Boost.Geometry

presented by Barend Gehrels

Formerly / aka GGL

Generic Geometry Library

presentation.cpp

```
int main()
{
   int tired = !1;
   long story;

   !1;

   !!!(0);

   !!!!!(–0);

   !!!!!!!(– – –0);

   return tired;
}
```

Your Comeau C/C++ test results are as follows:

```
Comeau C/C++ 4.3.10.1 (Oct  6 2008 11:28:09) for ONLINE_EVALUATION_BETA2
Copyright 1988-2008 Comeau Computing.  All rights reserved.
MODE:strict errors C++ C++0x_extensions

"ComeauTest.c", line 6: warning: expression has no effect
        !1;
        ^

"ComeauTest.c", line 8: warning: expression has no effect
        !!!(0);
        ^

"ComeauTest.c", line 10: warning: expression has no effect
        !!!!(-0);
        ^

"ComeauTest.c", line 12: warning: expression has no effect
        !!!!!!(- - -0);
        ^

"ComeauTest.c", line 4: warning: variable "story" was declared but never referenced
        long story;
             ^
```

In strict mode, with -tused, **Compile succeeded** (but remember, the Comeau online compiler does not link).
Compiled with C++0x extensions enabled.

Contents

- Introduction
- Features
- Usage / example (building a UI)
- Spatial set theory
- Design rationele

Introduction

Introduction

- What is Boost.Geometry
  - defines:
    - concepts for geometries
    - algorithms based on those concepts
    - Strategies (policies)
    - Other (iterators, ranges, policies, meta-functions)
  - dimension-agnostic
  - coordinate-system-agnostic
  - scalable kernel
  - based on generic programming

- Standards followed
  - std::
  - boost::
  - ISO / OGC

People

- Barend Gehrels
- Bruno Lalande
- Mateusz Loskot
- GGL Mailing list
  - Currently ~50 subscriptions
- Boost Mailing list
- Current users
  - Merkaartor (Open Street Map)
  - Open Graph Router
  - Flight Logbook
  - Games (Tangram)
  - Geodan
  - …

Timeline

- 1995 (Geodan Geographic Library)
- 2008, first preview (Geometry Library)
- 2009, fourth preview (Generic Geometry Library)
- November 2009: review and acceptance (Boost.Geometry)
- Boost 1.44? 1.45? 1.46?

Review and acceptance

- Review period: November 5, 2009 – November 23, 2009
- Review manager: Hartmut Kaiser
- 14 reviewers
- Review report: November 28, 2009
  - 12 votes Yes
  - 2 votes No
  - Several conditions
- Quote: *"The design is very clear. I think it can serve as a standard example of how to cover a big non trivial problem domain using meta-programming, partial specialization and tag dispatch to make it uniformly accessible by a set of generic algorithms"*

Challenges

- Build it generic
  - Combinatorial explosion of possibilities
- Make it fast and robust
- Scope
- Satisfy many

Features (1)

Geometry Models

0-dimensional	1-dimensional	2-dimensional
boost::geometry::point	boost::geometry::segment	boost::geometry::box
boost::geometry::point_xy	boost::geometry::segment_2d	boost::geometry::box_2d
boost::geometry::point_2d	boost::geometry::linestring	boost::geometry::box_3d
boost::geometry::point_3d	boost::geometry::linestring_2d	boost::geometry::box
	boost::geometry::linestring_3d	boost::geometry::linear_ring
		boost::geometry::ring_2d
		boost::geometry::ring_3d
		boost::geometry::polygon
		boost::geometry::polygon_2d
		boost::geometry::polygon_3d

Adapted:
Boost.Tuple, Boost.Array, C Array, std::vector, std::deque, std::pair

Features (2)

Geometry Concepts

0-dimensional

boost::geometry::concept::Point
boost::geometry::concept::ConstPoint

1-dimensional

boost::geometry::concept::Segment
boost::geometry::concept::ConstSegment
boost::geometry::concept::Linestring
boost::geometry::concept::ConstLinestring

2-dimensional

boost::geometry::concept::Box
boost::geometry::concept::ConstBox
boost::geometry::concept::Ring
boost::geometry::concept::ConstRing
boost::geometry::concept::Polygon
boost::geometry::concept::ConstPolygon

Functions

boost::geometry::concept::check
boost::geometry::concept::check_concepts_and_equal_dimensions

Features (3)

Core

Metafunctions

boost::geometry::cs_tag
boost::geometry::coordinate_type
boost::geometry::coordinate_system
boost::geometry::dimension
boost::geometry::geometry_id
boost::geometry::interior_type
boost::geometry::is_linear
boost::geometry::is_multi
boost::geometry::is_radian
boost::geometry::point_order
boost::geometry::point_type
boost::geometry::ring_type
boost::geometry::replace_point_type
boost::geometry::reverse_dispatch
boost::geometry::tag
boost::geometry::topological_dimension

