May 11, 2012

RV College of Engineering

Department of Computer Science

Advanced Algorithms Assignment

Voronoi Diagrams

By:
Samir Sheriff [1RV09CS093]
Satvik N [1RV09CS095]
Vinay BS [1RV09CS118]

May 11, 2012

1 Introduction

Voronoi diagram is a decomposition of a given space, determined by distances to a specified family of objects (subsets) in the space. These objects are usually called the sites or the generators and to each such object one associates a corresponding Voronoi cell, namely the set of all points in the given space whose distance to the given object is not greater than their distance to the other objects. It is named after Georgy Voronoi, and is also called a Voronoi tessellation, a Voronoi decomposition, or a Dirichlet tessellation (after Lejeune Dirichlet). Voronoi diagrams can be found in a large number of fields in science and technology, even in art, and they have found numerous practical and theoretical applications. It is the technique that enables the division of such multi-dimensional spaces into subspaces.

A simplest definition

In the simplest and most familiar case, we are given a finite set of points $\{p_1, ..., p_n\}$ in the Euclidean plane. In this case each site p_k is simply a point, and its corresponding Voronoi cell (also called Voronoi region or Dirichlet cell) R_k consisting of all points whose distance to p_k is not greater than their distance to any other site. Each such cell is obtained from the intersection of half-spaces, and hence it is a convex polygon. The segments of the Voronoi diagram are all the points in the plane that are equidistant to the two nearest sites. The Voronoi vertices (nodes) are the points equidistant to three (or more) sites.

Formal definition

Let X be a space (a nonempty set) endowed with a distance function d. Let K be a set of indices and let $(P_k)_{k \in K}$ be a tuple (ordered collection) of nonempty subsets (the sites) in the space X. The Voronoi cell, or Voronoi region,R_k , associated with the site is the set of all points in X whose distance to P_k is not greater than their distance to P_j the other sites , where j is any index different from k. In other words, if

$$d(x, A) = inf\{d(x, a) | a \in A\}$$

denotes the distance between the point x and the subset A, then

$$R_k = \{x \in X | d(x, P_k) \leq d(x, P_j) for all j \neq k\}$$

The Voronoi diagram is simply the tuple of cells $(R_k)_{k \in K}$. In principle some of the sites can intersect and even coincide (an application is described below for sites representing shops), but usually they are assumed to be disjoint. In addition, infinitely many sites are allowed in the definition (this setting has applications in geometry of numbers and crystallography), but again, in many cases only finitely many sites are considered.

In the particular case where the space is a finite dimensional Euclidean space, each site is a point, there are finitely many points and all of them are different, then the Voronoi cells are convex polytopes and they can be represented in a combinatorial way using their vertices, sides, 2-dimensional faces, etc. Sometimes the induced combinatorial structure is referred to as the Voronoi diagram. However, in general the Voronoi cells may not be convex or even connected.

2 Characteristics and Properties

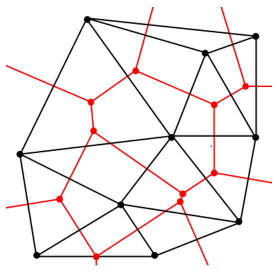
- The dual graph (the dual graph of a planar graph G is a graph which has a vertex corresponding to each plane region of G, and an edge joining two neighboring regions for each edge in G.) for a Voronoi diagram (in the case of a Euclidean space with point sites) corresponds to the Delaunay triangulation for the same set of points. The Delaunay triangulation of a discrete point set P in general position corresponds to the dual graph of the Voronoi tessellation for P. Special cases include the existence of three points on a line and four points on circle.

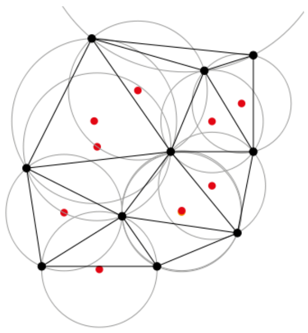
- Under relatively general conditions Voronoi cells enjoy a certain stability property: a small change in the shapes of the sites, e.g., a change caused by some translation or distortion, yields a small change in the shape of the Voronoi cells. This is the geometric stability of Voronoi diagrams . As shown there, this property does not hold in general, even if the space is two-dimensional (but non-uniformly convex, and, in particular, non-Euclidean) and the sites are points.

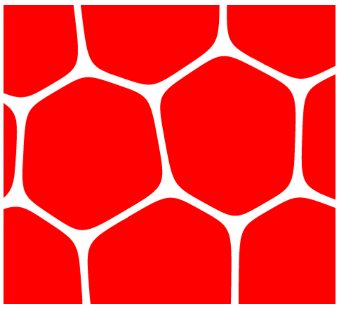
- The closest pair of points corresponds to two adjacent cells in the Voronoi diagram.

