16-350
Planning Techniques for Robotics

Planning Representations/Search Algorithms: Rapidly Exploring Random Trees (RRT)

Maxim Likhachev

Robotics Institute

Carnegie Mellon University

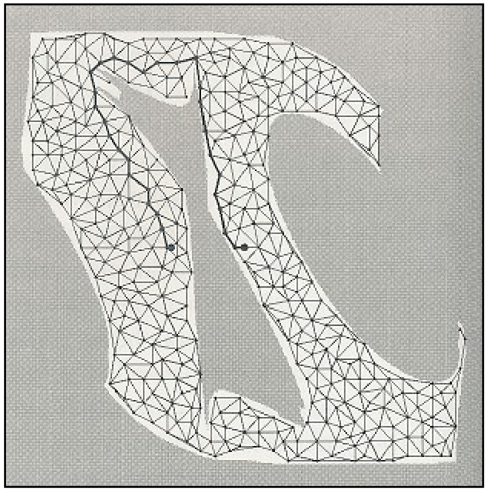
Probabilistic Roadmaps (PRMs)

Great for problems where a planner has to plan many times for different start/goal pairs (step 1 needs to be done only once)

Not so great for single shot planning

Step 1. Preprocessing Phase: Build a roadmap (graph) $\mathcal{G}$ which, hopefully, should be accessible from any point in C_{free}

Step 2. Query Phase: Given a start configuration q_I and goal configuration q_G, connect them to the roadmap $\mathcal{G}$ using a local planner, and then search the augmented roadmap for a shortest path from q_I to q_G



Rapidly Exploring Random Trees (RRTs)

No preprocessing step: starting with the initial configuration q_I build the graph (actually, tree) until the goal configuration g_G is part of it

Very effective for single shot planning

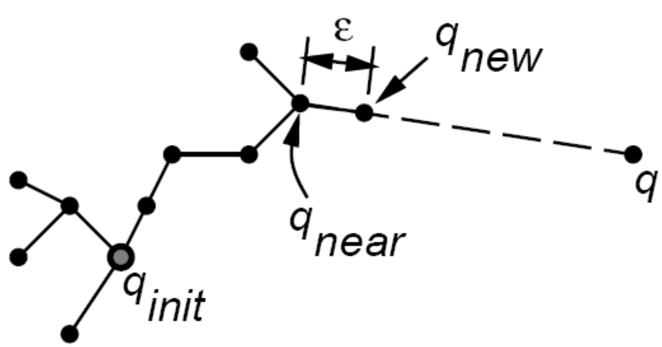
Rapidly Exploring Random Trees (RRTs)

BUILD_RRT(q_{init})

$\mathcal{T}$.init(q_{init});
for $k = 1$ **to** K **do**
    $q_{rand} \leftarrow$ RANDOM_CONFIG();
    EXTEND($\mathcal{T}, q_{rand}$);
Return $\mathcal{T}$

EXTEND($\mathcal{T}, q$)

$q_{near} \leftarrow$ NEAREST_NEIGHBOR($q, \mathcal{T}$);
if NEW_CONFIG(q, q_{near}, q_{new}) **then**
    $\mathcal{T}$.add_vertex(q_{new});
    $\mathcal{T}$.add_edge(q_{near}, q_{new});
    if $q_{new} = q$ **then**
        Return *Reached*;
    else
        Return *Advanced*;
Return *Trapped*;



EXTEND *operation*

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Rapidly Exploring Random Trees (RRTs)

Path to the goal is a path in the tree from q_{init} to the vertex closest to goal

BUILD_RRT(q_{init})

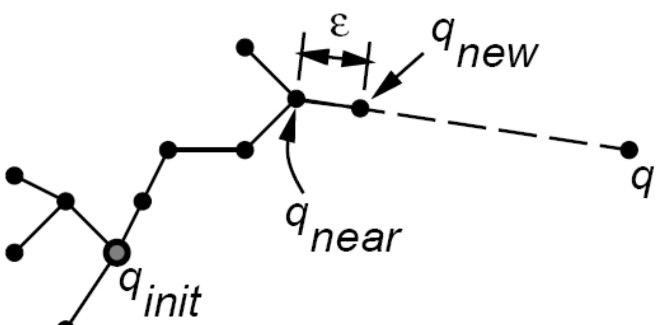
$\mathcal{T}$.init(q_{init});
for $k = 1$ **to** K **do**
    $q_{rand} \leftarrow$ RANDOM_CONFIG();
    EXTEND($\mathcal{T}, q_{rand}$);
Return $\mathcal{T}$

selects closest vertex in the tree

moves by at most ε from q_{near} towards q

EXTEND($\mathcal{T}, q$)

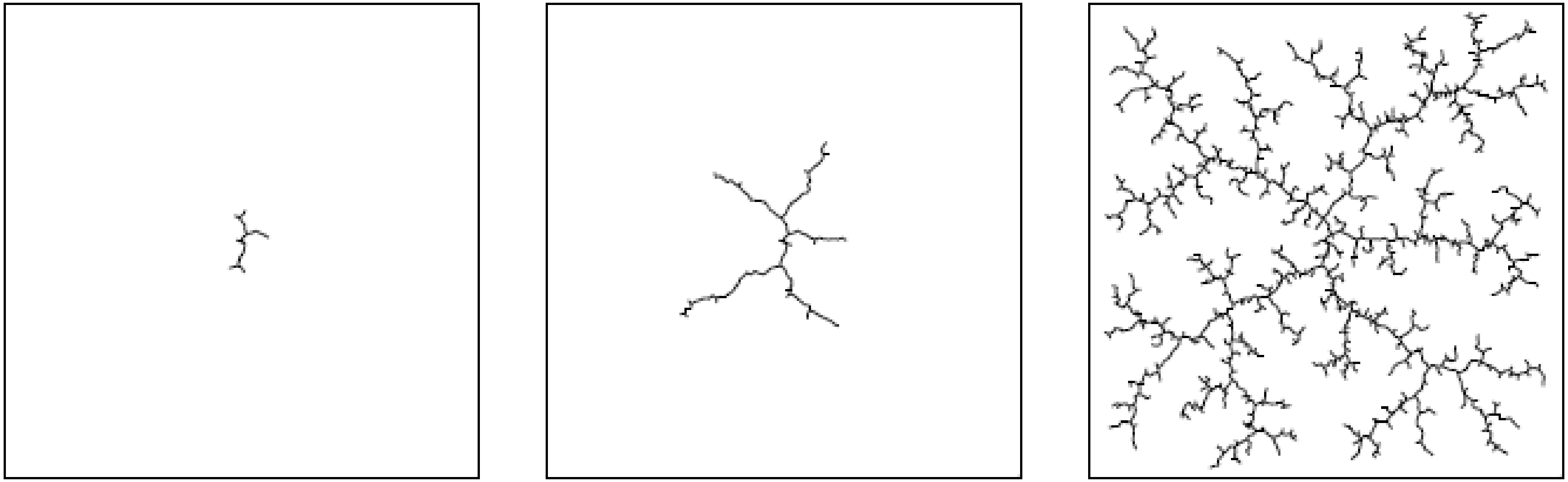
$q_{near} \leftarrow$ NEAREST_NEIGHBOR($q, \mathcal{T}$);
if NEW_CONFIG(q, q_{near}, q_{new}) **then**
    $\mathcal{T}$.add_vertex(q_{new});
    $\mathcal{T}$.add_edge(q_{near}, q_{new});
    if $q_{new} = q$ **then**
        Return *Reached*;
    else
        Return *Advanced*;
Return *Trapped*;



