Randomized Algorithms in Computational Geometry

Peter Palfrader

July 2013

Outline

Outline

Casting Problem

- Question: Given a polyhedral object $\mathcal{P}$, can we produce it by casting it from a single mold and then remove it by translations.

Casting Problem

- Question: Given a polyhedral object $\mathcal{P}$, can we produce it by casting it from a single mold and then remove it by translations.
- Necessary condition: $\mathcal{P}$ can be removed in direction $\vec{d}$ if $\vec{d}$ makes an angle greater than 90° with the outside normal of all ordinary faces.

Casting Problem, cont'd

How do we find $\vec{d}$, if it even exists?

- We will see an $\mathcal{O}(n)$ expected runtime algorithm that gives us a $\vec{d}$, given a fixed top face.

Casting Problem, cont'd

How do we find $\vec{d}$, if it even exists?

- We will see an $\mathcal{O}(n)$ expected runtime algorithm that gives us a $\vec{d}$, given a fixed top face.
- This results in an $\mathcal{O}(n^2)$ algorithm overall if we have to try different faces for the top face.

Half-Plane Intersection

- Constraints: $H = h_1, h_2, \ldots, h_n$ of the form

$$h_i : a_i x + b_i y \leq c_i$$

- $h_i \mathrel{\widehat{=}}$ closed half-plane in $\mathbb{R}^2$
- $h_i \mathrel{\widehat{=}}$ set of possible $\vec{d}$ for each face f_i of $\mathcal{P}$.
- Goal: Find all points in the common intersection.

Divide & Conquer

procedure IntersectHalfPlanes(H)
  if $|H| = 1$ **then**
    return unique $h \in H$
  else
    split H into H_1, H_2
    $C_1 \leftarrow$ `IntersectHalfPlanes(`H_1`)`
    $C_2 \leftarrow$ `IntersectHalfPlanes(`H_2`)`
    $C \leftarrow$ `IntersectConvexRegions(`C_1`, `C_2`)`
    return C
  end if
end procedure

Divide & Conquer, Complexity

- Intersecting Convex Regions can be done in linear time.
- Thus:

$$T(N) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1. \\ \mathcal{O}(n) + 2T(n/2) & \text{if } n > 1. \end{cases}$$

Divide & Conquer, Complexity

- Intersecting Convex Regions can be done in linear time.
- Thus:

$$T(N) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1. \\ \mathcal{O}(n) + 2T(n/2) & \text{if } n > 1. \end{cases}$$

- This solves to:

$$T(n) = \mathcal{O}(n \log n)$$

Incremental Approach – Linear Programming

Linear Programming:

- Maximize $c_1 x_1 + c_2 x_2 + \ldots + c_d x_d$
- Subject to:

$$\begin{aligned} a_{1,1}x_1 + \ldots + a_{1,d}x_d &\leq b_1 \\ a_{2,1}x_1 + \ldots + a_{2,d}x_d &\leq b_2 \\ &\vdots \\ a_{n,1}x_1 + \ldots + a_{n,d}x_d &\leq b_n \end{aligned}$$

Incremental Approach – Linear Programming

Linear Programming:

- Maximize: $c_x p_x + c_y p_y$
- Subject to:

$$\begin{aligned} a_{1,x} p_x + a_{1,y} p_y &\leq b_1 \\ a_{2,x} p_x + a_{2,y} p_y &\leq b_2 \\ &\vdots \\ a_{n,x} p_x + a_{n,y} p_y &\leq b_n \end{aligned}$$

Linear Programming

- Possible results from LP:
  (i) Problem is *infeasible*.
  (ii) Feasible region is unbounded in direction of $\vec{c}$.
  (iii) Feasible region is bounded by an edge e normal to $\vec{c}$.
  (iv) There is a unique solution.

Linear Programming

- Possible results from LP:
  (i) Problem is *infeasible*.
  (ii) Feasible region is unbounded in direction of $\vec{c}$.
  (iii) Feasible region is bounded by an edge e normal to $\vec{c}$.
  (iv) There is a unique solution.
- We would like to avoid (ii) so we add additional constraints: m_1, m_2 with: $m_1 := |p_x| \leq M, m_2 := |p_y| \leq M$.

Linear Programming

- Possible results from LP:
  1. (i) Problem is *infeasible*.
  2. (ii) Feasible region is unbounded in direction of $\vec{c}$.
  3. (iii) Feasible region is bounded by an edge e normal to $\vec{c}$.
  4. (iv) There is a unique solution.
- We would like to avoid (ii) so we add additional constraints: m_1, m_2 with: $m_1 := |p_x| \leq M, m_2 := |p_y| \leq M$.
- To avoid (iii) we establish a convention: When there are several optimal points, pick the lexicographically smallest one.

