# What's New in the JTS Topology Suite

*Martin Davis, OpenGeo*

*April 2012*

## What is JTS?

- API for representing and processing **2D linear vector Geometry**
- Implemented in Java; licensed under LGPL
- Provides the full OGC **Simple Features for SQL** geometry specification:
  - Points, Linestring, Polygons, collections
  - **Metrics:** Length, Area, Distance
  - **Predicates:** intersects, contains, etc.; relate for DE-9IM
  - **Overlay:** intersection, union, difference, symDifference
  - **Algorithms:** Convex Hull, Buffer
- Other features:
  - Validation, Polygonization, Simplification, Linear Referencing, etc.

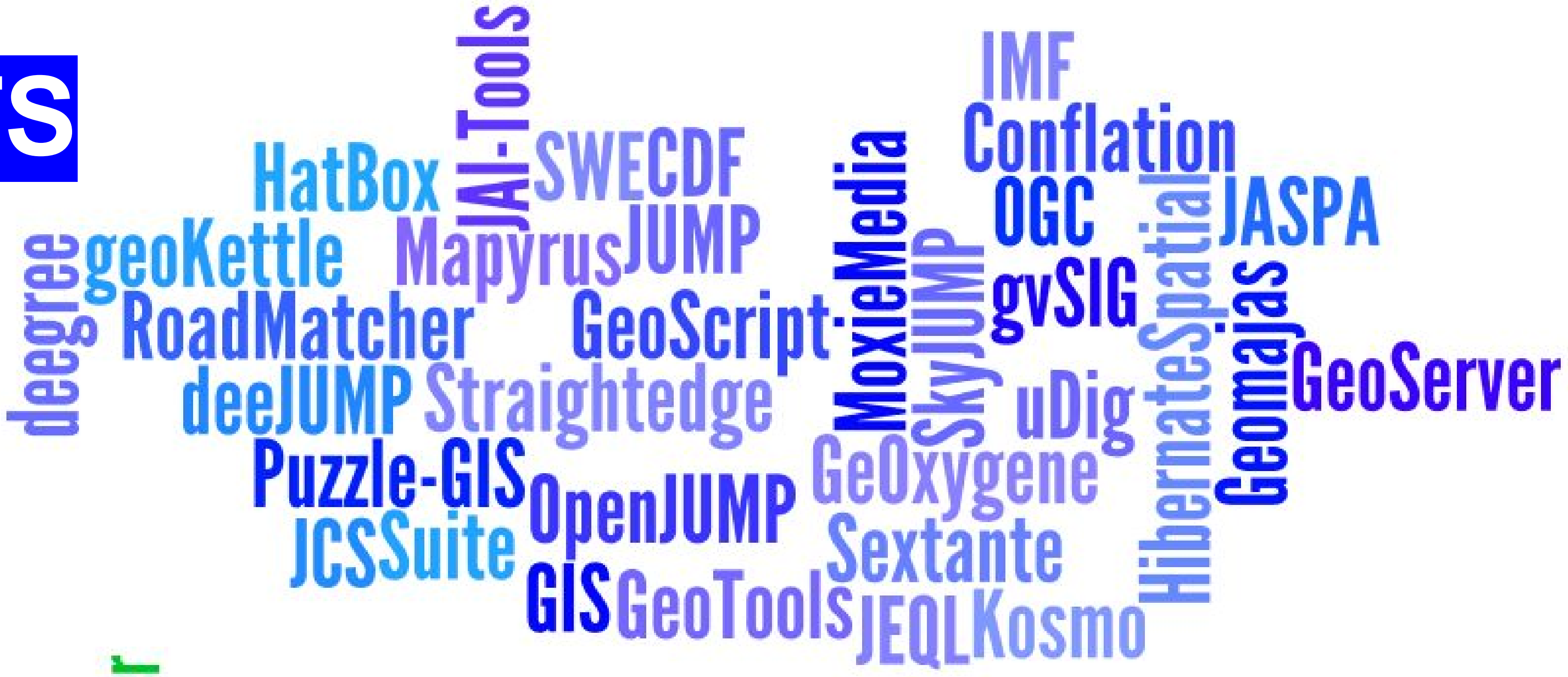
## Project History

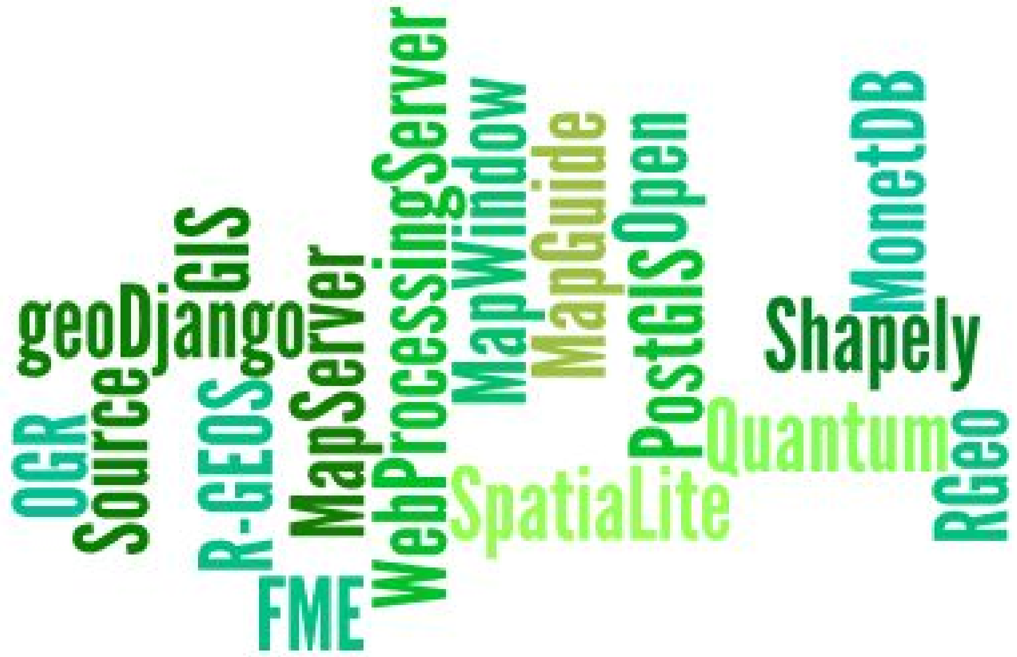
- **Version 1.0** - May 2001

  ...
- **Version 1.9** - January 2008
- **Version 1.10** - December 2008
- **Version 1.11** - March 2010
- **Version 1.12** - June 2011
- **Version 1.13** - *Coming Soon!*

