

# 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 nature of geometric problems** as opposed to continuous issues.

# Introduction

- Computational Geometry (CG) involves study of algorithms for solving **geometric problems** on a computer. The emphasis is more on **discrete nature of geometric problems** as opposed to continuous issues.
- 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 nature of geometric problems** as opposed to continuous issues.
- 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.
- If one assumes Michael Ian Shamos's thesis [[Shamos M. I., 1978](#)] as the starting point, then this branch of study is around forty years old.

# 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.
- 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

- 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.
- 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.
- Programming in CG is a little difficult. Fortunately, libraries like **LEDA** [[LEDA, www.algorithmic-solutions.com](http://www.algorithmic-solutions.com)] and **CGAL** [[CGAL, www.cgal.com](http://www.cgal.com)] are now available. These libraries implement various data structures and algorithms specific to CG.

# 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.

# 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.
- Then we study a few classical CG problems.

# Outline

# Area Computation

## Problem

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

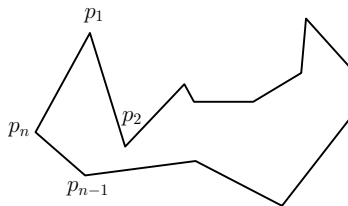
# Area Computation

## Problem

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

## Definition

A simple polygon is the region of a plane bounded by a finite collection of line segments forming a simple closed curve.



# Area Computation

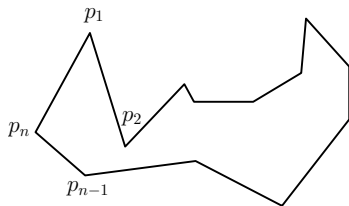
## Problem

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

## Definition

A simple polygon is the region of a plane bounded by a finite collection of line segments forming a simple closed curve.

- Let us first solve the problem for convex polygon.

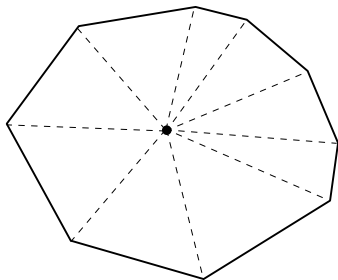




# Area Computation

## Area of a convex polygon

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



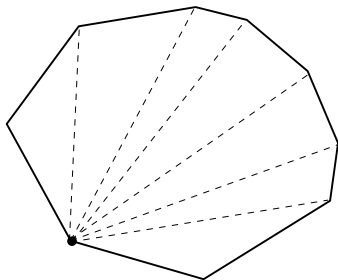
# Area Computation

## 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

## Area of a convex polygon

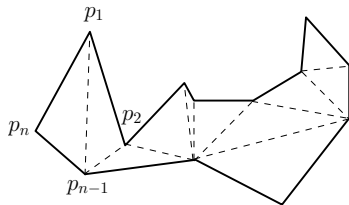
Find a point inside  $P$ , draw  $n$  triangles and compute the area.

## A better idea for convex polygon

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

## Area of a 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  $O(n)$  time algorithm [Chazelle B., 1991] OR by a *more or less simple*  $O(n \log n)$  time algorithm [Berg M. d. et. al., 1997].

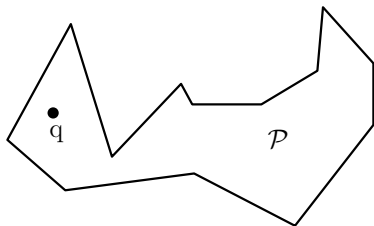


# Outline

# Point Inclusion

## Problem

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

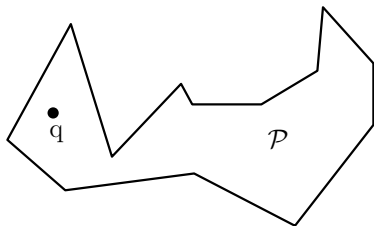


# Point Inclusion

## Problem

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

What if  $P$  is a triangle?



