May 14-31
Detecting Sparse Observation Zones

The Team

Team Members:

John Harding (Cpr E)
Michael Ore (Cpr E)
Nicholas McLaren (Cpr E)
Andrew Upah (Cpr E)
Bryce Wilson (Cpr E)

Advisor:

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

Client:

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

Project Overview & Problem

- Many databases share biodiversity data for biological research and collaboration
  - e.g. The Global Biodiversity Information Facility & The Avian Knowledge Network
  - Over 100 million bird observation records and millions of other species data
- Helps to understand the patterns and dynamics of various species across a region
- Interested in biggest regions with few or no species records

Problem Statement Formalized

Create a software tool that solves the following problem:

Problem (Detect Sparse Observation Zones):

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

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

Solution Overview

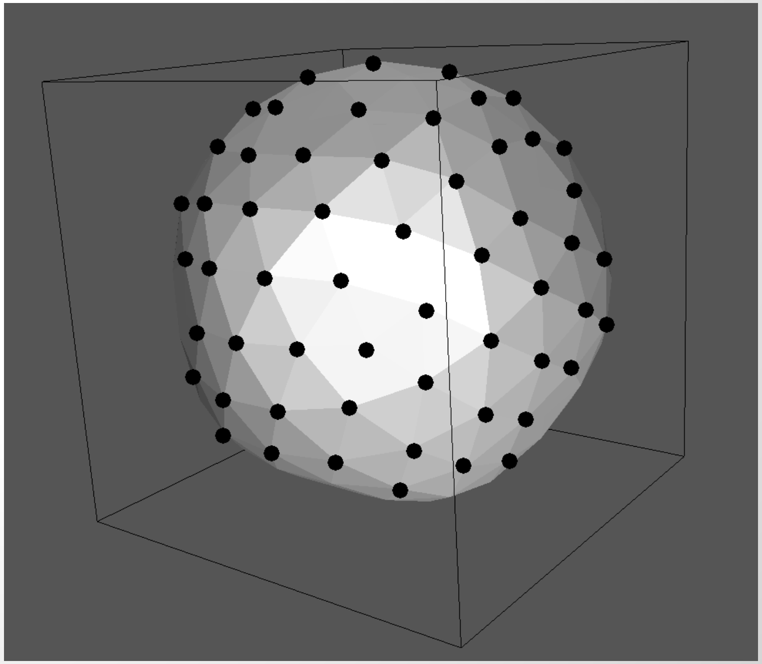
The algorithm we delivered:

1. Create a regularly spaced grid of points over Earth
2. At each grid point, find the largest radius that contains at most k input points
3. Output the grid points and their radii as k-circles

This provides a good idea of how sparse the data is over the whole area of interest

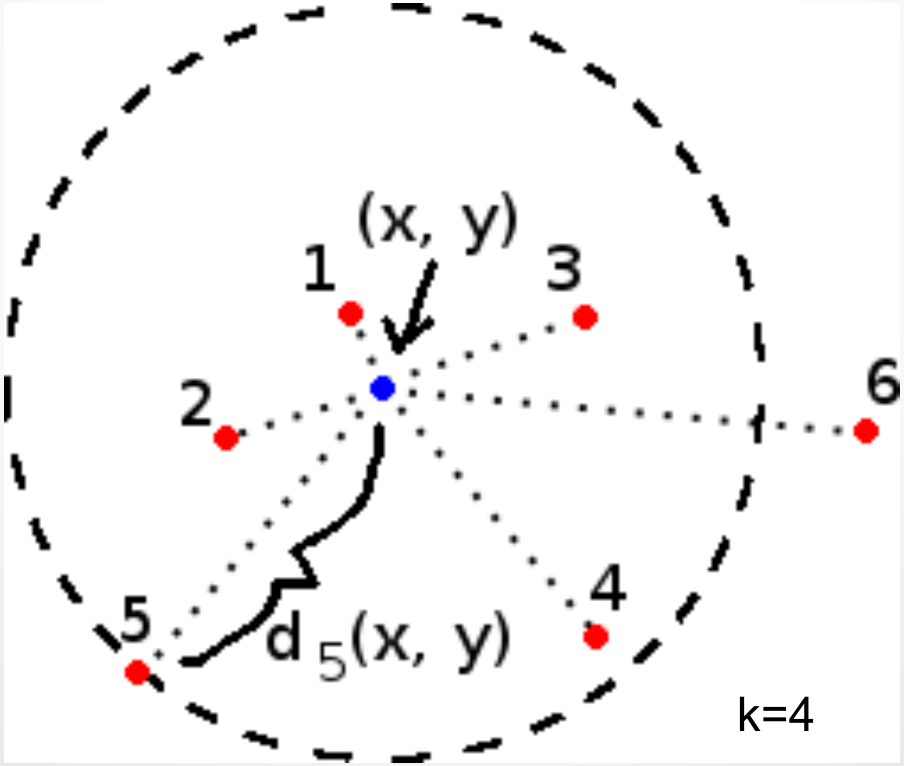
Grid on Earth: Geodesic Grid

- Compared to planar grid:
- Pros:
    - Useful for both global and local data
    - Minimal distortion
- Cons:
    - Memory consumption
    - Wasteful for local data



