Distance Computation on Boost.Geometry

Vissarion Fisikopoulos

FOSDEM 2018

Talk outline

Boost.Geometry

- Part of Boost C++ Libraries
- Header-only
- C++03 (conditionally C++11)
- Metaprogramming, Tags dispatching
- Primitives, Algorithms, Spatial Index
- Standards: OGC SFA
- used by MySQL for GIS

How to Get Started?

- Documentation: `www.boost.org/libs/geometry`
- Mailing list: `lists.boost.org/geometry`
- GitHub: `github.com/boostorg/geometry`

Hello World!

```
#include <boost/geometry.hpp>
#include <boost/geometry/geometries/geometries.hpp>
#include <iostream>
namespace bg = boost::geometry;
int main() {
  using point = bg::model::point
                <double, 2, bg::cs::geographic<bg::degree>>;

  std::cout << bg::distance(
             point(23.725750, 37.971536),   //Athens, Acropolis
             point(4.3826169, 50.8119483)); //Brussels, ULB
}
```

Hello World!

```
#include <boost/geometry.hpp>
#include <boost/geometry/geometries/geometries.hpp>
#include <iostream>
namespace bg = boost::geometry;
int main() {
  using point = bg::model::point
                <double, 2, bg::cs::geographic<bg::degree>>;

  std::cout << bg::distance(
             point(23.725750, 37.971536),   //Athens, Acropolis
             point(4.3826169, 50.8119483)); //Brussels, ULB
}
```

result=2088.389 km

Hello World!

```
#include <boost/geometry.hpp>
#include <boost/geometry/geometries/geometries.hpp>
#include <iostream>
namespace bg = boost::geometry;
int main() {
  using point = bg::model::point
                <double, 2, bg::cs::geographic<bg::degree>>;

  std::cout << bg::distance(
               point(23.725750, 37.971536),   //Athens, Acropolis
               point(4.3826169, 50.8119483)); //Brussels, ULB
}
```

result=2088.389 km

Main questions for today:

- how accurate is this result?
- how fast can we obtain it?

Models of the earth and coordinate systems

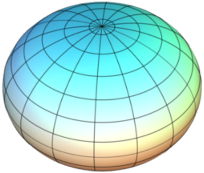
- Flat

Models of the earth and coordinate systems

- Flat
- Sphere (*Widely used e.g.* `google.maps`)

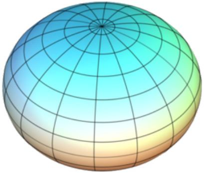
Models of the earth and coordinate systems

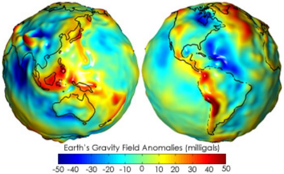
- Flat
- Sphere (*Widely used e.g.* `google.maps`)
- Ellipsoid of revolution (*GIS state-of-the-art*)



Models of the earth and coordinate systems

- Flat
- Sphere (*Widely used e.g.* `google.maps`)
- Ellipsoid of revolution (*GIS state-of-the-art*)
- Geoid (*Special applications, geophysics etc*)





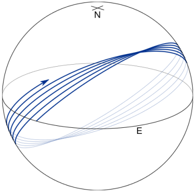
Coordinate systems in Boost.Geometry

```
namespace bg = boost::geometry;

bg::cs::cartesian

bg::cs::spherical_equatorial<bg::degree>
bg::cs::spherical_equatorial<bg::radian>

bg::cs::geographic<bg::degree>
bg::cs::geographic<bg::radian>
```

Geodesics

Definition: Geodesic = shortest path between a pair of points

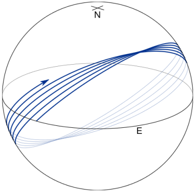
- flat: geodesic = straight line
- sphere: geodesic = great circle
- ellipsoid: geodesic = not closed curve (*except meridians and equator*)

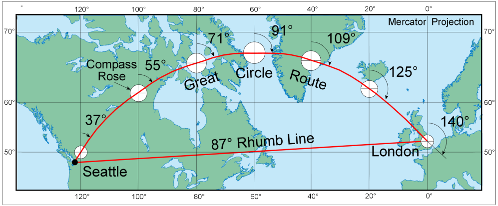


Geodesics

Definition: Geodesic = shortest path between a pair of points

- flat: geodesic = straight line
- sphere: geodesic = great circle
- ellipsoid: geodesic = not closed curve (*except meridians and equator*)





Note: loxodrome or rhump line is an arc crossing all meridians at the same angle (=azimuth). These are straight lines in Mercator projection and not shortest paths.

