Detect Sparse Observation Zones
Senior Design Group 14-31

Design Document

Team Members:
John Harding
Nicholas McLaren
Michael Ore
Andrew Upah
Bryce Wilson

Advisor:
Dr. Ruchi Chaudhary, Department of Biology, University of Florida

Client:
Prof. Gordon Burleigh, Department of Biology, University of Florida

Contents

Problem Statement

There are a number of big databases that share biodiversity data for biological research and collaboration. For example, the Global Biodiversity Information Facility (http://www.gbif.org) has millions of species records such as latitude and longitude coordinates. In addition, the Avian Knowledge network (http://www.avianknowledge.net) has over 100 million bird observation records. These databases provide an enormous amount of information that is useful in understanding the patterns and dynamics of various species across a region. Another interesting, yet not well explored, direction is finding out the biggest regions where there are few or no species records. This may represent different features in the landscape or areas where there has been little or no sampling. This project is focused on developing a software tool that inputs a collection of millions of points and outputs the largest areas where there are few or no observations. More formally, this problem is written in the following way:

Problem (Detect Sparse Observation Zones):

Input: a collection of latitude and longitude points: *k*, *n*.

Output: *n* latitude and longitude points with corresponding radiuses such that there are no more than *k* points in the respective circle of each point.

Term and Acronym Definitions

CGAL: Open source software library that provides efficient algorithms in computational geometry
Convex Hull: the smallest convex (no indent) region that contains any set of items
OpenGL: Multi-platform API used in rendering 2D and 3D computer graphics
Qt: Cross-platform application framework used in creating graphical user interfaces
Voronoi Diagram: A division of space using a set of input points in which each point has the corresponding region of all points in space nearer to that point than any other
Higher Order Voronoi Diagram: A generalization of Voronoi Diagrams. For any natural number k and given a set of input points, an order-k Voronoi Diagram associates every possible combination of k input points with the region where those k points are closest
kth Order Voronoi Diagram: A Higher Order Voronoi Diagram with a specific k value

kth Degree Voronoi Diagram: Another generalization, similar to Higher Order Voronoi Diagrams. Every region is associated with a single input point that is the kth-closest input point relative to every point in the region.
k-d tree: Multidimensional generalization of a binary search tree
k-circle: Relative to a set of input points, a k-circle is a circle with fewer than k points inside it

---

Algorithm

We'll model the problem as finding the largest circle containing fewer points than a parameter k (we'll call such circles "k-circles"), within a configurable area. We'll accomplish this with a generalization of Toussaint's LEC-2 algorithm for finding largest circles that contain no points (Toussaint 1983).

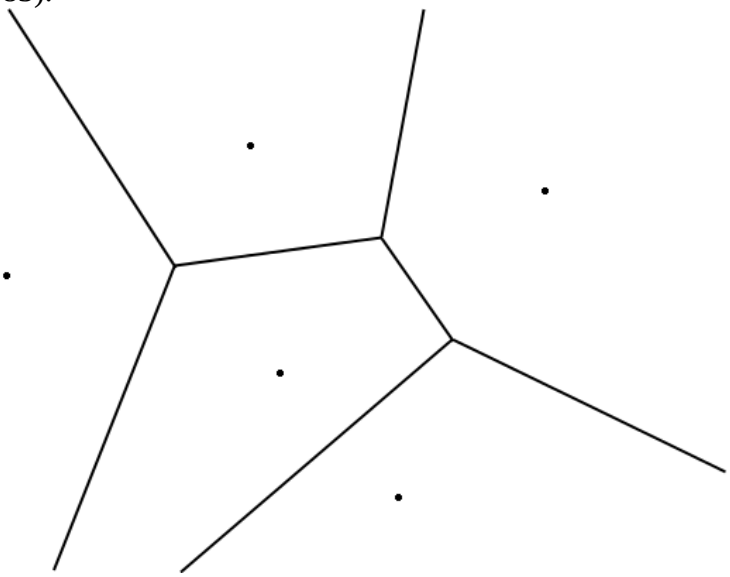

Figure #1: The weblike structure is the ordinary Voronoi Diagram for the five points.

A quick explanation of how Toussaint's algorithm works is necessary. Finding the largest empty circle requires searching through every maximal empty circle. Every such circle must be "bound" by three points on its edge, or else the circle could be made larger by shifting and/or expanding it.