Incremental Approach, cont'd

Let

- $H_i := \{m_1, m_2, h_1, h_2, \ldots, h_i\}$
- $C_i := m_1 \cap m_2 \cap h_1 \cap h_2 \cap \ldots \cap h_i$

Incremental Approach, cont'd

Let

- $H_i := \{m_1, m_2, h_1, h_2, \ldots, h_i\}$
- $C_i := m_1 \cap m_2 \cap h_1 \cap h_2 \cap \ldots \cap h_i$

Observe:

- C_i has a unique optimal vertex, v_i that maximizes $v_i \cdot \vec{c}$.

Incremental Approach, cont'd

Let

- $H_i := \{m_1, m_2, h_1, h_2, \ldots, h_i\}$
- $C_i := m_1 \cap m_2 \cap h_1 \cap h_2 \cap \ldots \cap h_i$

Observe:

- C_i has a unique optimal vertex, v_i that maximizes $v_i \cdot \vec{c}$.
- $C_0 \supseteq C_1 \supseteq C_2 \supseteq \ldots \supseteq C_n = C$

Incremental Approach: Step

- Have v_i.
- Step $i \longrightarrow i+1$
- If $v_i \in h_{i+1}$: $v_{i+1} = v_i$
- If $v_i \notin h_{i+1}$:
  - $C_{i+1} = \emptyset$, or
  - $v_{i+1} \in \ell_{i+1}$ where ℓ_{i+1} is the line bounding h_{i+1}.

Incremental Algorithm

```
procedure 2DBoundedLP(H)
    v_0 ← corner of C_0 = {m_1, m2}
    for i ← 0 ... n − 1 do
        if v_i ∈ h_{i+1} then
            v_{i+1} ← v_i
        else
            v_{i+1} ← point p on ℓ_{i+1} that
                          maximizes c⃗ · p subject to H_i
            if v_{i+1} = NULL then
                return NULL
            end if
        end if
    end for
    return v_n
end procedure
```

Incremental Approach: Complexity

- Finding that p on ℓ can be done in linear time.

Incremental Approach: Complexity

- Finding that p on ℓ can be done in linear time.
- Worst case: we have to do that every step of the way
- Therefore: Needs $\mathcal{O}(n \cdot n) = \mathcal{O}(n^2)$ time.

Incremental Approach: Complexity

- Finding that p on ℓ can be done in linear time.
- Worst case: we have to do that every step of the way
- Therefore: Needs $\mathcal{O}(n \cdot n) = \mathcal{O}(n^2)$ time.
- That's not quite the linear time algorithm we were promised...

Incremental Algorithm

procedure 2DBOUNDEDLP(H)
  $v_0 \leftarrow$ corner of $C_0 = \{m_1, m2\}$

  for do $i \leftarrow 0 \dots n-1$
    if $v_i \in h_{i+1}$ **then**
      $v_{i+1} \leftarrow v_i$
    else
      $v_{i+1} \leftarrow$ point p on ℓ_{i+1} that maximizes $\vec{c} \cdot p$ subject to H_i
      if $v_{i+1} =$ NULL **then**
        return NULL
      end if
    end if
  end for
  return v_n
end procedure

Randomized Algorithm

procedure 2DRandomizedBoundedLP(H)
    $v_0 \leftarrow$ corner of $C_0 = \{m_1, m2\}$
    $H \leftarrow$ `randomPermutation(`H`)`
    for do $i \leftarrow 0 \ldots n-1$
        if $v_i \in h_{i+1}$ **then**
            $v_{i+1} \leftarrow v_i$
        else
            $v_{i+1} \leftarrow$ point p on ℓ_{i+1} that maximizes $\vec{c} \cdot p$ subject to H_i
            if $v_{i+1} =$ NULL **then**
                return NULL
            end if
        end if
    end for
    return v_n
end procedure

Analysis

- Let $X_i = \begin{cases} 0 & \text{if } v_i \text{ stays the same.} \\ 1 & \text{if } v_i \text{ needs updating.} \end{cases}$

Analysis

- Let $X_i = \begin{cases} 0 & \text{if } v_i \text{ stays the same.} \\ 1 & \text{if } v_i \text{ needs updating.} \end{cases}$
- Then, total cost $T = \sum_{i=1}^{n} \mathcal{O}(i) \cdot X_i$

Analysis

- Let $X_i = \begin{cases} 0 & \text{if } v_i \text{ stays the same.} \\ 1 & \text{if } v_i \text{ needs updating.} \end{cases}$
- Then, total cost $T = \sum_{i=1}^{n} \mathcal{O}(i) \cdot X_i$
- $T = E[\sum_{i=1}^{n} \mathcal{O}(i) \cdot X_i] = \sum_{i=1}^{n} \mathcal{O}(i) \cdot E[X_i]$

Backwards Analysis

- $T = \sum_{i=1}^{n} \mathcal{O}(i) \cdot E[X_i]$
- But what is $E[X_i]$?

