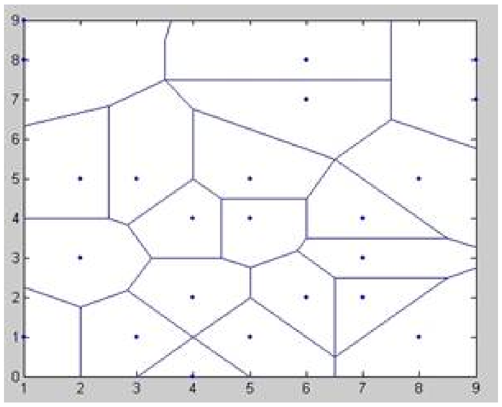
Introduction to Voronoi Diagrams

Keywords: voronoi, diagram, pathplanning, tutorial



The photo depicts an example of a voronoi diagram which displays a set of cells specified by a set of points. Any position within a specific cell is closer to the point which generated the cell than any other point in the diagram. The voronoi diagram can be very useful in robot path planning. If the points represent obstacles in a space, then restricting a robot to traverse the edges created by the voronoi diagram will insure that it is the maximum distance away from the nearest surrounding points at all times. This tutorial shows you how to create a voronoi diagram and introduces you to some other common methods used to create these diagrams. The tutorial takes approximately 1 hour to complete.

Motivation and Audience

This tutorial's motivation is to give an overview of Voronoi diagrams, demonstrates how to create one using matlab, and introduces the reader to other methods and programs used for the creation of these diagrams. This tutorial assumes the reader has the following background and interests:

- ***Access to Matlab***
- ***Familiar with programming in Matlab***
- ***Interested in robotic Path planning***

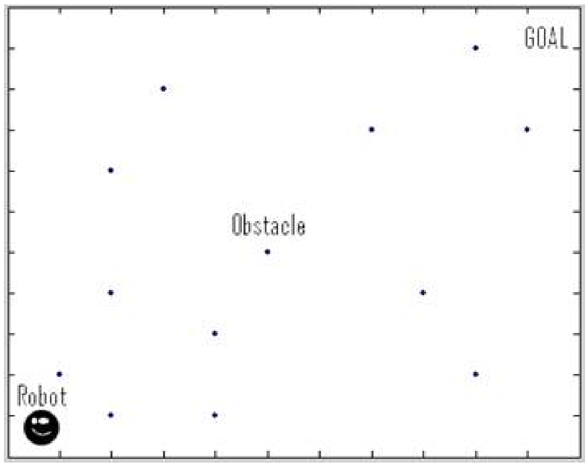
The rest of the tutorial is presented as follows:

- Introduction to Voronoi Diagrams
- Construction using a Geometric Construction algorithm and MATLAB
- Introduction to other methods used for Voronoi Diagram construction
- Final Words and Exercises

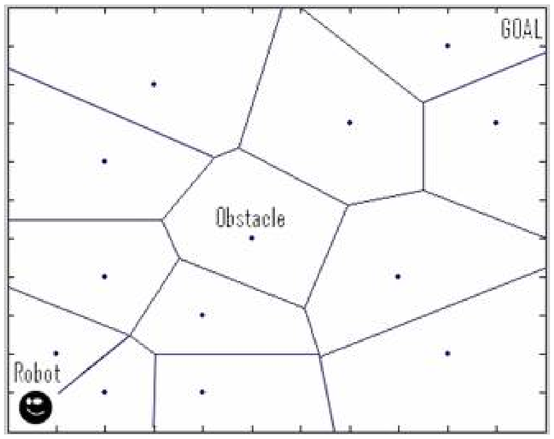
A Brief Introduction to Voronoi Diagrams

The Voronoi diagram itself is named after Greogry Voronoi who was a German mathematician. In 1908, Voronoi formalized the n dimensional case for the concept by which we now know as Voronoi diagrams. In short, a voronoi diagram records information about the distances between sets of points in any dimensional space. For path planning, voronoi tends to be used in two dimensional space, where sets of points all lie within a plane. As seen from the figure above, a plane is divided into cells so that each cell contains exactly one site. For every point in the cell, the Euclidean distance of the point to the site within the cell, must be smaller than the distance of that point to any other site in the plane. If this rule is followed across the entire plane, then the boundaries of the cells, known as Voronoi edges, will represent points equidistance from the nearest 2 sites. The point where multiple boundaries meet, called a voronoi vertex, is equidistance from its 3 nearest sites.