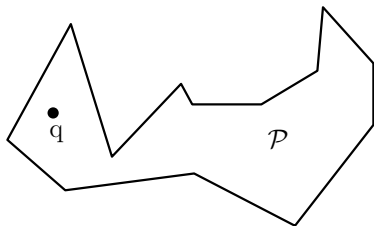
# Point Inclusion

## Problem

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

## What if $P$ is a triangle?

- Can be done in  $O(1)$  time.



# Point Inclusion

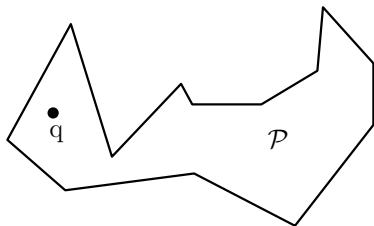
## Problem

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

## What if $P$ is a triangle?

- Can be done in  $O(1)$  time.

## What if $P$ is convex?



# Point Inclusion

## Problem

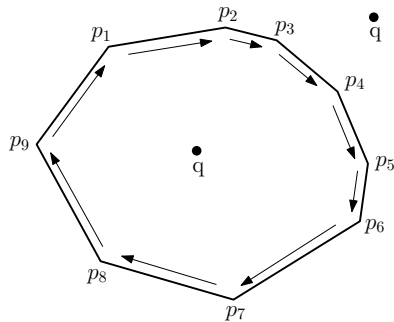
Given a simple polygon  $P$  of  $n$  points, and a query point  $q$ , is  $q \in P$ ?

## What if $P$ is a triangle?

- Can be done in  $O(1)$  time.

## What if $P$ is convex?

- Can be done in  $O(n)$  time.



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

# Point Inclusion

## Problem

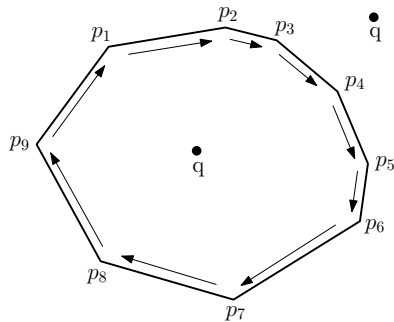
Given a simple polygon  $P$  of  $n$  points, and a query point  $q$ , is  $q \in P$ ?

## What if $P$ is a triangle?

- Can be done in  $O(1)$  time.

## What if $P$ is convex?

- Can be done in  $O(n)$  time.
- Takes a little effort to do it in  $O(\log n)$  time. Left as an **exercise**.



$q$  is always to the right 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 a triangle?

- Can be done in  $O(1)$  time.

## What if $P$ is convex?

- Can be done in  $O(n)$  time.
- Takes a little effort to do it in  $O(\log n)$  time. Left as an **exercise**.

What if  $P$  is an arbitrary simple polygon?



# Point Inclusion

## Problem

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

## What if $P$ is a triangle?

- Can be done in  $O(1)$  time.

## What if $P$ is convex?

- Can be done in  $O(n)$  time.
- Takes a little effort to do it in  $O(\log n)$  time. Left as an **exercise**.

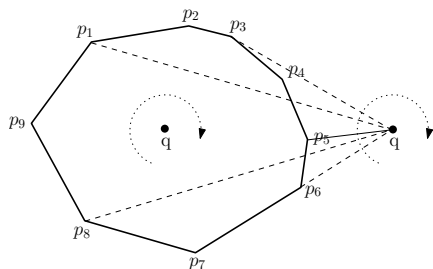
## What if $P$ is an arbitrary simple polygon?

- Can be done in  $O(n)$  time.

# Point Inclusion: Another Idea

For convex polygon

Walk around the polygon and compute total angle subtended at  $q$ . Time complexity is  $O(n)$ .



