Robot Autonomy

Lecture 7:
Motion Planning II

Oliver Kroemer

Piano Mover Problem

- Given a workspace W with obstacles $O \subset W$ and a c-space C with mapping function $A(q) \subset W$ where $q \in C$ then:

$$C_{obs} = \{q \in C | A(q) \cap O \neq \emptyset\}$$

$$C_{free} = C \setminus C_{obs}$$

- Want the robot to find a continuous path τ in C_{free} from initial configuration q_i to goal configuration q_g

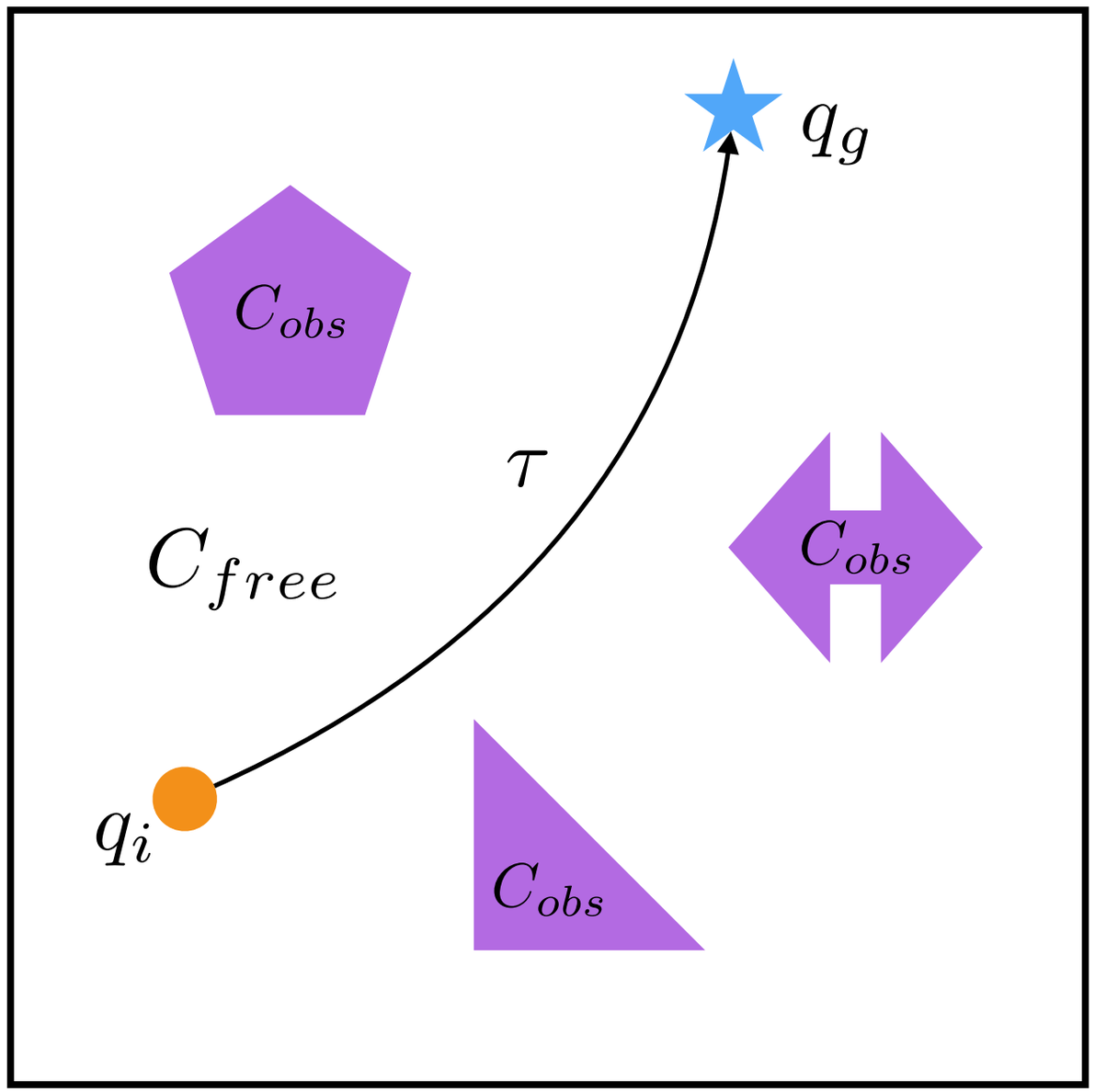
$$\tau : [0, 1] \rightarrow C_{free}$$

$$\tau(0) = q_i$$

$$\tau(1) = q_g$$

- Assume that robot can move in any direction in c-space

Piano Mover Problem



Topological Graphs

- Use graph to turn continuous problem into discrete one
- Define a topological graph $G(V, E)$ as:
  - Vertices

$$v_i \in C_{free}$$

  - Edges

$$e_{ij} \equiv \tau : [0, 1] \rightarrow C_{free}$$

General Procedure for Sample-based Planning

- Initialization:
  initialize graph $G = (V, E)$ as empty or with $V = \{q_i, q_g\}$
- Vertex Selection:
  Sample q_{new} in C_{free} to expand G
- Local Planning:
  Attempt to create paths between q_{new} and existing V
- Check if Done:
  Check termination conditions or goto vertex selection
- Search graph for to find a path for the query q_i to q_g

Completeness

- Complete:
  Planner is complete if 1) it always returns a solution if one exists and 2) otherwise returns a failure in bounded time

- Probabilistic Complete:
  Planner is probabilistic complete if the probability of a solution existing tends to zero as the number of sampled points increases and no solution is found. No time bound.

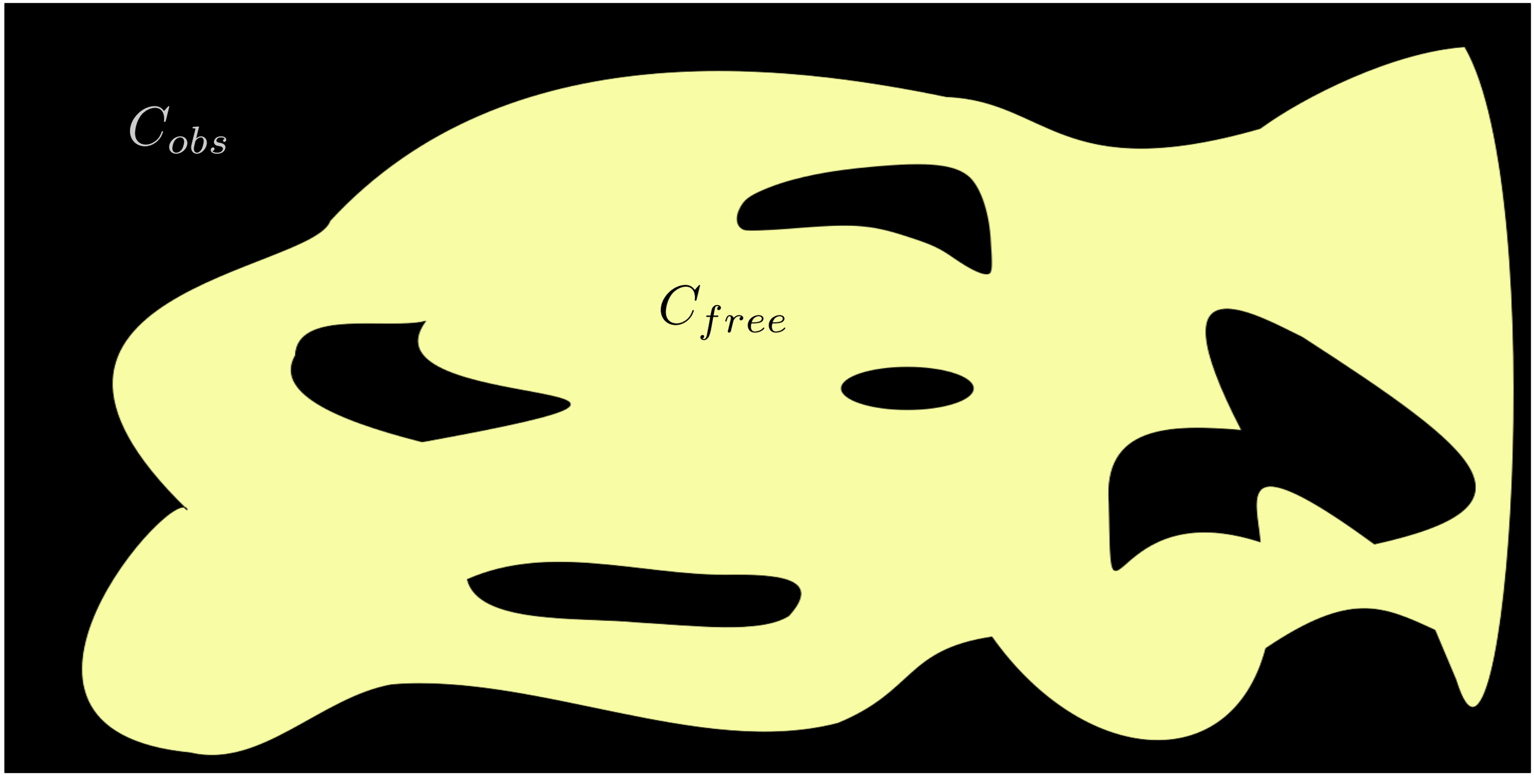
- Focus on feasibility rather than optimality

Multi- vs Single- Query Sample-Based Planners

- Multi-query Planner
  - Build a roadmap for multiple planning queries
  - Assumes fixed obstacles across multiple queries
  - Cover C_{free} and capture its connectivity
    - Probabilistic Roadmaps (PRMs)
- Single-query Planner
  - Build a graph for individual planning queries
  - Environment may change between queries
  - Connect the initial and goal nodes as directly as possible
  - Rapidly-exploring Random Trees (RRTs)

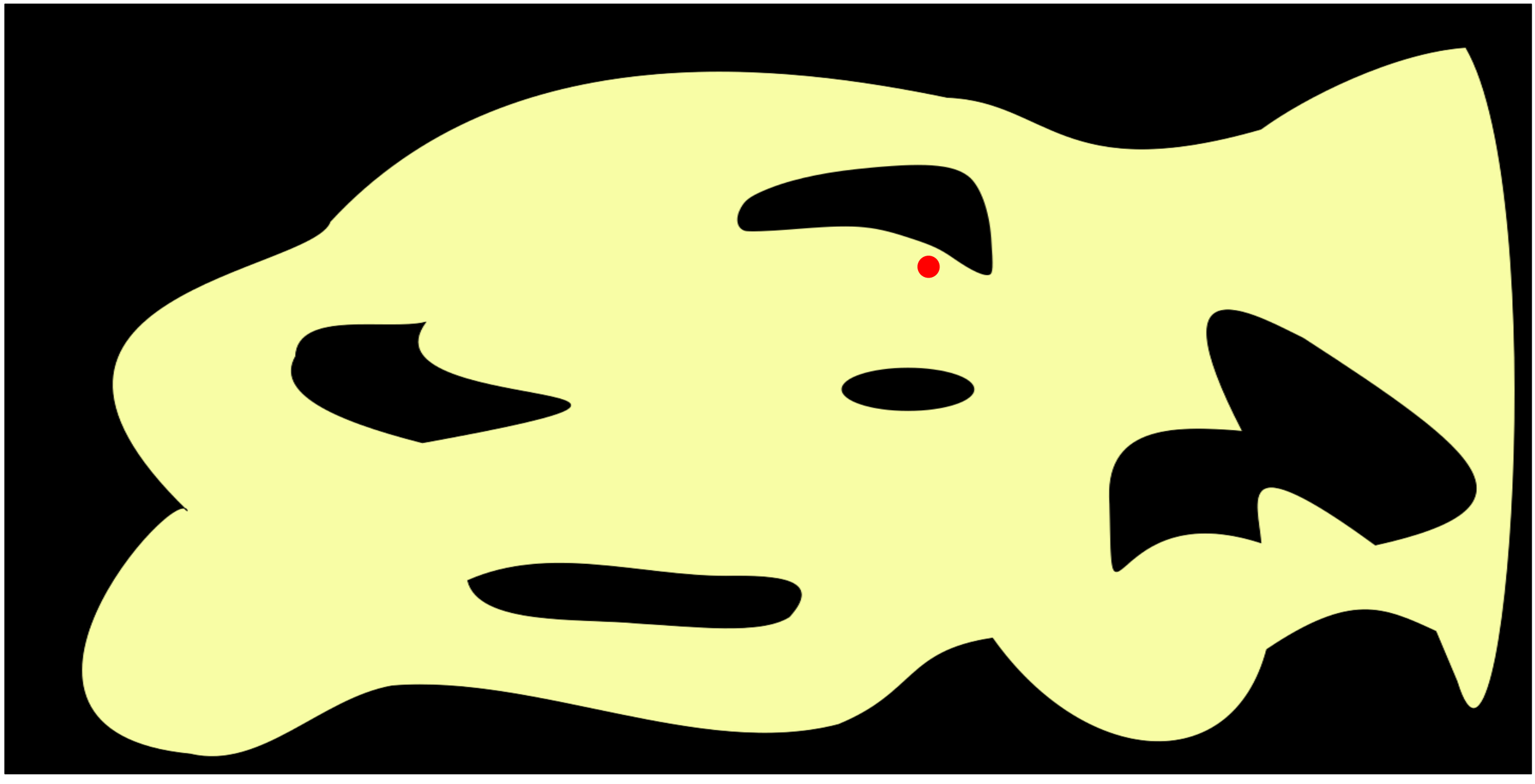
Probabilistic Road Maps

- Visualize c-space regions for clarity, NOT IN PRACTICE!



