COMPUTATIONAL GEOMETRY AND ALGORITHMS

ARAVIND BHARATHI

Abstract. A reading project taken under the guidance of Professor Akanksha Agrawal, wherein I was introduced to the mathematical modelling of computational problems, covering some algorithmic paradigms and data structures used to solve such problems. I also learnt analysis techniques with an emphasis on the design and analysis of efficient algorithms and covered topics ranging from divide-and-conquer, dynamic programming, greedy algorithms, amortization and randomization

This was followed by the study of geometric data structures, line segment intersections and overlay operations, polygon triangulation, Voronoi diagrams, Delaunay triangulations and convex hulls.

Contents

1. Preliminaries

In most geometric problems, sided-ness is important and the data structures are implemented s.t. it takes $\mathcal{O}(1)$ time to find out

- Degeneracies occur often and are generally ignored in initial analysis
- Floating point numbers may result in rounding errors that distort the outcomes

Definition 1.1: Correctness

The ability of an algorithm to handle all normal and special cases

Date: Summer 2021.

Definition 1.2: Robustness

The ability of an algorithm to handle small errors in computations

1.1. **Geometric Algorithms.** The steps towards development of a geometric algorithm involves three phases

(1) Developing a general algorithm ignoring any degenerate cases
(2) Altering the algorithm to deal with degenerate cases. This typically done taking the following into account
    - Degenerate cases dealt s.t. they can be incorporated within the general algorithm
    - Symbolic perturbation scheme is a general technique used to deal with degeneracies
(3) Implement the algorithm using geometric data structures and data types, typically done in exact arithmetic

1.2. **Line-Segment Properties.**

Definition 1.3: Convex Combination

A convex combination $p_3 = (x_3, y_3)$ of two distinct points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ where $p_3 = \alpha p_1 + (1 - \alpha)p_2$ for $0 \leq \alpha \leq 1$

Remark

Given two distinct points p_1 and p_2, the line segment $\overline{p_1p_2}$ is the set of convex combinations of p_1 and p_2

- p_1 and p_2 are called the endpoints of the line segment $\overline{p_1p_2}$
- When the ordering of p_1 and p_2 matters, $\overrightarrow{p_1p_2}$ is called a directed line segment
- The point $(0, 0)$ is defined as the origin
- If p_1 is the origin, then the directed line segment $\overrightarrow{p_1p_2} \equiv \vec{p}_2$

Definition 1.4: Clockwise vs Anticlockwise

For two vectors $\vec{p}_1$ and $\vec{p}_2$:

- If $p_1 \times p_2$ is positive, p_2 is anticlockwise wrt p_1
- If $p_1 \times p_2$ is negative, p_2 is clockwise wrt p_1
- If $p_1 \times p_2$ is zero, p_2 and p_1 are collinear

Remark

Takes $\Theta(1)$ time to compute for two multiplications ($2 \cdot \Theta(1)$) and one subtraction ($1 \cdot \Theta(1)$) (determinant of matrix)

Definition 1.5: Turning Direction of Consecutive Segments

For two line segments $\overline{p_0p_1}$ and $\overline{p_1p_2}$: If $(p_1 - p_0) \times (p_2 - p_0)$ is positive, $\overline{p_1p_2}$ is anticlockwise to $\overline{p_0p_1}$ (left turn at p_1)

Remark

Takes $\Theta(1)$ time

Definition 1.6: Straddle

$\overline{p_1p_2}$ is said to straddle a line if p_1 lies on one side and p_2 lies on the other side

Definition 1.7: Determining Whether Two Line Segments Intersect

Two line segments intersect iff either (or both) of the following conditions holds:

- Each segment straddles the line containing the other
- An endpoint of one segment lies on the other segment

Remark

Takes $\Theta(1)$ time

1.3. Determining Whether Any Pair of Segments Intersect.

Definition 1.8: Sweeping

An imaginary vertical sweep line passes through the given set of geometric objects usually from left to right

Remark

he spatial dimension that the sweep line moves across (in this case along x) is treated as a dimension of time. This provides a method for ordering geometric objects.

Ordering segments:

- Order the segments that intersect a vertical sweep line according to the y-coordinates of the points of intersection
- Two segments s_1 and s_2 are comparable at x_0 if the vertical sweep line intersects both of them at (x_0, y)
- A segment enters the ordering (insert segment into sweep line status) when its left endpoint is encountered by the sweep and it leaves the ordering (delete segment from sweep line status) when its right endpoint is encountered

Sweeping algorithms typically manage two sets of data::

(1) Sweep line status: Gives the relationships among the objects that the sweep line intersects
(2) Event-point schedule: A sequence of points (called event points, e.g. segment end point) which are ordered from left to right according to their x-coordinates
    - Sweep line reaches the x-coordinate of an event point $\rightarrow$ sweep halts $\rightarrow$ process event point $\rightarrow$ resume sweep
    - Changes to the sweep line status only occur at event points

2. Convex Hulls

Definition 2.1: Convex

A subset S of a plane is called convex if and only if for any pair of points $p, q \in S$ the line segment $\overline{pq}$ is completely contained in S

Definition 2.2: Convex Hull

Convex Hull $\mathcal{CH}(S)$ of a set S is the smallest convex set that contains S

Remark

For a finite set $P = \{p_1, p_2, \dots p_n\}$, the convex hull is the unique convex polygon whose vertices are points from P that contains all points in P

A polygon can be represented as follows:

- A natural way is to list its vertices in clockwise (or anticlockwise) order starting with an arbitrary vertex
- Vertices of $\mathcal{CH}(P) \subset P$: $\overline{uv}$ is an edge of $\mathcal{CH}(P)$ iff all points $P \setminus \{u, v\}$ lie on one side of $\overline{uv}$

Methods of computing a convex hull studied:

