Latest developments in Boost Geometry

Vissarion Fisikopoulos

FOSDEM 2019

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!

```cpp
#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

A bit more than “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
                 bg::strategy::distance::vincenty<>() );
}
```

A bit more than "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
                bg::strategy::distance::vincenty<>() );
}
```

result=2088389 m

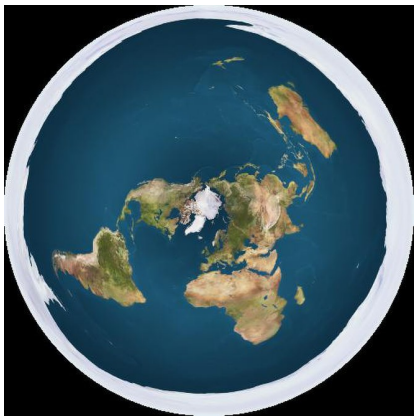
result with strategy=2088384 m

Boost Geometry Algorithms= CS-independent part + CS-specific part (strategies)

Models of the earth and coordinate systems

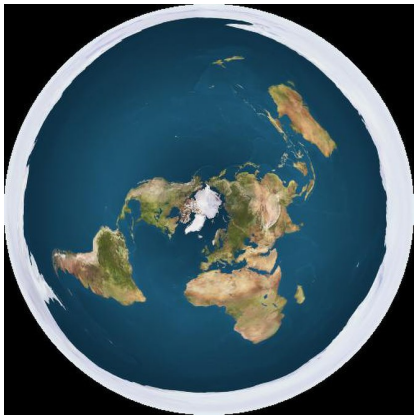
- Flat

  `boost::geometry::cs::cartesian`



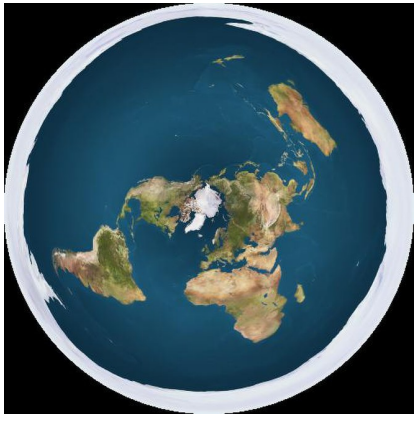
Models of the earth and coordinate systems

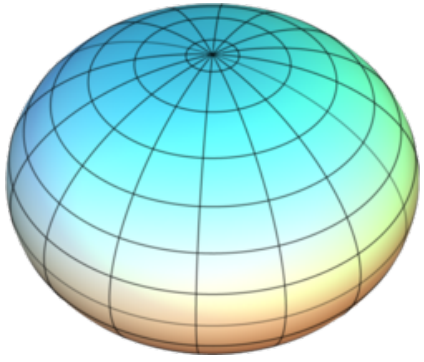
- Flat
  `boost::geometry::cs::cartesian`
- Sphere (*Widely used e.g.* `google.maps`)
  `boost::geometry::cs::spherical_equatorial<bg::degree>`
  `boost::geometry::cs::spherical_equatorial<bg::radian>`



Models of the earth and coordinate systems

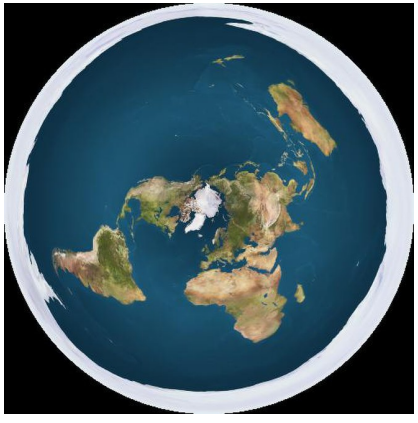
- Flat
  `boost::geometry::cs::cartesian`
- Sphere (*Widely used e.g.* `google.maps`)
  `boost::geometry::cs::spherical_equatorial<bg::degree>`
  `boost::geometry::cs::spherical_equatorial<bg::radian>`
- Ellipsoid of revolution (*geographic GIS state-of-the-art*)
  `boost::geometry::cs::geogrphic<bg::degree>`
  `boost::geometry::cs::geogrphic<bg::radian>`

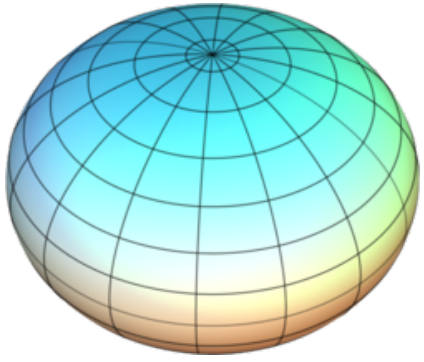


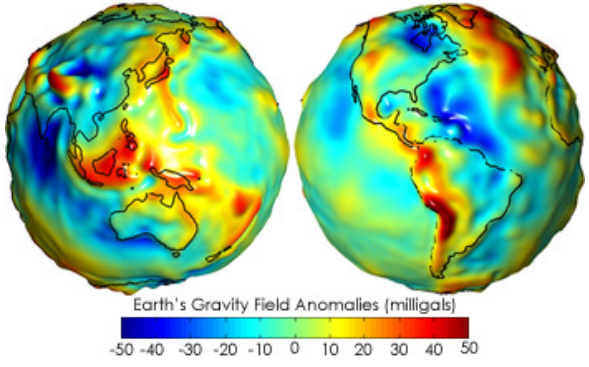


Models of the earth and coordinate systems

- Flat
  `boost::geometry::cs::cartesian`
- Sphere (*Widely used e.g.* `google.maps`)
  `boost::geometry::cs::spherical_equatorial<bg::degree>`
  `boost::geometry::cs::spherical_equatorial<bg::radian>`
- Ellipsoid of revolution (*geographic GIS state-of-the-art*)
  `boost::geometry::cs::geogrphic<bg::degree>`
  `boost::geometry::cs::geogrphic<bg::radian>`
- Geoid (*Special applications, geophysics etc*)







Geodesic problems

Direct given point, azimuth, distance compute new point

Inverse given two points compute distance, azimuth

Methods in Boost Geometry:

- Vincenty *(state-of-the-art iterative method)*
- Thomas *(series approximation order 2)*
- Andoyer *(series approximation order 1)*
- Spherical *(trigonometric approximation)*
- Higher series approximation *(up-to order 8)* *
- Projection + cartesian methods **

* Pull request: `https://github.com/boostorg/geometry/pull/431`

** Attend Adam Wulkiewicz talk at 15:55 !

Example 1: distance

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);
bg::strategy::distance::geographic_cross_track<bg::strategy::vincenty> str;
std::cout << bg::distance(poly, p, str) << std::endl;
```

