CS 4649/7649
Robot Intelligence: Planning

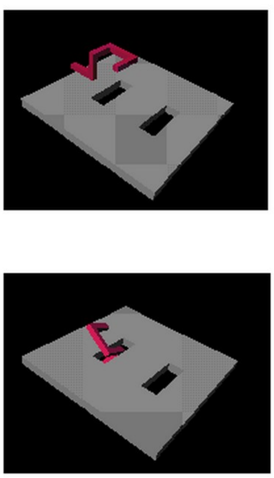
Probabilistic Roadmaps

Sungmoon Joo

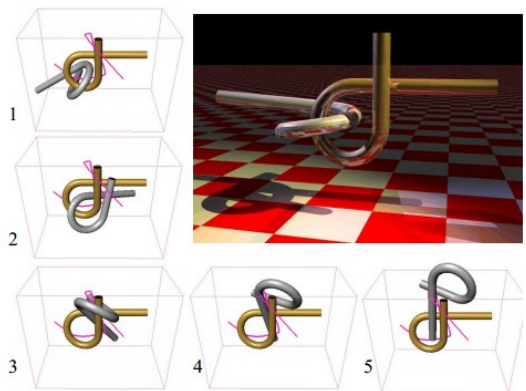
School of Interactive Computing
College of Computing
Georgia Institute of Technology

*Slides based in part on Dr. Mike Stilman and Dr. J.C. Latombe's lecture slides

Can we solve these planning problems?



http://www.kavrakilab.org/robotics/prm.html



"Planning Algorithms", S. Lavalle

Key Idea

- What did Visibility, Voronoi, Cells, Fields have in common?
  - Some form of explicit environment representation
  - Attempt at some form of optimality

- New concepts from 1990s:
  - Forget optimality altogether
  - Focus on Completeness
  - Think about Free Space

A New Kind of Roadmap

- Lydia Kavraki '94, '96 – Present
- Mark Overmars '92, '96 - Present

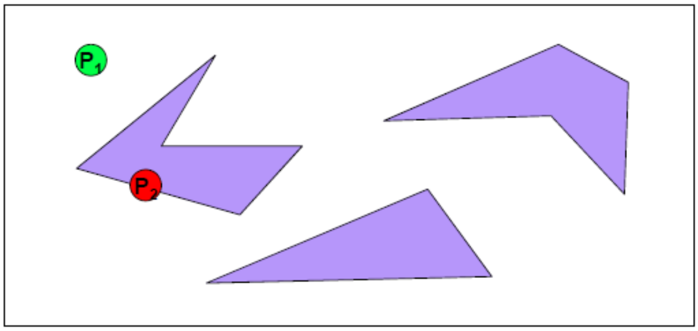
- Previous roadmaps used features related to actual obstacle features.

- Probabilistic Roadmaps (PRM)
  - Features: Sampled free points
  - Edges: Verified connections

"Probabilistic roadmaps for path planning in high-dimensional configuration spaces" By Kavraki, Svestka, Latombe, and Overmars, 1996, IEEE Transactions on Robotics and Automation

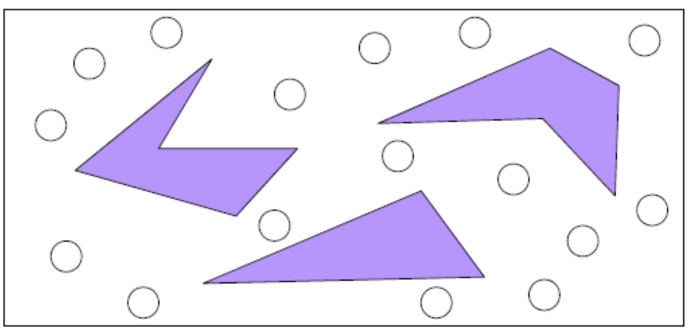
PRM idea: Step 1

Randomly sample a configuration P. Keep P only if P is in Free Space



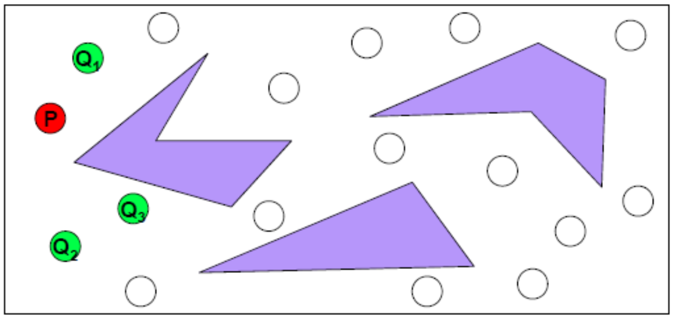
PRM idea: Step 1

Randomly sample a configuration P. Keep P only if P is in Free Space



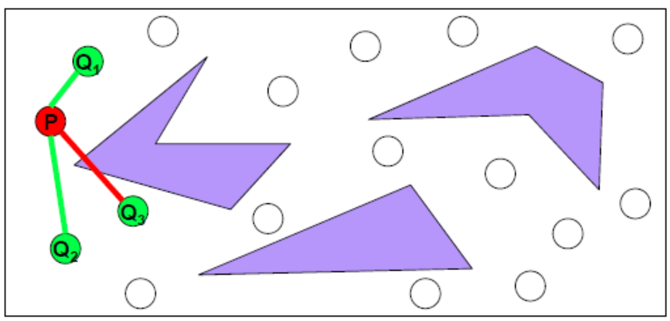
PRM idea: Step 2

For each node P find k nearest neighbors: $Q_1 \ldots Q_k$



PRM idea: Step 3

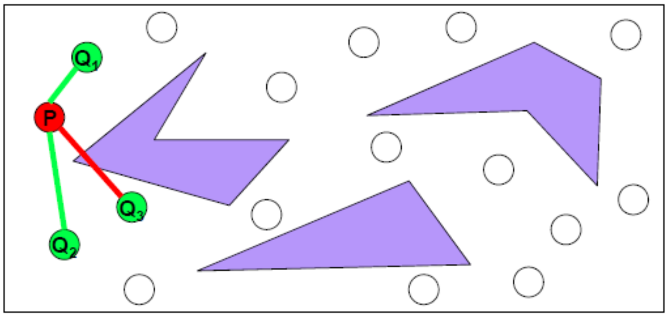
For each node P find k nearest neighbors: $Q_1 \ldots Q_k$