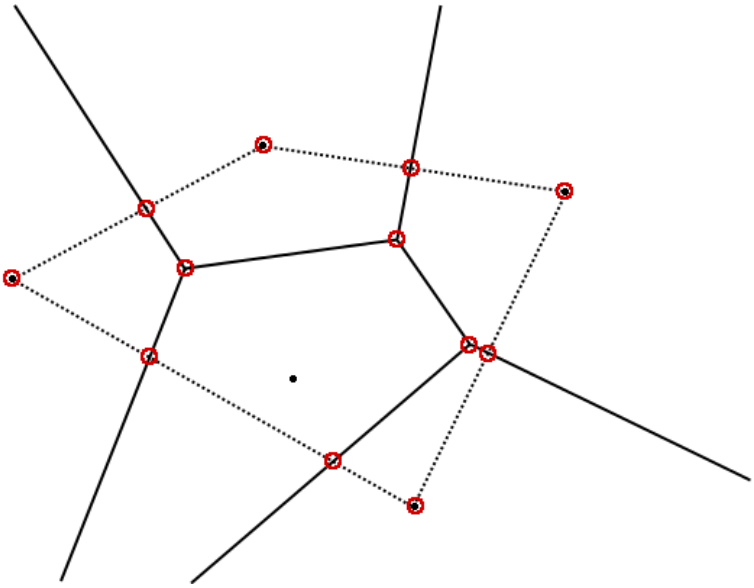

Figure #2: Every point that must be searched is circled in red, the convex hull is the dotted line

The Voronoi Diagram created from the input points partitions the plane into a region for each point, such that all points in each region are closer to the corresponding input point than to any other input point. Given a maximal empty circle, its center is a point that is closer to the three bounding points than any other input point. The center is therefore at the vertex for the intersection of the regions for the bounding points. Therefore, given a constraining polygon, we need only search every Voronoi vertex within the polygon and every extreme point of the polygon in order to find the largest empty circle. Figure #2 shows the points we need to check for the largest empty circle within the convex hull, and Figure #3 shows the result. The same algorithm can be used for any polygonal location constraint.

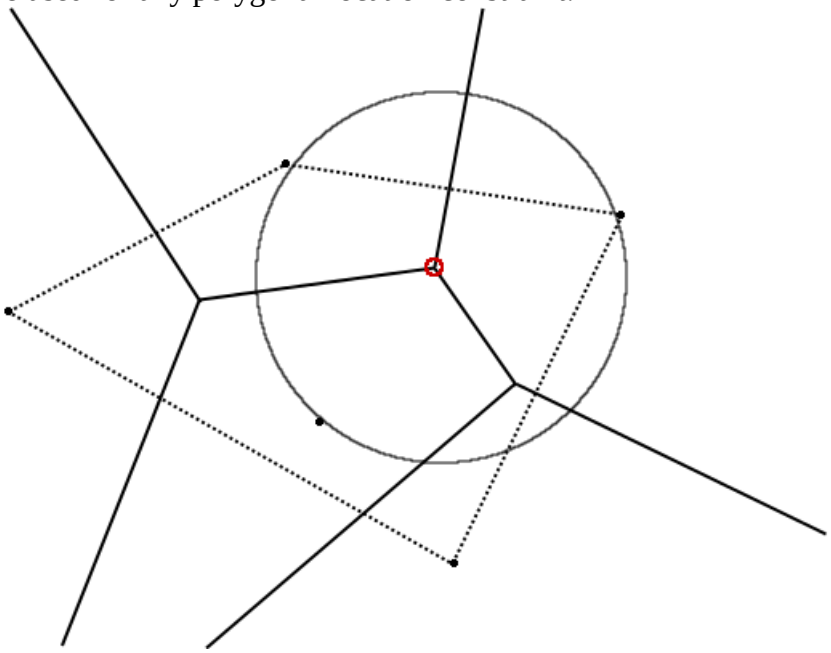

Figure #3: The largest empty circle whose center is inside the convex hull

Similarly, to find the largest k-circle, we should search every maximal k-circle. A maximal k-circle must also be bound by three points, and contain k-1 points (otherwise it could

consume one of the bounding points). Therefore, the bounding points are tied as the kth-closest input points to the circle's center, and the center must be a vertex intersecting three regions of the kth-Degree Voronoi Diagram of the input points. The kth-Degree Voronoi Diagram of any set of points can be created by overlaying the kth-Order Voronoi Diagram upon the (k-1)th-Order Voronoi Diagram. Figure #4 shows the points we would need to search, in addition to the points in Figure #2, in order to find the largest 2-circle within the convex hull.

