An Efficient Implementation of Fortune's Plane-Sweep Algorithm for Voronoi Diagrams

Ken Wong
kenw@csr.uvic.ca

Department of Computer Science
University of Victoria

Computational Geometry Victoria, BC

Outline

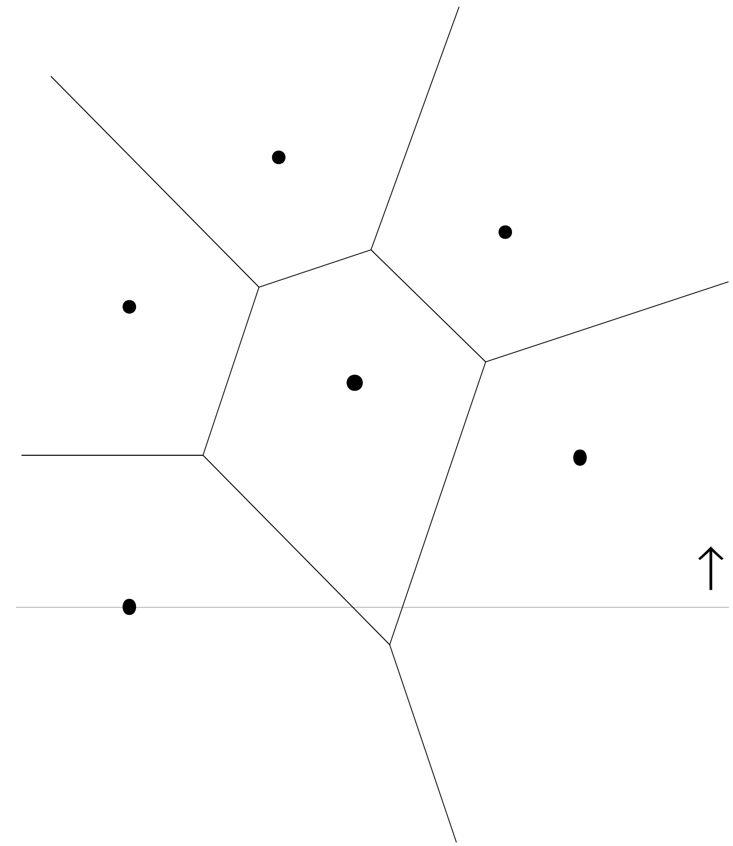
- Introduction
- Algorithm
- Data Structures
- Geometric Primitives
- Performance
- Applications

2-D Voronoi / Delaunay Algorithms

- Divide and conquer $O(n \log n)$
  - Shamos & Hoey 1975 algorithm outline
  - Lee & Schacter 1980 merge procedure
  - Guibas & Stolfi 1985 geometric primitives on quad-edge structure
  - Dwyer 1987 bucket scheme
  - Karasick et al. 1991 efficient version using rational arithmetic
- Incremental $O(n^2)$
  - Lawson 1972 flipped edges in a candidate triangulation
  - Green & Sibson 1978 construct polygons by local changes
  - Ohya et al. 1984 bucket scheme and quaternary tree traversal
- Plane sweep $O(n \log n)$
  - Fortune 1987 compute transformed diagram

