Geodan

GGL
Generic Geometry Library for Boost

Barend Gehrels

Bruno Lalande

Mateusz Loskot

April 2009



Contents

- Introduction and history (10 min)
- Usage (30 min)
- Internals (30 min)
- Extensibility (20 min)

- Questions or discussions
  - Allowed during session ☺

Introduction and History

Introduction (1)

- Generic Geometry Library
- Library concerning geometry based on generic programming and templates
- Target:

```
int a[2] = {1,1};
int b[2] = {2,3};
double d = distance(a, b);
cout << "Distance :" << d;
```

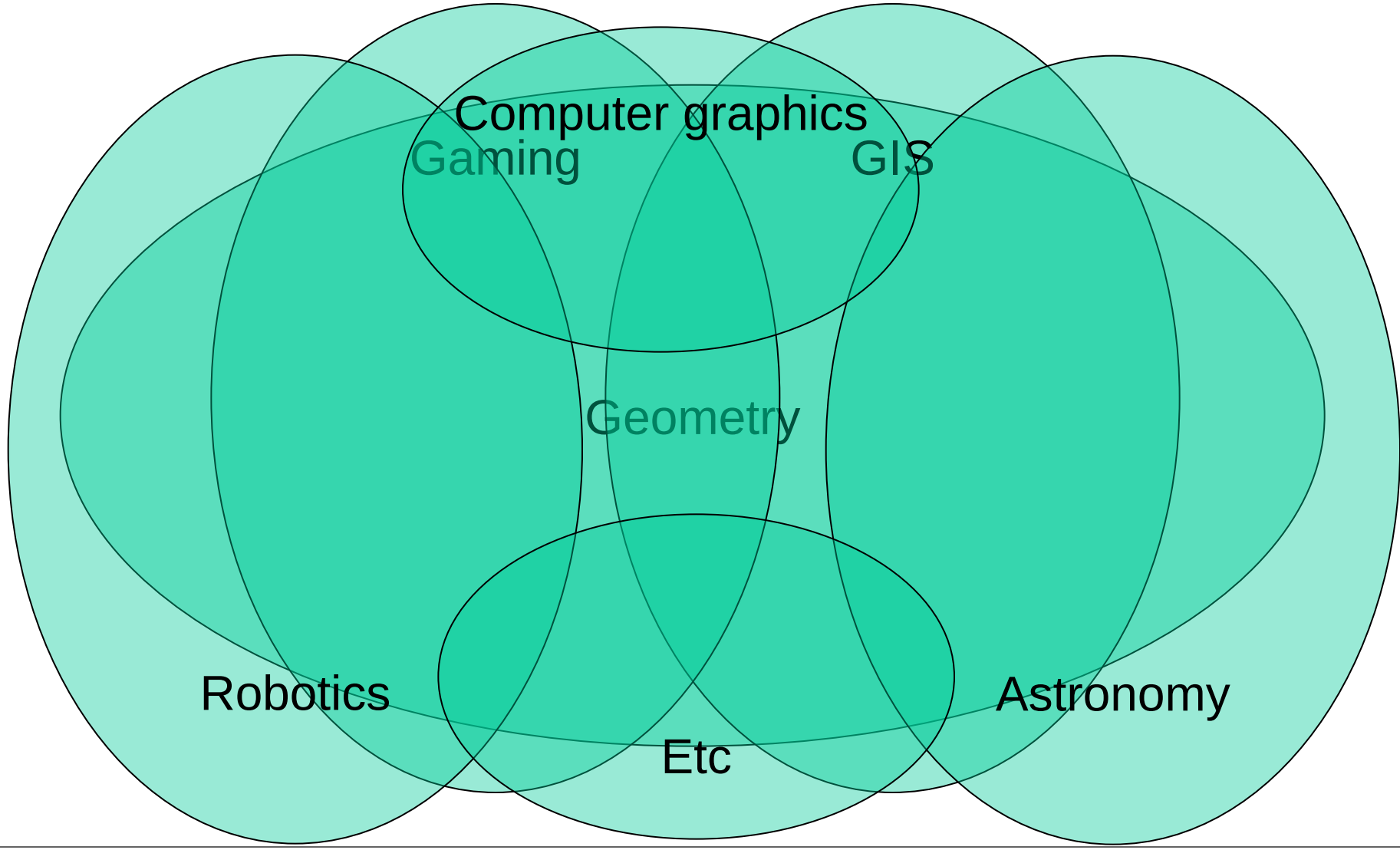
- Follows Boost conventions
- Follows std:: conventions where possible
- Follows OGC conventions where appropriate

Introduction (2)

- Dimension agnostic
- Coordinate system agnostic
- Based on concepts
- Non intrusive

- Rich
- Generic
- Blazing

Introduction (3)