Access Functions

boost::geometry::exterior_ring
boost::geometry::get
boost::geometry::get_as_radian
boost::geometry::interior_rings
boost::geometry::num_interior_rings
boost::geometry::num_points
boost::geometry::set
boost::geometry::set_from_radian

Classes

boost::geometry::exception
boost::geometry::centroid_exception

Features (4)

Coordinate Systems	Iterators	
Classes boost::geometry::cs::cartesian boost::geometry::cs::geographic boost::geometry::cs::polar boost::geometry::cs::spherical	**Metafunctions** boost::geometry::range_type **Classes** boost::geometry::circular_iterator boost::geometry::ever_circling_iterator boost::geometry::one_section_segment_iterator boost::geometry::section_iterator boost::geometry::segment_iterator	**Functions** boost::geometry::make_segment_iterator boost::geometry::operator== boost::geometry::operator!=

Features (5)

Algorithms

Geometry Constructors

boost::geometry::make
boost::geometry::make_inverse
boost::geometry::make_zero

Predicates

boost::geometry::disjoint
boost::geometry::equals
boost::geometry::intersects
boost::geometry::overlaps
boost::geometry::selected
boost::geometry::within

Append

boost::geometry::append

Area

boost::geometry::area

Assign

boost::geometry::assign
boost::geometry::assign_box_corners
boost::geometry::assign_inverse
boost::geometry::assign_point_from_index
boost::geometry::assign_point_to_index
boost::geometry::assign_zero

Buffer

boost::geometry::buffer
boost::geometry::make_buffer

Centroid

boost::geometry::centroid
boost::geometry::make_centroid

Clear

boost::geometry::clear

Combine

boost::geometry::combine

Convert

boost::geometry::convert

Convex Hull

boost::geometry::convex_hull
boost::geometry::convex_hull_inserter

Correct

boost::geometry::correct

Features (6)

Distance

boost::geometry::distance

Difference

boost::geometry::difference
boost::geometry::sym_difference

Dissolve

boost::geometry::dissolve

Envelope

boost::geometry::envelope
boost::geometry::make_envelope

for_each

boost::geometry::for_each_point
boost::geometry::for_each_segment

Intersection

boost::geometry::intersection_inserter

Length

boost::geometry::length

Overlay

boost::geometry::copy_segments
boost::geometry::copy_segment_point
boost::geometry::copy_segment_points
boost::geometry::enrich_intersection_points
boost::geometry::get_turns
boost::geometry::traverse

Perimeter

boost::geometry::perimeter

Reverse

boost::geometry::reverse

Section

boost::geometry::get_section
boost::geometry::sectionalize

Simplify

boost::geometry::simplify
boost::geometry::simplify_inserter

Transform

boost::geometry::transform

Union

boost::geometry::union_inserter

Unique

boost::geometry::unique

Miscellaneous Utilities

boost::geometry::parse

Features (7)

Strategies

Area	Buffer	Centroid
boost::geometry::strategy_area boost::geometry::area_result boost::geometry::strategy::area::by_triangles boost::geometry::strategy::area::huiller	boost::geometry::strategy::buffer::join_miter boost::geometry::strategy::buffer::join_bevel boost::geometry::strategy::buffer::join_round	boost::geometry::strategy_centroid boost::geometry::strategy::centroid_::bashein_detmer boost::geometry::strategy::centroid_::centroid_average
Compare	**Convex Hull**	**Distance**
boost::geometry::strategy_compare boost::geometry::strategy::compare::default_strategy boost::geometry::strategy::compare::circular_comparator	boost::geometry::strategy_convex_hull boost::geometry::strategy::convex_hull::graham_andrew	boost::geometry::strategy_distance boost::geometry::strategy_distance_segment boost::geometry::cartesian_distance boost::geometry::distance_result boost::geometry::make_distance_result boost::geometry::close_to_zero boost::geometry::fuzzy_equals boost::geometry::strategy::distance::projected_point boost::geometry::strategy::distance::pythagoras boost::geometry::strategy::distance::cross_track boost::geometry::strategy::distance::haversine

Features (8)

Intersection

- boost::geometry::de9im
- boost::geometry::de9im_segment
- boost::geometry::segment_intersection_points
- boost::geometry::strategy_intersection
- boost::geometry::strategy::intersection::liang_barsky
- boost::geometry::strategy::intersection::relate_cartesian_segments
- boost::geometry::strategy::intersection::relate_cartesian_segments

Side

- boost::geometry::strategy_side
- boost::geometry::side_info
- boost::geometry::strategy::side::course
- boost::geometry::strategy::side::side_by_triangle
- boost::geometry::strategy::side::side_by_cross_track