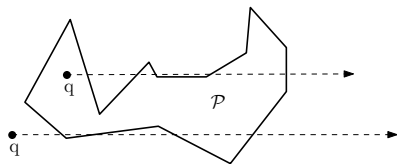
Total angular turn around  $q$  is  $2\pi$  if  $q \in \mathcal{P}$ ,  
else, 0



# Point Inclusion: Still another Idea

## Ray Shooting

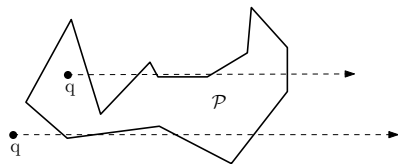
- Shoot a **ray** and count the number of **crossings** with edges of  $P$ . If it is odd, then  $q \in P$ . If it is even, then  $q \notin P$ .



# Point Inclusion: Still another Idea

## Ray Shooting

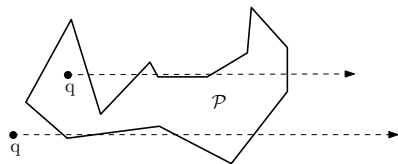
- Shoot a **ray** and count the number of **crossings** with edges of  $P$ . If it is odd, then  $q \in P$ . If it is even, then  $q \notin P$ .
- Time complexity is  $O(n)$ .



# Point Inclusion: Still another Idea

## 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  $q \notin P$ .
- Time complexity is  $O(n)$ .
- Some degenerate cases need to be taken care of.



# Outline

# Definitions

## Definition

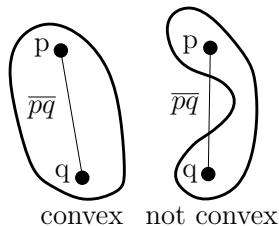
A set  $S \subset \mathcal{R}^2$  is convex if for any two points  $p, q \in S$ ,  $\overline{pq} \in S$ .



# Definitions

## Definition

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



# Definitions

## 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  $\mathcal{P}$ .

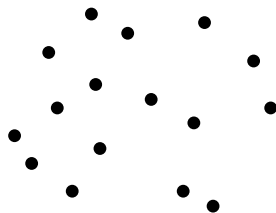
# Definitions

## 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  $\mathcal{P}$ .



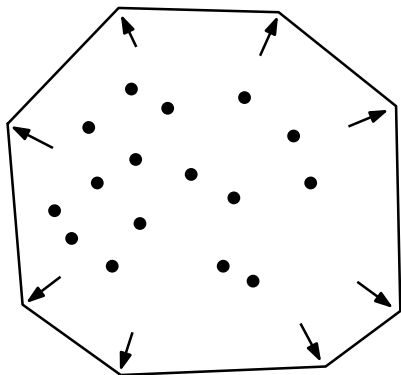
# Definitions

## Definition

A set  $S \subset \mathcal{R}^2$  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  $\mathcal{P}$ .



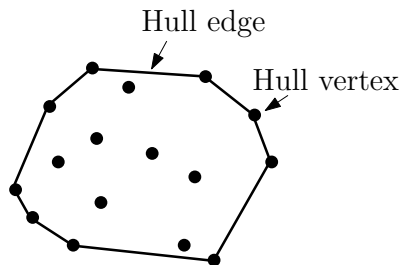
# Definitions

## 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  $\mathcal{P}$ .*

# A Naive Algorithm

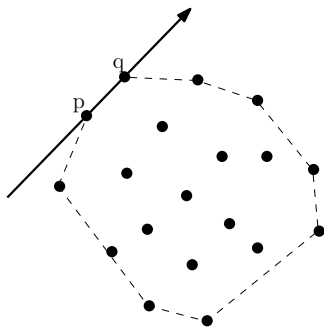
## Outline

- 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.

