Chapter 8

Computational Geometry

Computational Geometry

- A branch of algorithmics dealing with computation on geometric objects
- Algorithmics is the branch of computer science that consists of designing and analyzing computer algorithms
- It Involves
  - its description at an abstract level using a pseudo-language, and
  - the proof that the algorithm is correct
- Analysis deals with evaluating the performance of an algorithm; for example, its cost in time, a task called complexity analysis
- An algorithm is a sequence of operations performed on data that have to be organized in data structures.
- A careful design of data structures is required for optimizing the operations for a given algorithm

Basic Concepts of Algorithms

- 3 Cornerstones of algorithm
  i. the model of computation
  ii. the language used for describing the algorithm
  iii. the performance evaluation
- The algorithm is an abstraction of a program to be executed on a physical machine
- An abstraction, or model of computation, allows one to describe what type of information a machine can handle, as well as the operations available.
- Once the model of computation has been defined, an algorithm can be described using the primitive operations
- The performance evaluation of an algorithm is obtained by totaling the number of occurrences of each operation when running the algorithm
- For the sake of simplicity and ease of expression, one gets rid of unnecessary details in the computation
- Algorithm evaluation is a function f (n,c1,c2,...,cm), whose arguments are (1) the size n of the input and (2) the cost ci of each key operation

Basic Concepts of Algorithms

- The focus is on the dependence of computation time on input size rather than on the cost of each operation, which is implementation and machine dependent.
- Upper bound
  - A function g(n) of the input size is said to be O(f (n)) if there exist a constant C and an integer n0 such that $g(n) \leq Cf(n)$ for all $n > n0$.
- Lower bound
  - A function g(n) is said to be Ωf (n)) if there exist C and n0 such that $g(n) \geq Cf(n)$ for all $n > n0$.
- Same order
  - A function g(n) is said to be Θ(f (n)) if there exist C1, C2, and n0 such that $C1f(n) \leq g(n) \leq C2f(n)$ for all $n > n0$.

Algorithm Analysis

- The complexity of an algorithm is a function g(n) that gives the upper bound of the number of operations performed by an algorithm when the input size is n.
- The number of operations is also called algorithm “running time.”
- Worst-case complexity
  - In this case, the running time for any input of a given size will be lower than the upper bound, except possibly for some values of the input where the maximum is reached.
- Average-case complexity.
  - In this case, g(n) is the average number of operations over all problem instances for a given size.
- The average-case complexity is more representative of the actual behavior of an algorithm.
- Unfortunately, it is quite difficult to estimate, as the statistical behavior of the input is not easily predictable
- Most of the analyses give, therefore, a worst-case complexity.

Algorithm Analysis

- Even though this approximation provides a useful classification of algorithms, it leads to a loose estimate of the actual complexity because it does not take into account the multiplicative constant C and the fact that the complexity is true for n > n0.
- Thus, two algorithms with the same asymptotic complexity may lead to quite different computation times

Optimality

- An algorithm for a given problem is optimal if its complexity reaches the lower bound over all the algorithms solving this problem
- estimating a lower bound is an essential task, and unfortunately a difficult one

USEFUL ALGORITHMIC STRATEGIES

Incremental Algorithms: The Convex-Hull Example

- An incremental algorithm uses a straightforward strategy that works as follows.
- The idea is to first take a subset of the input small enough so that the problem is easily solved, and then to add, one by one, the remaining elements of the input while maintaining the solution at each step.

Divide-and-Conquer Strategy: The Half-Plane Intersection Example

- The divide-and-conquer strategy has for a long time proved effective in many situations.
- It relies on a recursive approach and has two steps.
- In the first step (top-down), the input is recursively divided until the subproblems are small enough in size to be solved easily.
- The second step (bottom-up) consists of recursively merging the solutions.
- This strategy is also used in sorting algorithms such as merge-sort and quicksort.
- **Building the structure**

  The set of n half-planes in the input is recursively halved until one obtains n singleton half-planes. This yields a binary tree

- **Solving an atomic problem.**

  The half-plane in each leaf singleton is intersected with rectangle R. This yields a convex polygon in each leaf.

