Rhodes University

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

Creating and Optimizing a Sky Tessellation Algorithm for Direction-Dependent Effects

Antonio Bradley Peters

supervised by

Prof. Karen Bradshaw

Prof. Denis Pollney

project originated by

Dr. Cyril Tasse & Prof. Oleg Smirnov

October 13, 2016

Contents

Chapter 1

WIP

1.1 Source Selection

The system is designed to take in three parameters, a list of all source points in the space, the dimensions of the space the points lie in and the intensity threshold. Each element of the list of sources has 3 parameters: x, y and z. The x and y parameters define the spatial coordinates of the source while the z parameter is seen as the intensity of the source, the list of n sources are passed into the function as an $n \times 3$ numpy array of doubles. The spatial dimension parameter takes in a tuple of two values, the width and height of the space the sources are in. The intensity threshold is a single double value which defines the minimum intensity a source needs to be a voronoi cell centre.

The sources are read in and are used to generate the voronoi centres as a points. Each point inherits the x, y and z values of the source and the rest are kept as default values. Once the list of points is created, it is sorted by the x then the y values of the sources. Once the centre points are generated, generation the voronoi diagram can begin.

For the sake of testing the system the coordinates are randomly generated on a 600×600 plane with the intensity randomly generated as the absolute value of a random normal distribution, centred at zero with a standard deviation of 3000. Using this, 10000 sources are randomly generated. From these sources, we generate centres to be used by the Voronoi, centres are sources with an intensity greater than one standard deviation of the original mean, since the absolute value of the source intensity is used, this is all points with an intensity greater than 3000, this accounts for approximately 32% of the sources.

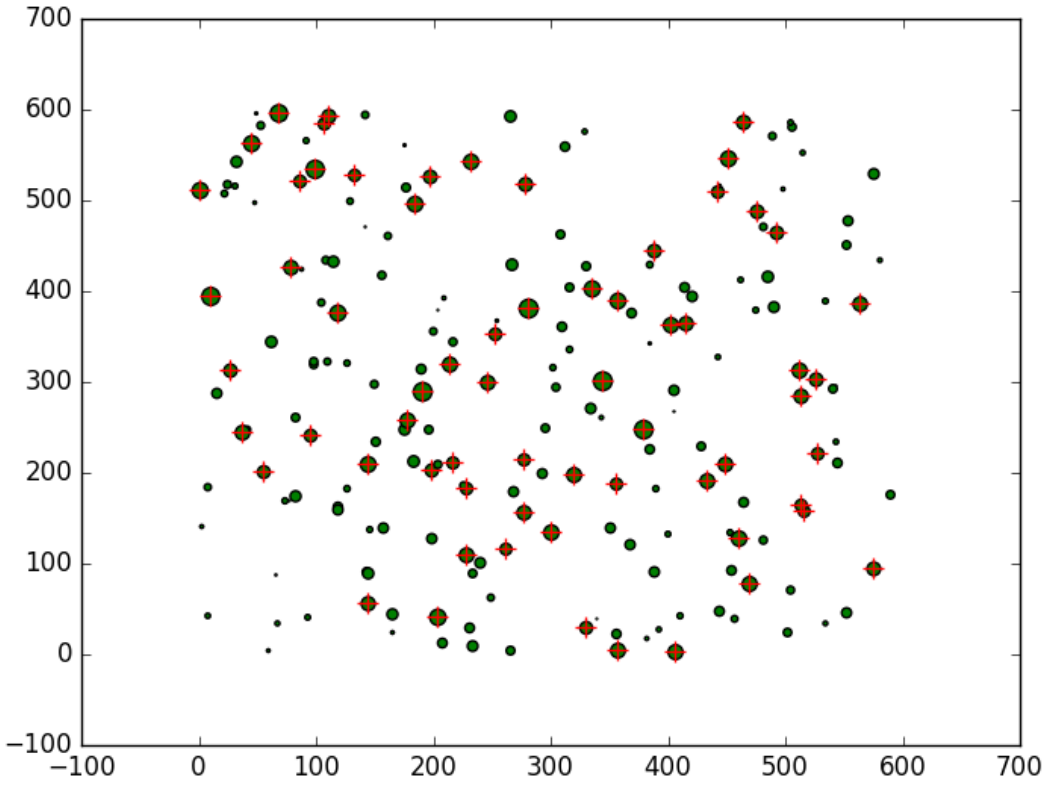


Figure 1.1: Sources and selected centres.

