Duality Transformation and its Applications to Computational Geometry

Partha P. Goswami

University of Kalyani
Kalyani, West Bengal, India.

Outline

Introduction

- In the Cartesian plane, a point has two parameters (x- and y-coordinates) and a (non-vertical) line also has two parameters (slope and y-intercept). We can thus map a set of points to a set of lines, and vice versa, in an one-to-one manner.

Introduction

- In the Cartesian plane, a point has two parameters (x- and y-coordinates) and a (non-vertical) line also has two parameters (slope and y-intercept). We can thus map a set of points to a set of lines, and vice versa, in an one-to-one manner.
- This natural duality between points and lines in the Cartesian plane has long been known to geometers.

Introduction

- In the Cartesian plane, a point has two parameters (x- and y-coordinates) and a (non-vertical) line also has two parameters (slope and y-intercept). We can thus map a set of points to a set of lines, and vice versa, in an one-to-one manner.
- This natural duality between points and lines in the Cartesian plane has long been known to geometers.
- The concept of duality is a powerful tool for the description, analysis, and construction of algorithms.

Introduction

- In the Cartesian plane, a point has two parameters (x- and y-coordinates) and a (non-vertical) line also has two parameters (slope and y-intercept). We can thus map a set of points to a set of lines, and vice versa, in an one-to-one manner.
- This natural duality between points and lines in the Cartesian plane has long been known to geometers.
- The concept of duality is a powerful tool for the description, analysis, and construction of algorithms.
- In this lecture we explore how geometric duality can be used to design efficient algorithms for a number of important problems in computational geometry.

Introduction

- There are many different point-line duality mappings possible, depending on the conventions of the standard representations of a line.

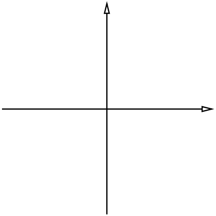
Introduction

- There are many different point-line duality mappings possible, depending on the conventions of the standard representations of a line.
- Each such mapping has its advantages and disadvantages in particular contexts.

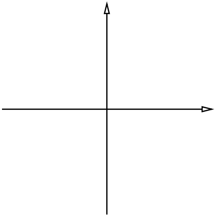
Outline

Definition

Let D be the duality transformation.



primal



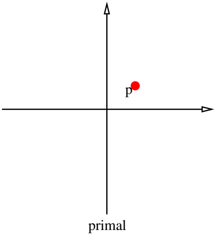
dual

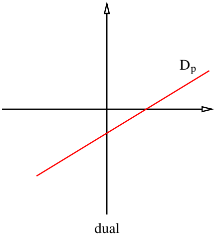
Definition

Let D be the duality transformation.

Definition

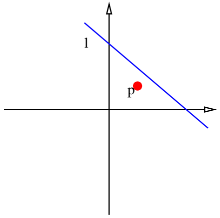
A point $p(a, b)$ is transformed to the line $D_p(y = ax - b)$.

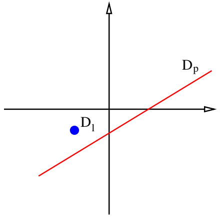




Definition

Let D be the duality transformation.





Definition

A point $p(a, b)$ is transformed to the line $D_p(y = ax - b)$.

Definition

A line $l(y = cx + d)$ is transformed to the point $D_l(c, -d)$.

Properties

Lemma

D is its own inverse, that is, $DD_p = p$ and $DD_l = l$.

Properties

Lemma

D is its own inverse, that is, $DD_p = p$ and $DD_l = l$.

Lemma

D is not defined for vertical lines since vertical lines can not be represented in the form $y = mx + c$.

Properties

Lemma

D is its own inverse, that is, $DD_p = p$ and $DD_l = l$.

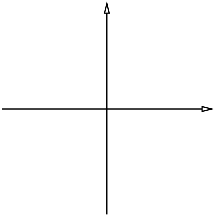
Lemma

D is not defined for vertical lines since vertical lines can not be represented in the form $y = mx + c$.

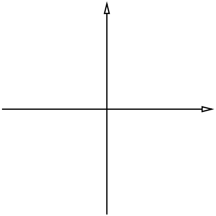
However this is not a problem in general. Because we can always rotate the problem space slightly so that none is vertical.

Properties

Incidence preserving



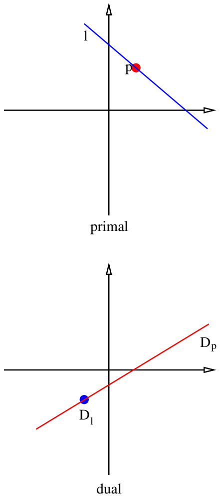
primal



dual

Properties

Incidence preserving

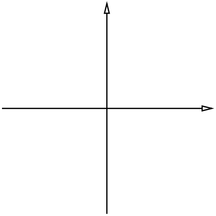


Lemma

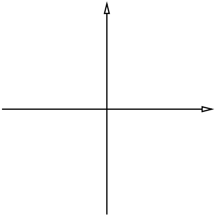
A point $p(a, b)$ is incident to the line $l(y = cx + d)$ in the primal plane iff point $D_l(c, -d)$ is incident to the line $D_p(y = ax - b)$ in the dual plane.

Properties

Order preserving



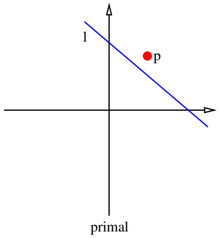
primal



dual

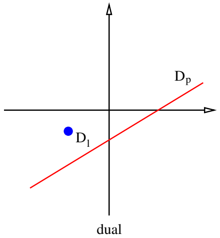
Properties

Order preserving



Lemma

A point $p(a, b)$ is above (below) the line $l(y = cx + d)$ in the primal plane iff line $D_p(y = ax - b)$ is below (above) the point $D_l(c, -d)$ in the dual plane.



Alternative Definition

- The duality transformation we have described so far is often called m-c duality.

Alternative Definition

- The duality transformation we have described so far is often called m-c duality.
- An alternative definition, called polar duality, is also used.

Polar Duality

Definition

A point p with coordinates (a, b) in the primal plane corresponds to a line T_p with equation $ax + by + 1 = 0$ in the dual plane and vice versa.

Polar Duality

Definition

A point p with coordinates (a, b) in the primal plane corresponds to a line T_p with equation $ax + by + 1 = 0$ in the dual plane and vice versa.

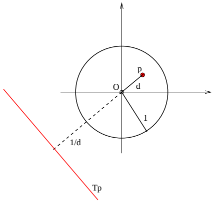
- Geometrically this means if d is the distance from the origin to the point p, The dual T_p of p is the line perpendicular to Op at distance $1/d$ from O and placed on the other side of O.

Polar Duality

Definition

A point p with coordinates (a, b) in the primal plane corresponds to a line T_p with equation $ax + by + 1 = 0$ in the dual plane and vice versa.

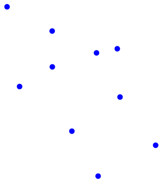
- Geometrically this means if d is the distance from the origin to the point p, The dual T_p of p is the line perpendicular to Op at distance $1/d$ from O and placed on the other side of O.



Outline

Convex Hull

Let $\mathcal{P}$ be a set of points in the plane.

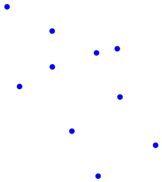


Convex Hull

Let $\mathcal{P}$ be a set of points in the plane.

Definition

Convex hull of $\mathcal{P}$, denoted by $CH(\mathcal{P})$, is the smallest convex set containing $\mathcal{P}$.



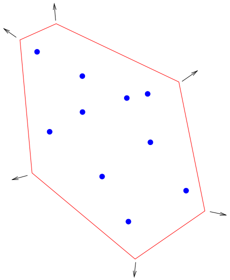
Convex Hull

Let $\mathcal{P}$ be a set of points in the plane.

Definition

Convex hull of $\mathcal{P}$, denoted by $CH(\mathcal{P})$, is the smallest convex set containing $\mathcal{P}$.

Informally, an elastic band stretched open to encompass the given set, when released, assumes the shape of the convex hull.



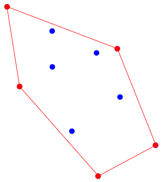
Convex Hull

Let $\mathcal{P}$ be a set of points in the plane.

Definition

Convex hull of $\mathcal{P}$, denoted by $CH(\mathcal{P})$, is the smallest convex set containing $\mathcal{P}$.

Informally, an elastic band stretched open to encompass the given set, when released, assumes the shape of the convex hull.



Convex Hull Problem

Problem

Given a set of points $\mathcal{P}$ in the plane, compute the convex hull $CH(\mathcal{P})$ of the set $\mathcal{P}$.

Convex Hull Problem

Problem

Given a set of points $\mathcal{P}$ in the plane, compute the convex hull $CH(\mathcal{P})$ of the set $\mathcal{P}$.

To compute the convex hull of a point set is a well known and fundamental problem in computational geometry.