- **Merging the results.**

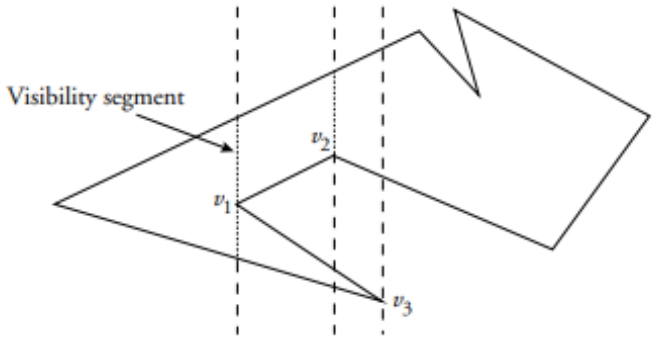
  Compute recursively the intersection of the convex polygons (going up in the structure defined in the first item in this list.)

Sweep-Line Method: The Rectangle Intersection Example

- The sweep-line (or plane-sweep) technique is intensively used in computational geometry.
- It consists of decomposing the geometric input into vertical strips, such that the information relevant to the problem is located on the vertical lines that separate two strips.
- Therefore, by “sweeping” a vertical line from left to right, stopping at the strip boundaries, we can maintain and update the information needed for solving a given problem.
- A structure called the state of the sweep or sweep-line active list, denoted in the following L. This structure is updated as the line goes through a finite number of positions, called events.
- A structure called event list and denoted in the following ε, which can be known beforehand or discovered step-by-step as the sweep goes on.
- In the first case, a chained list is used to store the information, whereas in the second case, another structure that supports searching, inserting, and possibly deleting is necessary (e.g., a priority queue).

POLYGON PARTITIONING

- Polygon decomposition, an important issue of computational geometry, is central to spatial databases because polygons are the most complex objects to be dealt with in 2D applications.
- Then, partitioning a polygon in simpler elements often simplifies algorithm design and implementation.