Use a 'local planner' to test connectivity between P and Q_i

Probabilistic Roadmap: Step 3

For each node P find k nearest neighbors: $Q_1 \ldots Q_k$

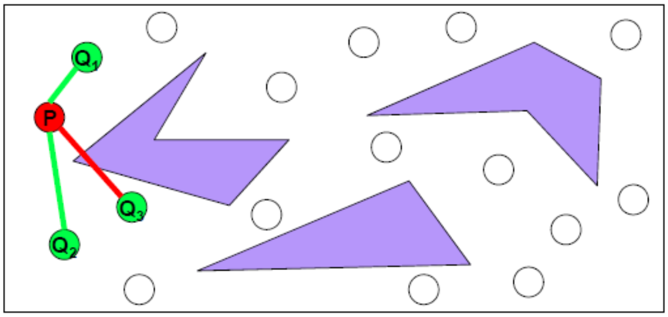


Use a local planner to test connectivity between P and Q_i

What could be a local planner?

PRM idea: Step 4

For each node P find k nearest neighbors: $Q_1 \ldots Q_k$



Use a local planner to test connectivity between P and Q_i

Find a path: Uniform Cost, A*, …

Probabilistic Roadmap

- Learning Phase: Construction and Expansion

  - Construct a PRM by generating random free configurations and connecting them using a simple, but very fast local planner

  - Store as a graph whose nodes are the configurations and whose edges are the paths computed by the local planner

  - Sometimes the graph consists of several large and small components which do not effectively capture the connectivity of free space. The graph even can be disconnected at some narrow region.

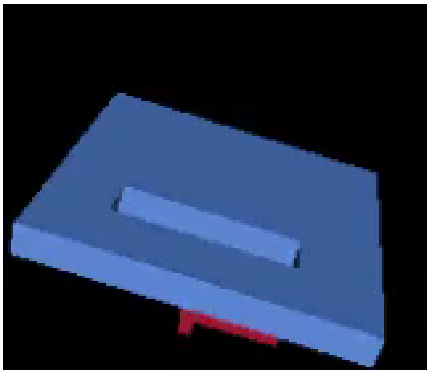
  - To expand a node, we compute a short, random-bounce walk starting from the node
- Query Phase

  - Find a path from the start and goal configurations to two nodes of the roadmap

  - Search the graph to find a sequence of edges connecting those nodes in the roadmap

  - Concatenating the successive segments gives a feasible path for the robot

We can solve these planning problem



http://www.kavrakilab.org/robotics/prm.html

http://www.youtube.com/watch?v=FRGzsyXHBqQ

Probabilistic Roadmap: Analysis

- **Sound**
  Yes

- **Complete**
  No
  Probabilistically Complete
  - The probability of success increases exponentially with the number of samples generated.
  - (RPP Barraquand & Latombe '89)

- **Efficient?**

Probabilistic Roadmap: Challenges

1. Connecting neighboring points
- Only easy for holonomic systems (e.g. linked manipulators)
(i.e., for which you can move each degree of freedom at will at any time).
- Typically solved w/o collision checking; later verified if valid by collision checking

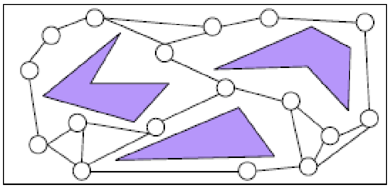
2. Collision checking
- Often takes majority of time in applications

3. Sampling
- How to sample uniformly (or biased according to prior information) over configuration space?

Making PRM Efficient

- Two procedures need to be extremely efficient:
  - Find Nearest Neighbor
    - → Identifies goals for local planner
  - Collision Detection
    - → Check if a sampled configuration is in free space
    - → Validate local plan

PRM



PRM: Pros and Cons

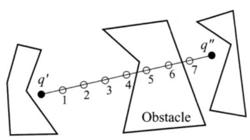
- Pros
  - Probabilistically complete: i.e., with probability one, if run long the graph will contain a solution path if one exists.
  - Apply easily to high-dimensional space
  - Fast with enough preprocessing

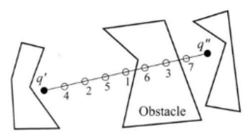
- Cons
  - Don't work well for some problems (e.g. narrow passage, constraints..)
  - Build graph over state space but no particular focus on generating a path
  - Post processing required: Shortening, Smoothing

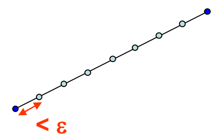
Planning Tools 1: Collision Detection

- Decide whether
  - a sample lies in free space, relatively easier
  - the local motion/path produced by the local planner is collision-free
- Local path collision checking (usually in configuration space)
  - Incremental: take small steps and check (early PRM)
  - Subdivision/Binary: use binary search

  (usually detect collision earlier than incremental methods)







