Nearly antipodal points distance accuracy improvement

Contents

1. Student Information

1.1. Personal Details

- **Name**: Adeel Ahmad
- **University**: National University of Computer and Emerging Sciences, Islamabad
- **Major**: Computer Science
- **Degree**: B.Sc.
- **Email**: adeel.ahmad.3a@gmail.com
- **Homepage**: https://adl1995.github.io
- **GitHub**: https://github.com/adl1995
- **Interests and hobbies**: Running, Cycling, Reading, Cooking, Automating repetitive tasks
- **Timezone**: UTC +05:00

1.1.1. Availability

> *How much time do you plan to spend on your GSoC?*

Since GSoC is equivalent to a full-time position, I plan to spend 40-45 hours per week. During this period, I will provide timely update on my progress.

> *What are your intended start and end dates?*

I will start doing preliminary work on my project once the accepted student proposals are announced i.e. April 23rd. Since this is my final year at college, I can easily work till the final week of GSoC i.e. August 14th.

> *What other factors affect your availability (exams, courses, moving, work, etc.)?*

I have my final exams in the last week of June, so I will have to take a few days off in between. To make up for lost progress, I will put in more time to meet the milestones mentioned in my timeline.

1.2. Background Information

> *Please summarize your educational background (degrees earned, courses taken, etc.).*

I'm currently in my final year of studies towards a bachelor's in Computer Science. My main interest lies towards computer vision and machine learning, and this is what my elective courses revolved around.

In machine learning, I had the opportunity to understand the underlying details of gradient descent (vanilla, mini-batch), linear, softmax and SVM classifiers, convolutional neural networks, and recurrent neural networks (RNNs, LSTM). I also made a blog post explaining the various activation functions used in neural networks [1].

Apart from this, I am also quite enthusiastic about computer vision. My final year project involves implementing a structure from motion pipeline using a monocular camera [2]. To learn all the intricate steps involved in this process, I followed Robotics: Perception course on Coursera [3]. Image processing algorithms have always fascinated me, to quench this fascination I implemented numerous filters (Gaussian, Sobel, Prewitt, Laplacian), edge detectors (Canny, Marr Hildreth), morphological operators (Convex hulling, Erosion, Dilation), and object detectors (Generalized Hough transform, simple convolution) [4, 5]. These were implemented from scratch making use of the `NumPy` Python library. The most interesting algorithm I implemented was Generalized Hough Transform [6]. I was amazed to see how something simple as the equation of line could lead to detection of objects in images.

My résumé [7] contains a more detailed overview on my interests and courses.

> *Please summarize your programming background (OSS projects, internships, jobs, etc.).*

I was a participant in Google Summer of Code 2017 [8] with the OpenAstronomy organization. My project involved the design and development of a Python client for Hierarchical Progressive Surveys (HiPS) [9], which is a scheme for describing astronomical images, source catalogues, and three-dimensional data cubes to manage large volumes of data. The package [10] enabled users to view astronomical figures in an interactive environment. I also maintained an extensive blog [11] throughout this period.

During the tenure of three months, I acquired a great deal of knowledge about software design principles and testing software from a user perspective. One major part of the development process was to write automated test cases and contiguously integrating code using Travis CI. Furthermore, I also composed API and user-level documentation [12] using Sphinx.

Apart from the hips repository [10], where I have been contributor till GSoC 2017, I have made small contributions to these projects: OpenCV [13], libLAS [14], Jailhouse [15], Jupyter notebook [16], and axios [17].

Moreover, I have been a Full-Stack web developer for almost three years, and have worked on multitude of projects ranging for SaaS to e-commerce websites. I work remotely on Upwork [18] and Fiverr [19]. Besides web development, I also work on web automation and data scraping using `Selenium` and `BeautifulSoup`.

The skills that I acquire are through self learning and consistency. My GitHub profile [20] contains some of my other open source work.

> *Please tell us a little about your programming interests. Please tell us why you are interested in contributing to the Boost C++ Libraries.*

Boost is considered to be one of the best crafted C++ libraries. I believe the most important step in software development is the design phase. All libraries associated with Boost provide an extensive design rationale, such as Boost Geometry [21].

