Introduction to Computational Geometry

Subir Kumar Ghosh
School of Technology & Computer Science
Tata Institute of Fundamental Research
Mumbai 400005, India
ghosh@tifr.res.in

Overview

1. Background
2. Time and space complexity
3. Polygon inclusion problems
4. Intersection problems
5. Convex hull problem
6. Polygon triangulation
7. Ary Gallery problem
8. Concluding remarks

Background

- Computational Geometry is concerned with the computational complexity of geometric problems within the framework of analysis of algorithms.
- Before computational geometry was accepted as a well-defined discipline in 80's, two groups of researchers were working very close to the field.
- The first group, obviously, consists of mathematicians, particularly geometers.
- The second group consists of researchers in computer application areas such as pattern recognition, computer graphics, image processing, VLSI design, computer-aided design and robotics.

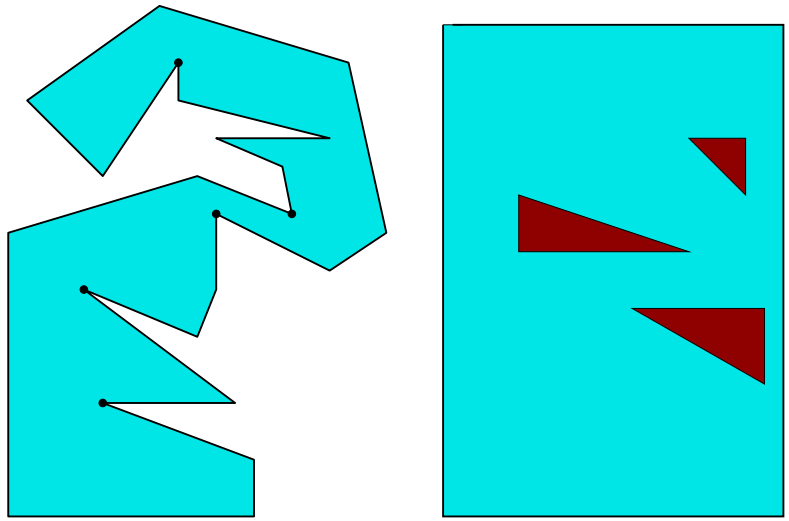
- Though geometric problems form the kernel of investigation to both the groups, their approaches to the problems are entirely different.
- The former group studies geometry with the intention of discovering properties of geometric objects. In contrast to that, the researchers of the latter group attempts to develop geometric algorithms for its application to real world problems.
- These two approaches indeed leave a big gap in studying geometric problems.
- In order to bridge the gap, the discipline "Computational Geometry" had evolved with an aim to extract out and exploit the properties of geometric objects that make an algorithm more efficient.

Time and Space Complexity

- In order to define an efficient algorithm, we need the notation $O(f(n))$ to measure the time and space complexity of the algorithm, where n is the size of the input to the algorithm.
- The notation $O(f(n))$ denotes the set of all functions $g(n)$ such that there exist positive constants c and n_0 with $|g(n)| \leq c|f(n)|$ for all $n \geq n_0$.
- We assume that addition, subtraction, multiplication, division and comparisons on real numbers can be performed in unit time.
- In addition, various other operations such as indirect addressing of memory (integer address only), computing the intersection of two lines, computing the distance between two points, testing whether the angle is convex are also available. These operations are assumed to take constant execution of time.

- Using this "O" notation, let us explain what an efficient algorithm means.
- Consider the problem of sorting of numbers $x_1, x_2, .., x_n$.
- A naive algorithm for sorting takes $O(n^2)$ time.
- Using the technique of divide-and-conquer, sorting can be done in $O(n \log n)$ time and this algorithm can be called efficient since it is faster than the naive algorithm of $O(n^2)$ time.