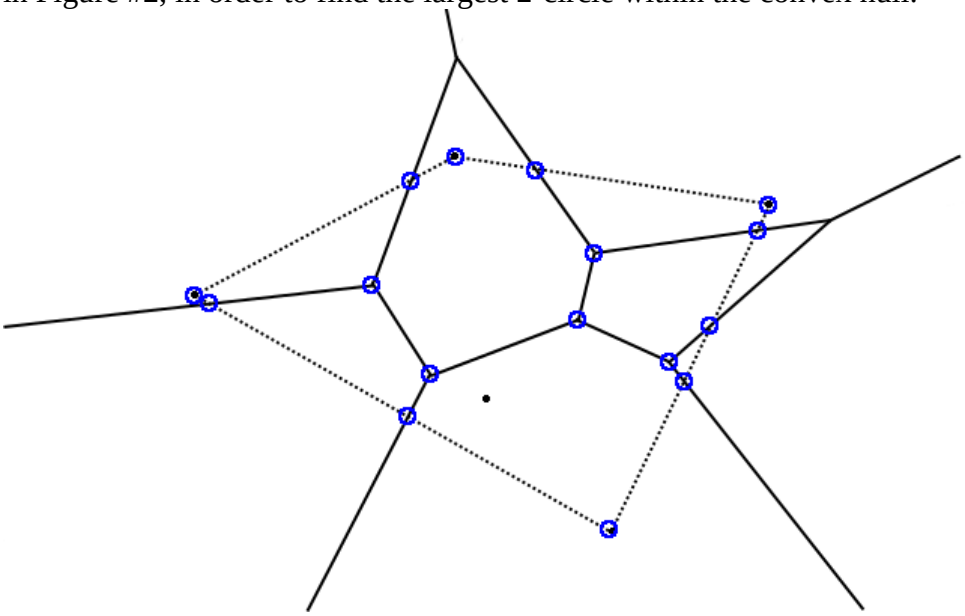

Figure #4: The 2nd-Order Voronoi Diagram with the new points of interest marked in blue.

Our generalized version of Toussaint's algorithm would replace the Voronoi Diagram with a kth-Degree Voronoi Diagram. kth-Degree Voronoi Diagrams are computable in polynomial time (Bernard 1987), and have vertices of polynomial order. Therefore, our generalization would also be computable in polynomial time.