## Where is it used ? (1)

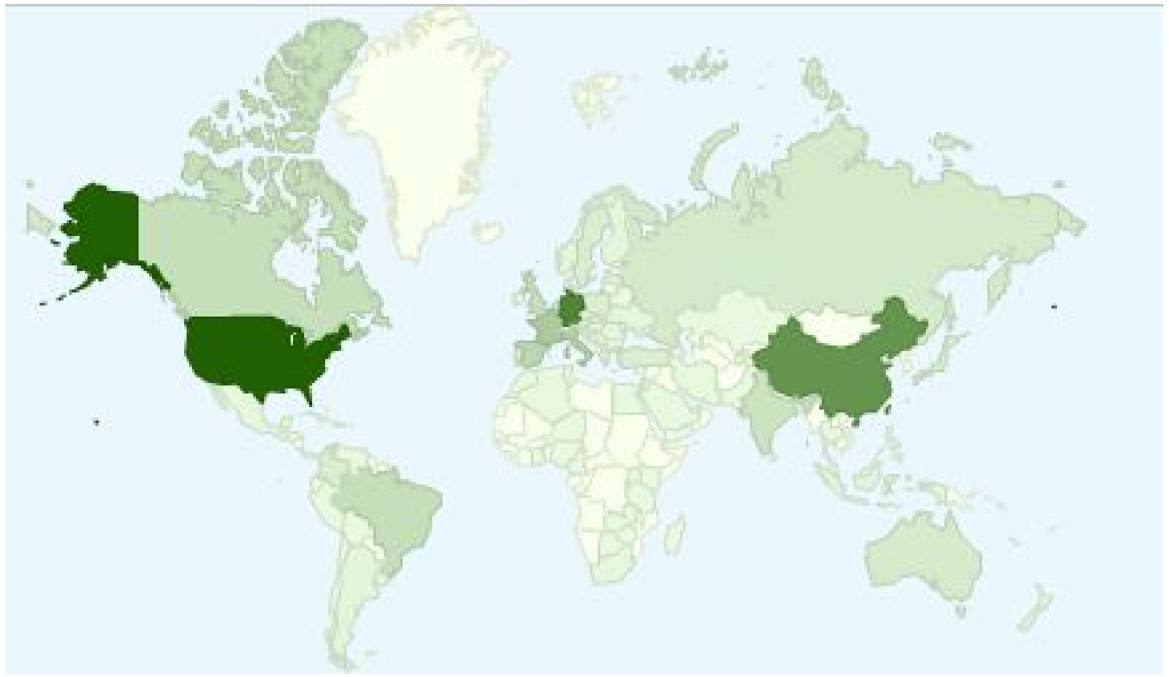
**JTS**





**GEOS**

## Where is it used ? (2)

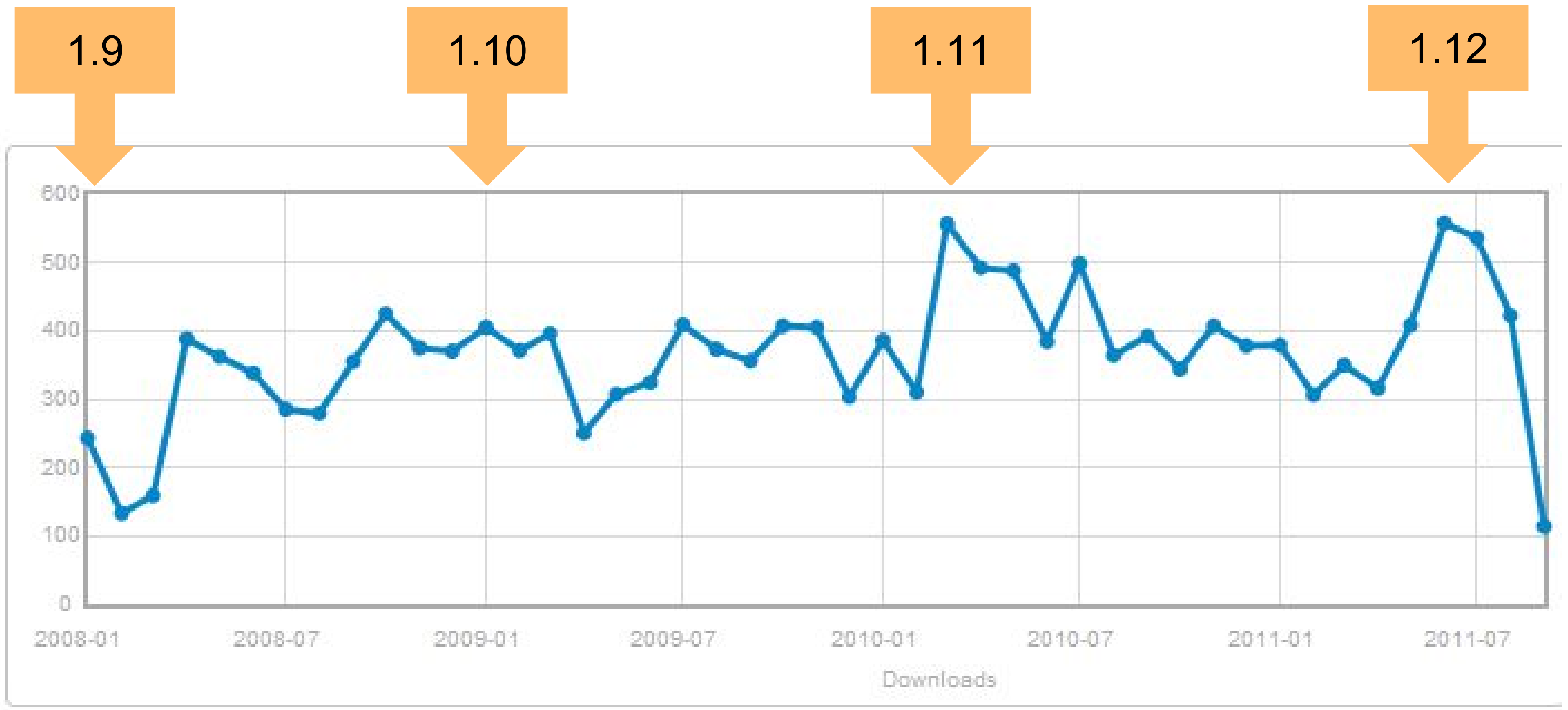


*Downloads [ Jan. 1, 2008 - Sept. 8 2011 ]*

1. United States -- 1,384
2. Germany -- 1,051
3. China -- 915
4. France -- 424
5. Italy -- 375

# Project Statistics

Total downloads [ Jan 2008 - Sept 2011 ] : **16,405**



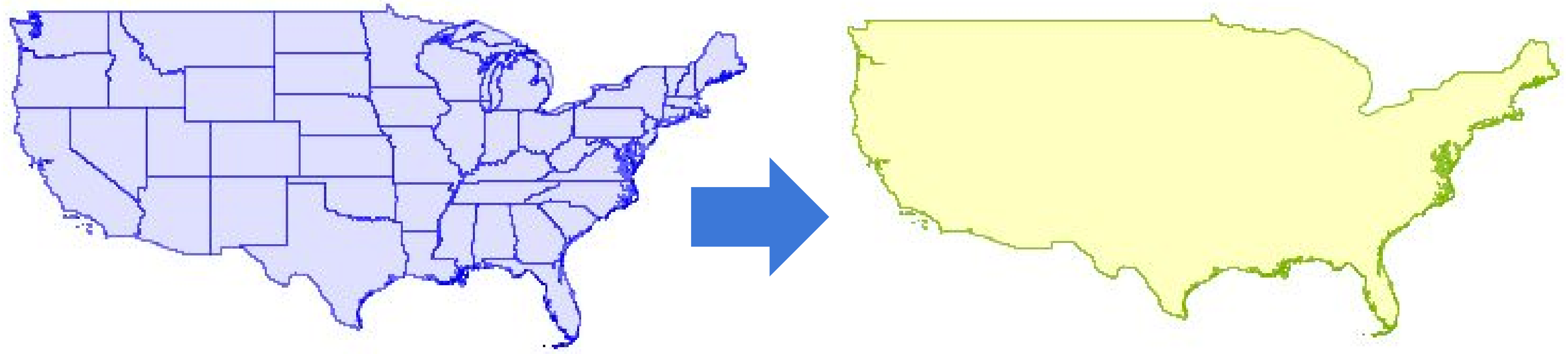
# JTS in other languages

- **Ports**
  - ***GEOS*** ---> C++
  - ***Net Topology Suite*** ---> C#
  - ***JSTS*** ---> JavaScript
- **Bindings *(on JVM)***
  - Groovy, Scala, Jython, JRuby, Clojure, etc
- **Bindings *(to GEOS)***
  - Shapely ( Python )
  - RGeo ( Ruby )
  - R-GEOS ( R )