History (1)

- GGL (geolib), Geodans Geographic Library
- Started in 1995
- Developed until 2003
- Frozen…
- Revived and revised in 2007
- New core (geometry) based on templates
- First preview: January 2008
- …
- Fourth preview: February 2009

History (2)

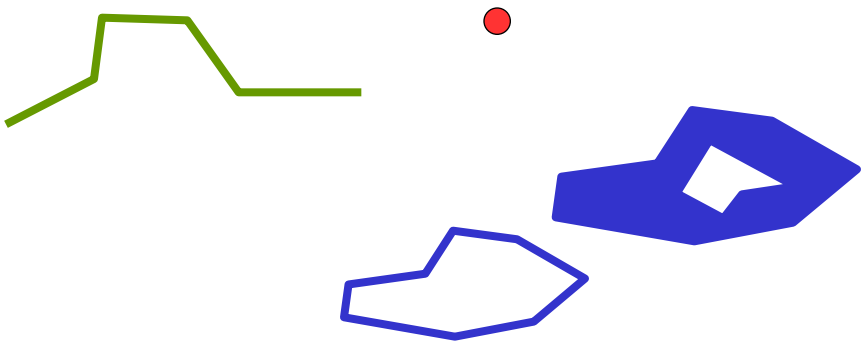
- Barend Gehrels (Geodan, Amsterdam) from (1995), 2007
- Bruno Lalande (Paris, Boost contributor) from April 2008
- Mateusz Loskot (Cadcorp, London) from March 2009

Usage

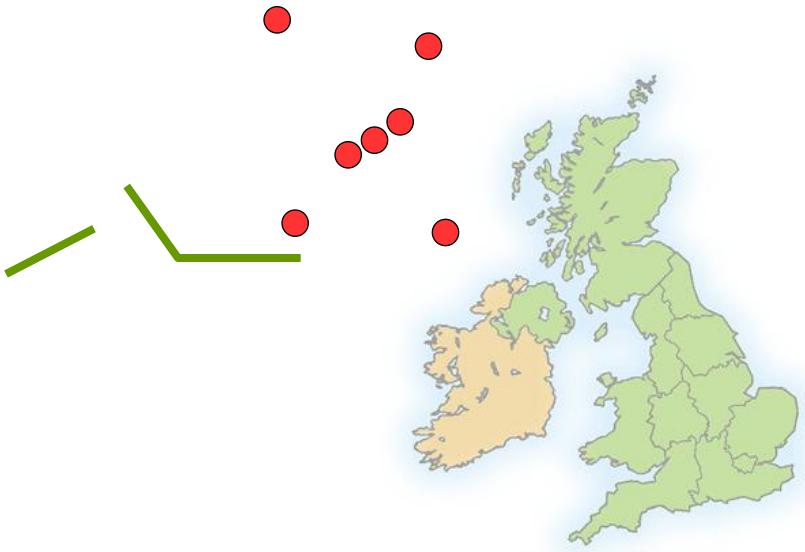


Geometries

- point
- linestring
- polygon
  - linear_ring

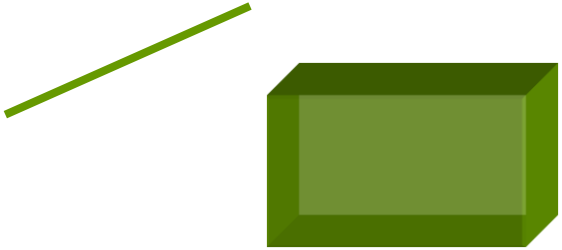


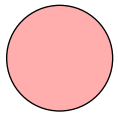
- multi_point
- multi_linestring
- multi_polygon

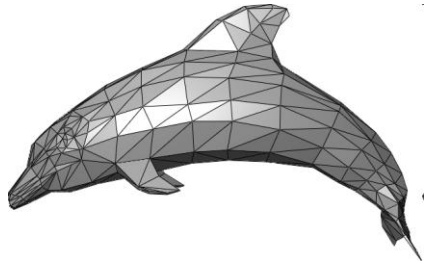


Other geometries

- "Helper" geometries
  - Segment
  - Box
- Other geometries
  - n-sphere
- 3D geometries
  - polyhedron
  - multi_polyhedron

