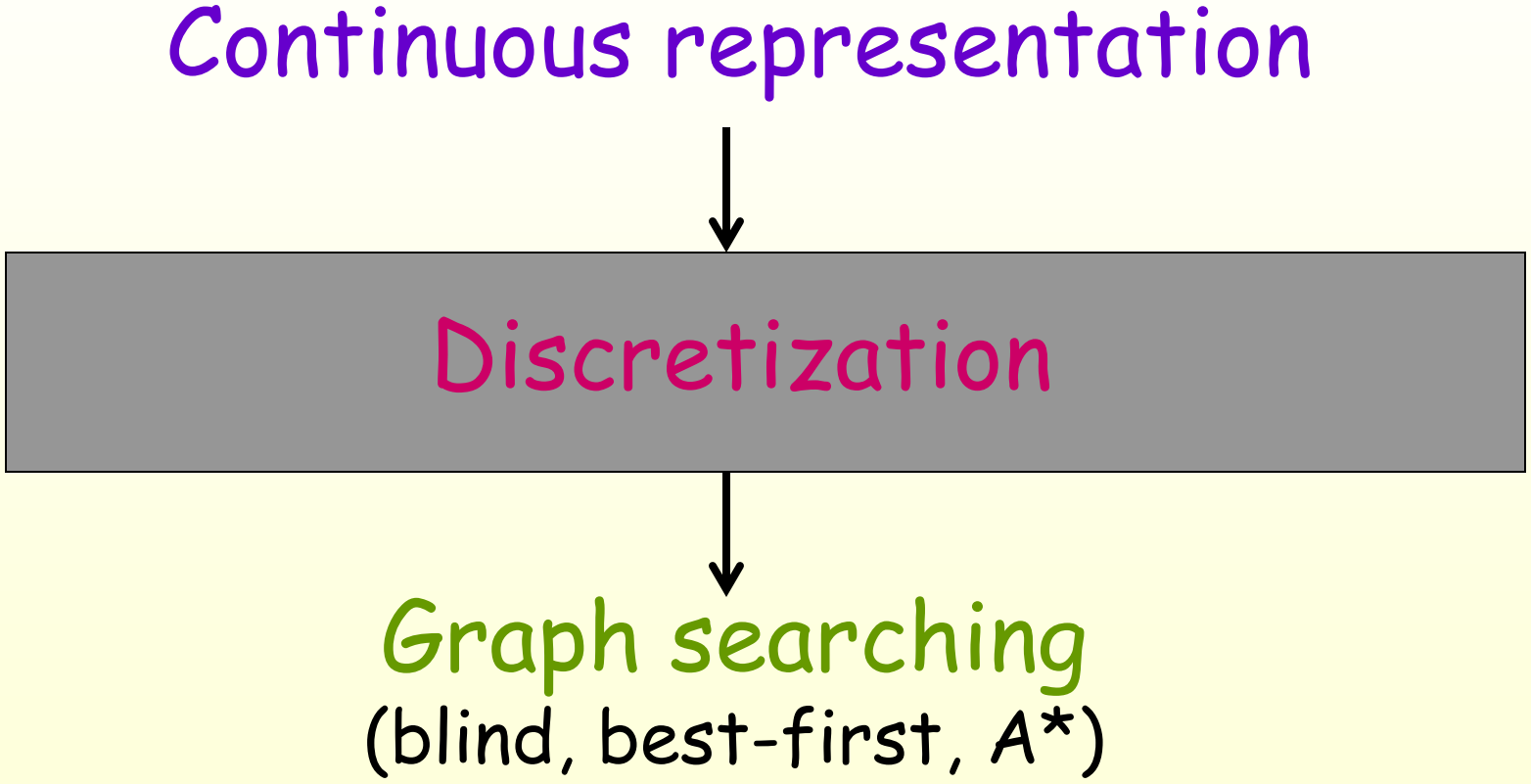
Types of Path Constraints

- Local constraints:
  lie in free space
- Differential constraints:
  have bounded curvature
- Global constraints:
  have minimal length

Homotopy of Free Paths



Motion-Planning Framework

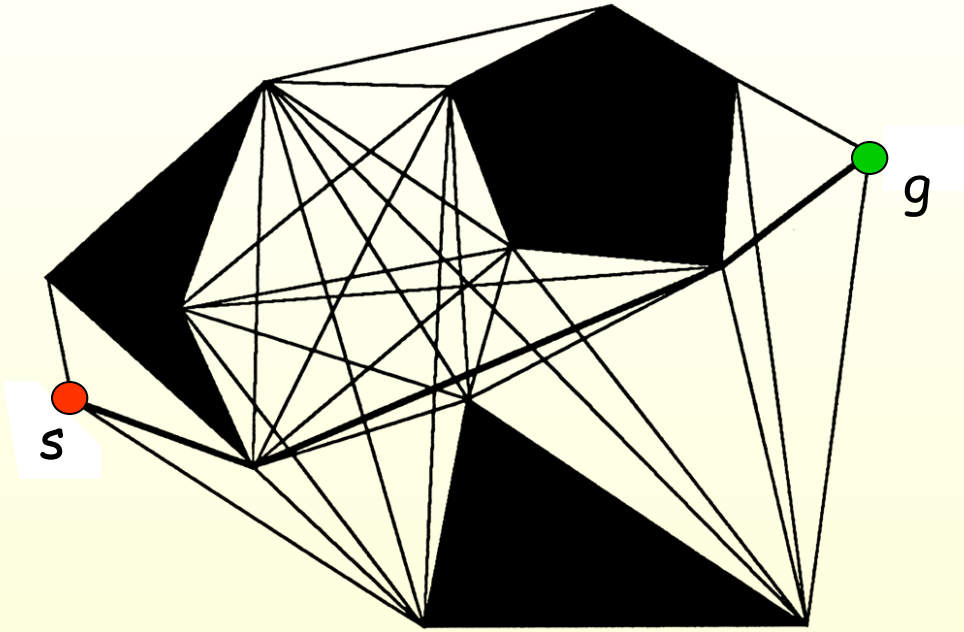


Path-Planning Approaches

1. **Roadmap**
   Represent the connectivity of the free space by a network of 1-D curves
2. **Cell decomposition**
   Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells
3. **Potential field**
   Define a function over the free space that has a global minimum at the goal configuration and follow its steepest descent

Roadmap Methods

- **Visibility graph (VG)**
  Introduced in the Shakey project at SRI in the late 60s. Can produce shortest paths in 2-D configuration spaces



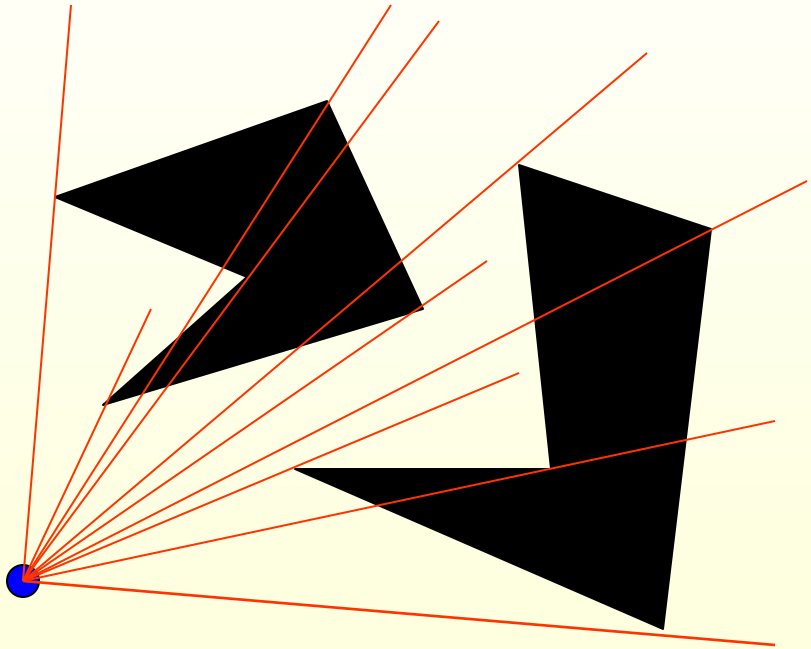
Simple (Naïve) Algorithm

1. Install all obstacles vertices in VG, plus the start and goal positions
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
10. Search VG using A*

