Robotic Motion Planning: Cell Decompositions

(with some discussion on coverage and pursuer/evader)

Robotics Institute 16-735

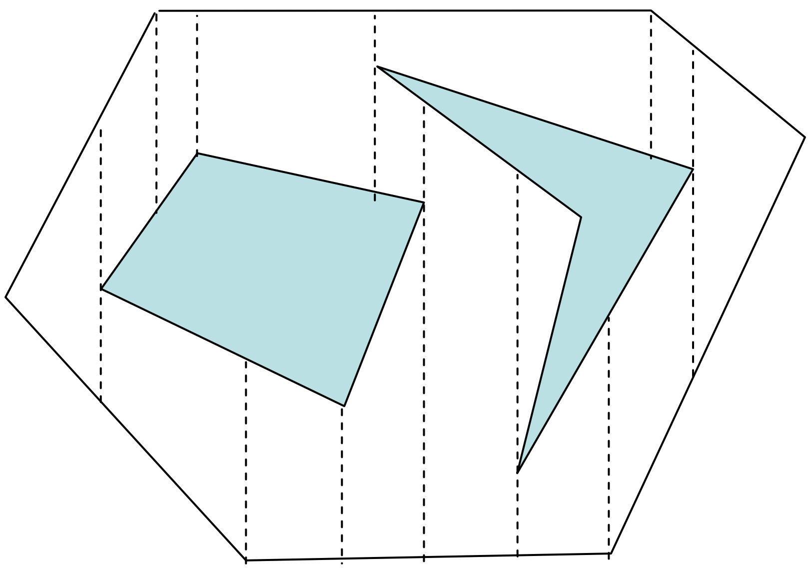
http://voronoi.sbp.ri.cmu.edu/~motion

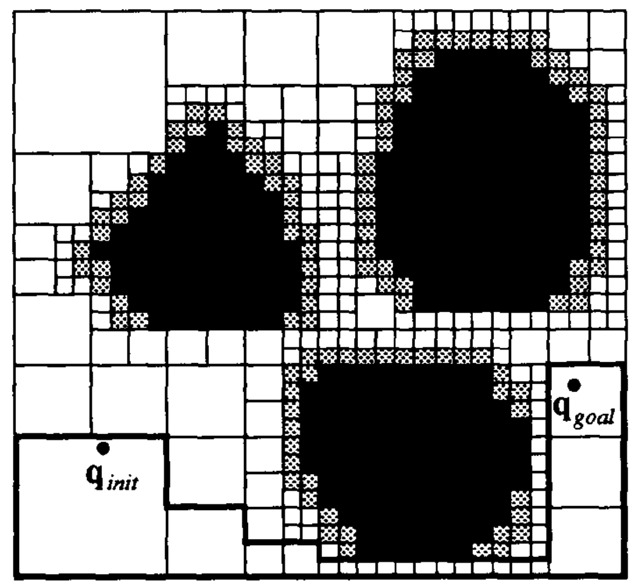
Howie Choset

http://voronoi.sbp.ri.cmu.edu/~choset

Exact Cell vs. Approximate Cell

- Cell: simple region



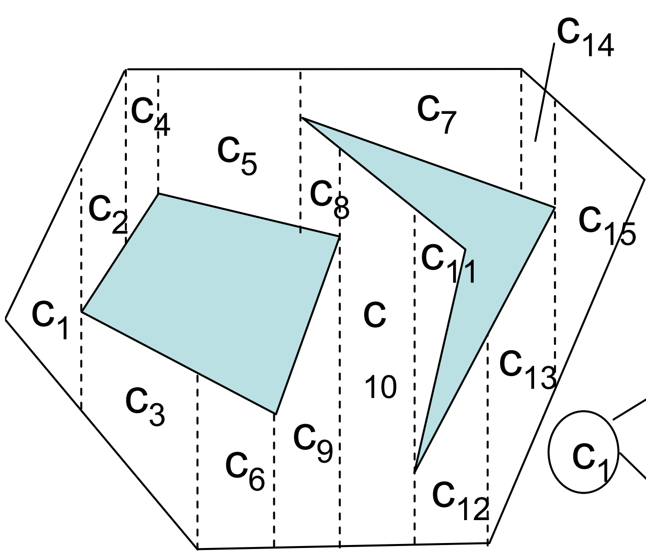


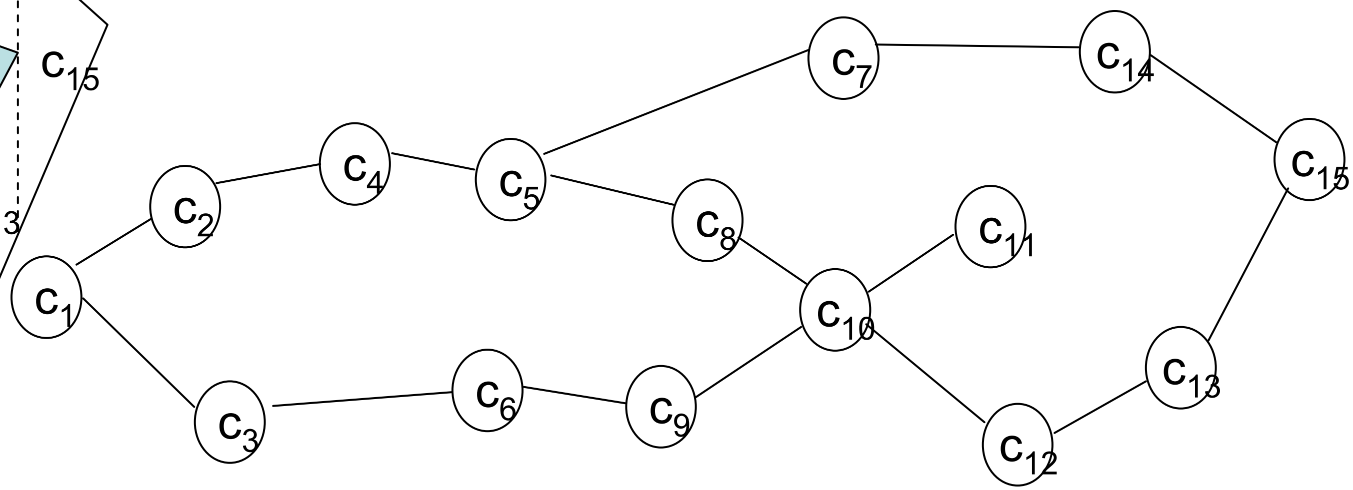
Adjacency Graph

– Node correspond to a cell

– Edge connects nodes of adjacent cells

- Two cells are *adjacent* if they share a common boundary





Path Planning

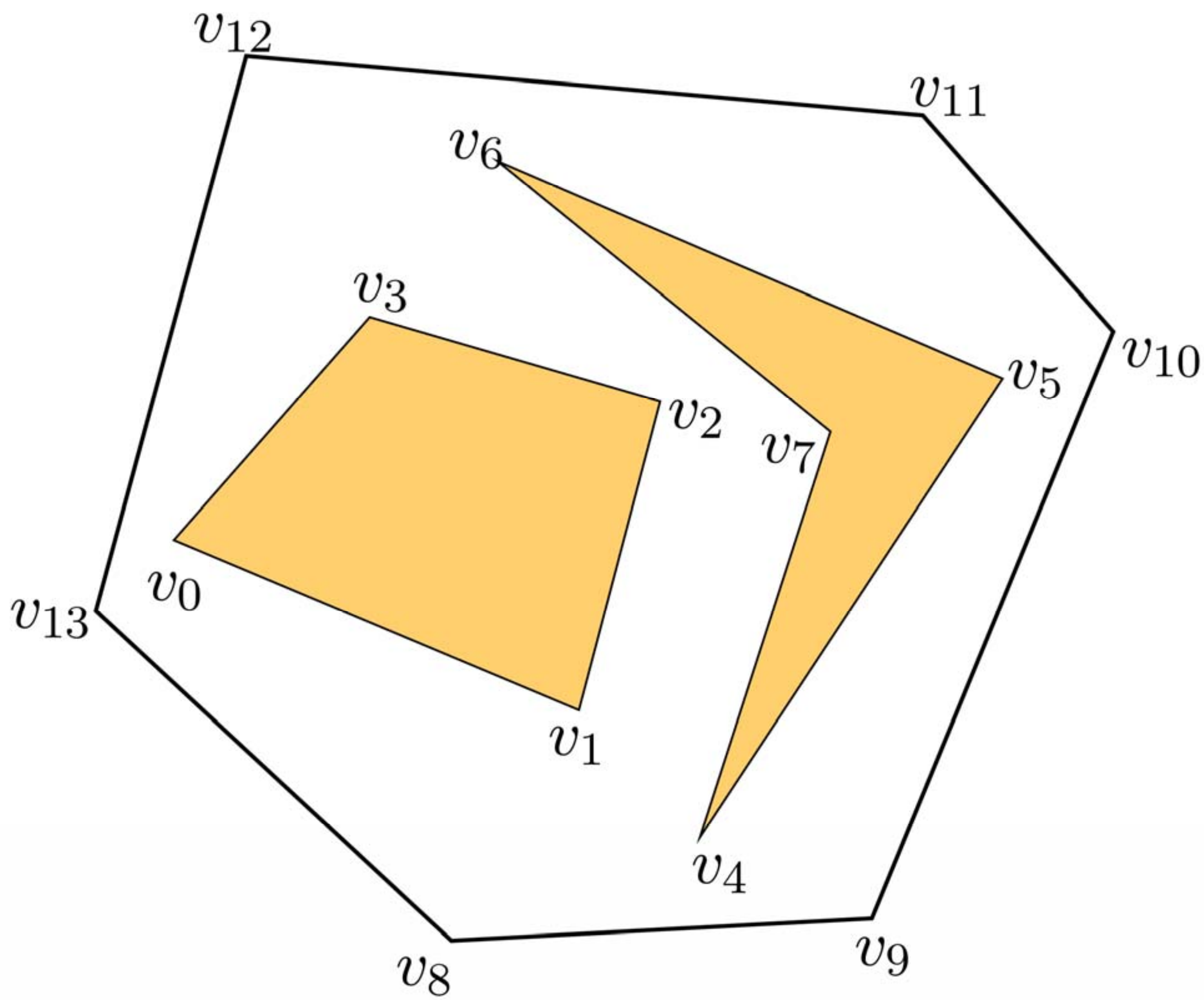
- Path Planning in two steps:
  - Planner determines cells that contain the start and goal
  - Planner searches for a path within adjacency graph

Types of Decompositions

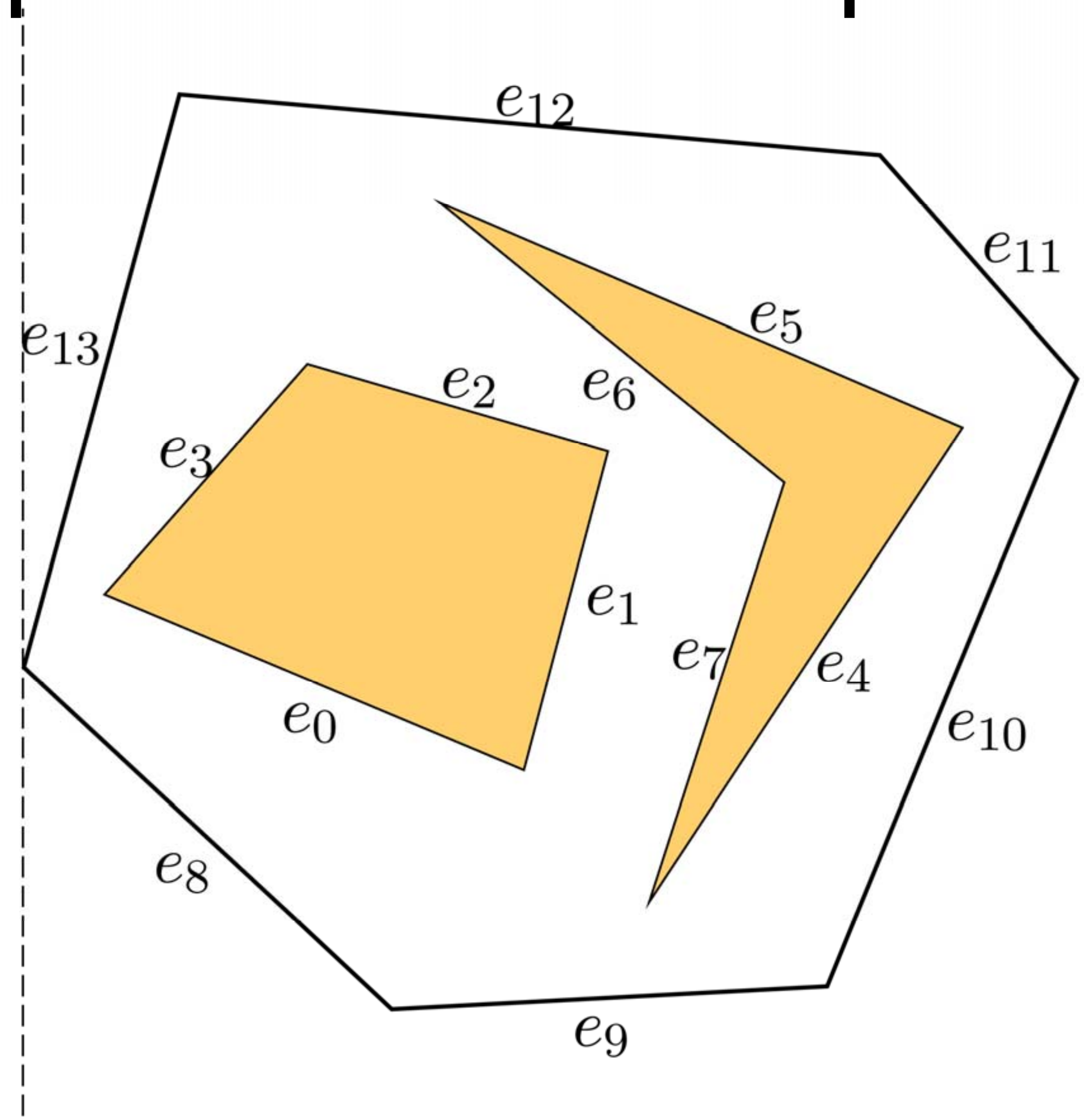
- Trapezoidal Decomposition
- Morse Cell Decomposition
  - Boustrophedon decomposition
  - Morse decomposition definition
  - Sensor-based coverage
  - Examples of Morse decomposition

