Book Ch 6.2 and 6.3.2-6.3.3

ED5215
Roadmaps

Nirav Patel

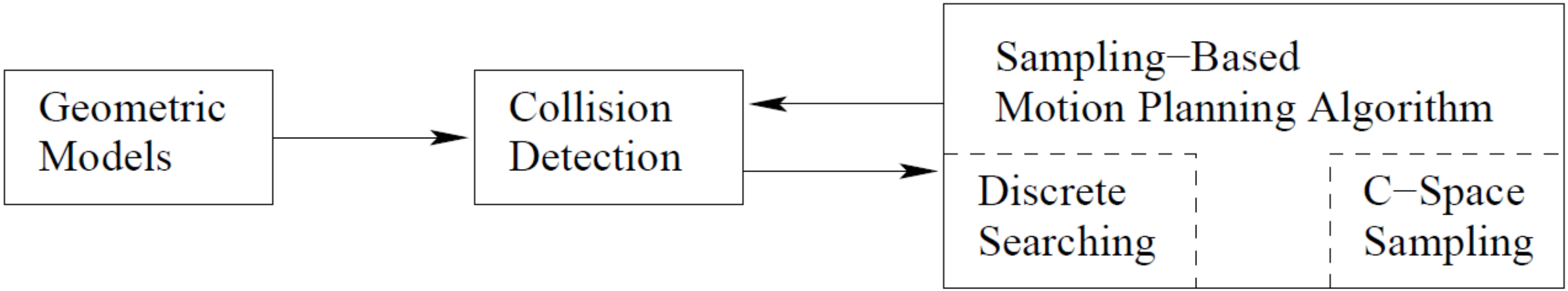
Assistant Professor, Department of Engineering Design

IIT Madras

Moodle page available at: https://courses.iitm.ac.in/

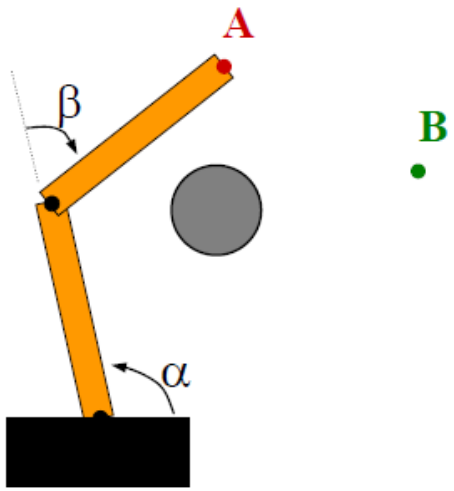
Sampling-Based Motion planning

- **Environment**
  - **Obstacles**
  - **Free space**
- **Collision checking**



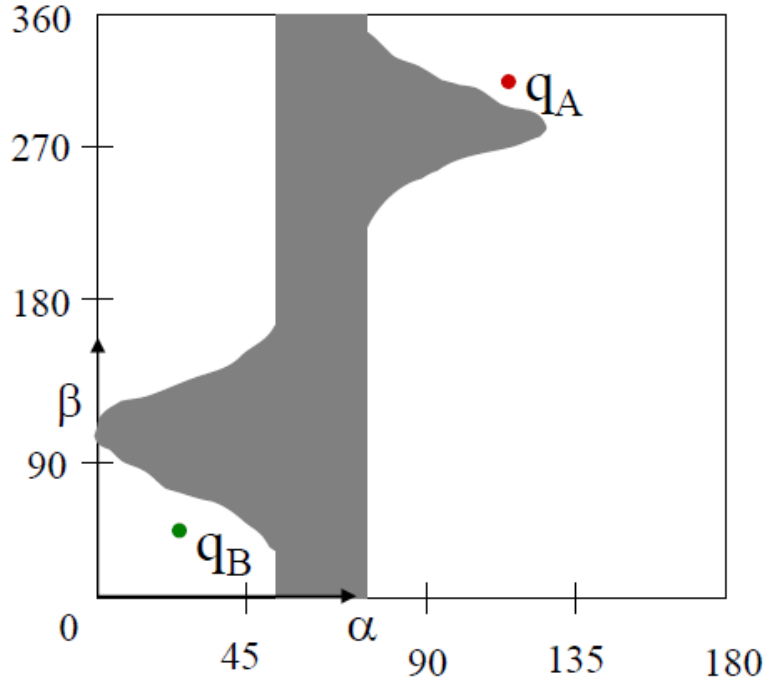
C-Space with obstacle

Reference *configuration*



An obstacle in the robot's workspace

How do we get from A to B ?

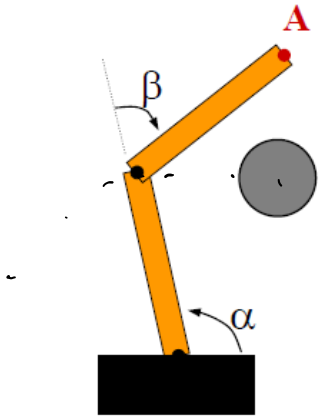


The C-space representation of this obstacle…

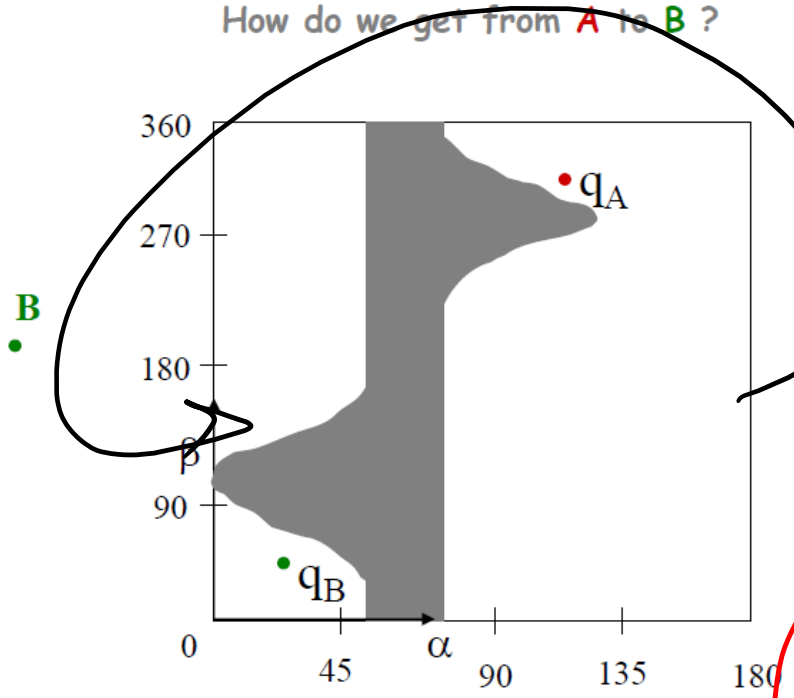
16-735, Howie Choset with slides from G.D. Hager, Z. Dodds, and Dinesh Mocha

C-Space with obstacle

Reference *configuration*

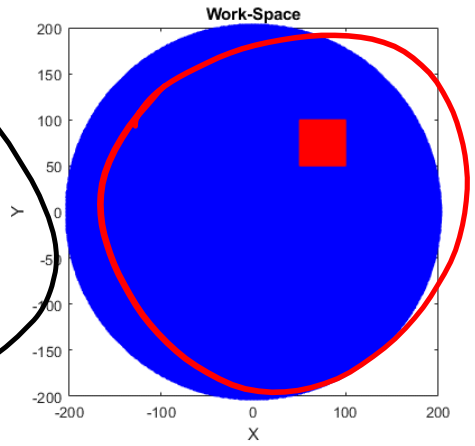


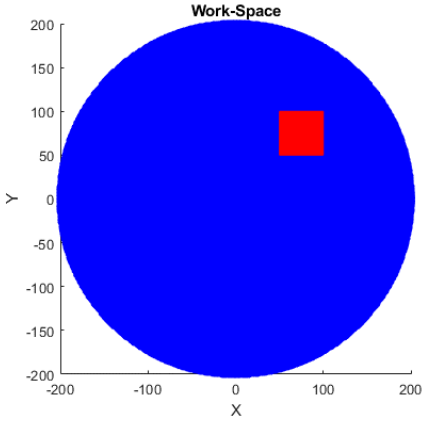
How do we get from A to B ?

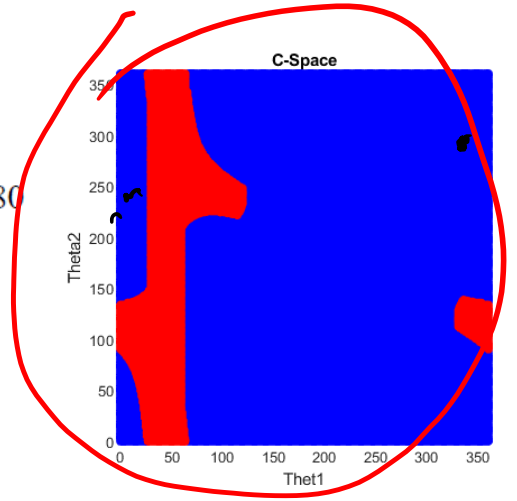


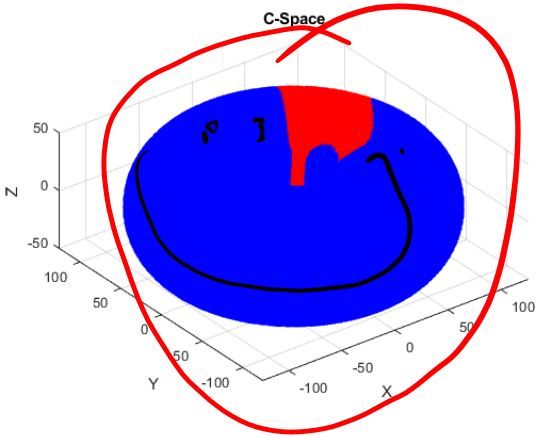
An obstacle in the robot's workspace

The C-space representation of this obstacle…









16-735, Howie Choset with slides from G.D. Hager, Z. Dodds, and Dinesh Mocha

A special case: perhaps a simpler one

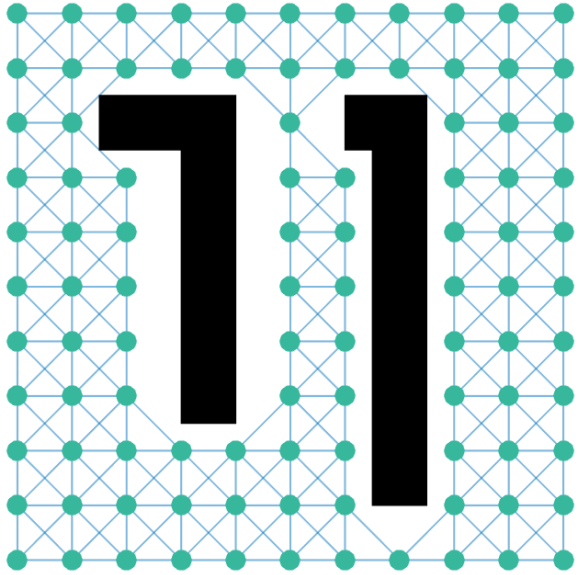
- Mobile robots
  - They are usually small
  - have simple geometric representation (polygon/point)
  - Explore/run in buildings that can be represented using geometric shapes (polygons)

Graph based algorithms (A*, D*, DFS etc)

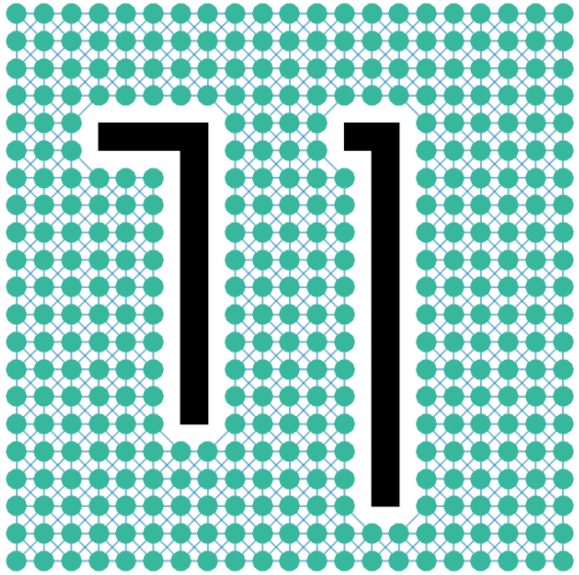
- How to compute the minimum cost path
  - Prerequisite: graph should already be available
- How do we get the graph?
  - Discretize the environment
  - Grid size??
  - Grid resolution??
  - Grid dimensions?
- What if environment has just a few obstacles or no obstacles?
  - We still need to search through the graph!!! (not the best idea!!)