Backwards Analysis

- $T = \sum_{i=1}^{n} \mathcal{O}(i) \cdot E[X_i]$
- But what is $E[X_i]$?
- Look at a specific fixed point in the algorithm
- What are the chances we have just updated v_i?

Backwards Analysis

- $T = \sum_{i=1}^{n} \mathcal{O}(i) \cdot E[X_i]$
- But what is $E[X_i]$?
- Look at a specific fixed point in the algorithm
- What are the chances we have just updated v_i?
- We updated v_i if v_i is not on an extreme vertex of C_{i-1}, that is, h_i is one of the half planes that define v_i.
- Half planes are sorted randomly, so the probability is at most $\frac{2}{i}$.

Backwards Analysis

- $T = \sum_{i=1}^{n} \mathcal{O}(i) \cdot E[X_i]$
- But what is $E[X_i]$?
- Look at a specific fixed point in the algorithm
- What are the chances we have just updated v_i?
- We updated v_i if v_i is not on an extreme vertex of C_{i-1}, that is, h_i is one of the half planes that define v_i.
- Half planes are sorted randomly, so the probability is at most $\frac{2}{i}$.
- $E[X_i] \leq \frac{2}{i}$.
- $T \leq \sum_{i=1}^{n} \mathcal{O}(i) \cdot \frac{2}{i} \in \mathcal{O}(n)$, expected.

Casting Problem

Summary:

- Have $\mathcal{O}(n)$ expected algorithm that tells us if a polyhedron $\mathcal{P}$ with a given top face can be removed from the mold.
- Therefore have $\mathcal{O}(n^2)$ algorithm to determine if $\mathcal{P}$ can be cast at all.

Outline

1 Casting Problem

2 Smallest Enclosing Disk

3 Point Location

4 Delaunay Triangulations

Smallest Enclosing Disk

- Problem: Given a set of points in the plane, find the smallest disk that covers all of them.

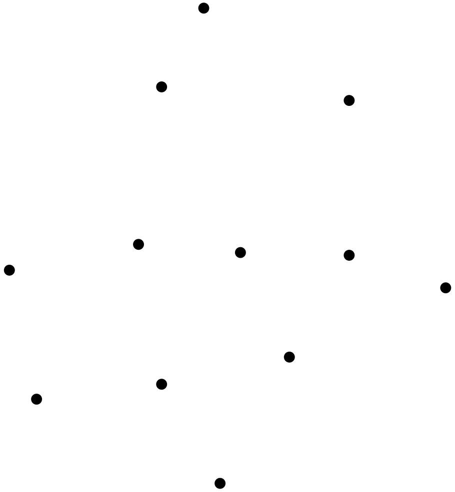
Smallest Enclosing Disk

- Problem: Given a set of points in the plane, find the smallest disk that covers all of them.
- Naive approaches do not perform very well.

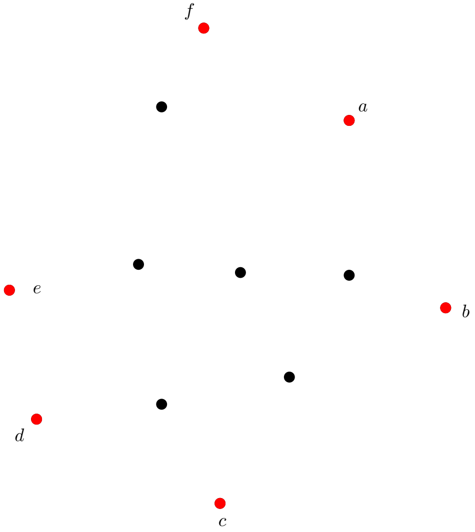
Smallest Enclosing Disk

- Problem: Given a set of points in the plane, find the smallest disk that covers all of them.
- Naive approaches do not perform very well.
- Given the *Farthest point Voronoi Diagram*, can be solved in $\mathcal{O}(n)$ time.[SH75]