## What's New in **JTS**

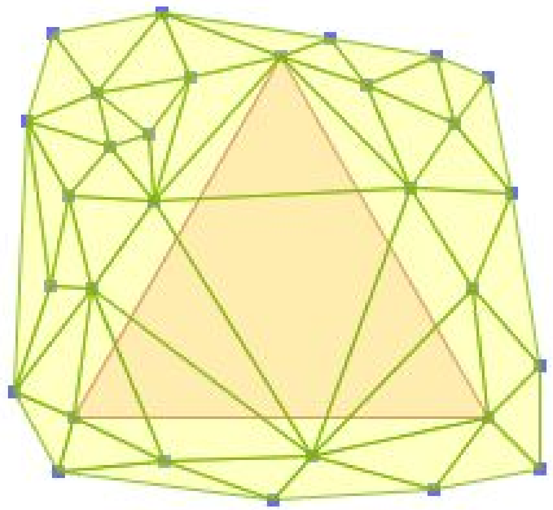
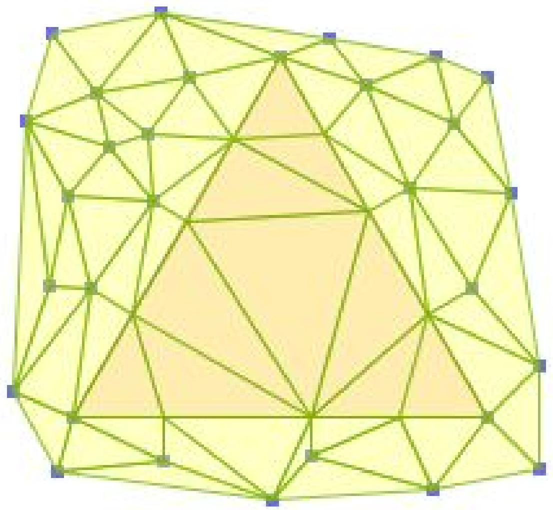
# Unary Union

- `Geometry.union()`
  - High-performance union of geometry collections
  - Uses spatial index to optimize union
  - In most situations much more efficient than iterating `Geometry.union(Geometry)`
  - handles heterogeneous `GeometryCollection`s



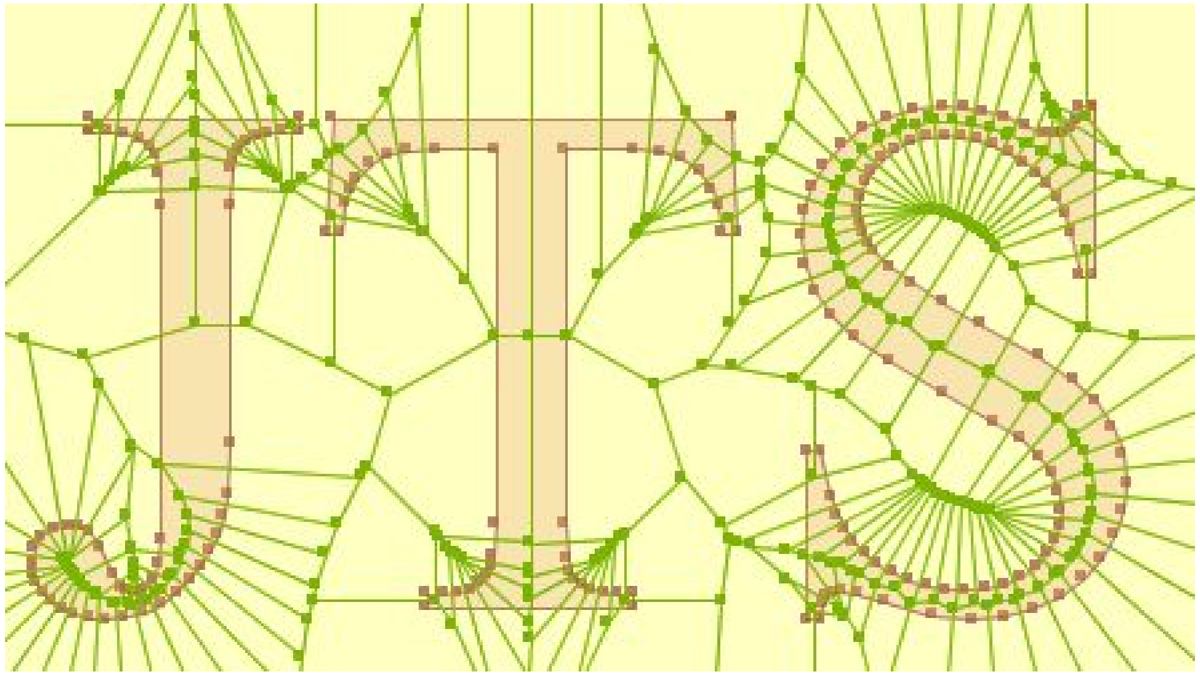
# Delaunay Triangulation

- `DelaunayTriangulationBuilder`
  - Optimal triangulation of point sets
  - Efficient, robust algorithm
  - Uses `QuadEdge` data structure
- `ConformingDelaunayTriangulationBuilder`
  - Delaunay triangulation with linear constraints
  - approximates constraints by adding vertices along segments

