CS 4649/7649
Robot Intelligence: Planning

Visibility Graph, Configuration Space

Sungmoon Joo

School of Interactive Computing
College of Computing
Georgia Institute of Technology

*Slides based in part on Dr. Mike Stilman's lecture slides

Course Info.

- HW#1 due Oct 6th
  - Wiki - Add your group info. (1-person group?)
  - Need a repo.?
  - Late policy – No late HWs
- Final Project Topic? Start early

Motion Planning Statement

"Compute a continuous sequence of collision-free robot configurations connecting the initial and goal configurations."

Notations/Definitions

- q: the robot's configuration, a complete specification of the position of every point in the system (a set of parameters that define the robot geometry in the world)
- W : the robot's workspace, the points the robot can possibly reach
- Wo_i : the i'th obstacle,
- W_{free} : the robot's free space, $W_{free} = W - (\cup WO_i)$
- a path $c \in C^0$, $c : [0,1] \rightarrow W_{free}$ where c(0) is q_{init} and c(1) is q_{goal}

*Class C^0 : all continuous functions

Bug Algorithms Summary

Algorithm	Bug 0	Bug 1	Bug 2
Completeness	X	0, Exhaustive	0, Greedy
Characteristic	-	Safe, Reliable	Better in some cases. But worse in other cases

***None of them is optimal**

A Useful Notion: Competitive Ratio (CR)

Lost Cow Problem

a short-sighted cow is following along an infinite fence and wants to find the gate

(a)If the cow is told that the gate is exactly distance 1unit away
- CR = 3

(b)If the cow is told only that the gate is at least distance 1unit away
- CR = 9
- The optimal solution consists of moving alternatively in the right and left directions, exploring 1 distance unit in one direction(say, right), then 2 units (from the original starting position) in the other direction(say, left), then 4 units in the right, and continuing in powers of 2 until the gate is found.

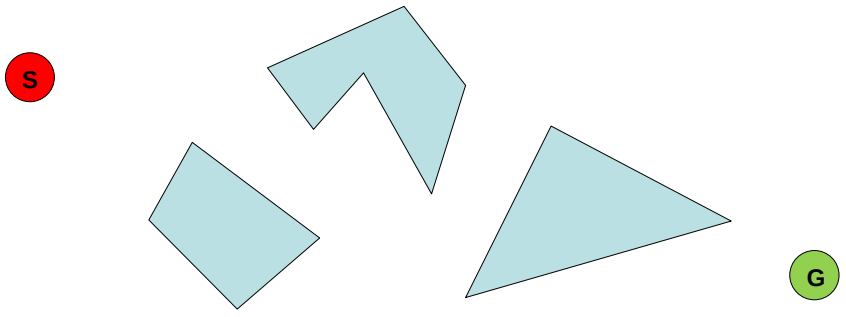
Figure from "Planning Algorithms" by Steven M. LaValle

Planning Requires Models

- Bug algorithms don't plan ahead.
- They are not really motion planners, but "reactive motion strategies"
- To plan its actions, a robot needs a (possibly imperfect) predictive model of the effects of its actions, so that it can choose among several possible combinations of actions

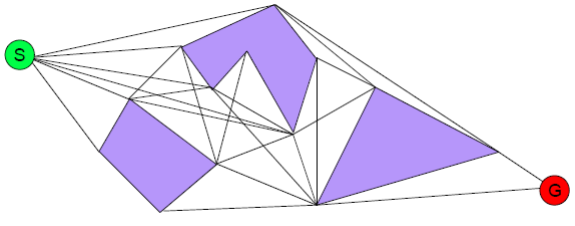
Path Planning

Assuming full knowledge, how does the robot 'plan' its path from S to G?