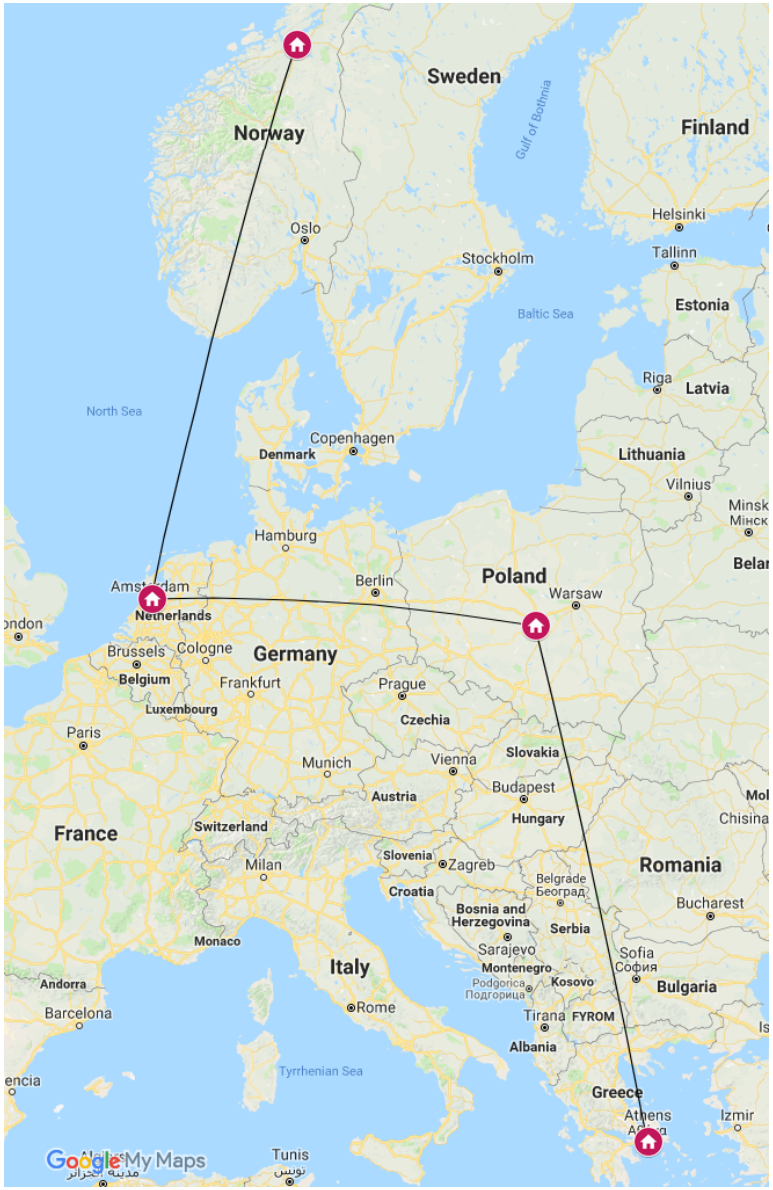
result	3365.565029 m
result with strategy	3365.541672 m

