Secrets of the JTS Topology Suite

Martin Davis

Refractions Research Inc.

Overview of presentation

- Survey of JTS functions and components
- Tips for using JTS as an engine for processing Geometry
- Tips for using JTS components and APIs for spatial algorithm development
- Future Enhancements

Overview of JTS

- Java API for modeling & manipulating *planar linear vector geometry*
  - License: LGPL
- Development History
  - Version 1.0 - May 2001
  - Version 1.8 - December 2006
  - *Version 1.9 – Q4 2007*
- Clients:
  - JUMP, GeoTools (uDig, GeoServer), eXist, etc.
  - (*as GEOS*) PostGIS, FME, OGR, MapServer, MapGuide Open Source, etc.
  - (*as NTS*) monoGIS, SharpMap, etc?

JTS as a Geometry Engine

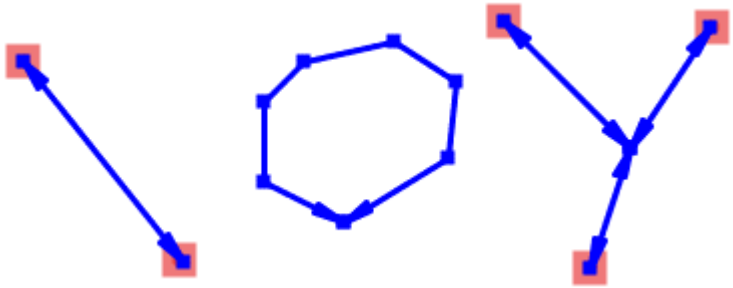
- Geometry types
  - `Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, GeometryCollection`
- Geometry methods
  - Spatial Predicates, `relate()`
  - Overlay ops, `buffer(), convexHull()`
  - Metrics: `area(), length()`
  - `distance(), withinDistance()`
- Geometry Processing
  - Line Merging
  - Noding & Polygonization
  - Simplification
  - Linear Referencing

Geometry Operation Classes

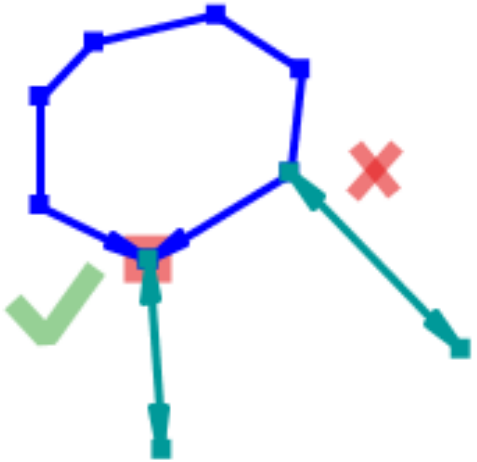
- Most non-trivial Geometry methods are implemented as classes
- Often classes provides extra functionality
- Examples:
  - `DistanceOp` can return two closest points
  - `IsSimpleOp` can return location of non-simplicity
  - `IsValid` gives option to check for SDE-style polygon topology
  - `RelateOp` allows `BoundaryNodeRule` to be specified

Spatial Relationships & Boundary Node Rules

- How boundary points of linear geometries are determined
  - OGC-SFS specifies “Mod-2” Rule



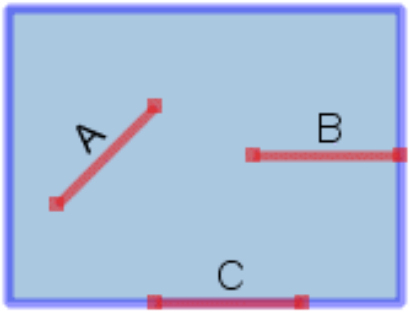
- Other rules sometimes useful
  - All Endpoints
    - Ex: Do roads touch only at nodes?
  - Monovalent Endpoints
  - Multivalent Endpoints



- `RelateOp` class allows specifying rule to use

Additional Spatial Predicates

- OGC-SFS spatial predicates have some subtle behaviour
- `contains()` : Polygons do not “contain” their boundary!
  - A & B : `contains()==true`
  - C : `contains()==`**`false`**



- JTS provides `covers()` and `coveredBy()`, which treat boundary and interior identically
- Side benefit – easier to optimize
  - e.g. <rectangle>.covers()