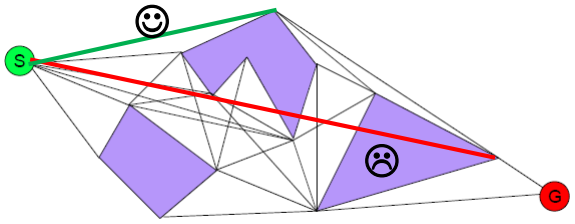
Visibility Graph

- Assuming polygonal obstacles, it looks like the shortest path is a sequence of straight lines joining the vertices of the obstacles



Visibility Graph

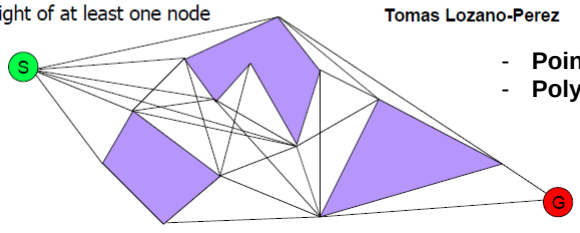
- Assuming polygonal obstacles, it looks like the shortest path is a sequence of straight lines joining the vertices of the obstacles
- Visibility graph G = set of unblocked lines between vertices of the obstacles + initial & goal configuration



Visibility Graphs*

- Early Motion Planning Algorithm
- Nodes share an edge if they are within "line of sight"
- All points in free space are within sight of at least one node

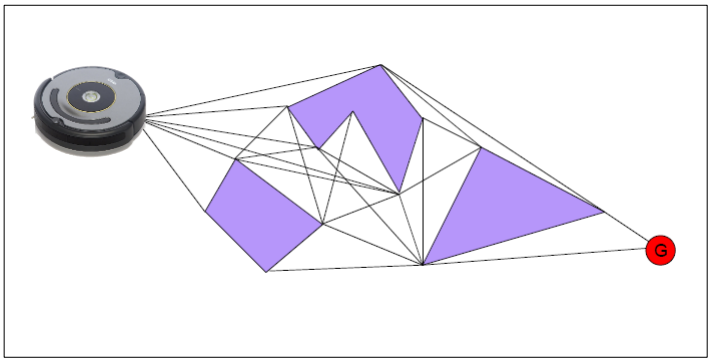
Tomas Lozano-Perez



- **Point robot**
- **Polygonal obstacles**

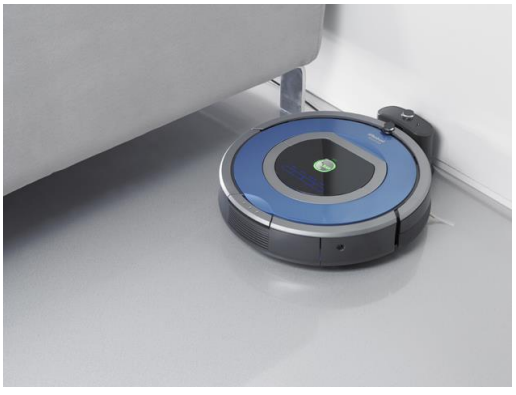
* "An algorithm for planning collision-free paths among polyhedral obstacles" 1979 T. Lozano-Perez & M. A. Wesley
* "A mobile automaton: An application of artificial intelligence techniques" 1969 N.J Nilson

Problem1



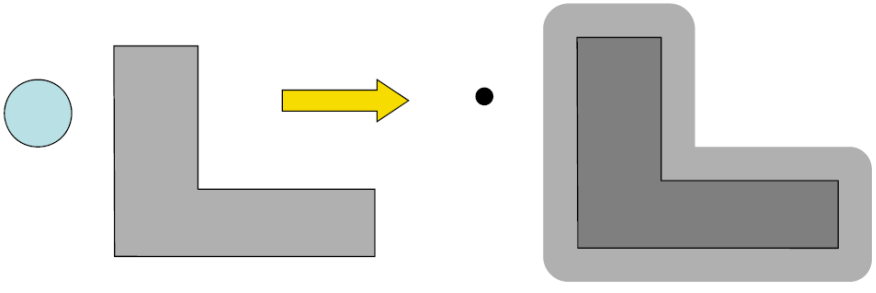
Robots are Not Points

Robots are Not Points



Robots are Not Points: How to Solve This?

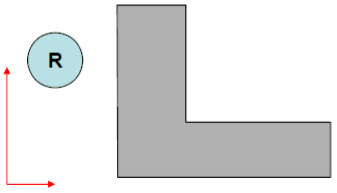
- 1983 **Tomas Lozano-Perez**
  - "**Spatial Planning: A Configuration Space Approach**"



What if we can 'transform' the 'work space'
such that the robot is represented by a point?