Through this experience, I can improve my knowledge regarding software development in general, and it will allow me to collaborate with people from diverse backgrounds. By contributing to Boost, I aim to further improve my skill set, and develop tools that will facilitate users globally.

> *What is your interest in the project you are proposing?*

Various organizations e.g. airports rely on extremely accurate results for the distance between two points. This project will allow the library users to achieve their desired accuracy. Additionally, I will be provided a chance to contribute to a well-known open source project.

> *Have you done any previous work in this area before or on similar projects?*

Apart from the programming competency task [22] that I have been working on for the past month, my previous Google Summer of Code project made extensive use of the celestial coordinate system. One of my task was to convert the input coordinates to be compatible with the tile drawing module [23].

> *What are your plans beyond this Summer of Code time frame for your proposed work?*

I propose to implement an algorithm for computing the similarity between geometries e.g. linestrings and polygons. This is achieved using various techniques, such as Frechet distance and Hausdorff distance. A solution is proposed in [32].

Apart from this, I will actively look into the issues raised by users and further optimize the algorithms, if required.

> Please rate, from 0 to 5 (0 being no experience, 5 being expert), your knowledge of the following languages, technologies, or tools:

- C++ 98/03 (traditional C++): 2, I used this for writing small scripts when starting out as a programmer.
- C++ 11/14 (modern C++): 3.5, I have made use of policy-based design, SFINAE, and type traits.
- C++ Standard Library: 3, I have used STL in almost all my C++ projects.
- Boost C++ Libraries: 2.5, Boost libraries I have worked with: Geometry, Chorno, Algorithm, and Test.
- Git: 4, This is my daily driver. I have experience in resolving merge conflicts, rebasing, and sending patches through email.

> What software development environments are you most familiar with (Visual Studio, Eclipse, KDevelop, etc.)?

I mainly use Vi and Sublime text as my development environment. My main goto compiler is g++. Furthermore, I use Emacs with Org-mode to organize my tasks.

> What software documentation tool are you most familiar with (Doxygen, DocBook, Quickbook, etc.)?

For the programming competency task, I used Doxygen to generate the documentation. This included the Getting Started guide and a detailed description of methods provided by the library. I made use of the default configuration file with a few changes related with adding a module to render LaTeX equations.

I am also familiar with Sphinx, as I used it for creating documentation for my previous year's Google Summer of Code project. One of the more interesting task involving this was to link a Python script to automatically generate an image output and display it to the user in different formats.

2. Project Proposal

2.1. Abstract

Nearly antipodal points refer to the most geographically distant points on a sphere or an ellipsoid i.e. the points are diametrically opposite to each other. Computing the great circle distance between these two points is often a corner case for most geodesic computations, and the distance is either overestimated or underestimated.

2.2. Problem Description

The solution proposed by Vincenty [24] models the Earth as an ellipsoid. It provides accurate distance within a few millimeters. However, it may not provide a solution between two nearly antipodal points, as the solution may never converge, or may converge slowly. To account for this, Vincenty proposed a modified formula [25] for the solution around the backside of the ellipsoid or nearly antipodal points. The modified Vincenty's formula for an ellipsoid with equal major and minor axes is given by:

$$\Delta\sigma = \arctan \frac{\sqrt{(\cos\phi_2 \cdot \sin(\Delta\lambda))^2 + (\cos\phi_1 \cdot \sin\phi_2 - \sin\phi_1 \cdot \cos\phi_2 \cdot \cos(\Delta\lambda))^2}}{\sin\phi_1 \cdot \sin\phi_2 + \cos\phi_1 \cdot \cos\phi_2 \cdot \cos(\Delta\lambda)}$$

But, this can still require thousands of iterations to converge, thus the method may be deemed inefficient. For rapid convergence for all pair of input points, a solution is proposed by Karney [26]. It makes use of the Newton's method [27]. Using this method, only two to four iterations are required for convergence.