Figure 1.1 shows sources in green with their intensities representing the radius of the on the plane, to simplify, only 200 points are generated. The crosses in red represent the centres that will be used to generate the Voronoi tessellation.

1.2 Voronoi

The current best model uses k-means clustering of of the points to the center points. For every point (the plane is seen as a finite number of pixels) in the plane, p, it finds the seed point, s_i, which is the shortest distance to it, i.e. such that $\sqrt{\big(p(x) - s_i(x)\big)^2 + \big(p(y) - s_i(y)\big)^2}$ is minimum. This is suboptimal as the time taken to create the diagram relies on the dimensions of the plane and the number of seed points in it. Instead we seek a method which is invariant of the plane size and relies solely on the seed points, for the sake of increasing the computation time, this method must also be parallelizable.

It was therefore decided that the divide and conquer method (Chapter **??**) would be used. The algorithm works by ordering the points, first by their x then their y values. The points are then divided into two subsets, a left and right set. The Voronoi diagrams are then generated for the left and right subsets using the divide and conquer method. The convex hulls of the left and right Voronoi diagrams are then found. The lowest common support line between the hulls is then found and from this a dividing polygonal chain is generated until it intercepts the upper bounds of the plane. The intersecting edges with the polygonal chain are then determined, and cut so that part of the chain is now part of the Voronoi cells edges.

Code for the Divide and Conquer was adapted from that in the git repository pyVoronoi[1]. pyVoronoi is an implementation for the Divide and Conquer Voronoi algorithm written in python 2.7. pyVoronoi uses a simple GUI to generate a voronoi diagram in a fixed plane using sources read in from a text file. This interface, while simple to use, is constraining as it does not allow the user to specify their own spacial constrains or generate the voronoi as part of a pipelined process. The visualiser and interface were therefore removed and the remaining code re-factored so that the process is a function which may be called by the user or used in a larger process.

1.2.1 Structures of the Voronoi

The voronoi structure is mainly made up of two structures, lines and points. Points are mainly used to define the sources, but are also used to generate line. Points are made up of an x, y and z value, which determines the position and intensity of the point. Each point also has a circumcentre attribute to determine whether it is the point of intercept of three lines (this is most commonly used in the points at the end of a line), two point parameters to indicate the points on either side of the point if the point lies on the convex hull, these are initialised as None, a list of related structures containing another point and their corresponding bisecting line, this list is set as empty. The point also has a list of all the sources contained in the cell and an error value associated to it, these values are set to an empty list and zero respectively and will only come into use after the voronoi diagram has been generated.

A line is made up of two points which define its endpoints, if the line is a bisector it

[1] https://github.com/twmht/pyVoronoi

contains two more points which define the source points used to generate it, if not these values are set to None. The line also contains a list of all the other lines the line is connected to as well. The line finally has an availability boolean parameter which defines whether or not the line is actively available to intercept with other lines or if it has been discarded from the overall voronoi structure.

1.2.2 Voronoi Function

The Voronoi function takes in the list of points and a range of points it will operate on (initially the entire set of points). Depending on the number of points in the subset range, one of four operations will occur. If the range is made up of more than three points, the range is divided into two equal sub ranges, one from the starting point of the range to the median and another from the point after the median to the end of the range. The Voronoi function is then called again for each sub range, denoted as VDL and VDR for left and right sub ranges respectively, with the full set of points. Once the voronoi structure for these two are calculated, the merge function is called with VDL and VDR and the function ends as it is not value returning.

If the range is three, the function generates three lines, one for the each pair of the three point. The intercepts of the lines is determined, if it does not exist, an invalid line exists, this line is found and removed. If the intercept does exist, it is found and the lines are clipped at the intercept. The lines are listed and returned with the range and the corresponding convex hull structure made up of all three points.
For a range of two points, the bisecting line is determined as the only line in the structure. The line is placed in a list where it is the only element and along with the point range and the convex hull, only made up of the two points, are returned.
The case of a single point is excluded as the first case of multiple points being divided in two equal or near equal sets and the case of three and two points in a set will make the case of a single point impossible.