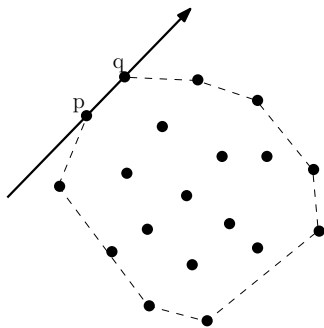




# A Naive Algorithm

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

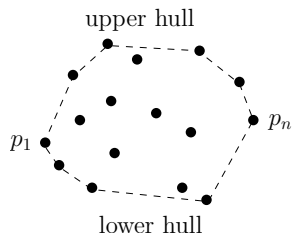
# Optimal Algorithms

- **Grahams scan**, time complexity  $O(n \log n)$   
(Graham, R.L., 1972).
- **Divide and conquer algorithm**, time complexity  $O(n \log n)$   
(Preparata, F. P. and Hong, S. J., 1977).
- **Jarvis's march** or **gift wrapping algorithm**, time complexity  $O(nh)$  where  $h$  is the 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  $O(n \log h)$   
(T. M. Chan, 1996).

# Definitions

## A better characterization

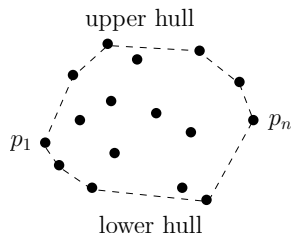
- Consider a walk in clockwise direction on the vertices of a closed polygon.



# 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.



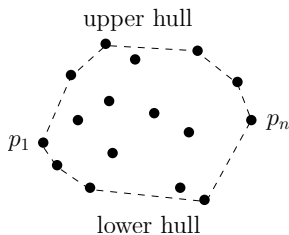


# Definitions

## 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 approach



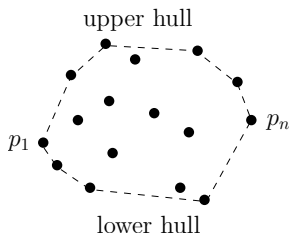
# Definitions

## 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 approach

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



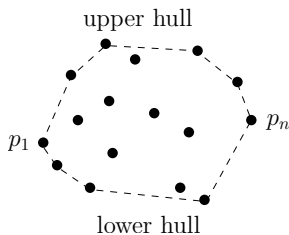
# Definitions

## 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 approach

- 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.



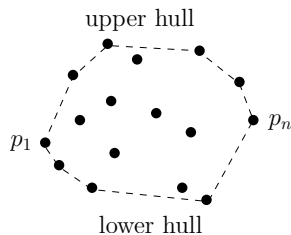
# Definitions

## 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 approach

- 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.
- The lower hull can be computed in a similar fashion.



# The pseudocode

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

# The pseudocode

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

Output: Vertices of  $CH(P)$  in clockwise order

# The pseudocode

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

Output: Vertices of  $CH(P)$  in clockwise order

Sort  $P$  according to  $x$ -coordinate to generate  
a sequence of points  $p[1], p[2], \dots, p[n]$ ;

# The pseudocode

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

Output: Vertices of  $CH(P)$  in clockwise order

Sort  $P$  according to  $x$ -coordinate to generate  
a sequence of points  $p[1], p[2], \dots, p[n]$ ;  
Insert  $p[1]$  and then  $p[2]$  in a list  $L_U$ ;



# The pseudocode

```
Input: A set P of n points in the plane
Output: Vertices of CH(P) in clockwise order
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 {

}
```

# The pseudocode

```
Input: A set P of n points in the plane
Output: Vertices of CH(P) in clockwise order
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;

}
```

# The pseudocode

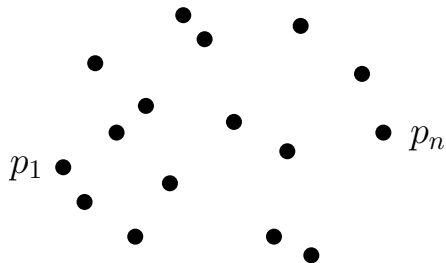
```
Input: A set P of n points in the plane
Output: Vertices of CH(P) in clockwise order
Sort P according to x-coordinate to generate
    a sequence of points p[1], p[2], ..., p[n];
Insert p[1] and then p[2] in a list L_U;
for i = 3 to n {
    Append p[i] to L_U;
    while(L_U contains more than two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

    }
}
```