- Visibility-based Decomposition

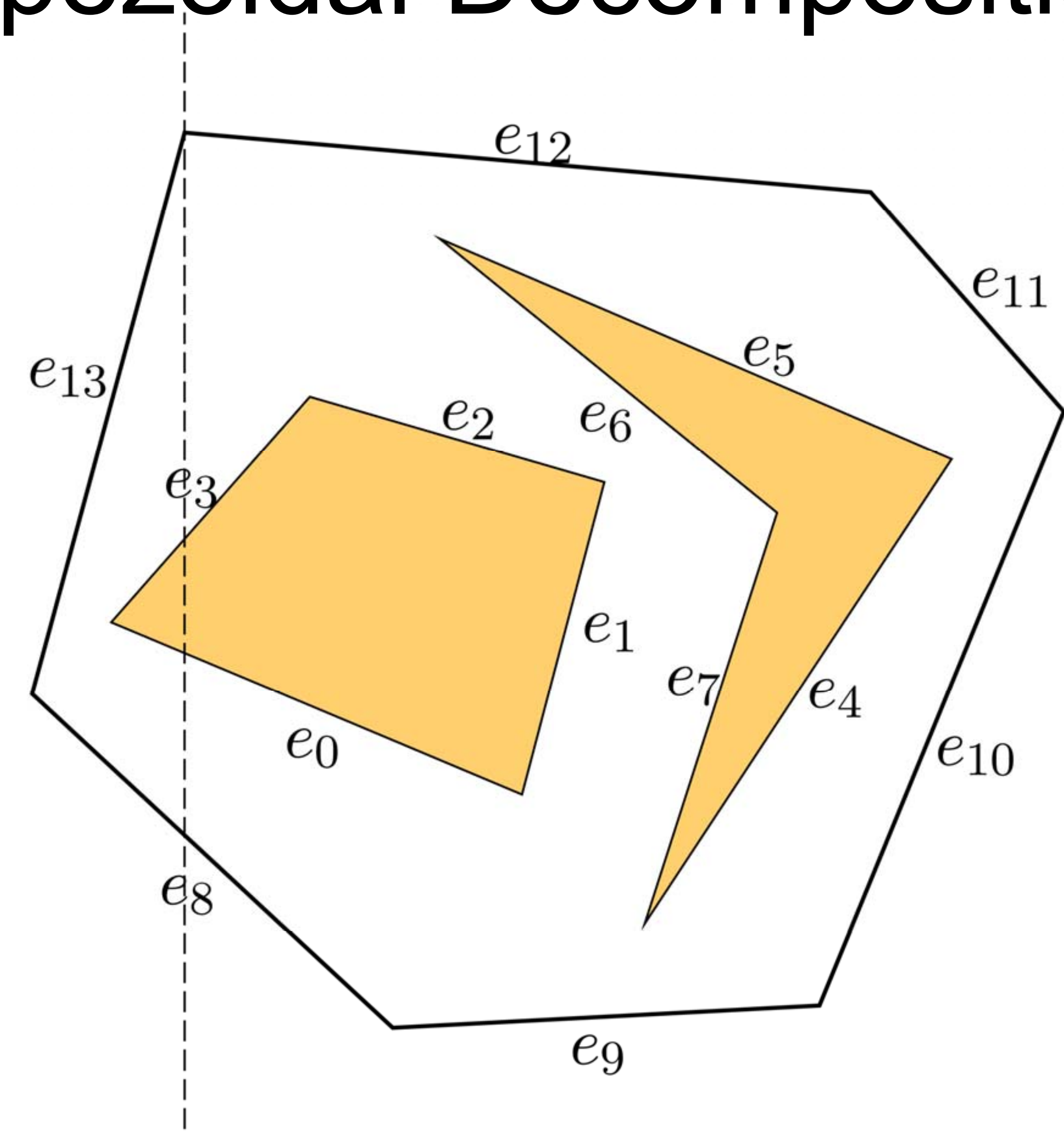
Trapezoidal Decomposition



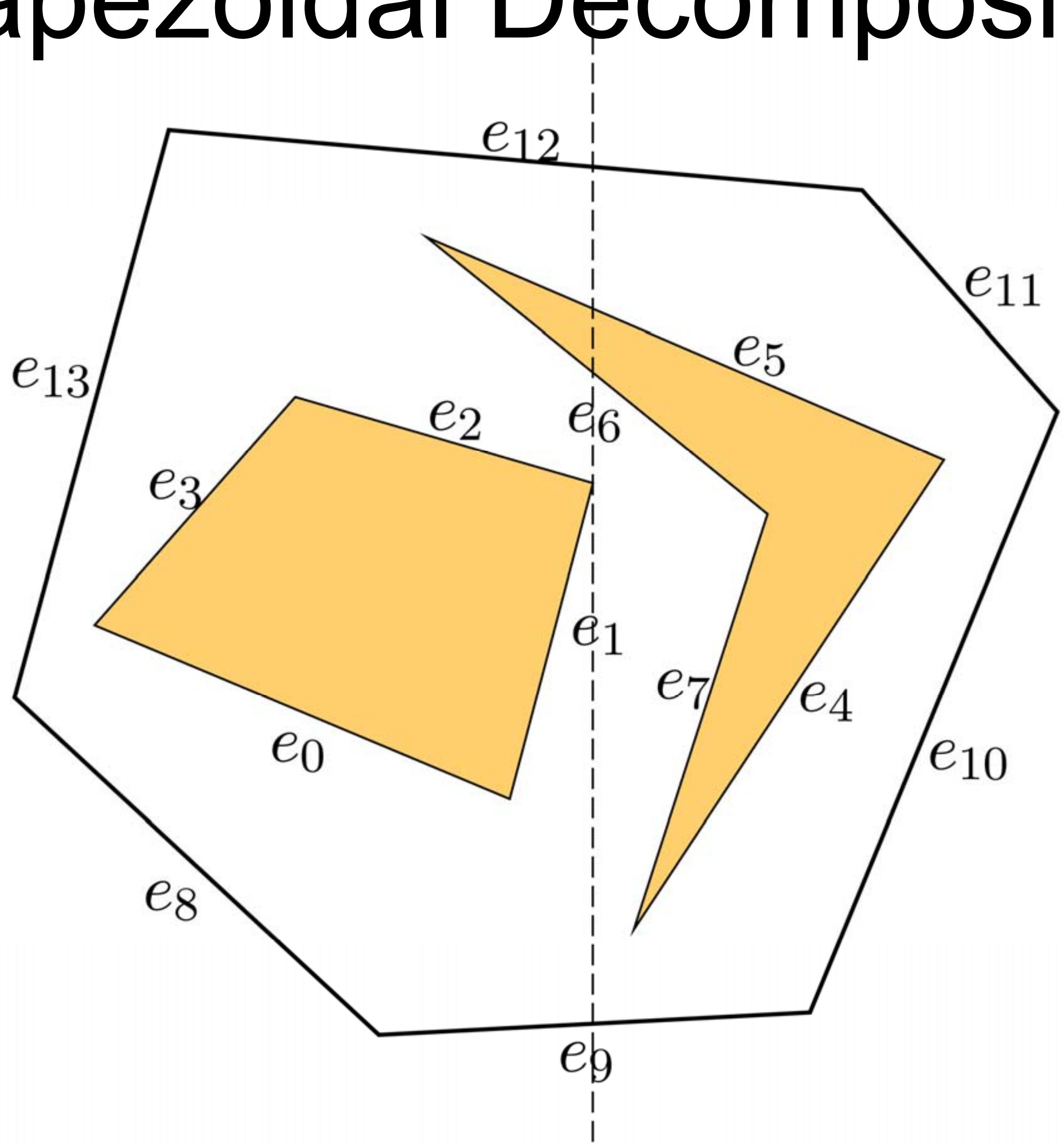
Trapezoidal Decomposition



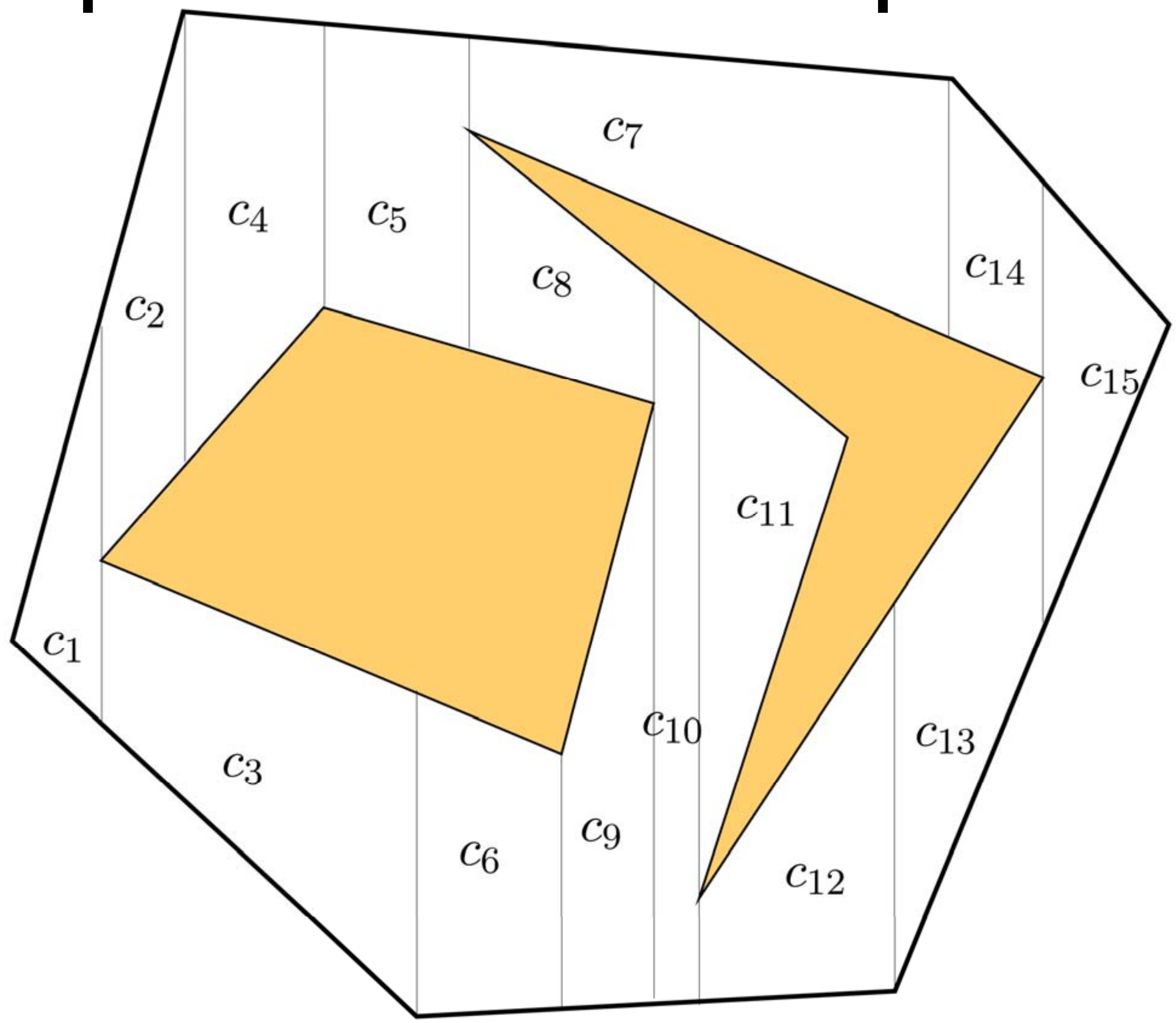
Trapezoidal Decomposition



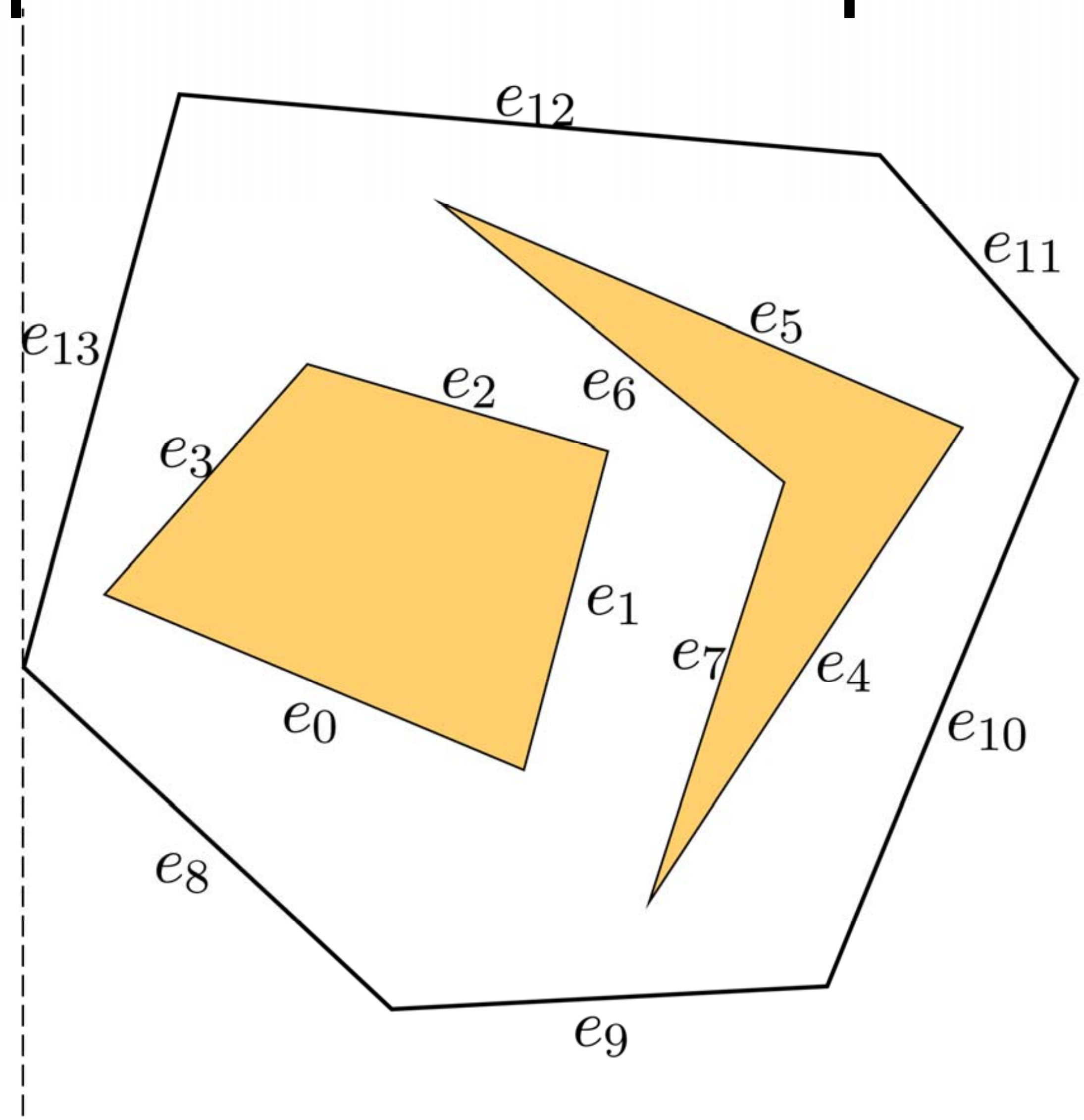
Trapezoidal Decomposition



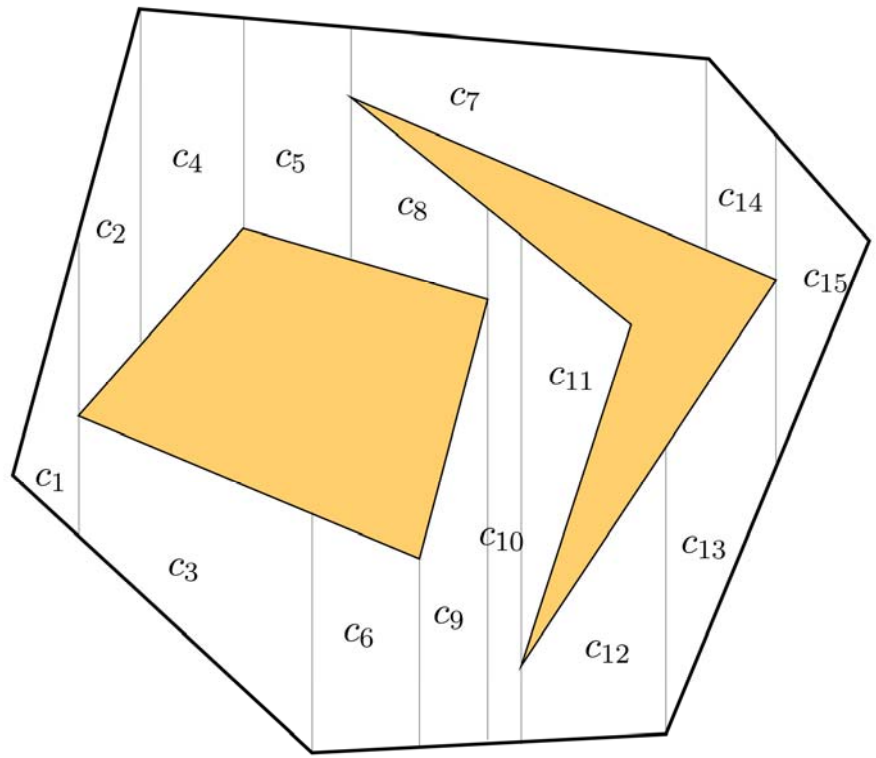
Trapezoidal Decomposition

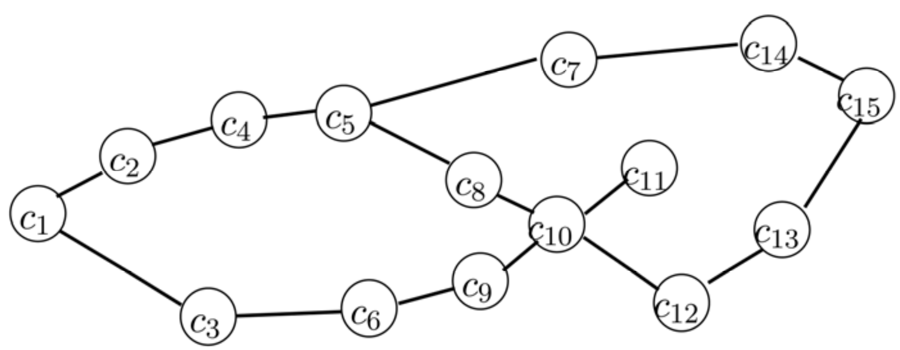


Trapezoidal Decomposition

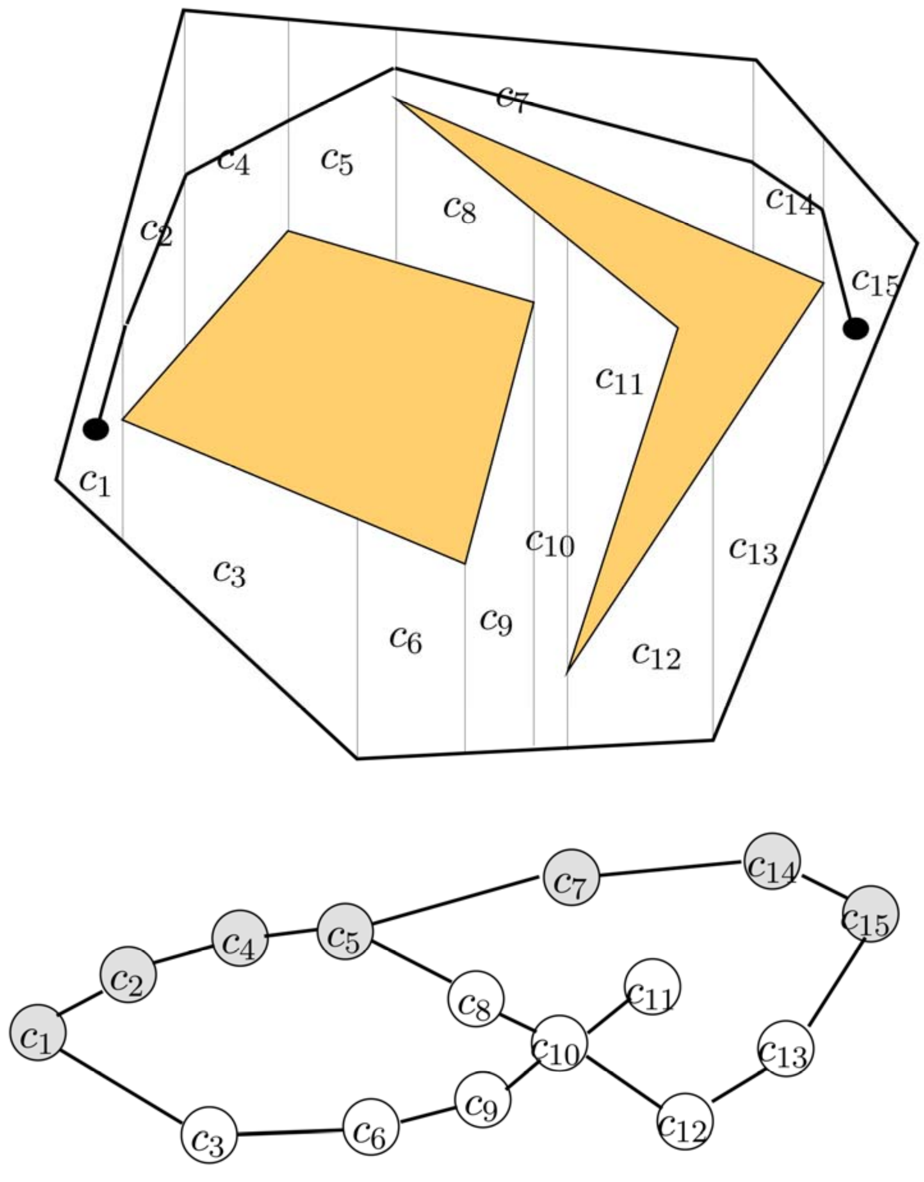


Trapezoidal Decomposition





Trapezoidal Decomposition Path



Implementation

- Input is vertices and edges
- Sort n vertices O(n logn)
- Determine vertical extensions
  - For each vertex, intersect vertical line with each edge – O(n) time
  - Total $O(n^2)$ time

Sweep line approach

Sweep a line through the space stopping at vertices which are often called events

Maintain a list L of the current edges the slice intersects

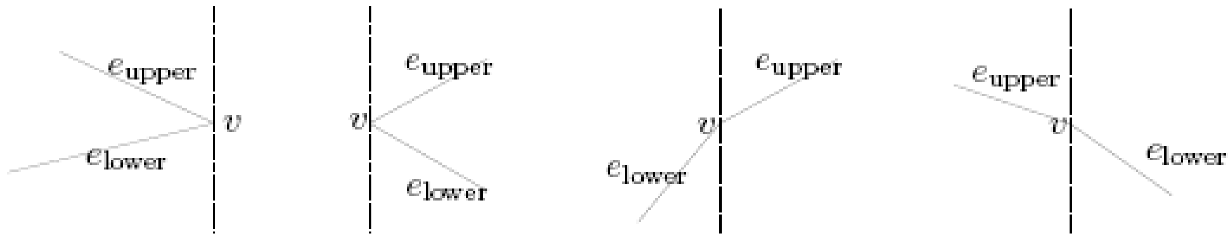
Determining the intersection of slice with L requires O(n) time but with an efficient data structure like a balanced tree, perhaps O(log n)

Really, determine between which two edges the vertex or event lies
These edges are e_{LOWER} and e_{UPPER}

So, really maintaining L takes O(n log n) – log n for insertions, n for vertices

Events

“other” vertex of e_{lower} has a y-coordinate lower than the “other” vertex of e_{upper}



Out

e_{lower} and e_{upper} are both to the left of the sweep line

- delete e_{lower} and e_{upper} from the list
- $(\ldots, e_{\text{LOWER}}, e_{\text{lower}}, e_{\text{upper}}, e_{\text{UPPER}}, \ldots)$
  $(\ldots, e_{\text{LOWER}}, e_{\text{UPPER}}, \ldots)$

Middle

e_{lower} is to the left and e_{upper} is to the right of the sweep line

- delete e_{lower} from the list and insert e_{upper}
- $(\ldots, e_{\text{LOWER}}, e_{\text{lower}}, e_{\text{UPPER}}, \ldots)$
  $(\ldots, e_{\text{LOWER}}, e_{\text{upper}}, e_{\text{UPPER}}, \ldots)$