EXTEND *operation*

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

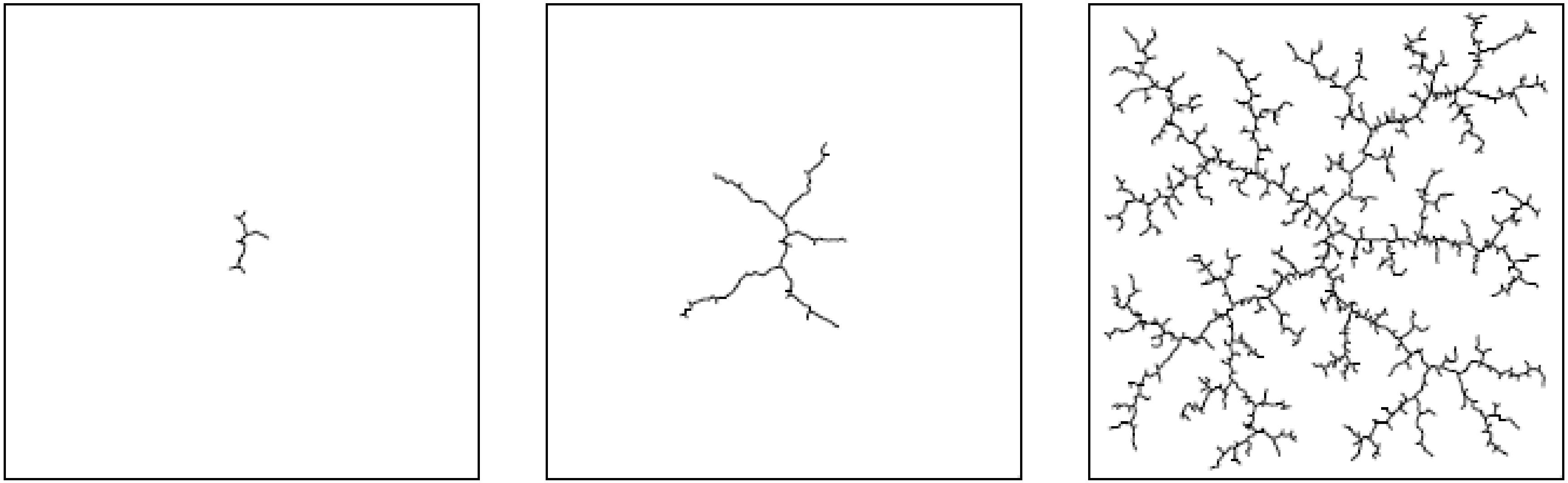
Rapidly Exploring Random Trees (RRTs)



- RRT provides uniform coverage of space

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Rapidly Exploring Random Trees (RRTs)

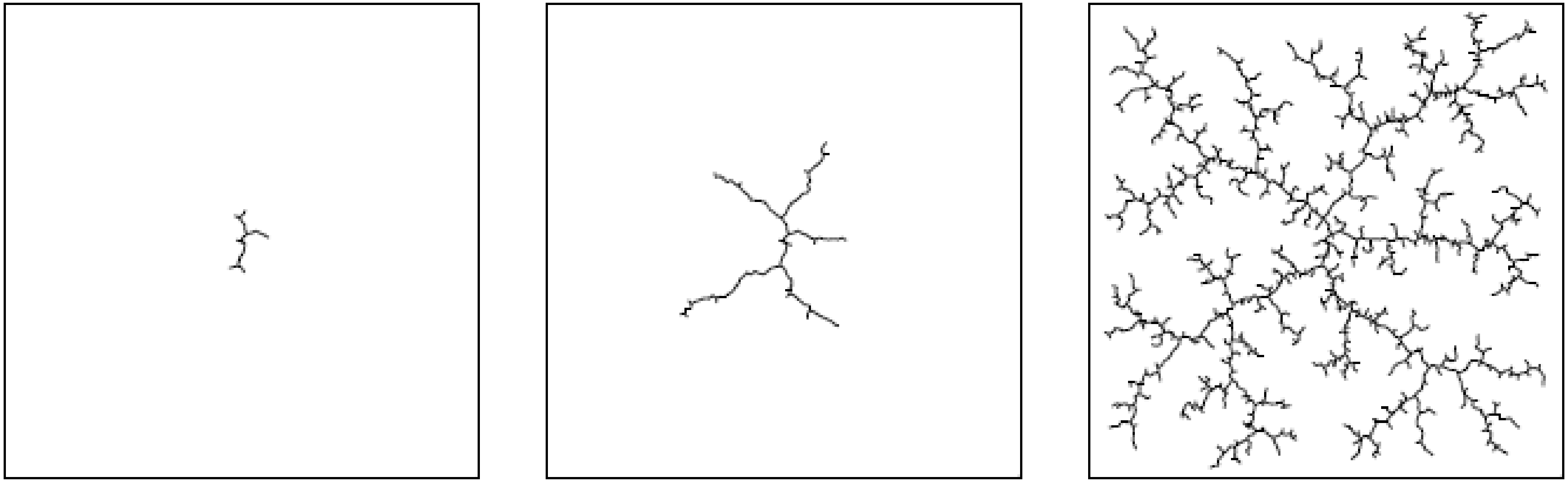


- RRT provides uniform coverage of space

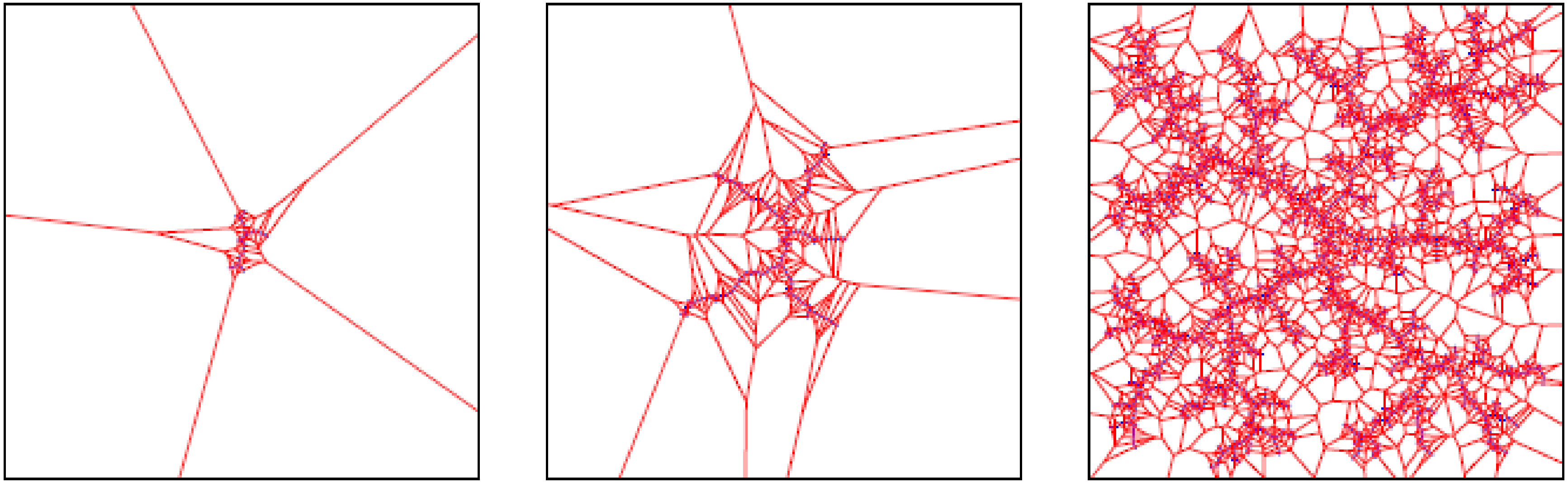
Pros/cons?

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Rapidly Exploring Random Trees (RRTs)