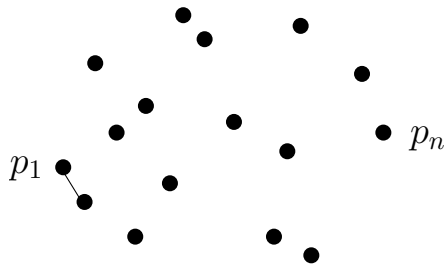
# The pseudocode

```
Input: A set P of n points in the plane
Output: Vertices of CH(P) in clockwise order
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;
    }
}
```

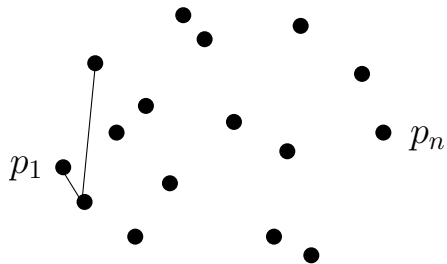
# The Algorithm in Action



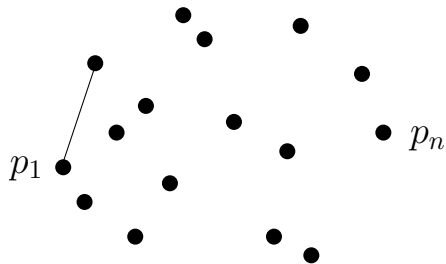
# The Algorithm in Action



# The Algorithm in Action

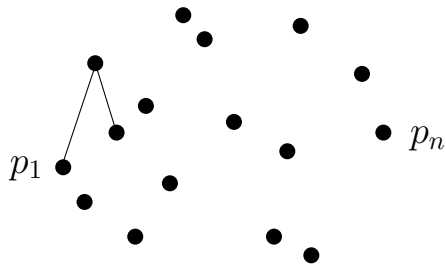


# The Algorithm in Action

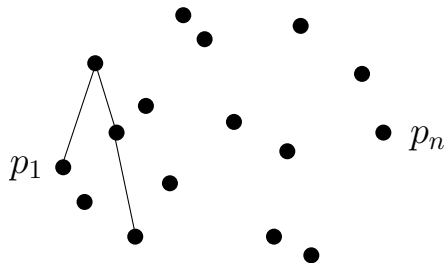




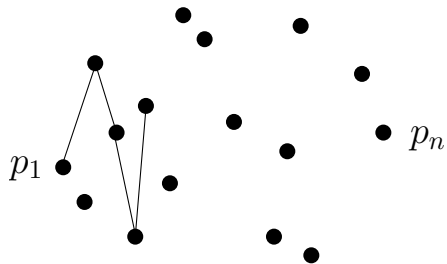
# The Algorithm in Action



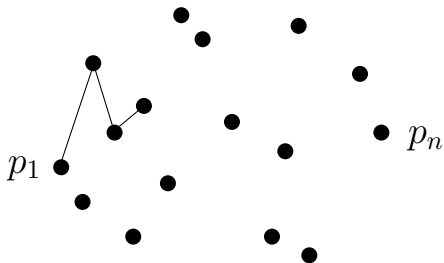
# The Algorithm in Action



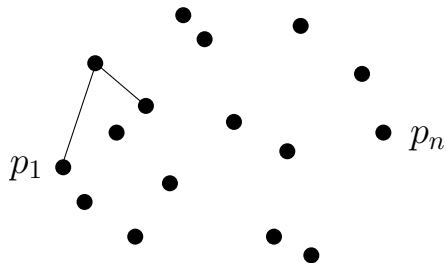
# The Algorithm in Action



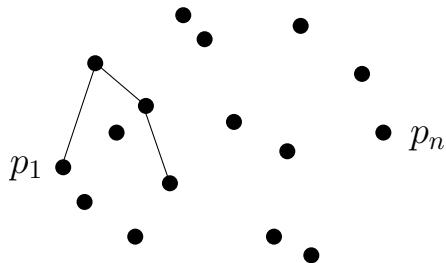
# The Algorithm in Action



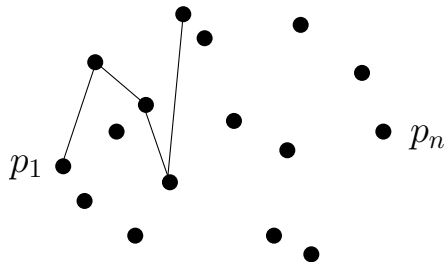
# The Algorithm in Action



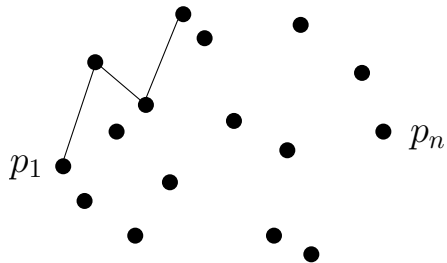
# The Algorithm in Action



# The Algorithm in Action

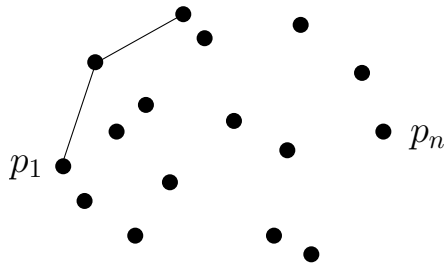


# The Algorithm in Action

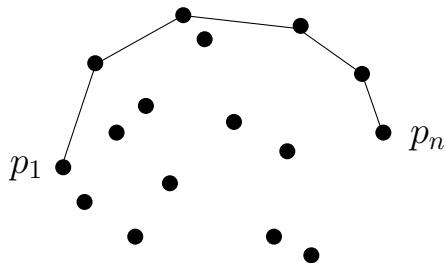




# The Algorithm in Action



# 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.

## 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.

## 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  $O(n)$ .



## Time complexity

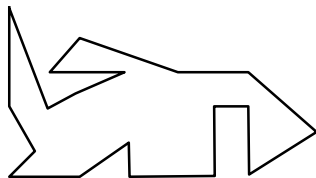