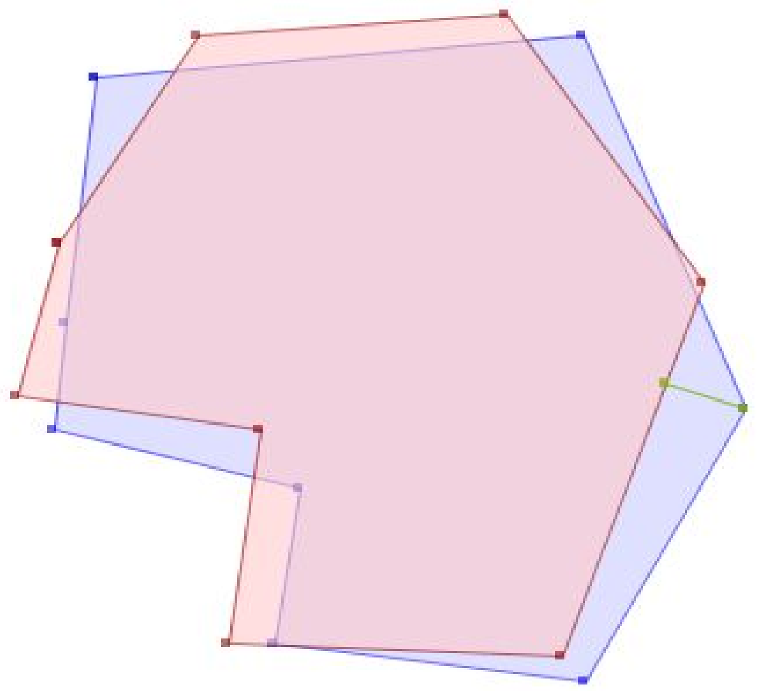
# Voronoi Diagram

- Dual of **Delaunay Triangulation**
- Voronoi & Delaunay scale to millions of points



# Hausdorff Distance

- `DiscreteHausdorffDistance` **distance metric**
  - "How far apart are two geometries"
  - useful for QA, geometry matching (conflation)
  - true Hausdorff distance is difficult/slow to compute, so uses faster discrete version

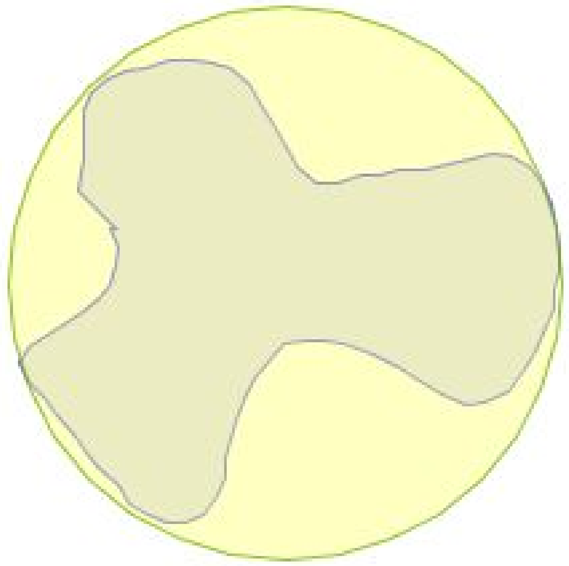
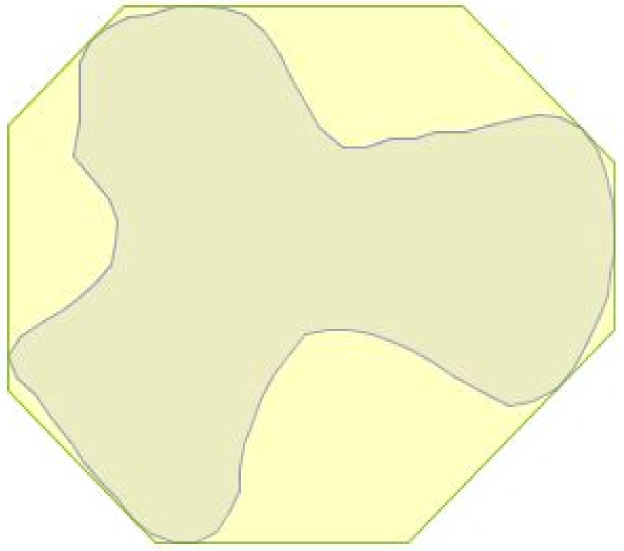
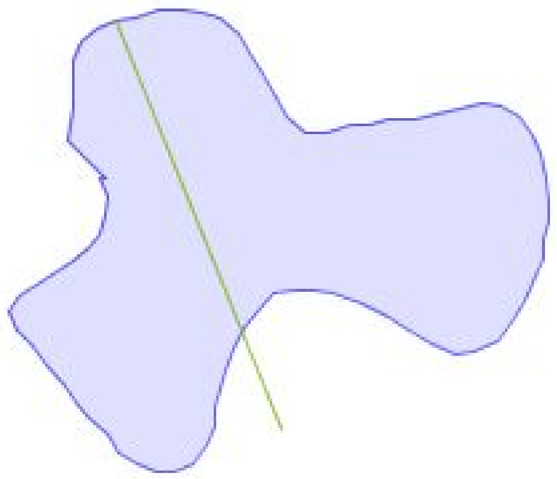
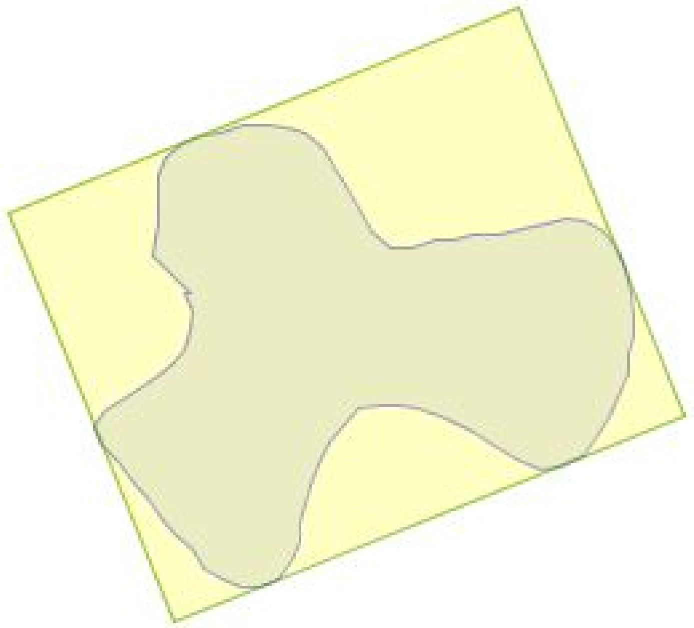


