Lecture 17: *Motion Planning I*

Scribe: *Duong Le, Keyu Han, Baiyu Huang, Seunghee Yoon*

1 Complete Motion Planning

1.1 Cell Decompose

1.2 Visibility Graph

The defining characteristics of a visibility map are that its nodes share an edge if they are within line of sight of each other, and that all points in the robot's free space are within line of sight of at least one node on the visibility map. This second statement implies that visibility maps, by definition, possess the properties of accessibility and departability. Connectivity must then be explicitly proved for each map for the structure to be a roadmap. In this section, we consider the simplest visibility map, called the visibility graph

1.2.1 Visibility Graph Definition

The standard visibility graph is defined in a two-dimensional polygonal configuration space (figure 2.1). The nodes v_i of the visibility graph include the start location, the goal location, and all the vertices of the configuration space obstacles. The graph edges $e_i j$ are straight-line segments that connect two line-of-sight nodes v_i and v_j, i.e.,

$$e_i j \neq \emptyset \iff sv_i + (1-s)v_i \in cl(Q_{free}) \forall s \in [0,1]$$

Note that we are embedding the nodes and edges in the free space and that edges of the polygonal obstacles also serve as edges in the visibility graph.

By definition, the visibility graph has the properties of accessibility and departability. We leave it to the reader as an exercise to prove the visibility graph is connected in a connected component of free space. Using the standard two-norm (Euclidean distance), the visibility graph can be searched for the shortest path . The visibility graph can be defined for a three dimensional configuration space populated with polyhedral obstacles, but it does not necessarily contain the shortest paths in such a space.

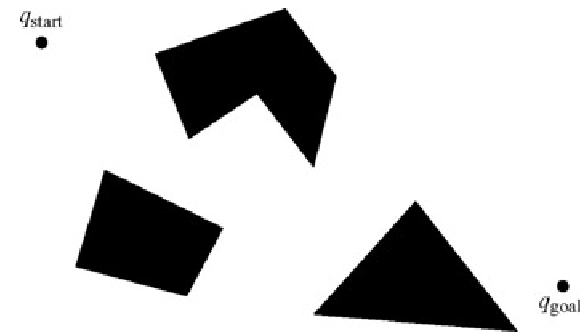

Figure 1: Polygonal configuration space with a start and goal

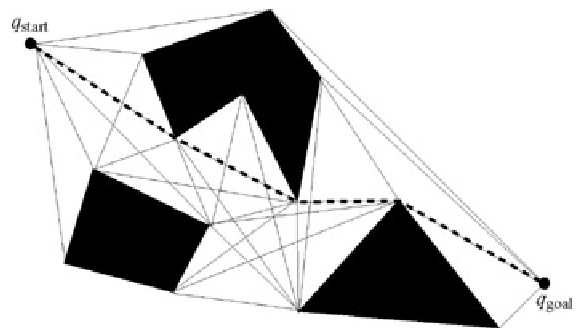

Figure 2: The thin solid lines delineate the edges of the visibility graph for the three obstacles represented as filled polygons. The thick dotted line represents the shortest path between the start and goal

Unfortunately, the visibility graph has many needless edges. The use of supporting and separating lines can reduce the number of edges. A supporting line is tangent to two obstacles such that both obstacles lie on the same side of the line. For nonsmooth obstacles, such as polygons, a supporting line l can be tangent at a vertex v_i if $\beta_\epsilon(v_i) \bigcap l \bigcap QO_i = v_i$. A separating line is tangent to two obstacles such that the obstacles lie on opposite sides of the line.

The reduced visibility graph is soley constructed from supporting and separating lines. In other words, all edges of the original visibility graph that do not lie on a supporting or separating line are removed.