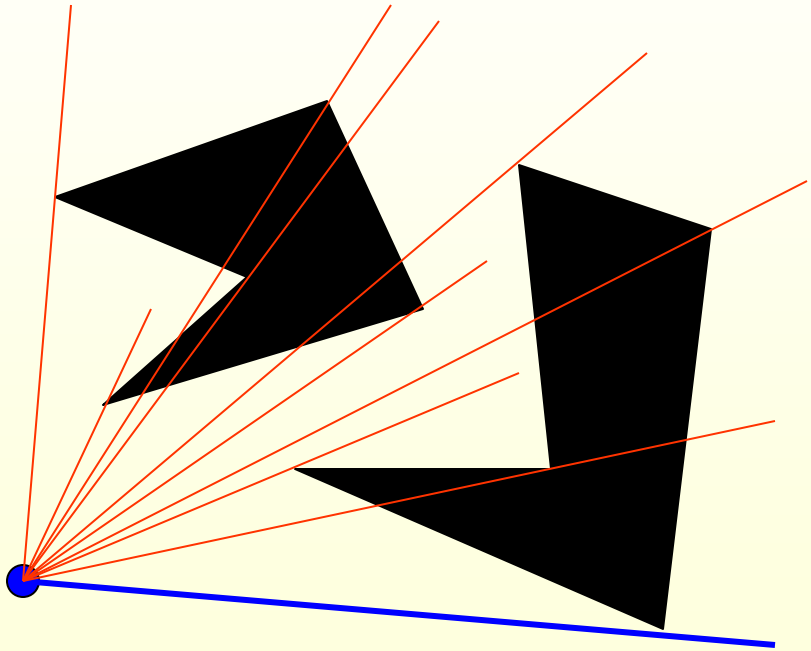
Complexity

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

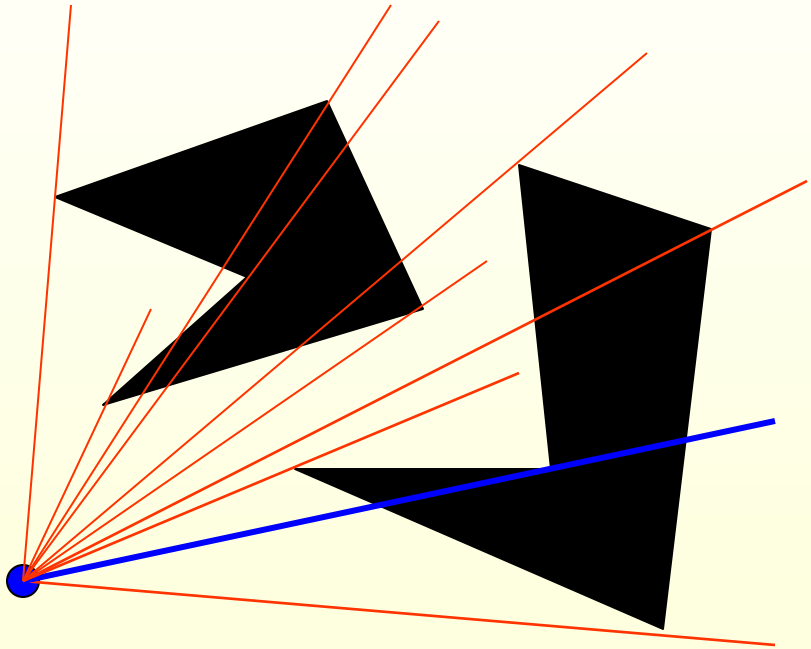
Rotational Sweep



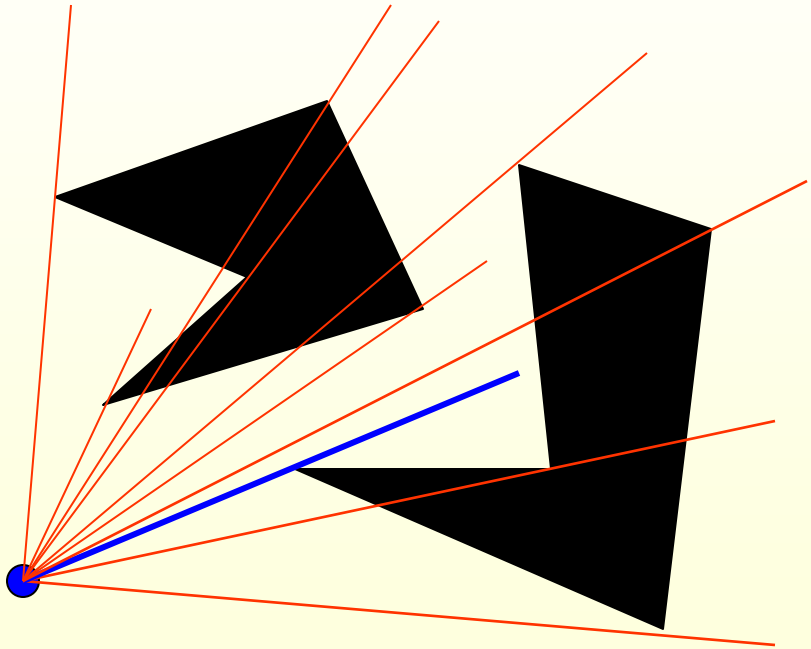
Rotational Sweep



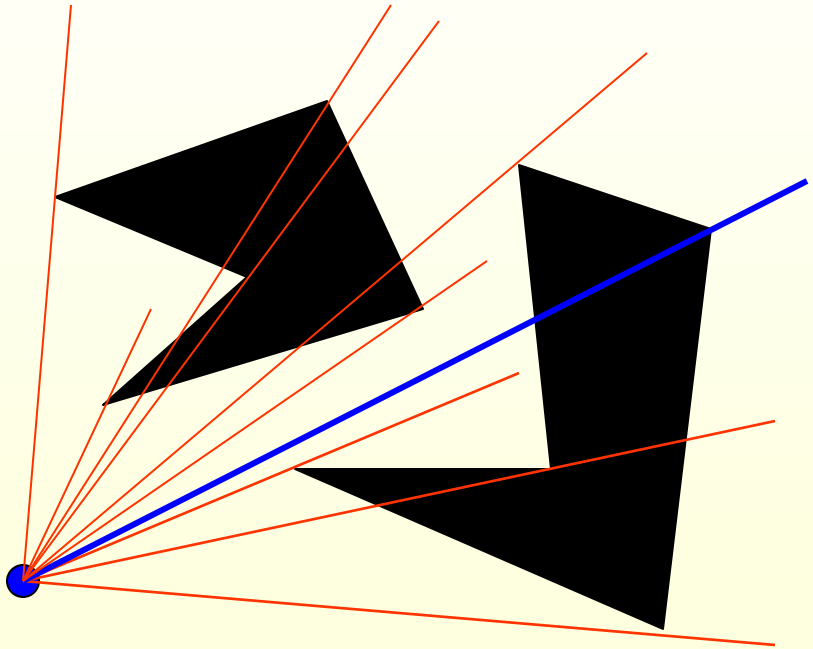
Rotational Sweep



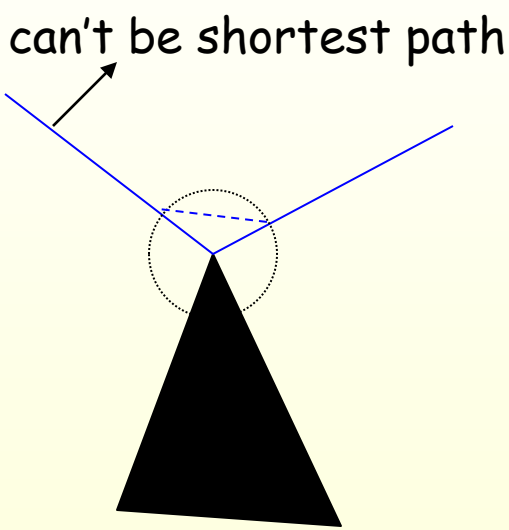
Rotational Sweep

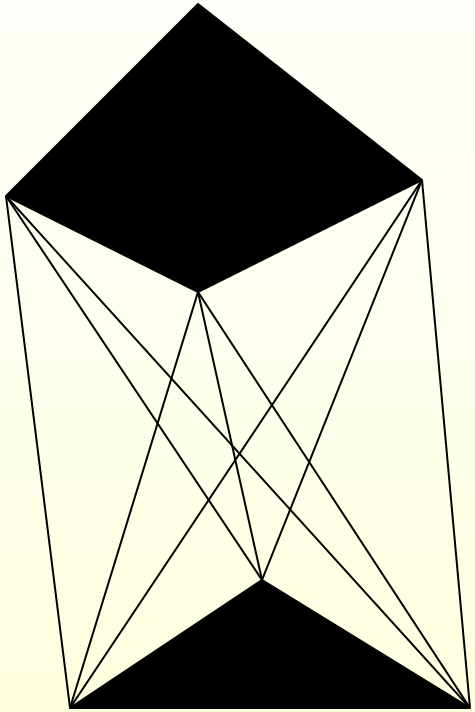


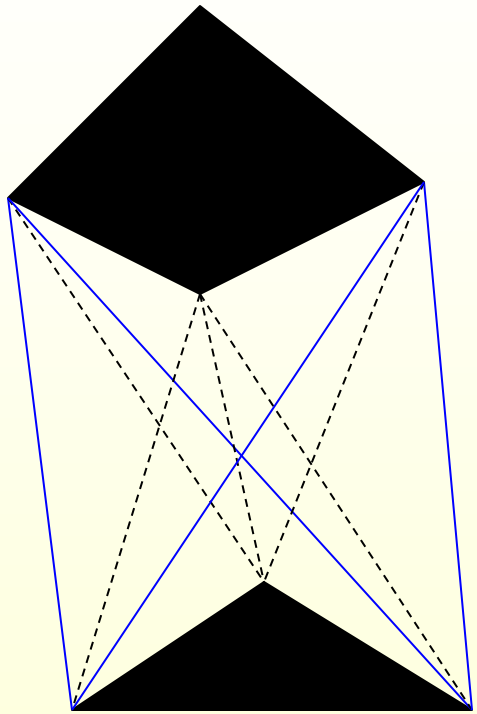
Rotational Sweep



Reduced Visibility Graph



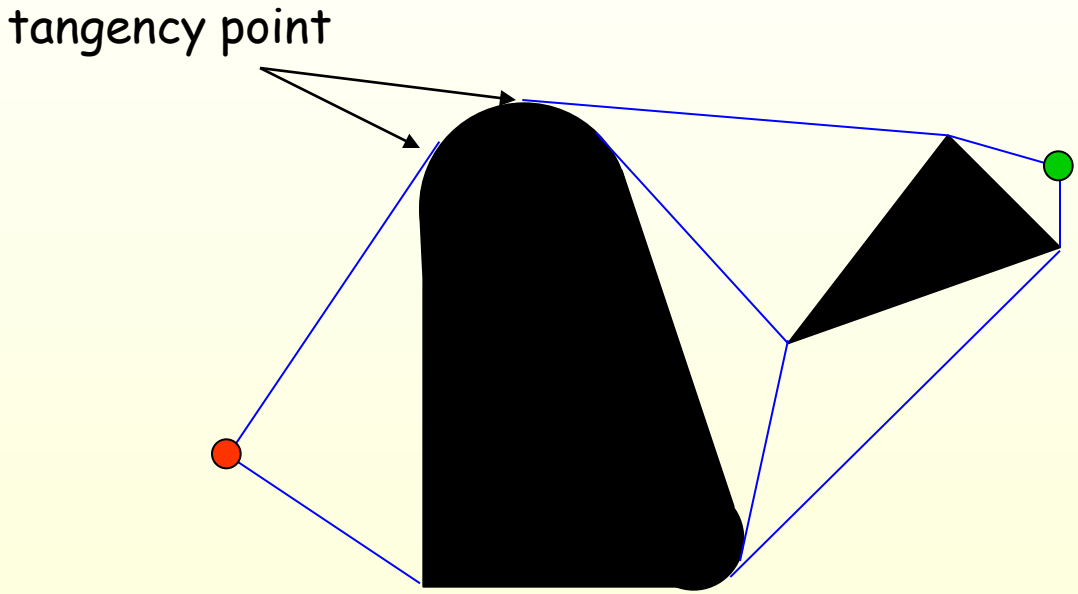




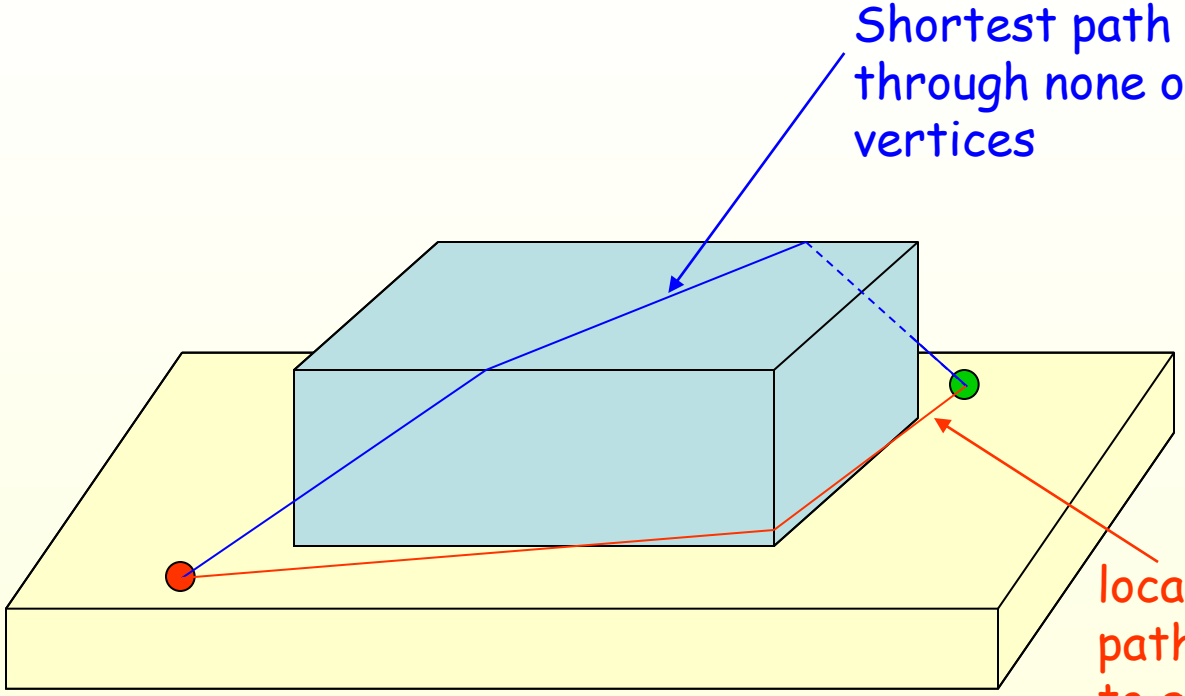
tangent segments

→ Eliminate concave obstacle vertices

Generalized (Reduced) Visibility Graph



Three-Dimensional Space

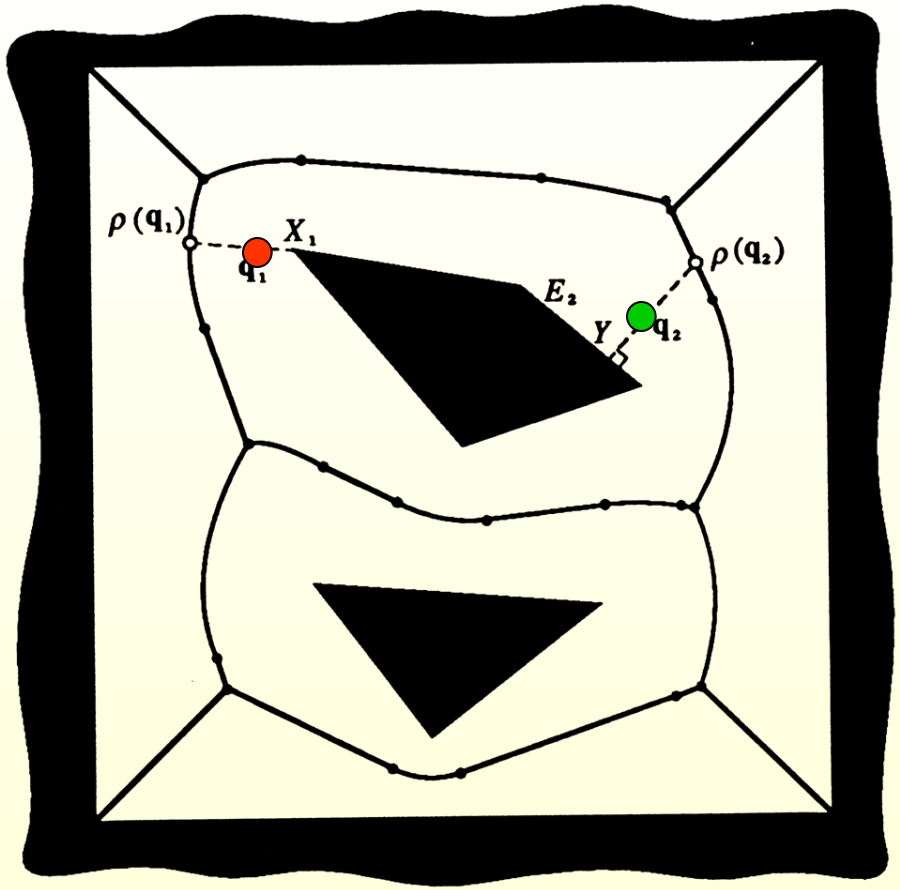


Computing the shortest collision-free path in a polyhedral space is NP-hard

Roadmap Methods

- **Voronoi diagram**
  Introduced by Computational Geometry researchers. Generate paths that maximizes clearance.