A Naive Algorithm

`input: a set S of n points in the plane`

A Naive Algorithm

```
input: a set S of n points in the plane
for each triple (u,v,w) of S{

}
```

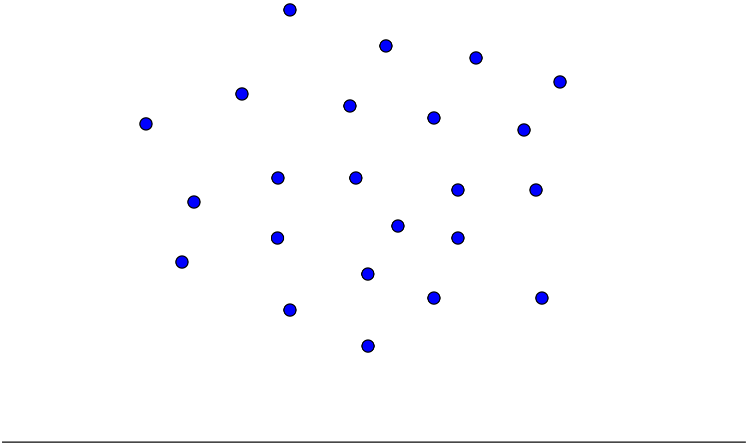
A Naive Algorithm

```
input: a set S of n points in the plane
for each triple (u,v,w) of S{
   remove all the points of S contained in Tr(u,v,w)
}
```

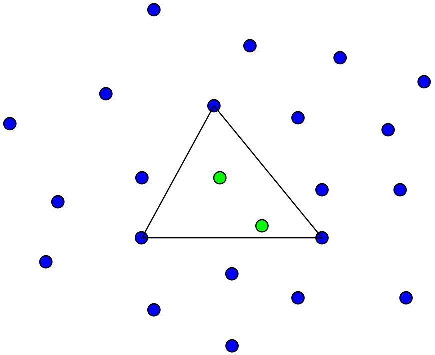
A Naive Algorithm

```
input: a set S of n points in the plane
for each triple (u,v,w) of S{
   remove all the points of S contained in Tr(u,v,w)
}
Order the remaining points of S and
  output the ordered list
```

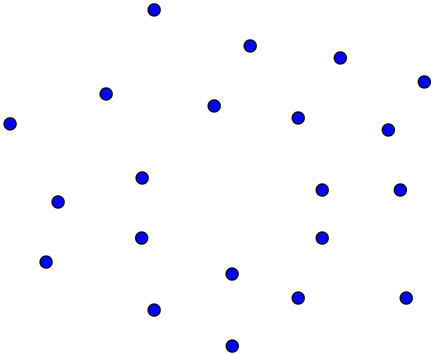
Example



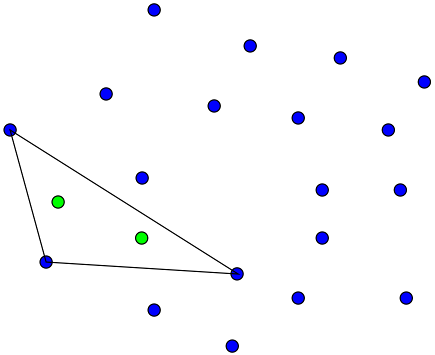
Example



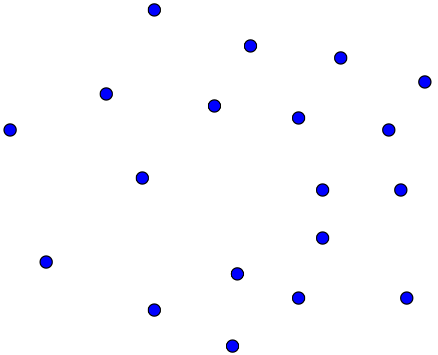
Example



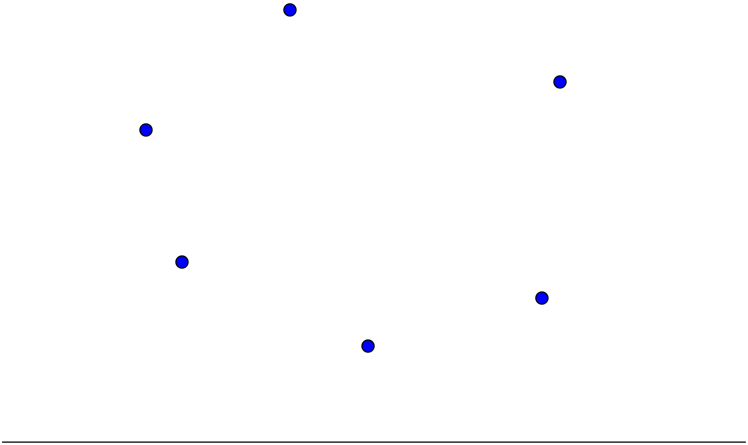
Example



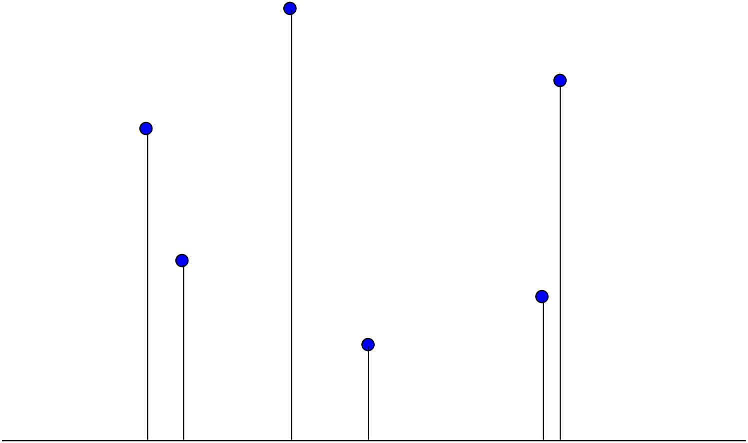
Example



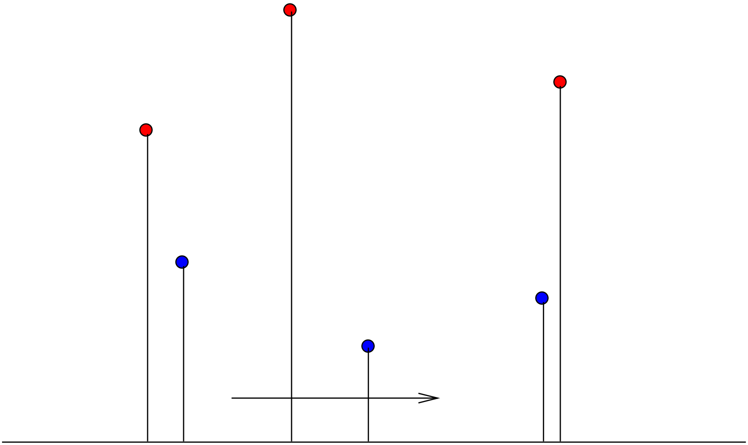
Example



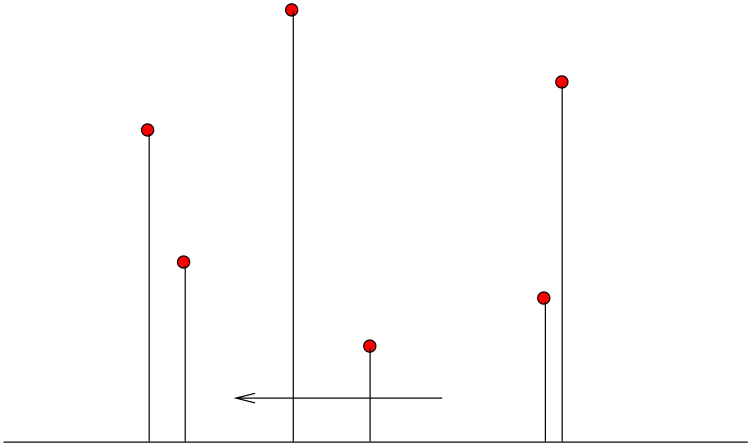
Example



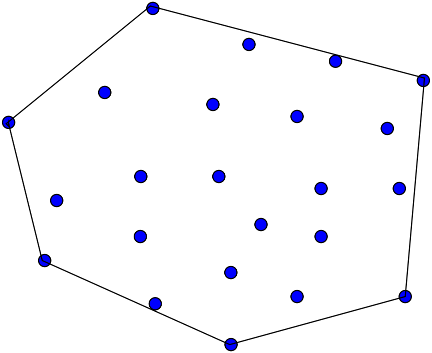
Example



Example



Example



Complexity

There are $O(n^3)$ triangles and it takes $O(n)$ time for each triangle.
So processing time for all triangles is $O(n^4)$.
The sorting step requires $O(n \log n)$ time.
Final reporting takes $O(n)$ time.

Complexity

There are $O(n^3)$ triangles and it takes $O(n)$ time for each triangle.
So processing time for all triangles is $O(n^4)$.
The sorting step requires $O(n \log n)$ time.
Final reporting takes $O(n)$ time.