Maximizing Radius

- (k+1)-distance function, $d_{k+1}(x, y)$
- Computable in O(klogN), N = # of input points



Output

- Evaluate $d_{k+1}(x, y)$ at each grid point, output
- Output Columns:
  - Latitude
  - Longitude
  - Radius
  - Grid Spacing
- $d_{k+1}(x, y)$ at grid points are good approximations for space in-between
- $d_{k+1}(x, y)$ for largest grid point is good approximation to global maximum

Performance Analysis

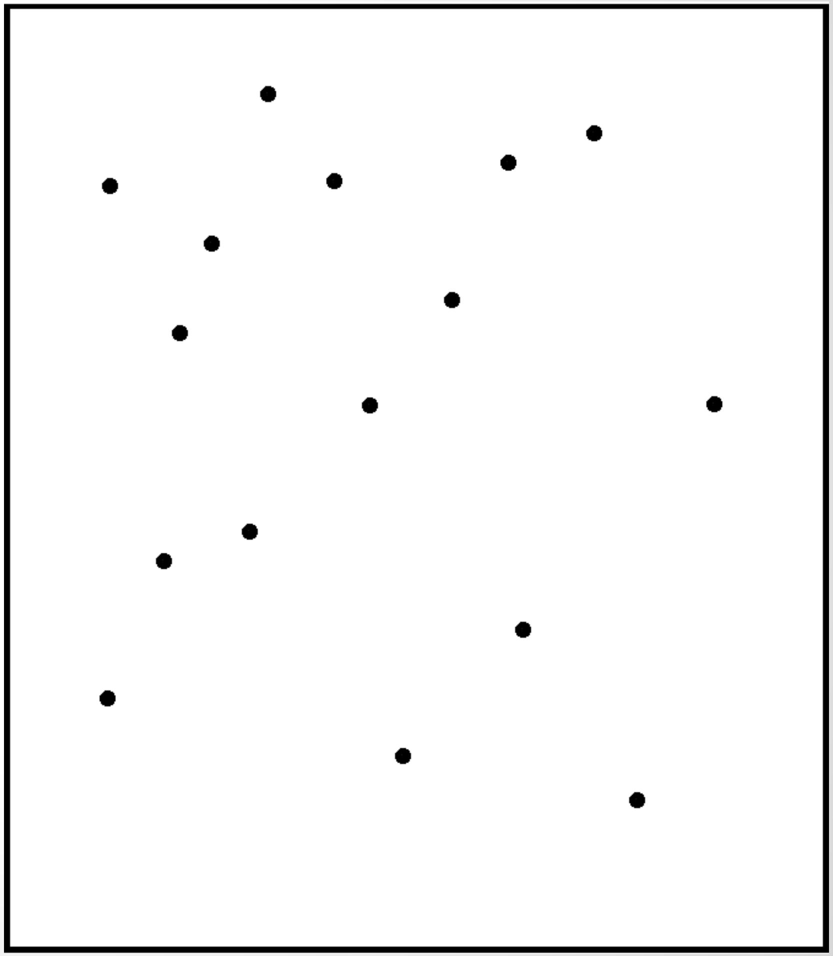
- A = area of search space (Earth's surface)
- g = grid spacing
- N = number of input points
- # of grid points ~ $O(A/g^2)$
- Time to sample ~ $O(k\log N\ A/g^2)$
- Memory use ~ $O(N + A/g^2)$
  - This is the bottleneck for low g

Testing

- Some unit testing
- Optimal result compared to approximation for small data
- Performance testing:
    - 1,000,000 random input points, k = 1000
    - 10,485,762 sampling points
    - 45 minutes
    - 8 GB

