Introduction to Boost.Geometry

Adam Wulkiewicz
Software Engineer
Agenda

1. Boost.Geometry
2. Hello World!
3. Primitives
4. Algorithms
5. Spatial Index
6. Debugging Helpers
Boost.Geometry

1. Part of Boost C++ Libraries
2. Header-only
3. C++03 support
4. Conditionally C++11
5. Metaprogramming, Tags dispatching
6. Primitives, Algorithms, Spatial Index
7. OGC SFA conformant
Boost.Geometry

1. Documentation: www.boost.org/libs/geometry
2. Mailing list: lists.boost.org/geometry
3. GitHub: github.com/boostorg/geometry
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
        >
    >;

    // Lodz -> Brussels
    std::cout << bg::distance(point(19.454722, 51.776667),
                               point(4.350000, 50.833333));
}
```
Hello World! - result

1056641.830203

maps.google.com
Primitives

1. Point, MultiPoint
2. Segment, Linestring, MultiLinestring
3. Ring, Polygon, MultiPolygon
4. Box
Primitives

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) << '
' << bg::distance(ls, mpoly);
Primitives - result

0.707107
0.707107
Primitives - adaptation

```cpp
#include <boost/geometry.hpp>
#include <boost/geometry/geometries/register/point.hpp>
#include <boost/geometry/geometries/register/linestring.hpp>
#include <iostream>
#include <vector>
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);
}
```
Primitives - result

0.707107
Algorithms

1. area, length, perimeter, num_points,
2. crosses, disjoint, distance, equals, intersects, overlaps, relate, relation, within,
3. centroid, convex_hull, envelope, buffer, simplify
4. difference, intersection, sym_difference, union_,
5. more...
Algorithms

\[
\text{linestring } \text{ls;}
\]
\[
\text{polygon } \text{poly;}
\]

\[
\text{bg::read_wkt("LINESTRING(0 3, 3 0, 4 0)"}, \text{ ls;})
\]
\[
\text{bg::read_wkt("POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))"}, \text{ poly);}
\]

\[
\text{std::cout << "length: \n" << bg::length(ls) << 'n'
}\]
\[
\text{<< "area: \n" << bg::area(poly) << 'n'
}\]
\[
\text{<< "perimeter: \n" << bg::perimeter(poly);}
\]
Algorithms - result

length 5.242641
area 1.000000
perimeter 4.000000
Algorithms

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'
  << "relation   " << bg::relation(ls, poly).str() << 'n'
  << "within      " << bg::within(ls, poly);
Algorithms - result

intersects 1
relation 101FF0212
within 0
Algorithms

```cpp
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);
```

Algorithms - result

centroid  POINT(1.882734 1.958075)
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))
libGeometry

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(32),
            bgsb::end_round(32),
            bgsb::point_circle(32));

std::cout << bg::wkt(mpoly);
MULTIPOLYGON(((1.588304 2.816228,3.700000 4.400000,3.781519 4.449740,3.871007 4.483074,3.965200 4.498788,4.060663 4.496306,4.153913 4.475721,4.241549 4.437783,4.320374 4.383876,4.387512 4.315966,4.440515 4.236530,4.477449 4.148487,4.496967 4.054987,4.498357 3.959502,4.481569 3.865495,4.447214 3.776399,2.447214 3.223607,2.327601 2.47615,0.434381 2.158114 0.474342,2.062536 0.496074,1.364555,0.498742,1.867937,0.482244,1.776393 0.447214,1.693443,0.394997,1.622273,0.327601,1.565619,0.247615,1.525658,0.158114,1.078363 1.183778,0.869395,1.027046,0.474342,0.158114,0.434381,0.247615,0.377627,0.327601,0.306557,0.394997,0.223607,0.447214,0.132063,0.482244,0.035445,0.498742,0.062536,0.496074,0.158114,0.474342,0.247615,0.434381,0.327601,0.377627,0.394997,0.306557,0.447214,0.223607,0.482244,0.132063,0.498742,0.035445,0.496074,0.062536,0.474342,0.158114,0.320943,0.618309,0.333787,0.627728,0.400000,0.700000,0.450841,0.783801,0.484357,0.875910,0.499259,0.972787,0.494975,1.070711,0.471669,1.165917,0.430237,1.254747,0.372272,1.333787,0.300000,1.400000,0.019494,1.639620,0.525658,3.158114,0.564586,3.245794,0.619542,3.324425,0.688504,0.391114,0.768932,0.443405,0.857867,0.479373,0.952034,0.497694,1.047966,3.497694,1.142133,3.479373,1.231068,3.443405,1.311496,3.391114,1.380458,3.324425,1.435414,3.245794,1.474342,3.158114,1.588304,2.816228,2.092621,1.303276,2.605573,2.329179,1.921637,1.816228,2.092621,1.303276)))
Algorithms

polygon poly1, poly2;
multi_polygon mpoly1, mpoly2;

bg::read_wkt("POLYGON((0 0,0 3,1 2.9,2 2.6,2.6 2,2.9 1,3 0,0 0))", poly1);
bg::read_wkt("POLYGON((1 1,1 4,4 4,4 1,1 1))", poly2);

bg::intersection(poly1, poly2, mpoly1);
bg::sym_difference(poly1, poly2, mpoly2);

std::cout << bg::wkt(mpoly1) << '\n';
<< bg::wkt(mpoly2);
Algorithms - result

MULTIPOLYGON(((1 3,2 3,3 2,3 1,1 1,1 3)))
MULTIPOLYGON(((1 3,1 1,3 1,3 0,0 0,0 3,1 3)),
((1 3,1 4,4 4,4 1,3 1,3 2,2 3,1 3)))
Spatial Index

1. R-tree
2. linear, quadratic or r*-tree
3. bulk-loading
4. user-defined value type
5. various spatial and knn query
6. stateful-allocator, move semantics, etc.
Spatial Index

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

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

Spatial Index

POLYGON((0 0,0 1,1 0,0 0)) intersects
POLYGON((1 1,1 2,2 1,1 1)) not intersects
Debugging Helpers

Visual Studio 2015 GraphicalDebugging extension

github.com/awulkiew/graphical-debugging
Debugging Helpers

QtCreator

Debugging Helpers

github.com/awulkiew/debugging-helpers
Thanks!
Integrated Cloud
Applications & Platform Services
Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.