- ε too large → collisions are missed
- ε too small → slow test of local paths

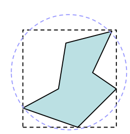
Planning Tools 1: Collision Detection

- Many different methods (usually in work space)

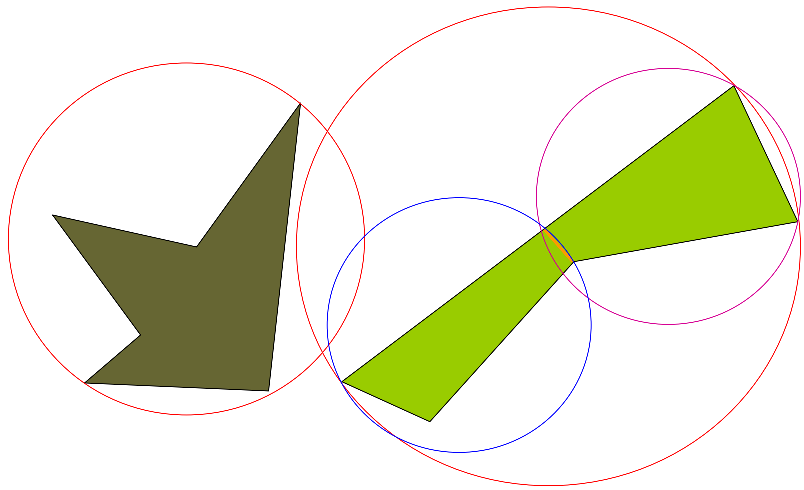
  - BVH, Grid method, Closest-feature tracking, Swept-volume intersection…

  * Bounding Volume Hierarchy(BVH) method in detail

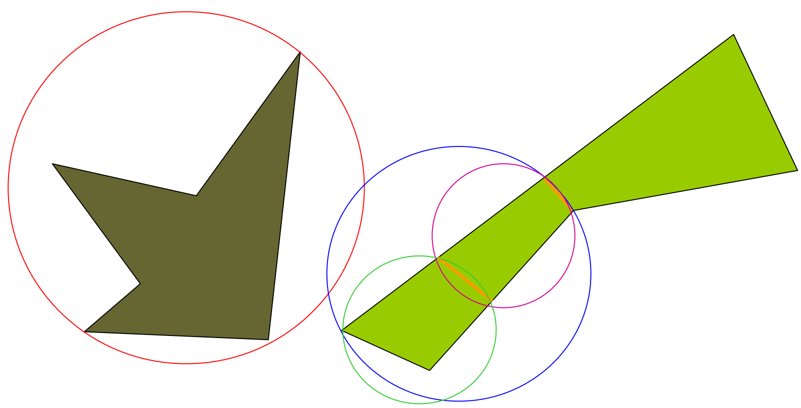
- BVH method: Idea
  - Enclose objects into bounding volumes (spheres or boxes)
  - Check the bounding volumes first
  - Decompose an object into two
  - Proceed hierarchically