Result

The naive algorithm takes $O(n^4)$ time and $O(n)$ space to compute the convex hull of a set of n points.

Optimal Algorithms

- The worst case computational complexity of the problem has been shown to be $O(n \log n)$, where n is the size of the given point set.

Optimal Algorithms

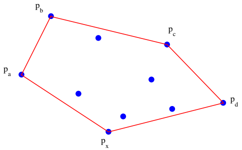
- The worst case computational complexity of the problem has been shown to be $O(n \log n)$, where n is the size of the given point set.
- A number of optimal algorithms have been devised for the convex hull problem.

Optimal Algorithms

- Grahams scan, time complexity $O(nlogn)$.
  (Graham, R.L., 1972)
- Divide and conquer algorithm, time complexity $O(nlogn)$.
  (Preparata, F. P. and Hong, S. J., 1977)
- Jarvis's march or gift wrapping algorithm, time complexity $O(nh)$ where h number of vertices of the convex hull.
  (Jarvis, R. A., 1973)
- Most efficient algorithm to date is based on the idea of Jarvis's march, time complexity $O(nlogh)$.
  T. M. Chan (1996)

Definitions

Let $\mathcal{P}$ be the given set of n points in the plane. Let $p_a \in \mathcal{P}$ be the point having smallest x-coordinate and $p_d \in \mathcal{P}$ be the point with largest x-coordinate. Obviously, both p_a and p_d belongs to $CH(\mathcal{P})$.

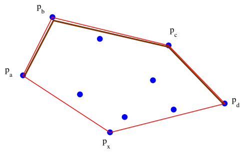


Definitions

Let $\mathcal{P}$ be the given set of n points in the plane. Let $p_a \in \mathcal{P}$ be the point having smallest x-coordinate and $p_d \in \mathcal{P}$ be the point with largest x-coordinate. Obviously, both p_a and p_d belongs to $CH(\mathcal{P})$.

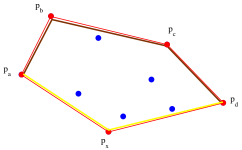
Definition

The c-wise polygonal chain $p_a, \ldots, p_d$ along the hull is called the upper hull.



Definitions

Let $\mathcal{P}$ be the given set of n points in the plane. Let $p_a \in \mathcal{P}$ be the point having smallest x-coordinate and $p_d \in \mathcal{P}$ be the point with largest x-coordinate. Obviously, both p_a and p_d belongs to $CH(\mathcal{P})$.



Definition

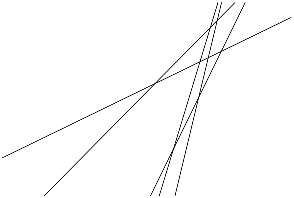
The c-wise polygonal chain $p_a, \ldots, p_d$ along the hull is called the upper hull.

Definition

The cc-wise polygonal chain $p_a, \ldots, p_d$ along the hull is called the lower hull.

Definitions

Let $\mathcal{L}$ be a set of lines in the plane.

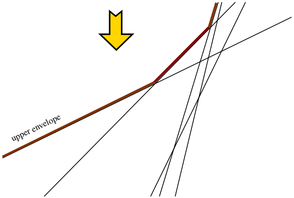


Definitions

Let $\mathcal{L}$ be a set of lines in the plane.

Definition

The upper envelope is a polygoal chain E_u such that no line $l \in \mathcal{L}$ is above E_u.



Definitions

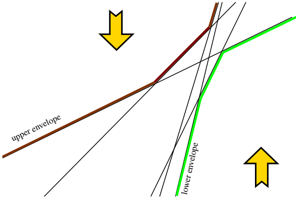
Let $\mathcal{L}$ be a set of lines in the plane.

Definition

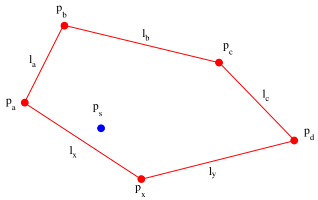
The upper envelope is a polygoal chain E_u such that no line $l \in \mathcal{L}$ is above E_u.

Definition

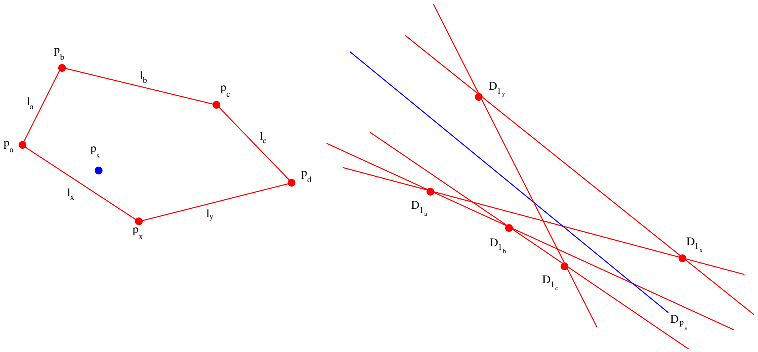
The lower envelope is a polygoal chain E_l such that no line $l \in \mathcal{L}$ is below E_l.



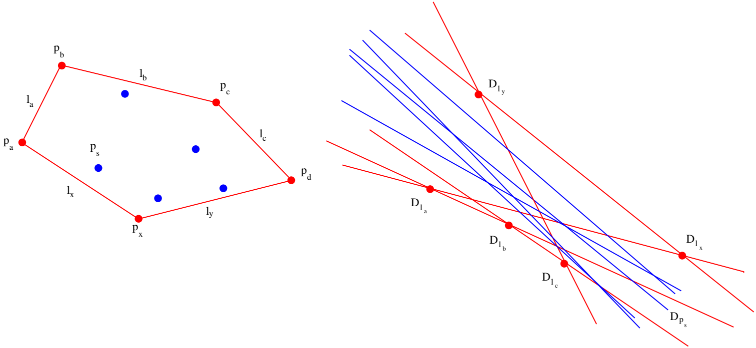
Connection Between Hull and Envelope



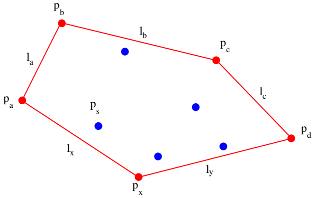
Connection Between Hull and Envelope

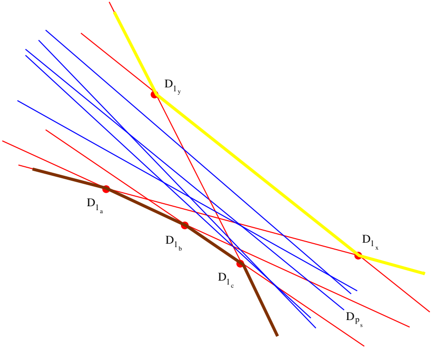


Connection Between Hull and Envelope



Connection Between Hull and Envelope





Connection Between Hull and Envelope

Conclusion

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

Connection Between Hull and Envelope

Conclusion

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

Thus the problem of computing convex hull of a point set in the primal plane reduces to the problem of computing upper and lower envelopes of the line set in the dual plane.

Algorithm

```
Input: I = (L1, L2, ..., Ln) is the list of dual lines
       in the increasing order of slopes.
```

Algorithm

```
Input: I = (L1, L2, ..., Ln) is the list of dual lines
       in the increasing order of slopes.
Output: O = (L1, L2, ..., Lk is the polygonal chain
        representing the upper hull.
```

Algorithm

```
Input: I = (L1, L2, ..., Ln) is the list of dual lines
       in the increasing order of slopes.
Output: O = (L1, L2, ..., Lk is the polygonal chain
        representing the upper hull.
O = (L1);
```

Algorithm

```
Input: I = (L1, L2, ..., Ln) is the list of dual lines
       in the increasing order of slopes.
Output: O = (L1, L2, ..., Lk is the polygonal chain
        representing the upper hull.
O = (L1);
for i = 2 to n do{




}
```

Algorithm

```
Input: I = (L1, L2, ..., Ln) is the list of dual lines
       in the increasing order of slopes.
Output: O = (L1, L2, ..., Lk is the polygonal chain
        representing the upper hull.
O = (L1);
for i = 2 to n do{
  L = last line in O;




}
```

Algorithm

```
Input: I = (L1, L2, ..., Ln) is the list of dual lines
       in the increasing order of slopes.
Output: O = (L1, L2, ..., Lk is the polygonal chain
        representing the upper hull.
O = (L1);
for i = 2 to n do{
  L = last line in O;
  while(the line segment L in O does not intersect Li)


}
```

Algorithm

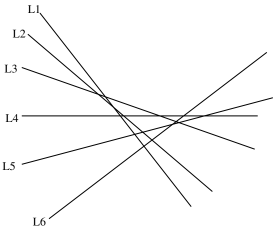
```
Input: I = (L1, L2, ..., Ln) is the list of dual lines
       in the increasing order of slopes.
Output: O = (L1, L2, ..., Lk is the polygonal chain
        representing the upper hull.
O = (L1);
for i = 2 to n do{
  L = last line in O;
  while(the line segment L in O does not intersect Li)
    remove L from O and replace L with its predecessor;

}
```