Optimized Spatial Predicates

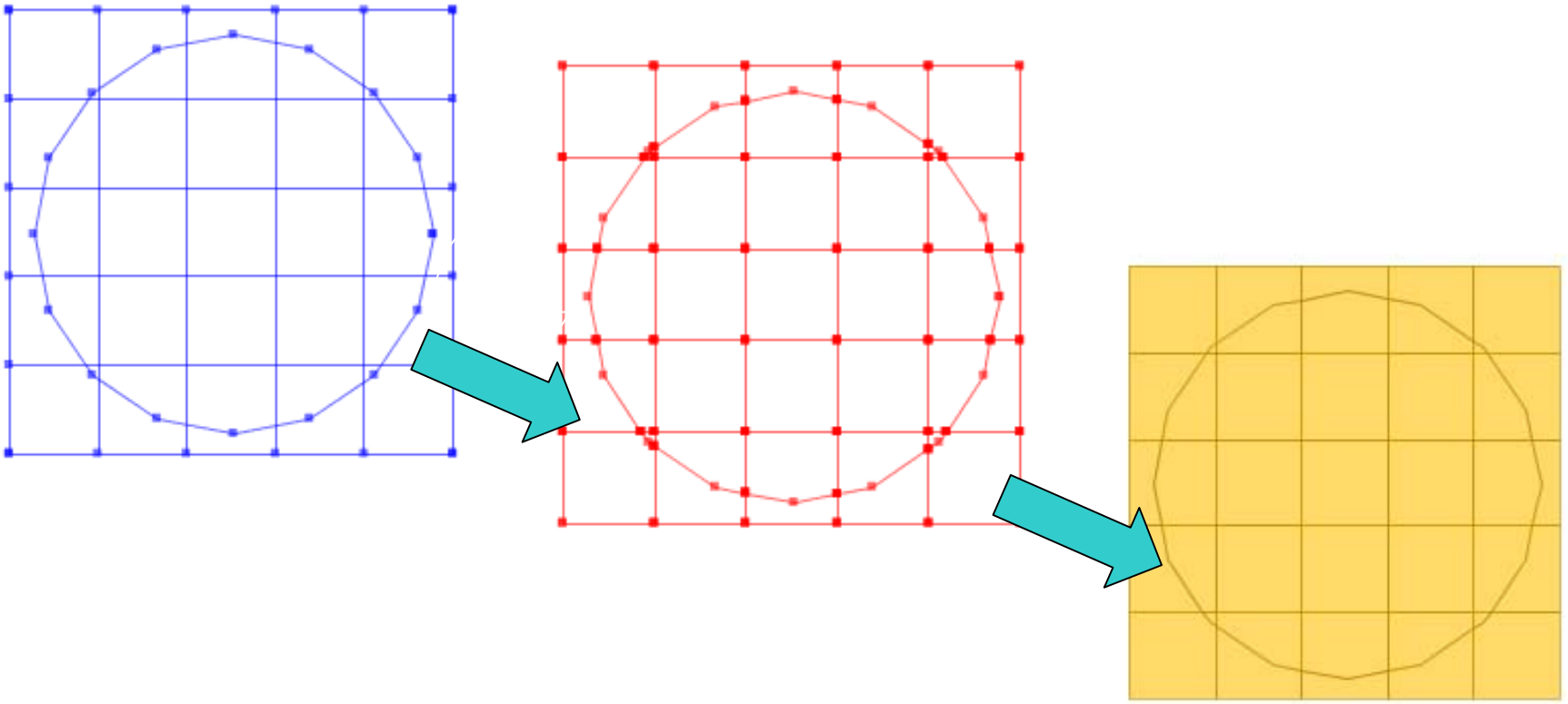
- Spatial query / join is common use pattern
  - i.e. repeated predicate operation on same geometry
- `PreparedGeometry` improves performance
  - Uses caching, algorithm optimizations
  - Over **100x** faster in some cases!

```
PreparedGeometry targetPrep
        = PreparedGeometryFactory.prepare(targetGeom);
for ( <geometries to test> ) {
  Geometry test = ...
  if (targetPrep.intersects(test)) {
    ...
```

- Currently provides most important predicates
  - `intersects`, `contains`, `covers`
- *New in JTS 1.9*