Examples

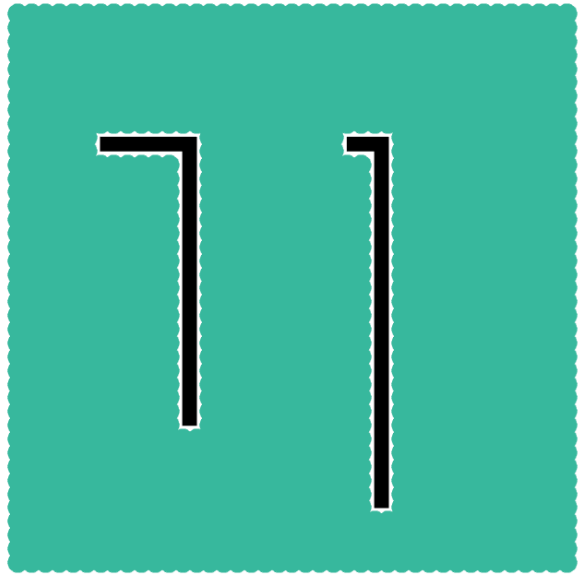
99 vertices with gridsize of 2



379 vertices with gridsize of 1



1565 vertices with gridsize of 0.5

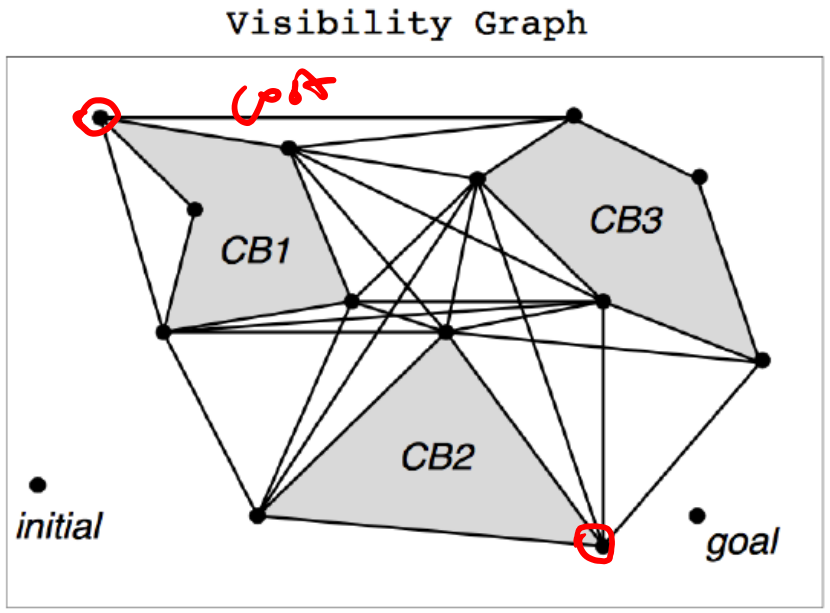


- Same environment
  - **Different scale** of problem/solution
  - What if I want my robot to be able to localize my robot with an **accuracy of 0.1 or go as close as 0.1 to an obstacle**?? (no units, assume all in m)

Image source: Pratap Tokekar, ECE 4984/5984: (Advanced) Robot Motion Planning, Virginia tech

Is there a better way: Visibility roadmaps

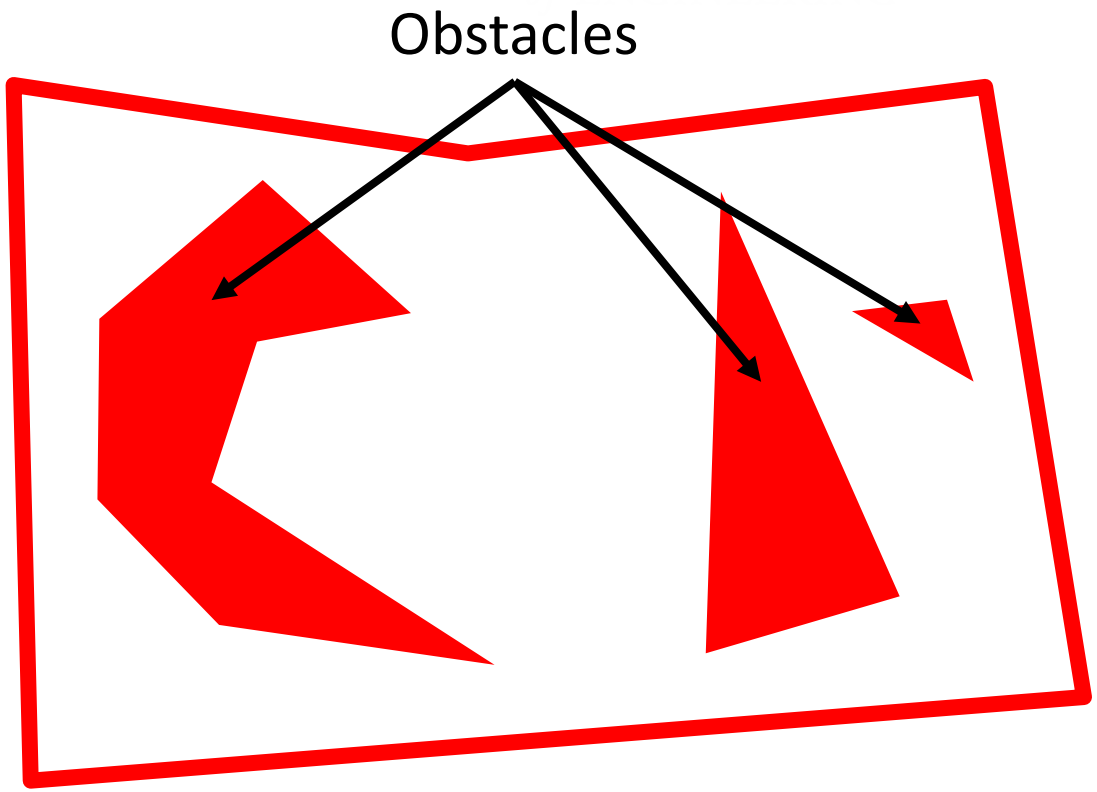
- 2D environment
  - Assume **point robot**
  - **Polygonal** obstacles
- We build a graph using
  - Vertices
    - All the **vertices** of the obstacles (remember they are **polygonal**)
    - **Start** and **goal** as **vertices**
  - Edges
    - Edge if two vertices can **see each other**, or in different words, if there is a line connecting them and not passing through an obstacle
- What do we do once we have this graph?
  - Use any graph search algorithm!!!



https://parasol.tamu.edu/~amato

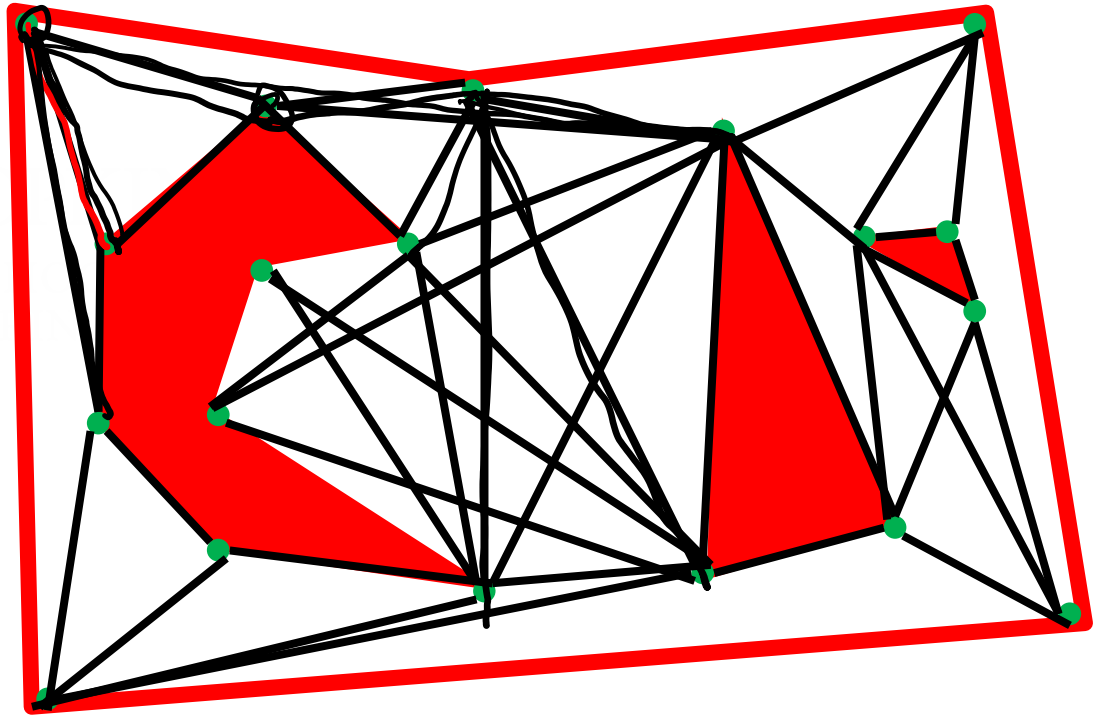
Example: lets build a visibility map!

- Assume the following environment
- Polygonal obstacles
- Point robot (of course)



Example : connect all the vertices (Visibility roadmap)

- Polygonal obstacles
- Vertices
  - All the **vertices** of the obstacles (remember they are **polygonal**)
  - **Start** and **goal** as **vertices**
- Edges
  - Edge if two vertices can **see each other**, or in different words, if there is a line connecting them and not passing through an obstacle



visibility map

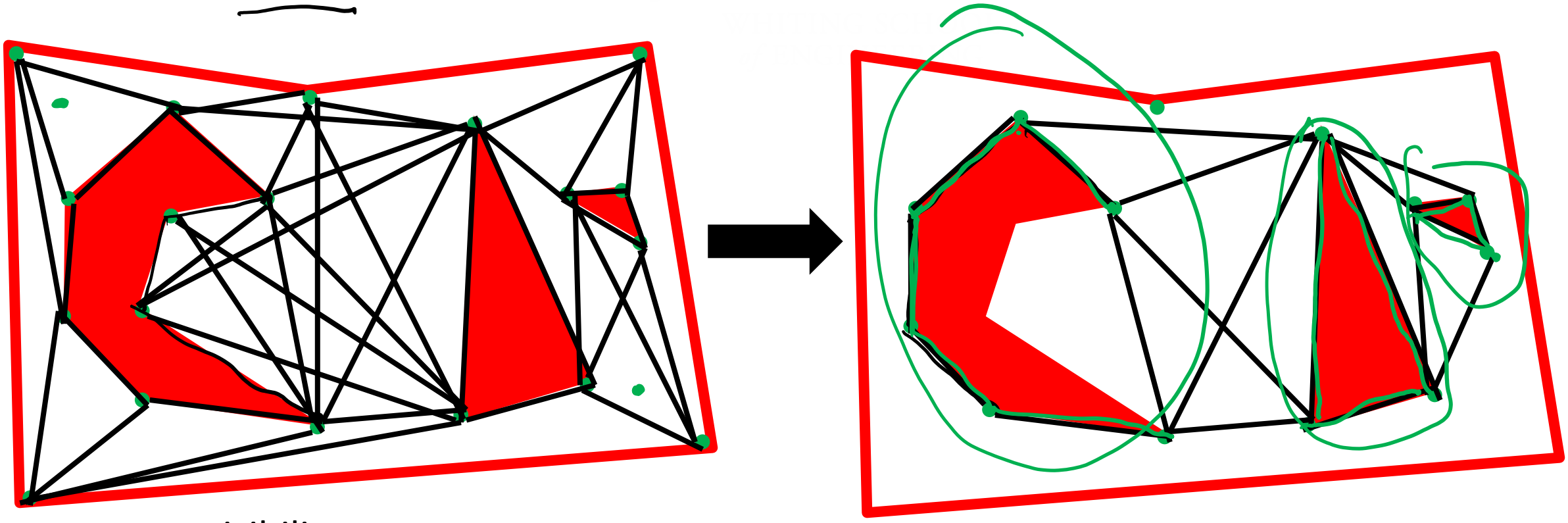
Example : connect all the vertices (Visibility roadmap)

- Polygonal obstacles
- Vertices
  - All the **vertices** of the obstacles (remember they are **polygonal**)
  - **Start** and **goal** as **vertices**
- Edges
  - Edge if two vertices can **see each other**, or in different words, if there is a line connecting them and not passing through an obstacle

Do we need ALL the edges??

Is there a better way: Visibility roadmap

- Do we need all the edges??
- Keep edges only between
  - Two reflex vertices on obstacles
  - All bitangents



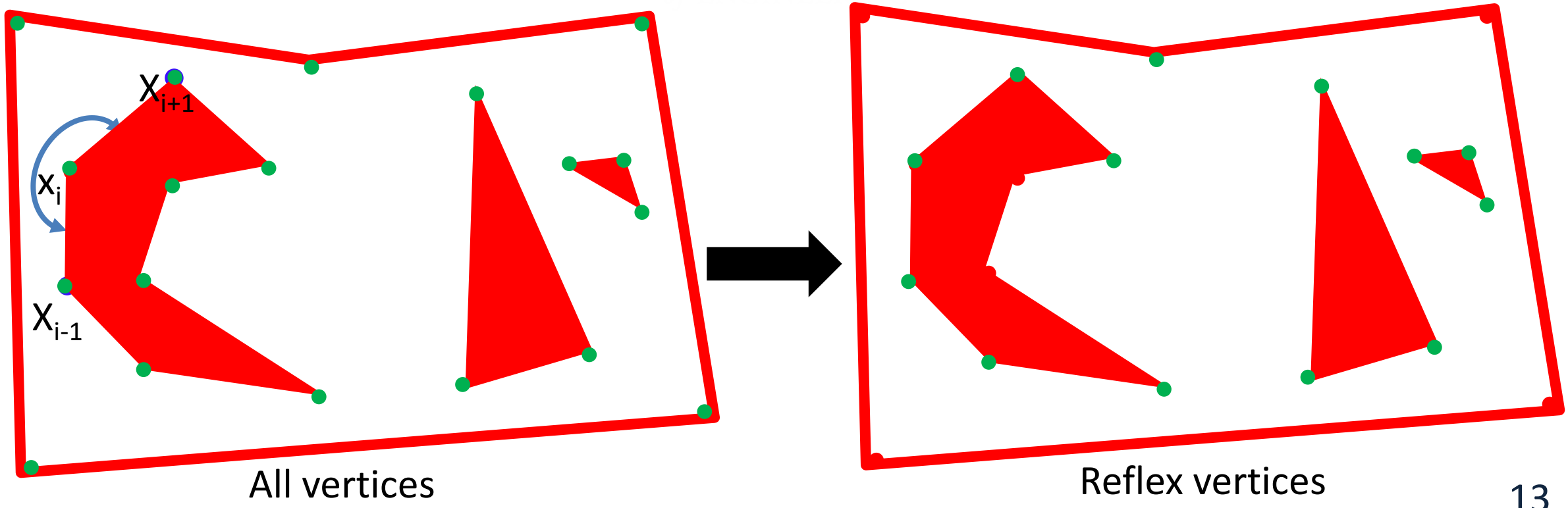
visibility map

Reduced visibility map

https://parasol.tamu.edu/~amato

Reflex vertices

- Reflex vertex
  - Angle between vertex $\angle x_{i-1}\, x_i\, x_{i+1} > \pi$
  - Angle in the free space
- Repeat this for each vertex

