Introduction to Computational Geometry

Partha P. Goswami
(ppg.rpe@caluniv.ac.in)

Institute of Radiophysics and Electronics
University of Calcutta
92, APC Road, Kolkata - 700009, 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 areas in computer science like computer graphics, computer vision and image processing, robotics, computer-aided designing (CAD), geographic information systems (GIS), 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 areas in computer science like computer graphics, computer vision and image processing, robotics, computer-aided designing (CAD), geographic information systems (GIS), 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 areas in computer science like computer graphics, computer vision and image processing, robotics, computer-aided designing (CAD), geographic information systems (GIS), 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 [6] 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 techniques to be applied to areas that involves 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 techniques to be applied to areas that involves continuous issues, discrete approximations to continuous curves or surfaces are needed.
- Programming in CG is also a little difficult. Fortunately, libraries like LEDA [7] and CGAL [8] are now available. These libraries implement various data structures and algorithms specific to 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 techniques to be applied to areas that involves continuous issues, discrete approximations to continuous curves or surfaces are needed.
- Programming in CG is also a little difficult. Fortunately, libraries like LEDA [7] and CGAL [8] are now available. These libraries implement various data structures and algorithms specific to CG.
- CG algorithms suffer from the curse of degeneracies. So, we would make certain simplifying assumptions at times like no three points are collinear, no four points are cocircular, etc.

Introduction

- In this lecture, we touch upon a few simple topics for having a glimpse of the area of computational geometry.

Introduction

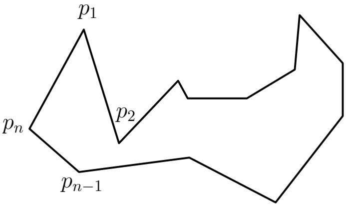
- In this lecture, we touch upon a few simple topics for having a glimpse of the area of computational geometry.
- First we consider some geometric primitives, that is, problems that arise frequently in computational geometry.

Outline

Area Computation

Problem

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



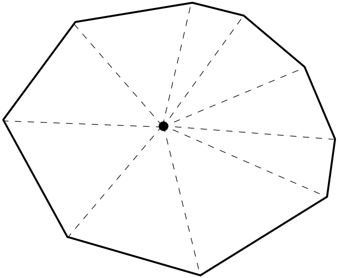
Area Computation

Problem

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

Area of a convex polygon

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



Area Computation

Problem

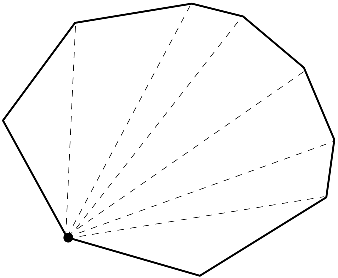
Given a simple polygon P of n vertices, compute its area.

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.



Area Computation

Problem

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

Area of a convex polygon

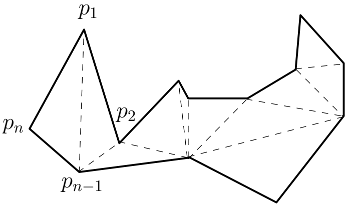
Find a point inside P, draw n triangles and compute the area.

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.



Area Computation

Result

If P be a simple polygon with n vertices with coordinates of the vertex p_i being (x_i, y_i), $1 \leq i \leq n$, then twice the area of P is given by

$$2\mathcal{A}(P) = \sum_{i=1}^{n}(x_i y_{i+1} - y_i x_{i+1})$$

Polygon Triangulation

Theorem

Any simple polygon can be triangulated.

Polygon Triangulation

Theorem

Any simple polygon can be triangulated.

Theorem

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

Polygon Triangulation

Theorem

Any simple polygon can be triangulated.

Theorem

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

Proof.

The proof is by induction on n. □

Polygon Triangulation

Theorem

Any simple polygon can be triangulated.

Theorem

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

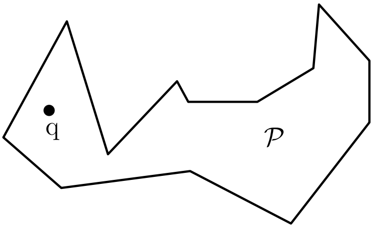
We can triangulate P by a very complicated $O(n)$ algorithm [2] OR by a *more or less simple* $O(n \log n)$ time algorithm [1].

Outline

Point Inclusion

Problem

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



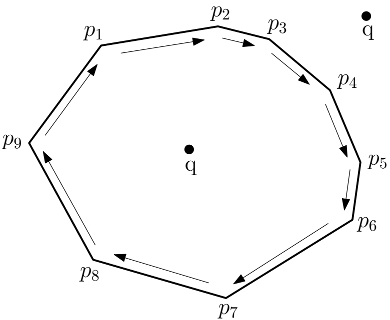
Point Inclusion

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 $O(\log n)$. Left as an exercise.



q is always to the right if $q \in \mathcal{P}$, else, it varies

Point Inclusion

Problem

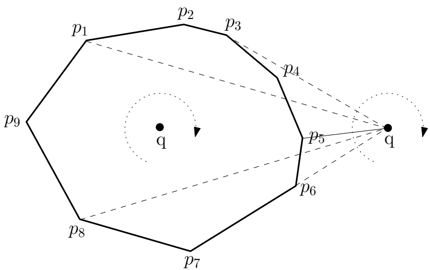
Given a simple polygon P of n points, and a query point q, is $q \in P$?

