Boost.Geometry: A C++ library for geometric computations

Vissarion Fisikopoulos

FOSDEM 2026

Outline

Introduction

Boost Geometry design

Using Boost Geometry

Spatial computations

Boost.Geometry

- Open source
- Part of Boost C++ Libraries
- Header-only
- Stable interface
- C++14 support
- Static polymorphism, Metaprogramming, Tag dispatching
- Primitives, Algorithms, Spatial Index
- Standards conformant (OGC Simple Feature Access)

boost GEOMETRY

Open Geospatial Consortium

Documentation and Resources

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

Main contributors:

- Barend Gehrels
- Bruno Lalande
- Mateusz Loskot
- Adam Wulkiewicz
- Menelaos Karavelas
- Vissarion Fisikopoulos
- Tinko Bartels
- and several other contributors

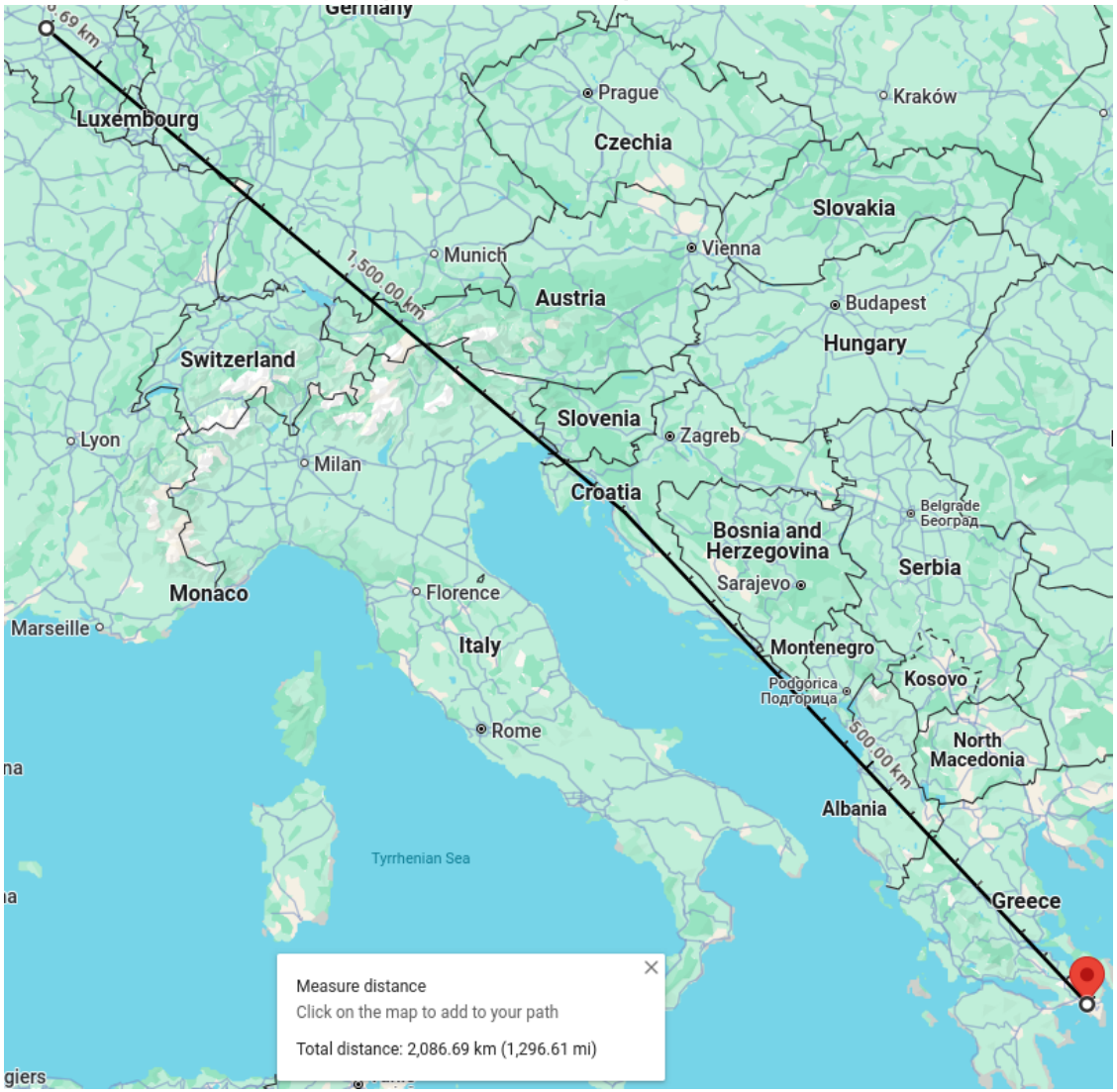
Hello World!

```
#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

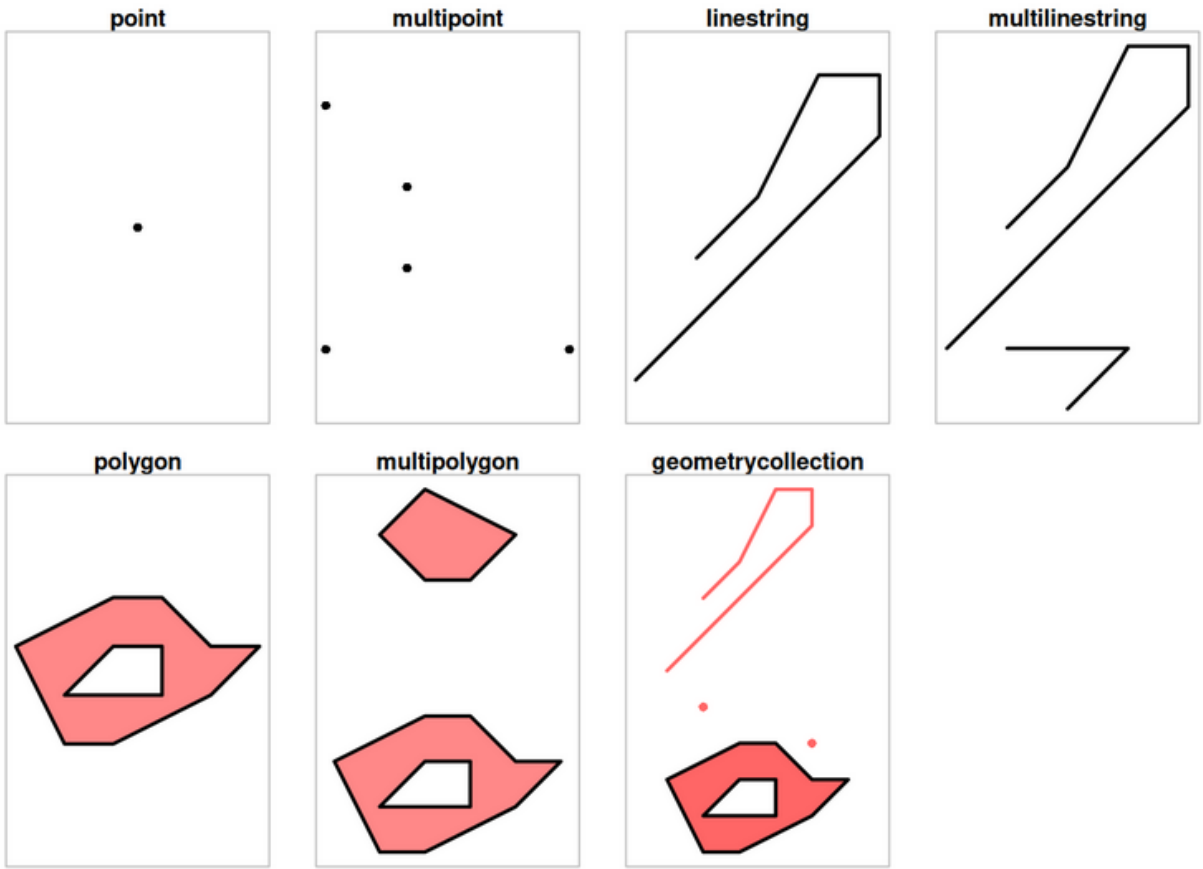
2.09071e+06 (in meters)



google maps

Primitives (Geometries)

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