Simplify

- boost::geometry::strategy::simplify::douglas_peucker

Transform

- boost::geometry::strategy_transform
- boost::geometry::strategy::copy_direct
- boost::geometry::strategy::copy_per_coordinate
- boost::geometry::strategy::degree_radian_vv
- boost::geometry::strategy::degree_radian_vv_3
- boost::geometry::strategy::from_spherical_2_to_cartesian_3
- boost::geometry::strategy::from_spherical_3_to_cartesian_3
- boost::geometry::strategy::from_cartesian_3_to_spherical_2
- boost::geometry::strategy::from_cartesian_3_to_spherical_3
- boost::geometry::strategy::inverse_transformer
- boost::geometry::strategy::map_transformer
- boost::geometry::strategy::ublas_transformer
- boost::geometry::strategy::translate_transformer
- boost::geometry::strategy::scale_transformer
- boost::geometry::strategy::rotate_transformer

Within

- boost::geometry::strategy::winding
- boost::geometry::strategy::crossings_multiply
- boost::geometry::strategy::franklin

Miscellaneous Utilities

- boost::geometry::strategy::not_implemented

Features (9)

Policies

Compare

boost::geometry::equal_to
boost::geometry::greater
boost::geometry::less

Relate

boost::geometry::policies::relate::direction_type
boost::geometry::policies::relate::segments_de9im
boost::geometry::policies::relate::segments_direction
boost::geometry::policies::relate::segments_intersection_points
boost::geometry::policies::relate::segments_tupled

Strategy Concepts

boost::geometry::concept::AreaStrategy
boost::geometry::concept::CentroidStrategy
boost::geometry::concept::ConvexHullStrategy
boost::geometry::concept::PointDistanceStrategy
boost::geometry::concept::PointSegmentDistanceStrategy
boost::geometry::concept::SegmentIntersectStrategy
boost::geometry::concept::SimplifyStrategy
boost::geometry::concept::WithinStrategy

Features (10)

Arithmetic

Add	Subtract	Multiply	Divide
boost::geometry::add_point boost::geometry::add_value	boost::geometry::subtract_point boost::geometry::subtract_value	boost::geometry::multiply_point boost::geometry::multiply_value	boost::geometry::divide_point boost::geometry::divide_value

Products

boost::geometry::cross_product
boost::geometry::dot_product

Extensions

TODO

...

Generic Programming Techniques

- Based on templates
- Based on Concepts implemented using Traits
- Based on tag dispatching **by type**
- Based on meta programming (MPL)
- Using metafunctions
  - type
  - value
- “Apply”
- Policies and Strategies
- Metafunction-finetuning, dispatching / specializations by metafunctions

Usage / example

Building a UI

Building a UI

- For example:
  - Using wxWidgets points
  - Read countries
  - Display them
  - Follow mouse

Point Concept

- wxWidgets has “wxPoint”
- wxPoint is adapted to Point Concept:

```
BOOST_GEOMETRY_REGISTER_POINT_2D(wxPoint, int, cs::cartesian, x, y)
```

- wxPoint can now be used in any Boost.Geometry algorithm
- Like:

```
wxPoint p1(1,3);
wxPoint p2(3,4);
double d = boost::geometry::distance(p1, p2);
```

Register Point

```
BOOST_GEOMETRY_REGISTER_POINT_2D(wxPoint, int, cs::cartesian, x, y)
```

- Is a shortcut for:

```
namespace boost { namespace geometry { namespace traits {

template<> struct tag<wxPoint> { typedef point_tag type; };
template<> struct dimension<wxPoint> : boost::mpl::int_<2> {};
template<> struct coordinate_type<wxPoint> { typedef int type; };
template<> struct coordinate_system<wxPoint>
{ typedef cs::cartesian type; };

template<> struct access<wxPoint, 0>
{
    static int get(wxPoint const& p) { return p.x; }
};

template<> struct access<wxPoint, 1>
{
    static int get(wxPoint const& p) { return p.y; }
};

}}}
```

Read countries

- E.g. from strings (Well-Known Text, ISO/OGC)
- Or from file (KML etc)
- File formats not in Kernel (WKT in extension)

```
boost::geometry::assign_inverse(m_box);

// Usually within a loop throughout a stream
std::getline(a_stream, line);
if (! line.empty())
{
   Geometry geometry;
   boost::geometry::read_wkt(line, geometry);
   m_countries.push_back(geometry);
   boost::geometry::combine(m_box,
        boost::geometry::make_envelope<Box>(geometry));
}
```

Transform

- Define map_transformer strategy

```
typedef boost::geometry::strategy::transform::map_transformer
        <
            boost::geometry::point_2d, wxPoint, true, true
        > map_transformer_type;
```

- Construct it (in this case with box)

```
wxSize sz = dc.GetSize();
map_transformer_type m(m_box, sz.x, sz.y));

BOOST_FOREACH(bg::polygon_2d const& poly, country)
{
   std::size_t n = boost::size(poly.outer());
   boost::scoped_array<wxPoint> points(new wxPoint[n]);
   bg::transform(poly.outer(),
        std::make_pair(points.get(), points.get() + n), m);
   dc.DrawPolygon(n, points.get());
}
```

Transform back

- To follow mouse in world coordinates

```
typedef boost::geometry::strategy::transform::inverse_transformer
        <
            wxPoint, boost::geometry::point_2d
        > inverse_transformer_type;

inverse_transformer_type(m_map_transformer);

bg::point_2d point;
bg::transform(event.GetPosition(), point, m_inverse_transformer);

std::ostringstream out;
out << "Position: " << bg::get<0>(point) << ", " << bg::get<1>(point);
m_owner->SetStatusText(wxString(out.str().c_str(), wxConvUTF8));
```

Result

- c:_svn\boost\sandbox\geometry\libs\geometry\example\Debug\x04_wxwidgets_world_mapper.exe

More mapping

- Country name?

