Introduction to Computational Geometry

Arijit Bishnu
(arijit@isical.ac.in)
(http://www.isical.ac.in/~arijit)

Advanced Computing and Microelectronics Unit
Indian Statistical Institute
203, B. T. Road, Kolkata - 700108, West Bengal, India.

Outline

Introduction

- Computational Geometry (CG) involves study of algorithms for solving geometric problems on a computer. The emphasis is more on discrete and combinatorial geometry.

Introduction

- Computational Geometry (CG) involves study of algorithms for solving geometric problems on a computer. The emphasis is more on discrete and combinatorial geometry.
- There are many fields of computer science like computer graphics, computer vision and image processing, robotics, computer-aided designing, geographic information systems, etc. that give rise to geometric problems.

Introduction

- Computational Geometry (CG) involves study of algorithms for solving geometric problems on a computer. The emphasis is more on discrete and combinatorial geometry.
- There are many fields of computer science like computer graphics, computer vision and image processing, robotics, computer-aided designing, geographic information systems, etc. that give rise to geometric problems.
- In CG, the focus is more on discrete nature of geometric problems as opposed to continuous issues. Simply put, we would deal more with straight or flat objects (lines, line segments, polygons) or simple curved objects as circles, than with high degree algebraic curves.

Introduction

- Computational Geometry (CG) involves study of algorithms for solving geometric problems on a computer. The emphasis is more on discrete and combinatorial geometry.
- There are many fields of computer science like computer graphics, computer vision and image processing, robotics, computer-aided designing, geographic information systems, etc. that give rise to geometric problems.
- In CG, the focus is more on discrete nature of geometric problems as opposed to continuous issues. Simply put, we would deal more with straight or flat objects (lines, line segments, polygons) or simple curved objects as circles, than with high degree algebraic curves.
- This branch of study is around thirty years old if one assumes Michael Ian Shamos's thesis [1] as the starting point.

Introduction

- Any problem that is to be solved using a digital computer has to be discrete in form. It is the same with CG.

Introduction

- Any problem that is to be solved using a digital computer has to be discrete in form. It is the same with CG.
- For CG to be applied to areas that deal with continuous issues, discrete approximations to continuous curves or surfaces are needed.

Introduction

- Any problem that is to be solved using a digital computer has to be discrete in form. It is the same with CG.
- For CG to be applied to areas that deal with continuous issues, discrete approximations to continuous curves or surfaces are needed.
- Programming in CG is also a little difficult. Libraries like LEDA [5] and CGAL [6] are now available.

Introduction

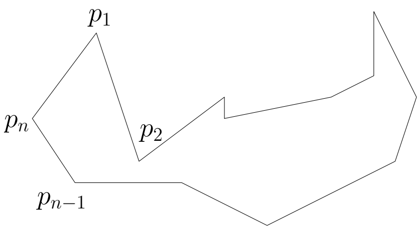
- Any problem that is to be solved using a digital computer has to be discrete in form. It is the same with CG.
- For CG to be applied to areas that deal with continuous issues, discrete approximations to continuous curves or surfaces are needed.
- Programming in CG is also a little difficult. Libraries like LEDA [5] and CGAL [6] are now available.
- CG algorithms suffer from the curse of degeneracies. So, we would make certain assumptions at times like no three points are collinear, no four points are cocircular, etc.

Outline

Area Computation

Problem

Given a simple polygon P of n points, compute its area.



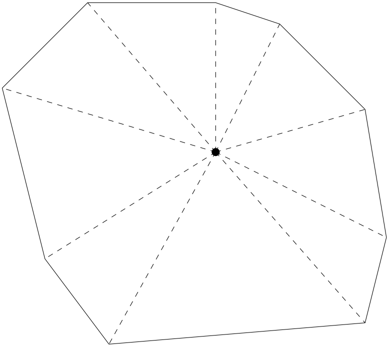
Area Computation

Problem

Given a simple polygon P of n points, compute its area.

Area of a convex polygon

Find a point inside P, draw n triangles and compute the area.



Area Computation

Problem

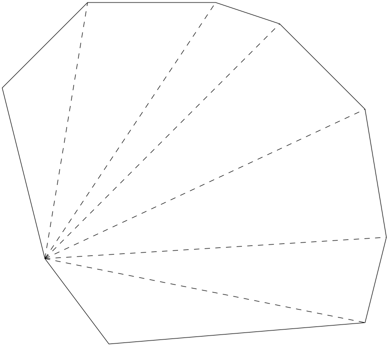
Given a simple polygon P of n points, compute its area.

Area of a convex polygon

Find a point inside P, draw n triangles and compute the area.

A better idea for convex polygon

We can triangulate P by non-crossing diagonals into $n - 2$ triangles and then find the area.



$(n-3)$ diagonals and $(n-2)$ triangles

Area Computation

Problem

Given a simple polygon P of n points, compute its area.

Area of a convex polygon

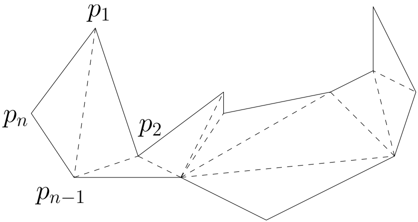
Find a point inside P, draw n triangles and compute the area.

A better idea for convex polygon

We can triangulate P by non-crossing diagonals into $n-2$ triangles and then find the area.

A better idea for simple polygon

We can do likewise.