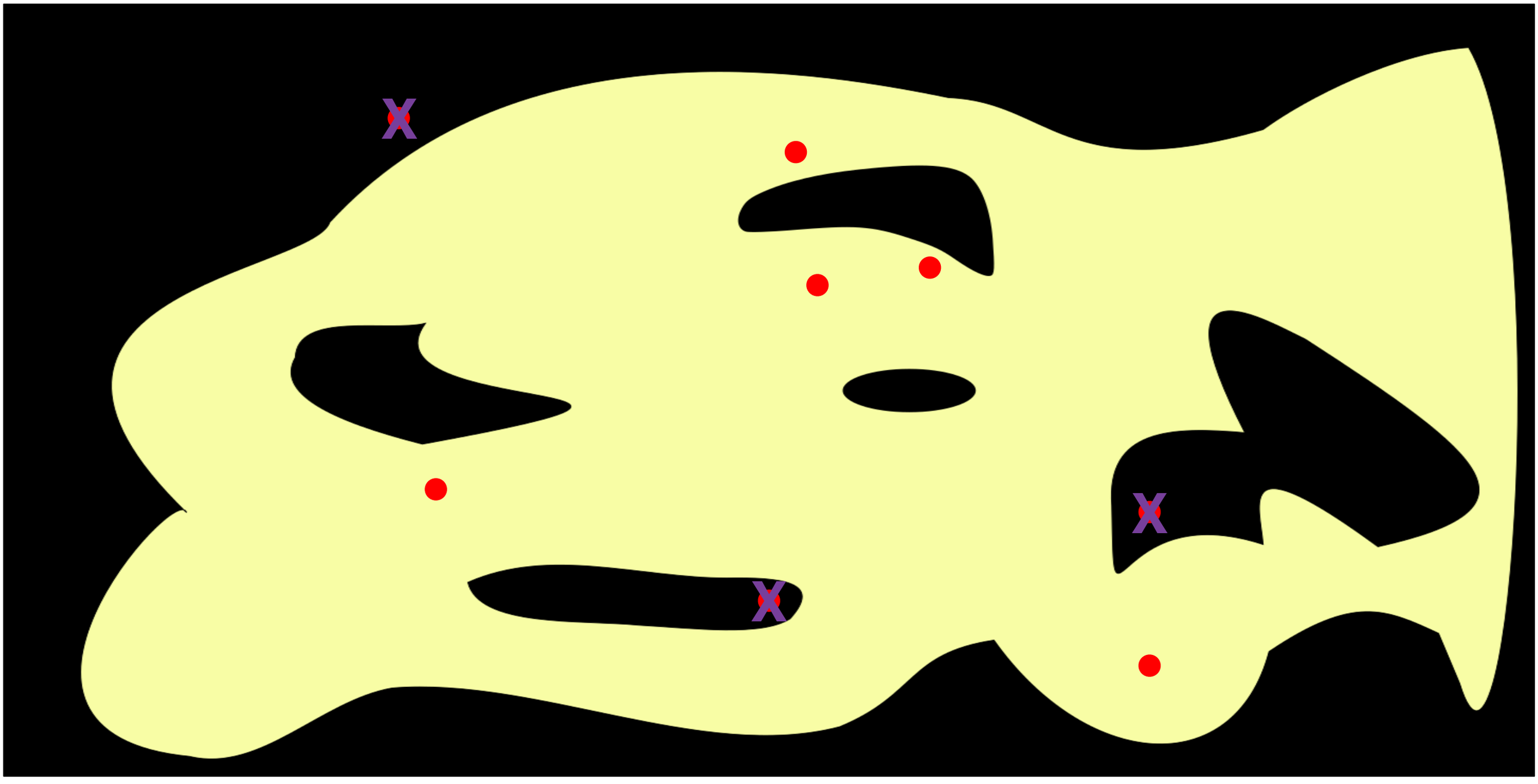
Probabilistic Road Maps

- Sample a points and test if it is in collision with obstacles



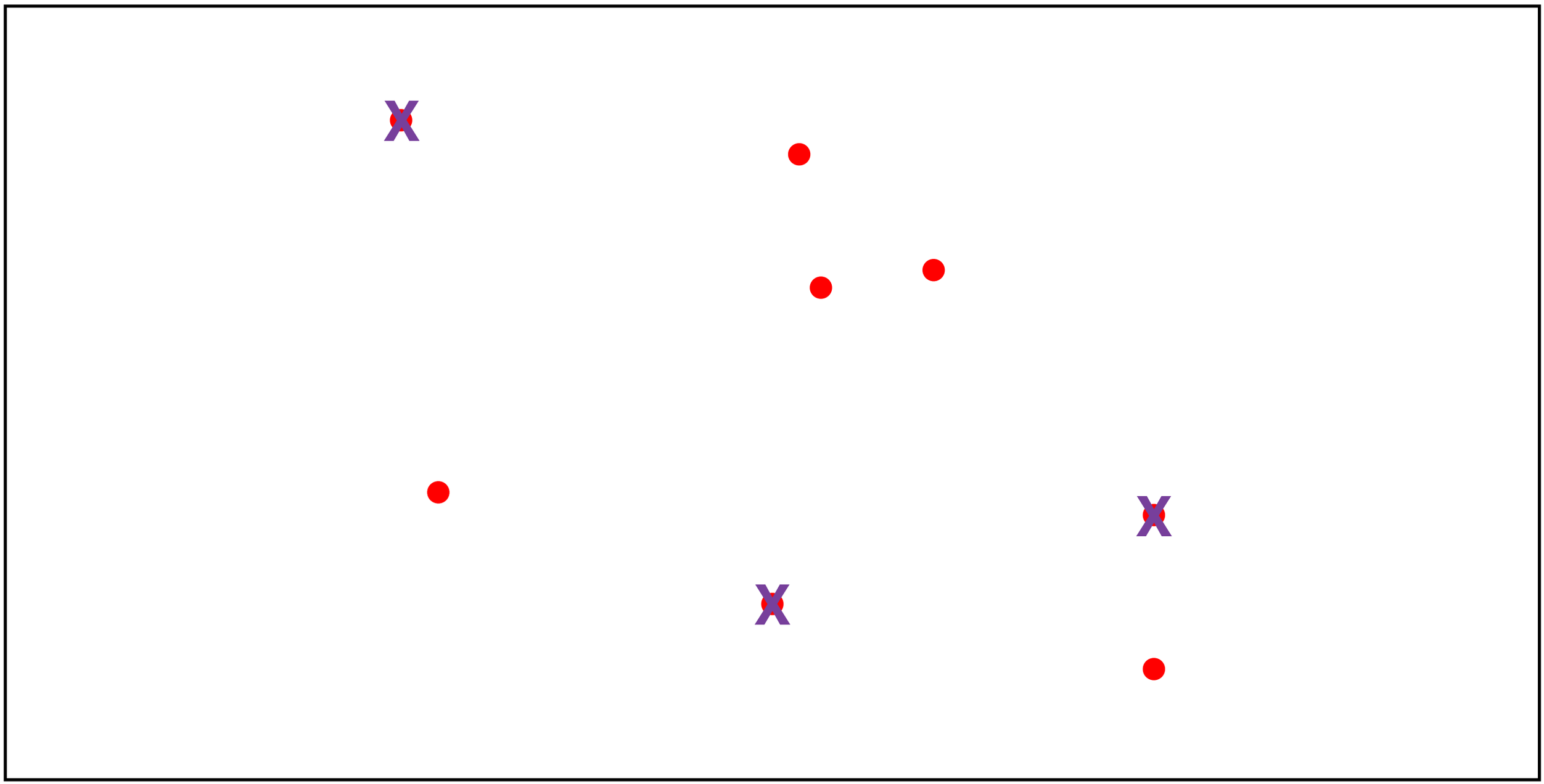
Probabilistic Road Maps

- Continue adding points and testing for collisions



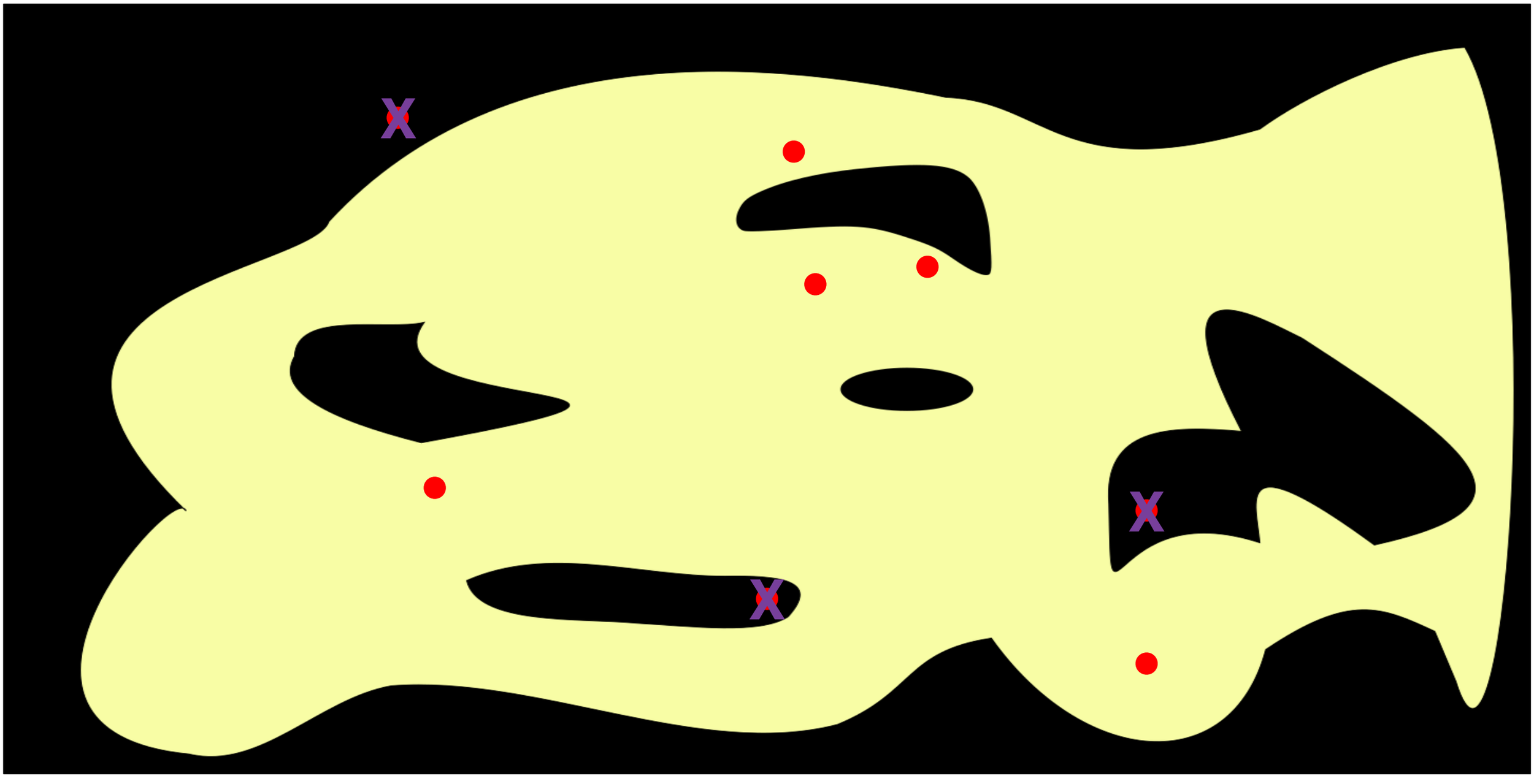
Probabilistic Road Maps

- Continue adding points and testing for collisions



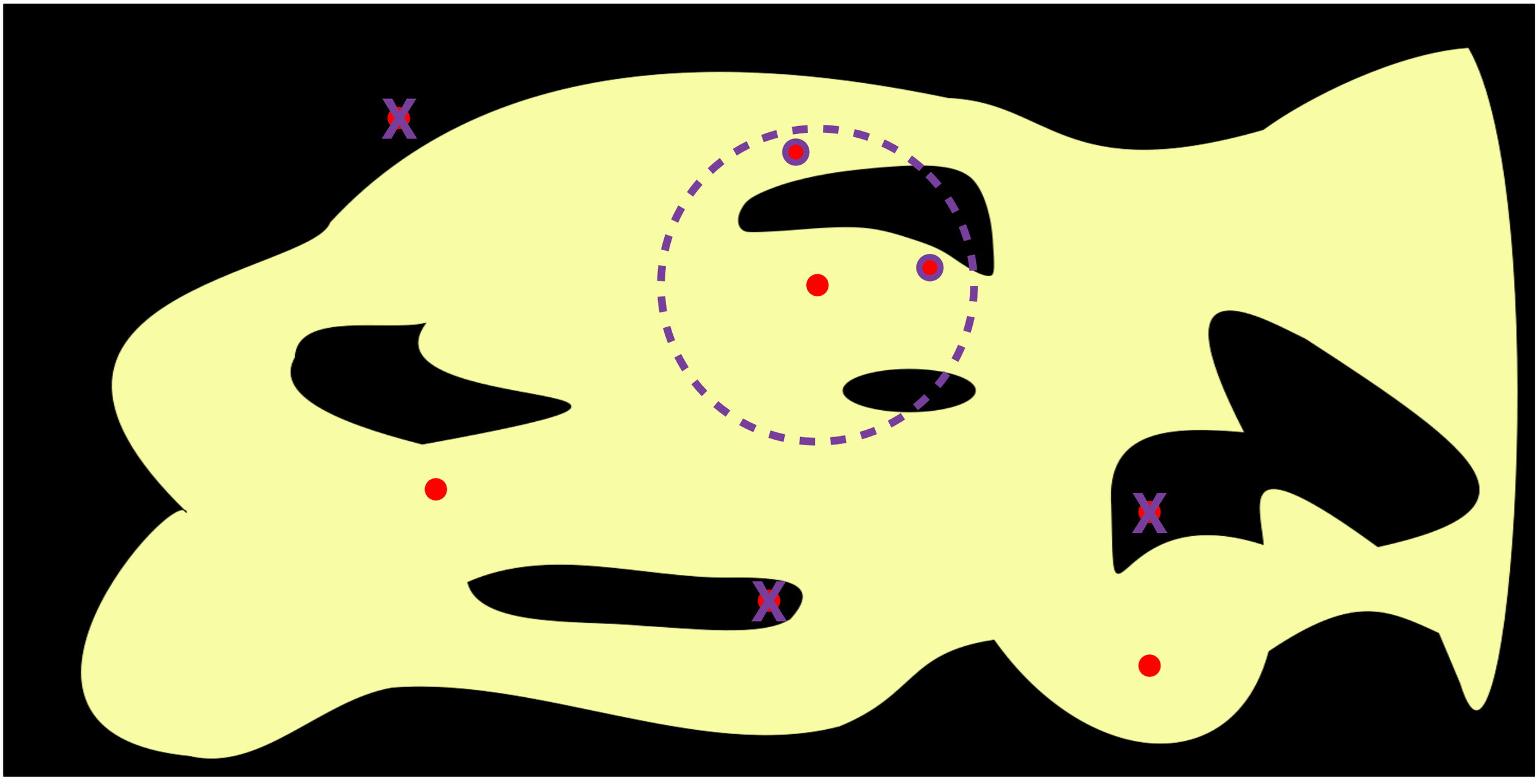
Probabilistic Road Maps

- Continue adding points and testing for collisions



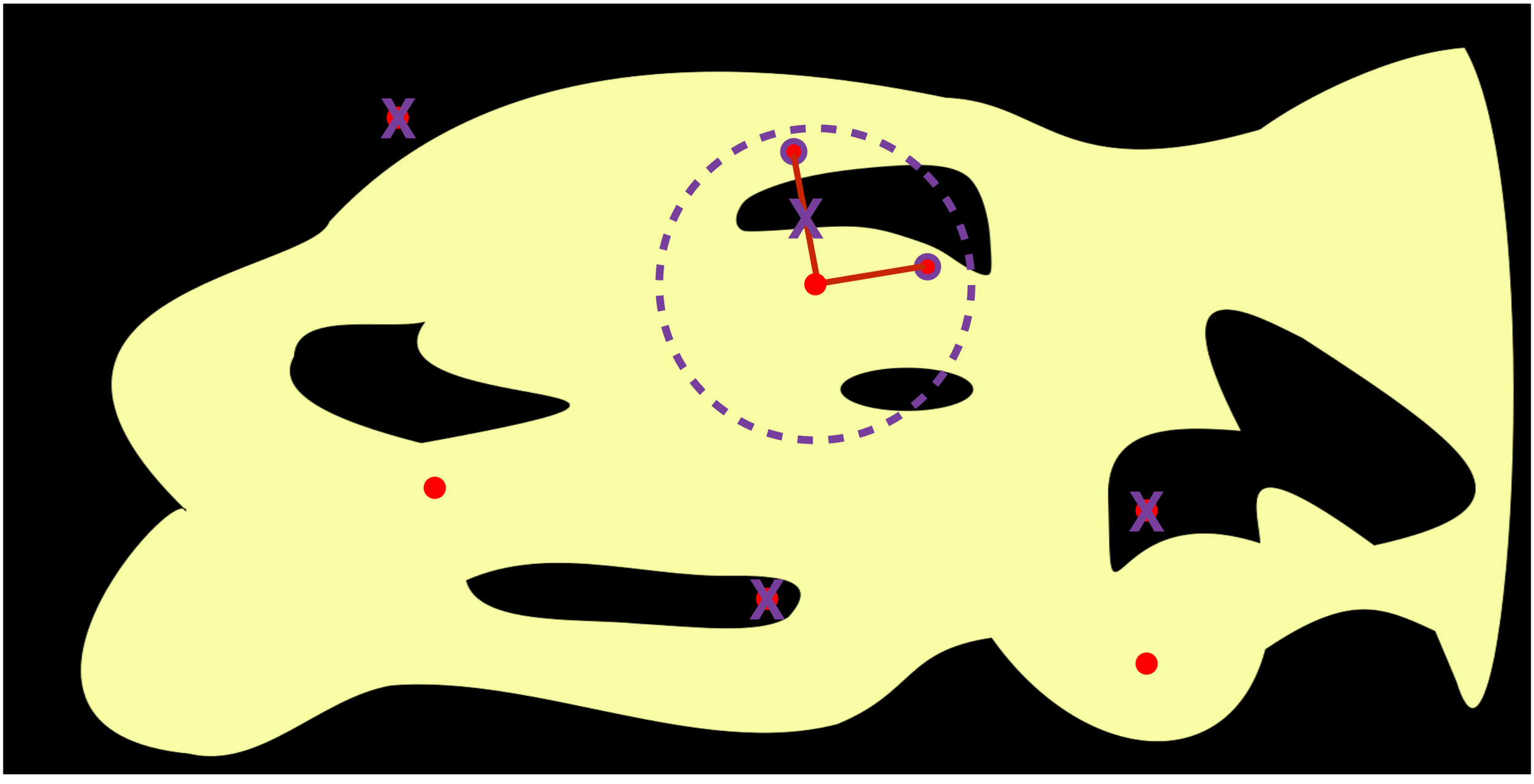
Probabilistic Road Maps

- For each new point, determine neighbouring points



Probabilistic Road Maps

- For each new point, determine neighbouring points



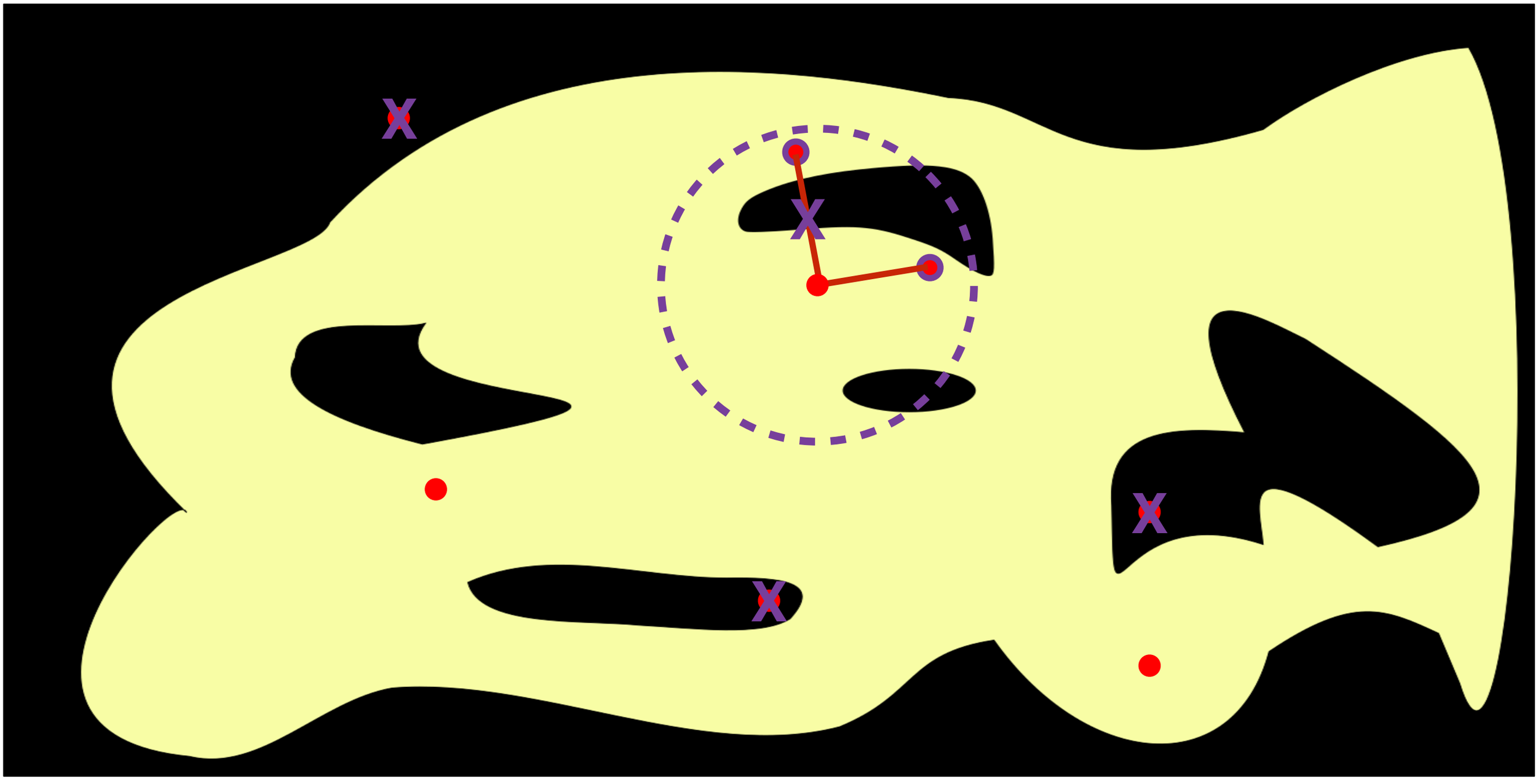
Neighbours

- Multiple methods for connect nodes (milestones)
- Radius:
  attempt to connect all neighbours within a fixed distance
- Nearest K:
  attempt to connect the node with the K nearest neighbours
- Component K:
  attempt to connect the node with the K nearest neighbours in each connected component

- More neighbours gives more connections and more tests

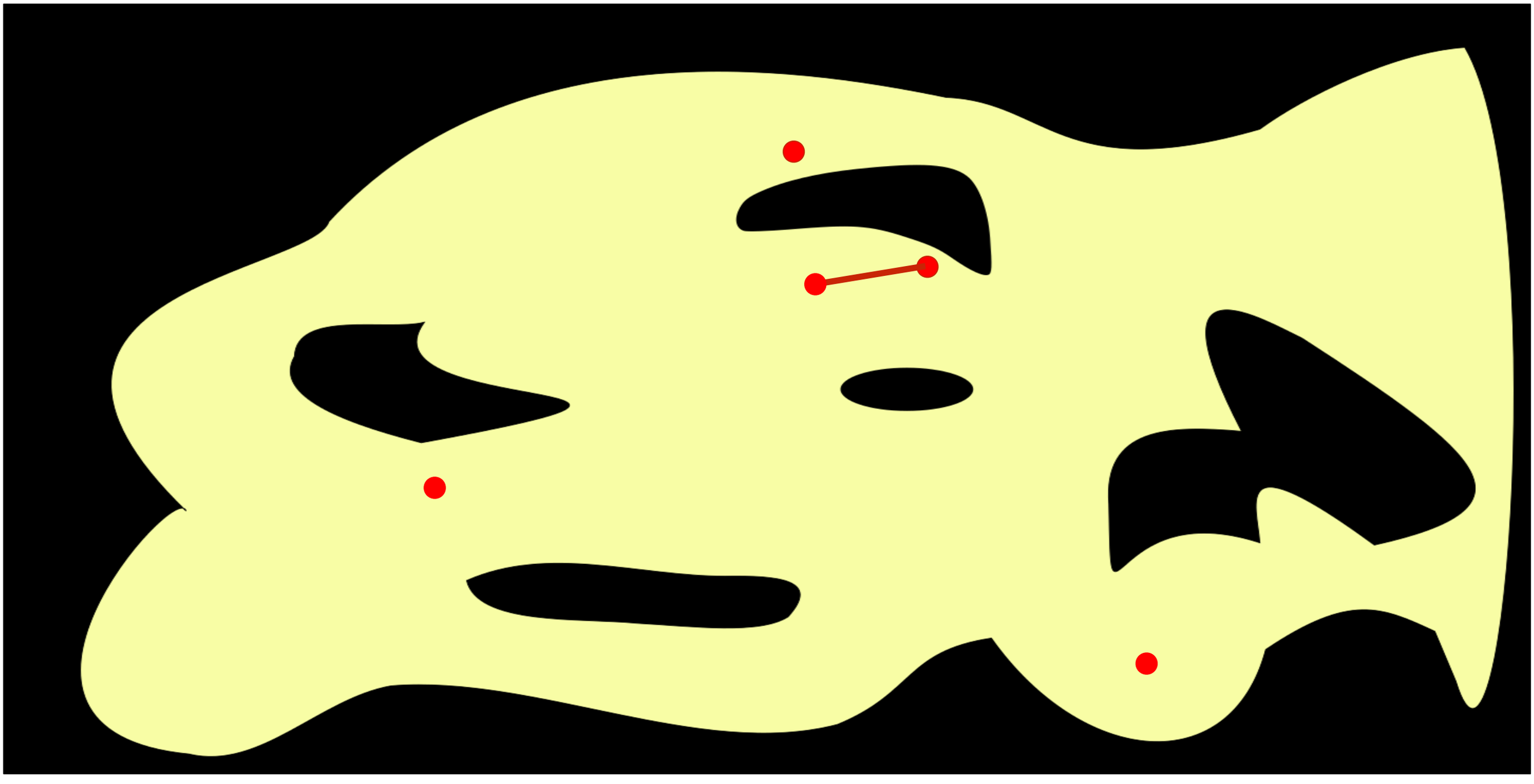
Probabilistic Road Maps

- For each new point, determine neighbouring points



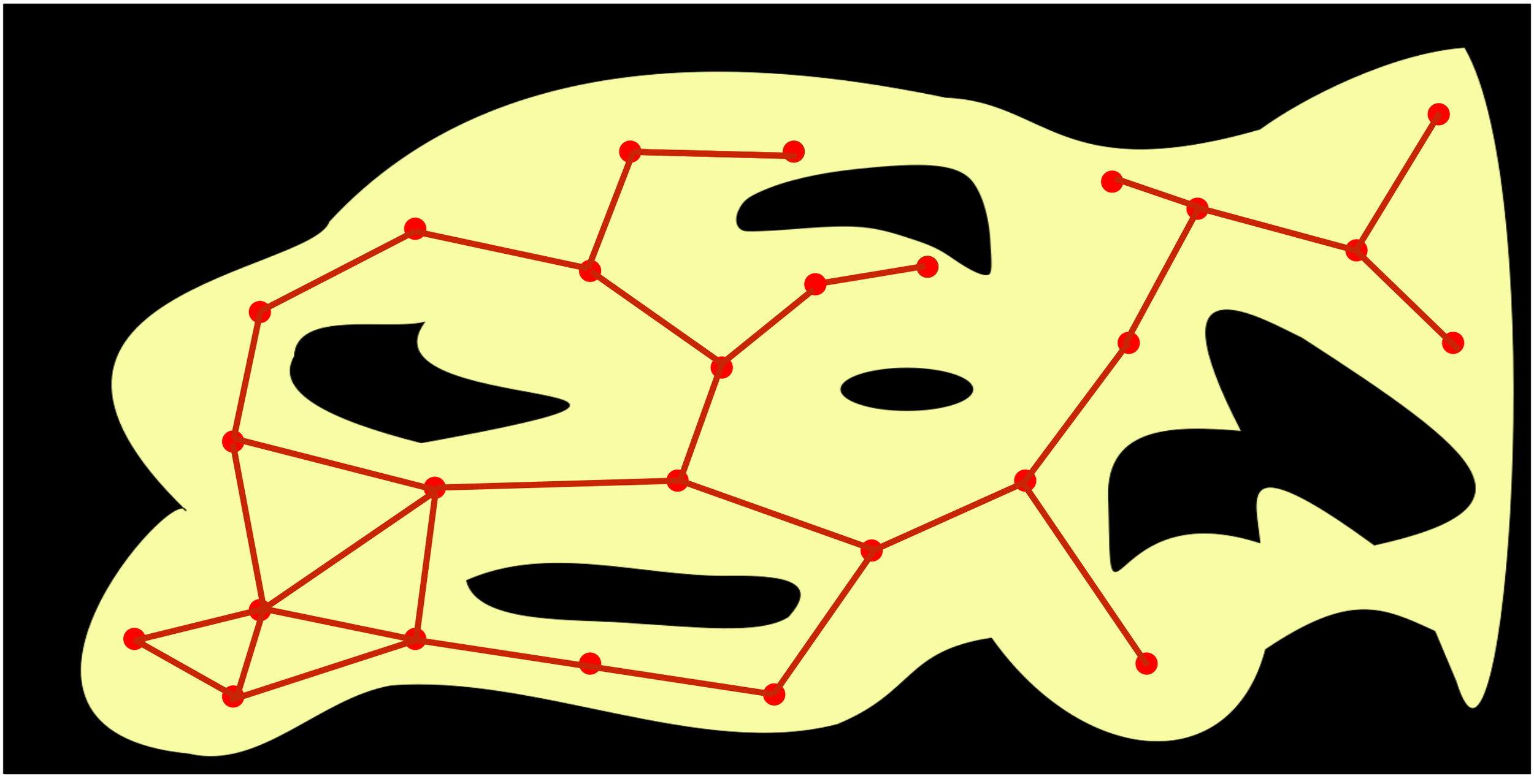
Probabilistic Road Maps

- For each new point, determine neighbouring points



Probabilistic Road Maps

- For each new point, determine neighbouring points



PRM Learning Phase

- Initialization

$$G = (V, E) \qquad V = \{\} \qquad E = \{\}$$

- Construction

while $|E| < N$

sample q_{new} from C

if $q_{new} \in C_{free}$

add q_{new} to V

for each $q \in$ Neighbor(V, q_{new})

if link(q, q_{new})