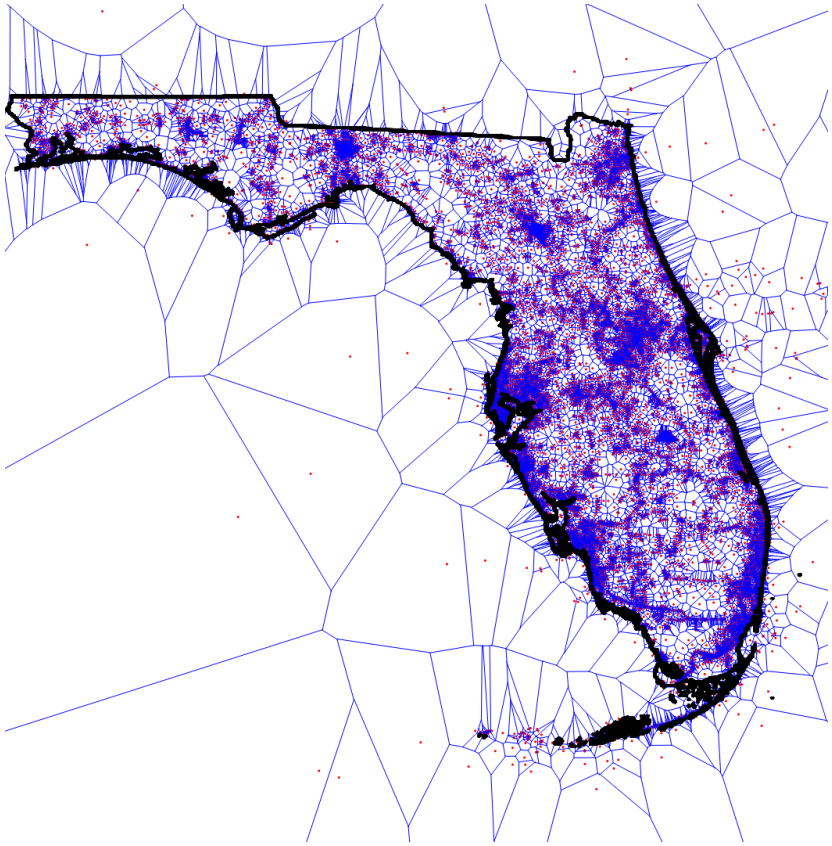

Figure #5: Visualization of LEC-2 on sample data from Prof. Burleigh

If k is proportional to n (in other words, the number of points in each circle is a fixed percentage of all points) then the vertices in the kth-Degree Voronoi Diagram is order n2 and the algorithm is then Ω(n2). This is most likely too slow for the scale we're looking for, so we most likely need to resort to an approximation algorithm based on this optimal algorithm. Achieving a satisfactory approximation algorithm requires a bit more research and a lot of empirical experimentation.

Prototype

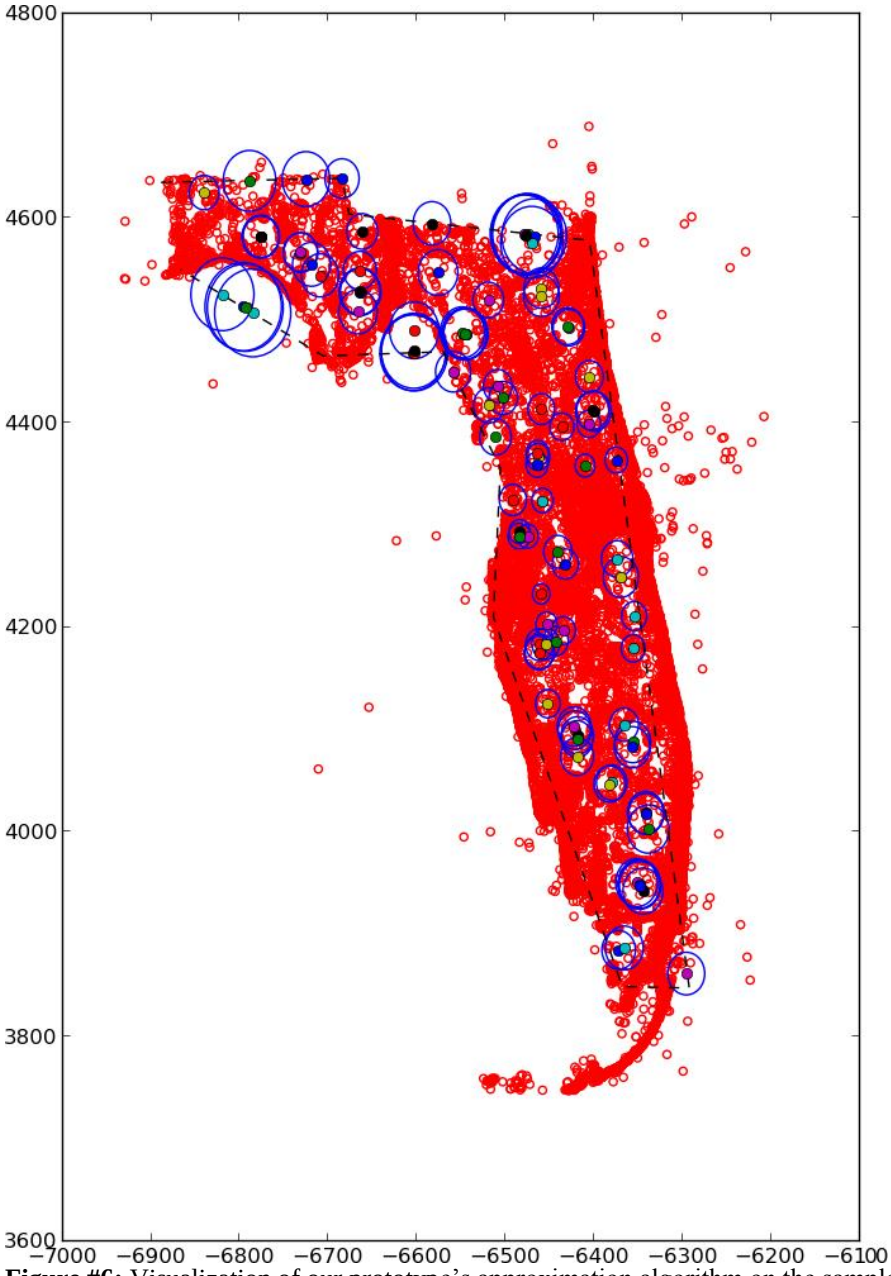

Figure #6: Visualization of our prototype's approximation algorithm on the sample data

We've made a functioning prototype of that uses a rough approximation algorithm based on the optimal algorithm in the previous section.