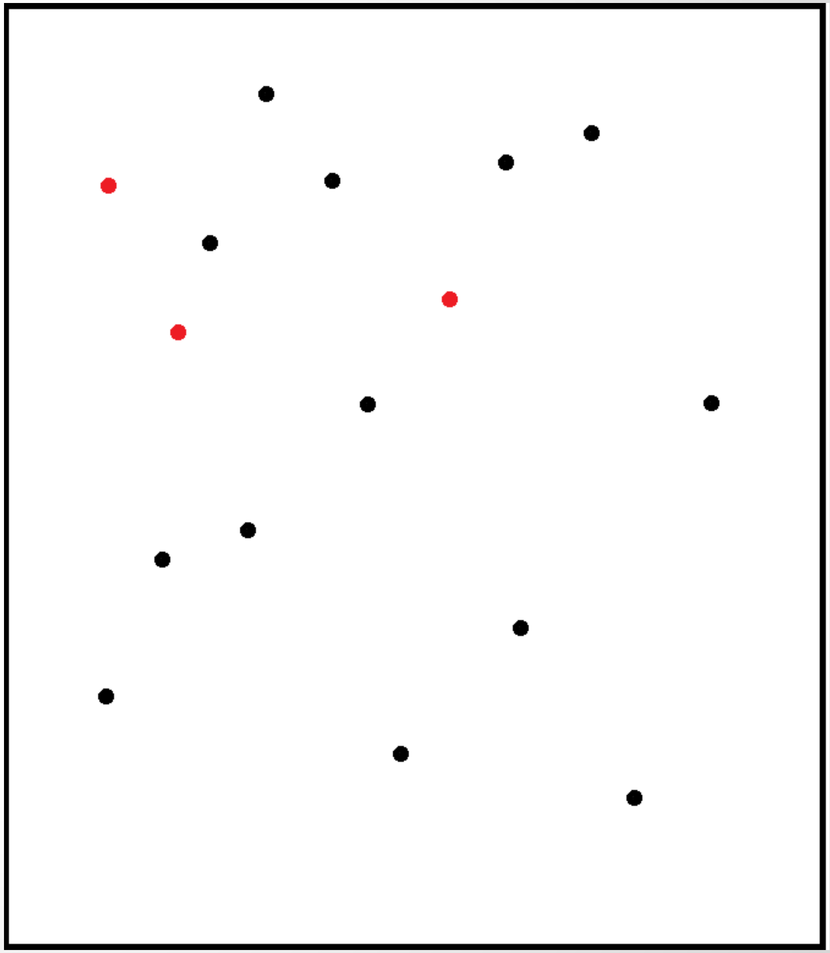
Simple Optimal Algorithm

- With a given set of points
- Pick a combination of 3
- Create a circle from these points
- Count points within the circle
- Save circle's center and radius if points inside = k
- Runs in $O(N^4)$



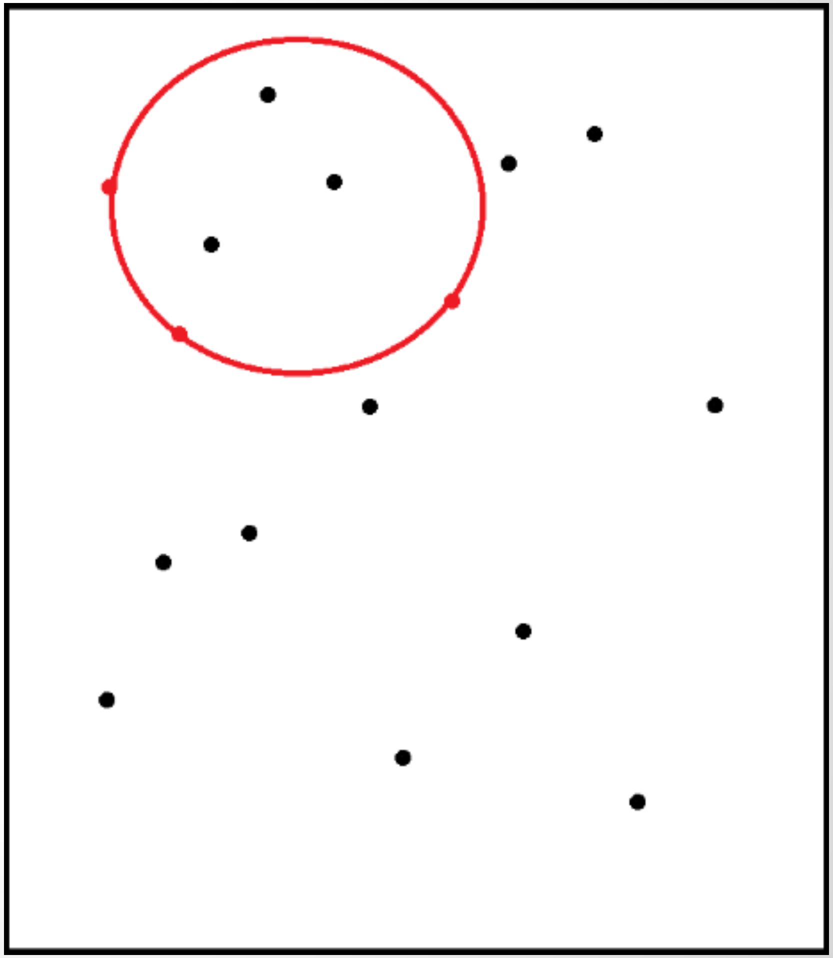
Simple Optimal Algorithm

- With a given set of points
- Pick a combination of 3
- Create a circle from these points
- Count points within the circle
- Save circle's center and radius if points inside = k
- Runs in $O(N^4)$