What if P is convex?

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

Another idea for convex polygon

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

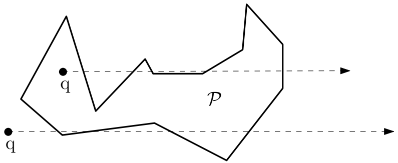


Total angular turn around q is 2π if $q \in \mathcal{P}$, else, 0

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 $O(n)$.

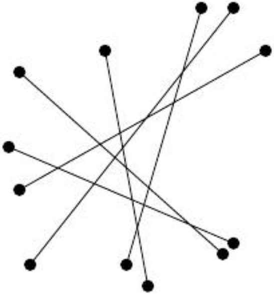


Outline

Line Segment Intersection

Input

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



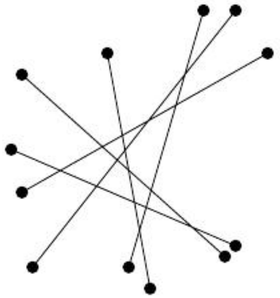
Line Segment Intersection

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

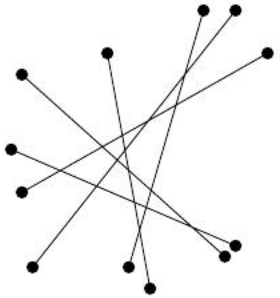
A set of line segments $\mathcal{L}$ in general position in the plane. $|\mathcal{L}| = n$.

Output

Report the intersections.

Output Sensitive Algorithm

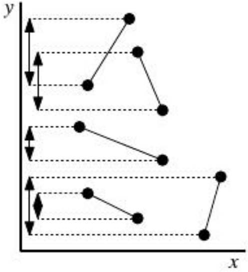
Number of intersections might vary from 0 to $\binom{n}{2} = O(n^2)$. So, the lower bound of the problem is $\Omega(n^2)$. The idea is now to look for an output sensitive algorithm.



An Output Sensitive Algorithm

The idea

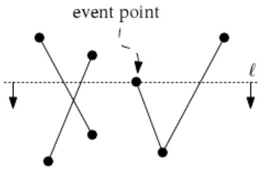
- Avoid testing pairs of segments that are far apart.



An Output Sensitive Algorithm

The idea

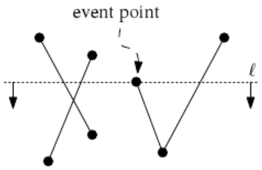
- Avoid testing pairs of segments that are far apart.
- To find such pairs, imagine sweeping a horizontal line ℓ downwards from above all segments.



An Output Sensitive Algorithm

The idea

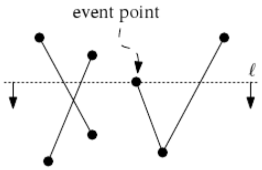
- Avoid testing pairs of segments that are far apart.
- To find such pairs, imagine sweeping a horizontal line ℓ downwards from above all segments.
- Keep track of all segments that intersect ℓ.



An Output Sensitive Algorithm

The idea

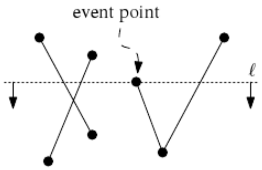
- Avoid testing pairs of segments that are far apart.
- To find such pairs, imagine sweeping a horizontal line ℓ downwards from above all segments.
- Keep track of all segments that intersect ℓ.
- ℓ is the sweep line and the algorithm paradigm is plane sweep.



An Output Sensitive Algorithm

The idea

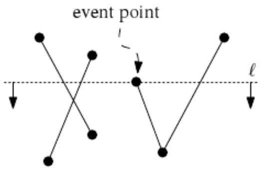
- Avoid testing pairs of segments that are far apart.
- To find such pairs, imagine sweeping a horizontal line ℓ downwards from above all segments.
- Keep track of all segments that intersect ℓ.
- ℓ is the sweep line and the algorithm paradigm is plane sweep.
- The status of the sweep line is the line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



An Output Sensitive Algorithm

The idea

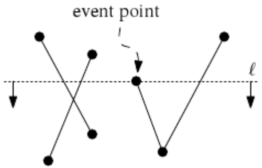
- Avoid testing pairs of segments that are far apart.
- To find such pairs, imagine sweeping a horizontal line ℓ downwards from above all segments.
- Keep track of all segments that intersect ℓ.
- ℓ is the sweep line and the algorithm paradigm is plane sweep.
- The status of the sweep line is the line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

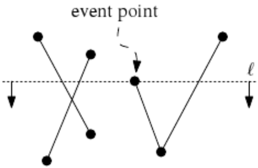
- The start and end points of each line segment. They are static.