$(n-3)$ diagonals and $(n-2)$ triangles

Area Computation and Polygon Triangulation

Moral of the story

A simple polygon can be triangulated into $(n-2)$ triangles by $(n-3)$ non-crossing diagonals.

Area Computation and Polygon Triangulation

Moral of the story

A simple polygon can be triangulated into $(n-2)$ triangles by $(n-3)$ non-crossing diagonals.

Proof

The proof is by induction on n.

Area Computation and Polygon Triangulation

Moral of the story

A simple polygon can be triangulated into $(n-2)$ triangles by $(n-3)$ non-crossing diagonals.

Proof

The proof is by induction on n.

Time complexity

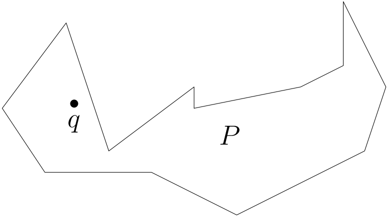
We can triangulate P by a very complicated $O(n)$ algorithm [7] OR by a *more or less simple* $O(n \log n)$ time algorithm [4].

Outline

Point Inclusion

Problem

Given a simple polygon P of n points, and a query point q, is $q \in P$?



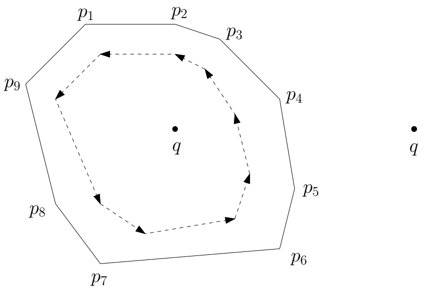
Point Inclusion

Problem

Given a simple polygon P of n points, and a query point q, is $q \in P$?

What if P is convex?

Easy in $O(n)$. Takes a little effort to do it in $O(\log n)$.



q is always to the left if $q \in P$, else, it varies.

Point Inclusion

Problem

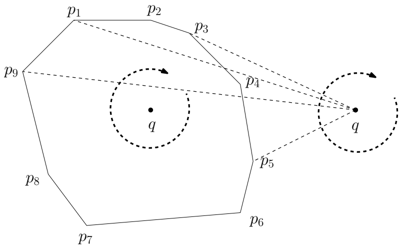
Given a simple polygon P of n points, and a query point q, is $q \in P$?

What if P is convex?

Easy in $O(n)$. Takes a little effort to do it in $O(\log n)$.

Another idea for convex polygon

- Stand at q and look around the polygon.
- We can show the same result for a simple polygon also.

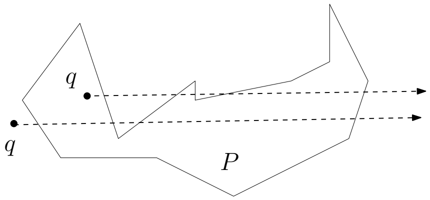


Total angular turn around$q$ is 2π if $q \in P$, else, 0

Point Inclusion

Another technique: Ray Shooting

Shoot a ray and count the number of crossings with edges of P. If it is odd, then $q \in P$. If it is even, then $q \notin P$. Some degenerate cases need to be handled. Time taken is $O(n)$.



Outline

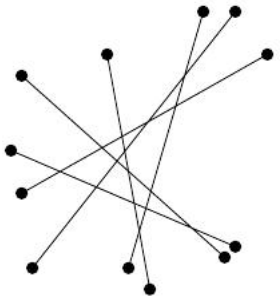
Line Segment Intersection

Input

A set of line segments $\mathcal{L}$ in general position in the plane. $|\mathcal{L}| = n$.

Output

Report the intersections.



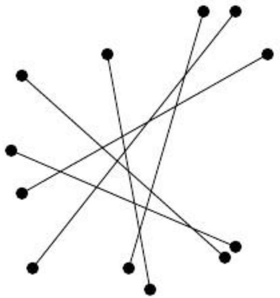
Line Segment Intersection

Input

A set of line segments $\mathcal{L}$ in general position in the plane. $|\mathcal{L}| = n$.

Output

Report the intersections.



Line Segment Intersection

Input

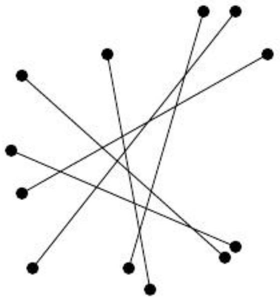
A set of line segments $\mathcal{L}$ in general position in the plane. $|\mathcal{L}| = n$.

Output

Report the intersections.

Output Sensitive Algorithm

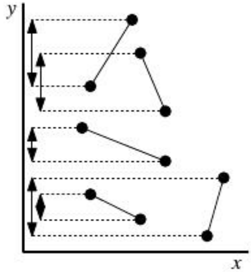
Number of intersections might vary from 0 to $\binom{n}{2} = O(n^2)$. So, the lower bound of the problem is $\Omega(n^2)$. The idea is now to look for an output sensitive algorithm.



An Output Sensitive Algorithm

The idea

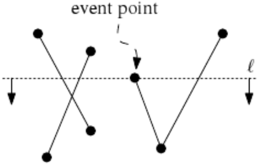
- Avoid testing pairs of segments that are far apart.



An Output Sensitive Algorithm

The idea

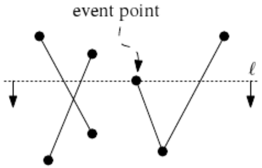
- Avoid testing pairs of segments that are far apart.
- To find such pairs, imagine sweeping a horizontal line ℓ downwards from above all segments.