1.2.3 Convex Hull

Andrew's monotone chain convex hull algorithm was used to determine the convex hull of a set of points. The function is first passed a range of points on which to operate. The points must be ordered lexicographically (first by the x coordinate then by y if there is a

tie), which they are as this is needed for generating the voronoi structure. A list of zeros, twice the size of the number of points in the range is generated, this list will hold the elements of the chain. The complete convex hull is calculated in two steps, the first finds the upper hull and the second, the lower hull.
The first point (the leftmost) and the second point in the range are added to the list. We then iterate over the rest of the range, adding the next point to our list, if the angle created by these three points is less than π radians, the points are left in the list, the next point in the range is added and the three latest points in the list are analysed for their angle again until we reach the. If the angle is greater than π radians, the two latest points in the list are removed and the process continues. This generates the upper hull of the convex hull.
To generate the lower hull, the same process is used, but the range is iterated over in reverse, starting at the rightmost point and ending at the leftmost.
One the list is populated with the points of the lower and upper hulls, they form the monotone chain. The chain is then iterated over and each point in the chain adds the preceding point to its fifth parameter and the next point to its fourth.

1.2.4 Voronoi Merge Function

The merge begins by finding the upper and lower tangents of the VDL and VDR sets. These tangent lines are defined as connections between the convex hulls of VDL and VDR with a point in each as the endpoints of the line. Starting from the rightmost point in VDL and the leftmost in VDR, it finds the upper tangent by finding the pair of points (one in VDL and one in VDR) who, together with their respective adjacent points, both generate an angle less than π radians, the order and the adjacent point being used is counter-clockwise for VDL and clockwise for VDR. To find the lower tangent, the order is reversed with VDL stepping clockwise and VDR counter-clockwise.
The upper tangent is used to find the starting point of the polygonal chain used to merge the voronoi. Bisector of the endpoints of the line, which both lie on convex hulls, are used as the first line segment in the chain which is appended to a list of lines called HP. The uppermost of the two points in the upper tangent is determined and a new tangent is generated with the lower point and the uppermost point's neighbour, not necessarily in the convex hull, so long as it is not a convex hull neighbour of the lower point and it's related line to the upper point is both available and intercepts the bisector of the upper tangent. The bisecting line of the new point which intercepts the new bisector along with the new bisector itself is added to a list of lines to be clipped, clip_lines. The same set of

operations then occur with the new point and the lower of the upper tangent defining the new tangent. This continues until the bisecting line of the lower tangent is determined. Once complete, we are left with a list, HP, which contains the bisecting lines between points in VDL and VDR and a list of bisecting lines from VDL and VDR together with lines from HP, clip_lines. The lines in clip_lines and HP are both cut at their intercepts and all are appended to a list of lines to be passed back along with the range of points and the new convex hull of the combined voronoi structure.

