Voronoi Diagram

Pronunciation : **American/English speakers pronounce “Voronoi” as "Vo - ro - noi" with a short "o" sound, like the "o" in "or", for the first two syllables. The third syllable is pronounced like the "noi" as in "noise". The stress is on the third syllable. On the other hand, Russian speakers pronounce the name with a short "o" in the first two syllables and it is such a short "o" that it almost sounds likes "uh" or even "ah".**

General Definitions

A Voronoi Diagram for a set S of N planar points is a partition of the plane into N polygonal regions, each of which is associated with some point p_i and S_1 and is the locus of points closer to that point than to any other points [1].

Mathematically, let $P = \{p_1, \ldots, p_n\}$, where $2 \leq n \leq \infty$ and $x_i \neq x_j$ for $i \neq j, \quad i, j \in I_n$. The region given by

$$V(p_i) = \{x: \| x - x_i \| \leq \| x - x_j \| \quad \text{for } j \neq i, \quad i \in I_n\}$$

is called ordinary Voronoi polygon associated with p_i or the Voronoi polygon of p_i and the set given by

$$V = \{ V(p_i), \ldots, V(p_n)\}$$

is called the planar ordinary Voronoi diagram generated by P or simply Voronoi diagram of P.

We can also define a planar ordinary Voronoi diagram with half planes as follows [2]: Let $P = \{ p_i, \ldots, p_n \} \subseteq R^2$, where $2 \leq n \leq \infty$ and $x_i \neq x_j$ for $i \neq j$, $i, j \in I_n$. We call the region

$$V(p_i) = \bigcap_{j \in In \setminus \{i\}} H(p_i, p_j)$$

the ordinary Voronoi polygon associated with p_i and set $V(P) = \{V(p_1), \ldots, V(p_n)\}$ the planar ordinary Voronoi diagram generated by P.

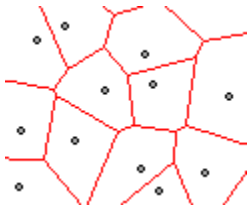
Problem Formulation [1]

Given a set S of N points in the plane, for each $p_i \in S$, what is the locus of points (x, y) in the plane that are closer to p_i than to any other point of S?