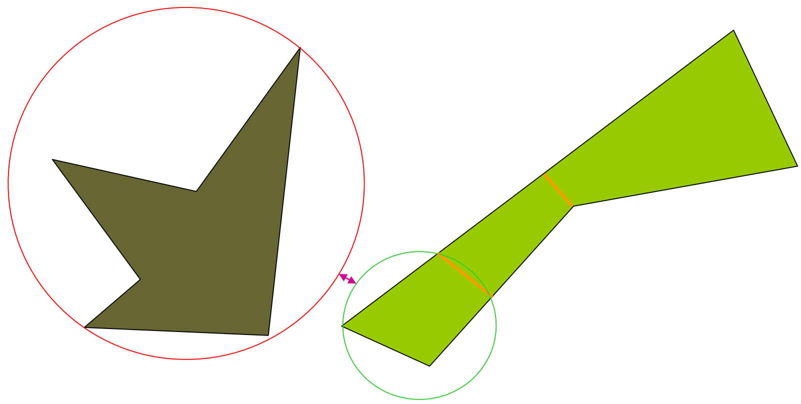
Collision Detection: BVH



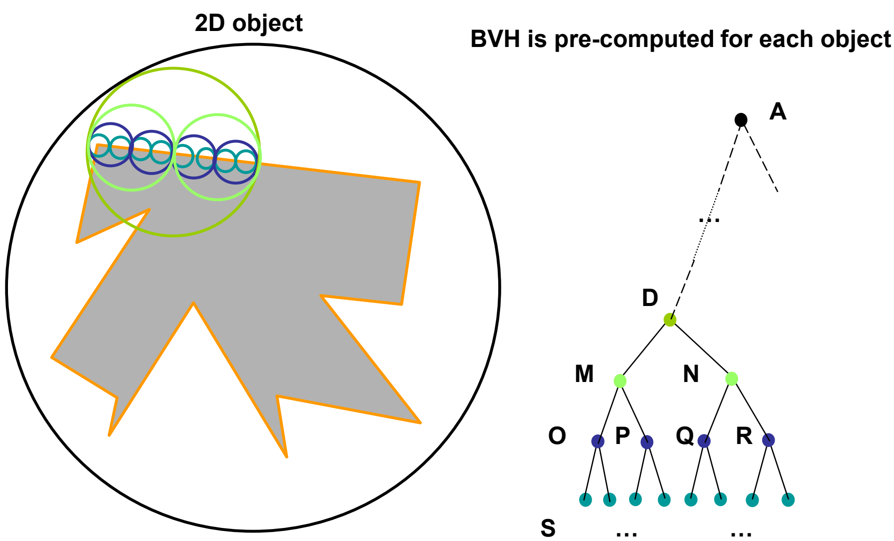
Collision Detection: BVH



Collision Detection: BVH



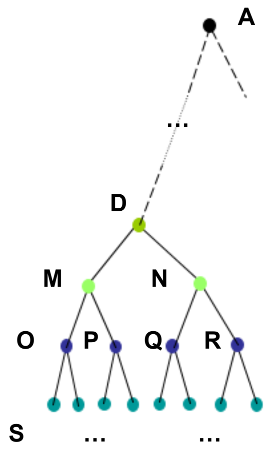
Collision Detection: BVH



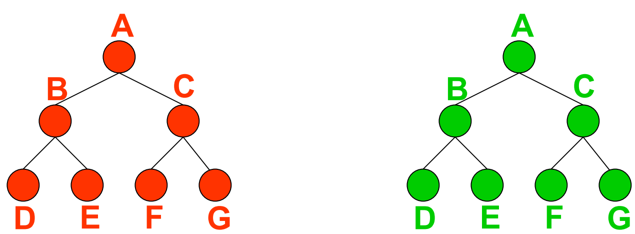
Collision Detection: BVH

3D object

BVH is pre-computed for each object



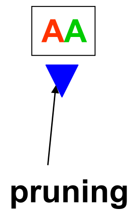
Collision Detection: BVH



Two objects described by their
pre-computed BVHs

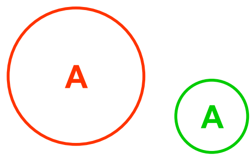
Collision Detection: BVH

Search tree



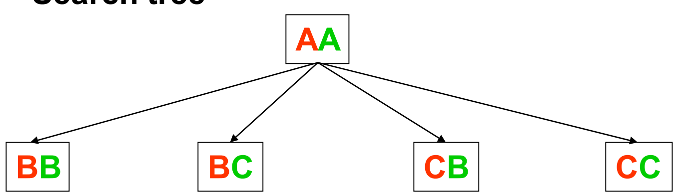
- Pruning discards subsets of the two objects that are separated by the BVs
- Each path is followed until pruning or until two leaves overlap
- When two leaves overlap, their contents are tested for overlap

No collision at top level → Don't need to test further



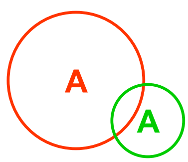
Collision Detection: BVH

Search tree

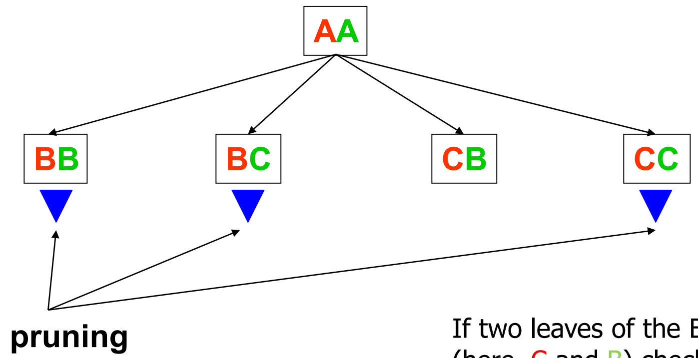


Collision at top level → Need to test sub levels

If two leaves of the BVH's overlap (here, A and A) check their content for collision



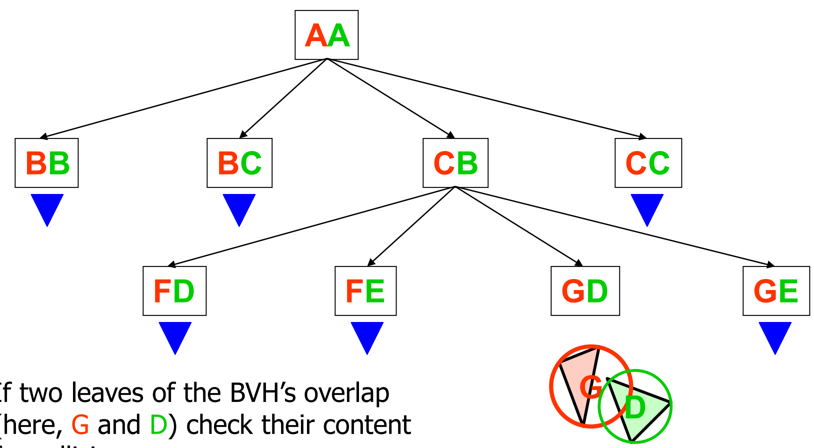
Collision Detection: BVH



If two leaves of the BVH's overlap (here, C and B) check their content for collision

Collision Detection: BVH

Search tree



If two leaves of the BVH's overlap (here, G and D) check their content for collision

Collision Detection: BVH

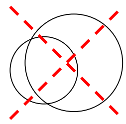
- Search strategy needed
  - If no collision, all paths must eventually be followed down to pruning or a leaf node
  - If collision, it is desirable to detect it as quickly as possible

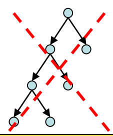
- Performance
  - O(several thousand) collision checks per second for 2 three-dimensional objects each described by 500,000 triangles, on a 1-GHz PC
  - Faster when objects are well separated or have much overlap.
  - Slower when objects barely overlap or are very close.