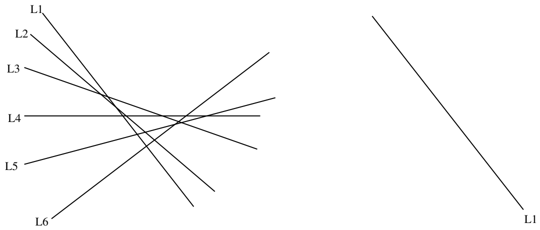
Algorithm

```
Input: I = (L1, L2, ..., Ln) is the list of dual lines
       in the increasing order of slopes.
Output: O = (L1, L2, ..., Lk is the polygonal chain
        representing the upper hull.
O = (L1);
for i = 2 to n do{
  L = last line in O;
  while(the line segment L in O does not intersect Li)
    remove L from O and replace L with its predecessor;
  insert the line Li at the tail of the list O;
}
```

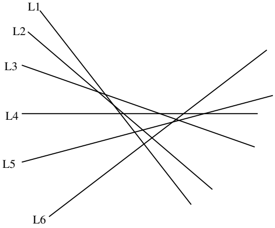
Example

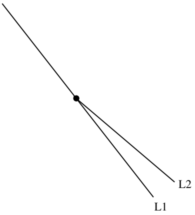


Example

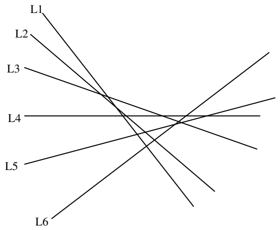


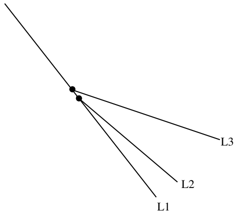
Example



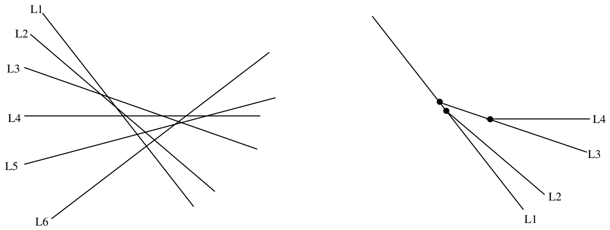


Example

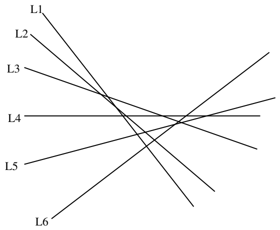


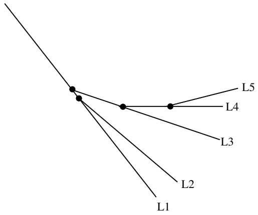


Example

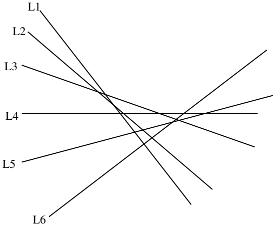


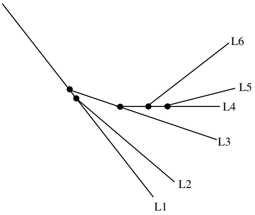
Example



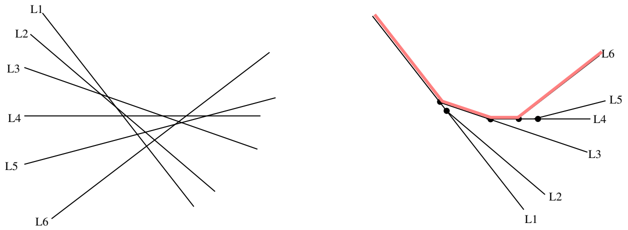


Example





Example



Result

Lemma

After sorting n lines by their slopes in $O(nlogn)$ time, the upper envelope can be obtained in $O(n)$ time.

Result

Lemma

After sorting n lines by their slopes in $O(nlogn)$ time, the upper envelope can be obtained in $O(n)$ time.

Proof.

It may check more than one line segment when inserting a new line, but those ones checked are all removed except the last one. □

Result

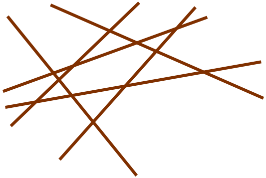
Result

Given a set $\mathcal{P}$ of n points in the plane, $CH(\mathcal{P})$ can be computed in $O(n \log n)$ time using n space.

Outline

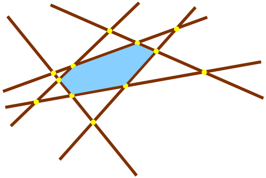
Definition

- Let $\mathcal{L}$ be a set of n lines in the plane. The embedding of $\mathcal{L}$ in the plane induces a planner subdivision that consists of vertices, edges, and faces where some of the edges and faces are unbounded. This subdivision is referred to as arrangement induced by $\mathcal{L}$, and is denoted by $A(\mathcal{L})$.



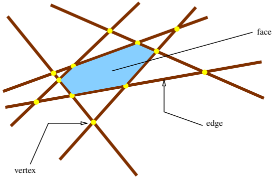
Definition

- Let $\mathcal{L}$ be a set of n lines in the plane. The embedding of $\mathcal{L}$ in the plane induces a planner subdivision that consists of vertices, edges, and faces where some of the edges and faces are unbounded. This subdivision is referred to as arrangement induced by $\mathcal{L}$, and is denoted by $A(\mathcal{L})$.



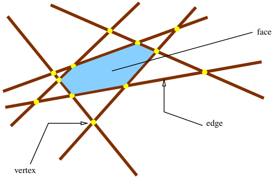
Definition

- Let $\mathcal{L}$ be a set of n lines in the plane. The embedding of $\mathcal{L}$ in the plane induces a planner subdivision that consists of vertices, edges, and faces where some of the edges and faces are unbounded. This subdivision is referred to as arrangement induced by $\mathcal{L}$, and is denoted by $A(\mathcal{L})$.



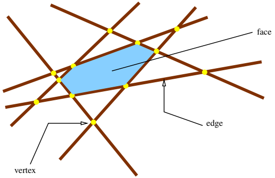
Definition

- Let $\mathcal{L}$ be a set of n lines in the plane. The embedding of $\mathcal{L}$ in the plane induces a planner subdivision that consists of vertices, edges, and faces where some of the edges and faces are unbounded. This subdivision is referred to as arrangement induced by $\mathcal{L}$, and is denoted by $A(\mathcal{L})$.
- An arrangement is called simple if no three lines passes through the same point and no two lines are parallel.



Definition

- Let $\mathcal{L}$ be a set of n lines in the plane. The embedding of $\mathcal{L}$ in the plane induces a planner subdivision that consists of vertices, edges, and faces where some of the edges and faces are unbounded. This subdivision is referred to as arrangement induced by $\mathcal{L}$, and is denoted by $A(\mathcal{L})$.
- An arrangement is called simple if no three lines passes through the same point and no two lines are parallel.
- The (combinatorial) complexity of an arrangement is the total number of vertices, edges, and faces. Observe that worst case complexity occurs when an arrangement is simple.



Result

Theorem

Let $\mathcal{L}$ be the set of n lines in the plane, and let $A(\mathcal{L})$ be the arrangement induced by $\mathcal{L}$.

(i) *The number of vertices of $A(\mathcal{L})$ is at most $n(n-1)/2$.*

(ii) *The number of edges of $A(\mathcal{L})$ is at most n^2.*

(iii) *The number of faces of $A(\mathcal{L})$ is at most $n^2/2 + n/2 + 1$.*

Equality holds in these three statements iff $A(\mathcal{L})$ is simple.

Result

Theorem

Let $\mathcal{L}$ be the set of n lines in the plane, and let $A(\mathcal{L})$ be the arrangement induced by $\mathcal{L}$.

(i) *The number of vertices of $A(\mathcal{L})$ is at most $n(n-1)/2$.*

(ii) *The number of edges of $A(\mathcal{L})$ is at most n^2.*

(iii) *The number of faces of $A(\mathcal{L})$ is at most $n^2/2 + n/2 + 1$.*

Equality holds in these three statements iff $A(\mathcal{L})$ is simple.

Can be proved easily by using Euler's formula:
For any connected planner embedded graph with m_v nodes, m_e arcs, and m_f faces the following relation holds

$$m_v - m_e + m_f = 2.$$

Computation of Arrangement

- One of the fundamental problems in computational geometry is constructing arrangements of lines, that is, explicitly building the regions formed by the intersections of a set of n lines.

Computation of Arrangement

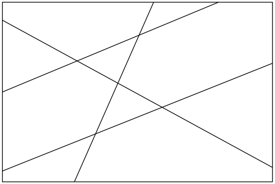
- One of the fundamental problems in computational geometry is constructing arrangements of lines, that is, explicitly building the regions formed by the intersections of a set of n lines.
- Algorithms for a surprising number of problems are based on constructing and analyzing the arrangement of a specific set of lines.