An Example of the Voronoi Tessellation can be seen in Figure 1.2

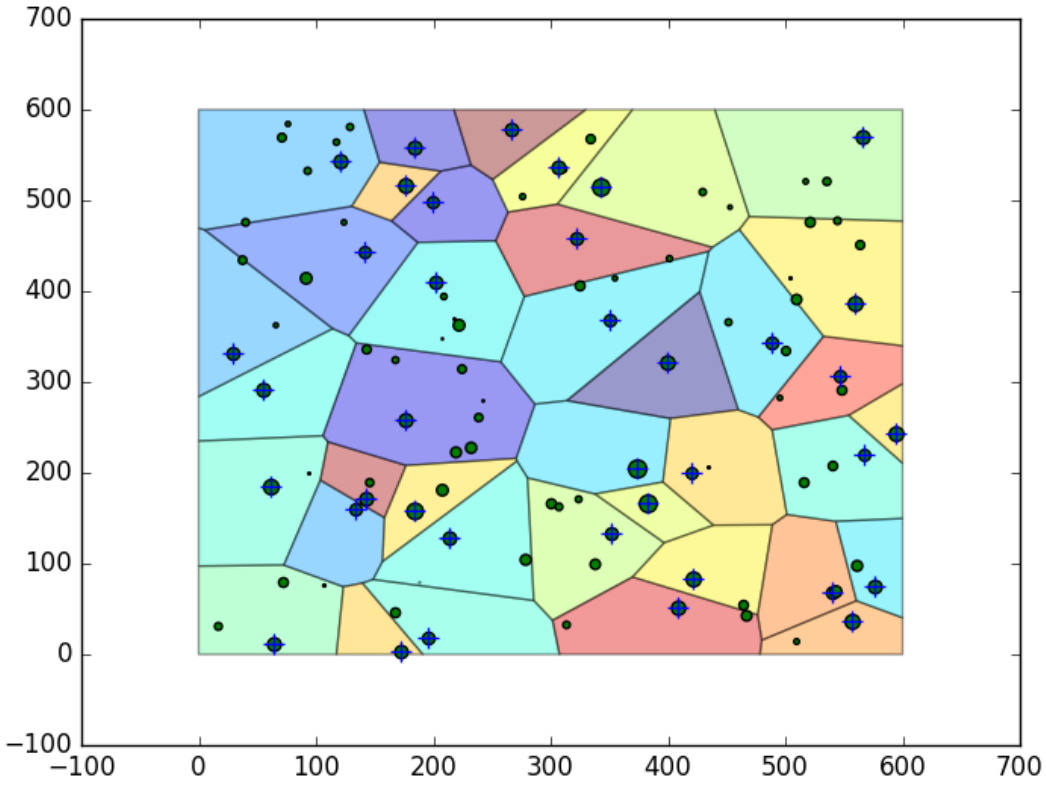


Figure 1.2: A Generated Voronoi Tessellation.

1.2.5 Weighted Voronoi Tessellation

An attempt was made to generate a weighted Voronoi tessellation using a distance transform which uses the intensities of the centres to redetermine the coordinates of the midpoint of the centres, this transform can be seen in equation 1.1 where z_1 and z_2 are the

intensities of $\vec{x_1}$ and $\vec{x_2}$ respectively.

$$d'(\vec{x_1}, \vec{x_2}) = \frac{z_1}{z_1 + z_2}\vec{x_1} + \frac{z_2}{z_1 + z_2}\vec{x_2} \tag{1.1}$$

This is an extension of the standard distance equation since, if $z_1 = z_2$ we obtain

$$\begin{aligned} d'(\vec{x_1}, \vec{x_2}) &= \frac{z_1}{z_1 + z_2}\vec{x_1} + \frac{z_2}{z_1 + z_2}\vec{x_2} \\ d'(\vec{x_1}, \vec{x_2}) &= \frac{1}{2}\vec{x_1} + \frac{1}{2}\vec{x_2} \\ d'(\vec{x_1}, \vec{x_2}) &= \frac{\vec{x_1} + \vec{x_2}}{2} \end{aligned}$$

Some complications were found in this, namely on the convex hull. Weighted centres which are not on the hull but due to their larger weighting still have cells which dominate areas of the hull. This leads to them not being included in the merging process and their line segments not being clipped or deactivated. Another problem with this is that it generates undefined regions in the space, regions where domains overlap due conflicting weightings, an example of this can be seen in Figure 1.3.

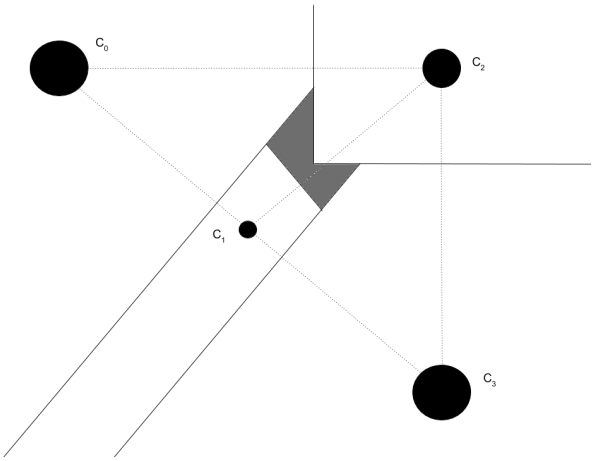


Figure 1.3: An unclassified area (grey) generated by a weighting conflict in c_1 and c_2.

It shows a conflicting area between c_1 and c_2, it should be expected that c_2, with its higher intensity, should claim the area, but in doing so it will cross into what then should be the domain of c_0 or c_3. It was for this reason that the choice to use the intensities to weight the Voronoi tessellation generated was abandoned and that the intensities would rather be used for calculating the error and calculating cell merges. An example of a failed weighted Voronoi tessellation can be seen in Figure 1.4.