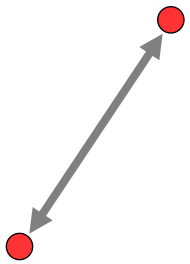






Samples: distance, points

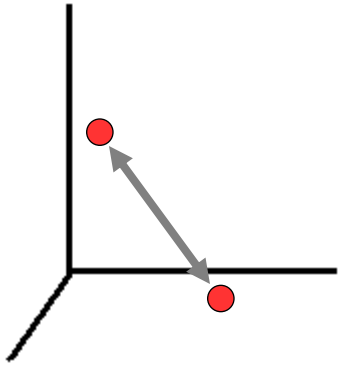
```
int a[2] = {1,1};
int b[2] = {2,3};
double d = ggl::distance(a, b);
cout << "Distance:" << d << endl;
```



```
boost::tuple<double, double> c = boost::make_tuple(3.7, 2.0);
boost::tuple<double, double> d = boost::make_tuple(5.1, 8.2);
cout << distance(c, d) << endl;

cout << distance(a, c) << endl; // on different types
```

```
typedef ggl::point<float, 3> p;
p e = ggl::make<p>(1, 2, 3);
p f = make<p>(4, 5, 6);
cout << distance(e, f) << endl; // 3D
```



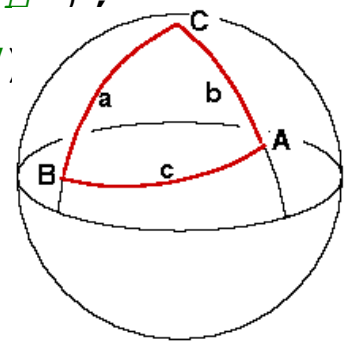
Distance, explanation

- GGL supports different point types
- Points following the concepts are supported
- double[2], double[3], tuple, ggl::point
- custom geometries

- same function **distance** for all types
- **distance** works also on different point types

Samples: distance, geographic

```
typedef point<float, 2, cs::geographic<degree> > lola;
lola amsterdam = parse<lola>("52 22 23 N", "4 53 32 E");
lola aspen = parse<lola>("39 11 32 N", "106 49 28 W");
cout << distance(amsterdam, aspen) << endl;
```



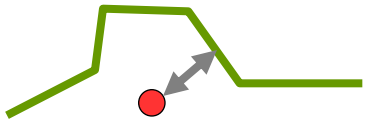
- Cartesian, spherical, geographic
- Behind the screens a different *calculation* is chosen
- This is called the **strategy**, a compile-time policy bound to coordinate system

```
cout << distance(amsterdam, aspen,
   strategy::distance::vincenty<lola>()) << endl;
```

- The strategy can also be specified by the user

Samples: distance, linestrings

```
typedef boost::tuple<double, double> p;
std::vector<p> line;
line.push_back(boost::make_tuple(1.0, 5.0));
line.push_back(boost::make_tuple(8.0, 3.4));
line.push_back(boost::make_tuple(7.5, 1.2));
p point = boost::make_tuple(3, 3);
cout << distance(line, point);
```



- Linestrings (curves, sequences of points, polylines) are std::vector's
- All algorithms work on boost::range's

```
boost::range_const_iterator<std::vector<p> >::type it;
it = boost::begin(line);
cout << distance(point, make_pair(it, it + 2)) << endl;
```

Samples: distance, distance result

```
typedef boost::tuple<int, int> p;
typedef ggl::distance_result<p, p>::type dr;
dr r = ggl::distance(make_tuple(1, 2), make_tuple(3, 4));
```

- The result is still **squared**: $a^2 + b^2$
- Expensive **sqrt** not applied

```
double d = r;
```

- Delayed calculation of sqrt
- Relevant for multi-comparisons on distance (e.g. in simplify)
- Spherical distances: no square root
- Distance might be compared in “degrees” instead (avoiding multiplying with radius each time)
- The distance result is strategy-dependant

Distance, summary

- one function
- many supported geometry-types
- many dimensions
- many behaviours
- different return types

	Pnt	Ls	Poly	MP	ML	MP
Pnt	*	*	*	*	*	*
Ls	*	*	*	*	*	*
Poly	*	*	*	*	*	*
MP	*	*	*	*	*	*
ML	*	*	*	*	*	*
MP	*	*	*	*	*	*

Cartesian	*
Spherical	*
Geographic	*

2D	*
3D	*
N-D	*

Algorithms

- All algorithms work on all geometry types…
- … and on custom types

- Many of them are “coordinate agnostic” and use strategies as “side”, “distance”
- Not all is implemented / applicable for 3D

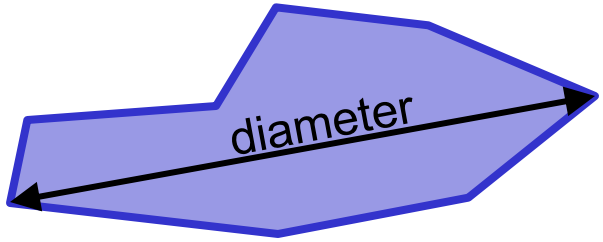
- Algorithms are modeled according to OGC
- More algorithms added or planned

