Computational Geometry

Geometric algorithms and software for GIS

Vissarion Fisikopoulos

Department of Informatics & Telecommunications
National & Kapodistrian University of Athens

Spring 2025

Outline

GIS intro

Spatial computation in Boost.Geometry

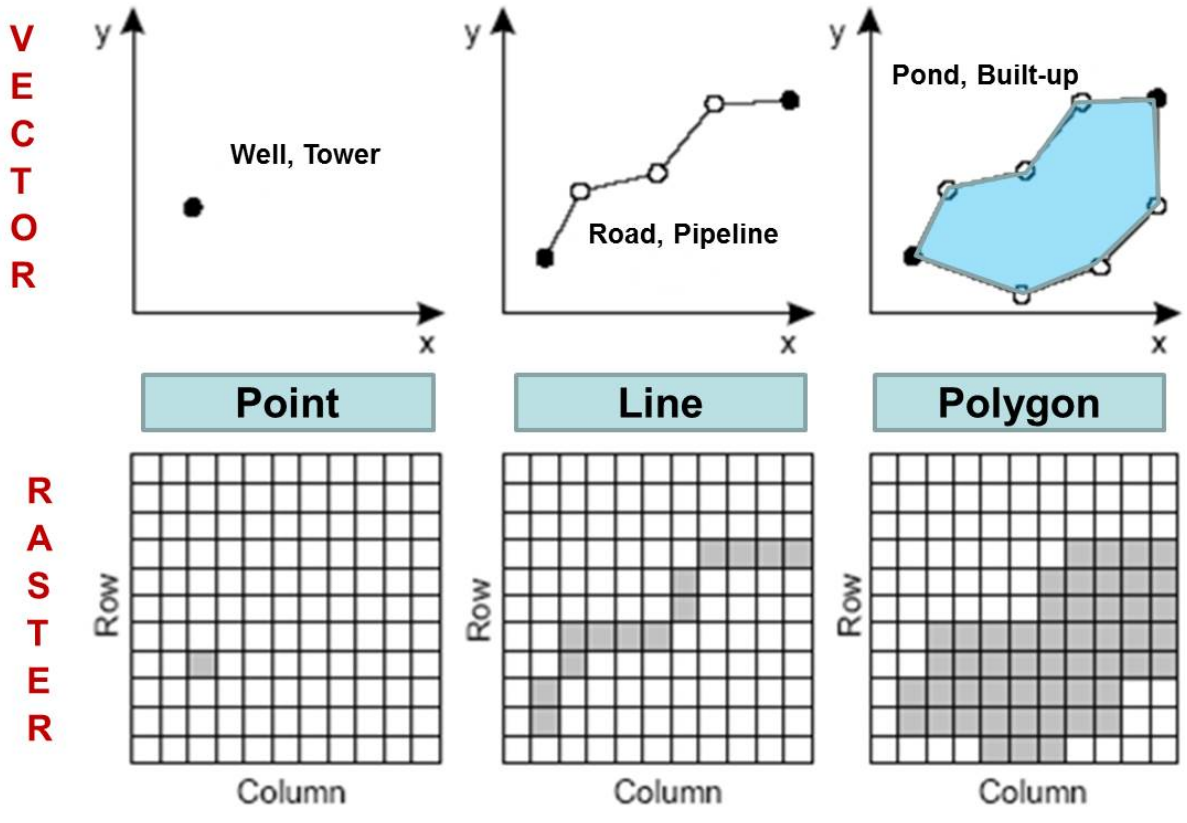
Geodesic algorithms in Boost.Geometry

What is GIS?

- Geographic Information System (GIS) = data + software + people
- Capture, store, manipulate, analyze, and visualize spatial data
- Examples: navigation apps, urban planning, disaster response

Vector and Raster Data

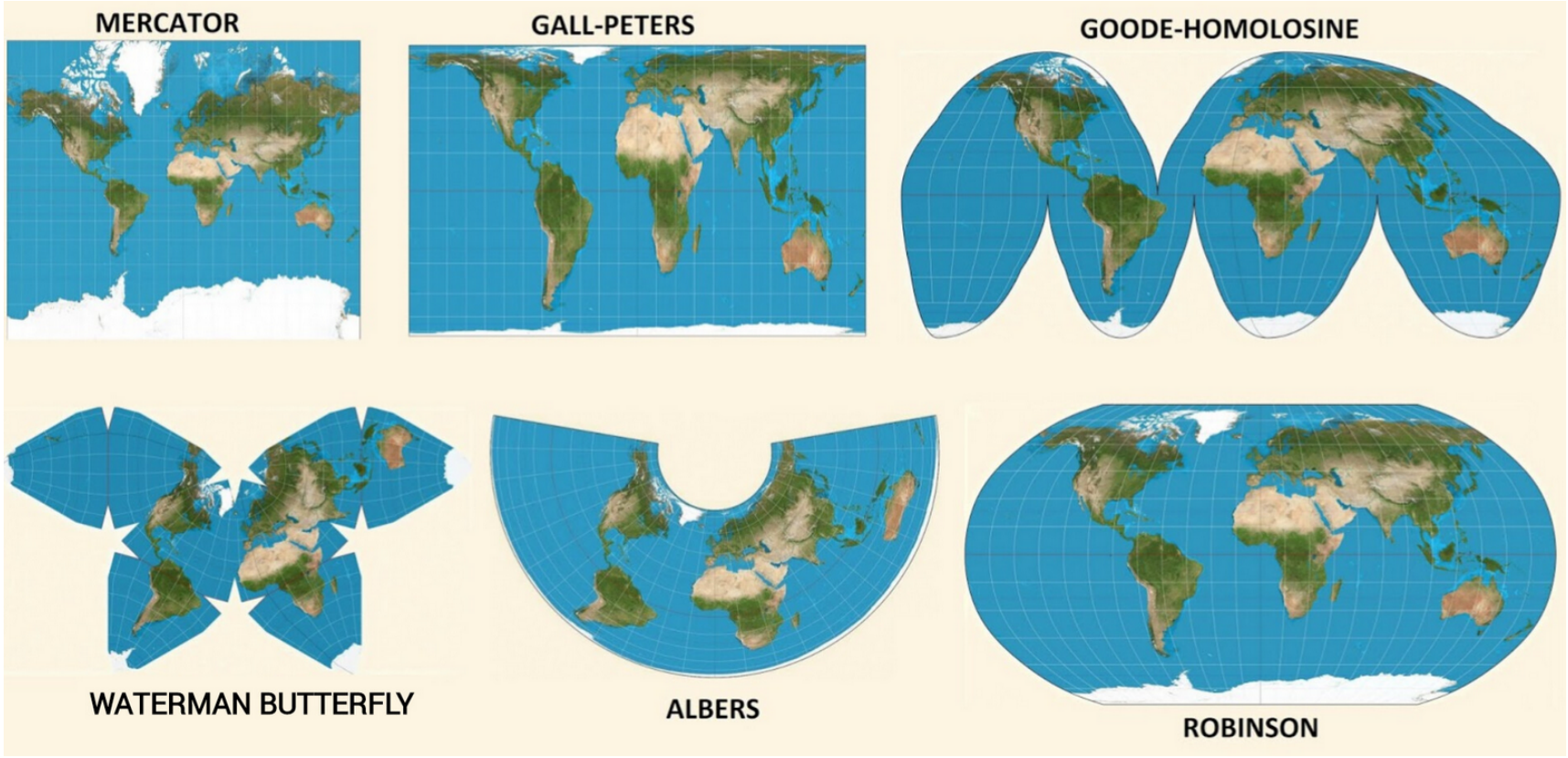
- Vector data: Points, Lines, Polygons
- Raster data: grid of pixels (e.g., satellite images, DEMs)
- Vector good for discrete features; raster for continuous phenomena



Vector and Raster Models

Map Projections

- Earth's surface is curved; maps are flat
- Projection = method to transform coordinates to 2D
- Common types: Mercator, Lambert, UTM
- Tradeoffs: area vs shape vs distance



Spatial Databases

Extend traditional db with the ability to store, index, and query spatial data such as geometries (points, lines, polygons) and rasters.

Core Features:

- Support for spatial data types: `POINT`, `LINESTRING`, `POLYGON`, `MULTIPOLYGON`
- Advanced spatial indexing: R-trees, QuadTrees
- Efficient spatial queries: containment, intersection, proximity, NN
- Built-in geometry functions: `ST_Intersects`, `ST_Contains`, `ST_Distance`, `ST_Union`, etc.