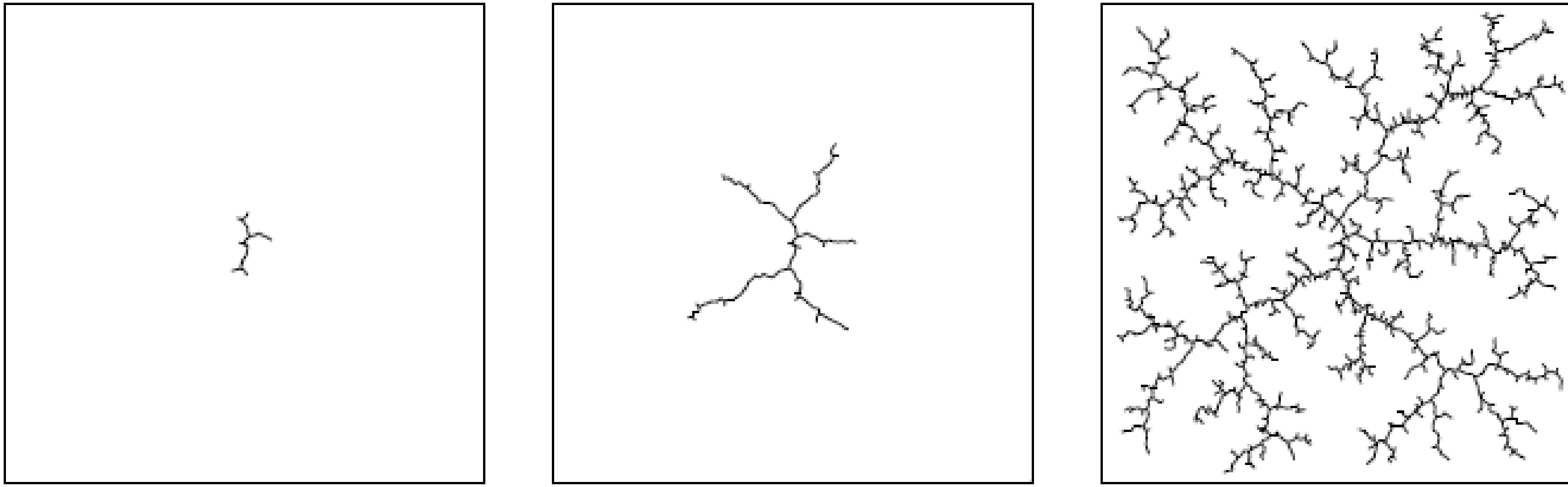


- Alternatively, the growth is always biased by the largest unexplored region

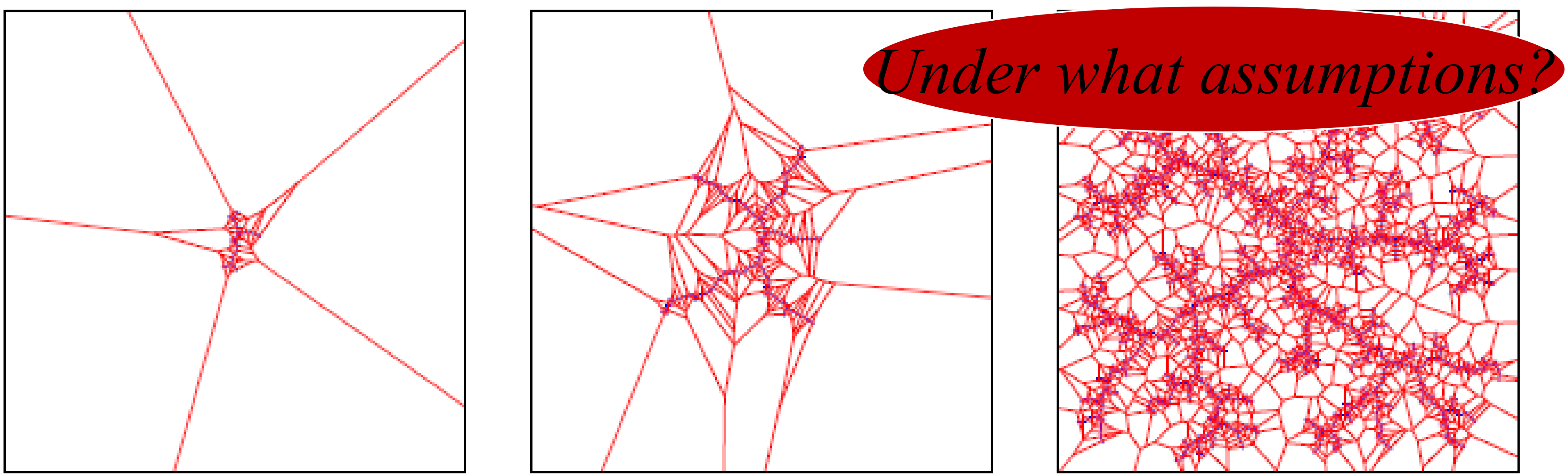


borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Rapidly Exploring Random Trees (RRTs)



- Alternatively, the growth is always biased by the largest unexplored region



borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

RRT-Connect

Bi-directional growth of the tree

$+$

relax the ε constraint on the growth of the tree

RRT-Connect

RRT_CONNECT_PLANNER(q_{init}, q_{goal})

$\mathcal{T}_a$.init(q_{init}); $\mathcal{T}_b$.init(q_{goal});
for $k = 1$ **to** K **do**
    $q_{rand} \leftarrow$ RANDOM_CONFIG();
    if not (EXTEND$(\mathcal{T}_a, q_{rand})$ = *Trapped*) **then**
        if (CONNECT$(\mathcal{T}_b, q_{new})$ = *Reached*) **then**
            Return PATH$(\mathcal{T}_a, \mathcal{T}_b)$;
    SWAP$(\mathcal{T}_a, \mathcal{T}_b)$;
Return *Failure*

CONNECT$(\mathcal{T}, q)$

repeat
    $S \leftarrow$ EXTEND$(\mathcal{T}, q)$;
until not (S = *Advanced*)
Return S;

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

RRT-Connect

RRT_CONNECT_PLANNER(q_{init}, q_{goal})

$\mathcal{T}_a$.init(q_{init}); $\mathcal{T}_b$.init(q_{goal});
for $k = 1$ **to** K **do**
    $q_{rand} \leftarrow$ RANDOM_CONFIG();
    if not (EXTEND($\mathcal{T}_a, q_{rand}$) $=$ *Trapped*) **then**
        if (CONNECT($\mathcal{T}_b, q_{new}$) $=$ *Reached*) **then**
            Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
    SWAP($\mathcal{T}_a, \mathcal{T}_b$);
Return *Failure*

tries to grow T_b to q_{new} that was just added to T_a

Why swap the trees?

CONNECT($\mathcal{T}, q$)

repeat
    $S \leftarrow$ EXTEND($\mathcal{T}, q$);
until not ($S =$ *Advanced*)
Return S;

CONNECT function grows the tree by more than just one ε

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

RRT-Connect

- For any $q \in C_{free}$, $lim_{k\rightarrow\infty} P[d(q) < \varepsilon] = 1$, where $d(q)$ is a distance from configuration q to the closest vertex in the tree, and assuming C_{free} is connected, bounded and open

- RRT-Connect is probabilistically complete: *as # of samples approaches infinity, the algorithm is guaranteed to find a solution if one exists*

Sampling-based approaches

Typical setup:

- Run PRM/RRT/RRT-Connect/…
- Post-process the generated solution to make it more optimal

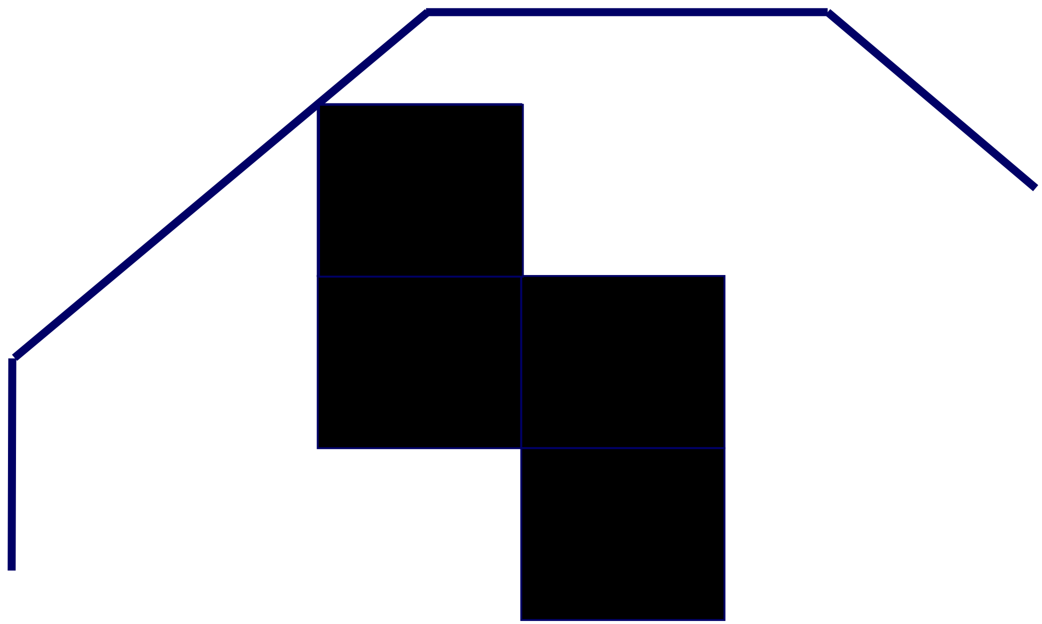
An important but
often time-consuming step

Could also be highly non-trivial

Post-processing

Any ideas how to post-process it?