Standard algorithms

- because linestrings, rings, multi* are std::vector / boost:: compatible…
- … all these algorithms can be applied
  - std::reverse(line.begin(), line.end(), …
  - std::copy(line.begin(), line.end(), …
  - boost::begin(line)
  - boost::end(line)
  - boost::size(line)
  - boost::range_const_iterator<Line>::type
  - std::for_each
  - etc

More algorithms

- geometric properties:
  - area
  - length
  - perimeter
  - centroid
  - *volume*
  - *contents*
  - envelope
  - is_convex
  - is_simple
  - diameter



Compile-time algorithms

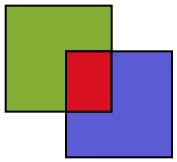
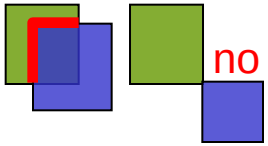
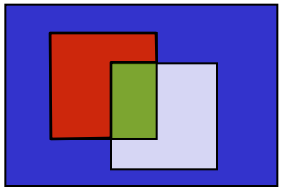
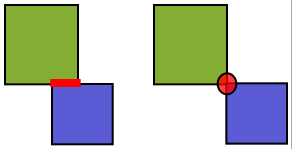
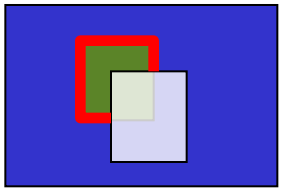
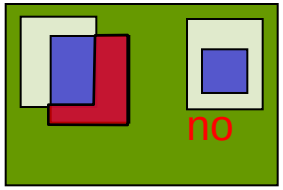
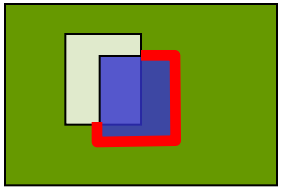
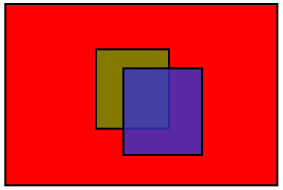
> compile-time functions / meta-functions :

- point_type
- tag
- coordinate_type
- coordinate_system
- replace_point_type
- dimension
- topological_dimension
- is_linear
- is_multi

More algorithms

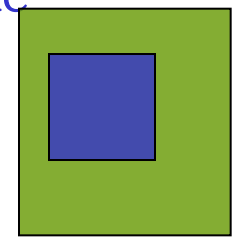
- boolean relations:
  - within
  - touches
  - intersects
  - overlaps
  - disjoint
  - contains
  - equals
  - crosses
- other relations
  - relate
  - distance

Theory of spatial relationships

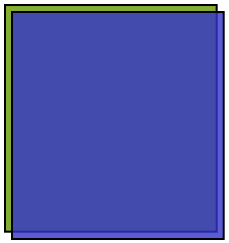
DE9IM dimension extended 9 intersection matrix (for polygons)	**Interior**	**Boundary**	**Exterior**
Interior	 *-1 / 2*	 *-1 / 1*	 *-1 (eq/in) / 2*
Boundary	 *-1 / 1*	 *-1 / 0 / 1*	 *-1 (eq/in) / 1*
Exterior	 *-1 (eq/in) / 2*	 *-1 (eq/in) / 1*	 *2*

Theory of spatial relationships

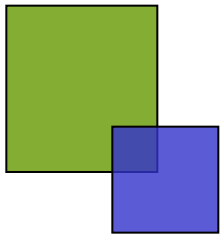
A: green
B: blue



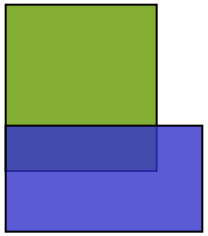
	I	B	E
I	2	1	2
B	-1	-1	1
E	-1	-1	2
within			
1--0--102			



	I	B	E
I	2	-1	-1
B	-1	1	-1
E	-1	-1	2
equals			
1---0---2			



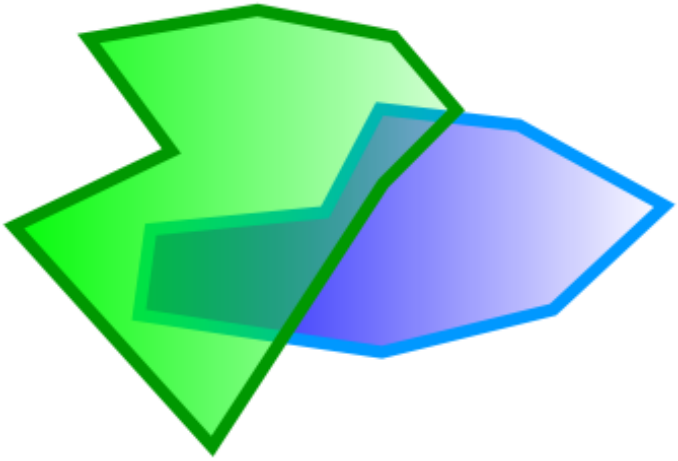
	I	B	E
I	2	1	2
B	1	0	1
E	2	1	2
overlaps			
212101212			