Popular Implementations:

- **PostGIS** (PostgreSQL): most feature-rich, supports topology, raster, 3D/4D, CRS transforms
- **MySQL Spatial**: since v5.7, native support, R-tree on InnoDB
- **SpatiaLite**: spatial extension for lightweight SQLite setups

Spatial Databases

Use Cases:

- Urban planning, routing, and logistics
- Web mapping and location-based apps
- Environmental and agricultural monitoring

Boost Geometry Agenda

- Boost.Geometry
- Hello World!
- Primitives
- Algorithms
- Spatial Index

Boost.Geometry

- Part of Boost C++ Libraries
- Header-only
- C++14 support
- Metaprogramming, Tag dispatching
- Primitives, Algorithms, Spatial Index
- OGC SFA conformant

Documentation and Resources

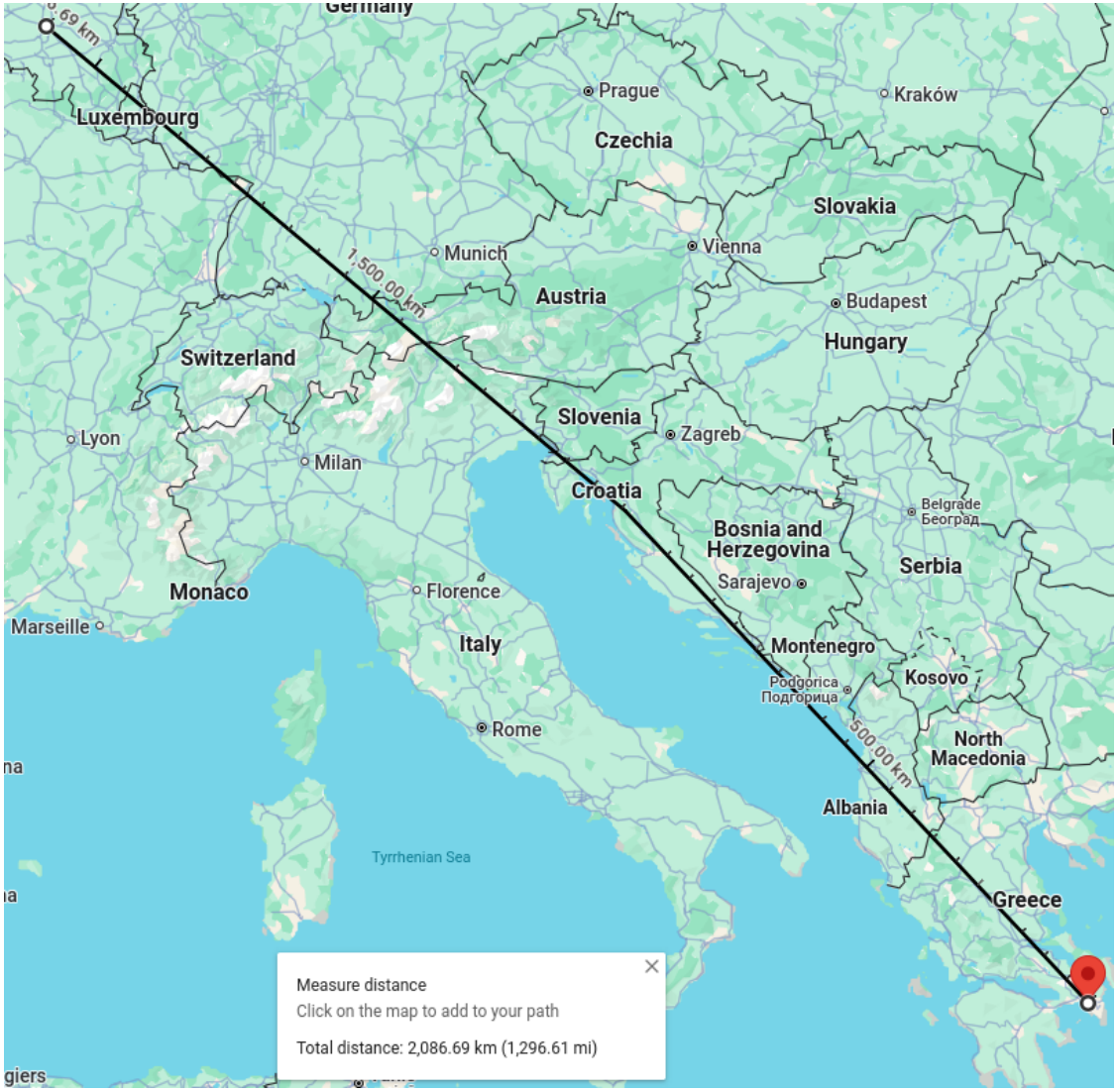
- `https://www.boost.org/libs/geometry`
- Mailing list: `lists.boost.org/geometry`
- GitHub: `https://github.com/boostorg/geometry`

Hello World!

```cpp
#include <boost/geometry.hpp>
#include <iostream>
namespace bg = boost::geometry;

int main() {
  using point = bg::model::point<double, 2,
        bg::cs::geographic<bg::degree>>;
  // Athens -> Brussels
  std::cout << bg::distance(point(23.727539, 37.983810),
                            point(4.350000, 50.833333));
}
```

Result

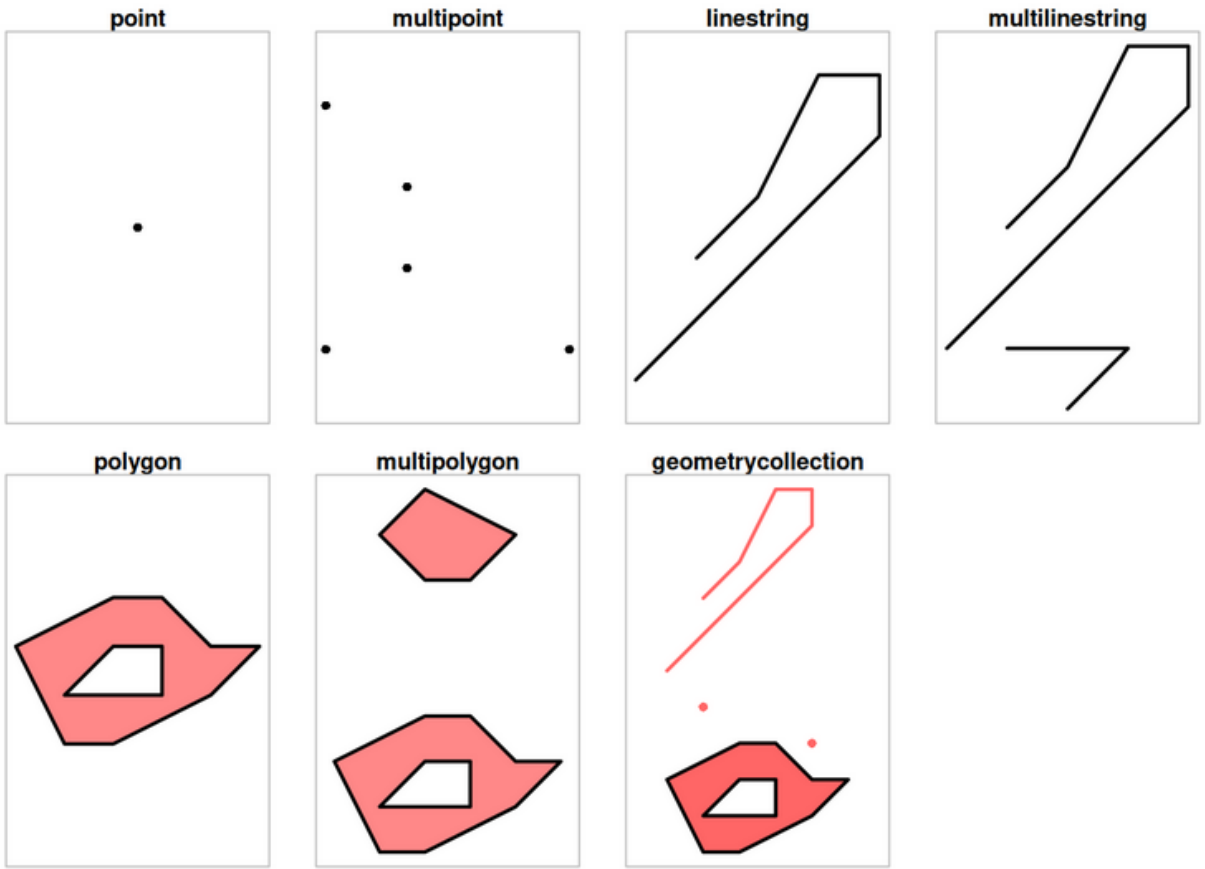
2.09071e+06 (in meters)



google maps

Primitives (Geometries)