Algorithms

Algorithms

Geometry Constructors
make
make_inverse
make_zero

Predicates
crosses
covered_by
disjoint
equals
intersects
is_empty
is_simple
is_valid
overlaps
touches
within

Append
append

Area
area

Assign
assign
assign_inverse
assign_zero
assign_points
assign_values (2 3 4 coordinate values)

Azimuth
azimuth

Buffer
buffer

Centroid
centroid

Clear
clear

Closest Points
closest_points

Convert
convert

Convex Hull
convex_hull

Correct
correct

Densify
densify

Distance
distance

Difference
difference
sym_difference

Envelope
envelope

Expand
expand

For Each
for each (point, segment)

Intersection
intersection

Length
length

Line Interpolate
line_interpolate

Num_ (counting)
num_interior_rings
num_geometries
num_points
num_segments

Perimeter
perimeter

Relate
relate
relation

Reverse
reverse

Similarity
discrete_frechet_d
discrete_hausdorff

Simplify
simplify

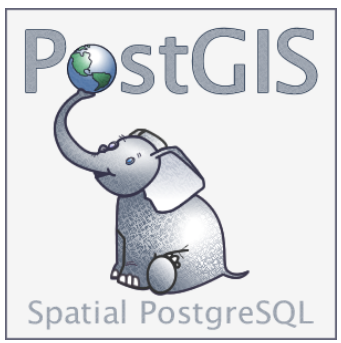
Transform
transform

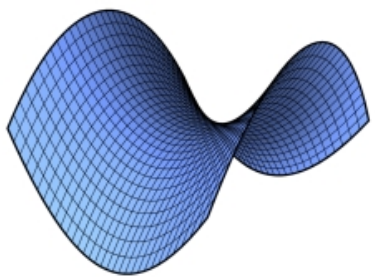