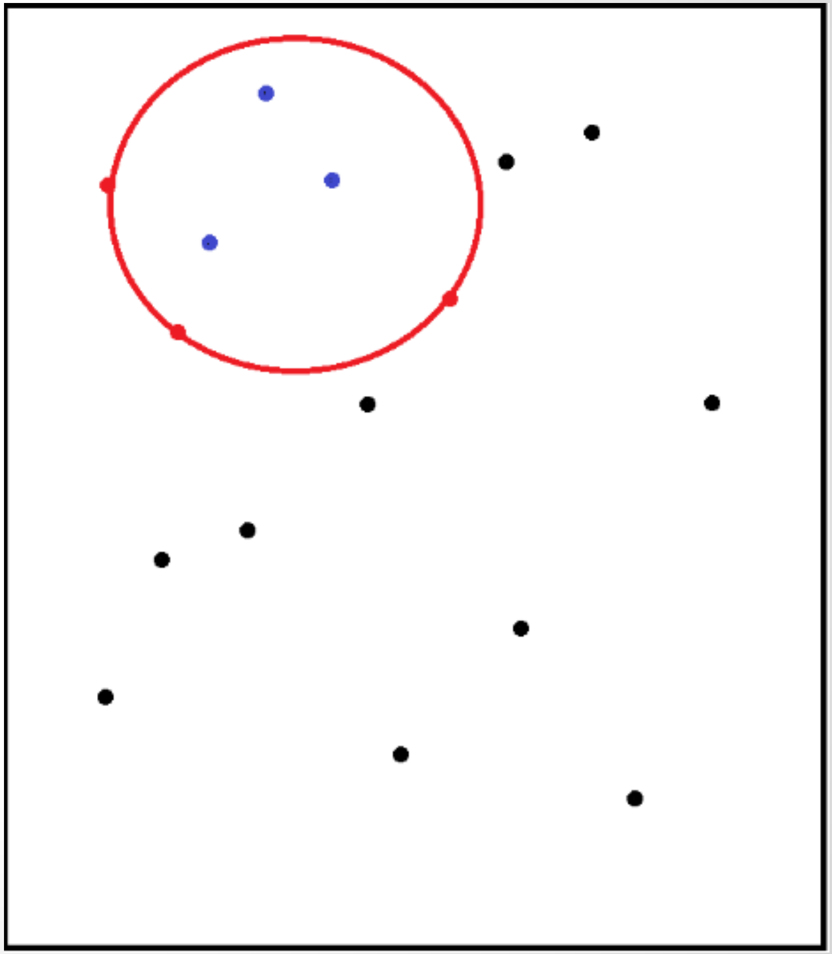
Simple Optimal Algorithm

- With a given set of points
- Pick a combination of 3
- Create a circle from these points
- Count points within the circle
- Save circle's center and radius if points inside = k
- Runs in $O(N^4)$



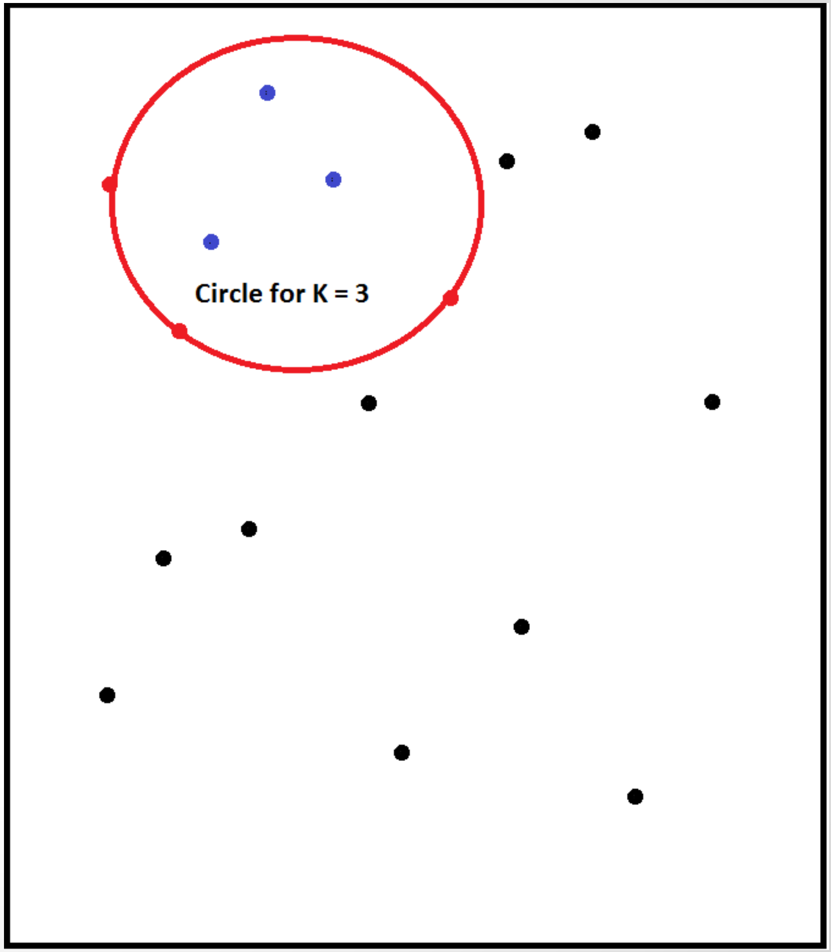
Simple Optimal Algorithm

- With a given set of points
- Pick a combination of 3
- Create a circle from these points
- Count points within the circle
- Save circle's center and radius if points inside = k
- Runs in $O(N^4)$



Simple Optimal Algorithm

- With a given set of points
- Pick a combination of 3
- Create a circle from these points
- Count points within the circle
- Save circle's center and radius if points inside = k
- Runs in $O(N^4)$



Algorithm Implementation Details

The algorithm code was written in C++, using the library CGAL (Computational Geometry Algorithms Library).

CGAL was chosen as it performs well and provided most of the algorithms needed to build our algorithm.

