GeoDev Maester Training:
Forging Your Chain With Python and GDAL

Stephen Christensen
Software Developer

mySidewalk

Agenda

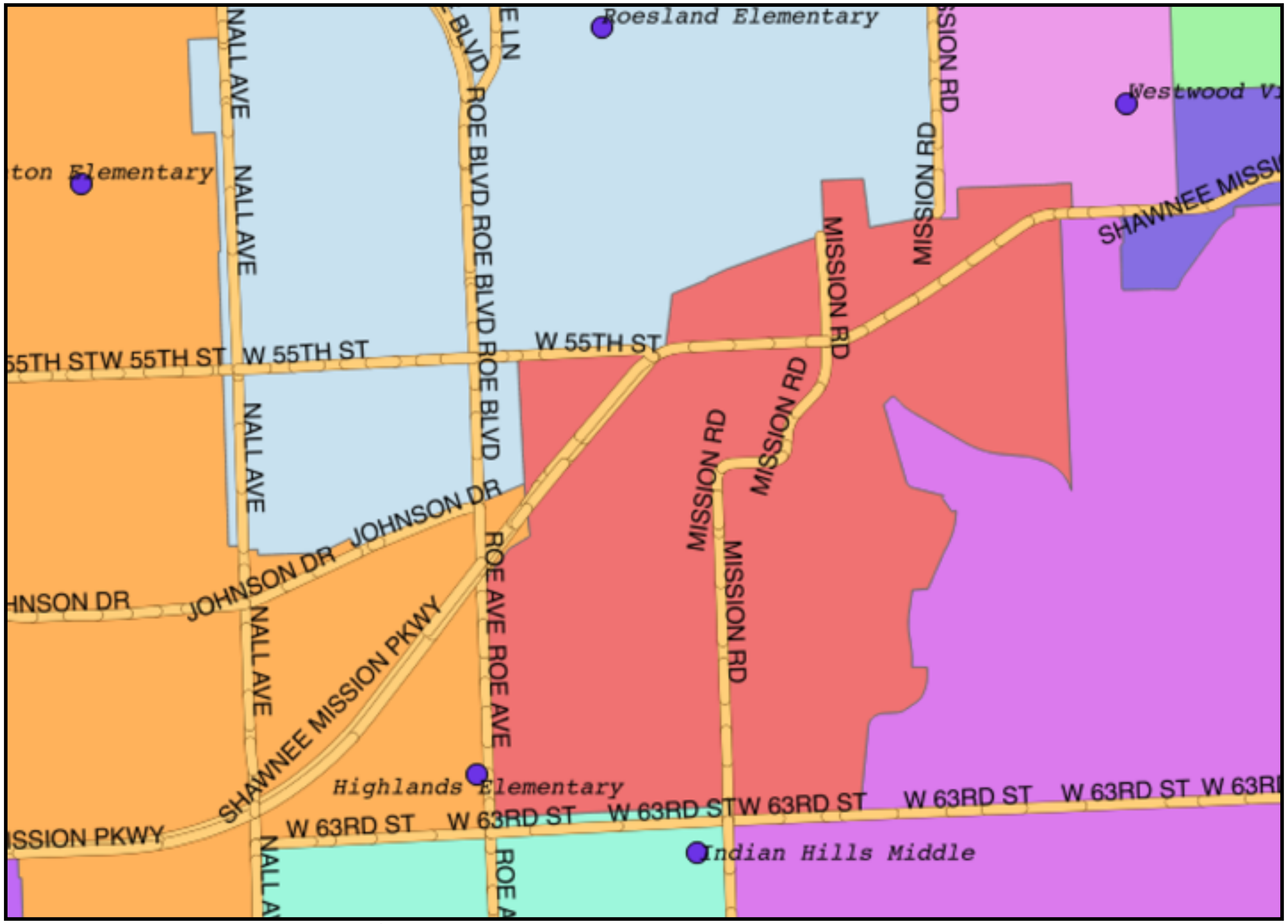
- Intro to GDAL
- 5 key concepts for geospatial development
- GDAL Insights
- Where to go from here
- Quick demo

Maester Luwin: *There are some who call my order the knights of the mind… Have you ever thought that you might wear a maester's chain? There is no limit to what you might learn.*
Bran: *I want to learn magic. The crow promised that I would fly.*

Main GIS Concepts

- vector data/raster data
- data sources
- layers
- features
- attributes
- projections
- resolution
- topology

● schools
▬ major roads
▭ city boundaries



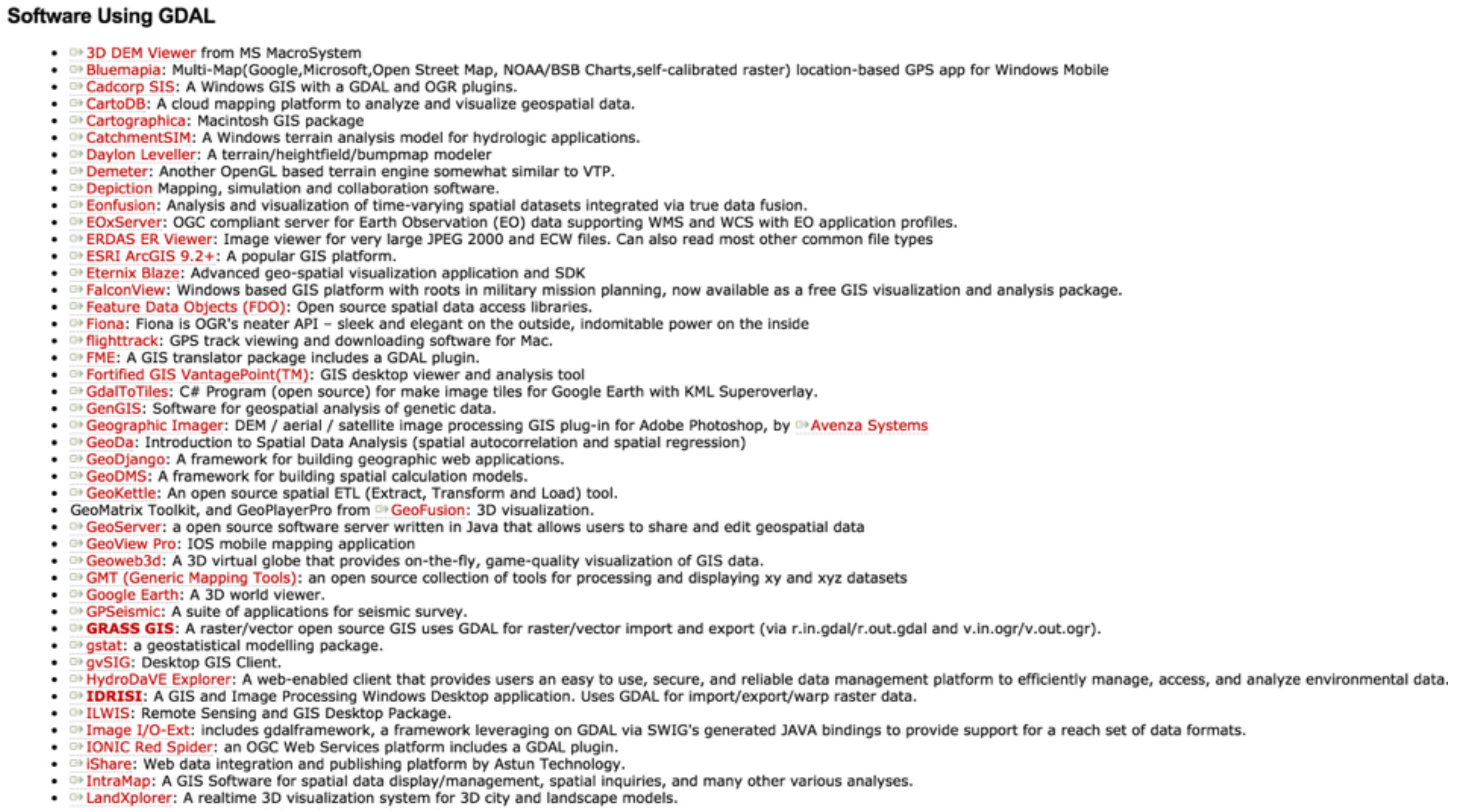
GDAL/OGR

- provides an abstract data model for numerous raster (142 drivers) and vector (84 drivers) formats
- OGR (vector) used to be separate from GDAL (raster), now they're combined under GDAL
- command line tools for common processing tasks
- language bindings that wrap the native C++ data model
- read/write geometries, fields, and attributes
- perform geoprocessing tasks
- has no required dependencies, but it relies on GEOS (Geometry Engine, Open Source) for geometric operations and PROJ4 for reprojecting
- Overseen by the OSGeo Foundation and follows OGC standards
- officially pronounced "goodle"

How do I install GDAL?

