Robotic Motion Planning: Cell Decompositions
(with some discussion on coverage)

Robotics Institute 16-735
http://www.cs.cmu.edu/~motionplanning
Howie Choset
http://www.cs.cmu.edu/~choset/
Homework 6

- Handout: Wed Oct 17th
- Due: Mon Nov 5th

For a two-dimensional configuration space, implement the Voronoi diagram using incremental methods.

- You may want to consider breaking your code down into separate modules
- Compute distance to the individual obstacles (assume you have a sensor with infinite sensor range)
- Move away from the closest obstacle until reaching double equidistance
- Compute the tangent space of the GVD by passing a line through the two closest points on the two closest obstacle and taking the line perpendicular to it
- Correct by rotating 90 degrees and move onto the GVD
- Detect the nodes (meet points)
- Compute tangents pairwise of three closest obstacles
- Create the graph structure
- On your web page, show at least two images in different environments.
Reminding Show-Tell (Monday)

• Prepare slides and send them to Hyungpil by Sunday 9pm
• Each group has maximum 75min / 8group = 9 min/group
• So, prepare 8-9 slides to report your progress
• Please include the final demo that you plan to do
Definition

Exact Cellular Decomposition (as opposed to approximate)

- $\nu_i$ is a cell
- $\text{int}(\nu_i) \cap \text{int}(\nu_j) = \emptyset$ if and only if $i \neq j$
- $F_s \cap (\text{cl}(\nu_i) \cap \text{cl}(\nu_j)) \neq \emptyset$ if $\nu_i$ and $\nu_j$ are adjacent cells
- $F_s = \bigcup_i (\nu_i)$
Exact Cell vs. Approximate Cell

**Exact cell decomposition**
- $\nu_i$ is a cell
- $\text{int}(\nu_i) \cap \text{int}(\nu_j) = 0$ iff $i \neq j$
- $F_s \cap (\text{cl}(\nu_i) \cap \text{cl}(\nu_j)) \neq 0$
  - if $\nu_i$ and $\nu_j$ are adjacent cells
- $F_s = \cup_i(\nu_i)$

- **Cell**: simple region
- **Approximate cell**

**Neighboring cells share boundary**

**“Exact” cell decomposition**
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 (point location query)
  – 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

\[ \text{Diagram with labeled trapezoids: } c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15} \]
Trapezoidal Decomposition
Trapezoidal Decomposition
Trapezoidal Decomposition Path
Implementation

- Input is vertices and edges
- Sort n vertices $O(n \log n)$ – ex. Binary tree sort
- 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 $e_{\text{LOWER}}$ and $e_{\text{UPPER}}$

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

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

Out

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

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

In

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

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

Middle

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

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

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

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

Anything left to the sweep line is deleted
Anything right to the sweep line is inserted

RI 16-735 Howie Choset
Example

\[ L : \emptyset \to \{ e_8, e_{13} \} \]

\( e_{\text{lower}} \) and \( e_{\text{upper}} \) are both to the right of the sweep line

- Insert \( e_{\text{lower}} \) and \( e_{\text{upper}} \) into the list
- \((..., e_{\text{LOWER}}, e_{\text{UPPER}}, ...) \to (...), e_{\text{LOWER}}, e_{\text{lower}}, e_{\text{upper}}, e_{\text{UPPER}}, ...)\)
Example

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

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

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

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

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

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

\[ \{e_9, e_1, e_2, e_6, e_5, e_{12}\} \rightarrow \{e_9, e_6, e_5, e_{12}\} \]

delete \( e_{\text{lower}} \) and \( e_{\text{upper}} \) from the list

\[
(\ldots, e_{\text{LOWER}}, e_{\text{lower}}, e_{\text{upper}}, e_{\text{UPPER}}, \ldots) \\
(\ldots, e_{\text{LOWER}}, e_{\text{UPPER}}, \ldots)
\]

RI 16-735 Howie Choset
Trapezoidal Decomposition
A Search Structure

- What data structure supports point location queries?
- Does $q$ lie to the left or to the right of the vertical line thru the endpoint stored at this node?
- Does $q$ lie above or below the segment $s$ stored here?
Randomized Incremental Algorithm