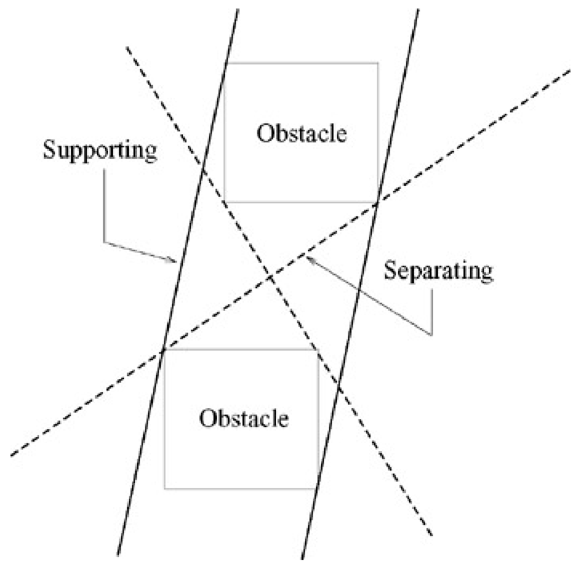

Figure 3: Supporting and separating line segments. Note that for polygonal obstacles, we use a nonsmooth notion of tangency

1.2.2 Visibility Graph Construction

Let $V = v_1, ..., v_n$ be the set of vertices of the polygons in the configuration space as well as the start and goal configurations. To construct the visibility graph, for each be the set of vertices of the polygons in the configuration space as well as the start and goal configurations. To construct the visibility graph, for each $v \in V$ we must determine which other vertices are visible to v.The most obvious wayto make this determination is to test all line segments $vv_i, v \neq v_i$to see if they intersect an edge of any polygon. For a particular vv_i, there are $O(n)$ intersections to check because there are $O(n)$edges from the obstacles. Now, there are $O(n)$ potential segments emanating from v,so for a particular v, there are $O(n_2)$ tests to determine which vertices are indeed visible from v, This must be done for all $v \in V$ and thus the construction of the visibility graph would have complexity $O(n_3)$

There is a more efficient way to compute the set of vertices that are visible from v. Imagine a rotating beam of light emanating from a lighthouse beacon. At any moment, the beam illuminates the object that is closest to the lighthouse. Furthermore, as the beam rotates, the obstacle that is illuminated changes only at a finite number of orientations of the beam. If the obstacles in the space are polygons, these orientations occur when the beam is incident on a vertex of some polygon. This insight motivates a class of algorithms known in the computational geometry literature as plane sweep algorithms.

A plane sweep algorithm solves a problem by sweeping a line, called the sweep line,

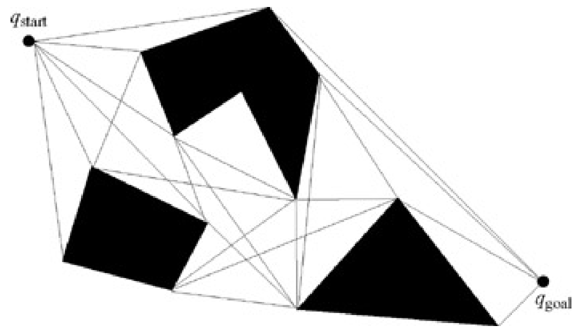

Figure 4: Reduced visibility graph

across the plane, pausing at each of the vertices of the obstacles. At each vertex, the algorithm updates a partial solution to the problem. Plane sweep algorithms are used to efficiently compute the intersections of a set of line segments in the plane, to compute intersections of polygons, and to solve many other computational geometry problems.
For the problem of computing the set of vertices visible from$v$, we will let the sweep line, l, be a half-line emanating from v,and we will use a rotational sweep, rotating l from 0 to 2π. The key to this algorithm is to incrementally maintain the set of edges that intersect l , sorted in order of increasing distance from v. If a vertex v_i is visible to v, then it should be added to the visibility graph.It is straightforward to determine if v_i is visible to v (see figure 5). Let S be the sorted list of edges that intersects the half-line emanating from v; the set S is incrementally constructed as the algorithm runs. If the line segment vv_i does not intersect the closest edge in S, and if$l$ does not lie between the two edges incident on v,(the sweep line does not intersect the interior of the obstacle at v), then the v_i is visible to v.