- Use one of my Dockerfiles
  - https://github.com/christensenst/gdal-mangling/blob/master/gdal_2_1_1/Dockerfile
- Use the version of GDAL provided with the QGIS install

What tools use GDAL?



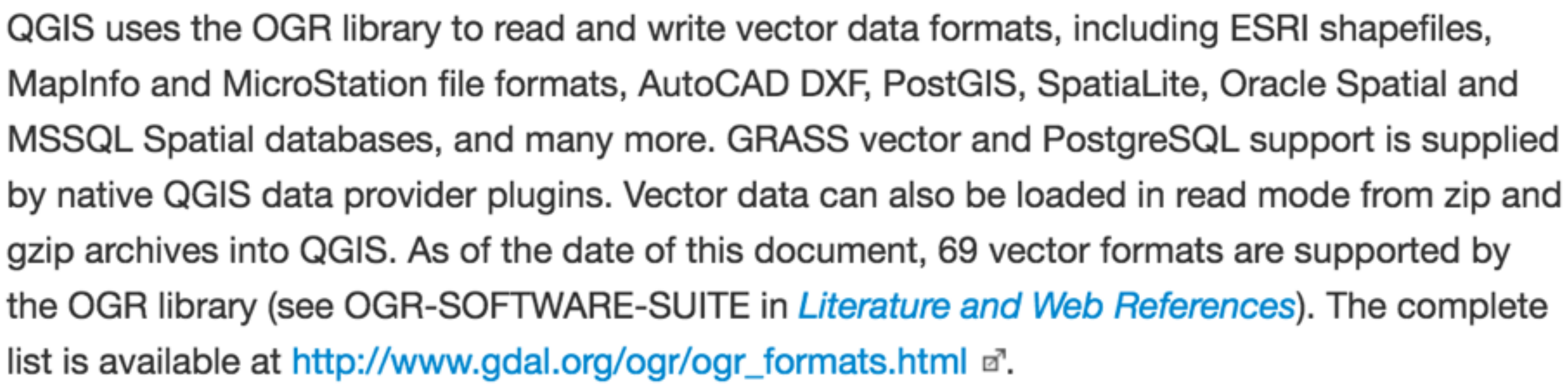
Software Using GDAL

- 3D DEM Viewer from MS MacroSystem
- Bluemapia: Multi-Map(Google,Microsoft,Open Street Map, NOAA/BSB Charts,self-calibrated raster) location-based GPS app for Windows Mobile
- Cadcorp SIS: A Windows GIS with a GDAL and OGR plugins.
- CartoDB: A cloud mapping platform to analyze and visualize geospatial data.
- Cartographica: Macintosh GIS package
- CatchmentSIM: A Windows terrain analysis model for hydrologic applications.
- Daylon Leveller: A terrain/heightfield/bumpmap modeler
- Demeter: Another OpenGL based terrain engine somewhat similar to VTP.
- Depiction Mapping, simulation and collaboration software.
- Eonfusion: Analysis and visualization of time-varying spatial datasets integrated via true data fusion.
- EOxServer: OGC compliant server for Earth Observation (EO) data supporting WMS and WCS with EO application profiles.
- ERDAS ER Viewer: Image viewer for very large JPEG 2000 and ECW files. Can also read most other common file types
- ESRI ArcGIS 9.2+: A popular GIS platform.
- Eternix Blaze: Advanced geo-spatial visualization application and SDK
- FalconView: Windows based GIS platform with roots in military mission planning, now available as a free GIS visualization and analysis package.
- Feature Data Objects (FDO): Open source spatial data access libraries.
- Fiona: Fiona is OGR's neater API – sleek and elegant on the outside, indomitable power on the inside
- flighttrack: GPS track viewing and downloading software for Mac.
- FME: A GIS translator package includes a GDAL plugin.
- Fortified GIS VantagePoint(TM): GIS desktop viewer and analysis tool
- GdalToTiles: C# Program (open source) for make image tiles for Google Earth with KML Superoverlay.
- GenGIS: Software for geospatial analysis of genetic data.
- Geographic Imager: DEM / aerial / satellite image processing GIS plug-in for Adobe Photoshop, by Avenza Systems
- GeoDa: Introduction to Spatial Data Analysis (spatial autocorrelation and spatial regression)
- GeoDjango: A framework for building geographic web applications.
- GeoDMS: A framework for building spatial calculation models.
- GeoKettle: An open source spatial ETL (Extract, Transform and Load) tool.
- GeoMatrix Toolkit, and GeoPlayerPro from GeoFusion: 3D visualization.
- GeoServer: a open source software server written in Java that allows users to share and edit geospatial data
- GeoView Pro: IOS mobile mapping application
- Geoweb3d: A 3D virtual globe that provides on-the-fly, game-quality visualization of GIS data.
- GMT (Generic Mapping Tools): an open source collection of tools for processing and displaying xy and xyz datasets
- Google Earth: A 3D world viewer.
- GPSeismic: A suite of applications for seismic survey.
- **GRASS GIS**: A raster/vector open source GIS uses GDAL for raster/vector import and export (via r.in.gdal/r.out.gdal and v.in.ogr/v.out.ogr).
- gstat: a geostatistical modelling package.
- gvSIG: Desktop GIS Client.
- HydroDaVE Explorer: A web-enabled client that provides users an easy to use, secure, and reliable data management platform to efficiently manage, access, and analyze environmental data.
- **IDRISI**: A GIS and Image Processing Windows Desktop application. Uses GDAL for import/export/warp raster data.
- ILWIS: Remote Sensing and GIS Desktop Package.
- Image I/O-Ext: includes gdalframework, a framework leveraging on GDAL via SWIG's generated JAVA bindings to provide support for a reach set of data formats.
- IONIC Red Spider: an OGC Web Services platform includes a GDAL plugin.
- iShare: Web data integration and publishing platform by Astun Technology.
- IntraMap: A GIS Software for spatial data display/management, spatial inquiries, and many other various analyses.
- LandXplorer: A realtime 3D visualization system for 3D city and landscape models.