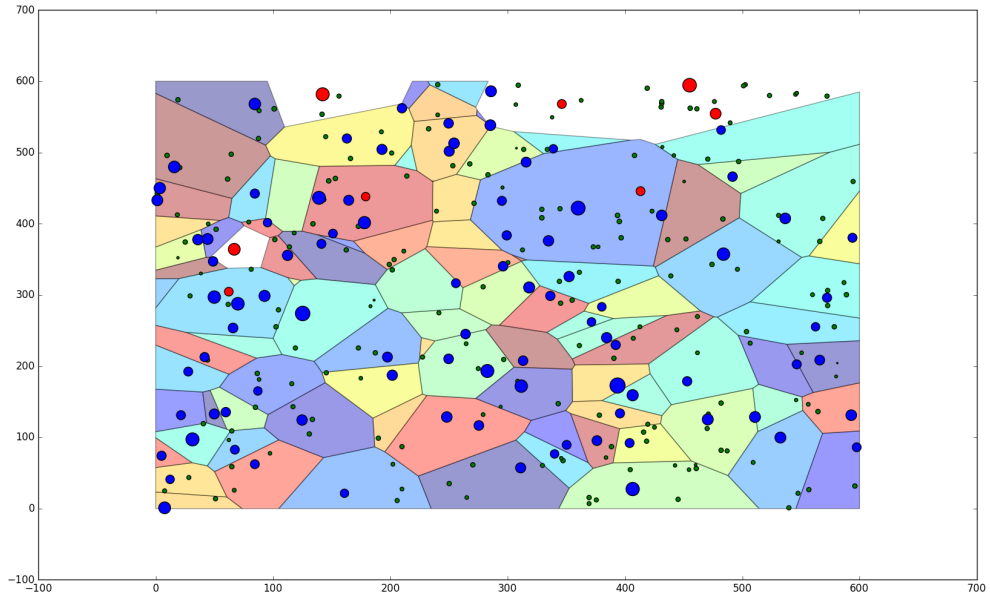


Figure 1.4: A failed visualisation of a weighted Voronoi tessellation.

1.3 Cell Error

Once the Voronoi tessellation is determined, it is re-centred based on the weighted average of the points in the cell. Sources are added to cells by determining the cell centre which is closest to it using the standard distance equation:

$$d = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2}, \tag{1.2}$$

where (x_i, y_i) is the location of a source in the plane and (x_c, y_c) is the location of a centre. The closest centre is such that d is minimum. This is done to add the influence of weaker sources in overall correction. This is especially necessary when the cell is generated by a source slightly above the intensity threshold and contains a source slightly below the threshold. We seek a new weighted centre such that the error for a cell is minimum. The error for a cell containing N sources is defined as

$$\epsilon = \sum_{i=0}^{N} z_i |\vec{r_i} - \vec{r_c}|^2, \tag{1.3}$$

with $\vec{r_i} = (x_i, y_i)$ is the location and z_i is the intensity of some source in the cell and $\vec{r_c}$ as the location of a new centre.
This error function will have a local minima at the point where its derivative with regards to $\vec{r_c}$ is zero, or

$$\begin{aligned}\frac{d\epsilon}{d\vec{r_c}} &= \sum_{i=0}^{N} \frac{d}{d\vec{r_c}} z_i(|\vec{r_i}|^2 - 2\vec{r_i} \cdot \vec{r_c} + |\vec{r_c}|^2) \\ &= \sum_{i=0}^{N} z_i(2\vec{r_i} - 2\vec{r_c}) = 0\end{aligned}$$

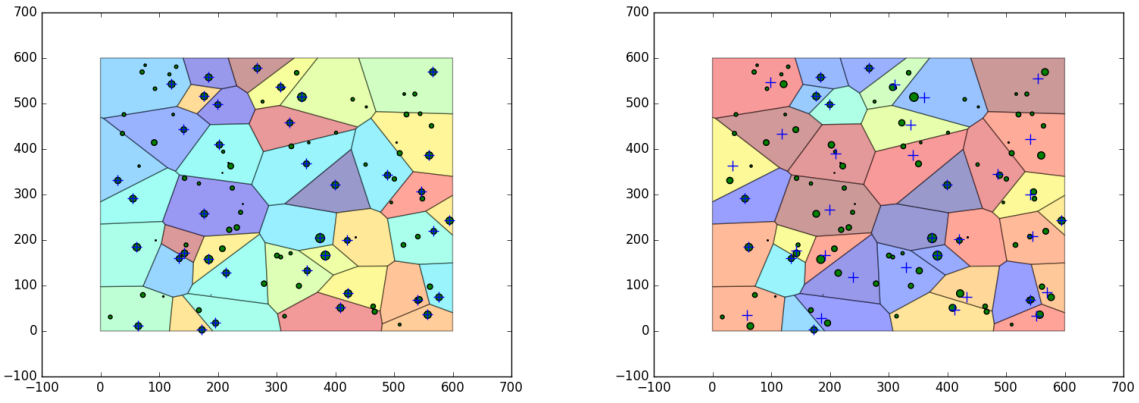
or

$$2\sum_{i=0}^{N} z_i\vec{r_i} = 2\sum_{i=0}^{N} z_i\vec{r_c}$$

Since $\vec{r_c}$ is not dependant on the sum, it can be removed and the equation reordered to give