- Desirable Properties of BVs and BVHs

BV
- Tightness
- Efficient testing
- Invariance

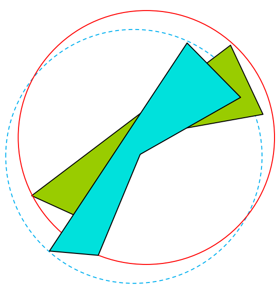
BVH
- Separation
- Tree balance

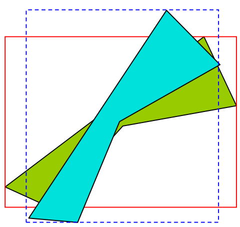


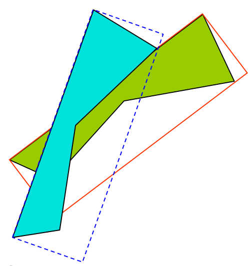


Collision Detection: BVH

No type of BV is optimal for all situations

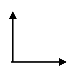






Sphere **Axis-Aligned Bounding Box** **Oriented Bounding Box**

	Sphere	AABB	OBB
Tightness	-	--	+
Testing efficiency	+	+	0
Invariance	yes	no	yes



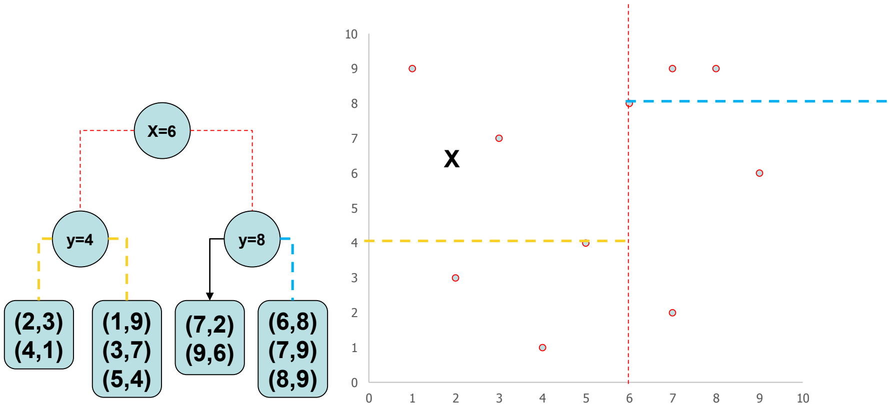
Planning Tools 2: Nearest Neighbor

- k-d tree: common choice for graph building
  - If there is just one point, form a leaf with that point.
  - Otherwise, divide the points in (roughly) half by a line perpendicular to one of the axes.
  - Recursively construct k-d trees for the two sets of points.
  - Requires O(dn) storage, built in O(dn log n) time
  - Query takes $O(n^{1-1/d} + m)$ time where m is # of neighbors
    → asymptotically linear in n and m with large d

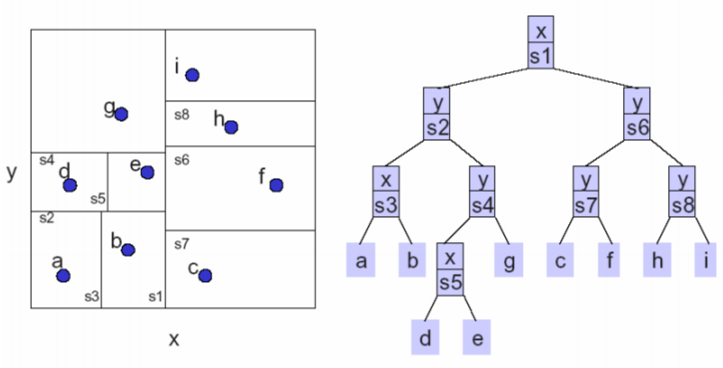
- k-d tree search
  - Search over a circular region → update(decrease) the radius

k-d Tree Building

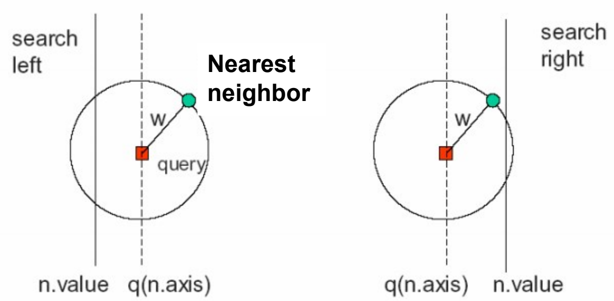
Points {(2,3),(4,1),(1,9),(3,7),(5,4),(7,2),(9,6),(6,8),(7,9),(8,9)}



