Spatial trajectories in Boost Geometry

Vissarion Fisikopoulos

FOSDEM 2020
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
Who is Boost.Geometry?

- Boost.Geometry is an open source project (as any other Boost library)
- Anybody can, and is welcome, to contribute
- Core development team:
  - Barend Gehrels
  - Bruno Lalande
  - Mateusz Loskot
  - Adam Wulkiewicz
  - Menelaos Karavelas
  - Vissarion Fysikopoulos
- Contributions from about a dozen of other developers
- See Boost.Geometry website for credits and GitHub repository's history
MySQL (since 5.7) relies on Boost geometry for GIS support (geographic support since 8)

no homegrown set of GIS functions for MySQL

both aim in OGC standard compliance

compatible licences

MySQL benefit from BG open source community (maintenance, bug fixing, gsoc)

BG is C++/header only → no problems with versions of a shared library on different platforms for MySQL
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
}
```
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
Hello strategies!

```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
                bg::strategy::distance::vincenty<>());
}
```
#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
  
  ```cpp
  boost::geometry::cs::cartesian
  ```

- Sphere
  - Widely used e.g. google.maps
  - ```cpp
  boost::geometry::cs::spherical<bg::degree>
  ```
  - ```cpp
  boost::geometry::cs::spherical<bg::radian>
  ```

- Ellipsoid of revolution
  - geographic GIS state-of-the-art
  - ```cpp
  boost::geometry::cs::geographic<bg::degree>
  ```
  - ```cpp
  boost::geometry::cs::geographic<bg::radian>
  ```

- Geoid
  - Special applications, geophysics etc
  - ```cpp
  boost::geometry::cs::geographic<bg::degree>
  ```
  - ```cpp
  boost::geometry::cs::geographic<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::geographic<bg::degree>
  boost::geometry::cs::geographic<bg::radian>

- Geoid *(Special applications, geophysics etc)*
Models of the earth and coordinate systems

- Flat
  
  ```cpp
  boost::geometry::cs::cartesian
  ```

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

- Ellipsoid of revolution (*geographic GIS state-of-the-art*)
  
  ```cpp
  boost::geometry::cs::geographic<bg::degree>
  boost::geometry::cs::geographic<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::geographic<bg::degree>
  boost::geometry::cs::geographic<bg::radian>

- **Geoid** *(Special applications, geophysics etc)*
Spatial trajectories

- Trajectories are sequences of time-stamped locations.
- Generated by GPS, smartphones, infrastructure, computer games, natural phenomena, etc.
- Here we study only the spatial and not the temporal information, i.e. trajectories are modelled as linestrings.

Trajectories of major hurricanes in Atlantic [Wang et al.'17]
Trajectories data-set

GeoLife GPS Trajectories dataset [download]

Two trajectories
Simple operations: size, length, distance

```cpp
using point = bg::model::point<
double, 2, bg::cs::geographic<bg::degree>>;
bg::model::linestring<point> ls1, ls2;

std::ifstream myfile1("Geolife_Trajectories_1.3/Data/000/ Trajectory/20090516091038.plt");
std::ifstream myfile2("Geolife_Trajectories_1.3/Data/010/ Trajectory/20081224011945.plt");
read_linestring(myfile1, ls1);
read_linestring(myfile2, ls2);

std::cout << boost::size(ls1) << std::endl;
std::cout << boost::size(ls2) << std::endl;

std::cout << bg::length(ls1) << std::endl;
std::cout << bg::length(ls2) << std::endl;

std::cout << bg::distance(ls1, ls2) << std::endl;
```

317
75
2196.14
718.456
369.504

Note: distances in meters, result by use of non default strategies neglectable
Closest points

```cpp
using point = bg::model::point
    <double, 2, bg::cs::geographic<bg::degree>>;

using linestring = bg::model::linestring<point>;;

linestring ls1, ls2;

std::ifstream myfile1 ("Geolife_Trajectories_1.3/Data/000/Trajectory/20090516091038.plt");
std::ifstream myfile2 ("Geolife_Trajectories_1.3/Data/010/Trajectory/20081224011945.plt");

read_linestring(myfile1, ls1);
read_linestring(myfile2, ls2);

bg::model::segment<point> sout;
bg::closest_points(ls1, ls2, sout);
```
Closest points
Simplification of trajectories

- simplification using Douglas-Peucker algorithm
- quadratic worst case complexity [Hershberger et.al’92]
- line_interpolate: interpolate points on linestring at a fixed distance
- sampling points on linestrings
  (https://github.com/boostorg/geometry/pull/618)
Simplify and line_interpolate