Parallelization

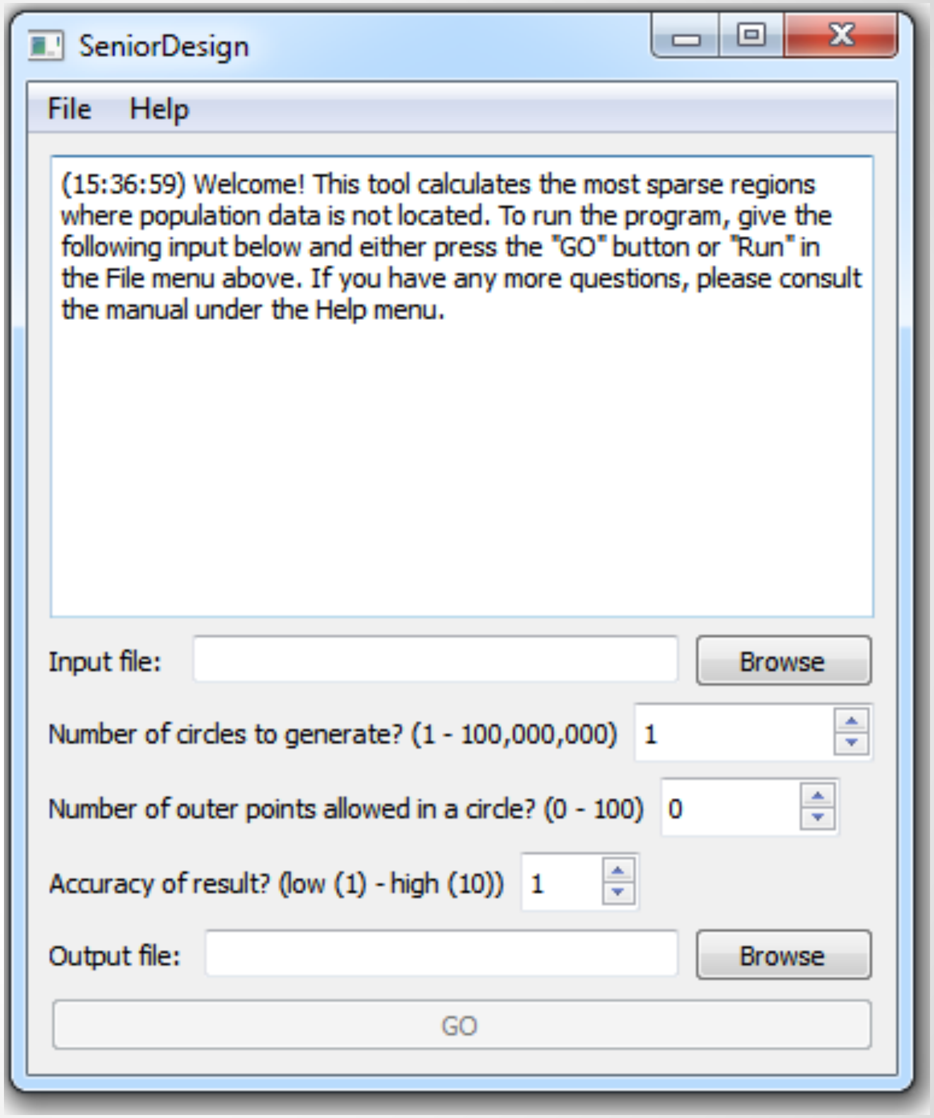
Results without parallelization:

- 2,000,000 sampling points with data set of approx. 350,000 points runs in about 94.2 sec

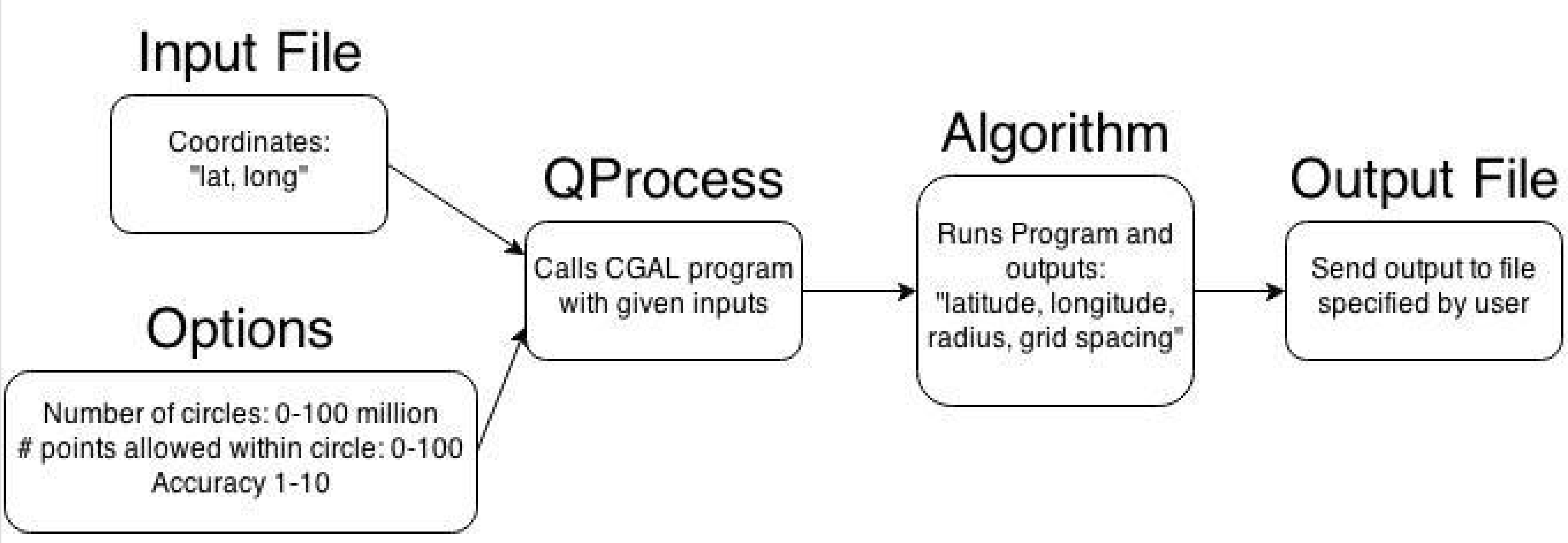
Results with 4 cores in parallelization:

- 2,000,000 sampling points with data set of approx. 350,000 points runs in about 94.1 sec

GUI Design


SeniorDesign
File
Help
(15:36:59) Welcome! This tool calculates the most sparse regions where population data is not located. To run the program, give the following input below and either press the "GO" button or "Run" in the File menu above. If you have any more questions, please consult the manual under the Help menu.
Input file:
Browse
Number of circles to generate? (1 - 100,000,000)
1
Number of outer points allowed in a circle? (0 - 100)
0
Accuracy of result? (low (1) - high (10))
1
Output file:
Browse
GO

Overall Design



Future Possibilities

- A faster optimal algorithm may be useful
- An iterative algorithm could find larger k-circles centered near grid points
- Distribute points with constant memory
- Think statistically; what if the input points are drawn from a larger population?

Demo

Questions

Extra slides start here

Basic Definitions

Geodesic Grid: a technique used to model the surface of sphere with a subdivided polyhedron

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

Basic Definitions

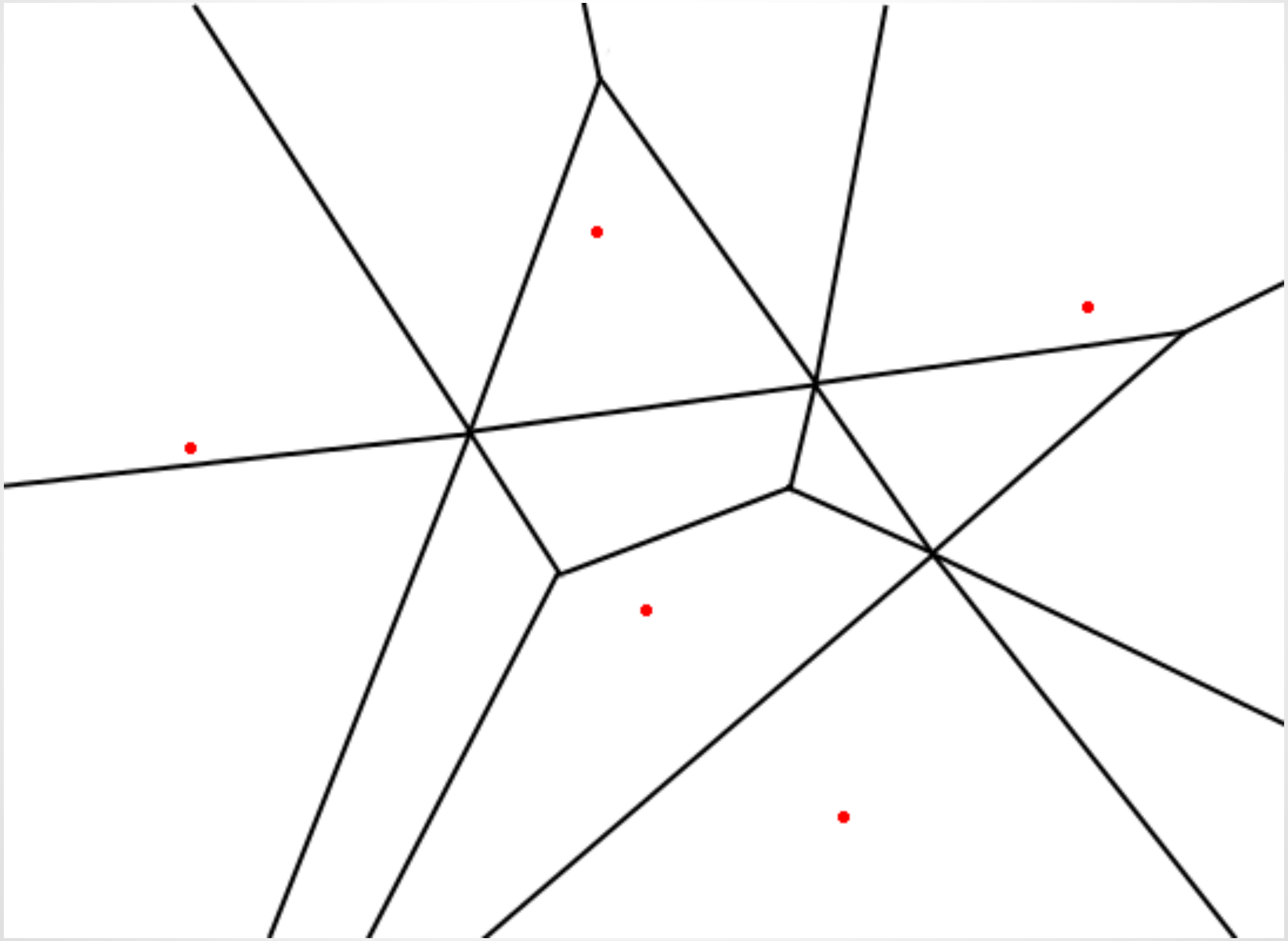
kth Degree Voronoi Diagram: A generalization of Voronoi Diagrams. For any natural number k and given a set of input points, a kth-degree Voronoi diagram associates every input point with the region where that point is kth closest