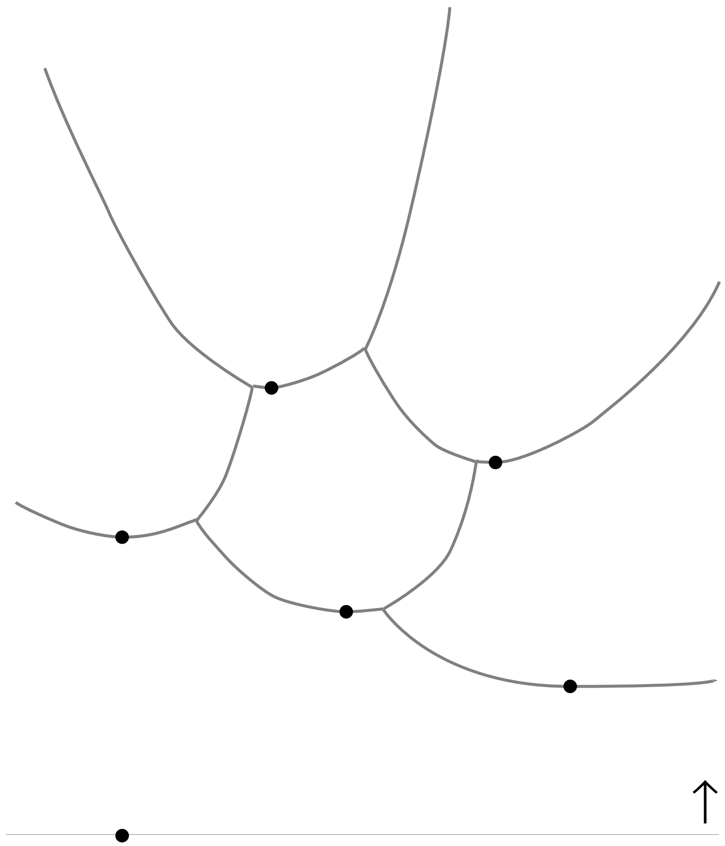
Plane Sweep and Voronoi Diagrams

- Approach
  - sweep horizontal line across the sites (bottom to top)
  - diagram *V* is constructed behind moving front
  - maintain intersection of diagram with current sweepline in *sweep table*
  - process events where sweepline momentarily stops (at sites and vertices) according to *event queue*
  - problem: edges and regions are encountered by the sweepline before the defining sites themselves …



Geometric Mapping

- Solution
  - compute a transformed diagram: $*(V)$
  - maintain semi-infinite *boundary rays* in the sweep table …
  - determine where these rays intersect to become boundary segments
  - these intersections are transformed Voronoi vertices
  - the boundary segments map back to subsets of Voronoi edges
  - can perform the inverse mapping at the same time

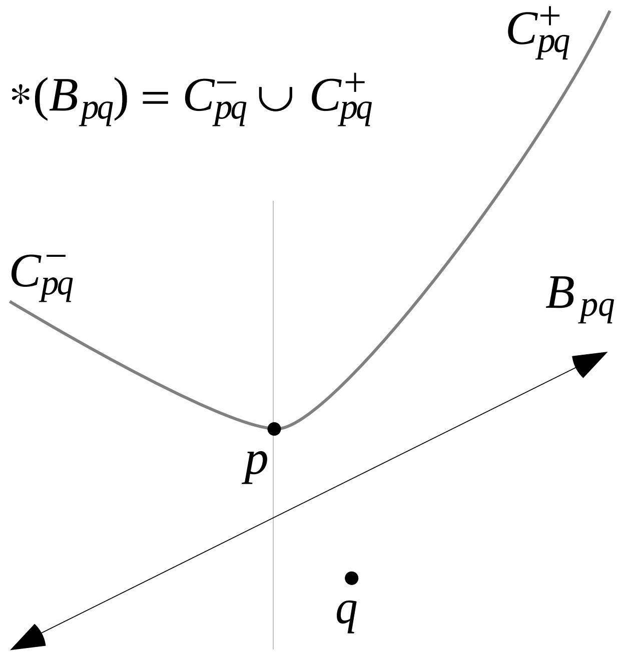


$*$ Transform

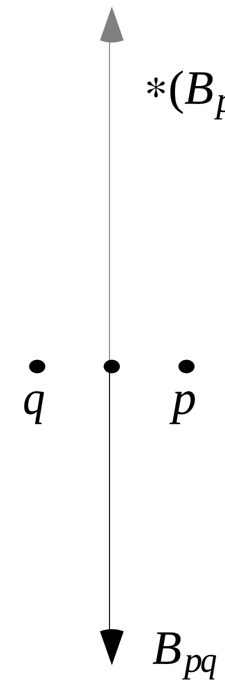
- Definition
  - $* : \Re^2 \to \Re^2$
    $\forall\, z = (z_x, z_y) \in \Re^2$
    $*(z) = (z_x, z_y + d(z))$ where d is the Euclidean distance to the nearest site

- Use
  - only ever applied to the perpendicular bisector between neighboring sites …
  - form *boundaries* (rays and segments) based on these bisectors