Distance between points

flat: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ (Pythagoras)

Distance between points

flat: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ (Pythagoras)

sphere: $\mathsf{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\mathsf{hav}(\lambda_2 - \lambda_1)$, where
$\mathsf{hav}(\theta) = \sin^2(\frac{\theta}{2})$ (Haversine formula)

λ, ϕ: longitude, latitude

Distance between points

flat: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ (Pythagoras)

sphere: $\mathrm{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\mathrm{hav}(\lambda_2 - \lambda_1)$, where
$\mathrm{hav}(\theta) = \sin^2(\frac{\theta}{2})$ (Haversine formula)

ellipsoid:

$$\frac{s}{b} = \int_0^\sigma \sqrt{1 + k^2 \sin^2 \sigma'}\, \mathrm{d}\sigma', \tag{1}$$

$$\lambda = \omega - f \sin \alpha_0 \int_0^\sigma \frac{2 - f}{1 + (1 - f)\sqrt{1 + k^2 \sin^2 \sigma'}}\, \mathrm{d}\sigma'. \tag{2}$$

where λ, ϕ are longitude, latitude, s the distance and
$k = e' \cos \alpha_0$ and f, e', b constants

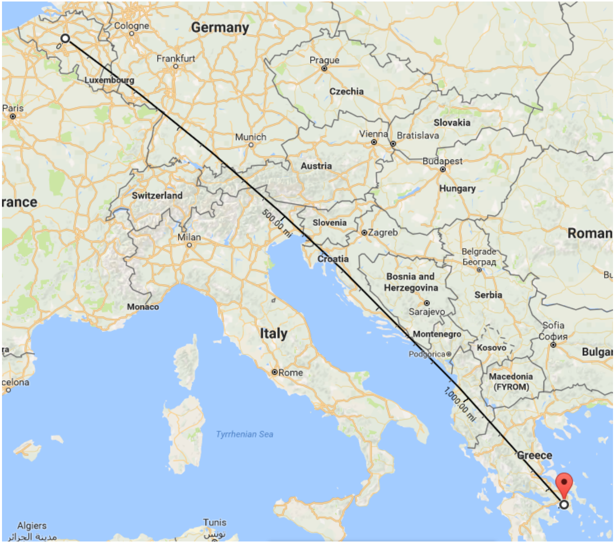
Distance computation algorithms

- Vincenty *(state-of-the-art iterative method)*
- Thomas *(series approximation order 2)*
- Andoyer *(series approximation order 1)*
- Elliptic arc *(great elliptic arc)* (order 0, 1, 2, 3)
- Flat Earth approximation
- Spherical

Code: `https://github.com/boostorg/geometry/pull/431`

Distance point-point example

How far away from home ?



Distance example I

How far away from home ?

```
namespace bg = boost::geometry;
typedef bg::model::point<double, 2,
                         bg::cs::geographic<bg::degree> > point;

typedef bg::srs::spheroid<double> stype;
typedef bg::strategy::distance::thomas<stype> thomas_type;

std::cout << bg::distance(
               point(23.725750, 37.971536), //Athens, Acropolis
               point(4.3826169, 50.8119483),//Brussels, ULB
               thomas_type())
          << std::endl;
```

Distance example results

vincenty	2088384.36606831
andoyer	2088389.07865908
thomas	2088384.36439399
elliptic-0	2087512.07128654
elliptic-1	2088385.16226434
elliptic-2	2088384.42226239
elliptic-3	2088384.4215802
spherical	2085992.80103907
flat approx	2213471.75971644

Distance example II: Brussels center polygon to ULB

Distance polygon-point example

Brussels center polygon to ULB

```
namespace bg = boost::geometry;
typedef bg::model::point<double, 2,
                         bg::cs::geographic<bg::degree> > point;

point p(4.3826169, 50.8119483);   // ULB
bg::model::polygon<point> poly;
bg::read_wkt("POLYGON((4.346693 50.858306,
                       4.367945 50.852455,
                       4.366227 50.840809,
                       4.344961 50.833264,
                       4.338074 50.848677,
                       4.346693 50.858306))", poly);
std::cout << bg::distance(poly, p) << std::endl;
```

Distance point-polygon example results

andoyer	3365.56502748817
thomas	3365.54629915499
vincenty	3365.54630190101
spherical	3361.88636450697
elliptic-0	3363.88943439124
elliptic-1	3365.54806340098
elliptic-2	3365.54630563542
elliptic-3	3365.54630383983
flat approx	3344.76309172576

Exotic distance computation: Caracas!