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



Roadmap Methods

- **Visibility graph**
- **Voronoi diagram**
- **Silhouette**

  First complete general method that applies to spaces of any dimension and is singly exponential in # of dimensions [Canny, 87]
- **Probabilistic roadmaps**

Probabilistic RoadMaps (PRM)

Probabilistic Roadmap Method

Construction ($G = V, E$)

Loop

$c \leftarrow$ a free sample

add c to the vertices V

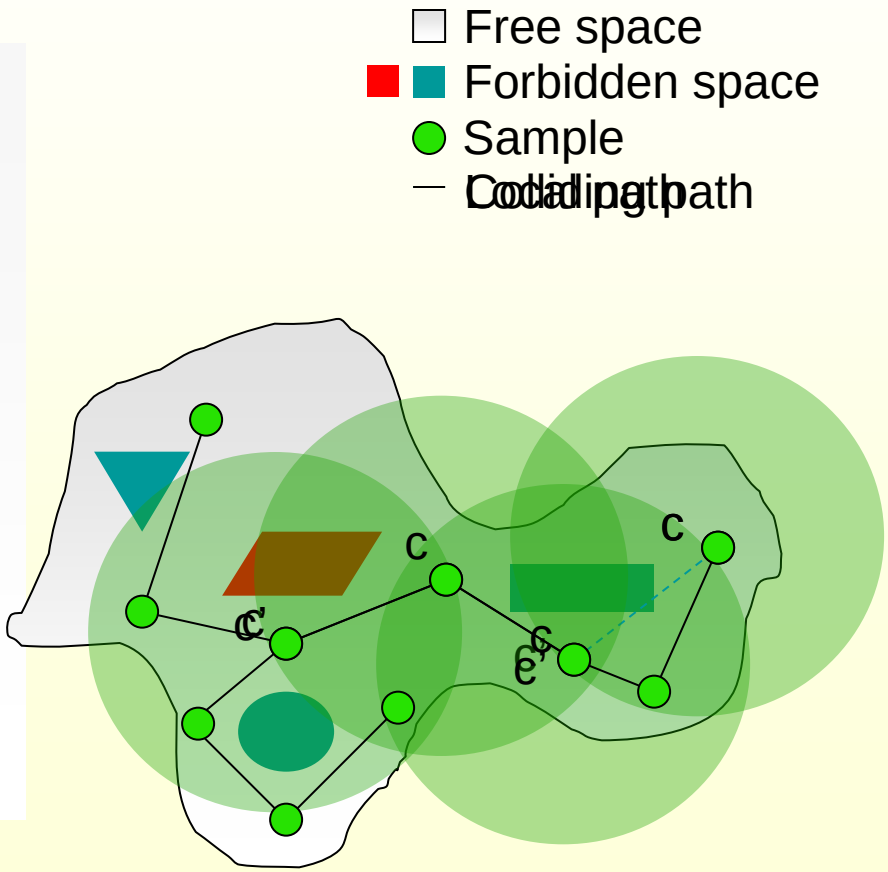
$N_c \leftarrow$ a set of nodes

for all c' in N_c in increasing distance

if c' and c are not connected in G **then**

if local path between c and c' exists **then**

add the edge $c'c$ to E



Probabilistic Roadmap Method

Construction ($G = V, E$)

Loop

$c \leftarrow$ a free sample

add c to the vertices V

$N_c \leftarrow$ a set of nodes

for all c' in N_c in increasing distance

if c' and c are not connected in G **then**

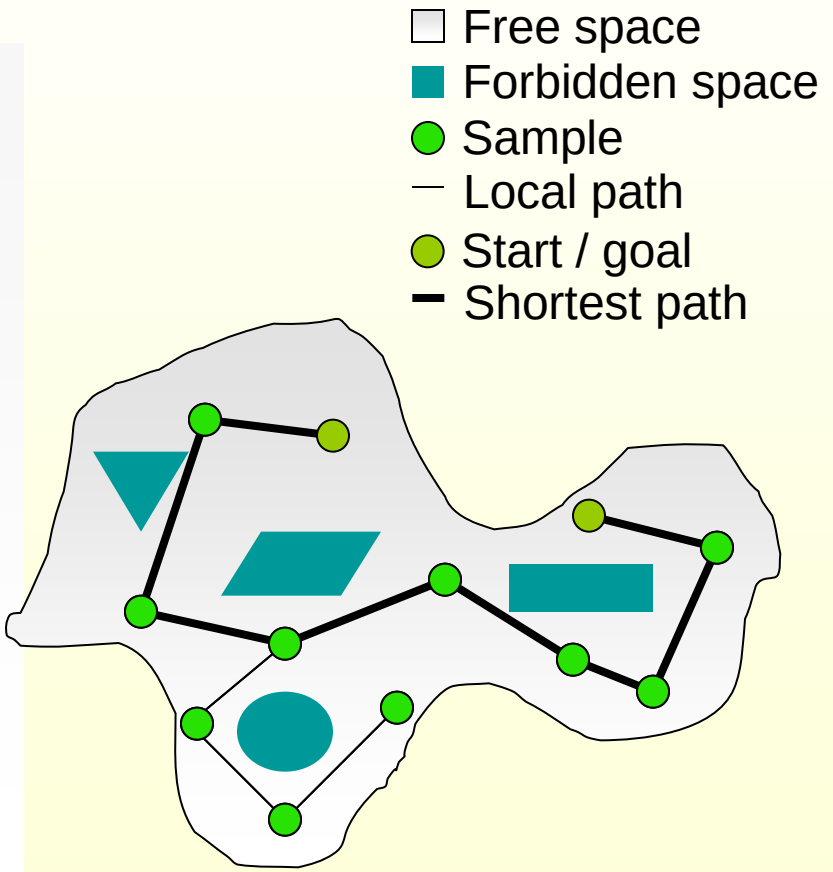
if local path between c and c' exists **then**

add the edge $c'c$ to E

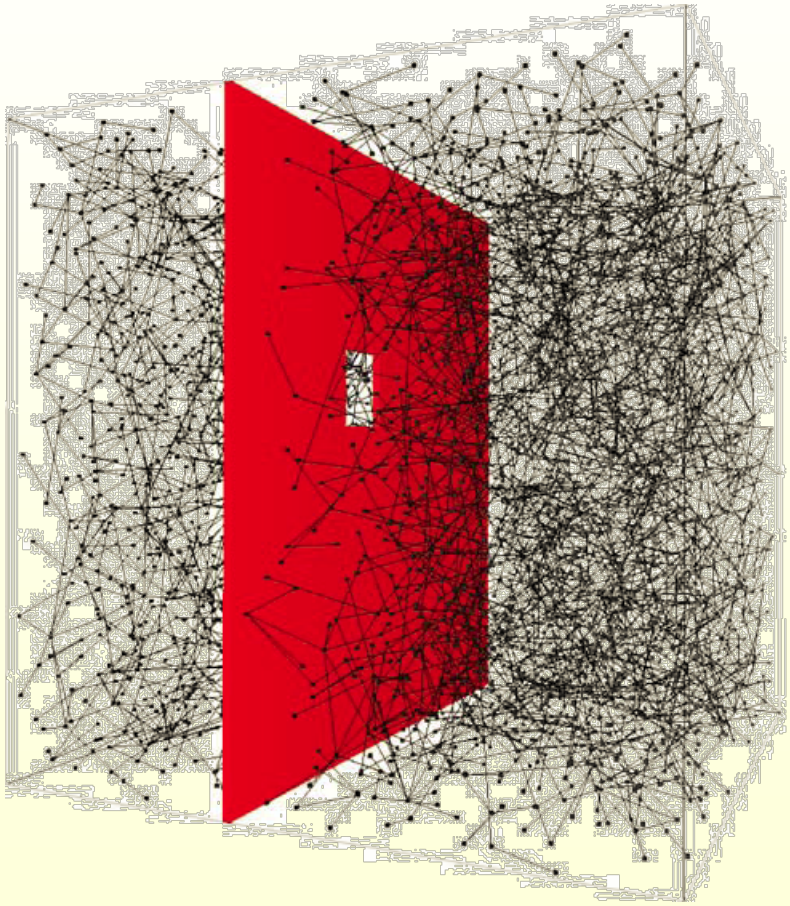
Query

connect sample s and g to roadmap

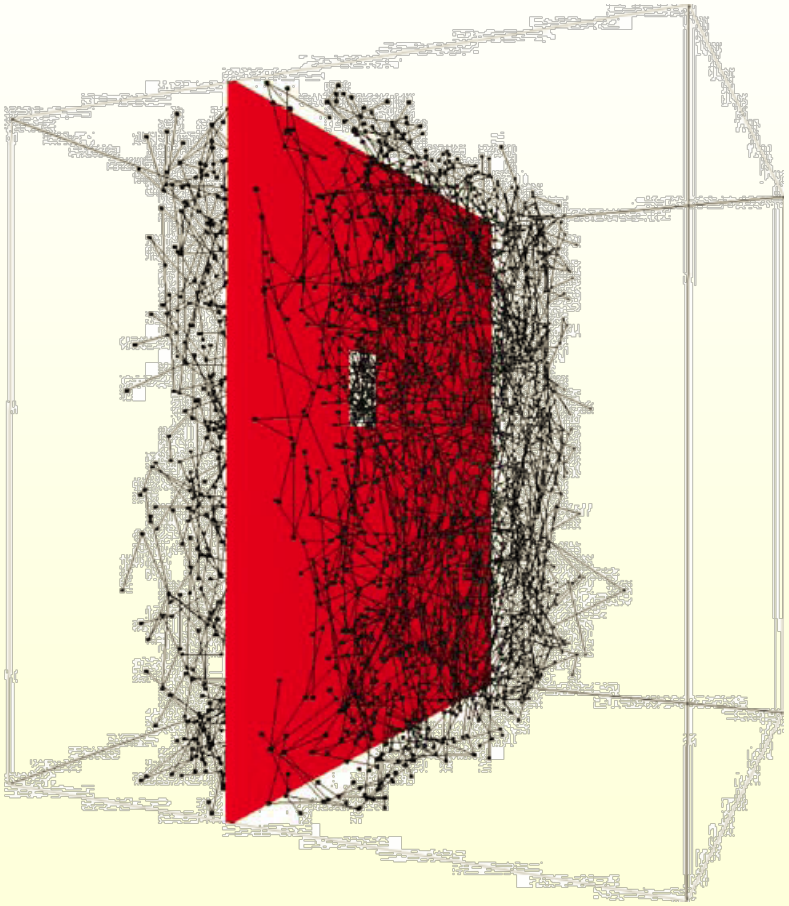
Dijkstra's shortest path



PRM Example



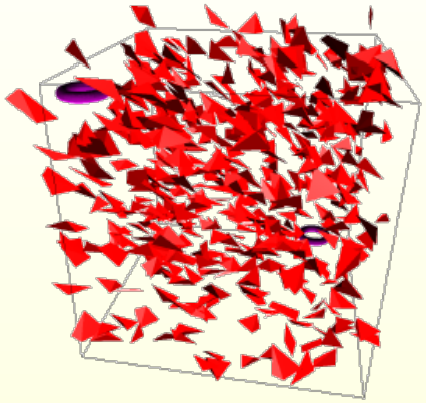
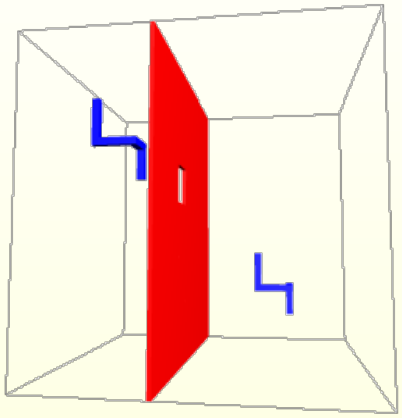
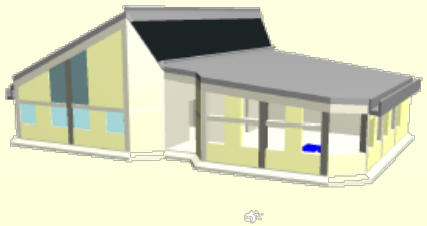
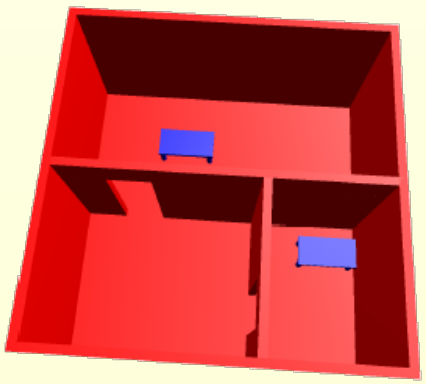
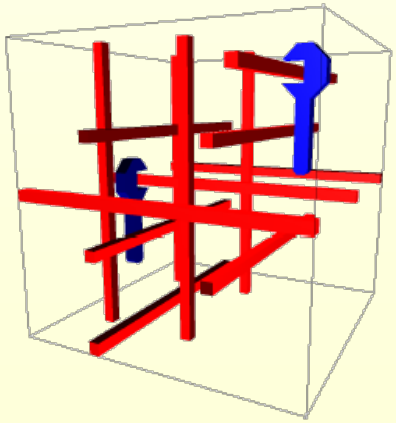
Halton Sampling



Gaussian Sampling

Test Scenes

- Comparison of techniques

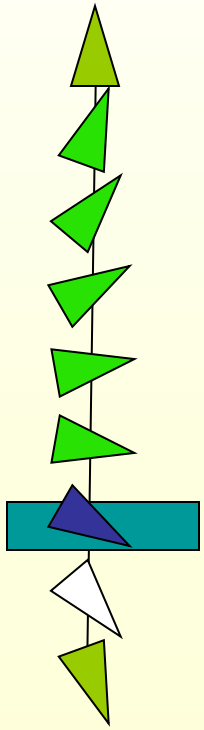







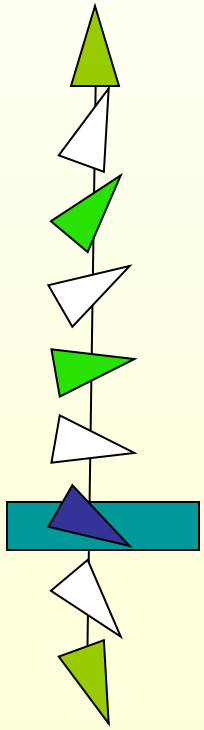
PRM Choices

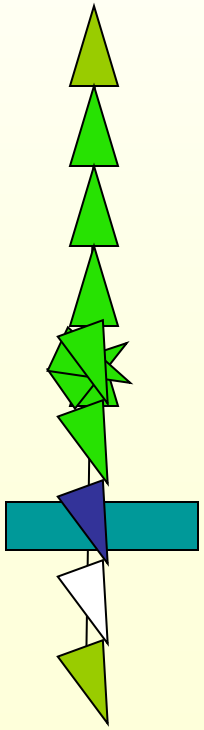
- Collision checking on local path
  - Incremental
  - Binary
  - Rotate-at-*s*
- Neighbor selection
  - Nearest-*k*
  - Component-*k*
  - Visibility

