Cell Decomposition

⇒ Here we consider a different type of representation of the free space called an exact cell decomposition.

These structures represents the free space by the union of simple regions called cells.

→ The shared boundaries of cells often have a physical meaning.

⇒ Two cells are adjacent if they share a common boundary.

⇒ An adjacency graph as its name suggests, encodes the adjacency relationships of the cells.

→ node corresponds to a cell and an edge connects nodes to adjacent cell.

⇒ Assuming the decomposition is computed, Path planning with a cell decomposition is usually done in two steps:

① The planner determines the cells that contains the start & goal respectively.

② Then the planner searches for a path within the adjency graph.

⇒ Note: Adjacency graph could serve as a roadmap of the free space as well.

⇒ Cell decompositions, however, distinguish themselves from other methods in that they can be used to achieve coverage.

A coverage path planner determines a path that passes an effector (e.g. robot) over all points in a free space.

⇒ Since each cell has a simple structure, each cell can be covered with simple motions.
(back-and-forth forming maneuvers)

→ Once the robot visits each cell, coverage is achieved.

→ So coverage can be reduced to finding an exhaustive walk through the adjacency graph.

⇒ Sensor-based coverage is achieved by simultaneously covering an unknown space & constructing its adjacency graph.

⇒ The most popular cell decomposition is the trapezoidal decomposition.

→ Heavily relies on the polygonal representation of the planar configuration space.

⇒ A more general class of decompositions, which are termed (Moorse Decompositions)

↓

{Allow for representation of non polygonal & nonplanar space}

⇒ Moorse decomposition are based on ideas from Canny's roadmap work.

⇒ We then consider a broader class of decompositions which includes those based on visibility constraints.

↳ One such decomposition serves as a basis for the pursuit/evasion problem.

★ Trapezoidal Decomposition

⇒ The trapezoidal decomposition comprises two-dimensional cells that are shaped like trapezoids.

↳ Some cells can be shaped like triangles.

↳ Can be views as degenerate trapezoids where one of the parallel sides has a zero-length edge.

$V_i \Rightarrow$ Vertex of the polygons

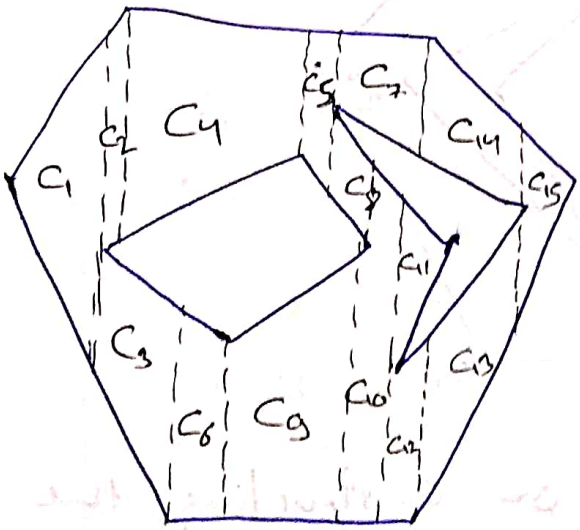
$i \neq j \quad V_{ix} \neq V_{iy}$ (assumption)

$\Rightarrow$ To form the decomposition, at each vertex V draw two segments

- One called an upper vertical extension.
- Other called a lower vertical extension.

{down corresponds to ↓ y coordinate} {Up corresponds to ↑ y coordinate}

$\Rightarrow$ The upper and lower vertical extension start at the vertex and terminate when they first intersect an edge of the polygon.