It makes use of a k-d tree, which can be used to find the k closest points from any query point quickly. Effectively, it contains the same information as all higher-order Voronoi diagrams and is much faster to construct but does not explicitly provide vertex and edge positions.

The approximation algorithm is iterative. In each iteration, the next approximation is found by moving the current approximation a small distance away from the kth closest input point.

Figure #6 shows the results of the approximation algorithm on a dataset with n ≈ 25,000 and k=25, generating 100 k-circles from random starting points and 100 iterations each. Runtime was about a minute. We've not measured its accuracy by any metric yet.

Our next step for implementation would be to implement the optimal version of the algorithm which can be used as a standard to evaluate this and any other approximate implementations.

Software Architecture

Our plan is to rely heavily on CGAL and Qt with OpenGL as the user interface between them. CGAL already has algorithms for generating Voronoi Diagrams, but not Higher Order ones. Nonetheless, we can calculate and represent Higher Order diagrams with CGAL's 2D arrangement module. We determined Qt as our user interface of choice because this API was designed to work well with CGAL. In addition, OpenGL will be implemented as our graphical viewing of the input, output, and boundaries because this software will be less intensive on the CPU and is able to draw polygons and other geometrical shapes with more precision. Below is our internal software flow diagram that shows how the main modules will function together for the backend.

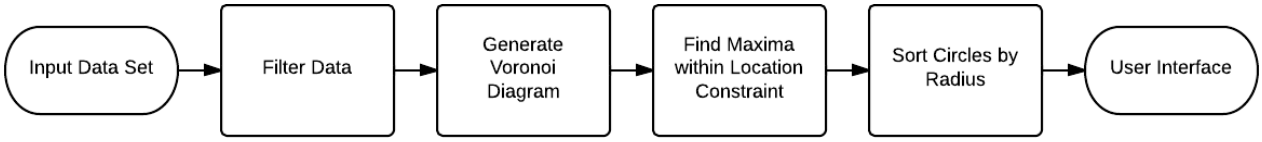

Figure #7: Flowchart showing the flow of the algorithmic core of the project

---

User Interface

As for the user interface, we are planning to make it very easy to use and with some additional functionality to analyze the output data. This software will be made available for Windows, Linux, and MacOS, and thus we turned to Qt since it will look good on all operating systems. Our screensketch of one possible look is shown below.