Example 2: Distance nearly-antipodal points

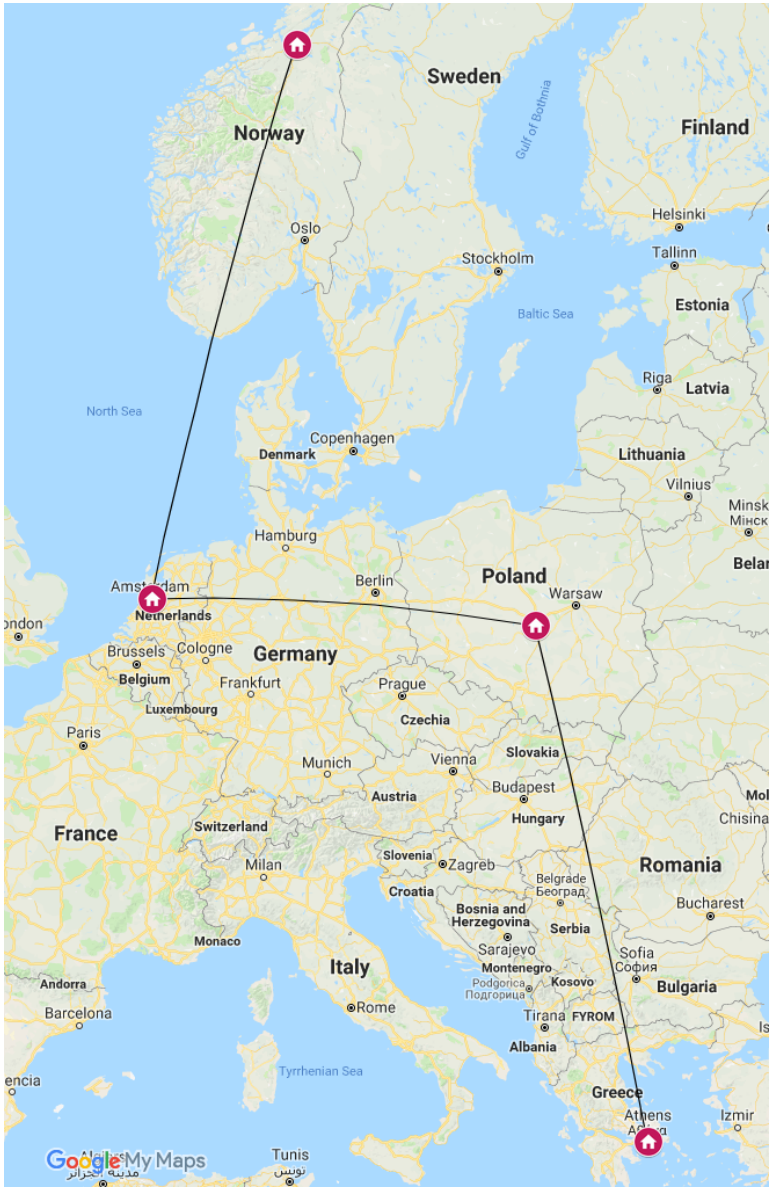
```
namespace bg = boost::geometry;
typedef bg::model::point<double, 2,
                         bg::cs::geographic<bg::degree> > point;

std::cout << bg::distance(
               point(-73.787500,  2.179167),
               point(106.139064, -2.162200),
               bg::strategy::distance::karney<>() );
```

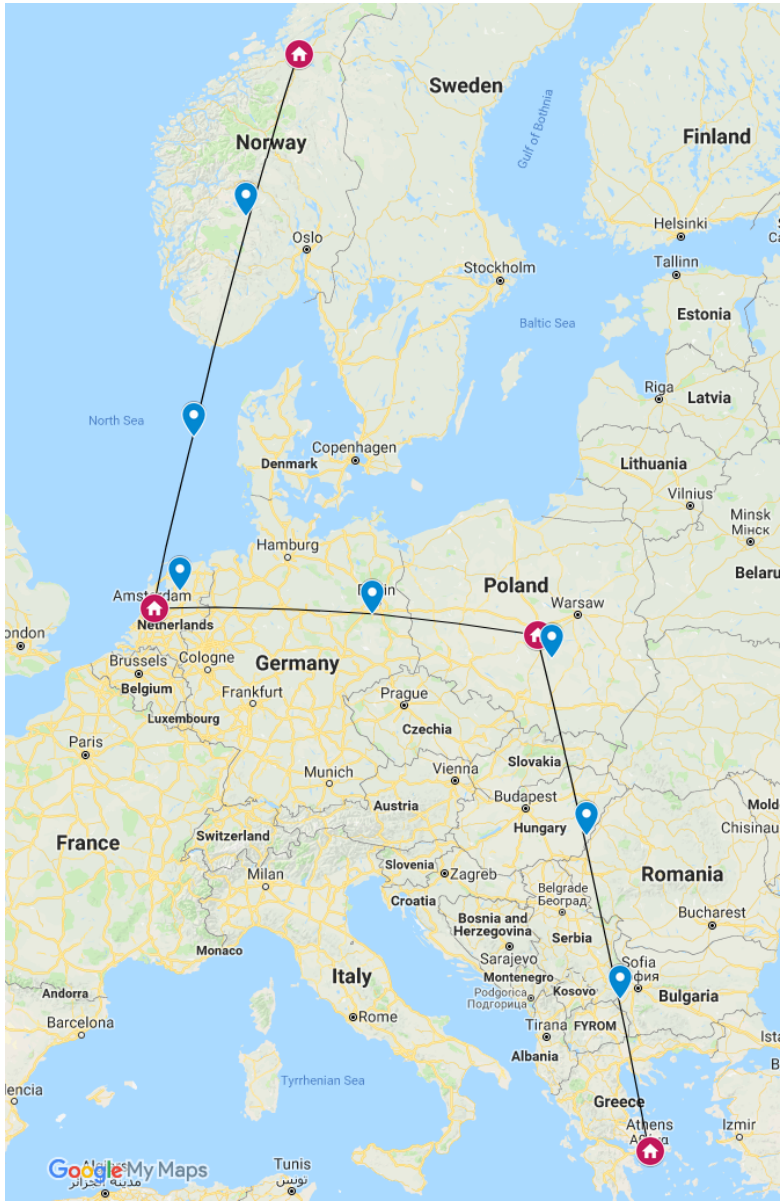
result	20027376.31 m	> 25 km difference
result with strategy	20001571.51 m	