Algorithm 5: Rotational Plane Sweep Algorithm

```
Input: A set of vertices {v_i} (whose edges do not intersect) and a vertex v
Output: A subset of vertices from {v_i} that are within line of sight of v
For each vertex v_i, calculate α_i, the angle from the horizontal axis to the line segment
vv_i.
Create the vertex list ε, containing the α_i 's sorted in increasing order.
Create the active list S, containing the sorted list of edges that intersect the horizontal
half-line emanating from v.
for all α_i do
  if v_i is visible to v then
     Add the edge (v, v_i )to the visibility graph.
  end if
  if v_i is the beginning of an edge, E, not in S then
     Insert the E into S.
  end if
  if v_i is the end of an edge in S then
     Delete the edge from S.
  end if
end for
```

Figure 5: Rotational Plane Sweep Algorithm

2 Sampling-based Motion Planning

Sampling-based motion planning is not complete: not guarantee to find a solution when one exists. However, as the number of samples goes to infinity, there is a strong guarantee that it can find a solution.
There are 2 types of sample-based planning algorithms:

Multiple Query Algorithm

- Roadmap is built beforehand. Then it can be used multiple times for different queries.
- Query is fast since roadmap is already generated.
- Keeping roadmap all the time is expensive.
- Can only deal with static environment - has to rebuild the roadmap when environment changed.

Single Query Algorithm

- No roadmap is built beforehand. Find a path from start to goal then finish
- Query is slower.
- No saved roadmap - better memory
- Can deal with dynamic environment

2.1 Probabilistic Road Map (PRM)

Probabilistic Roadmap (PRM) is a multi-query algorithm.
There are 2 steps:

1. Preprocessing state: build roadmap
2. Query state: search roadmap given start and goal

2.1.1 Roadmap Construction

Algorithm 6 Roadmap Construction Algorithm

Input:
n : number of nodes to put in the roadmap
k : number of closest neighbors to examine for each configuration
Output:
A roadmap $G = (V, E)$

```
V ← Ø
E ← Ø
while |V| < n do
    repeat
        q ← a random configuration in Q
    until q is collision-free
    V ← V ∪ {q}
end while
for all q ∈ V do
    N_q ← the k closest neighbors of q chosen from V according to dist
    for all q' ∈ N_q do
        if (q, q') ∉ E and Δ(q, q') ≠ NIL then
            E ← E ∪ {(q, q')}
        end if
    end for
end for
```

Figure 6: Roadmap Construction Algorithm

Let $\Delta(q, q')$ be a function that returns either a collision-free path from q to q′ or NIL if it cannot find such a path. There are many algorithms can be used for Δ. For example, if we want to check whether there is a straight-line path from q to q, we can use binary search to check for collision.
Figure 6 shows an algorithm to construct a roadmap:

- Line (1) and (2): Initialize graph $G = (V, E)$ to be empty.
- Line (3) to (8): describes sampling process: keep sampling to get n numbers of collision-free samples.
- Line (9) to end: describes process to build roadmap from n samples: for every sample node $q \in V$, create a set N_q of k closest neighbors. Then from every $q' \in N_q$, whenever function $\Delta(q, q')$ succeeds to compute a collision-free path from q to q', the edge (q, q') is added to E. Figure 7 shows an example of roadmap using Δ as straight-line planner

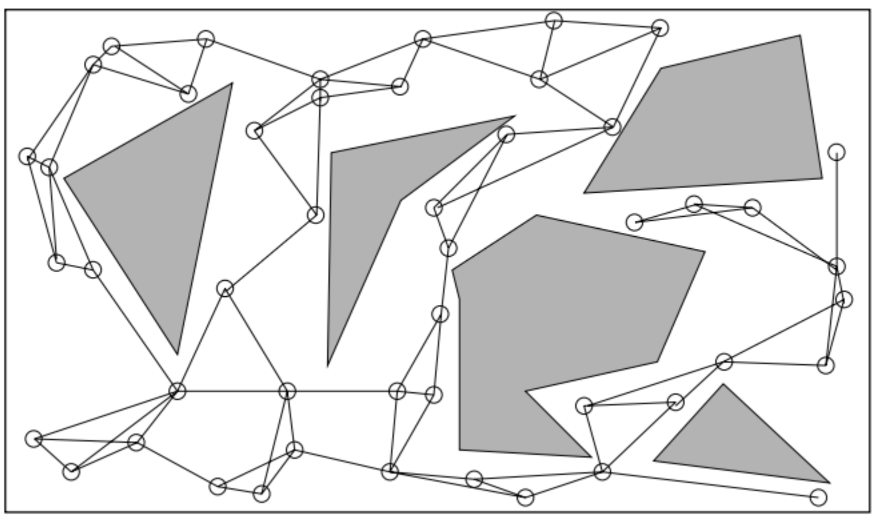

Figure 7: An example of a roadmap

2.1.2 Query

When you have a start point and end point, you connect them to the graph and then using graph searching algorithm to find path from start to end.