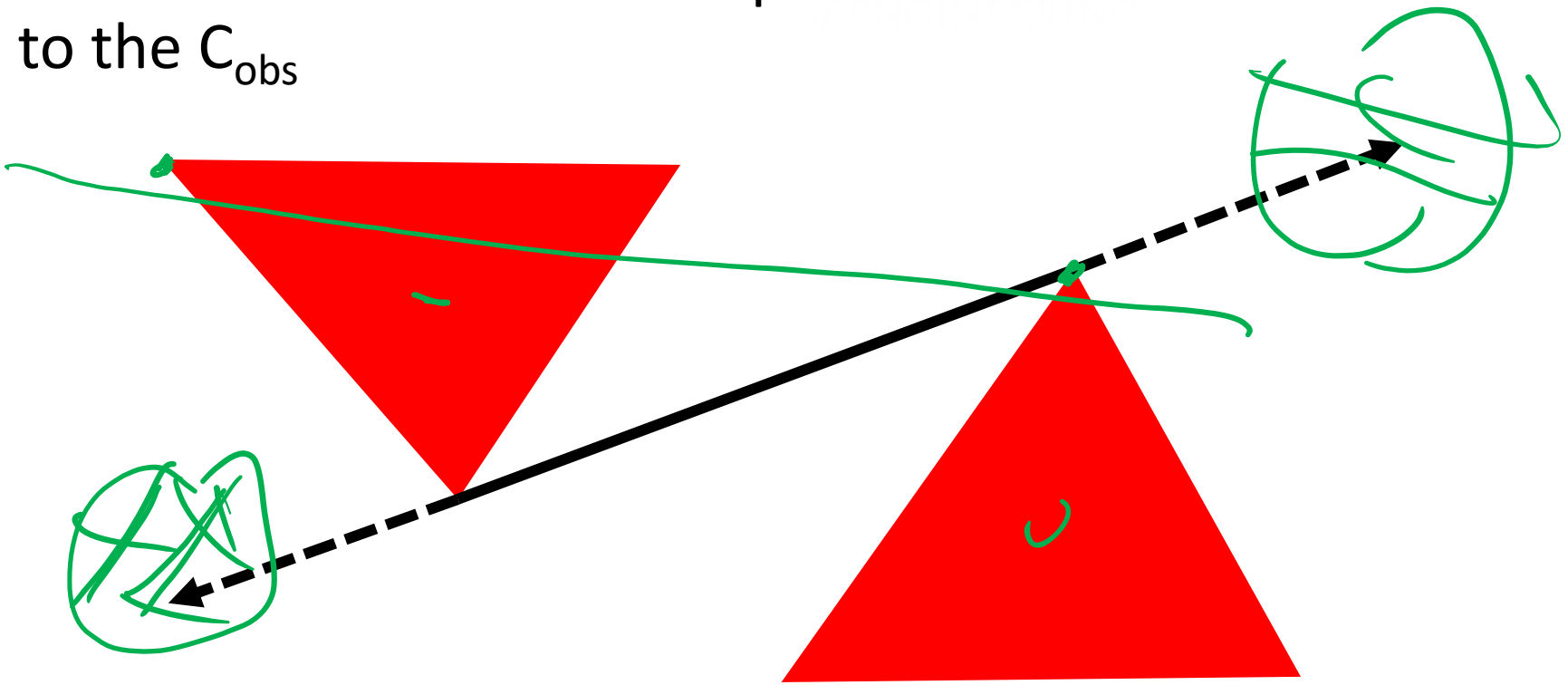


All vertices

Reflex vertices

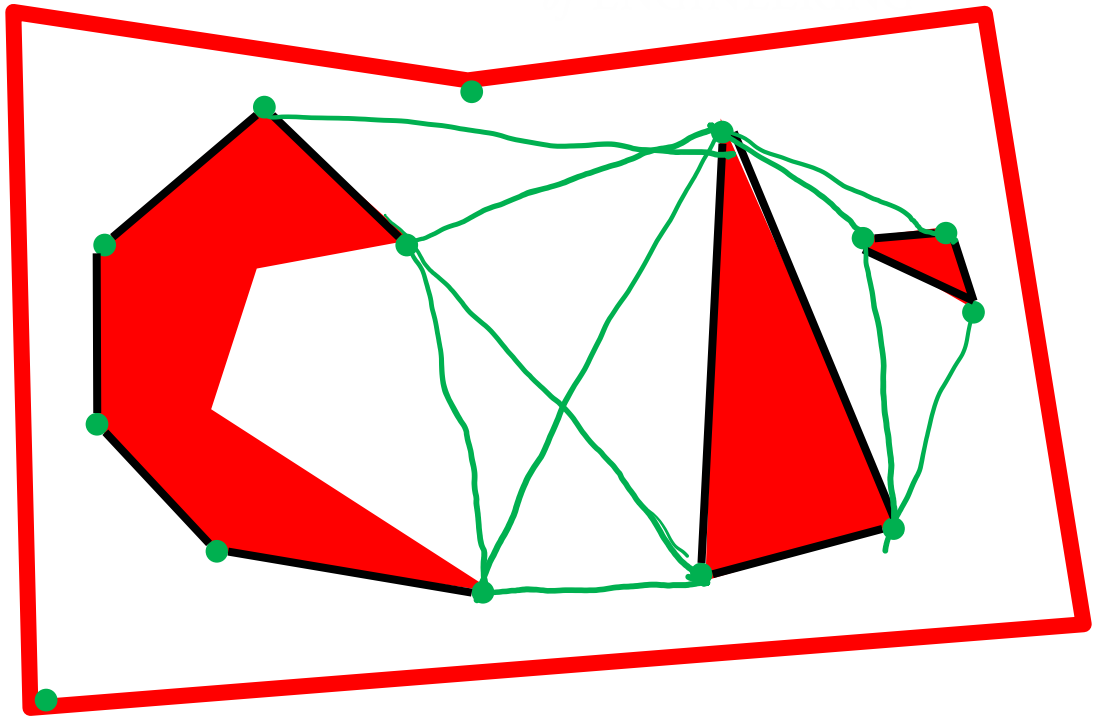
Bitangent edge

- A bitangent edge
  - Must touch two reflex vertices that are mutually visible from each other
  - Line must extend outward past each of the vertices without poking in to the C_{obs}



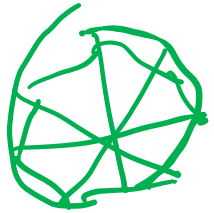
Bitangent edge

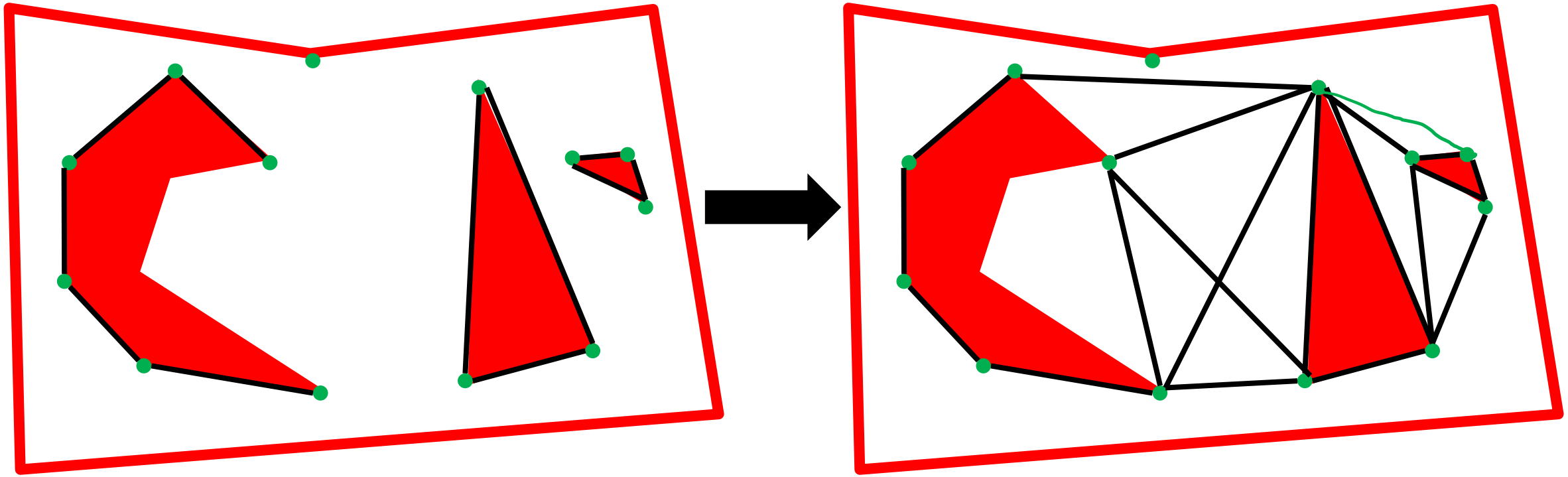
- Keep edges only between
  - Two reflex vertices on obstacles
  - All bitangents



Bitangent edge

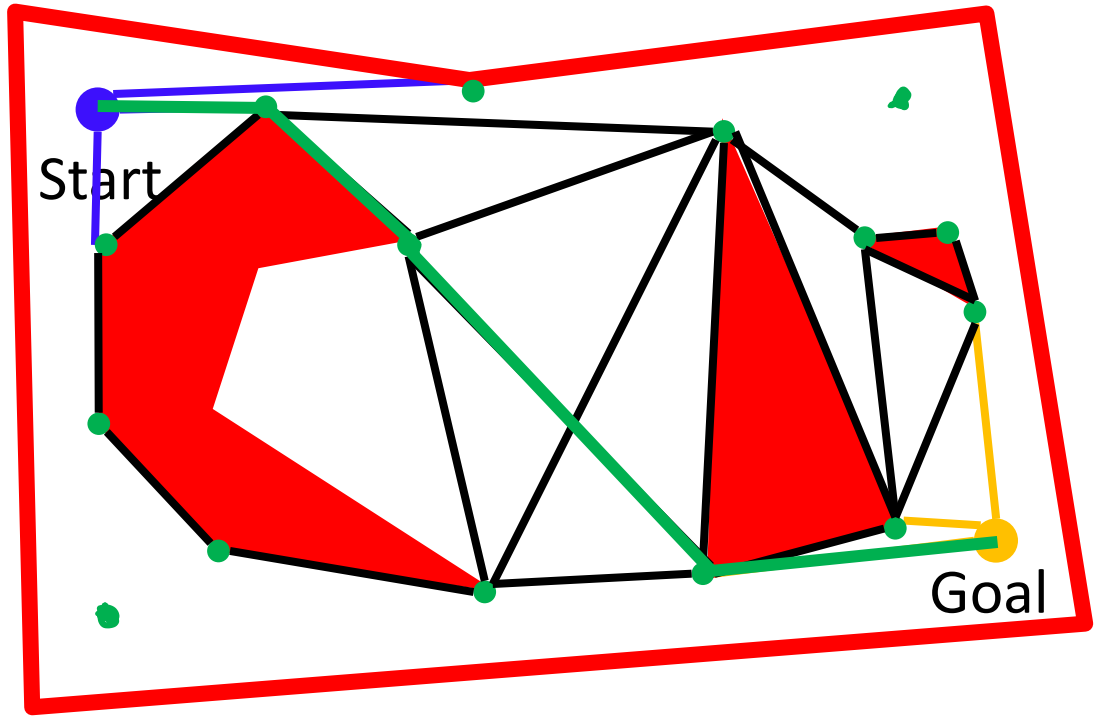
- Keep edges only between
  - Two reflex vertices on obstacles
  - All bitangents





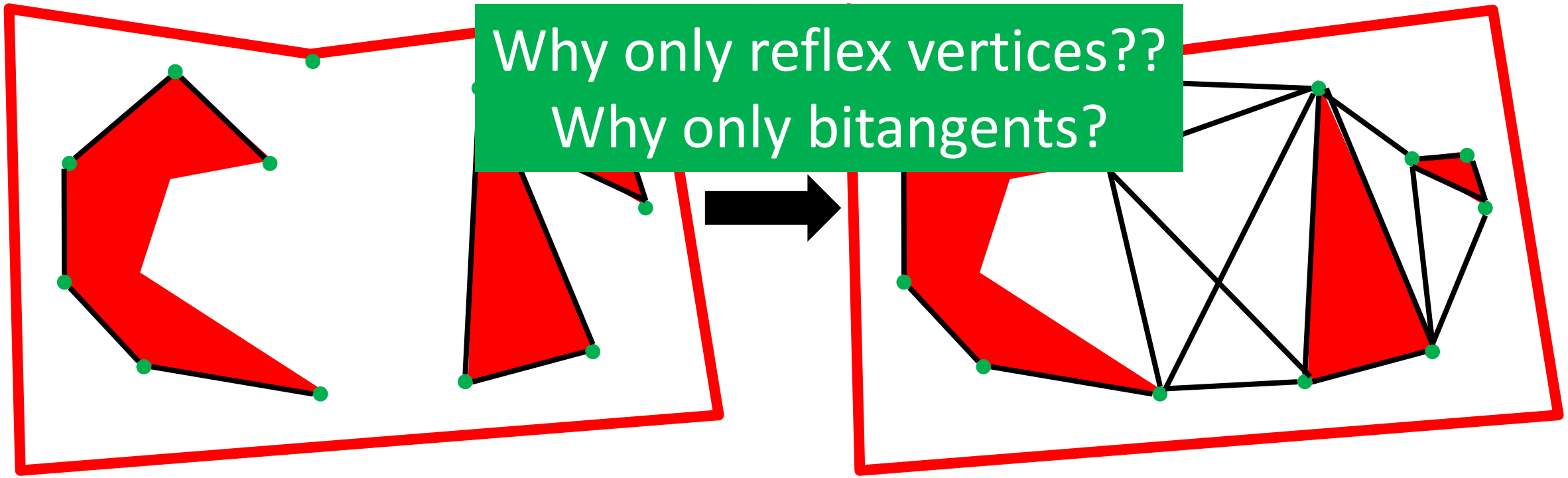
Then what???