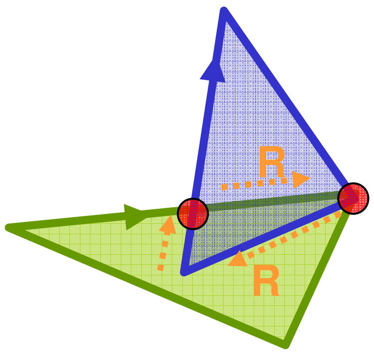
```
struct country : public bg::multi_polygon<...>
{
     std::string name;
     int nr_of_inhabitants;
};
// Register it using traits or registration macro
```

- Display country name

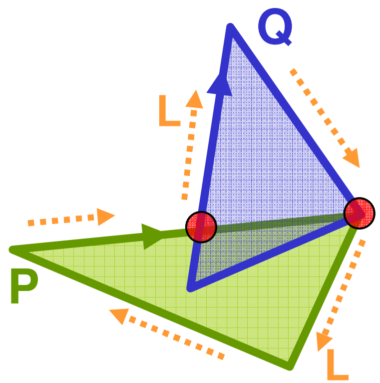
```
bg::point_2d point;
bg::transform(event.GetPosition(), point, m_inverse_transformer);
BOOST_FOREACH(country const& c, countries)
{
     if (boost::geometry::within(point, c))
     {
        // display c.name
     }
```

Pieces of
Spatial Set Theory

Spatial Set Theory (1)

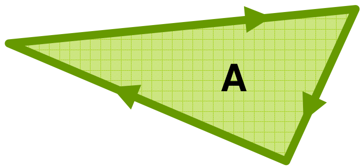


Intersection:
Take the right turn everywhere

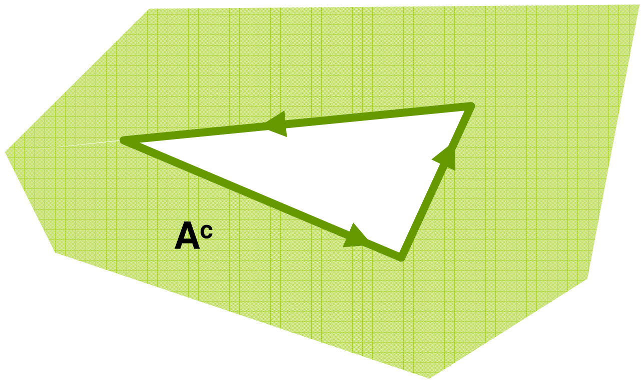


Union:
Take the left turn everywhere

Spatial Set Theory (2)



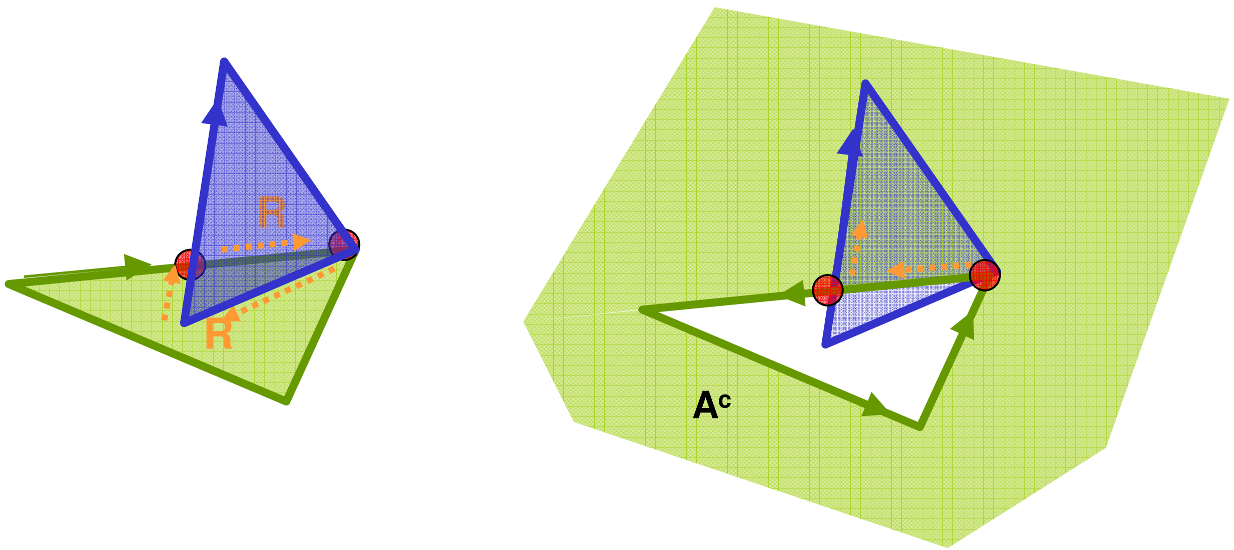
A: Polygon (clockwise)