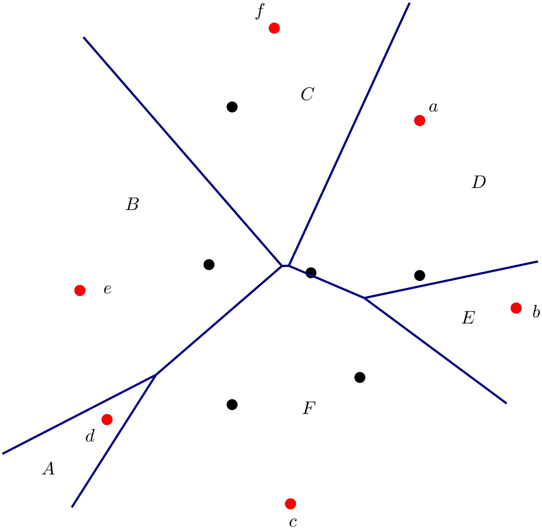
Farthest Point Voronoi Diagram



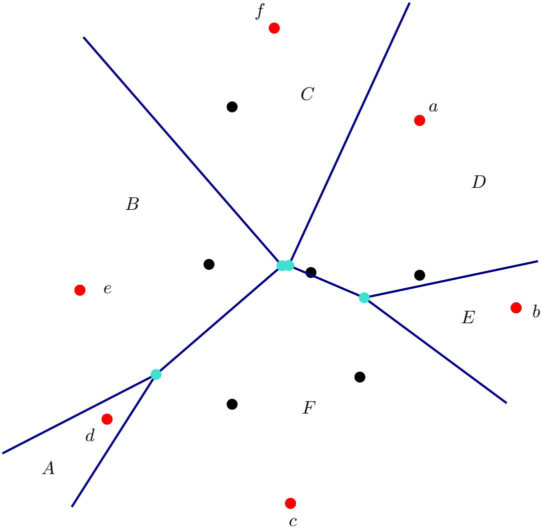
Farthest Point Voronoi Diagram



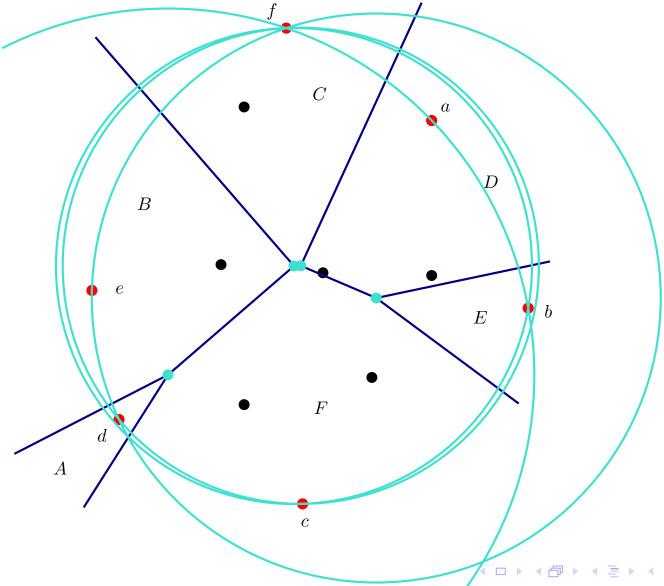
Farthest Point Voronoi Diagram



Farthest Point Voronoi Diagram



Farthest Point Voronoi Diagram



Incremental Algorithm

- Set of Points $P := \{p_1, p_2, \ldots, p_n\}$
- $P_i := \{p_1, \ldots, p_i\}$
- $D_i :=$smallest disk enclosing P_i

Incremental Algorithm

- Set of Points $P := \{p_1, p_2, \ldots, p_n\}$
- $P_i := \{p_1, \ldots, p_i\}$
- D_i :=smallest disk enclosing P_i

Incremental Step:
Observation:

(i) if $p_i \in D_{i-1}$ then $D_i = D_{i-1}$

(ii) if $p_i \notin D_{i-1}$ then p_i lies on ∂D_i

Smallest Enclosing Disk

procedure MINIDISK(P)
    $P \leftarrow$ `randomPermutation(`P`)`
    $D_2 \leftarrow$ smallest disk of $\{p_1, p_2\}$.
    for $i \leftarrow 3 \ldots n$ **do**
        if $p_i \in D_{i-1}$ **then**
            $D_i \leftarrow D_{i-1}$
        else
            $D_i \leftarrow$ `DiskWithPoint`$(\{p_1, \ldots, p_{i-1}\}, p_i)$
        end if
    end for
    return D_n
end procedure

Smallest Enclosing Disk, cont'd

procedure DiskWithPoint(P, q)
    $P \leftarrow$ `randomPermutation`(P)
    $D_1 \leftarrow$ smallest enclosing disk for $\{p_1, q\}$.
    for $j \leftarrow 2 \ldots n$ **do**
        if $p_j \in D_{j-1}$ **then**
            $D_j \leftarrow D_{j-1}$
        else
            $D_j \leftarrow$ `DiskWith2Points`($\{p_1, \ldots, p_{j-1}\}, p_j, q$)
        end if
    end for
    return D_n
end procedure