- The shortest-path roadmap includes edges between consecutive reflex vertices and C_{obs} and also bitangent edges
- Extend the roadmap to include Start and Goal: **connect them to all visible vertices**
- Any graph search algorithm for the shortest path



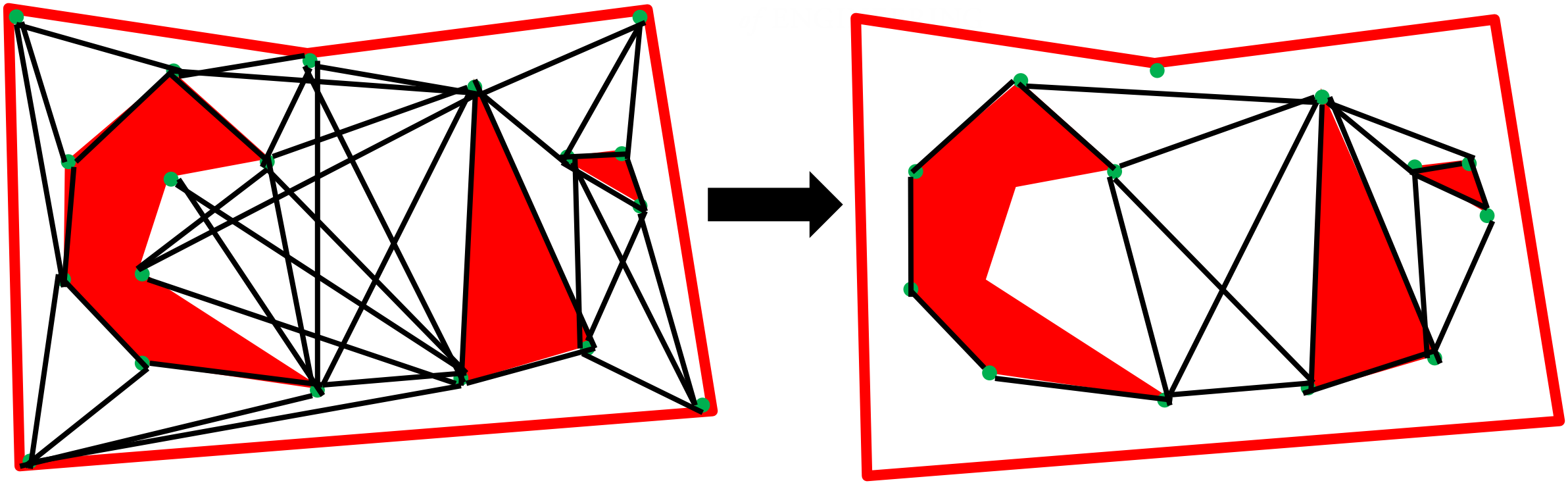
Reduced visibility roadmap

- Two reflex vertices on obstacles
- All bitangents



Recap: Reduced Visibility roadmap

- Keep edges only between
  - Two reflex vertices on obstacles
  - All bitangents



visibility map

Reduced visibility map

How do we implement?

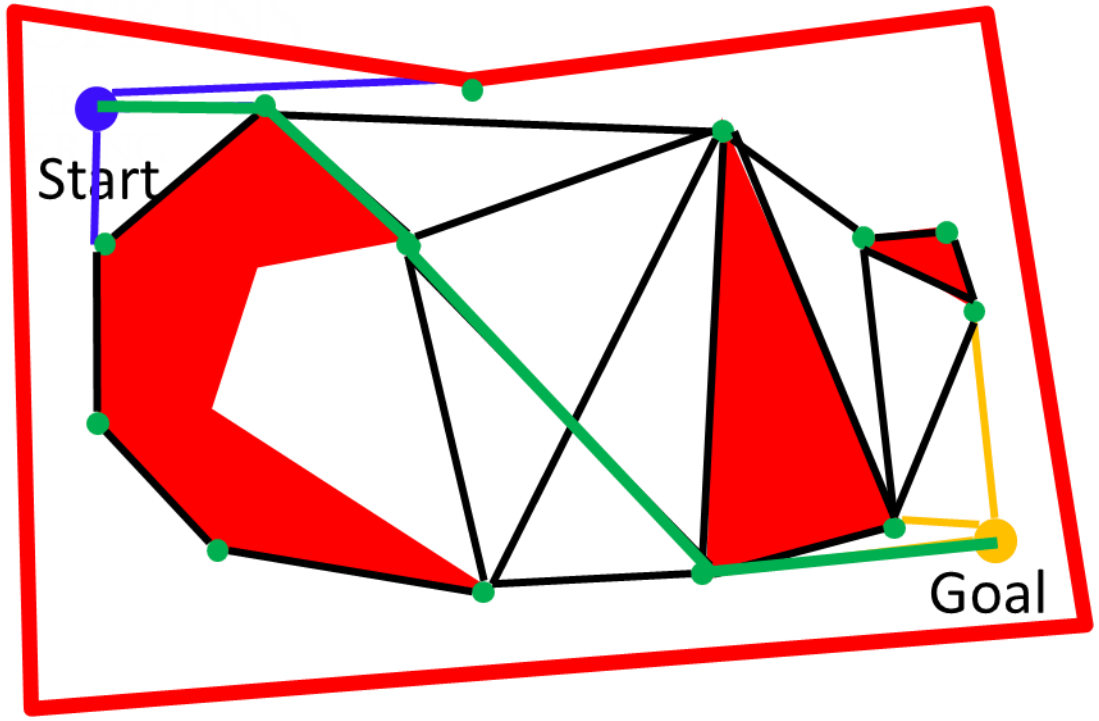
- We need **three subroutines** to check
  - if a vertex is reflex
  - If two vertices are mutually visible
  - If there exists a bitangent

- We need to run this algorithm only once for a given environment

- A graph search algorithm
  - Dijkstra's shortest path
  - DFS
  - BFS

When do we want to use visibility roadmaps?

- Static environment
  - **Pre-compute** the **reduced** visibility roadmap
- Allows us to find a path (not necessarily the shortest) between any given **start** and **goal**
  - Just add the start and goal to the graph
- Can query multiple times by changing the **start** and **goal**



Shortest-Path Roadmaps

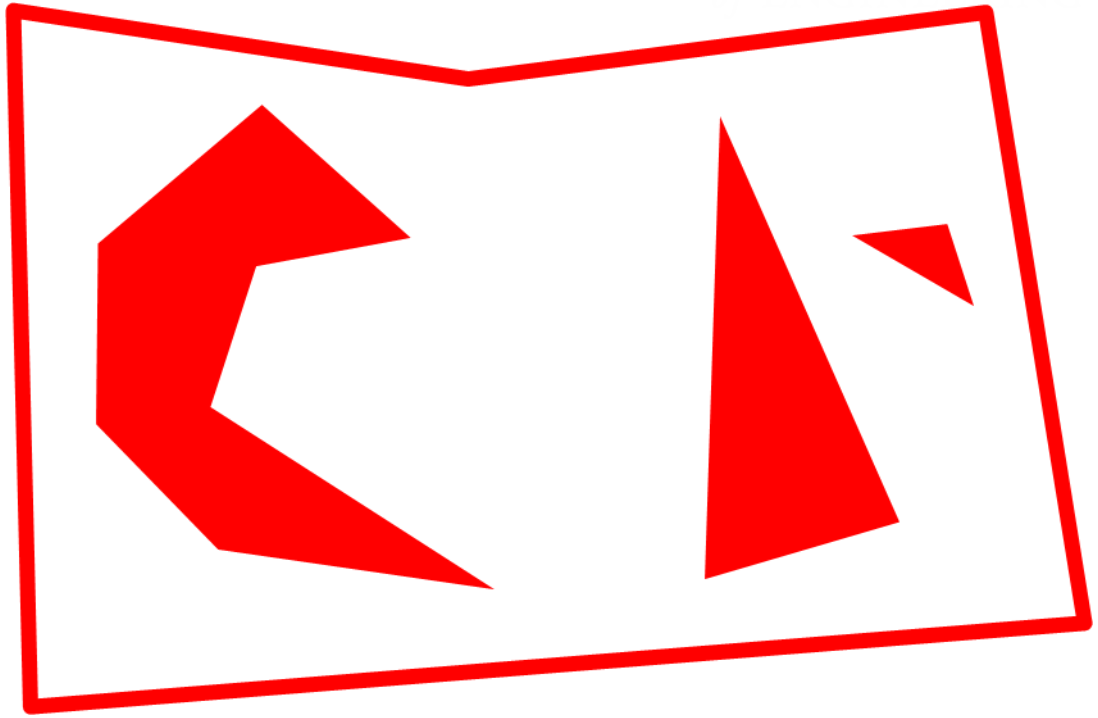
- ***Graze*** the obstacle vertices in the reduced visibility roadmap

- This also means that robot is very close to the obstacle (but not colliding) all the time
  - This works only if we assume **a point robot**
  - Also robot must have really **good collision detection sensors and control** otherwise **collision can not be avoided**

- What to do if the robot is **not a point robot???**

Minkowski Sum

- Inflate the obstacles: the inflated obstacle is called as C-space obstacles or C_{obs}
- Then plan assuming a point robot

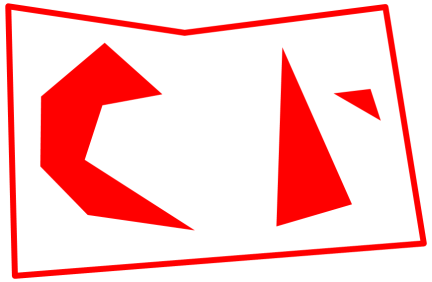


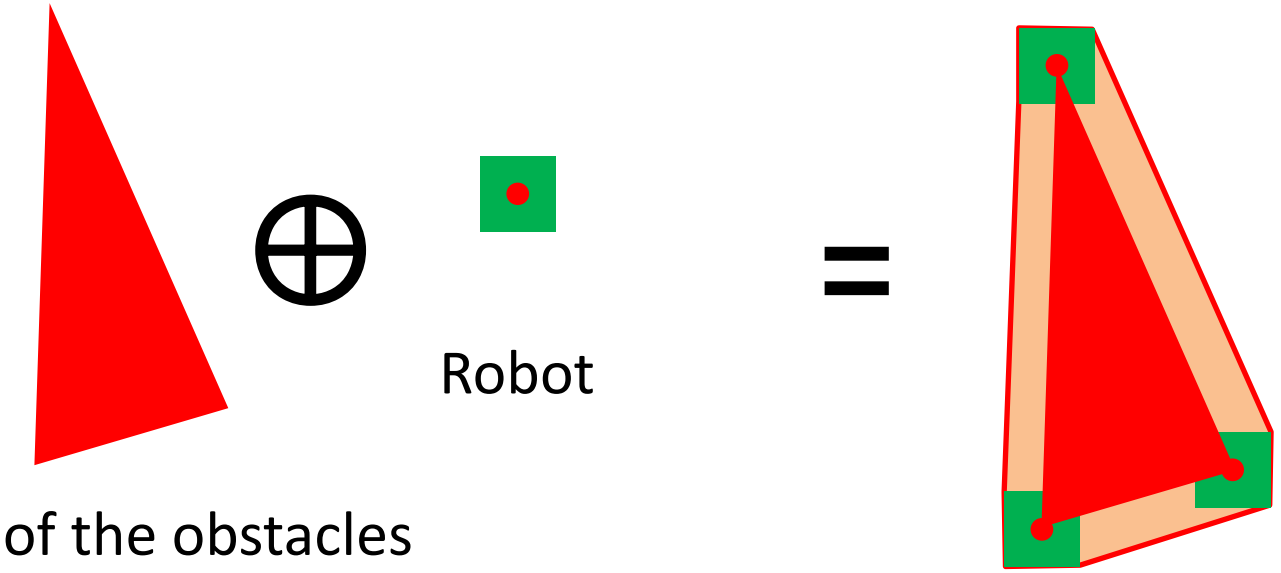
Environment

Robot

Minkowski Sum

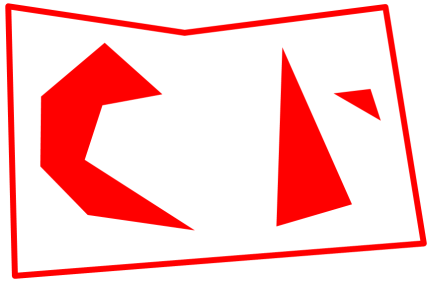
- A: obstacle and B: robot
- Minkowski Sum
  - $A \oplus B = \{ a + b \mid a \in A, b \in B\}$
- Slide origin of B on every point (boundary) of A
- Its kind of a higher dimensional convolution