Configuration Space

- A configuration is a set of parameters that define the robot geometry in the world
- How many parameters? **TWO: (X,Y)**



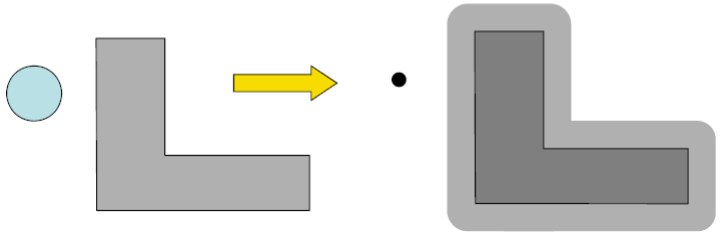
Configuration Space

- Configuration, q: A complete specification of the position of every point of the robot
- C-Space, Q = space of all possible configurations of the robot
  ex. Cleaning robot in circular shape with radius r, center position (C_x,C_y)
  Points on the robot, Robot(C_x,C_y) = $\{(x,y) \mid (x-C_x)^2 + (y-C_y)^2 = r^2\}$
  Unbounded room, Q = $\mathbb{R}^2$
- Collision-free configuration, q_{free}
  - q is collision-free iff the robot does not collide with any obstacles when in configuration q, i.e. Robot(q) $\cap$ $(\cup_i Obstacle_i) = \emptyset$
  - Q_{free} = space of all collision-free configurations of the robot
- Path planning problem

Compute path in Q_{free} from q_{init} to q_{goal}

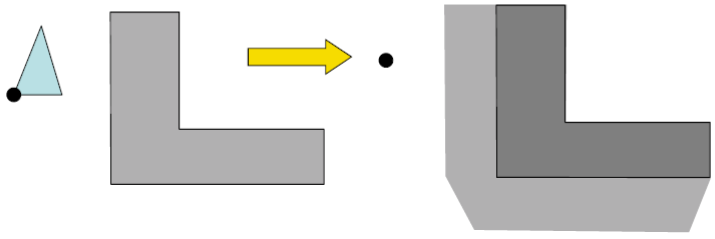
Configuration Space Transform

- A configuration is a set of parameters that define the robot geometry in the world
- Plan in Parameter Space! (Space Dimension = 2)
- **How does an obstacle look like in configuration space?**



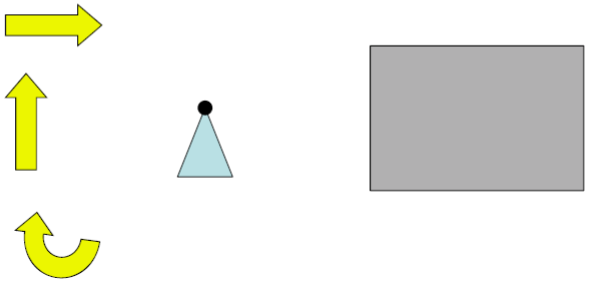
Configuration Space Transform

- A configuration is a set of parameters that define the robot geometry in the world
- Plan in Parameter Space! (Space Dimension = 2)
- **How does an obstacle look like in configuration space?**



Configuration Space Transform

- Configuration Space Transform with Rotation:
  - Configuration Space Dimensions?
    - Three: x, y, θ



Configuration Space Transform

Configuration Space Transform with Rotation (3D Configuration Space)