Union
union

Unique
unique

Applications

- GIS, spatial databases
- Robotics and autonomous systems
- Computer graphics and gaming
- Computer vision and image analysis





Outline

Introduction

Boost Geometry design

Using Boost Geometry

Spatial computations

Three-layer design

- **Algorithms layer**: Namespace `boost::geometry` constains user-facing API (e.g. `boost::geometry::distance`).
- **Dispatch layer**: Namespace `boost::geometry::dispatch` contains compile-time selection using traits, tags, concepts.
- **Implementation layer**: Namespace `boost::geometry::detail` contains implementation details.

Algorithms

- Free functions (e.g., `distance`, `area`, `buffer`).
- Input: geometries, strategies (optional)
- Algorithms are independent of geometry representation
- Work across cartesian, spherical, geographic coordinate systems.
- Internally resolved via dispatching and strategy selection.

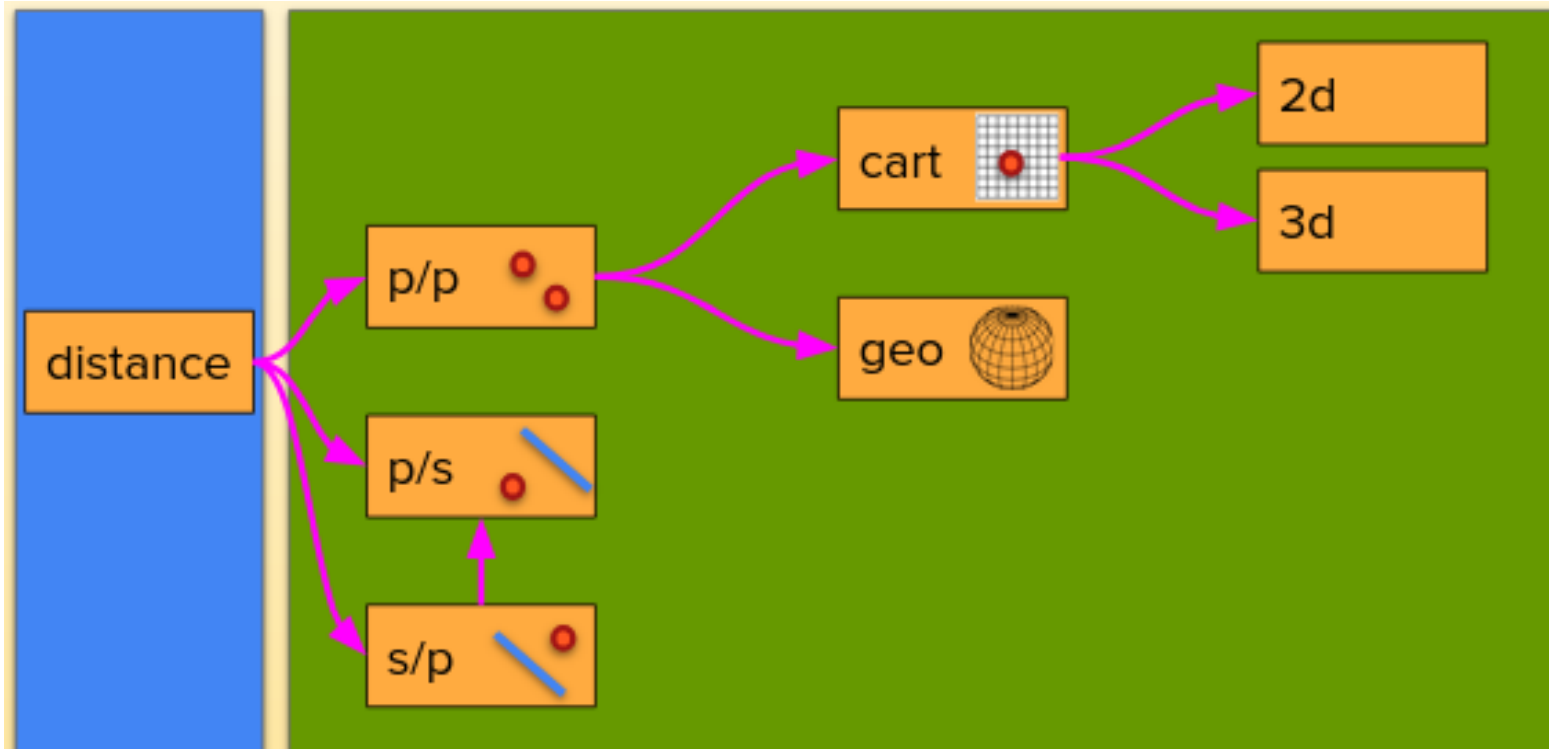
Strategies

- Encapsulate mathematical models and formulas.