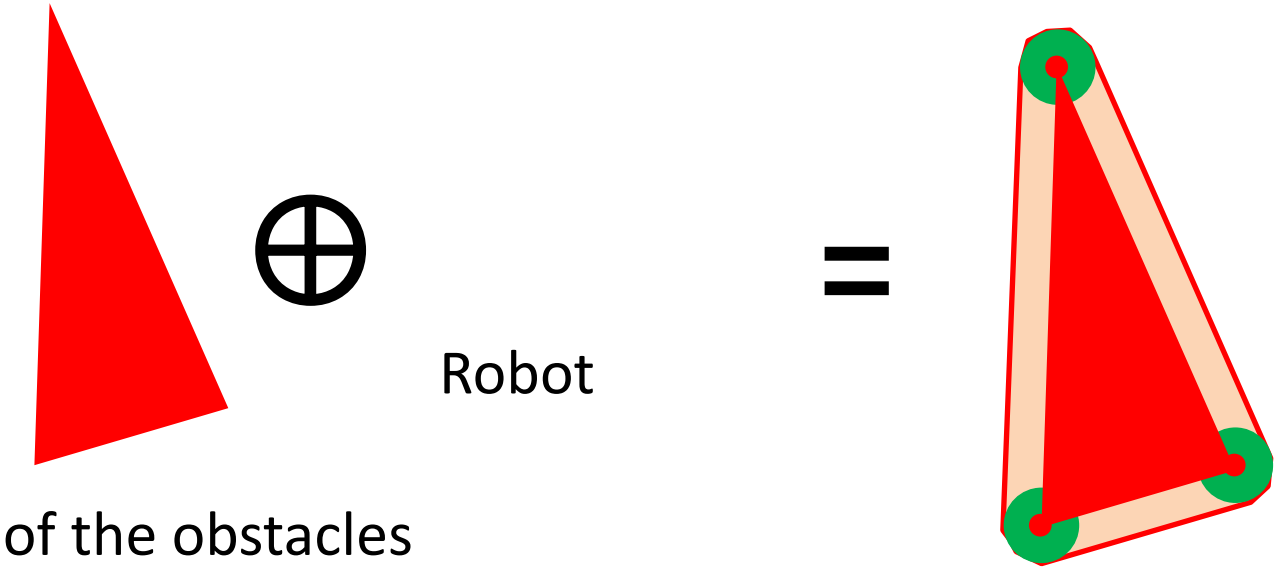




Minkowski Sum

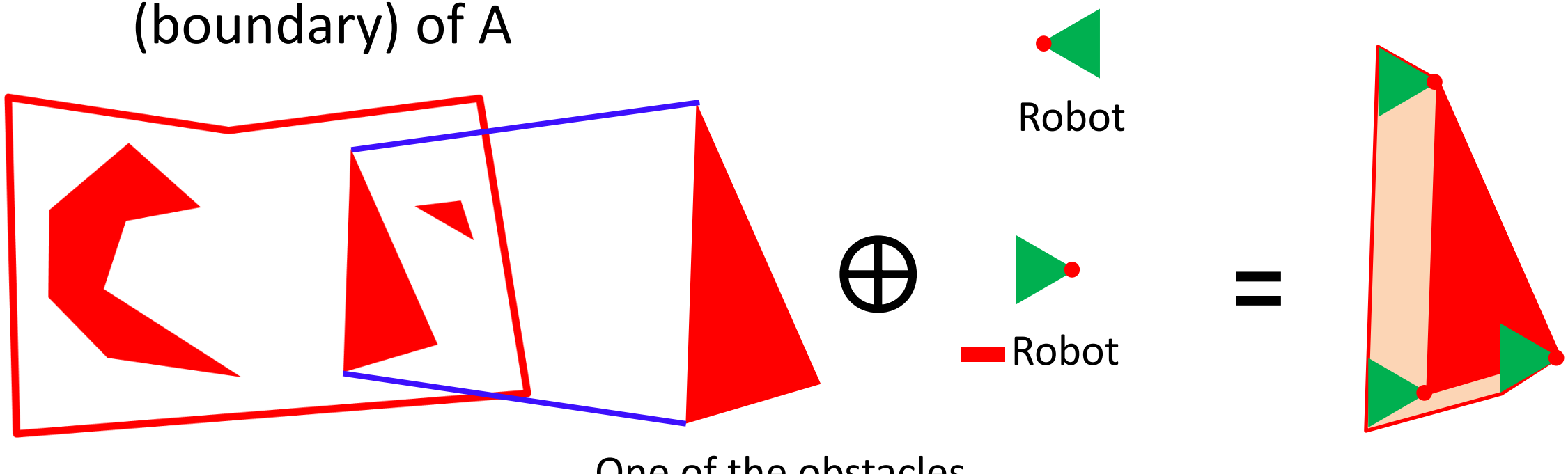
- A: obstacle and B: robot
- Minkowski Sum
  - $A \oplus B = \{ a + b \mid a \in A, b \in B\}$
- Slide origin of B on every point (boundary) of A
- Its kind of a higher dimensional convolution





Minkowski **Difference: non-symmetric robot**

- A: obstacle and B: robot
- Minkowski **Difference**
  - $A + (-B) = \{ a - b \mid a \in A, b \in B \}$
- Flip the robot and then slide origin of B on every point (boundary) of A



What if shortest-path isn't what we want?

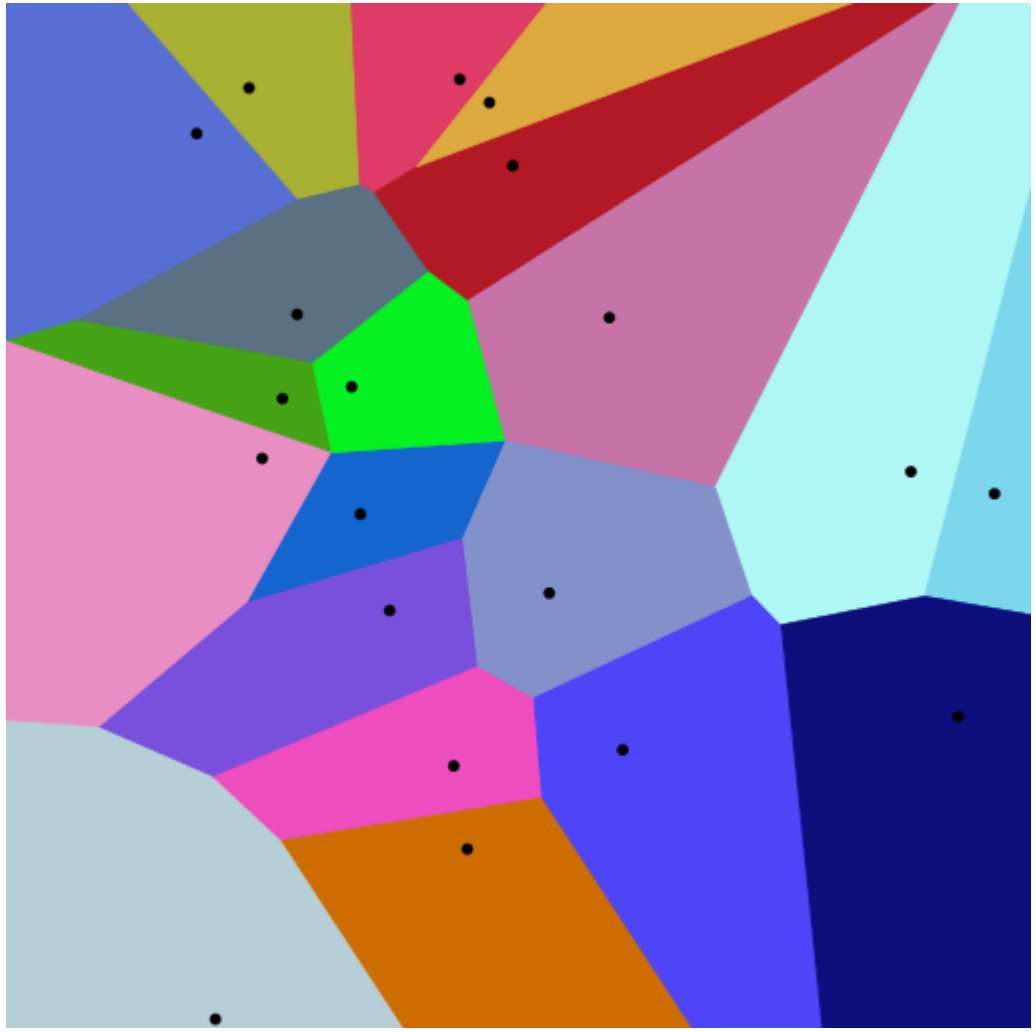
- Safest path/maximum clearance path?
  - maximum possible obstacle distance

- Take a simple case
  - Two **point** obstacles
  - Midway point between them is the maximum clearance path point

- What if we extend it to n **point** obstacles?
  - We get **Voronoi Diagram** !!

Voronoi Diagram

- Partition the plane (2D) into regions (like city zones/pin codes)
- Each region/cell (zone) is **closer to a single point x_i** than any other point in the whole plane
  - We must define the distance metrics
- Do we use such an arrangement?
  - Post office location and their serving are?
  - Fire stations and their serving are?
  - Its an optimal solution to make every other point in the plane have best possible access to a given resource located at X_i



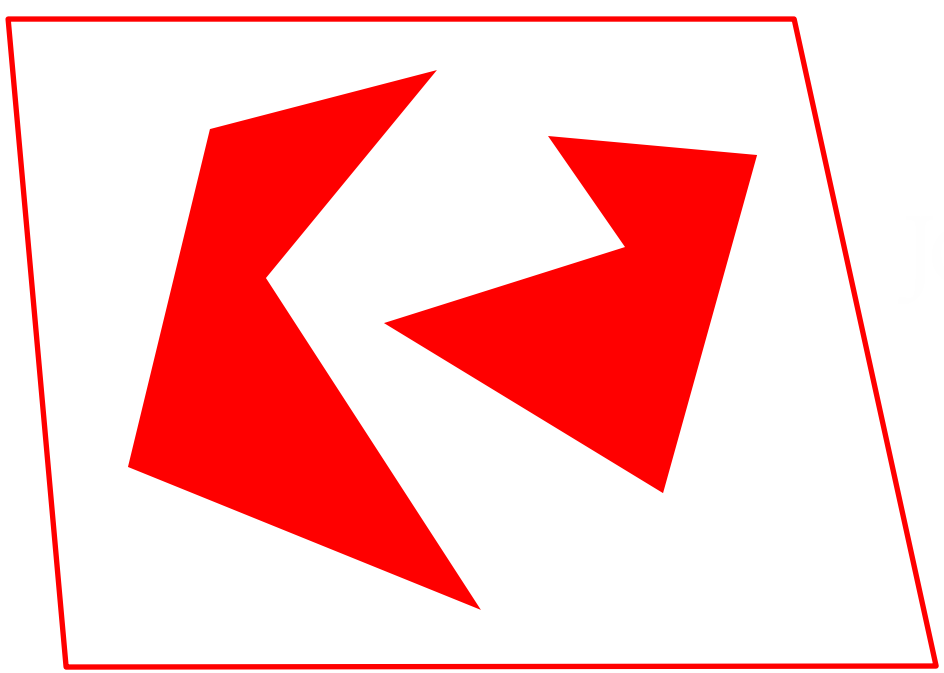
Source: Wikipedia

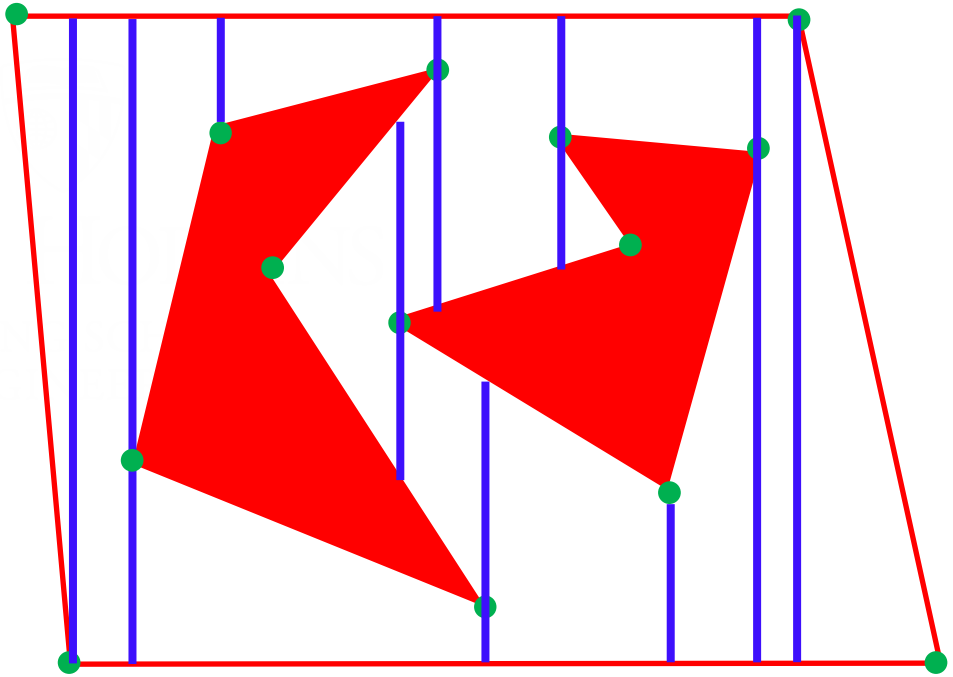
Maximum Clearance Roadmaps

- How do we generate such a roadmap?
  - Generalized Voronoi diagrams
  - Skeletonization

- There are many other ways

Vertical Cell Decomposition





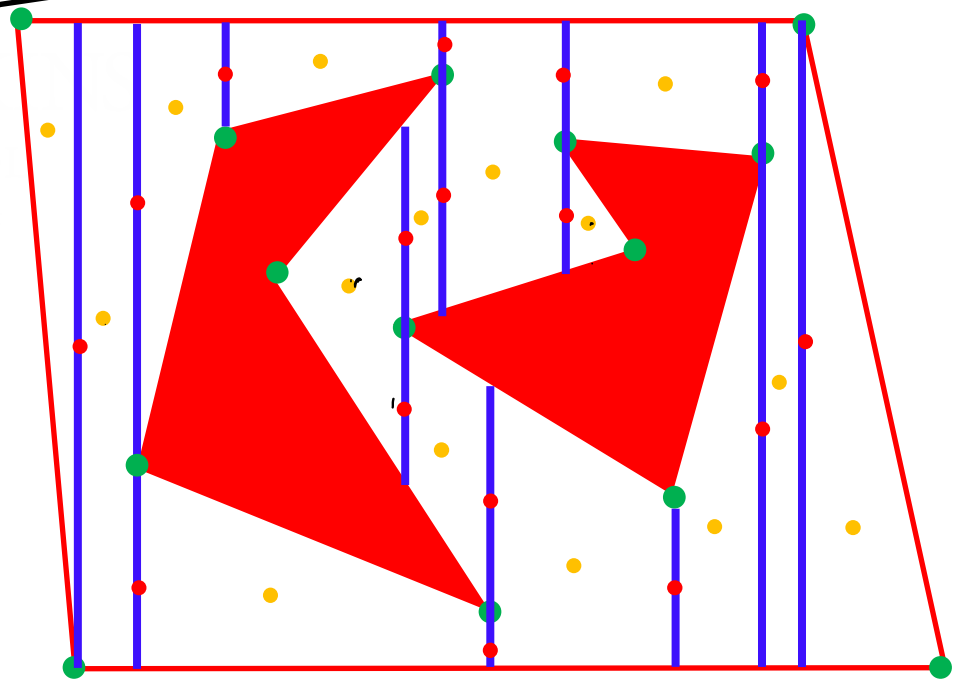
- Consider the above environment
- Mark each vertex
- Draw a vertical line from each vertex until the line hits an obstacle

Vertical Cell Decomposition

- Create vertices at
  - Centroid of the cell
  - At the mid point of the vertical lines

do we need them?