Collision Checking

- Incremental
- Binary
- Rotate-at-s







Collision Checking – Results

	incremental	binary	rotate-at-*s*
cage	2.4	1.9	2.7
clutter	1.8	1.3	3.1
hole	431.9	422.3	1206.7
house	6.4	4.9	47.1
rooms	0.5	0.4	1.1
wrench	0.9	0.5	1.1

Neighbor Strategy

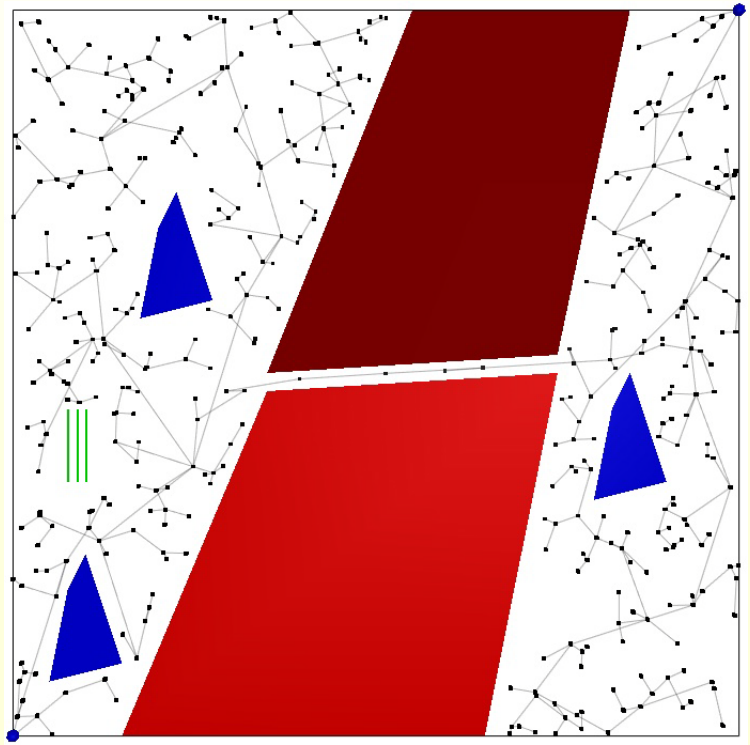
- Nearest-*k*
  - Nearest *k* neighbors
- Component
  - Nearest neighbor in each connected component
- Component-*k*
  - Nearest *k* neighbors in each connected component
- Visibility
  - Only useful nodes, add node/connection only if:
    - Node cannot be connected to other nodes
    - Node can be connected to two (or more) connected components

Neighbor Strategy – Results

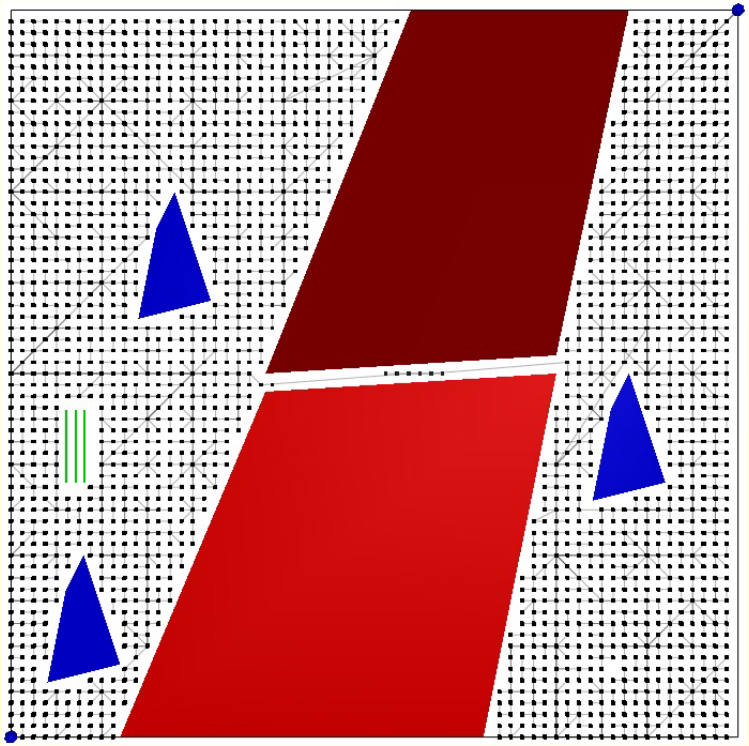
	nearest-*k*	comp.	comp.-*k*	visibility
cage	1.9	3.4	1.6	3.0
clutter	1.3	1.3	1.4	2.3
hole	409.5	7428.2	7554.4	102.5
house	4.9	3.0	13.0	45.5
rooms	0.4	0.2	0.2	6.3
wrench	0.5	0.4	0.4	1.6

Sampling

- Random

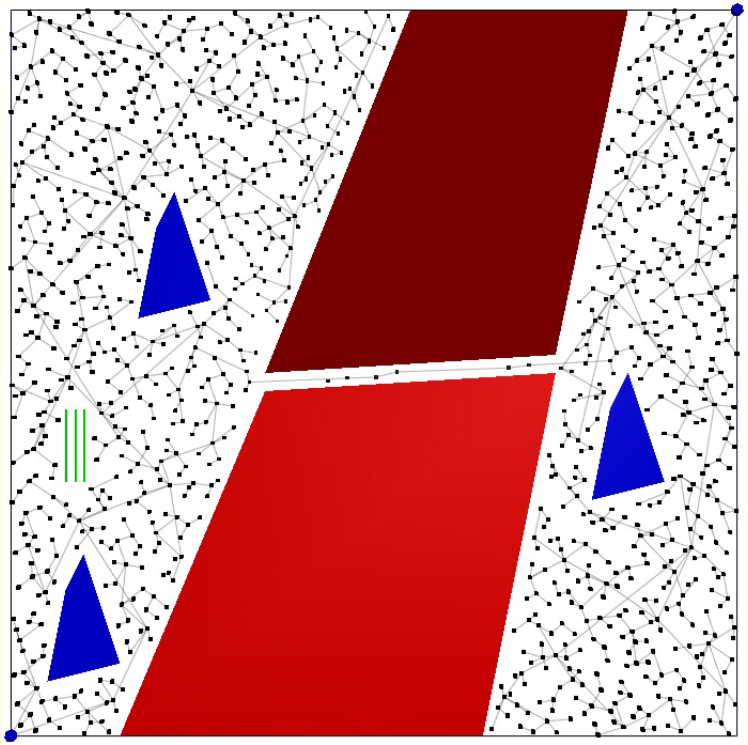


- Grid

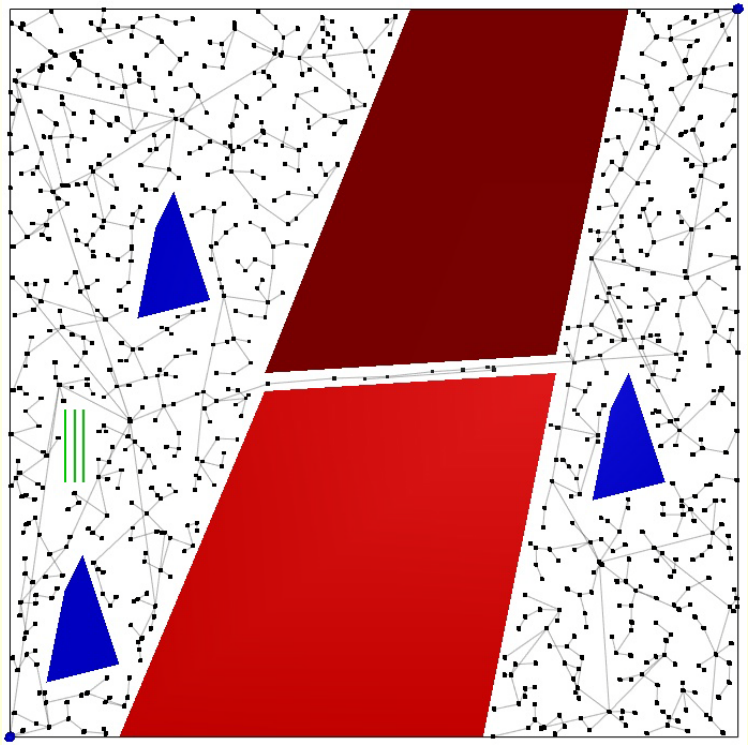


Sampling

- Halton



- Cell-based



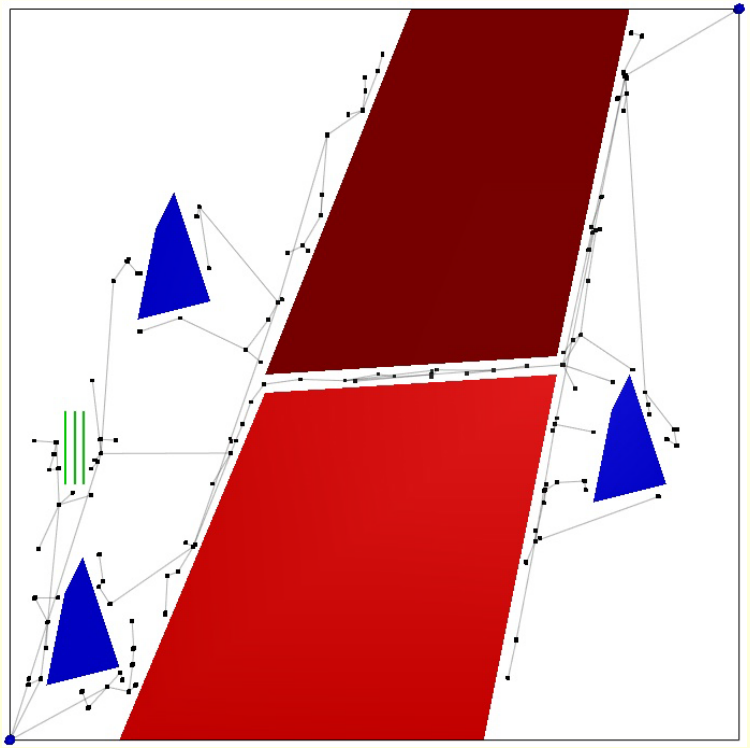
Sampling

- Basic sampling strategy

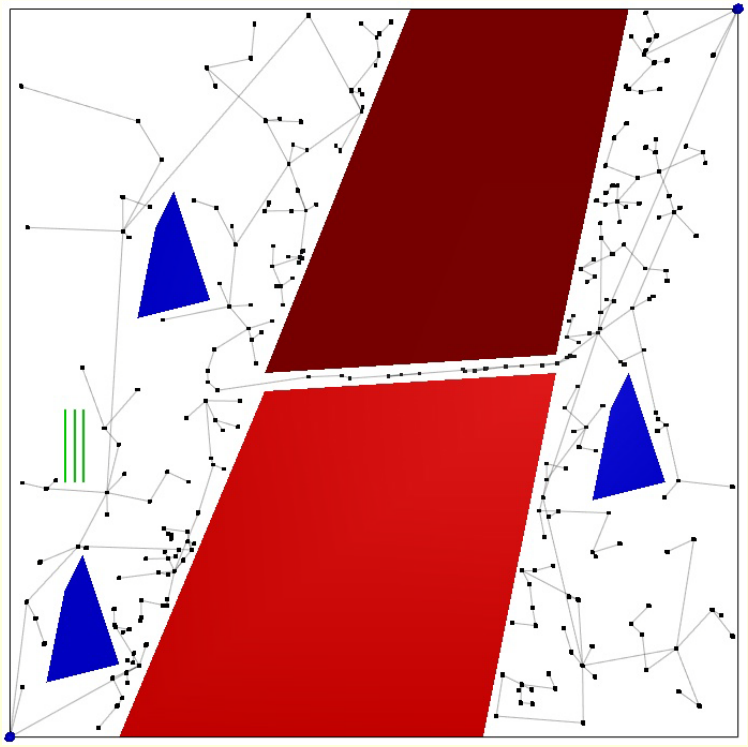
	random	grid	halton	cell-based
Corridor	0.4	1.4	0.2	0.4
Rooms	0.8	0.6	0.3	0.7
Clutter	2.7	7.3	4.0	3.1
Hole	33.9	26.2	43.3	42.3
House	210.0	262.5	207.4	225.9

Sampling

- Gaussian



- Obstacle-based



Sampling

- Sampling around obstacles

	gaussian	obstacle	obstacle*	halton
Corridor	0.4	0.5	0.5	0.3
Rooms	0.4	0.7	0.5	0.2
Clutter	5.6	3.7	7.9	2.2
Hole	3.1	8.0	2.1	15.8
House	268.0	197.1	209.4	152.9