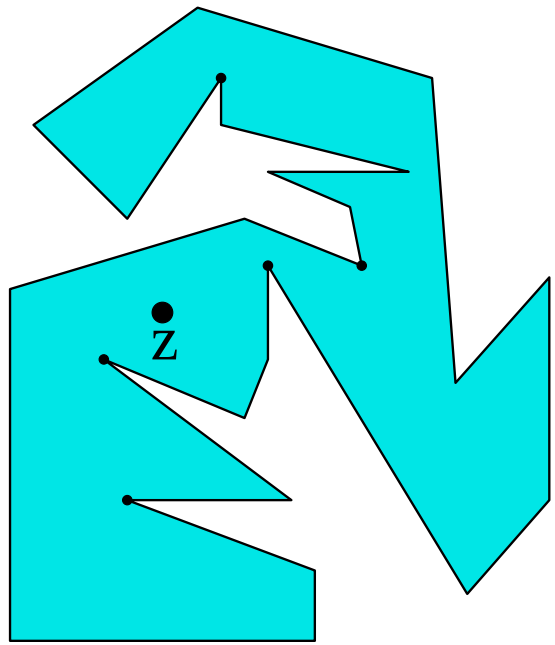
Polygon



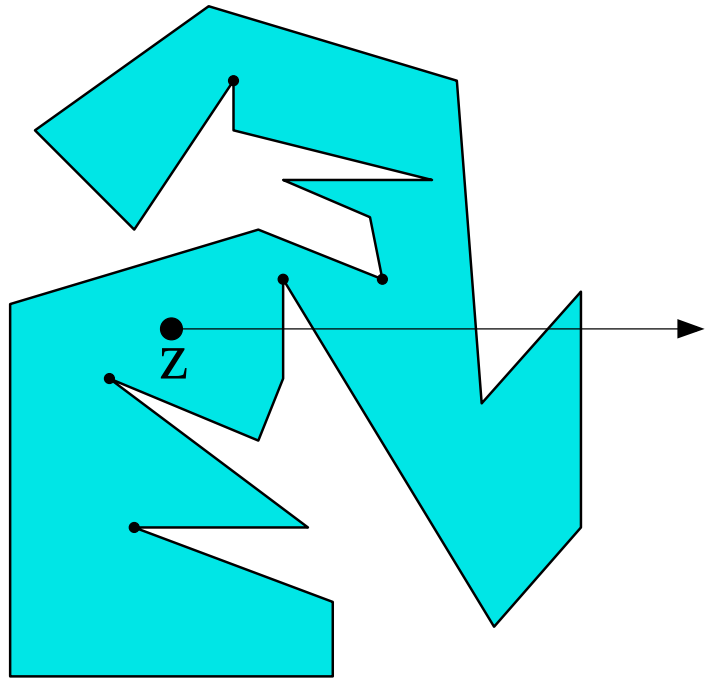
A polygon P is given as a counterclockwise or clockwise sequences of *vertices* with their respective x and y coordinates and no three vertices of P are collinear.

If the boundary of P consists of two or more cycles, then P is called a *polygon with holes*. Otherwise, P is called a *simple polygon* or a *polygon without holes*.

Polygon inclusion problems



Problem: Given a simple polygon P and a point z, determine whether z is internal to P.



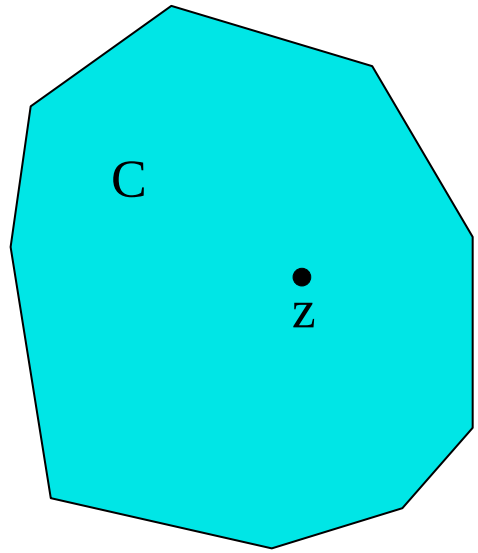
Theorem: Whether a point is internal to a simple polygon of n vertices can be determined in $O(n)$ time.

Polygon query problem

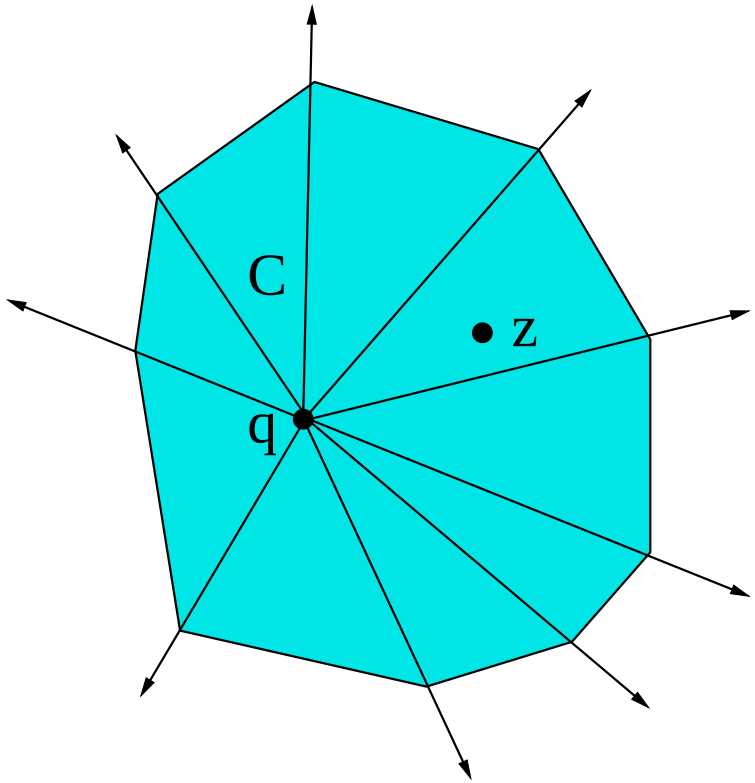
- Suppose we have m points $z_1, z_2, ..., z_m$ and a simple polygon P of n vertices, and we wish to know for all i, whether z_i (called *query point*) is internal to P.
- Using the above method, we can report the answer by spending $O(n)$ time for each query point z_i. Therefore, the total time required is $O(nm)$.
- If m is very large compare to n, this method is certainly inefficient. Can we do better?

Problem: Given a simple polygon P and a query point z, determine whether z is internal to P.