add edge $e(q, q_{new})$ to E

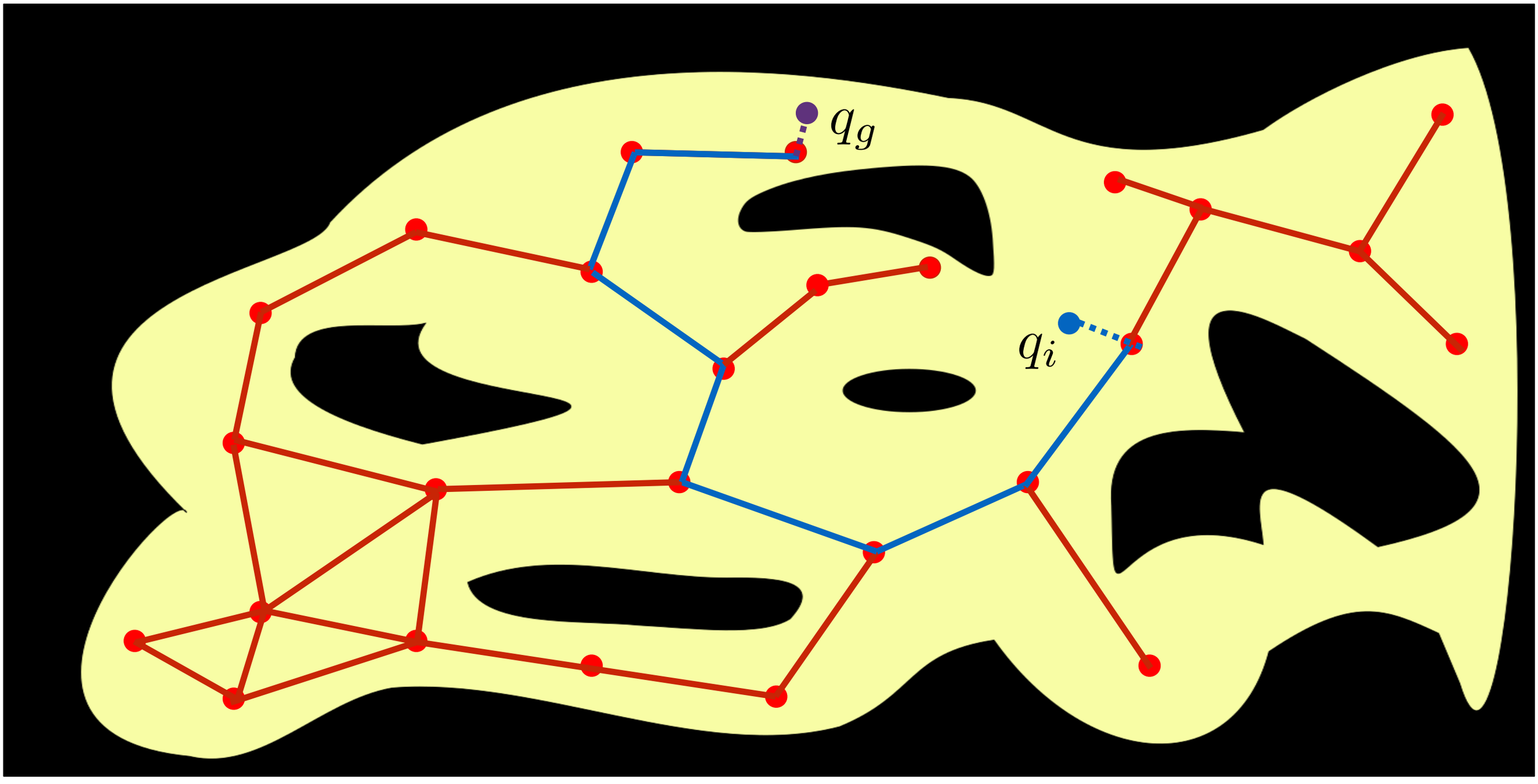
Generate Candidate Vertices and Edges

Evaluate Candidate Vertices and Edges (Collision Checking)