In

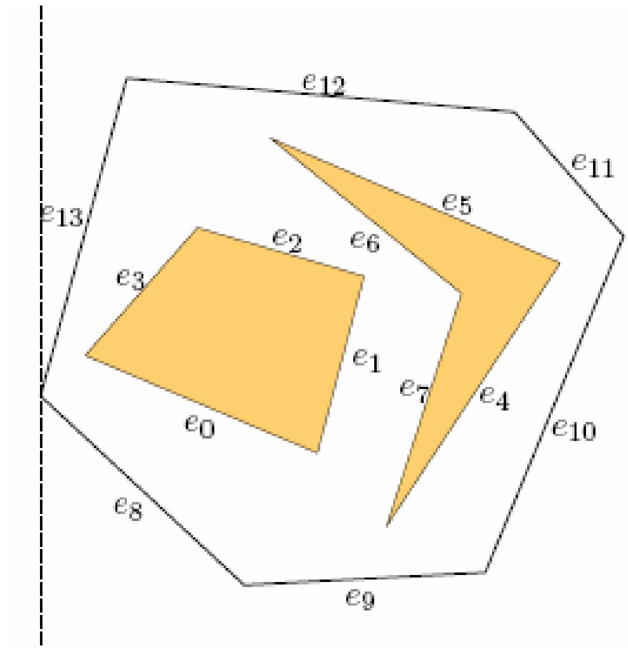
e_{lower} and e_{upper} are both to the right of the sweep line

- insert e_{lower} and e_{upper} into the list
- $(\ldots, e_{\text{LOWER}}, e_{\text{UPPER}}, \ldots) \rightarrow (\ldots, e_{\text{LOWER}}, e_{\text{lower}}, e_{\text{upper}}, e_{\text{UPPER}}, \ldots)$

e_{lower} is to the right and e_{upper} is to the left of the sweep line

- delete e_{upper} from the list and insert e_{lower}
- $(\ldots, e_{\text{LOWER}}, e_{\text{upper}}, e_{\text{UPPER}}, \ldots)$
  $(\ldots, e_{\text{LOWER}}, e_{\text{lower}}, e_{\text{UPPER}}, \ldots)$

Example

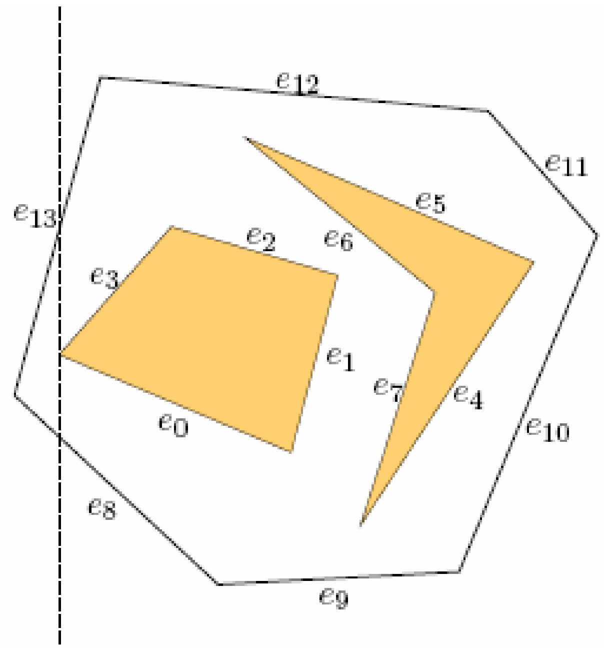


$$L : \emptyset \rightarrow \{e_8, e_{13}\}$$

e_{lower} and e_{upper} are both to the right of the sweep line

- insert e_{lower} and e_{upper} into the list
- $(..., e_{\text{LOWER}}, e_{\text{UPPER}}, ...) \rightarrow (..., e_{\text{LOWER}}, e_{\text{lower}}, e_{\text{upper}}, e_{\text{UPPER}}, ...)$

Example

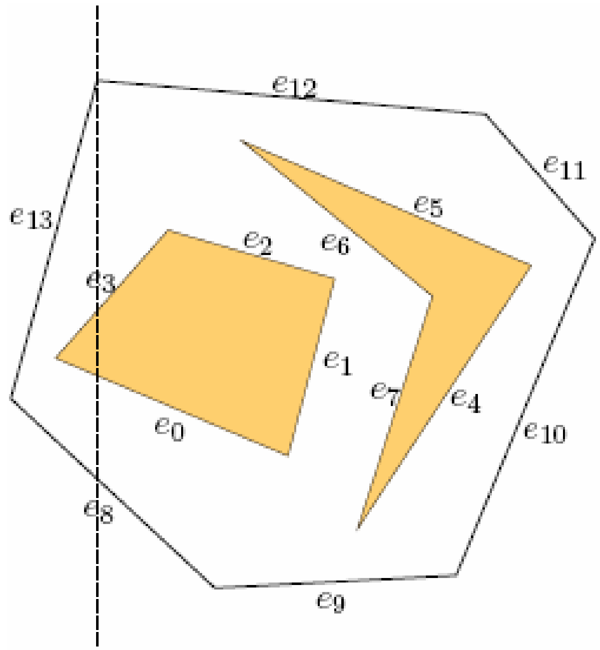


$$L : \{e_8, e_{13}\} \rightarrow \{e_8, e_0, e_3, e_{13}\}$$

e_{lower} and e_{upper} are both to the right of the sweep line

- insert e_{lower} and e_{upper} into the list
- $(..., e_{\text{LOWER}}, e_{\text{UPPER}}, ...) \rightarrow (..., e_{\text{LOWER}}, e_{\text{lower}}, e_{\text{upper}}, e_{\text{UPPER}}, ...)$

Example

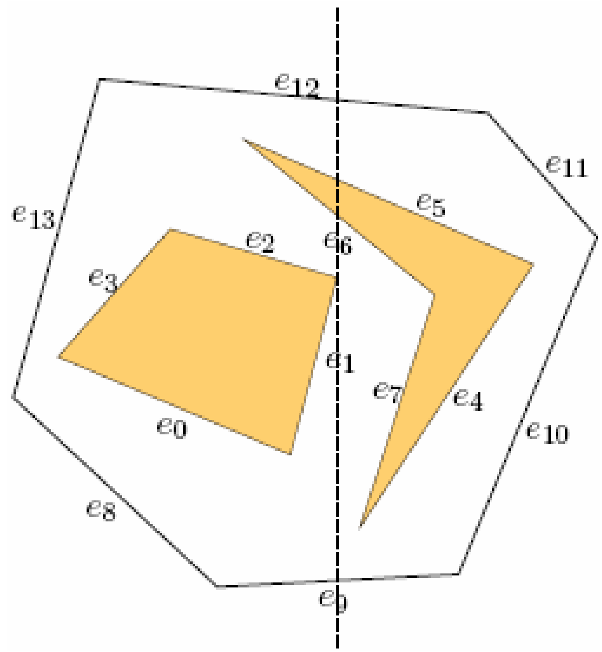


$L: \{e_8, e_0, e_3, e_{13}\} \rightarrow \{e_8, e_0, e_3, e_{12}\}$

e_{lower} is to the left and e_{upper} is to the right of the sweep line

- delete e_{lower} from the list and insert e_{upper}
- $(\ldots, e_{\text{LOWER}}, e_{\text{lower}}, e_{\text{UPPER}}, \ldots)$
  $(\ldots, e_{\text{LOWER}}, e_{\text{upper}}, e_{\text{UPPER}}, \ldots)$

Example



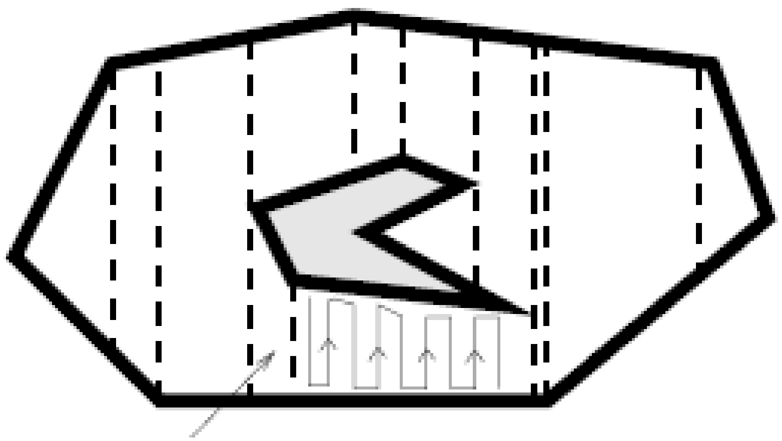
$$\{e_9, e_1, e_2, e_6, e_5, e_{12}\} \rightarrow \{e_9, e_6, e_5, e_{12}\}.$$

delete e_{lower} and e_{upper} from the list

$(..., e_{\text{LOWER}}, e_{\text{lower}}, e_{\text{upper}}, e_{\text{UPPER}}, ...)$

$(..., e_{\text{LOWER}}, e_{\text{UPPER}}, ...)$

Coverage



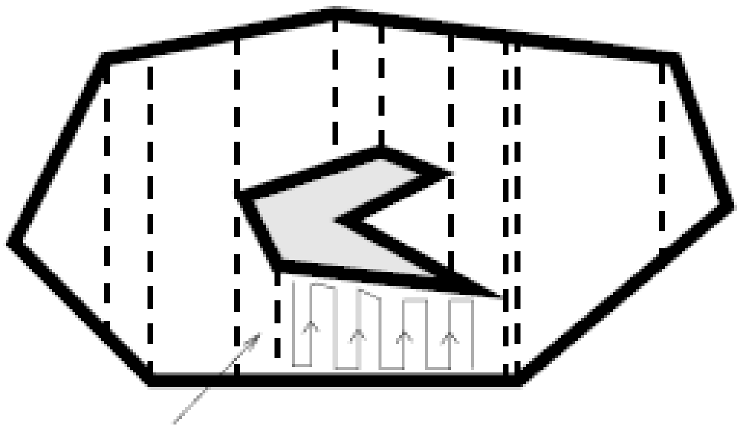
Planner determines an exhaustive walk through the adjacency graph

Planner computes explicit robot motions within each cell

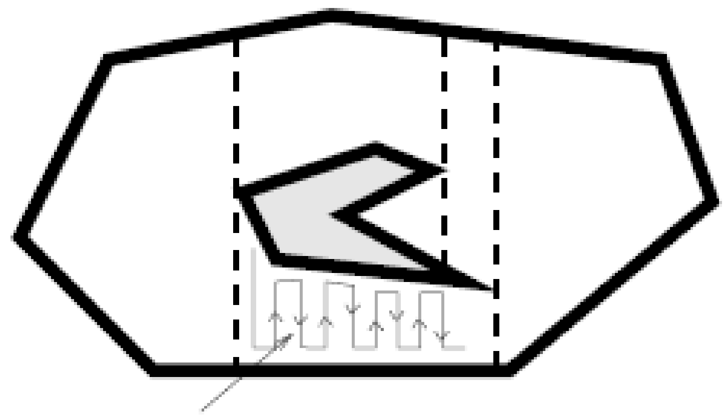
Problems

1. Polygonal representation
2. Quantization
3. Position uncertainty
4. Full information
5. What else?

Boustrophedon Decomposition

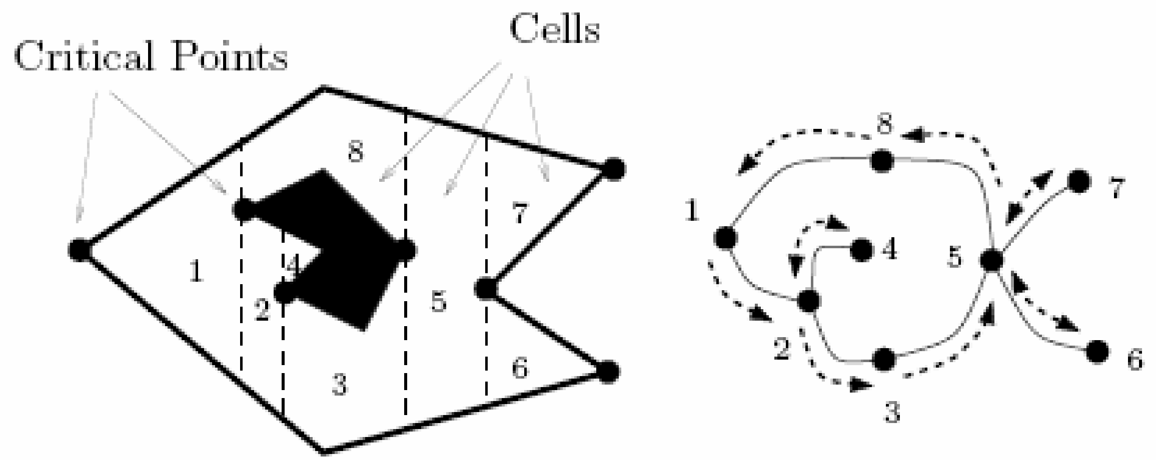


Coverage Path in a Cell.



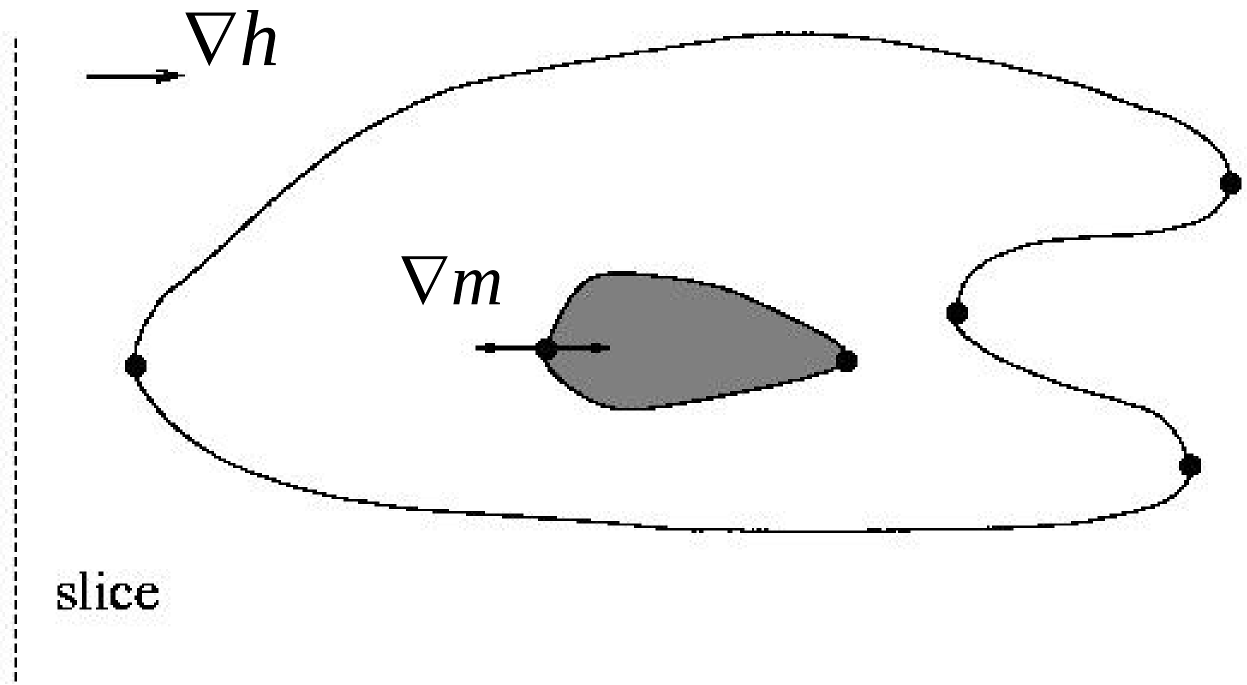
Coverage Path in a Cell.

Complete Coverage

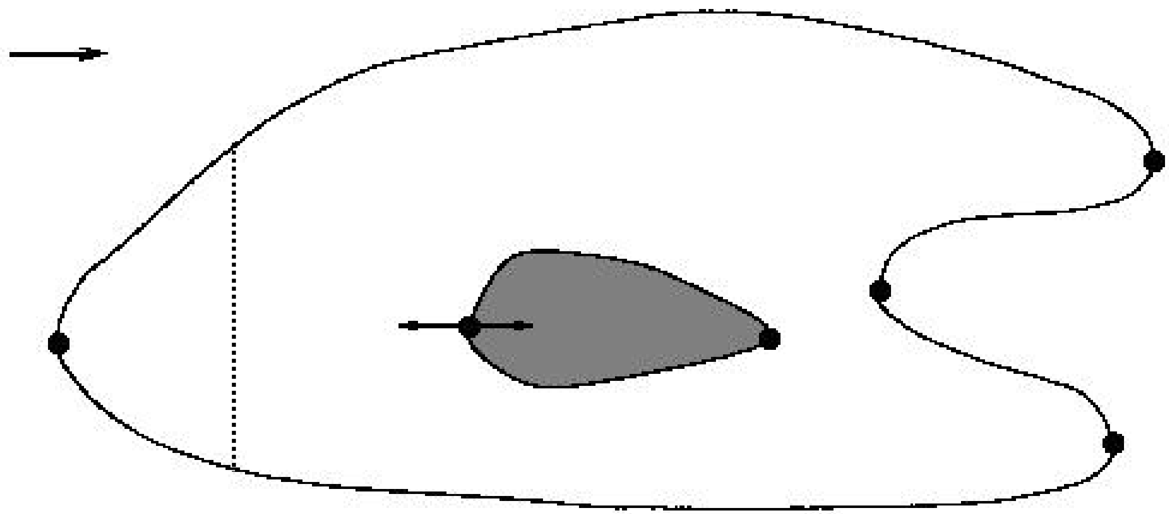


Exhaustive walk 1–2–4–2–3–5–6–5–7–5–8–1

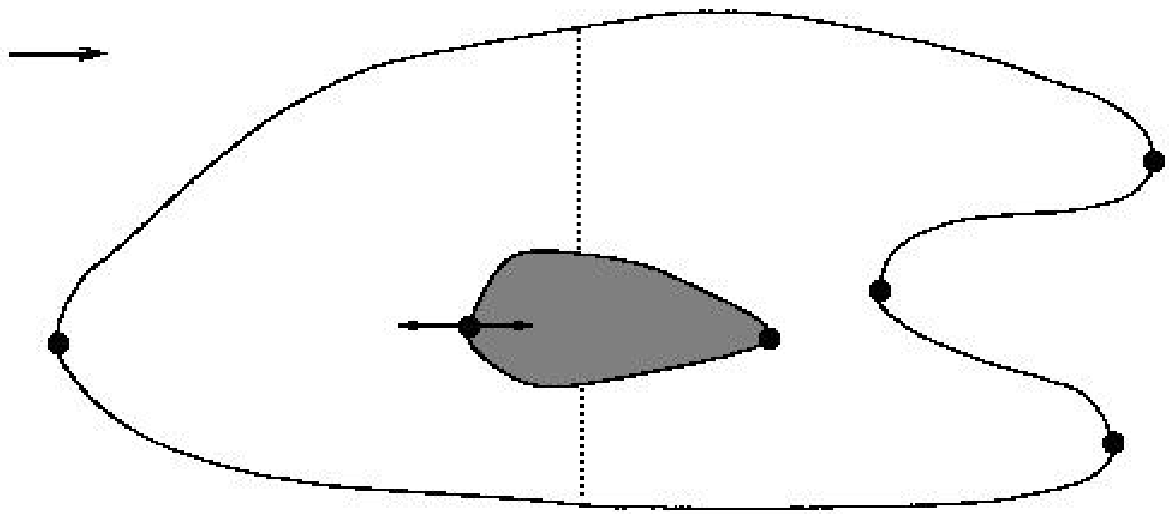
Morse Decomposition in Terms of Critical Points