Convex polygon query problem



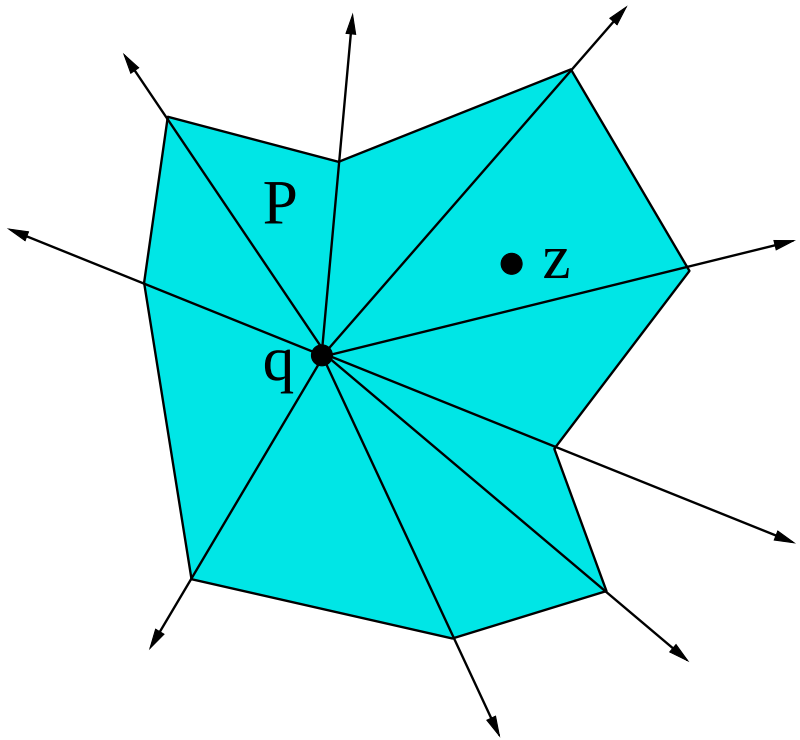
Problem: Given a convex polygon C and a query point z, determine whether z is internal to C.



- Vertices of C are in sorted angular order with respect to any internal point q of P.
- The entire plane can be divided into wedges.
- The wedge containing z can be located by binary search.

Theorem: Whether a query point z is internal to a convex polygon C of n vertices can be determined in $O(\log n)$ query time with $O(n)$ preprocessing time using $O(n)$ space.

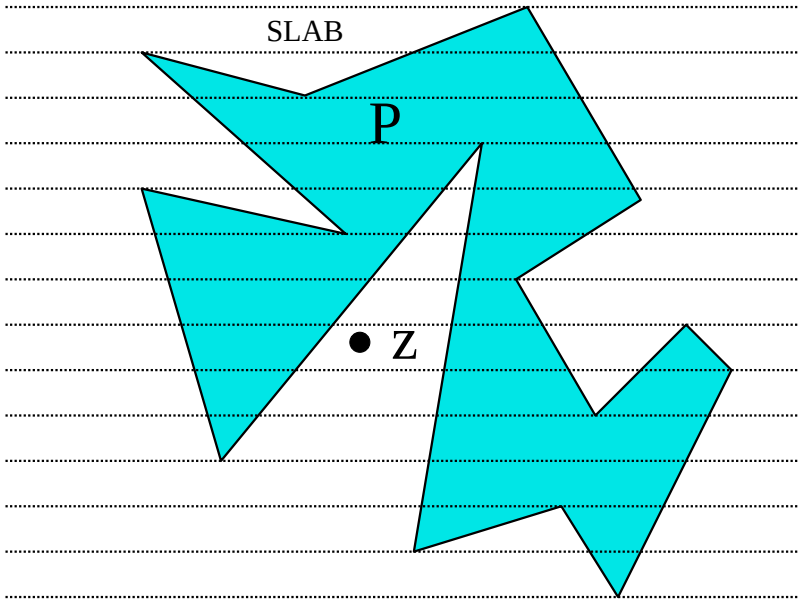
Star-shaped polygon query problem



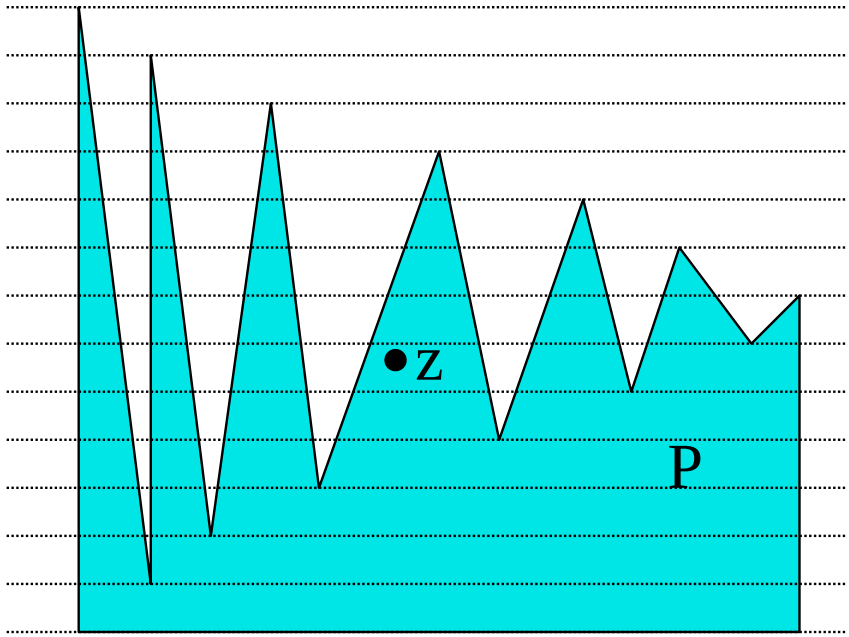
Problem: Given a star-shaped polygon P and a query point z, determine whether z is internal to P.