An Output Sensitive Algorithm

The idea

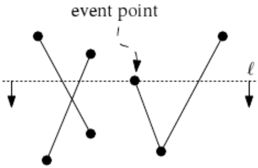
- Avoid testing pairs of segments that are far apart.
- To find such pairs, imagine sweeping a horizontal line ℓ downwards from above all segments.
- Keep track of all segments that intersect ℓ.



An Output Sensitive Algorithm

The idea

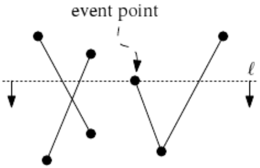
- Avoid testing pairs of segments that are far apart.
- To find such pairs, imagine sweeping a horizontal line ℓ downwards from above all segments.
- Keep track of all segments that intersect ℓ.
- ℓ is the sweep line and the algorithm paradigm is plane sweep.



An Output Sensitive Algorithm

The idea

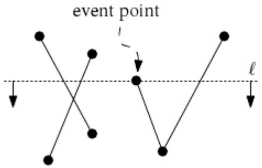
- Avoid testing pairs of segments that are far apart.
- To find such pairs, imagine sweeping a horizontal line ℓ downwards from above all segments.
- Keep track of all segments that intersect ℓ.
- ℓ is the sweep line and the algorithm paradigm is plane sweep.
- The status of the sweep line is the line segments intersecting it.



An Output Sensitive Algorithm

The idea

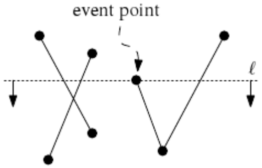
- Avoid testing pairs of segments that are far apart.
- To find such pairs, imagine sweeping a horizontal line ℓ downwards from above all segments.
- Keep track of all segments that intersect ℓ.
- ℓ is the sweep line and the algorithm paradigm is plane sweep.
- The status of the sweep line is the line segments intersecting it.
- Only at particular points known as event points, the status needs to be updated.



Event Points and Sweep Line Status

Event Points and the Event Queue

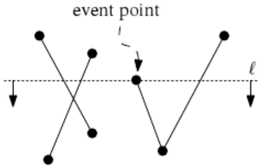
- The start and end points of each line segment. They are static.



Event Points and Sweep Line Status

Event Points and the Event Queue

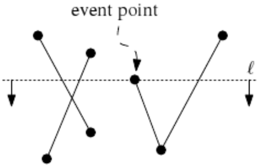
- The start and end points of each line segment. They are static.
- The intersection points. They are dynamic and are generated as the sweep line ℓ sweeps down.



Event Points and Sweep Line Status

Event Points and the Event Queue

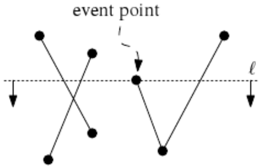
- The start and end points of each line segment. They are static.
- The intersection points. They are dynamic and are generated as the sweep line ℓ sweeps down.
- The event points are to be arranged in a data structure in a way in which the sweep line sees them.



Event Points and Sweep Line Status

Event Points and the Event Queue

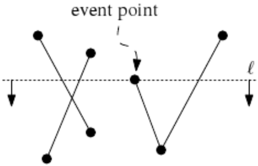
- The start and end points of each line segment. They are static.
- The intersection points. They are dynamic and are generated as the sweep line ℓ sweeps down.
- The event points are to be arranged in a data structure in a way in which the sweep line sees them.
- The data structure should support extracting the minimum y-coordinate, insertion and deletion.



Event Points and Sweep Line Status

Event Points and the Event Queue

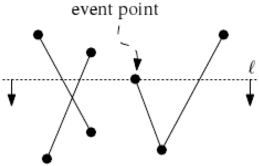
- The start and end points of each line segment. They are static.
- The intersection points. They are dynamic and are generated as the sweep line ℓ sweeps down.
- The event points are to be arranged in a data structure in a way in which the sweep line sees them.
- The data structure should support extracting the minimum y-coordinate, insertion and deletion.
- A heap or a balanced binary search tree can support these operations in $O(\log n)$ time.



Event Points and Sweep Line Status

Sweep Line Status

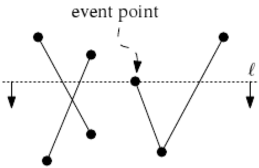
- We need to store the left to right order in which the line segments intersect ℓ. This data structure has to be dynamic.



Event Points and Sweep Line Status

Sweep Line Status

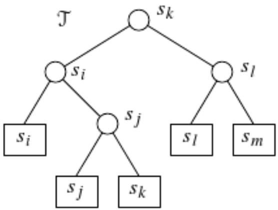
- We need to store the left to right order in which the line segments intersect ℓ. This data structure has to be dynamic.
- A line segment might come in (insertion) or go off (deletion) the sweep line. We need to search for its position.

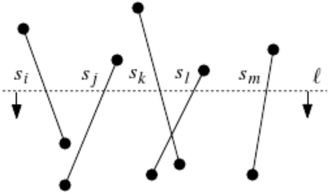


Event Points and Sweep Line Status

Sweep Line Status

- We need to store the left to right order in which the line segments intersect ℓ. This data structure has to be dynamic.
- A line segment might come in (insertion) or go off (deletion) the sweep line. We need to search for its position.
- A balanced binary search tree can support these operations in $O(\log n)$ time.