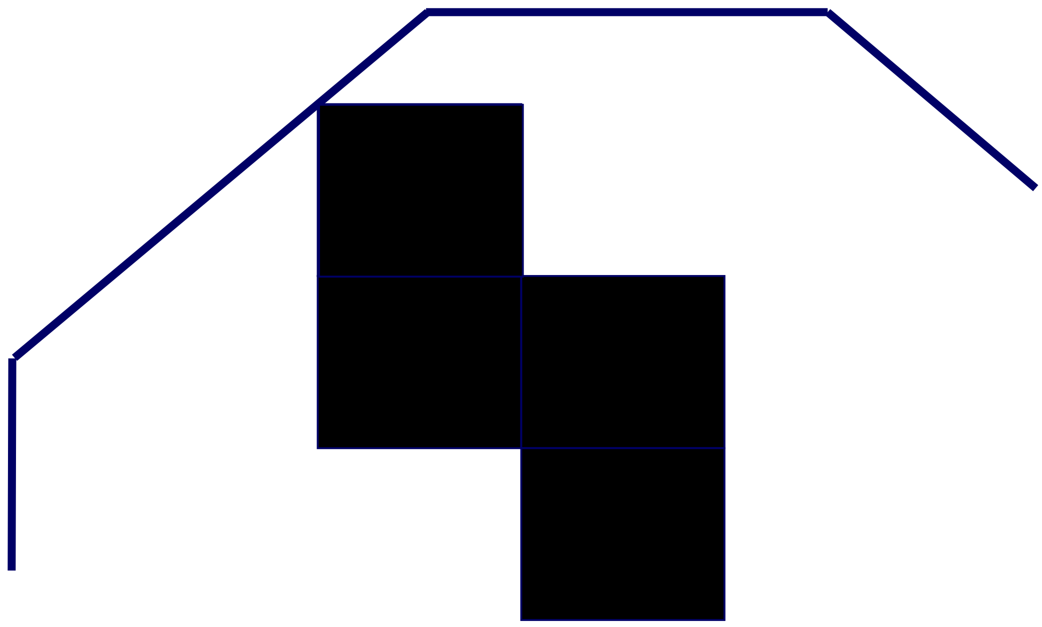
Consider this path generated by RRT or PRM or A on a grid-based graph:*



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

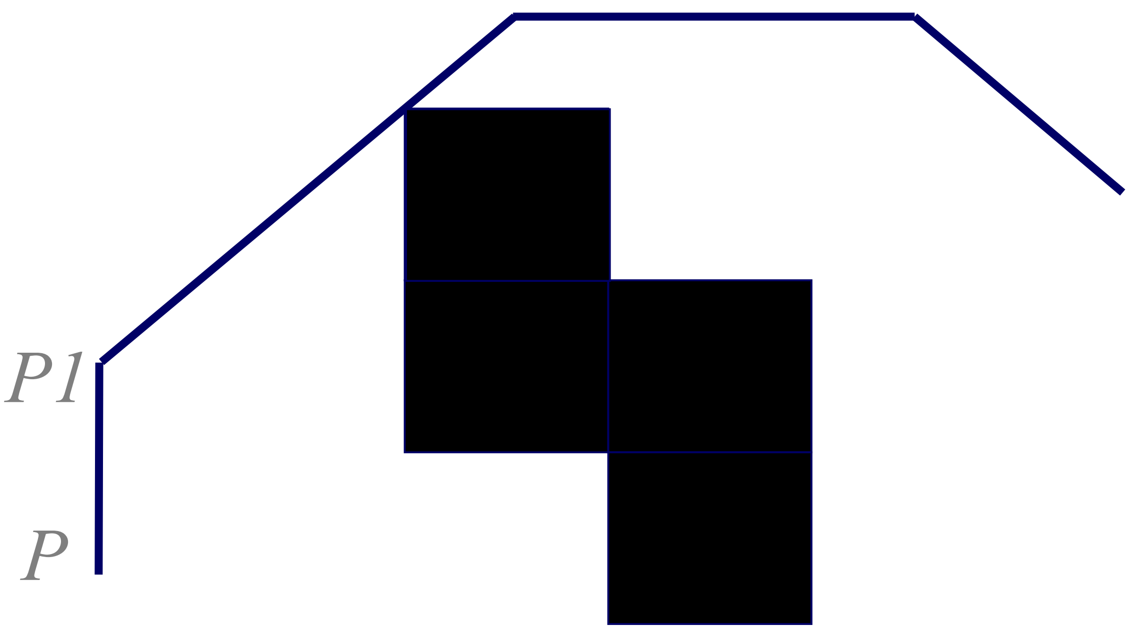
NewPath=[]; P=start point, P1 = point P+1 along the path
while P != goal point
    while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point
        P1 = point P1+1 along the path;
    NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

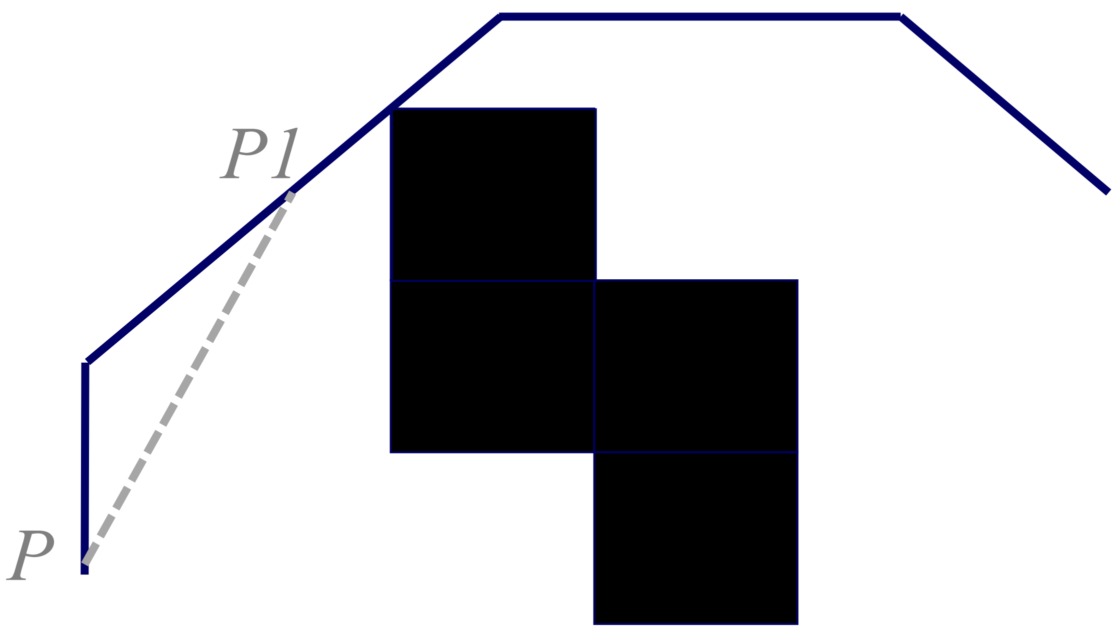
NewPath=[]; P=start point, P1 = point P+1 along the path
while P != goal point
while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point
P1 = point P1+1 along the path;
NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

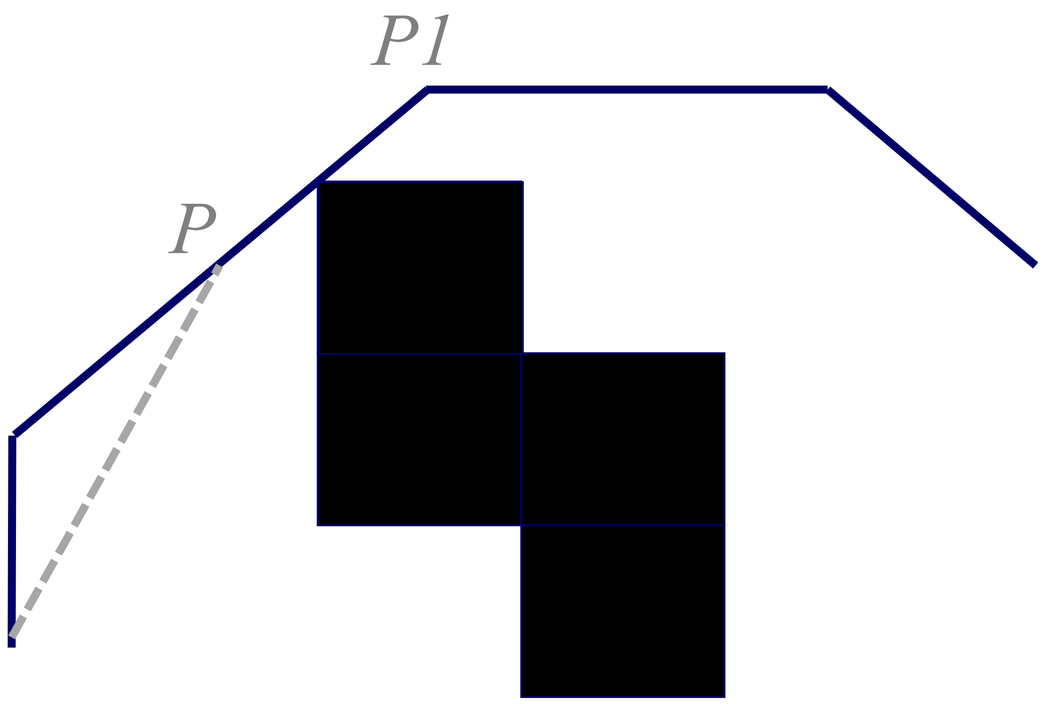
NewPath=[]; P=start point, P1 = point P+1 along the path
while P != goal point
    while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point
        P1 = point P1+1 along the path;
    NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