- Examples: Vincenty, Andoyer, Thomas for geographic distance.
- Strategies chosen explicitly by the user or defaulted by the library.
- Provide accuracy/performance flexibility without changing algorithm signatures.

Tag dispatching

- Tags for: geometries, strategies, coordinate systems
- Tag types available through meta-functions
- Dispatching is done w.r.t.
  - Geometry type (point, linestring, polygon, etc.)
  - Coordinate system (cartesian, geographic, etc.)
  - Dimension (2d, 3d, etc.)
  - Strategy (e.g. different distance strategies for geographic cs)
- Provides type-safe, zero-overhead static polymorphism.

Distance example flow



Outline

Introduction

Boost Geometry design

Using Boost Geometry

Spatial computations

Using Geometries

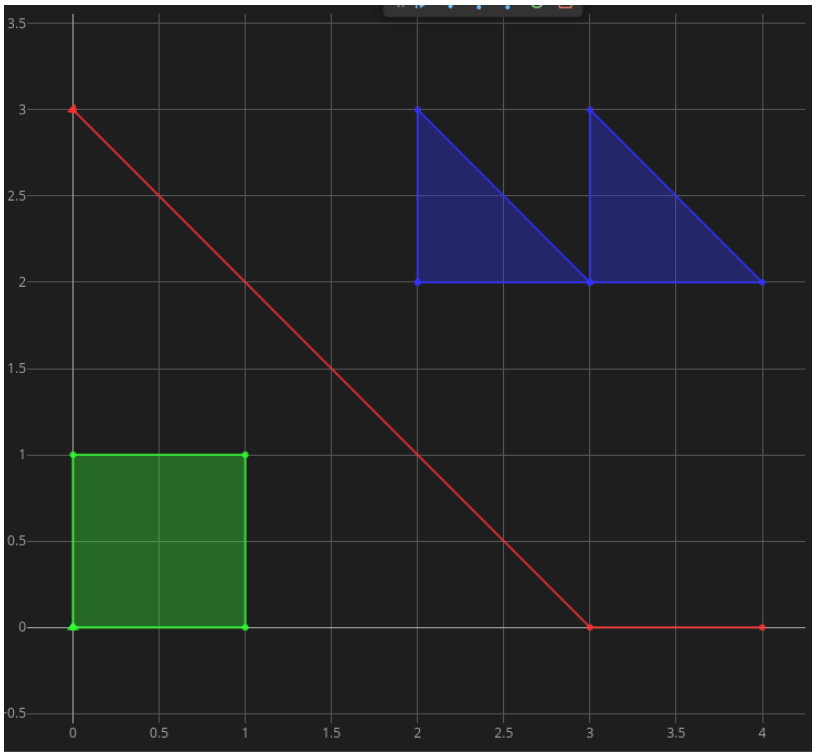
```
using point = bg::model::point<double, 2, bg::cs::cartesian>;
using linestring = bg::model::linestring<point>;
using polygon = bg::model::polygon<point>;
using multi_polygon = bg::model::multi_polygon<polygon>;