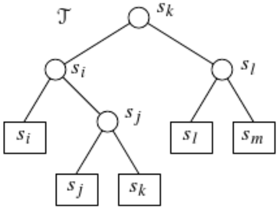


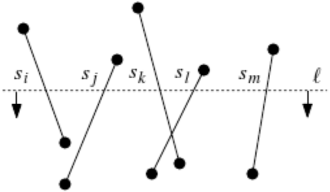


Event Points and Sweep Line Status

Sweep Line Status

- The sweep line status changes during three events: start and end points and intersection points and nowhere else.

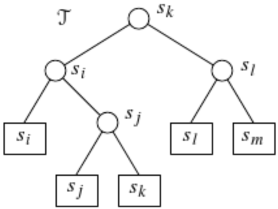


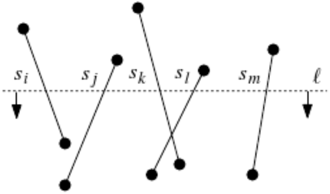


Event Points and Sweep Line Status

Sweep Line Status

- The sweep line status changes during three events: start and end points and intersection points and nowhere else.
- s_k and s_l are two segments intersecting at a point.

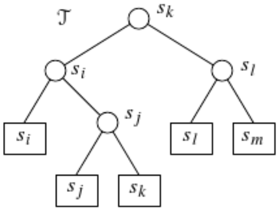


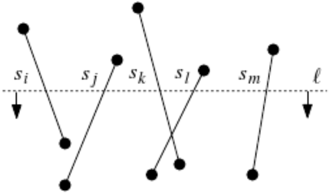


Event Points and Sweep Line Status

Sweep Line Status

- The sweep line status changes during three events: start and end points and intersection points and nowhere else.
- s_k and s_l are two segments intersecting at a point.
- There is an event point above the intersecting point where s_k and s_l are adjacent and are tested for intersection. So, no intersection point is ever missed.

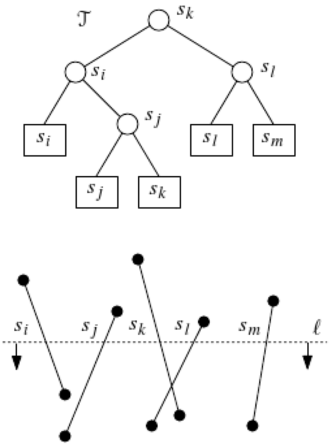




The Algorithm

Algorithm

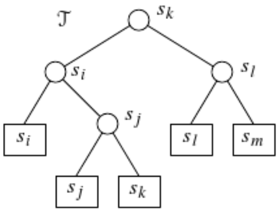
- Create a heap $\mathcal{H}$ with the y-coordinates of end points of $\mathcal{L}$. Create sweep status data structure $\mathcal{T}$ on x-coordinates of the points. Initially $\mathcal{T}$ is empty.

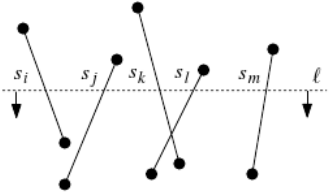


The Algorithm

Algorithm

- Create a heap $\mathcal{H}$ with the y-coordinates of end points of $\mathcal{L}$. Create sweep status data structure $\mathcal{T}$ on x-coordinates of the points. Initially $\mathcal{T}$ is empty.
- Keep on extracting points from $\mathcal{H}$ till it is non-empty.

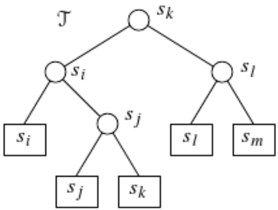


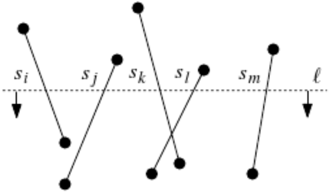


The Algorithm

Algorithm

- Create a heap $\mathcal{H}$ with the y-coordinates of end points of $\mathcal{L}$. Create sweep status data structure $\mathcal{T}$ on x-coordinates of the points. Initially $\mathcal{T}$ is empty.
- Keep on extracting points from $\mathcal{H}$ till it is non-empty.
- Based on the three cases: segment top end point, segment bottom end point and intersection point, take necessary actions on $\mathcal{T}$.

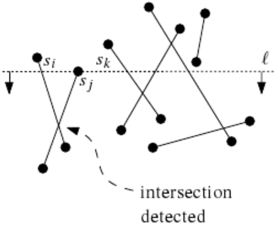




The Algorithm

Algorithm: The three cases

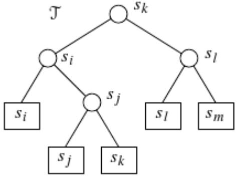
- **[Top end point]** Insert the line segment into $\mathcal{T}$ based on x- coordinates.

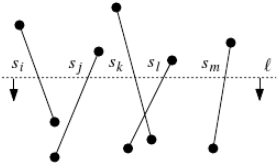


The Algorithm

Algorithm: The three cases

- **[Top end point]** Insert the line segment into $\mathcal{T}$ based on x- coordinates.
- Test for intersections with line segments to the left and right. Insert intersection point, if any, into $\mathcal{H}$.

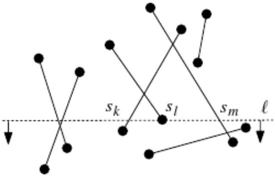




The Algorithm

Algorithm: The three cases

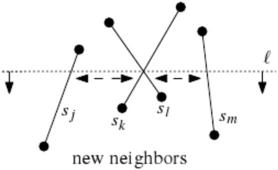
- **[Top end point]** Insert the line segment into $\mathcal{T}$ based on x- coordinates.
- Test for intersections with line segments to the left and right. Insert intersection point, if any, into $\mathcal{H}$.
- **[Bottom end point]** Delete this line segment from $\mathcal{T}$. Test for intersections between preceding and succeeding entries in $\mathcal{T}$.