NewPath=[]; P=start point, P1 = point P+1 along the path
while P != goal point
    while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point
        P1 = point P1+1 along the path;
    NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

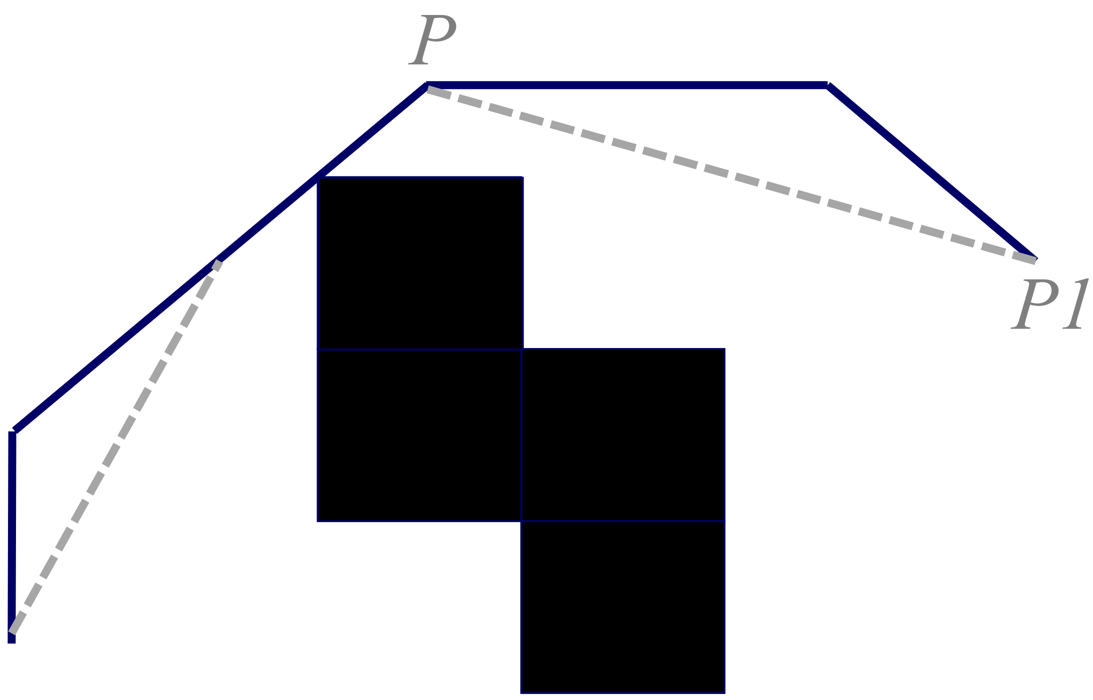
  NewPath=[]; P=start point, P1 = point P+1 along the path

  while P != goal point

      while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point

          P1 = point P1+1 along the path;

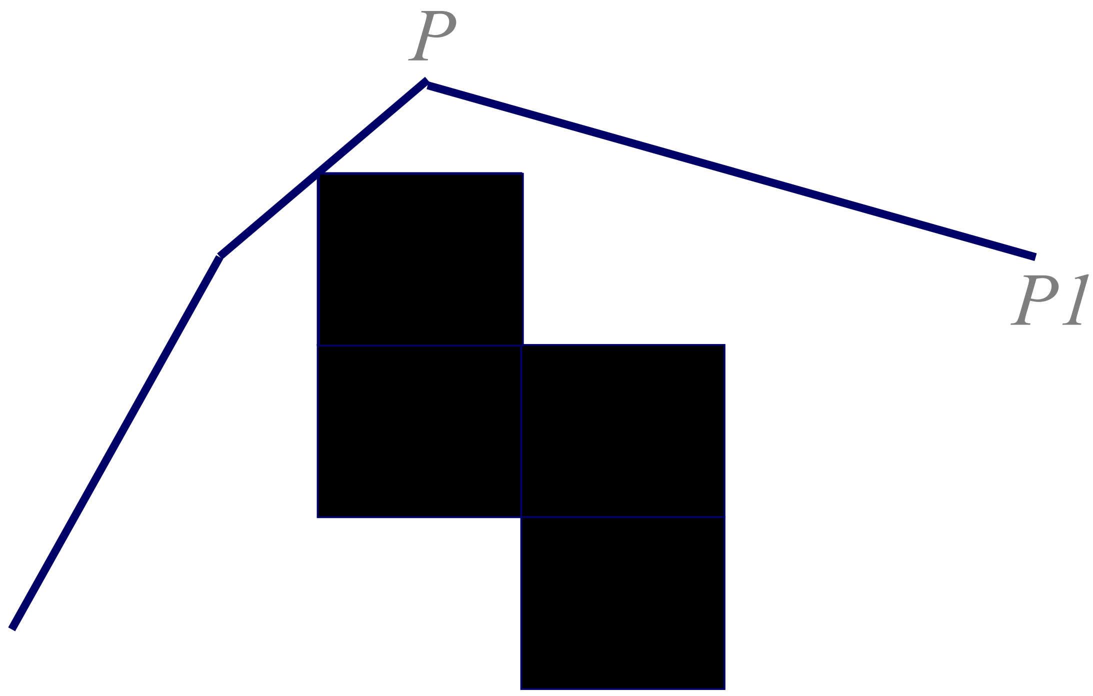
      NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



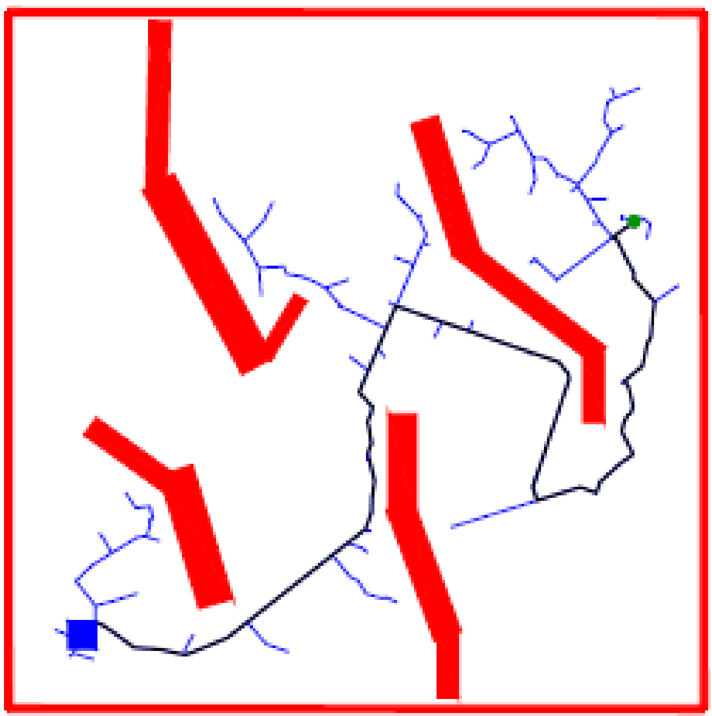
Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