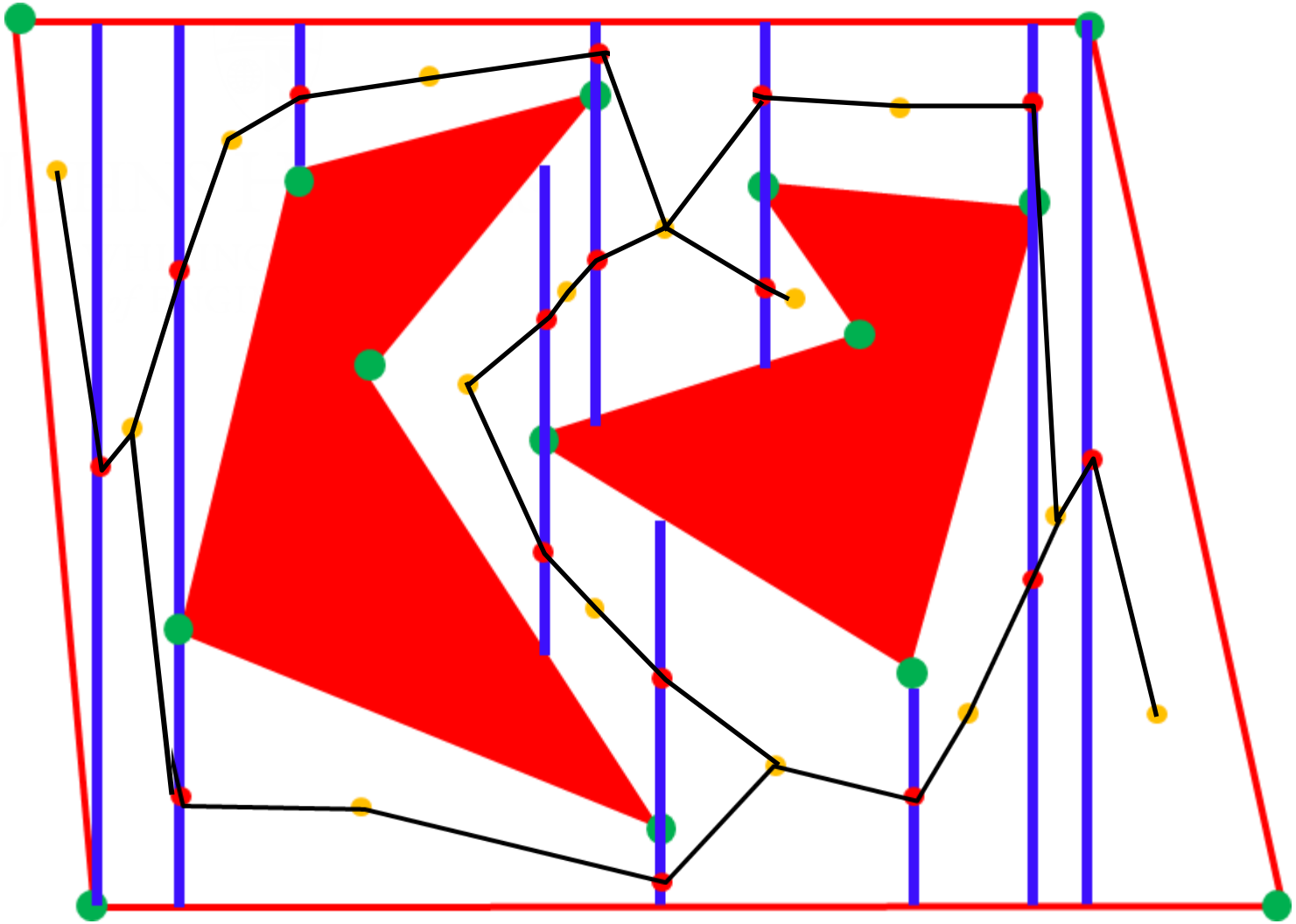
- Add edges only to left and right neighbours



Vertical Cell Decomposition

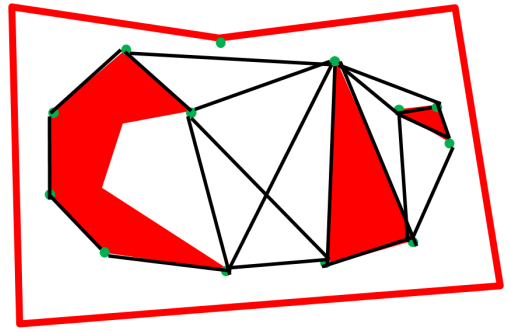
- Create vertices at
  - Centroid of the cell
  - At the mid point of the vertical lines

- Add edges only to left and right neighbours

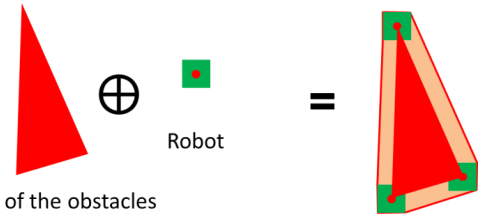


Recap

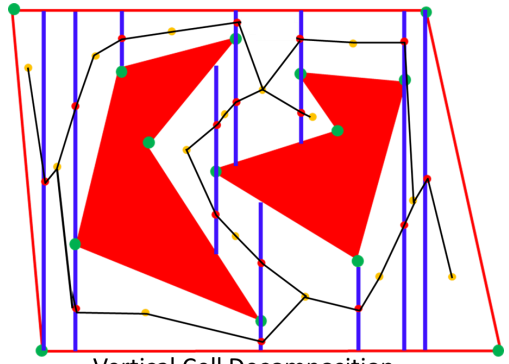
- Visibility roadmaps
  - Reduced visibility roadmap
    - Still a point robot
    - Minimum distance roadmap
    - Always grazing the obstacles
  - Minkowski sum
    - Inflate the obstacles to get C_{obs}
    - Works well for polygonal obstacles and robot
  - Vertical cell decomposition
    - A simple approach to divide the environment into smaller cells



Reduced visibility map



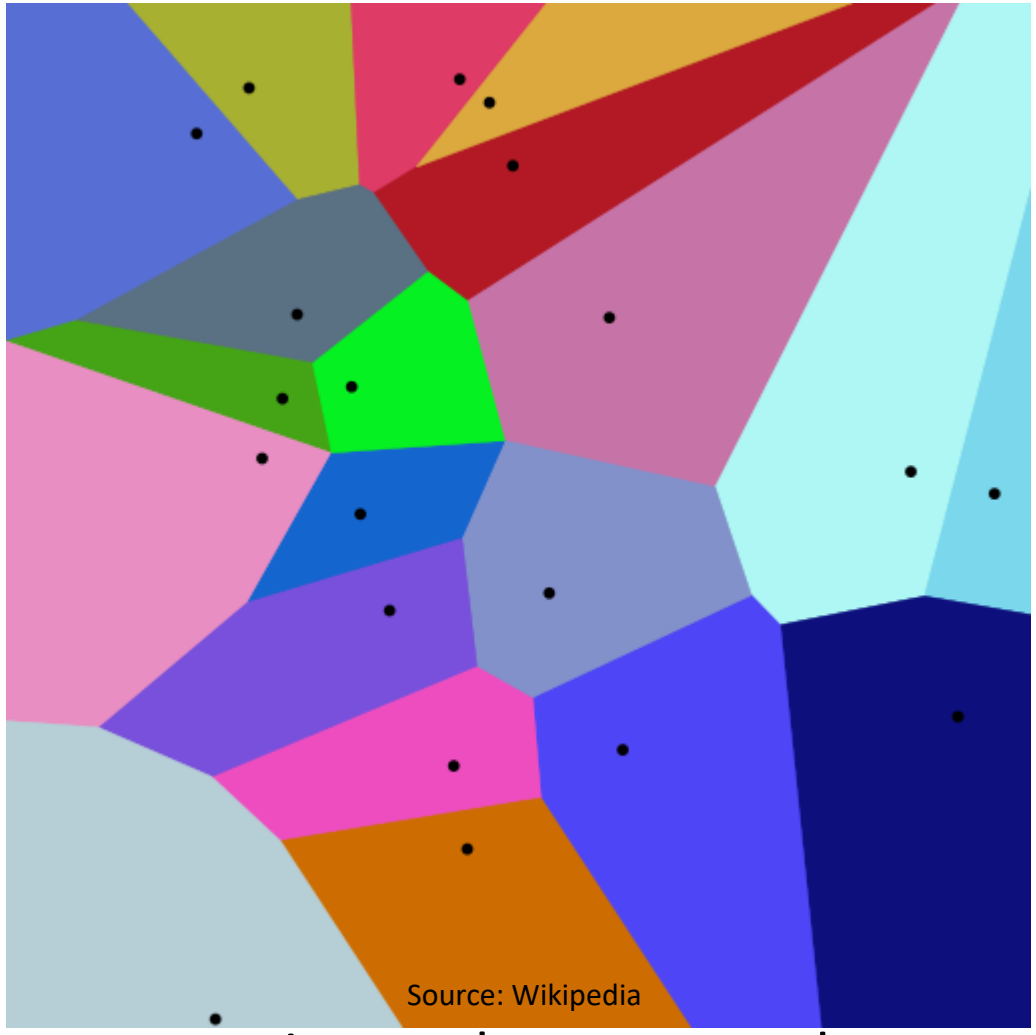
One of the obstacles



Vertical Cell Decomposition

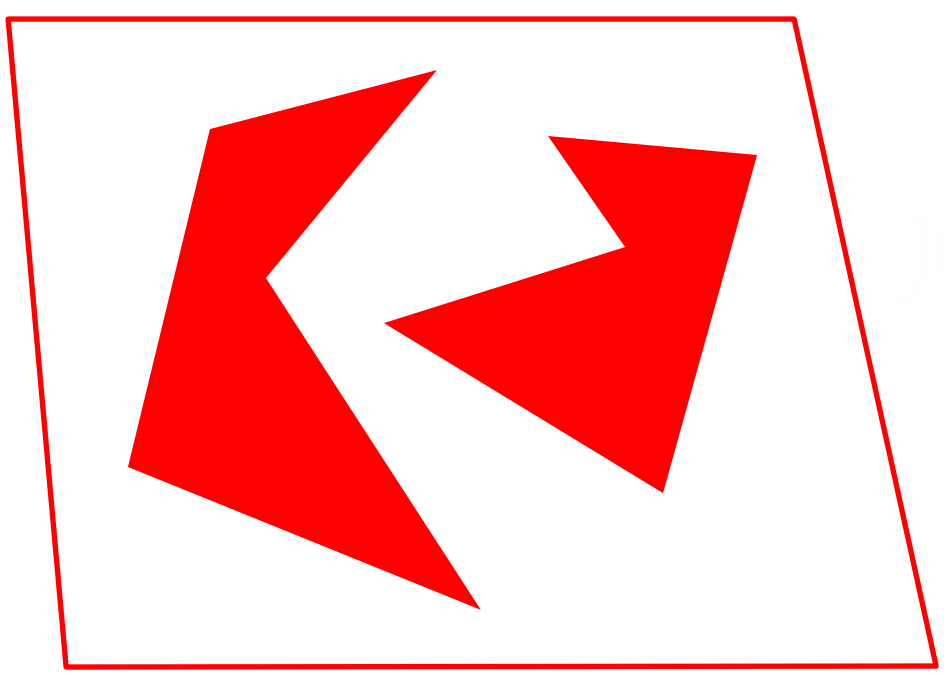
Voronoi Diagram: maximum clearance roadmap