The Algorithm

Algorithm: The three cases

- **[Top end point]** Insert the line segment into $\mathcal{T}$ based on x- coordinates.
- Test for intersections with line segments to the left and right. Insert intersection point, if any, into $\mathcal{H}$.
- **[Bottom end point]** Delete this line segment from $\mathcal{T}$. Test for intersections between preceding and succeeding entries in $\mathcal{T}$.
- **[Intersection point]** Swap the line segments' status in $\mathcal{T}$. Check for intersections of preceding and succeeding entries.



The Analysis

Analysis

- Total number of event points is $2n + I$, where I is the number of intersections.

The Analysis

Analysis

- Total number of event points is $2n + I$, where I is the number of intersections.
- The heap $\mathcal{H}$ grows to a size at most $2n + I$. Each operation takes $O(\log(2n + I))$. As $I < n^2$, so $O(\log(2n + I)) = O(\log n)$.

The Analysis

Analysis

- Total number of event points is $2n + I$, where I is the number of intersections.
- The heap $\mathcal{H}$ grows to a size at most $2n + I$. Each operation takes $O(\log(2n + I))$. As $I < n^2$, so $O(\log(2n + I)) = O(\log n)$.
- The balanced binary search tree $\mathcal{T}$ grows also to a size at most $2n + I$. So, each operation takes $O(\log n)$.

The Analysis

Analysis

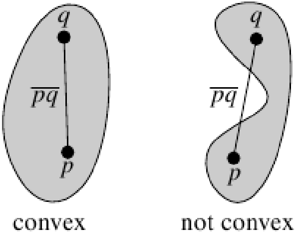
- Total number of event points is $2n + I$, where I is the number of intersections.
- The heap $\mathcal{H}$ grows to a size at most $2n + I$. Each operation takes $O(\log(2n + I))$. As $I < n^2$, so $O(\log(2n + I)) = O(\log n)$.
- The balanced binary search tree $\mathcal{T}$ grows also to a size at most $2n + I$. So, each operation takes $O(\log n)$.
- So, the total time taken is $O((2n + I) \log n) = O(n \log n + I \log n)$.

Outline

Convex Hull

Definition

A set $S \subset \mathcal{R}^2$ is convex if for any two points $p, q \in S$, $\overline{pq} \in S$.



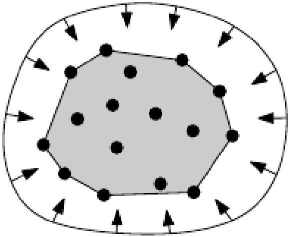
Convex Hull

Definition

A set $S \subset \mathcal{R}^2$ is convex if for any two points $p, q \in S$, $\overline{pq} \in S$.

Definition

Let $\mathcal{P}$ be a set of points in $\mathcal{R}^2$. Convex hull of $\mathcal{P}$, denoted by $CH(\mathcal{P})$, is the smallest convex set containing $\mathcal{P}$.



Convex Hull Problem

Problem

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

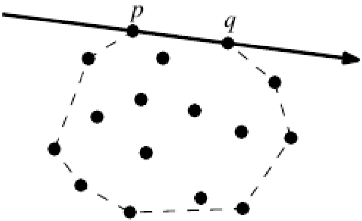
Convex Hull Problem

Problem

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

A Naive Algorithm

- Consider all line segments determined by $\binom{n}{2} = O(n^2)$ pairs of points.
- If a line segment has all the other $n - 2$ points on one side of it, then it is a hull edge.



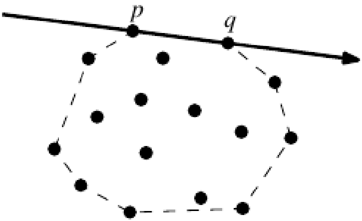
Convex Hull Problem

Problem

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

A Naive Algorithm

- Consider all line segments determined by $\binom{n}{2} = O(n^2)$ pairs of points.
- If a line segment has all the other $n - 2$ points on one side of it, then it is a hull edge.
- We need $\binom{n}{2}(n-2) = O(n^3)$ time.



Towards a Better Algorithm

Way forward, but how much?

- **Better characterizations lead to better algorithms.**

Towards a Better Algorithm

Way forward, but how much?

- **Better characterizations lead to better algorithms.**
- How much better can we make?

Towards a Better Algorithm

Way forward, but how much?

- **Better characterizations lead to better algorithms.**
- How much better can we make?
- Leads to the notion of lower bound of a problem.

Towards a Better Algorithm

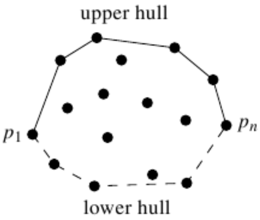
Way forward, but how much?

- **Better characterizations lead to better algorithms.**
- How much better can we make?
- Leads to the notion of lower bound of a problem.
- The problem of Convex Hull has a lower bound of $\Omega(n \log n)$. This can be shown by a reduction from the problem of sorting which also has a lower bound of $\Omega(n \log n)$.