- Assume the setting is the Euclidean plane and a group of different points are given. Then two points are adjacent on the convex hull if and only if their Voronoi cells share an infinitely long side.

- If the space is a normed space and the distance to each site is attained (e.g., when a site is a compact set or a closed ball), then each Voronoi cell can be represented as a union of line segments emanating from the sites.


(a) Connecting the centers of the circumcircles produces the Voronoi diagram


(b) The Delaunay triangulation with all the circumcircles and their centers


(c) The stability property of voronoi diagrams.

3 Algorithms

There are several ways to compute Voronoi diagrams, four of which are described below:

Incremental algorithm

Green and Sibson calculated the Voronoi diagram by incremental insertion i.e., obtained V(S) from V(S{s}) by inserting the site s. As the region of s can have up to (n_1) edges, for n = $\|S\|$, this leads to a runtime of O(n^2). Ohya, Iri, Murota [2] and Sugihara [3] have polished up the technique of inserting Voronoi regions, and make incremental algorithm efficient and numerically robust. In fact, well-distributed sets of sites achieve average time complexity of O(n).

The incremental method set up with a simple Voronoi diagram for two or three sites, and modified the diagram by adding sites one by one. For p = 1, 2, . . . , n, let V_p denoted the Voronoi diagram for the first p sites $s_1, s_2, ..., s_p$.

The major part of the incremental method is to translate V_{p-1} to $V_p$for each p.

The basic idea the incremental Voronoi diagram is as follows: Suppose that we have already built the Voronoi diagram V_{p-1} as shown by solid lines, and would like to add a new sites s_p. First, find the site, say s_i, whose Voronoi polygon contains s_p, and draw the perpendicular bisector between s_l and s_i, denoted by B(s_p, s_i). The bisector crosses the boundary of V(s_i) at two points, point x_1 and point x_2. Site s_p is to the left of the directed line segment x_1 x_2. The line segment x_1 x_2 divides the Voronoi polygon V(s_i) into two pieces, the one on the left belonging to the Voronoi polygon of s_p. Thus, we get a Voronoi edge on the boundary of the Voronoi polygon of s_i.

Starting with the edge x_1 x_2, expand the boundary of the Voronoi polygon of s_l by the following procedure. The B(s_i, s_l) crosses the boundary of V(s_i) at x_2, entering the adjacent Voronoi polygon, say V(s_j). Therefore, next draw the B(s_i, s_j), and find the point, x_3, at which the bisector crosses the boundary of V(s_j). Similarly, find the sequence of segments of perpendicular bisectors of s and the neighboring sites until we reach the starting point x_1. Let this sequence be (x_1 x_2, x_2 x_3, , x_{m-1} x_m, x_m x_1). This sequence forms a counterclockwise boundary of the Voronoi polygon of the new site s. Finally, we delete from V_{p-1} the substructure inside the new Voronoi polygon, and thus get V_p.

The Basic Idea

Starting with the Voronoi diagram of $\{p_1, ...p_i\}$,

- add point p_{i+1}. Explore all candidates to find the site p_j ($1 \leq j \leq i$) closest to p_{i+1}.
- compute its region. Build its boundary starting from bisector $b_{i+1,j}$.
- and prune the initial diagram.
- Each time an edge e, generated by p_{i+1} and p_j, intersects a preexistent edge, e_0, a new vertex v is created and a new edge starts, e + 1. Then, these are the tasks to perform:
  - Assign $v_E(e) = v, e_{N(e)} = e^{'}$, $f_L(e) = i + 1$, $f_R(e) = j$
  - Create e+1 and assign $v_B(e + 1) = v$, $e_P(e + 1) = e$
  - Delete all edges of the region of p_j, that lie between $v_B(e)$ and $v_E(e)$ in clockwise order
  - Update e(p_j) = e
  - Create v with e(v) = e

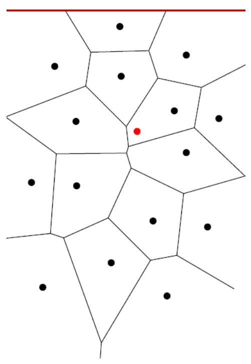


Figure 1: A New Point

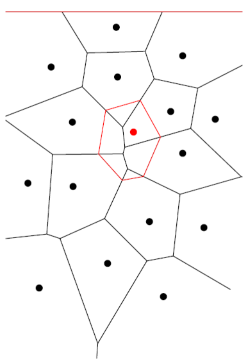


Figure 2: Constructing the Region

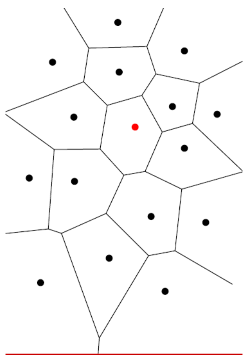


Figure 3: Voronoi Diagram after Pruning

Figures

Algorithmic Analysis

Each step runs in O(i) time, therefore the total running time of the algorithm is O(n$\hat{2}$).