2.3. Problem Demonstration

The strategies currently offered by Boost Geometry do not correctly handle corner cases for nearly antipodal points [28]. To illustrate this, two geographically distant points are chosen on the Earth's surface. The values are represented in latitude / longitude pairs (degrees):

```
2.179167    -73.787500
-2.162200    106.139064
```

Using Vincenty's strategy to compute the distance:

```
using namespace boost::geometry;

typedef model::point
    <double, 2, cs::spherical_equatorial
    <degree>> spherical_point;
typedef srs::spheroid<double> stype;

// Define the strategy.
typedef strategy::distance::vincenty<stype> vincenty_type;

spherical_point point1(-73.787500, 2.179167),
                        point2(106.139064, -2.162200);

// The distance is returned in meters.
double d = distance(point1, point2, vincenty_type());
```

The distance is returned as 19963713.360 meters. We compare this with GeographicLib [29], which correctly handles the case for nearly antipodal points. The CLI tool `geod` is passed the latitude and longitude positions of our two points:

```
geod +ellps=WGS84 -I +units=m <<< "2.179167 -73.787500 -2.162200 106.139064"
```

This returns the distance as 20001571.135 meters. Thus, the distance computed using Boost was off my 37857.775 meters.

To illustrate the problem with the current implementation of direct Vincenty strategy, we use the latitude and longitude positions of our first pair of points we used earlier and try to recover the latitude and longitude positions for the second pair or destination. In Boost Geometry, the direct strategy can be specified as follows:

```
using namespace boost::geometry;

// For storing the resulting values.
formula::result_direct<double> result;

// WGS-84 spheroid.
srs::spheroid<double> spheroid(6378137.0, 6356752.3142451793);
```

```
// Define the strategy.
typedef formula::vincenty_direct<double, true, true, true, true> vincenty_direct_type;

result = vincenty_direct_type::apply(-73.787500, 2.179167, 20001571.135, 6.80724316,
spheroid);
```

This gives us the latitude and longitude positions as 0.962372 and -76.9321, respectively. We again compare this with GeographicLib:

```
geod +ellps=WGS84 -I +units=m <<< "2.179167 -73.787500 6.80724316 20001571.135"
```

The resulting values for latitude and longitude positions are -2.16220000 and 106.13906400, respectively. Thus, we can see that the result provided by GeographicLib is almost identical to our input points (second pair), whereas, the result provided by Boost Geometry provides incorrect values.

2.4. Proposed Solution

The issue regarding low accuracy when dealing with nearly antipodal points has already been raised in [28, 30]. This project aims to improve the accuracy for nearly antipodal points by following the method proposed by Karney [26]. To test the current implementation of Boost Geometry, I computed distance between two nearly antipodal points across Vincenty, Thomas, and Andoyer strategy. The least accurate result was provided by Vincenty, as discussed above, whereas, the result provided by Thomas and Andoyer had a difference of 1736.43 meters and 1747.53 meters with the correct result, respectively.

For this project I propose to implement the direct and inverse geodesic problem solutions, which provide the computation of geodesics on an ellipsoid of revolution. The paper proposed by Karney also provides a method for computing the area of a geodesic polygon (a polygon whose sides are geodesics) on an ellipsoid.

Initially the solution will be bundled with Vincenty's algorithm, and rigorous testing will be done to ensure the algorithm provides accurate results for all input types. Once this is done, the Andoyer and Thomas strategy will be updated to correctly handle the case for nearly antipodal points.

2.4.1. Direct Geodesic Problem

In the direct geodesic problem we are provided with the latitude ϕ1, longitude λ1, azimuth α1 at the first point, and the distance s_{12} between the two points. Our goal is to find the find latitude ϕ2, longitude λ2, and azimuth α2 for the second point.

The method starts off by using an auxiliary sphere which allows us to correspond exactly between a geodesic and a great circle on a sphere. On the sphere, the latitude ϕ is replaced with the reduced latitude β (see section 2.4.2.).