Add Valid Vertices and Edges

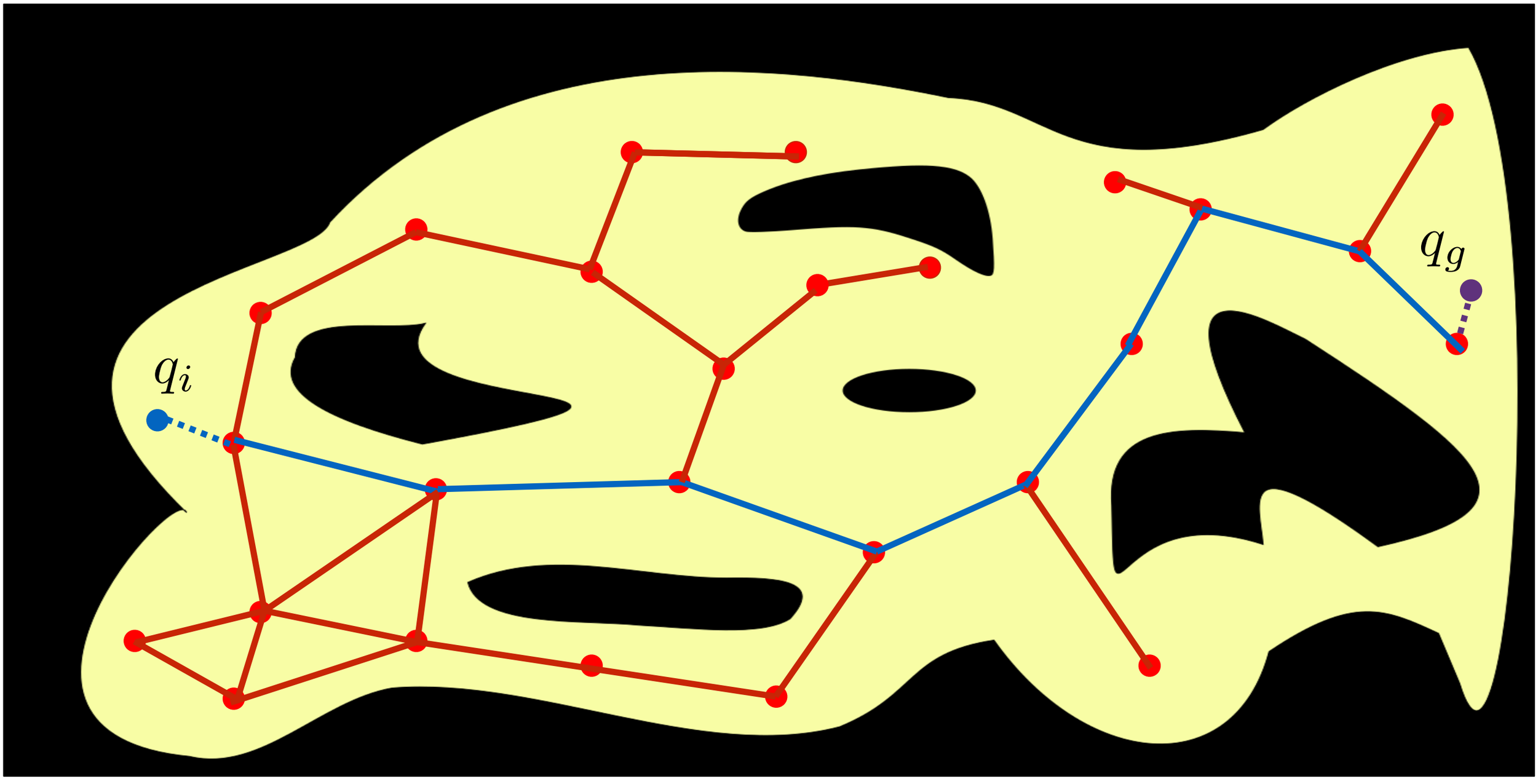
Query Phase

- Connect query configurations q_i and q_g and find a path



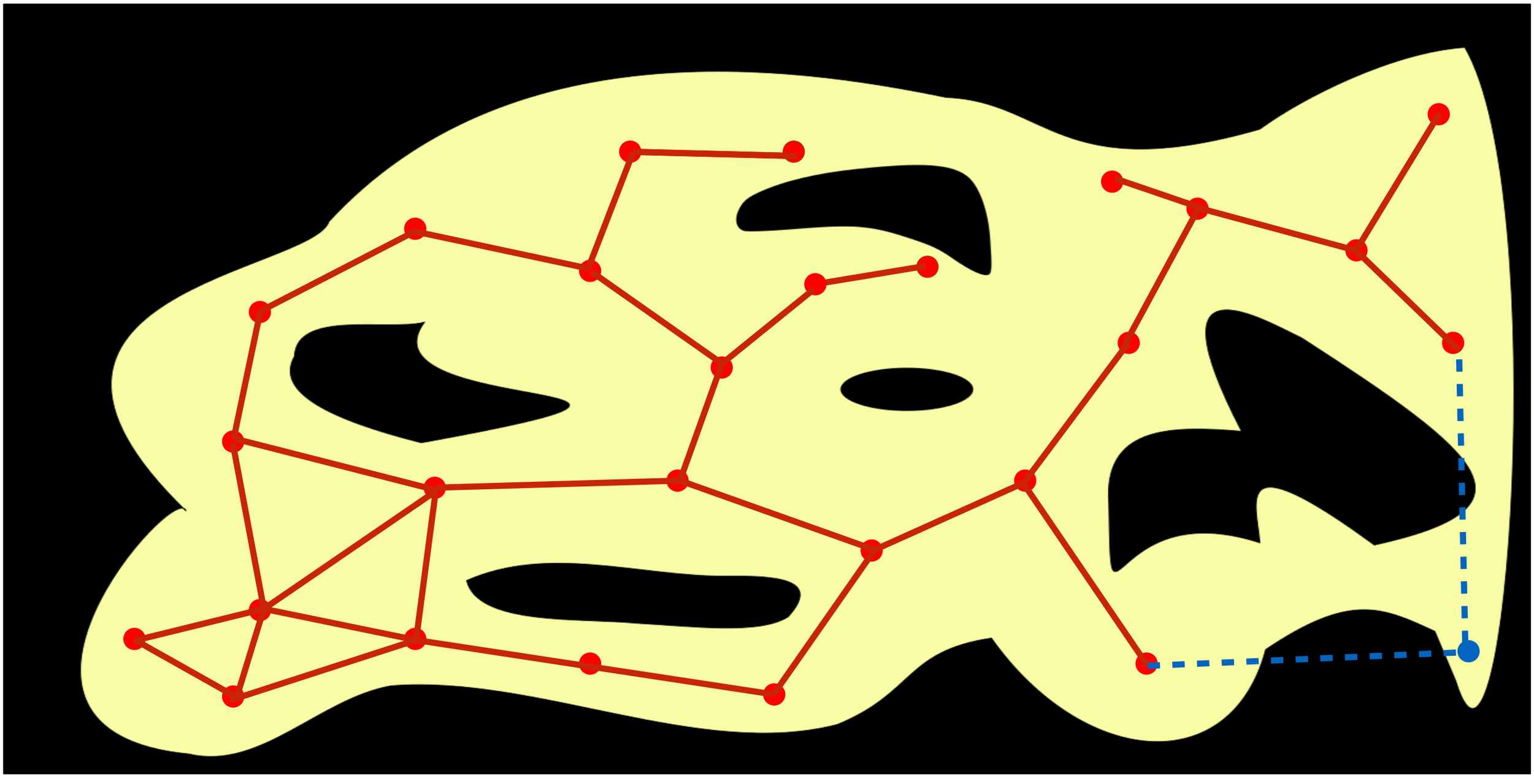
Query Phase

- Connect query configurations q_i and q_g and find a path



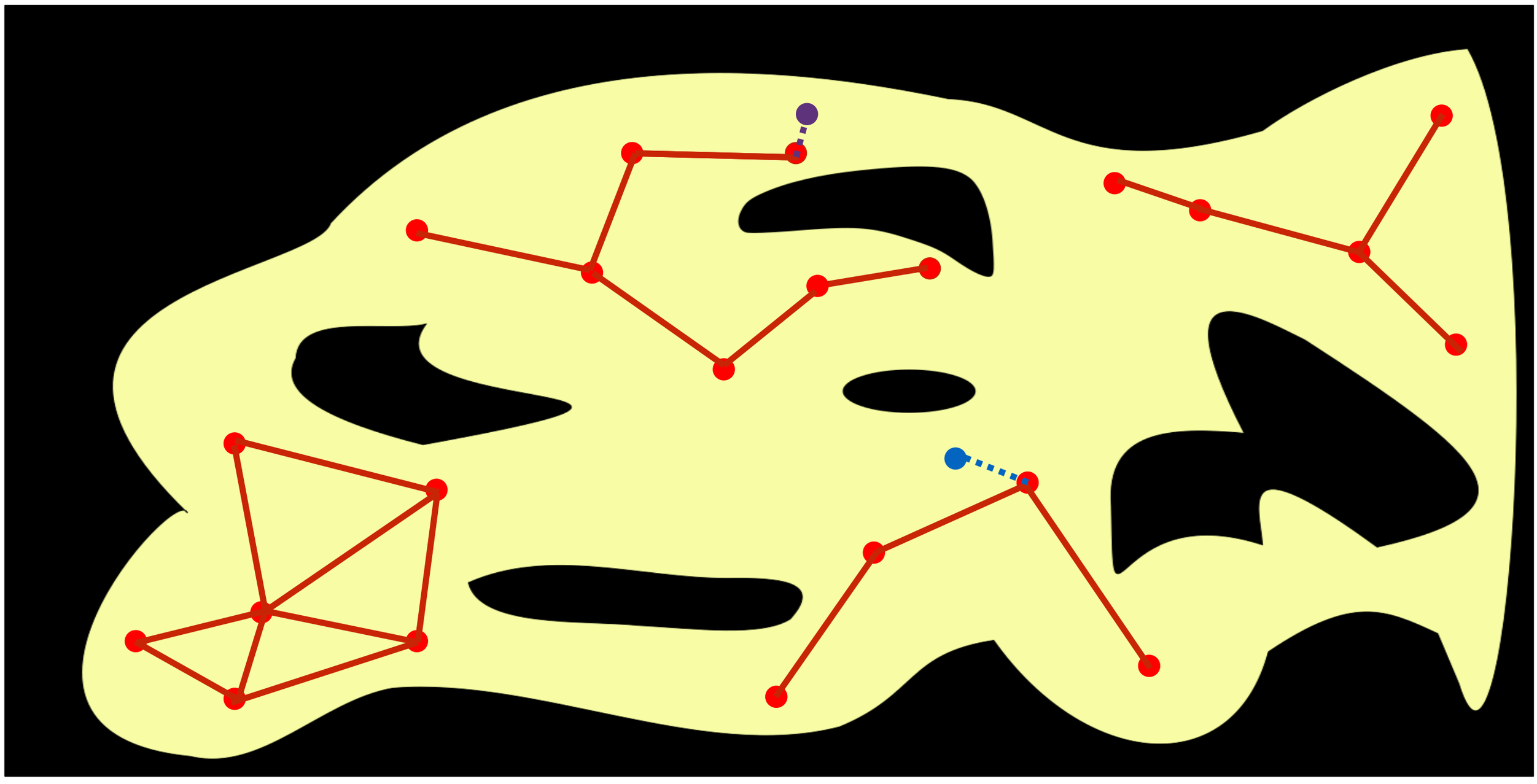
Coverage Issues

- Lack of accessibility and departability in some regions



Connectivity Issues

- Graphs may be disconnected with no path between nodes



- Need to improve connectivity in expansion step

Expansion Step

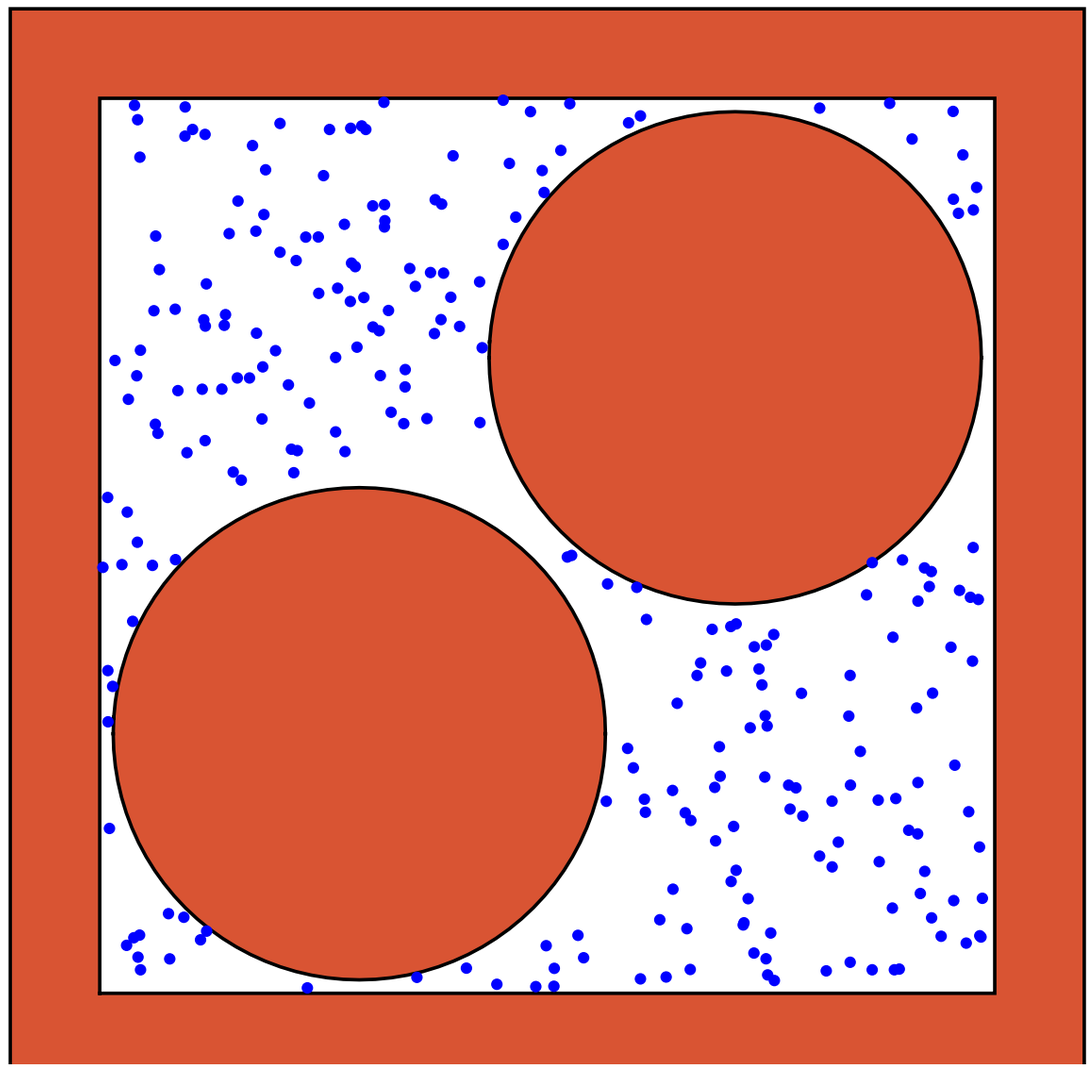
- Use heuristics to improve connectivity

- Sample q_{new} to favour unconnected/difficult regions
  - Gaussian Sampling
  - Bridge Sampling
  - Vertex Enhancement
  - Visibility Roadmaps

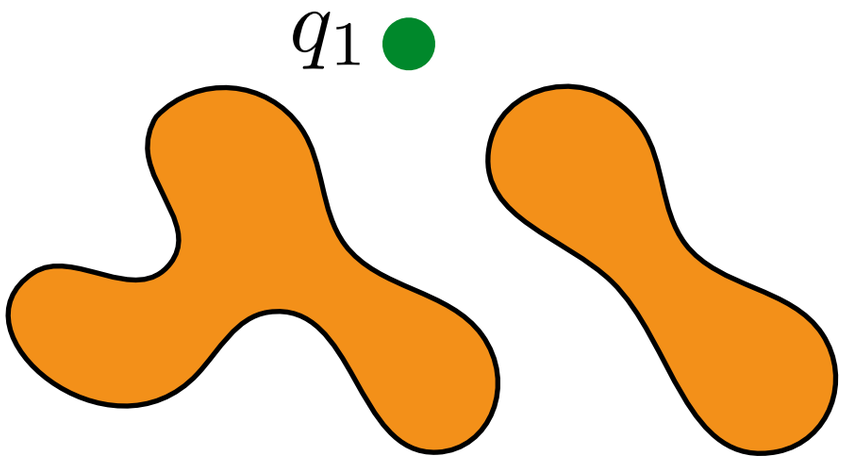
(Standard) Uniform Sampling

- Sample configurations

$$q_1 \sim \text{Uniform}$$



- Test configurations



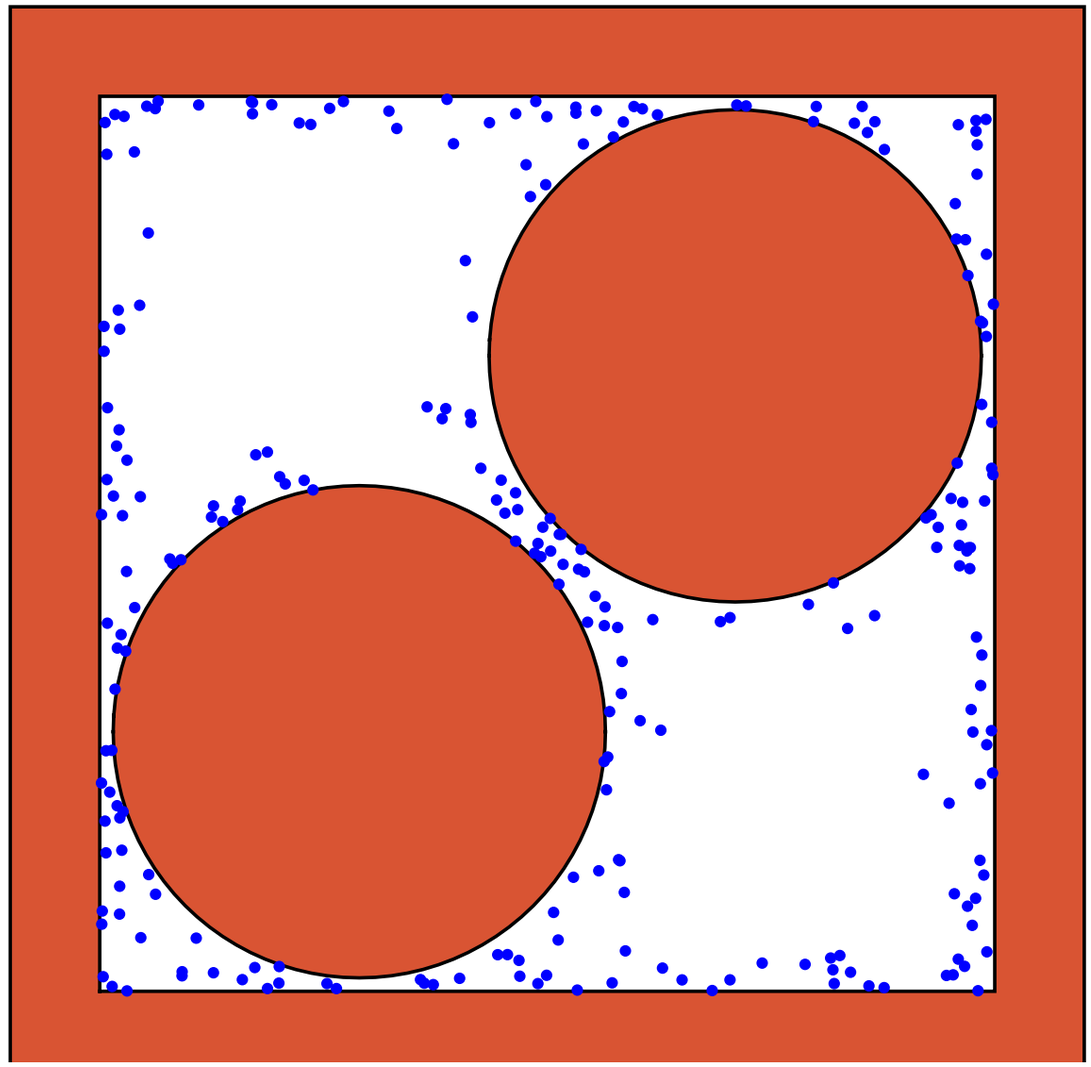
- Tends to favor: large open regions

Gaussian Sampling

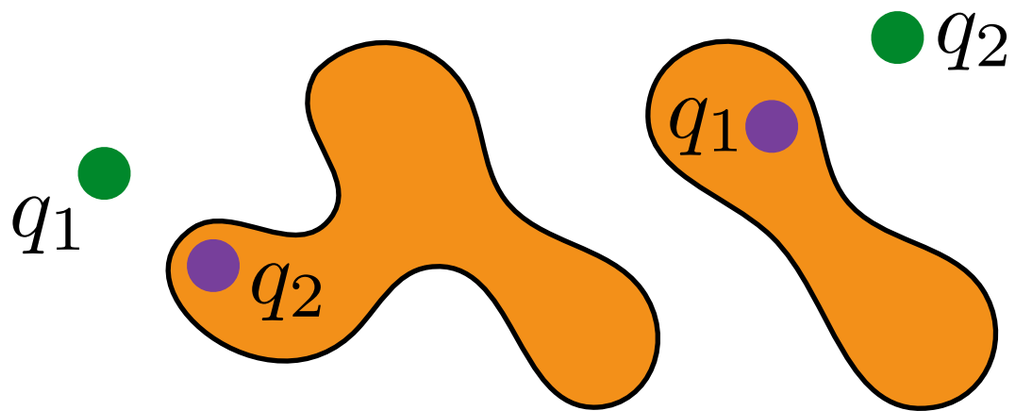
- Sample configuration pairs

$$q_1 \sim \text{Uniform}$$

$$q_2 \sim \mathcal{N}(q_1|\Sigma)$$



- Then test configurations



- Tends to favor: boundaries of obstacles

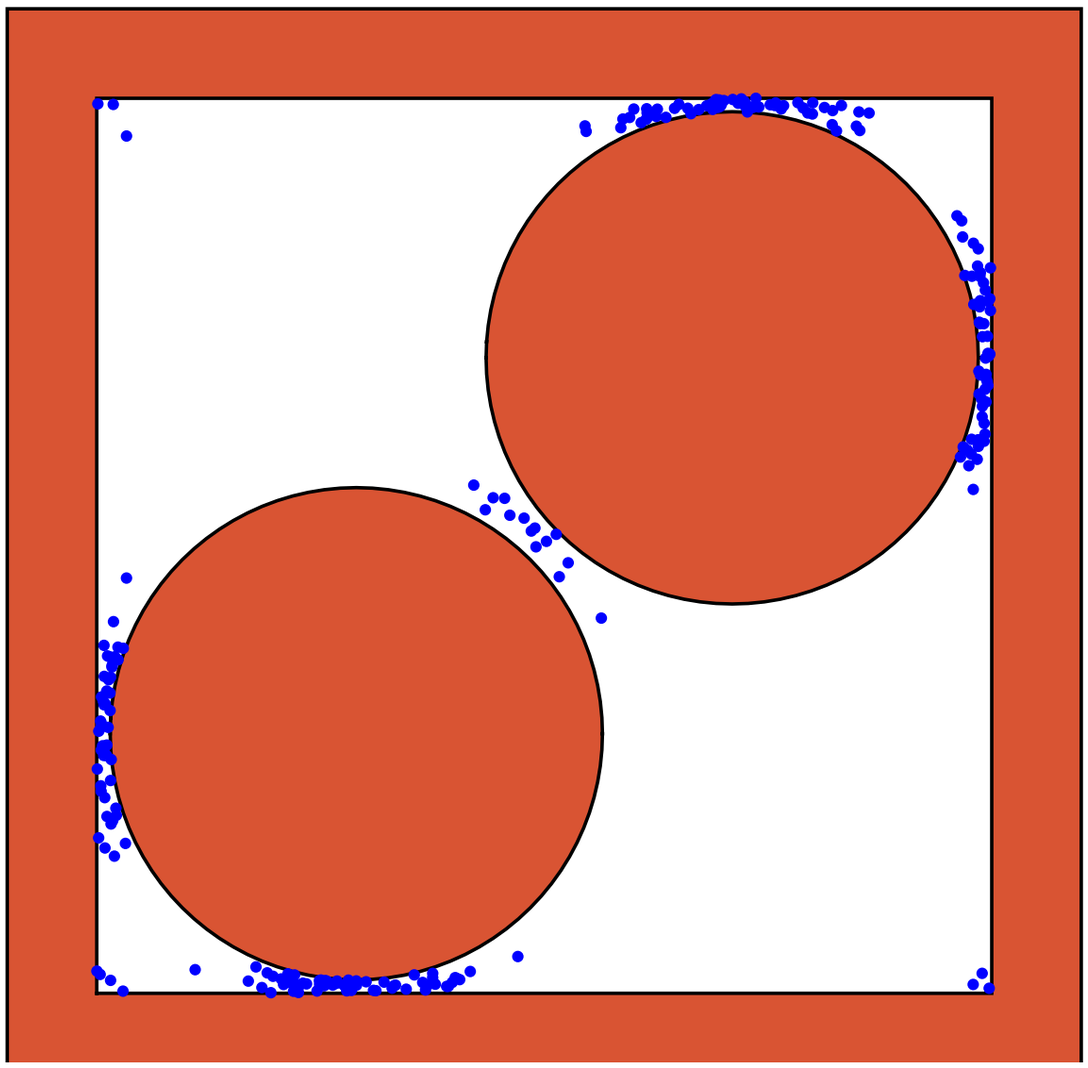
Bridge Test Sampling

- Sample configuration triplets

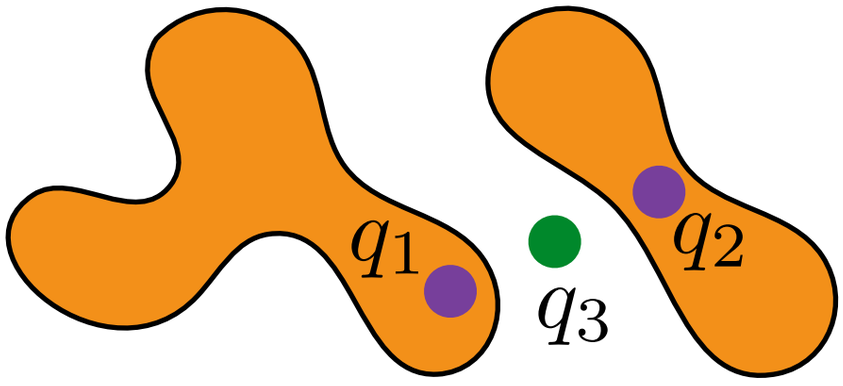
$$q_1 \sim \text{Uniform}$$

$$q_2 \sim \mathcal{N}(q_1|\Sigma)$$

$$q_3 = (q_1 + q_2)/2$$



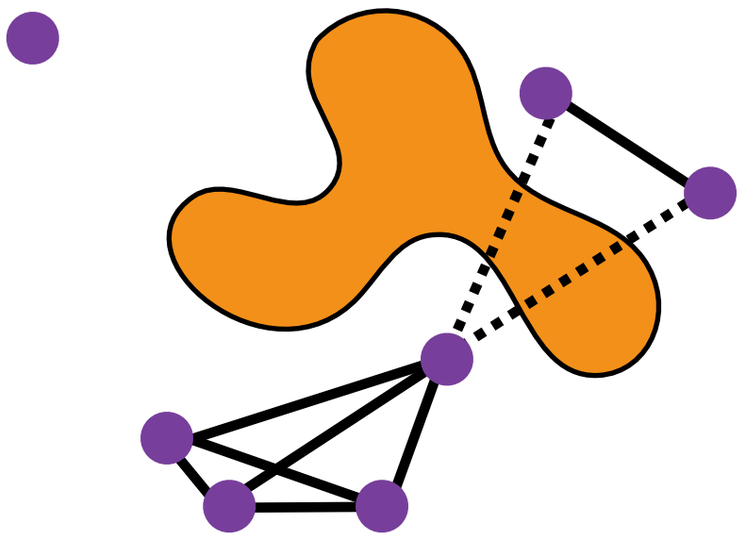
- Then test configurations



- Tends to favor: gaps between obstacles

Vertex Enhancement

- Randomly select vertices around which to explore
- Probability of selecting vertex v is given by