2.1.3 Failure

Some reasons to failure:

- Connectivity: disconnected graph

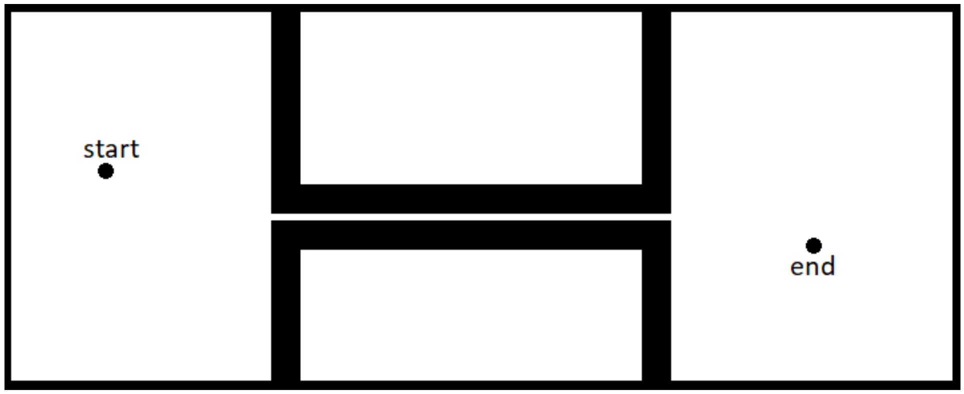

Figure 8: An example when obstacles are very closed to each other

- Obstacles are very closed to the others.

Solution: generate more samples.

But in the case when obstacles are closed to the others, the probability to sample between the obstacles is very unlikely. For example, in figure 8, the chance to sample points in the narrow road between 2 obstacles is very low. If you keep sampling more points, the graph will become so dense and will take a lot of time to find a path between start and end point.
Solution: sample more points toward edges of the obstacles. We can use binary search to find the closest point to the obstacle that's collision-free, and then sample more points near that point. The process of sampling becomes more expensive, but result graph is less dense and faster to compute path.

2.2 Visibility PRM

The roadmap is constructed incrementally by randomly sampling the configuration space and attempting to connect some pairs of collition-free samples by the local method. The visibility roadmaps are build without any explicit computation of the visibility domains.

2.2.1 Principle

The algorithm that we propose below is general. It allows us to build visibility roadmaps without requiring any explicit computation of the visibility domains. The roadmap is constructed incrementally by randomly sampling the conguration space and attempting to connect some pairs of collision-free samples by the local method. Figure 9 illustrates the principle of the sampling strategy used at each iteration of the algorithm. Randomly

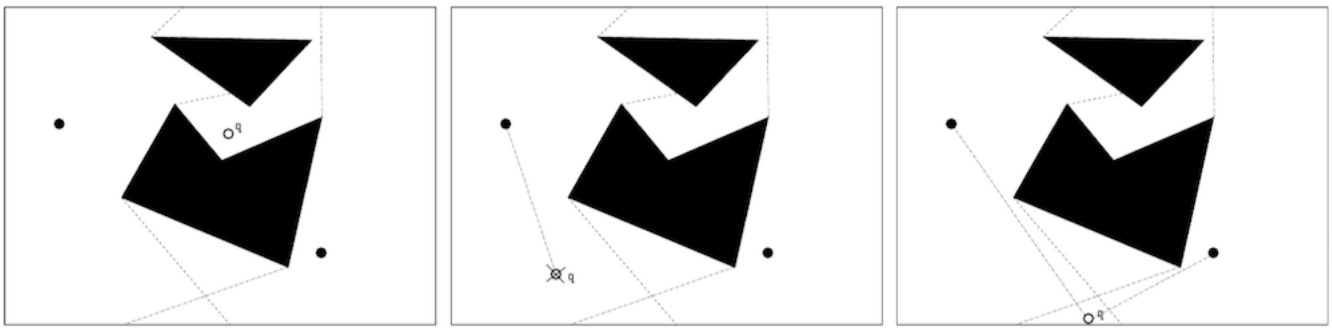

Figure 9: node added as a guard; node rejected; connection node merging two connected components

chosen congurations are checked for collision to generate samples in CS_free; when a free sample is found, it is added to the roadmap either if it does not 'see' any another node of the current roadmap (i.e. it is a new guard) or if it is seen by at least two nodes belonging to two distinct connected components of the roadmap (i.e. it is a connection node). The end of the roadmap's construction is controlled by a termination condition related to the volume of free space currently covered by the roadmap.