Smallest Enclosing Disk, cont'd

procedure DiskWith2Points(P, q_1, q_2)
    $D_0 \leftarrow$ smallest disk with $\{q_1, q_2\}$ on the boundary.
    for $k \leftarrow 1 \ldots n$ **do**
        if $p_k \in D_{k-1}$ **then**
            $D_k \leftarrow D_{k-1}$
        else
            $D_k \leftarrow$ disk with $\{q_1, q_2, p_k\}$ on the boundary.
        end if
    end for
    return D_n
end procedure

Correctness

- P a set of points
- R a possibly empty set of points
- $P \cap R = \emptyset$.
- $p \in P$

Correctness

- P a set of points
- R a possibly empty set of points
- $P \cap R = \emptyset$.
- $p \in P$

(i) If there is a disk that encloses P and has R on its boundary, then the smallest such disk is unique. We denote it by $md(P, R)$.

Correctness

- P a set of points
- R a possibly empty set of points
- $P \cap R = \emptyset$.
- $p \in P$

(i) If there is a disk that encloses P and has R on its boundary, then the smallest such disk is unique. We denote it by $md(P, R)$.

(ii) If $p \in md(P \setminus \{p\}, R)$, then $md(P, R) = md(P \setminus \{p\}, R)$

Correctness

- P a set of points
- R a possibly empty set of points
- $P \cap R = \emptyset$.
- $p \in P$

(i) If there is a disk that encloses P and has R on its boundary, then the smallest such disk is unique. We denote it by $md(P, R)$.

(ii) If $p \in md(P \setminus \{p\}, R)$, then $md(P, R) = md(P \setminus \{p\}, R)$

(iii) If $p \notin md(P \setminus \{p\}, R)$, then $md(P, R) = md(P \setminus \{p\}, R \cup \{p\})$

Complexity

- `DiskWith2Points()` runs in linear time.
- Ignoring calls to `DiskWith2Points()`, `DiskWithPoint()` also runs in linear time. What are the chances we call `DiskWith2Points()`?

Complexity

- `DiskWith2Points()` runs in linear time.
- Ignoring calls to `DiskWith2Points()`, `DiskWithPoint()` also runs in linear time. What are the chances we call `DiskWith2Points()`?
- The probability of having to call it is bounded by $\frac{2}{i}$.
- Thus, `DiskWithPoint()` runs in time $\mathcal{O}(n) + \sum_{i=2}^{n} \mathcal{O}(i) \cdot \frac{2}{i} \in \mathcal{O}(n)$, expected.

Complexity

- `DiskWith2Points()` runs in linear time.
- Ignoring calls to `DiskWith2Points()`, `DiskWithPoint()` also runs in linear time. What are the chances we call `DiskWith2Points()`?
- The probability of having to call it is bounded by $\frac{2}{i}$.
- Thus, `DiskWithPoint()` runs in time $\mathcal{O}(n) + \sum_{i=2}^{n} \mathcal{O}(i) \cdot \frac{2}{i} \in \mathcal{O}(n)$, expected.
- Using the same argument, we can see that `MiniDisk` runs also ins $\mathcal{O}(n)$ expected time.

Complexity

- `DiskWith2Points()` runs in linear time.
- Ignoring calls to `DiskWith2Points()`, `DiskWithPoint()` also runs in linear time. What are the chances we call `DiskWith2Points()`?
- The probability of having to call it is bounded by $\frac{2}{i}$.
- Thus, `DiskWithPoint()` runs in time $\mathcal{O}(n) + \sum_{i=2}^{n} \mathcal{O}(i) \cdot \frac{2}{i} \in \mathcal{O}(n)$, expected.
- Using the same argument, we can see that `MiniDisk` runs also ins $\mathcal{O}(n)$ expected time.
- Linear space complexity.

Summary

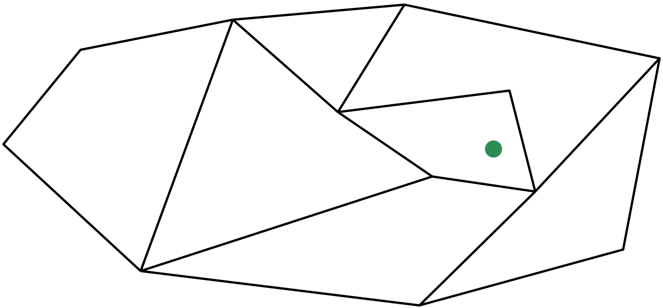
- We have seen an easy to implement algorithm to find the smallest enclosing disk for a set of points in the plane.
- The algorithm runs in expected linear time and linear storage.

Outline

1. Casting Problem
2. Smallest Enclosing Disk
3. **Point Location**
4. Delaunay Triangulations

Point location

- Problem: Given a partition of $\mathbb{R}^2$ and a query point q, find the face that q is in.