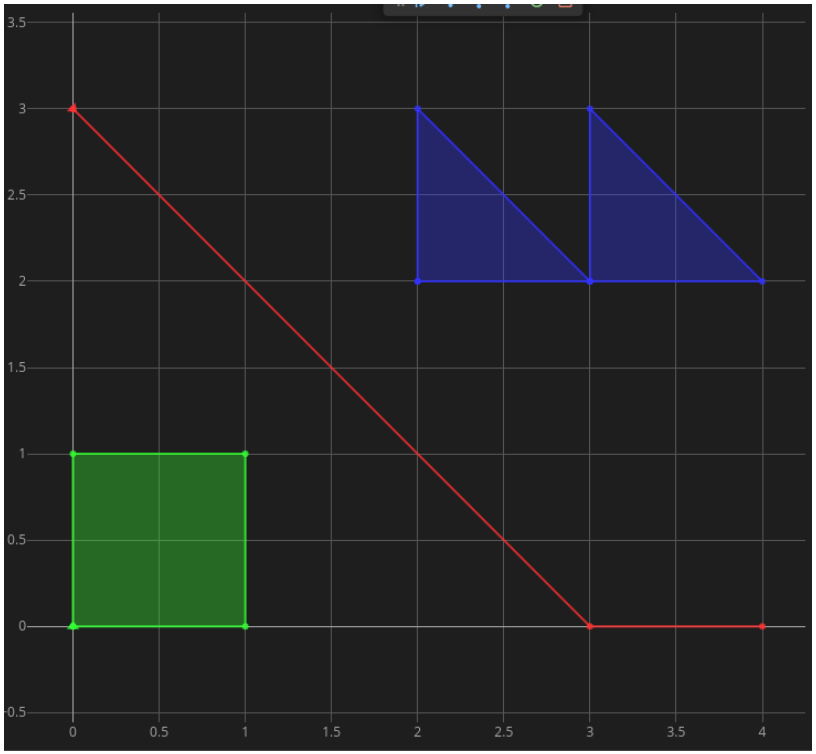
- Point, MultiPoint
- Segment, Linestring, MultiLinestring
- Ring, Polygon, MultiPolygon
- Box
- GeometryCollection



Using Geometries

```
using point = bg::model::point<double, 2, bg::cs::cartesian>;
using linestring = bg::model::linestring<point>;
using polygon = bg::model::polygon<point>;
using multi_polygon = bg::model::multi_polygon<polygon>;

linestring ls;
polygon poly;
multi_polygon mpoly;

bg::read_wkt("LINESTRING(0 3,3 0,4 0)", ls);
bg::read_wkt("POLYGON((0 0,0 1,1 1,1 0,0 0))", poly);
bg::read_wkt("MULTIPOLYGON(((2 2,2 3,3 2,2 2)),
                           ((3 2,3 3,4 2,3 2)))", mpoly);

std::cout << bg::distance(ls, poly) << '\n'
          << bg::distance(ls, mpoly);
```

Distance Output



0.707107 (to polygon)
0.707107 (to multipolygon)

Adapting Custom Types

```
#include <boost/geometry.hpp>
#include <boost/geometry/geometries/register/point.hpp>
#include <boost/geometry/geometries/register/linestring.hpp>
#include <iostream>
namespace bg = boost::geometry;

struct my_point { double x, y; };

BOOST_GEOMETRY_REGISTER_POINT_2D(my_point, double,
        bg::cs::cartesian, x, y)
BOOST_GEOMETRY_REGISTER_LINESTRING(std::vector<my_point>)

int main()
{
  my_point pt{0, 0};
  std::vector<my_point> ls{{0, 1}, {1, 0}, {1, 1}, {0.5, 1}};
  std::cout << bg::distance(pt, ls);
}
```

Result: 0.707107

Algorithms Overview

Measure

- distance, area, length, perimeter
- discrete_frechet_distance, discrete_hausdorff_distance

Geometric

- centroid, convex_hull, envelope, buffer, simplify

Relational and set operations

- crosses, disjoint, equals, intersects, overlaps, relate, within
- difference, intersection, union, sym_difference

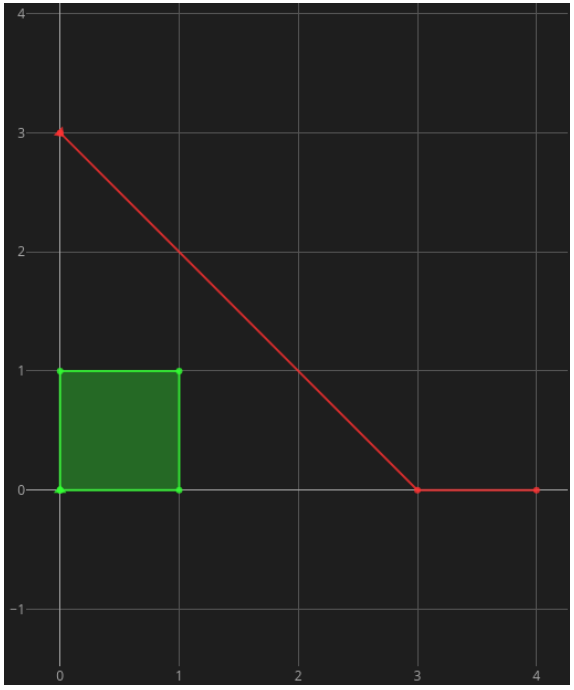
More . . .

`https://www.boost.org/doc/libs/latest/libs/geometry/doc/html/geometry/reference/algorithms.html`

Area, Length, Perimeter

```
linestring ls;
polygon poly;

bg::read_wkt("LINESTRING(0 3, 3 0, 4 0)", ls);
bg::read_wkt("POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))", poly);

std::cout << "length    " << bg::length(ls) << '\n'
          << "area      " << bg::area(poly) << '\n'
          << "perimeter " << bg::perimeter(poly);
```

Algorithm Output



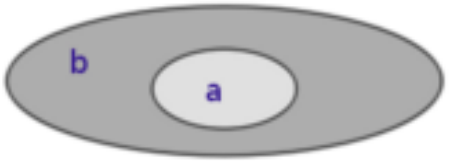
length: `5.242641`

area: `1.000000`

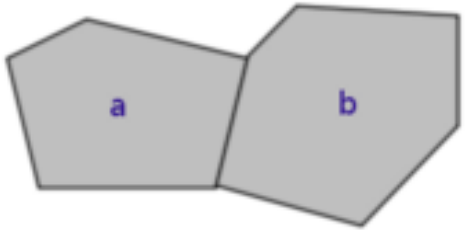
perimeter: `4.000000`

Geospatial topology

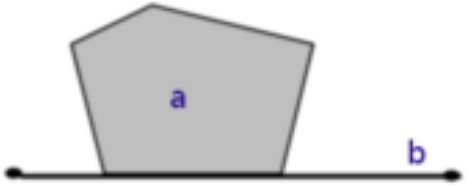
Within(a,b)



Touches(a,b)

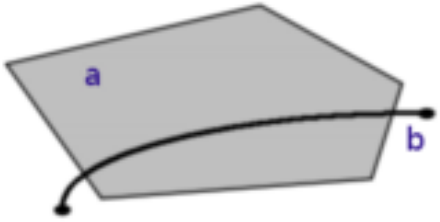


Touches(a,b)

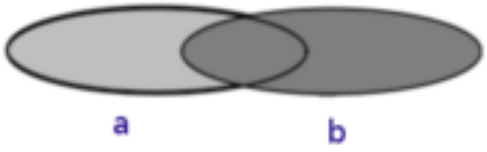


Crosses(a,b)

Crosses(a,b)



Overlaps(a,b)