Computation of Arrangement

- One of the fundamental problems in computational geometry is constructing arrangements of lines, that is, explicitly building the regions formed by the intersections of a set of n lines.
- Algorithms for a surprising number of problems are based on constructing and analyzing the arrangement of a specific set of lines.
- A variety of data structures have been proposed for this purpose.

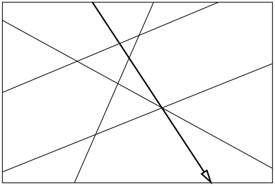
Computation of Arrangement

- Algorithms for constructing arrangements are usually incremental. Beginning with an arrangement of one or two lines, subsequent lines are inserted into the arrangement one at a time, thereby building larger and larger arrangements.



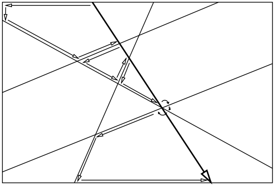
Computation of Arrangement

- Algorithms for constructing arrangements are usually incremental. Beginning with an arrangement of one or two lines, subsequent lines are inserted into the arrangement one at a time, thereby building larger and larger arrangements.
- To insert a new line, we start on the leftmost cell containing the line and walk over the arrangement to the right, moving from cell to neighboring cell and splitting into two pieces those cells that contain the new line.



Computation of Arrangement

- Algorithms for constructing arrangements are usually incremental. Beginning with an arrangement of one or two lines, subsequent lines are inserted into the arrangement one at a time, thereby building larger and larger arrangements.
- To insert a new line, we start on the leftmost cell containing the line and walk over the arrangement to the right, moving from cell to neighboring cell and splitting into two pieces those cells that contain the new line.



Computation of Arrangement

- A geometric fact called the zone theorem implies that the k-th line inserted cuts through k cells of the arrangement and, moreover, $O(k)$ total edges form the boundary of these cells.

Computation of Arrangement

- A geometric fact called the zone theorem implies that the k-th line inserted cuts through k cells of the arrangement and, moreover, $O(k)$ total edges form the boundary of these cells.
- We can thus scan through each edge of every cell encountered on our insertion walk in linear time.

Computation of Arrangement

- A geometric fact called the zone theorem implies that the k-th line inserted cuts through k cells of the arrangement and, moreover, $O(k)$ total edges form the boundary of these cells.
- We can thus scan through each edge of every cell encountered on our insertion walk in linear time.
- The total time to insert all n lines in constructing the full arrangement is $O(n^2)$.

Levels

- We consider an alternative concept, called levels, for structuring an arrangement of lines.

Levels

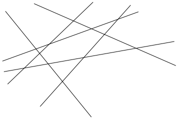
- We consider an alternative concept, called levels, for structuring an arrangement of lines.
- It is simple both from understanding and implementations point of view.

Definition

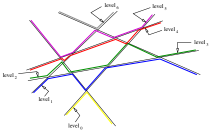
Definition

Let $\mathcal{L}$ be a set on n lines in the plane inducing an arrangement $A(\mathcal{L})$. A point π in the plane is at level θ ($0 \leq \theta \leq n$) if there are exactly θ lines in $\mathcal{L}$ that lie strictly below π. The θ-level of $A(\mathcal{L})$ is the closure of a set of points on the lines of $\mathcal{L}$ whose levels are exactly θ in $A(\mathcal{L})$, and is denoted as λ_θ.

definition

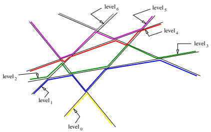


definition



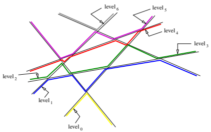
definition

- Clearly, the edges of λ_θ form a monotone polychain from $x = -\infty$ to $x = \infty$. Each vertex of the arrangement $A(\mathcal{L})$ appears in two consecutive levels, and each edge of $A(\mathcal{L})$ appears in exactly one level.



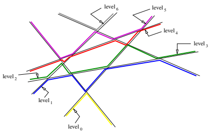
definition

- Clearly, the edges of λ_θ form a monotone polychain from $x = -\infty$ to $x = \infty$. Each vertex of the arrangement $A(\mathcal{L})$ appears in two consecutive levels, and each edge of $A(\mathcal{L})$ appears in exactly one level.
- We can thus store each level simply as an array of segments.



definition

- Clearly, the edges of λ_θ form a monotone polychain from $x = -\infty$ to $x = \infty$. Each vertex of the arrangement $A(\mathcal{L})$ appears in two consecutive levels, and each edge of $A(\mathcal{L})$ appears in exactly one level.
- We can thus store each level simply as an array of segments.
- Observe that the upper and the lower envelops mentioned earlier, are simply the n-th and 0-th levels respectively.



Computing Levels

- Once an arrangement is computed using traditional data structures mentioned earlier, levels of an arrangement can be computed in optimal $O(n^2)$ time.

Computing Levels

- Once an arrangement is computed using traditional data structures mentioned earlier, levels of an arrangement can be computed in optimal $O(n^2)$ time.
- Here we consider an alternative method using plane sweep paradigm.

Computing Levels

- Once an arrangement is computed using traditional data structures mentioned earlier, levels of an arrangement can be computed in optimal $O(n^2)$ time.
- Here we consider an alternative method using plane sweep paradigm.
- The method was first introduced by Bentley and Ottmann (1979) in the context of solving the problem of line segment intersections.

Plane Sweep Method

Basic method consists of:

- A vertical line l, called the sweep line, sweeps over the plane from $x = -\infty$ to $x = \infty$.

Plane Sweep Method

Basic method consists of:

- A vertical line l, called the sweep line, sweeps over the plane from $x = -\infty$ to $x = \infty$.
- The status of the sweep line at any instant is the set of lines intersecting it.

Plane Sweep Method

Basic method consists of:

- A vertical line l, called the sweep line, sweeps over the plane from $x = -\infty$ to $x = \infty$.
- The status of the sweep line at any instant is the set of lines intersecting it.
- The status changes only when the sweep line reaches some particular points, called event points.

Plane Sweep Method

Basic method consists of:

- A vertical line l, called the sweep line, sweeps over the plane from $x = -\infty$ to $x = \infty$.
- The status of the sweep line at any instant is the set of lines intersecting it.
- The status changes only when the sweep line reaches some particular points, called event points.
- The algorithm performs some computational steps when the sweep line reaches event points.

Data structure

- In our case, events are the points of intersection of the lines. Apart from storing the events, we also need to insert new events and extract the event nearest to the sweep line on its right. Clearly, a queue is a suitable data structure for this.

Data structure

- In our case, events are the points of intersection of the lines. Apart from storing the events, we also need to insert new events and extract the event nearest to the sweep line on its right. Clearly, a queue is a suitable data structure for this.
- At every instant, the sweep line intersects each element of $\mathcal{L}$. Order the lines from bottom to top according to their intersection points with the sweep line. Data structure we use for maintaining the sweep line status are arrays storing the levels. At an instant, portion of the line at the i-th position, $0 \leq i \leq n$, is part of the i-th level.

Processing

- Let the next event be the intersection point of the lines currently at i-th and $(i + 1)$-th positions respectively. Processing steps to be performed at this event point are as follows.

Processing

- Let the next event be the intersection point of the lines currently at i-th and $(i+1)$-th positions respectively. Processing steps to be performed at this event point are as follows.
- Portion of the line at the i-th position before the event point will become part of the $(i+1)$-th level after the event point. Similarly, portion of the line at the $(i+1)$-th position before the event point will become part of the i-th level after the event point.

Processing

- Let the next event be the intersection point of the lines currently at i-th and $(i+1)$-th positions respectively. Processing steps to be performed at this event point are as follows.
- Portion of the line at the i-th position before the event point will become part of the $(i+1)$-th level after the event point. Similarly, portion of the line at the $(i+1)$-th position before the event point will become part of the i-th level after the event point.
- If the line at the $(i+1)$-th position after the event point intersect the line at the $(i+2)$-th position on the right of the sweep line, then insert the intersection point as a future event point. Similarly, if the line at the i-th position after the event point intersect the line at the $(i-1)$-th position on the right of the sweep line, then insert the intersection point as a future event point.

Initialization

- Compute the left most intersection point. Position the sweep line before this intersection point.

Initialization

- Compute the left most intersection point. Position the sweep line before this intersection point.
- Compute the intersection points of the lines with the sweep line and order the lines from bottom to top according to the order of the intersection points.

Initialization

- Compute the left most intersection point. Position the sweep line before this intersection point.
- Compute the intersection points of the lines with the sweep line and order the lines from bottom to top according to the order of the intersection points.
- Initialize the level arrays with the lines according to their position on the sweep line.

Initialization

- Compute the left most intersection point. Position the sweep line before this intersection point.
- Compute the intersection points of the lines with the sweep line and order the lines from bottom to top according to the order of the intersection points.
- Initialize the level arrays with the lines according to their position on the sweep line.
- Check each pair of lines from bottom to top if they insert on the right of the sweep line. If yes, insert these intersection points in the queue as an event point.