- Simultaneously construct the Trapezoidal map and a data structure

Algorithm: TrapezoidalMap(E)

\textit{Input}: A set \( E \) of \( n \) non-crossing line segments

\textit{Output}: \textbf{Trapezoidal map} \( T(E) \) and a search structure \( D \)

1. Initialize the trapezoidal map structure \( T \) and \( D \) for it
2. Compute a random permutation \( e_1, e_2, \ldots, e_n \) of the elements of \( E \)
3. \textbf{for} \( i = 1 \) to \( n \)
4. \hspace{1em} \textbf{do} Find the set \( \Delta_0, \ldots, \Delta_k \) of trapezoids in \( T \) intersected by \( e_i \)
5. \hspace{2em} remove \( \Delta_0, \ldots, \Delta_k \) from \( T \), replace them with new trapezoids that appear due to insertion of \( e_i \)
6. \hspace{2em} remove the leaves for \( \Delta_0, \ldots, \Delta_k \) from \( D \), create leaves for the new trapezoids. Link new leaves to the existing ones
Algorithm: FollowSegment(T,D,s_i)

**Input:** A trapezoidal map T, a search structure D for T, and a new segment s_i

**Output:** The sequence of trapezoidal intersected by s_i

1. Let p and q be the left and right endpoint of s_i
2. Search with p in D to find Δ₀
3. j ← 0
4. **While** q lies to the right of rightp(Δᵢ)
5. **do if** rightp(Δᵢ) lies above s_i
6. **then** Let Δᵢ₊₁ be the lower right neighbor of Δᵢ
7. **else** Let Δᵢ₊₁ be the upper right neighbor of Δᵢ
8. j ← j+1
9. **Return** Δ₀,..., Δᵢ
RI 16-735 Howie Choset

RIA Example

T(E)

A

C

D

B

\{e_1, e_2, e_3\}

x-nodes

v_1

right

left

y-nodes

e_1

above

below

D

v_3

v_1

v_2

v_3

A

C

D

B
RIA Example
RIA Example

{e1, e2, e3}
RIA Example
RIA Example
RIA Example: point location query

\[ \{e_1, e_2, e_3\} \]
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.

RI 16-735 Howie Choset
Complete Coverage
Morse Decomposition in Terms of Critical Points

- **Slice function:** \( h(x,y) = x \)

- **At a critical point** \( x \) of \( h \), \( \nabla h(x) = \nabla m(x) \) \( \text{where } M = \{ x \mid m(x) = 0 \} \)
$1$-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

RI 16-735 Howie Choset
Incremental construction (cont’d)

**Stage 1**

**Stage 2**

**Stage 3**
Incremental construction (cont’d)

Stage 1

Stage 2

Stage 3

Stage 4

RI 16-735 Howie Choset
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

$\text{Canny } \pi_1: Q \rightarrow \mathbb{R}$
Morse Decomposition
\[ h(x,y) = x^2 + y^2 \]
Morse Decomposition

\[ h(x,y) = |x| + |y| \]
Morse Decomposition

\[ h(x,y) = \tan(y/x) \]
Brushfire Decomposition
Brushfire Decomposition

\[ h(x, y) = D(x, y) \]
Brushfire Decomposition
\[ h(x,y) = D(x,y) \]
Brushfire Decomposition

\[ h(x,y) = D(x,y) \]
Brushfire Decomposition

\[ h(x,y) = D(x,y) \]
Brushfire Decomposition Coverage Path
Wavefront Decomposition
Notation

• A slice is a codimension one manifold \((Q_\lambda)\)
• Slices are parameterized by \(\lambda\)
  – varying \(\lambda\) sweeps a slice through the space
• The portion of the slice in the free configuration space \((Q_{\text{free}})\) is \(Q_{\text{free} \lambda}\)
• \(Q_{\text{free} \lambda} = Q_\lambda \cap Q_{\text{free}}\)
Slice Definition

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