A^c:
- Complement of A
- Counter clockwise

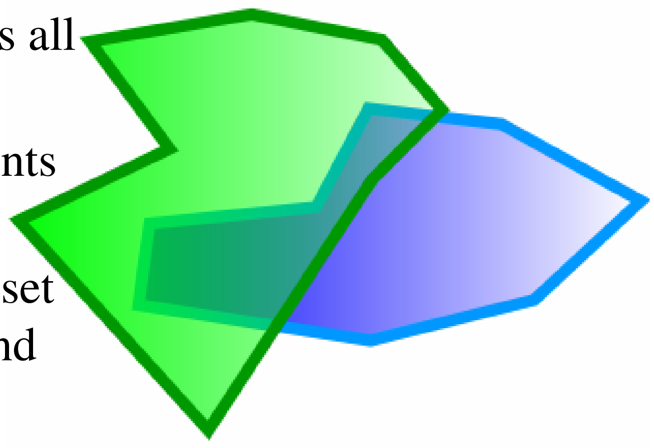
Spatial Set Theory (3)

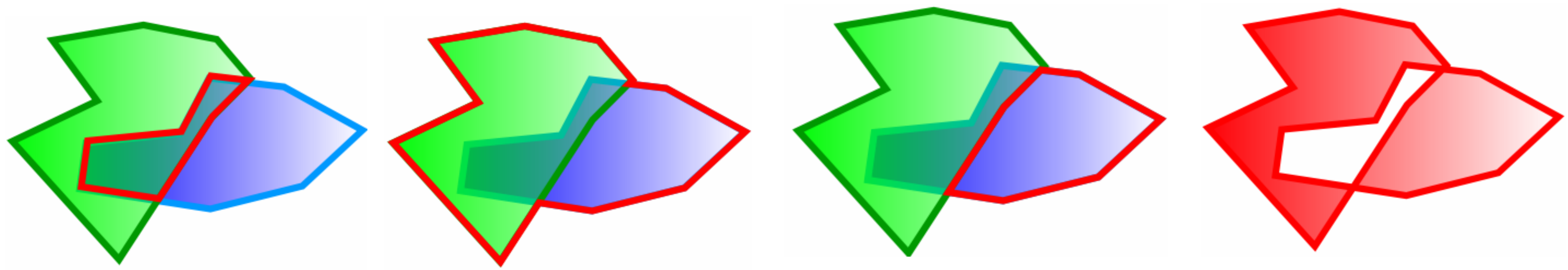
- B \ A : the difference of two sets A and B is the set of elements which belong to B but not to A
- B \ A = $A^c \cap B$



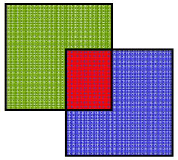
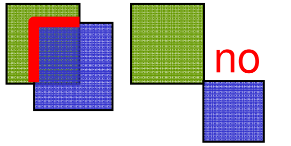
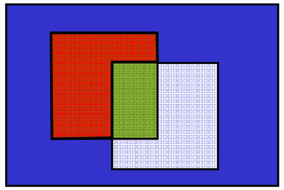
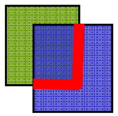
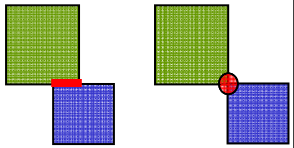
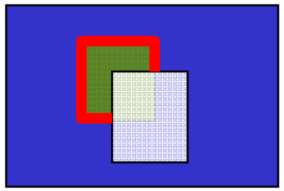
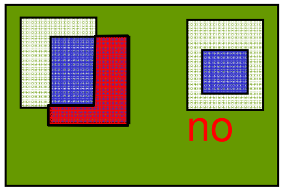
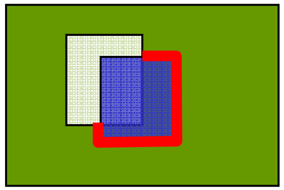
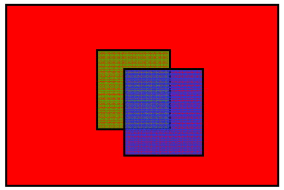
Spatial Set Theory (4)