Node Addition

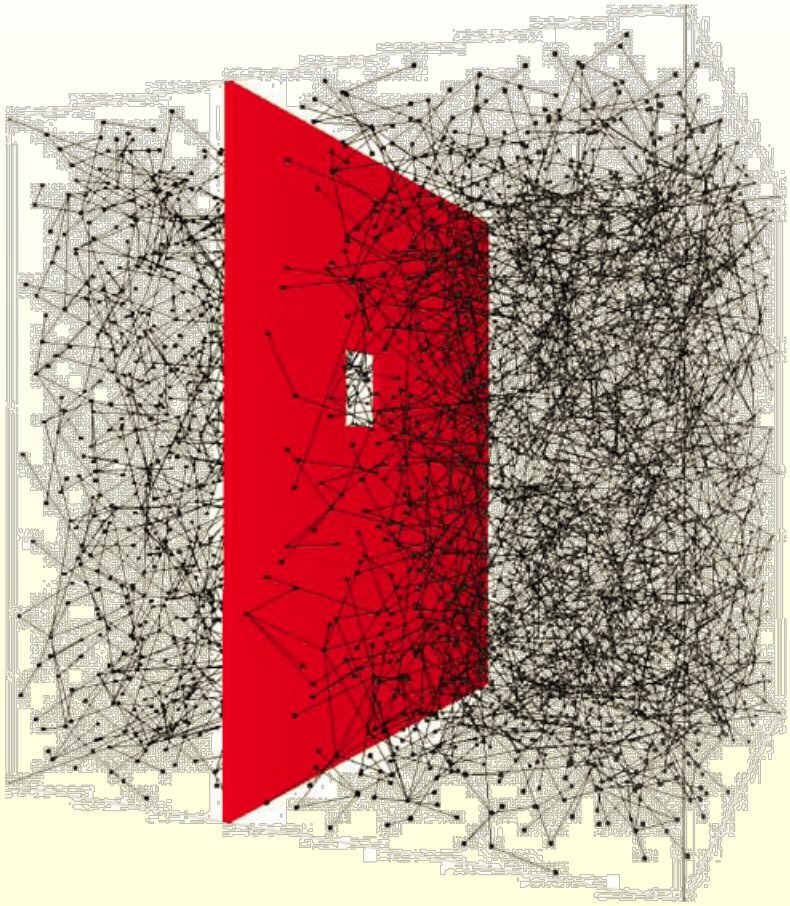
- Nearest-*k*
  - Connect to nearest *k* nodes in graph
- Component
  - Connect to nearest nodes in connected component
- Component-*k*
  - Idem, but connect to at most *n* nodes in each cc
- Visibility
  - Connect to *useful* nodes

Node Addition

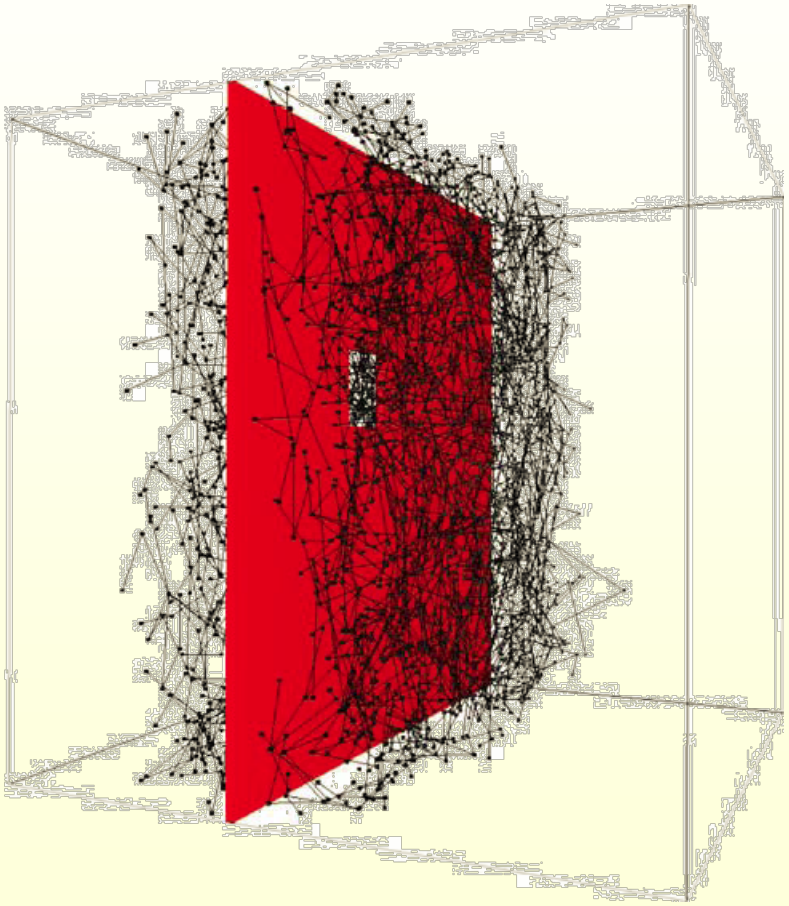
- Node addition strategy

	nearest-*k*	comp	comp-*k*	visibility
Corridor	0.2	0.8	0.5	1.5
Rooms	0.3	0.5	0.7	2.3
Clutter	4.0	4.4	3.4	7.6
Hole	43.3	>120	31.9	20.9
House	207.4	279.6	189.1	>600

PRM Final



Halton Sampling



Gaussian Sampling

Path-Planning Approaches

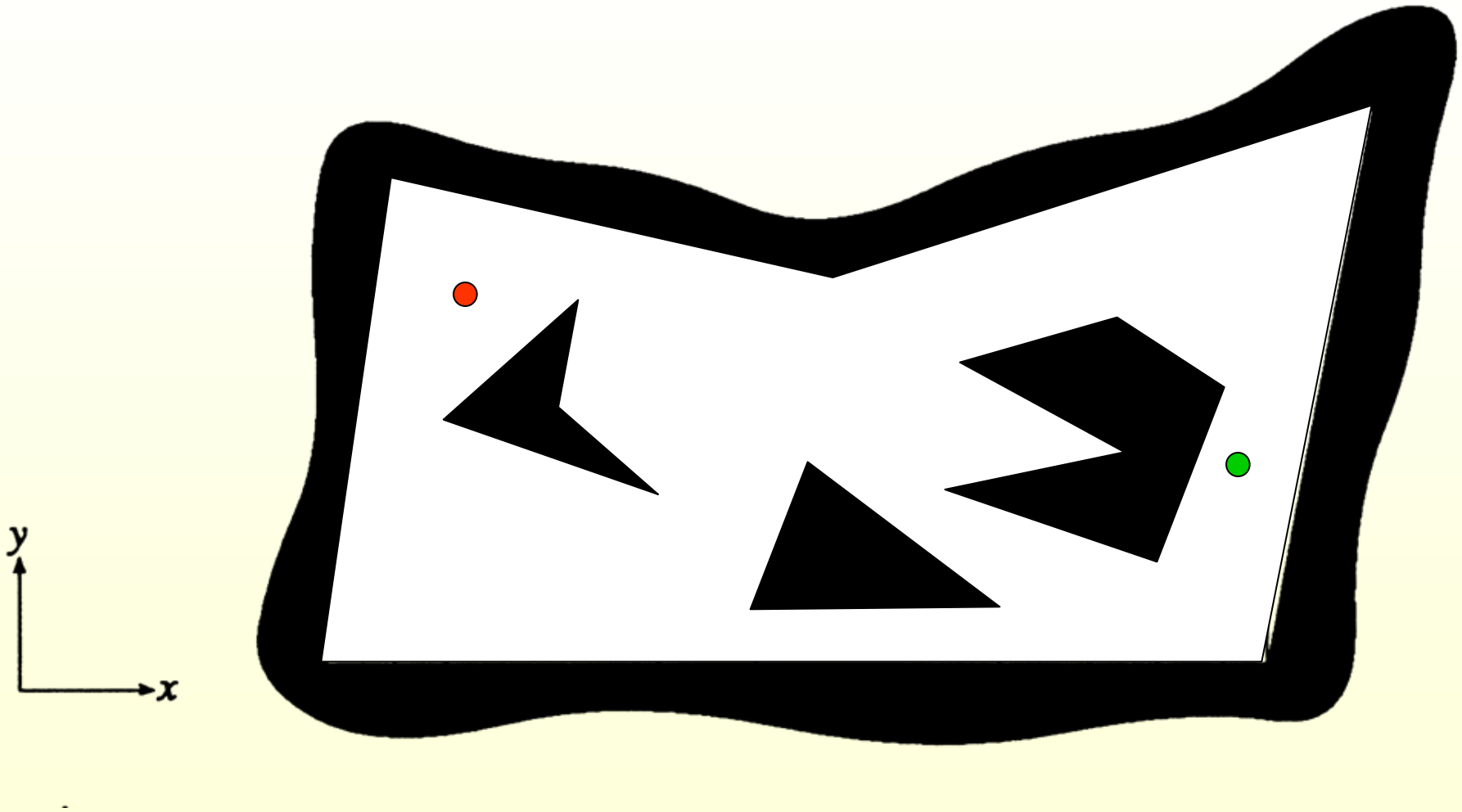
1. **Roadmap**
   Represent the connectivity of the free space by a network of 1-D curves
2. **Cell decomposition**
   Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells
3. **Potential field**
   Define a function over the free space that has a global minimum at the goal configuration and follow its steepest descent

Cell-Decomposition Methods

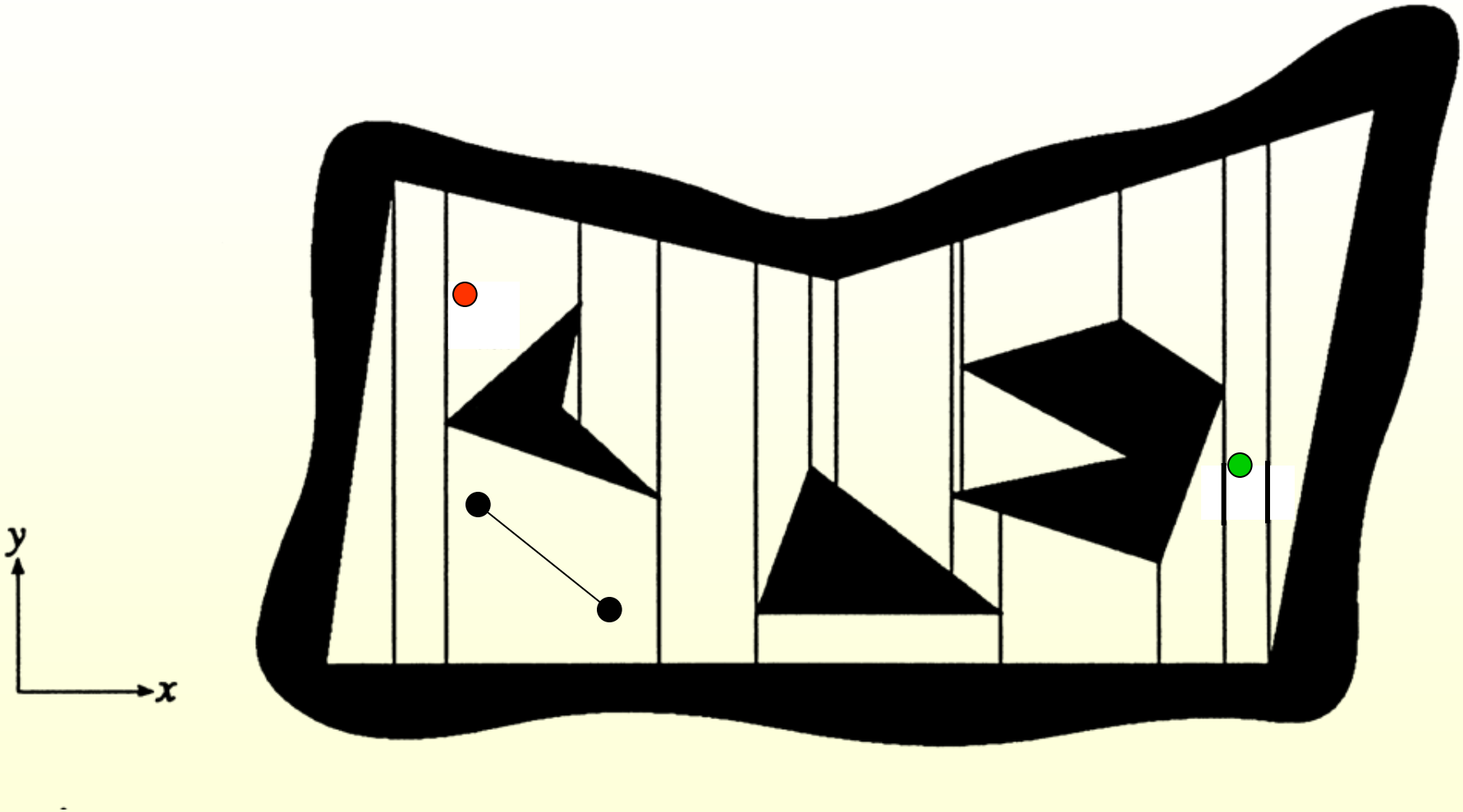
Two classes of methods:

- Exact cell decomposition
  The free space F is represented by a collection of non-overlapping cells whose union is **exactly** F
  Example: trapezoidal decomposition