Example 3: Line interpolate points



Example 3: Line interpolate points





Example 3: Line interpolate points

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

using linestring_type = bg::model::linestring<point>;
using multipoint_type = bg::model::multi_point<point>;

linestring_type l {{23.7275, 37.9838}, //Athens
                   {19.4560, 51.7592}, //Lodz
                   {4.9036, 52.3680}, //Amsterdam
                   {10.3951, 63.4305}}; //Trondheim
multipoint_type mp;

bg::line_interpolate_point(l,500000.0,mp);
std::cout << wkt(mp) << std::endl;
```

Example 3: Line interpolate points

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

using linestring_type = bg::model::linestring<point>;
using multipoint_type = bg::model::multi_point<point>;

linestring_type l {{23.7275, 37.9838}, //Athens
                   {19.4560, 51.7592}, //Lodz
                   {4.9036, 52.3680}, //Amsterdam
                   {10.3951, 63.4305}}; //Trondheim
multipoint_type mp;

bg::line_interpolate_point(l,500000.0,mp);
std::cout << wkt(mp) << std::endl;
```

Output: MULTIPOINT((22.57502513 42.39930172),(21.33184367

46.81043626),(19.97378916 51.21703093),(13.19702028

52.24398028),(5.88051868 52.7698247),(6.416288309

56.16967713),(8.386069808 60.53816825))