k-circle: A k-circle is a circle with k input points inside it

Example Output

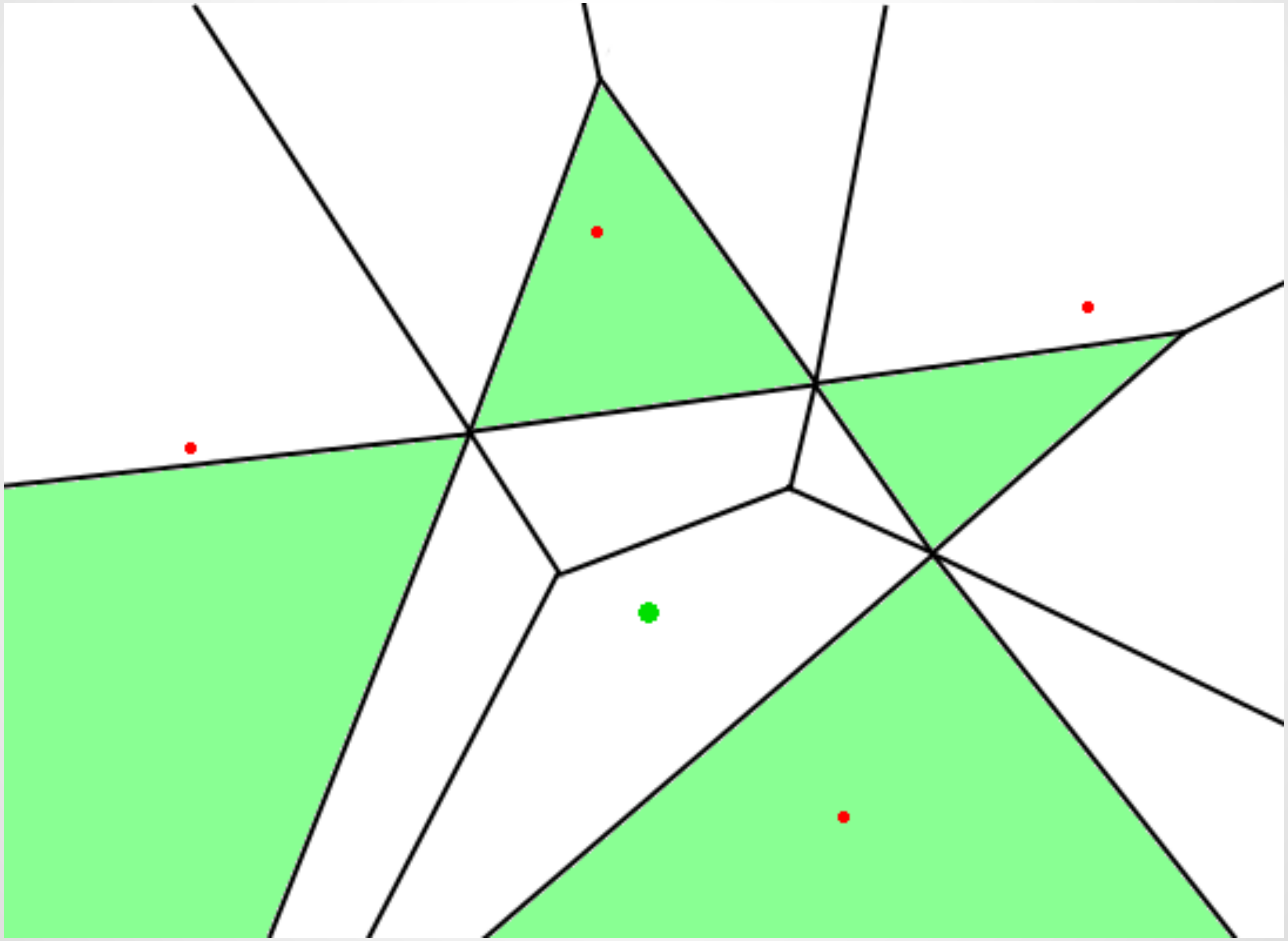
Latitude	Longitude	Radius (m)	Grid spacing (m)
3.3694030	-165.1577028	528304.0110501	4370.4272605
3.3701742	-165.2230401	527766.1398303	4371.1253014
-7.5518313	-1.7331102	527486.3614580	4390.6032108
3.3086040	-165.1926503	527411.5997402	4370.7009520
3.3093587	-165.2579801	527360.3486624	4371.3948051
-7.7376293	-1.8329856	527272.5754768	4390.4780559
1.6433310	150.0065616	527179.1483774	3617.1577611
-3.3504708	121.3624331	527115.6345737	4394.8683768
-7.7377402	-1.7674941	527096.1302746	4390.4134684
-3.7367498	-114.8439781	527085.7350453	4379.2457942