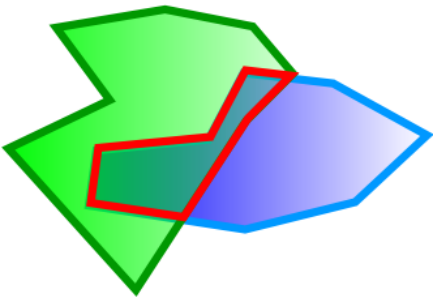


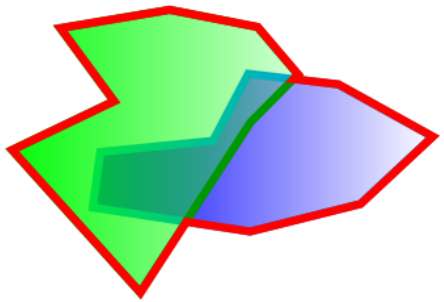
	I	B	E
I	2	1	2
B	1	1	1
E	2	1	2
overlaps (t)			
212111212			

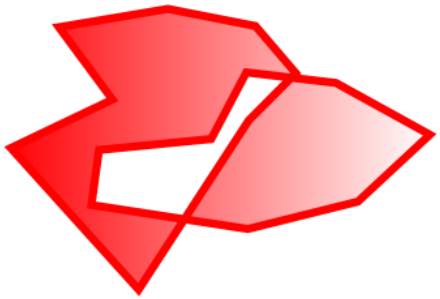
More algorithms

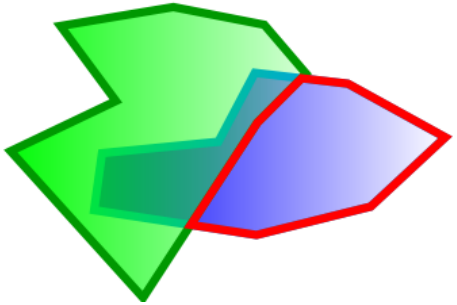
- point set relationships
  - intersection (AND)
    - clip
  - union (OR)
  - symmetric_difference (XOR)
  - difference (AND NOT)











More algorithms

>Change: – simplify – *disperse* – *spline* – *densify* – buffer – convex_hull – transform	>Utilities / internals – sectionalize – midpoints – remove_identations – remove_holes – convert
>Construction / editing – assign – parse – make – append – correct – clear	>Other: – closest_pair – *minimum bounding ball* – *largest empty circle* – *oriented bounding box*