Example 4: Area

Belgium

Example 4: Area

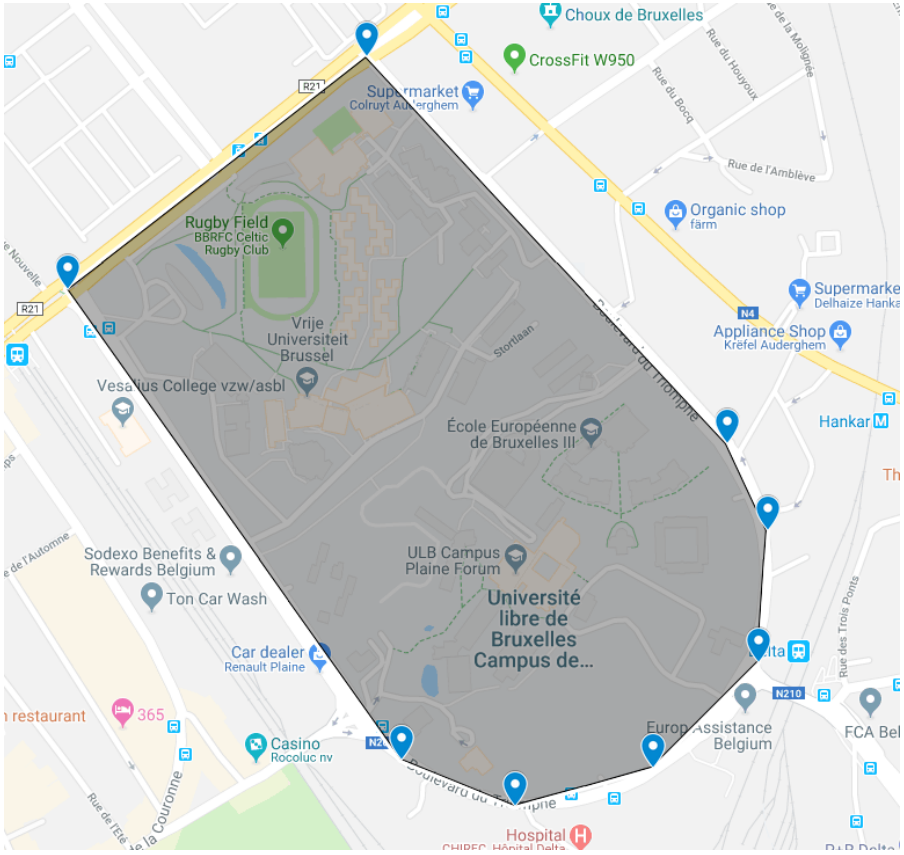
Belgium

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

bg::read_wkt("POLYGON((2.54166319863 51.0911090428 3.37086325679 51.373854107 \
               3.43986325156 51.2447821855 3.89541801404 51.2056910007 \
               4.23890019286 51.3504270229 4.32770008701 51.290127276 \
               4.25236303814 51.3751450878 5.03847307305 51.4869450919 \
               5.23897322671 51.2622820907 5.84713612726 51.1531911176 \
               5.63881817249 50.8488820987 6.01180012435 50.7572730935 \
               6.27041821549 50.6198541375 6.39820016212 50.3231729857 \
               6.13440926324 50.1278451369 5.9730542739 50.170000075 \
               5.74777319107 49.9074911044 5.89916310333 49.6627732262 \
               5.80788233348 49.5450452048 5.47278222549 49.508882154 \
               4.86847332403 49.8022182559 4.82472716541 50.1675641263 \
               4.51055415514 49.9474912228 4.1492361245 49.9783731717 \
               4.16500013748 50.2830541795 3.29708205679 50.5243001057 \
               3.15857311809 50.7843642428 2.65055417947 50.8161090248 \
               2.54166319863 51.0911090428))", poly);
std::cout << bg::area(poly, bg::strategy::area::geographic
                            <
                              bg::strategy::andoyer
                            >()) << std::endl;
```

andoyer	30822.6163 km^2
thomas	30819.5846 km^2
vincenty	30819.5844 km^2
series	30875.8950 km^2

Example 4: Area

ULB



Example 4: Area

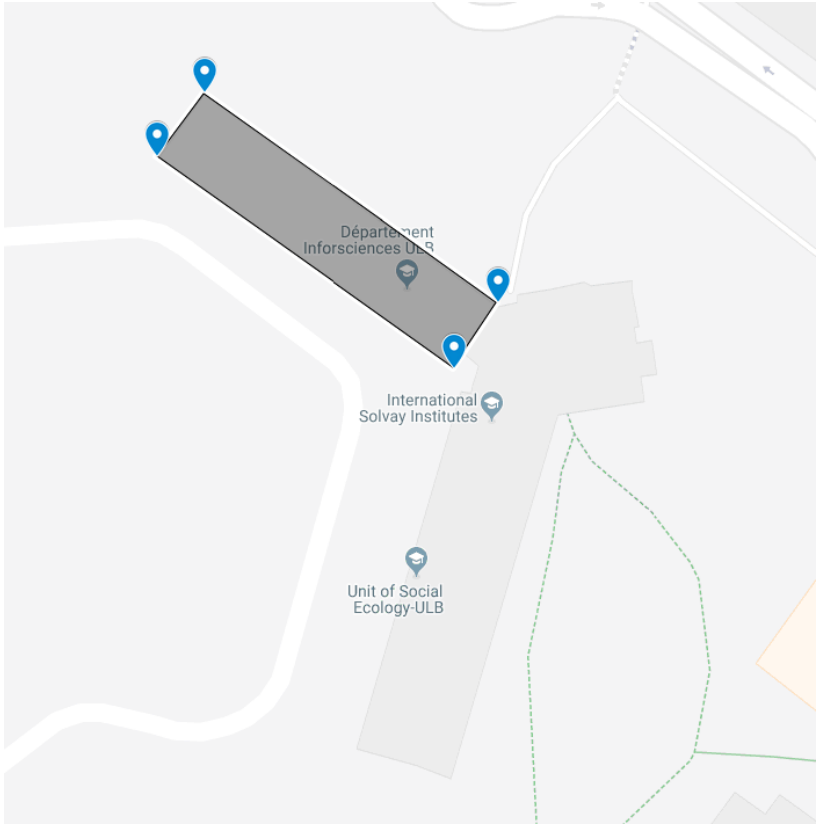
ULB

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

bg::model::polygon<point> poly;
bg::read_wkt("POLYGON((4.395971 50.8256969\
                4.4026444 50.8211695\
                4.4033954 50.8201528\
                4.4032237 50.8186074\
                4.4012711 50.8173737\
                4.3987176 50.8169264\
                4.3966148 50.8174686\
                4.3904349 50.8229724\
                4.395971 50.8256969))", poly);
  std::cout << bg::area(poly, bg::strategy::area::geographic
                                <
                                  bg::strategy::andoyer
                                >()) << std::endl;
```

