WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

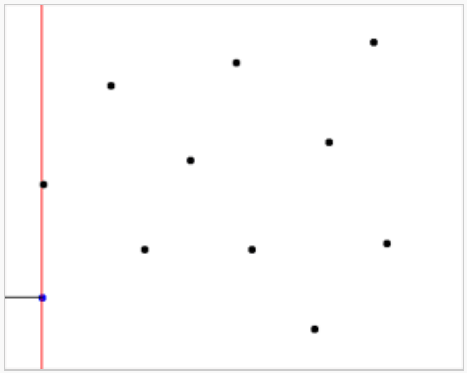
Print/export
Create a book
Download as PDF
Printable version

Languages
فارسی
Српски / srpski
ไทย
Українська
Edit links

Fortune's algorithm

From Wikipedia, the free encyclopedia
(Redirected from Fortune's Algorithm)

Fortune's algorithm is a sweep line algorithm for generating a Voronoi diagram from a set of points in a plane using O(*n* log *n*) time and O(*n*) space.[1][2] It was originally published by Steven Fortune in 1986 in his paper "A sweepline algorithm for Voronoi diagrams."[3]



Fortune's algorithm animation

Contents [hide]

Algorithm description [edit]

The algorithm maintains both a sweep line and a *beach line*, which both move through the plane as the algorithm progresses. The sweep line is a straight line, which we may by convention assume to be vertical and moving left to right across the plane. At any time during the algorithm, the input points left of the sweep line will have been incorporated into the Voronoi diagram, while the points right of the sweep line will not have been considered yet. The beach line is not a line, but a complicated, piecewise curve to the left of the sweep line, composed of pieces of parabolas; it divides the portion of the plane within which the Voronoi diagram can be known, regardless of what other points might be right of the sweep line, from the rest of the plane. For each point left of the sweep line, one can define a parabola of points equidistant from that point and from the sweep line; the beach line is the boundary of the union of these parabolas. As the sweep line progresses, the vertices of the beach line, at which two parabolas cross, trace out the edges of the Voronoi diagram. The beach line progresses by keeping each parabola base exactly half way between the points initially swept over with the sweep line, and the new position of the sweep line.

The algorithm maintains as data structures a binary search tree describing the combinatorial structure of the beach line, and a priority queue listing potential future events that could change the beach line structure. These events include the addition of another parabola to the beach line (when the sweep line crosses another input point) and the removal of a curve from the beach line (when the sweep line becomes tangent to a circle through some three input points whose parabolas form consecutive segments of the beach line). Each such event may be prioritized by the *x*-coordinate of the sweep line at the point the event occurs. The algorithm itself then consists of repeatedly removing the next event from the priority queue, finding the changes the event causes in the beach line, and updating the data structures.

As there are O(*n*) events to process (each being associated with some feature of the Voronoi diagram) and O(log *n*) time to process an event (each consisting of a constant number of binary search tree and priority queue operations) the total time is O(*n* log *n*).

Pseudocode [edit]

Pseudocode description of the algorithm.[4]

```
let ∗(z) be the transformation ∗(z) = (z_x, z_y + d(z)),
  where d(z) is the Euclidean distance between z and the nearest site
let T be the "beach line"
let R_p be the region covered by site p.
let C_pq be the boundary ray between sites p and q.
let p_1, p_2, ..., p_m be the sites with minimal y-coordinate, ordered by x-coordinate
Q ← S − p_1, p_2, ..., p_m
create initial vertical boundary rays C^0_{p1,p2}, C^0_{p2,p3}, ...C^0_{p_{m−1},p_m}
```

$T \leftarrow *(R_{p1}), C^0_{p1,p2}, *(R_{p2}), C^0_{p2,p3}, ..., *(R_{p_{m-1}}), C^0_{p_{m-1},p_m}, *(R_{p_m})$
while not IsEmpty(Q) **do**
    $p \leftarrow$ DeleteMin(Q)
    case p of
    p is a site in $*(V)$:
        find the occurrence of a region $*(R_q)$ in T containing p,
          bracketed by C_{rq} on the left and C_{qs} on the right
        create new boundary rays C^-_{pq} and C^+_{pq} with bases p
        replace $*(R_q)$ with $*(R_q), C^-_{pq}, *(R_p), C^+_{pq}, *(R_q)$ in T
        delete from Q any intersection between C_{rq} and C_{qs}
        insert into Q any intersection between C_{rq} and C^-_{pq}
        insert into Q any intersection between C^+_{pq} and C_{qs}
    p is a Voronoi vertex in $*(V)$:
        let p be the intersection of C_{qr} on the left and C_{rs} on the right
        let C_{uq} be the left neighbor of C_{qr} and
          let C_{sv} be the right neighbor of C_{rs} in T
        create a new boundary ray C^0_{qs} if $q_y = s_y$,
          or create C^+_{qs} if p is right of the higher of q and s,
          otherwise create C^-_{qs}
        replace $C_{qr}, *(R_r), C_{rs}$ with newly created C_{qs} in T
        delete from Q any intersection between C_{uq} and C_{qr}
        delete from Q any intersection between C_{rs} and C_{sv}
        insert into Q any intersection between C_{uq} and C_{qs}
        insert into Q any intersection between C_{qs} and C_{sv}
        record p as the summit of C_{qr} and C_{rs} and the base of C_{qs}
        output the boundary segments C_{qr} and C_{rs}
    endcase
endwhile
output the remaining boundary rays in T

Weighted sites and disks [edit]

As Fortune describes in [1] a modified version of the sweepline algorithm can be used to construct an additively weighted Voronoi diagram, in which the distance to each site is offset by the weight of the site; this may equivalently be viewed as a Voronoi diagram of a set of disks, centered at the sites with radius equal to the weight of the site.

Weighted sites may be used to control the areas of the Voronoi cells when using Voronoi diagrams to construct treemaps. In an additively weighted Voronoi diagram, the bisector between sites is in general a hyperbola, in contrast to unweighted Voronoi diagrams and power diagrams of disks for which it is a straight line.

References [edit]

1. ^ [a] [b] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf (2000), *Computational Geometry* (2nd revised ed.), Springer-Verlag, ISBN 3-540-65620-0 Section 7.2: Computing the Voronoi Diagram: pp.151–160.
2. ^ Austin, David, *Voronoi Diagrams and a Day at the Beach*, Feature Column, American Mathematical Society.
3. ^ Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Proceedings of the second annual symposium on Computational geometry*. Yorktown Heights, New York, United States, pp.313–322. 1986. ISBN 0-89791-194-6. ACM Digital Library SpringerLink
4. ^ Kenny Wong, Hausi A. Müller, *An Efficient Implementation of Fortune's Plane-Sweep Algorithm for Voronoi Diagrams*.

External links [edit]

- Steven Fortune's C implementation
- Fortune's Voronoi algorithm implemented in C++
- Fortune's algorithm implemented in JavaScript

Categories: Computational geometry