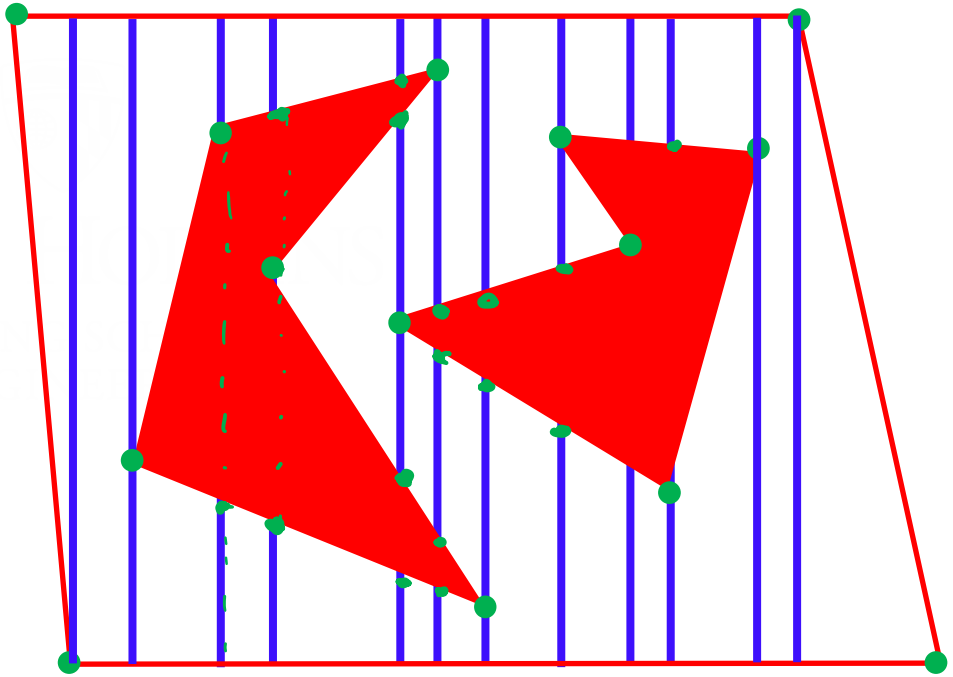
- Partition the plane (2D) into regions (like city zones/pin codes)
- Each region/cell (zone) is **closer to a single point x_i** than any other point in the whole plane
  - We must define the distance metrics
- Do we use such an arrangement?
  - Post office location and their serving are?
  - Fire stations and their serving are?
  - Its an optimal solution to make every other point in the plane have best possible access to a given resource located at X_i



maximum clearance roadmap

Cylindrical Decomposition

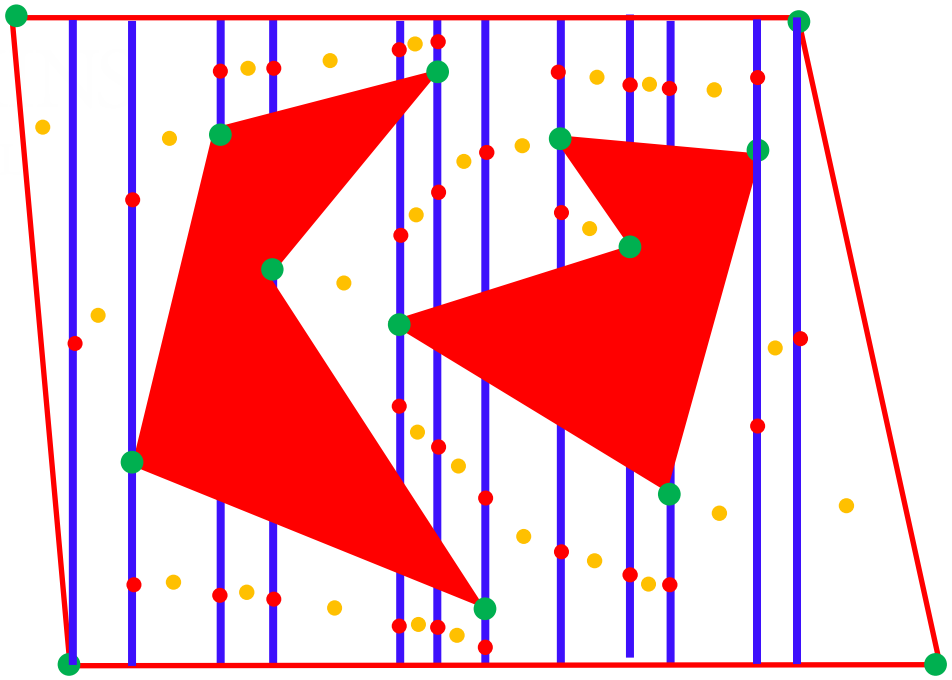




- Consider the above environment
- Mark each vertex
- Draw a vertical line from each vertex ~~until the line hits an obstacle~~ and let them continue after the obstacles (**-inf Y** - **+inf Y**)

Cylindrical Decomposition

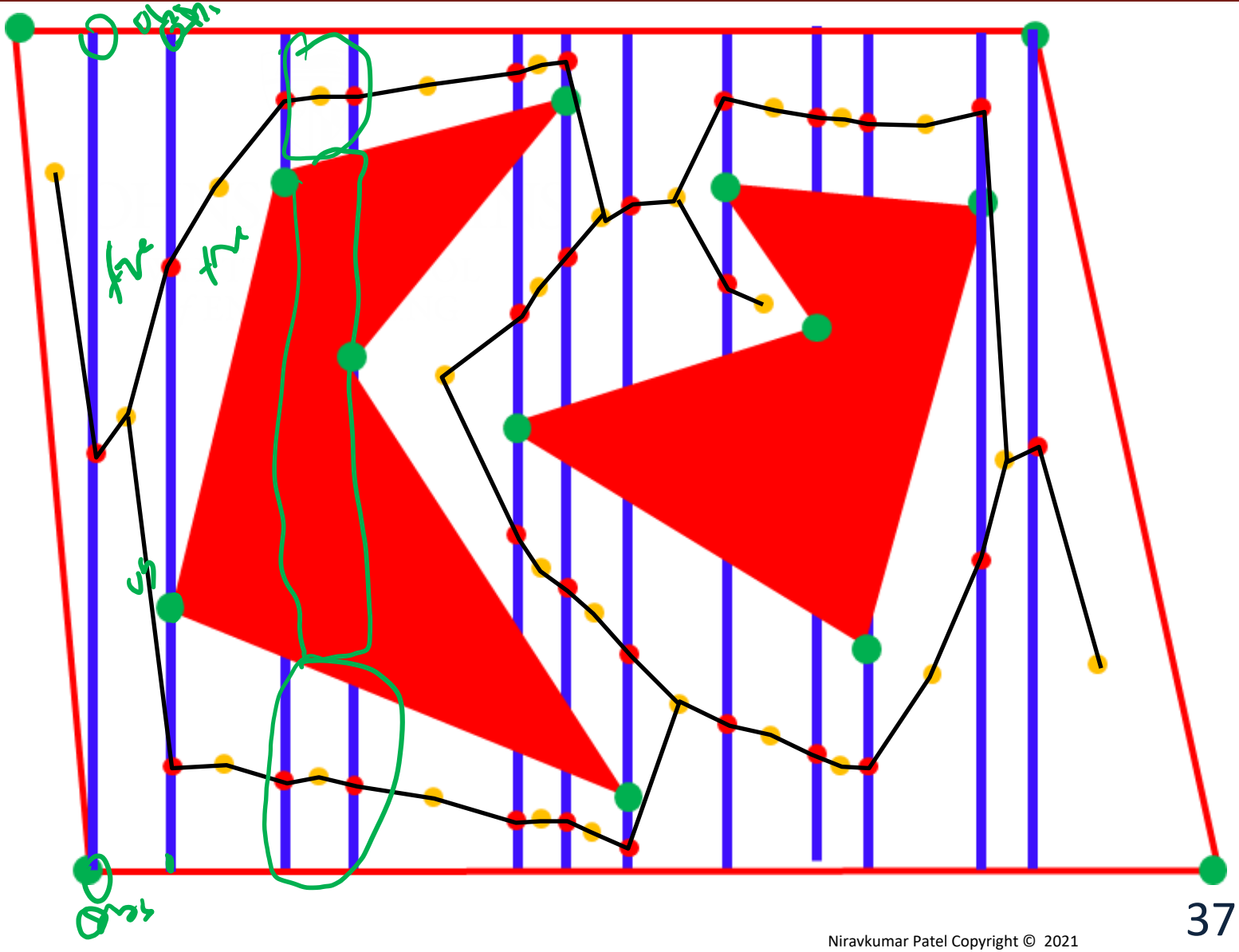
- Create vertices at
  - Centroid of the cell
  - At the mid point of the vertical lines
- Add edges only to left and right neighbours



Cylindrical Decomposition

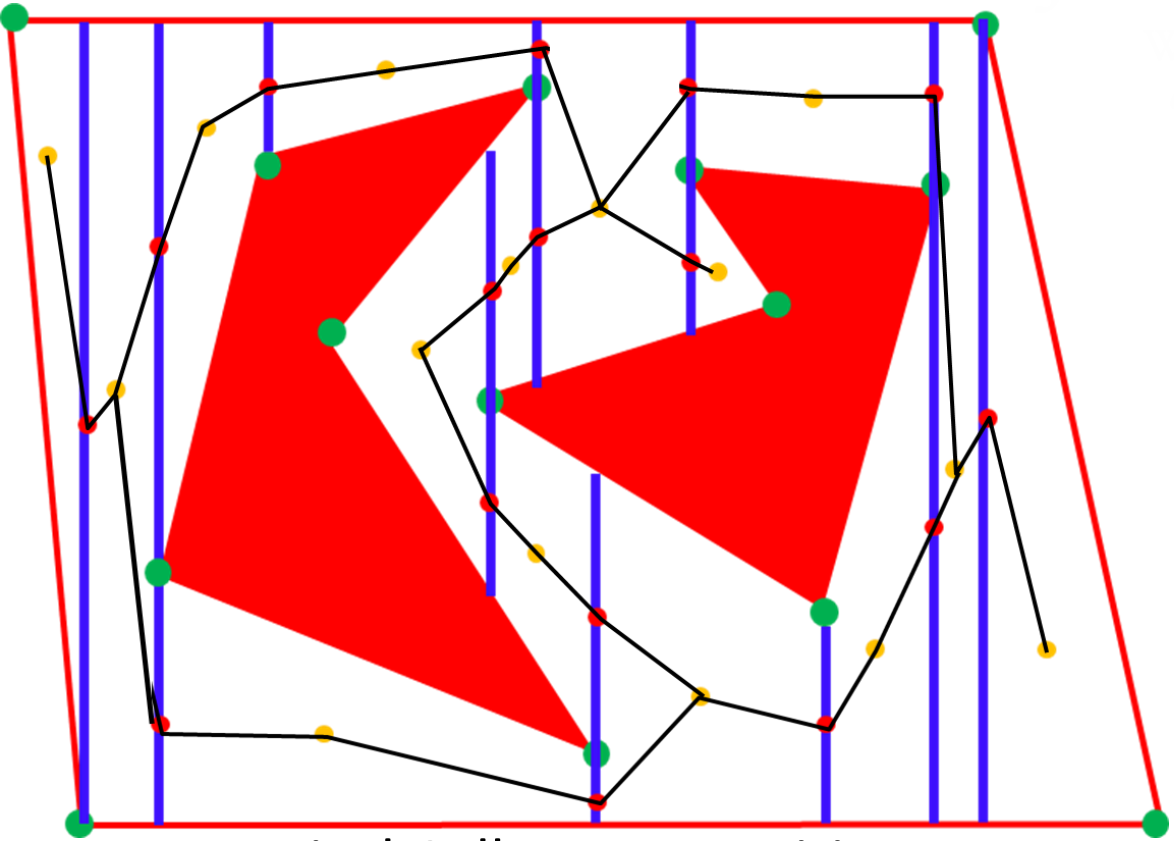
- Create vertices at
  - Centroid of the cell
  - At the mid point of the vertical lines

- Add edges only to left and right neighbours

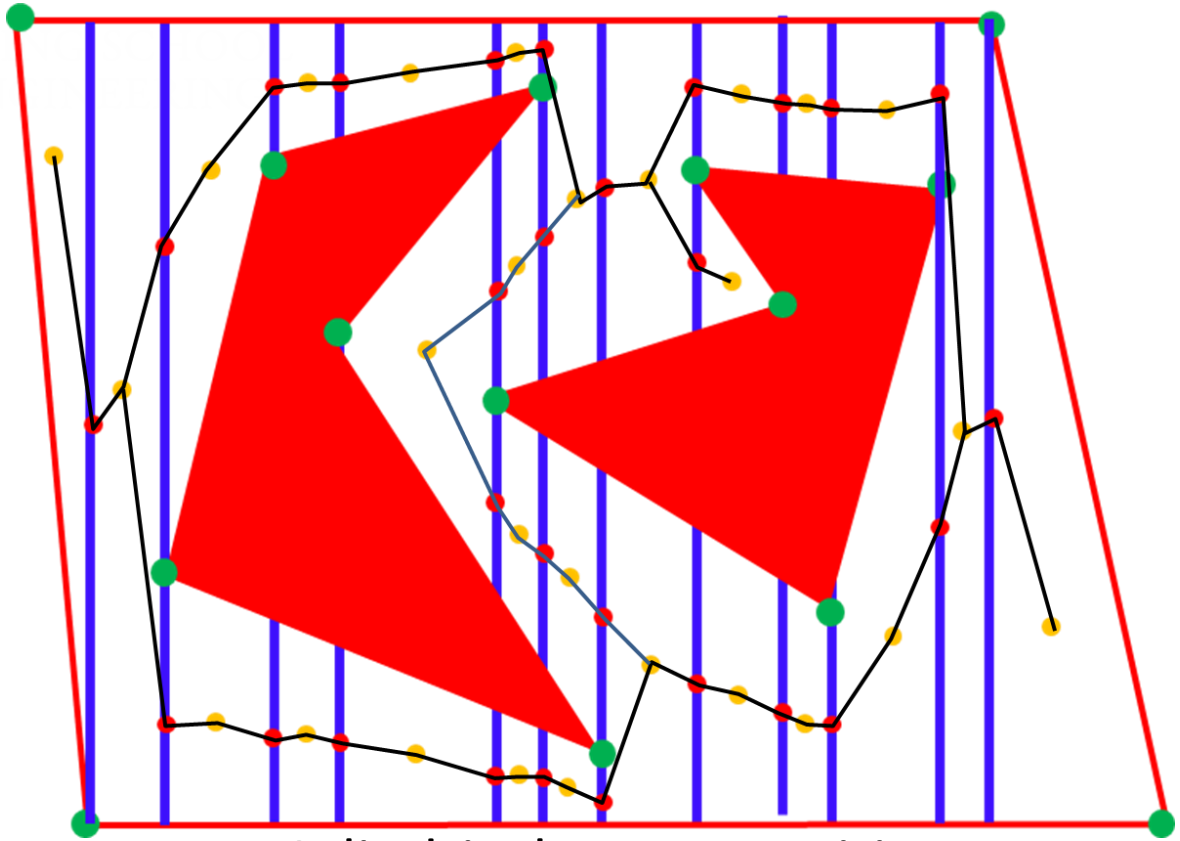


Vertical Cell Vs. Cylindrical Decomposition

- Alternate events on the vertical lines are C_{obs} and C_{free}
- Each strip (cylinder) is a stack of cells made of alternating C_{obs} and C_{free}
- Generalizes better to higher dimensions and more complex configuration spaces