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`

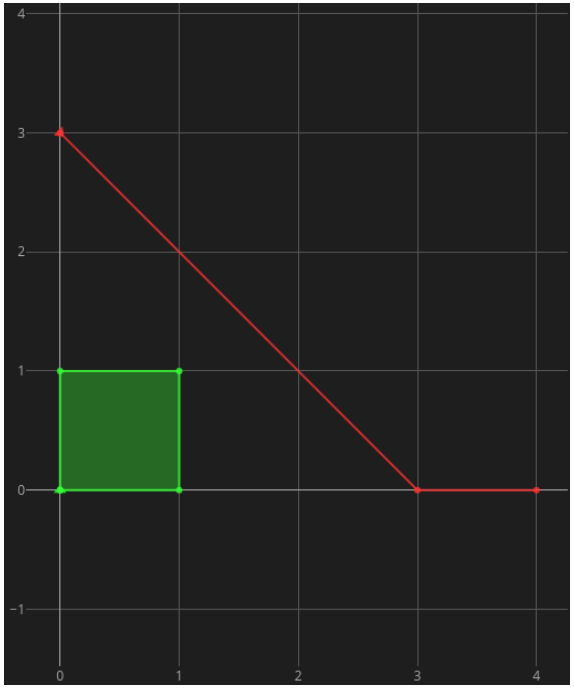
Area, Length, Perimeter

```
linestring ls;
ls.push_back(point(0.0, 3.0));
ls.push_back(point(3.0, 0.0));
ls.push_back(point(4.0, 0.0));

polygon poly = {{{0,0},{0,1},{1,1},{1,0},{0,0}}};

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

Algorithm Output



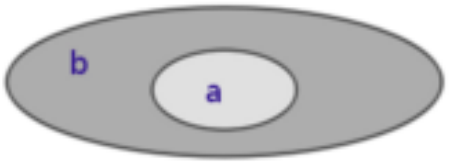
length: 5.242641

area: 1.000000

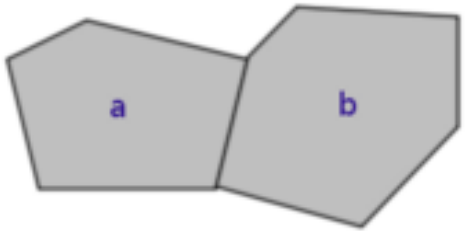
perimeter: 4.000000

Geospatial topology

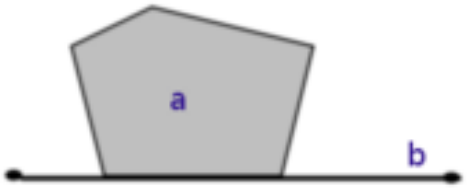
Within(a,b)



Touches(a,b)

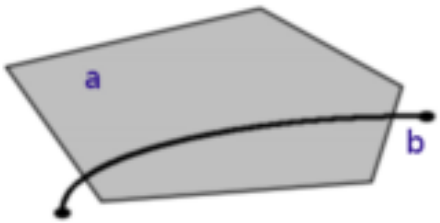


Touches(a,b)

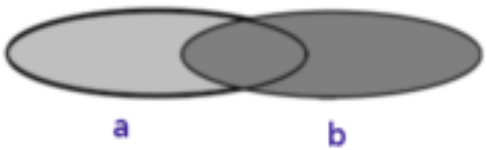


Crosses(a,b)

Crosses(a,b)



Overlaps(a,b)



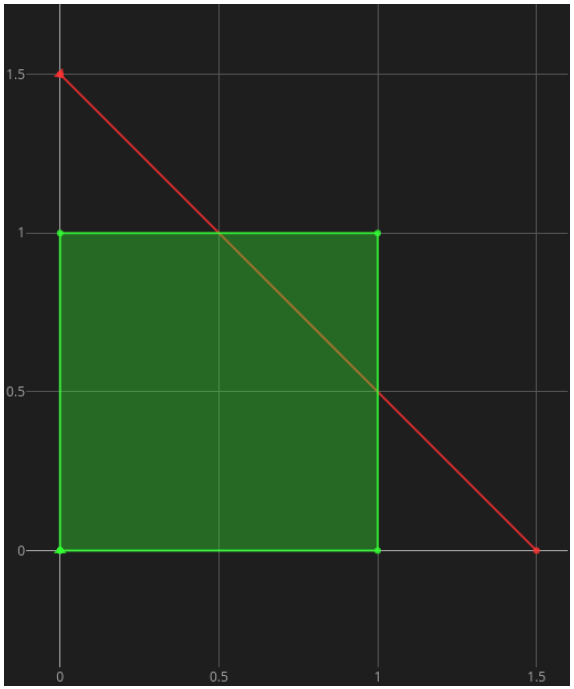
Intersects, Within