LineString Noding, Polygonization

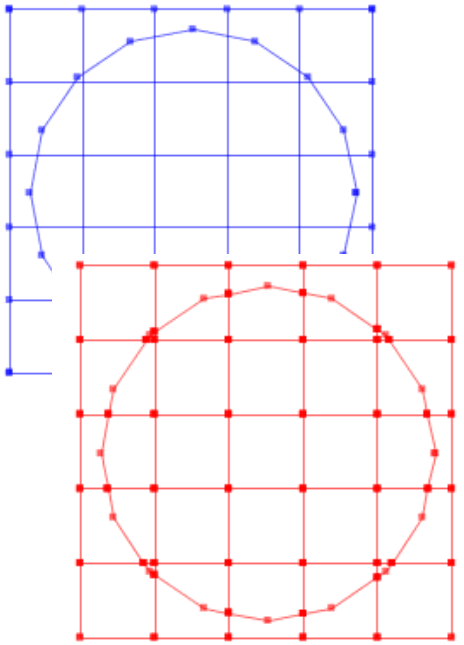
- Problem: Node & Dissolve a set of LineStrings, then Polygonize



LineString Noding, Polygonization *cont'd*

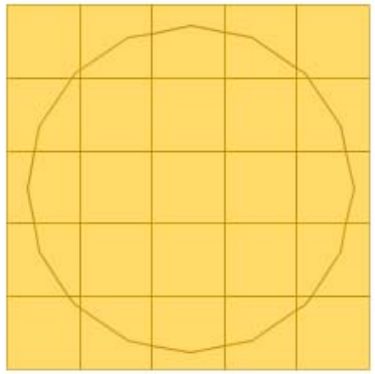
- Trick: to node & dissolve, combine LineStrings into a `MultiLineString`, then union them with a `Point` from one of the lines

```
Collection lines = ...
Geometry mls = geomFactory.buildGeometry(lines);
Point mlsPt = geomFactory.createPoint(mls.getCoordinate());
Geometry nodedLines = mls.union(pt);
```



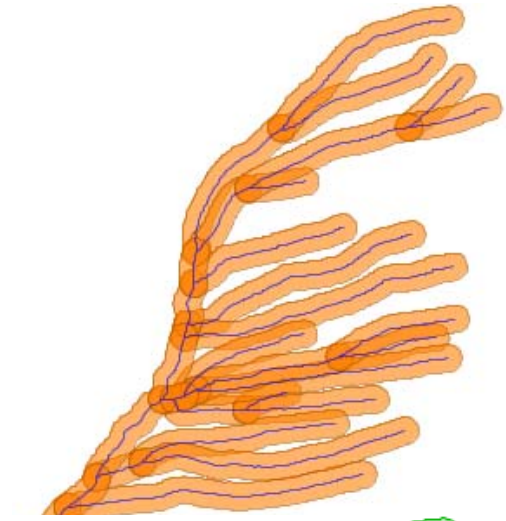
- Noded lines can be polygonized using the `Polygonizer` class

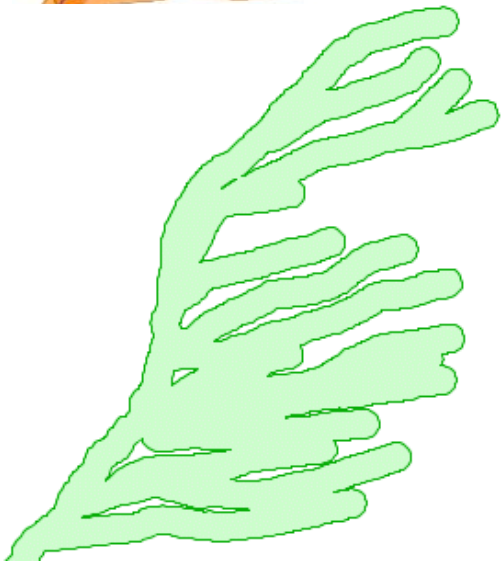
- *New in JTS 1.9:* `Geometry.union()`



Polygon Union using buffer(0)

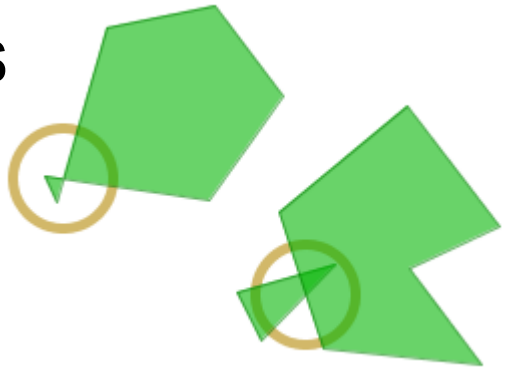
- Merging a large set of Polygons using repeated `polyUnion.union(poly)` can be slow
- Trick: combine Polygons into a `GeometryCollection`, then compute `gc.buffer(0.0)`
- *Warning - doesn't work for non-polygonal features!*
- *New in JTS 1.9:* `Geometry.union(`