As you can imagine, there are many possible applications for the voronoi diagrams: nearest neighbor queries for data structure problems in computational geometry, computational morphology such as modeling how fire spreads and crystals grow, and also business applications such as determining where to locate a store so it is no closer to any existing store of its kind. As stated earlier, voronoi diagrams can also be used for path planning. Imagine you have a robot that needs to move through a cluttered room without hitting any objects.



The safest route would have the robot move in a path such that it is as far away from the nearest obstacles as possible. In order to generate this route, you would need to first generate the voronoi diagram for the room to obtain the voronoi edges which represent the maximum distance between the nearest objects.

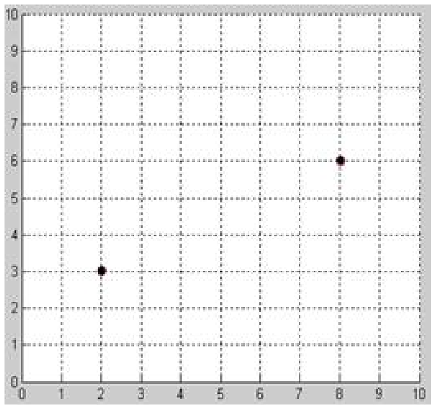


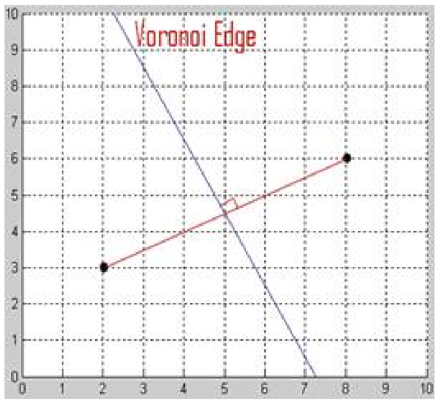
The next step is to then follow the associated voronoi edges to the end goal. Finding the optimal path, or even any path to the goal itself requires using search algorithms which I describe in my other tutorials. This tutorial concentrates on the first portion of this task; generating the voronoi diagram.

Construction of a Voronoi Diagram

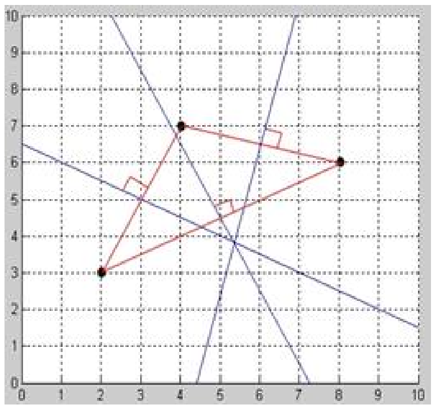
There are many approaches to constructing Voronoi diagrams. Some methods are more efficient in terms of time than others. The method I will be describing is a simple geometric construction algorithm using perpendicular bisectors. In terms of time efficiency, it is VERY inefficient but it is easy to understand and helps to solidify the understanding of voronoi diagrams. In the later sections of this tutorial, I will give a brief introduction into some more common and MUCH MORE efficient algorithms.

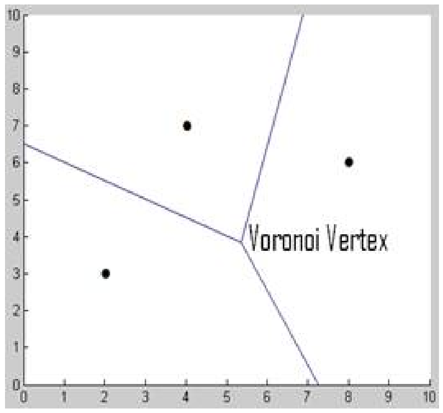
We know that a voronoi edge represents points that are equidistant from the nearest two sites. So let's start with just two points on the graph as seen in figures below. The voronoi edge for these two points would be the perpendicular bisector of the segment joining the two sites.





If you add a third point to the diagram, you must calculate the perpendicular bisector between each of the three points: point1 to point 2, point 1 to point 3, and point 2 to point 3. The next step is to remove each section of the perpendicular bisector lines that violate the equidistant rule. You should now be left with the voronoi diagram for the three sites as seen in figure below.





