Recent Developments in JTS and GEOS

Martin Davis

CUGOS Spring Fling 2023

April 2023
Martin Davis

- Geospatial Engineer at crunchydata

- Developer on:
  - JTS Topology Suite
  - GEOS
  - PostGIS
  - pg_featureserv

I ❤️ Math & Geometry!
JTS Topology Suite

- Library for representing and processing vector geometry
- Written in Java
- Since 2001; now at version 1.19
- Open source, on GitHub
- License
  - EPL: Eclipse Public License
  - EDL: Eclipse Distribution License (BSD-style)
- Widely used in Java spatial applications
GEOS Geometry Library

- JTS port to C++ with a C API
- Open source, on GitHub
- License: GPL (GNU Public License)
- VERY widely used

**Language Bindings**
- Shapely (Python)
- R-sf
- GeoPHP
- GoGEOS
- Node-geos (Javascript)
- rgeos (Rust)

**Databases**
- PostGIS
- SpatialLite
- CockroachDB
- DuckDB
- MonetDB

**Applications**
- QGIS
- GDAL
- MapServer
- GRASS
Functionality Overview

● 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
  ○ **Constructions:** Convex Hull, Buffer

● Other functions:
  ○ Validation, Polygonization, Simplification, Linear Referencing, Delaunay/Voronoi…
Circles
Maximum Inscribed Circle

- Largest circle inside a polygon
  - Furthest point from polygon boundary
- Iterative approximation - uses an accuracy distance tolerance

```
MaximumInscribedCircle( geom, accuracy );
```
Largest Empty Circle

- Largest circle containing no obstacles (lines / points)
  - Furthest point from obstacles
- Optional: constrain center to a boundary polygon

```c
LargestEmptyCircle( geom, [ boundary ], accuracy );
```
Hulls
Convex Hull

- The *unique* convex polygon containing input vertices
- As per the Simple Features specification
- Works for all geometry types

```c
ConvexHull( geom );
```
Concave Hull - Points

- A *(possibly)* concave polygon containing input vertices
- Many possible hulls, determined by param \( \text{pctconvex} \)

\[
\text{ConcaveHull( geom, pctconvex );}
\]

\[
pctconvex = 1.0 \quad 0.6 \quad 0.4 \quad 0.0
\]
Concave Hull - Points: How it works

- Build Delaunay Triangulation on points
- Sort triangles by longest edge length
- Remove triangles, until tolerance is reached

\[ \text{Pctconvex} = 0.6 \quad 0.4 \quad 0.0 \]
Concave Hull - Points, allowing holes

- Concave hull can contain holes
  - via optional parameter `allow_holes = true`

```cpp
ConcaveHull( geom, pctconvex, true );
```

```
pctconvex   = 0.6  = 0.5  = 0.25  = 0.0
```
Concave Hull - Polygons?

- Standard Concave Hull algorithms only support points
- **Problem!** Does not respect polygon boundaries
Concave Hull - Polygons

- New algorithm to compute Concave Hull for polygon(s)
  - constrained by polygon boundaries
Polygon Hull Simplification

- Computes **Outer** and **Inner Hulls** of polygonal geometry
- Preserves polygonal topology, including holes and MultiPolygons
- Parameter: \( \text{vertex\_fraction} = \text{fraction of vertices kept} \)

```c
SimplifyPolygonHull( geom, vertex\_fraction, is\_outer );
```
Polygon Outer Hull VS Concave Hull

- Preserves Holes/MultiPolygon VS Single Polygon
- Parameter: Vertex Fraction VS Percent Convex

Outer Hull
Vertex Fraction = 0

Concave Hull
PctConvex = 0.2
Delaunay Triangulation

- Computes the **Delaunay Triangulation** of points
- Processes vertices **only**
  - *does not respect polygon linework*
  - *does not handle holes or MultiPolygons*

```sql
SELECT ST_DelaunayTriangles( geom );
```
Polygon Triangulation

- Computes the **Constrained Delaunay Triangulation** of polygons
  - respects polygon linework
  - handles holes and MultiPolygons

```sql
SELECT ST_TriangulatePolygon( geom );
```
Polygonal Coverages
Polygonal Coverages

- A set of non-overlapping polygons
- Many use cases
  - Cadastral parcels
  - Political jurisdictions
  - Land use
  - Geological regions
  - Etc, etc
- Can be represented as a full topological model
  - e.g. PostGIS Topology
- Another option…
Simple Polygonal Coverage

- Represent Polygonal Coverage as **discrete polygons**
  - A set of Polygons and MultiPolygons
  - Allows holes, disjoint regions
  - Implicit topology

- Advantages
  - Simple
  - Performant
  - Works with existing functions
Polygonal Coverage - Validity

- Coverage Validity required for:
  - Correct operation of coverage functions
  - Accurate modelling and analysis

- A set of polygons is a valid coverage if:
  - Polygons are valid
  - Polygons are non-overlapping
    - interiors do not intersect
  - Adjacent polygons are edge-matched
    - shared lines have identical vertices
Polygonal Coverage - Validation

- Tests if a set of valid polygons is a valid coverage
- For coverage-invalid polygons, reports invalid sections of polygon boundary:
  - Overlapping edges
  - Non edge-matched adjacent edges
- For each polygon returns
  - Invalid: invalid edges (MultiLineString)
  - Valid: empty or null

CoverageValidate( geom[] ) => MultiLineString[]
Polygonal Coverage - Union

- Computes the union of a set of coverage polygons
- Aggregate function, returns polygonal geometry
- Very fast (can be 100x faster than general-purpose union)

```
CoverageUnion( geom[] ) => MultiPolygon
```
Polygonal Coverage - Simplification

- Simplifies the boundaries of a set of coverage polygons
- Preserves topology; result is a valid coverage with identical structure

coverageSimplify( geom[], tolerance ) => geom[]

Size: 11,481 pts  =>  Size: 739 pts
Polygonal Coverage - Inner Simplification

- Simplifies the **inside boundaries** of a set of coverage polygons
- Preserves topology; result is a valid coverage with identical structure

CoverageSimplifyInner( geom[], tolerance ) => geom[]
Future Work

- Polygonal Coverage functions
  - Find Gaps
  - Clean
  - Precision Reduce
  - Overlay