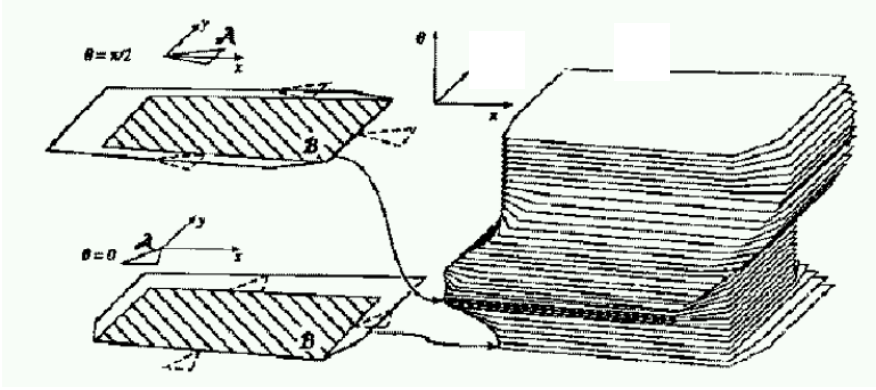


Figure from "Robot Motion Planning" by J.-C.Latombe

Minkowski Sum

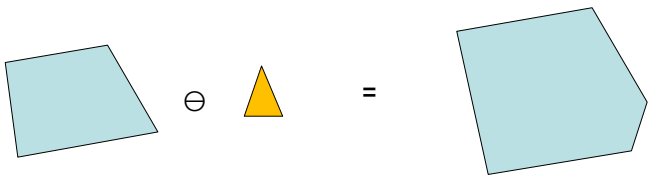
The Minkowski addition of two sets A and B

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

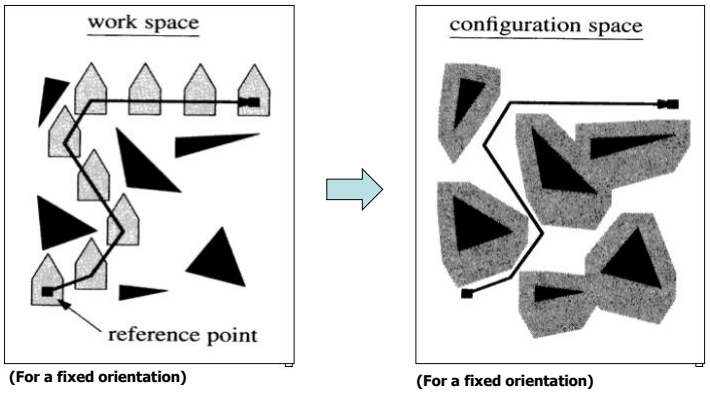
The Minkowski difference of two sets A and B

$$A \ominus B = \{a - b : a \in A, b \in B\}$$

$Q_{Obstacle}$ = {q : q ∈ Q and Robot(q) ∩ Obstacle ≠ ∅}
= set of all infeasible configurations due to obstacles
= Obstacle ⊖ Robot (for polygons and polyhedra)



Path Planning for Robots with Geometric Shapes

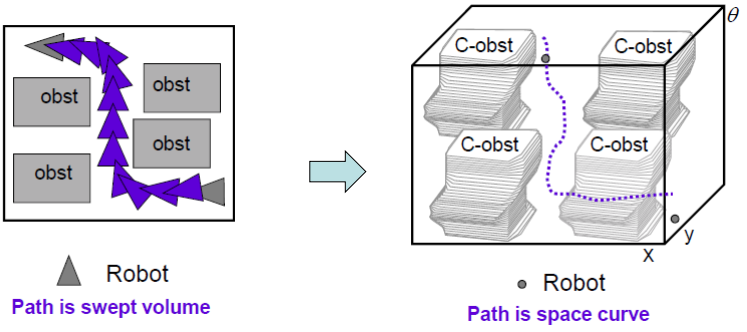


(For a fixed orientation) (For a fixed orientation)

Step 1: Reduce robot to a point in the configuration space
Step 2: Compute configuration-space obstacles (not a trivial job)
Step 3: Search for a path in the collision-free configuration space

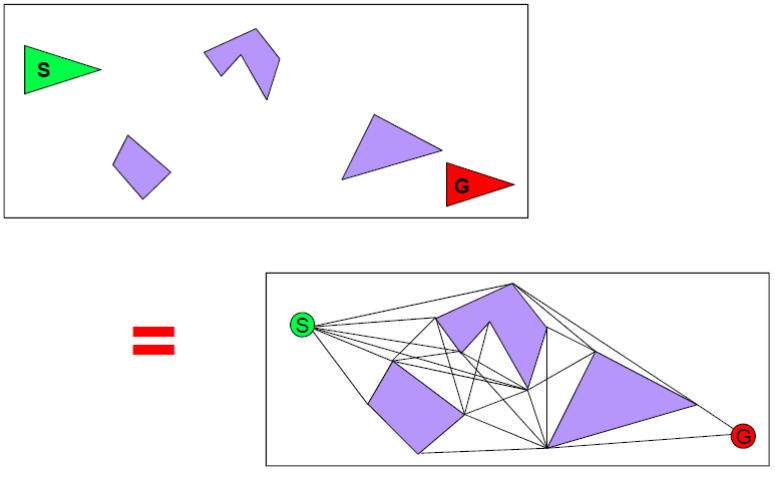
Figure from "Robot Motion Planning" by J.-C.Latombe