$$\vec{r_c} = \frac{\sum_{i=0}^{N} z_i\vec{r_i}}{\sum_{i=0}^{N} z_i} \tag{1.4}$$

From this, the new centre is determined, since it is required for the cell merge, the new centre retains the old centres intensity value i.e. the highest intensity from a source in the cell. Once the cell's new centre is obtained, its error is calculated using Equation 1.3. An example of a corrected centre can be seen in Figure 1.5.



(a) Highest intensity centre.

(b) Weighted average centre.

Figure 1.5: Changing the centre of a Voronoi cell.

1.4 Cell Merge

The merge process is iterative and runs dependant on the total sum of the error of the tessellation. Initially, this error is relatively low as a multiple cells are generated with one or very few sources contained in it. In order to keep the maximum error threshold relative, unless it is given as an input by the user, it is calculated as the product of the set standard deviation, the size of the plane and the number of sources in the plane. The process begins by summing the errors of the cells and then goes through an iterative process of finding the best merge, checking if implementing the best merge exceeds the maximum error threshold and, if not, implementing the best merge.

1.4.1 Obtaining the best Merge

The best merge is obtained by iterating over the list of points and, for each point, testing its active neighbouring points.

The merge test works by determining a new centre, determined by two existing centres, $\vec{p_1}$ and $\vec{p_2}$, with intensities z_1 and z_2 as

$$P = t\vec{p_1} + (1-t)\vec{p_2} \text{ with } t = \frac{z_1}{z_1 + z_2} \tag{1.5}$$

the new weight for the merged centre is now defined as

$$Z = z_1 + z_2 \tag{1.6}$$

Since p_1 and p_2 are centred sums of the positions of the sources in their cells, by expanding them to their original forms

$$\vec{p_1} = \frac{\sum_{i=0}^{N} z_{1i}\vec{x_{1i}}}{\sum_{i=0}^{N} z_{1i}} \text{ and } \vec{p_2} = \frac{\sum_{i=0}^{M} z_{2i}\vec{x_{2i}}}{\sum_{i=0}^{M} z_{2i}}$$

and by noting that

$$z_j = \sum_{i=0}^{N} z_{ji}$$

Substituting these into equation 1.5, we obtain

$$\begin{aligned}
P &= t\vec{p_1} + (1-t)\vec{p_2} \\
&= \frac{z_1}{z_1+z_2}\frac{\sum_{i=0}^{N} z_{1i}\vec{x_{1i}}}{z_1} + (\frac{z_1+z_2}{z_1+z_2} - \frac{z_1}{z_1+z_2})\frac{\sum_{i=0}^{M} z_{2i}\vec{x_{2i}}}{z_2} \\
&= \frac{\sum_{i=0}^{N} z_{1i}\vec{x_{1i}}}{z_1+z_2} + \frac{\sum_{i=0}^{M} z_{2i}\vec{x_{2i}}}{z_1+z_2} \\
&= \frac{\sum_{i=0}^{N} z_{1i}\vec{x_{1i}} + \sum_{i=0}^{M} z_{2i}\vec{x_{2i}}}{z_1+z_2} \\
&= \frac{\sum_{i=0}^{N} z_{1i}\vec{x_{1i}} + \sum_{i=0}^{M} z_{2i}\vec{x_{2i}}}{\sum_{i=0}^{N} z_{1i} + \sum_{i=0}^{M} z_{2i}}
\end{aligned}$$

This shows that equation 1.5 is equivalent to finding the centre of all the sources in both p_1 and p_2.