Event Points and Sweep Line Status

Event Points and the Event Queue

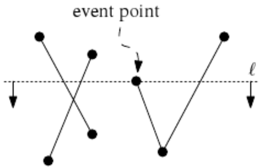
- The start and end points of each line segment. They are static.
- The intersection points. They are dynamic and are generated as the sweep line ℓ sweeps down.



Event Points and Sweep Line Status

Event Points and the Event Queue

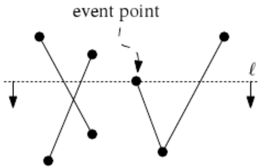
- The start and end points of each line segment. They are static.
- The intersection points. They are dynamic and are generated as the sweep line ℓ sweeps down.
- The event points are to be arranged in a data structure in a way in which the sweep line sees them.



Event Points and Sweep Line Status

Event Points and the Event Queue

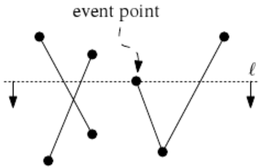
- The start and end points of each line segment. They are static.
- The intersection points. They are dynamic and are generated as the sweep line ℓ sweeps down.
- The event points are to be arranged in a data structure in a way in which the sweep line sees them.
- The data structure should support extracting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



Event Points and Sweep Line Status

Event Points and the Event Queue

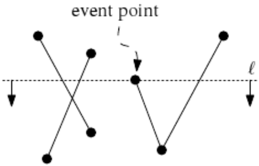
- The start and end points of each line segment. They are static.
- The intersection points. They are dynamic and are generated as the sweep line ℓ sweeps down.
- The event points are to be arranged in a data structure in a way in which the sweep line sees them.
- The data structure should support extracting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

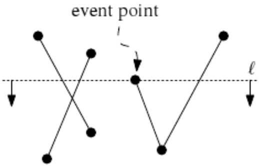
- We need to store the left to right order in which the line segments intersect ℓ. This data structure has to be dynamic.



Event Points and Sweep Line Status

Sweep Line Status

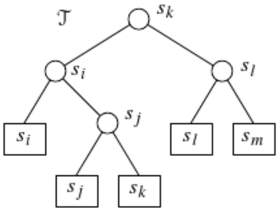
- We need to store the left to right order in which the line segments intersect ℓ. This data structure has to be dynamic.
- A line segment might come in (insertion) or go off (deletion) the sweep line. We need to search for its position.

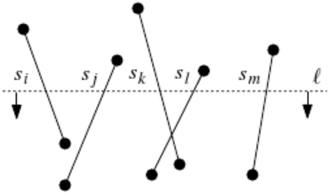


Event Points and Sweep Line Status

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.

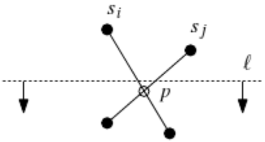




Event Points and Sweep Line Status

Sweep Line Status

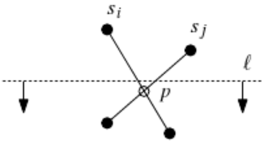
- The sweep line status changes during three events: start and end points and intersection points and nowhere else.



Event Points and Sweep Line Status

Sweep Line Status

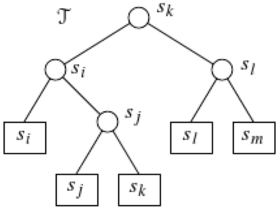
- The sweep line status changes during three events: start and end points and intersection points and nowhere else.
- s_i and s_j are two segments intersecting at p.

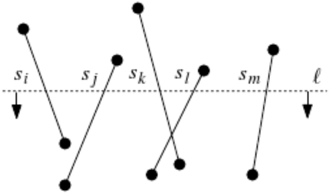


Event Points and Sweep Line Status

Sweep Line Status

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

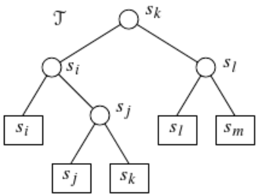


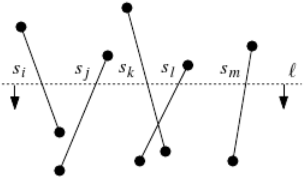


The Algorithm

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.

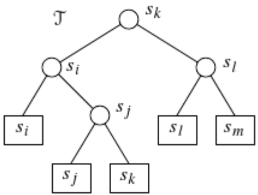


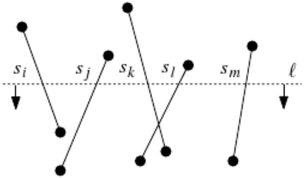


The Algorithm

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.

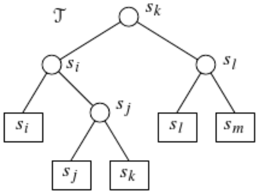


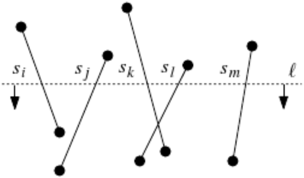


The Algorithm

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, we take necessary actions.