- Sorting takes time  $O(n \log n)$ .
- The for loop is executed  $O(n)$  times.
- For each execution of the for loop, the while loop is encountered once.
- For each execution of the while loop body, a point gets deleted.
- A point once deleted, is never deleted again.
- So, the total number of executions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



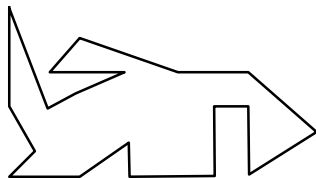
# Art Gallery Problem

## The problem

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

## Hardness

The problem is NP-Hard.



# Art Gallery Problem

## The problem

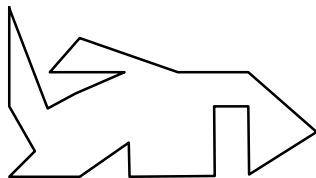
Given a simple polygon  $\mathcal{P}$  of  $n$  vertices, find the minimum number of cameras that can guard  $\mathcal{P}$ .

## Hardness

The problem is NP-Hard.

## Simplified version

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  $\mathcal{P}$ .

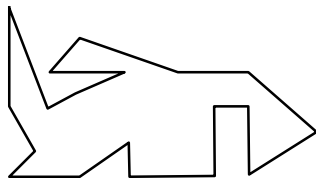
## Hardness

The problem is NP-Hard.

## Simplified version

Can we find, as a function of  $n$ , the number of cameras that suffices to guard  $\mathcal{P}$ ?

## What if $\mathcal{P}$ is convex?



# 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

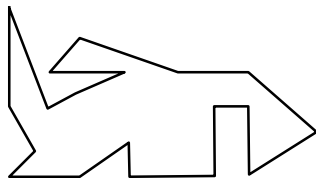
The problem is NP-Hard.