DE-9IM intersection matrix

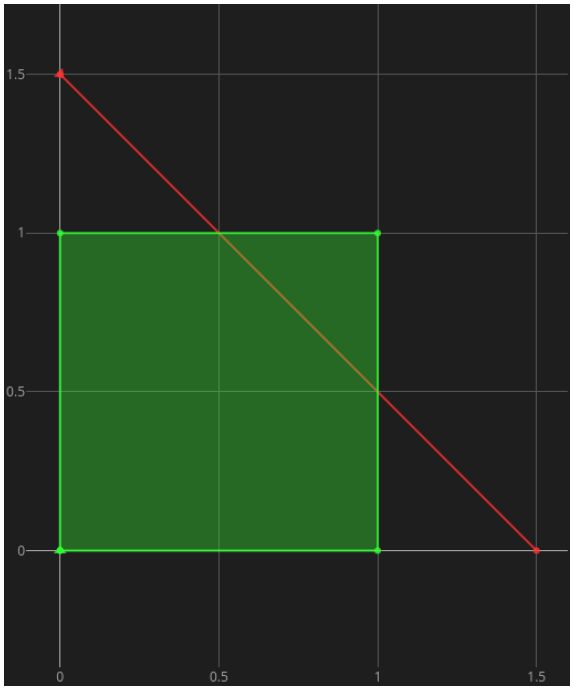
	Interior	Boundary	Exterior
Interior	 $\dim[I(a) \cap I(b)] = 2$	 $\dim[I(a) \cap B(b)] = 1$	 $\dim[I(a) \cap E(b)] = 2$
Boundary	 $\dim[B(a) \cap I(b)] = 1$	 $\dim[B(a) \cap B(b)] = 0$	 $\dim[B(a) \cap E(b)] = 1$
Exterior	 $\dim[E(a) \cap I(b)] = 2$	 $\dim[E(a) \cap B(b)] = 1$	 $\dim[E(a) \cap E(b)] = 2$

within = T*F**F***

Intersects, Within

```
linestring ls;
polygon poly;

bg::read_wkt("LINESTRING(0 1.5, 1.5 0)", ls);
bg::read_wkt("POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))", poly);

std::cout << "intersects " << bg::intersects(ls, poly) << '\n'
          << "within     " << bg::within(ls, poly);
```

Relation Output



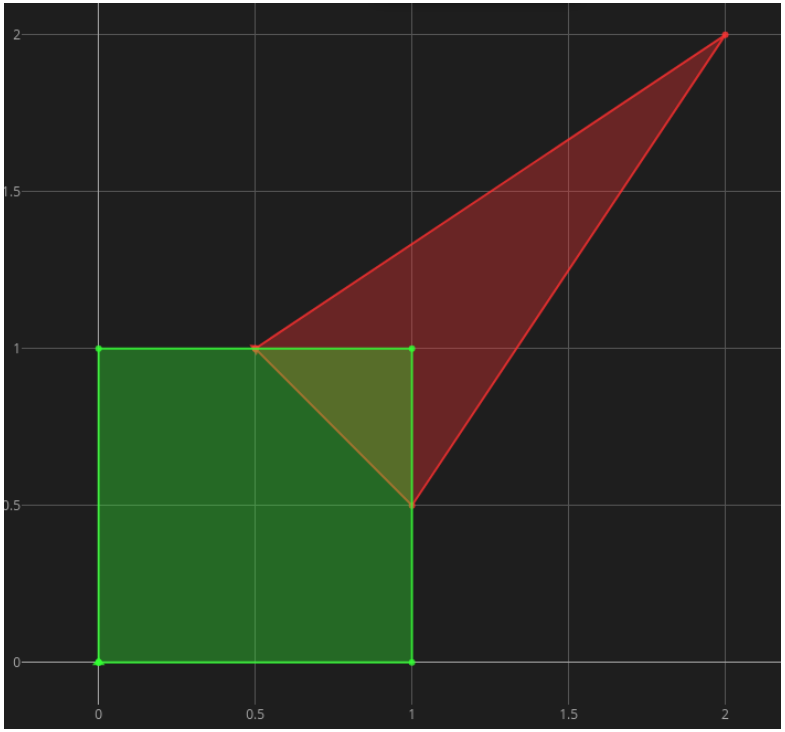
intersects: 1

within: 0

Intersects, Relation, Within

```
polygon poly1;
polygon poly2;


bg::read_wkt("POLYGON((0.5 1, 2 2, 1 0.5, 0.5 1))", poly1);
bg::read_wkt("POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))", poly2);

std::cout << "intersects " << bg::intersects(poly1, poly2) << '\n'
          << "within     " << bg::within(poly1, poly2) << '\n'
          << "relation   " << bg::relation(poly1, poly2).str();
```

Relation Output

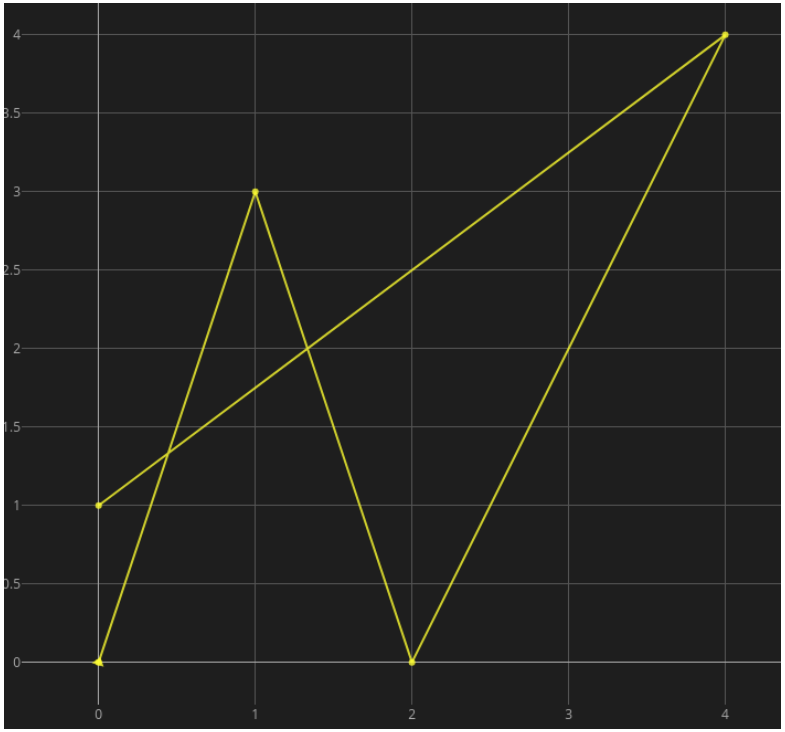


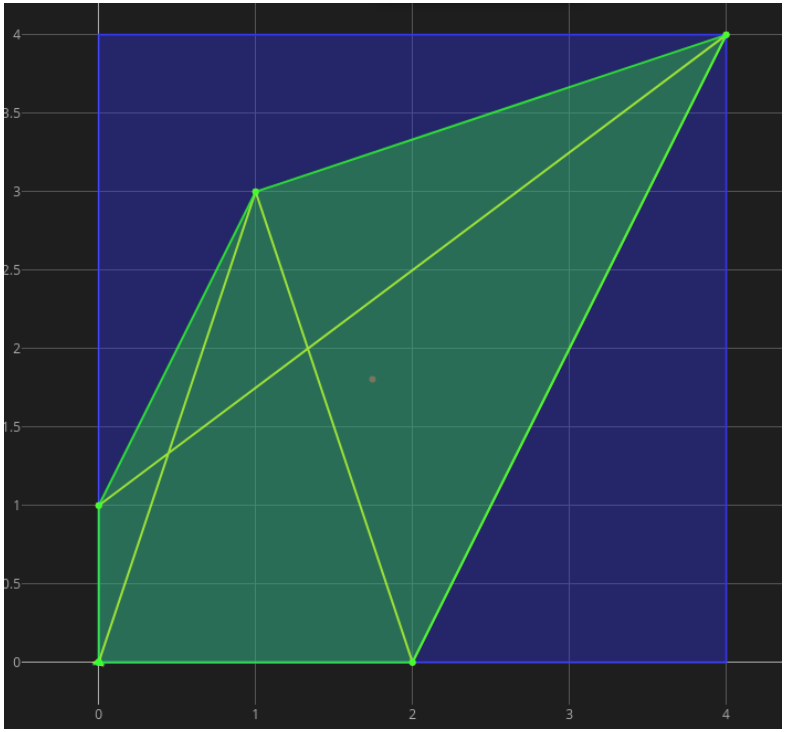

```
intersects 1          (T********)or(*T*******)or...
within     0          (T*F**F***)
relation   212101212
```

Centroid, Envelope, Convex Hull

```
point p;
linestring ls;
box b;
polygon poly;

bg::read_wkt("LINESTRING(0 0, 1 3, 2 0, 4 4, 0 1)", ls);
bg::centroid(ls, p);
bg::envelope(ls, b);
bg::convex_hull(ls, poly);

std::cout << "centroid " << bg::wkt(p) << '\n'
          << "envelope " << bg::wkt(b) << '\n'
          << "hull     " << bg::wkt(poly);
```