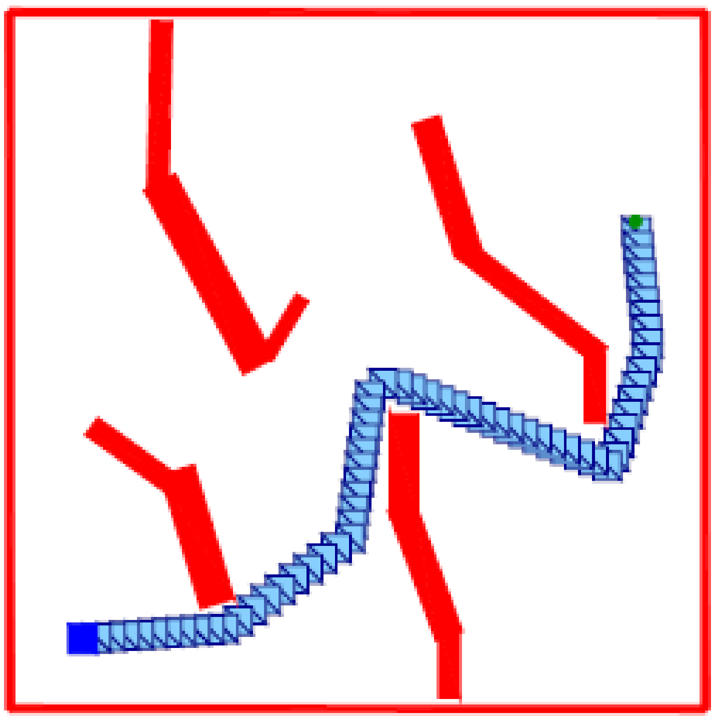
NewPath=[]; P=start point, P1 = point P+1 along the path
while P != goal point
  while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point
    P1 = point P1+1 along the path;
  NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Examples of RRT in action