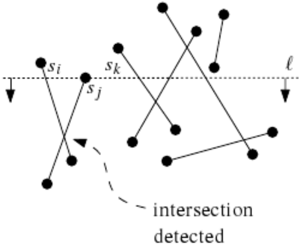




The Algorithm

Algorithm

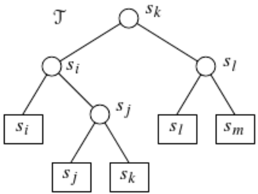
- **[Top end point]** Insert the line segment into $\mathcal{T}$ based on x-coordinates.

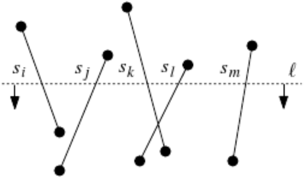


The Algorithm

Algorithm

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

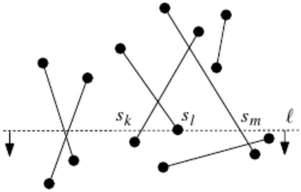




The Algorithm

Algorithm

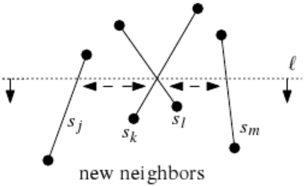
- **[Top end point]** Insert the line segment into $\mathcal{T}$ based on x-coordinates.
- Test for intersections with line segmentes to the left and right. Insert intersection point, if any, into $\mathcal{H}$.
- **[Bottom end point]** Delete this line segment from $\mathcal{T}$. Test for intersections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



The Algorithm

Algorithm

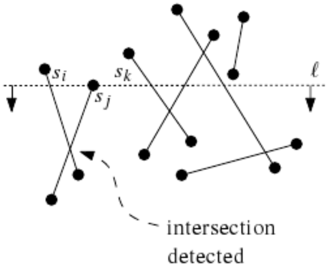
- **[Top end point]** Insert the line segment into $\mathcal{T}$ based on x-coordinates.
- Test for intersections with line segmentes to the left and right. Insert intersection point, if any, into $\mathcal{H}$.
- **[Bottom end point]** Delete this line segment from $\mathcal{T}$. Test for intersections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

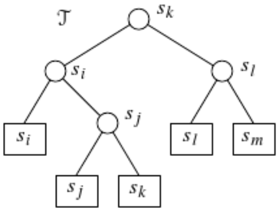
- The heap $\mathcal{H}$ grows to a size at most $2n + I$ where I is the number of intersections. Each operation takes $O(\log(2n + I))$. As $I \leq n^2$, so $O(\log(2n + I)) = O(\log n)$.

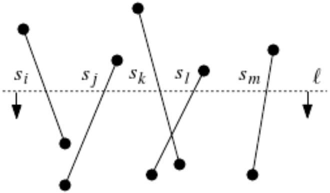


The Analysis

Analysis

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

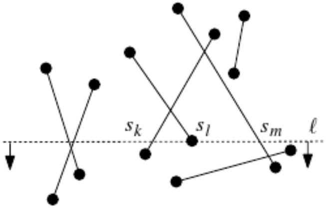




The Analysis

Analysis

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

Definitions

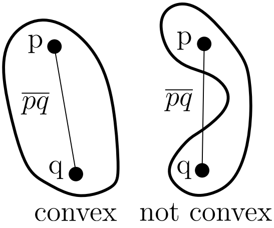
Definition

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

Definitions

Definition

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



Definitions

Definition

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

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 $\mathcal{P}$.

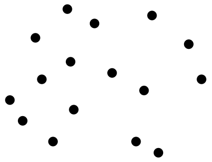
Definitions

Definition

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

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 $\mathcal{P}$.



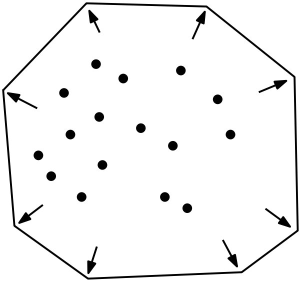
Definitions

Definition

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

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 $\mathcal{P}$.



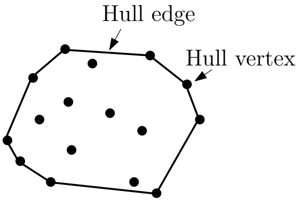
Definitions

Definition

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

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 $\mathcal{P}$.

A Naive Algorithm

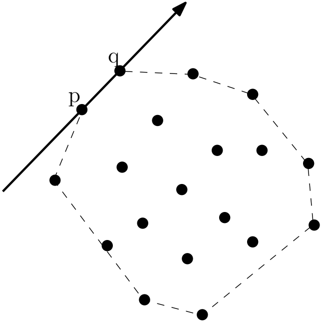
Outline

- Consider all line segments determined by $\binom{n}{2} = O(n^2)$ pairs of points.