Centroid/convex hull output



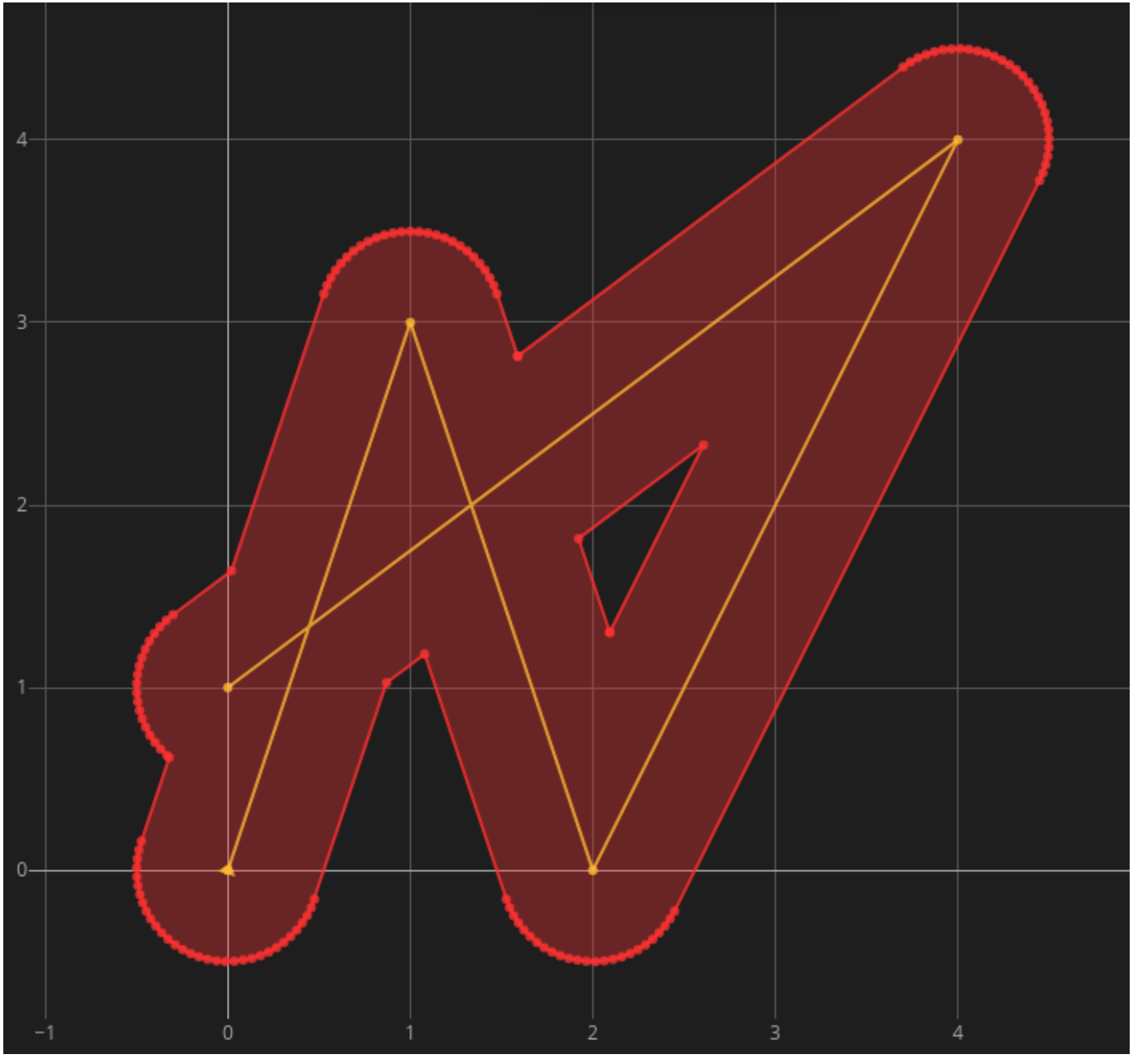



```
centroid POINT(1.7451 1.80392)
envelope POLYGON((0 0,0 4,4 4,4 0,0 0))
hull     POLYGON((0 0,0 1,1 3,4 4,2 0,0 0))
```

Geometric Buffer

```
linestring ls;
multi_polygon mpoly;

bg::read_wkt("LINESTRING(0 0, 1 3, 2 0, 4 4, 0 1)", ls);
namespace bgsb = bg::strategy::buffer;

bg::buffer(ls, mpoly,
           bgsb::distance_symmetric<double>(0.5),
           bgsb::side_straight(),
           bgsb::join_round(64),
           bgsb::end_round(64),
           bgsb::point_circle(64));
```