```
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

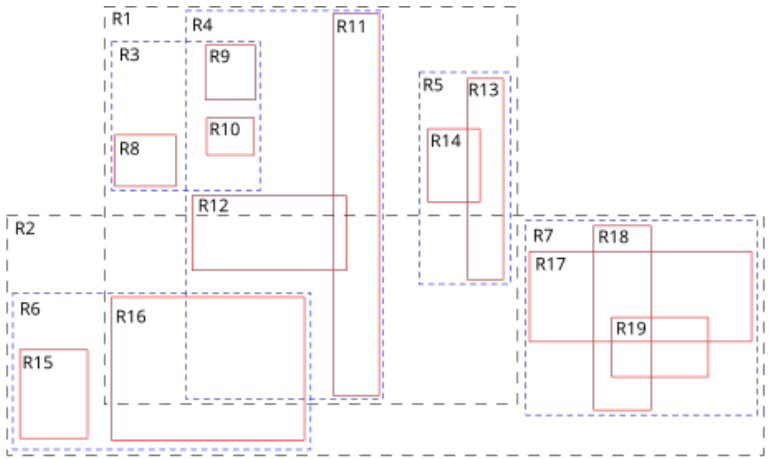


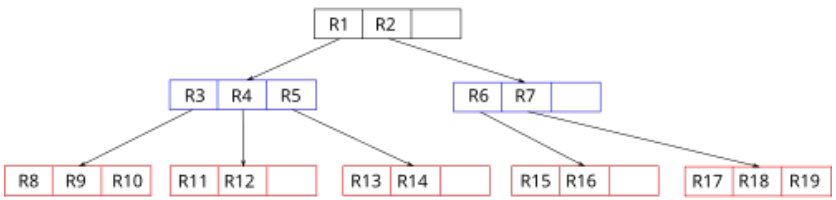
intersects: 1

within: 0

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 Types

```
#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 Insertion