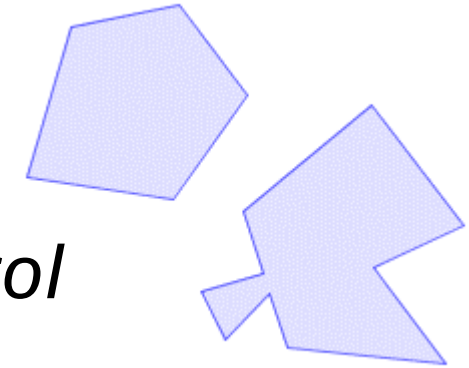




Polygon Cleaning using buffer(0)

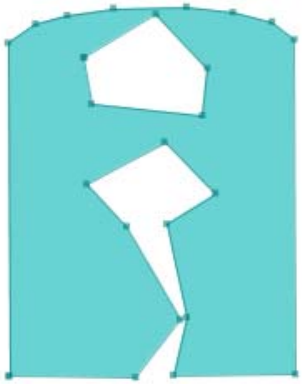
- Polygons from external data sources can be invalid because of self-intersections or overlaps
- Trick: computing `buffer(0.0)` removes pinch-offs, merges overlapping polygons

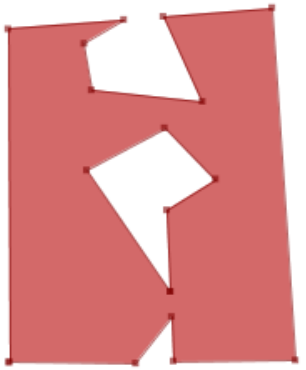




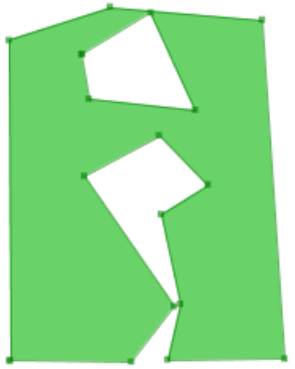
- *It would be nice to have more control over cleaning behaviour!*
- *Future:* `PolygonRectifier`

Geometry Simplification





- Two types
  - `DouglasPeuckerSimplifier`
    - standard Douglas-Peucker
    - Faster, but does not preserve topology
  - `TopologyPreservingSimplifier`
    - Slower, but preserves topology (lines will not cross, holes are preserved)



- Not `Geometry` methods – use classes directly

JTS as an algorithm library

- JTS contains many algorithms and components for building spatial algorithms & processes
  - Fast Point-in-Polygon
  - Robust Line-Point orientation, Ring orientation
  - Line segment intersection detection & computation
  - Spatial indexes (and `MonotoneChain`)
  - Indexed Noding & Intersection detection for line arrangements
  - `PlanarGraph` package
  - Primitive Geometric objects & methods
    - `LineSegment`, `Triangle`, `Angle`

Fast Point-In-Polygon

- Common use case is repeated P-I-P queries against a fixed polygon
- This case can be optimized by using spatial indexing
- Options:
  - As component: `IndexedPointInAreaLocator`
    - Result in {INTERIOR, BOUNDARY, EXTERIOR}
  - Also `PreparedGeometry.intersects(), contains()`
- Uses incremental `RayCrossingCounter` – easy to use over custom Ring data structures

Spatial Indexes

- Several types of spatial index available
  - 2-D: `QuadTree`, `STRtree`
  - 1-D: `Bintree`, `SortedPackedIntervalRTree`
- Used in many internal JTS operations to improve performance
  - Line noding
  - Line segment intersection detection
  - Point-in-polygon
- Often useful for improving performance of "naive" spatial algorithms
  - In theory takes $O(n^2)$ into $O(n \log n)$!

Spatial Indexes – STRtree VS QuadTree

- STRtree
  - Packed R-Tree
  - Cannot be modified once built (no insert or delete)
  - Fastest performance
- QuadTree
  - Slower performance (but still good!)
  - Supports insert & delete
  - Useful for “online” algorithms

Future Enhancements

- Polygon Fixing/Cleaning
- `PreparedGeometry.intersection()`
- Rectangle clipping (intersection)
- Buffer enhancements: variable-width, single-sided, offset lines
- Geometry Smoothing, Densification
- Measures support for Linear Referencing
- `Geometry.cut(Geometry)`
- Topology API
- Interface-based `Geometry` model
  - Easier to use JTS over other geometry implementations
  - `Coordinate` interface too!

Downloads & Information

- Download JTS

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

- JTS Mailing List

http://lists.jump-project.org/mailman/listinfo/jts-devel