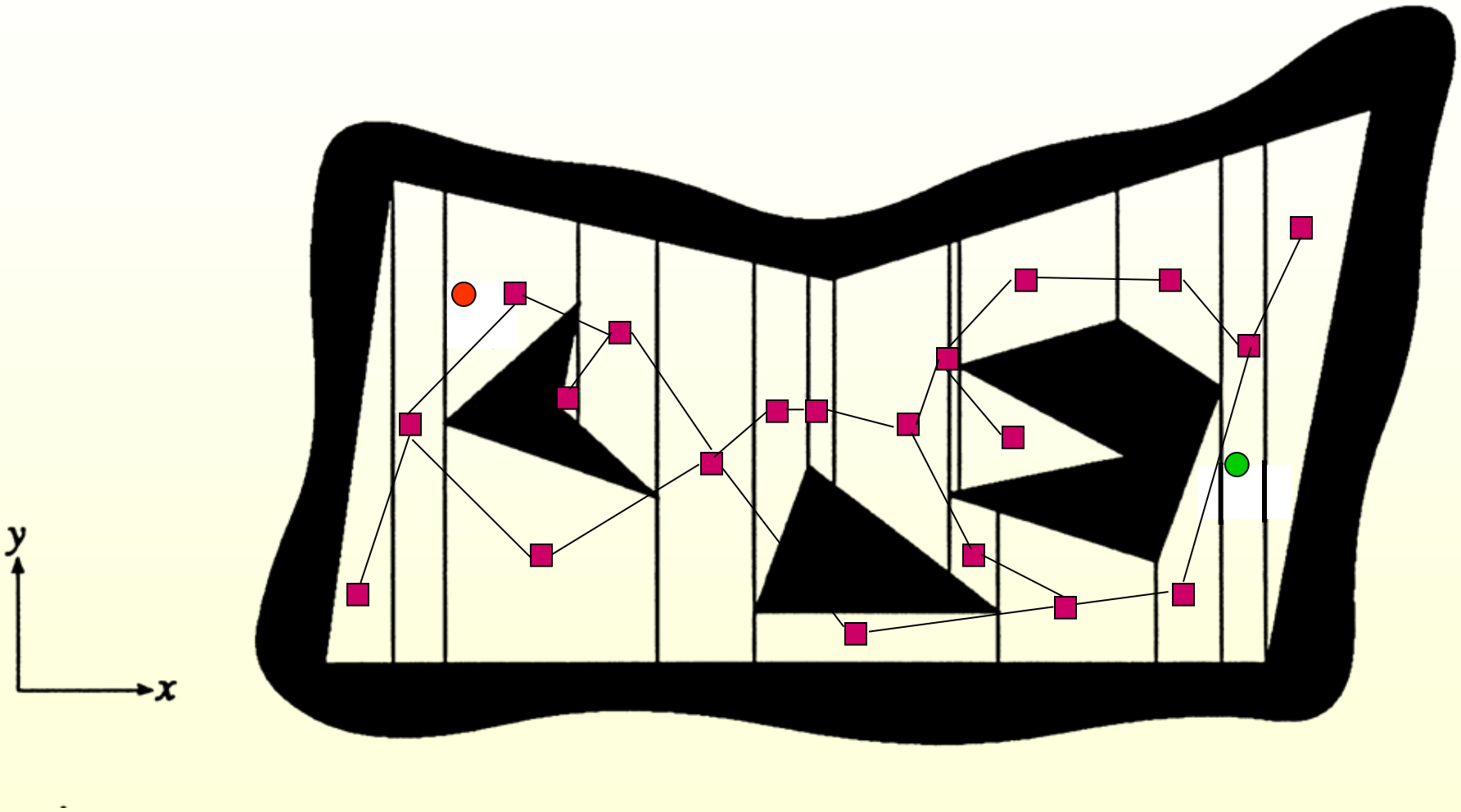
Trapezoidal Decomposition



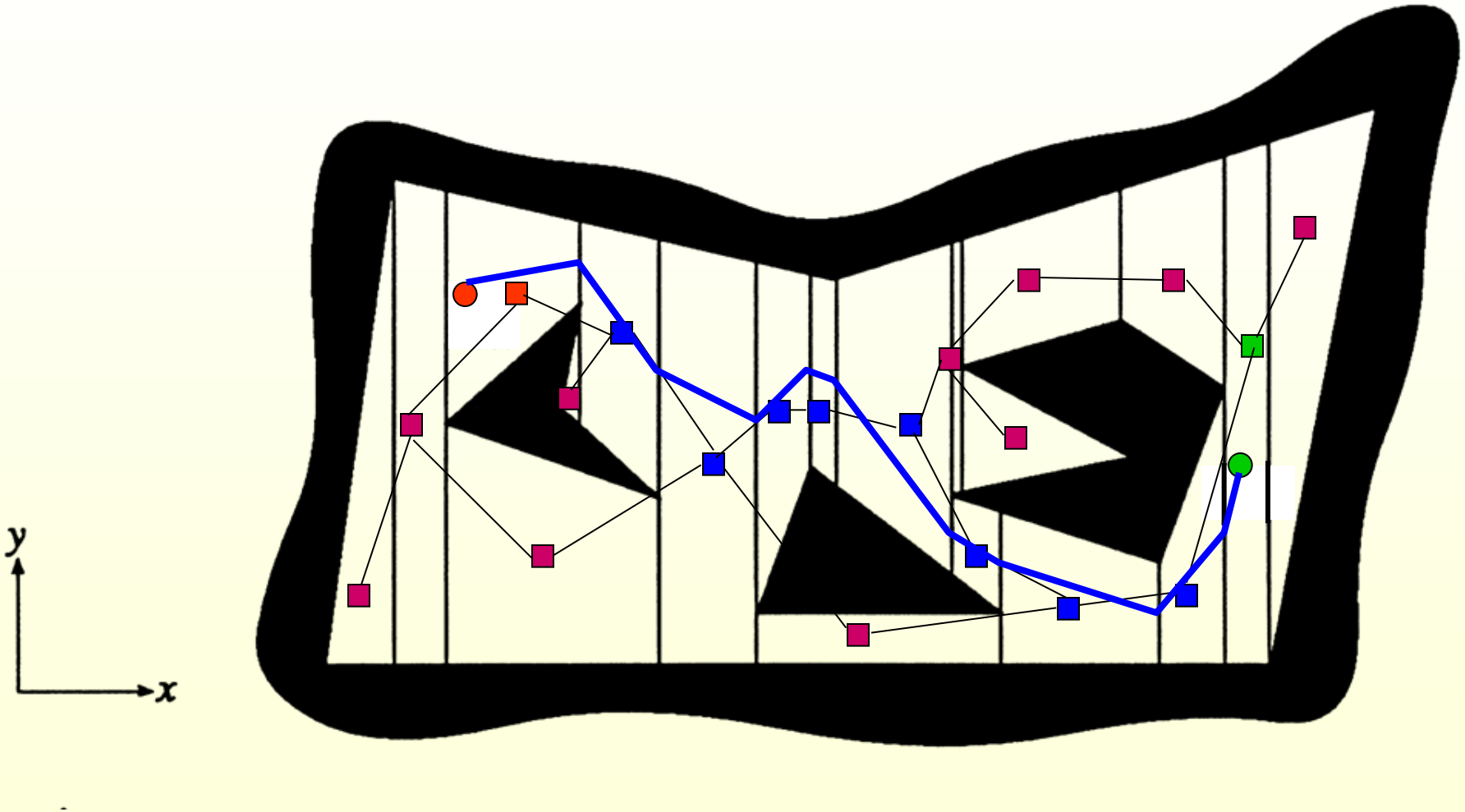
Trapezoidal Decomposition



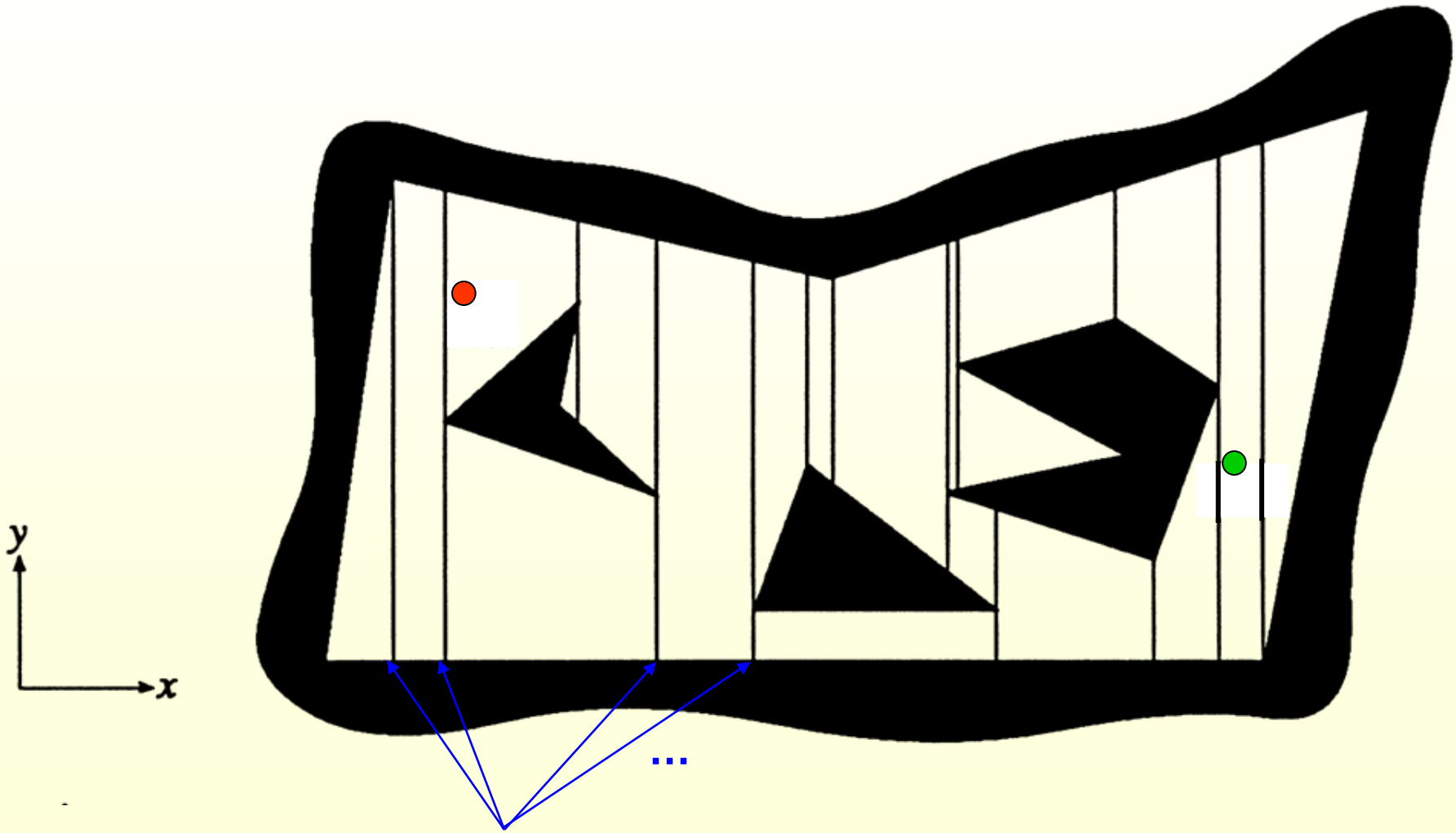
Trapezoidal Decomposition



Trapezoidal Decomposition

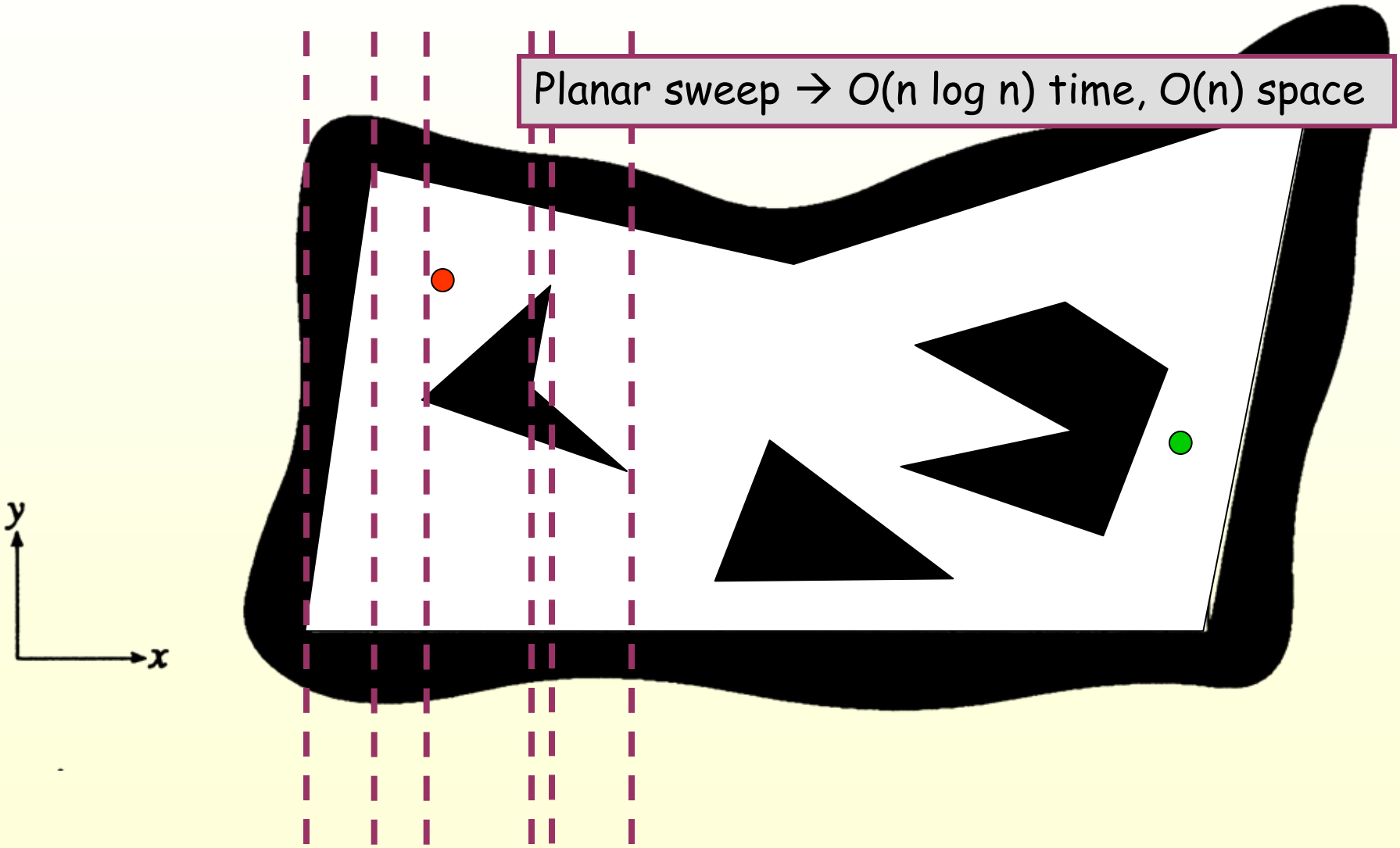


Trapezoidal Decomposition



critical events → criticality-based decomposition

Trapezoidal Decomposition

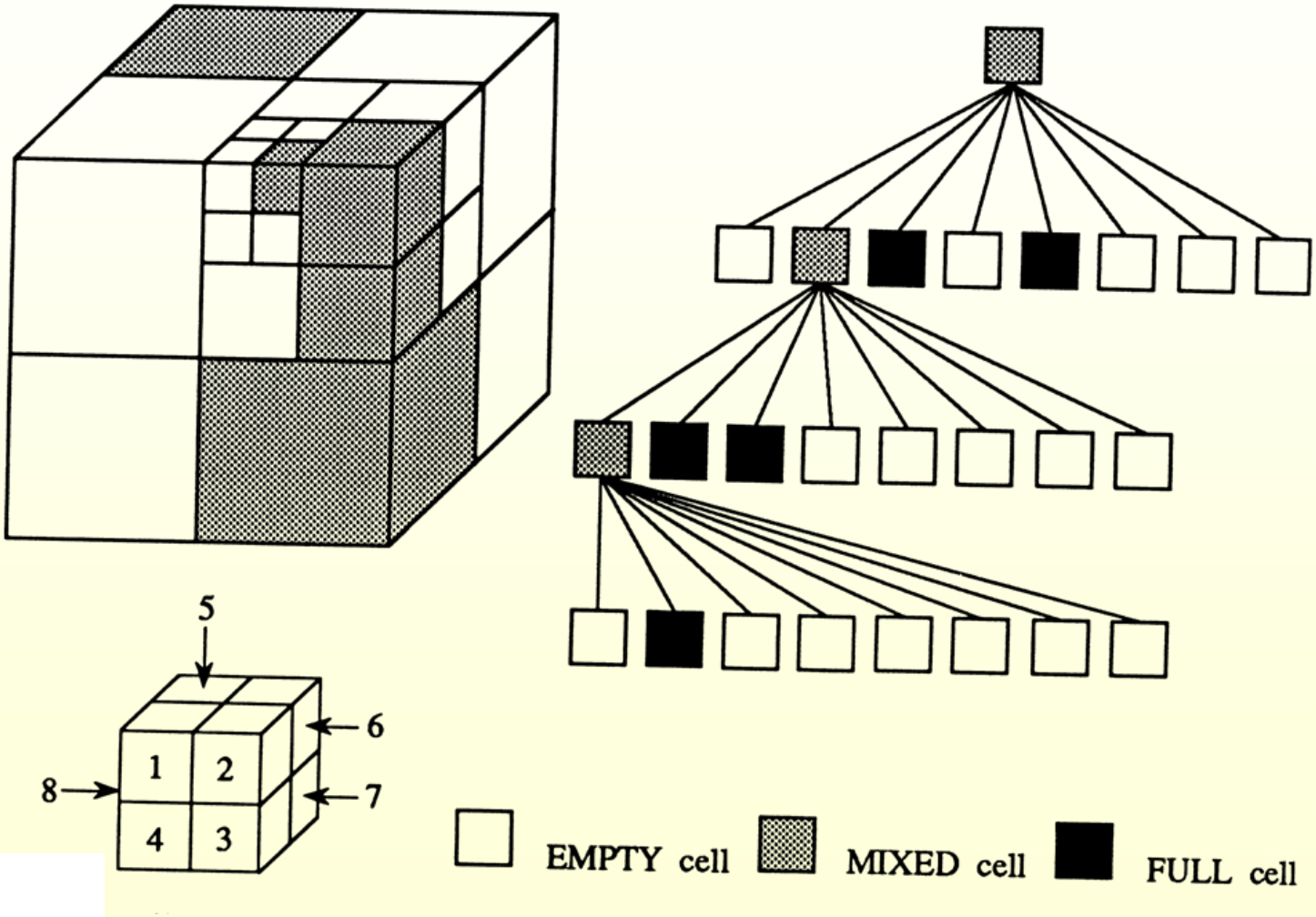


Cell-Decomposition Methods

Two classes of methods:

- Exact cell decomposition
- Approximate cell decomposition
  F is represented by a collection of non-overlapping cells whose union is contained in F
  Examples: quadtree, octree, 2^n-tree

Octree Decomposition



Sketch of Algorithm

1. Compute cell decomposition down to some resolution
2. Identify start and goal cells