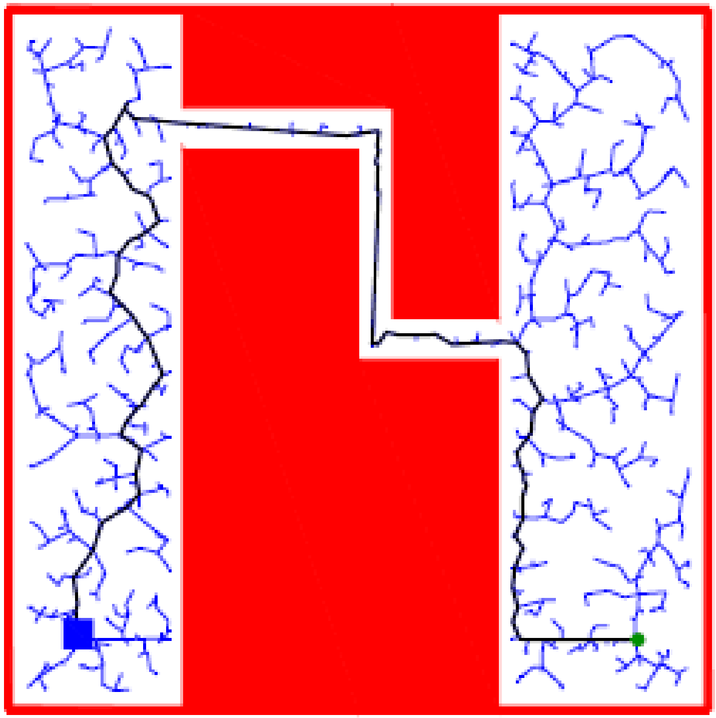
RRT-connect



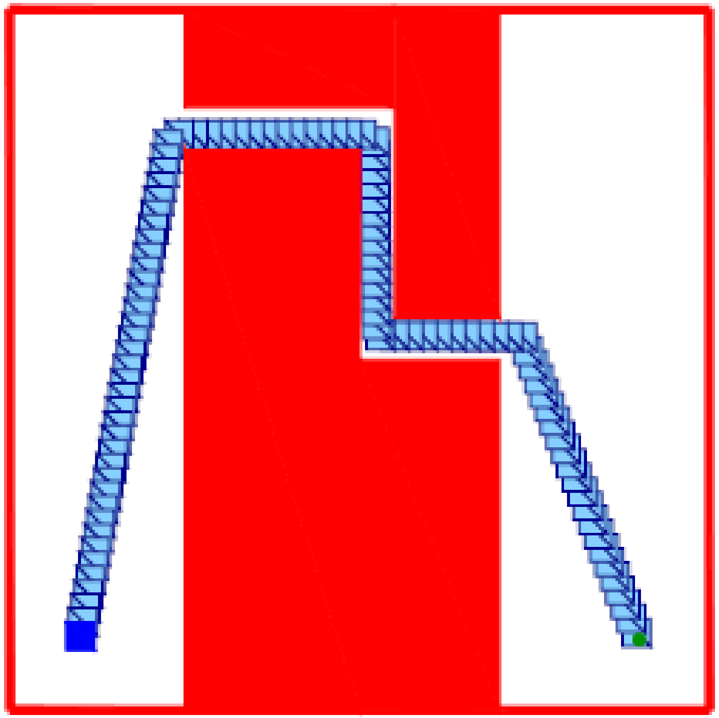
path after postprocessing

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Examples of RRT in action



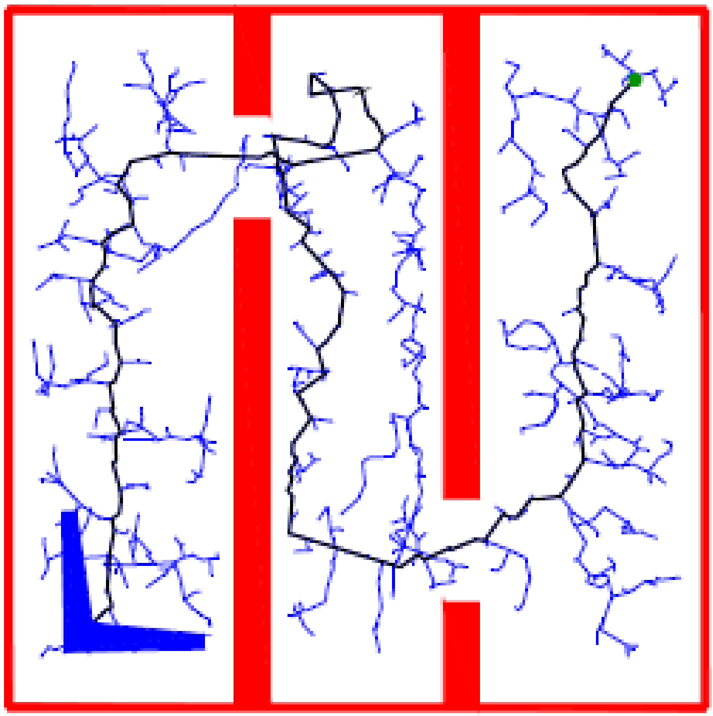
RRT-connect



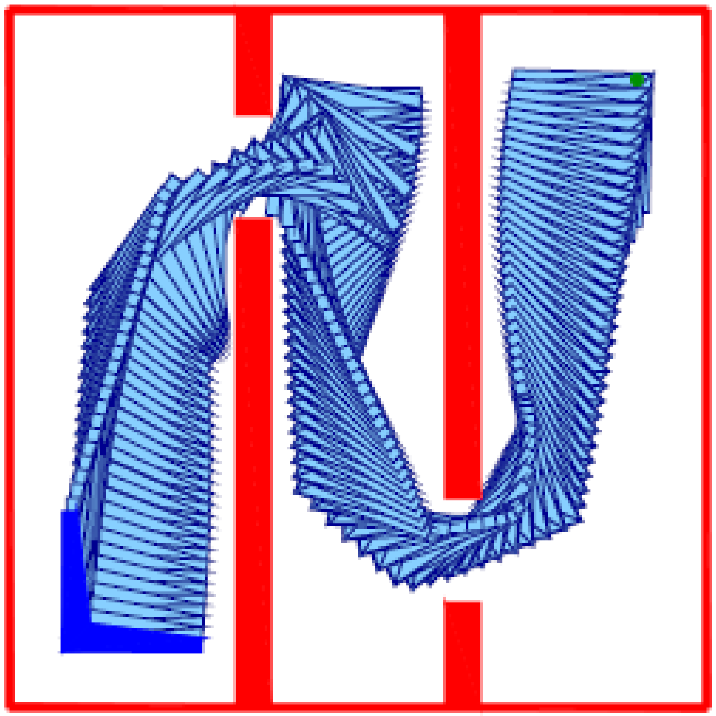
path after postprocessing

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Examples of RRT in action



RRT-connect



path after postprocessing

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

PRMs vs. RRTs

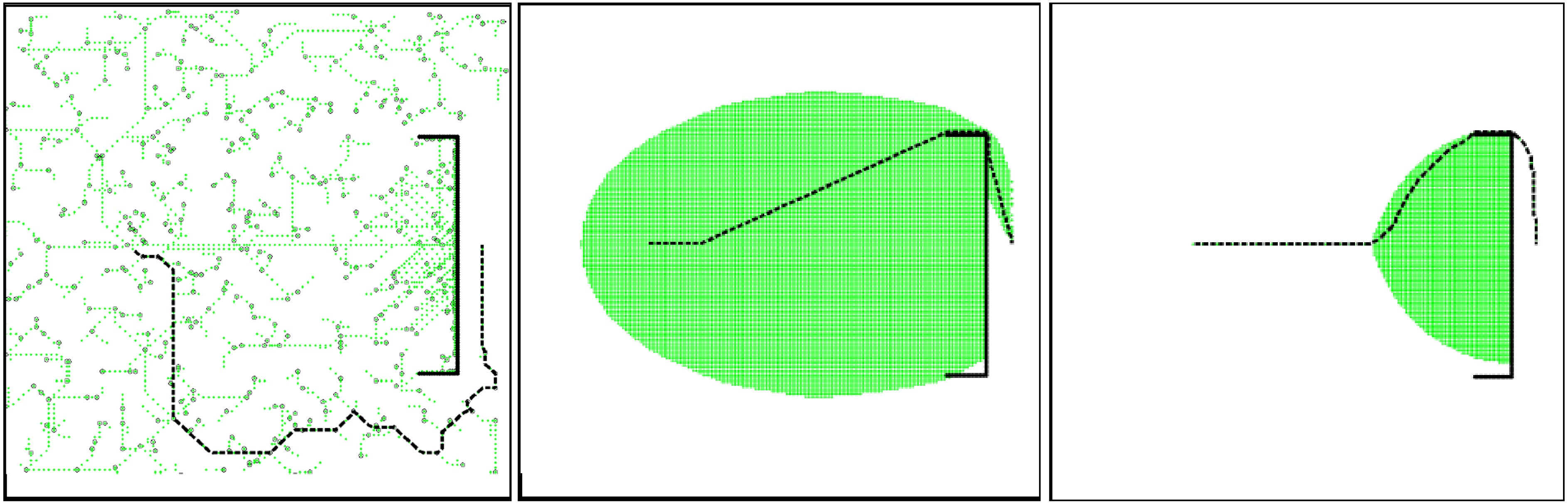
- PRMs construct a roadmap and then searches it for a solution whenever q_I, g_G are given
  - well-suited for repeated planning in between different pairs of q_I, g_G *(multiple queries)*

- RRTs construct a tree for a given q_I, q_G until the tree has a solution
  - well-suited for single-shot planning in between a single pair of q_I, g_G *(single query)*

  - There exist extensions of RRTs that try to reuse a previously constructed tree when replanning in response to map updates

RRTs vs A*-based planning

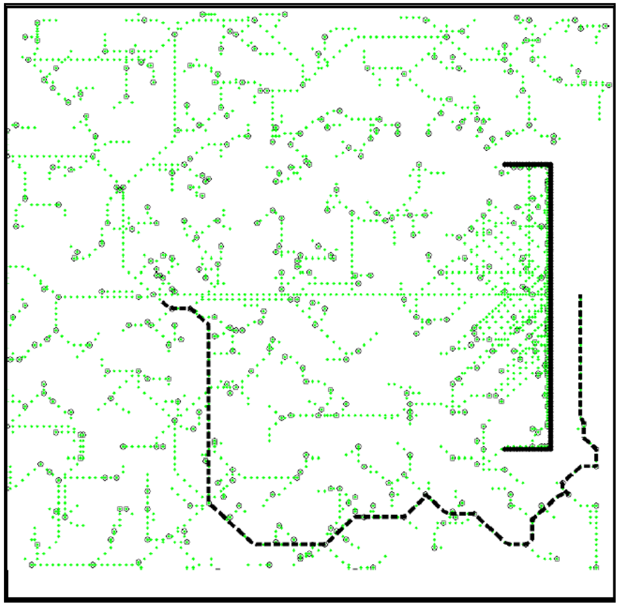
RRT *A** *wA* with ε = 3*



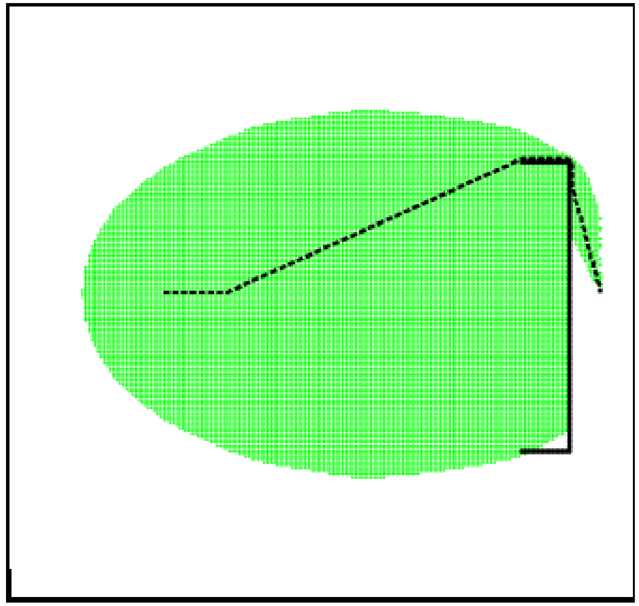
- RRTs:
  - sparse exploration, usually little memory and computations required, works well in high-D
  - solutions can be highly sub-optimal, requires post-processing, which in some cases can be very hard to do, the solution is still restricted to the same homotopic class