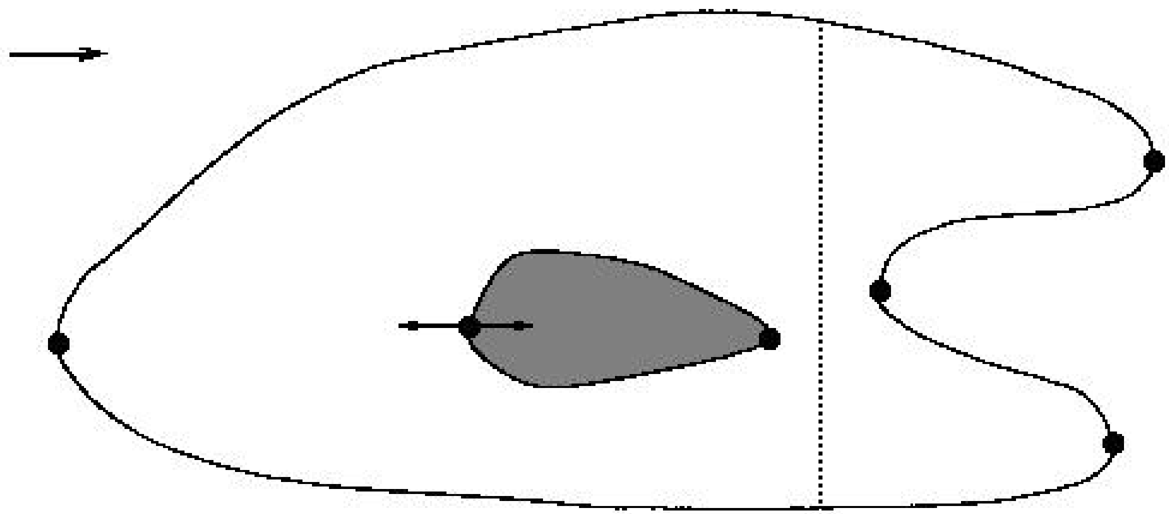
- *Slice function: h(x,y)= x*
- *At a critical point x of* $h|_M, \nabla h(x) = \nabla m(x)$ *where* $M = \{x | m(x)=0\}$



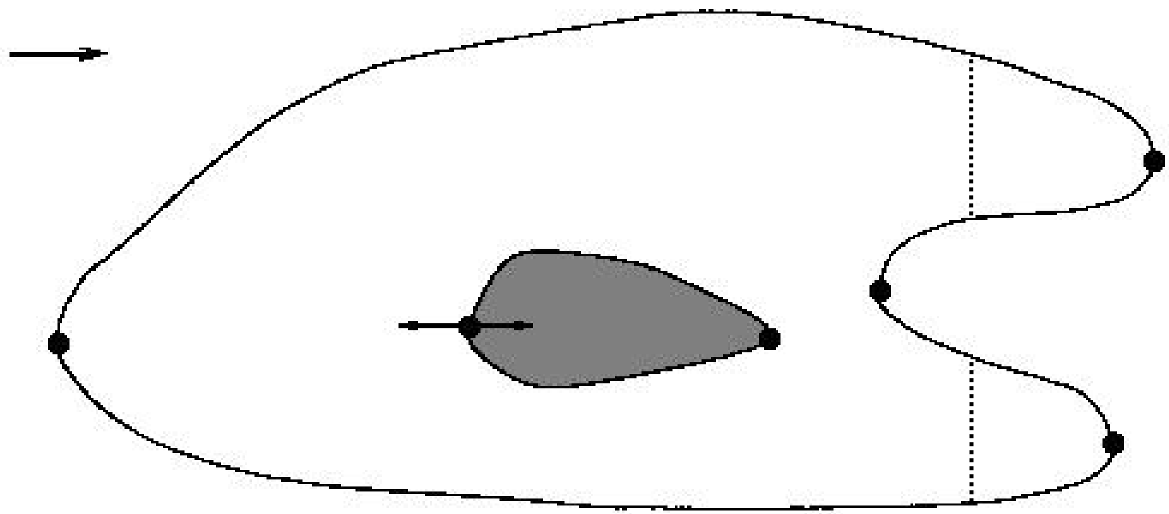
1-connected



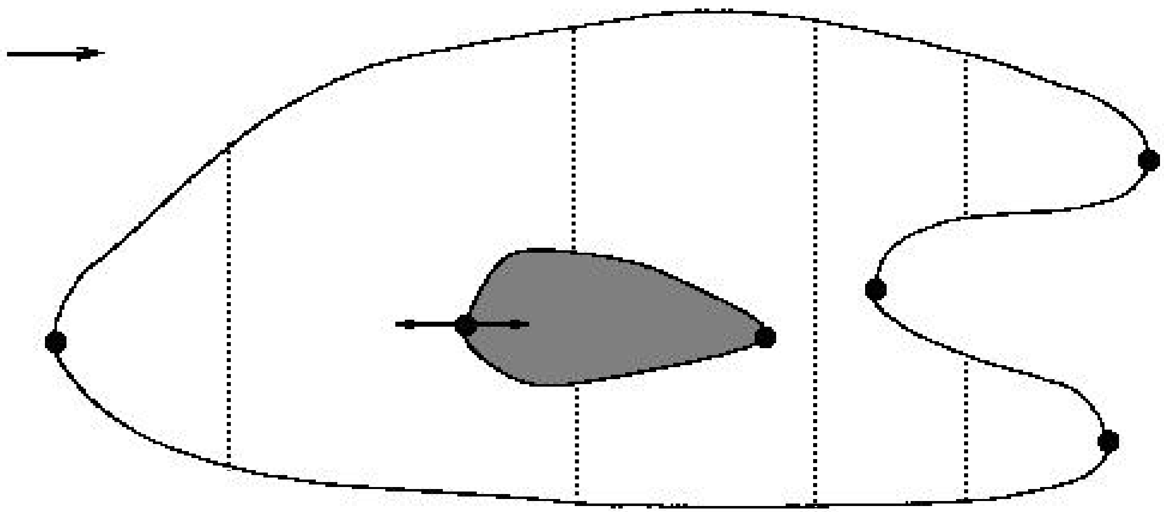
2-connected



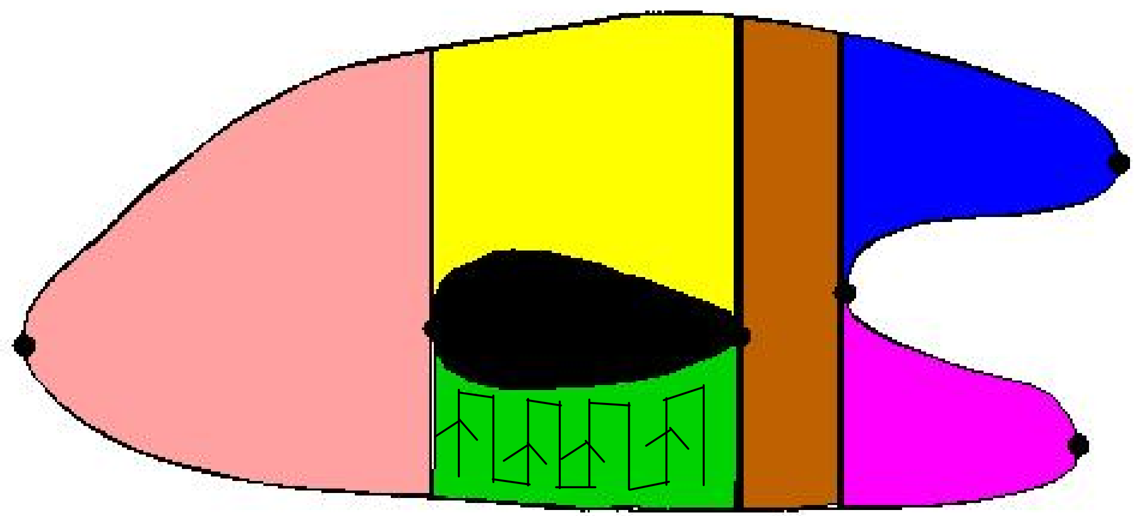
1-connected



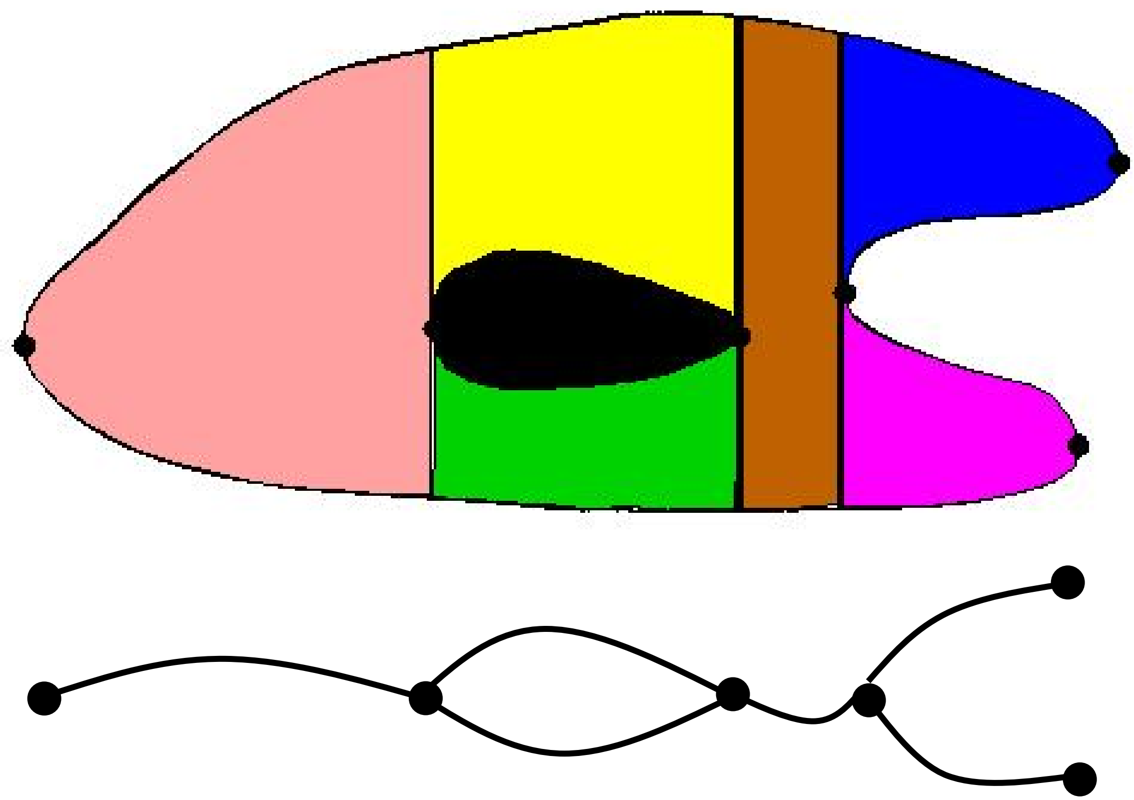
2-connected



• ***Connectivity of the slice in the free space changes at the critical points***



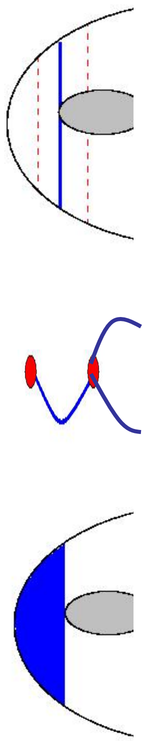
- ***Each cell can be covered by back and forth motions***



- ***Reeb graph represents the topology of the cellular decomposition***

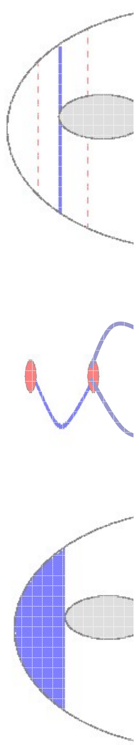
Incremental construction

- ***While covering the space, look for critical points***

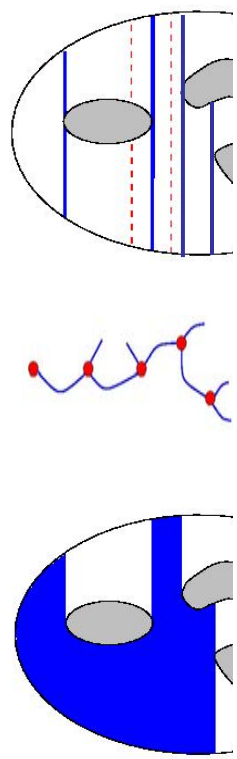


Stage 1

Incremental construction (cont'd)

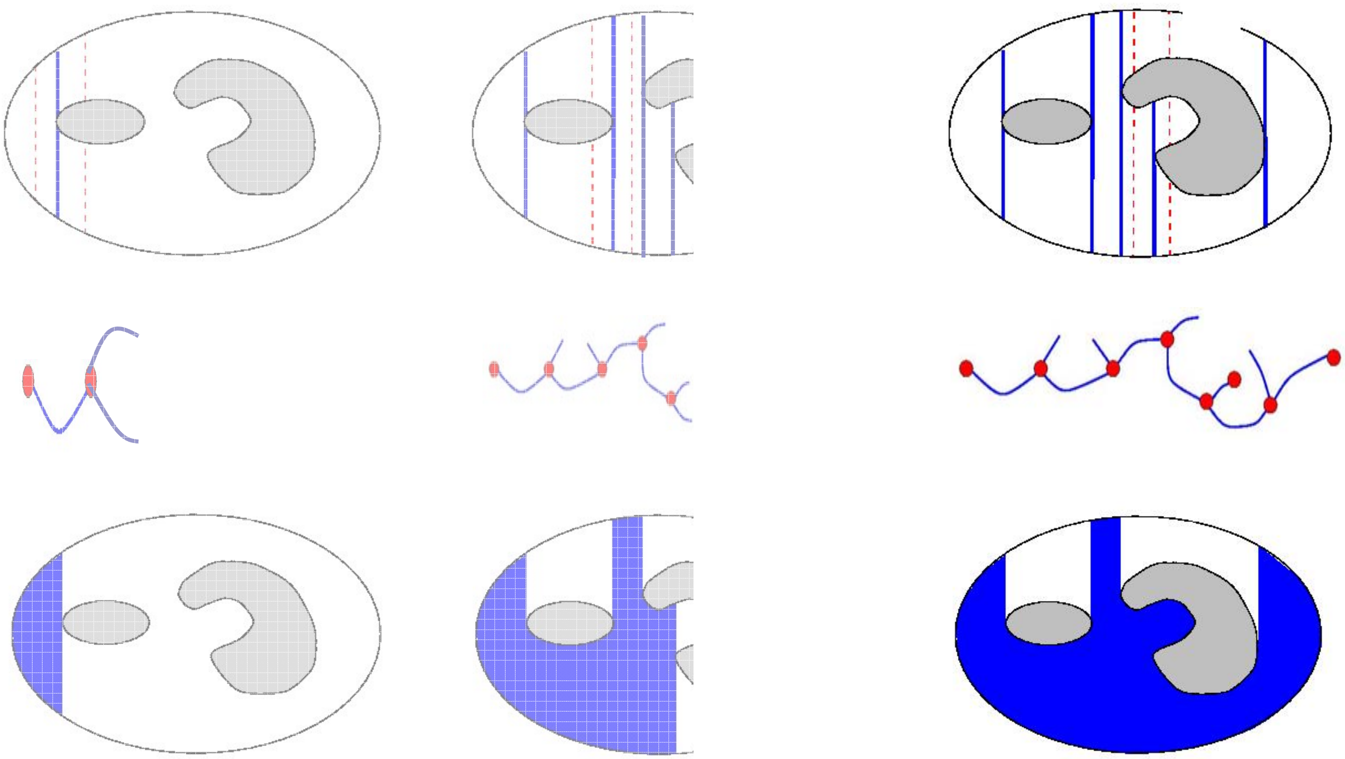


Stage 1



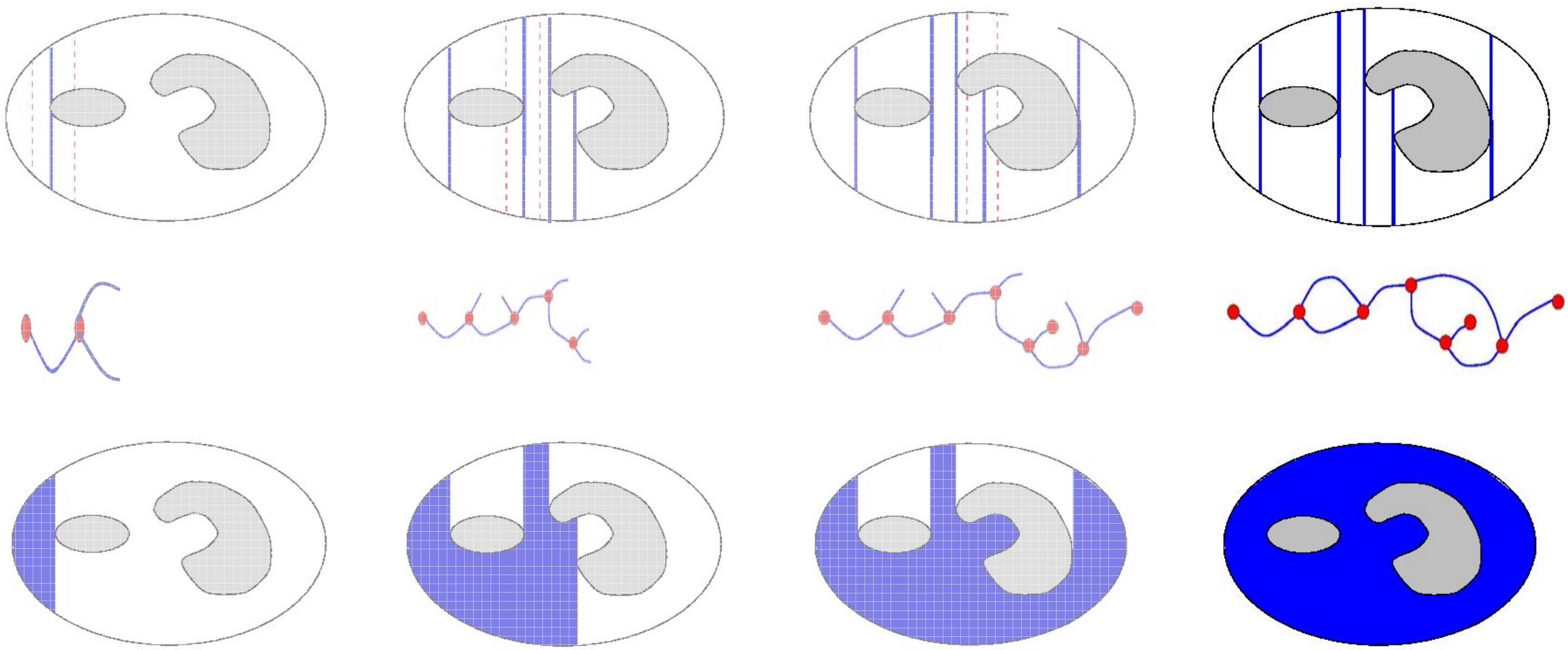
Stage 2

Incremental construction (cont'd)



Stage 1 *Stage 2* *Stage 3*

Incremental construction (cont'd)