## Simplified version

Can we find, as a function of  $n$ , the number of cameras that suffices to guard  $\mathcal{P}$ ?

## What if $\mathcal{P}$ is convex?

- Only one camera is required.



# Art Gallery Problem

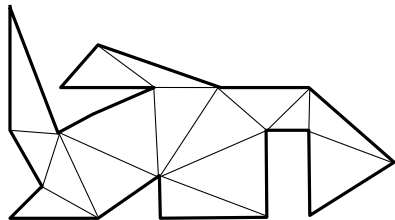
Easy solution



# Art Gallery Problem

## Easy solution

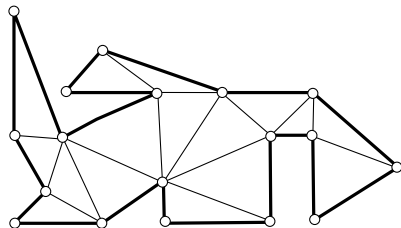
- Recall  $\mathcal{P}$  can be triangulated into  $n - 2$  triangles. Place a guard in each triangle.



# Art Gallery Problem

## Easy solution

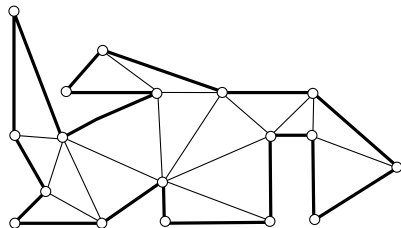
- Recall  $\mathcal{P}$  can be triangulated into  $n - 2$  triangles. Place a guard in each triangle.
- Or place guards at vertices of the triangulation  $\mathcal{T}$  of  $\mathcal{P}$ .



# Art Gallery Problem

## Easy solution

- Recall  $\mathcal{P}$  can be triangulated into  $n - 2$  triangles. Place a guard in each triangle.
- Or place guards at vertices of the triangulation  $\mathcal{T}$  of  $\mathcal{P}$ .
- We get an  $O(n)$  bound on the number of guards.



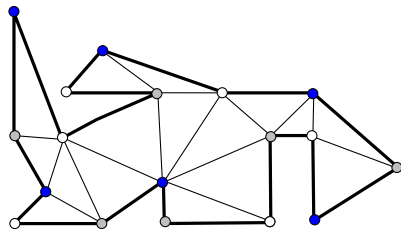
# Art Gallery Problem

Can the bound be reduced?

# Art Gallery Problem

Can the bound be reduced?

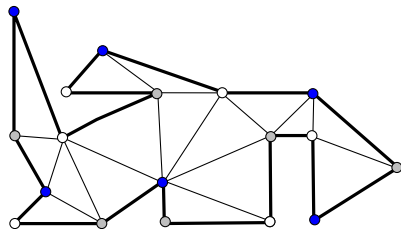
- We do a 3-coloring of the vertices of  $\mathcal{T}$ . Each triangle of  $\mathcal{T}$  has a blue, gray and white vertex.



# Art Gallery Problem

## Can the bound be reduced?

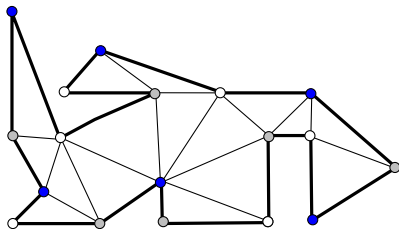
- We do a 3-coloring of the vertices of  $\mathcal{T}$ . Each triangle of  $\mathcal{T}$  has a blue, gray and white vertex.
- Choose the smallest color class to guard  $\mathcal{P}$ .