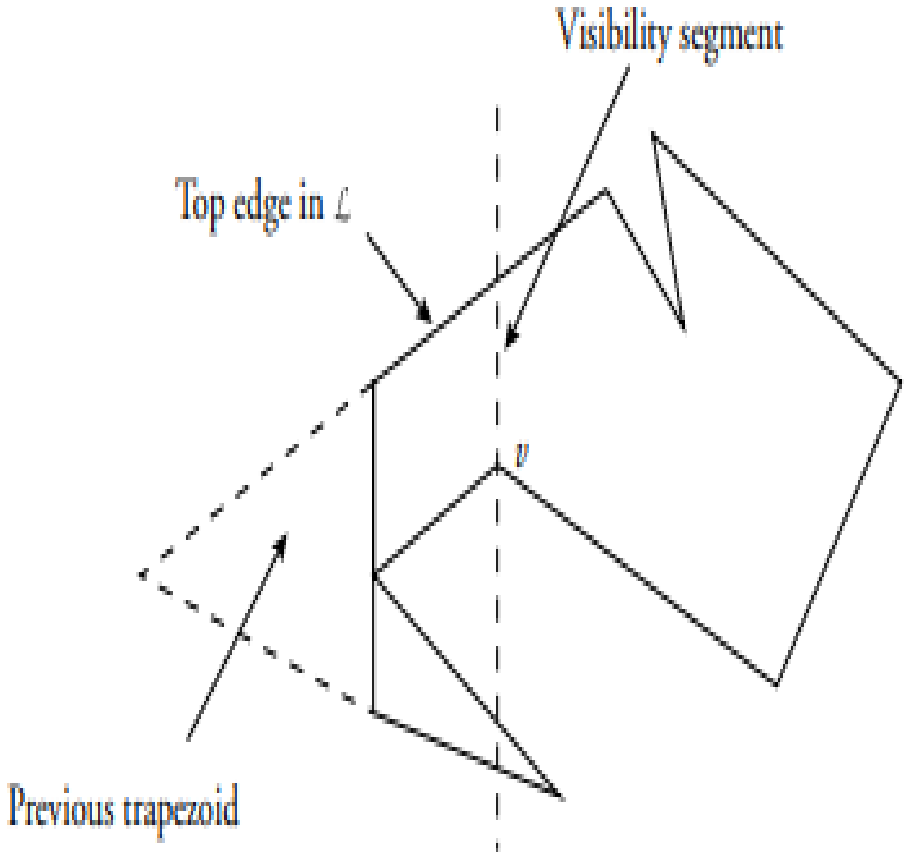


Trapezoidalization of a Simple Polygon

- A trapezoid is a quadrilateral with at least two parallel edges.
- A triangle can be viewed as a degenerated trapezoid with a null-length edge.
- The trapezoidalization algorithm relies on the sweep-line technique.
- First, the vertices of polygon P are sorted according to their x coordinate.
- Then a vertical line l scans the sorted vertices.
- For each vertex v, we compute the maximal vertical segment on l , internal to P and containing v

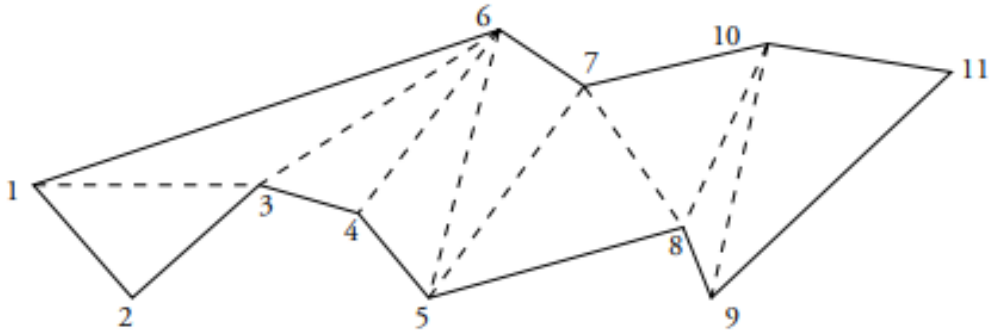
POLYGON PARTITIONING

- The visibility segments define the trapezoidalization.
- The polygon is decomposed into trapezoids with vertical parallel edges (visibility segments)



Triangulation of Simple Polygons

- Triangulating a polygon P from scratch consists of finding diagonals within this polygon.
- A diagonal is a segment ViVj between two vertices of P whose interior does not intersect the boundary of P. Vi and Vj are said to be visible from each other.
- The triangulation is not deterministic, but one can show that every triangulation of a polygon P of n vertices has n − 3 diagonals and results in n − 2 triangles.
- Algorithms that directly triangulate a simple polygon but they are rather slow.
- Monotone polygons can be linearly triangulated, and thus that partitioning a simple polygon into monotone polygons is the key to efficient triangulation algorithms.

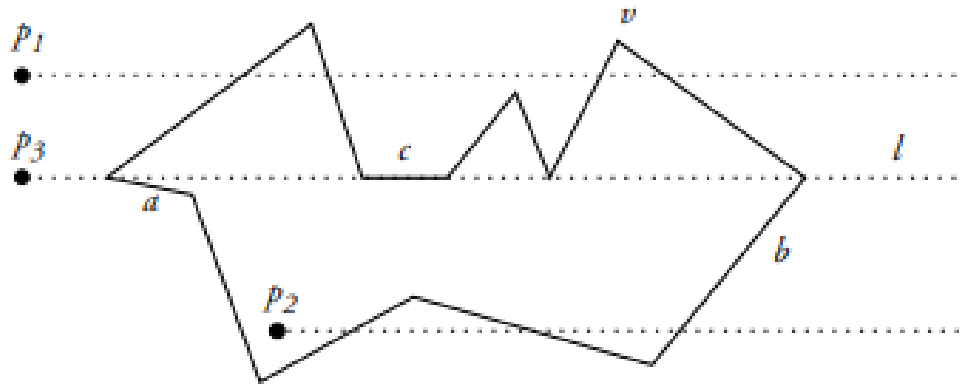


ALGORITHMS FOR SPATIAL DATABASES

I. **Area Size of a Polygon and Related Operations**

II. **Point in Polygon**

III. **Polyline Intersections**



Point in Polygon