…and it keeps going, too many for one slide

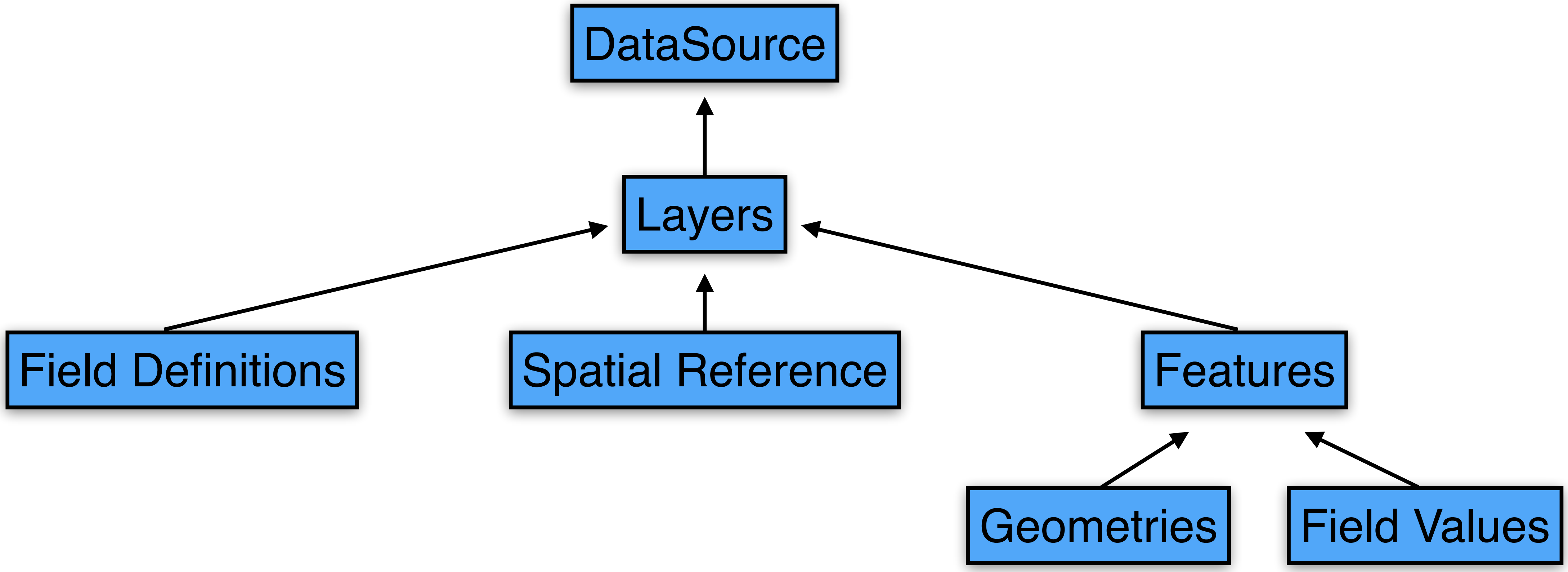
How do these tools use GDAL?

…from the QGIS docs:



QGIS uses the OGR library to read and write vector data formats, including ESRI shapefiles, MapInfo and MicroStation file formats, AutoCAD DXF, PostGIS, SpatiaLite, Oracle Spatial and MSSQL Spatial databases, and many more. GRASS vector and PostgreSQL support is supplied by native QGIS data provider plugins. Vector data can also be loaded in read mode from zip and gzip archives into QGIS. As of the date of this document, 69 vector formats are supported by the OGR library (see OGR-SOFTWARE-SUITE in *Literature and Web References*). The complete list is available at http://www.gdal.org/ogr/ogr_formats.html.

GDAL/OGR Layer Data Model



Reading Geo Data

```
import ogr
data_source = ogr.Open("/home/st_ephen/sample_polygons.shp")
layer = data_source.GetLayer()
for feature in layer:
    print feature.GetGeometryRef().ExportToWkt()
```

```
POLYGON ((-94.491189182471643 39.720777195800032,-94.210245049016194
39.84512951716556,-93.984568613945413 39.58951641213641 9,-94.366836861106108
39.331600486341252,-94.491189182471643 39.720777195800032))
POLYGON ((-95.568909300972877 38.841099663177225,-95.368563894328418
38.864127870837507,-95.202760799174371 38.813465813984891,-95.165915666917925
38.659176822660996,-95.228091827600693 38.537127322061494,-95.400803385052811
38.433500387590222,-95.317901837475787 38.573972454317946,-95.283359525985361
38.682205030321278,-95.354746969732247 38.769712219430353,-95.472190828799683
38.656874001894963,-95.568909300972877 38.841099663177225))
```

Reading Geo Data

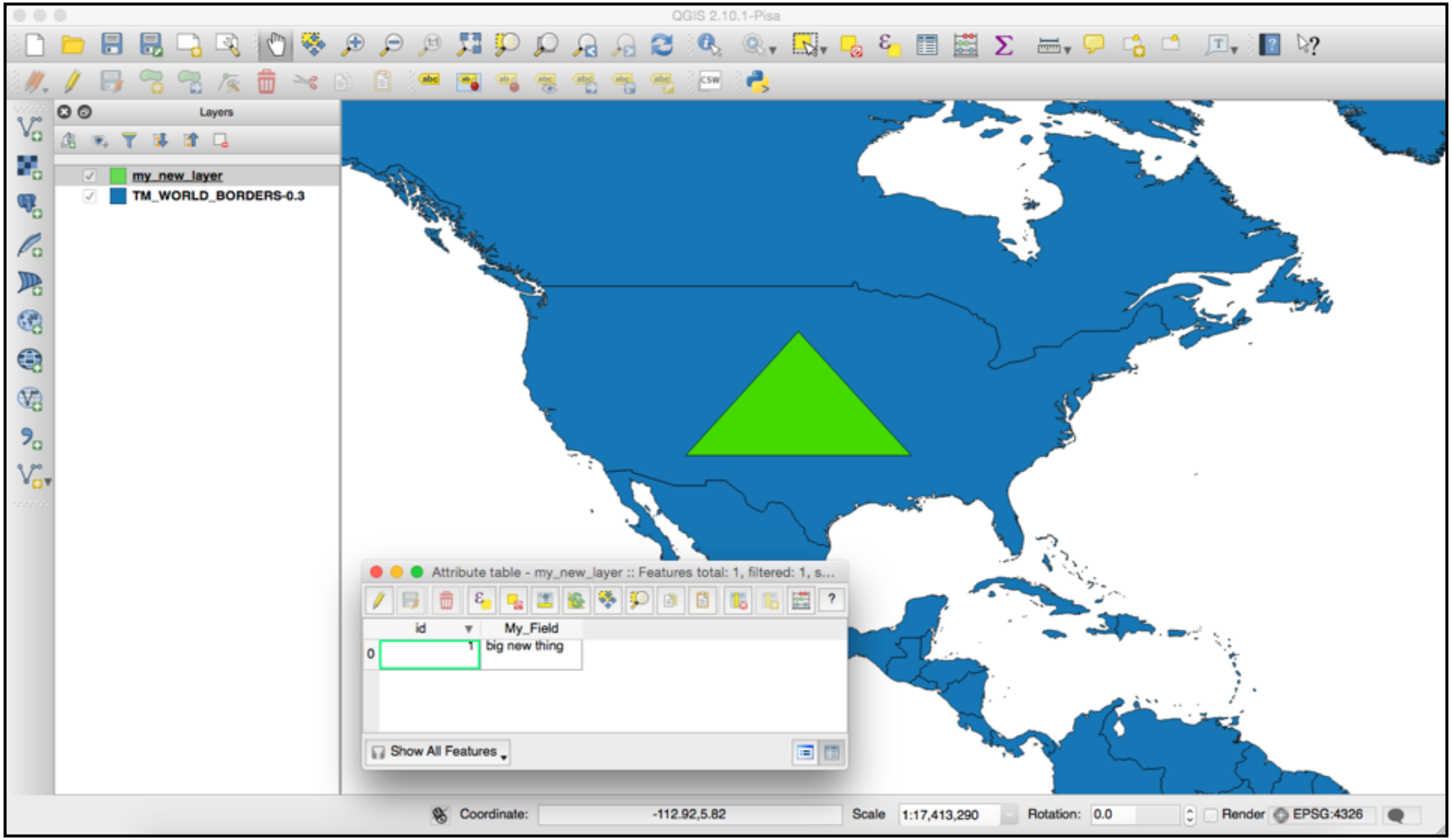
```
layer.SetNextByIndex(0) # reset the underlying C++ iterator
for feature in layer:
    print feature.GetField('my_label')
```

```
first
second
```

Writing Geo Data

```
import ogr, osr
driver = ogr.GetDriverByName("ESRI Shapefile")
data_source = driver.CreateDataSource("my_new_layer.shp")
srs = osr.SpatialReference()
srs.ImportFromEPSG(4326)
layer = data_source.CreateLayer("my_new_layer", srs, ogr.wkbPolygon)
field = ogr.FieldDefn("My_Field", ogr.OFTString)
field.SetWidth(24)
layer.CreateField(field)
feature = ogr.Feature(layer.GetLayerDefn())
feature.SetField("My_Field", "big new thing")
wkt = "POLYGON((-110 34, -100 45, -90 34, -110 34))"
polygon = ogr.CreateGeometryFromWkt(wkt)
feature.SetGeometry(polygon)
layer.CreateFeature(feature)
feature.Destroy()
data_source.Destroy()
```

Output



Attribute table - my_new_layer :: Features total: 1, filtered: 1, s...