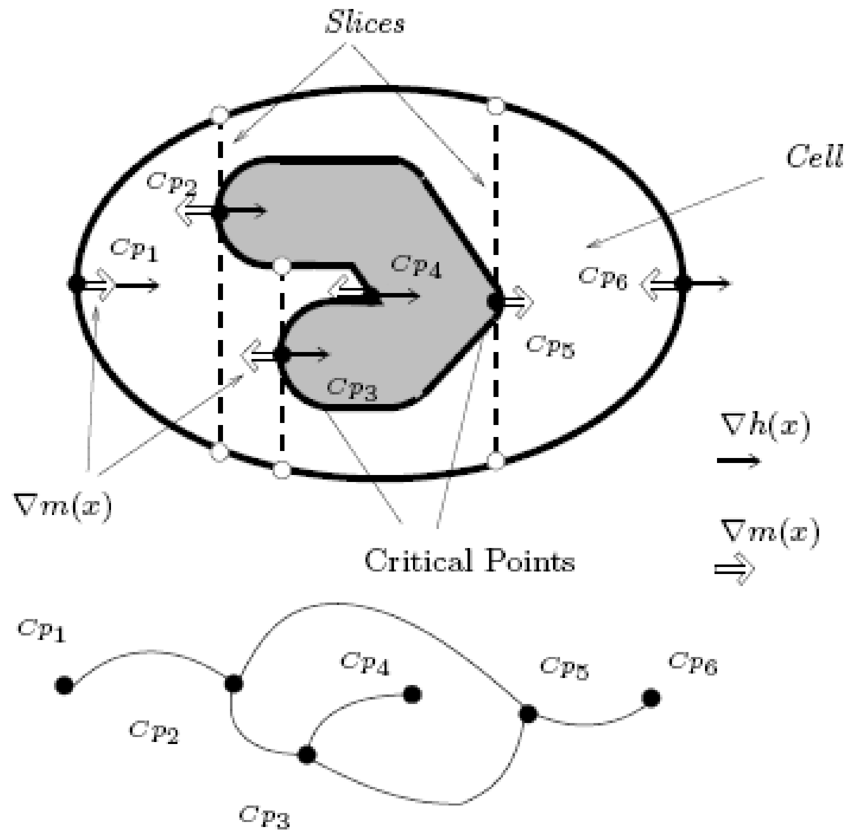


Stage 1 *Stage 2* *Stage 3* *Stage 4*

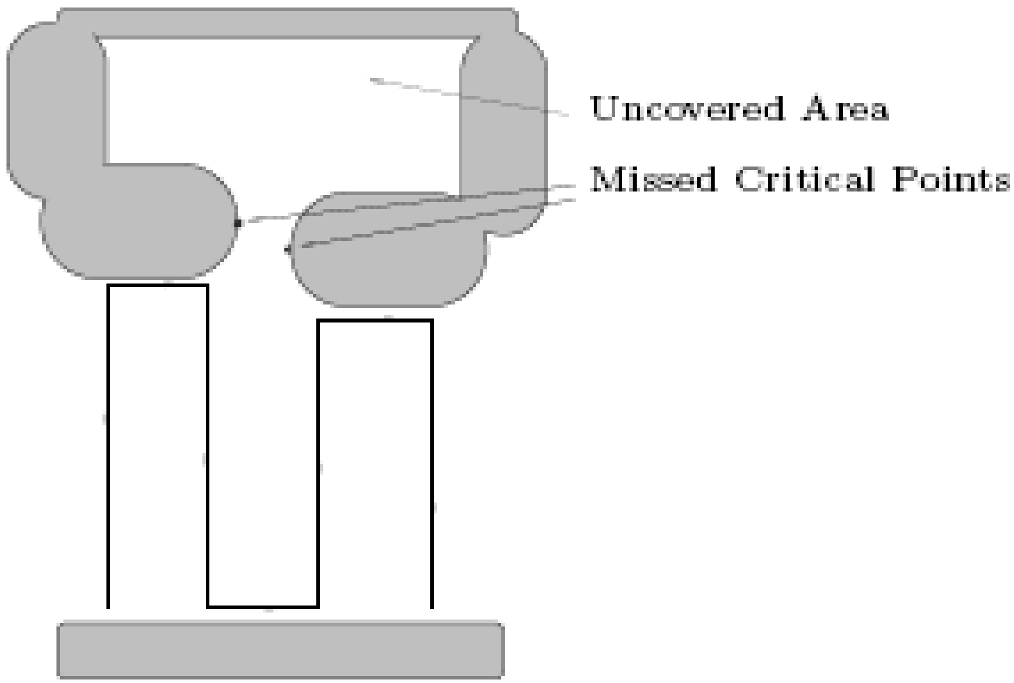
Algorithm

- Cover a cell until the closing critical point is detected
- If the closing critical point has “uncleaned” cells associated with it, chose one and cover, repeat
- If the closing critical point has no uncleaned cells,
  - search reeb graph for a critical point with an uncleaned cell
  - Plan a path (on average shorter than bug2) to critical point
  - Cover cell, repeat
- Else coverage is complete

Detect Critical Points



Encountering Critical Points: Problem



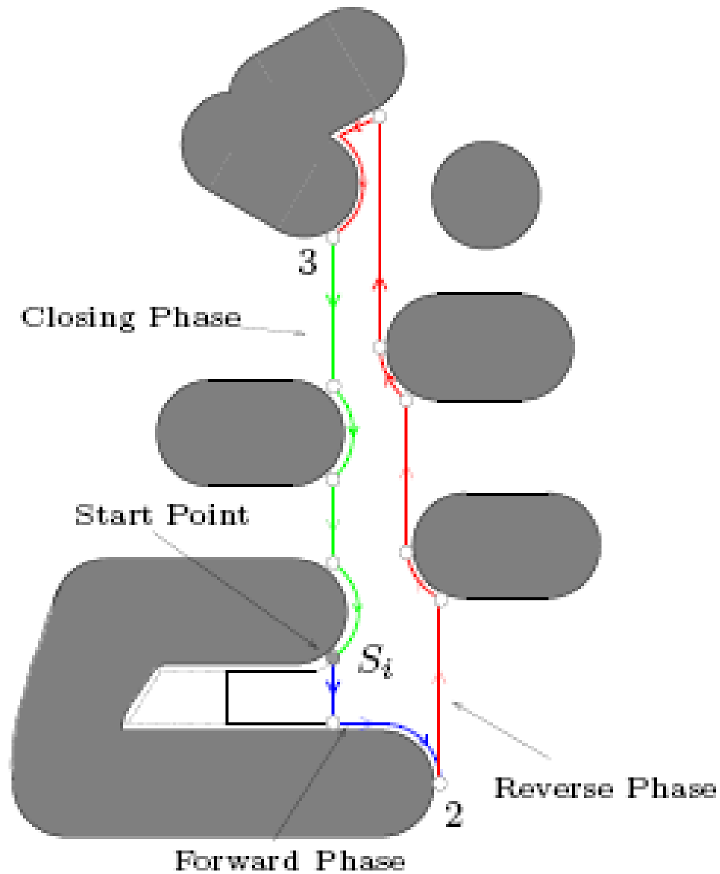
closing critical point

Cycle Algorithm

Forward phase: The robot follows a slice, i.e., laps, until it encounters an obstacle. Then the robot follows the boundary of the obstacle in the forward sweep direction until either the robot moves laterally one lap width or until the robot encounters a critical point in the floor.

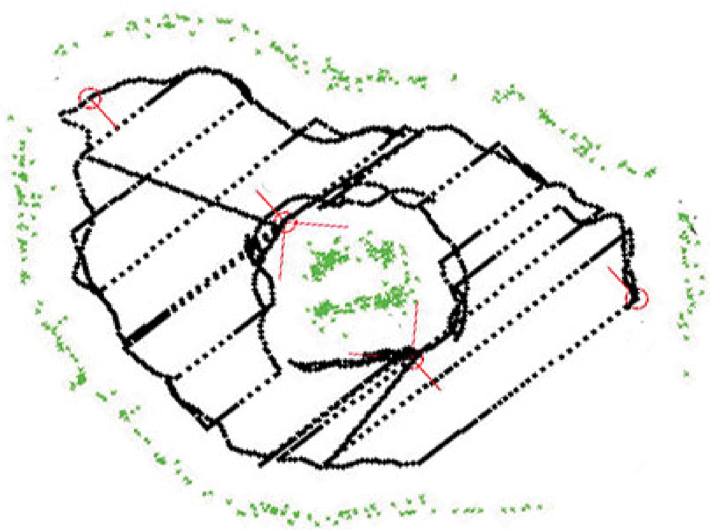
Reverse phase: The robot executes one or more laps in the reverse direction, intermixed with reverse boundary-following. Each reverse boundary-following operation terminates when the robot finds a critical point or when the aggregate lateral motion in the reverse direction is one lap width.

Closing phase: The robot executes one or more laps along the slice, possibly intermixed with boundary-following. Each boundary-following operation terminates when the robot encounters S_i or the slice in which S_i lies.

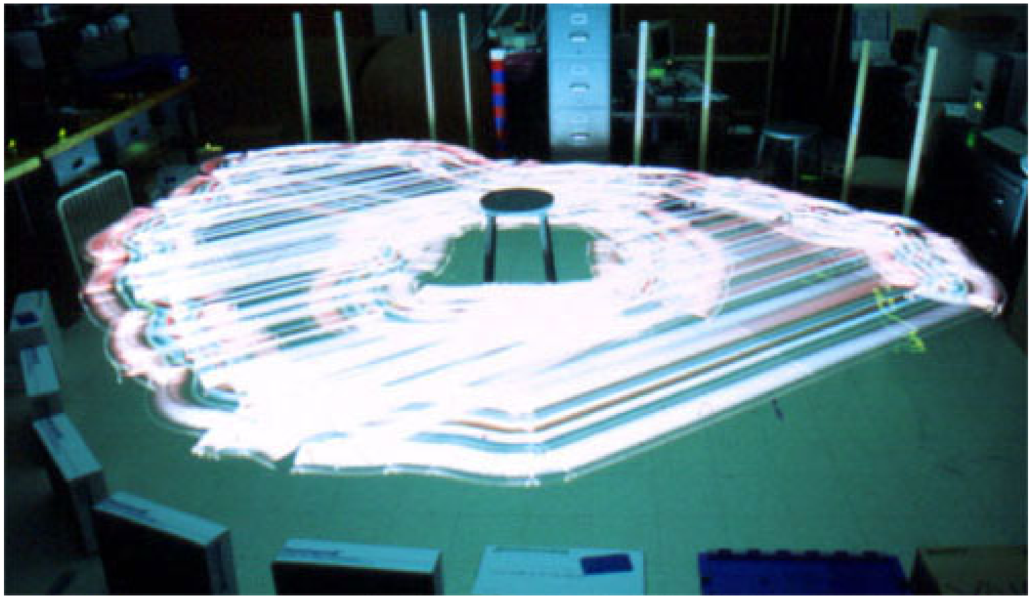


Sensor-based Complete Coverage

***Goal**: Complete coverage of an unknown environment*

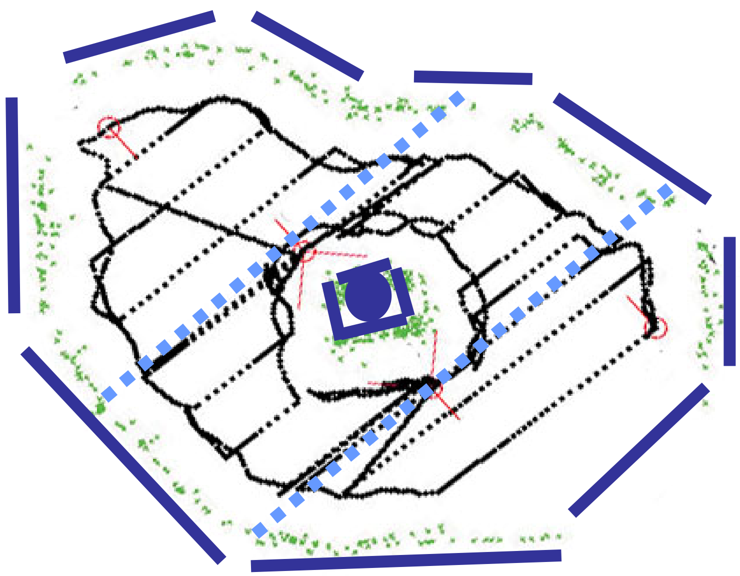


Time-exposure photo of a coverage experiment

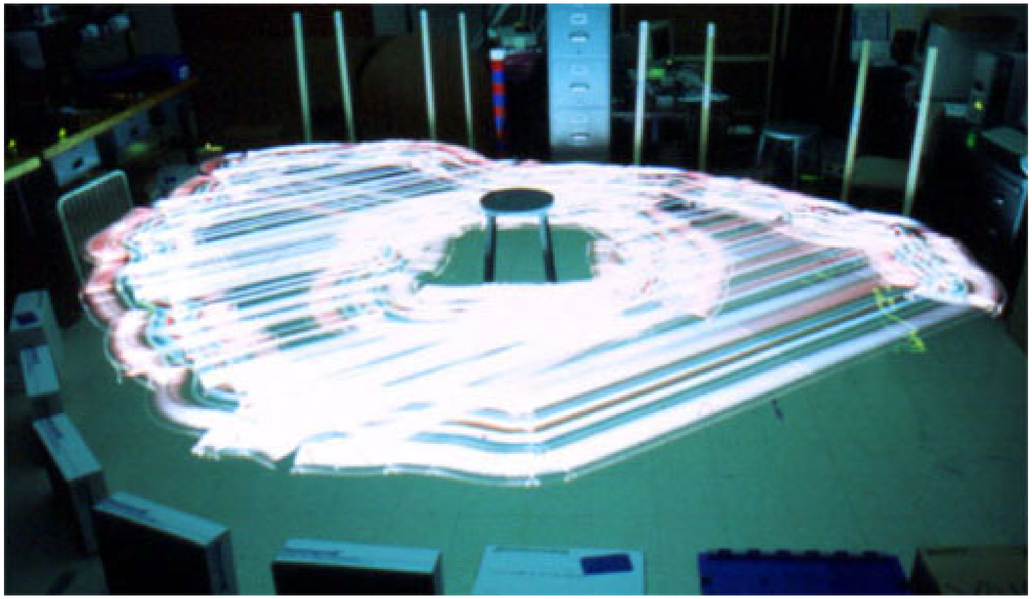


Sensor-based Complete Coverage

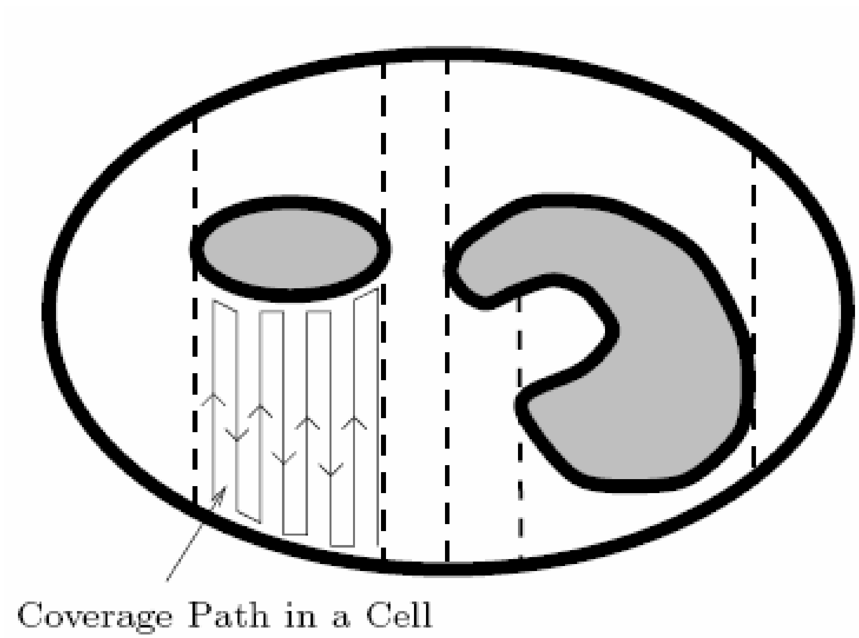
***Goal**: Complete coverage of an unknown environment*



Time-exposure photo of a coverage experiment



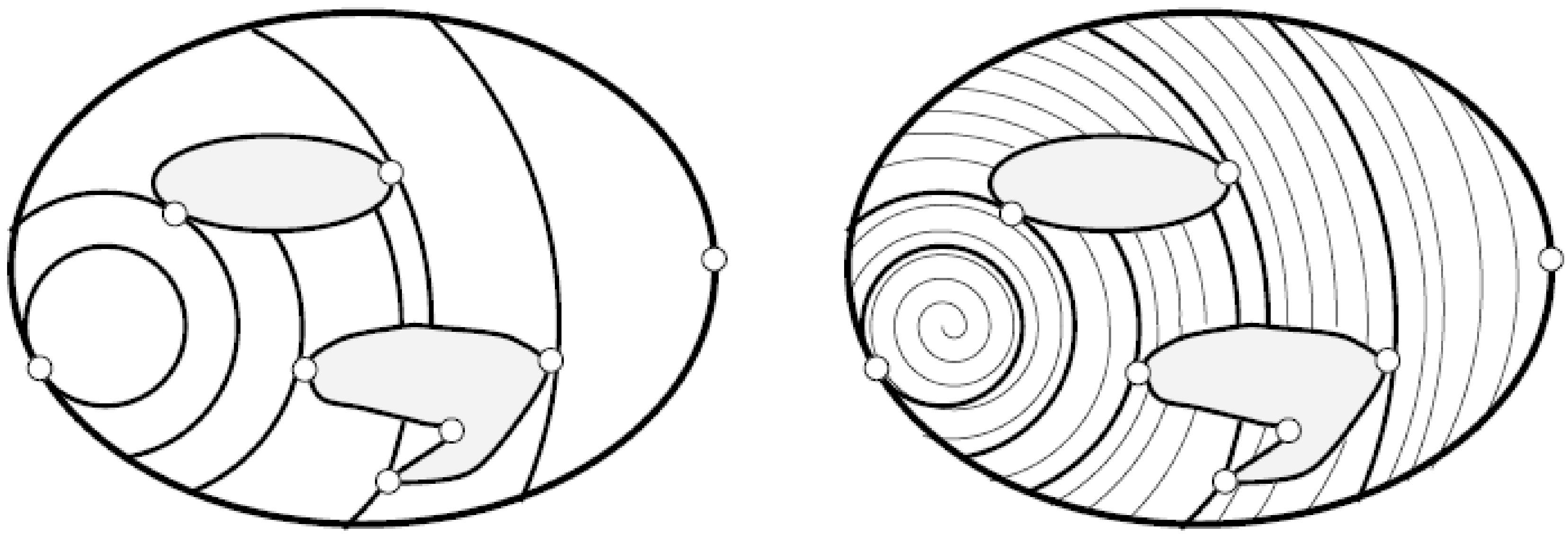
Morse Decomposition h(x,y) = x



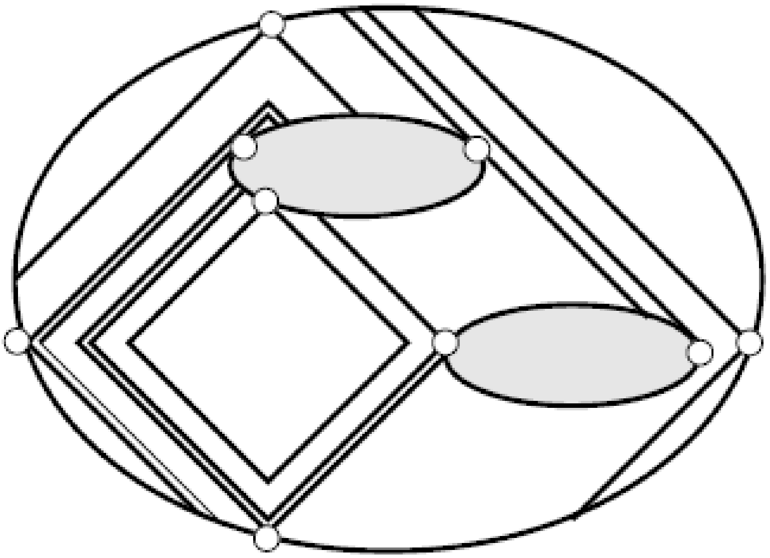
Boustrophedon decomposition

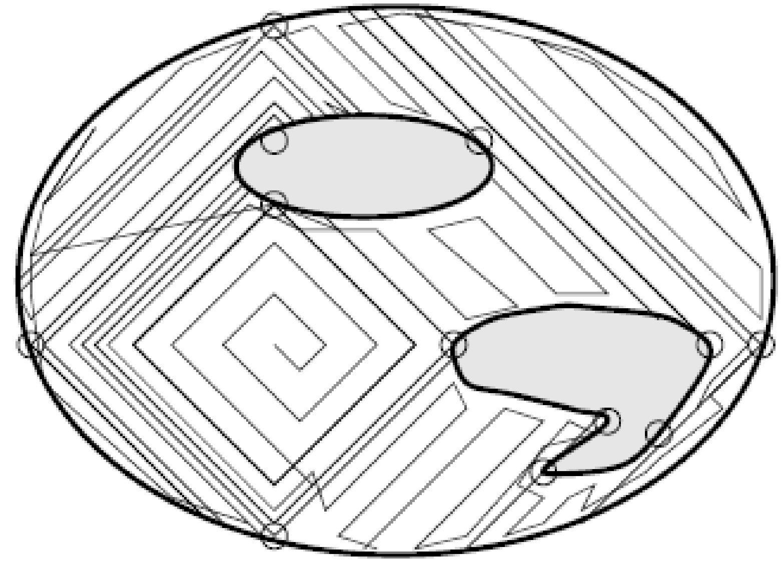
•Canny π_1: Q $\rightarrow \Re$)