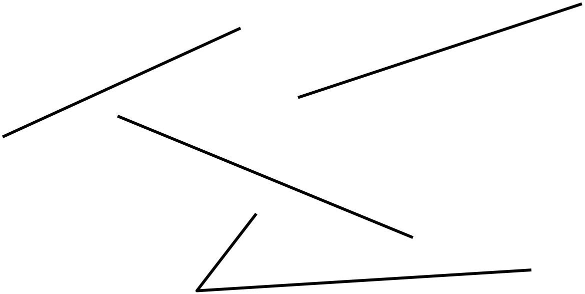


Point location

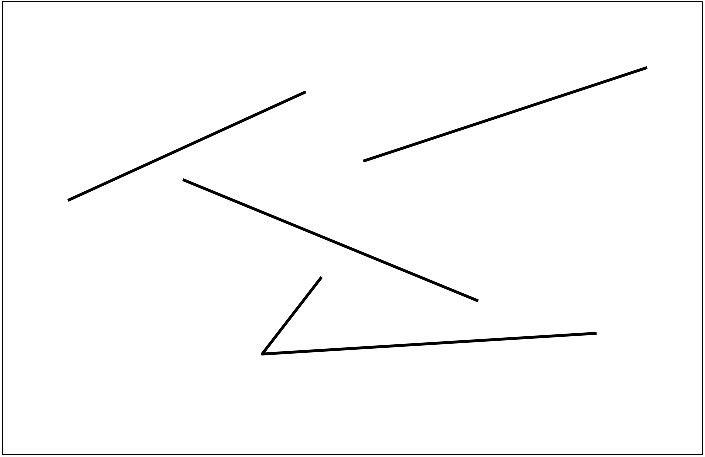
Plethora of Algorithms:

- Slab Method
- Chain Method
- Triangulation Refinement

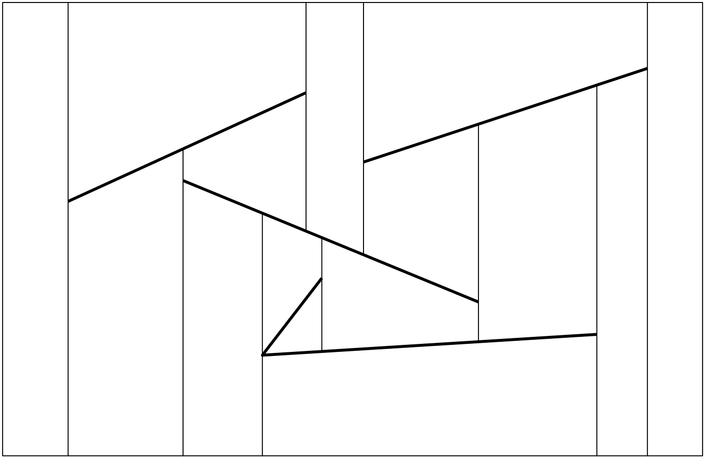
Trapezoidal Maps



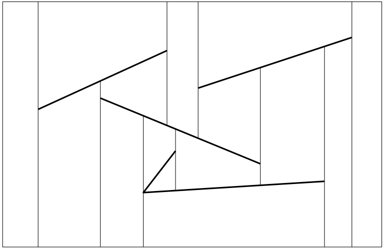
Trapezoidal Maps



Trapezoidal Maps



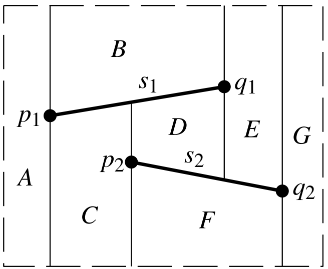
Trapezoidal Maps, cont'd

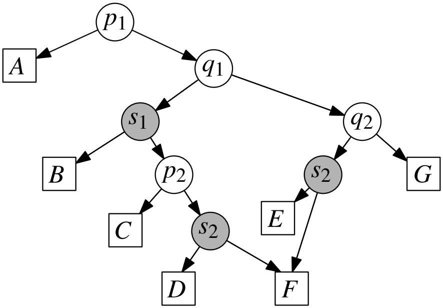


Each face Δ has

- up to two vertical edges.
- exactly two non-vertical edges, *bottom*(Δ) and *top*(Δ).
- a unique vertex that defines its left vertical edge, *leftp*(Δ).
- a unique vertex that defines its right vertical edge, *rightp*(Δ).
- up to four neighbors (two to the left, two to the right) - we do not count those above or below.