```
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));
}
```

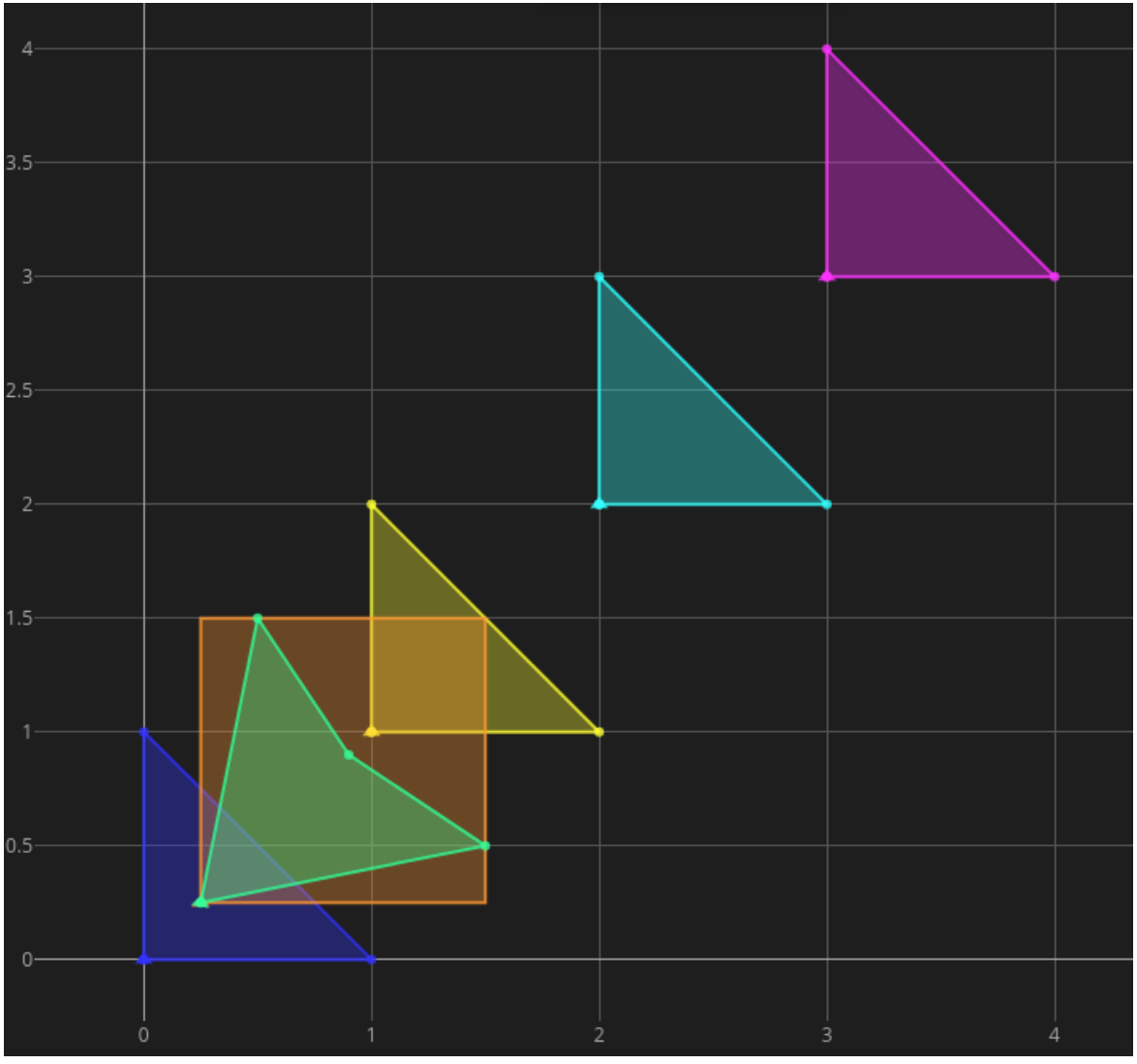
Spatial Index Query

```
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

Introduction

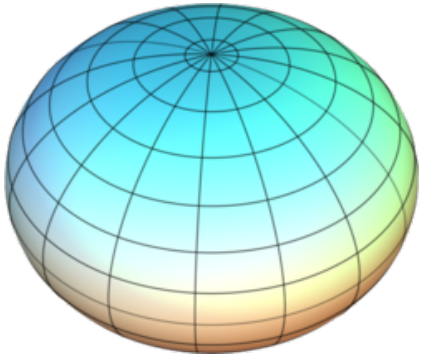
Boost Geometry design

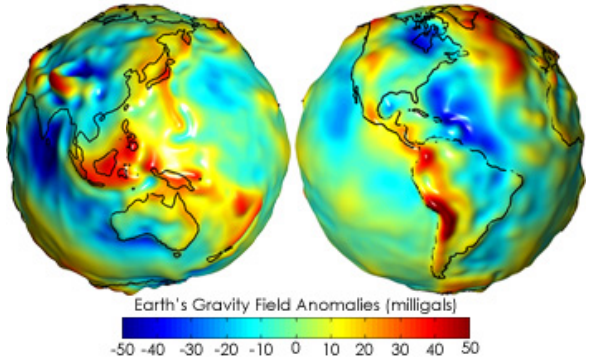
Using Boost Geometry

Spatial computations

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>
```

Distance example (revisited)