Algorithm

`Input: A set L of n lines in the plane`

Algorithm

```
Input: A set L of n lines in the plane
Compute initial position of the sweep line. Initialize
  event queue Q and level arrays LA[i], 0 <= i <= n.
```

Algorithm

```
Input: A set L of n lines in the plane
Compute initial position of the sweep line. Initialize
  event queue Q and level arrays LA[i], 0 <= i <= n.
while Q is not empty do
```

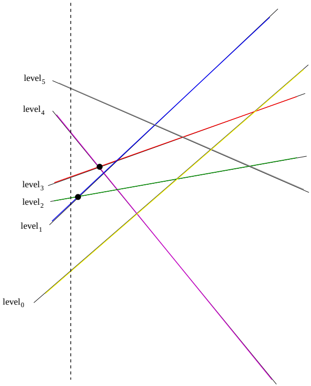
Algorithm

```
Input: A set L of n lines in the plane
Compute initial position of the sweep line. Initialize
  event queue Q and level arrays LA[i], 0 <= i <= n.
while Q is not empty do
  retrieve the next event point from Q and delete it.
```

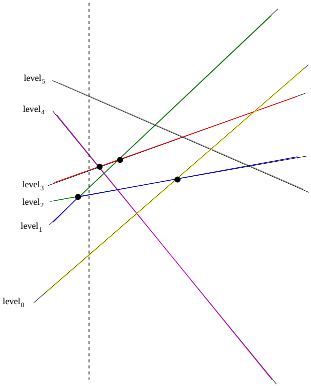
Algorithm

```
Input: A set L of n lines in the plane
Compute initial position of the sweep line. Initialize
  event queue Q and level arrays LA[i], 0 <= i <= n.
while Q is not empty do
  retrieve the next event point from Q and delete it.
  process the event point.
```

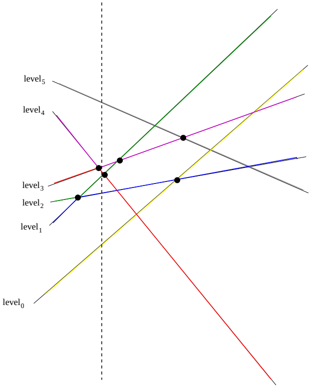
Example



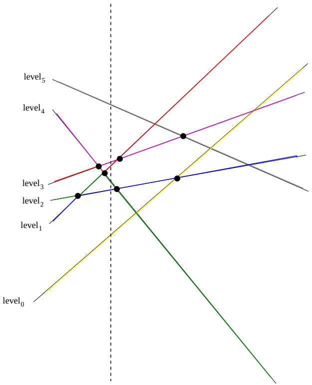
Example



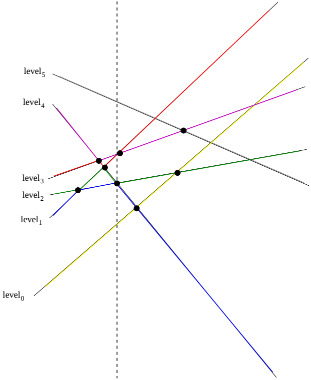
Example



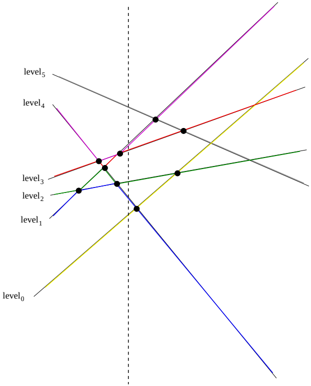
Example



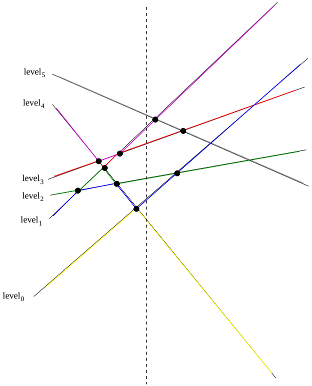
Example



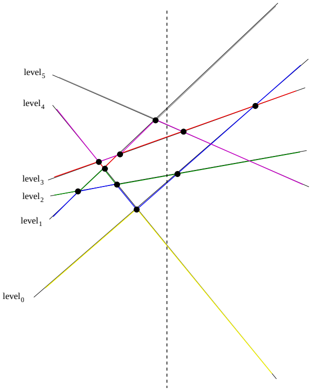
Example



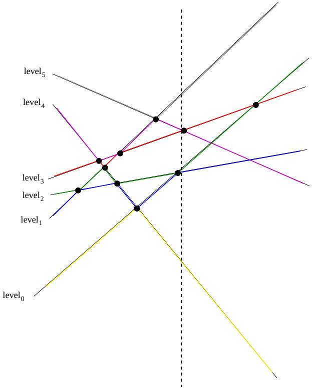
Example



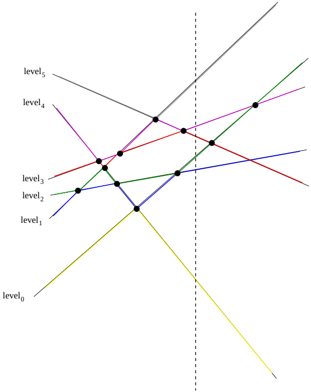
Example



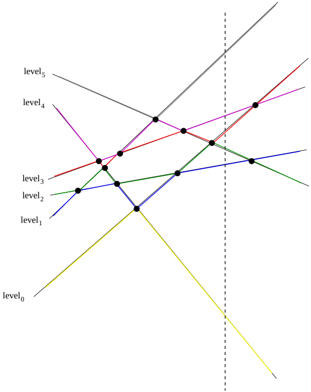
Example



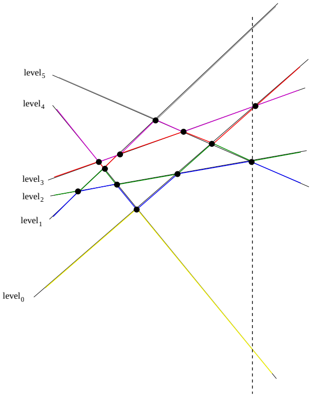
Example



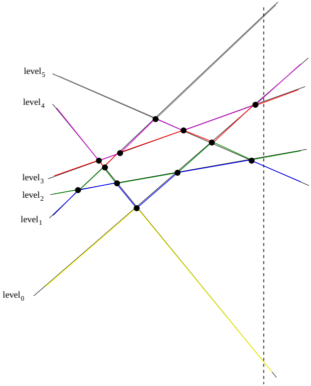
Example



Example



Example



Complexity

- Initialization step requires $O(n^2)$ time.

Complexity

- Initialization step requires $O(n^2)$ time.
- At each event point, processing requires $O(\log n)$ time.

Complexity

- Initialization step requires $O(n^2)$ time.
- At each event point, processing requires $O(\log n)$ time.
- Since there are $O(n^2)$ event points, overall time complexity is $O(n^2 \log n)$.

Complexity

- Initialization step requires $O(n^2)$ time.
- At each event point, processing requires $O(\log n)$ time.
- Since there are $O(n^2)$ event points, overall time complexity is $O(n^2 \log n)$.
- Space complexity is $O(n^2)$.

Result

Theorem

Using plane sweep, levels of an arrangement of n lines can be computed in $O(n^2 \log n)$ time using $O(n^2)$ space.

Outline

Smallest Area Triangle Problem

Problem

Let $\mathcal{P}$ be a set of n points in the plane. The problem is to determine which of the $\binom{n}{3}$ triangles with vertices in $\mathcal{P}$ has the smallest area.

Smallest Area Triangle Problem

Problem

Let $\mathcal{P}$ be a set of n points in the plane. The problem is to determine which of the $\binom{n}{3}$ triangles with vertices in $\mathcal{P}$ has the smallest area.

The solution of the above problem allows us to solve the following problem also.

Smallest Area Triangle Problem

Problem

Let $\mathcal{P}$ be a set of n points in the plane. The problem is to determine which of the $\binom{n}{3}$ triangles with vertices in $\mathcal{P}$ has the smallest area.

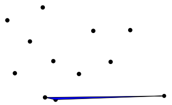
The solution of the above problem allows us to solve the following problem also.

Problem

Let $\mathcal{P}$ be a set of n points in the plane. The problem is to determine whether three points in $\mathcal{P}$ are collinear.

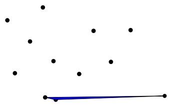
Smallest Area Triangle Problem

- The difficulty of the problem arises from the fact that the vertices of the smallest triangle can be arbitrarily apart (i.e., absence of locality).



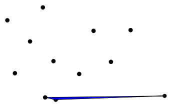
Smallest Area Triangle Problem

- The difficulty of the problem arises from the fact that the vertices of the smallest triangle can be arbitrarily apart (i.e., absence of locality).
- The best known algorithm, without using duality, for this problem has time and space complexities $O(n^2 \log n)$ and $O(n)$ respectively.
  (Edelsbrunner and Welzl, 1982).



