Geography on Boost.Geometry

The Earth is not flat (but it's not round either)

Vissarion Fisikopoulos

`fisikop@gmail.com`

FOSDEM, Brussels, 2017

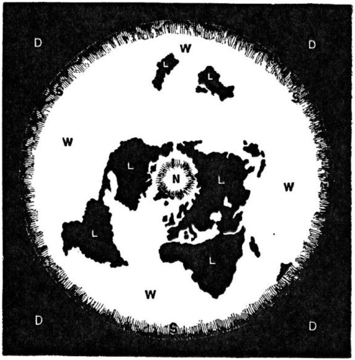
Talk outline

Geodesic algorithms in Boost.Geometry

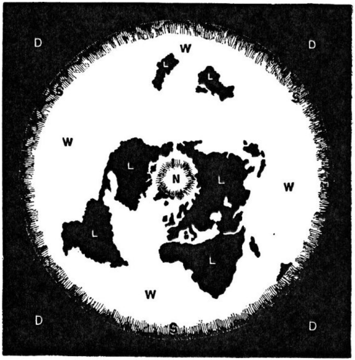
Examples using Boost.Geometry

Discussion

Flat Earth



Flat Earth


D
W
D
S
S
W
L
N
L
W
L
L
W
D
S
D

EARTH NOT A GLOBE
THE FLAT EARTH SOCIETY
theflatearthsociety.org
IN VERITATE VICTORIA

Models of the earth and coordinate systems

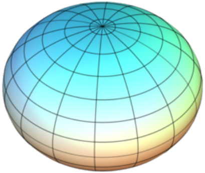
- Flat
  Accurate only locally. Euclidean geometry. Very fast and simple algorithms.

Models of the earth and coordinate systems

- Flat
  Accurate only locally. Euclidean geometry. Very fast and simple algorithms.
- Sphere
  Widely used (e.g. `google.maps`*). Not very accurate. Fast algorithms.*

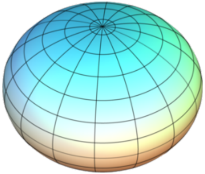
Models of the earth and coordinate systems

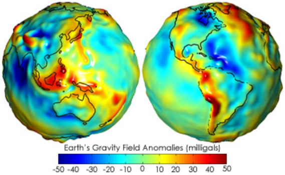
- Flat
  Accurate only locally. Euclidean geometry. Very fast and simple algorithms.
- Sphere
  Widely used (e.g. `google.maps`*). Not very accurate. Fast algorithms.*
- Ellipsoid of revolution (or shperoid or ellipsoid)
  State-of-the-art in GIS. More involved algorithms.



Models of the earth and coordinate systems

- Flat
  Accurate only locally. Euclidean geometry. Very fast and simple algorithms.
- Sphere
  Widely used (e.g. `google.maps`*). Not very accurate. Fast algorithms.*
- Ellipsoid of revolution (or shperoid or ellipsoid)
  State-of-the-art in GIS. More involved algorithms.
- Geoid
  Special applications, geophysics etc