A Naive Algorithm

Outline

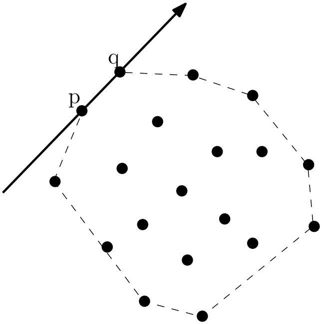
- Consider all line segments determined by $\binom{n}{2} = O(n^2)$ pairs of points.
- If a line segment has all the other $n - 2$ points on one side of it, then it is a hull edge.



A Naive Algorithm

Outline

- 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

How much betterment is possible?

- Better characterizations lead to better algorithms.

Towards a Better Algorithm

How much betterment is possible?

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

Towards a Better Algorithm

How much betterment is possible?

- 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

How much betterment is possible?

- 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 $\Omega(n \log n)$.

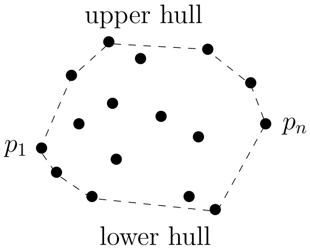
Optimal Algorithms

- 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

A better characterization

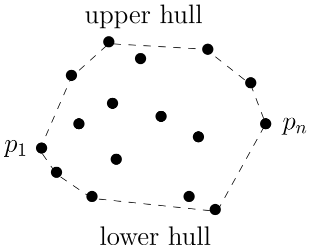
- Consider a walk in clockwise direction on the vertices of a closed polygon.



Definitions

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.

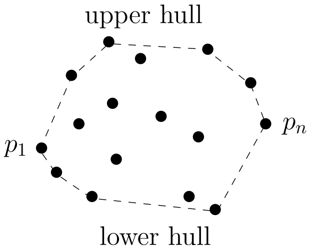


Definitions

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



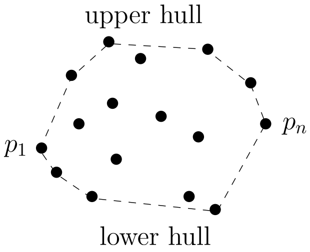
Definitions

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.



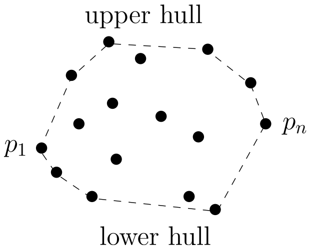
Definitions

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.



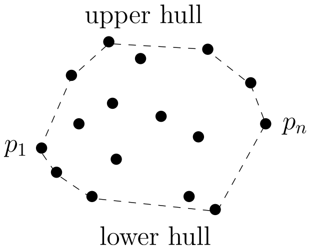
Definitions

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 $\mathcal{P}$ from left to right.



A Naive Algorithm

`Input: A set P of n points in the plane`

A Naive Algorithm

```
Input: A set P of n points in the plane
Output: Convex Hull of P
```

A Naive Algorithm

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

A Naive Algorithm

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

A Naive Algorithm

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







}
```

A Naive Algorithm

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







}
```

A Naive Algorithm

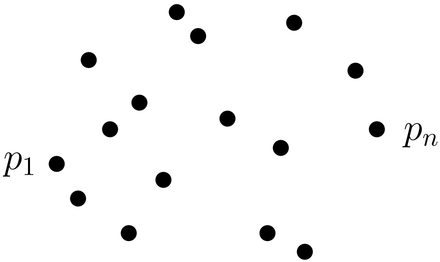
```
Input: A set P of n points in the plane
Output: Convex Hull of P
Sort P according to x-coordinate to generate
   a sequence of points p[1], p[2], ..., p[n];
Insert p[1] and then p[2] in a list L_U;
for i = 3 to n {
   Append p[i] to L_U;
   while(L_U contains more than two points AND
      the last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


   }
}
```