(1) Rotational sweep method: Process vertices in the order of the polar angles they form with a reference vertex
(2) Incremental method: Sort the points from left to right. At i^{th} stage, update the convex hull of the $i-1$ leftmost points
(3) Divide and conquer method: Divide the set of n points into two subsets of $n/2$ points. Recursively compute the convex hulls of the subsets and combine the hulls

All the three methods run in $\mathcal{O}(n \lg n)$ time

2.1. Graham's Scan.

```
- Maintain a stack $S$ of candidate points
- Push each point of the input set $Q$ ($|Q| \geq 3$) onto $S$ and
pop each point that is not a vertex of $CH(Q)$
- When the algorithm terminates, $S$ contains exactly the vertices of $CH(Q)$
1. Let $p_0$ be the point in $Q$ with the min $y$-coord (plus the min $x$-coord
in case of a tie)
2. Let $\langle p_1, p_2, \dots, p_{n-1}\rangle$ be the remaining points in $Q$
sorted by polar angle around $p_0$ (range in the interval $[0,\pi)$)
3. If more than one point has the same angle, remove all but the farthest point from $p_0$
4. Declare an empty stack $S$
5. Push the first three points into the stack
6. Push the next point and check if there are consecutive turns in the same direction
(in this case, without loss of generality, it is left)
7. If there is a non-left turn, delete the point just prior to the newly added point
from the stack and check consecutive turns again
8. Iterate over all the points in $Q$
- Sorting takes $\Theta(n\lg n)$ time
- Subsequently, iterating over $n$ points take $\mathcal{O}(n)$ time
  - The repeated checks for consecutive turns take $\mathcal{O}(n)$ overall as once
  a point is popped from the stack, it is never checked again
- $\Rightarrow \mathcal{O}(n\lg n)$ time
```

Correctness can be shown as follows:

(1) Since no point in Q is below p_0, it must be a vertex of $CH(Q)$
(2) If two or more points have the same polar angle relative to p_0, all but the farthest such point q are convex combinations of p_0 and q, and so are removed from consideration
(3) At the start of each iteration, S consists exactly the vertices of $CH(Q_{i-1})$ in anti-clockwise order

2.2. Jarvis March.

```
- Aka gift-wrapping algorithm
- Runs in $\mathcal{O}(nh)$ time. When $h = o(\lg n)$, Jarvis March runs asymptotically faster than Gra
1. Select $p_0$ as in Graham's Scan
2. Choose the vertex which subtends the smallest polar angle with respect to $p_0$ as $p_1$
3. Continue procedure till the highest point is reached (thereby constructing the right chain)
4. Choose the vertex which subtends the smallest polar angle wrt negative $x$-axis and continue procedu
```

3. Line Segment Intersection

The geometric meaning of an overlay of two network of line segments can be described as follows:

- Given two sets of line segments, compute all intersections between a segment from one set and a segment from the other
- An assumption needs to be made if the line segments are open or closed (at its ends)

A modified version of the problem can be defined as follows: Given a set $S = \{s_1, s_2, \dots s_n\}$ of n closed segments in the plane, report all intersection points among the segments in S

The brute force algorithm would be to compute all intersection $\Rightarrow T(n) = \mathcal{O}(n^2)$

Definition 3.1: Output Sensitive Algorithm

An algorithm whose running time is sensitive to the size of the output

Definition 3.2: Plane Sweep Algorithm

Moving (sweep) line sweeps entire plane with an array updated (not continuously but discreetly) at specific event points

4. Polygon Triangulation

Let $\mathcal{P}$ be a simple polygon with n vertices

Definition 4.1: Triangulate

The process of dividing the vertices into a maximal set of non-intersecting diagonals (within $\mathcal{P}$)

Remark

$\mathcal{T}_\mathcal{P}$ represents the triangulation of P

Maximal $\Rightarrow$ No triangle has a polygon vertex in the interior of one of its edges. This could happen if the polygon has three consecutive collinear vertices

Triangulations are usually not unique

Theorem 4.1

Every simple polygon admits a triangulation and any triangulation of a simple polygon with n vertices consists of exactly $n-2$ triangles

Definition 4.2: Dual Graph

Dual graph of $\mathcal{T}_\mathcal{P} = \mathcal{G}(\mathcal{T}_\mathcal{P})$ is an entity that has a node for every triangle in $\mathcal{T}_\mathcal{P}$ (for every node ν in $\mathcal{G}(\mathcal{T}_\mathcal{P})$, $t(\nu)$ is the triangle in $\mathcal{T}_\mathcal{P}$)

4.1. Art Gallery Problem.

For a simple polygonal region (no self-intersections and no holes), let the n = #cameras

Trivial upper bound n = #vertices (follows from simple polygon assumption)

Another trivial case: Convex polygons requires only one camera $\Rightarrow$ To guard a triangle, only one camera is required

Need to compute worst-case scenario and minimum bound for any simple polygon with n vertices

Theorem 4.2

For a simple polygon with n vertices, $[n/3]$ cameras are sufficient to have every point in the polygon visible from at least one of the cameras

Theorem 4.3

Let $\mathcal{P}$ be a simple polygon with n vertices. A set of $[n/3]$ camera positions in $\mathcal{P}$ s.t. any point inside $\mathcal{P}$ is visible from at least one of the cameras can be computed in $\mathcal{O}(n \lg n)$ time

4.2. Computing a Triangulation.

Definition 4.3: Monotonicity

A simple polygon is called monotone wrt a line l if for **any** line $l' \perp l$, $\mathcal{P}$ intersects with l' at most twice

Remark

This can be extended to situations where intersection of $\mathcal{P}$ with l' is connected

Remark

A polygon that is monotone wrt y-axis is called *y-monotone*
If one walks from a topmost to a bottommost vertex along the left (or the right) boundary chain, then we always move downwards or horizontally, never upwards

When walking from topmost to bottommost vertex, a turn vertex v is where **both** incident edges go up or down. There are 4 types of turn vertices:

(1) Start vertices (topmost with internal angle $< \pi$)
(2) End vertices (bottommost with internal angle $< \pi$)
(3) Merge vertices (not bottommost and both incident edges go up)
(4) Split vertices (not topmost and both incident edges go down)