Theorem: Whether a query point z is internal to a star-shaped polygon P of n vertices can be determined in $O(\log n)$ query time with $O(n)$ preprocessing time using $O(n)$ space.

Simple polygon inclusion problem

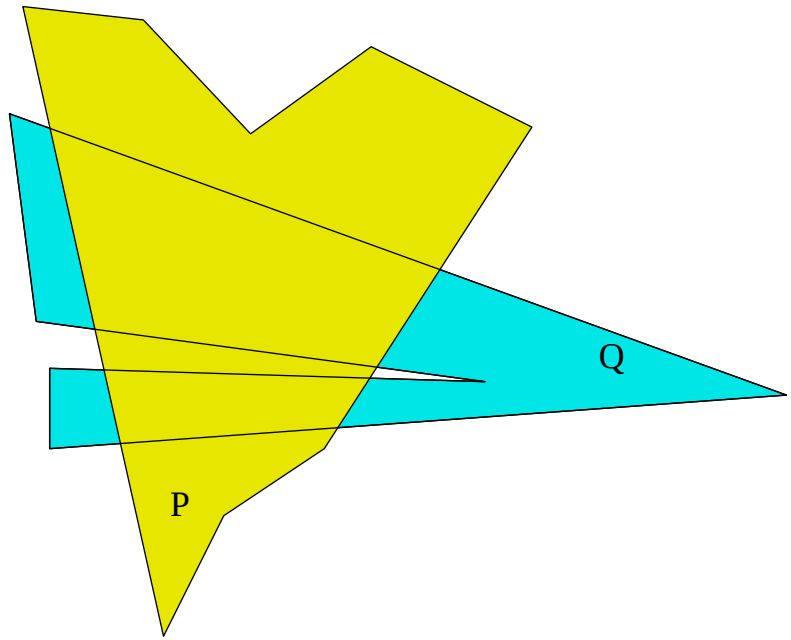


- Draw horizontal lines through each vertex of P dividing the plane into $n + 1$ horizontal strips called "slabs".
- Sort these slabs by y-coordinate as a preprocessing step.
- Perform a binary search to locate the query point z in a slab.
- Within this slab, perform a binary search to locate the query point z in a trapezoid.



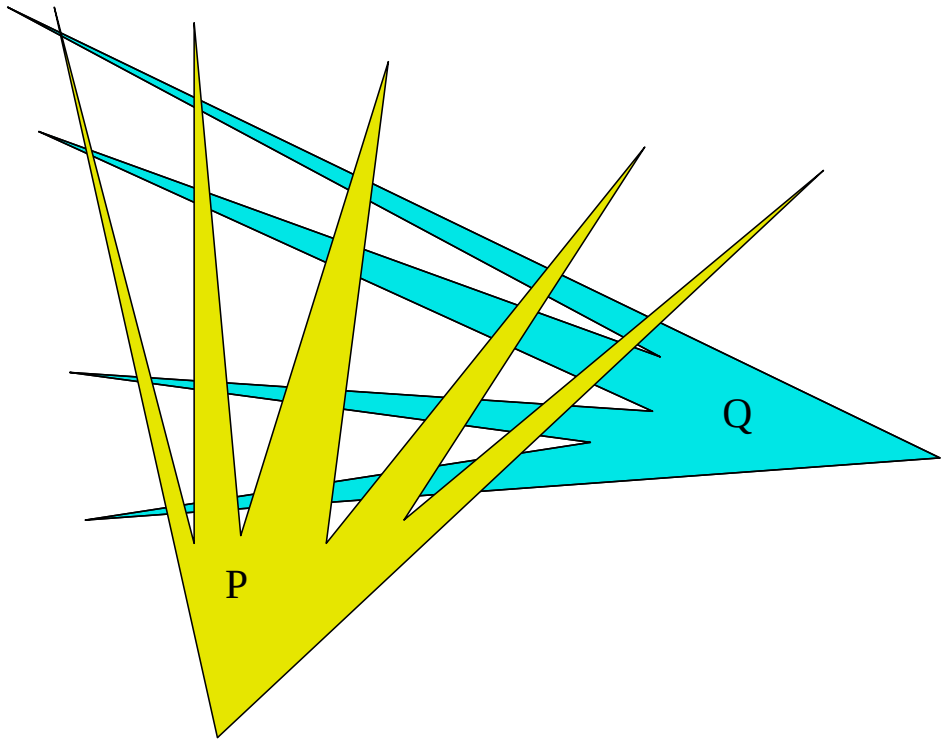
Theorem: Whether a query point z is internal to a simple polygon of n vertices can be determined in $O(\log n)$ time using $O(n^2 \log n)$ preprocessing time and $O(n^2)$ space.