$*(B_{pq}) = C^-_{pq} \cup C^+_{pq}$

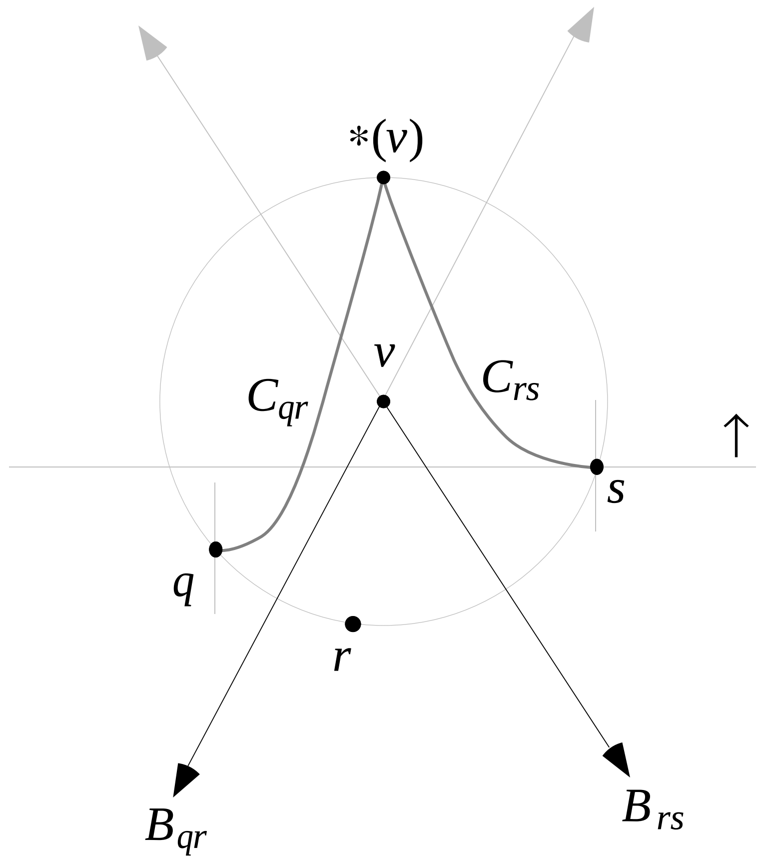


$*(B_{pq}) = C^0_{pq}$



$*$ Transform

- Properties of $*$
  - continuous
  - one-to-one on the edges of V
  - sites stay fixed
  - Voronoi vertices map to the top of Voronoi circles …
  - Voronoi edges map to sections of hyperbolas or vertical lines
  - each site p is the unique lowest point of the transformed Voronoi region around p



Algorithm

- Definitions
  - input: $S \equiv$ the n sites
  - output: subsets of $*(V)$
  - [can compute doubly-connected edge representation of the (primal) Voronoi diagram and dual Delaunay triangulation at the same time]
  - $Q \equiv$ event queue of sites and transformed (candidate) vertices
  - $T \equiv$ sweep table of boundary rays and transformed regions (ordered left-to-right by their intersection with the sweepline)
  - [since sites are fixed under *, they can be stored in a separate, presorted array and merged with the vertices coming out of Q]
  - [maintaining the transformed regions in T is not necessary]

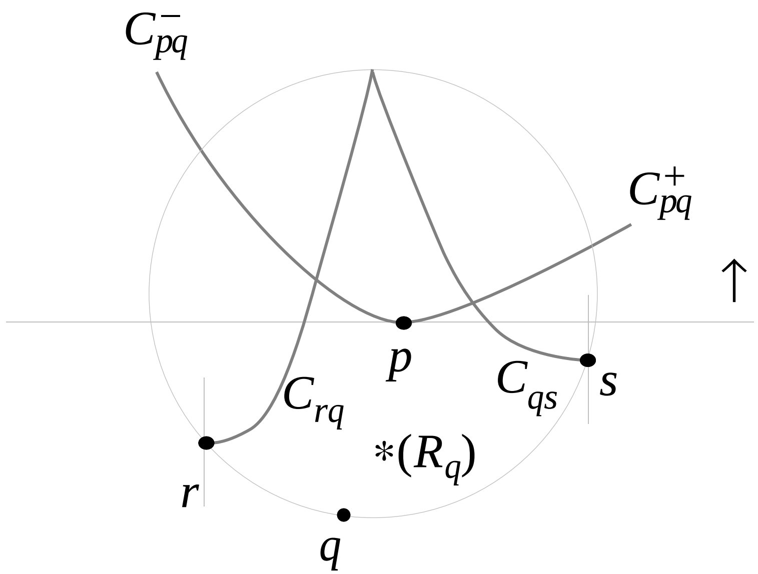
Algorithm

- Outline

```
initialize Q
initialize T

while not Q.IsEmpty( ) do
    p ← Q.DeleteMin( )

    case p of
    p is a site in ∗(V):
        ProcessSite( p )

    p is a Voronoi vertex in ∗(V):
        ProcessVertex( p )          -- produces output
    endcase
endwhile

output the remaining boundary rays in T as unbounded edges of ∗(V)
```

