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

August 19, 2016

Chapter 1

Introduction

Chapter 2

Background Information

2.1 Models and Algorithms

2.1.1 Voronoi Tessellations

A Voronoi Diagram is a partitioning of a space S by a set of points. Given n points (seed points) the space, $P = \{p_0, p_2, ..., p_{n-1}\}, P \subset S$, is partitioned into n regions, known as Voronoi regions or Voronoi cells, where every point, $s \in S_i, 0 \leq i \leq n-1$ in a region, $S_i \subset S$, is closest to a single seed point, $p_i \in P$, in terms of the space's distance measurement operation, d (Okabe & Chiu, 2009). An example of a Voronoi Diagram can be seen in Figure 2.1.

Weighted Voronoi Tessellations

The basic form of the Voronoi tessellation has the seed points as being indistinguishable from one another, other than their different positions in the space. An extension of the tessellation is to break this rule and to have the seeds have some bias or weighting associated with them. These weightings can represent a property of the data, for example, in terms of radio interferometry, they can represent the intensity of each source detected. This weighting can affect d in different ways, depending on how the weighting is accounted for; this is know as the "weighted distance". Some of these methods, discussed in Okabe & Chiu (2009) include multiplicative, additive, compound and power Voronoi diagrams. These diagrams have distance operators described as d_M, d_A, d_C, and d_P respectively.

$$\begin{aligned} d_M(s, p_i) &= \frac{1}{w_i} d \\ d_A(s, p_i) &= d - w_i \\ d_C(s, p_i) &= \frac{1}{w_{i1}} d - w_{i2} \\ d_P(s, p_i) &= d^2 - w_i \end{aligned} \tag{2.1}$$

Problems arise in attempting to compute the tessellations for the multiplicative, additive and compound Voronoi diagrams as the edges of these diagrams, could potentially be curved by a circular arc (d_M,d_C), a hyperbolic arc (d_A,d_C) or a fourth order polynomial arc (d_C) (Okabe & Chiu, 2009). This leaves the power diagram as the only Voronoi diagram which enforces that the edges are straight lines and the resulting tessellation is a convex polygon, similar to the standard Euclidean Voronoi diagram. For the power diagram, if the weighting is equal for all points, the resulting diagram is the same as that

of a standard Euclidean Voronoi diagram. It is possible in the power diagram, that a seed point will not be contained within its own associated Voronoi polygon. This occurs when two seed points ($p_i, p_j \in P$, $w_i < w_j$, $i \neq j$) are close enough together such that the weighted bisector, defined by

$$b(p_i, p_j) = \frac{1}{2}(||\mathbf{x}_i||^2 - ||\mathbf{x}_j||^2 + w_i - w_j) \qquad \mathbf{x}_i = (x_i, y_i), \tag{2.2}$$

does not lie on the line segment $p_i\bar{p}_j$. When this occurs, p_i lies in the region of V_j. If the difference in weighting between p_i and p_j is great enough and the distance between them small enough, the points in V_i may be an empty set. It is worth noting that Power Diagrams are also referred to as General Voronoi Diagrams (Aurenhammer, 1987).

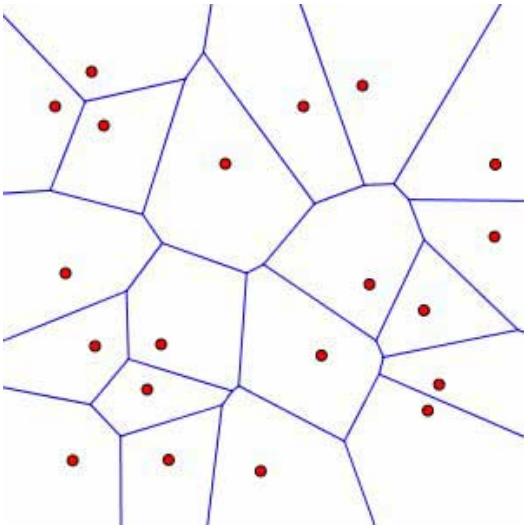

Figure 2.1: Voronoi diagram[1]

2.1.2 Voronoi Tessellation Generation Algorithms

Although Voronoi tessellations extend to multiple dimensions, for the simplicity we will only discuss those in a two dimensional plane.