Polygon intersection problem



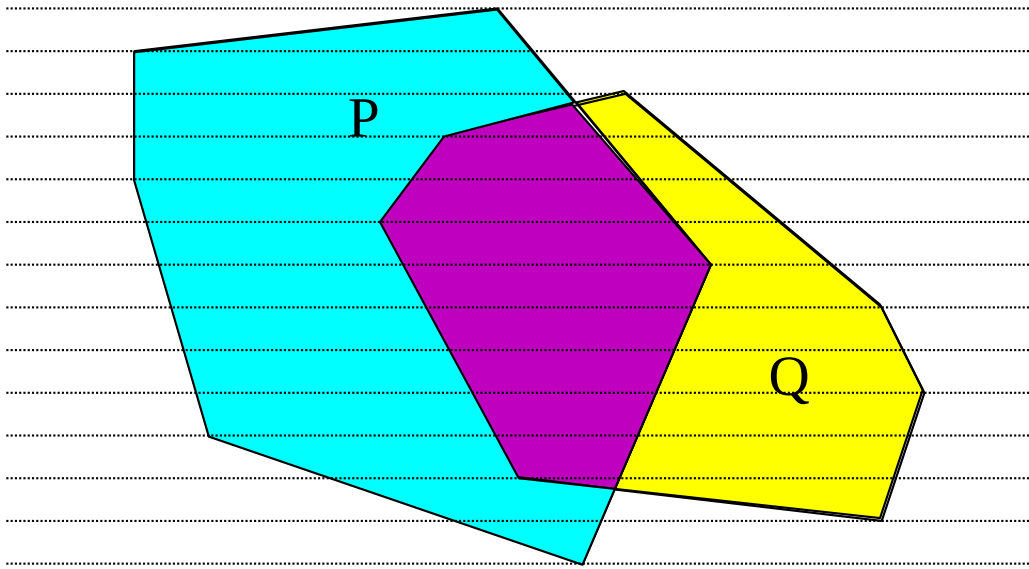
Problem: Given two simple polygons P and Q of n and m vertices, compute their intersection.

Can there be $O(nm)$ intersection points between edges of P and Q?



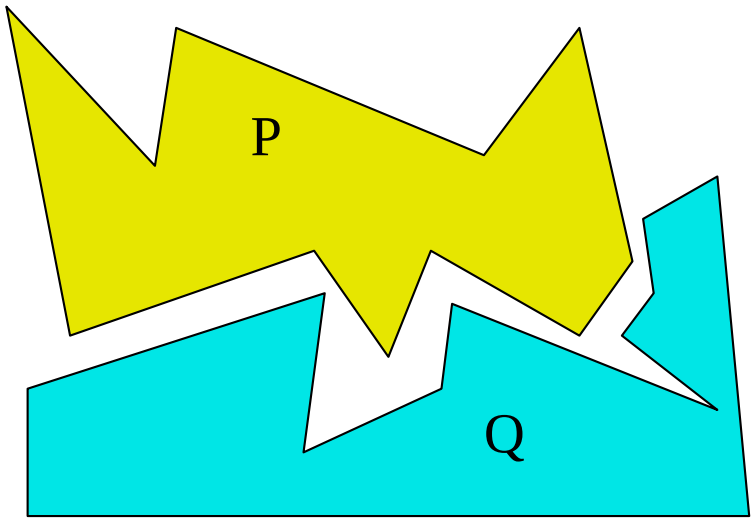
Theorem: Computing the intersection of two simple polygons of n and m vertices requires $O(nm)$ time in the worst case.

Convex polygon intersection problem



Theorem: The intersection of two convex polygons of n and m vertices can be computed in $O(n + m)$ time.

Testing polygons intersection

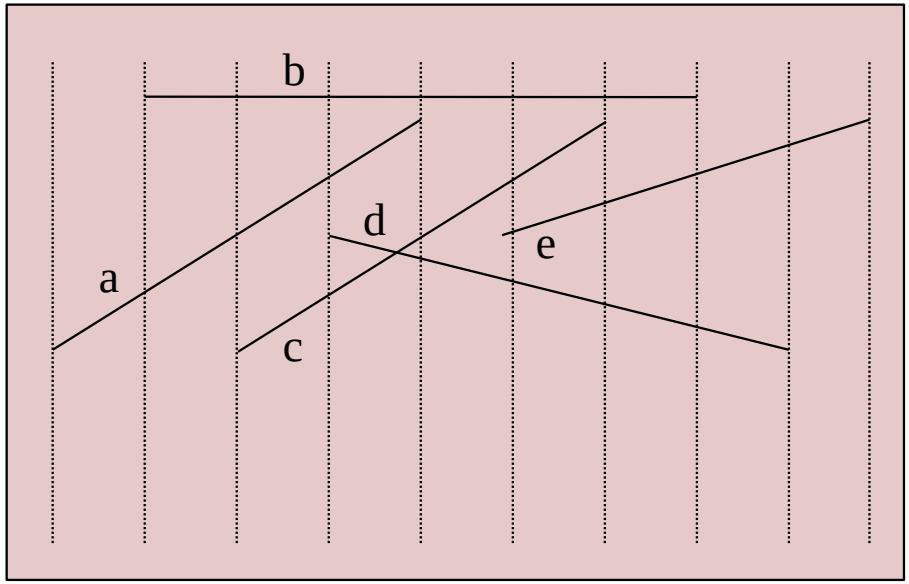


Problem: Given two simple polygons P and Q of n and m vertices, do they intersect?

Problem: Given n line segments in the plane, determine whether any two intersect?

Theorem: Intersection of simple polygons is linear time transformable to line-segment intersection testing.