Graham's Scan: An optimal algorithm for Convex Hull

A better characterization

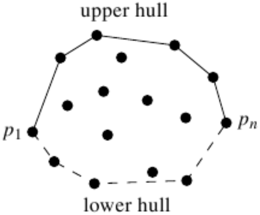
- Consider a walk in clockwise direction on the vertices of a closed polygon.



Graham's Scan: An optimal algorithm for Convex Hull

A better characterization

- Consider a walk in clockwise direction on the vertices of a closed polygon.
- Only for a convex polygon, we will make a right turn always.

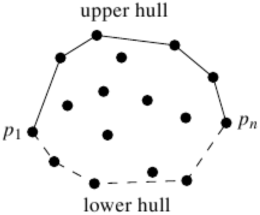


Graham's Scan: An optimal algorithm for Convex Hull

A better characterization

- Consider a walk in clockwise direction on the vertices of a closed polygon.
- Only for a convex polygon, we will make a right turn always.

The incremental paradigm



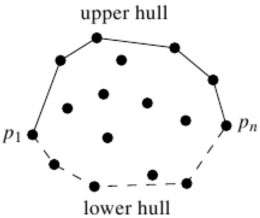
Graham's Scan: An optimal algorithm for Convex Hull

A better characterization

- Consider a walk in clockwise direction on the vertices of a closed polygon.
- Only for a convex polygon, we will make a right turn always.

The incremental paradigm

- Insert points in P one by one and update the solution at each step.



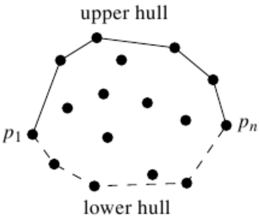
Graham's Scan: An optimal algorithm for Convex Hull

A better characterization

- Consider a walk in clockwise direction on the vertices of a closed polygon.
- Only for a convex polygon, we will make a right turn always.

The incremental paradigm

- Insert points in P one by one and update the solution at each step.
- We compute the upper hull first. The upper hull contains the convex hull edges that bound the convex hull from above.



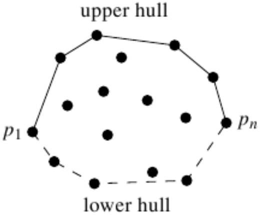
Graham's Scan: An optimal algorithm for Convex Hull

A better characterization

- Consider a walk in clockwise direction on the vertices of a closed polygon.
- Only for a convex polygon, we will make a right turn always.

The incremental paradigm

- Insert points in P one by one and update the solution at each step.
- We compute the upper hull first. The upper hull contains the convex hull edges that bound the convex hull from above.
- Sort the points in P from left to right.



Algorithm

```
Input: A set of points P
```

Algorithm

```
Input: A set of points P
Output: Convex Hull of P
```

Algorithm

```
Input: A set of points P
Output: Convex Hull of P
Sort P according to x-coordinate to generate
       a sequence of points p[1], p[2], ..., p[n];
```

Algorithm

```
Input: A set of points P
Output: Convex Hull of P
Sort P according to x-coordinate to generate
       a sequence of points p[1], p[2], ..., p[n];
Put p[1] first and then p[2] in a list L_U;
```

Algorithm

```
Input: A set of points P
Output: Convex Hull of P
Sort P according to x-coordinate to generate
       a sequence of points p[1], p[2], ..., p[n];
Put p[1] first and then p[2] in a list L_U;
for i = 3 to n {



}
```

Algorithm

```
Input: A set of points P
Output: Convex Hull of P
Sort P according to x-coordinate to generate
       a sequence of points p[1], p[2], ..., p[n];
Put p[1] first and then p[2] in a list L_U;
for i = 3 to n {
    Append p[i] to L_U;







}
```

Algorithm

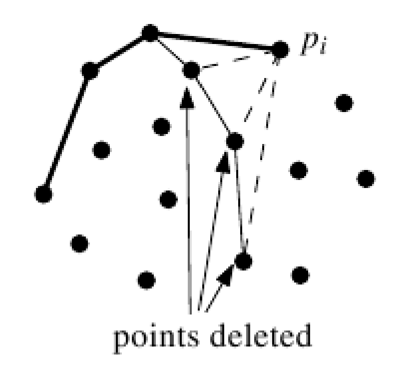
```
Input: A set of points P
Output: Convex Hull of P
Sort P according to x-coordinate to generate
       a sequence of points p[1], p[2], ..., p[n];
Put p[1] first and then p[2] in a list L_U;
for i = 3 to n {
    Append p[i] to L_U;
    while(L_U contains more than two points AND
          the last three points in L_U
          do not make a right turn) {


    }
}
```

Algorithm

```
Input: A set of points P
Output: Convex Hull of P
Sort P according to x-coordinate to generate
       a sequence of points p[1], p[2], ..., p[n];
Put p[1] first and then p[2] in a list L_U;
for i = 3 to n {
    Append p[i] to L_U;
    while(L_U contains more than two points AND
          the last three points in L_U
          do not make a right turn) {
             Delete the middle of the last
             three points from L_U;
    }
}
```

The Algorithm in Action

The algorithm in action



The Algorithm

Time Complexity

- Sorting takes time $O(n \log n)$.

The Algorithm

Time Complexity

- Sorting takes time $O(n \log n)$.
- The `for` loop is executed $O(n)$ times.

The Algorithm

Time Complexity

- Sorting takes time $O(n \log n)$.
- The `for` loop is executed $O(n)$ times.
- For each execution of the `for` loop, the `while` loop is encountered once.

