

# Geometric data structures

Sudebkumar Prasant Pal

Department of Computer Science and Engineering  
IIT Kharagpur, 721302.

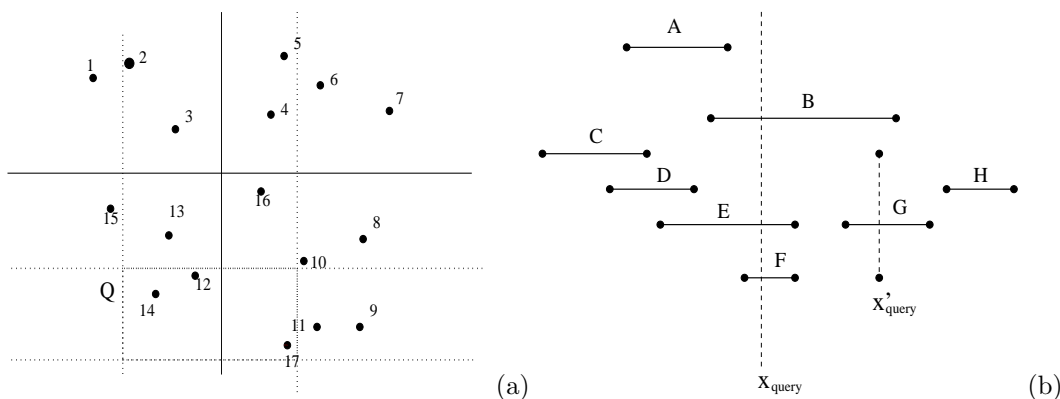
## 1 Introduction

We present a few important data structures for fundamental geometric problems that arise in several applications. The subject matter is vast with developments occurring over several decades. We concentrate only on a small and simple set of data structures in order to introduce a few important and fundamental techniques. The common feature in the problems considered here is the *output-sensitivity* of the time required to retrieve data. The time complexity is partly proportional to the size of the reported output and partly proportional to a sub-linear function of the size of the total database.

## 2 Orthogonal range queries: Kd-trees and range search trees

We begin with *orthogonal range queries*, also called *rectangular queries*. Such queries are common in databases, computer graphics, and in VLSI design. Here, a set  $S$  of  $n$  points is given off-line. We preprocess  $S$  to create some data structure, so that arbitrary 2-dimensional (2-d) range queries can be answered subsequently. The notion of preprocessing is relevant when all the data points are given apriori as the set  $S$ . The processing of each query rectangle  $Q[a, b] \times [c, d]$ , is expected to scan only a subset of  $S$ , necessarily, the points that would appear inside the query rectangle  $Q$ .

We may use Kd-trees for 2-dimensional range searching (see [1]). Given a set  $S$  of  $n$  points in the plane, report points inside a query rectangle  $Q$  whose sides are parallel to the axes. In Figure 1(a), the points inside  $Q$  are 14, 12 and 17. We construct a Kd-tree in  $O(n \log n)$  time using  $O(n)$  space, so that orthogonal range queries may be answered in  $O(\sqrt{n} + k)$  time. Here,  $k$  is the number of points of  $S$  inside the query rectangle



Q.

Using two 1-d range queries, one along each axis, one can answer 2-d range queries by selecting points common to the outputs of the two queries. The cost incurred may however exceed the actual (much smaller) output size of the 2-d range query. The query time can be reduced to  $O(\log^2 n + k)$  from the above  $O(\sqrt{n} + k)$  time, if the use of  $O(n \log n)$  space is permitted; this is possible using *range trees* [1]. The query time can be further reduced to  $O(\log n + k)$  by using the technique of *fractional cascading* for the *range tree* data structure [1], keeping the space requirement within  $O(n \log n)$ .

### 3 The windowing problem and the interval tree data structure

The window clipping problem in computer graphics requires displaying those parts of line segments which lie inside an arbitrary window. The set of line segments need to be preprocessed, so that arbitrary window queries can be answered by reporting portions of such clipped segments. We can use 2-d rectangular range queries as in the previous section, for reporting segments with at least one endpoint inside the query window rectangle. However, segments that cut across the window (with their endpoints outside the window), cannot be determined in this manner. For such segments we need to solve the following problem: given a set  $S$  of  $n$  horizontal intervals, we need to report those which intersect the vertical segment joining points  $(x'_{query}, y_{min})$  and  $(x'_{query}, y_{max})$  (see Figure 1(b)). A simpler problem is to consider the infinite vertical line at abscissa  $x_{query}$  (Figure 1(b)). We discuss the solution to these problem using the data structure called *interval trees* as in [1]. The preprocessing time required is  $O(n \log n)$ , and the query time is  $O(\log n + k)$ .

### 4 Plane sweep and angular sweep algorithms

The plane sweep technique helps in reporting all intersections between  $n$  given segments in the plane. The main idea is to sweep a vertical line from left to right, taking steps at endpoints of each of the  $n$  segments; the algorithm discovers all intersections and takes a step also at each intersection point. Some actions are taken to maintain the data structures at each step so that the reporting of all intersections to the left of the sweep line is ensured. Using simple data structures, all the  $k$  intersection can be reported in  $O((n + k) \log n)$  time [3].

The sweep paradigm is useful also in computing the region visible from a point inside a polygon with holes. Typically, the region visible to a robot inside a non-convex room with several obstacles, can be modelled as the visible region from a point inside a polygon with holes. If there are  $n$  edges defining the boundary of the polygon and  $h$  triangular holes/obstacles, then the visible region can be computed in  $O(n \log h)$  time [2].

## References

- [1] Mark de Berg, Otfried Schwarzkopf, Marc van Kreveld and Mark Overmars, *Computational Geometry: Algorithms and Applications*, Springer.
- [2] S. K. Ghosh, *Visibility Algorithms in the Plane*, Cambridge University Press, Cambridge, UK, 2007.
- [3] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, New York, NY, Springer-Verlag, 1985.