3. Search for sequence of empty/mixed cells between start and goal cells
4. If no sequence, then exit with no path
5. If sequence of empty cells, then exit with solution
6. If resolution threshold achieved, then exit with failure
7. Decompose further the mixed cells
8. Return to 2

Path-Planning Approaches

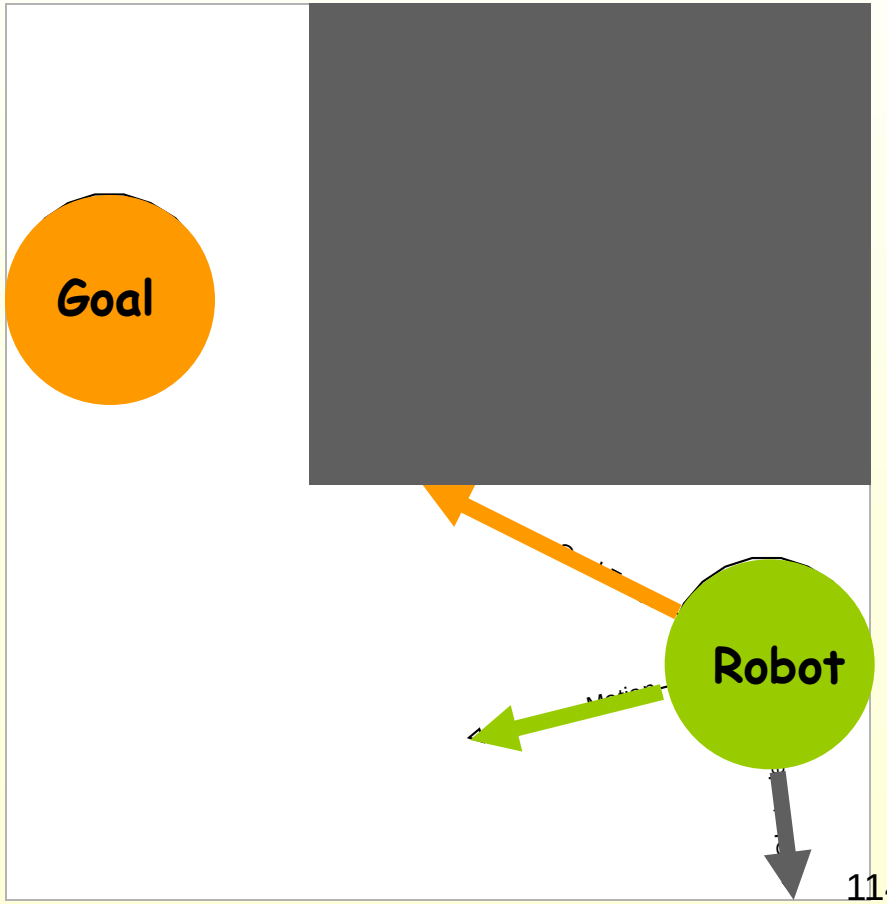
1. **Roadmap**
   Represent the connectivity of the free space by a network of 1-D curves
2. **Cell decomposition**
   Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells
3. **Potential field**
   Define a function over the free space that has a global minimum at the goal configuration and follow its steepest descent

Potential Field Methods

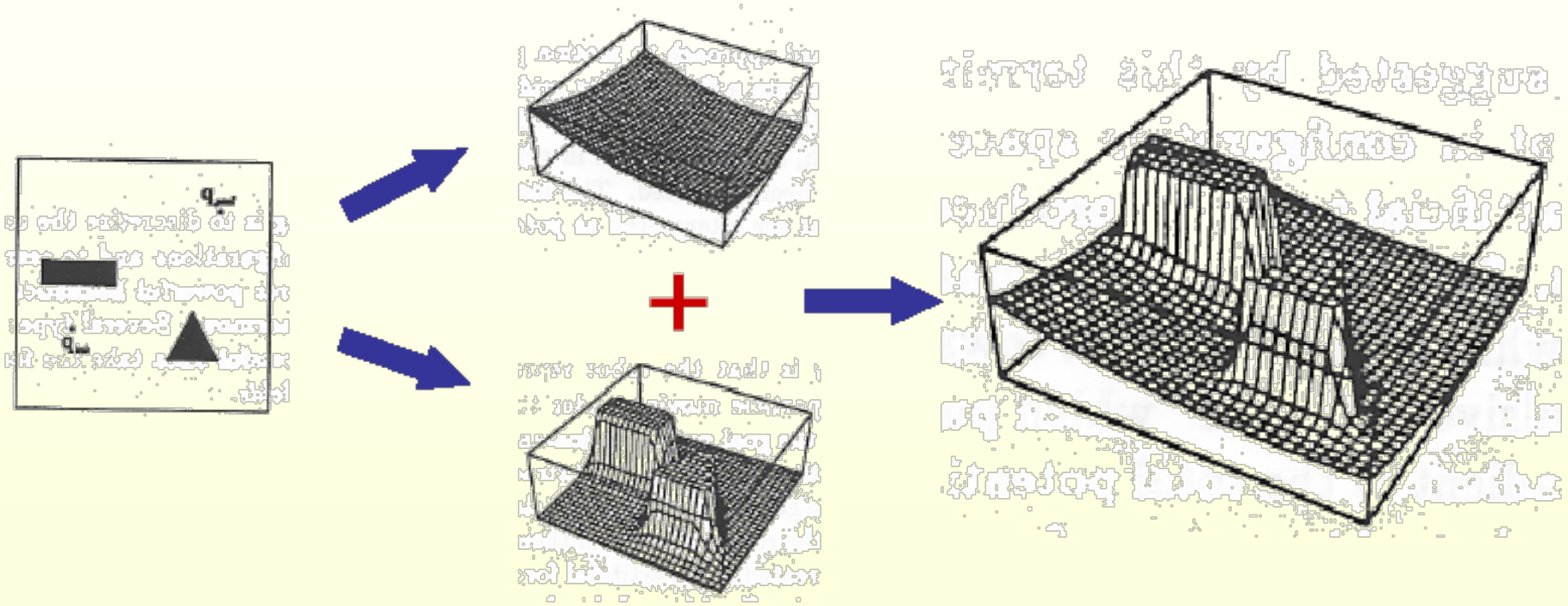
- Approach initially proposed for real-time collision avoidance [Khatib, 86]. Hundreds of papers published on it.