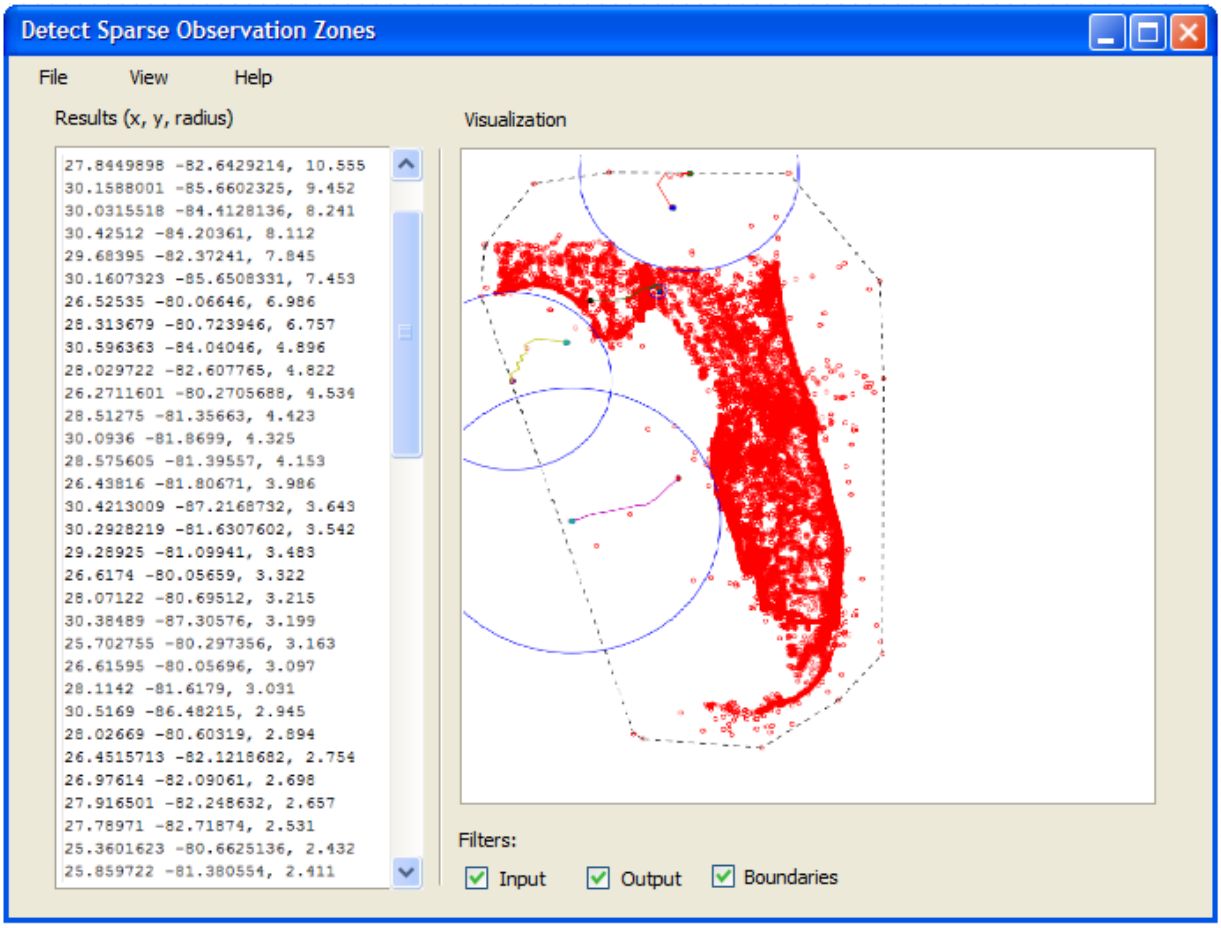

Figure #8: Concept sketch of our GUI

The input file will be selected by the user at the start of the program by clicking the select new file menu item under the file menu option. Then the user may select a predefined boundary if they so wish from http://gadm.org/ or just use the convex hull boundary by default. The program data is then processed when the user selects the run option. After finishing, the CGAL algorithm outputs the largest empty circles data analytically by text on the left and visually onto a data map on the right. Finally, a popup allows user to save analytical results to their defined destination.

Use Cases

User:

- Inputs text file of lat/long coordinates which are checked for correctness
- Specifies number of points to accept in largest circle
- Specifies number of areas to display
- Exports output data to .txt
- Exports visualization to .png
- Changes the number of output circles displayed
- Add or remove visualization module

Assumptions and Limitations

Assumptions:

- Both Toussaint's algorithm and our generalization of it assume that no 4 points are cocircular and no 3 points are colinear. We will need to carefully investigate the effects of breaking this assumption.

Limitations:

- Our algorithm only outputs circles; these may not always accurately represent the underlying cause of any sparsity.

Alternative Options

We decided to approach this problem with geometric optimization. That is, we're optimally maximizing circle size while minimizing number of points contained, with a configurable balance between them. We had other design ideas that had advantages and disadvantages over our final choice.

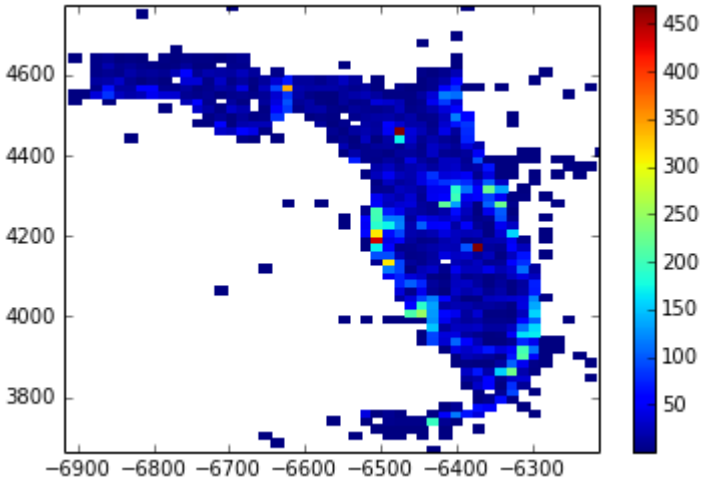

Figure #9: Plot of bird observations in Florida, in rectangular cells

One line of thought was to partition the map into regular polygonal "cells", and count the number of points in each. Then, we could look for contiguous regions that have a large number of cells, but not many total points.We could not think of a compelling way to construct such a region, so we pursued other leads. Such an approach would be advantageous in that the regions constructed wouldn't need to be circles; they could have arbitrary shape if necessary. The biggest drawback of this approach is the reduced granularity that comes in throwing out information about where points are inside the cells.