The Algorithm

Time Complexity

- Sorting takes time $O(n \log n)$.
- The `for` loop is executed $O(n)$ times.
- For each execution of the `for` loop, the `while` loop is encountered once.
- For each extra execution of the `while` loop, a point gets deleted.

The Algorithm

Time Complexity

- Sorting takes time $O(n \log n)$.
- The `for` loop is executed $O(n)$ times.
- For each execution of the `for` loop, the `while` loop is encountered once.
- For each extra execution of the `while` loop, a point gets deleted.
- A point once deleted, is never deleted again.

The Algorithm

Time Complexity

- Sorting takes time $O(n \log n)$.
- The `for` loop is executed $O(n)$ times.
- For each execution of the `for` loop, the `while` loop is encountered once.
- For each extra execution of the `while` loop, a point gets deleted.
- A point once deleted, is never deleted again.
- So, the total number of extra executions is bounded by $O(n)$.

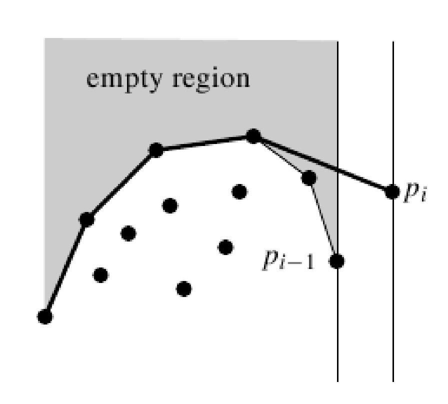
The Algorithm

Time Complexity

- Sorting takes time $O(n \log n)$.
- The `for` loop is executed $O(n)$ times.
- For each execution of the `for` loop, the `while` loop is encountered once.
- For each extra execution of the `while` loop, a point gets deleted.
- A point once deleted, is never deleted again.
- So, the total number of extra executions is bounded by $O(n)$.
- Hence, the total time complexity is $O(n \log n)$.

Proof of Correctness

The Proof of Correctness

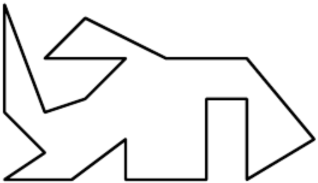


Outline

Art Gallery Problem

Problem

Given a simple polygon P of n vertices, find the minimum number of cameras that can guard P.



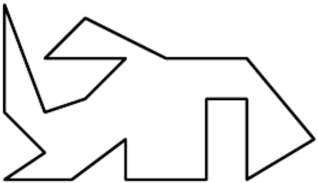
Art Gallery Problem

Problem

Given a simple polygon P of n vertices, find the minimum number of cameras that can guard P.

Hardness

The above problem is NP-Hard.



Art Gallery Problem

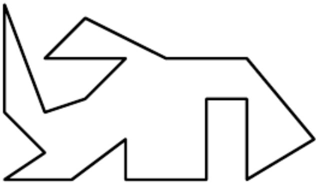
Problem

Given a simple polygon P of n vertices, find the minimum number of cameras that can guard P.

Hardness

The above problem is NP-Hard.

Any solution?



Art Gallery Problem

Problem

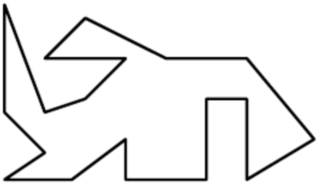
Given a simple polygon P of n vertices, find the minimum number of cameras that can guard P.

Hardness

The above problem is NP-Hard.

Any solution?

- Can we find as a function of n the number of cameras that suffices to guard P?



Art Gallery Problem

Problem

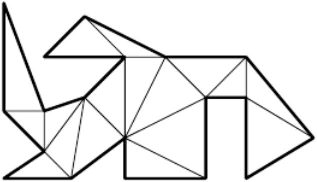
Given a simple polygon P of n vertices, find the minimum number of cameras that can guard P.

Hardness

The above problem is NP-Hard.

Any solution?

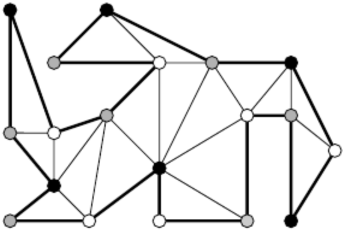
- Can we find as a function of n the number of cameras that suffices to guard P?
- Recall P can be triangulated into $n-2$ triangles. Place a guard in each triangle.



Art Gallery Problem

Can we bring the bound down?

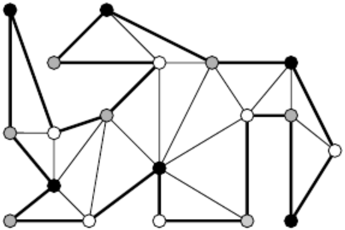
- Place guards at vertices of the triangulation $\mathcal{T}$ of P.



Art Gallery Problem

Can we bring the bound down?

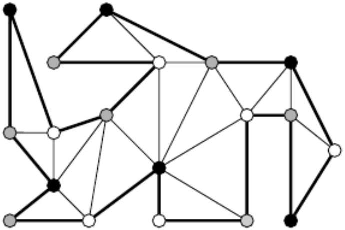
- Place guards at vertices of the triangulation $\mathcal{T}$ of P.
- We do a 3-coloring of the vertices of $\mathcal{T}$. Each triangle of $\mathcal{T}$ has a black, gray and white vertex.



Art Gallery Problem

Can we bring the bound down?