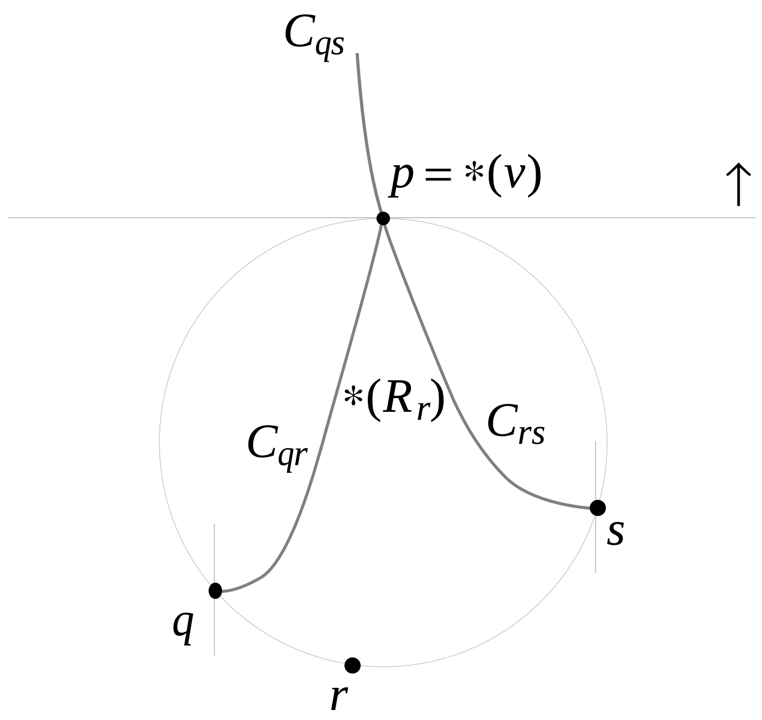
Algorithm

- Processing a site
  - find a region in T which contains site p
  - create a region and minus and plus boundary rays whose bases are p
  - split the region containing p and insert the three new objects
  - delete invalid candidate vertices in Q
  - insert candidate vertices in Q



Algorithm

- Processing a vertex
  - determine the two boundary rays for which p is their intersection
  - create a boundary ray whose base is p
  - replace the two rays and the region between them with the new ray
  - delete invalid candidate vertices in Q
  - insert candidate vertices in Q
  - mark p as the endpoint of the two boundary rays
  - output the two boundary segments as part of $*(V)$



Data Structures

- Event Queue

$Q \leftarrow$ New(EQ)	
Q.Free( )	
$i \leftarrow Q$.Size( )	$O(1)$
$b \leftarrow Q$.IsEmpty( )	$O(1)$
$h \leftarrow Q$.Insert(p, e)	$O(\log k)$
$e \leftarrow Q$.EventAt(h)	$O(1)$
$e \leftarrow Q$.Min( )	$O(1)$
$e \leftarrow Q$.DeleteMin( )	$O(\log k)$
$e \leftarrow Q$.Delete(h)	$O(\log k)$
Q.Replace(h, p, e)	$O(\log k)$

  - [implicit heap with handles]
  - [average maximum size is about O($\sqrt{n}$) on uniformly random sites]