(10, -66) — (10.1,-66.001)

andoyer	4541.72061848821
thomas	4541.75261545522
vincenty	4541.75261516954
spherical	4523.98895663893
elliptic-0	4535.41683179165
elliptic-1	4541.75263210892
elliptic-2	4541.75263210867
elliptic-3	4541.75263210867
flat approx	4541.81163222931

Accuracy vs. Performance: experimental study

Motivation:

- function that given a user accuracy utilizes the most efficient method
- distance computation in the core of demanding tasks: NN, similarity of curves (e.g. GPS trajectories)

Accuracy vs. Performance: experimental study

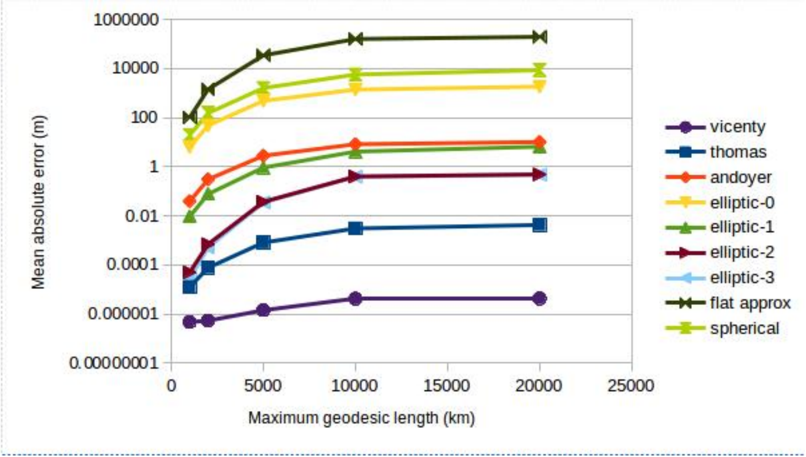
Motivation:

- function that given a user accuracy utilizes the most efficient method
- distance computation in the core of demanding tasks: NN, similarity of curves (e.g. GPS trajectories)

Dataset:

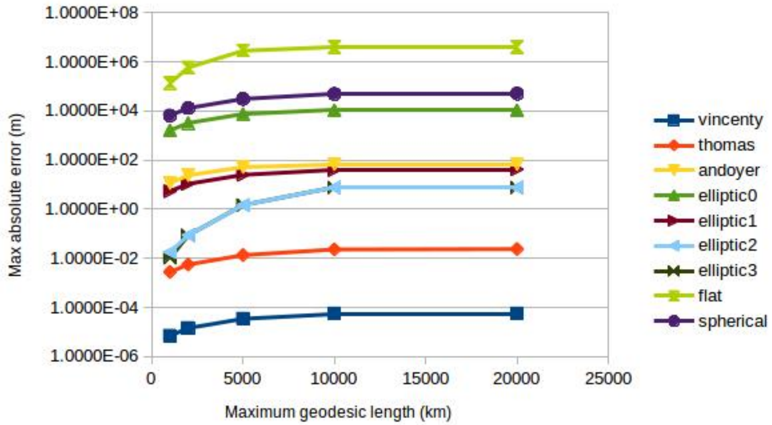
- `https://zenodo.org/record/32156#.WOyaMrUmv7C`
- set of 500K geodesics for the WGS84 ellipsoid
- computation using high-precision and GeographicLib

Accuracy: mean absolute error



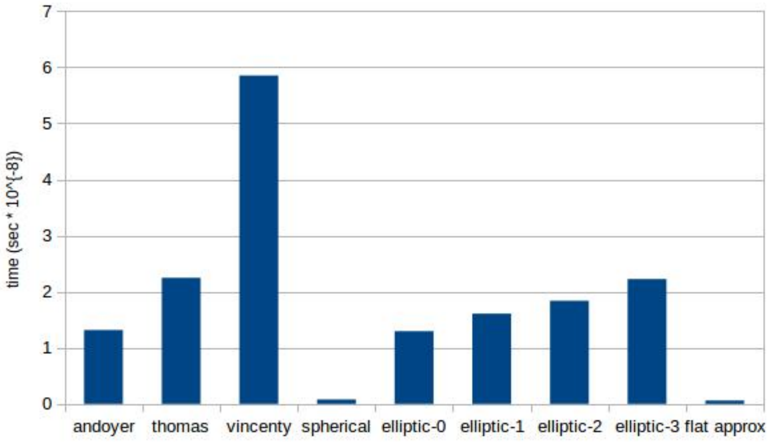
*tests on 100K geodesics with randomly distributed points

Accuracy: max absolute error



*tests on 100K geodesics with randomly distributed points

Performance



Some conclusions

- vincenty most accurate 2x, 5x slower than thomas, andoyer
- elliptic arc: time-accuracy trade-off between andoyer and thomas
- flat earth approximation fastest BUT accurate only for short distances close to equator

Related work

- Distance between any two geometries
- Point clouds snap to geometries
- Similarity of geometries (Hausdorff, Fréchet distance)

Boost Geometry GSoC'18

Topics:

1. Similarity between geometries
2. Nearly antipodal points distance accuracy improvement
3. R-tree serialization

More details:

https://github.com/boostorg/boost/wiki/Boost-Google-Summer-of-Code-2018

Thank you! Questions?