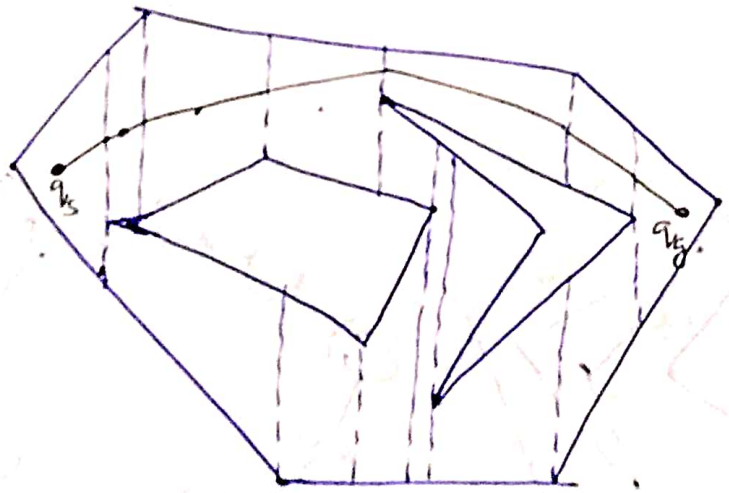
Adjacency graph

Trapezoidal decomposition

⇒ Once the cells that contain the start & goal are determined, the planner searches the adjacency graph to determine the path.

- → However, the result of the graph search is just a sequence of nodes, not a sequence of points embedded in the free space.
- → Next step is to determine the explicit path

⇒ Since a trapezoid is a convex set, any two points on the boundary of a trapezoidal cell can be connected by a straight line segment that do not intersect any obstacle.



⇒ The next issue centers on constructing the decomposition itself.

⇒ Input: → list of polygons
  → {Each represented by a list of vertices}

⇒ First Step: Sort the vertices based on the x-coordinate of each vertex.

  → This takes O(nlogn) time and O(n) storage where n is the number of edges (or vertices) in all of the polygons.

→ Next Step: Determine the vertical extensions.
  → A naive algorithm can intersect a line through V_i with edge e_j for all j.
    → This will require O(n) time resulting in $O(n^2)$ time to construct the trapezoidal decomposition.

⇒ We can do better: the extensions can be determined by sweeping a sweep line.
  → through the free space stopping at the vertices which are sometimes termed events.
  → While passing the sweep line, the planner can maintain a list L that contains the current edges which the slice intersects.

⇒ With the list L, determining the vertical extensions at each event requires $O(n)$ time with a simple search, but if the list is stored in a efficient data structure like a balanced tree, then the search requires $O(\log n)$ time.

⇒ It is easy to determine the y-coordinates of the intersection of the line that passes through V_i and and each edge e_i.

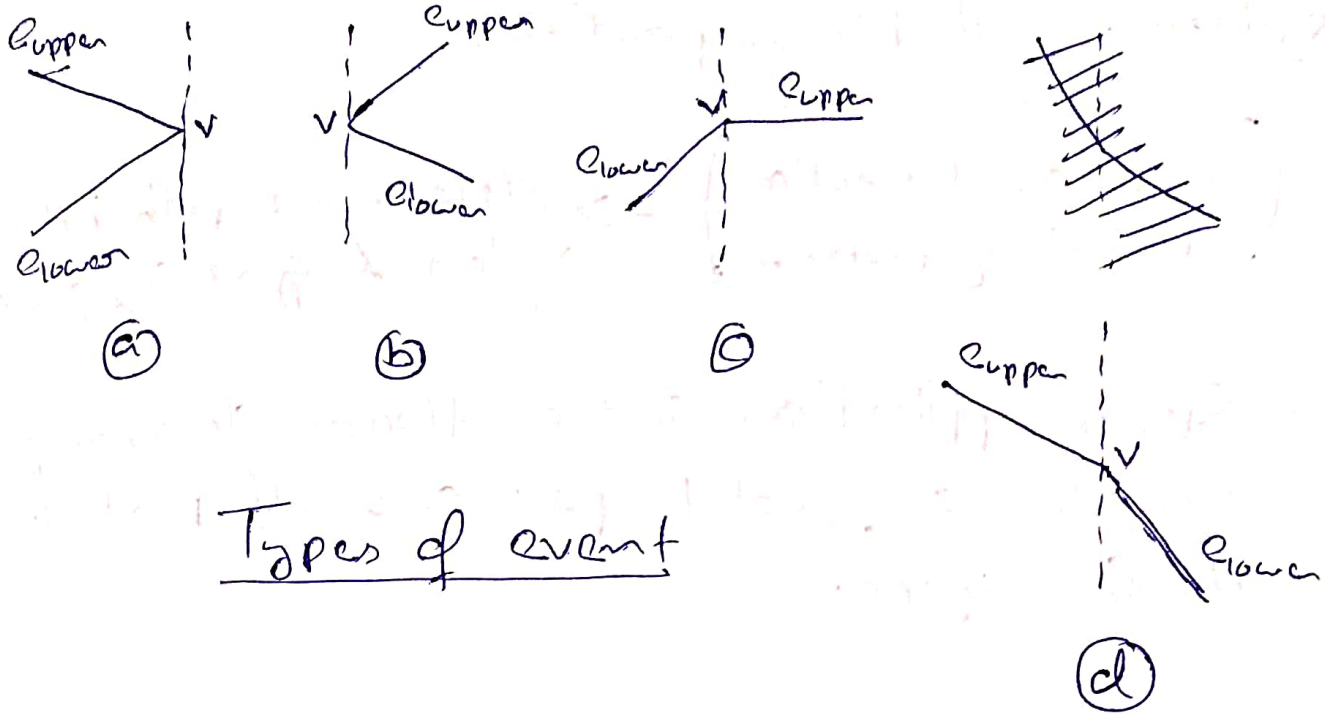
→ The trick is to find the appropriate edge or edges for the vertical extensions. (i.e. the two edges that V lies between)