	id	My_Field
0	1	big new thing

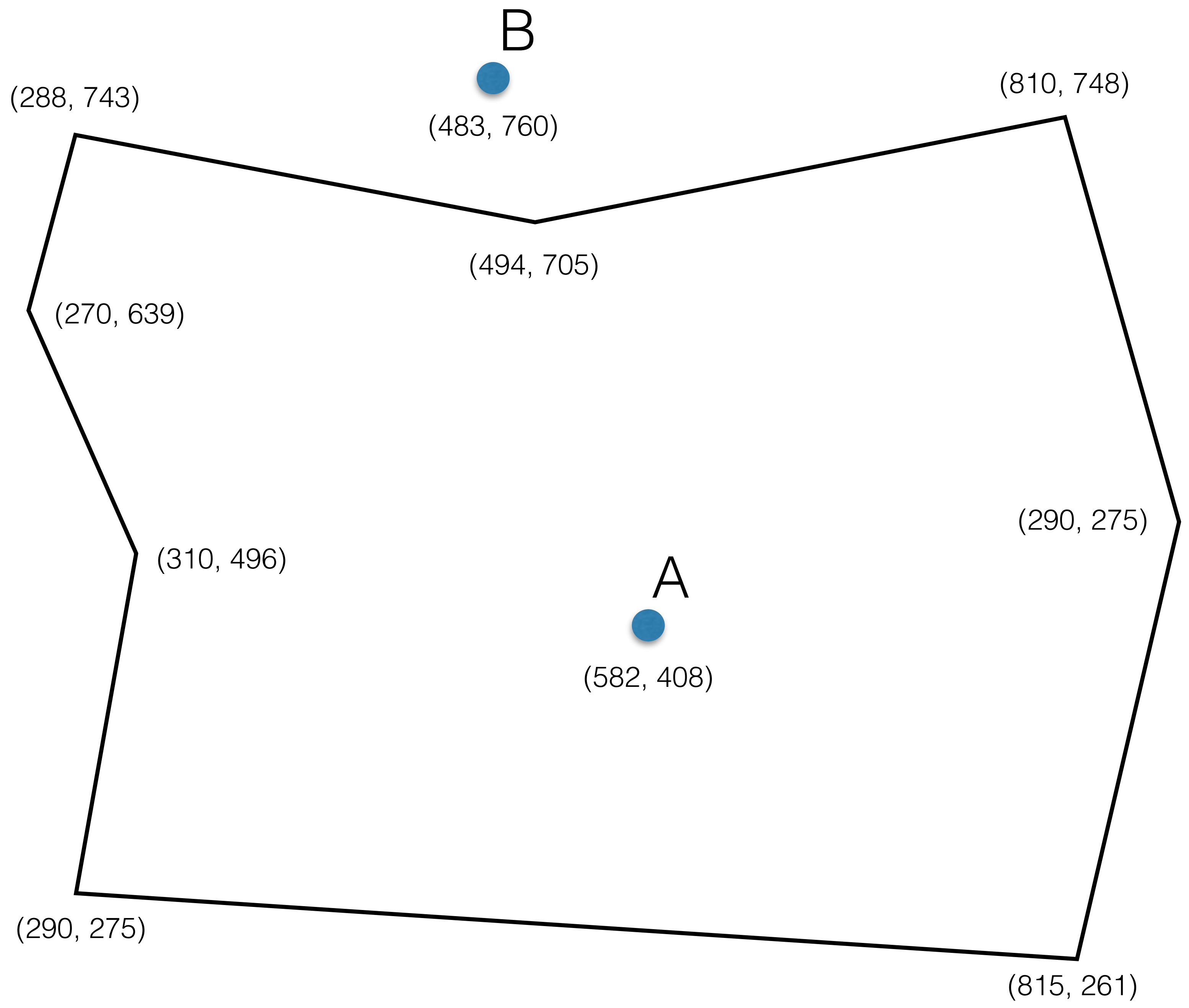
Geoprocessing

- Layer methods:
  - Clip()
  - Erase()
  - Intersection()
  - Union()
- Geometry methods:
  - Area()
  - Buffer()
  - Centroid()
  - Contains()
  - ConvexHull()
  - Intersection()
  - Intersects()
  - Union()
  - Within()

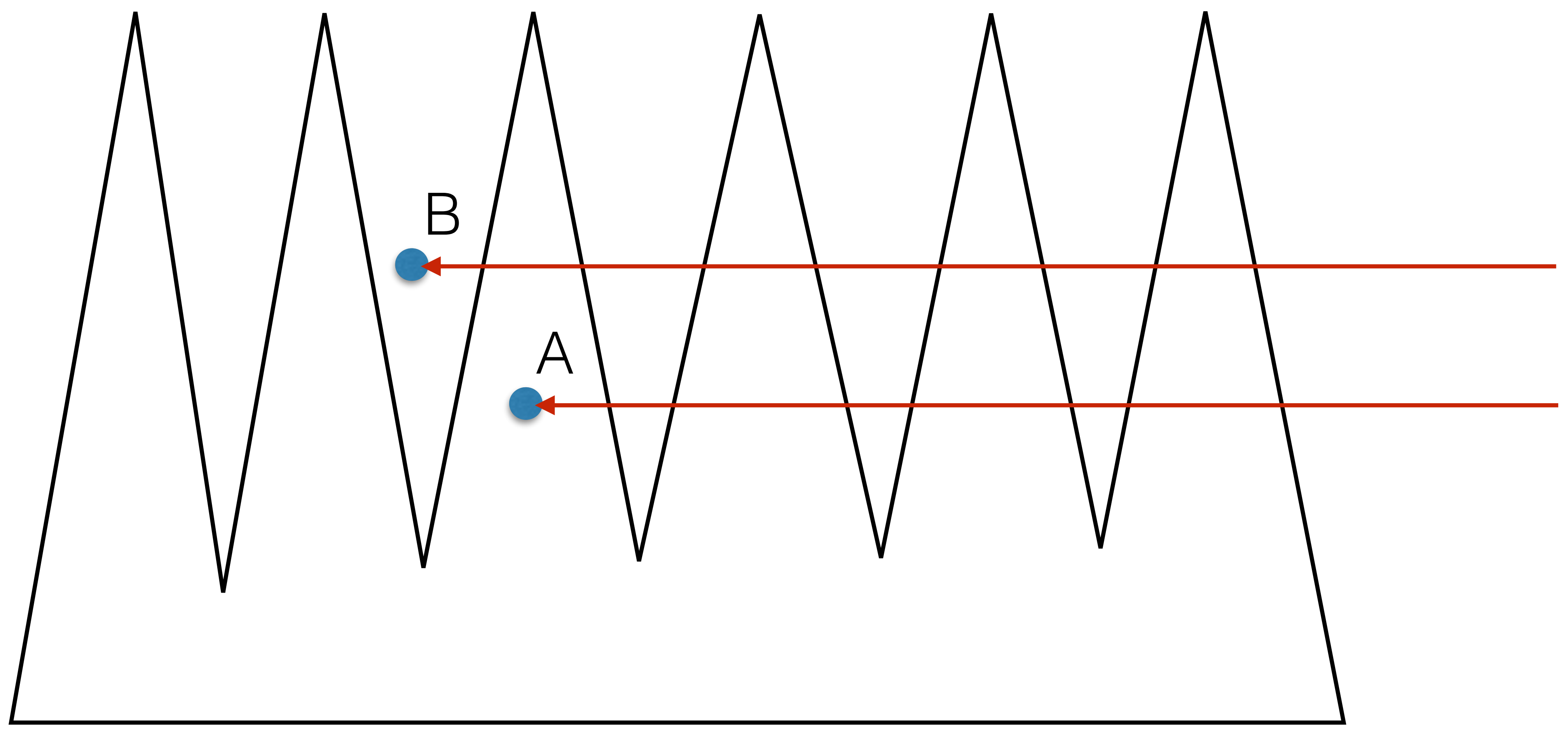
Five Essential geodev concepts

Point in Polygon Problem

How do you determine if a point is within a polygon?