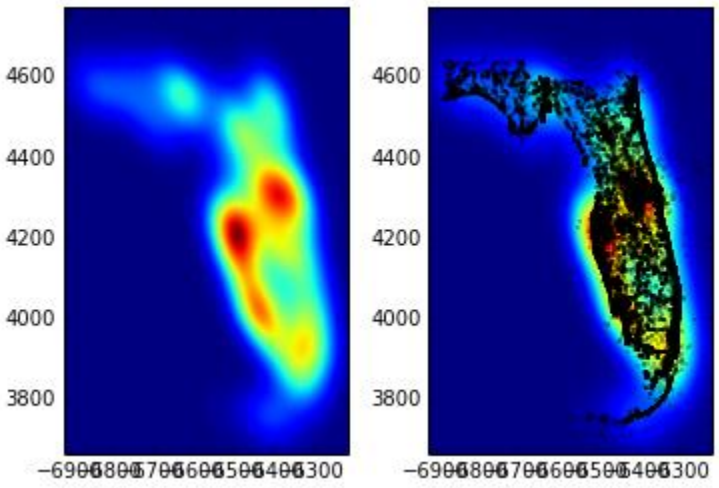

Figure #10: Plot of bird observations in Florida, with a heatmap generated with a kernel density estimation

Another candidate approximation method was statistical density estimation. This is similar to the above approach, but with a continuous result. This avoids the issue of loss of granularity to an extent, but it still throws out some information. We couldn't think of a good way to automatically highlight sparse regions with this representation either, but a plot of the result would still suggest to the user where sparse areas are. It may still be a good idea to use this, and overlay the results of our optimal algorithm on top of a heatmap.

Testing Approach

Our approach to testing will be defined using the following:

- Small case of kth degree to verify algorithm correctness
- Large case to ensure it can handle the millions of points it could be receiving
- Large case of a large degree to ensure acceptable runtimes in the worst cases
- Cases that could potentially not work, i.e. cases with 4 cocircular points
- Try to change the boundaries in order to break the application
- Try to enter improper input data and see if the program crashes

Testing will be considered done when all of these have been completed to a satisfactory level.

Team Information

Team Members	**Role**	**Contact Information**
John Harding (CprE)	UI Designer	otangs@iastate.edu
Nicholas McLaren (CprE)	Algorithm/Implementation	nmclaren@iastate.edu
Michael Ore (CprE)	Researcher/Webmaster	orem@iastate.edu
Andrew Upah (CprE)	CGAL-UI Integration/Leader	aupah@iastate.edu
Bryce Wilson (CprE)	UI Rendering	bmwilson@iastate.edu
Ruchi Chaudhary	Advisor	ruchic@ufl.edu
Prof. Gordon Burleigh	Client	gburleigh@ufl.edu

Closing Summary

The above information describes our current design of our backend algorithm using CGAL and the current position of the user interface. If any information was not presented here, you can find more information on our project plan or by contacting one of the team members. We would like to thank Professor Burleigh as well as our advisor, Ruchi Chaudhary for allowing us to help

solve this problem and for guiding us along the way. Finally, the list below shows the research papers used while investigating the best algorithm for our design.

Bibliography

Toussaint, Godfried T. "Computing Largest Empty Circles with Location Constraints." *International Journal of Computer & Information Sciences* 12.5 (1983): 347-58. Print.

Lee, Der-Tsai. "On K-Nearest Neighbor Voronoi Diagrams in the Plane." *IEEE Transactions on Computers* C-31.6 (1982): 478-87. Print.

Chazelle, Bernard, and Herbert Edelsbrunner. "An Improved Algorithm for Constructing Kth-Order Voronoi Diagrams." *IEEE Transactions on Computers* C-36.11 (1987): 1349-354. Print.