→ Let these two edges be called e_{LOWER} and e_{UPPER}.

⇒ Let e_{lower} & e_{upper} be the two edges that contain V.

→ e_{lower} has a higher y-coordinate than the "other" vertex of e_{upper}.

{the other vertex of}



Types of event

(a) delete e_{lower} & e_{upper} from the list.

(b) Insert e_{lower} & e_{upper} into the list.

(c) delete e_{lower} from the list and insert e_{upper}

(d) delete e_{upper} from the list & insert e_{lower}.

⇒ Finally we need to determine which cells contain the start and goal.

⇒ First, we will seemingly construct a fine cell decomposition which will have cells that are subsets of the trapezoid and then infer

↳ from these which trapezoids contains the start & goal.

★ Morse Cell Decompositions

Conventional motion Planning problem → determine path between start & goal configuration

⇒ Some application such as floor cleaning requires a robot to pass over all points in its free space.

⇒ A planner can use an exact cell decomposition to cover an unknown space by simply covering each cell and then using the adjacency graph to ensure each cell is visited & hence covered.

⇒ We measure efficiency of such planner in terms of area covered Vs path length traversed.

⇒ Morse function

→ function whose critical points are nondegenerate; for a practical perspective

This means that critical points are isolated, and if a morse function is used for gradient descent any random perturbation will destabilize saddles & maxima.

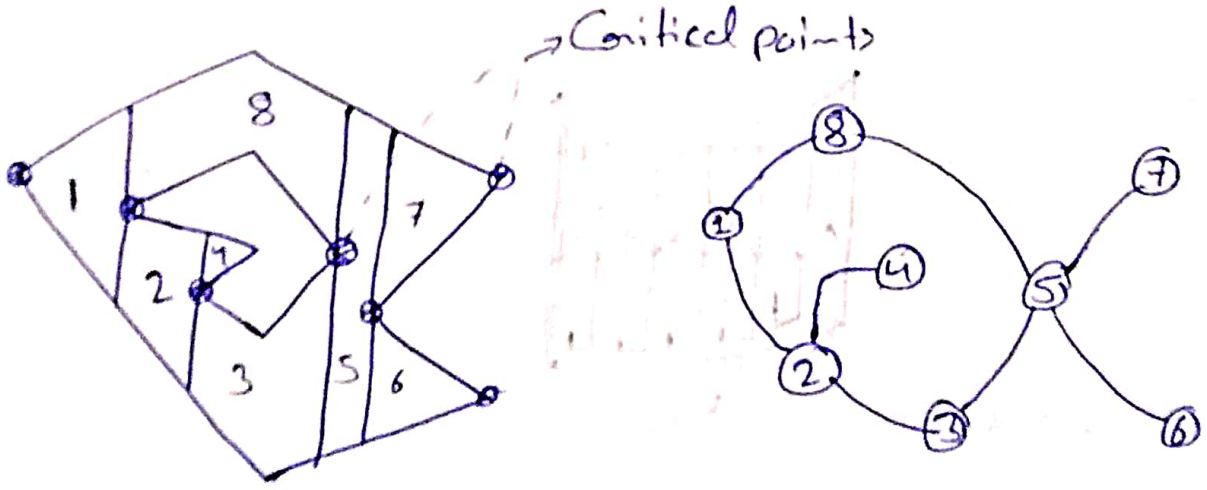
⇒ In this section, we evolve the trapezoidal decomposition to a new decomposition called the boustrophedon decomposition.