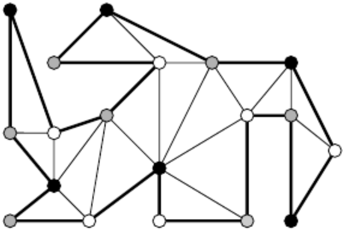
- Place guards at vertices of the triangulation $\mathcal{T}$ of P.
- We do a 3-coloring of the vertices of $\mathcal{T}$. Each triangle of $\mathcal{T}$ has a black, gray and white vertex.
- Choose the smallest color class to guard P.



Art Gallery Problem

Can we bring the bound down?

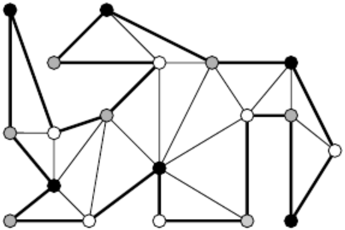
- Place guards at vertices of the triangulation $\mathcal{T}$ of P.
- We do a 3-coloring of the vertices of $\mathcal{T}$. Each triangle of $\mathcal{T}$ has a black, gray and white vertex.
- Choose the smallest color class to guard P.
- Hence, $\lfloor \frac{n}{3} \rfloor$ guards suffice.



Art Gallery Problem

Can we bring the bound down?

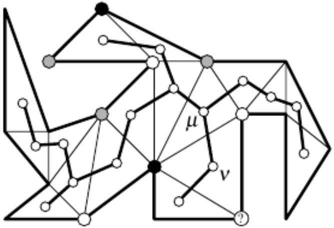
- Place guards at vertices of the triangulation $\mathcal{T}$ of P.
- We do a 3-coloring of the vertices of $\mathcal{T}$. Each triangle of $\mathcal{T}$ has a black, gray and white vertex.
- Choose the smallest color class to guard P.
- Hence, $\lfloor \frac{n}{3} \rfloor$ guards suffice.
- But, does a 3-coloring always exist?



Art Gallery Problem

A 3-coloring always exists

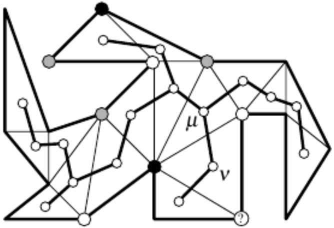
- Consider the dual graph $\mathcal{G}_{\mathcal{T}}$ of $\mathcal{T}$ of P.



Art Gallery Problem

A 3-coloring always exists

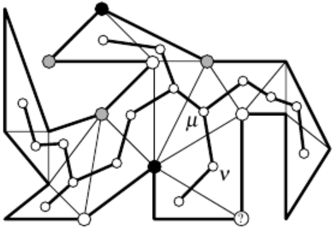
- Consider the dual graph $\mathcal{G}_{\mathcal{T}}$ of $\mathcal{T}$ of P.
- $\mathcal{G}_{\mathcal{T}}$ is a tree as P has no holes.



Art Gallery Problem

A 3-coloring always exists

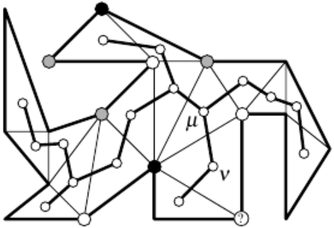
- Consider the dual graph $\mathcal{G}_{\mathcal{T}}$ of $\mathcal{T}$ of P.
- $\mathcal{G}_{\mathcal{T}}$ is a tree as P has no holes.
- Do a DFS on $\mathcal{G}_{\mathcal{T}}$ to obtain the coloring.



Art Gallery Problem

A 3-coloring always exists

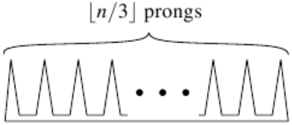
- Consider the dual graph $\mathcal{G}_{\mathcal{T}}$ of $\mathcal{T}$ of P.
- $\mathcal{G}_{\mathcal{T}}$ is a tree as P has no holes.
- Do a DFS on $\mathcal{G}_{\mathcal{T}}$ to obtain the coloring.
- Place guards at those vertices that have color of the minimum color class. Hence, $\lfloor \frac{n}{3} \rfloor$ guards are sufficient to guard P.



Art Gallery Problem

A 3-coloring always exists

- Consider the dual graph $\mathcal{G}_{\mathcal{T}}$ of $\mathcal{T}$ of P.
- $\mathcal{G}_{\mathcal{T}}$ is a tree as P has no holes.
- Do a DFS on $\mathcal{G}_{\mathcal{T}}$ to obtain the coloring.
- Place guards at those vertices that have color of the minimum color class. Hence, $\lfloor \frac{n}{3} \rfloor$ guards are sufficient to guard P.



Necessity?

Are $\lfloor \frac{n}{3} \rfloor$ guards sometimes necessary?

Art Gallery Theorem

The Final Theorem

For a simple polygon with n vertices, $\lfloor \frac{n}{3} \rfloor$ cameras are always sufficient and occasionally necessary to have every point in the polygon visible from at least one of the cameras.

Bibliography

- Michael Ian Shamos, *Computational Geometry*, PhD thesis, Yale University, New Haven.
- Franco P. Preparata and Michael Ian Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
- Joseph O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1998.
- Mark de Berg, Marc van Kreveld, Mark Overmars and Otfried Schwarzkof, *Computational Geometry: Algorithms and Applications*, Springer, 1997.
- http://www.algorithmic-solutions.com
- http://www.cgal.org
- B. Chazelle, *Triangulating a simple polygon in linear time*, Discrete Comput. Geom., 6:485524, 1991.