*Euclidean distance = 0*

***Hausdorff distance = 18.23***

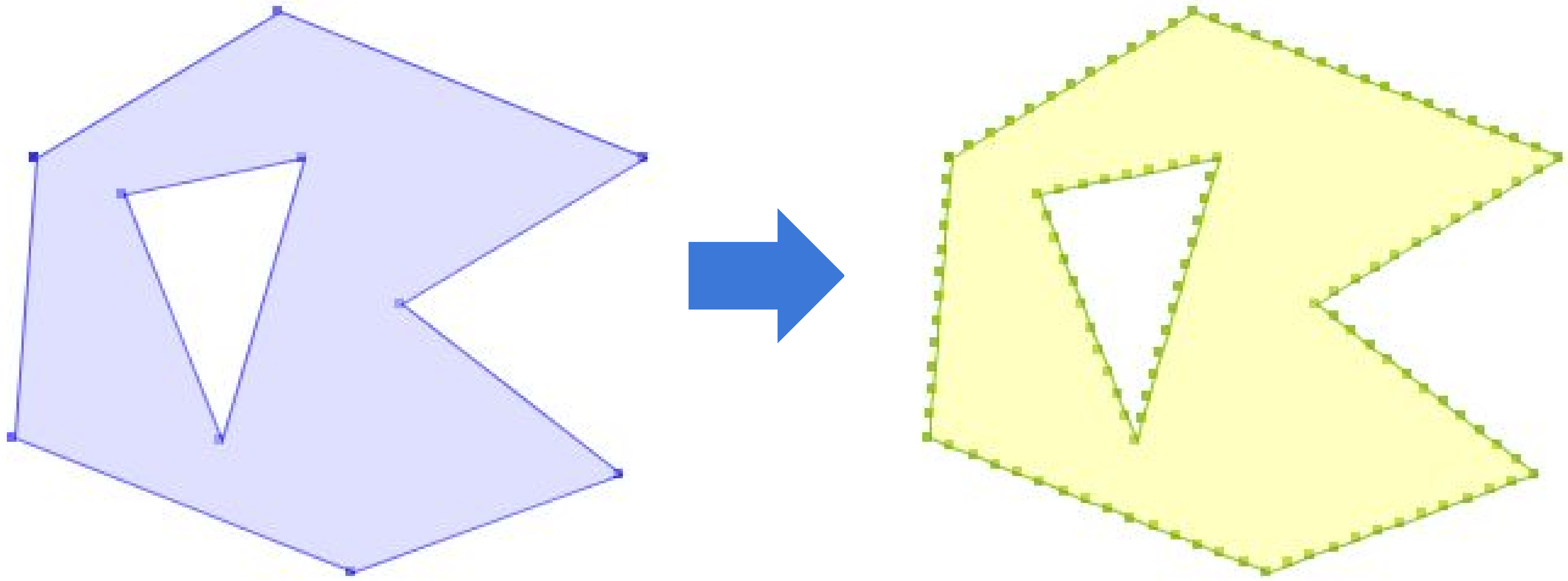
# Bounding Containers

- `MinimumBoundingCircle`
- `OctagonalEnvelope`
- `MinimumDiameter`
  - *also* **Minimum Rectangle**

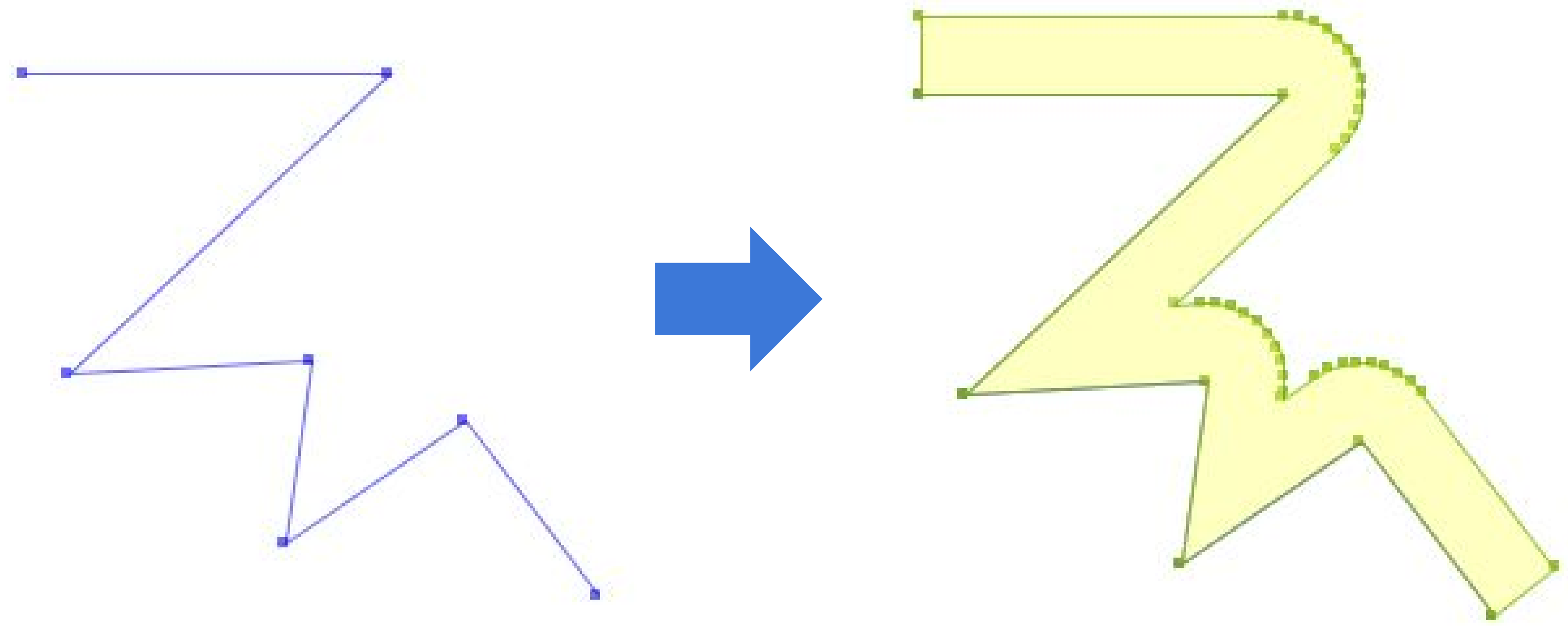
## Densification

- `Densifier`
  - specify maximum length of segments
  - ensures result has valid topology