k-d Tree Building



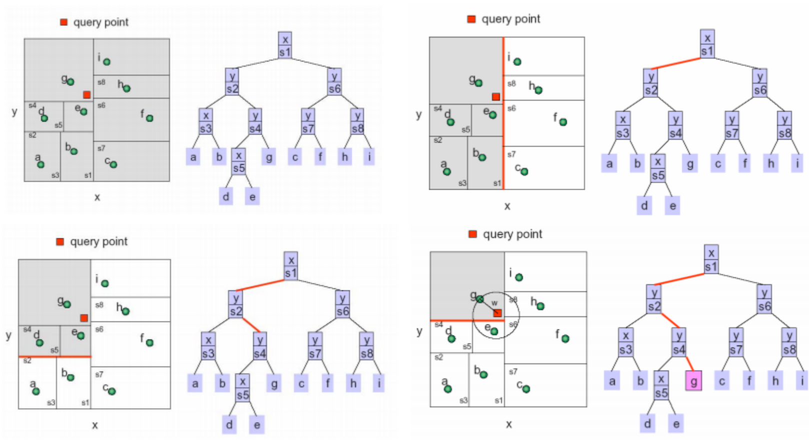
k-d Tree Search



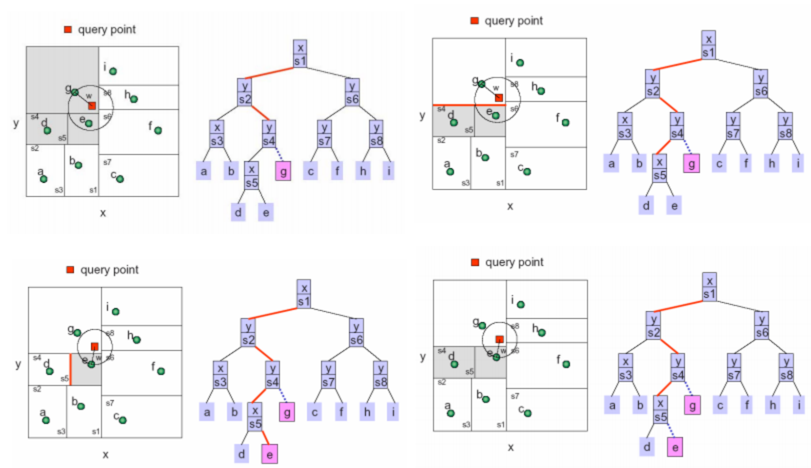
q(n.axis) – w ≤ n.value
means the circle overlaps
the left subtree.

q(n.axis) + w > n.value
means the circle overlaps
the right subtree.

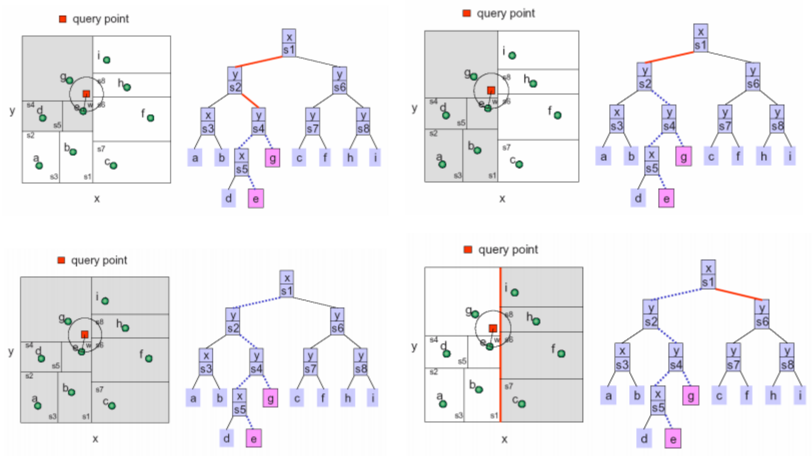
k-d Tree Search



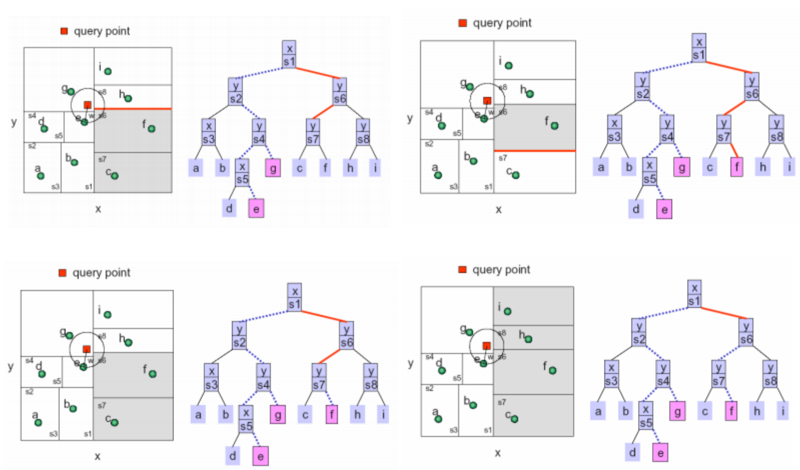
k-d Tree Search



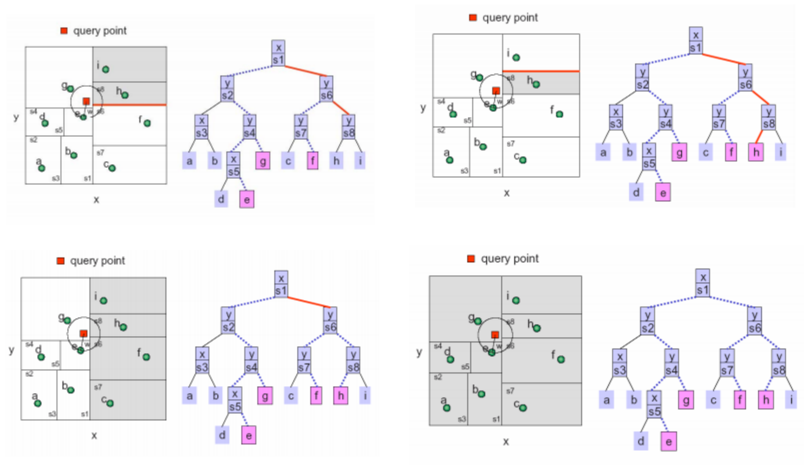
k-d Tree Search



k-d Tree Search



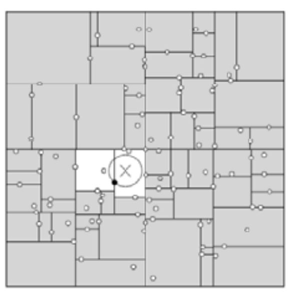
k-d Tree Search



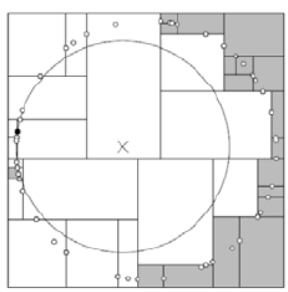
Planning Tools 2: Nearest Neighbor

- K-D tree search **(white leaf nodes searched)**

Sampling is important!