Search Trees





[Image: [M4]]

Incremental Randomized Algorithm

Basic Idea: Given a set $\mathcal{S}$ of line segments, construct the trapezoidal map $\mathcal{T}(\mathcal{S})$ incrementally, while at the same time also constructing the search structure $\mathcal{D}(\mathcal{T}(\mathcal{S}))$.

Incremental Randomized Algorithm, cont'd

procedure TRAPEZOIDALMAP(S)
    Find bounding box R.
    Initialize $\mathcal{T}$ and $\mathcal{D}$ for R.
    Shuffle S.
    for $i \leftarrow 1 \dots n$ **do**
        Find the set $\Delta_0, \Delta_1, \dots, \Delta_k$ of trapezoids in $\mathcal{T}$ that intersect s_i.
        Remove these trapezoids from $\mathcal{T}$ and replace them with new trapezoids that appear due to the intersection with s_i.
        Remove the leaves for $\Delta_0, \dots, \Delta_k$ from $\mathcal{D}$ and create new ones for the new trapezoids. Link them to the search tree appropriately by adding new inner nodes.
    end for
    return $(\mathcal{T}, \mathcal{D})$.
end procedure

Analysis

- Correctness: Follows from construction, in particular the loop invariants.
- Search Complexity: depends on the depth of the search structure.
  - Depth of $\mathcal{D}$ increases by at most 3 every iteration. Therefore the query time is bounded by $3n$.
  - Consider a fixed search path for q in $\mathcal{D}$. Let X_i be a random variable denoting the number of nodes added on that path in iteration i.
  - So the search path has length $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$.
  - Let P_i be the probability that we added a node in iteration i. $E[X_i] \leq 3P_i$.
  - What is P_i?

Analysis

- Correctness: Follows from construction, in particular the loop invariants.
- Search Complexity: depends on the depth of the search structure.
  - Depth of $\mathcal{D}$ increases by at most 3 every iteration. Therefore the query time is bounded by $3n$.
  - Consider a fixed search path for q in $\mathcal{D}$. Let X_i be a random variable denoting the number of nodes added on that path in iteration i.
  - So the search path has length $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$.
  - Let P_i be the probability that we added a node in iteration i. $E[X_i] \leq 3P_i$.
  - $P_i = Pr[\Delta_q(\mathcal{S}_i) \neq \Delta_q(\mathcal{S}_{i-1})]$.

Analysis

- Correctness: Follows from construction, in particular the loop invariants.
- Search Complexity: depends on the depth of the search structure.
  - Depth of $\mathcal{D}$ increases by at most 3 every iteration. Therefore the query time is bounded by $3n$.
  - Consider a fixed search path for q in $\mathcal{D}$. Let X_i be a random variable denoting the number of nodes added on that path in iteration i.
  - So the search path has length $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$.
  - Let P_i be the probability that we added a node in iteration i. $E[X_i] \leq 3P_i$.
  - $P_i = Pr[\Delta_q(\mathcal{S}_i) \neq \Delta_q(\mathcal{S}_{i-1})] = Pr[\Delta_q(\mathcal{S}_i) \notin \mathcal{T}(S_{i-1})]$.

Analysis

- Correctness: Follows from construction, in particular the loop invariants.
- Search Complexity: depends on the depth of the search structure.
  - Depth of $\mathcal{D}$ increases by at most 3 every iteration. Therefore the query time is bounded by $3n$.
  - Consider a fixed search path for q in $\mathcal{D}$. Let X_i be a random variable denoting the number of nodes added on that path in iteration i.
  - So the search path has length $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$.
  - Let P_i be the probability that we added a node in iteration i. $E[X_i] \leq 3P_i$.
  - $P_i = Pr[\Delta_q(\mathcal{S}_i) \neq \Delta_q(\mathcal{S}_{i-1})] = Pr[\Delta_q(\mathcal{S}_i) \notin \mathcal{T}(S_{i-1})] \leq \frac{4}{i}$.

Analysis

- Correctness: Follows from construction, in particular the loop invariants.
- Search Complexity: depends on the depth of the search structure.
  - Depth of $\mathcal{D}$ increases by at most 3 every iteration. Therefore the query time is bounded by $3n$.
  - Consider a fixed search path for q in $\mathcal{D}$. Let X_i be a random variable denoting the number of nodes added on that path in iteration i.
  - So the search path has length $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$.
  - Let P_i be the probability that we added a node in iteration i. $E[X_i] \leq 3P_i$.
  - $P_i = Pr[\Delta_q(\mathcal{S}_i) \neq \Delta_q(\mathcal{S}_{i-1})] = Pr[\Delta_q(\mathcal{S}_i) \notin \mathcal{T}(S_{i-1})] \leq \frac{4}{i}$.
  - So the total length is $12 \sum_{i=1}^{n} \frac{1}{i} \in \mathcal{O}(\log n)$ expected.