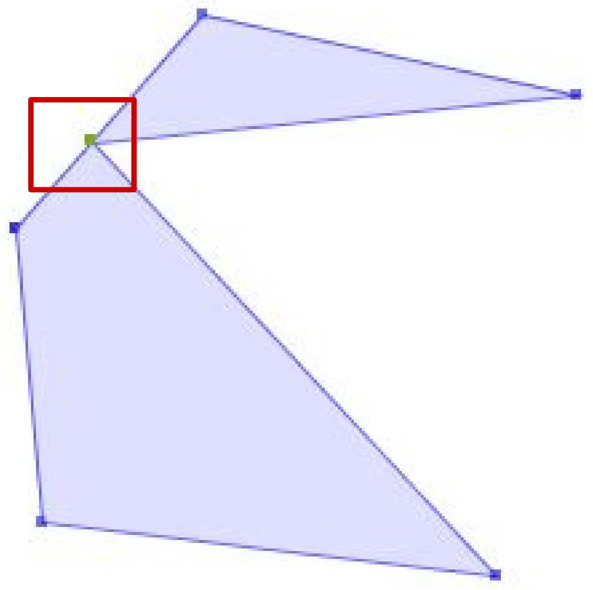
## Single-Sided Buffers

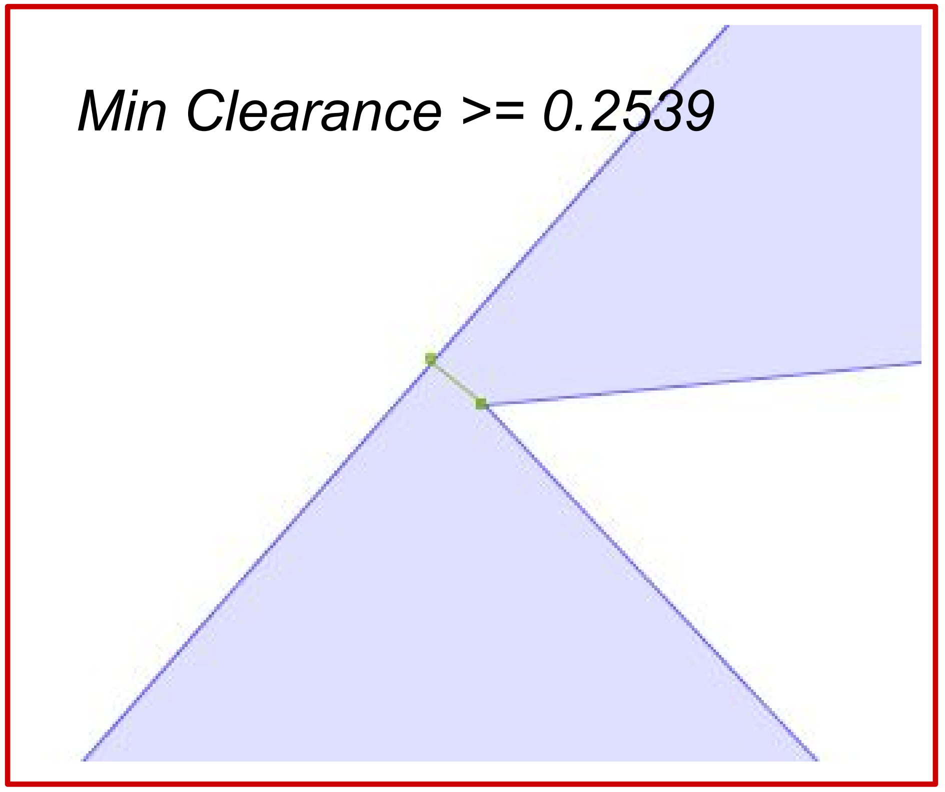
- Invoke by `BufferParameters.setSingleSided()`
    - Sign of distance determines side
- *Some warnings apply!*



# Minimum Clearance

- Determines if Precision Reduction might product invalid result
- Uses `STRtree` Nearest Neighbour for efficient computation



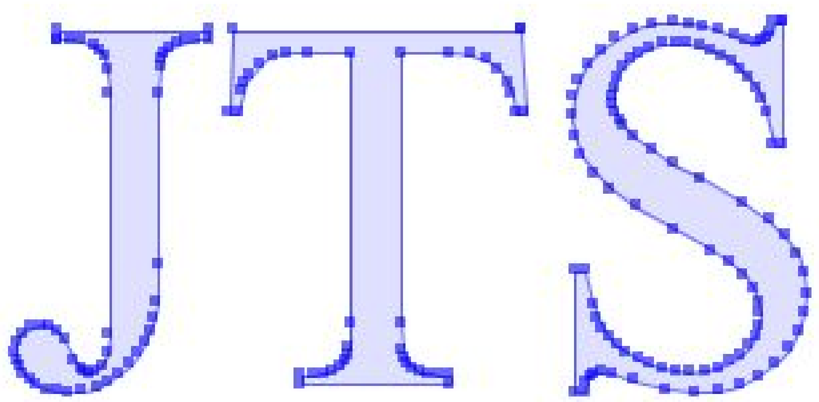


# Nearest Neighbour

- **Nearest Neighbour**
  - between an object and a set
  - within a set
  - between two sets
- **implemented via** `STRtree` **index**
  - efficient search
  - user-definable distance metric
- **Uses**
  - MinimumClearance
  - Fast distance calculation

## Java2D utilities

- `ShapeReader`
  - converts `java.awt.Shape` to `Geometry`
- `ShapeWriter`
  - converts `Geometry` to `java.awt.Shape`
  - provides `PointTransformation` to map coordinates
  - supports **decimation** for faster rendering
- `FontGlyphReader`
  - converts `Font` text to a `Polygon` geometry