2nd degree Voronoi diagram

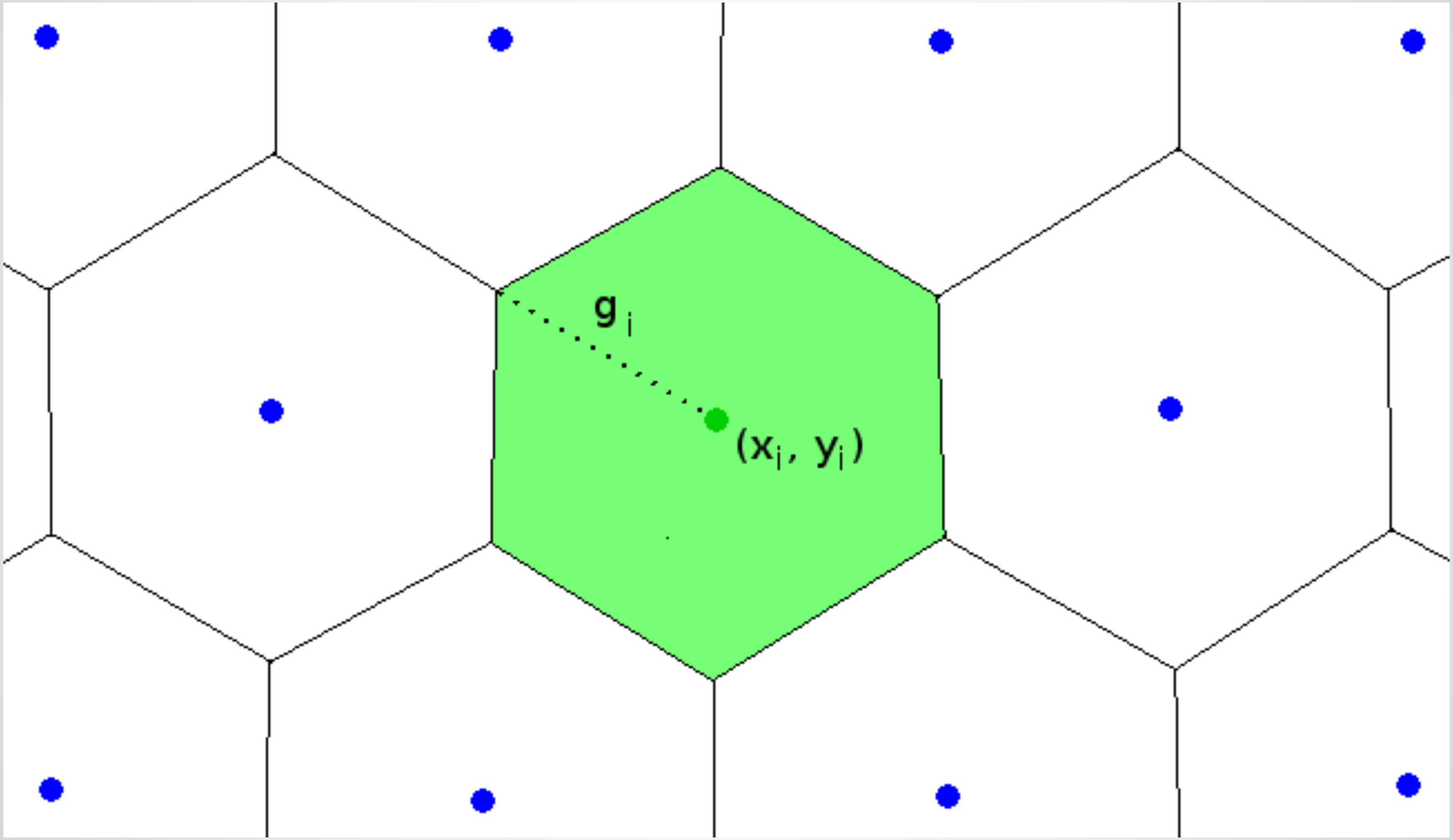


May14-31

2nd degree Voronoi diagram



Grid Spacing



Design Approach

- Numerical optimization
  - Define a function d(x, y) that takes in the (x, y) coordinates of the center of a circle and outputs the largest possible radius it can have without including more than k data points.
  - This is the same as the distance to the (k+1)th closest data point. d(x, y) changes slowly, so its value at sampling points approximate the values at nearby points.

Graphic Visualization Design

- prompts user for window boundaries
- read longitude, latitude, radius values from a txt.file
- draws points and circles to the map
- <latitude> <longitude> <radius>