```cpp
using point = bg::model::point
    <double, 2, bg::cs::geographic<bg::degree>>;
using linestring = bg::model::linestring<point>;
linestring ls;

std::ifstream myfile2("Geolife_Trajectories_1.3/Data/010/
    Trajectory/20081224011945.plt");
read_linestring(myfile2, ls);

std::cout << "# points in ls = " << boost::size(ls2) << std::endl;
std::cout << "ls length (m) = " << bg::length(ls2) << std::endl;

linestring ls_simplified;
bg::simplify(ls2, ls_simplified, 20);
std::cout << "# points in simplified = " << boost::size(ls_simplified) << std::endl;

using multipoint_type = bg::model::multi_point<point>;
multipoint_type mp;
between_line_interpolate(ls2, 70, mp);
std::cout << "# points interpolated = " << boost::size(mp) << std::endl;
```

#points in ls = 75
ls length (m) = 718.456
#points in simplified = 6
#points interpolated = 9
Simplification and line_interpolate
Measuring similarity of trajectories

- Hausdorff distance

\[ H(f, g) = \max_{a \in f} \{ \min_{b \in g} \{ \text{dist}(a, b) \} \} \]

- Fréchet distance

\[ F(f, g) = \min \{ \|L\| \text{ such that } L \text{ is a coupling between } f \text{ and } g \} \]

coupling is a sequence of pairs from \( f, g \) that respect the order
Three trajectories

$l_1$: red, $l_2$: green, $l_3$: blue
using point = bg::model::point
    <double, 2, bg::cs::geographic<bg::degree>>;
bg::model::linestring<point> ls1, ls2, ls3;
std::ifstream myfile1 ("Geolife_Trajectories_1.3/Data/000/
    Trajectory/20090516091038.plt");
std::ifstream myfile2 ("Geolife_Trajectories_1.3/Data/010/
    Trajectory/20081224011945.plt");
std::ifstream myfile3 ("Geolife_Trajectories_1.3/Data/000/
    Trajectory/20081026134407.plt");
read_linestring(myfile1, ls1);
read_linestring(myfile2, ls2);
read_linestring(myfile3, ls3);
std::cout << bg::discrete_hausdorff_distance(ls1, ls2) << ","
    << bg::discrete_hausdorff_distance(ls2, ls3) << ","
    << bg::discrete_hausdorff_distance(ls1, ls3)
    << std::endl;
std::cout << bg::discrete_frechet_distance(ls1, ls2) << ","
    << bg::discrete_frechet_distance(ls2, ls3) << ","
    << bg::discrete_frechet_distance(ls1, ls3)
    << std::endl;

919.467, 7266.3, 8175.84
1260.76, 12601.7, 12837.9
Hausdorff & Fréchet distance
Comparing similarity of 160 pairs of trajectories

```cpp
namespace bf = boost::filesystem;
using point = bg::model::point
   <double, 2, bg::cs::geographic<bg::degree>>;
linestring = bg::model::linestring<point> ls1, ls2;

bf::path p{"Geolife_Trajectories_1.3/Data/000/Trajectory/"};
bf::directory_iterator it1{p};
double min_frechet = 10000000;

bf::directory_iterator it2{bf::path{"Geolife_Trajectories_1.3/Data/010/Trajectory/"}};
for (; it2 != bf::directory_iterator {}; it2++) {
    std::ifstream myfile1 ((*it1).path().string());
    std::ifstream myfile2 ((*it2).path().string());
    read_linestring(myfile1, ls1);
    read_linestring(myfile2, ls2);
    double frechet = bg::discrete_frechet_distance(ls1, ls2);
    min_frechet = frechet < min_frechet ? frechet : min_frechet;
}
```
cartesian: 9.97[sec]
spherical: 28.47[sec]
geographic: 52.30[sec]
Most similar trajectories

Same result for cartesian, spherical, geographic
Thank you! Questions?