```
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)

Transformations between coordinate systems

- WGS 84 (4326): GPS lat/lon on Earth's surface.
- British National Grid (27700): flat map, units in meters.
- Same place ⇒ different numbers in each system.
- To overlay cities on hiking trails: **project** 4326 → 27700.

```
namespace bg = boost::geometry;
namespace srs = bg::srs;

using point_car = bg::model::point<double, 2, bg::cs::cartesian>;
using point_geo = bg::model::point<double, 2, bg::cs::geographic<bg::degree>>;

// Create projection from EPSG:4326 to EPSG:27700
srs::transformation<> tr1(srs::epsg(4326), srs::epsg(27700));
srs::transformation<srs::static_epsg<4326>,
                    srs::static_epsg<27700> > tr2;

point_geo pt(0.1278, 51.5074); // Longitude, Latitude for London
point_car pt_out1, pt_out2;

tr1.forward(pt, pt_out1);
tr2.forward(pt, pt_out2);
```

Projected coordinates: 547766, 180865

Projected Hiking trails

Hiking trails (orange) in Lake District (northwest England). Map data from OpenStreetMap; visualization with QGIS

Analyzing trajectories with Boost.Geometry

- Each walker's path represented as a `linestring` of GPS coordinates (lon, lat).
- Fréchet distance: captures similarity of two curves while respecting traversal order.
- Preprocess: simplify algorithm and R-tree
- No projections: same algorithm for geographic and cartesian CS

Walkers' trajectories example

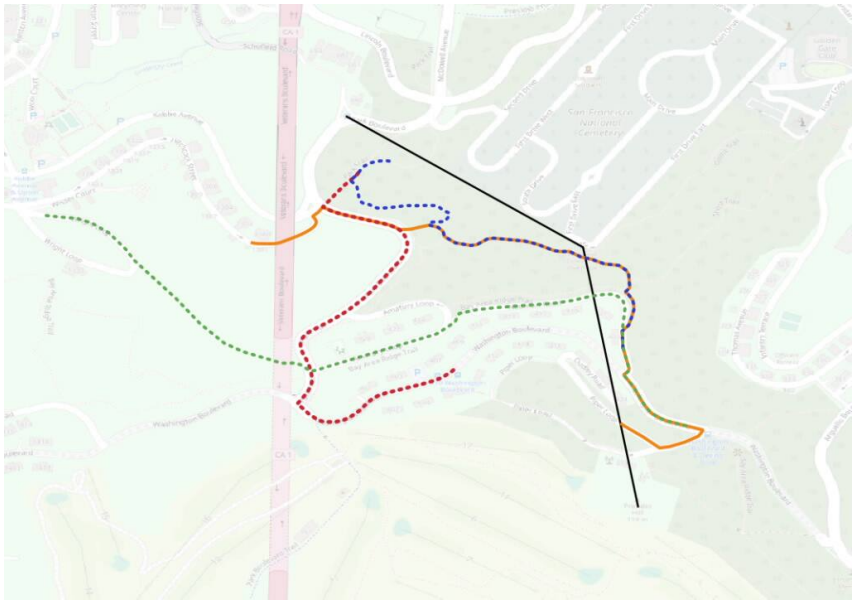
```
using point_t = bg::model::point<double, 2, bg::cs::geographic<bg::degree>>;
using linestring_t = bg::model::linestring<point_t>;

linestring_t red, black;

bg::read_wkt("LINESTRING(-122.46428847312926 37.79360088412314,...)", black);

bg::read_wkt("LINESTRING(-122.46702694852982 37.79562087383118,...)", red);

std::cout << "d: " << bg::discrete_frechet_distance(black, red) << std::endl;
```



red : 453.59273618286403 orange : 241.65190882342154 blue : 261.68893003364565 green : 429.19109867209164

Possible future steps

- 3D support (recent: PolyhedralSurface limited support)
- Support for curved geometries
- Robustness (predicates and constructions)
- Integrate newer standard features C++17/20

Thank you!

Questions?