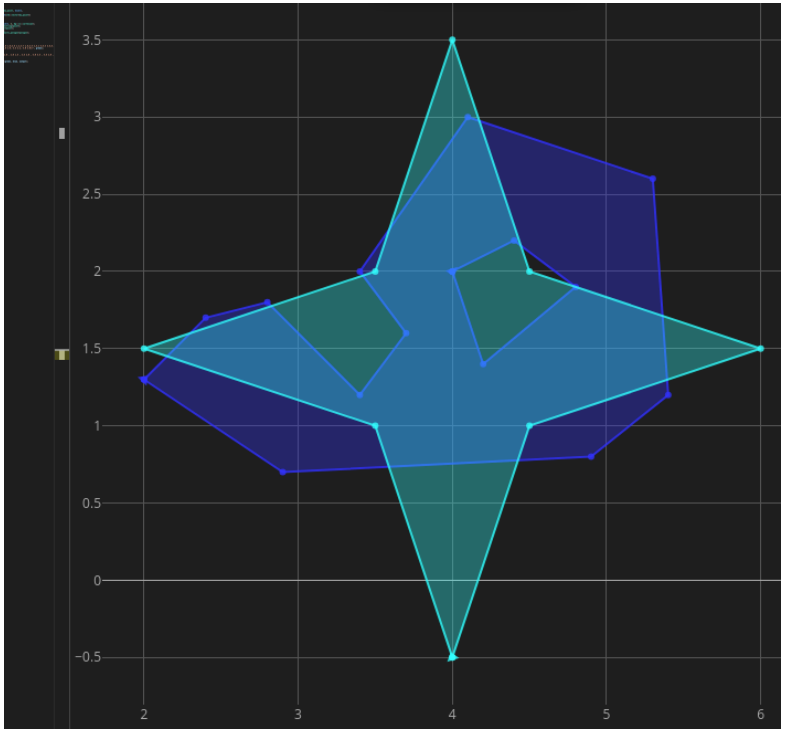
Buffer result

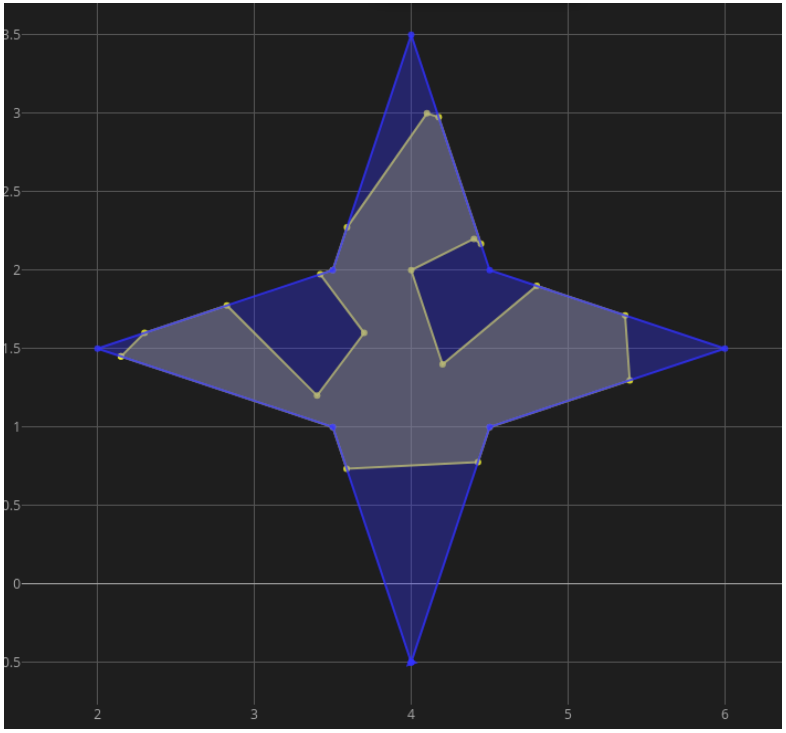


Intersection and Symmetric Difference

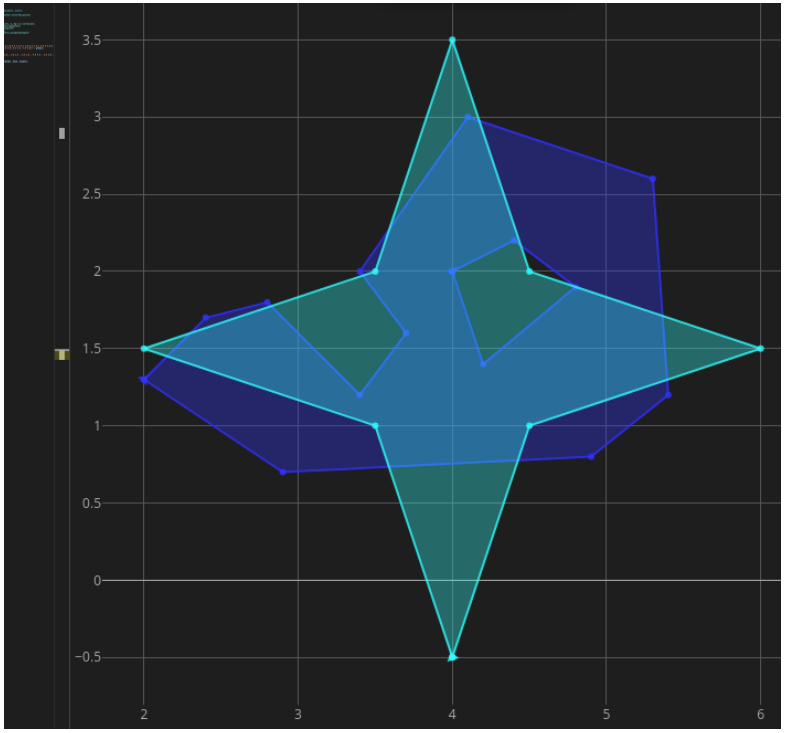
```
polygon green, blue;

    boost::geometry::read_wkt(
        "POLYGON((2 1.3,2.4 1.7,2.8 1.8,3.4 1.2,3.7 1.6,3.4 2,4.1
            "(4.0 2.0, 4.2 1.4, 4.8 1.9, 4.4 2.2, 4.0 2.0))", gree

    boost::geometry::read_wkt(
        "POLYGON((4.0 -0.5 , 3.5 1.0 , 2.0 1.5 , 3.5 2.0 , 4.0 3.5

    multi_polygon mpoly1, mpoly2;
    boost::geometry::intersection(green, blue, mpoly1);
    boost::geometry::sym_(green, blue, mpoly2);
```

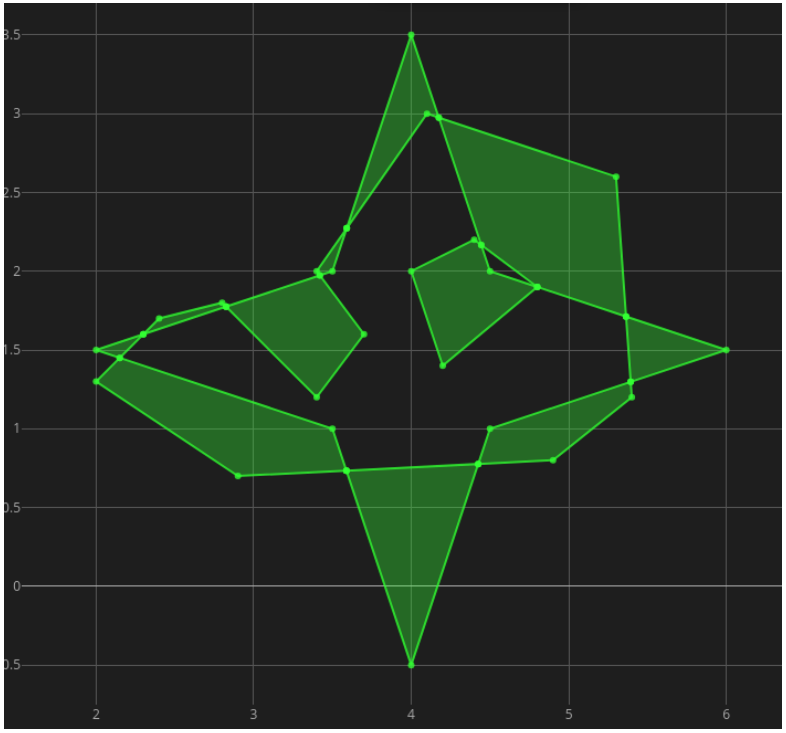
Intersection





Symmetric Difference (XOR)





Spatial Index Basics

- R-tree
- Split heuristics: linear, quadratic or r*-tree
- bulk-loading
- user-defined value type
- various spatial and knn query

Spatial Index Basics

```
#include <boost/geometry/index/rtree.hpp>

using point = bg::model::point<double, 2, bg::cs::cartesian>;
using polygon = bg::model::polygon<point>;
using box = bg::model::box<point>;
using value = std::pair<box, std::size_t>;
using rtree = bg::index::rtree<value, bg::index::rstar<16>>;
```

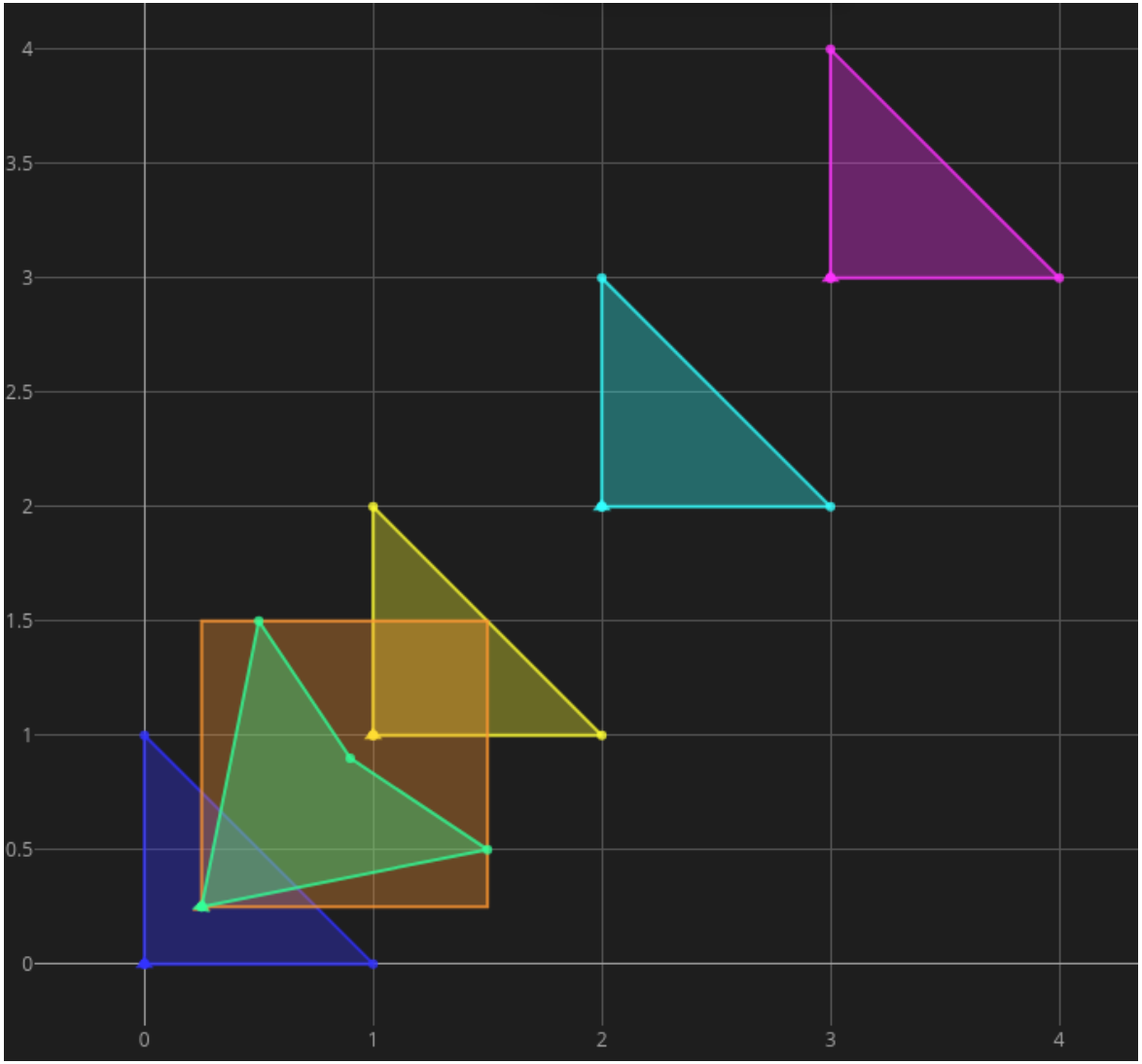
Spatial Index Example

```
std::vector<polygon> polys(4);
bg::read_wkt("POLYGON((0 0, 0 1, 1 0, 0 0))", polys[0]);
bg::read_wkt("POLYGON((1 1, 1 2, 2 1, 1 1))", polys[1]);
bg::read_wkt("POLYGON((2 2, 2 3, 3 2, 2 2))", polys[2]);
bg::read_wkt("POLYGON((3 3, 3 4, 4 3, 3 3))", polys[3]);

rtree rt;
for (std::size_t i = 0; i < polys.size(); ++i) {
  box b = bg::return_envelope<box>(polys[i]);
  rt.insert(std::make_pair(b, i));
}
```