Answer: Use Ray Casting

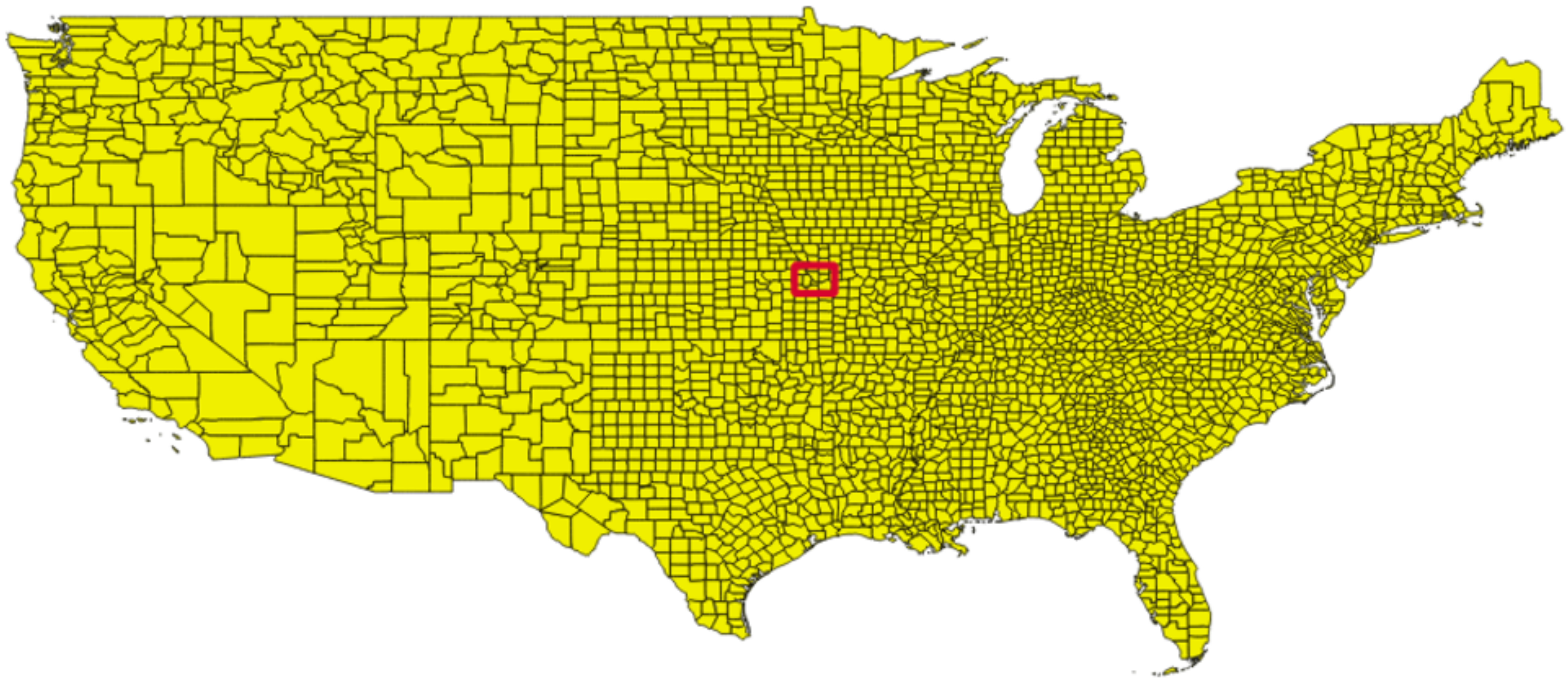


The ray will have an odd # of intersections with the polygon if the point is inside, an even number if it is outside (there are a few edge scenarios to handle)

Key Point: Determining if a point is in a polygon is an O(N) operation and gets increasingly expensive as resolution increases.

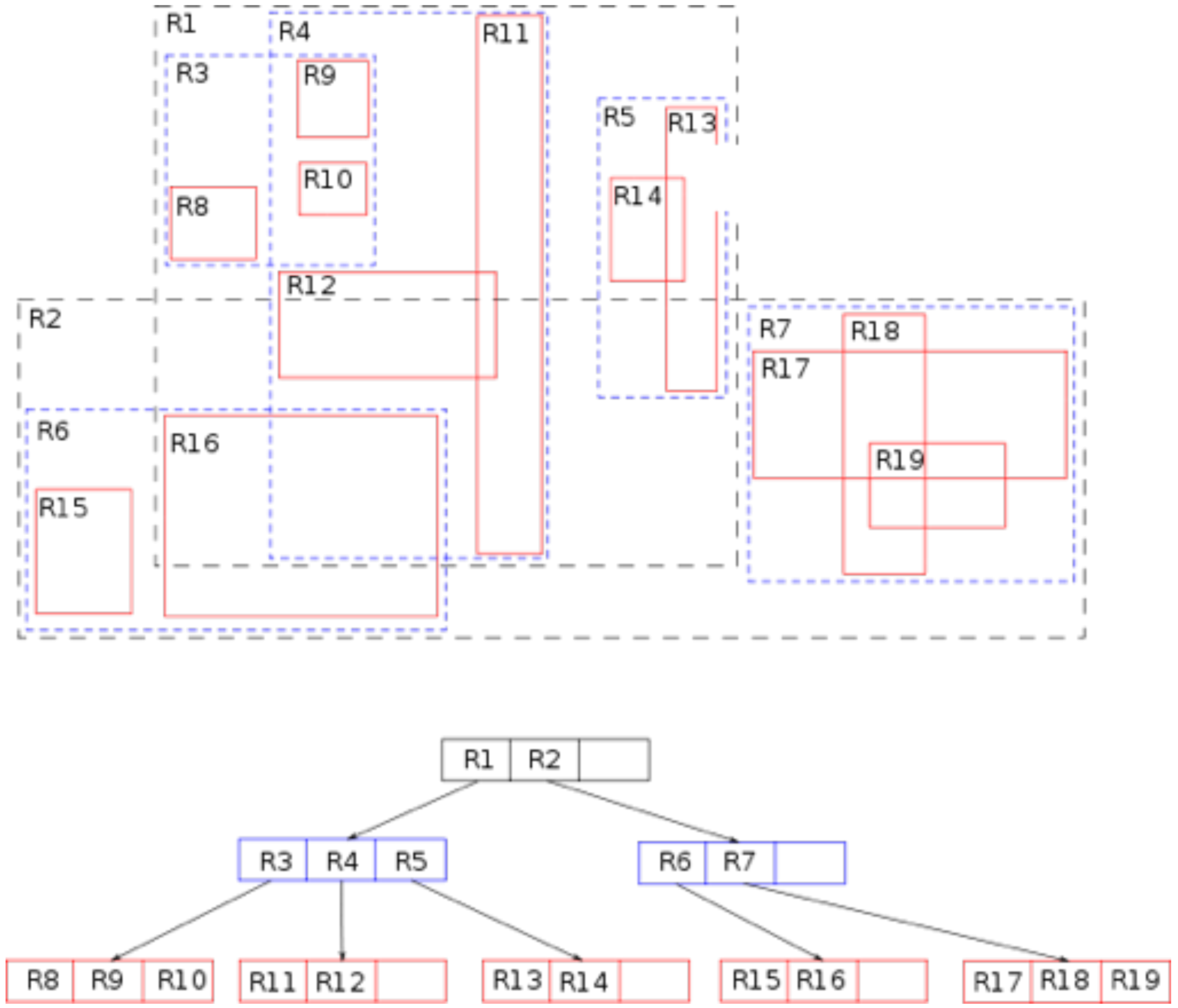
Random Access By Location

How can you efficiently access a small set of features based on location?