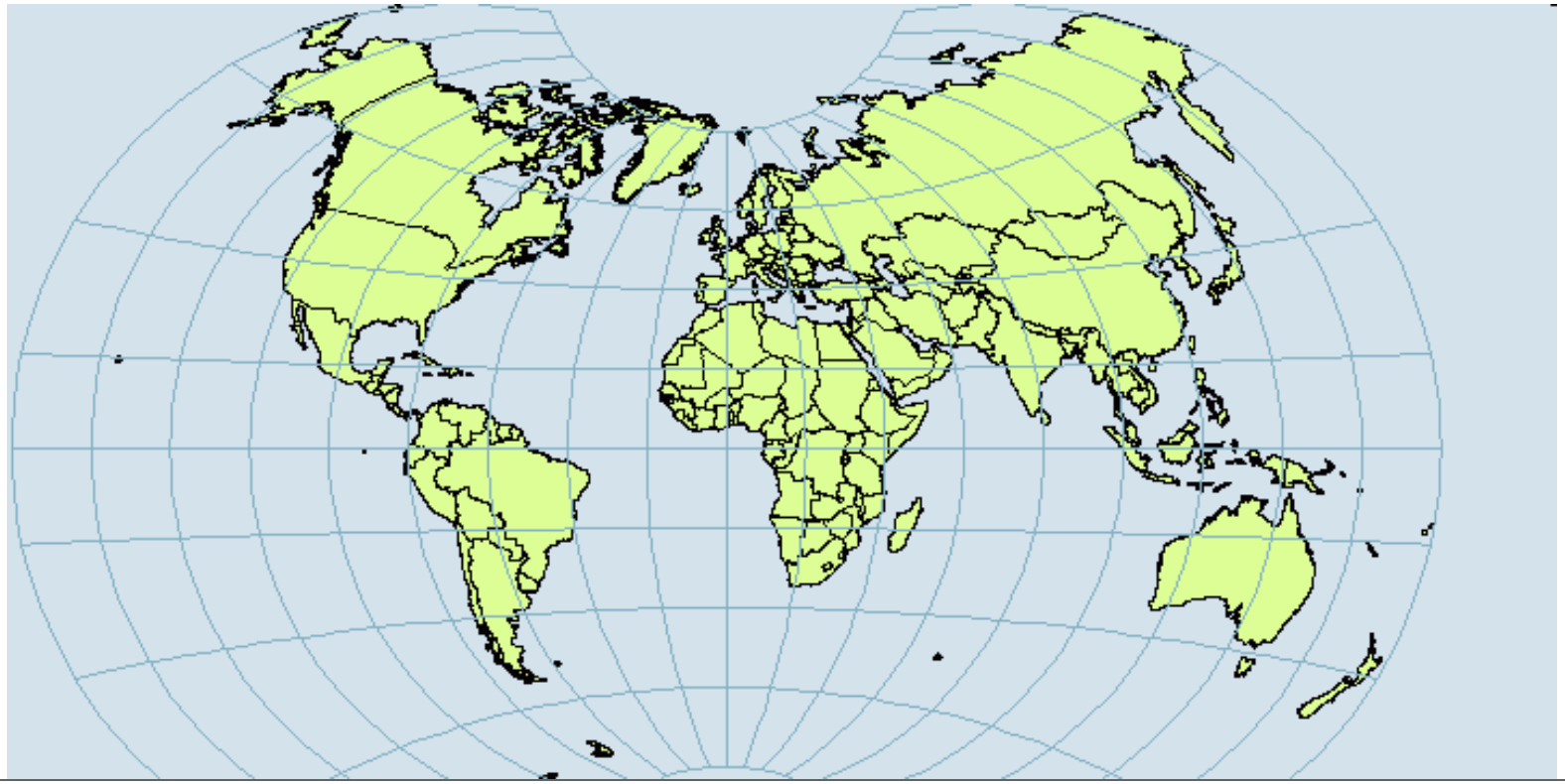
Spatial indexes

- Rtree
- BSP
- PRTree
- QuadTree

- Working on:
  - a set of geometries
  - a set of vertices within a geometry
  - a set of geometries within a multi-geometry

Projections

- Map Projections
- Proj4 conversion
- Transform



Transformations

- Matrix transformations
- uBLAS is used
- Scaling, rotation, etc

- Transform from radian to degree v.v.

- Transform via map projection

- Generic Transform

Internals and details

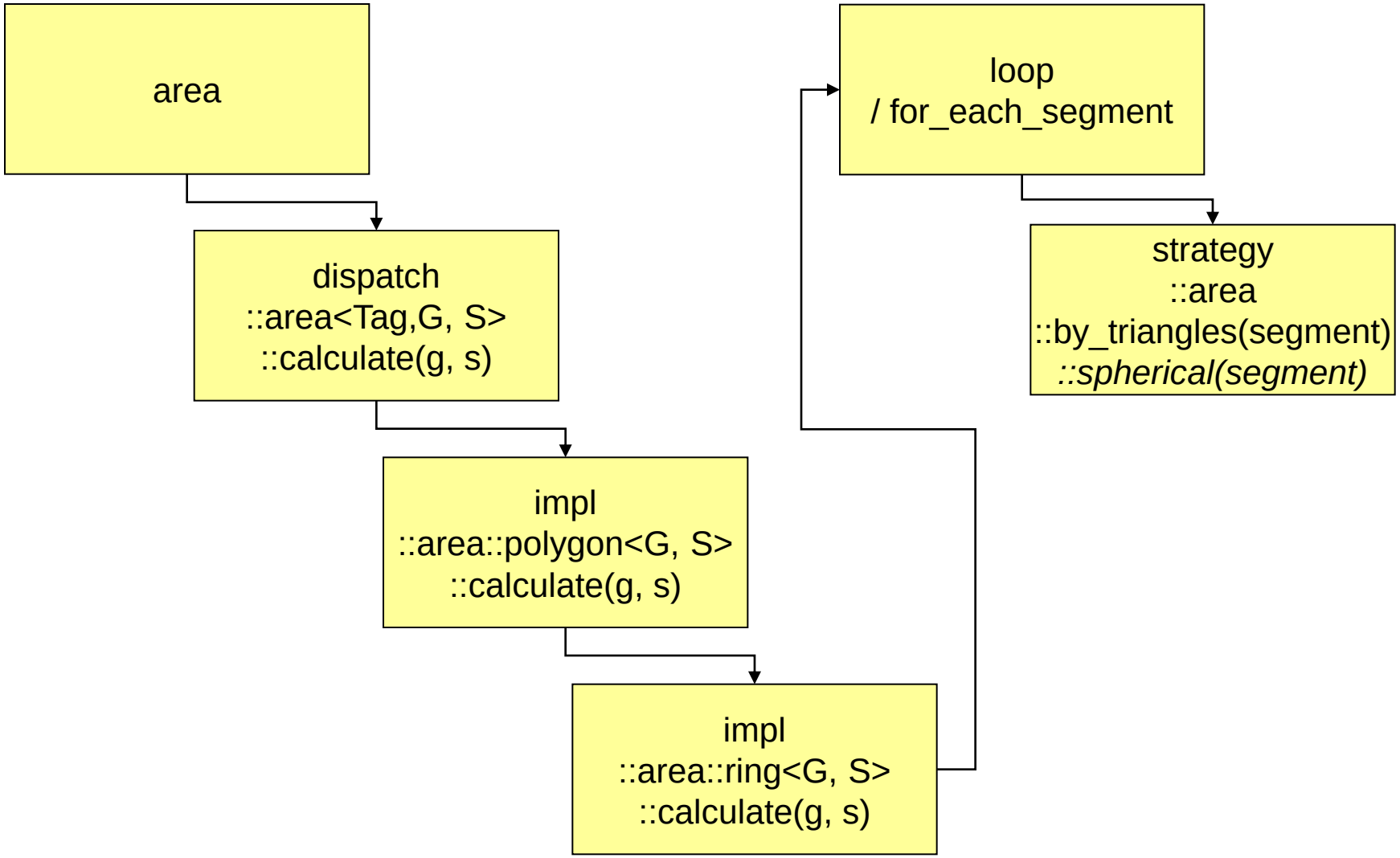
Use of other boost libraries (1)