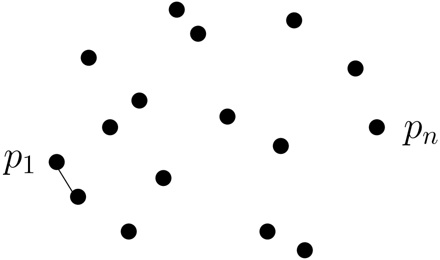
A Naive Algorithm

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

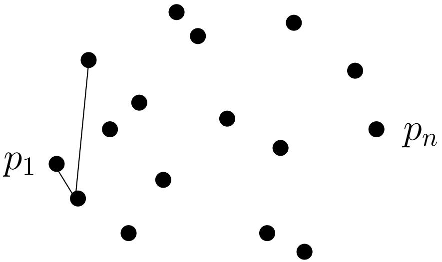
The Algorithm in Action



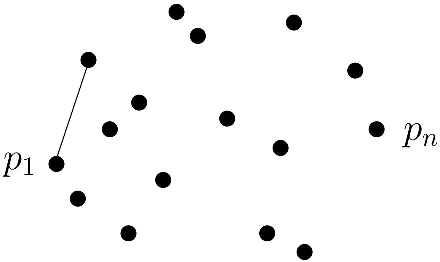
The Algorithm in Action



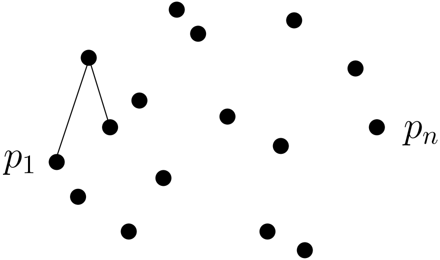
The Algorithm in Action



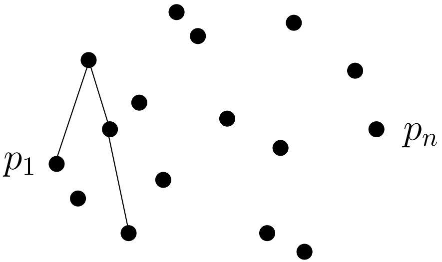
The Algorithm in Action



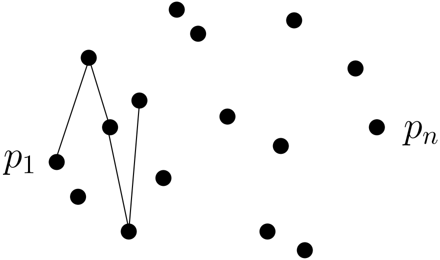
The Algorithm in Action



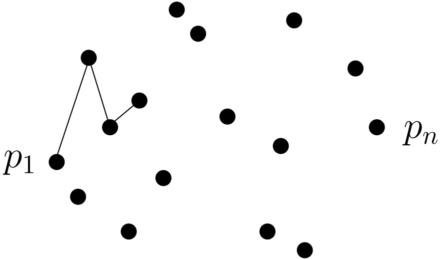
The Algorithm in Action



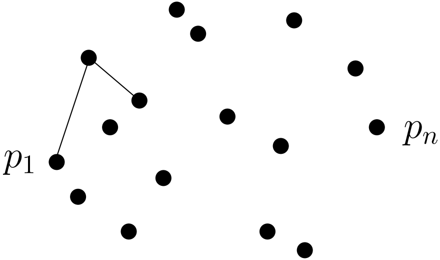
The Algorithm in Action



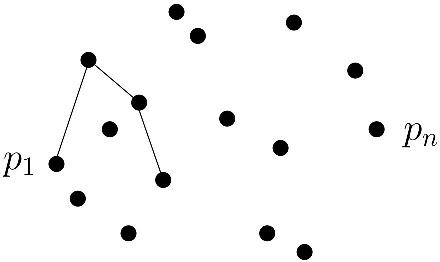
The Algorithm in Action



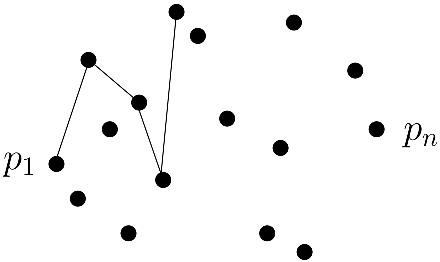
The Algorithm in Action



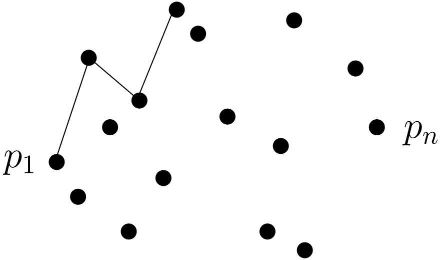
The Algorithm in Action



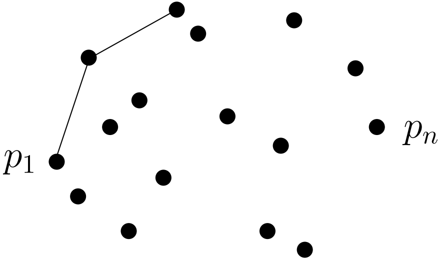
The Algorithm in Action



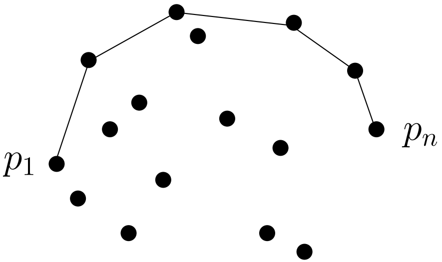
The Algorithm in Action



The Algorithm in Action



The Algorithm in Action



Analysis

Time complexity

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

Analysis

Time complexity

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

Analysis

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.

Analysis

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 execution of the while loop body, a point gets deleted.

Analysis

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 execution of the while loop body, a point gets deleted.
- A point once deleted, is never deleted again.