Data Structures

- Sweep Table

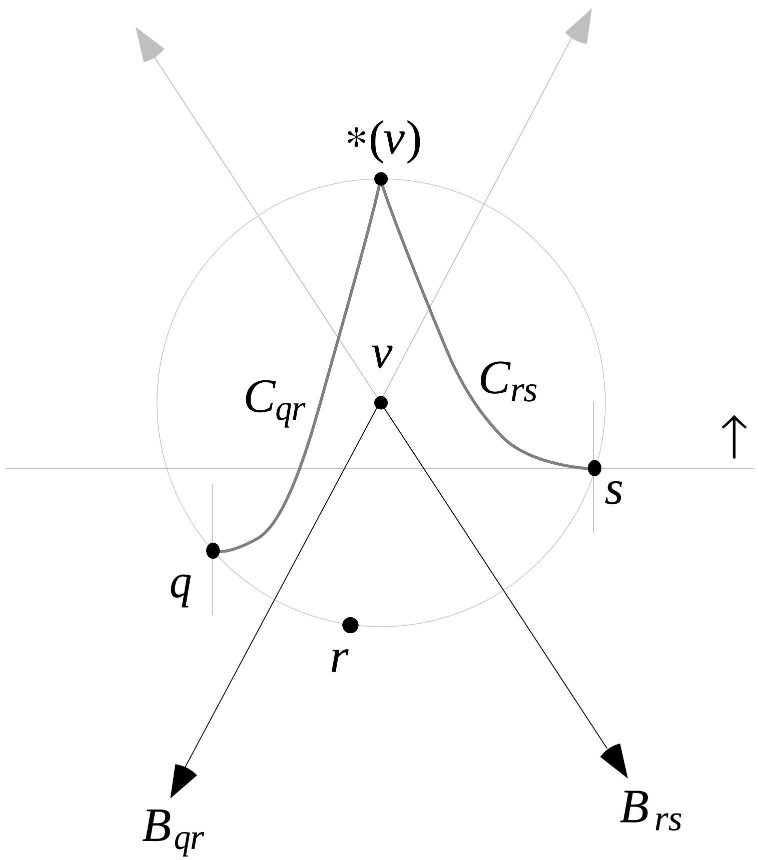
$T \leftarrow$ New ST	
T.Free( )	
$h \leftarrow T$.Insert($h1$, $h2$, x)	$O(1)$
$x \leftarrow T$.ObjectAt(h)	$O(1)$
$x \leftarrow T$.Before(h)	$O(1)$
$x \leftarrow T$.After(h)	$O(1)$
$h \leftarrow T$.PlaceHolder(s)	$O(1)$
T.Delete(h)	$O(1)$
T.Replace(h, x)	$O(1)$
T.InOrder(f, z)	$O(k)$
$x \leftarrow T$.Search(g, d)	$O(\log k)$

- [randomized, threaded binary tree with handles]
- [optional balancing using treap rotation technique]

- [average maximum size is about O($\sqrt{n}$) on uniformly random sites]

Geometric Primitives

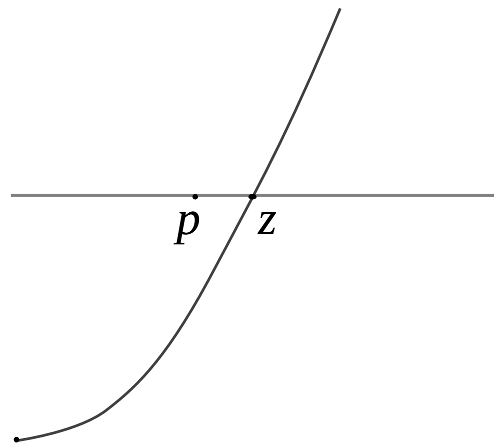
- Candidate vertices
  - determine whether consecutive boundary rays in the sweep table intersect to form a candidate vertex
  - compute that intersection
  - [fast left-turn test]
  - [result of test can be used within intersection calculation]



Geometric Primitives

- Region find
  - determine the relative position of a boundary (hyperbolic curve) and a site
  - [efficient test (no divisions or square roots)]
  - [history test (exploit past decisions and the monotonicity of boundaries)]

$$p_x \stackrel{?}{=} z_x$$



Performance

- Robustness
  - handles co-circular and co-linear degeneracies
  - hand verified test suite

- Efficiency
  - minimize effect of $*$-transform
  - manage memory allocation and free lists
  - reduce size of data structures (avoid vertices that cannot be Voronoi)
  - compare alternative data structure representations (splay trees)
  - cache subexpressions
  - special cases
  - doubly connected edge representation for output (write once)
  - tuned implementation (reduce hotspots)

  - 100 000 sites in 5.7 s on RS/6000 C20 server (120 MHz PPC 604)

Applications

- Graph layout
  - layout adjustment by “cluster busting” [Lyons]

  - compute Voronoi diagram
  - move each site to centroid of surrounding Voronoi polygon
  - recompute diagram and iterate

- Constrained triangulation
  - force specific edges to appear in constructed triangulation
  - (these edges were parts of streams and ridges in terrain data)

  - adaptively subdivide edges into shorter segments
  - compute Delaunay triangulation