Spatial Index Query

```
polygon qpoly;
bg::read_wkt("POLYGON((0.25 0.25,0.5 1.5,0.9 0.9,
  1.5 0.5,0.25 0.25))", qpoly);
box qbox = bg::return_buffer<box>(bg::return_envelope<box>(qpoly),
  0.0001);

std::vector<value> result;
rt.query(bg::index::intersects(qbox), std::back_inserter(result));

for (const auto& v : result) {
  std::cout << bg::wkt(polys[v.second])
            << (bg::intersects(polys[v.second], qpoly)
              ? " intersects" : " not intersects") << std::endl;
}
```

Result




```
POLYGON((0 0,0 1,1 0,0 0)) intersects
POLYGON((1 1,1 2,2 1,1 1)) not intersects
```

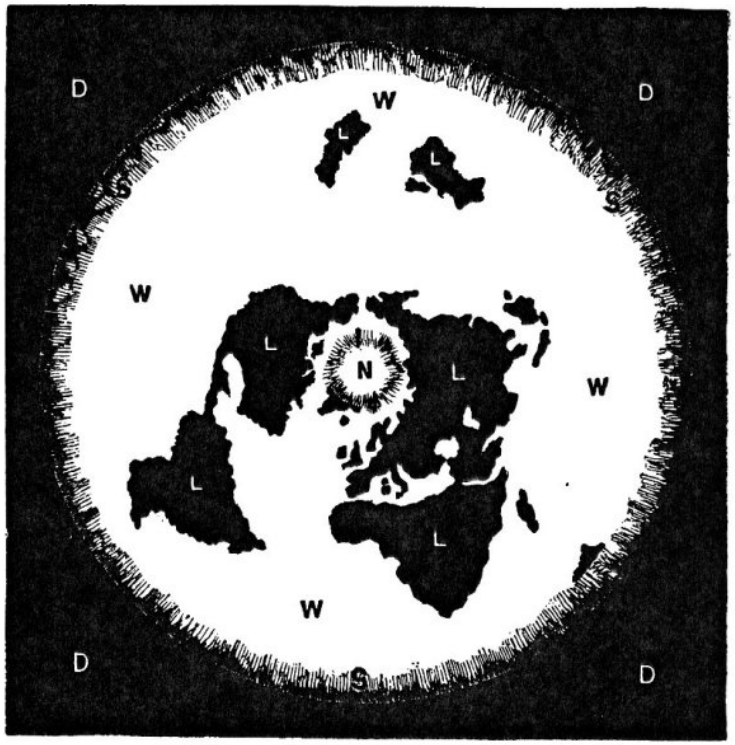
Outline

GIS intro

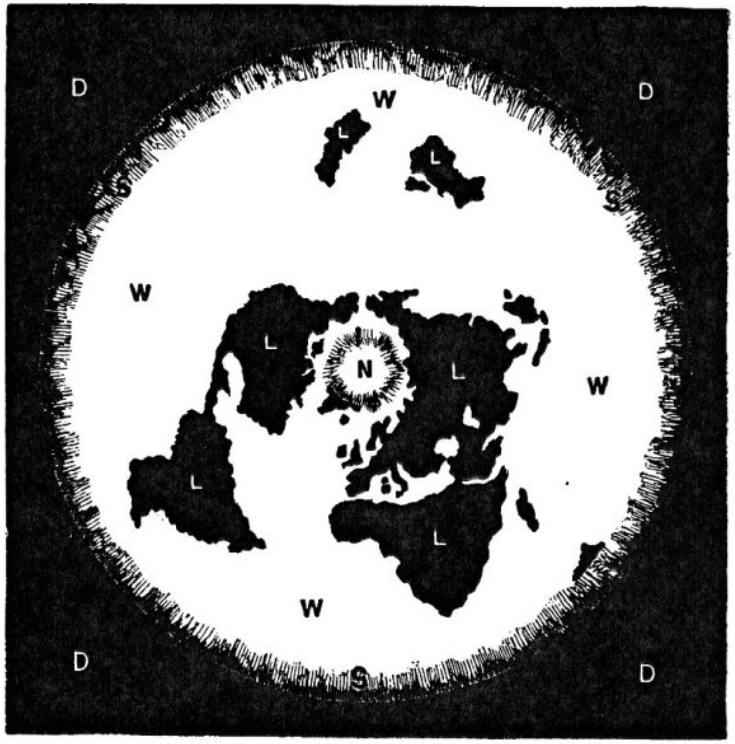
Spatial computation in Boost.Geometry

Geodesic algorithms in Boost.Geometry

Flat Earth



Flat Earth


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


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

Models of the earth and coordinate systems

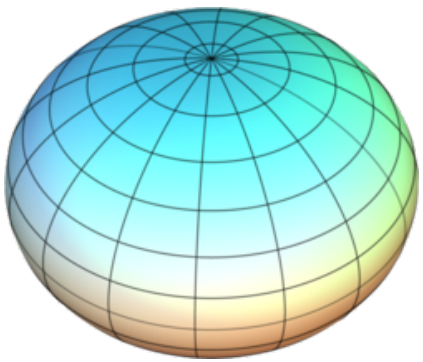
- Flat
  Accurate only locally. Euclidean geometry. Very fast and simple algorithms.

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.*

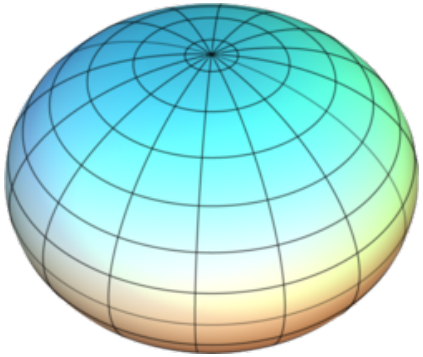
Models of the earth and coordinate systems

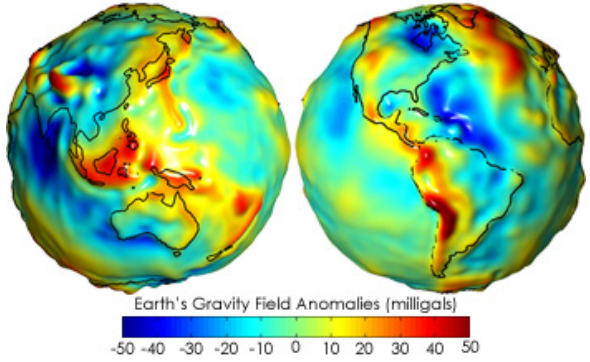
- Flat
  Accurate only locally. Euclidean geometry. Very fast and simple algorithms.
- Sphere
  Widely used (e.g. `google.maps`*). Not very accurate. Fast algorithms.*
- Ellipsoid of revolution (or shperoid or ellipsoid)
  State-of-the-art in GIS. More involved algorithms.