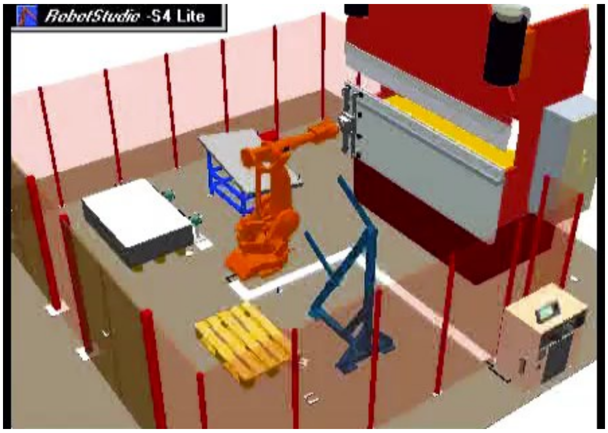


Good distribution



Bad distribution

PRM in Action



http://www.kavrakilab.org/robotics/lazyprm.html

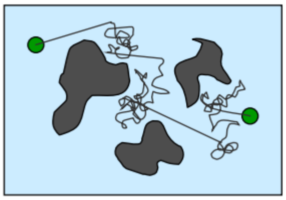
PRM Summary

- Concept
  - samples to find free configurations
  - connects the configurations (creates a graph)
  - search
- Does not require explicit calculation of obstacle features
  - does require efficient Collision Detection
  - does require efficient Nearest Neighbor
- Create a roadmap once, queries are very fast - **Multi-Query Planner**

Single vs Multi-Query

Single

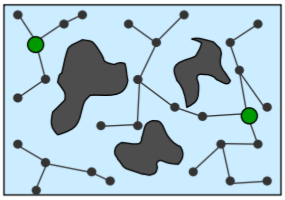
EXAMPLE: Potential-Field



Greedy, can take a long time but good when you can dive into the solution

Multi

EXAMPLE: PRM



Spreads out like uniformity but need lots of sample to cover space

Single Query Alternatives

- **Randomized Potential Field Planner (Barraquand & Latombe '89)**
  - Combines: potential fields w/
  - Random motions to escape local minima

- **Ariadne's Clew (Bessiere, Mazer, Ahuactzin '95)**
  - Places new configurations far apart from old ones
  - Interleaves attempts to directly reach the goal

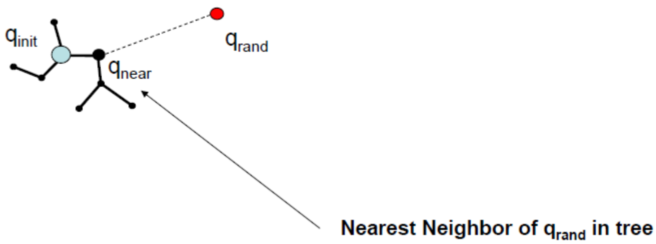
- **Rapidly Exploring Random Trees (LaValle '98, Kuffner & LaValle '99)**
  - Exploration is biased to achieve fast coverage of space

- **More Options:**
  - Expansive Space Trees (Hsu et. al. '00)
  - LazyPRM (Bohlin & Kavraki '01)

* Probabilistic Roadmap of Tree (PRT) combines both (single & multi-query) ideas

Rapidly-Exploring Random Trees (RRT)

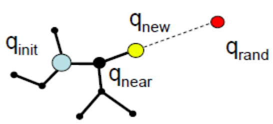
- Planning is search
- Search happens over a search tree
- RRT defines a simple rule for growing high quality trees



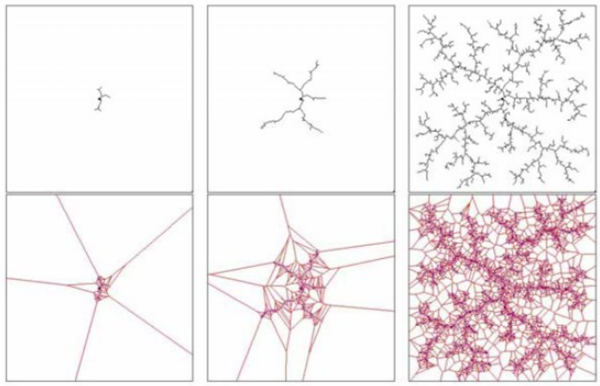
[LaValle '98, LaValle & Kuffner '00]

RRT: Sampling Paths

- Planning is search
- Search happens over a search tree
- RRT defines a simple rule for growing high quality trees



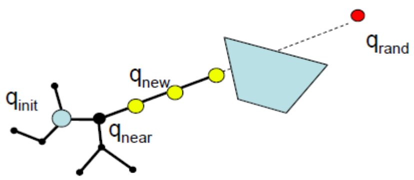
RRT: Voronoi Bias (Evaluating Trees)



The probability that a path is found increases exponentially with the number of iterations.