★ Boustrophedon Decomposition

⇒ From a coverage perspective, a minor shortcoming of the trapezoidal decomposition is that many small cells are formed that can seemingly be aggregated with neighboring cells.

↳ Reorganizing the cells can result in a shorter and more efficient path to cover the same area.

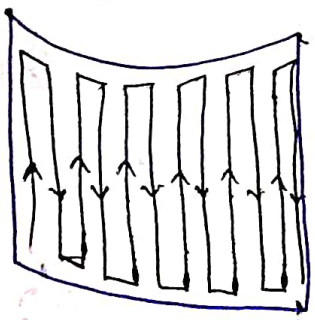
⇒ The boustrophedon decomposition is formed by considering the vertices at which a vertical line can be extended both up & down in the free space.



⇒ With the decomposition in hand, the planner determines a coverage path in two steps.

① First the planner determines an exhaustive walk through the adjacency graph.
   ↳ This list can be computed by using a depth-first search algorithm.

② Once the ordered list of cell is determined, the planner then computes the explicit robot motion within each cell.
   ↳ The path in each cell consists of a repeated sequence of straight-line segment separated by one robot width and short segments connecting the straight line-segments



★ Morse Decomposition Definition

⇒ We generalize the boustrophedon decomposition background polygons by borrowing ideas from Canny's work, which first applied a "slicing method" to motion planning.

⇒ Slice is a codimension one manifold denoted by Q_λ.

↳ The slices are parameterized by λ.

⇒ The portion of the slice in the free configuration space, Q_{free} is denoted by $Q_{free\lambda}$.

$$\boxed{Q_{free\lambda} = Q_\lambda \cap Q_{free}}$$

⇒ Slice can be defined in terms of the preimage of the projection operator

$$\pi_1 : Q \to \mathbb{R}$$

{On the plane $\pi_1(x,y) = x$ and the slice $Q_\lambda = \pi_1^{-1}(\lambda)$ corresponds to a vertical slice}

⇒ As the slice is swept through the target region obstacles intersect (or stop intersecting) the slice.

↳ Severing free space into smaller pieces as the slice first encounters an obstacle.

↳ Or merging smaller pieces into larger pieces as the slice immediately departs an obstacle.

⇒ The connectivity changes occurs at points termed <u>critical points</u>.

⇒ Slice that contain critical points are termed <u>critical slices</u>.

⇒ Naturally, Q_{free} contains one or more connected components, which are termed slice intervals and are denoted Q^j_{free}

⇒ So $\boxed{Q_{free} = \bigcup_j Q^j_{free}}$ { for the j^{th} open connected slice interval }

⇒ Denote the set of slice intervals that contain a critical points by I^*.

<u>Defination 6.2.1 (Morse Decomposition)</u>

"A Morse decomposition is an exact cell decomposition whose cells are the connected components of $Q_{free} \setminus I^*$."

⇒ Morse theory assures us that between critical slices, "merging" & "severing" of slices do not occur.

↳ This is useful for tasks such as Coverage because the robot can trivially perform simple motions between critical points & guarantee complete coverage of a cell.

⇒ A bulk of the coverage operation occurs with motions along the slices, sometime called <u>laps</u>.

⇒ We call the distance between subsequent laps as the inter-lap distance.

★ Examples of Morse Decomposition: Vanicble Slice

⇒ The definition of a Morse decomposition is not specific to a particular slice.

⇒ Lets rewrite the definition of the slice as the Preimage of a general real-valued function.