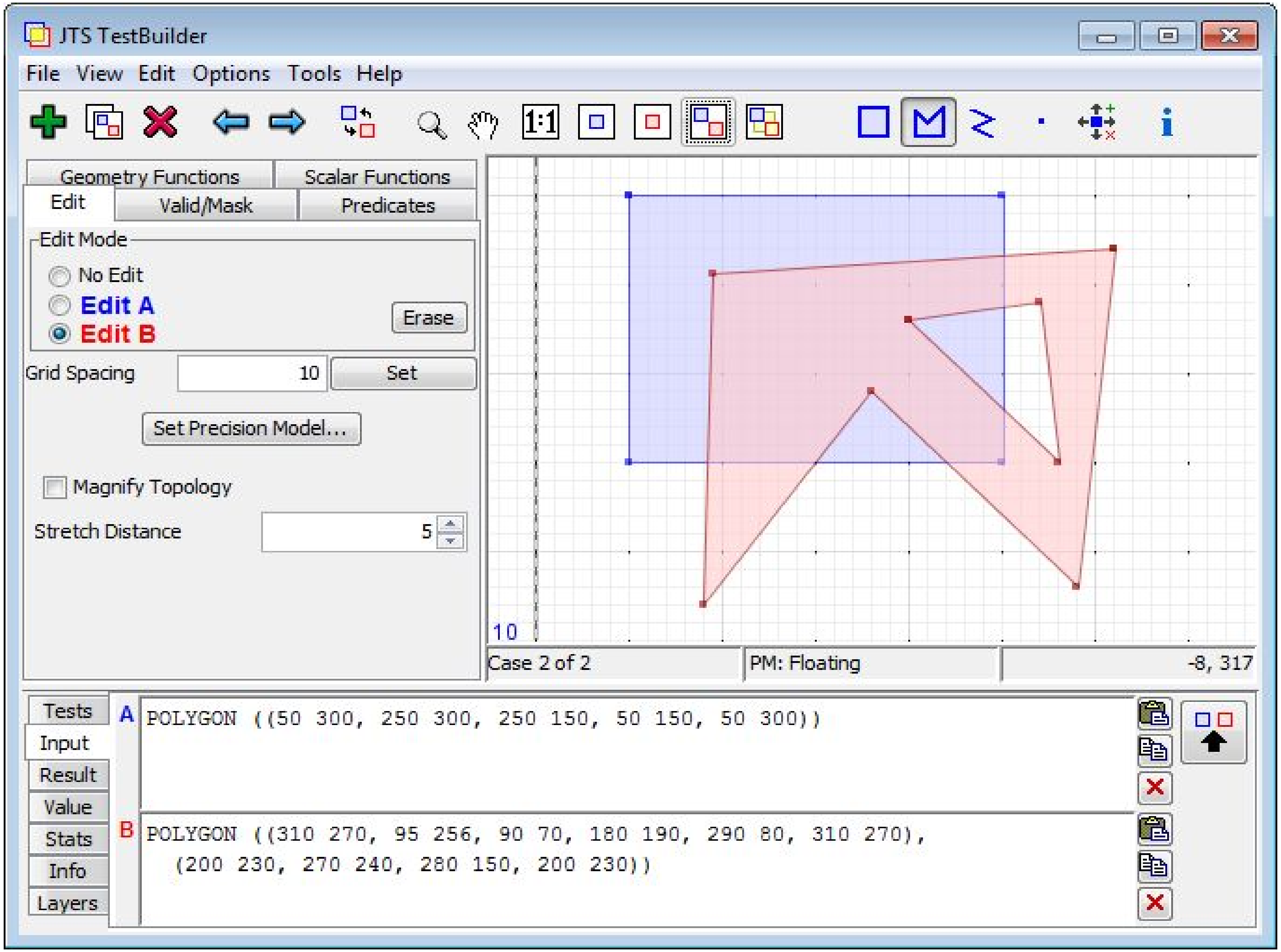
## Mathematics utilities

- `Vector2D`
  - vector structure & operations

- `DD` - DoubleDouble
  - higher-precision floating-point arithmetic
  - 106 bits of precision
  - provides robust computation of:
    - **inCircle** test for Delaunay triangulation
    - triangle **area** & **orientation**

```
public static DD triAreaDDFast(
     Coordinate a, Coordinate b, Coordinate c) {
  DD t1 = DD.valueOf(b.x).selfSubtract(a.x)
        .selfMultiply(DD.valueOf(c.y).selfSubtract(a.y));
  DD t2 = DD.valueOf(b.y).selfSubtract(a.y)
        .selfMultiply(DD.valueOf(c.x).selfSubtract(a.x));
  return t1.selfSubtract(t2);
}
```

# What's New in **TestBuilder**



Grid Spacing: 10

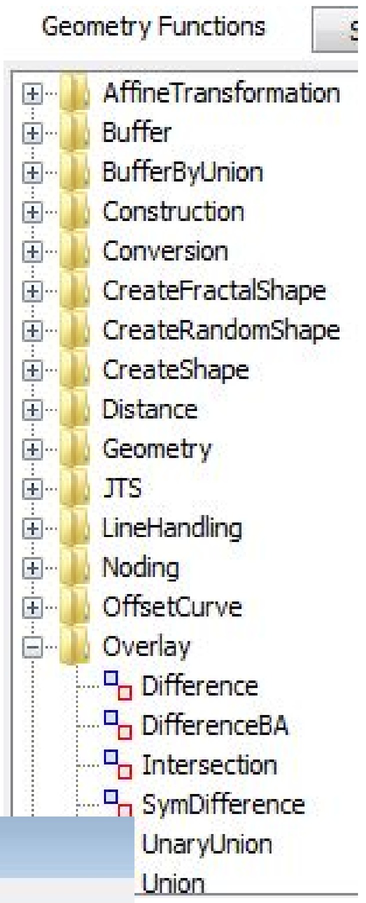
Stretch Distance: 5

Case 2 of 2 | PM: Floating | -8, 317

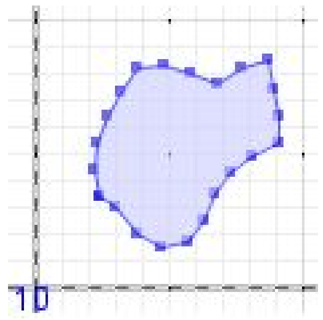
```
A POLYGON ((50 300, 250 300, 250 150, 50 150, 50 300))

B POLYGON ((310 270, 95 256, 90 70, 180 190, 290 80, 310 270),
    (200 230, 270 240, 280 150, 200 230))
```

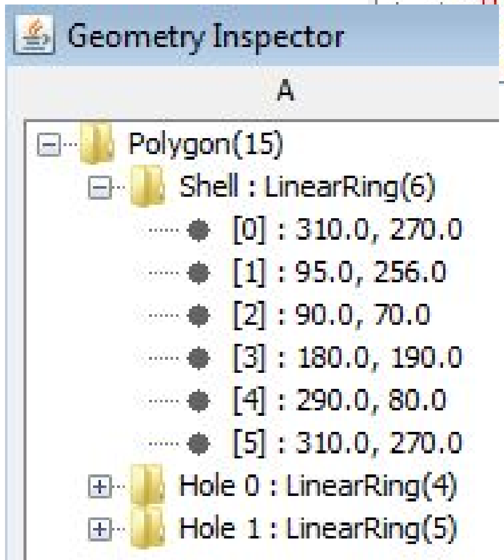
# What's New in **TestBuilder**

- **User-Defined Functions**
  - via Java `public static` methods
- **Many new functions**
- **Dynamic digitizing grid**
- **Stream digitizing**
- **Drag-and-drop data load**
  - WKT, XML tests, Shapefile
- **Threading**
  - Function execution
  - Rendering
- **Display function run time**
- **Geometry Inspector**


Geometry Functions

- AffineTransformation
- Buffer
- BufferByUnion
- Construction
- Conversion
- CreateFractalShape
- CreateRandomShape
- CreateShape
- Distance
- Geometry
- JTS
- LineHandling
- Noding
- OffsetCurve
- Overlay
  - Difference
  - DifferenceBA
  - Intersection
  - SymDifference
  - UnaryUnion
  - Union



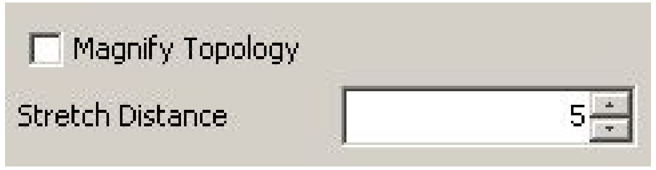

Geometry Inspector

A

- Polygon(15)
  - Shell : LinearRing(6)
    - [0] : 310.0, 270.0
    - [1] : 95.0, 256.0
    - [2] : 90.0, 70.0
    - [3] : 180.0, 190.0
    - [4] : 290.0, 80.0
    - [5] : 310.0, 270.0
  - Hole 0 : LinearRing(4)
  - Hole 1 : LinearRing(5)