Other vertices are called regular vertices

Partition $\mathcal{P}$ into y-monotones by adding diagonals from any split vertex v to any vertex above v and from any merge vertex w to any vertex below w

Lemma 4.1

A polygon is y-monotone if it has no split vertices or merge vertices

Remark

$\Rightarrow \mathcal{P}$ is partitioned into y-monotone pieces once the split and merge vertices have been ridden off

4.2.1. *Algorithm to partition into y-monotone pieces.*

```
- Plane sweep algorithm
- Vertices of $\mathcal P$ are the event points and no new points
points will be created during the sweep
- Priority queue $Q$ supports a priority of events based on y-coordinate
  - Either pre-sort the coordinates ($\mathcal O(n\lg n)$ time) and
  each insert would take $\mathcal O(1)$ time
  - Or let ‘insert_Q‘ operation occur after a binary search of
  $\mathcal O(\lg n)$ time at each event point
  - Both eventually take $\mathcal O(n\lg n)$ time
```

4.3. **Triangulating a Monotone Polygon.**

```
- Let $\mathcal P$ be a strictly y-monotone polygon
- Sort vertices in descending order of y-coordinates (break the tie
by handling leftmost vertex first)
- Declare an empty stack $\mathcal S$ as an auxiliary data structure
- Push a vertex $v_i$ to the stack and check if diagonals can be drawn from
$v_i$ to other vertices in the stack from the bottom of the stack
  - Pop all the triangulated vertices that are *split-off* from the drawn diagonal
- Only the highest polygon vertex remaining (which is at the bottom of the stack)
is definitively convex wrt diagonal / edges. This is an invariant
```

Theorem 4.4

A strictly y-monotone polygon with n vertices can be triangulated in linear time

Theorem 4.5

A simple polygon with n vertices can be triangulated in $\mathcal{O}(n \lg n)$ time and with $\mathcal{O}(n)$ space

Remark

Space worst-case when stack is filled with all vertices

As neither the splitting algorithm for polygons nor the triangulation algorithm uses the initial assumption that $\mathcal{P}$ is simple, the following theorem holds

Theorem 4.6

A planar subdivision with n vertices in total can be triangulated in $\mathcal{O}(n \lg n)$ time with an algorithm that uses $\mathcal{O}(n)$ storage

5. Voronoi Diagrams

Definition 5.1: Euclidean Distance

Euclidean distance: $dist(p, q) := \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$

Let $P := \{p_1, p_2, \dots p_n\}$ be a set of n distinct points (sites) in the plane

Definition 5.2: Voronoi Diagram

Voronoi diagram of P is the subdivision of the plane into n cells (one for each site in P) with the property that a point q lies in the cell corresponding to a site p_i iff $dist(p_i, q) < dist(p_j, q)\ \forall\ p_j \in P, j \neq i$

Denoted by $Vor(P)$

Remark

buse of notation: $Vor(P)$ may be used to denote the edges and vertices and not the planar subdivision itself

The cell of $Vor(P)$ that corresponds to a site p_i is denoted by $\mathcal{V}(p_i)$ (aka Voronoi cell of p_i)

Definition 5.3: Bisector

For two points p and q in the plane, the perpendicular bisector of $\overline{pq}$ is called the **bisector** of p and q

Remark

his bisector splits the plane into two half-planes $h(p,q)$ which contains p and $h(q,p)$ which contains q

A point r lies in $h(p,q)$ iff $dist(p,r) < dist(q,r)$

$$\Rightarrow \mathcal{V}(p_i) = \cap_{1 \leq j \leq n, j \neq i} \; h(p_i, p_j)$$

Intersection of $n-1$ half planes $\Rightarrow$ has at most $n-1$ vertices and at most $n-1$ edges

Theorem 5.1

Let P be a set of n point sites in the plane. If all the sites are collinear then $Vor(P)$ consists of $n-1$ parallel lines. Otherwise, $Vor(P)$ is connected and its edges are either line segments or rays (half-lines)

Definition 5.4: Complexity of a Voronoi Diagram

Complexity of a Voronoi diagram = total # vertices and edges

n sites with at most $n-1$ vertices and edges $\Rightarrow$ complexity of $Vor(P) \subset \mathcal{O}(n^2)$

Theorem 5.2

For $n \geq 3$, the number of vertices in the Voronoi diagram of a set of n point sites in the plane is at most $2n-5$ and the number of edges is at most $3n-6$

Proof

Follows from the Euler's Formula: For any connected planar embedded graph with V nodes, E arcs and F faces, $V - E + F = 2$ and a vertex at infinity v_∞ □

$\Rightarrow$ complexity$(Vor(P)) \subset \mathcal{O}(n)$

Definition 5.5: Largest Empty Circle

Largest empty circle** $C_P(q)$ of q wrt P is the largest circle with q as its centre that does not contain any site of P in its interior

Theorem 5.3

or the Voronoi diagram $Vor(P)$ of a set of points P, the following holds:

(1) A point q is a vertex of $Vor(P)$ iff $C_P(q)$ contains three or more sites on its boundary
(2) The bisector between sites p_i and p_j defines an edge of $Vor(P)$ iff there is a point q on the bisector st $C_P(q)$ contains both p_i and p_j on its boundary but no other site

Theorem 5.4: Converse

Let the bisector of p_i and p_j define a Voronoi edge, then $C_P(q)$ of any point q in the interior of this edge must contain p_i and p_j on its boundary and no other sites

5.1. **Voronoi Diagram Computation.** Computing intersection of one half plane takes $\mathcal{O}(n \lg n)$ time using linear programming $\Rightarrow$ n cells take $\mathcal{O}(n^2 \lg n)$ time

Can do better using a plane sweep algorithm called *Fortune's algorithm* in $\mathcal{O}(n \lg n)$ time. Sorting problem is reducible to Voronoi diagram computation $\Rightarrow$ takes $\Omega(n \lg n)$ time in the worst case. Fortune's algorithm is optimal

