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

September 27, 2016

Contents

Chapter 1

Introduction

Chapter 2

WIP

2.1 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 **??**) 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

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

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.

2.1.1 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 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.

2.1.2 Voronoi Function

The Voronoi function 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.

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.

2.1.3 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 180, $thepointsareleftinthelist, thenextpointintherangeisaddedandt$ 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.

2.1.4 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 tnan 180, $theorderandtheadjacentpointbeingusediscounter - clockwiseforVDLandclockwiseforVDR.Tofindthelowertangent, theorderisreversedwithVDLstep clockwise.$
$Theuppertangentisusedtofindthestartingpointofthepolygonalchainusedtomergethevoronoi.Bisect$

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.