RRTs vs A*-based planning

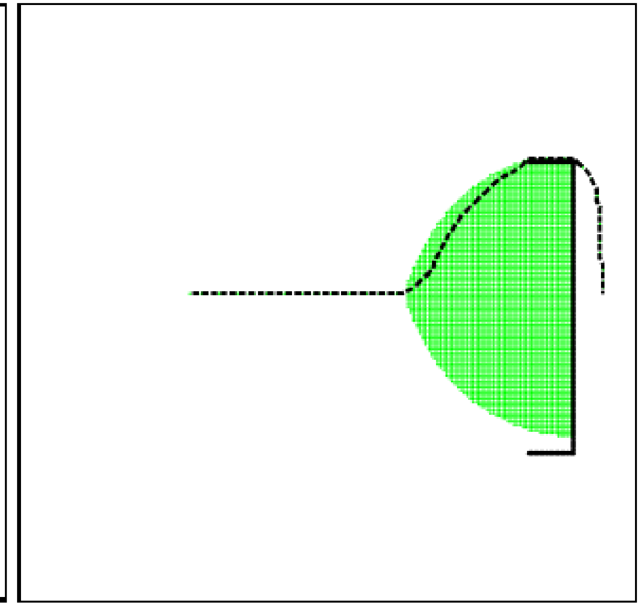
RRT



*A**

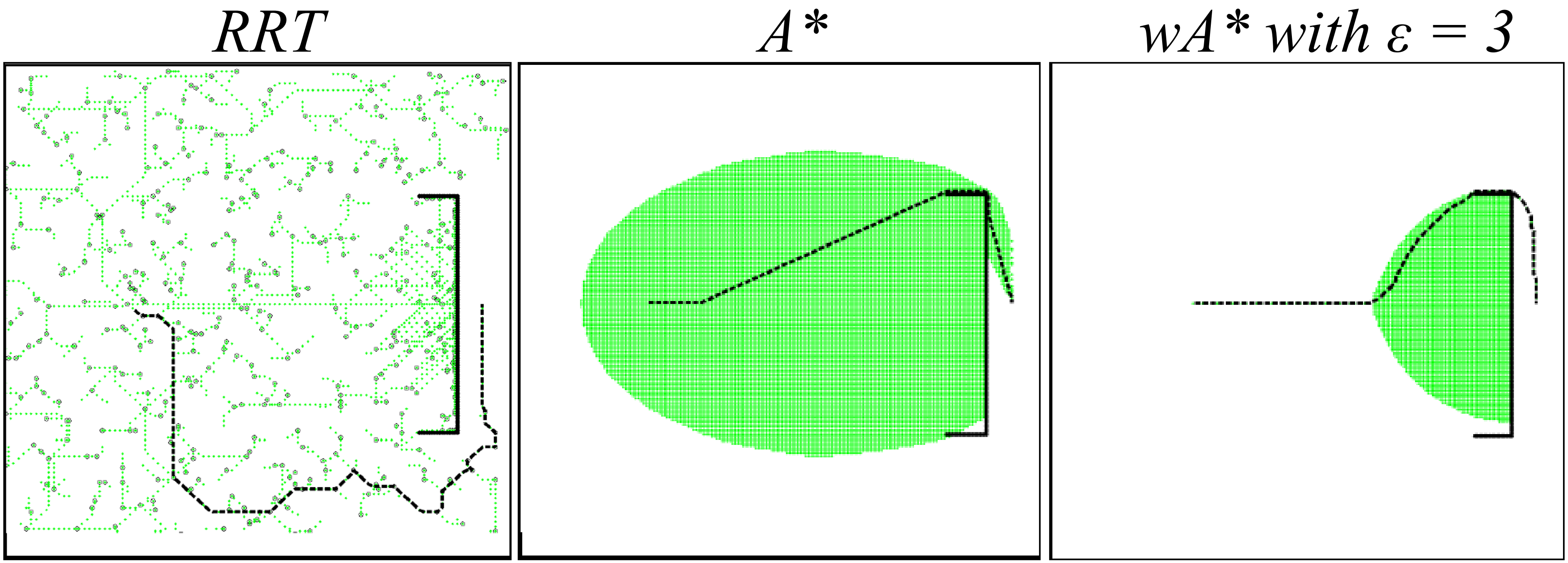


wA with* $\varepsilon = 3$



- RRTs:
  - does not incorporate a (potentially complex) cost function
  - there exist versions (e.g., RRT*) that try to incorporate the cost function and converge to a provably least-cost solution in the limit of samples (but typically computationally more expensive than RRT)

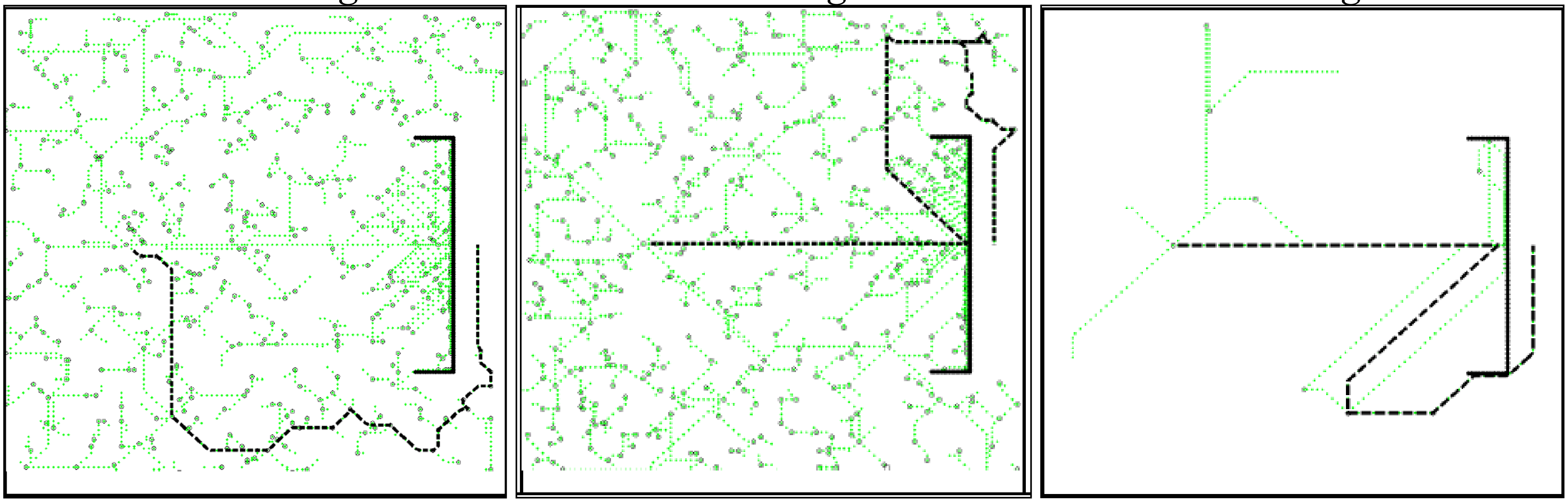
RRTs vs A*-based planning



- A* and weighted A* (wA*):
  - returns a solution with optimality (or sub-optimality) guarantees with respect to the discretization used
  - explicitly minimizes a cost function
  - requires a thorough exploration of the state-space resulting in high memory and computational requirements

Sampling in RRTs

RRT, $P_g=0$ *RRT,* $P_g=0.1$ *RRT,* $P_g=0.5$

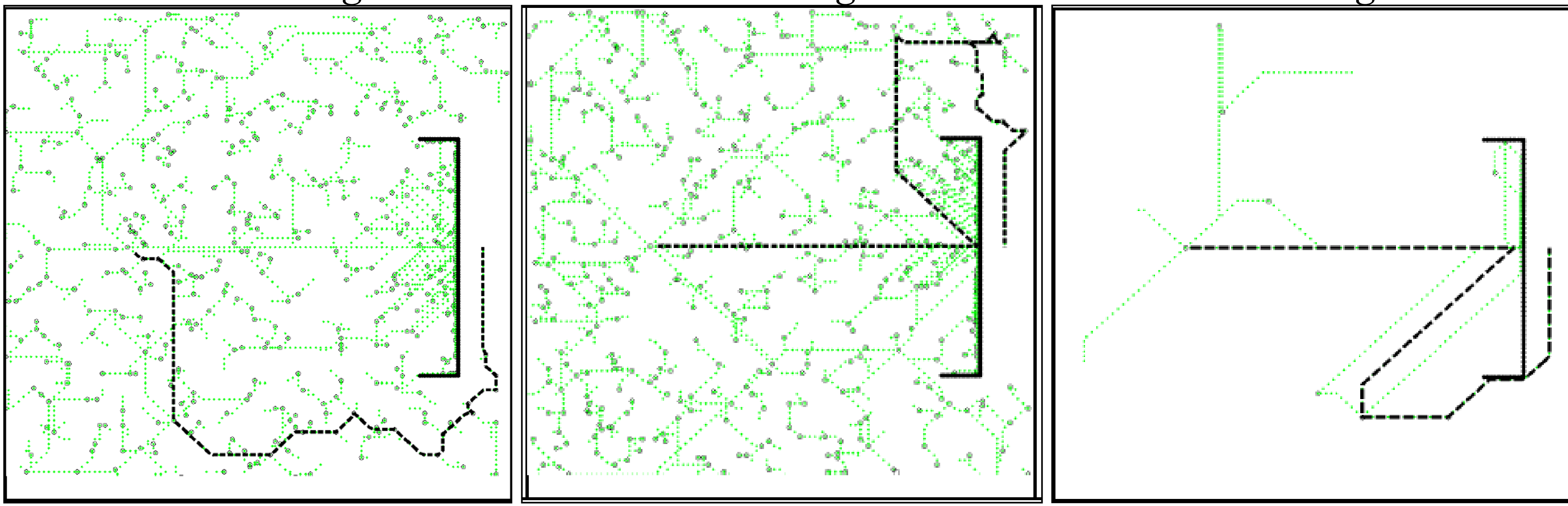


- Uniform: q_{rand} is a random sample in C_{free}

- Goal-biased: with a probability $(1\text{-}P_g)$, q_{rand} is chosen as a random sample in C_{free}, with probability P_g, q_{rand} is set to g_G

Sampling in RRTs

RRT, P_g=0 *RRT, P_g=0.1* *RRT, P_g=0.5*



- Uniform: q_{rand} is a random sample in C_{free}

- Goal-biased: with a probability $(1\text{-}P_g)$, q_{rand} is chosen as a random sample in C_{free}, with probability P_g, q_{rand} is set to g_G

Very useful!

Summary

- RRT
  - interleaves tree construction and search
  - great for high-dimensional planning in continuous spaces
  - RRT-Connect typically much faster than RRT
  - Good post-processing is <u>very</u> important

- Sampling-based approaches are heavily used in planning for articulated robots (e.g., arms, humanoids, etc.)

- Provide probabilistic guarantees (i.e., in the limit of the number of samples)