Difficulty arises as the usual invariant that the Voronoi cells for the sites above the sweep line cannot be satisfied. Need to maintain information about the part of $Vor(P)$ above l that cannot be changed by sites below l

Let the closed half-plane above l be denoted by l^+. The nearest site p_i for $q \in l^+$ is known for certain if $dist(q, p_i) < dist(q, l)$. The locus of points that are closer to some site $p_i \in l^+$ than to l is bounded by a parabola. 'Skyline' formed by intersection of parabolae β_i of all sites taken as focii wrt l forms the **beach line**, which is x-monotone for a horizontal sweeping line. The *breakpoints* between different parabolic arcs forming the beach line lie on edges of $Vor(P)$

Status of sweep line l = maintaining the beach line. Changes to the beach line:

(1) When a new arc appears - l reaches a new site (**site event**)
   - At a site event, two new breakpoints appear which trace out an edge
   - A growing edge gets connected to another edge and eventually connected to the rest of the diagram

Lemma 5.1

The only way in which a new arc can appear on the beach line is through a site event

Consequently, a beach line consists of at most $2n - 1$ parabolic arcs

(2) When an existing arc on the beach line disappears
   - Right before disappearing, the arc of the parabola would have shrunk to a point $q \Rightarrow$ this point is a vertex of the Voronoi diagram
   - This occurs at **circle events**

Lemma 5.2

The only way in which an existing arc can disappear from the beach line is through a circle event

Data structures in the algorithm:

- Event queue Q
- Status of sweep line, i.e., representation of beach line
- Subdivisions stored in DCEL including a outer bounding box for consistency
- Beach line represented by the balanced binary search tree $\mathcal{T}$
  - Leaves = Arcs of beach line (in left-to-right ordering)
  - Each leaf μ stores the site that defines the arc α it represents
    - Each leaf stores one pointer to the node in Q that represents the circle event in which α disappears
    - 'nil' if no circle event exists or hasn't been detected
  - Internal nodes of $\mathcal{T}$ represent breakpoints
    - Ordered tuple $\langle p_i, p_j \rangle$ where p_i is the focus of the parabola on the left and p_j is that on the right
    - Insert, Search and Delete operations in $\mathcal{O}(\lg n)$ time
    - Every internal node ν also has a pointer to the half edge being traced out by the breakpoint represented by ν in the DCEL
  - x-coordinate of breakpoint can be computed from tuple of sites and the position of the sweep line in constant time
- Parabolae are not explicitly stored
- Q is implemented as a priority queue where the priority of an event is its y-coordinate
  - Stores the upcoming events that are already known

  - Site events are stored as it is
  - Circle events are stored by the lowest point of the circle along with a pointer to the leaf in $\mathcal{T}$ that represents the arc that will disappear in that event
  - Potential circle events are stored in Q. Could fail if:
    * Breakpoints do not converge
    * New arc arises before breakpoints fully converge (false alarm)
- All the site events are known in advance but the circle events are not

Detection of circle events:

- Topological structure of beach line changes at every event
- At a site event, three new triple arcs (triples - $(\alpha, \alpha', \alpha'')$) appear: New arc is 1. left arc 2. middle arc (can't form converging breakpoints) 3. right arc
- When such a triple has converging breakpoints, add event to Q
- If site associated to a disappearing triple is present in Q, it is a false alarm

Lemma 5.3

Every Voronoi vertex is detected by means of a circle event

After all the events have been handled (and Q is empty), the beach line would not have disappeared - the breakpoints that are still present correspond to the half-infinite edges of the Voronoi diagram

```
    Input: $P:=\{p_1,p_2,\dots p_n\}$
Output: $Vor(P)$ given inside a bounding box in DCEL $\mathcal D$
‘‘‘
1. Initialise $Q$, $\mathcal T$ and $\mathcal D$
2. while $Q$ is not empty
3.   Remove the event with largest y-coordinate from $Q$
4.   if event p_i == site event
       HandleSiteEvent(p_i)
5.   else
       HandleCircleEvent(k) // where k is the leaf of $\mathcal T$ representing
                            // the arc that will disappear
6. Compute a bounding box st it contains all vertices of the Voronoi diagram in its interior. Attach ha
7. Complete the description of DCEL (uptil this point, only half edges of the DCEL would have been dete
‘‘‘
- HandleSiteEvent($p_i$)
‘‘‘
1. if $\mathcal T$ is empty
     insert $p_i$
     return
2. search($\alpha$) in $\mathcal T$ vertically above $p_i$
3. if leaf representing $\alpha$ has a pointer to a circle event in $Q$
     print (false alarm)
     delete event from $Q$
4. Replace the leaf of $\mathcal T$ that represents $\alpha$ with a subtree having
three leaves. Middle leaf = stores new site $p_i$. Left and right store $p_j$ $\Rightarrow$
Store tuples $\langle p_j,p_i \rangle$ and $\langle p_i,p_j \rangle$
5. Rebalance BST
6. Create new half edge records in $\mathcal D$ for the edge separating $\mathcal V(p_i)$
and $\mathcal V(p_j)$
7. Check if triple of $p_i$ converges
8. if yes
     insert circle event into $Q$
     add pointers between $\mathcal T$ and node in $Q$
‘‘‘
```

```
- HandleCircleEvent($k$)
‘‘‘
1. Delete $k$ from $\mathcal T$ (for disappearing arc $\alpha$)
2. Update tuples
3. Rebalance BST
4. Delete all circle events involving $\alpha$ from $Q$ // these can be found using
                                  // pointers from pred and succ of $k$ in
                                  // $\mathcal T$
5. Add circle event point as vertex to DCEL
6. Create two half edge records corresponding to the new breakpoint of the beach
line
7. Check if new triple that has the former left (or right) neighbour of $\alpha$
as its middle arc to see if the two breakpoints converge
8. if yes
     insert corresponding circle event into $Q$
     set pointers between new circle event in $Q$ and corresponding leaf in
     $\mathcal T$
‘‘‘
```

