

3. Computational Geometry

Overview

Learning Objectives

- Review the basic concepts of algorithms
- Understand some useful algorithmic strategies (e.g., the sweep-line method)
- Learn some algorithms for spatial databases (e.g., point in polygon, polygon intersection, ...)

Literature

- [RSV02] Chapter 5 and/or
[BKO00] Chapters 1 and 2
- For many of the basic principles and algorithms presented in the following, you can find very useful resources on the Web that often go beyond what is presented in the following.

Background

Design of algorithms involves

1. its *description at an abstract level* by means of a *pseudo-code*, and
2. the *proof* that the algorithm is *correct*.

Analysis deals with the performance of an algorithm, such as its cost in time or storage space, known as *complexity analysis*.

The data to be processed by an algorithm is organized in *data structures*.

Some notions:

Random Access Machine

Algorithm evaluation is a function $f(n, c_1, c_2, \dots, c_m)$

Upper bound: A function $g(n)$ of the input size is said to be $O(f(n))$ if there exists a constant c and integer n_0 such that $g(n) \leq c * f(n) \forall n > n_0$

Lower bound: A function $g(n)$ is said to be $\Omega(f(n))$ if there exists a c and n_0 such that $g(n) \geq c * f(n) \forall n > n_0$

Background (2)

The complexity of an algorithm is a function $g(n)$ that gives the upper bound of the number of operations performed by algorithm when the input size is n .

- *Worst-case complexity*: the running time for any input of a given size will be lower than the upper bound.
- *Average-case complexity*: $g(n)$ is the average number of operations over all problem instances for a given size.


Often, the complexity $g(n)$ is approximated by its family $O(f(n))$ where $f(n)$ is n , $\log n$, n^a $a \geq 2$, a^n

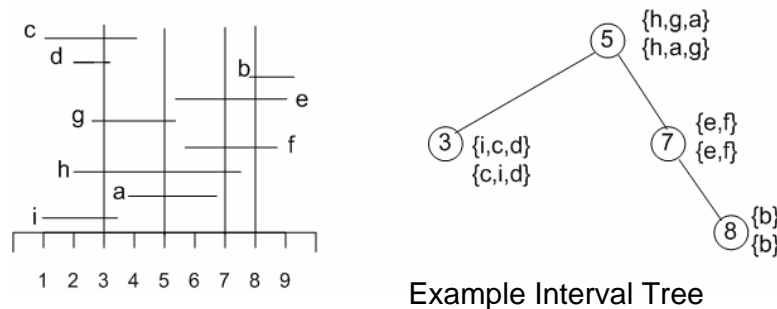
An algorithm for a given problem is *optimal* if its complexity reaches the lower bound over all the algorithms that solve the problem. Estimating lower bound is an essential yet often very difficult task.

Background (3)

A fundamental way of obtaining efficient algorithms is to organize the data with *data structures* that are capable of supporting basic operations on the structures (search, insert, delete) in sublinear time.

For example, self-adjusting, height balanced search trees (e.g., AVL trees, 2-3-4 trees) allow for operations in $O(\log n)$.

A useful structure for many geometric problems is the *interval tree*, which supports efficient *stabbing queries*. 



Algorithmic Strategies

Most of the algorithms rely on well-known general algorithmic strategies:

Incremental algorithms:

Start with a small input so that problem can be solved easily. Then add other elements step by step while maintaining the solution at each step.

Example: Convex Hull (naïve versus optimal)

Divide-and-Conquer strategy:

Relies on a recursive approach and has two steps:

- (1) input is recursively divided into manageable subproblems,
- (2) recursively merge the solutions bottom-up

Example: reporting the intersection of a set of half-planes

Sweep-line (or plane-sweep) technique.

Sweep-Line Method

Basic idea:

- decompose input into vertical strips such that the information relevant to the problem is located on the vertical lines that separate two strips.
- By sweeping a vertical line from left to right, stopping at the strip boundaries, one can maintain the information needed for solving the problem.

Algorithm may vary widely, based on what the geometric input is (sweep-line of planar curves, line segments, rectangles etc)

Commonality among algorithms is the use of two data structures:

- *Sweep-line active list L*: this structure is updated as the line goes through a finite number of positions, called *events*.
- *Event list E*: known beforehand (managed as a chained list) or discovered step-by-step as sweep goes on (managed by a more flexible structure, e.g., priority queue).

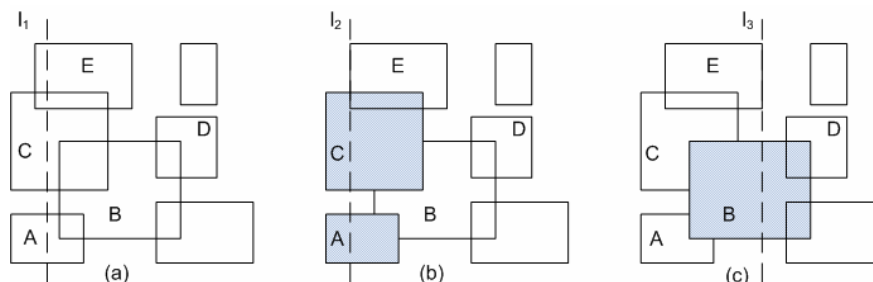
Sweep-Line Method (2)

Example: Given a set of rectangles S . Find all pairwise intersections.

Given the worst case complexity, an *output sensitive algorithm* is worthwhile.