$$P(v) \propto \frac{n_f}{n_t + 1}$$

failed attempts to connect

total attempts to connect

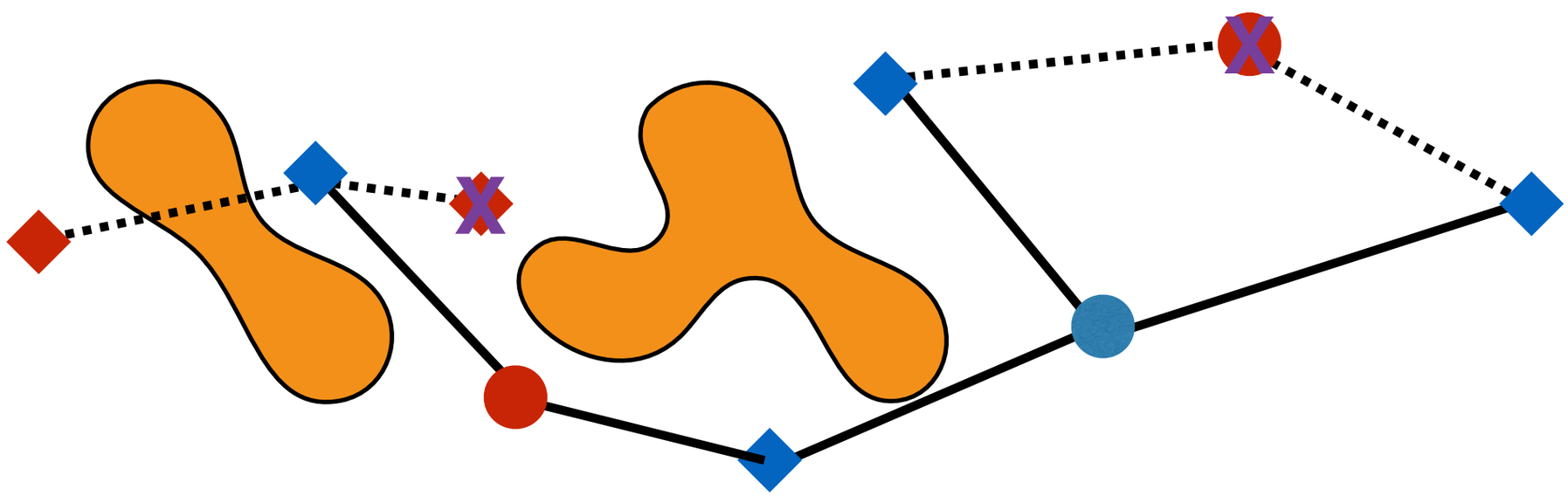
  - Prefers nodes with many failed attempts to connect
  - Avoids nodes that are completely isolated
- Perform random walk from vertex to find new candidate

Lazy PRMs

- Number of vertices results in a tradeoff:
  - Sparse PRM results in less optimal paths
  - Dense PRM requires more collision checking (expensive)
- Lazy PRMs gives density while reducing collision check
  - Create full PRM without any collision checking
  - At query time, find shortest path and check for collisions
  - Remove collision edges/vertices and repeat as necessary

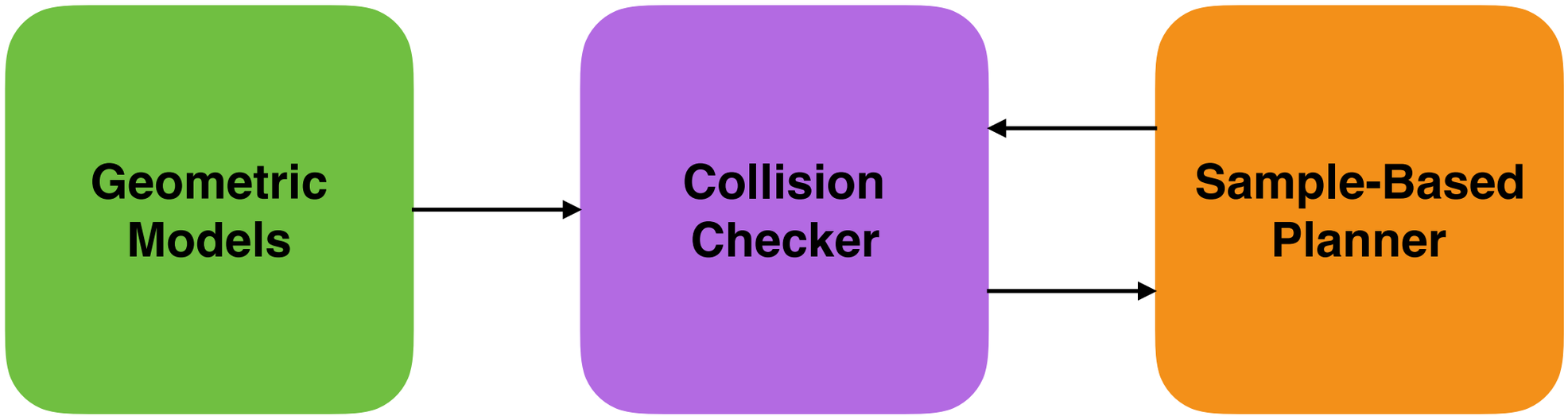
Visibility Roadmaps

- Dense roadmaps include many "wasted" nodes/edges
- Use visibility to create a spare roadmap that covers C_{free}
- Define two types of vertices:
  - Guards:
    Vertices that does not see any other vertices at creation
  - Connectors:
    Must see at least two guards from different components



Collision Checking

- Sample-based planner does not see full C_{free} and C_{obs}
- Can only probe for collisions using collision checker



- Check needs to be FAST for efficient planning

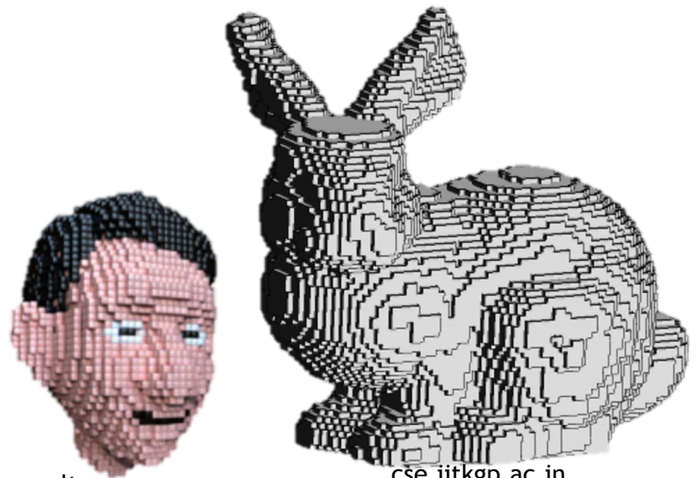
Based on slides of Katharina Muelling

3D Object Representations

- Composite of primitive shapes
  - Mostly used in simulation
  - Very fast, but overestimates geometry

From slides D. Berenson

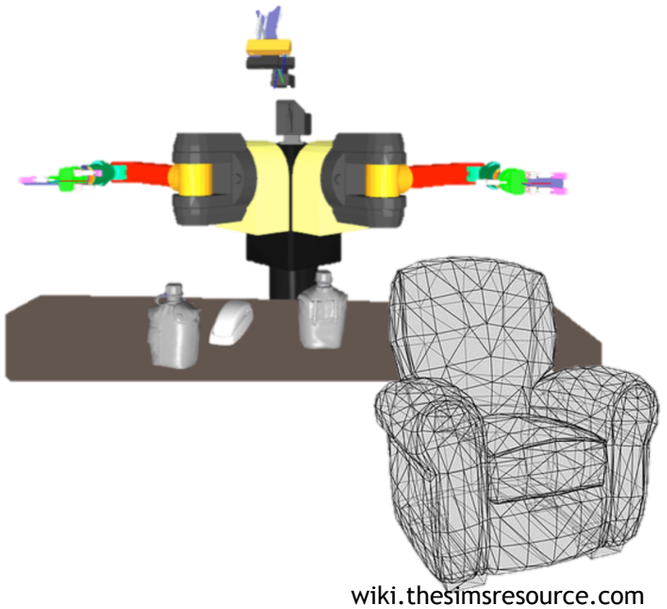
- Voxel Grids
  - Easy to check intersections
  - Difficult to rotate



maltaannon.com

cse.iitkgp.ac.in

- Triangle Meshes
  - Often used for real-world applications
  - Naive approach: check all triangle pairs



wiki.thesimsresource.com

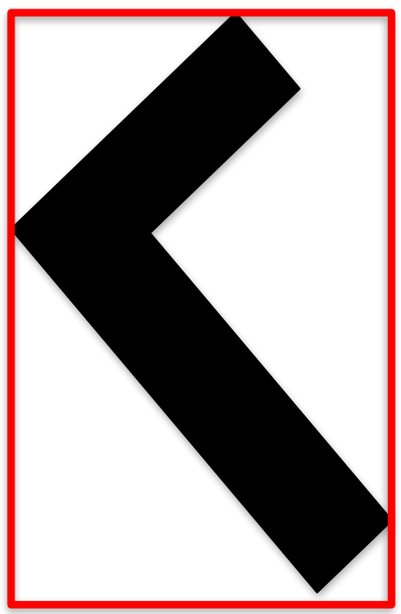
Two Phase Collision Detection

- Broad Phase:
  - Use cheap coarse check to remove many collision candidates
  - Can be performed using simple bounding boxes
  - Find all potential collision candidates

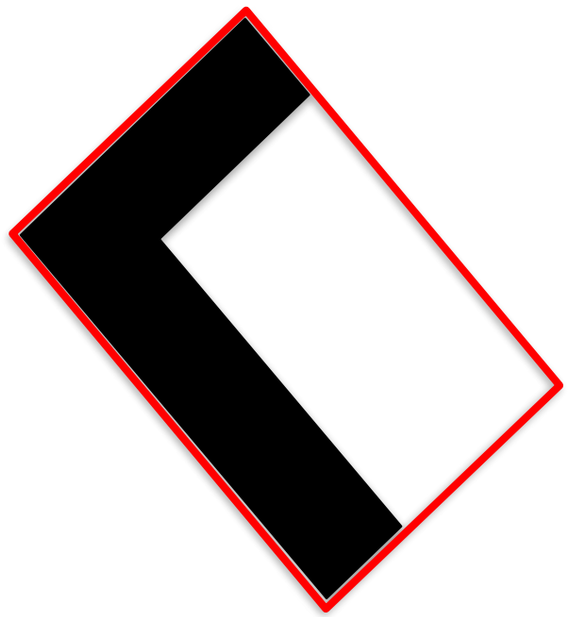
- Narrow Phase:
  - Carefully check individual pairs of bodies for collisions
  - Broad phase selects the hard problems for the narrow phase

Collision Checking Volumes

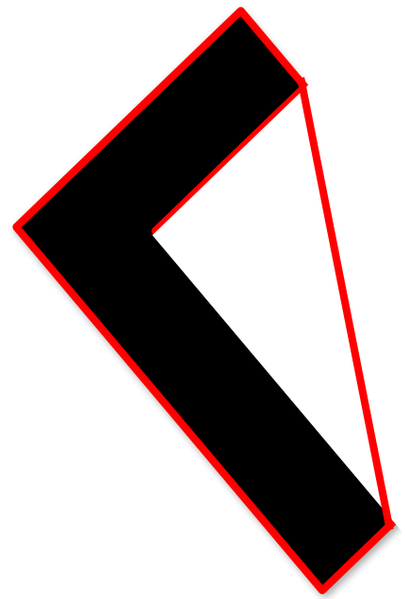
Approximate shapes using bounding volumes



Axis-Aligned Bounding Box
(AABB)



Oriented Bounding Box
(OBB)



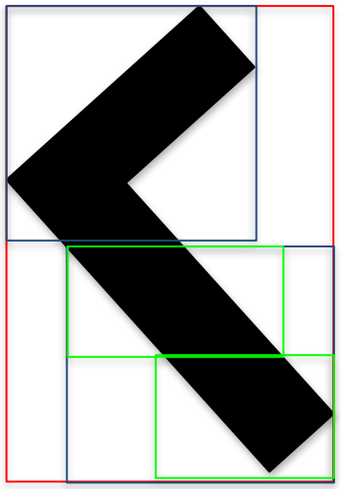
Convex Hull

Looser Approximation
Cheaper Test

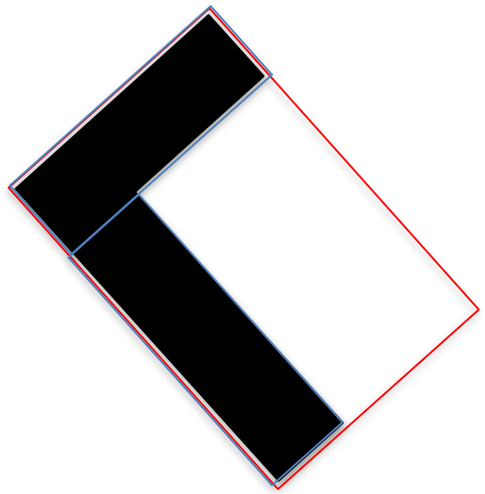
Tighter Approximation
Expensive Test

Hierarchical Collision Checking Volumes

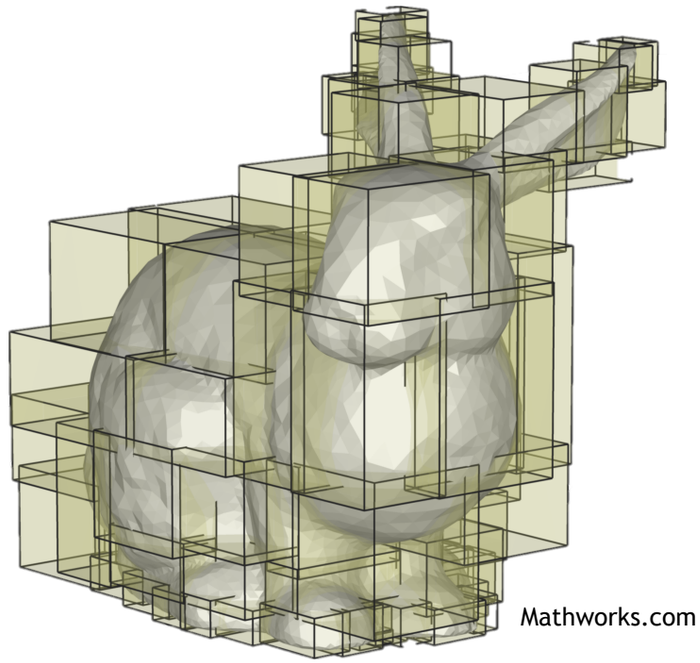
- Decompose object's bounding box into a tree structure