Lemma 5.4

The algorithm runs in $\mathcal{O}(n \lg n)$ time and uses $\mathcal{O}(n)$ space

Proof

n site events of at most constant number of BST operations $\mathcal{O}(\lg n)$ time □

Theorem 5.5

The Voronoi diagram of a set of n points in the plane can be computed with a sweep line algorithm in $\mathcal{O}(n \lg n)$ time using $\mathcal{O}(n)$ space

5.2. Voronoi Diagrams of Line Segments.

- Bisector of two points = line
- Bisector of two lines = two lines
- Bisector of two line segments = combination of upto 7 line segments and parabolic arcs. Parabolic arcs occur only if the closest point of one line segment is an endpoint and the of the other is in its interior

Complexity of $Vor(S)$ remains $\mathcal{O}(n)$. For simplicity, assume all line segments are disjoint. Let $S := \{s_1, s_2, \dots s_n\}$ be a set of n disjoint line segments (Segments = sites). Beach line is defined by only taking into account the part of any line segment above the sweep line l

Types of breakpoints considering a point p is closest and equidistant to:

(1) Two site endpoints and l, then p is a breakpoint that traces a line segment (point site case)
(2) Two site interior and l, then p traces a line segment
(3) A site endpoint and a site interior of different sites and l, then p traces a parabolic arc
    When only one site in involved:
(4) If p is closest to a site endpoint and equidistant from l, then p traces a line segment
(5) If a site interior intersects the sweep line, then the intersection is a breakpoint that traces a line segment

Types of events:

(1) Site event (upper endpoint): Arc of beach line splits into two and four new arcs appear in between. The breakpoints between these four arcs are of the last two types

(2) Site event (lower endpoint): Breakpoint due to intersection of site interior with l is replaced by two breakpoints of the fourth type with a parabolic in between for the newly discovered site event
(3) Circle event: Corresponds to the disappearance of an arc in the beach line and they occur when the sweep line reaches the bottom of an empty circle that is defined by two or three sites above l. The centres of these empty circles are at locations where two consecutive breakpoints will meet
   - Three sites are involved when the breakpoints are of types 1, 2 and 3
   - Only two sites are involved if one of the breakpoints is of type 4
   - Type 5 can't occur for disjoint line segments

Vornoi diagram stored in DCEL $\mathcal{D}$. Algorithm still has only $\mathcal{O}(n)$ events

Theorem 5.6

The Voronoi diagram of a set of n disjoint line segment sites can be computed in $\mathcal{O}(n \lg n)$ time using $\mathcal{O}(n)$ space

6. Delaunay Triangulation

Definition 6.1: Terrain

A terrain is a graph of a function $f : A \subset \mathbb{R}^2 \to \mathbb{R}$ that assigns a height $f(p)$ to every point p in the domain, A

6.1. Triangulations of Planar Point Sets.

Definition 6.2: Maximum planar subdivision

A maximal planar subdivision is a subdivision $\mathcal{S}$ such that no edge connecting two vertices can be added to $\mathcal{S}$ without destroying its planarity

Remark

In other words, $\mathcal{S}$ is fully polygonated with triangles

Definition 6.3: Triangulation

A triangulation of $P := \{p_1, p_2, \dots p_n\}$, a set of n points in the plane, is a maximum planar subdivision whose vertex set is P

Unbounded face == complement of $CH(P)$
triangles and edges is the same in any triangulation of P

Theorem 6.1

Let P be a set of n points in the plane, not all collinear, and let k denote the number of points in P that lie on the boundary of $CH(P)$, then any triangulation of P has $2n-2-k$ triangles and $3n-3-k$ edges

Proof

Let $\mathcal{T}$ be a triangulation of P with n vertices and m triangles, then $n_f = m + 1$ (including the unbounded face): A triangle has 3 edges and the unbounded face has k edges, and each edge is incident to exactly two faces $\Rightarrow n_e = \frac{3m+k}{2}$ The result follows from Euler's formula $n_v - n_e + n_f = 2$ □

Consider the $3m$ angles of the triangles in $\mathcal{T}$ sorted in ascending order $\alpha_1, \alpha_2, \dots \alpha_{3m}$

Definition 6.4: Angle vector

The angle vector of $\mathcal{T}$ is defined as $A(\mathcal{T}) := (\alpha_1, \alpha_2, \dots \alpha_{3m})$

$A(\mathcal{T}) > A(\mathcal{T}') \equiv A(\mathcal{T})$ is lexicographically larger than $A(\mathcal{T}')$ if $\exists\ i$ s.t. $\alpha_j = \alpha'_j\ \forall\ j < i$ and $\alpha_i > \alpha'_i$

Definition 6.5: Angle-optimality

$\mathcal{T}$ is called **angle-optimal** if $A(\mathcal{T}) \geq A(\mathcal{T}')\ \forall\ \mathcal{T}' \in P$

Theorem 6.2: Thales' Theorem

Angle subtended by a line l on different points on the same side of l in a circle C are equal

Definition 6.6: Illegal Edge

An edge e incident to two triangles is illegal if $\min \alpha < \min \alpha'$ where α and α' correspond to the angles in $\mathcal{T}$ and $\mathcal{T}'$ respectively

Lemma 6.1

Let $\mathcal{T}$ be a triangulation with an illegal (non-optimal) edge e, let $\mathcal{T}'$ be obtained via an edge-flip of $\mathcal{T}$, then $A(\mathcal{T}') > A(\mathcal{T})$

Lemma 6.2

Let $e = \overline{p_i p_j}$ be incident to triangles $p_i p_j p_k$ and $p_i p_j p_l$ and let C be the circle passing through p_i, p_j and p_k. e is illegal iff p_l lies in the interior of C

Remark

If the points form a cyclic quadrilateral with no right angles, then exactly one edge will be illegal