Analysis

- Similarly, it can be shown that the size of $\mathcal{D}$ is $\mathcal{O}(n)$ expected, and
- that the running time of `TrapezoidalMap` is $\mathcal{O}(n \log n)$ expected.

Summary

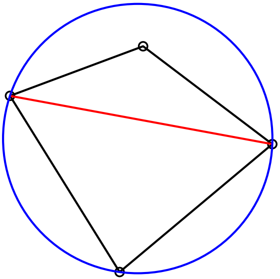
We have seen an algorithm that given a set $\mathcal{S}$ of n line segments builds a trapezoidal map and a search structure in $\mathcal{O}(n \log n)$ expected time and $\mathcal{O}(n)$ expected space. These structures support point location queries in $\mathcal{O}(\log n)$ expected time.

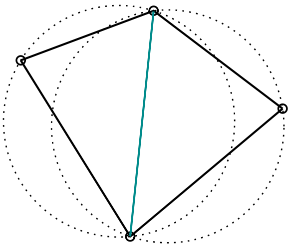
Outline

Delaunay Triangulation

- Definition: Let $\mathcal{P}$ be a set of points in the plane, and let $\mathcal{T}$ be a triangulation of $\mathcal{P}$. Then $\mathcal{T}$ is a *Delauney Triangulation* if the circumcircle of any triangle of $\mathcal{T}$ does not contain a point of $\mathcal{P}$ in its interior.
- Dual Graph of the point Voronoi diagram.

Legal and Illegal Edges





- $\mathcal{T}$ is a Delauney Triangulation if it has no *illegal edges*.
- Every triangulation can be transformed into a DT by continuously flipping illegal edges.

Incremental Algorithm

```
procedure DELAUNEYTRIANGULATION(P)
    Let p_0 be a point on CH(P).
    Create p_{-1}, p_{-2} so that p_0, p_{-1}, p_{-2} are a bounding Δ.
    Randomly permute p_2, ... p_n.
    Initialize T with Δ p_0, p_{-1}, p_{-2}.
    for i ← 2 ... n do
        Find Δ that contains p_i.
        Split triangles.
        Legalize affected edges.
    end for
    Discard p_{-1}, p_{-2} and all incident edges.
    return T
end procedure
```

Analysis

The expected number of total triangles created is bounded by $1 + 9n$.

- In iteration r we insert p_r and get $\mathcal{T}_r$.
- For every triangle created during the "split triangles" step, we create one edge incident at p_r. During the "legalize edges" step we add one incident edge for every two triangles created.
- If the degree of p_r after insertion is k, we have created at most $2k - 3$ triangles. What is this k?

Analysis

The expected number of total triangles created is bounded by $1 + 9n$.

- In iteration r we insert p_r and get $\mathcal{T}_r$.
- For every triangle created during the "split triangles" step, we create one edge incident at p_r. During the "legalize edges" step we add one incident edge for every two triangles created.
- If the degree of p_r after insertion is k, we have created at most $2k - 3$ triangles. What is this k?
- p_r is just a random element of P_r. T_r has at most $3(r + 3) - 6)$ edges. Therefore, $\sum_{i=1}^{r} deg(p_i) \leq 6r$.
- It follows that $E[\textit{number of triangles created in step r}] \leq 2 \cdot 6 - 3 = 9$

Analysis, cont'd

- To support the point location queries, we create a search structure $\mathcal{D}$. This will have a node for every triangle created. Thus expected space is in $\mathcal{O}(n)$.
- Expected running time - ignoring point location - is proportional to the number of triangles created. Therefore – ignoring point location – expected running time is in $\mathcal{O}(n)$ as well.

Analysis, cont'd

- To support the point location queries, we create a search structure $\mathcal{D}$. This will have a node for every triangle created. Thus expected space is in $\mathcal{O}(n)$.
- Expected running time - ignoring point location - is proportional to the number of triangles created. Therefore – ignoring point location – expected running time is in $\mathcal{O}(n)$ as well.
- Point location dominates this however. Amortized over the entire run it requires $\mathcal{O}(n \log n)$ [omitted].

Summary

We have seem a randomized incremental algorithm to construct a Delauney Triangulation. It runs in $\mathcal{O}(n \log n)$ expected time and requires linear expected space.

Thank you for your attention.

Questions?

References I

- Mark de Berg, Marc van Kreveld, Mark Overmars, Otfried Cheong ”Computational Geometry: Algorithms and Applications”, Third Edition, Springer 2008
- Michael Ian Shamos, Dan Hoey ”Closest Point Problems”, in Proceedings of 16th Annual IEEE Symposium on Foundations of Computer Science (1975)