Incremental Algorithm

The most simplistic of the generation algorithms, the Incremental is an iterative algorithm as described below (Green & Sibson, 1978) (Okabe & Chiu, 2009):

[1]Taken from `http://www.ams.org/samplings/feature-column/fcarc-voronoi`

1. Starting from $i = 0$ and an empty plane
2. A seed point, p_i is placed into the plane.
3. The nearest neighboring seed point $p_f = p_{nn}$ is found
4. A perpendicular bisector is drawn between p_i and p_f (if it exists).
5. The bisecting line is followed in both directions until it intercepts an existing edge or the plane's boundary on both ends.
6. A new edge is defined by this segment of the bisector as part of both p_i and p_f.
7. The seed point of the polygon that shares the found edge clockwise to p_f (anticlockwise to p_i) is then set to p_f.
8. Continue from step 4 until $p_f = p_{nn}$ again.
9. Set $i = i + 1$ and repeat from step 2 until $i = n$

In its most naive form, this algorithm achieves an efficiency of $O(n^2)$.

Divide and Conquer Algorithm

The Divide and Conquer algorithm was first proposed by Shamos & Hoey (1975) and also described in Okabe & Chiu (2009). It is a recursive algorithm that improves on the Incremental algorithm by having a construction time of $O(n \log n)$.

1. If the space contains only one point, return it with the entire plane as its Voronoi region.
2. Divide the space, S containing the set of n seed points, P, into two subspaces, S_L and S_R, such that S_L and S_R contain $n/2$ seed points and every seed point of P_L lies to the left of every seed point of P_R (this is made easier if P is ordered).
3. Recursively compute the Voronoi tessellations for P_L in S_L and P_R in S_R; V_L and V_R respectively.
4. A polygonal line, Q, must now be found such that Q merges V_L and V_R into a single Voronoi tessellation, V:

(a) Starting with the polygon of V_R which contains the top-left corner of S_R, p_R and the polygon of V_L which contains the top-right corner of S_L, p_L. Since p_L must lie to the left of p_R, they must overlap when V_L and V_R are extended into S.

(b) A perpendicular bisector is drawn between p_L and p_R and segmented between its two closest edge intercepts from the shortest distance between p_L and p_R and add this segment to Q.

(c) If the lower edge, intercepted by the bisector is in V_R then p_R is set to the seed point polygon which shares this edge and similarly if the edge is in V_L.

(d) Continue from step 4b until the bottom of S is reached.

5. Remove all line segments of V_L to the right of Q and all those of V_R to the left of Q to form V.

6. Return V recursively until the full Voronoi tessellation is complete.

Part of achieving this efficiency is assuming P is co-lexicographically ordered, meaning for all $p_i, p_j \in P$, $0 \leq i < j < n$; $x_i > x_j$ or ($x_i = x_j$ and $y_i > y_j$). This speeds up the partitioning of P into P_R and P_L at each level of recursion.

Fortune's Algorithm (Sweep-Line Method)

Fortune (1987) describes an algorithm where the tessellations are found by a line "sweeping" over the space and solving the problem at each step of the sweep. This can be problematic for Voronoi tessellations as the line may intercept the Voronoi Region of a seed point before it intercepts the point. Therefore the Voronoi Tessellation is not computed directly, but through a geometric transform. The transform $\phi(x(s), y(s))$ works such that for any point, $s \in S$ with coordinates $(x(s), y(s))$,

