GSoC2017 Boost.Geometry:Filtering of compare distance predicates Proposal

1.Personal Details

Name: Ruoyun Jing

College/University: Northwest University

Course/Major: Software Engineering

Degree Program: Bachelor of Engineering

Email: jingry0321@gmail.com

Homepage: https://github.com/Rylynnn

Availability:

I will spend at least 40 hours a week for the work. I do not have any conflicts during the official coding periods, because my semester course will ended by the end of April. If time permits, I will try to make efforts doing more work beyond my proposal. Furthermore,I am willing to keep contributing to Boost and Boost.Geometry after this GSoC period.

2.Background Information

Education Background

I'm a third-year undergraduate student majoring in Software Engineering at Northwest University, China. I have taken many courses including:

Computer Science

- Principles of Computer Organization
- Operating System
- Software Engineering
- Artificial Intelligence
- Data Structure and Algorithm

Mathematics

- Linear Algebra
- Advanced Mathematics
- Discrete Mathematics
- Number Theory

Programming Background

I've been writing C++ since I entered university, and I became to a contestant of algorithm competition since first semester. For the study of algorithms, I am familiar with C++ and STL.

Operating systems

Ubuntu 16.04 LTS, Windows 10, Mac OS X

Languages

C,C++,Pascal,Python,Java,C#,HTML,CSS,JavaScript

Algorithms

Search, Graph Theory, Computational Geometry, etc.[1]

Programming Contest

Bronze Medal, 46th, ACM International Collegiate Programming Contest(ACM-ICPC) Asia Shenyang Regional Contest, Dec, 2016.

Bronze Medal, 52th, China Collegiate Programming Contest(CCPC) Hefei Regional Contest, Dec, 2016.

Programming Interest

I extremely enjoy learning and spending time on algorithms using C++, and I also want to challenge my programming capabilities in different aspects. When I found Boost, a set of algorithms libraries for C++, I felt this is a good opportunity for me. Honestly, I did not have lots of experience contributing to open source community, however, I will try my best to communicate and learn!

Because I am good at mathematical algorithms like geometry, I decided to propose the project of Boost.Geometry. After I finished the Programming Competency Test from Boost.Geometry, I read the source code of Boost.Geometry, I found there have plenty of basic geometry formula. It will be better if we can reduce the time complexity by calculating for different conditions or using other methods. So I decide to contribute to project1 to avoid expensive geographic distance calculation whenever possible.

Previous Work

I have finished the following work since I decided to contribute to Boost:

- Learn C++ meta-programming and understand the architecture of Boost.Geometry.
- Learn Boost.Geometry functions with documents and source code.
- Learn knowledge of geodesic from differential geometry and map projection.

Future Plan

I am willing to continue developing and contributing to Boost.Geometry and Boost community . And I would like to expand and optimize other functions from Boost, the work I

[1] https://github.com/Rylynnn/Arithmetic

do will indeed improve engineering ability and abundant my experience of coding for open source community.

Skillng Core

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++) (3.5, using frequently before during coding to solve algorithms' problems and courses' work)
- C++ 11/14 (modern C++) (4, frequently use during coding to solve algorithms' problems and courses' work)
- C++ Standard Library (4, frequently use during coding to solve algorithms' problems)
- Boost C++ Libraries (2.5, never used them before GSoC 2017, but understand the architecture and learn functions)
- Git (3.5, frequently use during study)

Dev Environment

- Visual Studio for C# development
- Eclipse for JAVA development
- Code Blocks, Sublime Test + Gcc/Clang for C++ devlopment.

Documentation Tools

I tried to used Doxygen during my curriculum design of Software Engineering. If there is need for me to use other software documentation tools, I would like to learn!

3.Project Proposal

Introduction

In some algorithms there is the need to compare two distances of two point pairs, just like . Especially, computing distances on ellipsoid in Boost.Geometry used compare_distance (p_1, p_2, q_1, q_2)[2] function, which is a predicate that is it returns three possible values (larger than, less than, equal). Boost.Geometry has 3 strategies for distance: andoyer, tomas, vincenty.. Although Newton's method has been successfully used to give rapid convergence for this formula, the time complexity of this algorithm is hard to estimate.

To reduce the time complexity, this project will try to use the following approaches:

1. **Condition Division:** Compute the Euclidean distances[3] first, if the length of two segments are very close, defined not “far enough” means “very close”, then fall back

[2] Boost.Geometry comparable_distance-1.63.0
[3] Euclidean distance - Wikipedia

to expensive geographic distance calculation, otherwise return the result by comparing those numbers obtained by less expensive cartesian computation.