- $A \cap B$: the intersection of two sets A and B is the set that contains all elements of A that also belong to B (aka AND)
- $A \cup B$: the union of two sets A and B is the set that contains all elements that belong to A or B (aka OR)
- $B \setminus A$: the difference of two sets A and B is the set of elements which belong to B but not to A: $\mathbf{A^c \cap B}$
- $A \Delta B$: the symmetric difference of two sets A and B is the set of elements which belong to either A or to B, but not to A and B (aka XOR): $\mathbf{(A^c \cap B) \cup (B^c \cap A)}$





Dimensionly Extended 9 IM

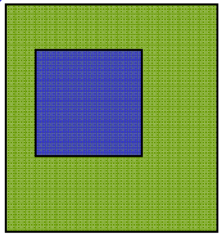
DE9IM **dimension extended** **9 intersection** **matrix** **(for polygons)**	**Interior**	**Boundary**	**Exterior**
Interior	 *-1 / 2*	 *-1 / 1*	 *-1 (eq/in) / 2*
Boundary	 *-1 / 1*	 *-1 / 0 / 1*	 *-1 (eq/in) / 1*
Exterior	 *-1 (eq/in) / 2*	 *-1 (eq/in) / 1*	 2

0,1,2:
topological
dimension

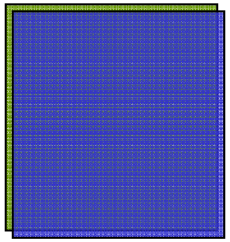
Theory

B: blue

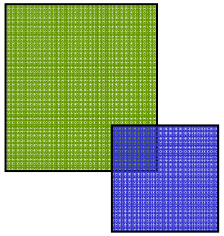
A: green



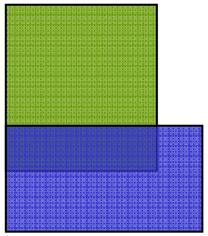
	I	B	E
I	2	1	2
B	-1	-1	1
E	-1	-1	2
within			
1--0--102			



	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			

Design Rationele

Design Rationele

- Paper “Generic Programming for Geometry”
  - BoostCon ’10
- Function “distance”
- Step by step more generic

Step 1: trivial

```
struct mypoint
{
   double x, y;
};

double distance(mypoint const& a, mypoint const& b)
{
   double dx = a.x - b.x;
   double dy = a.y - b.y;
   return sqrt(dx * dx + dy * dy);
}
```

Step 1: drawbacks of trivial

- for **any point class**, not on just this ***mypoint*** type
- in **more than two** dimensions
- for **other coordinate systems**, e.g. over the earth
- between a point and a line, or between **other geometry combinations**
- in **other (e.g. higher) precision** than double
- **avoiding** √ : often we want to avoid the square root; for comparing distances it is not necessary.

Step 2: templates

Make distance a template function

```
template <typename P1, typename P2>
inline double distance(P1 const& a, P2 const& b)
{
   double dx = a.x - b.x;
   double dy = a.y - b.y;
   return sqrt(dx * dx + dy * dy);
}
```

Drawbacks:

- nearly same drawbacks
- only progress is P1/P2

Step 3: traits (a)

Function distance: no usage of .x and .y

```
template <int D, typename P>
inline double get(P const& p)
{
   // Explained on next slide
   return traits::access<P, D>::get(p);
}


template <typename P1, typename P2>
inline double distance(P1 const& a, P2 const& b)
{
   double dx = get<0>(a) - get<0>(b);
   double dy = get<1>(a) - get<1>(b);
   return sqrt(dx * dx + dy * dy);
}
```

Step 3: traits (b)

Necessary implementation:

```
// By Boost.Geometry
namespace traits
{
   template<typename P, int D> class access {};
}

// By User
namespace traits
{
   template<> class access<mypoint, 0>
   {
        static double get(mypoint const& p) { return p.x; }
   };

   template<> class access<mypoint, 1>
   {
        static double get(mypoint const& p) { return p.y; }
   };
}
```

Step 4: dimension agnosticism (a)

- For 2, 3 or other dimensions
- Example: distance independent on #dimensions:

```
template <typename P1, typename P2, int D>
class pythagoras
{
   static double apply(P1 const& a, P2 const& b)
   {
        double d = get<D-1>(a) - get<D-1>(b);
        return d * d + pythagoras<P1, P2, D-1>::apply(a, b);
   }
};

template <typename P1, typename P2 > class pythagoras<P1, P2, 0>
{
   static double apply(P1 const&, P2 const&) { return 0; }
};
```

Step 4: dimension agnosticism (b)

Modified distance function:

```
template <typename P1, typename P2>
double distance(P1 const& a, P2 const& b)
{
   BOOST_STATIC_ASSERT((dimension<P1>::value
                  == dimension<P2>::value ));

   return sqrt(pythagoras<P1, P2, dimension<P1>::value>::apply(a, b));
}
```

Step 4: dimension agnosticism (c)