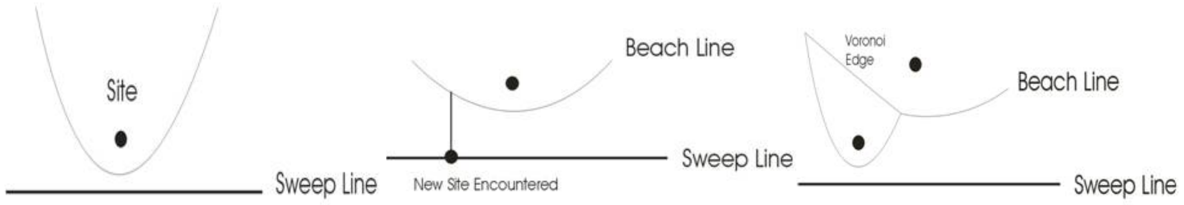
Adding yet another point to the diagram requires following the same method described above. As you might be able to tell, as the number of sites increase, the time it takes to compute the voronoi diagram using this method increases significantly. You can download my Matlab program "voronoitut.m", found in the download section of this tutorial, that successfully generates the voronoi diagram for three points using the method just described. In the code you will find text fully describing what each portion of the code does. Although when run, the code allows for entering "n" number of points, I have not worked out the bugs for generating a correct voronoi diagram over 3 points. The purpose of my code is only to demonstrate the method described above.

If interested, the reader can easily use the Matlab function "voronoi.m". It is called by using the prototype VORONOI(X,Y), which then plots the Voronoi diagram for the points X,Y. X and Y can also be an array of points. The figure at the top of the page was generated using the voronoi function in Matlab.

Introduction to Other Methods for Generating Voronoi Diagrams

Outside of the intersecting perpendicular bisector half planes approach, there are a number of other algorithms that are more commonly used for generating voronoi diagrams because they are much more time efficient. There are other incremental construction algorithms similar to the method shown above. There are divide and conquer algorithms that divide the problem into subsets of sites which decreases the problem size. Each subset of sites has its own voronoi diagram from which each voronoi diagram is then merged with the other subset voronoi diagrams. At the merge sections, extra computation has to go into recalculating the voronoi edges and vertices. Steven Fortune developed a plane sweep algorithm called the "fortune's algorithm" that uses a sweep line and a beach line to generate the diagrams.

Fortunes algorithm is of interest in particular because it is very efficient and conceptually easy to understand. The algorithm uses a sweep line and a beach line. As the sweep line crosses sites on the plane, a beach line is generated which represents the voronoi boundary between the sites passed by the sweep line and the sweep line itself. Remember, the bisector between two points is a straight line but the bisector between a point and a line is a parabola. The beach line is made of these parabolas and the voronoi diagram is completely correct up to the beach line. As the sweep line crosses another site, a vertical edge (very narrow parabola) connects the point to the beach line above it. As the sweep line continues farther, the vertical edge opens up to a wider and wider parabola. An existing arc of a beach line will eventually shrink to a point and create a voronoi vertex. When the sweep line has passed by the entire plane, the voronoi diagram will be complete. The C program files that implement this code can be found in the download section of this tutorial.



(Note: These figures were not drawn to scale)

Final Words

While voronoi diagrams are applicable to three or more dimensions, for path planning in general, working with voronoi diagrams in a two dimensional plane offers a quick visualization of the paths that are the farthest away from any of the nearest two or three obstacles. As stated earlier, after the voronoi edges have been found, the next step is to generate a path of a series of voronoi edges for the robot to follow. Parts of this next step can be found in my other tutorials (ie. Breadth first search, Depth first search etc.).

Downloadable Files from this Tutorial

voronoitutorial.zip
FortuneAlgorithmcCode.zip

References

Mumm, Michael. “Voronoi Diagrams”. TMME Vol1, no2, p.44-55
http://en.wikipedia.org/wiki/Voronoi
Lecture Notes CMSC 754 (www.math.gatech.edu/~randall/AlgsF06/mount.pdf)
http://mapviewer.skynet.ie/docs/Voronoi_Diagram_Slides.ppt
http://netlib.bell-labs.com/who/sjf/
Blaer, Paul. “Robot Path Planning Using Generalized Voronoi Diagrams” (http://www.cs.columbia.edu/~pblaer/projects/path_planner/voronoi.html

If you have any questions or comments about this tutorial, please email me at jth23@drexel.edu