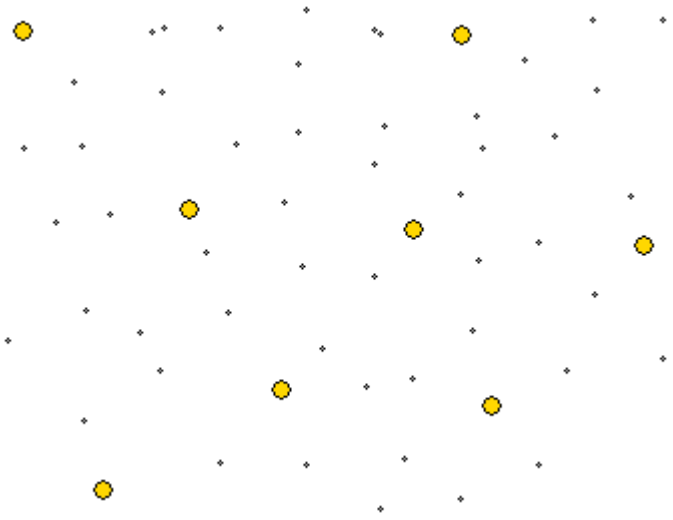
Examples

Example 1 Suppose we are given following points

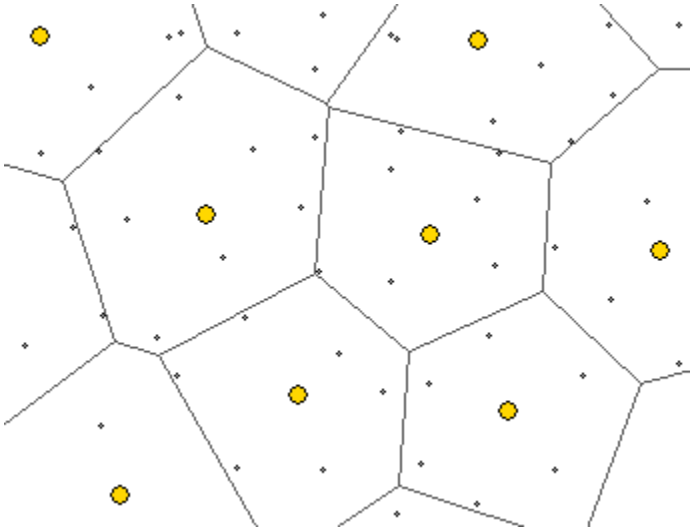
A Voronoi diagram divides the drawing into a region around each point such that the borders of the regions are equidistance from the two nearest points.



Example 2 Suppose we have a few hundred environmental sampling stations scattered throughout a region. Also, suppose that we have a eight data collection centers (yellow points).



The problem is to assign each sampling station to the nearest collection center. The solution is to create a Voronoi Area around each data center (yellow points) as follows:



Incremental Method

The incremental method is a powerful method for the construction of Voronoi diagram. This method has been one of the most popular methods for studying the concept of Voronoi diagram and for practical implementation. This method starts with a simple Voronoi diagram for three sites and modifies the diagram by adding a new point one at time. In the worst case, the addition of points is time proportional to the number of points added so far. Consequently, the worst-case time complexity is $\theta(n^2)$.

In mathematics, a **Voronoi diagram** is a special kind of decomposition of a metric spacedetermined by distances to a specified discrete set of objects in the space, e.g., by a discrete set of points. It is named after Georgy Voronoi, also called a **Voronoi tessellation**, a **Voronoi decomposition**, or a **Dirichlet tessellation** (after Lejeune Dirichlet),

Given a set of points, aVoronoi diagram defines a series of cells surrounding each point. Each cell contains all points that are closer to its defining point than to any other point in the set. Subsequently, the "borders" of the cells are equidistant between the defining points of adjacent cells. I'll give you a diagram later.

In the simplest case, we are given a set of points *S* in the plane, which are the Voronoi sites. Each site *s* has a Voronoi cell, also called a Dirichlet cell, *V(s)* consisting of all points closer to *s* than to any other site. The segments of the Voronoi diagram are all the points in the plane that are equidistant to the

two nearest sites. The Voronoi nodes are the points equidistant to three (or more) sites.

More introduction

Say you have a map of your city, and on it you have located a set of points representing every fast food restaurant. Creating a Voronoi diagram from these points would enable you (on your walk through the city) to know which restaurant you're closest to at any given time. Of course, an application that simple could be solved more efficiently via other methods, but let's raise the stakes a bit. What if you want to know the percentage of your walk that will be closest to one particular McDonalds. Suddenly, without a Voronoi diagram, this is tricky. With a Voronoi diagram, however, it's a simple matter of intersecting the line that represents your walk with the cell that surrounds that particular restaurant.

Voronoi diagrams can also be used to *make* maps, not just analyze them. Say you have a set of points that represent air quality sample locations. To quickly generalize the sample points into a local map… bam! Voronoi diagram!
There are other more abstract information processing uses for the diagrams as well, but I'm not going to get into them here.

A few more notes

- The most efficient way to create a Voronoi diagram is via Fortune's sweepline method, which reminds me of how police departments use lines of people to do a walking search of an open area. We're not going to use that method here. The math is less intuitive.
- The dual graph of the Voronoi diagram for a set of points is called a Delaunay triangulation. It's a pretty great way of generating a mesh from a set of points, because it allows for an efficient non-uniform distribution of detail.
- All Voronoi cells are convex hulls, assuming that the boundary we are working within is a convex hull. This will be important.

Applications

One of the early applications of Voronoi diagrams was by John Snow to study theepidemiology 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.

A point location data structure can be built on top of the Voronoi diagram in order to answernearest 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 isvector 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.

The Voronoi diagram is useful in polymer physics. It can be used to represent free volume of the polymer.

It is also used in derivations of the capacity of a wireless network.