$$\phi(x(s), y(s)) = (x(s) + r(s), y(s)), \tag{2.3}$$

where $r_i(s)$ is defined as the distance to the seed point $p_i \in P$ and $r(s) = min\{r_i(s) | 1 \leq i \leq n-1\}$, is the distance to the closest seed point to s. This transform can then easily be reversed to re-obtain S and its set of Voronoi tessellations. Now, for the transform of S, $\phi(S)$ denoted by Φ, the left-most point of each Voronoi Region is its seed point (except the left-most seed point), this is tessential for the algorithm. It is important to note that the perpendicular bisectors of seed points in S, through the transform, become

hyperbolas in Φ (provided they are not horizontal in S). For $p_i, p_j \in P$, the hyperbola is denoted as h_{ij} which can be split into h_{ij}^+ and h_{ij}^- as the upper and lower half-hyperbolas about the left-most point, respectively. Set Q is denoted as the set of all event points in the algorithm. Q is initially populated with the seed points (in co-lexicographical order) but the edge interceptions will be added as they are found. The algorithm, as described by Okabe & Chiu (2009) goes as follows:

1. Add P to Q.
2. Choose and delete the leftmost seed point, p_i from Q.
3. Create a list, L containing the transformed Voronoi region of p_i, $\phi(V_i)$.
4. While Q is not empty do the following:
   (a) Choose and delete the leftmost element, w of Q.
   (b) If w is a seed point:
      i. Set $p_i = w$.
      ii. Find the region, $\phi(V_j)$, containing p_i.
      iii. Replace $\phi(V_j)$ in L with $(\phi(V_j), h_{ij}^-, \phi(V_i), h_{ij}^+, \phi(V_j))$
      iv. The half-hyperbola intercept(s) with any other hyperbolas are found, if they exist, and are appended to the front of Q.
      v. Repeat from step 4.
   (c) If w is a half-edge:
      i. Set $\phi(q_t) = w$ where $\phi(q_t)$ is the intercept of $h_{ij}^\pm$ and $h_{jk}^\pm$.
      ii. Replace all sequences of the form $(h_{ij}^\pm, \phi V_j, h_{jk}^\pm)$ on L with $h = h_{ik}^+$ or $h = h_{ik}^-$ appropriately.
      iii. Remove from Q any intersections of $h_{ij}^\pm$ and $h_{jk}^\pm$ with other half-hyperbolas.
      iv. Move any intersections of h in L to Q.
      v. Mark $\phi(q_t)$ as a Voronoi vertex incident to $h_{ij}^\pm$, $h_{jk}^\pm$ and h.
      vi. Repeat from step 4.
5. Return the half-hyperbolas on L, the set of marked intersections from step 4(c)v and the relations among them.

2.1.3 Clustering Algorithms

It may be the case that the number of potential seed points in a space, N_p, is much larger than the optimal number of facets, N_v. In these cases it would reduce the overall computation time drastically if the N_p points were grouped into N_v clusters. From each of these clusters, a point is then chosen as a seed point to be used to find the corresponding Voronoi tessellation. Some key examples of such clustering algorithms are described in this subsection.

K-Means Algorithm

K-means clustering is an iterative process where an initial guess at the center of a cluster, c, is made and improved with each iteration. It is named as such because it seeks to separate n objects into k clusters where, for each object in a cluster ($o^i \in C_i$, $i \in \mathbb{R}$, $0 \leq i \leq k-1$) the mean point of that cluster, c_i, is closer to it than any other mean point and the c_i is representative of the average value of all points, $\frac{1}{m}\sum_{j=1}^{m} o_j^i$, in C_i. Way *et al.* (2012) describe the algorithm as follows:

1. Randomly choose k mean points $(c_0, \ldots, c_{k-1})$.
2. Assign each c_i an empty object set, C_i.
3. Iterate through all the objects in the space $(o_0, \ldots, o_{n-1})$ and assign the object to the object set of the mean point closest to it.
4. Set all c_i to be the average of all points in their respective C_i.
5. If the sum of the changes in c_i, $\sum_0^{k-1} \Delta c_i$, is greater than some given tolerance, ϵ, then repeat from step 2, else return the set of means (and their object sets if necessary).