Path Planning for Robots with Geometric Shapes



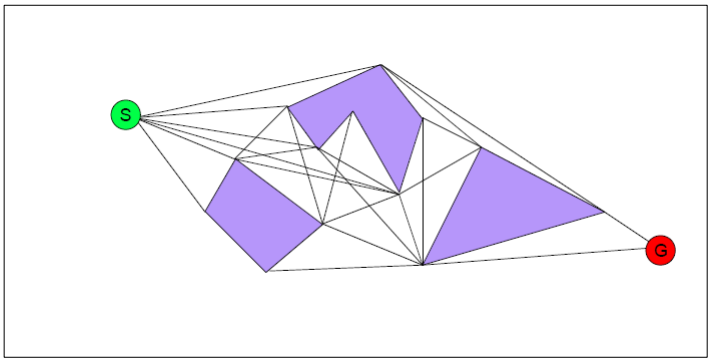
Step 1: Reduce robot to a point in the configuration space
Step 2: Compute configuration-space obstacles (not a trivial job)
Step 3: Search for a path in the collision-free configuration space

Problem1: Solved*

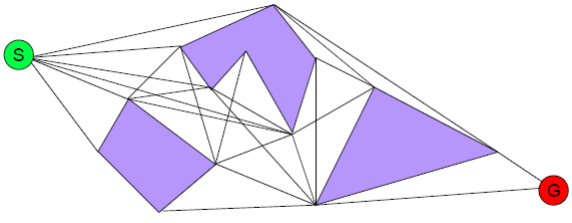


***even though it is difficult to do in general**

Problem2



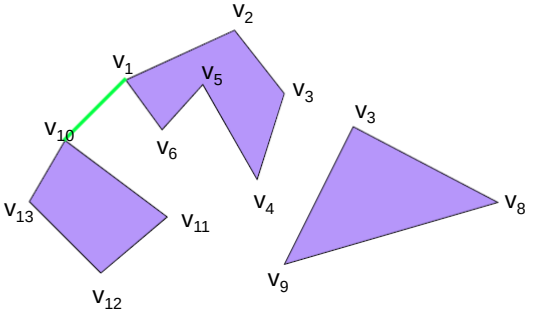
How to Build a Visibility Graph?



Visibility Graph Construction

Method 1

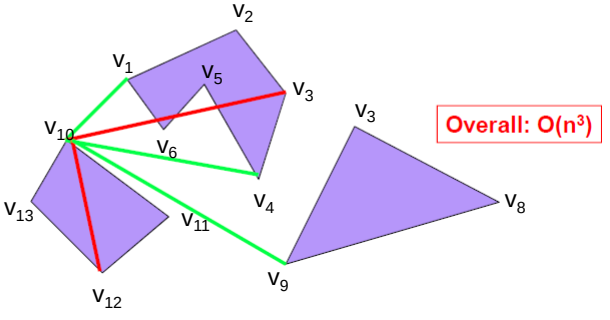
- Set of vertices $\{v_1, \dots, v_n\}$
- For each vertex v_i:
  For each vertex v_j, test whether edge (v_i, v_j) is collision-free



Visibility Graph Construction

Method 1 (Naïve)

- Set of vertices $\{v_1, \dots, v_n\}$
- For each vertex v_i:
  For each vertex v_j, test whether edge (v_i, v_j) is collision-free:
  (Collision test = for each edge *e* check if (v_i, v_j) intersects *e*)

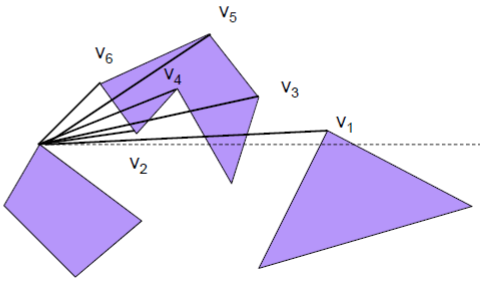


Visibility Graph Construction

Method 2 Sweep method

For each vertex v_i:

1) Sort vertices by angle

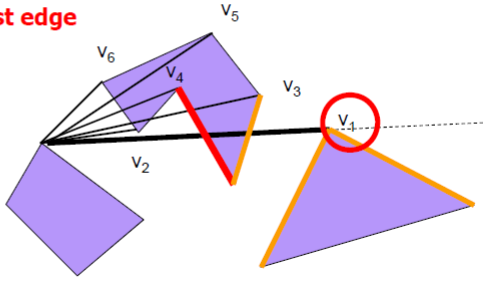


Visibility Graph Construction

Method 2

For each vertex v_i:

1) Sort vertices by angle
2) Test visibility in order of angle:
   Keep a list of sorted active edges
   Check if vertex on first edge

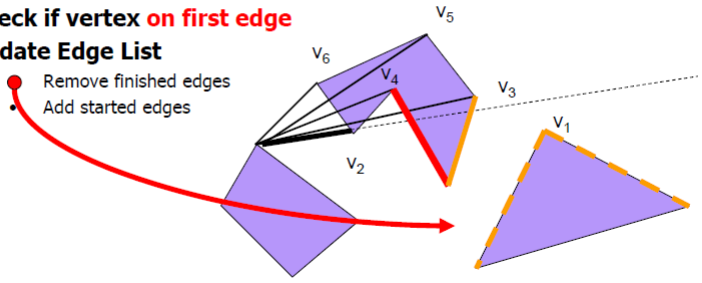


Visibility Graph Construction

Method 2

For each vertex v_i:

1) Sort vertices by angle
2) Test visibility in order of angle:
   - **Keep a list of sorted active edges**
   - **Check if vertex on first edge**
   - **Update Edge List**
     - Remove finished edges
     - Add started edges

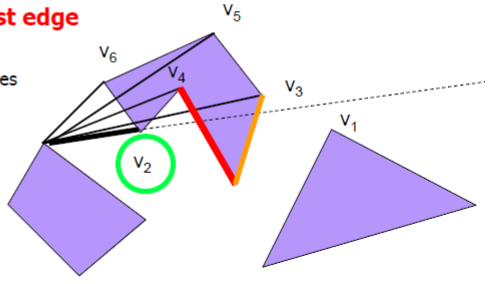


Visibility Graph Construction

Method 2

For each vertex v_i:

1) Sort vertices by angle
2) Test visibility in order of angle:
   - **Keep a list of sorted active edges**
   - **Check if vertex on first edge**
   - **Update Edge List**
     - Remove finished edges
     - Add started edges

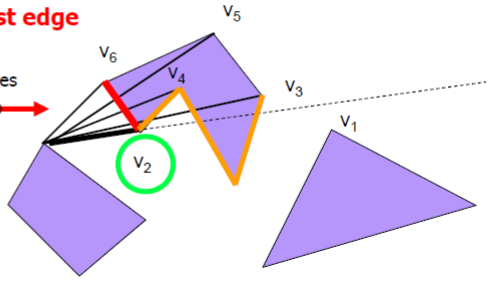


Visibility Graph Construction

Method 2

For each vertex v_i:

1) Sort vertices by angle
2) Test visibility in order of angle:
   - **Keep a list of sorted active edges**
   - **Check if vertex on first edge**
   - **Update Edge List**
     - Remove finished edges
     - Add started edges

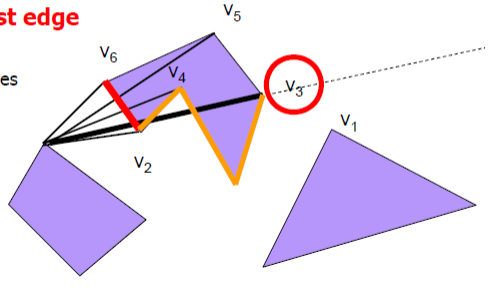


Visibility Graph Construction

Method 2

For each vertex v_i:

1) Sort vertices by angle
2) Test visibility in order of angle:
   - **Keep a list of sorted active edges**
   - **Check if vertex on first edge**
   - **Update Edge List**
     - Remove finished edges
     - Add started edges