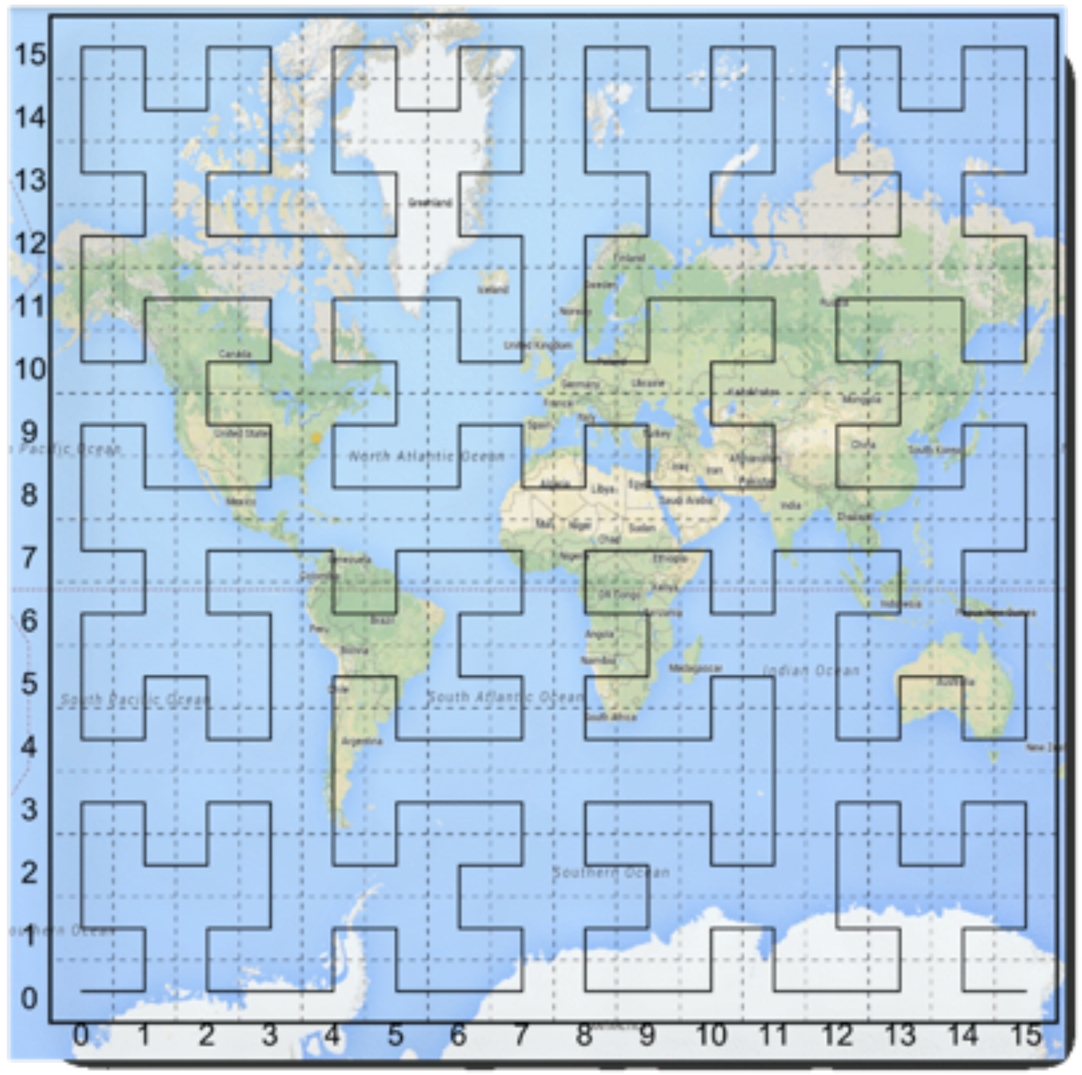


Answer: Use a Spatial Index

Rtree



Hilbert Space-filling Curve



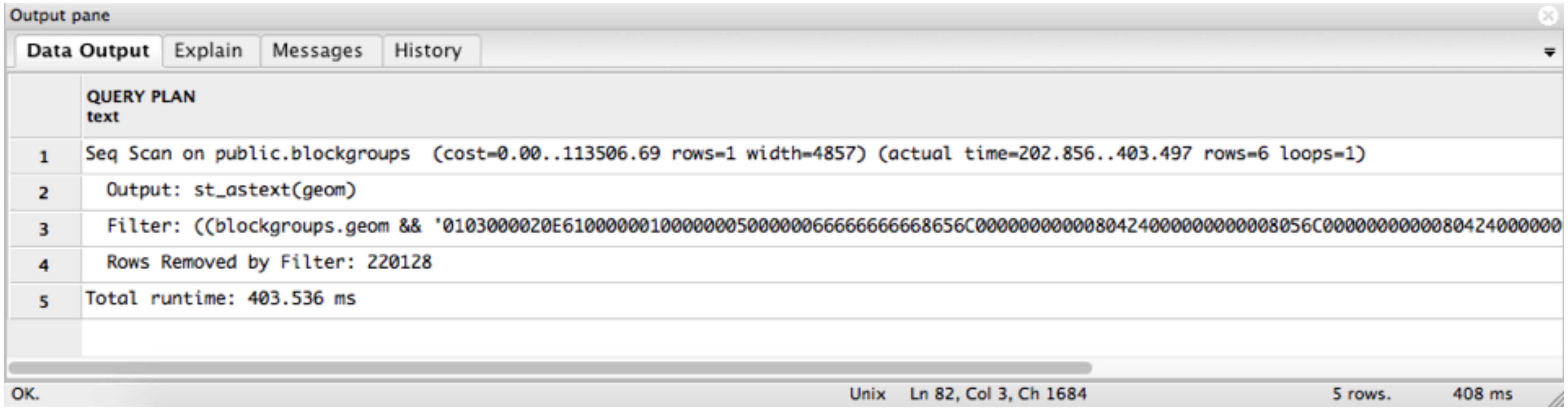
Key Point: Use a spatial index to optimize random access of geospatial features

How do I know if GDAL’s SetSpatialFilter() uses a layer’s spatial index if the index exists?

What should I use instead?

PostGIS: Indexes

```
EXPLAIN ANALYZE VERBOSE
SELECT ST_AsText(geom)
FROM blockgroups
WHERE ST_Intersects(
  geom,
  ST_GeomFromText(
    'POLYGON((-90.1 37, -90 37, -90 37.1, -90.1 37.1, -90.1 37))', 4326
  )
);
```



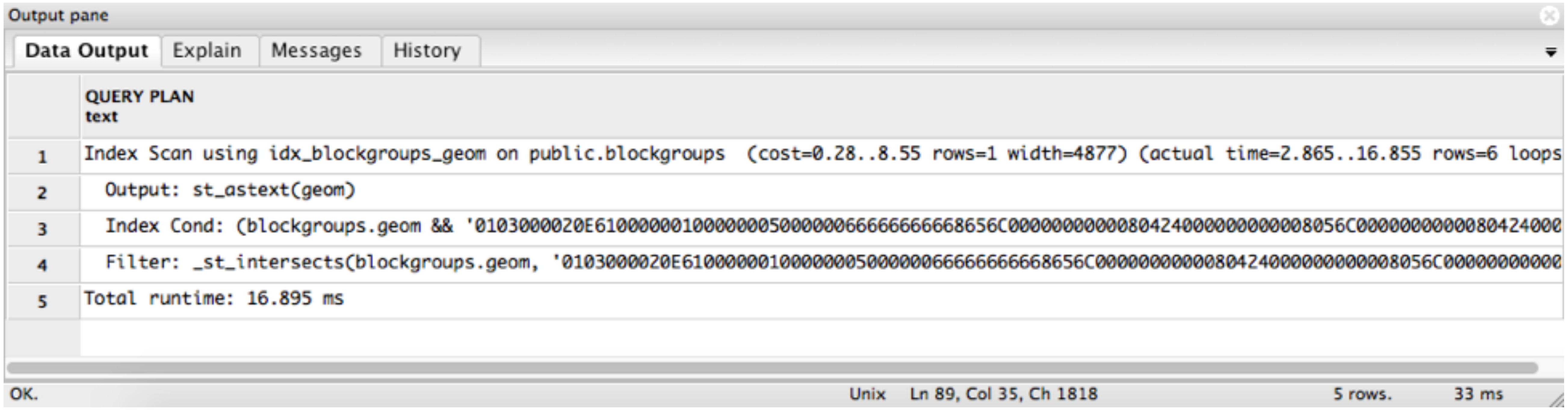
```
QUERY PLAN
text
1 Seq Scan on public.blockgroups  (cost=0.00..113506.69 rows=1 width=4857) (actual time=202.856..403.497 rows=6 loops=1)
2   Output: st_astext(geom)
3   Filter: ((blockgroups.geom && '0103000020E6100000010000000500000066666666668656C000000000008042400000000000008056C00000000000804240000000
4   Rows Removed by Filter: 220128
5 Total runtime: 403.536 ms
```

```
OK.                    Unix   Ln 82, Col 3, Ch 1684          5 rows.     408 ms
```