We can now solve for the spherical triangle to get α2, ω2, and β2 (this can be used to find the longitude ϕ2) using the following equations:

$$\alpha_0 = \phi(|\cos\alpha + i\sin\alpha\sin\beta| + i\sin\alpha\cos\beta)$$
$$\sigma = \phi(\cos\alpha\cos\beta + i\sin\beta)$$
$$\omega = \phi(\cos\sigma + i\sin\alpha_0\sin\sigma)$$
$$\beta = \phi(|\cos\alpha_0\cos\sigma + i\sin\alpha_0| + i\cos\alpha_0\sin\sigma)$$
$$\alpha = \phi(\cos\alpha_0\cos\sigma + i\sin\alpha_0)$$

Where $i = \sqrt{-1}$. Finally, the longitude λ2 can be found by solving:

$$\lambda = \omega - f\sin\alpha_0 I_3(\sigma)$$

2.4.2. Inverse Geodesic Problem

The inverse problem can be solved using techniques similar to those employed for an ellipsoid of revolution. Our goal here is to find the distance between the two points s_{12}, and the azimuths α1 and α2.

To efficiently solve this problem, Karney's method assumes that α1 is already known i.e. the azimuth at the first point. This is used in addition with the two latitudes ϕ1 and ϕ2, which gives us the longitudinal difference λ12 corresponding to the first intersection of the geodesic with ϕ2. The longitude obtained this way differs from the original value. So, to get accurate results, we need to adjust α1 until we get the correct longitude. This is solved by the hybrid geodesic problem, given by:

$$\cos\alpha_2 = \frac{\sqrt{\cos^2\alpha_1\cos^2\beta_1 + (\cos^2\beta_2 - \cos^2\beta_1)}}{\cos\beta_2}$$

Here β denotes the reduced latitude or parametric latitude. This can be found by:

$$\tan\beta = (1-f)\tan\phi$$

The next step is to compute σ and ω. These can be found by the equations mentioned in section 2.4.1. Using this we can compute the distance s_{12} between the two points.

3. Programming Competency

The programming competency task related with this project was the creation of a header-only library that allowed users to compute distance between two points on the Earth's surface.

The implementation is hosted on GitHub: https://github.com/adl1995/boost-geometry-proposal

3.1. Design

In implementing this, several design choices were made. At first, the library provided two classes `GeoPoint` and `GeoDistance`. The former class contained the latitude and longitude position, while the latter class extended numerous distance methods. However, this did not provide enough flexibility to the user. To account for this, I made use of type traits, similar to Boost Geometry.

The library user now has the ability to define their custom point type which is used for distance computation. The traits have to be specialized with the generic functions `getRadian` and `getDegree` in the `PointTrait` namespace. Assuming a user-defined `CustomPoint` structure holds values in degrees, the specialization can be done as follows:

```
namespace PointTrait
{
   template <>
   struct AccessPoint<CustomPoint, 0>
   {
      static double getRadian(CustomPoint const& p)
      { return p.latitude * M_PI / 180; }

      static double getDegree(CustomPoint const& p)
      { return p.latitude; }
   };
   template <>
   struct AccessPoint<CustomPoint, 1>
   {
      static double getRadian(CustomPoint const& p)
      { return p.longitude * M_PI / 180; }
```

```
        static double getDegree(CustomPoint const& p)
        { return p.longitude; }
    };
}
```

The structure members can now be accessed via `AccessPoint::getRadian<0>(point)`. Also, the GeoLib library now provides distance computation via functions rather than class methods.

