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 22, 2016

Contents

Chapter 1

Introduction

Chapter 2

Background Information

2.1 Introduction

In this paper we discuss the literature and resources required to create the sky tessellation algorithm and the techniques which could be used to optimize it.
We begin by looking at the background of the problem in radio astronomy. We discuss how radio telescopes work, how the image is created and how direction-dependent effects occur and how they can be corrected.
We then look at Voronoi tessellations, what they are and variations in how they work. We focus especially on Voronoi tessellations of weighted points. Tessellation algorithms are also be explained as well as algorithms for clustering data. Their efficiencies and complexities will also be discussed.
Lastly we look at parallelism, Graphical Processing Unit (GPU) architecture and the technicalities of programming on a GPU. We discuss the hardware to be used, the NVIDIA GTX 750 Ti, and the GPU programming language CUDA. The optimizations of GPUs and CUDA are also discussed.

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 **??**) 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 implimentation for the Divide and Conquer Voronoi algorithm written in

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

python 2.7. pyVoronoi uses a simple GUI to genertate a voronoi diagram in a fixed plane using sources read in from a textfile. 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 pipleined process. The visualiser and interface were therefore removed and the remaining code refactored so that the process is a function which may be called by the user or used in a larger process.

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 elelment 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 intesity threshold is a single double value which defines the minimum intensity a source needs to be a voronoi cell centre.

Once the sources are read in, they are used to generate the voronoi centres structurally defined as a point. Each point inherits the x, y and z values of the source. Each point also has a related attribute which is a list of structures containing another point and their corresponding bisecting line, this list is set as empty. The point also has an error value associated to it, this error value is set to zero and will only come into use after the voronoi diagram has been generated. Once the list of points is created, it is sorted by the x then the y values of the sources.

Once we have our centre points we can begin generating the voronoi diagram. The function Voronoi takes in the list of points and the range of these points it has access to

References