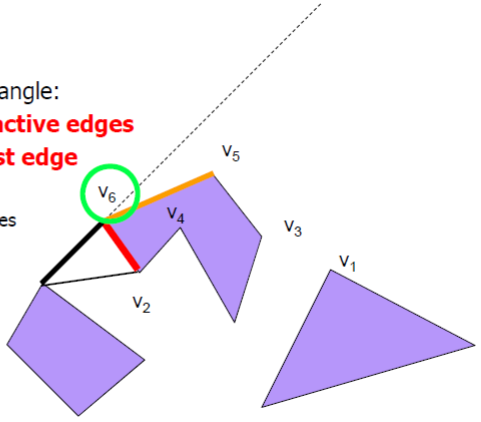


Visibility Graph Construction

Method 2

For each vertex v_i:

1) Sort vertices by angle
2) Test visibility in order of angle:
   - **Keep a list of sorted active edges**
   - **Check if vertex on first edge**
   - **Update Edge List**
     - Remove finished edges
     - Add started edges



Visibility Graph Construction

Method 2 (Smarter bookkeeping)

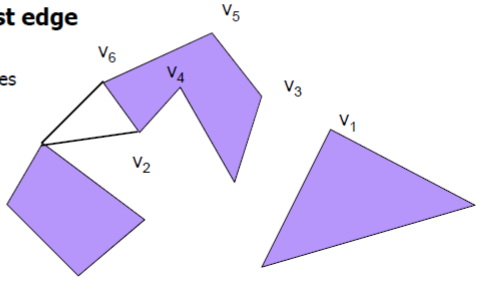
For each vertex v_i: ⟶ O(n)

1) Sort vertices by angle ⟶ O(n log n)

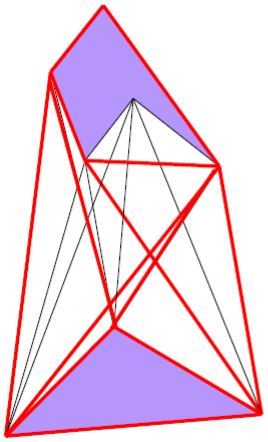
2) Test visibility in order of angle:
   - **Keep a list of sorted edges**
   - **Check if vertex on first edge**
   - **Update Edge List**
     - Remove finished edges
     - Add started edges

O(log n)

$O(n^2 \log n)$



Visibility Graph Reduction



Necessary edges:

Edge between Reflex Vertices

Reflex: Inner Angle > π

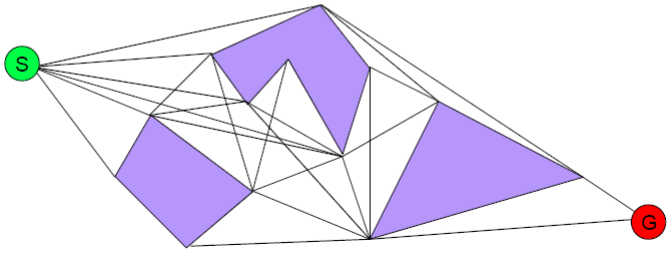
Bitangent Lines

Connects two Reflex Vertices

Does not "poke into interior"

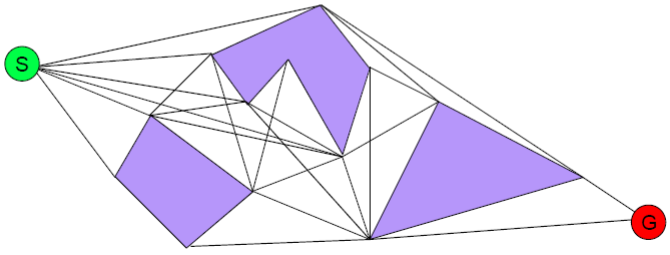
Visibility Graph Search

- Add the start and goal to the graph
- Search: Cost = Distance along edge



Visibility Graph Analysis

- Visibility Graphs are complete? Yes (Assuming Polygonal Obstacles)
- Visibility Graphs are optimal? Yes Metric: Distance Traveled



Questions to Ask

- **Completeness – What are the assumptions?**
  - Bug Algorithms: Finite Obstacles in Bounded Space
  - Visibility Graph: **Polygonal Obstacles**
- **Optimality – What is the metric?**
  - Bug Algorithms: Not optimal, Lower/Upper bound
  - Visibility Graph: **Euclidian Distance Traveled**