Morse Decomposition
$h(x,y) = x^2 + y^2$



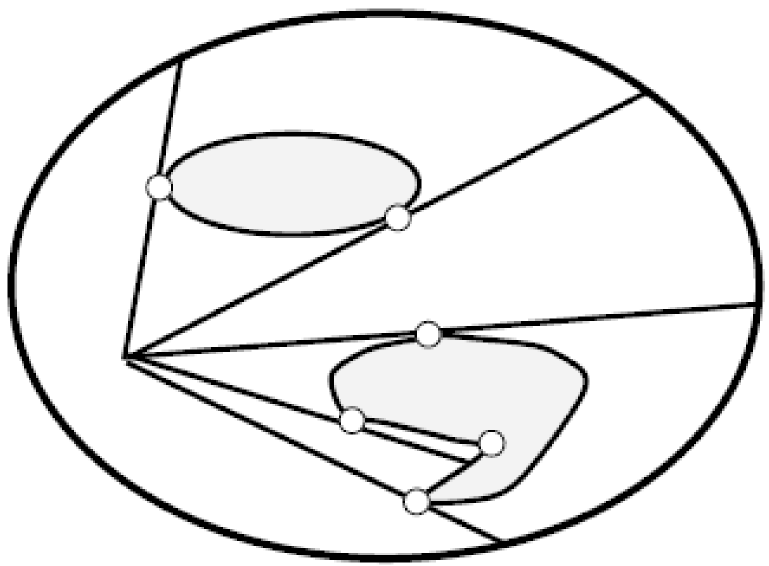
Morse Decomposition
$h(x,y) = |x| + |y|$

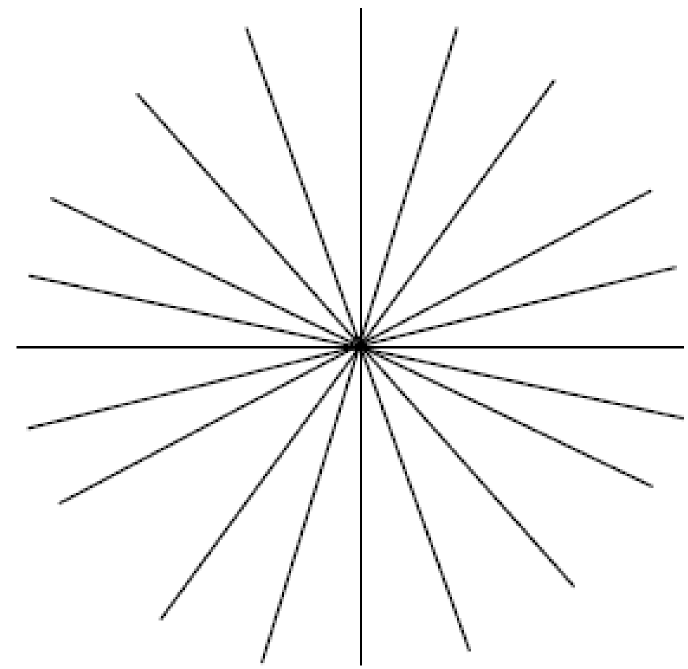




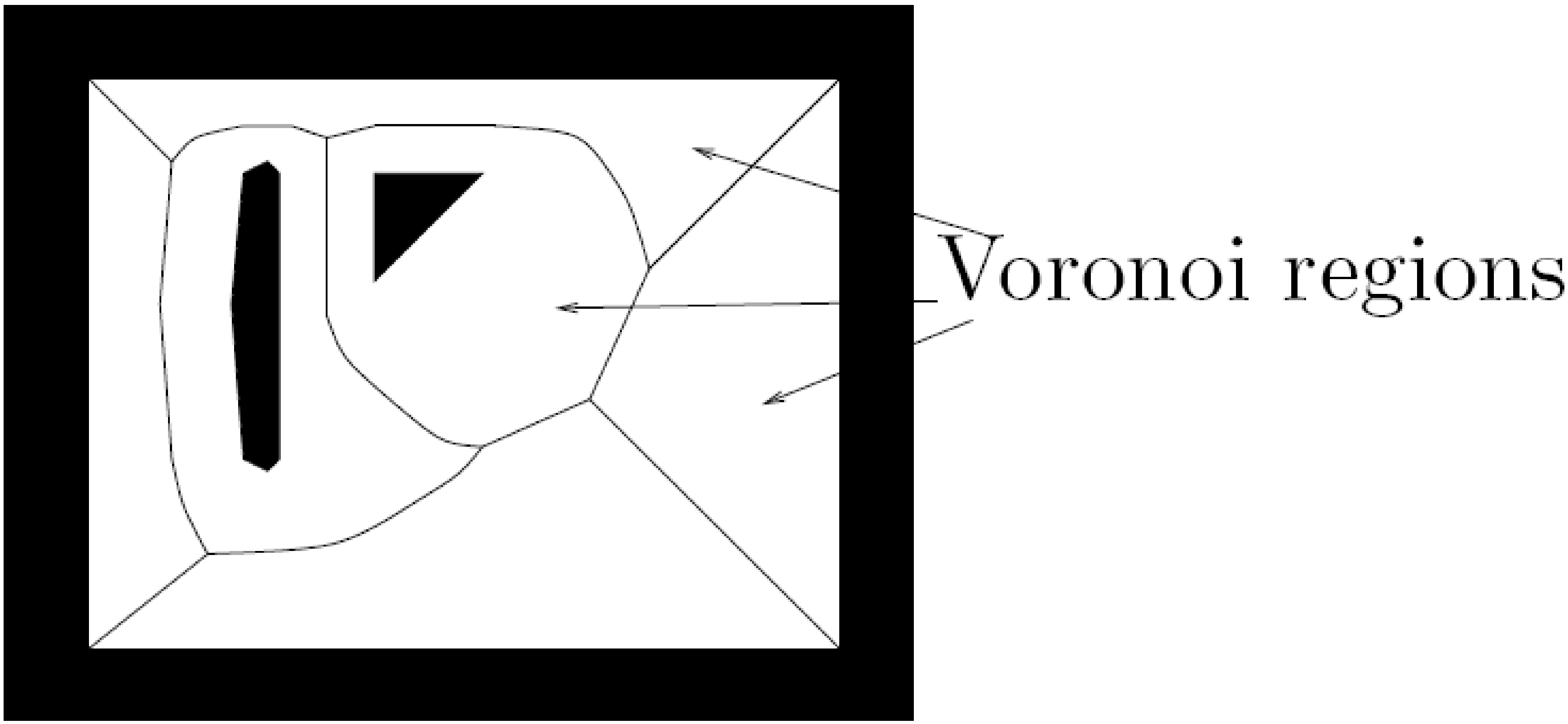
squarels

Morse Decomposition
h(x,y) = tan(y/x)

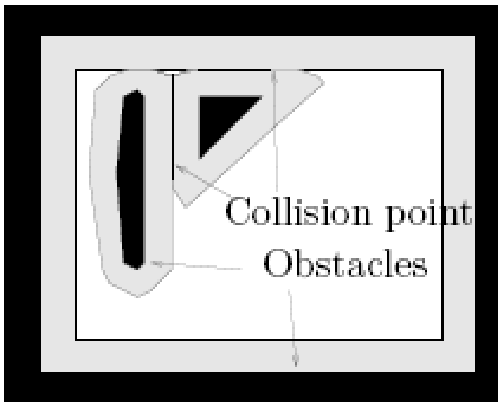




Brushfire Decomposition



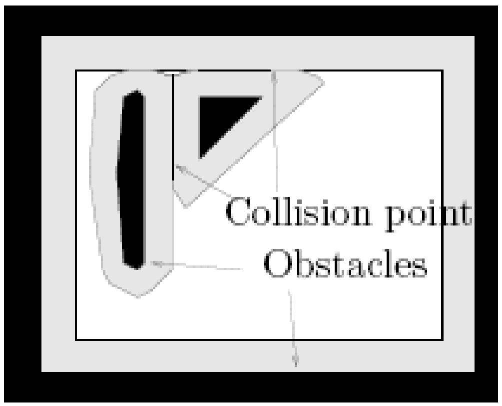
Brushfire Decomposition
h(x,y) = D(x,y)



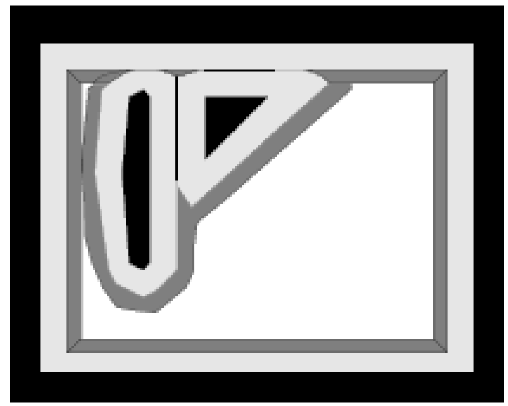
Stage 1

Brushfire Decomposition

$h(x,y) = D(x,y)$

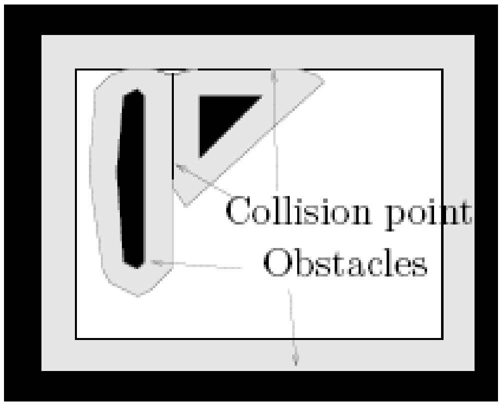


Stage 1

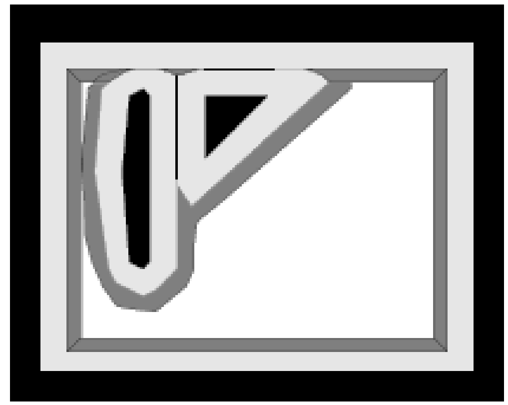


Stage 2

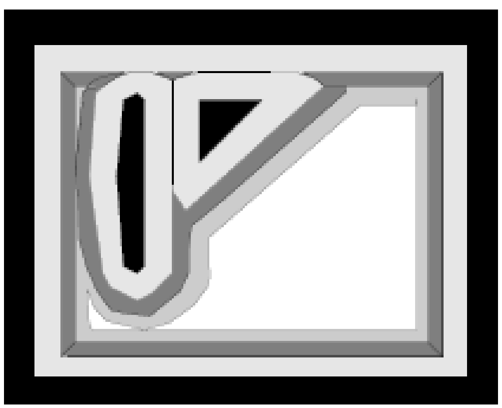
Brushfire Decomposition
h(x,y) = D(x,y)



Stage 1

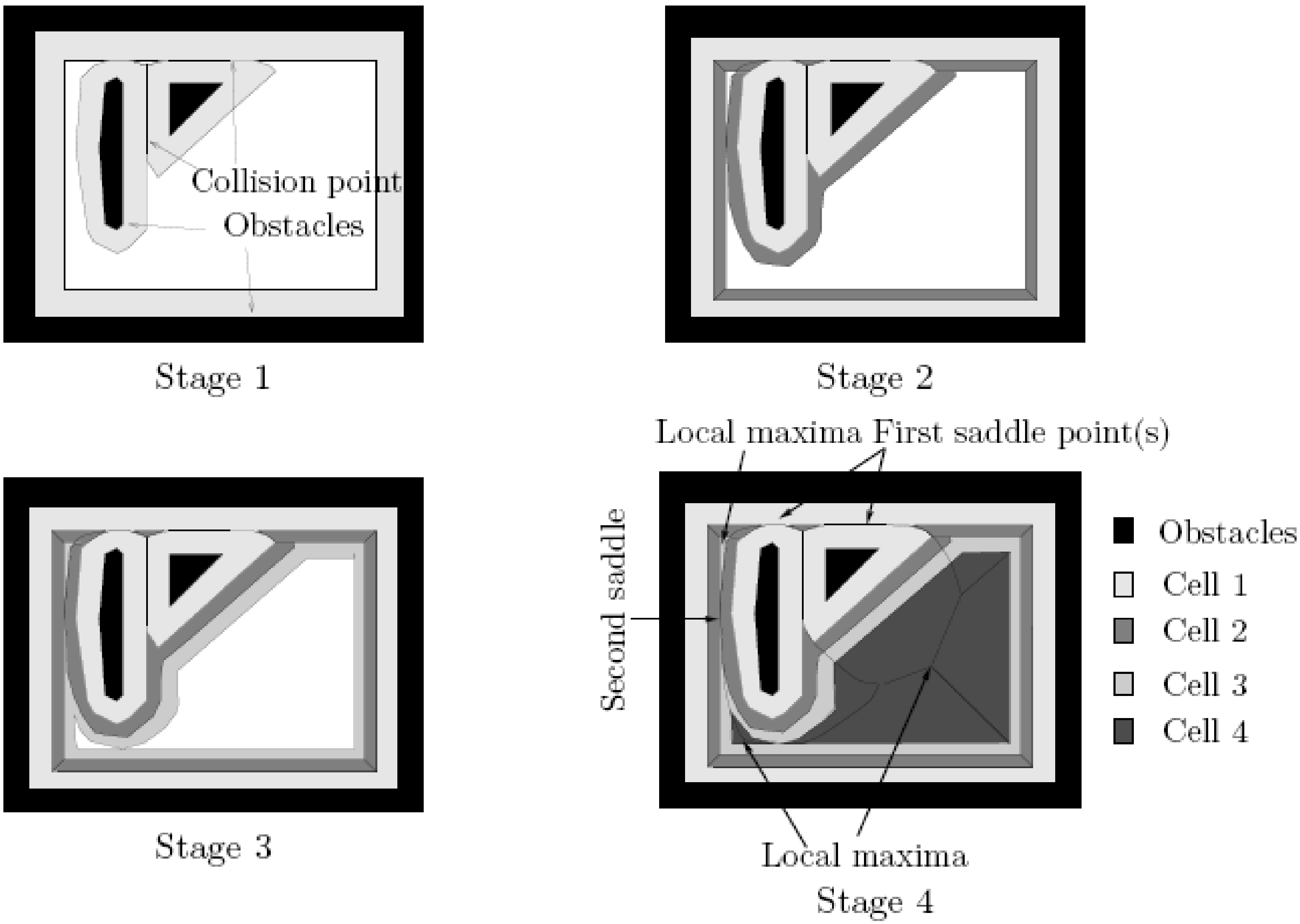


Stage 2

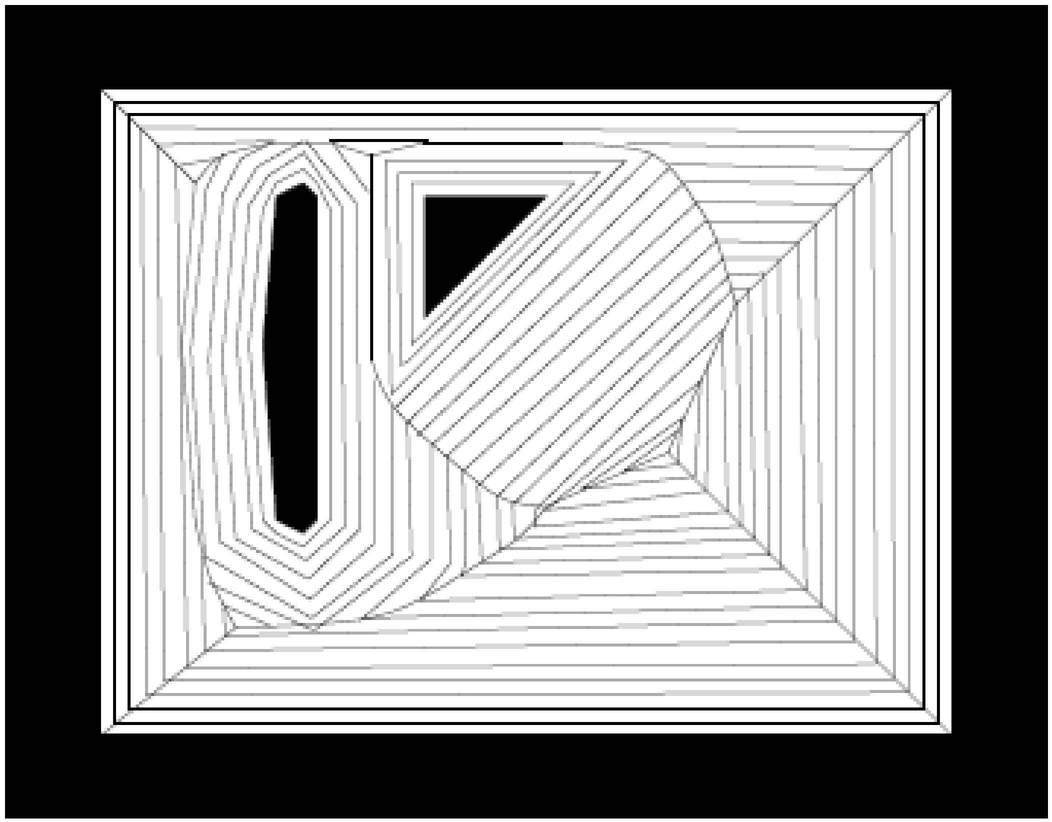


Stage 3

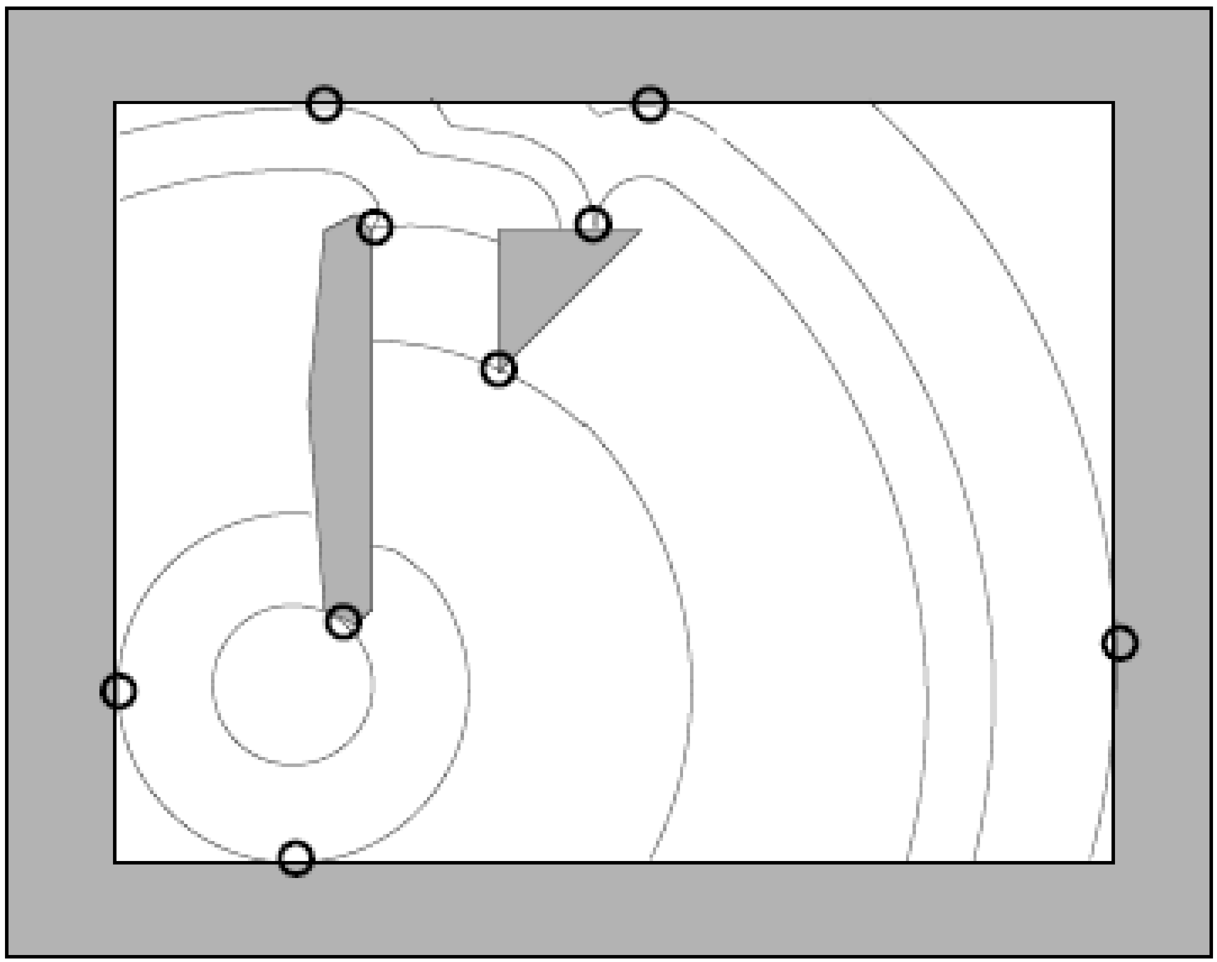
Brushfire Decomposition
h(x,y) = D(x,y)