AABB-Tree



OBB-Tree

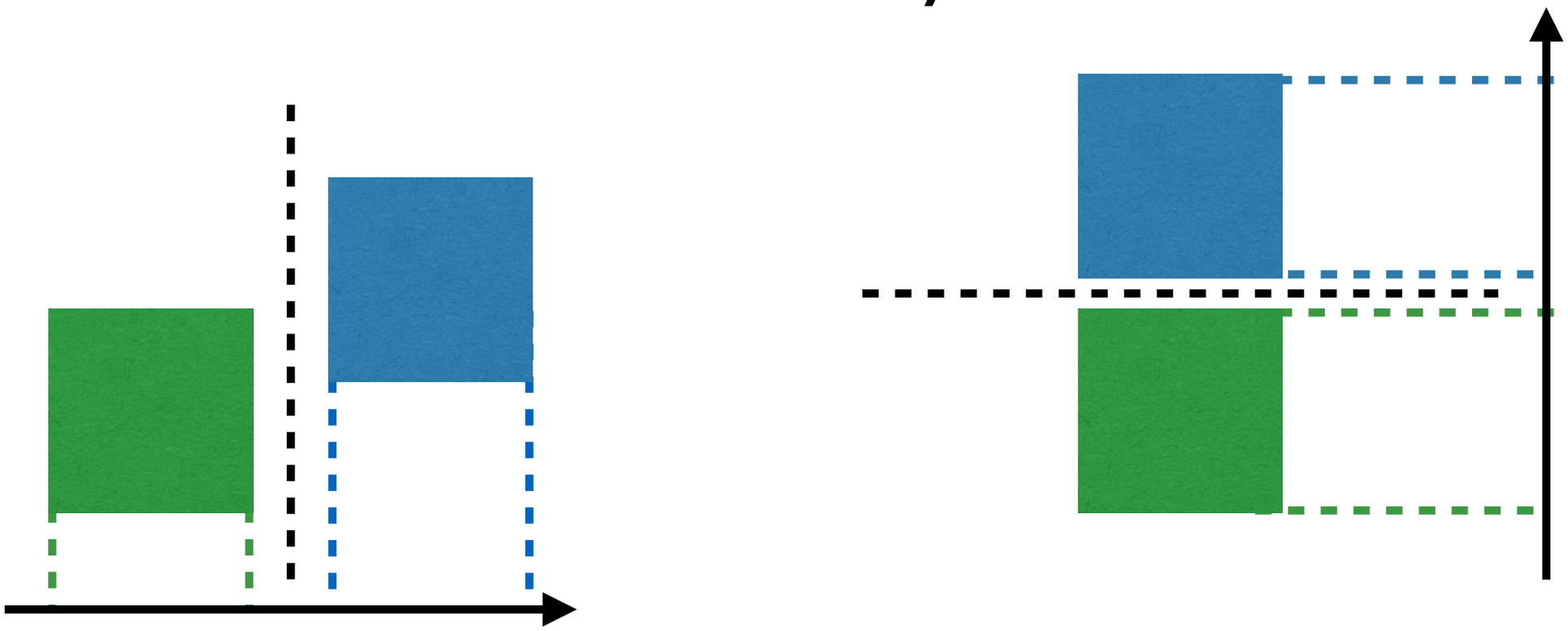


AABB-Tree

- Construction: divide regions into two of similar size but more refined to shape of the object
- Collision Check: Move on if no collision was detected, otherwise evaluate the region's sub regions

SAT Collision Checking

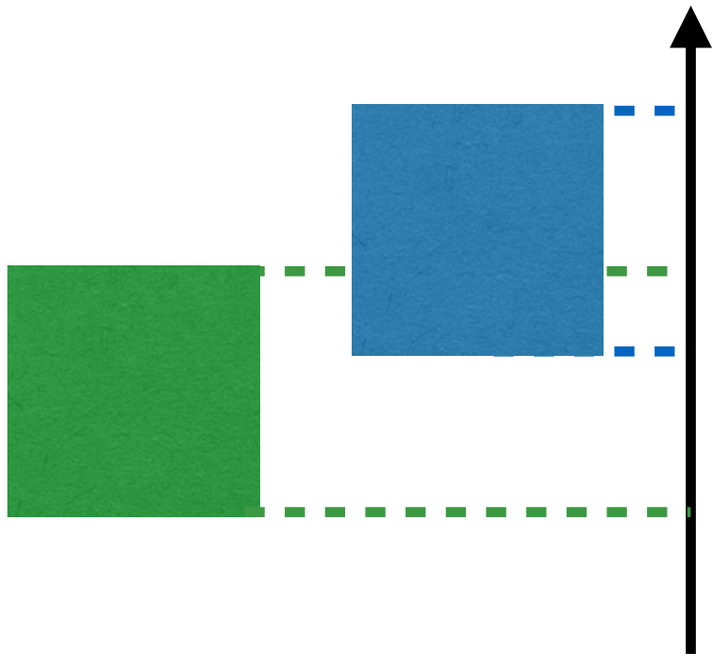
- Consider two cuboids, are they in collision?



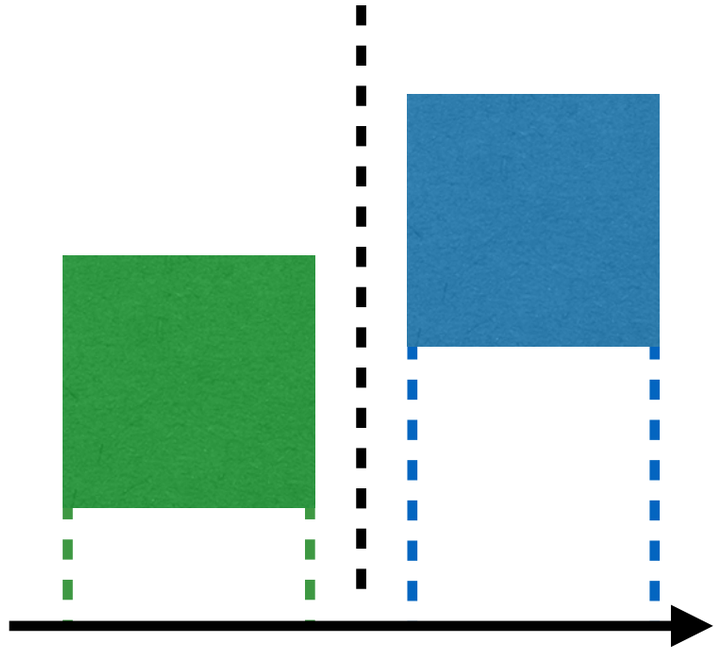
- Shapes separated by at least one axis if not in collision
  - Separating Axis Theorem (SAT)
  - Check overlap by projecting corners onto axis

SAT Collision Checking

- Need to check each axes, until one separates shapes



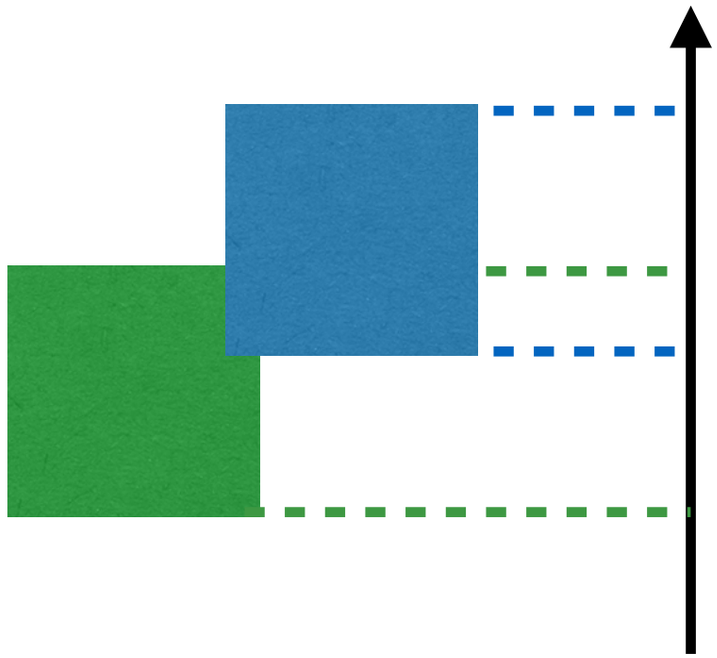
No Separation



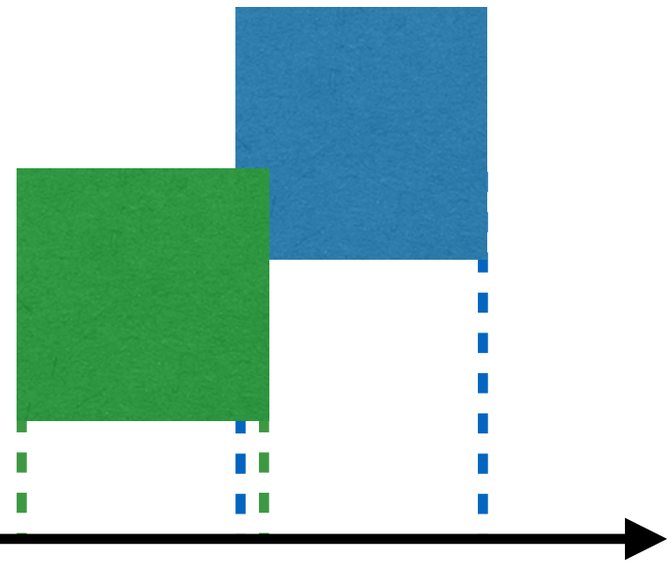
Separation
No collision!
Stop checking

SAT Collision Checking

- Need to check each axes, until one separates shapes



No Separation



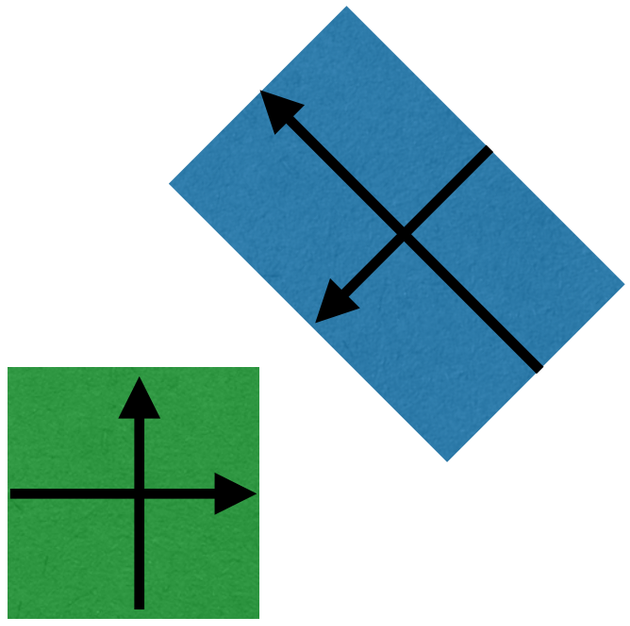
No Separation

(No Separation
in 3rd direction)

Collision!

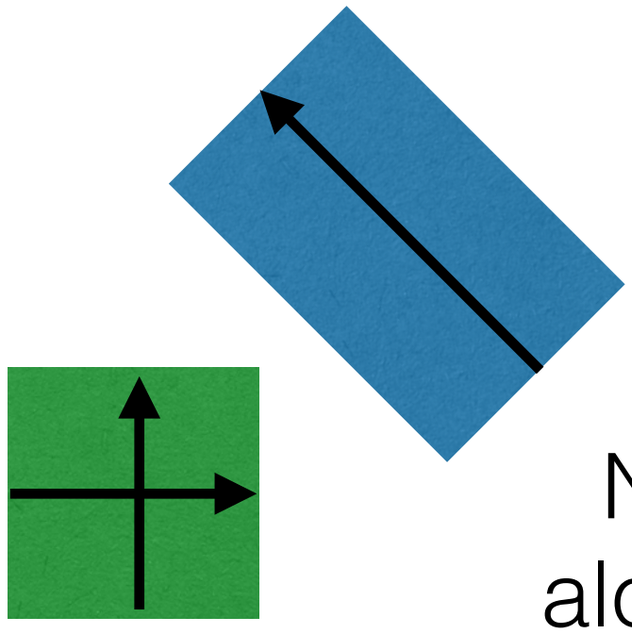
SAT Collision Checking

- Need to check axes corresponding to shapes' normals



SAT Collision Checking

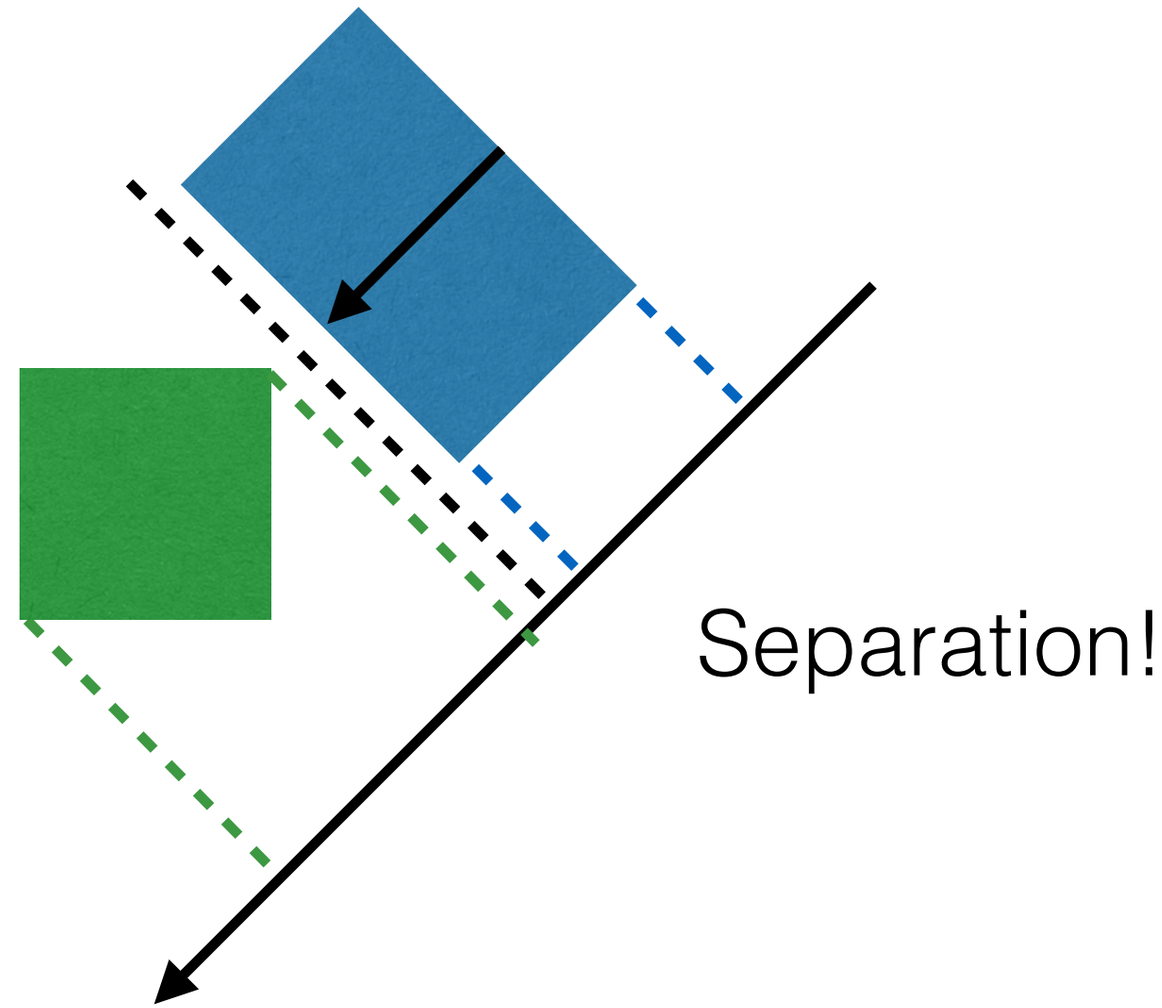
- Need to check axes corresponding to shapes' normals