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 generally referred to as Thiessen polygons.

Voronoi diagrams are used to study the growth patterns of forests and forest canopies, and may also be helpful in developing predictive models for forest fires.

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

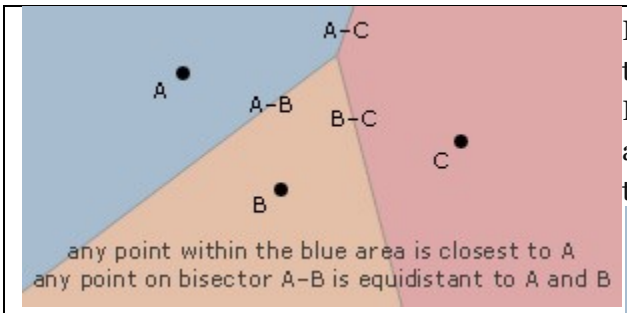
In autonomous robot navigation, Voronoi diagrams are used to find clear routes. If the points are obstacles, then the edges of the graph will be the routes furthest from obstacles (and theoretically any collisions).

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.

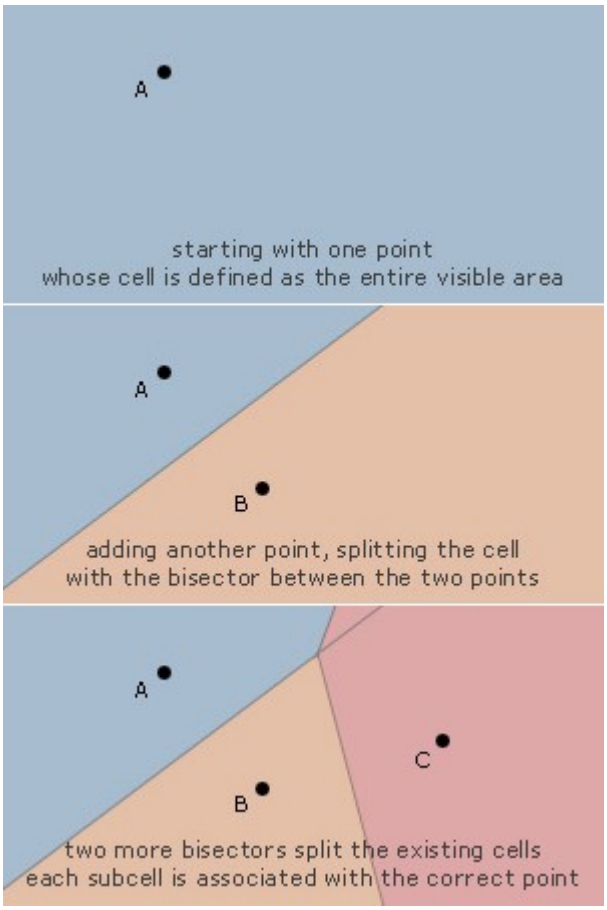
Voronoi Polygons have been used in mining to estimate the reserves of valuable materials, minerals or other resources. Exploratory drillholes are used as the set of points in the Voronoi polygons.

The procedure

First, let's review what exactly a Cell will define. The first image here shows the basics.



Every point within the blue area is closer to Point A than it is to B or C. Likewise for the other two. Points that lie on the segments between the colored areas are equidistant from the Points that define those areas.