Definition 6.7: Legal Triangulation

A legal triangulation is a triangulation with no illegal edges

Remark

egality is a necessary condition for optimality

```
Legalisation(\mathcal T):
   Input: Some triangulation $\mathcal T$ of $P$
   Output: A legal triangulation of $P$
   while $\mathcal T$ contains an illegal edge
     do edge_flip()
   return $\mathcal T$
```

Cause of termination: angle-vector of $\mathcal{T}$ increases in every iteration of the loop and there is only a finite number of triangulations possible

6.2. **Delaunay Triangulation.** Let $P := \{p_1, p_2, \dots p_n\}$ be a set of n points (or sites). The dual graph $\mathcal{G}$ of $Vor(P)$ has a node for every Voronoi cell of p, $\mathcal{V}(p)$ (i.e., at each site p). Equivalently, $\mathcal{G}$ has an arc for

each edge in $Vor(P)$. There is a one-to-one correspondance between the bounded faces of $\mathcal{G}$ and the vertices of $Vor(P)$

Definition 6.8: Delaunay Graph

The straight line embedding of $\mathcal{G}$ is the Delaunay graph of P, $DG(P)$

Theorem 6.3

The Delaunay graph of a planar point set is a plane graph

Proof

An edge $\overline{p_ip_j}$ is in $DG(P)$ iff $\exists$ a closed disk C_{ij} with p_i and p_j on its boundary and no other site of P contained in it

This follows from a Voronoi diagram edge property that the centre of such a C_{ij} lies on the common edge of $\mathcal{V}(p_i)$ and $\mathcal{V}(p_j)$ □

If a vertex v of $Vor(P)$ is a vertex of Voronoi cells for the sites $p_1, p_2, \dots p_k$, then the corresponding face f in $DG(P)$ has $p_1, p_2, \dots p_k$ as its vertices $\Rightarrow p_1, p_2, \dots p_k$ lie on a circle and f is a convex k-gon

Definition 6.9: General Position

If no four points of the set of points lie on a circle, the set is said to be in general position

Remark

f P is in general position, then all vertices of of $Vor(P)$ have degree three. Consequently, all bounded faces of $DG(P)$ are triangles

Definition 6.10: Delaunay Triangulation

A Delaunay triangulation is any triangulation obtained by adding edges to $DG(P)$

Remark

t is unique iff $DG(P)$ is a triangulation

Theorem 6.4

Let P be a set of points in the plane, then:

(1) Three points p_i, p_j and $p_k \in P$ are vertices of the same face of the Delaunay graph of P iff the circle passing through p_i, p_j and p_k contains no point of P in its interior
(2) Two points p_i and $p_j \in P$ form an edge of the Delaunay graph of P iff there is a closed disc C that contains p_i and p_j on its boundary and does not contain any other point of P

Theorem 6.5

Let P be a set of points in the plane and let $\mathcal{T}$ be a triangulation of P, then $\mathcal{T}$ is a Delaunay Triangulation of P iff the circumcircle of any triangle of $\mathcal{T}$ does not contain a point of P in its interior

Theorem 6.6

Let P be a set of points in the plane. A triangulation $\mathcal{T}$ of P is legal iff $\mathcal{T}$ is a Delaunay triangulation of P

Proof

Legality of all Delaunay Triangulation follows from its definition. The corollary is proved by contradiction □

Angle-optimality ⇒ legality ⇒ Delaunay Triangulation
P is in general position ⇒ only one legal triangulation
P not in general position ⇒ any triangulation is legal

Theorem 6.7

Let P be a set of points in the plane. Angle-optimal triangulation of $P \Leftrightarrow$ Delaunay triangulation of P

Proof

Follows from above theorem and remarks □

6.3. **Computing the Delaunay Triangulation.**

$$P := \{p_1, p_2, \dots p_n\} \Rightarrow Vor(P) \Rightarrow \mathcal{DG}(P) \Rightarrow Delaunay\ Triangulation$$

6.3.1. *Direct computation using randomized incremental algorithm.* Bound $P := \{p_0, p_1, \dots p_{n-1}\}$, sorted in descending order of y-coordinate by a large triangle with vertex p_0 from P. and newly added vertices p_{-1} and p_{-2} then compute the Delaunay triangulation of $P \cup \{p_{-1}, p_{-2}\}$. Later discard p_{-1} and p_{-2} together with all their incident edges. But for this to work, p_{-1} and p_{-2} need to be carefully chosen s.t. they are far away enough. In particular, ensure that they do not lie in any circle defined by three points in P. Finally, add points in random order to a list which maintains a Delaunay triangulation of the current point set

Consider the addition of point p_r

(1) Find triangle in $\mathcal{T}$ which contains p_r
(2) Add edges from p_r to vertices of triangle under consideration. If p_r is on an edge of the triangle e, add edge from p_r to the opposite vertices of the two triangles incident to e
(3) Legalise the edges

```
Algorithm Delaunay Triangulation($P$)
Input: A set $P$ of $n+1$ points in the plane
Output: A Delaunay Triangulation of $P$
1. Let $p_0$ be the lexicographically highest point of $P$ (rightmost to split ties)
2. Let $p_{-1}$ and $p_{-2}$ be two points in $\mathbb R^2$ sufficiently far away and s.t.
 $P$ is contained in the triangle $p_0$, $p_{-1}$ and $p_{-2}$
3. Initialise $\mathcal T$ as the triangulation consisting of the single triangle
$p_0 p_{-1} p_{-2}$
4. Compute a random permutation $p_1, p_2, \dots p_n$ of $P\backslash \{p_0\}$
5. for r in range (1,n):
  6. Insert $p_r$ into $\mathcal T$
    7. Find a triangle $p_i p_j p_k \in \mathcal T$ containing $p_r$
    8. if $p_r$ lies in the interior of the triangle $p_i p_j p_k$
      9. then Add edges from $p_r$ to the three vertices of $p_i p_j p_k$ thereby splitting
       $p_i p_j p_k$ into three triangles
        10. LegaliseEdge($p_r$, $\overline{p_i p_j}$, $\mathcal T$)
        11. LegaliseEdge($p_r$, $\overline{p_j p_k}$, $\mathcal T$)
        12. LegaliseEdge($p_r$, $\overline{p_k p_i}$, $\mathcal T$)
```

```
    13. else $p_r$ lies on an edge of $p_i p_j p_k$ (let edge $e$ be $\overline{p_i p_j}$)
      14. Add edges from $p_r$ to $p_k$ and to the third vertex $p_l$ of other triangle
       incident to $e$, thereby splitting two triangles into four
      15. LegaliseEdge($p_r$, $\overline{p_i p_l}$, $\mathcal T$)
      16. LegaliseEdge($p_r$, $\overline{p_l p_j}$, $\mathcal T$)
      17. LegaliseEdge($p_r$, $\overline{p_j p_k}$, $\mathcal T$)
      18. LegaliseEdge($p_r$, $\overline{p_k p_i}$, $\mathcal T$)
19. Discard $p_{-1}$ and $p_{-2}$ and all their incident edges
20. Return $\mathcal T$
```