Plane-sweep technique



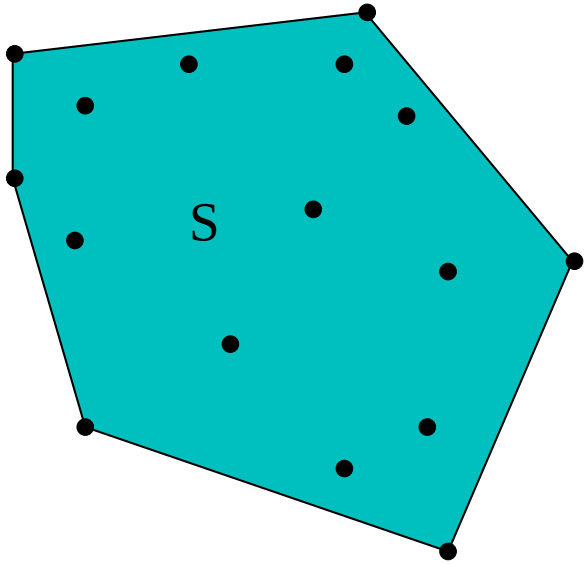
- Sweep a vertical line from left to right.
- *The left endpoint of a segment is encountered.* In this case, the segment is added to the ordering.
- *The right endpoint of a segment is encountered.* In this case, the segment is removed from the ordering.
- *The intersection point of two segments is reached.* Here the segments exchange places in the ordering.

Theorem: Whether any two of n line segments in the plane intersect can be determined in $O(n \log n)$ time, and it is optimal.

Theorem: The problem of detecting the intersection of two simple polygons of n and m vertices can be determined in $O((n+m)\log(n+m)$ time in the worst case.

Theorem: The intersection type of two star-shaped polygons can be determined in linear time.

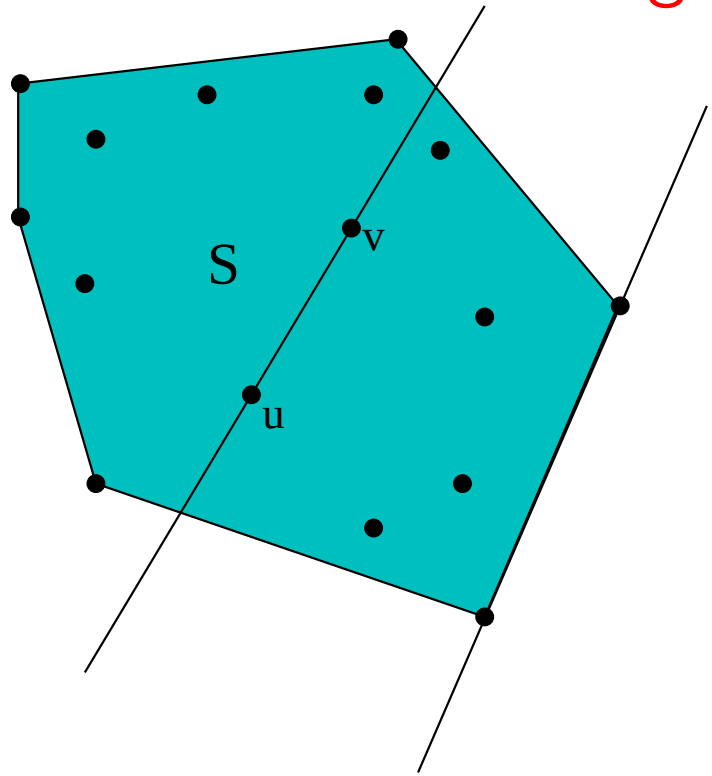
Convex hull



Problem: Given a set of n points in the plane, compute the convex hull of S.

The convex hull of S is the smallest convex set containing all points of S.

A naive algorithm

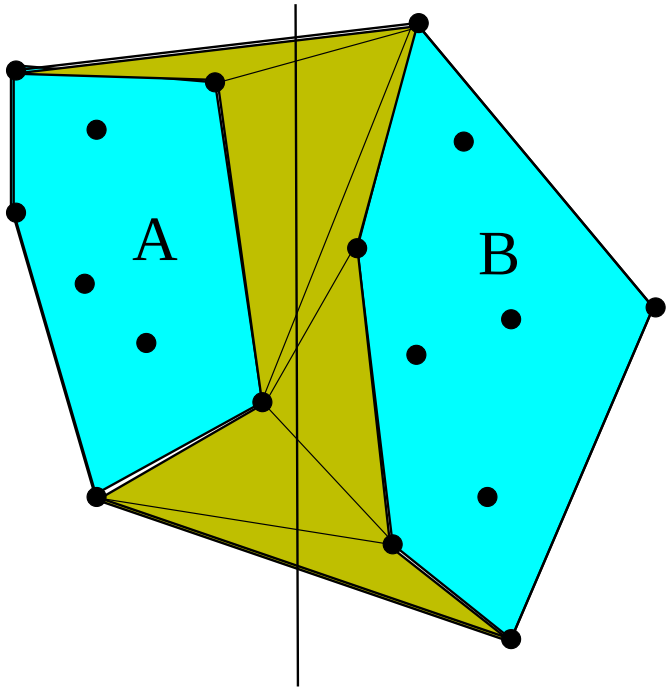


- For every pair of points u and v, draw the line L_{uv} passing through u and v.
- If every point of S lie on one side of L_{uv}, take uv as a convex hull edge.
- Therefore, the convex hull of S can be computed in $O(n^3)$ time.
- Is it possible to compute the convex hull of S in $O(n^2)$ time?

Divide and Conquer technique

- "Divide and Conquer" is a general paradigm for solving problems in computer science.
- The essence is to partition the problem into two (nearly) equal halves.
- Solve the problem for each half recursively.
- Create a full solution by "merging" the two half solutions.
- Let $T(n)$ be the time complexity of a divide and conquer hull algorithm. If merging step can be done in linear time, then $T(n) = T(n/2) + O(n)$. Hence, $T(n) = O(n \log n)$.