Divide and Conquer

Let P be a set of n points in the plane.If the points are vertically partitioned into two subsets R and B, consider the Voronoi diagram of the sets R and B, then the Voronoi diagram of P substantially coincides with the Voronoi diagrams of R and B!

The Algorithm

1. Sort the points of P by abscissa (only once) and vertically partition P into two subsets R and B, of approximately the same size.

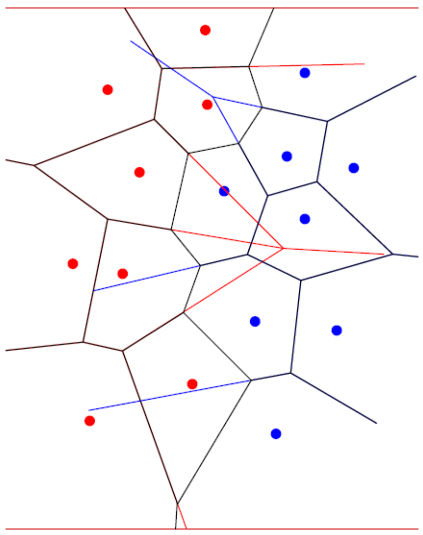


Figure 4: Divide and Conquer

2. Recursively compute Vor(R) and Vor(B).
3. Compute the separating chain.
4. Prune the portion of Vor(R) lying to the right of the chain and the portion of Vor(B) lying to its left.

Computing the chain

1. Find the two halfines
2. Advance Starting with one of the halfines, and until getting to the other one, do:

   Each time an edge $e \in b(R;B)$ begins, such that $e \subset b_{ij}, p_i \in R$ and $p_j \in B$, do:

   - Detect its intersection with $Vor_R(pi)$
   - Detect its intersection with $Vor_B(pj)$
   - Choose the first of the two intersection points
   - Detect the site p_k corresponding to the new starting region
   - Replace p_i or p_j (as required) by pk
   - Restart with the new edge

Merging R and B

Each time a face $Vor_{B(p_i)}$ is left through an edge $\{e^{'} \in b_{ij}\}$, while staying in the same face $Vor_{R(p_k)}$, a new vertex v is created, an edge e ends and another edge e + 1 begins:

- Create e+1 and assign to it $v_B = v$and $e_P = e^{'}$
- Assign to e: v_E = v, e_N = e + 1, f_L = i and f_R = k
- Delete all edges of $Vor_B(p_k)$ found in counterclockwise order between the entry and exit points
- Update e(p_i) = e
- Create the new vertex v and assign e(v) = e

Algorithmic Analysis

The total running time of the algorithm is O(n log n). This running time is optimal, because ch(P) can be computed from Vor(P) in O(n) time.

Fortune's Algorithm

The algorithm maintains both a sweep line and a beach line, which both move through the plane as the algorithm progresses.

Pseudo-Code

```
let $*(z)$ be the transformation $*(z) = (z_x, z_y + d(z))$, where $d(z)$ is a parabola with minimum at $z$
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
$Q \leftarrow S - p_1, p_2, ..., p_m$
create initial vertical boundary rays $C^0_{p1,p2}, C^0_{p2,p3}, ...C^0_{pm-1,pm}$
$T \leftarrow *(R_{p1}), C^0_{p1,p2}, *(R_{p2}), C^0_{p2,p3}, ..., *(R_{pm-1}), C^0_{pm-1,pm}, *(R_{pm})$
while not IsEmpty($Q$) do
    $p$ ← 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 $T$
```

Technique

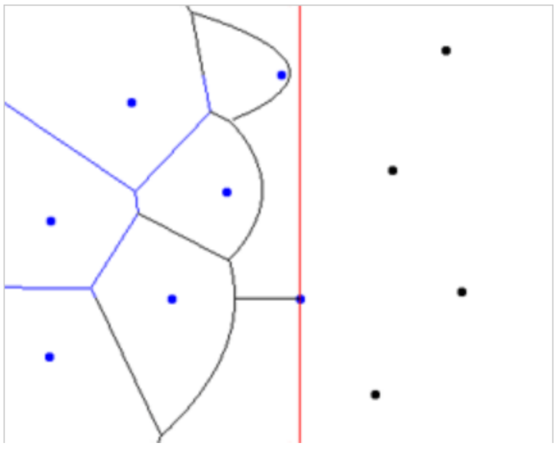


Figure 5: Fortune's Algorithm

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 complex 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 algorithm maintains as data structures a binary search tree describing the combinatorial structure of the beach line, and a priority queuelisting 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.

Algorithmic Analysis

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).