3.2. Testing

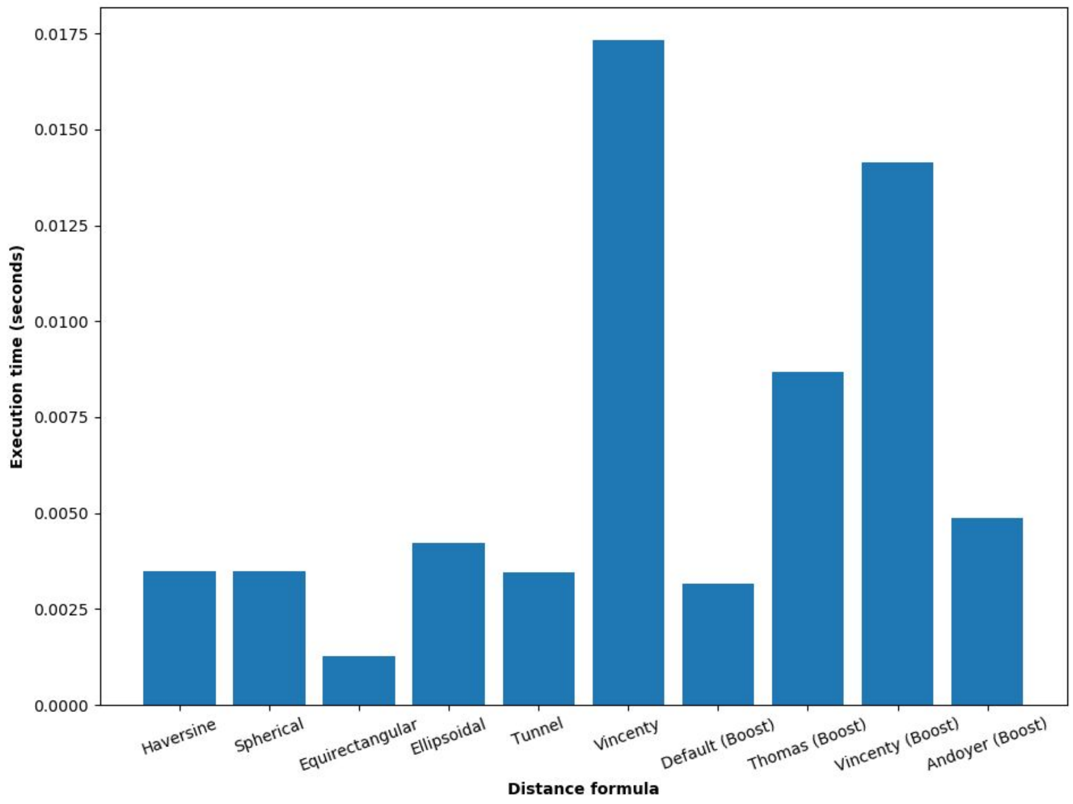
To test the implementation on a large heterogeneous data, the GeodTest dataset [31] was chosen. The testing was done using the Boost Test library. In addition to the functions provided by the library, I also compared the results with Boost Geometry algorithms, namely: Vincenty, Thomas, Andoyer, and Haversine.

3.3. Results

Each algorithm provides results within a unique tolerance. Higher tolerance values represents lower accuracy. The most accurate algorithm is the Vincenty's formulae, while the least accurate being the Ellipsoidal approximation.

It should be noted that testing was done on a diverse dataset, where some distances were large and some were short. So, the algorithms with a higher tolerance might still produce good results for shorter distances. For benchmarking, the execution time was computed using Boot Chrono. The bar chart below tries to visually compare the average execution time for each distance formula over 15000 runs. This was calculated using the g++ compiler (version 7.2.0 on Ubuntu 14.04) with the O1 optimization level.

Note: The default Boost Geometry strategy used here is Haversine. This is because the point types are defined as `spherical_equatorial`, which use Haverine as their default strategy [33].


Execution time (seconds)
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
Haversine
Spherical
Equirectangular
Ellipsoidal
Tunnel
Vincenty
Default (Boost)
Thomas (Boost)
Vincenty (Boost)
Andoyer (Boost)
Distance formula

4. Timeline

Time Period	Milestones
April 24- May 13 **(Community Bonding Period)**	● Understand the intricate details of the algorithm proposed by Karney ● Familiarise myself with the design patterns followed by Boost Geometry, such as SFINAE, lazy / eager evaluation, etc.
Phase 1 begins	
May 14 - May 21 (1 week)	● Start implementing the direct geodesic problem that provides accurate results for nearly antipodal points (this will be bundled with Vincenty's algorithm)
May 22 - May 29 (1 week)	**Final exams**
May 30 - June 14 (2 weeks)	● Finish the implementation of direct geodesic problem ● Add extensive test cases for verifying the implementation
Phase 1 ends **Phase 2 begins**	
June 15 - June 30 (2 weeks)	● Implement the inverse geodesic problem that provides accurate results for nearly antipodal points (this will be bundled with Vincenty's algorithm)
July 01 - July 13 (~2 weeks)	● Add extensive test cases for verifying the implementation