# Art Gallery Problem

## Can the bound be reduced?

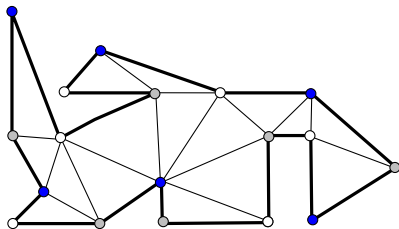
- We do a 3-coloring of the vertices of  $\mathcal{T}$ . Each triangle of  $\mathcal{T}$  has a blue, gray and white vertex.
- Choose the smallest color class to guard  $\mathcal{P}$ .
- Hence,  $\lfloor \frac{n}{3} \rfloor$  guards suffice.



# Art Gallery Problem

## Can the bound be reduced?

- 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

## Theorem

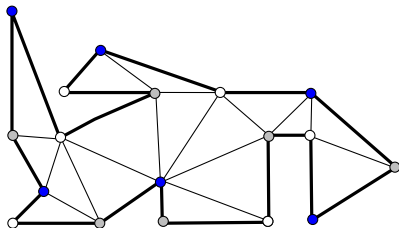
The triangulation graph of a simple polygon  $P$  may be 3-colored.

# Art Gallery Problem

## Theorem

The triangulation graph of a simple polygon  $P$  may be 3-colored.

A 3-coloring always exist



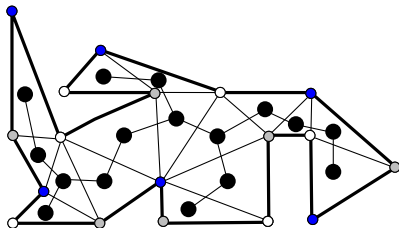
# Art Gallery Problem

## Theorem

The triangulation graph of a simple polygon  $P$  may be 3-colored.

## A 3-coloring always exist

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



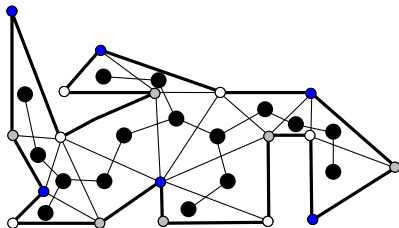
# Art Gallery Problem

## Theorem

The triangulation graph of a simple polygon  $P$  may be 3-colored.

## A 3-coloring always exist

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



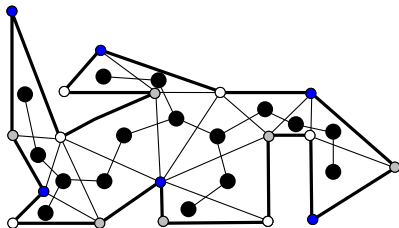
# Art Gallery Problem

## Theorem

The triangulation graph of a simple polygon  $P$  may be 3-colored.

## A 3-coloring always exist

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



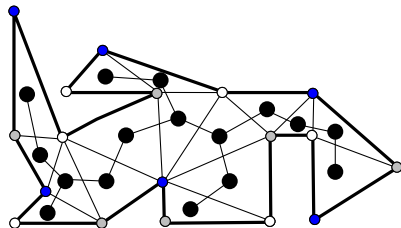
# Art Gallery Problem

## Theorem

The triangulation graph of a simple polygon  $P$  may be 3-colored.

## A 3-coloring always exist

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

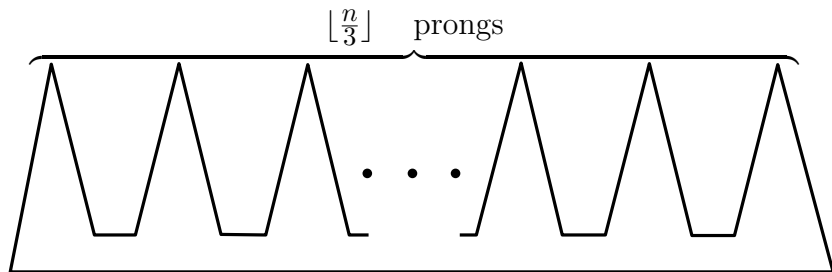
Necessity?

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

# Art Gallery Problem

## 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., 6:485-524, 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](http://en.wikipedia.org/wiki/Computational_geometry)

Thank you!