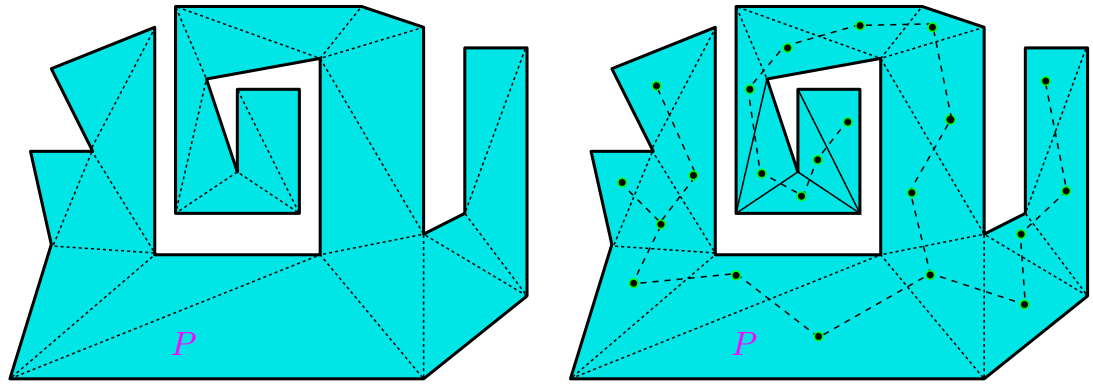
Convex hull algorithm



- Divide S into two sets A and B of nearly equal size.
- Compute the convex hulls of A and B recursively.
- Construct tangents between the convex hulls of A and B.
- Remove the appropriate portions of the convex hulls of A and B to construct the convex hull of S.

Theorem: The convex hull of a set of n points can be computed in $O(n \log n)$ time and it is optimal.

Polygon triangulation



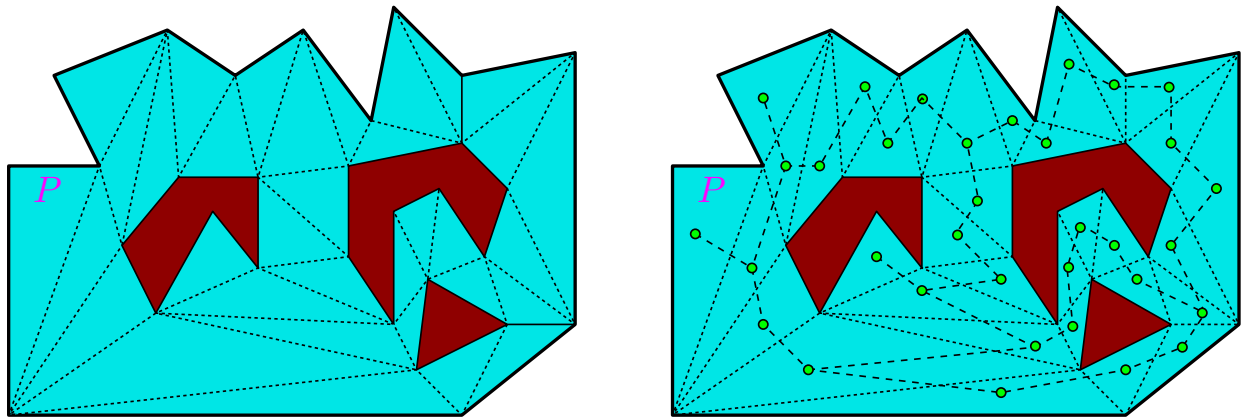
A line segment joining any two mutually visible vertices of a polygon is called a *diagonal* of the polygon.

Lemma: Every triangulation of a simple polygon P of n vertices uses $n-3$ diagonals and has $n-2$ triangles.

Corollary: The sum of the internal angles of a simple polygon of n vertices is $(n-2)\pi$.

Lemma: The dual of a triangulation of P is a tree.

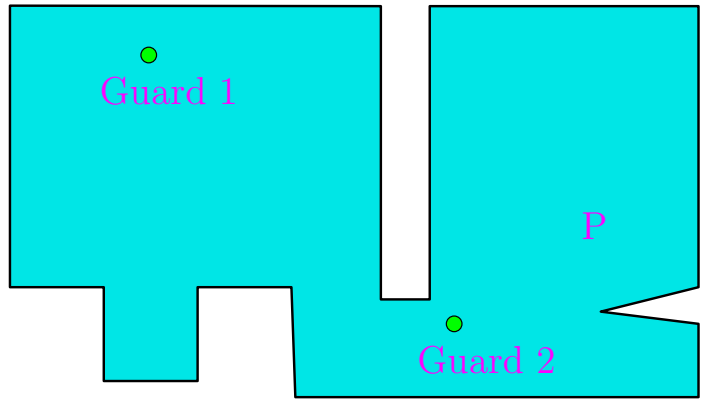
Lemma: Every polygon P has at least two disjoint ears.



Lemma: Every triangulation of a polygon with h holes with a total of n vertices uses $n + 3h - 3$ diagonals and has $n + 2h - 2$ triangles.

Lemma: The dual graph of a triangulation of a polygon with holes must have a cycle.

Art gallery problem



An art gallery can be viewed as a polygon P with or without holes with a total of n vertices and guards as points in P.