Smallest Area Triangle Problem

- The difficulty of the problem arises from the fact that the vertices of the smallest triangle can be arbitrarily apart (i.e., absence of locality).
- The best known algorithm, without using duality, for this problem has time and space complexities $O(n^2 \log n)$ and $O(n)$ respectively. (Edelsbrunner and Welzl, 1982).
- Using duality, it is possible to improve upon the complexity.



Assumption

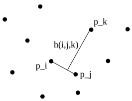
- The definition of duality implies that if two points p_i and p_j in the primal plane have same x-coordinate values, then corresponding duals D_{p_i} and D_{p_j} are parallel in the dual plane.

Assumption

- The definition of duality implies that if two points p_i and p_j in the primal plane have same x-coordinate values, then corresponding duals D_{p_i} and D_{p_j} are parallel in the dual plane.
- To avoid this we assume that no two points in $\mathcal{P}$ have same x-coordinates. This may possibly require rotating the axes by a small angle which can be determined in $O(n \log n)$ time.

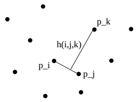
Sketch of the solution

- Let $h(i, j, k)$ be the perpendicular distance from the point p_k to the segment $p_i p_j$ and let the line through p_k that is parallel to $p_i p_j$ is $l(i, j, k)$.



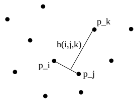
Sketch of the solution

- Let $h(i, j, k)$ be the perpendicular distance from the point p_k to the segment $p_i p_j$ and let the line through p_k that is parallel to $p_i p_j$ is $l(i, j, k)$.
- Smallest area triangle with $p_i p_j$ as an edge minimizes $h(i, j, k)$ for all $k \neq i, j$; $1 \leq k \leq n$.



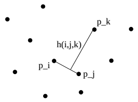
Sketch of the solution

- Let $h(i,j,k)$ be the perpendicular distance from the point p_k to the segment $p_i p_j$ and let the line through p_k that is parallel to $p_i p_j$ is $l(i,j,k)$.
- Smallest area triangle with $p_i p_j$ as an edge minimizes $h(i,j,k)$ for all $k \neq i,j$; $1 \leq k \leq n$.
- Straight forward use of this scheme leads to an $O(n^3)$ time algorithm.



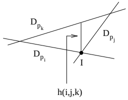
Sketch of the solution

- Let $h(i, j, k)$ be the perpendicular distance from the point p_k to the segment $p_i p_j$ and let the line through p_k that is parallel to $p_i p_j$ is $l(i, j, k)$.
- Smallest area triangle with $p_i p_j$ as an edge minimizes $h(i, j, k)$ for all $k \neq i, j$; $1 \leq k \leq n$.
- Straight forward use of this scheme leads to an $O(n^3)$ time algorithm.
- However, when taken to dual plane, this leads to efficient algorithm.



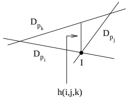
Dualization

- In the dual plane, the edge $p_i p_j$ becomes the intersection point I of D_{p_i} and D_{p_j}.



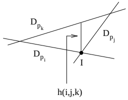
Dualization

- In the dual plane, the edge $p_i p_j$ becomes the intersection point I of D_{p_i} and D_{p_j}.
- The line $l(i, j, k)$ becomes the point on D_{p_k} having same x-coordinate as I.



Dualization

- In the dual plane, the edge $p_i p_j$ becomes the intersection point I of D_{p_i} and D_{p_j}.
- The line $l(i,j,k)$ becomes the point on D_{p_k} having same x-coordinate as I.
- Expression for the vertical distance $h(i,j,k)$ between I and D_{p_k} is



$$h(i,j,k) = y(p_k) + \frac{x(p_k)[y(p_j) - y(p_i)] + x(p_j)y(p_i) - x(p_i)y(p_j)}{x(p_i) - x(p_j)}$$

Algorithm

- We use the plane sweep method. Basic steps are as follows.

Algorithm

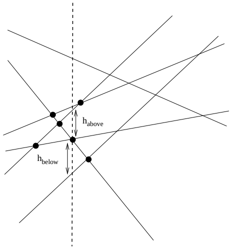
- We use the plane sweep method. Basic steps are as follows.
- Sweep a vertical line over the arrangement of n lines in the dual plane.

Algorithm

- We use the plane sweep method. Basic steps are as follows.
- Sweep a vertical line over the arrangement of n lines in the dual plane.
- Here event points are the intersection points between pairs of lines.

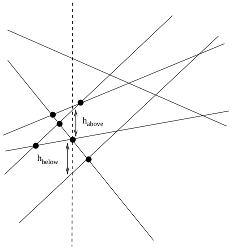
Algorithm

- We use the plane sweep method. Basic steps are as follows.
- Sweep a vertical line over the arrangement of n lines in the dual plane.
- Here event points are the intersection points between pairs of lines.
- When sweep line reaches an event point, the intersection point between D_{p_i} and D_{p_j} say, compute the vertical distances, along the sweep line, between the event point and the lines just above and below it.



Algorithm

- We use the plane sweep method. Basic steps are as follows.
- Sweep a vertical line over the arrangement of n lines in the dual plane.
- Here event points are the intersection points between pairs of lines.
- When sweep line reaches an event point, the intersection point between D_{p_i} and D_{p_j} say, compute the vertical distances, along the sweep line, between the event point and the lines just above and below it.
- Let the minimum distance occurs for the line D_{p_k}. Compute the minimum area of the triangle with $p_i p_j$ as base.



Complexity

- Observe that during sweep we need not store the arrangement. Moreover, at any instance, number of event points stored in the event queue is $O(n)$.

Complexity

- Observe that during sweep we need not store the arrangement. Moreover, at any instance, number of event points stored in the event queue is $O(n)$.
- Hence space complexity of the algorithm is $O(n)$.

Complexity

- Observe that during sweep we need not store the arrangement. Moreover, at any instance, number of event points stored in the event queue is $O(n)$.
- Hence space complexity of the algorithm is $O(n)$.
- Time complexity of the algorithm is, clearly, $O(n^2 \log n)$.

Complexity

- Observe that during sweep we need not store the arrangement. Moreover, at any instance, number of event points stored in the event queue is $O(n)$.
- Hence space complexity of the algorithm is $O(n)$.
- Time complexity of the algorithm is, clearly, $O(n^2 \log n)$.
- The $\log n$ factor in the time complexity can be avoided by using topological line sweep.
  (Edelsbrunner, H. and Guibas, L. J., 1989)

Outline

Problem

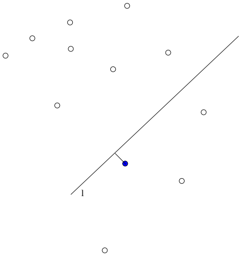
Problem

Given a set $\mathcal{P}$ of n points in the plane and a query line l, compute the nearest neighbor (in the perpendicular distance sense) of the query line l.

Problem

Problem

Given a set $\mathcal{P}$ of n points in the plane and a query line l, compute the nearest neighbor (in the perpendicular distance sense) of the query line l.



Multi-shot Query

- For a single query line, the problem can be solved in optimal $O(n)$ time.

Multi-shot Query

- For a single query line, the problem can be solved in optimal $O(n)$ time.
- We are interested in multi-shot query version.

Multi-shot Query

- For a single query line, the problem can be solved in optimal $O(n)$ time.
- We are interested in multi-shot query version.
- Here we are allowed to preprocess the point set so that each query can be answered efficiently.

Strategy

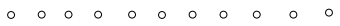
- We use duality to solve the problem.

Strategy

- We use duality to solve the problem.
- Since our definition of duality does not allow vertical line, we need to have separate algorithm for handling vertical query lines.

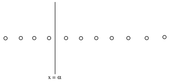
Nearest Neighbor Query Vertical Line

- Sort the points of the given set $\mathcal{P}$ on their x-coordinates. This can be done in $O(n \log n)$ time using $O(n)$ space.



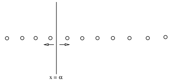
Nearest Neighbor Query Vertical Line

- Sort the points of the given set $\mathcal{P}$ on their x-coordinates. This can be done in $O(n \log n)$ time using $O(n)$ space.
- Using binary search find the position of the query vertical line $x = \alpha$ in the sorted array. This will take $O(\log n)$ time.



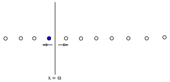
Nearest Neighbor Query Vertical Line

- Sort the points of the given set $\mathcal{P}$ on their x-coordinates. This can be done in $O(n \log n)$ time using $O(n)$ space.
- Using binary search find the position of the query vertical line $x = \alpha$ in the sorted array. This will take $O(\log n)$ time.
- Then a pair of scan from α towards left and right determine the nearest neighbor in constant time.



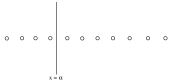
Nearest Neighbor Query Vertical Line

- Sort the points of the given set $\mathcal{P}$ on their x-coordinates. This can be done in $O(n \log n)$ time using $O(n)$ space.
- Using binary search find the position of the query vertical line $x = \alpha$ in the sorted array. This will take $O(\log n)$ time.
- Then a pair of scan from α towards left and right determine the nearest neighbor in constant time.