2.2.2 Guard and connection node

When a free sample is found, it is added to the roadmap in two cases:

- If it does not "see" another node already in R . This will be a new guard.
- If it is seen at leas by two nodes belonging to two distint connected components of will be a connection node.

2.2.3 Algorithm

The algorithm, called Visib-PRM, iteratively processes two sets of nodes: Guard and Connection. The nodes of Guard belonging to a same connected component (i.e. connected by nodes of Connection) are gathered in subsets G_i.

At each elementary iteration, the algorithm randomly selects a collision-free configuration q. The main loop processes all the current components G_i of Guard. The algorithm loops over the nodes g in G_i , until it finds a node visible from q. The first time the algorithm succeeds in finding such a visible node g, it memorizes both g and its component G_i, and switches to the next component $G_i + 1$. When q 'sees' another guard g 0 in another component G_j , the algorithm adds q to the Connection set and the component G_j is merged with the memorized G_i. If q is not visible from any component, it is added to the Guard set. The main loop fails to create a new node when q is visible from only one component; in that case q is rejected. Parameter ntry is the number of failures before

$Guard \leftarrow \emptyset$; $Connection \leftarrow \emptyset$; $ntry \leftarrow 0$
while $(ntry < M)$
  Select a random free configuration q
  $g_{vis} \leftarrow \emptyset$; $G_{vis} \leftarrow \emptyset$
  for all $G_i \in Guard$ do
    $found \leftarrow false$
    for all $g \in G_i$ do
      if $q \in Vis(g)$ then
        $found \leftarrow true$
        if $g_{vis} = \emptyset$ then $g_{vis} \leftarrow g$; $G_{vis} \leftarrow G_i$
        else $Connection \leftarrow Connection \cup \{q\}$; Create (g, q) and (q, g_{vis}); Merge G_{vis} and G_i
    until $found = true$
    if $g_{vis} = \emptyset$ then $Guard \leftarrow Guard \cup \{q\}$; $ntry \leftarrow 0$
    else $ntry \leftarrow ntry + 1$
end

Figure 10: Visibility PSM Algorithm

the insertion of a new guard node. 1/ntry gives an estimation of the volume not yet covered by visibility domains. It estimates the fraction between the non-covered volume and the total volume of CS_free. This is a critical parameter which controls the end of the algorithm. Hence, the algorithm stops when ntry becomes greater than a user set value M, which means that the volume of the free space covered by visibility domains becomes probably greater than (1-1/M).

2.2.4 Failure

The random generation of the guards may produce in some cases guards that will be difficult to connect. This effect is illustrated in Fig. 11 where two guards have been generated near the boundary of the black triangular obstacle. They fully cover CS_free, however the intersection of both visibility domains is 'unfortunately' small. The only way to complete the roadmap is to pick a connection node in the small triangle. Then the algorithm will fail if the parameter M is not sufficiently high.Nevertheless this case is only a side-effect of the algorithm. Indeed, in this example, the probability to select the first two guards with a small intersection domain is very low. Moreover, this undesirable

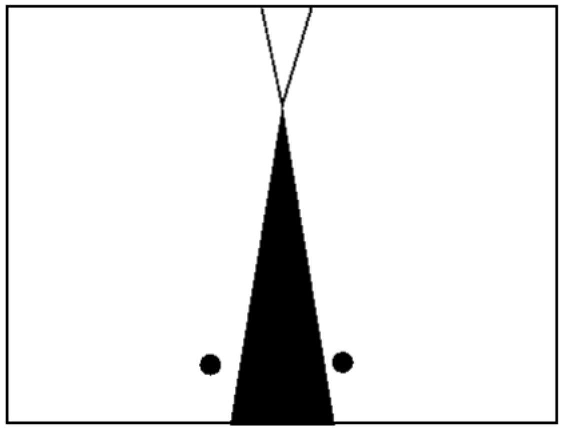

Figure 11: Visibility PSM Failure

effect was never observed in practice in all the examples we experimented on with the algorithm.