Brushfire Decomposition Coverage Path



Wavefront Decomposition



Notation

- A slice is a codimension one manifold (Q_λ)
- Slices are parameterized by λ
  - varying λ sweeps a slice through the space
- The portion of the slice in the free configuration space (Q_{free}) is $Q_{free\lambda}$
- $Q_{free\lambda} = Q_\lambda \cap Q_{free}$

Slice Definition

- Slice can be defined in terms of the preimage of the projection operator
- $h: Q \rightarrow \Re$ (Canny $\pi_1: Q \rightarrow \Re$)
- Vertical slice are defined by
- $Q_\lambda = h^{-1}(\lambda)$, with $h(x,y) = x$ for the plane
- Increasing λ sweeps the slice to the right through the plane

The Pursuer-Evader Problem

- Problem definition

  How do you plan the motion of a pursuer(s) in a polygonal environment so that it will eventually "see" an unpredictable evader?

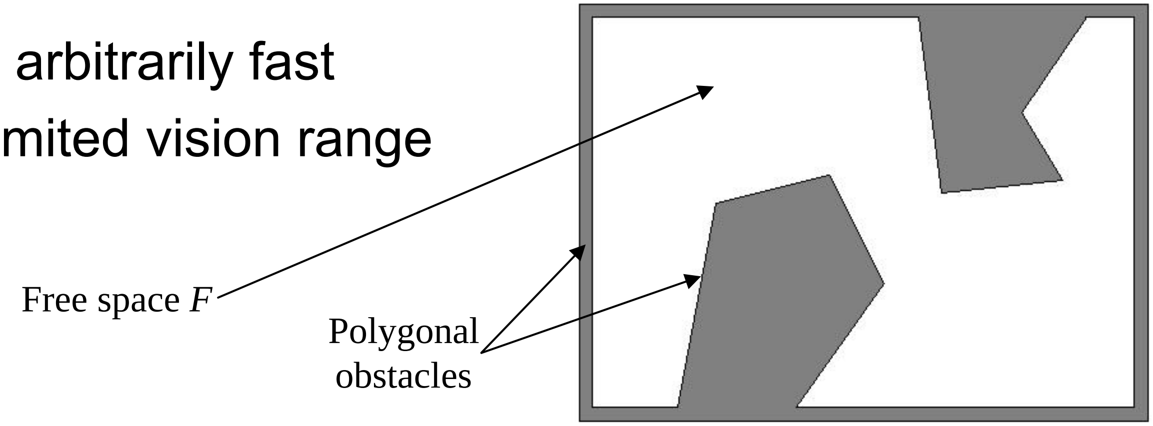
- Assumptions

  -Polygonal environment, freespace denoted *F*

  -If the evader is within line of sight of the pursuer, it has been "captured"

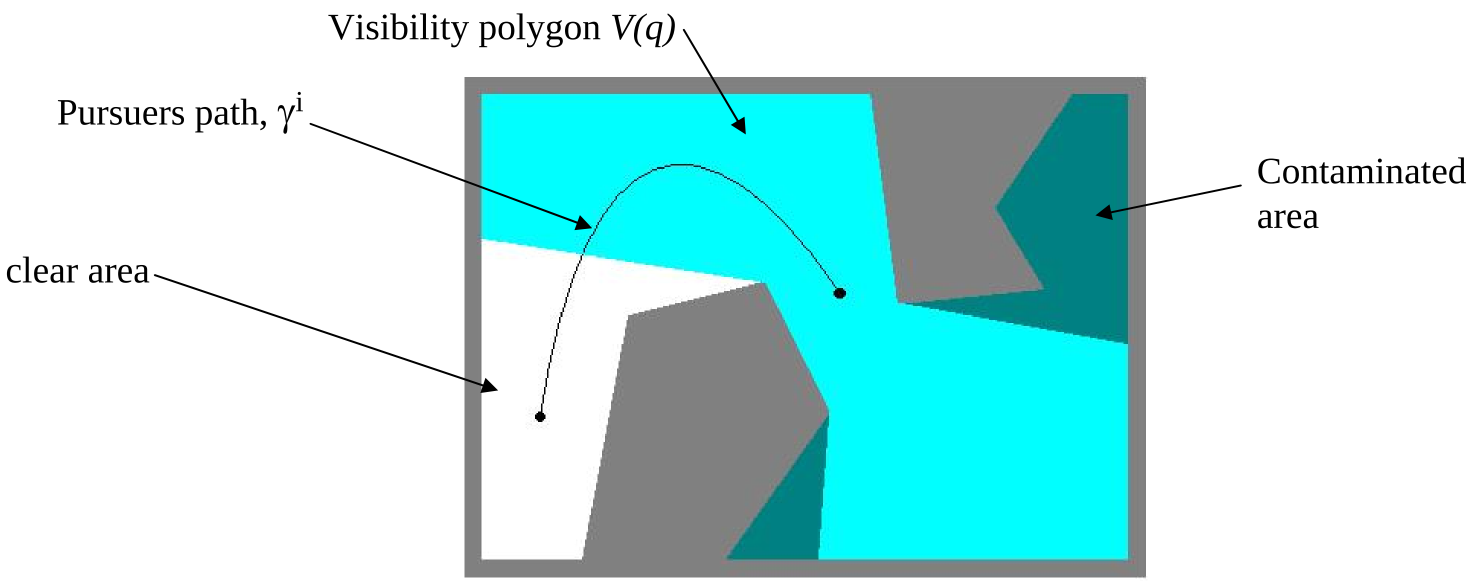
  -Evaders can move arbitrarily fast

  -Pursuers have unlimited vision range



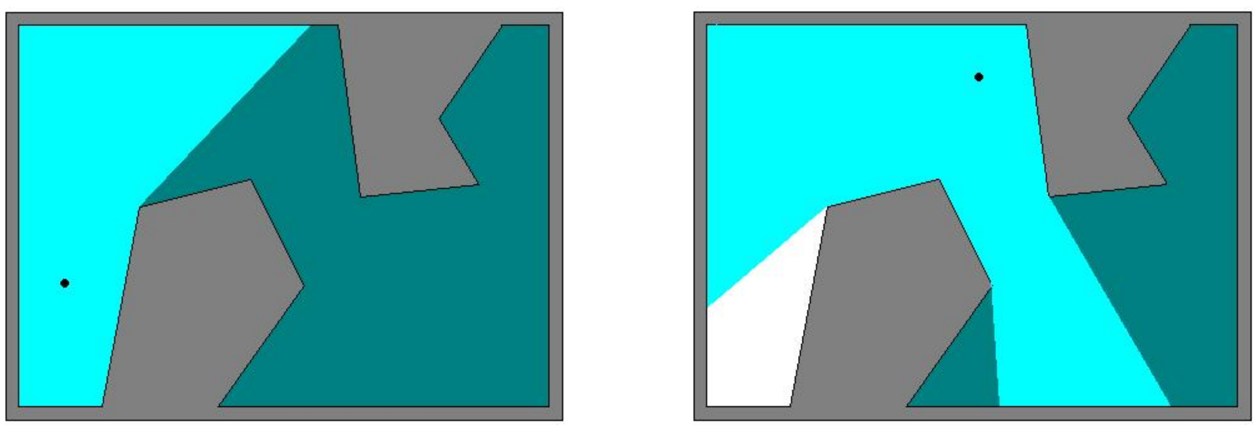
Terminology and Definitions

- $\gamma^i(t)$ position of i^{th} pursuer at time t
- $V(q)$ set of points in F visible from $q \in F$
- *Contaminated:* region of *F* that might contain the evader
- *Cleared:* region that is not contaminated.
- *Recontaminated:* A region that was contaminated, then cleared, and then contaminated
- *Solution strategy:* A strategy γ for any given evader path if there is at some time a point where the pursuer sees the evader.



Information State and Space

- Let $q \in F$ be the current pursuer position, let $S \subseteq F$ be the set of all contaminated points in F, then $\eta = (q, S)$ is an ***information state***. In other words, it is a set of data that uniquely describes state of the environment at a given point. Is $\eta = (q, S)$ a function of time?
- The set of all possible information states is the ***information space***.



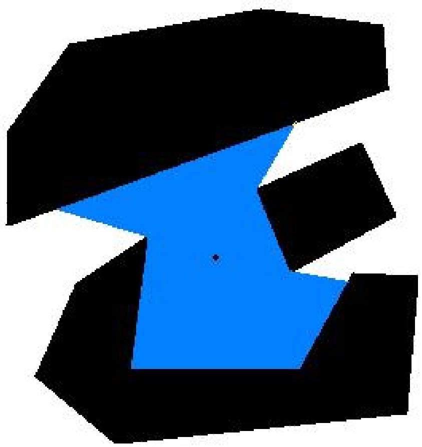
Two different information states

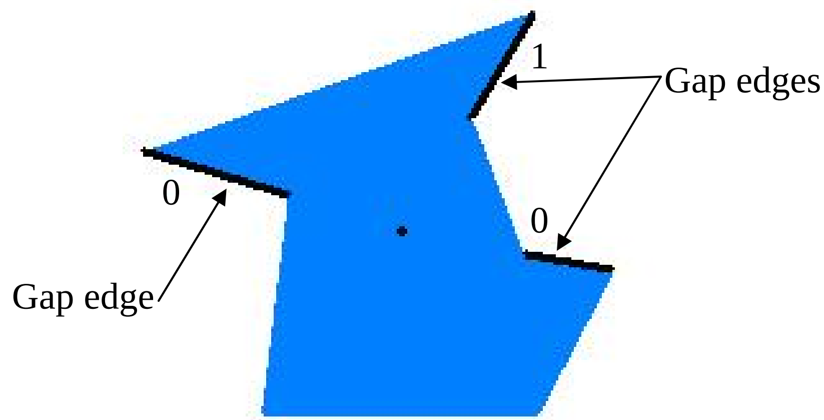
Information State

•How do we use the information state in our search for the evader?

At a point *q*, the edges of the visibility polygon *V(q)* alternate between being on the boundary of *F* and the interior of *F.* We will call the edges of *V(q)* that enter the free space *gap edges.*

The blue area is *V(q)* for the point shown





•We can assign each gap edge a binary value– if the edge borders a contaminated region, it is assigned a "1", and "0" for all other edges.

•For each point *q*, we can assign a binary vector *B(q)* that contains all the gap edge labels

•The pair *(q, B(q))* then uniquely describes the information state, for example (q, {010})

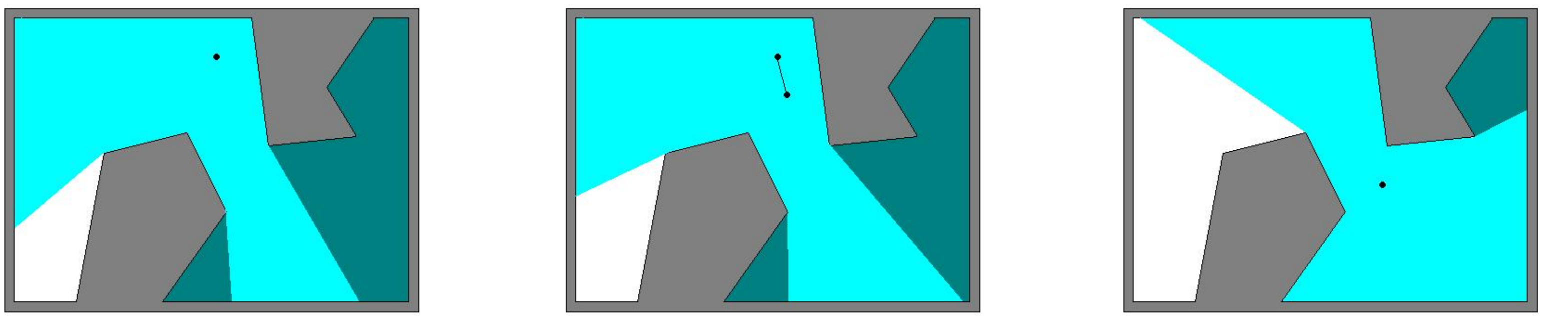
Recall a contaminated region *might* have an evader

Conservative Regions

- A connected set $D \subseteq F$ is *conservative* if for all $q \in F$, and for all γ:[t0, t1] $\rightarrow D$ such that γ is continuous and $\gamma(t_0) = \gamma(t_1) = q$, then the same information state is obtained.

Ok, but what does that mean?

As long as we stay in the same conservative region, the information state will not change. If we break the free space *F* down into conservative regions, then if we visit one point in a region, we will obtain the same information that we would have gotten from any other point in the same region



Position q1 Position q2 (with path shown from q1) Position q3

Moving the robot from q1 to q2, the information state does not change.

But when we move from q1 to q3 the information state does change- the region in the lower center is cleared

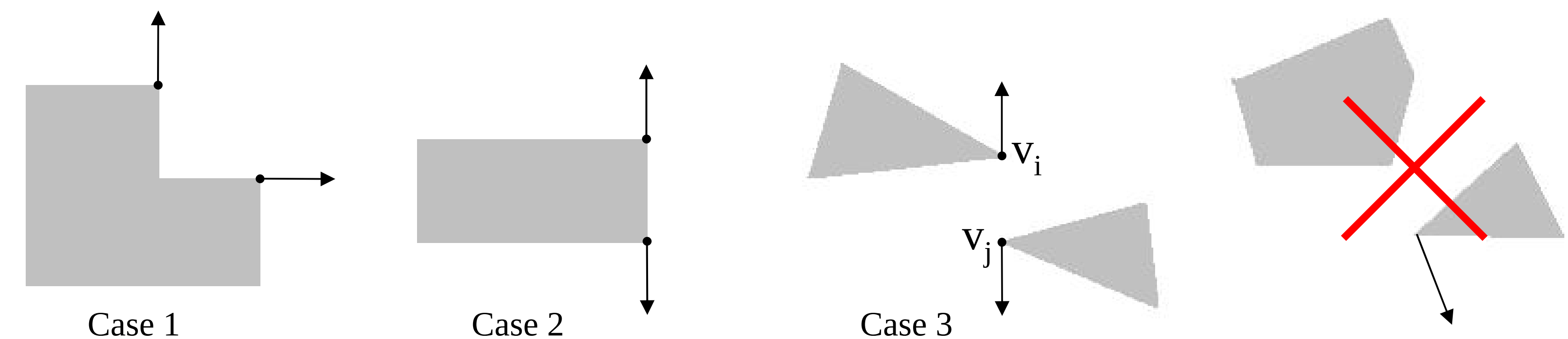
Thus q1 and q2 are in the same conservative region

Constructing Conservative Regions

So how do we construct them?

Draw rays extending the edges of obstacles until they hit another obstacle. We also draw rays extending away from any two vertices that don't have an obstacle between them.

The three general cases are shown below.



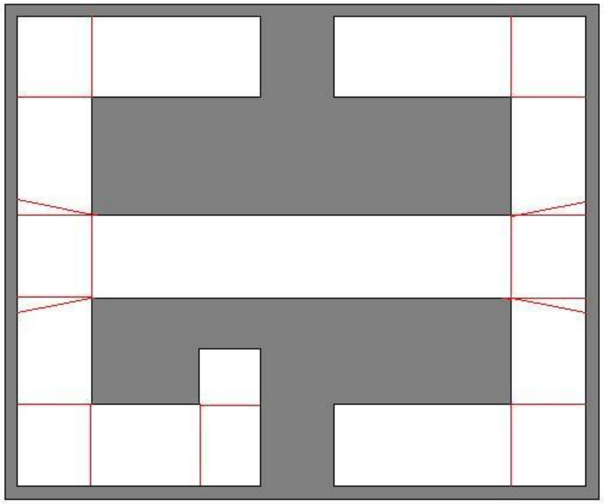
Extend edges in all possible directions

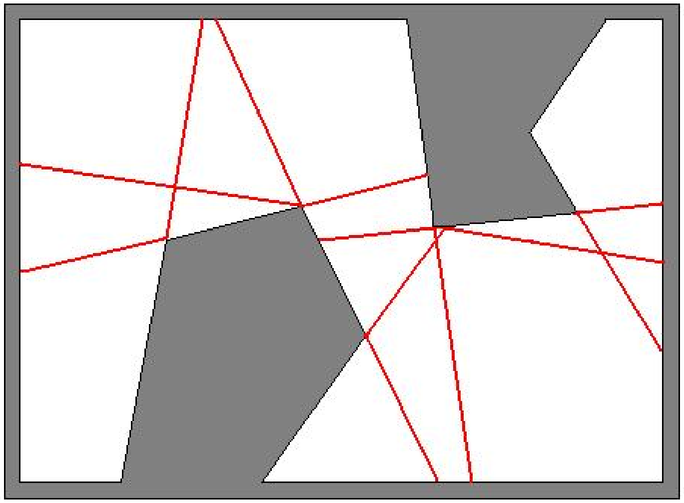
(Note that we have not shown all the possible rays for the obstacles)

Extend pairs of vertices outwards only if it is free in both directions along a line through the two vertices

For Case 3: Let $v_i \in C_i$, $v_j \in C_j$. Then if $\lambda v_i + (1-\lambda) v_j \in F$ for all $\lambda \in (-\varepsilon, 1+\varepsilon)$ then we draw a ray extending from v_i away from v_j and vice versa until they hit an obstacle (except lamda = 0,1??)