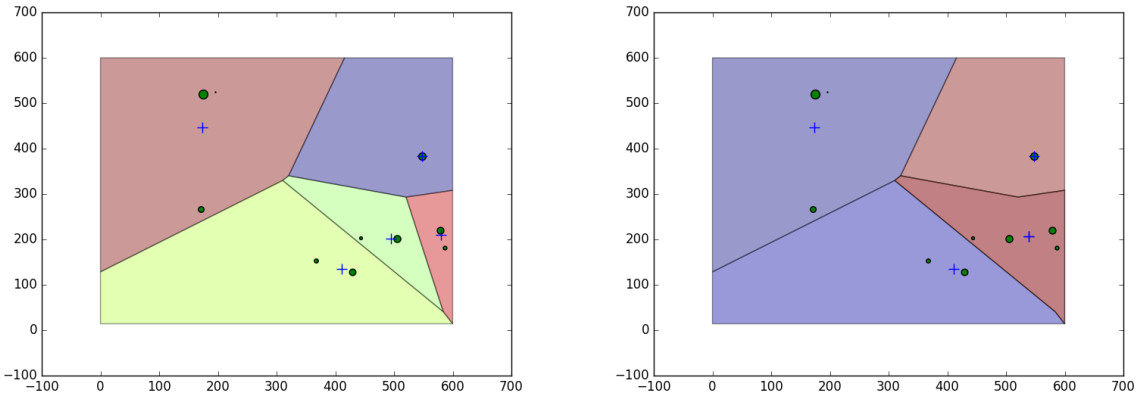
Once the new centre is found, the error must be determined, this is again found by summing the square of the weighted distances to the new centre from each source. Once determined, the new centres coordinates and intensity, as well as the new error are returned. The error itself is not compared to find the best merge, but rather the change in error, that is, the merge which produces the lowest increase in the error, this is done by taking the tested merge error and subtracting from it the errors of the centres used to generate it. If it is lower, it is then stored along with the new coordinates and the centres used to generate it.

Once the best merge is found, it is determined if the new sum of errors will exceed the threshold, if it does, the merge process is halted and the structure returned as the best possible merge. If the threshold is set too high, it may occur that all the cells merge into a single cell, in this case, the process is again halted as no best merge could be found as the threshold is set too high for the system and the structure is returned as only the set of points with no active lines left and so no merged Voronoi can be generated. If the system finds a valid merge which is still under the threshold, it adds the difference in the merge error to the total error and executes the merge.

1.4.2 Executing the Merge

The merge execution algorithm takes in the new coordinates and intensity, P, the new error, as well as the centre, p_1, and its related centre with which it is to be merged, p_2.

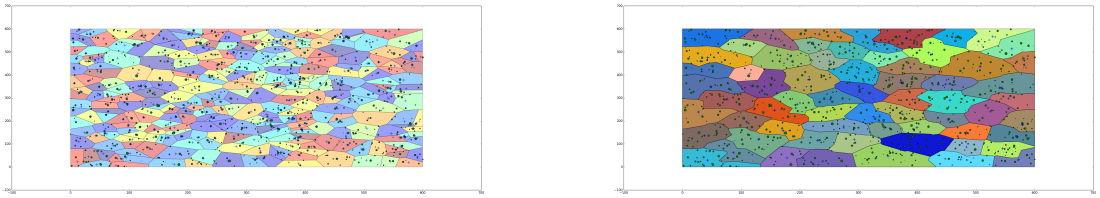
It starts by setting the coordinates and intensity of p_1 to those of P. It then sets active status the line which relates p_1 to p_2 to false and appends the list of sources in p_2 to that of p_1. The error of p_1 is then set to that of the new error and p_2 and it's list of consumed centres is added to the list of consumed centres of p_1. Finally, the list of centres is iterated over and any centre which references p_2 is changed to reference p_1 and all lines which relate other centres to p_2 are changed to relate to p_1 instead. Once this is complete, the process of finding the best merge begins again until the threshold is reached.



(a) Simple re-centred Voronoi before merge. (b) Simple re-centred Voronoi after single merge.

Figure 1.6: Single merge.

Figure 1.6 shows the effects of a single merge. As shown, the weighted distance between the merged centres is less than those of any other neighbouring centre pairs.



(a) Re-centred Voronoi before merge. (b) Re-centred Voronoi after merge.

Figure 1.7: Completed merge.

Figure 1.7 shows a Voronoi Tessellation and its completed merged structure, it shows larger, more centralised cell structures.

References

Arthur, David, & Vassilvitskii, Sergei. 2007. k-means++: The advantages of careful seeding. *Pages 1027–1035 of: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics.