2. **Calculation with The Area of a Sector of an Ellipse:** Get cross-section through the center of ellipsoid and the geodesic segments, then calculate the area, which is easy to compare distances.

(this is low priority and may be tested if you have the time.)Calculation with Performing a local spheroid approximation and return the 2D distance by map projection: Perform a local spheroid approximation, then use methods of map projection to calculate the distances. In this project, I use Azimuthal equidistant projection, accommodating equatorial, polar, oblique and Two-point equidistant projection make the 3D point to 2D with reliable functions, then do approximate calculation.**Related problems**

1.Compare azimuths: Given p1, p2, q1, q2 we want to compare azimuth(p1,p2) vs. azimuth (q1,q2). The result again will be {larger than, less than, equal}. This problem appear in relational operations in Boost Geometry. See for example https://github.com/boostorg/geometry/blob/develop/include/boost/geometry/strategies/geographic/intersection.hpp#L336 where we test if two points are on the same side of some geodesic segment.

2.Not directly related but a useful reference for filtering in computational geometry is the following: https://www.cs.cmu.edu/~quake/robust.html

Milestones and Schedule

Community Bonding Period (May 4 to May 29, 2017):

1. Understand Boost.Geometry libraries and architecture by source code and documents.
2. Make a wiki page (TODO lists and weekly reports).
3. Learn meta-programming and other essential knowledge of C++.
4. Get familiar with development[4] for Boost community.
5. Understand the algorithms of map projection and mathematical calculation on ellipsoid.

Official Coding Period (May 30 to August 21, 2017):

Official Coding Period Phase 1 (May 30 to June 26, 2017):

Week 1 (May 30 to June 5, 2017):

1. Learn how to create documentation and tests.
2. Create initial documentation.

Week 2 to 3 (June 6 to June 19, 2017):

1. Implement **Condition Division.**
2. Check the section to fit the architecture of Boost.
3. Create documentation for these section.

[4] Boost Development

Week 4 (June 20 to June 26, 2017):

1. Design the test data, and review the code.
2. Analyze the test data to define when fall back to expensive geographic distance calculation or return the result by comparing approximately.

Milestone 1: Finish implementation of **Condition Division** and define “far enough”.

Official Coding Period Phase 2 (June 27 to July 24, 2017):

Week 5 (June 27 to July 10, 2017):

1. Learn differential geometry, check the geodesic segments of two points whether can get a cross-section through the center of ellipsoid, and whether the cross-section is ellipse that we can calculate the sectorial area easy to compare.
2. Create documentation for this section.

Week 6 to 8 (July 11 to July 24, 2017):

1. Implement the algorithm of **Calculation with Sectorial Area**.
2. Design the test data, and review the code.
3. Analyze the test data, guarantee the correct of algorithms.

Milestone 2: Finish implementation of **Calculation with Sectorial Area**.

Official Coding Period Phase 3 (July 25 to August 21, 2017):

Week 9 to 10 (July 25 to August 7, 2017):

1. Implement the architecture of **Calculation with Performing a local spheroid approximation and return the 2D distance by map projection.**
2. Implement the algorithm of Azimuthal equidistant projection accommodate equatorial.
3. Implement the algorithm of Azimuthal equidistant projection accommodate polar.
4. Implement the algorithm of Azimuthal equidistant projection accommodate oblique.
5. Implement the algorithm of Two-point equidistant projection
6. Create documentation for this section.

Week 11 (August 8 to August 14, 2017):

1. Design the test data, and review the code.
2. Analyze the test data, guarantee the correct of algorithms.

Week 12 (August 15 to August 21, 2017):

1. Fix bugs and documentation details.
2. Prepare for final delivery.

Milestone 3: Finish implementation of **Calculation with Performing a local spheroid approximation and return the 2D distance by map projection**.

Versions

macOS 10.12.3

g++ 5.3

g++ 6

Clang++ 3.8

Clang++ 3.9

Boost 1.64.0 (from github develop branch)

Boost.Geometry 1.64.0 (from github develop branch)

Deliverables

The deliverables would be:

1. Implementation of **Condition Division** and define the range of Division on Boost.Geometry.
2. Implementation of **Calculation with Sectorial Area** on Boost.Geometry.
3. Implementation of **Calculation with Performing a local spheroid approximation and return the 2D distance with four kinds of map projection** on Boost.Geometry.
4. Documentation and tests for the implementations.