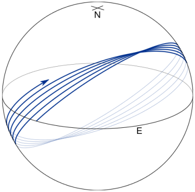
Coordinate systems in Boost.Geometry

```
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

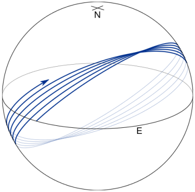
- flat: geodesic = straight line
- sphere: geodesic = great circle
- ellipsoid: geodesic = not closed curve (*except meridians and equator*)

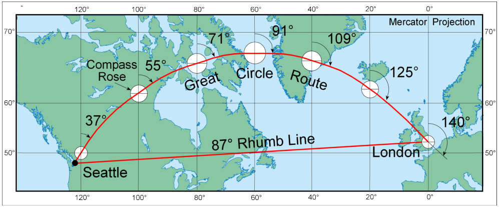


Geodesics

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.

Geographic algorithms

Two main geodesic problems

- direct: given point p, azimuth a and distance s compute point q and distance s from p on the geodesic defined by p, a
- inverse: given two points compute their distance and corresponding azimuths

Geographic algorithms

Two main geodesic problems

- direct: given point p, azimuth a and distance s compute point q and distance s from p on the geodesic defined by p, a
- inverse: given two points compute their distance and corresponding azimuths

Algorithms:

- core geodesic algorithms: point-point distance, area, intersection, envelope, point-segment distance, segment-segment distance
- higher level algorithms: geometry-geometry distance, set operations between geometries (union, intersection etc), relational operations among geometries (contains, crosses, disjoint etc)

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: $(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\,\mathrm{hav}(\lambda_2 - \lambda_1)$ (Haversine formula)

λ, ϕ: longitude, latitude

Distance between points

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

sphere: $(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\,\mathrm{hav}(\lambda_2 - \lambda_1)$ (Haversine formula)

ellipsoid:

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

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

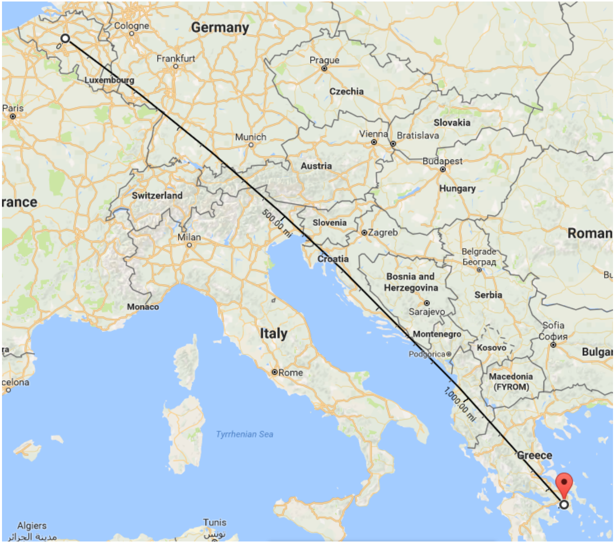
where λ, ϕ are longitude, latitude, s the distance and $k = e' \cos \alpha_0$ and f, e', b constants

Geodesic computation in Boost.Geometry

- Different formulas are selected w.r.t. the coordinate system
- 3 different algorithms for distance on ellipsoid implemented as strategies (andoyer, thomas, vincenty)
  → time-accuracy trade-offs
- State-of-the-art approach:
  closed formula for the spherical solution plus small ellipsoidal integral approximation (series expansion or ~~numerical integration~~)

Distance example

How far away from home ?



Distance example

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

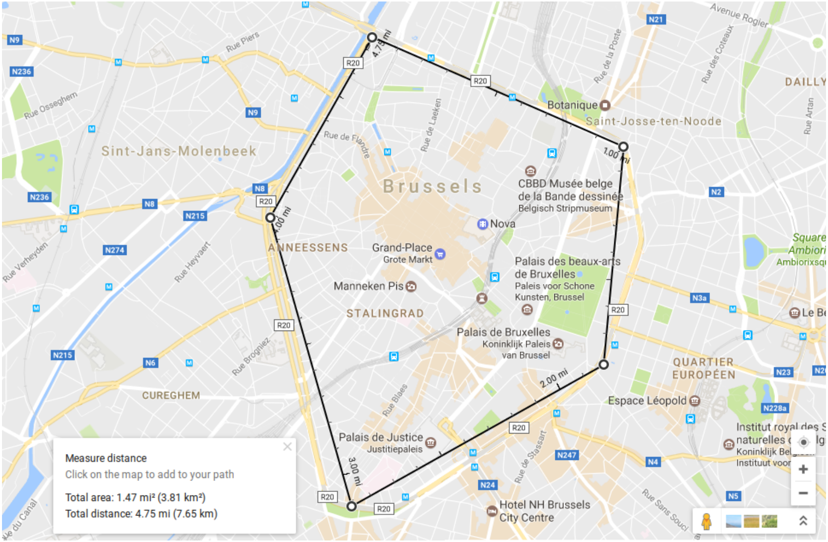
spherical	2,085.993 km *
spherical	2,088.327 km **
geographic (andoyer)	2,088.389 km
geographic (thomas)	2,088.384 km
geographic (vincenty)	2,088.385 km
google maps	2,085.99 km

* radius = 6371008.8 (mean Earth radius)

** radius = 6378137 (WGS84 major axis)

Area example

Brussels center polygon



Measure distance

Click on the map to add to your path

Total area: 1.47 mi^2 (3.81 km^2)

Total distance: 4.75 mi (7.65 km)

Area example

Brussels center polygon

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

bg::strategy::area::geographic<
                                  point,
                                  bg::formula::vincenty_inverse
                              > geographic_vincenty;

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::area(poly, geographic_vincenty)
          << std::endl;
```

Area example results

spherical	3.81045 km^2 *
spherical	3.81898 km^2 **
geographic (andoyer)	3.84818 km^2
geographic (thomas)	3.82414 km^2
geographic (vincenty)	3.82413 km^2
google maps	3.81 km^2

* radius = 6371008.8 (mean Earth radius)

** radius = 6378137 (WGS84 major axis)

Performance

- Expect: spherical $<$ geographic (andoyer) $<$ geographic (thomas) $<$ geographic (vincenty)
- No detailed performance analysis done yet
- Some timings appear on `github` Boost.Geometry pull requests

Similar work

`GeographicLib`

- C++ library that implements ellipsoidal distance, area and projections
- robust and fast
- used by posGIS $>= 2.2.0$
- lack of variety of algorithms e.g. intersection, point-segment distance etc.

Future work

- More geodesic algorithms on ellipsoid: segment-segment distance, projections, convex hull, centroid, ...
- Distance of nearly antipodal points in geographic algorithms
- Google summer of code proposals :-/

Thank you!

Questions?