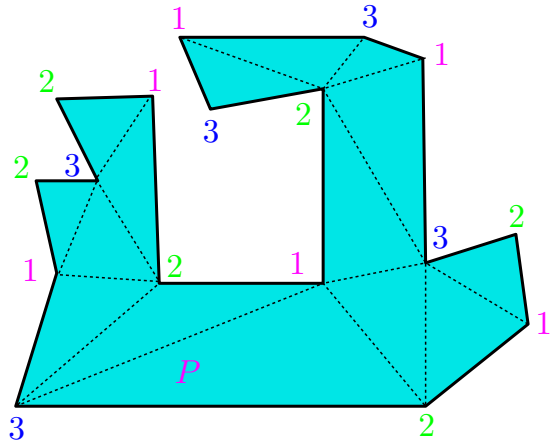
An art gallery can be viewed as a polygon P with or without holes with a total of n vertices and guards as points in P.

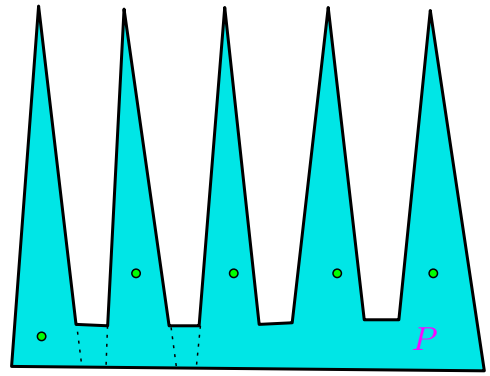
During a conference at Stanford in 1976, Victor Klee asked the following question:

How many guards are always sufficient to guard any polygon with n vertices?

1. R. Honsberger, *Mathematical games II*, Mathematical Associations for America, 1979.

Chvatal's art gallery theorem



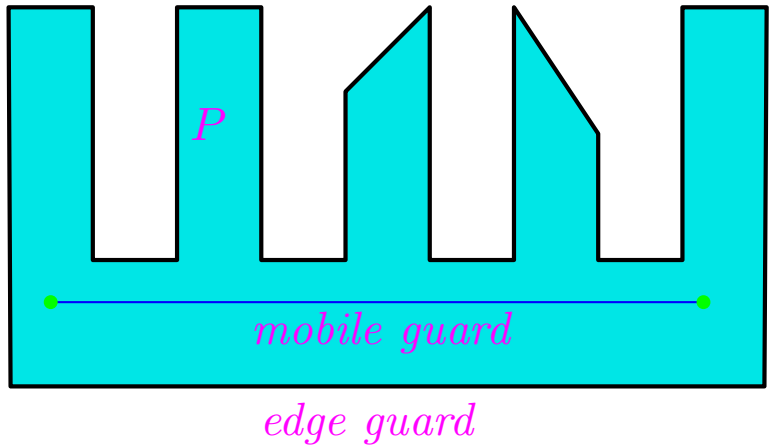


Theorem: A simple polygon P of n vertices needs at most $\lfloor \frac{n}{3} \rfloor$ guards.

Lemma: All vertices of P can be coloured using three colours (say, $\{1, 2, 3\}$) such that two vertices joined by an edge of P or by a diagonal in the triangulation of P receive different colours.

1. V. Chvatal, *A combinatorial theorem in plane geometry*, Journal of Combinatorial Theory, Series B, 18 (1975), 39-41.
2. S. Fisk, *A short proof of Chvatal's watchman theorem*, Journal of Combinatorial Theory, Series B, 24 (1978), 374.

Different types of guards



- *Point guards*: These are guards that are placed anywhere in the polygon.
- *Vertex guards*: These are guards that are placed on vertices of the polygon.
- *Edge guards*: These are guards that are allowed to patrol along an edge of the polygon.
- *Mobile guards*: These are guards that are allowed to patrol along a segment lying inside a polygon.

Minimum number of guards

Let P be a polygon with or without holes. What is the minimum number of guards required for guarding a polygon P with or without holes?

Suppose, a positive integer k is given. Can it be decided in polynomial time whether k guards are sufficient to guard P?

The problem is NP-complete.

Theorem: The minimum vertex, point and edge guard problems for polygons with or without holes (including orthogonal polygons) are NP-hard.

Theorem: The minimum vertex and point guard problems for orthogonal polygons with or without holes are NP-hard.

Further Reading

1. M. de Berg, M. Van Kreveld, M. Overmars and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer, 1997.
2. S.K. Ghosh, *Visibility Algorithms in the Plane*, Cambridge University Press, Cambridge, UK, 2007.
3. J. O'Rourke, *Art gallery theorems and algorithms*, Oxford University Press, 1987.
4. J. O'Rourke, *Computational Geometry in C*, Cambridge University Press, New York, USA, 1998.
5. F. P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, USA, 1990.

Concluding remarks

- In this talk, we have presented a few basic algorithms and techniques of computational geometry which were designed in the late 70's and early 80's.
- In the last three decades, several algorithms have been designed for many interesting geometry problems in two and higher dimensions.
- The main impetus for the development of geometric algorithms came from the progress in computer graphics, computer-aided design and manufacturing.
- In addition, algorithms are also designed for geometric problems that are classical in nature.
- The success of the field can be explained from the beauty of the geometry problems studied, the solutions obtained, and by the many application domains- computer graphics, geographic information systems, robotics and others, in which geometric algorithms play a crucial role.

Thank You.