## An Algorithm for Vectorising Spatial Maps based on the Sweep-Line Voronoi Algorithm – Shane O'Sullivan

A method of Vectorising a spatial map of points has been developed as part of the MapViewer (http://mapviewer.skynet.ie) project that uses a filtering techniques and the sweep line Voronoi algorithm by Stephan Fortune to convert a grey-scale map into a series of lines.

This can have many practical applications, such as reducing the dimensionality of a map, as well as adding orientation information to the map instead of it simply consisting of individual pixels.

For example, the first picture below is a map of an indoor environment in grid format, where white areas represent free space, black areas represent space occupied by obstacles, and blue represents areas about which there is no information.

Fig 1

After the Vectorising algorithm has been applied to it, we are left with a vector map such as the one in the picture below. All areas sufficiently dark (the darker they are the more likely they are to have an obstacle in them) have had a line fitted to them, representing the fact that there is an obstacle present.

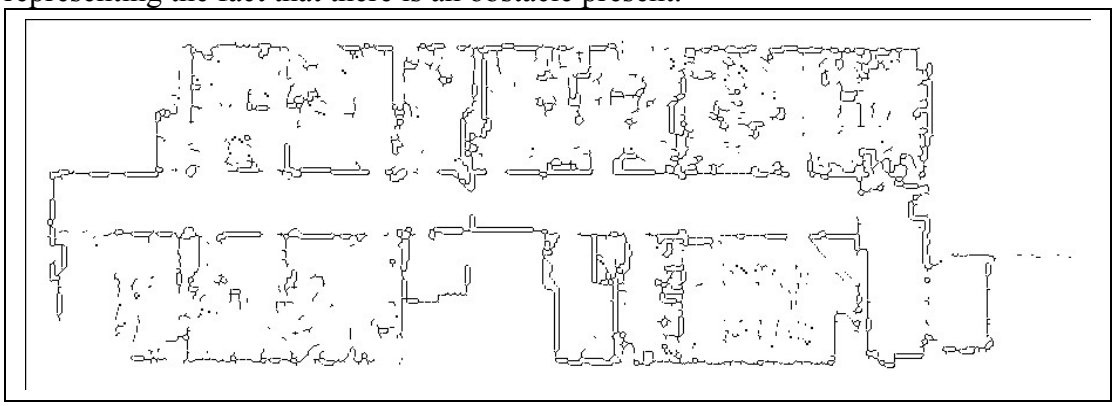

Fig 2

Each of the lines in the vector map has both a length and orientation related to it, so there is now much more information about the environment than before. It can also be manipulated as an entity, which the group of cells in the grid map could not.

**Filtering the map**

The first step in performing this transformation is to decide how a group of cells is to be represented by lines in the vector map. If no filtering is performed, then a Voronoi algorithm will not give an accurate representation of the obstacle. For example, in Fig 3, we see a square block with a Voronoi diagram created in it in blue.

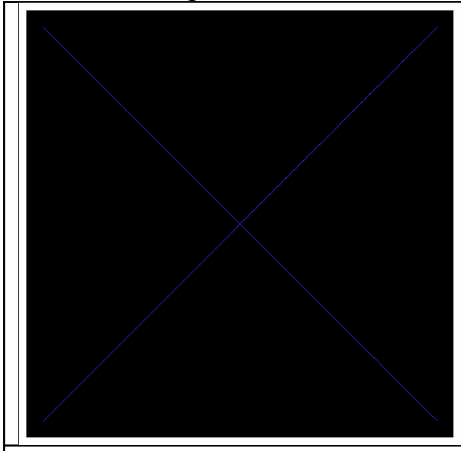

Fig 3

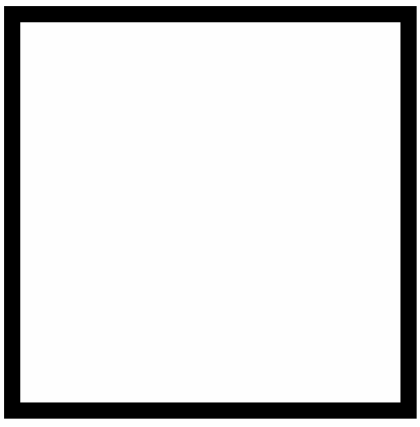

Fig 4

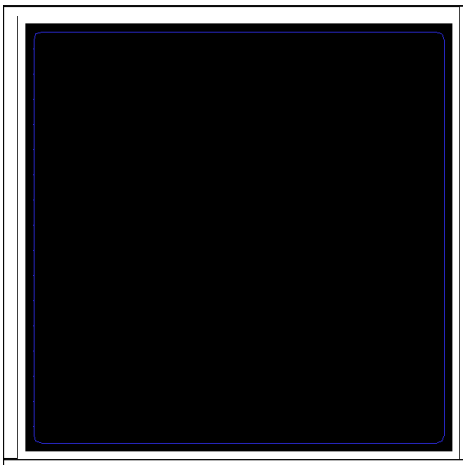

Fig 5

The approach taken in this algorithm is to filter out any occupied cells that are completely surrounded by other occupied cells, leaving an object similar to that in Fig 4. The resulting Voronoi Diagram appears as in Fig 5. This can be done in O(n) time, and results in vectors that circumscribe objects.

The use that the Voronoi algorithm is put to here is different to the most common usage. Usually, Voronoi diagrams define the free space in a map, and contains all of the points in free space that are equidistant from their two closest occupied points (or basis points). To vectorise a map however, the Voronoi diagram must represent the occupied areas of the map. For this reason, *rather than using the occupied points of the map to generate the Voronoi diagram, the unoccupied points of the map must be used.*

**Reducing the dimensionality of the problem**

The sweep line algorithm designed by Stephan Fortune is very fast at calculating Voronoi Diagrams, and runs in O(*nlogn*) time. However, to achieve this it makes liberal use of hash maps, which are quite expensive with regards to memory. Therefore it is desirable to reduce the dimensionality of the problem as much as possible.