Aurenhammer, Franz. 1987. Power diagrams: properties, algorithms and applications. *SIAM Journal on Computing*, **16**(1), 78–96.

Cheng, Jingquan. 2009. Radio Telescope Design. *The Principles of Astronomical Telescope Design*.

Fortune, Steven. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, **2**(1-4), 153–174.

Green, Peter J, & Sibson, Robin. 1978. Computing Dirichlet tessellations in the plane. *The Computer Journal*, **21**(2), 168–173.

Hamerly, Greg. Making k-means even faster. SIAM.

Moore, Gordon E. 2006. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp. 114 ff. *IEEE Solid-State Circuits Newsletter*, **3**(20), 33–35.

NVIDIA. 2014. *GeForce GTX 750 Ti Whitepaper*. `http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce-GTX-750-Ti-Whitepaper.pdf`. Accessed on: 26/04/2016.

NVIDIA. 2015. *CUDA C Programming Guide*. `http://docs.nvidia.com/cuda/`. Accessed on: 03/05/2016.

NVIDIA. 2016. *CUDA*. `http://www.nvidia.com/object/cuda_home_new.html`. Accessed on 26/05/2016.

NVIDIA. 2016a. *GeForce GTX 750 Ti.* `http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-750-ti`. Accessed on: 26/04/2016.

NVIDIA. 2016b. *GPU-Accelerated Libraries.* `https://developer.nvidia.com/gpu-accelerated-libraries`. Accessed on: 22/05/2016.

Okabe, Atsuyuki, Boots, Barry, Sugihara, Kokichi, & Chiu, Sung Nok. 2009. *Spatial tessellations: concepts and applications of Voronoi diagrams.* Vol. 501. John Wiley & Sons.

Rajan, Krishna. 2013. *Informatics for materials science and engineering: data-driven discovery for accelerated experimentation and application.* Butterworth-Heinemann.

Rong, Guodong, & Tan, Tiow-Seng. 2006. Jump flooding in GPU with applications to Voronoi diagram and distance transform. *Pages 109–116 of: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games.* ACM.

Sault, RJ, & Wieringa, MH. 1994. Multi-frequency synthesis techniques in radio interferometric imaging. *Astronomy and Astrophysics Supplement Series*, **108**, 585–594.

Shamos, Michael Ian, & Hoey, Dan. 1975. Closest-point problems. *Pages 151–162 of: Foundations of Computer Science, 1975., 16th Annual Symposium on.* IEEE.

Smirnov, Oleg M. 2011. Revisiting the radio interferometer measurement equation-I. A full-sky Jones formalism. *Astronomy & Astrophysics*, **527**, A106.

Smirnov, OM, & Tasse, Cyril. 2015. Radio interferometric gain calibration as a complex optimization problem. *Monthly Notices of the Royal Astronomical Society*, **449**(3), 2668–2684.

Steinbach, Michael, Karypis, George, Kumar, Vipin, *et al.* 2000. A comparison of document clustering techniques. *Pages 525–526 of: KDD workshop on text mining*, vol. 400. Boston.

Subhlok, Jaspal, Stichnoth, James M, O'hallaron, David R, & Gross, Thomas. 1993. Exploiting task and data parallelism on a multicomputer. *Pages 13–22 of: ACM SIGPLAN Notices*, vol. 28. ACM.

Tasse, Cyril. 2014. Applying Wirtinger derivatives to the radio interferometry calibration problem. *arXiv preprint arXiv:1410.8706.*

Tasse, Cyril. 2016. *DDFacet imager.* TBD. Draft in preparation.

Thompson, A Richard, Moran, James M, & Swenson Jr, George W. 2008. *Interferometry and synthesis in radio astronomy.* John Wiley & Sons.

van Weeren, RJ, Williams, WL, Hardcastle, MJ, Shimwell, TW, Rafferty, DA, Sabater, J, Heald, G, Sridhar, SS, Dijkema, TJ, Brunetti, G, *et al.* 2016. LOFAR facet calibration. *arXiv preprint arXiv:1601.05422.*

Vuduc, Richard, & Choi, Jee. 2013. A brief history and introduction to GPGPU. *Pages 9–23 of: Modern Accelerator Technologies for Geographic Information Science.* Springer.

Way, Michael J, Scargle, Jeffrey D, Ali, Kamal M, & Srivastava, Ashok N. 2012. *Advances in machine learning and data mining for astronomy.* CRC Press.