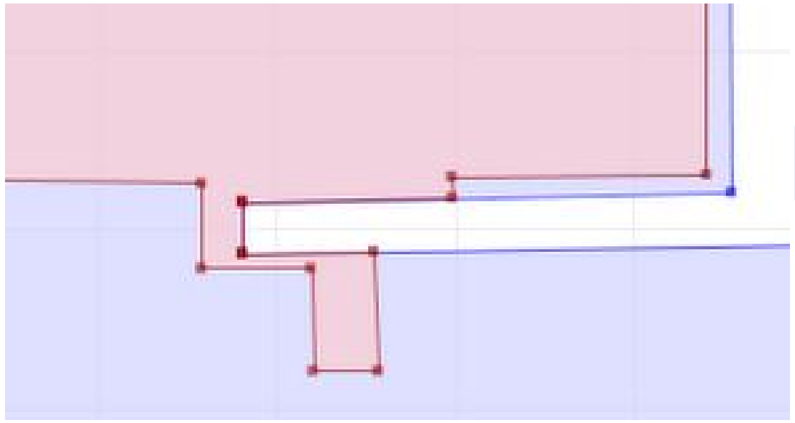
# What's New in the **TestBuilder**

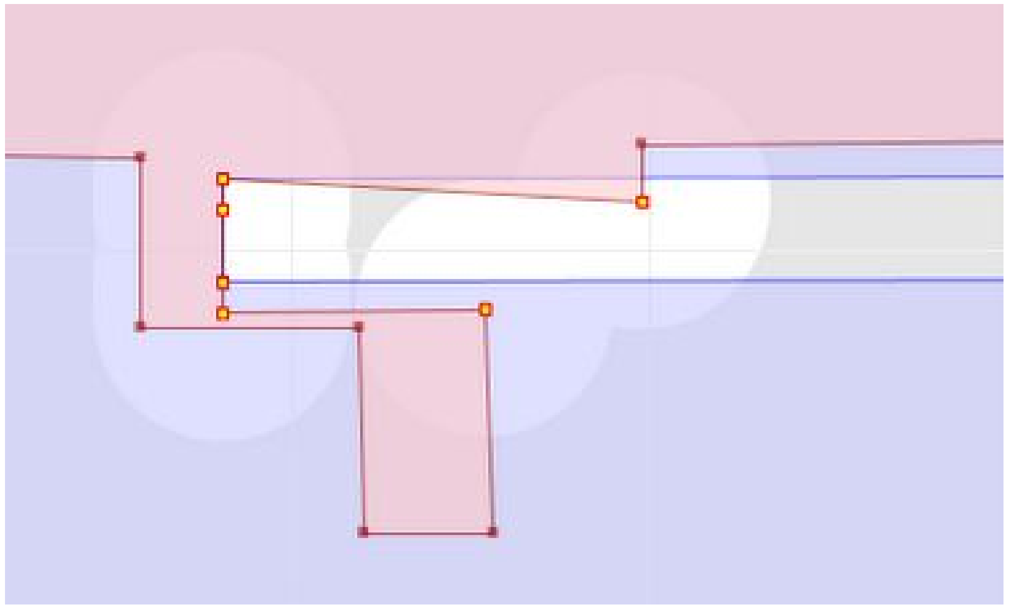
- **Magnify Topology**
  - Visualize very small geometry & topology discrepancies



☐ Magnify Topology

Stretch Distance 5





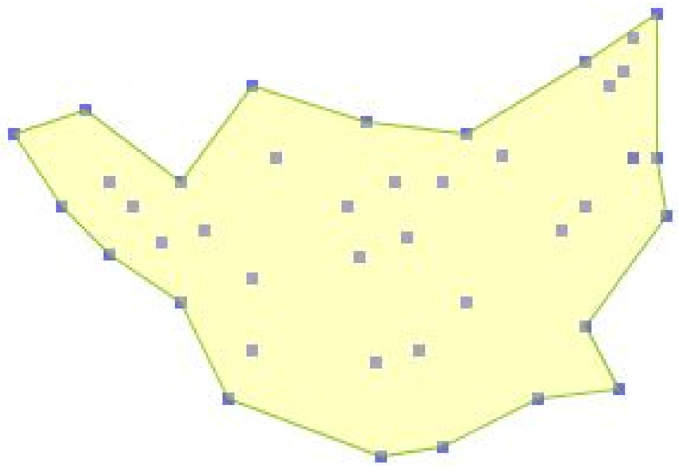
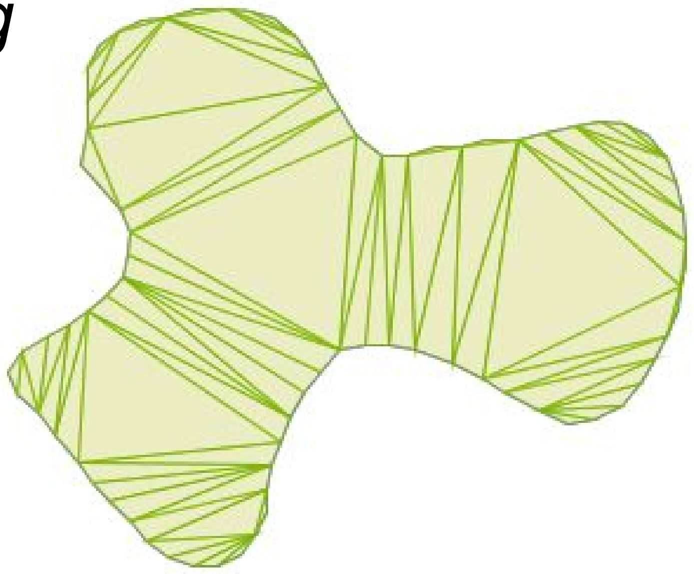
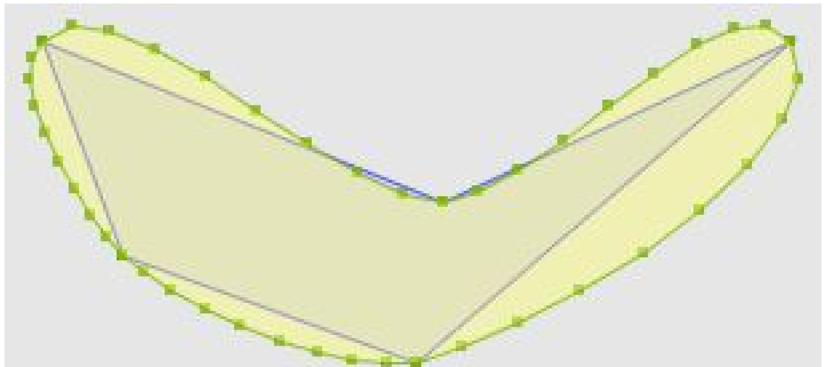
## What's New in the **TestRunner**

- **Custom operations**
  - Implement as Java code, configure in test file or cmd line
  - Uses:
    - Experiment with different algorithms
    - Re-use test corpus with different operations
    - Compare JTS results with external code
- **Custom Result Matching strategies**
  - use for operations which produce approximate results
  - e.g. `buffer()`
- **Ability to run single Test Case out of a set**

# In the Lab

- **Performance improvements**
  - **Buffer** (again!)
  - Fast **Distance** computation

- **New algorithms:**
  - **Concave Hull**
  - **Point Clustering (***e.g. K-means)*
  - **Polygon triangulation** (*Ear Clipping with Delaunay improvement*)
  - **Orthogonalization**
  - **Bezier Smoothing**

# Future Plans

- **Computation in Geodetic coordinate systems**
  - Area, Distance first
  - Other operations  ...somehow
- **Support measures on coordinates**
- **Improve performance, robustness**
  - Constant quest...
- **Split packaging into Core and Algorithms**
- **Refactor `Geometry` classes to use interfaces**
  - allows alternate geometry representations
  - => *JTS 2.0*

## Distribution & Support

- **JTS available from SourceForge**

`http://sourceforge.net/projects/jts-topo-suite/`

- **Mailing List**

`https://lists.sourceforge.net/lists/listinfo/jts-topo-suite-user`

- **Other JTS resources**
  - Javadoc
  - References
  - FAQ
  - more to come...

**`http://tsusiatsoftware.net/jts/main.html`**