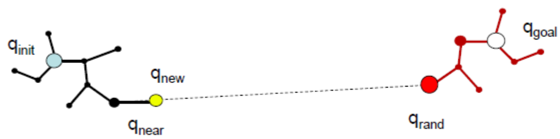
[Kuffner & LaValle '00]

RRT Connect



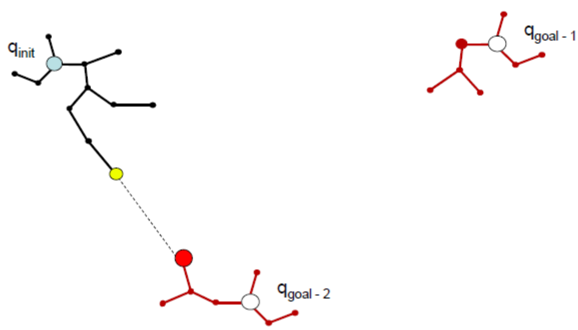
[LaValle '98, LaValle & Kuffner '01]

Merging Trees



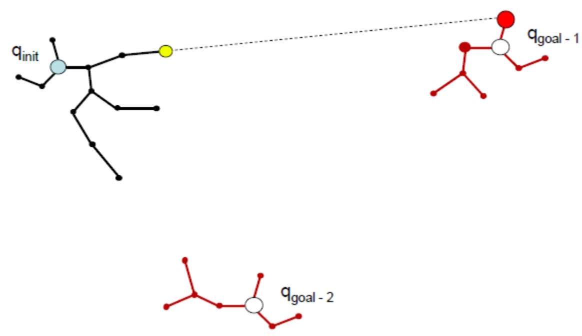
[Kuffner & LaValle '99]

Multi-Tree RRT Connect



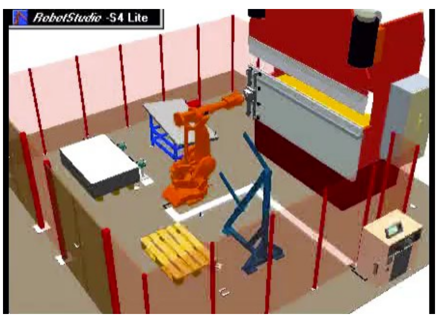
[Hirano et. al. '05]

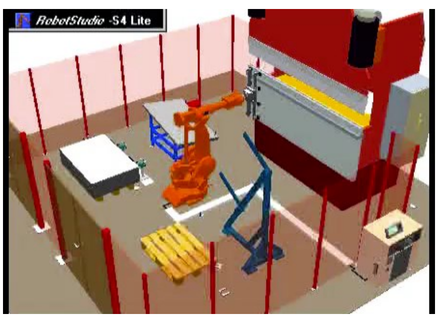
Multi-Tree RRT Connect



[Hirano et. al. '05]

Challenge 1: Refinement

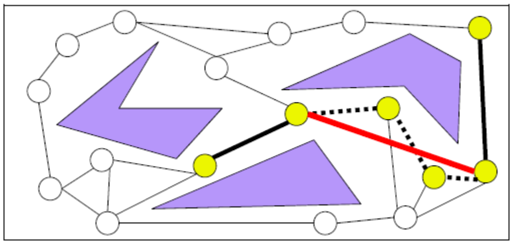




http://www.cse.unr.edu/robotics/tc-apc/videos

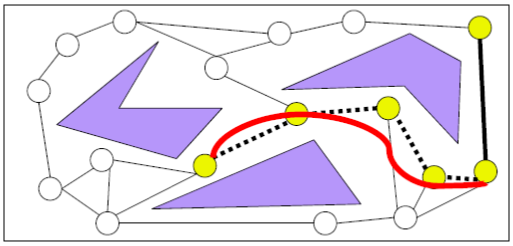
Back to Optimality

- This is an optimal path given the roadmap / sampled tree
  - **Path Shortening!**
  - Extra care with collision detection



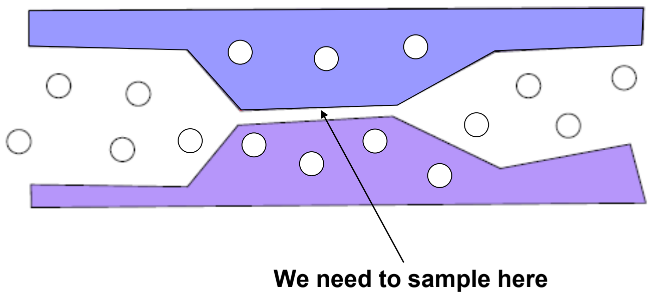
Back to Optimality

- This is an optimal path given the roadmap / sampled tree
  - **Path Smoothing!**
  - Extra care with collision detection



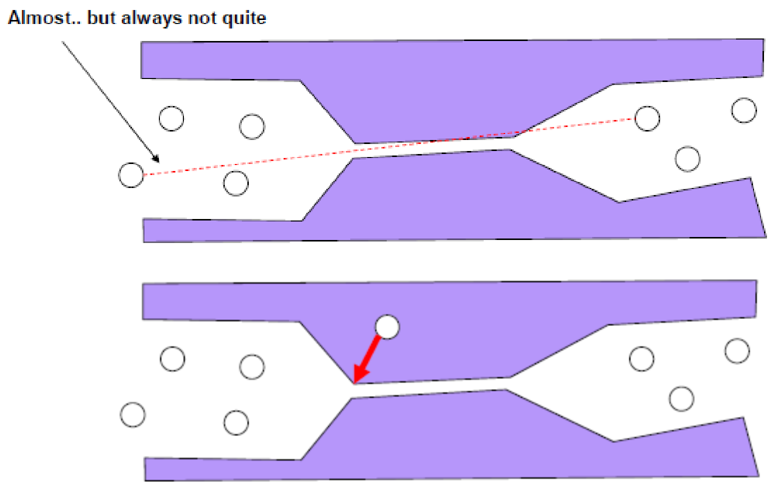
Challenge 2: Sampling

'Uniform sampling" is good because it is easy to implement but could be bad…



Different strategies: Near obstacles, Narrow passages, Visibility-based, Manipulability-based, Quasi-random, Grid-based…

Narrow Passages



Use workspace information to guide joint space motion – usually also randomized.