One obvious problem with this algorithm is that the number of iterations can be unpredictable; this is addressed by having the sum of changes only converge to ϵ, instead of complete convergence. With large data sets and large k-values where the mean points converge in smaller steps with every iteration, this can help drastically reduce the runtime of the algorithm. The clusters produced are also dependent on the initial placement of the mean points (Way *et al.*, 2012). Other methods of improving the runtime include probabilistic choices of starting mean points (Arthur & Vassilvitskii, 2007) and constraining the distance and using the triangle inequality (Hamerly, 2010).

Bisecting K-Means Algorithm

A variation on the k-means algorithm is to embed it into another iterative method, which, by design, reduces the computation time and also improves the quality of the clusters produced. The algorithm works by branching large clusters into smaller ones. The algorithm, described by Steinbach *et al.* (2000), goes as follows:

1. Start with the entire set of objects in the space as part of a single cluster.
2. Choose the largest cluster in the space.
3. Split the objects into two sub-clusters and refine iteratively as by way of the k-means algorithm.
4. Repeat from step 2 until k clusters are produced.

Agglomerative Clustering Algorithm

Contrary to the k-means algorithm (and more specifically the bisecting k-means) is the agglomerative clustering algorithm. Instead of starting with a single cluster containing all points, this algorithm instead places every point in its own cluster and merges them until the required number of clusters are produced. Way *et al.* (2012) describe the algorithm as:

1. Begin with each object in its own cluster.
2. Merge the two closest clusters.
3. Repeat step 2 until k clusters remain.

Although this algorithm will always yield the same result for a given data set, it is far more expensive than the k-mean. Improvements can be made on this, however by instead building a minimum spanning tree, weighted by the distance between the data and iteratively removing the links with the highest weights until the number of required clusters is produced.

2.1.4 Related Work

Standard Voronoi Faceting

In Tasse (2014), Smirnov & Tasse (2015) and van Weeren *et al.* (2016), a series of observed or simulated extragalactic points are clustered into facets using a Voronoi tessellation algorithm with the seed points for these facets set as the brightest points in each facet. An example of this can be seen in Figure 2.2 where the facets are superimposed over the source image from which they are derived.

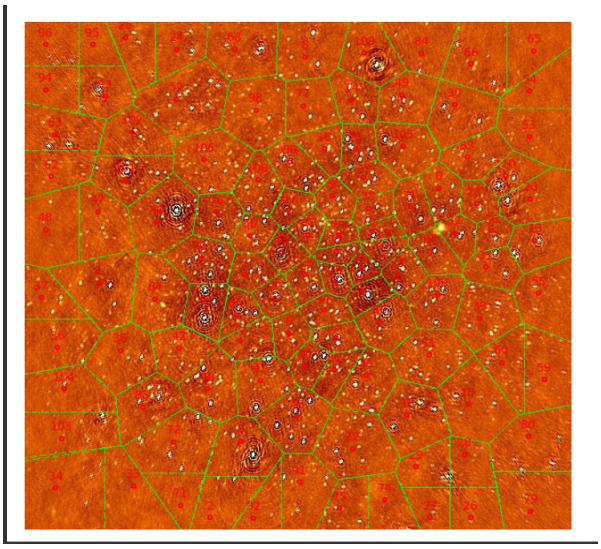

Figure 2.2: Example of Voronoi faceting to group extragalactic points for DD-calibration

Chapter 3

WIP

3.1 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 2.1.2) 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.

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. 2010. Making k-means Even Faster. *Pages 130–140 of: SDM*. 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, Boots, Sugihara, & Chiu. 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.

SKA. 2016. *The SKA Project.* `http://www.ska.ac.za/about/project.php`. Accessed on: 21/02/2016.

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, Stichnoth, O'hallaron, & Gross. 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.