- Boost Type Traits (e.g. remove_const)
- Boost Range
- Boost Concept Check
- Boost Numeric Conversion (cast, bounds)
- Boost MPL
- Boost Static Assert
- Boost Iterator
- Boost Smart Pointer (shared_ptr, in spatial index / projections)
- Boost uBLAS (for transformations)

Use of other Boost Libraries (2)

- Adapted geometries:
  - Boost Array
  - Boost Tuple
- IO and parsing
  - Boost Tokenizer
  - Boost Conversion (lexical cast)
  - Boost String Algo
- Testing
  - Boost Test
  - Boost Timer
- Examples and applications
  - Boost Graph
  - Boost Lambda
  - Boost Function

Generic programming and tag dispatching



Tag dispatching

- Two types of tags:
  - tag identifying kind of geometry
    - choose algorithm
  - tag identifying coordinate system
    - choose strategy
- Specializations of “dispatch”:
  - per tag (possibly of more than one geometry)
  - per dimension
  - per coordinate system
  - per other property

Concepts and tag dispatching

- "Other property": meta-functions as:
  - is_linear
  - is_multi
  - topological_dimension
- Used for specializations
- No "concept refinement"
- "meta-function finetuning"

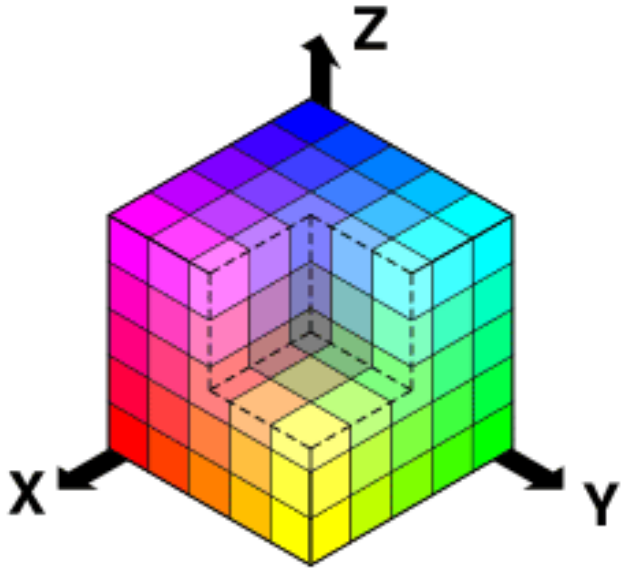
Extensibility

Customization

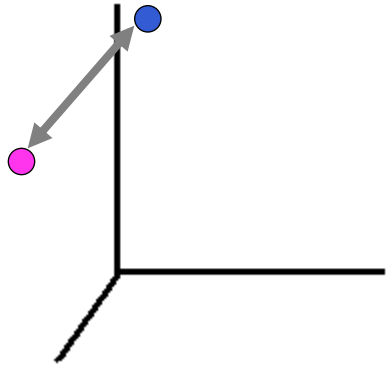
- Concepts
- Generic
- Non intrusive

- Library can handle other point types
- Add traits
- Or use “REGISTER”

```
struct my_color
{
    double red, green, blue;
};
```