Define the appropriate range of “far enough”

To define the appropriate range of “far enough”, here are four steps to do:

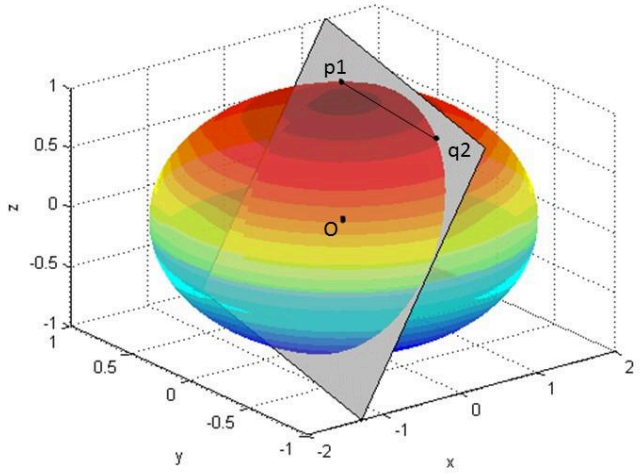
1. Implement function compare_distance_Euclidean with Euclidean distances.
2. Test the result between function compare_distance (p_1, p_2, q_1, q_2) and compare_distance_Euclidean (p_1, p_2, q_1, q_2).
3. Adjust parameters to limit deviation in an acceptable range (maybe it would be 1e-6 corresponds, the accurate value should be discussed by specific condition).
4. Define “far enough” with adjusted parameters.

Mathematical definition

In Cartesian coordinates, if $p_1 = (p_{1x}, p_{1y}, p_{1z})$ and $p_2 = (p_{2x}, p_{2y}, p_{2z})$ are two points in Euclidean 3-space, then the distance d from p_1 to p_2, or from p_2 to p_1 is given by the Pythagorean formula:

$$d(p_1, p_2) = \sqrt{(p_{1x} - p_{2x})^2 + (p_{1y} - p_{2y})^2 + (p_{1z} - p_{2z})^2}$$

Example



Calculation with Sectorial Area

To calculate with Sectorial Area, here are four steps to do:

1. Cut the ellipse with a planar cross-section passing through the center of the ellipsoid, and points p_1 & p_2.
2. Cut the ellipse with a planar cross-section passing through the center of the ellipsoid, and points q_1 & q_2.
3. Calculate the area of sector p_1Op_2 and q_1Oq_2.
4. Compare the area between sector p_1Op_2 and q_1Oq_2 to two distances of two point pairs.

Mathematical definition

In Cartesian coordinates, if $p_1 = (p_{1x}, p_{1y}, p_{1z})$ and $p_2 = (p_{2x}, p_{2y}, p_{2z})$are two points in Euclidean 3-space. The cross-section passing through the center of the ellipsoid, and points p_1 & p_2 is ellipse.

Then $p_1 = (p_{1x}, p_{1y})$, $p_2 = (p_{2x}, p_{2y})$, and the implicit equation of the ellipse[5] p_1Op_2 is:

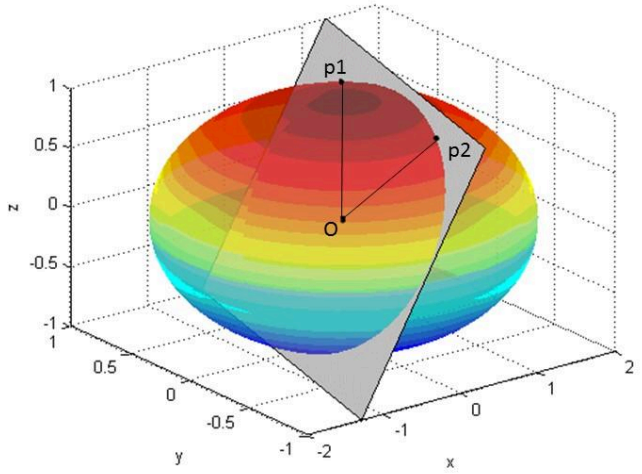
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

The area of sector p_1Op_2 is:

$$S_{p_1Op_2} = \frac{ab}{2}(arctan\frac{ap_{1y}}{bp_{1x}} - arctan\frac{ap_{2y}}{bp_{2x}})$$

Example

[5] Ellipse - Wikipedia



Map Projection[6]

A map projection is a systematic transformation of the latitudes and longitudes of locations on the surface of a sphere or an ellipsoid into locations on a plane. Many properties can be measured, just like area, shape, direction, distance, etc. Map projections can be constructed to preserve at least one of properties, though only in a limited way for most. The purpose of the map determines which projection should form the base for the map.