6.3.2. *Legalising a triangulation.* At each iteration, the edges need to be flipped until the triangulation obtained is legal

```
LegaliseEdge($p_r$, $\overline{p_i p_j}$, $\mathcal T$)
// Where $\overline{p_i p_j}$ is the edge that may need to be flipped
1. if $\overline{p_i p_j}$ is illegal
  // Check by lemma
  // $p_i p_j p_k$ be the triangle adjacent to $p_r p_i p_j$ along $\overline{p_i p_j}$
  2. then flip $\overline{p_i p_j}$
  3. LegaliseEdge($p_r$, $\overline{p_i p_k}$, $\mathcal T$)
  4. LegaliseEdge($p_r$, $\overline{p_k p_j}$, $\mathcal T$)
```

To prove the correctness of the algorithm:

- Need to prove that no illegal edges remain after 'LegaliseEdge()' is invoked

Lemma 6.3

Every new edge created in 'DelaunayTriangulation()' or in 'LegaliseEdge()' during the insertion of p_r is an edge of the $\mathcal{DG}(\{p_{-2}, p_{-1}, p_0, \ldots, p_r\})$

Proof

(1) Delaunay Triangulation - A triangle $p_i p_j p_k$ and its circumcircle C split into three by p_r lying in its interior (and no other point of the iteration of the algorithm lies in the interior) will inevitably obtain 3 legal edges in its Delaunay graph as a circle can be drawn with p_r and $p_{i/j/k}$ on its boundary fully contained within C
(2) LegaliseEdge - Originally, $p_i p_j p_k$ form a Delaunay triangle. With the addition of p_r in the interior of the circumcircle of the triangle C, the edge $\overline{p_i p_j}$ becomes illegal. $\overline{p_r p_l}$ is fully contained within C implying that it is an edge of the Delaunay graph at that iteration

□

- An edge can only become illegal if an incident triangle changes and 'LegaliseEdge' tests them too

6.3.3. *Implementing the algorithm.* There is a need to find the triangle in the r^{th} iteration containing p_r and to deal with removing p_{-1} and p_{-2} at the end of the algorithm

(1) While building a Delaunay Triangulation, the corresponding point location structure (DAG)$\mathcal{D}$ is also built. Leaves of $\mathcal{D} \rightarrow$ triangles of $\mathcal{T}$ with cross pointers between them. Internal nodes in $\mathcal{D} \rightarrow$ triangles that were in the triangulation at some earlier stage but have already been destroyed
Initialise $\mathcal{D}$ as a DAG with a single leaf node $\rightarrow p_0 p_{-1} p_{-2}$ At some iteration where $\Delta = p_i p_j p_k$ is split $\rightarrow$ add three new leaves to $\mathcal{D}$ and make leaf for Δ into an internal node, with outgoing pointers to those three new leaves
When $p_i p_j p_k$ and $p_i p_j p_l$ are replaced by an edge flip by triangles $p_k p_l p_j$ and $p_k p_i p_l$, two new leaves are created and the old leaf nodes become internal nodes pointing to these new leaves
Find the triangle that contains p_r by descending the DAG. Since the out-degree of any node is at most 3, this takes linear time in the number of nodes on the search path
(2) Let l_{-1} be a horizontal line lying below the entire set of $n+1$ points P and l_{-2} be a horizontal line lying above P

p_{-1} is chosen to be sufficiently far to the right on l_{-1} (outside every circle defined by three non-collinear points of P) and s.t. the clockwise ordering of the points of P around p_{-1} is identical to their lexicographic ordering. Similarly choose p_{-2} sufficiently far to the left on l_{-2} (of $P \cup \{p_{-1}\}$)
The Delaunay Triangulation of $P \cup \{p_{-1}, p_{-2}\}$ consists of:

- The Delaunay Triangulation of P
- Edges connecting p_{-1} to every point on the left convex hull of P
- Edges connecting p_{-2} to every point on the right convex hull of P
- Edge $\overline{p_{-1}p_{-2}}$

The lowest point of P and highest point p_0 of P are connected to both p_{-1} and p_{-2}
Point location step:

- p_j is lexicographically larger than p_i
- $\Leftrightarrow p_j$ lies to the left of the line from p_i to p_{-1}
- $\Leftrightarrow p_j$ lies to the left of the line from p_{-2} to p_i

Checking if an edge is illegal in the presence of p_{-1} and p_{-2}. Let $e := \overline{p_i p_j}$ be the edge to be tested, and p_k and p_l be the other vertices of the triangles incident to e (if they exist):