PostGIS: GiST Index

```
CREATE INDEX idx_blockgroups_geom
ON blockgroups USING gist (geom);
VACUUM ANALYZE blockgroups;

-- rerun the query
```



Output pane

Data Output | Explain | Messages | History

```
   QUERY PLAN
   text
1  Index Scan using idx_blockgroups_geom on public.blockgroups  (cost=0.28..8.55 rows=1 width=4877) (actual time=2.865..16.855 rows=6 loops
2    Output: st_astext(geom)
3    Index Cond: (blockgroups.geom && '0103000020E6100000010000000500000066666666668656C000000000008042400000000000008056C0000000000080424000
4    Filter: _st_intersects(blockgroups.geom, '0103000020E6100000010000000500000066666666668656C000000000008042400000000000008056C00000000000
5  Total runtime: 16.895 ms
```

OK.   Unix   Ln 89, Col 35, Ch 1818   5 rows.   33 ms

Reprojecting Geo Data

```
import ogr, osr

def transform(geom, input_sr, output_srid):
    spatial_reference = osr.SpatialReference()
    spatial_reference.ImportFromEPSG(output_srid)
    coordinate_transform = osr.CoordinateTransformation(input_sr, spatial_reference)
    geom.Transform(coordinate_transform)
    print 'EPSG {}: {}'.format(output_srid, geom.ExportToWkt())

ds = ogr.Open('SCHOOL_PT.shp')
layer = ds.GetLayer()
feature = layer.GetFeature(0)
geom = feature.GetGeometryRef()
print 'Initial coordinates: {}'.format(geom.ExportToWkt())
input_sr = layer.GetSpatialRef()
transform(geom.Clone(), input_sr, 4326)
transform(geom.Clone(), input_sr, 2796)
transform(geom.Clone(), input_sr, 3857)
```

Reprojecting Geo Data (cont.)

```
Initial coordinates: POINT (2198992.499961942434311 256438.265605181455612)
EPSG 4326: POINT (-94.879709306258292 38.995485594378088)
EPSG 2796: POINT (670254.25449690897949 78162.539681516442215)
EPSG 3857: POINT (-10561960.926586495712399 4721024.94320560619235)
```

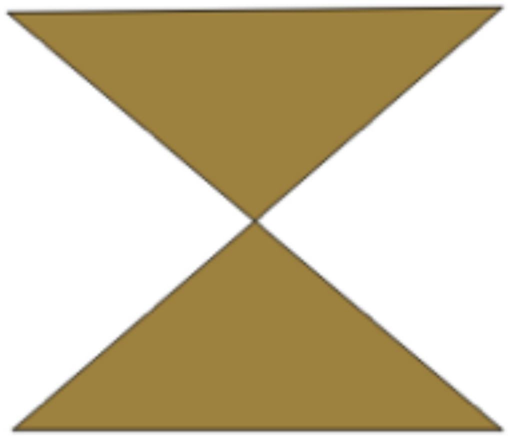
Key Point: Geometries by themselves don’t know where they are on the earth. A spatial reference defines context for a geometry. Geometries can become mismatched with spatial references if care is not taken

How can I ensure that the features in my layer are actually on the earth?

Feature Validity

```
print geom.ExportToWkt()
print geom.IsValid()
```

```
POLYGON ((-94.665201275495789 38.970752922740395,-94.623597027230815
38.971093405441131,-94.664763336040366
38.943509003928412,-94.623597027230815
38.943509003928412,-94.665201275495789 38.970752922740395))
Warning 1: Self-intersection at or near point -94.644416900386773
38.957142586016573
False
```



Key Point: Many geospatial operations assume that the features operated on are valid and will fail if they are invalid. Use ST_MakeValid() in PostGIS to clean invalid geometries

Simplification

https://bost.ocks.org/mike/simplify/

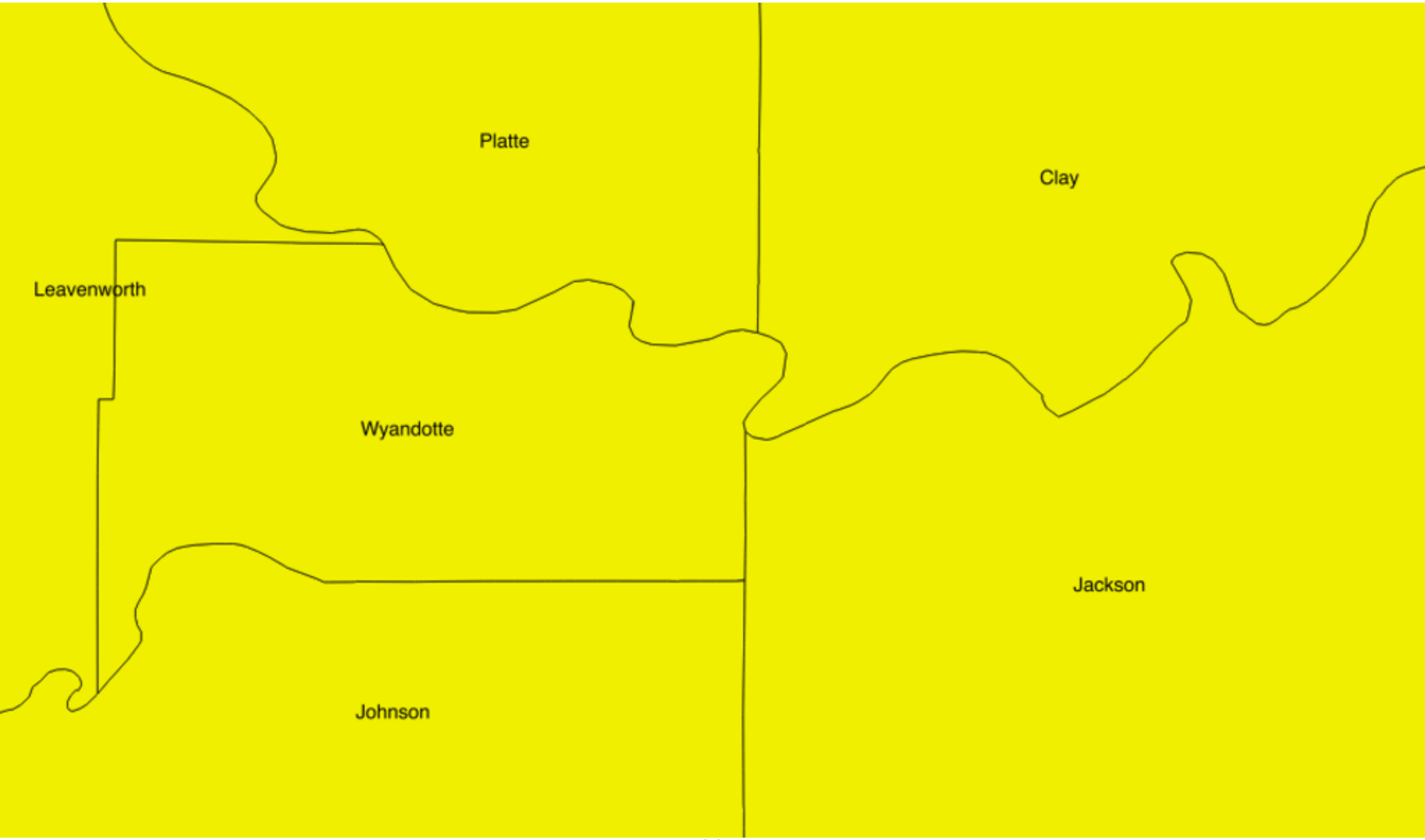
```
import ogr, osr

ds = ogr.Open(‘cb_2015_us_county_500k.shp')
layer = ds.GetLayer()
feature = layer.GetFeature(0)
geom = feature.GetGeometryRef()
print geom.GetGeometryRef(0).GetPointCount()
simple_geom = geom.Clone().Simplify(0.0008)
print simple_geom.GetGeometryRef(0).GetPointCount()
simple_geom = geom.Clone().Simplify(0.008)
print simple_geom.GetGeometryRef(0).GetPointCount()
```