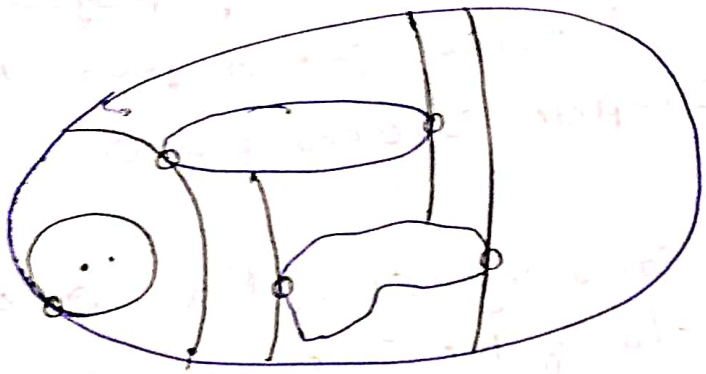
$$\boxed{h: Q \to R}$$

↓

{ For boustrophedon decomposition in the Plane $h(x,y) = x$ }

★ Spiral, Spike, and Squared Patterns

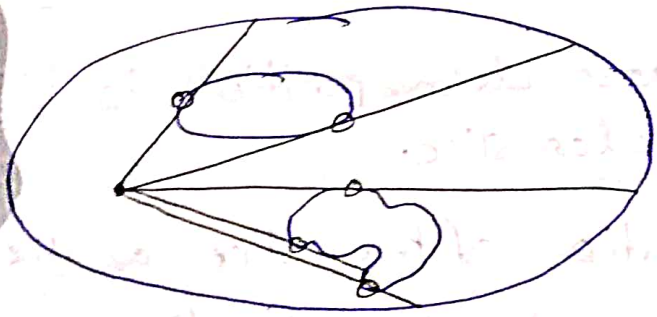
⇒ We can use the function $h(x,y) = \sqrt{x^2+y^2}$ to produce a pattern of concentric circles in the plane.



Spiral

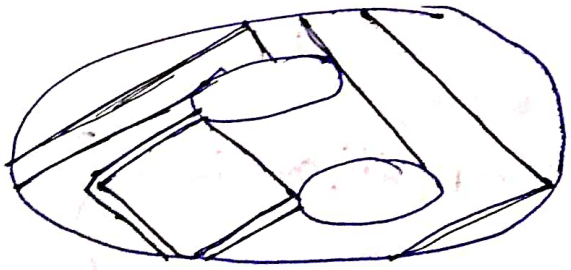
$h(x,y) = \sqrt{x^2+y^2}$

⇒ This yields a path that maximizes the area covered per unit distance traveled in regions sparsely populated with obstacles.



Spike

$h(x,y) = \tan\left(\frac{y}{x}\right)$



Squared

$h(x,y) = \|x\| + |y|$

★ Brushfire decomposition

⇒ The brushfire algorithm is a popular technique to construct the GVD.

⇒ The algorithm, however, induces a decomposition that is not the Voronoi regions of the GVD.

⇒ The distance function D, which measures the distance between the point x and the nearest point c on the closest obstacle QO_i admits a decomposition termed the brushfire decomposition.

⇒ Each slice of D is a wave front where each point on the front has propagated a distance λ from the closest obstacle.

- → As λ↑ the wave fronts progress.
- → Cells of the brushfire decomposition are formed when these wave fronts initially collide.

⇒ The pattern induced by the brushfire algorithm is ideally suited for coverage with mobile robots experiencing dead-reckoning error but have a large sensing range.

★ Wave-front Decomposition

⇒ Let $h(x,y)$ be the length of the shortest path between a point (x,y) and a fixed location.

⇒ The level sets $h^{-1}(\lambda)$ foliate the free space where for a given λ, the set of points in $h^{-1}(\lambda)$ are a distance λ away from the fixed point in the free space.

↳ This function is sometimes called wave-front potential.

⇒ Imagine a wave front starting at q_{start} & expanding into the free space.

↳ The value λ parameterizes each wave front.

⇒ Once the wave front crosses q_{goal}, the planner can backtrack a path from q_{goal} to q_{start}.

⇒ Critical points of this function occur both when wave fronts becomes tangent to obstacles & when wave fronts collide.

⇒ This decomposition is especially useful for coverage by a tethered robot where the robot's tether is incrementally fed and the robot sweeps out curves each of constant tether length.

★ Sensor-based Coverage

⇒ Lets place the robot in an unknown environment, but assume it has the standard range Sensor.

⇒ The task is to simultaneously cover and explore the unknown space.

↳ This can be reduced to concurrently and incrementally covering each cell while constructing the adjacency graph.

⇒ For sensor-based coverage, however, we incrementally construct a "dual" graph called a (Reeb graph)

↳ This graph is dual in the sense that the nodes of the Reeb graph are the Critical Points and the edges connect neighbouring Critical points.