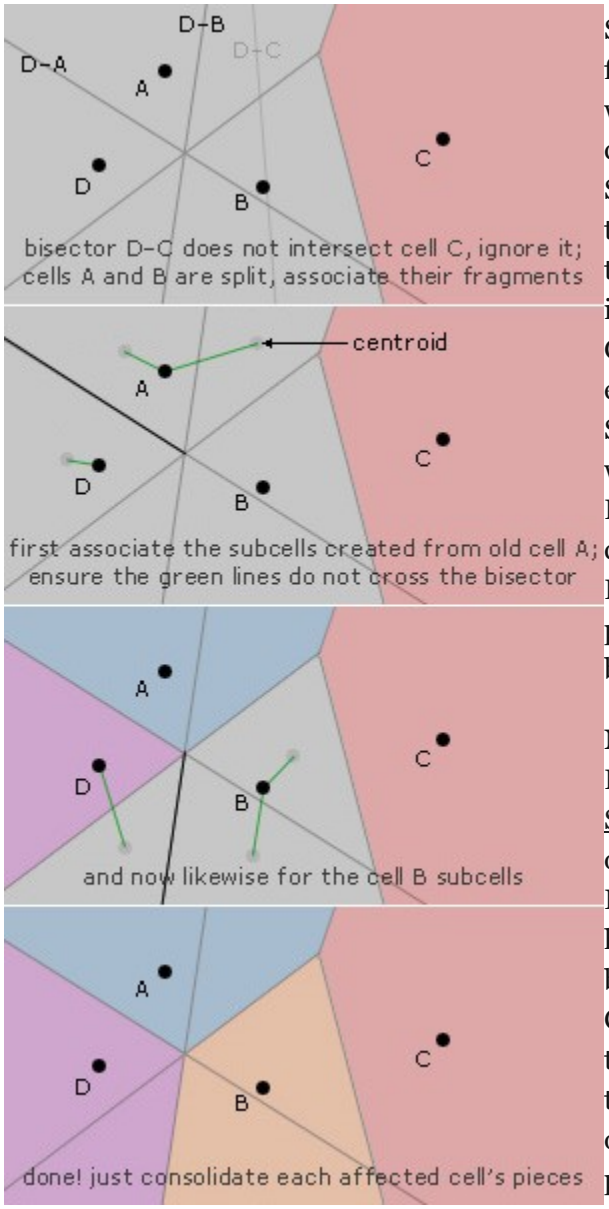
We'll be using a procedural approach to create our Voronoi diagram. One point at a time. The images on the left illustrate how we'd go about creating the previous example.
Start with Point A. Its initial Cell is the entire boundary of the image. Now add Point B and a Bisector between the two Points, which splits Cell A into two pieces (Subcells). We'll get into how to determine which Subcell belongs to which Point later.

Now we'll add Point C, along with a Bisector between A and C, and between B and C. If Bisector A-C intersects Cell A, we split Cell A along that line. Likewise for Bisector B-C.

If the Bisector does not intersect the Cell whose Point defines it, that Cell is not affected by the insertion of the new Point.
If it *does* intersect and we split it, we'll now need to associate all of its pieces with either the old Cell's defining Point, or the Point that we're inserting.

Associating and consolidating Subcells

In the example below, we have a few Cells that have been affected by the addition of the new Point D (shown in grey). The issue now is that we need to match each Cell fragment (Subcell) with a Point. Here's how we do it.



Start with all of the Subcells that have been created from an old Cell (in the example image, we start with old Cell A). Create a Segment between the centroid of each Subcell and the new Point. Create a Segment between the centroid of each Subcell and the old Cell's defining Point. Whichever of those two Segments does *not* intersect the Bisector indicates the owner the new Subcell.
Get it? We're testing which side of the Bisector line each Subcell falls on. We choose the centroid of the Subcell because (since each Subcell is a convex hull) we know that it will fall within the Subcell's Polygon. The Segment connecting any internal coordinate of the Polygon and the Subcell's owner Point should not cross the Bisector. Repeat this procedure for each old Cell that has been affected by the inserted Point.

Now we've got each new Subcell associated with a Point. Overall, our applet looks like this: Many Subcell fragments, correctly colored. (You can click on the applet to add new Points.)
Each Subcell belongs to the correct Point, but we have so many Polygons on the screen that it's become a disaster in terms of visual clarity and CPU-usage. We'd rather not have any more Subcells than we need. The number of Subcells should equal the number of Cells, which is equal to the number of input Points. We only want to split one Polygon per Cell operation.

To consolidate the Subcells of a particular Cell, we combine all of the Points of all of its Subcells into an array, create a funny looking Polygon from them, then simplify that Polygon to be a convex hull, effectively removing all internal Points.