Analysis

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 execution of the while loop body, a point gets deleted.
- A point once deleted, is never deleted again.
- So, the total number of executions of the while loop body is bounded by $O(n)$.

Analysis

Time complexity

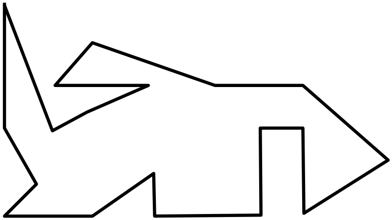
- Sorting takes time $O(n \log n)$.
- The for loop is executed $O(n)$ times.
- For each execution of the for loop, the while loop is encountered once.
- For each execution of the while loop body, a point gets deleted.
- A point once deleted, is never deleted again.
- So, the total number of executions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

Outline

Art Gallery Problem

The problem

Given a simple polygon $\mathcal{P}$ of n vertices, find the minimum number of cameras that can guard $\mathcal{P}$.



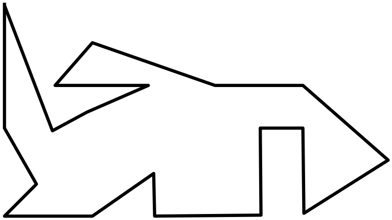
Art Gallery Problem

The problem

Given a simple polygon $\mathcal{P}$ of n vertices, find the minimum number of cameras that can guard $\mathcal{P}$.

Hardness

The above problem is NP-Hard.



Art Gallery Problem

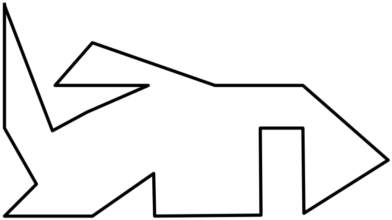
The problem

Given a simple polygon $\mathcal{P}$ of n vertices, find the minimum number of cameras that can guard $\mathcal{P}$.

Hardness

The above problem is NP-Hard.

Any solution?



Art Gallery Problem

The problem

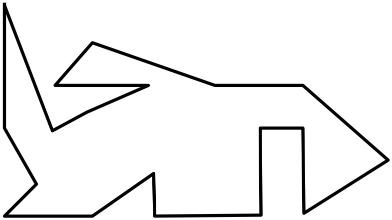
Given a simple polygon $\mathcal{P}$ of n vertices, find the minimum number of cameras that can guard $\mathcal{P}$.

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 $\mathcal{P}$?



Art Gallery Problem

The problem

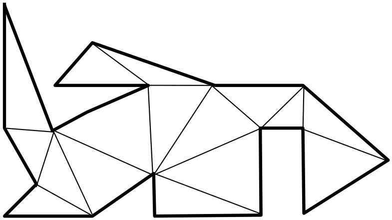
Given a simple polygon $\mathcal{P}$ of n vertices, find the minimum number of cameras that can guard $\mathcal{P}$.

Hardness

The above problem is NP-Hard.

Any solution?

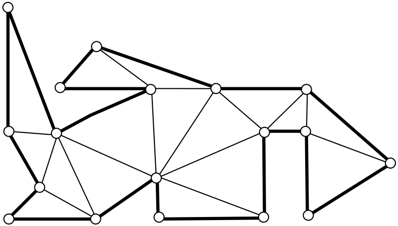
- Can we find, as a function of n, the number of cameras that suffices to guard $\mathcal{P}$?
- Recall $\mathcal{P}$ can be triangulated into $n-2$ triangles. Place a guard in each triangle.



Art Gallery Problem

Can the bound be reduced?

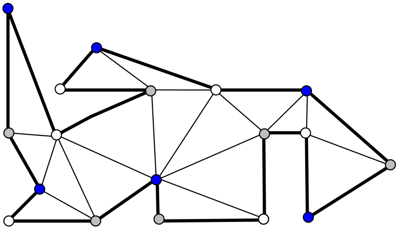
- Place guards at vertices of the triangulation $\mathcal{T}$ of $\mathcal{P}$.



Art Gallery Problem

Can the bound be reduced?

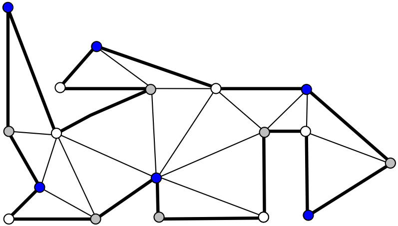
- Place guards at vertices of the triangulation $\mathcal{T}$ of $\mathcal{P}$.
- We do a 3-coloring of the vertices of $\mathcal{T}$. Each triangle of $\mathcal{T}$ has a blue, gray and white vertex.



Art Gallery Problem

Can the bound be reduced?