Key observations:

- given a vertical line l and the set S_l of all rectangles cut by l , reporting pairwise intersections in S_l is reduced to testing the overlap of the projection of rectangles on the y axis. E.g., $S_l = \{A, C, E\}$ in Fig. (a)
- When sweeping l from left to right, content of S_l changes at a finite number of locations of l , called *event points*, namely when l reaches the left or right vertical edges of each rectangle in S .



Sweep-Line Method (3)

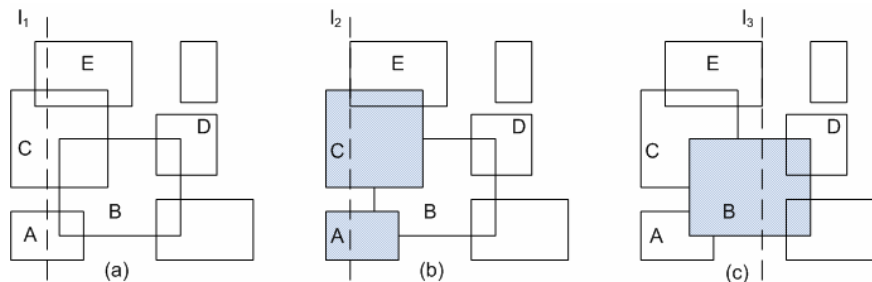
Event points are the $2 \times |S|$ left and right x coordinates, ordered in E .

A data structure L is maintained for the list of active rectangles that intersect with l .

Operations at each event point:

- When left edge of rectangle $r \in E$ is reached by l , insert r into L .
 r 's x interval intersects with those currently in L . Hence, the problem is reduced to searching for those rectangles in L that overlap in the y dimension.
- When right edge of rectangle $r \in E$ is reached by l , r does not overlap with any of the remaining rectangles in E .

Both cases are illustrated in Fig. (b) and (c).



Sweep-Line Method (4)

Algorithm *RectangleIntersection* (S : set of rectangles)

begin

sort the $2n$ x_{\min} and x_{\max} values of S and place them in E

$L = \{\}$

while ($E \neq \{\}$) **do**

begin

$p \leftarrow \min(E)$ // extract the next event from E

if (p is the left edge of a rectangle r) **then**

begin

insert (r, L)

search and report the rectangles in L intersecting r

end if

if (p is the right edge of a rectangle r) **then** delete (r, L)

end while

end

Complexity: $O(n \log n + k)$, with k being the number of intersections

Polygon Partitioning

Polygon decomposition is central to spatial databases, because polygons are the most complex 2-D objects. Polygon partitioning into simpler elements often simplifies subsequent algorithm design and implementation.

Stronger constraints on polygon types imply easier algorithms:

Dealing with convex polygons is simpler than dealing with monotone polygons, which, in turn, are simpler to deal with than simple polygons.

In the following, we only outline key ideas of different strategies and algorithms. For some more detailed discussions, see, e.g.,

<http://www.personal.kent.edu/~rmuhamma/Compgeomtry/MyCG/PolyPart/polyPartition.htm>

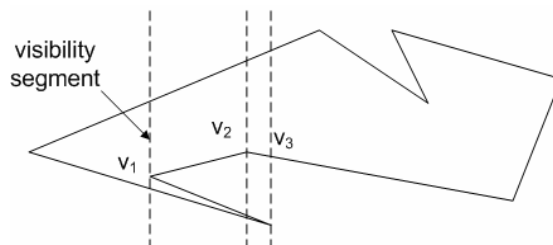
[BKO00] Chapter 3


Polygon Partitioning (2)

Trapezoidalization of a simple polygon P (recall that a *trapezoid* is a quadrilateral with at least two parallel edges).

Trapezoidalization relies on sweep-line technique in which vertices of P are sorted by their x coordinates. For each vertex v_i intersecting with l , compute maximal segment (called *visibility segments*) of l that is internal to P and contains v_i .

The visibility segments define trapezoidalization, which can be computed in $O(n \log n)$.

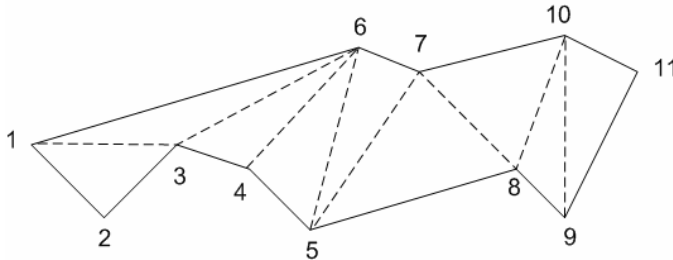


By cutting non-degenerated trapezoids, a triangulation can be obtained. 

Polygon Partitioning (3)

Triangulation of a simple polygon consists of finding diagonals within the polygon.

- Triangulation is non-deterministic
- Every triangulation of polygon P with n vertices has $n-3$ diagonals and results in $n-2$ triangles.
- Monotone polygons can be triangulated linearly, thus partitioning a simple polygon into monotone polygons is key to efficient triangulation. Such a partitioning (similar to the trapezoidalization) can be done in $O(n \log n)$.
- Subsequent triangulation can be done in linear time.
- There exists a linear algorithm for triangulating simple polygons.



Some basics of 2-D Lines

Given two points $p_1(x_1, y_1)$ and $p_2(x_2, y_2)$ in a 2-dimensional space, derive the explicit/implicit equation for the line.

Given a point $p_1(x_1, y_1)$ and a line, what is the distance between the point and the line?