Vertical Cell Decomposition



Cylindrical Decomposition

Triangulation

- Divide C_{free} into triangles
- Place vertices at
  - Centroid of each triangle
  - Mid-point of each of the sides of the triangle that is in C_{free}
- Have to avoid thin triangles: more of a computational geometry problem and solutions exist for this

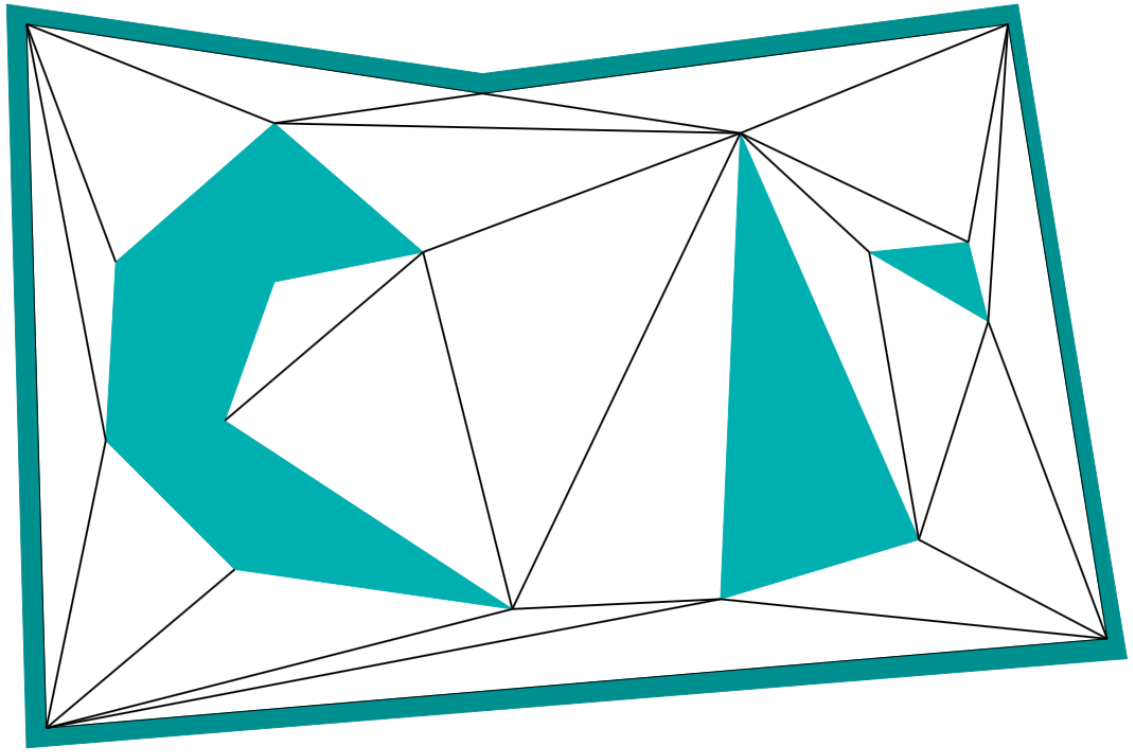


Figure 6.16: A triangulation of $\mathcal{C}_{free}$.

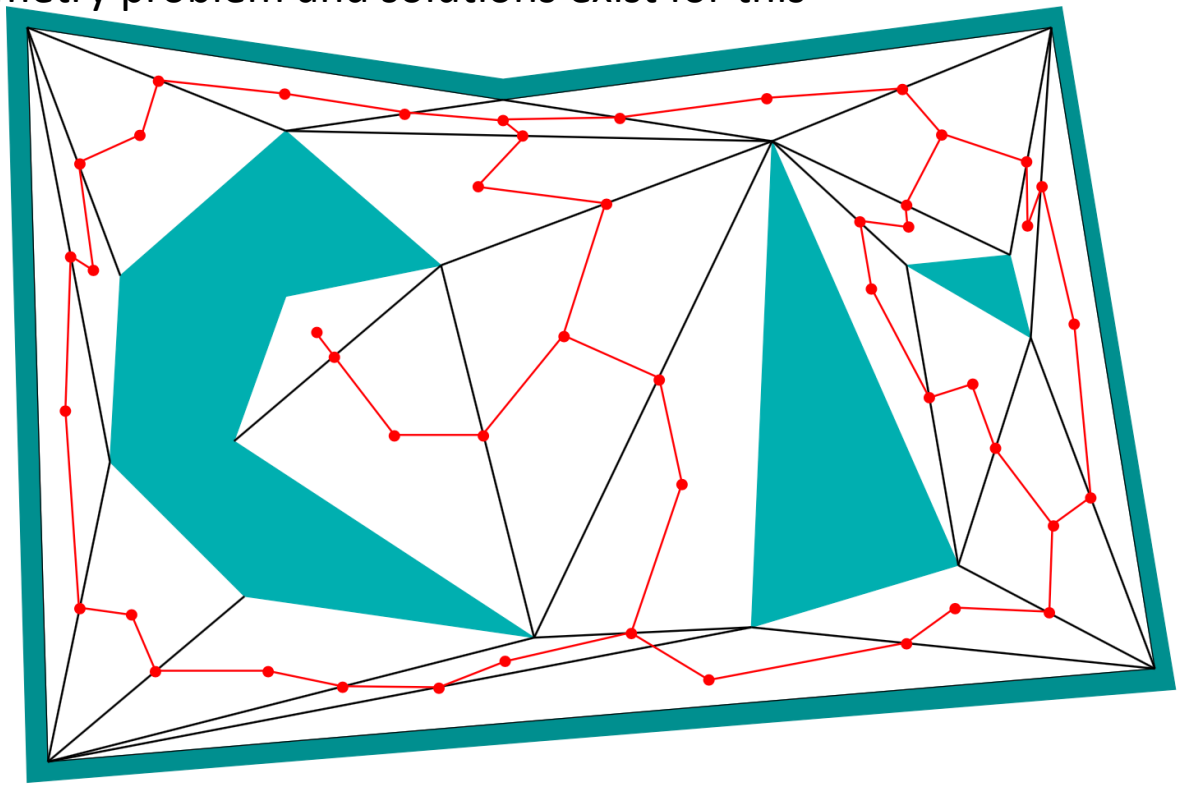
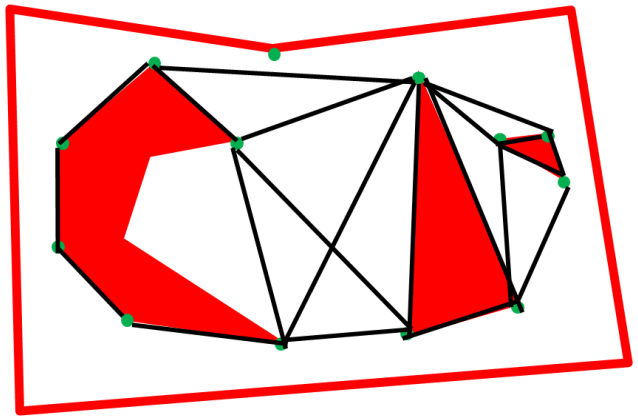
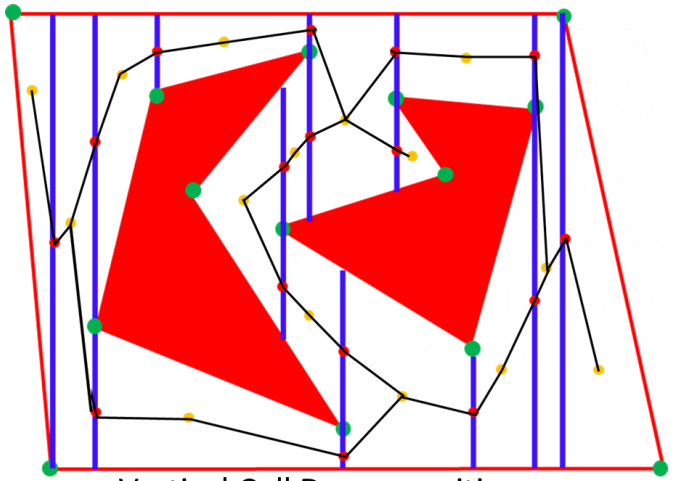


Figure 6.17: A roadmap obtained from the triangulation.

Summery: visibility roadmaps



Reduced visibility map



Vertical Cell Decomposition

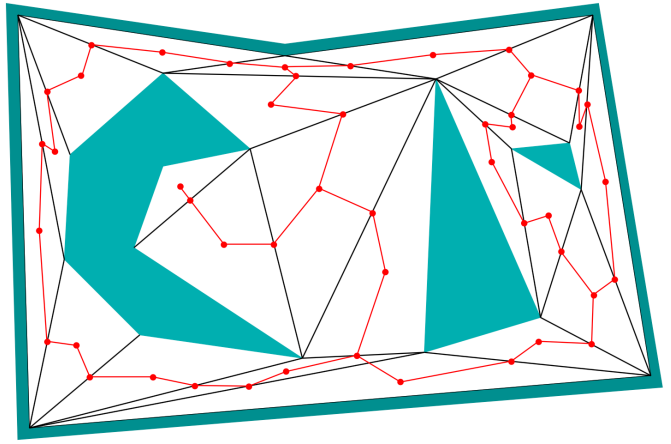
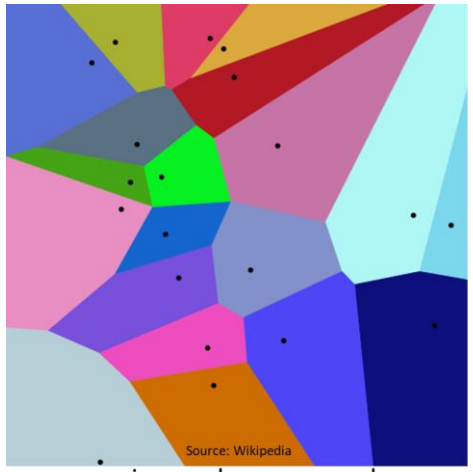
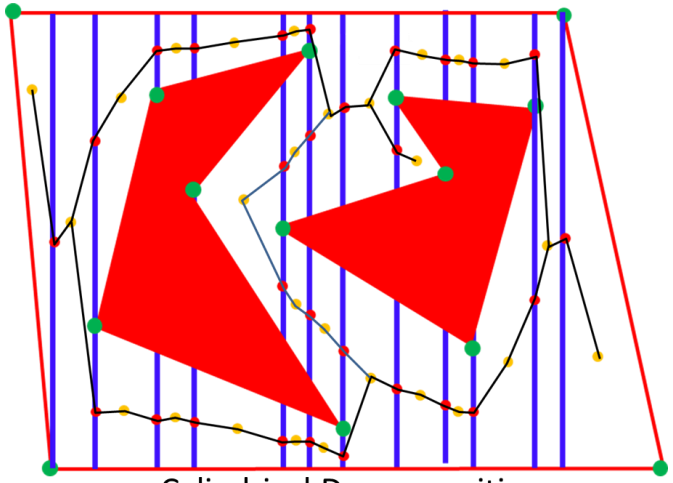


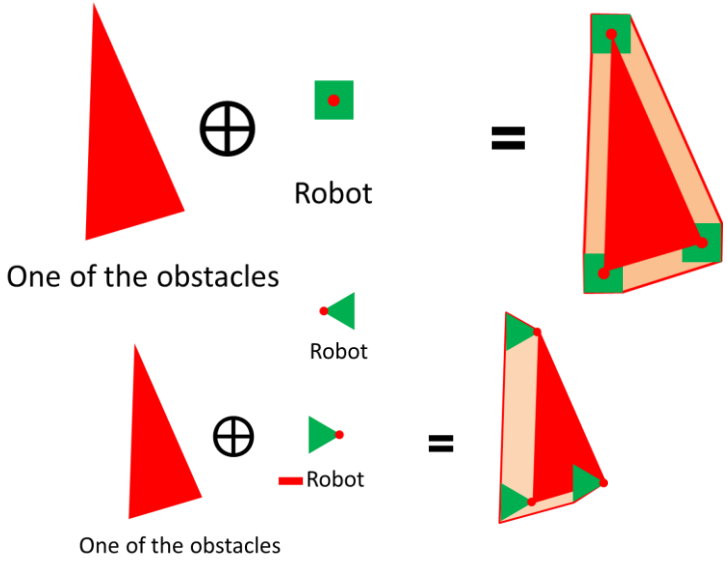
Figure 6.17: A roadmap obtained from the triangulation.



maximum clearance roadmap



Cylindrical Decomposition



Minkowski Sum

World isn't that simple/nice !

- Relax most assumptions
  - Complex robot model and environments
  - Robot may not be polygonal
  - C-space obstacles may not be polygonal
  - Robot can rotate and translate
  - Robot with chain of rigid bodies
- Grid-based and visibility graph based methods don't generalize
- Finding a feasible path might become challenging
- Solution
  - Sampling based algorithms to deal with complex C-spaces

How do we implement?

- We need **three subroutines** to check
  - if a vertex is reflex
  - If two vertices are mutually visible
  - If there exists a bitangent

- A graph search algorithm
  - Dijkstra's shortest path

Credits

- Many of the figures are adapted from the textbook: Planning Algorithms by Steven M. LaValle
- Some of the slides are adapted from lecture notes by Pratap Tokekar, ECE 4984/5984: (Advanced) Robot Motion Planning, Virginia tech