Necessary metafunction:

```
// By Boost.Geometry
namespace traits
{
   template<typename P> struct dimension {};
}


// By User
namespace traits
{
   template<> struct dimension<mypoint> : boost::mpl::int_<2> {};
}
```

After step 4:

- Done:
  - Point type agnostic, any point type
  - Concepts implemented by traits
  - Dimension agnostic

- Still todo:
  - Different geometries (point/line)
  - Coordinate type (double)
  - Coordinate systems

Step 5: tag dispatching (a)

Purpose: Support different geometry types

(e.g. point / linestring)

```
template <typename G1, typename G2>
inline double distance(G1 const& g1, G2 const& g2)
{
   return dispatch::distance
         <
                 typename tag<G1>::type,
                 typename tag<G2>::type,
                 G1, G2
         >::apply(g1, g2);
}
```

Step 5: tag dispatching (b)

Implement tags

```
// By Boost.Geometry
namespace traits
{
   template<typename G> struct tag {};
}

// By user
namespace traits
{
   template<> struct tag<mypoint> { typedef point_tag type; };
   template<> struct tag<mytrack> { typedef linestring_tag type; };
}
```

Step 5: tag dispatching (c)

Add base class and specializations for distance, to implement TD

```
namespace dispatch
{
   template<typename Tag1, typename Tag2, typename G1, typename G2>
   class distance {};

   template<typename P1, typename P2>
   class distance<point_tag, point_tag, P1, P2>
   {
         static double apply(P1 const& a,  P2 const& b)
         {
                 // call pythagoras
         }
   };

   // versions for point/segment, etc
}
```

Step 5: tag dispatching – side note

- Boost.org: *Tag dispatching is a way of using* ***function overloading*** *to dispatch based on properties of a type (…)*
- Literature: always std::advance()

```
template <class InputIterator, class Distance>
void advance(InputIterator& i, Distance n)
{
     typename iterator_traits<InputIterator>
             ::iterator_category category;
     detail::advance_dispatch(i, n, category);
}
```

- Boost.Geometry: tag dispatching by **type** not by **instance**
- Then also usable for metafunctions

After step 5:

- Supports different geometry types
- Starts looking like it is in Boost.Geometry

```
mypoint a(1,1);
other_point b(2,2);
std::cout << distance(a,b) << std::endl;

segment s1(0,0,5,3);
std::cout << distance(a, s1) << std::endl;

rgb red(255, 0, 0);
rgb orange(255, 128, 0);
std::cout << "color distance: "
    << distance(red, orange) << std::endl;
```

Step 6: coordinate types (a)

Purpose: Support different coordinate types

(e.g. double / GMP)

```
// By Boost.Geometry
namespace traits
{
   template<typename P> struct coordinate_type {};
}

// By user
namespace traits
{
   template<> struct coordinate_type<mypoint>
   {
         typedef double type;
   };
}
```

Step 6: coordinate types (b)

- “Select Most Precise” of two point types
- Adapted Pythagoras:

```
template <typename P1, typename P2, int D>
class pythagoras
{
   typedef typename select_most_precise
         <
                 typename coordinate_type<P1>::type,
                 typename coordinate_type<P2>::type
         >::type selected_type;

   static selected_type apply(P1 const& a, P2 const& b)
   {
         selected_type d = get<D-1>(a) - get<D-1>(b);
         return d * d + pythagoras<P1, P2, D-1>::apply(a, b);
   }
};
```

Step 6: coordinate types (c)

- Select_most_precise
  - int + int → int
  - int + float → float
  - float + double → double
  - GMP + double → GMP
- != promote
  - Because int + int → double

Step 6: coordinate types (d)

- After coordinate types:
  - Mixing of point types
  - Type agnostic:
    - float
    - integer
    - double
    - long double
    - UDT
    - “ttmath”
    - “GMP”
  - Double (almost) nowhere
- Sometimes (distance)
  - Integer coordinate types will go to double (sqrt)

Step 7: coordinate systems (a)

- Cartesian
- Spherical
- Geographic
- Astronomic
- …

```
// By Boost.Geometry
namespace traits { template<typename P> struct coordinate_system {}; }

// By user
namespace traits
{
   template<> struct coordinate_system<mypoint>
   { typedef cartesian type; };
}
```

Step 7: coordinate systems (b)

- Computation method (often per coordinate system)
- Called “strategy”
- Default strategy (in “strategy_distance” specializations):

```
template <typename T1, typename T2, typename P1, typename P2>
struct strategy_distance {};

template <typename P1, typename P2>
struct strategy_distance<cartesian, cartesian, P1, P2>
{
    typedef pythagoras<P1, P2> type;
};
```

Step 7: coordinate systems (c)

- Adapted distance function:

```
template <typename G1, typename G2>
inline double distance(G1 const& g1, G2 const& g2)
{
   typedef typename strategy_distance
         <
                 typename coordinate_system<G1>::type,
                 typename coordinate_system<G2>::type,
                 typename point_type<G1>::type,
                 typename point_type<G2>::type
         >::type strategy;

   return dispatch::distance
         <
                 typename tag<G1>::type,
                 typename tag<G2>::type, G1, G2, strategy
         >::apply(g1, g2, strategy());
}
```

Step 7: coordinate systems (d)

- After step 7:
- Distance function is coordinate system agnostic
- “Side effect”
- Library user:
  - Can call distance(a, b)
  - Can call distance(a, b, my_own_strategy());

Point concept

- Point Concept is explained across different slides...
- … and now finished, consisting of 5 elements.