- e is an edge of the triangle $p_0 p_{-1} p_{-2}$. These edges are always legal
- The indices i, j, k and l are all non-negative - normal case; treated as seen above
- *All other cases* - e is legal iff $\min(k, l) < \min(i, j)$
  - Either p_k or p_l is the point p_r, so at most one of the indices of k and l is negative
  - $e = \overline{p_{-1}p_{-2}}$ is handled in case 1, so at most one of the indices of i and j is negative
  - Correctness:
    (a) If only one of the four indices is negative, then this point lies outsidethe circle defined by the other three points and the method is correct
    (b) Otherwise, both $\min(i, j)$ and $\min(k, l)$ are negative and the fact that p_{-2} lies outside any circle defined by three points in $P \cup \{p_{-1}\}$ implies that this method is correct

6.4. **Analysis.** Notations: $P_r := \{p_1, p_2, \ldots p_r\}$ and $\mathcal{DG}_r := \mathcal{DG}(\{p_{-2}, p_{-1}, p_0\} \cup P_r)$. The subset of points in P that lie in the circumcircle of a given triangle Δ is denoted by $K(\Delta)$

Lemma 6.4

The expected number of triangles created by 'DelaunayTriangulation' is at most $9n + 1$

Proof

(1) One triangle $p_0 p_{-1} p_{-2}$ to begin with
(2) After insertion of p_r, there are k edges of $\mathcal{DG}_r$ incident to p_r
Created at most $2(k-3)+3 = 2k-3$ triangles where k is the degree of p_r in $\mathcal{DG}_r$ denoted as $deg(p_r, \mathcal{DG}_r)$
(3) Bound on expected degree $= E[\#\Delta' s] \leq E[2deg(p_r, \mathcal{DG}_r) - 3] = 9$

□

Definition 6.11: Structural change

Structural change generated by an algorithm is the number of entities created and deleted during the course of the algorithm

Theorem 6.8

The Delaunay Triangulation of a set P of n points in the plane can be computed in $\mathcal{O}(n \lg n)$ expected time using $\mathcal{O}(n)$ expected storage

Proof

(1) Correctness of the algorithm follows from the proof of the lemma - Legalising a triangulation
(2) From analysis lemma, expected number of triangles is $\mathcal{O}(n)$
Every node of $\mathcal{D}$ corresponds to a triangle, and the former could use more than linear storage
(3) Apart from the DAG-search step, the algorithm is proportional to the number of created triangles $\Rightarrow \mathcal{O}(n)$
(4) Time to locate p_r in triangulation = linear in number of nodes in path of $\mathcal{D}$ Node corresponding to triangle in current triangulation $\rightarrow \mathcal{O}(1)$
Node corresponding to destroyed triangles $\rightarrow \mathcal{O}(n)$ in number of triangles that were present at some earlier stage
Destroying a triangle $\Delta = p_i p_j p_k$:
- A new point p_s has been inserted inside $\Delta \Rightarrow \Delta$ was a Delaunay triangle in previous iteration
- An edge flip has replaced Δ and an adjacent Δ' by the pair $p_k p_i p_l$ and $p_k p_j p_l \Rightarrow$ either Δ was a DT and p_l was inserted (inside the circumcircle of Δ) or vice versa

Definition 6.12: Charged

The visit to a triangle during the location of p_r is said to be charged to a triangle Δ where $p_r \in K(\Delta)$

- A triangle Δ can be charged at most once for every one of the points in $K(\Delta)$
- Time required for point location step:

$$\Rightarrow \mathcal{O}(n + \sum_{\Delta} card(K(\Delta)))$$

where the summation is over all Delaunay triangles created by the algorithm. This is equal to $\mathcal{O}(n \lg n)$

□

Let Δ be a triangle of the Delaunay triangulation $\mathcal{DG}_r$

- For $r = 1$, $K(\Delta) = n$
- For $r = n$, $K(\Delta) = 0$
- For random r, $K(\Delta) = \mathcal{O}(n/r)$

Proof

Under the (simplifying) assumption that P is in general position (result is true for general case as well)

Lemma 6.5

- If P is a point in general position, then $\sum_{\Delta} card(K(\Delta)) = \mathcal{O}(n \lg n)$ where the summation is over all Delaunay triangles created by the algorithm

- Since P is assumed to be in general position, every subset P_r is in general position $\Rightarrow$ triangulation obtained after adding p_r is the unique graph $\mathcal{DG}_r = \mathcal{T}_r$
- Set of triangles created in stage $r = \mathcal{T}_r \backslash \mathcal{T}_{r-1}$ (by definition)
-

$$\Rightarrow \sum_{\Delta} card(K(\Delta)) \equiv \sum_{r=1}^{n} \left(\sum_{\Delta \in \mathcal{T}_r \backslash \mathcal{T}_{r-1}} card(K(\Delta)) \right)$$

- For a point q, let $k(P_r, q)$ denote the number of triangles $\Delta \in \mathcal{T}_r$ s.t. $q \in K(\Delta)$ and let $k(P_r, q, p_r)$ be the number of triangles $\Delta \in \mathcal{T}_r$ s.t. not only $q \in K(\Delta)$ but p_r is also incident to Δ
  - Any Delaunay triangle created in stage r is incident to p_r

–

$$\Rightarrow \sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} card(K(\Delta)) = \sum_{q \in P \setminus P_r} k(P_r, q, p_r)$$

– A triangle $\Delta \in \mathcal{T}_r$ is incident to a random point p with probability at most $3/r$

- From arithmetic calculation above, the # of triangles destroyed by the insertion of point p_{r+1} = # of triangles created -2
- Summing over r proves the lemma

□

7. astro

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

[2] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. Computational Geometry: Algorithms and Applications (3rd ed. ed.). Springer-Verlag TELOS, Santa Clara, CA, USA.

[3] David M. Mount. 2002. Computational Geometry Lecture Notes (Fall 2002).

[4] Zaninetti, L. (2018). Filaments of galaxies and Voronoi diagrams. Open Astronomy, 27(1), 335-340. https://doi.org/10.1515/astro-2018-0040

[5] Vavilova I. (2012). The Voronoi tessellation method in astronomy. https://arxiv.org/pdf/2012.08965.pdf

IIT Bombay