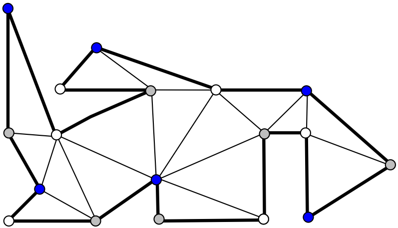
- Place guards at vertices of the triangulation $\mathcal{T}$ of $\mathcal{P}$.
- We do a 3-coloring of the vertices of $\mathcal{T}$. Each triangle of $\mathcal{T}$ has a blue, gray and white vertex.
- Choose the smallest color class to guard $\mathcal{P}$.



Art Gallery Problem

Can the bound be reduced?

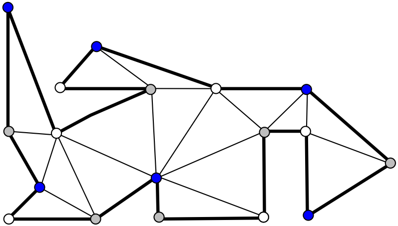
- Place guards at vertices of the triangulation $\mathcal{T}$ of $\mathcal{P}$.
- We do a 3-coloring of the vertices of $\mathcal{T}$. Each triangle of $\mathcal{T}$ has a blue, gray and white vertex.
- Choose the smallest color class to guard $\mathcal{P}$.
- Hence, $\lfloor \frac{n}{3} \rfloor$ guards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



Art Gallery Problem

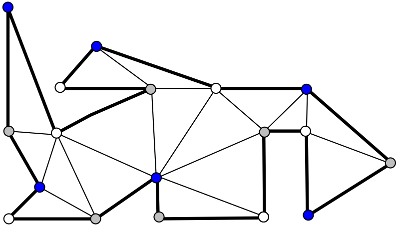
Can the bound be reduced?

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



Art Gallery Problem

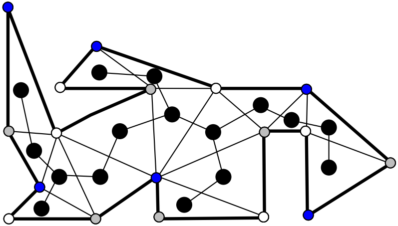
A 3-coloring always exist



Art Gallery Problem

A 3-coloring always exist

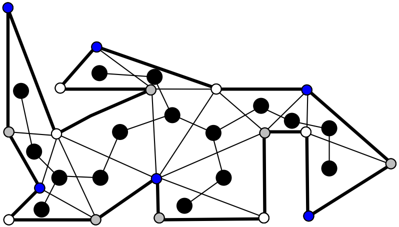
- Consider the dual graph $\mathcal{G}_{\mathcal{T}}$ of $\mathcal{T}$ of $\mathcal{P}$.



Art Gallery Problem

A 3-coloring always exist

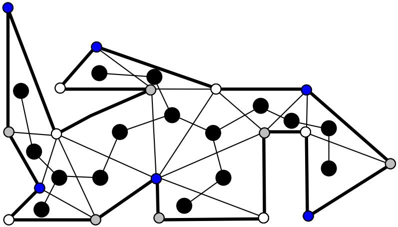
- Consider the dual graph $\mathcal{G}_{\mathcal{T}}$ of $\mathcal{T}$ of $\mathcal{P}$.
- $\mathcal{G}_{\mathcal{T}}$ is a tree as $\mathcal{P}$ has no holes.



Art Gallery Problem

A 3-coloring always exist

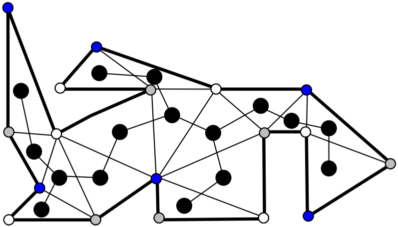
- Consider the dual graph $\mathcal{G}_{\mathcal{T}}$ of $\mathcal{T}$ of $\mathcal{P}$.
- $\mathcal{G}_{\mathcal{T}}$ is a tree as $\mathcal{P}$ has no holes.
- Do a DFS on $\mathcal{G}_{\mathcal{T}}$ to obtain the coloring.



Art Gallery Problem

A 3-coloring always exist

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



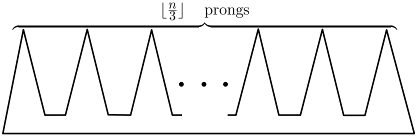
Art Gallery Problem

A 3-coloring always exist

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

Necessity?

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



Art Gallery Theorem

Final Result

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.

References I

- Mark de Berg, Marc van Kreveld, Mark Overmars and Otfried Schwarzkof, *Computational Geometry: Algorithms and Applications*, Springer, 1997.
- B. Chazelle, *Triangulating a simple polygon in linear time*, Discrete Comput. Geom., 6:485524, 1991.
- Herbert Edelsbrunner, *Algorithms in Computational Geometry*, Springer, 1987.
- Joseph O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1998.
- Franco P. Preparata and Michael Ian Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
- Michael Ian Shamos, *Computational Geometry*, PhD thesis, Yale University, New Haven., 1978.

References II

- http://www.algorithmic-solutions.com
- http://www.cgal.org
- http://en.wikipedia.org/wiki/Computational_geometry

Thank you!