andoyer	493001 m^2
thomas	495767 m^2
vincenty	495800 m^2
series	494520 m^2

Example 4: Area

ULB building



Example 4: Area

ULB building

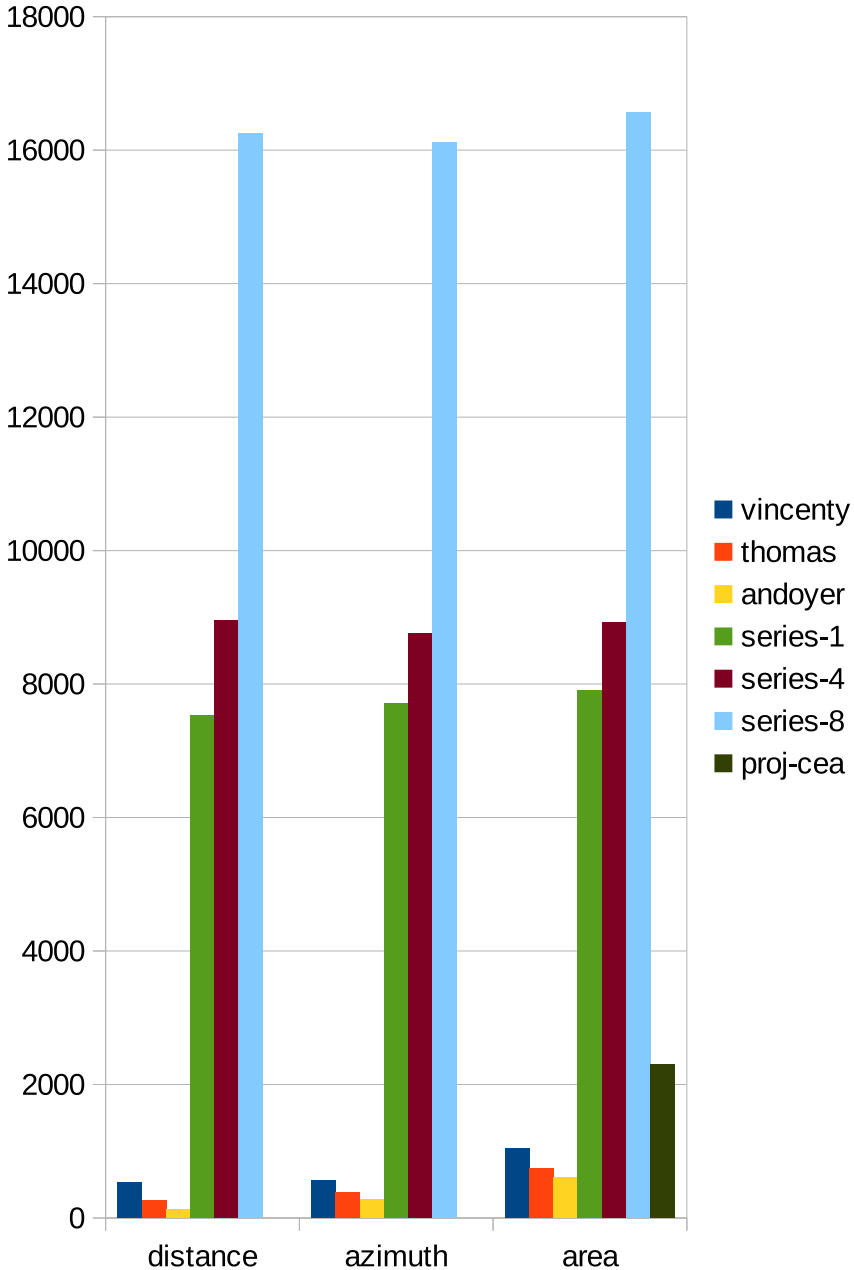
```
namespace bg = boost::geometry;
typedef bg::model::point<double, 2,
                         bg::cs::geographic<bg::degree> > point;

bg::model::polygon<point> poly;

bg::read_wkt("POLYGON((4.3969779 50.8204388\
                4.3976565 50.8201321\
                4.3975546 50.8200355\
                4.3968679 50.8203456\
                4.3969779 50.8204388))", poly);
std::cout << bg::area(poly, bg::strategy::area::geographic
                            <
                              bg::strategy::andoyer
                            >()) << std::endl;
```

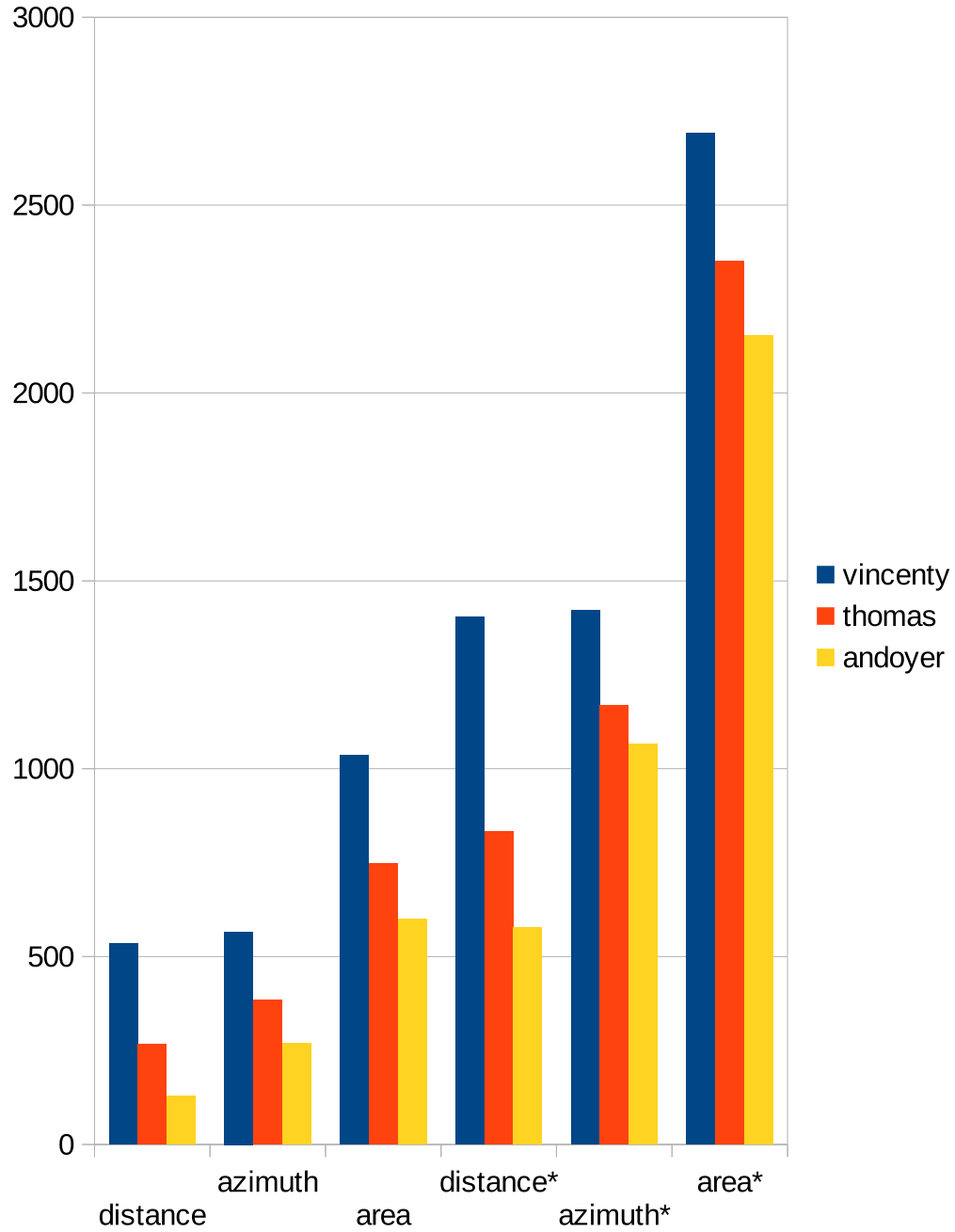
andoyer	979 m^2
thomas	763 m^2
vincenty	850 m^2
series	774 m^2