No separation
along these axes

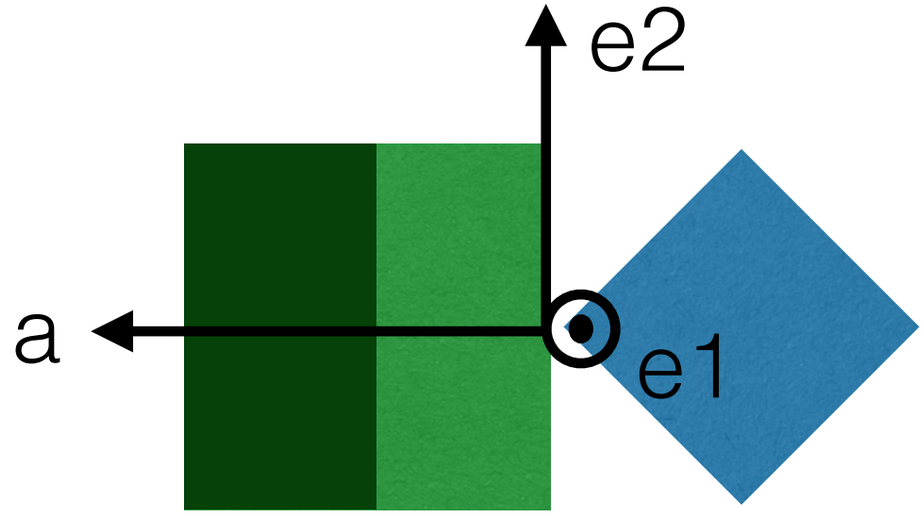
SAT Collision Checking

- Need to check axes corresponding to shapes' normals



SAT Collision Checking

- Which axes to check?
  - Surface normals (6 for cuboids)
  - Cross product between edges (9 for cuboids, can use normals)
    Needed for edge-edge conditions

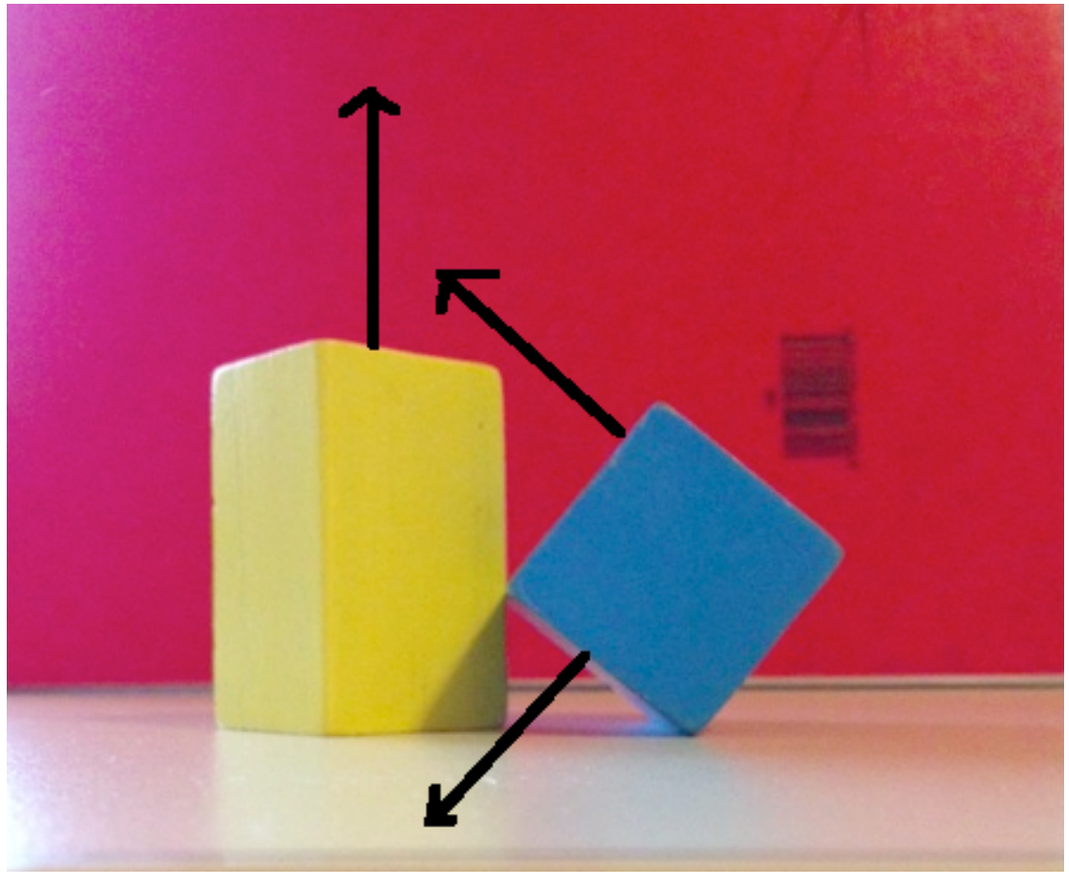


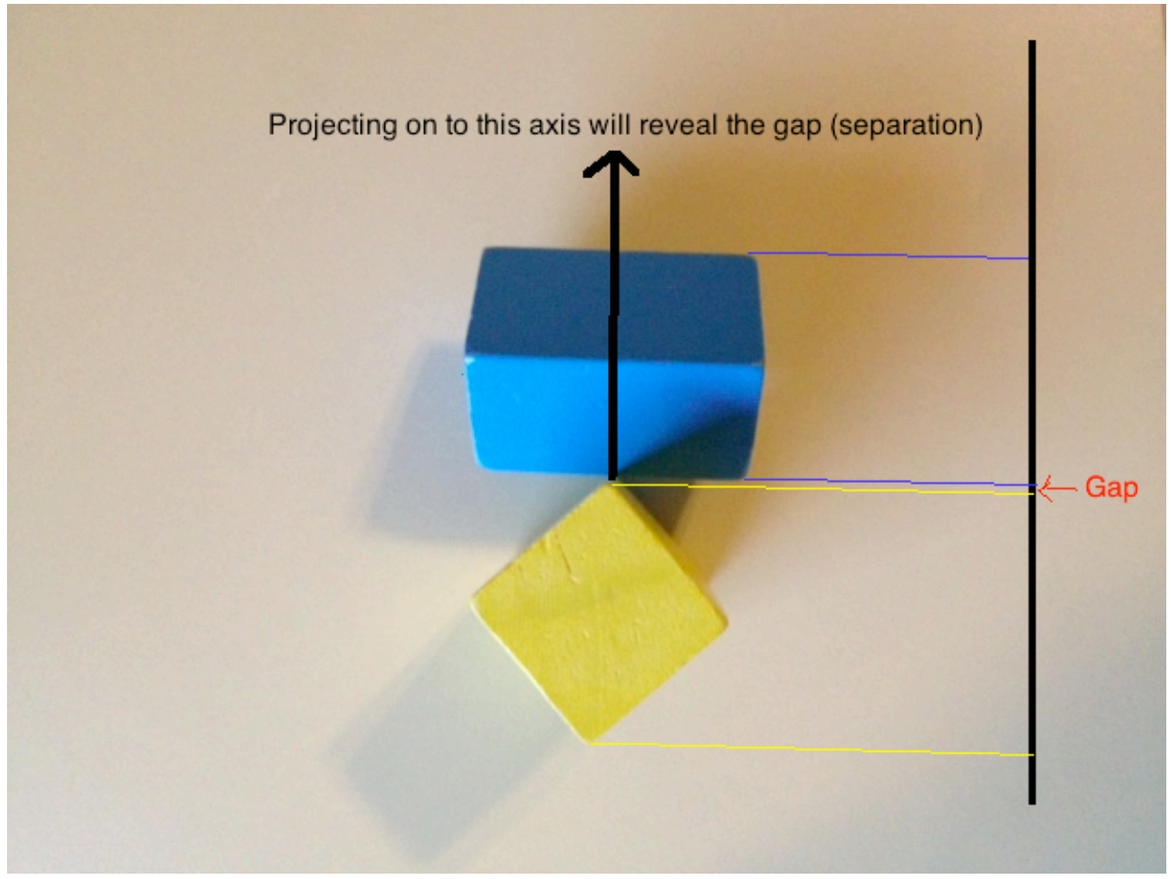
a=e1xe2

Separation! No need to continue

SAT Collision Checking

- Which axes to check?
  - Surface normals (6 for cuboids)
  - Cross product between edges (9 for cuboids, can use normals)
    Needed for edge-edge conditions





Thanks to Ken on stack exchange

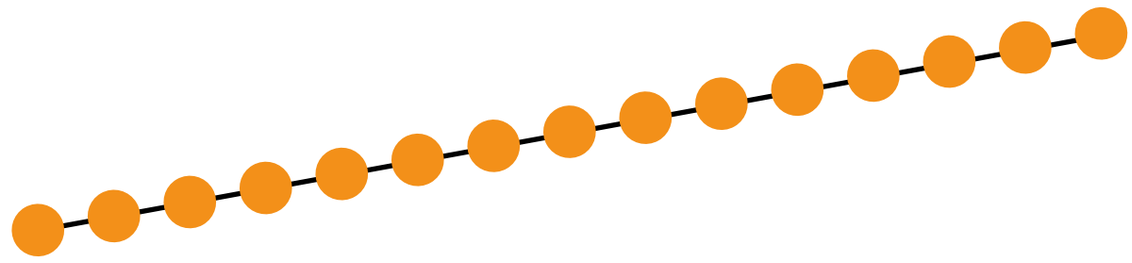
SAT Collision Checking

- SAT collision checking works for convex shapes

- Would need to decompose shape/mesh first

Collisions Along Edges

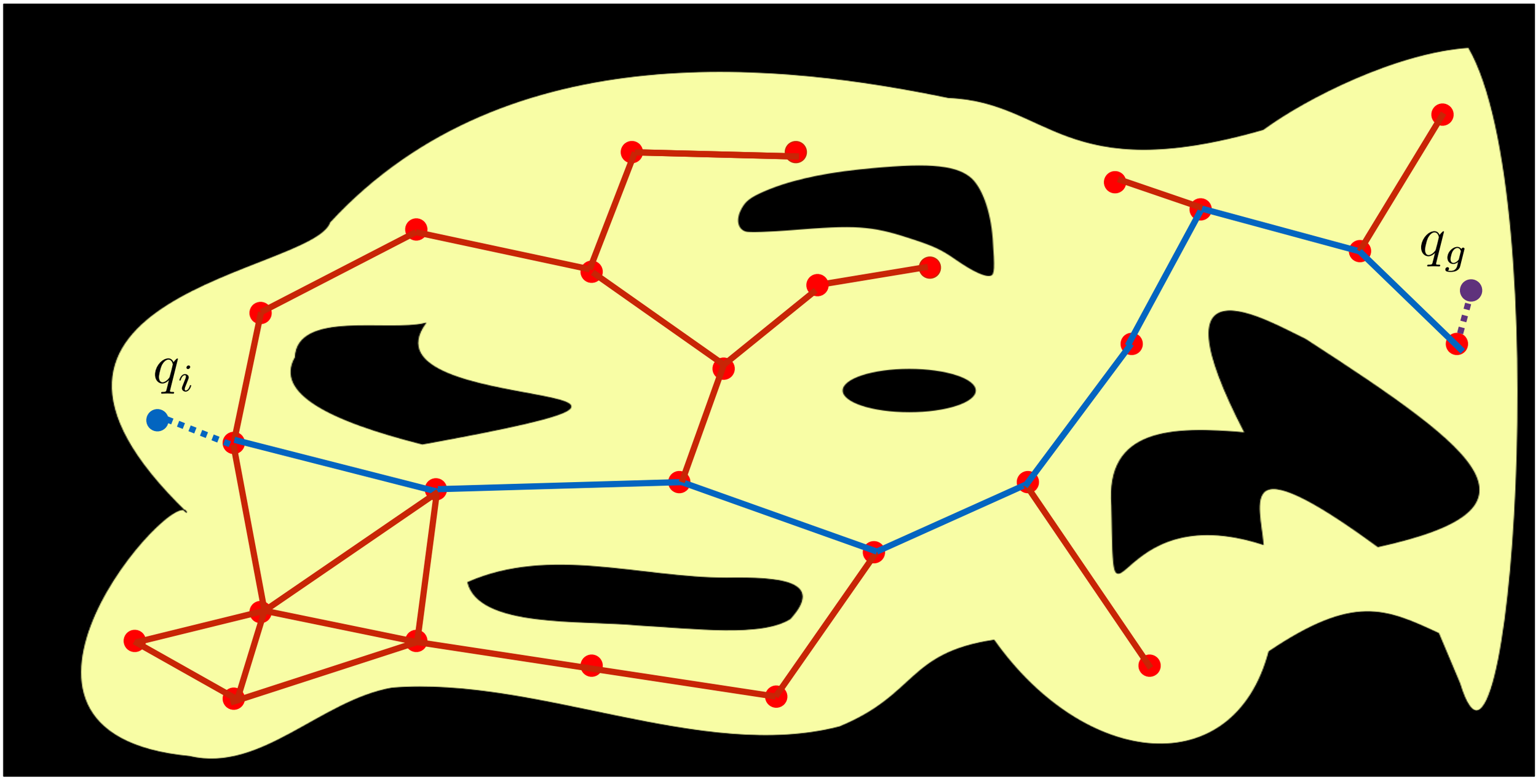
- Need to check for collisions along trajectories
- In practice, approximated by sampling points along path



  - Denser sampling is less likely to jump over collisions
  - Sparser sampling is faster to evaluate
- In theory, one can also achieve guaranteed performance by considering max possible point movement in each step
- In practice, use a library: FCL, Bullet, I-Collide, Rapid, etc.

Collision Free Paths

- Applicable to high dimensions and multiple queries



- Probabilistic Complete

Questions?