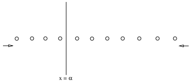
Farthest Neighbor Vertical Query Line

- Same scheme can also be used for determining the farthest neighbor of a query vertical line.



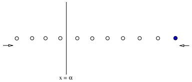
Farthest Neighbor Vertical Query Line

- Same scheme can also be used for determining the farthest neighbor of a query vertical line.
- Here a pair of scan from the end points of the array will determine the farthest neighbor of the query vertical line $x = \alpha$.



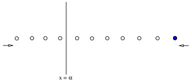
Farthest Neighbor Vertical Query Line

- Same scheme can also be used for determining the farthest neighbor of a query vertical line.
- Here a pair of scan from the end points of the array will determine the farthest neighbor of the query vertical line $x = \alpha$.



Farthest Neighbor Vertical Query Line

- Same scheme can also be used for determining the farthest neighbor of a query vertical line.
- Here a pair of scan from the end points of the array will determine the farthest neighbor of the query vertical line $x = \alpha$.



Result

Lemma

With $O(n \log n)$ preprocessing time using $O(n)$ space, nearest and farthest neighbors of a query vertical line can be found in $O(\log n)$ time.

Farthest Neighbor of a Non-Vertical Query Line

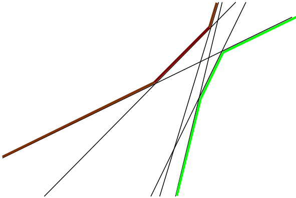
- Suppose the problem is to report the farthest neighbor of a given query line which is non-vertical.

Farthest Neighbor of a Non-Vertical Query Line

- Suppose the problem is to report the farthest neighbor of a given query line which is non-vertical.
- As the preprocessing step, compute the upper envelope and the lower envelope of the set of lines dual to the given set of points $\mathcal{P}$. This can be done in in $O(n \log n)$ time using $O(n)$ space as mentioned previously.

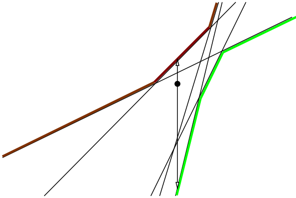
Farthest Neighbor of a Non-Vertical Query Line

- Let E_u and E_l are the arrays storing the upper and the lower envelops respectively.



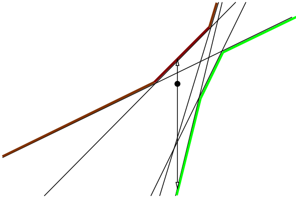
Farthest Neighbor of a Non-Vertical Query Line

- Let E_u and E_l are the arrays storing the upper and the lower envelops respectively.
- Given a query line l, shoot a vertical ray from the point D_l in upward and downward direction and find the intersection points with the upper and the lower envelope respectively.



Farthest Neighbor of a Non-Vertical Query Line

- Let E_u and E_l are the arrays storing the upper and the lower envelops respectively.
- Given a query line l, shoot a vertical ray from the point D_l in upward and downward direction and find the intersection points with the upper and the lower envelope respectively.
- This can be done in $O(\log n)$ time by using two binary searches on the arrays E_u and E_l holding the envelopes.



Result

Lemma

With $O(n \log n)$ preprocessing time using $O(n)$ space, farthest neighbors of a query non-vertical line can be found in $O(\log n)$ time.

Nearest Neighbor of a Query Non-vertical Line

- Let $\mathcal{L}$ be the set of lines which are dual to the points of the given set $\mathcal{P}$. Also let D_l be the point dual to to the query non-vertical line l.

Nearest Neighbor of a Query Non-vertical Line

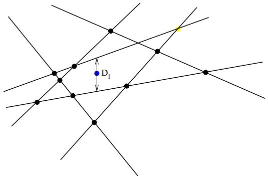
- Let $\mathcal{L}$ be the set of lines which are dual to the points of the given set $\mathcal{P}$. Also let D_l be the point dual to to the query non-vertical line l.
- Let $A(\mathcal{L})$ be the arrangement of lines of the set $\mathcal{L}$.

Nearest Neighbor of a Query Non-vertical Line

- Let $\mathcal{L}$ be the set of lines which are dual to the points of the given set $\mathcal{P}$. Also let D_l be the point dual to to the query non-vertical line l.
- Let $A(\mathcal{L})$ be the arrangement of lines of the set $\mathcal{L}$.
- Let f be the cell of the arrangement $A(\mathcal{L})$ containing D_l.

Nearest Neighbor of a Query Non-vertical Line

- Let $\mathcal{L}$ be the set of lines which are dual to the points of the given set $\mathcal{P}$. Also let D_l be the point dual to to the query non-vertical line l.
- Let $A(\mathcal{L})$ be the arrangement of lines of the set $\mathcal{L}$.
- Let f be the cell of the arrangement $A(\mathcal{L})$ containing D_l.
- Then one of the points corresponding to the lines just above D_l is the nearest neighbor of l in the primal plane.



Point Location Problem

- Given an arrangement of lines $A(\mathcal{L})$, the problem of finding the component of $A(\mathcal{L})$ containing a given query point p is known as point location problem in computational geometry.

Point Location Problem

- Given an arrangement of lines $A(\mathcal{L})$, the problem of finding the component of $A(\mathcal{L})$ containing a given query point p is known as point location problem in computational geometry.
- With standard data structure for storing an arrangement of lines, point location problem can be solved in optimal $O(\log n)$ time.

Point Location Problem

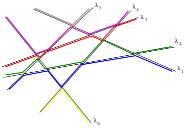
- Given an arrangement of lines $A(\mathcal{L})$, the problem of finding the component of $A(\mathcal{L})$ containing a given query point p is known as point location problem in computational geometry.
- With standard data structure for storing an arrangement of lines, point location problem can be solved in optimal $O(\log n)$ time.
- So with standard data structure, nearest neighbors of a non-vertical query line can be determined in $O(\log n)$ time. The reqired preprocessing time and space is $O(n^2)$.

Point Location Problem

- Given an arrangement of lines $A(\mathcal{L})$, the problem of finding the component of $A(\mathcal{L})$ containing a given query point p is known as point location problem in computational geometry.
- With standard data structure for storing an arrangement of lines, point location problem can be solved in optimal $O(\log n)$ time.
- So with standard data structure, nearest neighbors of a non-vertical query line can be determined in $O(\log n)$ time. The reqired preprocessing time and space is $O(n^2)$.
- Here we describe an algorithm for point location using levels of arrangement.

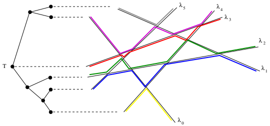
Point Location Using Level Structure

- First compute the levels of the arrangement $A(\mathcal{L})$ in $O(n^2 \log n)$ time using $O(n^2)$ space.
- Let λ_θ be the linear array containing vertices and edges of level θ, $\theta = 0, 1, \ldots, n$, of the arrangement $A(\mathcal{L})$.



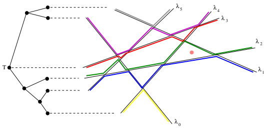
Point Location Using Level Structure

- First compute the levels of the arrangement $A(\mathcal{L})$ in $O(n^2 \log n)$ time using $O(n^2)$ space.
- Let λ_θ be the linear array containing vertices and edges of level θ, $\theta = 0, 1, \ldots, n$, of the arrangement $A(\mathcal{L})$.
- Create a balanced binary search tree T, called the primary structure, whose nodes correspond to the levels θ, $0 \leq \theta \leq n$. Each node of T, representing a level θ, is attached with the corresponding array λ_θ, called the secondary structure. This requires $O(n \log n)$ time and $O(n)$ space.



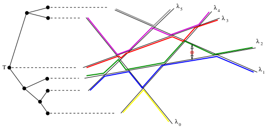
Point Location Using Level Structure

- Given the query line l, we perform two level binary search on the tree T with the point D_l.



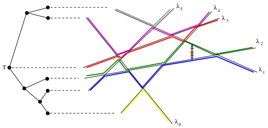
Point Location Using Level Structure

- Given the query line l, we perform two level binary search on the tree T with the point D_l.
- This will enable us to locate the two edges just above and below D_l.



Point Location Using Level Structure

- Given the query line l, we perform two level binary search on the tree T with the point D_l.
- This will enable us to locate the two edges just above and below D_l.
- Time complexity for performing this point location is $O(\log^2 n)$.



Complexity

Lemma

With $O(n^2 \log n)$ preprocessing time and $O(n^2)$ space, nearest neighbor of a non-vertical query line can be determined in $O(\log^2 n)$ time.

Complexity

Lemma

With $O(n^2 \log n)$ preprocessing time and $O(n^2)$ space, nearest neighbor of a non-vertical query line can be determined in $O(\log^2 n)$ time.

- It may be mentioned that the query time complexity can be reduced to $O(\log n)$, by using a data structuring technique, called fractional cascading.
(Lueker, G. S., 1978)

Thank you!