	● Update the Thomas and Andoyer strategy to produce accurate results for nearly antipodal points
Phase 2 ends	
July 14 - July 21 (1 week)	● Improve the API design
July 22 - July 31 (~1 week)	● Benchmark the performance on the basis of different factors e.g. accuracy, execution time, etc.
Aug 01 - Aug 07 (1 week)	● Provide diverse examples to help users get quickly started
Aug 08 - Aug 14 (~1 week)	● Provide API and user-level documentation

References

[1] An overview of activation functions used in neural networks,
https://adl1995.github.io/an-overview-of-activation-functions-used-in-neural-networks.html
[2] Structure from Motion pipeline, https://github.com/reconstruct3d/SfM
[3] Robotics: Perception | Coursera, https://www.coursera.org/learn/robotics-perception
[4] An implementation of Marr Hildreth and Canny edge detector,
https://github.com/adl1995/edge-detectors
[5] Multiple Choice Questions detector, https://github.com/adl1995/exam-mcq-recognition
[6] An implementation of Generalised Hough transform,
https://github.com/adl1995/generalised-hough-transform
[7] Résumé, https://adl1995.github.io/personal/resume.pdf
[8] HiPS client for Python,
https://summerofcode.withgoogle.com/archive/2017/projects/5440053895495680
[9] Hierarchical progressive surveys,
https://www.aanda.org/articles/aa/pdf/2015/06/aa26075-15.pdf
[10] Python astronomy package for HiPS, https://github.com/hipspy/hips
[11] GSoC 2017 blog, https://adl1995.github.io/category/gsoc.html
[12] Getting started page for hips, http://hips.readthedocs.io/en/latest/getting_started.html
[13] Commits for OpenCV, https://github.com/opencv/opencv/commits/master?author=adl1995
[14] Commits for libLAS, https://github.com/libLAS/libLAS/commits/master?author=adl1995
[15] Commits for Jailhouse, https://github.com/siemens/jailhouse/commits/next?author=adl1995
[16] Commits for Jupyter notebook,
https://github.com/jupyter/notebook/commits/master?author=adl1995
[17] Issue for axios, https://github.com/axios/axios/issues/853
[18] Upwork profile, https://www.upwork.com/freelancers/~018e56b8591046f889
[19] Fiverr profile, https://www.fiverr.com/adl1995
[20] GitHub profile, https://github.com/adl1995
[21] Boost Geometry design rationale,
http://www.boost.org/doc/libs/1_66_0/libs/geometry/doc/html/geometry/design.html
[22] Programming Competency task, https://github.com/adl1995/boost-geometry-proposal
[23] HiPS draw module, https://github.com/hipspy/hips/tree/master/hips/draw
[24] Direct and inverse solutions of geodesics on the ellipsoid with application of nested equation, http://www.cqsrg.org/tools/GCDistance/inverse.pdf
[25] Geodetic inverse solution between antipodal points,
https://geographiclib.sourceforge.io/geodesic-papers/vincenty75b.pdf
[26] Algorithms for geodesics, Charles F. F. Karney, https://arxiv.org/pdf/1109.4448.pdf
[27] Newton's method, https://en.wikipedia.org/wiki/Newton's_method
[28] "haversine method is inaccurate", https://svn.boost.org/trac10/ticket/11929

[29] GeographicLib, https://sourceforge.net/projects/geographiclib
[30] “geometry default geographic (andoyer) antipodal distance is zero”, https://svn.boost.org/trac10/ticket/11917
[31] Test set for geodesics, Charles F. F. Karney, https://zenodo.org/record/32156
[32] Similarity Measure Using Hausdorff Distance in 2D Shape Recognition System, by Enayatifar et al., https://www.atlantis-press.com/php/download_paper.php?id=10165
[33] Default strategy Haversine, https://github.com/boostorg/geometry/blob/master/include/boost/geometry/strategies/spherical/distance_haversine.hpp#L329