Here are some projections which can be constructed to preserve distance based on spheroid approximation:

1. Azimuthal equidistant projection[7] which can accommodate all aspects: equatorial, polar, and oblique. Distances for all aspects are accurate from the center point outward.
2. Two-point equidistant projection[8] which shows the true distance from either of two chosen points to any other point on a map.Correct from either of two chosen points to any other point on the map.

To calculate with performing a local spheroid approximation and return the 2D distance by map projection, here are steps to do:

1. For most of these projections use longitude and latitude to calculate, I should understand the algorithm of projections and implement them with cartesian coordinate.
2. Classify the two points pairs based on different projections.
3. Calculate the distance with the center point outward.

---

[6] Map Projection - Wikipedia

[7] Azimuthal Equidistant--Help | ArcGIS for Desktop

[8] Two-Point Equidistant--Help | ArcGIS for Desktop

4. Compare the approximate distance.

4.Programming Competency

Boost.Geometry programming competency test

I have finished the Boost.Geometry programming competency test on Boost GSoC2017 page:

> PROJECT 1 and 3 Implement a library that provides:
> a representation of points on Earth's surface
> a function to compute the shortest distance between two given points

The source code and test case are on my github repository.

I have implemented the test with two different model of Earth: the spherical model and the ellipsoidal model, and test them as my test case. The two implementations have three constructed functions of points on Earth's surface, which provide constructing points between longitude & latitude and cartesian coordinate.

The spherical model used Great circle distance[9], which need cartesian coordinates to calculate an equation with 3 multiplications, 2 additions, 1 minus, 1 trigonometric computation. O(1). And the ellipsoidal model used the Vincenty formula. Vincenty formulae need geodesic coordinates to calculate. Newton's method has been successfully used to give rapid convergence for all pairs of input. During calculate with Newton's method, there are 21 multiplications, 4 subtractions, 7 additions, 5 minus, 3 trigonometric functions, 1 radication in loop, 27 multiplications, 4 subtractions, 9 additions, 7 minus, 8 trigonometric functions out of loop, and the number of loop is hard to estimate but at least 1000. Above all, the time complexity of this algorithm is hard to estimate.

Comparision between `earth_point.hpp` and Boost.Geometry `vincenty_inverse.hpp`

During this test, I applied the Vincenty formula to calculate the distance. Compared with Boost.Geometry, I found that Boost.Geometry used `vincenty_inverse.hpp` to calculate distance between points and `vincenty_direct.hpp` to calculate the destination point, in my code, it just can finished calculate distance between points. After I read the source code of them, the implementation of `vincenty_inverse.hpp` and `earth_point.hpp` are similar, and I extracted more constants out of the Newton iteration loop which will reduce constant coefficient. About the constructed function, I finished constructed the point on the Earth's surface more accurate with longitude, latitude and height, but Boost.Geometry just constructed points with longitude and latitude. And in earth_point.hpp you can construct point with three methods and set points with Redian system or Angle system which provide more convenient for users.

I tested `earth_point.hpp` and Boost.Geometry `vincenty_inverse.hpp` with 12 test cases[10] on timing and distance result.

---

[9] Great-circle distance - Wikipedia
[10] Boost.Geometry-geometry/test/strategies/vincenty.cpp

Test Case	P1(longitude, latitude)	P2(longitude, latitude)	`earth_point.hpp` distance result(km)	`earth_point.hpp` timing(ms)	Boost.geometry `vincenty_inverse.hpp` distance result(km)	Boost.geometry `vincenty_inverse.hpp` timing(ms)
Flinders Peak -> Buninyong	(144.425,-37.951)	(143.926,-37.6528)	55.015606	16	54.972271	215
Lodz -> Trondheim	(19.4667,51.7833)	(10.35,63.3833)	1399.033666	20	1399.032724	18
London -> New York	(0.1275,51.5072)	(-74.0058,40.7128)	5602.041895	16	5602.044851	14
Shanghai -> San Francisco	(121.5,31.2)	(-122.417,37.7833)	9899.678374	12	9899.698550	17
N	(0,0)	(0,50)	5540.847041	15	5540.847042	23
S	(0,0)	(0,-50)	5540.847041	18	5540.847042	17
E	(0,0)	(50,0)	5565.974540	16	5565.974540	16
W	(0,0)	(-50,0)	5565.974540	15	5565.974540	16
NE	(0,0)	(50,50)	7284.879297	15	7284.879297	9
sub-polar	(0,89)	(1,80)	1005.150874	16	1005.150874	13
no point difference	(4,52)	(4,52)	0	0	0	0
normal case	(4,52)	(3,40)	1336.027219	14	1336.027219	37

Algorithm Problem codes.

I have been code for solving several algorithm problems, during my competation experiences.

- Competation online
- Practice code for algorithm