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





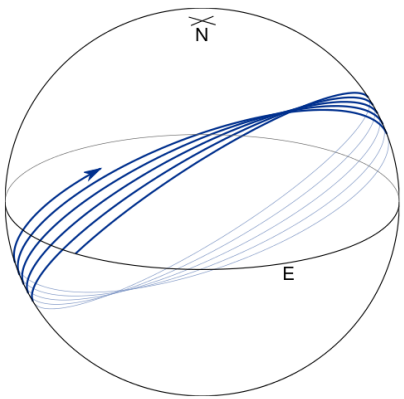
Coordinate systems in Boost.Geometry

```
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

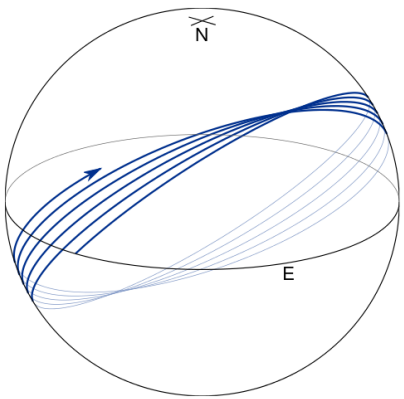
- flat: geodesic = straight line
- sphere: geodesic = great circle
- ellipsoid: geodesic = not closed curve (*except meridians and equator*)

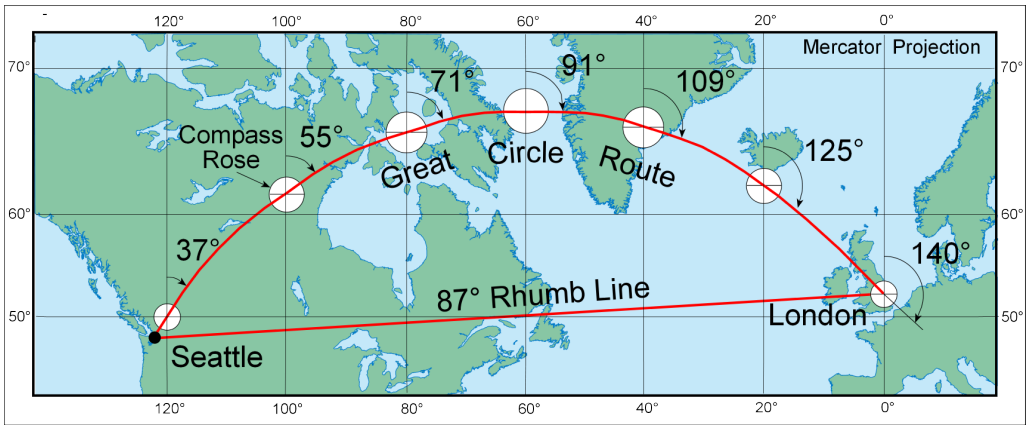


Geodesics

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)

Karney - Geodesics on an ellipsoid of revolution, 2011

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

λ, ϕ: longitude, latitude

Distance between points

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

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

Geodesic computation in Boost.Geomerty

- Different formulas are selected w.r.t. the coordinate system
- 3 different algorithms for distance on ellipsoid implemented as strategies (andoyer, thomas, vincenty)
  $\rightarrow$ 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 (revisited)

How far away from home ?

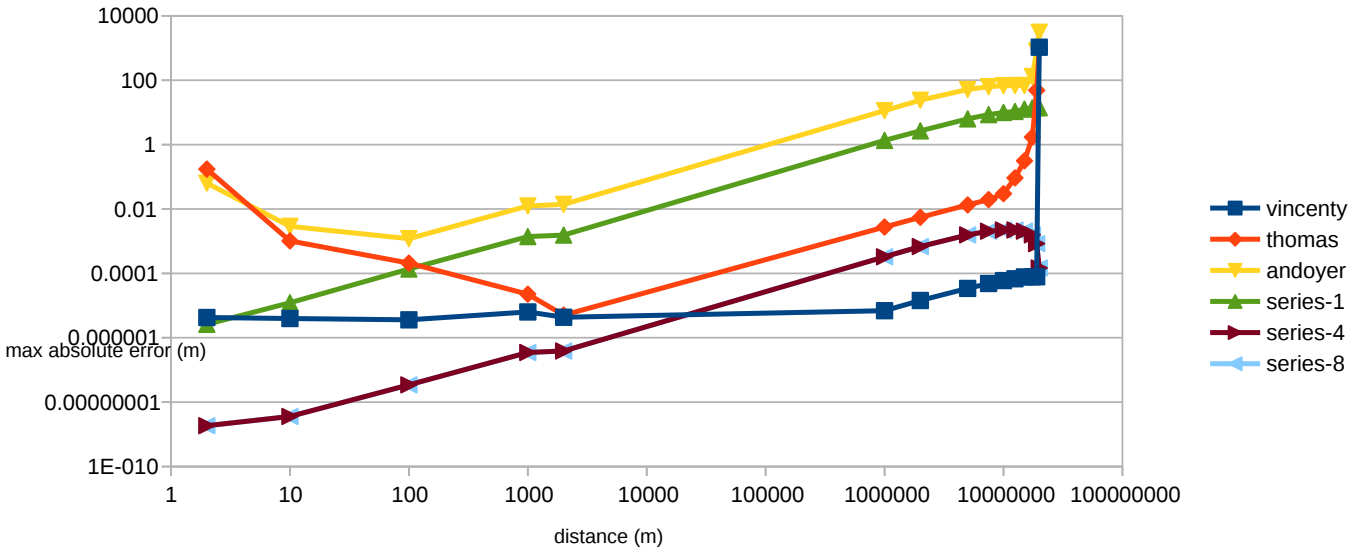
```
namespace bg = boost::geometry;
typedef bg::model::point<double, 2,
                         bg::cs::geographic<bg::degree> > point;

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)

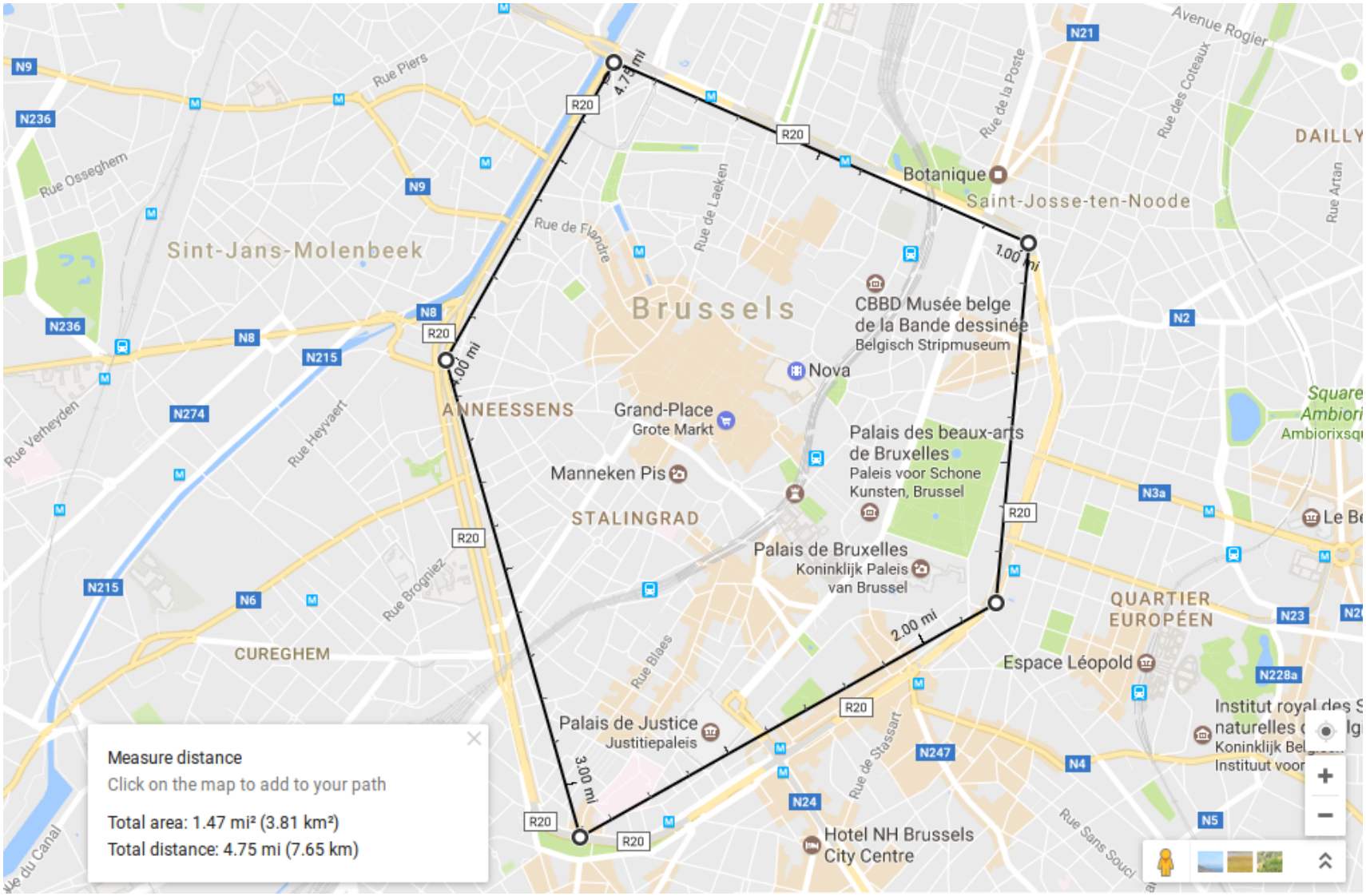
Distance experiment



Data: `https://zenodo.org/records/12510796`

Area example

Brussels center polygon



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)

GeographicLib

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