- For each point type, specify:
  - specialization for metafunction traits::**tag**
  - specialization for metafunction traits::**coordinate_system**
  - specialization for metafunction traits::**coordinate_type**
  - specialization for metafunction traits::**dimension**
  - specialization for traits::**access**

Avoid square root

- Avoid sqrt in distance:
  - Only in cartesian distance, not in spherical distance
  - Class “cartesian_distance”
  - Return_type per strategy

Combinations
of geometries and representations

Concept checking

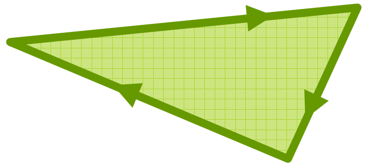
- BOOST_CONCEPT_ASSERT
- BOOST_CONCEPT_REQUIRES

- As early as possible
- But not in algorithm: dispatched, concept not known
- Within dispatch: code duplication…

- concept::check<Geometry>();
  - dispatches using geometry tag
  - and using is_const
  - compiletime only

Clockwise / counter clockwise

- area: clockwise: positive, counter clockwise: negative
- Solution: **reversible_view**
- Forward range iterator



```
for(typename boost::range_iterator<Ring>::type
        it = boost::begin(ring); it != boost::end(ring); ++it)
```

- Backward range iterator

```
for(typename boost::range_reverse_iterator<Ring>::type
        it = boost::rbegin(ring); it != boost::rend(ring); ++it)
```

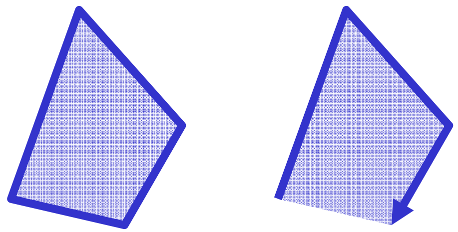
- → <Forward> or <backward>

```
typedef reversible_view<Ring, iterate_reverse> view_type; // or forward

view_type view(ring);
for(typename boost::range_iterator<range_type>::type
        it = boost::begin(view); it != boost::end(view); ++it)
```

Closed / open polygons

- Some polygons are open, some repeat first point
- Solution: **closeable_view**
- → <Closing> or <not closing>



```
typedef closeble_view<Ring, true> view_type;
// or false

view_type view(ring);
for(typename boost::range_iterator<range_type>::type
        it = boost::begin(view);
        it != boost::end(view); ++it)
```

Area

- So for area:
  - Closed / open
  - Forward / reversed
  - In two views, based on traits of polygon at hand
  - Views are, here, ranges based on another view

```
typedef closeable_view
  <
      reversible_view<Ring, Direction>,
      Closing
  > range_type;
```

Combinatorial explosion

- Algorithms with one geometry (e.g. area):
  - # geometries * #coordinate systems * 2 (cw/ccw) * 2 (op/cl)
- Algorithms with two geometries (e.g. distance):
  - # geometries * # geometries * # cs ….
- (Enable if cannot easily handle that)

- Partial specializations:
  - Tag dispatching
  - Metafunction grouping on similarity
  - Dispatch reuse as policy
  - (Views on) ranges iterating in same direction
  - Etc.
- Reversibility (next slide)

Reversibility

- Reversibility
  - Avoid separate implementations for point/segment and segment/point
  - Class “reverse_dispatch”
  - Exchanges template arguments and runtime parameters

Dispatch / Policy reuse

- Multi geometries reuse dispatches of single versions as a policy

```
template <typename MultiTag, typename MultiGeometry, typename Strategy>
class length<MultiTag, MultiGeometry, Strategy, true> : multi_sum
   <
     typename length_result<MultiGeometry>::type,
     MultiGeometry, S,
     length
          <
               typename tag<
                         typename range_value<MultiGeometry>::type
                         >::type,
               typename range_value<MultiGeometry>::type,
               Strategy, false
          >
   > {};
```

Summary

- Algorithms
- Geometries / Concepts
- Dispatching by tag
- Handle different conventions for polygon representations
- Handle combinatorial explosion

Future

- Many many possibilities
  - 3D
  - Meshes
  - Triangulation
  - Infinite lines, rays, ellipses, etc
  - Coordinate systems
  - …
- Extensions

Questions?

Thanks!

barend@geodan.nl

Geodan

Amsterdam, the Netherlands