Benchmarks

- Dataset:
  `https://zenodo.org/record/32156#.WOyaMrUmv7C`
- 1M geodesics, WGS84 ellipsoid
- `https://github.com/vissarion/geometry/wiki/Accuracy-and-performance-of-geographic-algorithms`

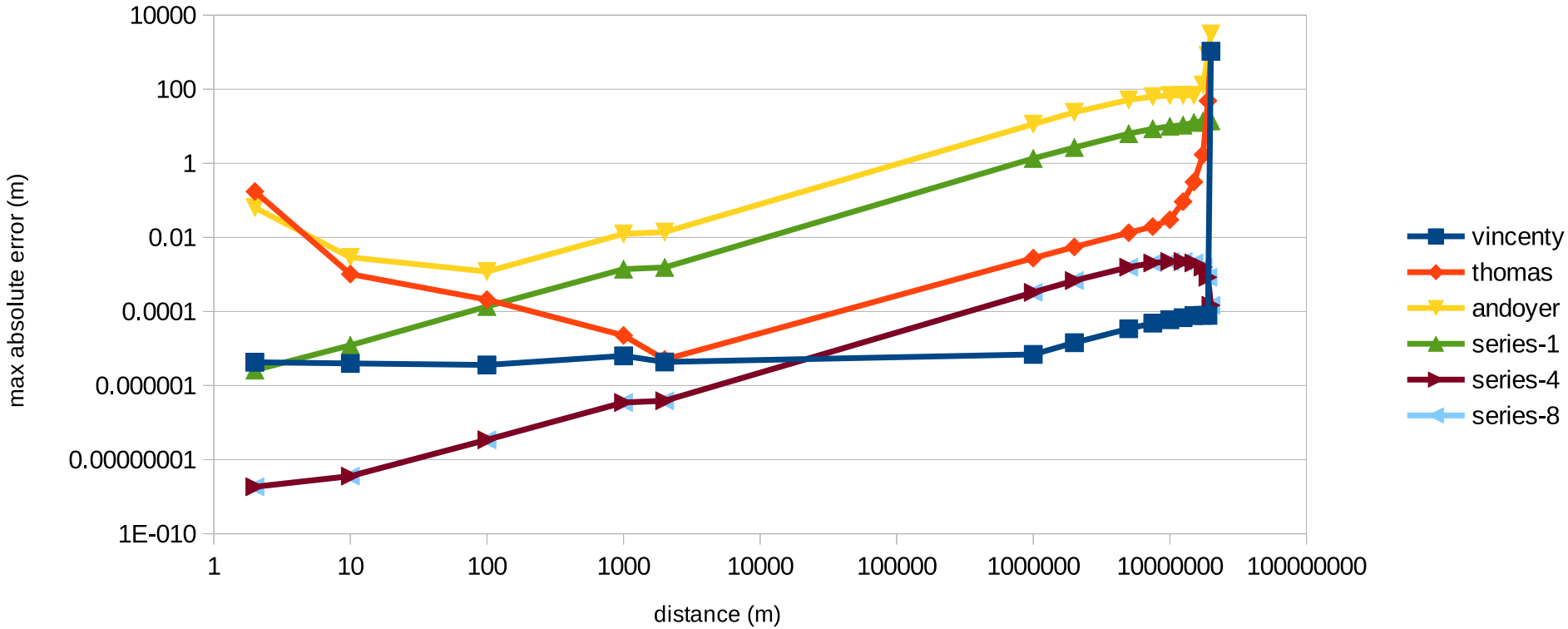
Performance





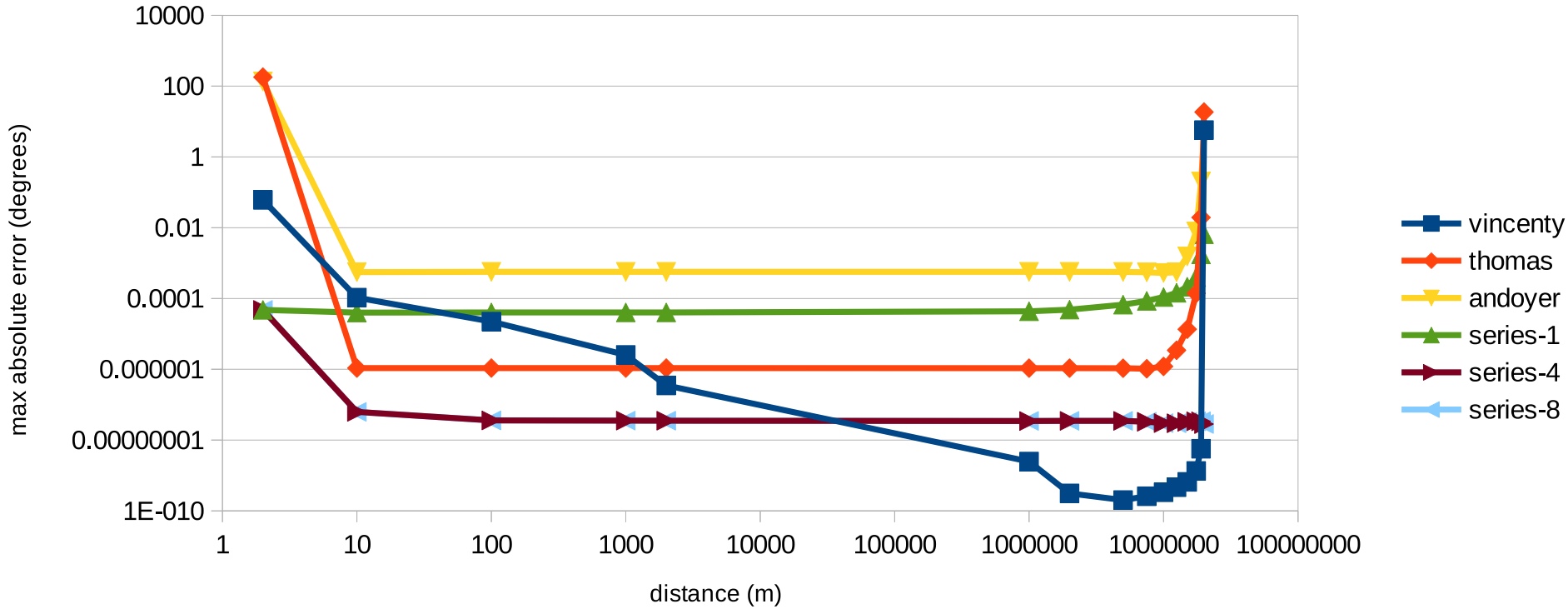
Accuracy

distance



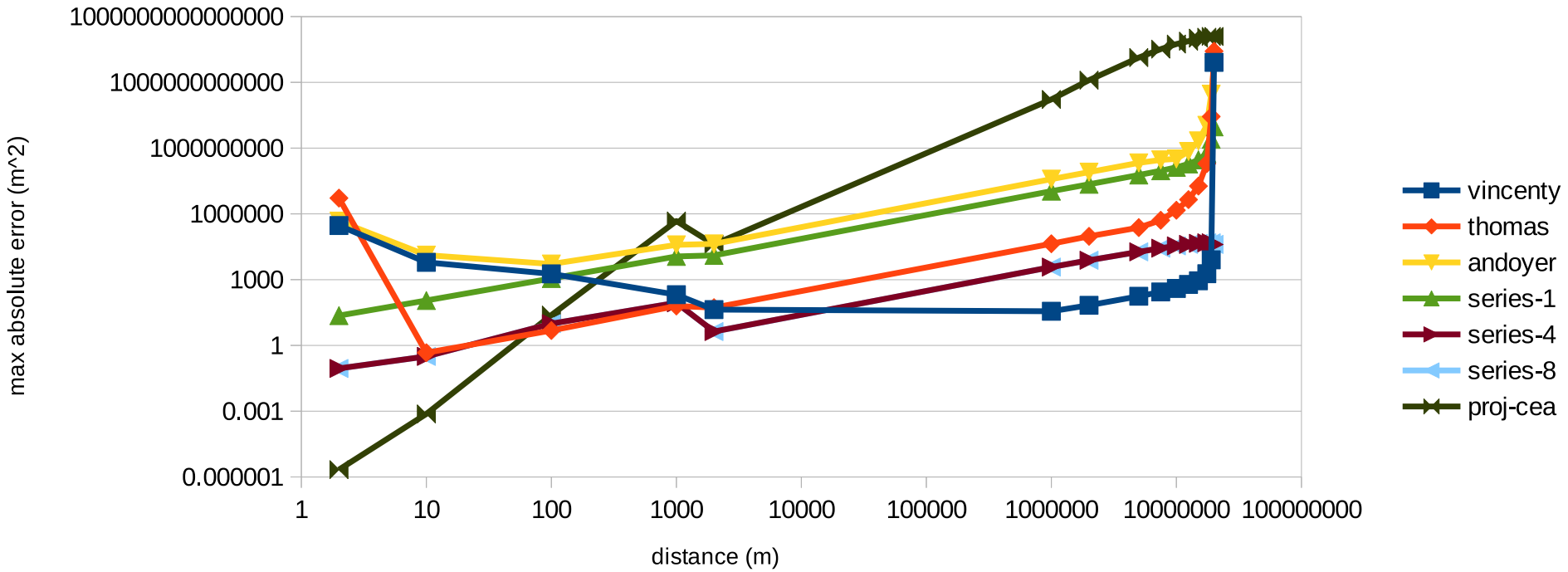
Accuracy

azimuth



Accuracy

area



Thank you!

Questions?