Examples of Conservative Regions

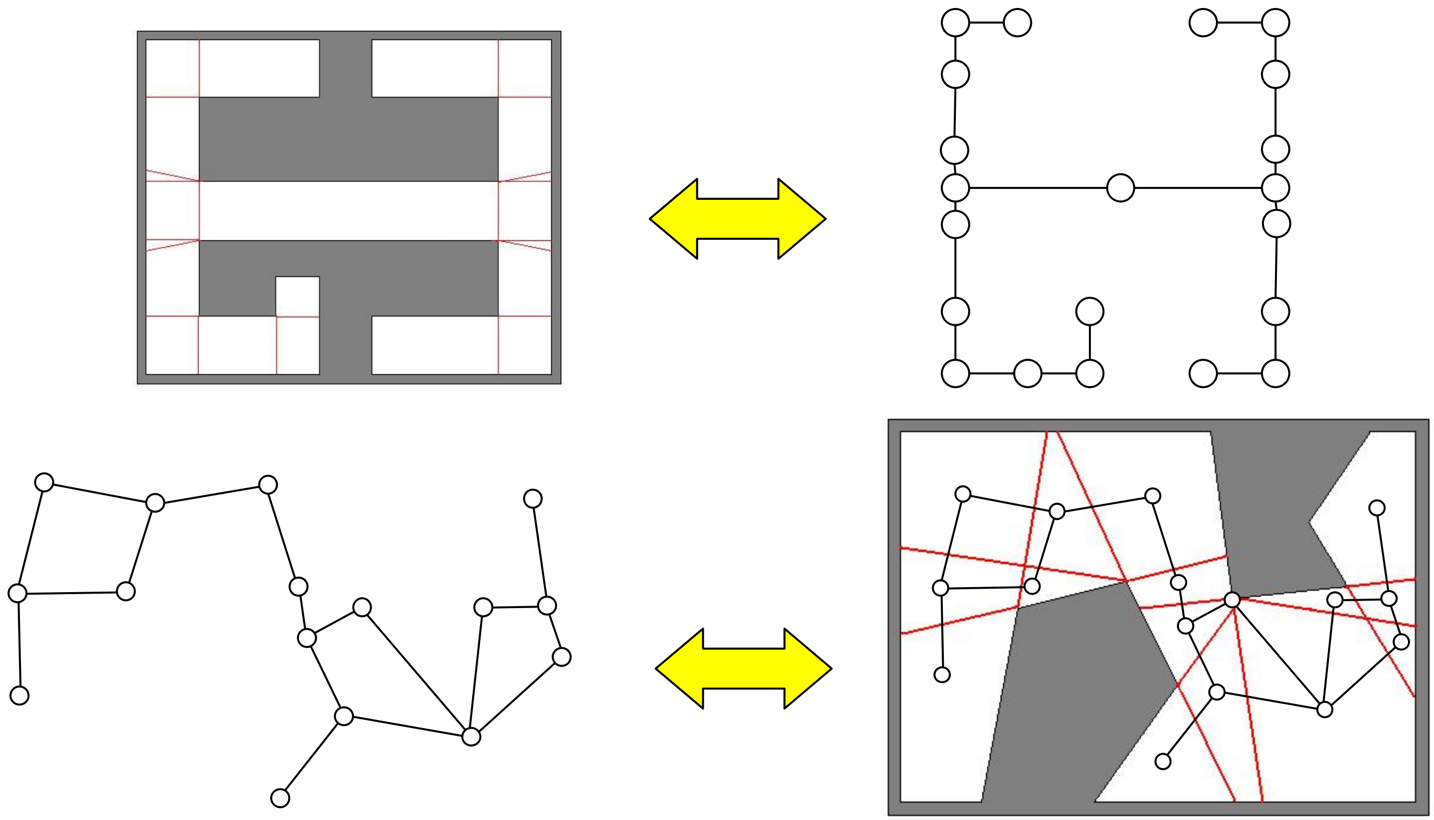




- Because the information state is the same in a given conservative region, all we really need to do is visit the center of each region to obtain the state.

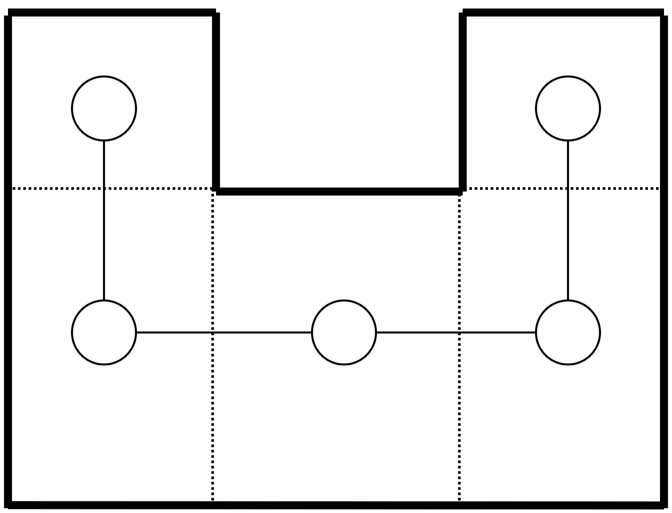
Conservative Regions to Graphs

- Now that we have the space decomposed into conservative regions, we can represent each region as the node on a finite, planar graph *G*.

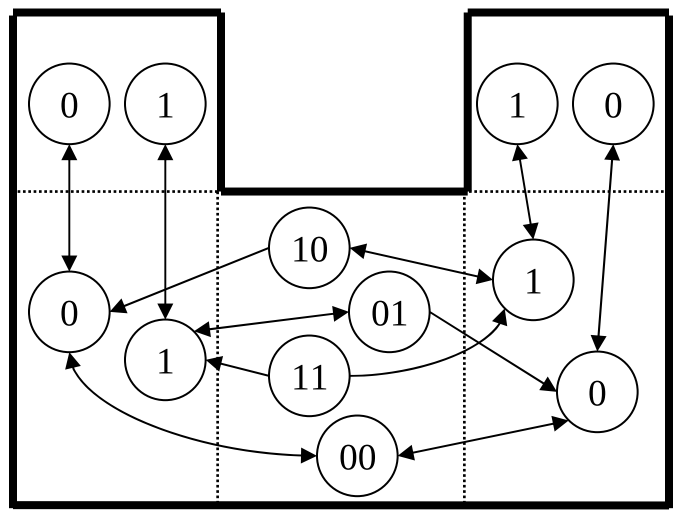


Directed Information Graph

- Given a graph G, we can derive the *information graph* G_I that includes the labels for the gap edges. For each node in G, we include a set of vertices in G_I, one for each possible gap edge label.
- For example, for a given point and region $q \in D$, there are two gap edges in $B(q)$. But we include all possible combinations of $B(q)$ in G_I: {00, 01, 10, 11}. Thus we can identify a vertex in G_I with the pair $(q, B(q))$



Freespace with overlaid graph G



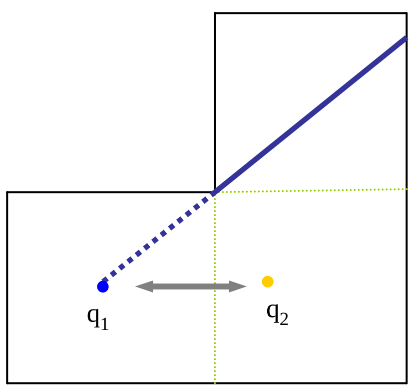
Information Graph G_I

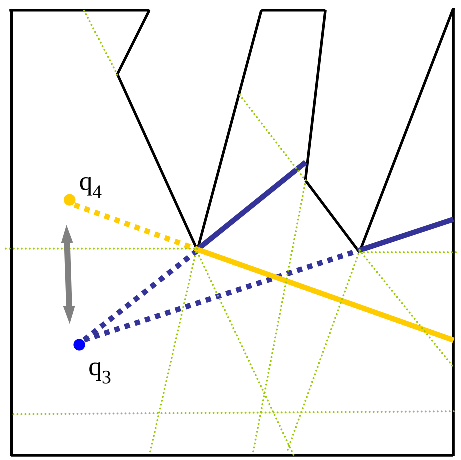
Gap Edge transitions

- What happens to the gap edges when we move from region to region?

There are four cases:

1. *A gap edge disappears* ⇒Don't worry about it, the area has been cleared
2. *A gap edge appears* ⇒ Assign it a "0" (clear) label
3. *Two or more gap edges merge into one* ⇒ If any of the original edges had a "1" (contaminated) , then the new edge will be a "1"
4. *One gap edge splits into two* ⇒ Assign new edges the same value as the old edge





•Moving from q1 to q2, the single gap edge disappears **(Case 1)**

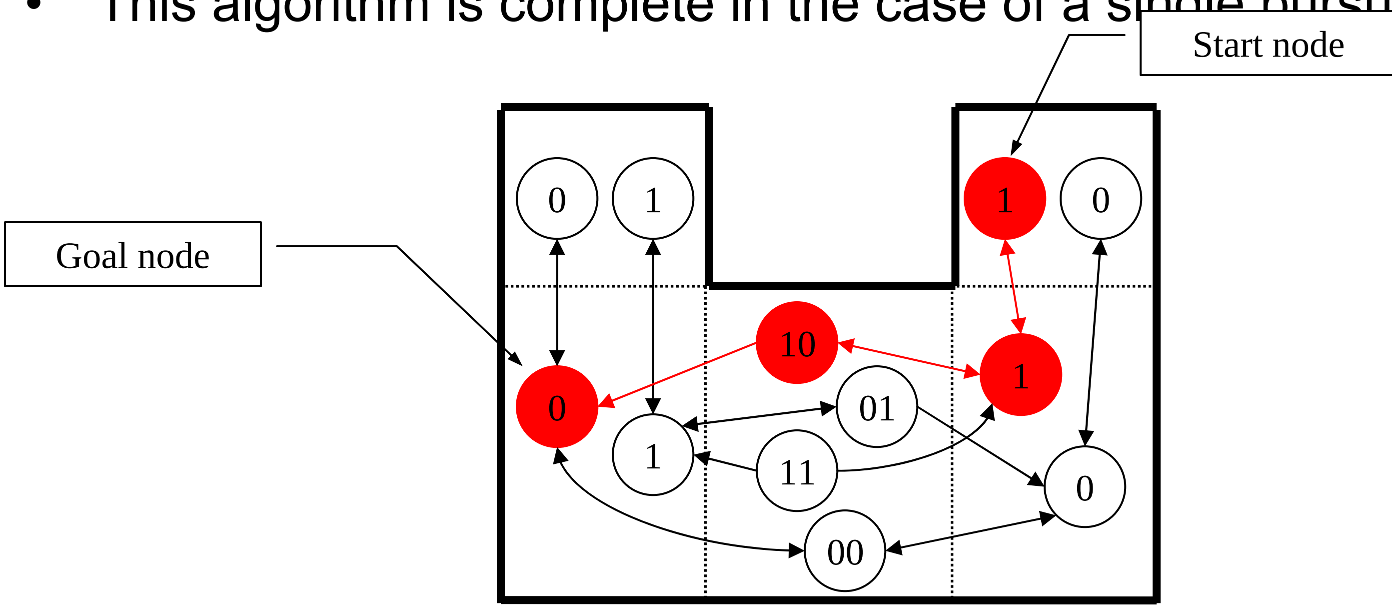
• From q2 to q1, a gap edge reappears **(Case 2)**

• Moving from q3 to q4, two gap edges merge into one **(Case 3)**

•From q4 to q3, a single edge splits into two **(Case 4)**

Graph search and solution

- The final step is to simply apply any graph searching algorithm to the information graph G_I and update the vector *B(q)* for each region.
- Any node on G_I of the form *(q, B(q))* such that *B(q)* = “00…0” (all gap edges are 0) or a node with no gap edges is a *goal node*.
- This algorithm is complete in the case of a single pursuer.

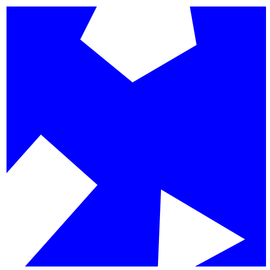


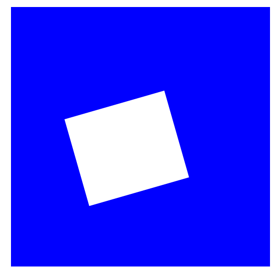
Worst case bounds

How many pursuers do you need to have to find an evader in a given space?

That depends on the geometry of the space

- For a simply connected free space, *F*, with *n* edges, *H(F)* = Θ(log *n*), where *H(F)* is the number of pursuers needed.
- For a free space *F* with *h* holes and *n* edges, $H(F) = \Theta(\sqrt{h} + \log n)$


Simply connected


Free space with hole

- Simply connected means all the edges can be connected into a single continuous path

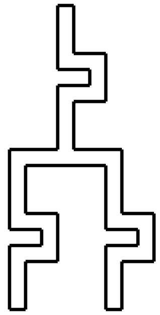
Why does the hole matter? If there is a hole, the evader can always be on the side opposite a single pursuer. Thus a space with one hole requires two pursuers.

Quick review: O(n) = "at most" Ω(n) = "at least" Θ(n) = asymptotically equal

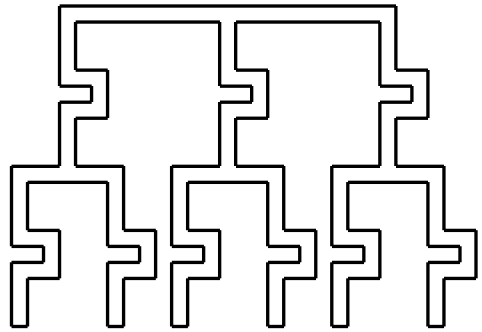
Intuition on Bounds for *H(F)*

- For a simply connected environment, ***H(F)*= Θ(log *n*).** We can see this by using a "Ω" shaped free space.

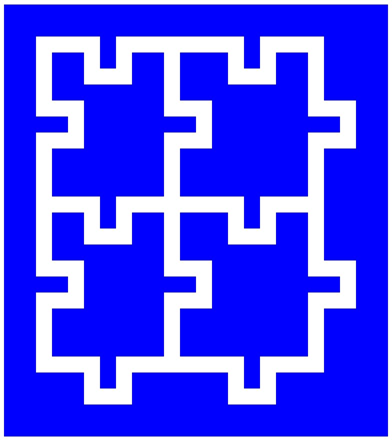
12 edges, one pursuer



36 edges, two pursuers



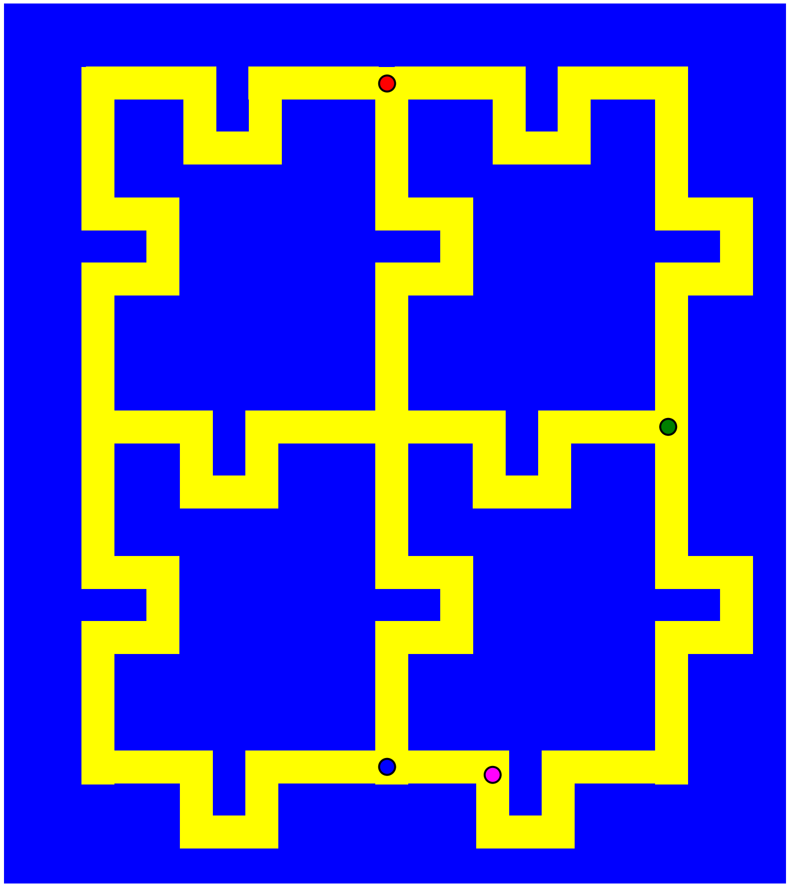
108 edges, three pursuers



4 holes, 111 edges, 4 pursuers

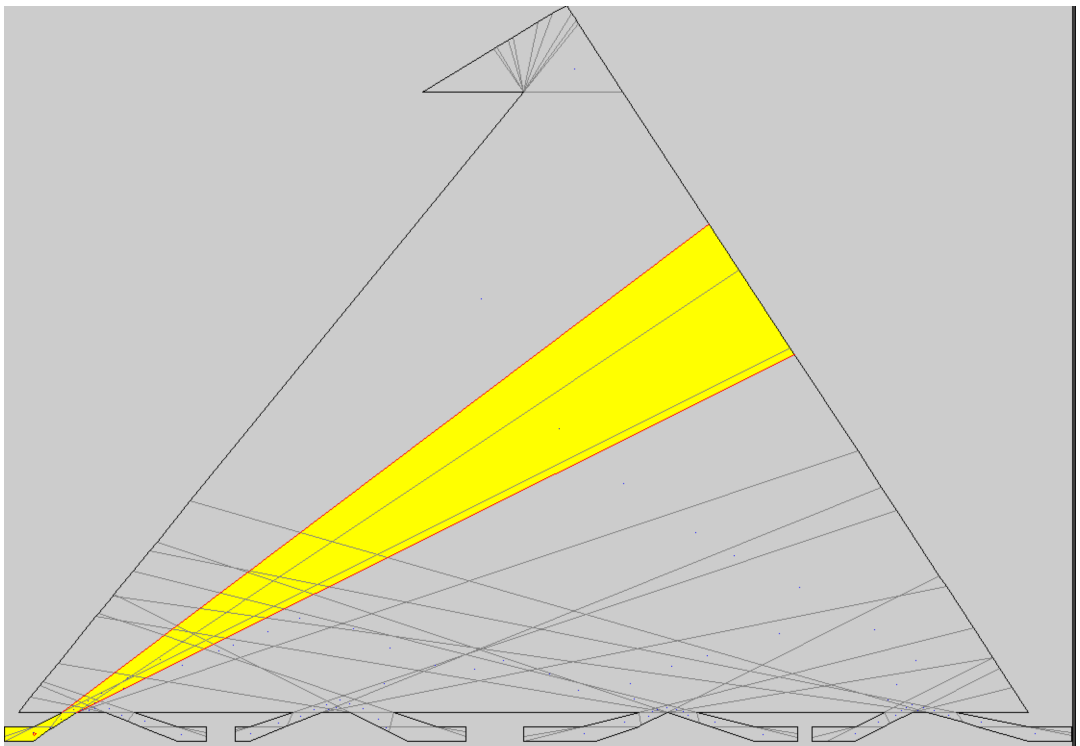
- For a space with h holes and n edges, $H(F) = \Theta(\sqrt{h} + \log n)$
- The $\sqrt{h}$ pursuers are used to divide the space into simply connected components, while the log(n) pursuers search the remaining space

Multiple Pursuers Demo



Recontamination

- There are some simply connected free spaces with *H(F)*=1 where recontamination will be required $\Omega(n)$ times



Here the peak will be recontaminated 3 times, requiring 2 extra visits to the peak (see web animation)

Conclusions / Questions

- The algorithm presented is complete for a single pursuer
- Any graph search algorithm will provide a solution once a information graph is extracted from the conservative region decomposition.
- Tight bounds exist for the number of pursuers necessary for a given free space.
- A complete and correct algorithm does not exist yet for *H(F)* > 1

Based on the paper

"A Visibility-Based Pursuit-Evasion Problem", Guibas, Latombe, LaValle, Lin, Motwani

Animations are on the web at: http://robotics.stanford.edu/groups/mobots/pe.html