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

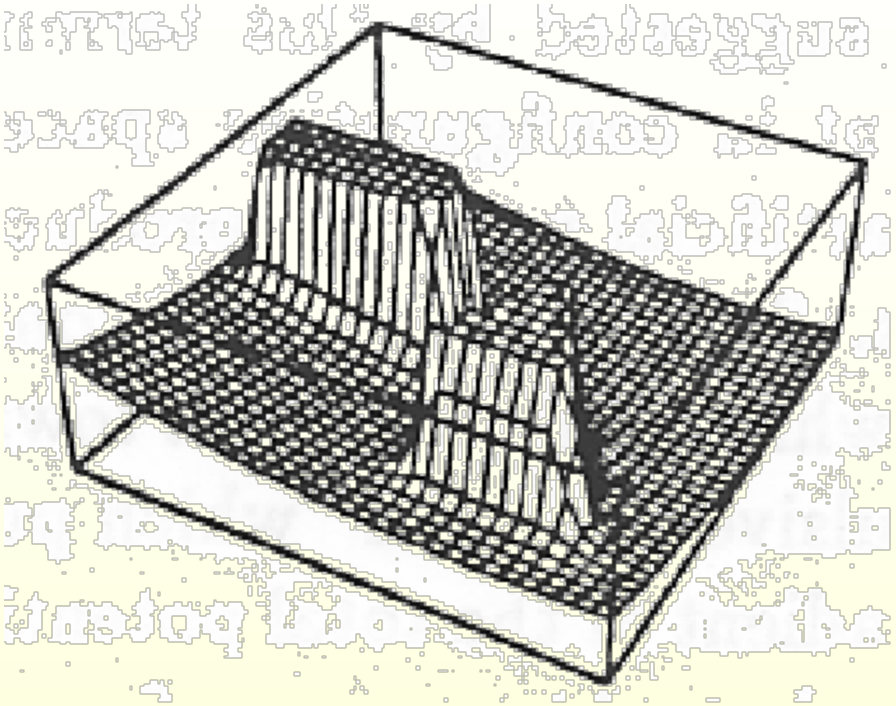
$$F_{Obstacle} = \begin{cases} \eta \left(\frac{1}{\rho} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2} \frac{\partial \rho}{\partial x} & \text{if } \rho \le \rho_0, \\ 0 & \text{if } \rho > \rho_0 \end{cases}$$



Attractive and Repulsive Fields



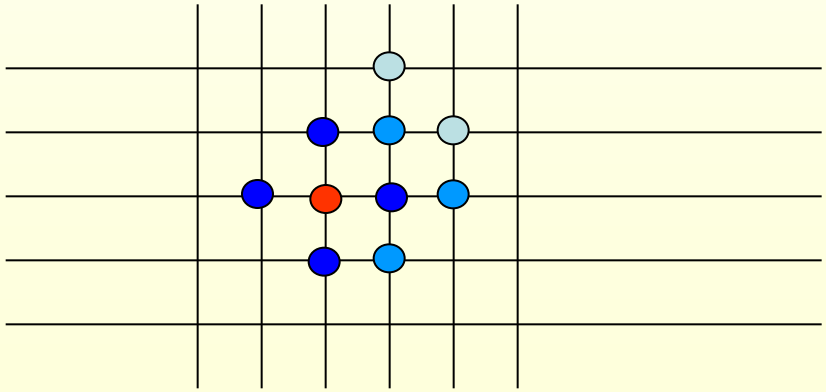
Local-Minimum Issue



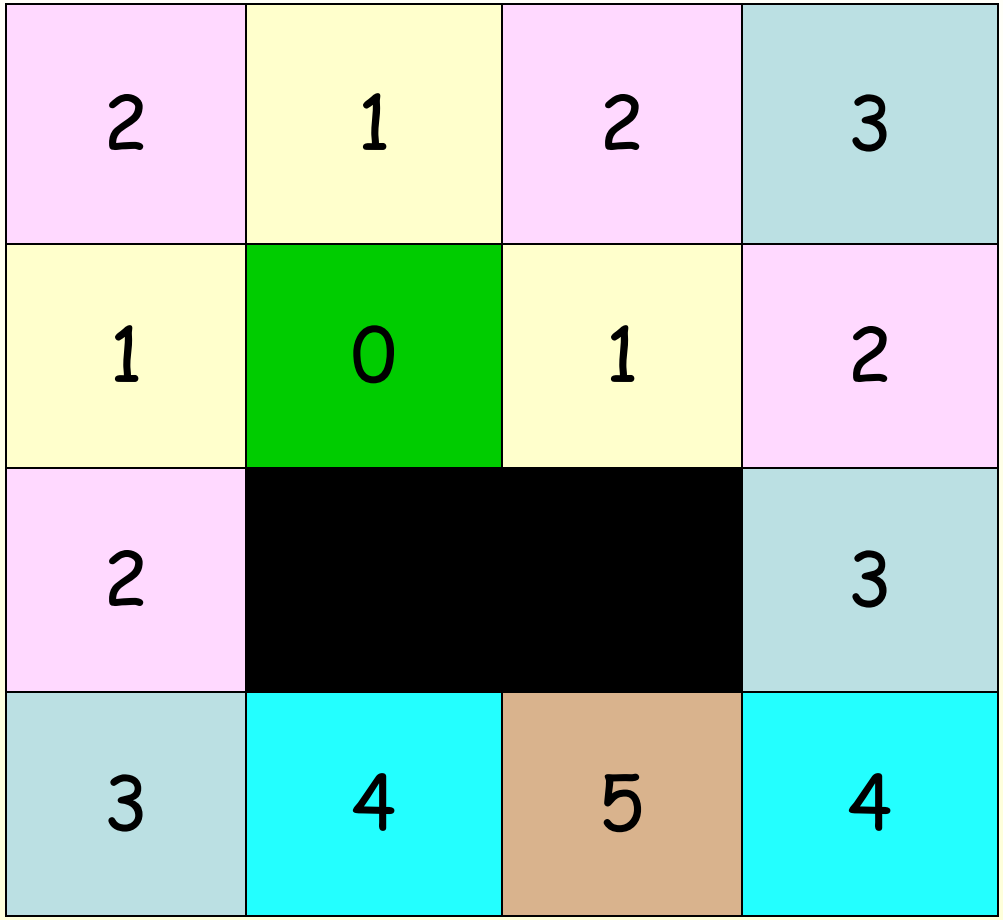
- Perform best-first search (possibility of combining with approximate cell decomposition)
- Alternate descents and random walks
- Use local-minimum-free potential (navigation function)

Sketch of Algorithm (with best-first search)

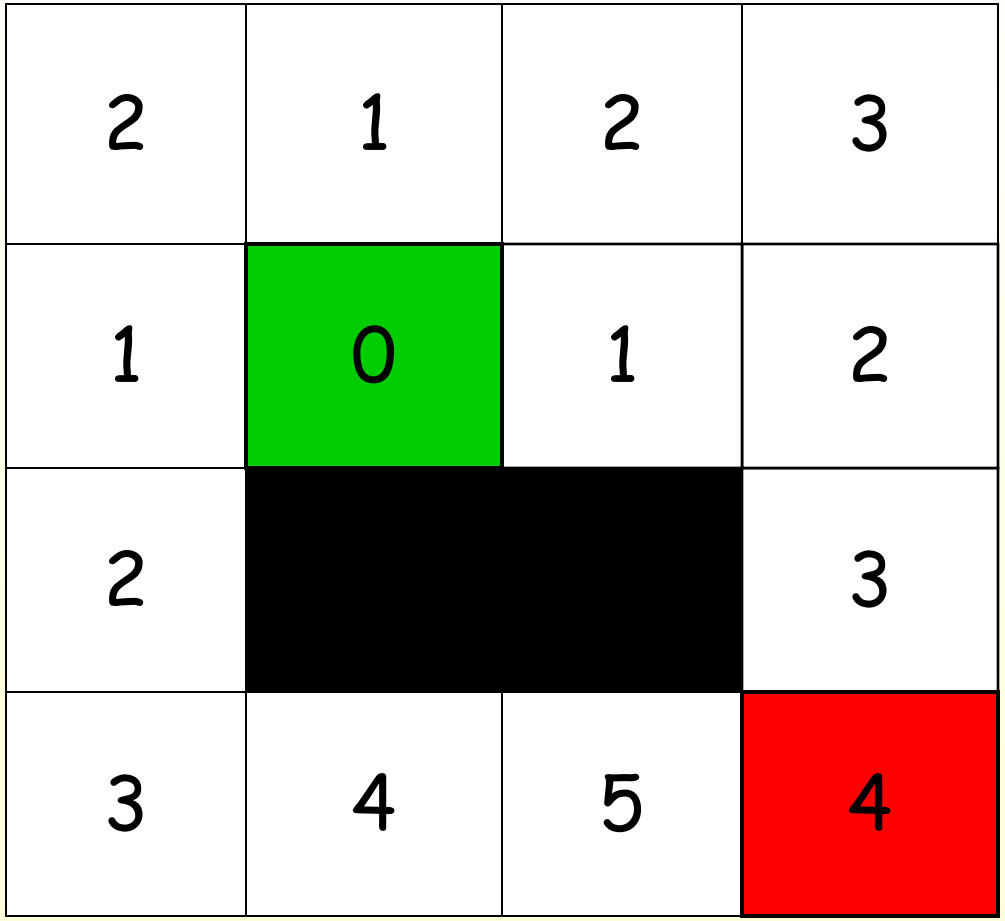
1. Place regular grid G over space
2. Search G using best-first search algorithm with potential as heuristic function



Simple Navigation Function



Simple Navigation Function



Completeness of Planner

- A motion planner is **complete** if it finds a collision-free path whenever one exists and returns failure otherwise.
- Visibility graph, Voronoi diagram, exact cell decomposition, navigation function provide complete planners
- Weaker notions of completeness, e.g.:
  - resolution completeness
    (path discovery with best-first search)
  - probabilistic completeness
    (path discovery with random walks)

- A probabilistically complete planner returns a path with high probability if a path exists. It may not terminate if no path exists.

- A resolution complete planner discretizes the space and returns a path whenever one exists in this representation.

Preprocessing / Query Processing

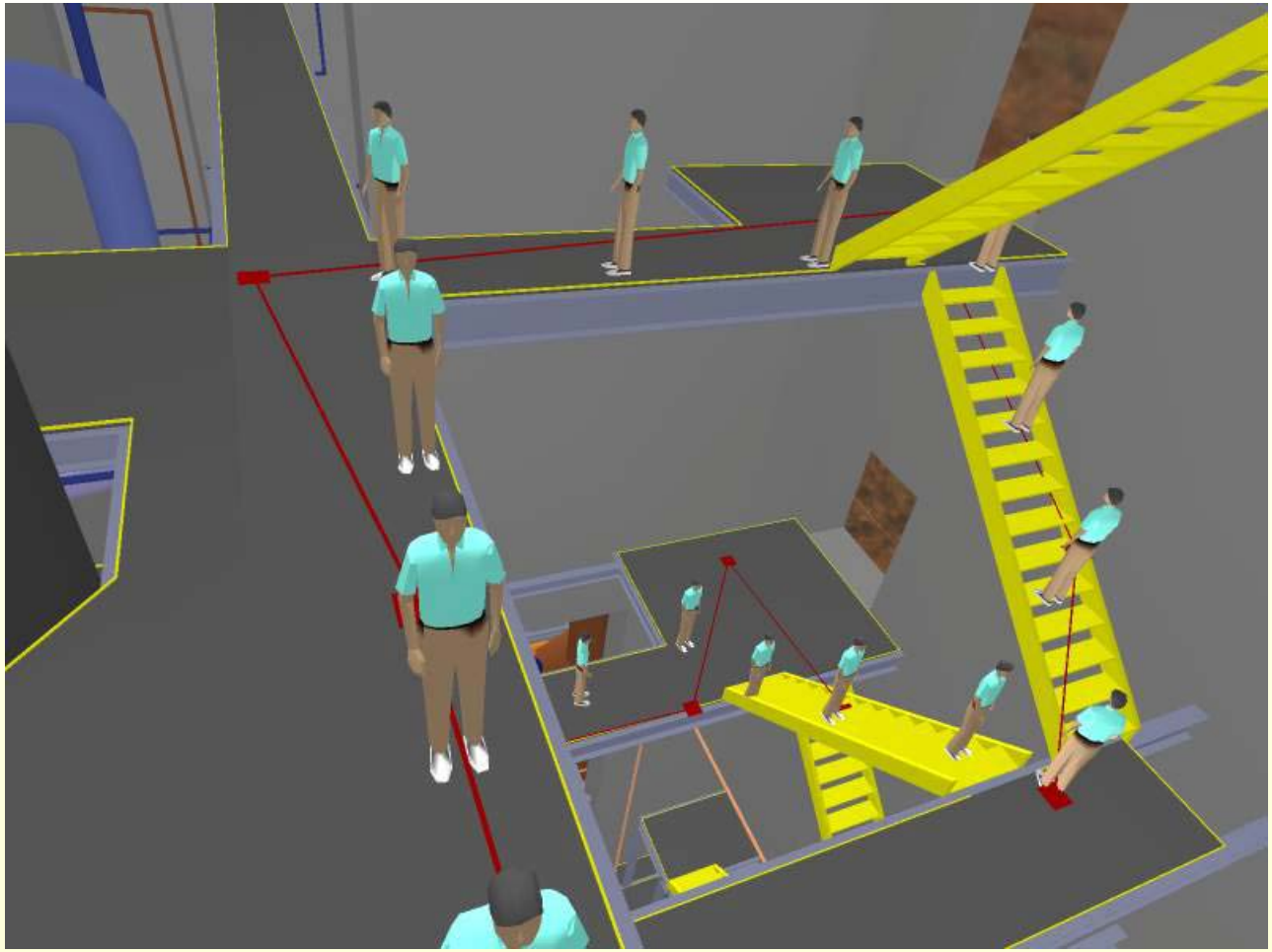
- Preprocessing:
  Compute visibility graph, Voronoi diagram, cell decomposition, navigation function

- Query processing:
  - Connect start/goal configurations to visibility graph, Voronoi diagram
  - Identify start/goal cell
  - Search graph

Many Remaining Issues

- Space dimensionality
- Geometric complexity of the free space
- Constraints other than avoiding collision
- The goal is not just a position to reach
- Etc …

The End



Course Review

Topics Covered, I

- 2D and 3D Transformations
- Smooth Shape Representations
  - 2D Parametric Polynomial Arcs, Bézier Control Points/Polygons
  - De Casteljau Subdivision Algorithm, Spline Curves
- Elementary Differential Geometry, Notions of Curvature
- Shape Registration: ICP (Iterated Closest Pair)
- Shape Family Analysis: PCA (Principal Components Analysis)
- Shape Matching: Descriptors and Correspondences
- Voronoi and Delaunay Diagrams
- Shape Distance Functions

Topics Covered, II

- Nearest Neighbor Search
- Surface Reconstruction
  - From Densities: Marching Cubes
  - From Point Clouds: CRUST and Related Algorithms
- Smoothing and Remeshing
- Topology, Meshes and Simplicial Complexes, Homology, Persistence
- Collision Detection
- Motion Planning

Related Courses

- CS205. Mathematical Methods for Robotics, Vision & Graphics
- CS268. Geometric Algorithms
- CS326a. Motion Planning
- CS348a. Computer Graphics: Geometric Modeling

The End