Lloyd's algorithm

An ordinary Voronoi diagram is formed by a set of points in the plane called the generators or generating points. Every point in the plane is identified with the generator which is closest to it by some metric. The common choice is to use the Euclidean distance metric

$$|x_1 - x_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

where $x_1 = (x_1, y_1)$ and $x_2 = (x_2, y_2)$ are any two points in a plane.

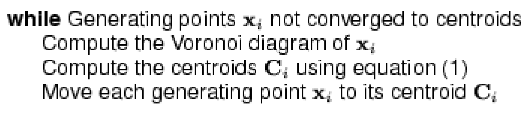
Centroidal Voronoi diagrams

A centroidal Voronoi diagram has the odd property that each generating point lies exactly on the centroid of their Voronoi region. The centroid of a region is defined as

$$C_i = \frac{\int_A x\rho(x)dA}{\int_A \rho(x)dA}$$

where A is the region, x is the position and $\rho(x)$ is the density function.

Lloyd's method is an iterative algorithm to generate a centroidal Voronoi diagram from any set of generating points. The algorithm can simply be stated as in figure 6:



```
while Generating points $\mathbf{x}_i$ not converged to centroids
    Compute the Voronoi diagram of $\mathbf{x}_i$
    Compute the centroids $\mathbf{C}_i$ using equation (1)
    Move each generating point $\mathbf{x}_i$ to its centroid $\mathbf{C}_i$
```

Figure 6: The Lloyd's algorithm

4 Applications

- Voronoi diagrams are also used in computer graphics to procedurally generate some kinds of organic looking textures.

- The Voronoi diagram is useful in polymer physics. It can be used to represent free volume of the polymer.
- Voronoi diagrams are used to study the growth patterns of forests and forest canopies, and are helpful in developing predictive models for forest fires.
- Voronoi diagrams together with farthest-point Voronoi diagrams are used for efficient algorithms to compute the roundness of a set of points. Roundness is the measure of the sharpness of a particle's edges and corners.
- In computational chemistry, Voronoi cells defined by the positions of the nuclei in a molecule are used to compute atomic charges. This is done using the Voronoi deformation density method.
- One of the early applications of Voronoi diagrams was by John Snow to study the epidemiology of the 1854 Broad Street cholera outbreak in Soho, England. He showed the correlation between areas on the map of London using a particular water pump, and the areas with most deaths due to the outbreak.
- In autonomous robot navigation, Voronoi diagrams are used to find clear routes. If the points are obstacles, then the edges of the graph will be the routes furthest from obstacles (and theoretically any collisions).
- The Voronoi diagrams find applications in the evaluation of circularity / roundness while using assesing the dataset from a Coordinate-measuring machine.
- A point location data structure can be built on top of the Voronoi diagram in order to answer nearest neighbor queries, where one wants to find the object that is closest to a given query point. Nearest neighbor queries have numerous applications. For example, one might want to find the nearest hospital, or the most similar object in a database. A large application is vector quantization, commonly used in data compression.
- With a given Voronoi diagram, one can also find the largest empty circle amongst a set of points, and in an enclosing polygon; e.g. to build a new supermarket as far as possible from all the existing ones, lying in a certain city.
- It is also used in derivations of the capacity of a wireless network.
- In machine learning, Voronoi diagrams are used to do 1-NN classifications.

- Voronoi polygons have been used in mining to estimate the reserves of valuable materials, minerals or other resources. Exploratory drillholes are used as the set of points in the Voronoi polygons.
- In materials science, polycrystalline microstructures in metallic alloys are commonly represented using Voronoi tessellations.
- In climatology, Voronoi diagrams are used to calculate the rainfall of an area, based on a series of point measurements. In this usage, they are referred to as Thiessen polygons.

References

[1] Applet for the Incremental Algorithm- `http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/Voronoi/Incremental2/incremental2.htm`

[2] Applet for Fortune's Algorithm- `http://www.diku.dk/hjemmesider/studerende/duff/Fortune/`

[3] Applet for the Divide And Conquer Algorithm- `http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/Voronoi/DivConqVor/divConqVor.htm`

[4] Applet for the Divide And Conquer Algorithm- `http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/Voronoi/DivConqVor/divConqVor.htm`

[5] Video of Lloyd's algorithm- `http://www.youtube.com/watch?v=qqRtSGvy0no`

[6] A PPT explaining the workings of the Incremental Algorithm and the Divide and Conquer Algorithm, pictorially- `http://www-ma2.upc.es/~geoc/mat1q1112/construccio_voronoi_EN.pdf`

[7] An explanation of Lloyd's Algorithm- `http://cs.nyu.edu/~ajsecord/npar2002/html/stipples-node2.html`

[8] A Javascript implementation of Voronoi Diagrams using the 'paper.js' framework- `http://paperjs.org/examples/voronoi/`

[9] Wikipedia - Voronoi diagram `http://en.wikipedia.org/wiki/Voronoi_diagram`