We observe that only those unoccupied cells that actually touch occupied cells are of interest to us. This is because the Voronoi algorithm is interested in only the *nearest* unoccupied cell, and all unoccupied cells that do *not* touch an occupied cell cannot be the closest one.

For example, taking the block from Fig 3 again, the only unoccupied (non-black) cells that will be passed to the Voronoi algorithm for computation are the light-grey cells in Fig 6.

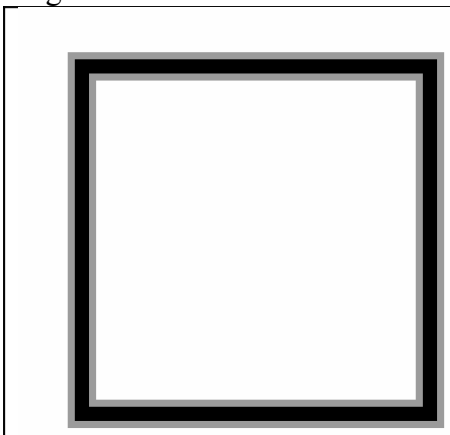

Fig 6

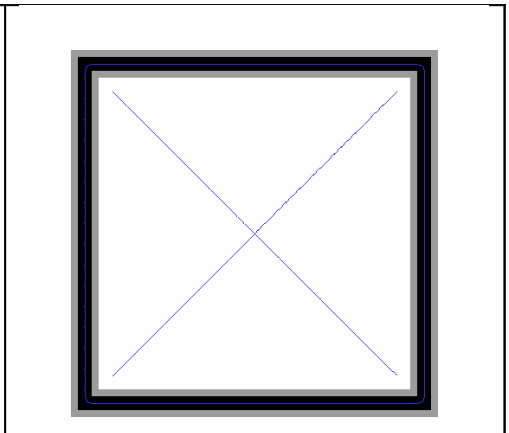

Fig 7

Unfortunately this does lead to a lot of incorrect vectors being created, as in Fig 7, but these can be quickly and easily removed at the end. For sparse maps, with mostly unoccupied space, this technique vastly reduces both the memory and computation requirements of the algorithm.

**Generating the Voronoi Diagram**

Once the above filtering steps have been performed, it is time to pass a list of Cartesian coordinates to the Voronoi algorithm. An implementation of this algorithm is available on the MapViewer website at http://mapviewer.skynet.ie .

The exact workings of the sweep-line algorithm and theory behind it have been very well explained by Stephan Fortune, and a simple search on Google should find the correct paper. Alternatively, two of his papers on the subject are hosted on the MapViewer website, along with other information regarding the algorithm.

The implementation of the algorithm hosted on the MapViewer site is an altered version of C code that Fortune gives away on his website. While his code is extremely good at what it does, it is not very accessible as it simply prints information to the screen, is more or less uncommented, and suffers from extremely severe memory leak issues. I fixed up the memory leaks, encapsulated the code in a C++ class, and provided storage and accessor methods to retrieve the line segments of the Voronoi diagram. I also added some comments here and there. However, the basic algorithm is unchanged, and is as described in his publications.

**Filtering out incorrect vertices generated as a result of performance improvements**

As mentioned earlier, by reducing the dimensionality of the problem, erroneous vertices would be created. These can be filtered out by checking both either endpoint of the line against the map that resulted from the first filtering pass, i.e. the map in Fig 4. If the endpoint does *not* land on an occupied cell, then the vector should be discarded. Otherwise it is kept, and becomes part of the vector map.

**Reducing the Dimensionality of the Vector Map**

One of the disadvantages of using a Voronoi algorithm to convert a grid-based map to a vector based map is that it often tends to break up continuous lines into many smaller lines, even though all the small lines have the same slope and all join together. However, this is relatively easily countered. Below I detail how this is achieved in MapViewer. Only horizontal and vertical lines are joined together, since practice has shown that it is very rare for other lines to be joined together.

1. Create a two-dimensional array the same size as the map. Each element of the array contains a pointer to a list (each element of the list is a line), which initially is initialised to 0, i.e. no list exists there.
2. Cycle through the lines in the Voronoi diagram. As each one is encountered, if it is either horizontal or vertical push it onto the list in the position of the map where it's two end points land. Otherwise store it separately in another list. If no list exists in that element of the array, create one. This means that each line is pushed onto two lists. For example, if the end points of the line were (2.6,3.7) and (15.3, 78.9), it would be pushed onto the lists in the array positions [2][3] and [15][78].
3. As the line is being pushed onto list in the array, cycle through the lines in that list (this isn't very expensive, there's probably max 3 lines there). If either of the two points in a line already in the list is the same as the line being pushed, and the slope of both lines is equal, then remove both lines from all lists, join them together, and push the joined line onto the array.
4. Finally, once all lines have been concatenated, step through the array of lists. Copy each line into the final list. Note that every line has been stored twice, once for each end point, so as you copy the line from the array list, make sure to delete it's corresponding entry for it's other point. Also, copy all the non-horizontal, non-vertical lines from the list they were pushed on in step 2 onto the final list.

Lo and behold, you have a Vector Map!

If you can't be bothered to implement this yourself, and would rather just use it, download MapViewer from http://mapviewer.skynet.ie and load a map into it. Then click "Tools/Convert Grid Map To Vector Map With Line Fitting". The map can be in a number of formats, a number to do with various robot simulators, as well as Bitmap (BMP) and JPEG. See the site for more information.