Customization: color cube point



```
// Sample point, defining three color values
struct my_color
{
    double red, green, blue;
};

GEOMETRY_REGISTER_POINT_3D(my_color, double,
                cs::cartesian, red, green, blue)

my_color c1 = ggl::make<my_color>(255, 3, 233);
my_color c2 = ggl::make<my_color>(0, 50, 200);

cout << "color distance " << c1 << " to " << c2 << " is " <<
   ggl::distance(c1,c2) << endl;
```

Customization

- Custom linestring
- GPS track

- Custom polygon (triangle)
- Overriding one algorithm

Customization: GPS point

```
struct gps_point
{
    double latitude, longitude, height;
    double speed;
    // Date/time, heading, etc

    gps_point() {}

    gps_point(const std::string& c1,
        const std::string& c2,
        double h, double s)
        : height(h)
        , speed(s)
    {
        ggl::parse(*this, c1, c2);
    }
};
GEOMETRY_REGISTER_POINT_2D(gps_point, double,
        cs::geographic<degree>, longitude, latitude)
```

Customization: GPS track

```
// Declare a custom linestring which will have the GPS points
struct gps_track : std::vector<gps_point>
{
    std::string owner;
    int route_identifier;
    // etc

    gps_track(int i, const std::string& o)
        : owner(o)
        , route_identifier(i)
    {}
};

// Register the track, using traits

namespace ggl { namespace traits {

template <>
struct tag<gps_track>
{
    typedef linestring_tag type;
};

}}
```

Customization: GPS point

```
int main()
{
    // Declare a "GPS Track" and add some GPS points
    gps_track track(23, "Mister G");
    track.push_back(gps_point("52 22 23 N", "4 53 32 E", 50, 180));
    track.push_back(gps_point("52 10 00 N", "4 59 59 E", 110, 170));
    track.push_back(gps_point("52 5 20 N", "5 6 56 E", 0, 90));

    std::cout
        << "track:  " << track.route_identifier << std::endl
        << "from:   " << track.owner << std::endl
        << "as wkt: " << ggl::wkt(track) << std::endl
        << "length: " << ggl::length(track)/1000.0 << " km" << std::endl;

    return 0;
}
```

Customization: Custom triangle

```
template <typename P>
struct triangle : public boost::array<P, 3> { };


namespace ggl { namespace traits {

// Register the triangle as being a ring
template <typename P>
struct tag<triangle<P> >
{
    typedef ring_tag type;
};

}}

// somewhere
triangle<boost::tuple<double, double> > t;
    t[0] = boost::make_tuple(0, 0);
    t[1] = boost::make_tuple(5, 0);
    t[2] = boost::make_tuple(2.5, 2.5);

    std::cout << "Triangle: " << kml(t) << std::endl;
    std::cout << "Area: " << area(t) << std::endl;
```

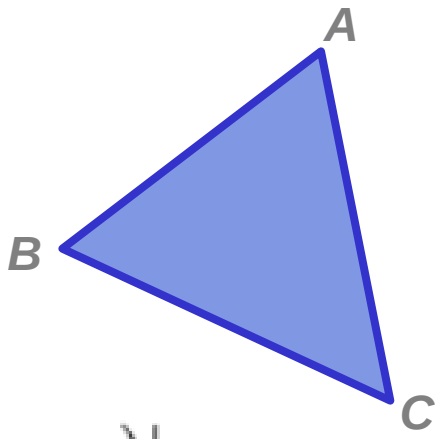
Customization, triangle

- All algorithms for triangle are that of RING
- But wait…
- For a triangle there is also an easier formula

Customization: Custom triangle

```
// Specializations of area dispatch structure, implement algorithm
namespace ggl { namespace dispatch {

template<typename P, typename Strategy>
struct area<ring_tag, triangle<P>, Strategy >
{
    static inline double calculate(const triangle<P>& t, const Strategy &)
    {
        return
          0.5  * (
             (get<0>(t[2]) - get<0>(t[0])) * (get<1>(t[1]) - get<1>(t[0]))
           - (get<0>(t[1]) - get<0>(t[0])) * (get<1>(t[2]) - get<1>(t[0]))
                );
    }
};
```



$$S = \frac{1}{2}|(x_C - x_A)(y_B - y_A) - (x_B - x_A)(y_C - y_A)|$$

Summary

- **Generic** library for **Geometry**
- First library for Geography based on Generic Programming
- More than Geography

- Boost needs Geometry!

Thanks!

barend@geodan.nl

Geodan
President Kennedylaan 1
1079 MB Amsterdam (NL)
Tel: +31 (0)20 – 5711 311
Fax: +31 (0)20 – 5711 333
E-mail: info@geodan.nl
Web: www.geodan.nl