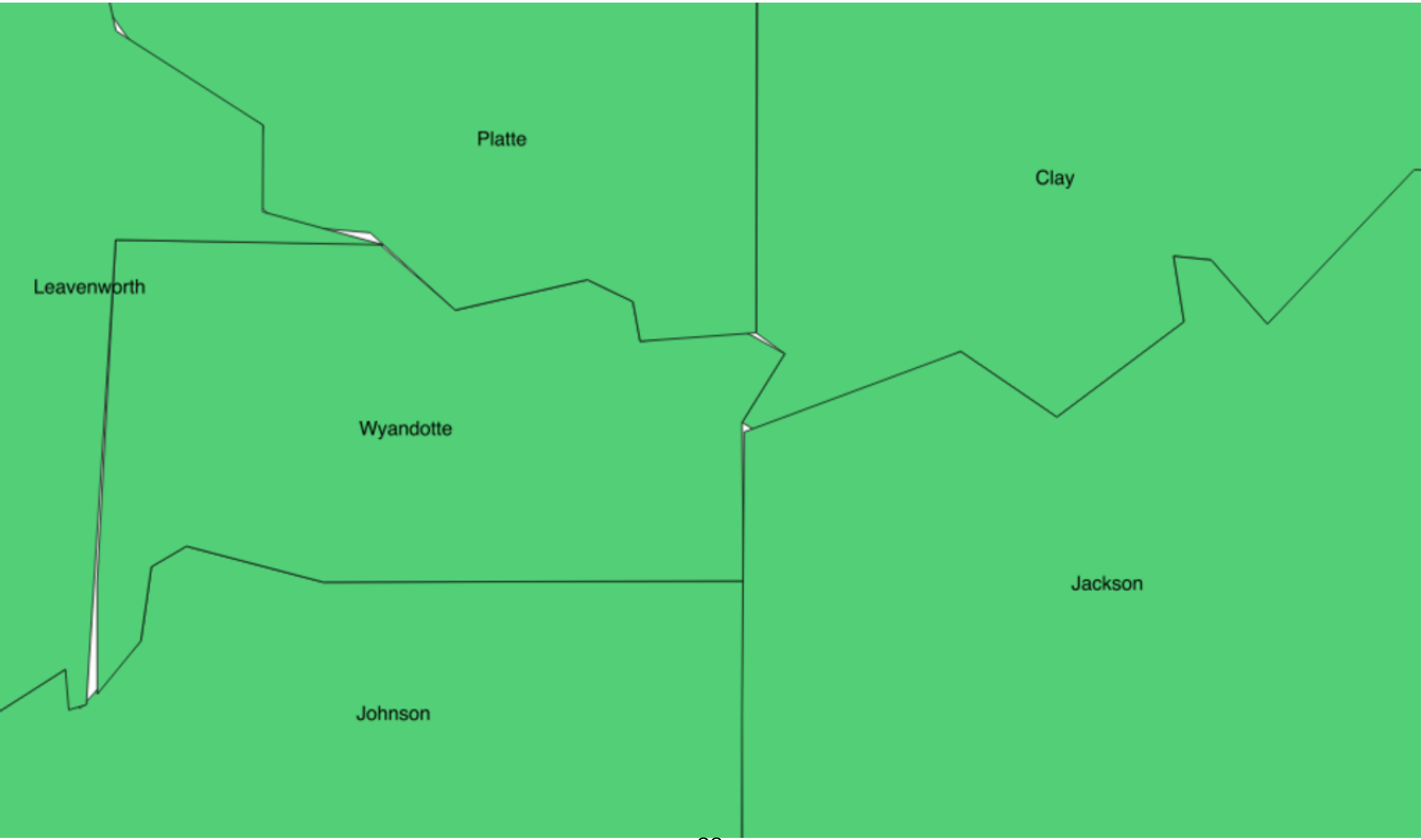
```
540
340
39
```

Key Point: Simplify geometries to reduce data transmission sizes as well as optimize geographic operations

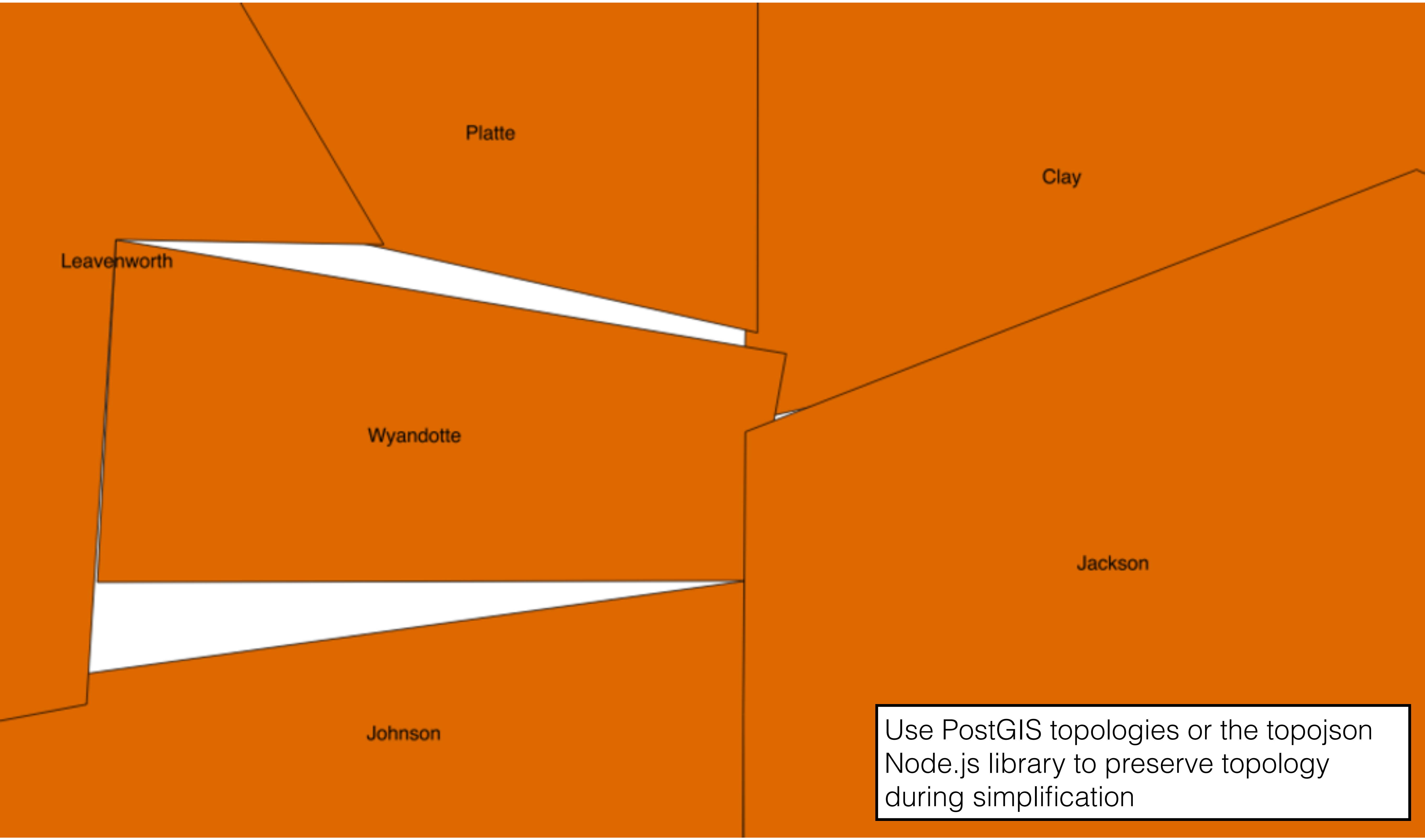
Original High Res Polygons



We lost our topology!



Simply… useless



Use PostGIS topologies or the topojson Node.js library to preserve topology during simplification

GDAL Insights

Common questions about GDAL

When should I use the Python GDAL bindings vs using PostGIS?

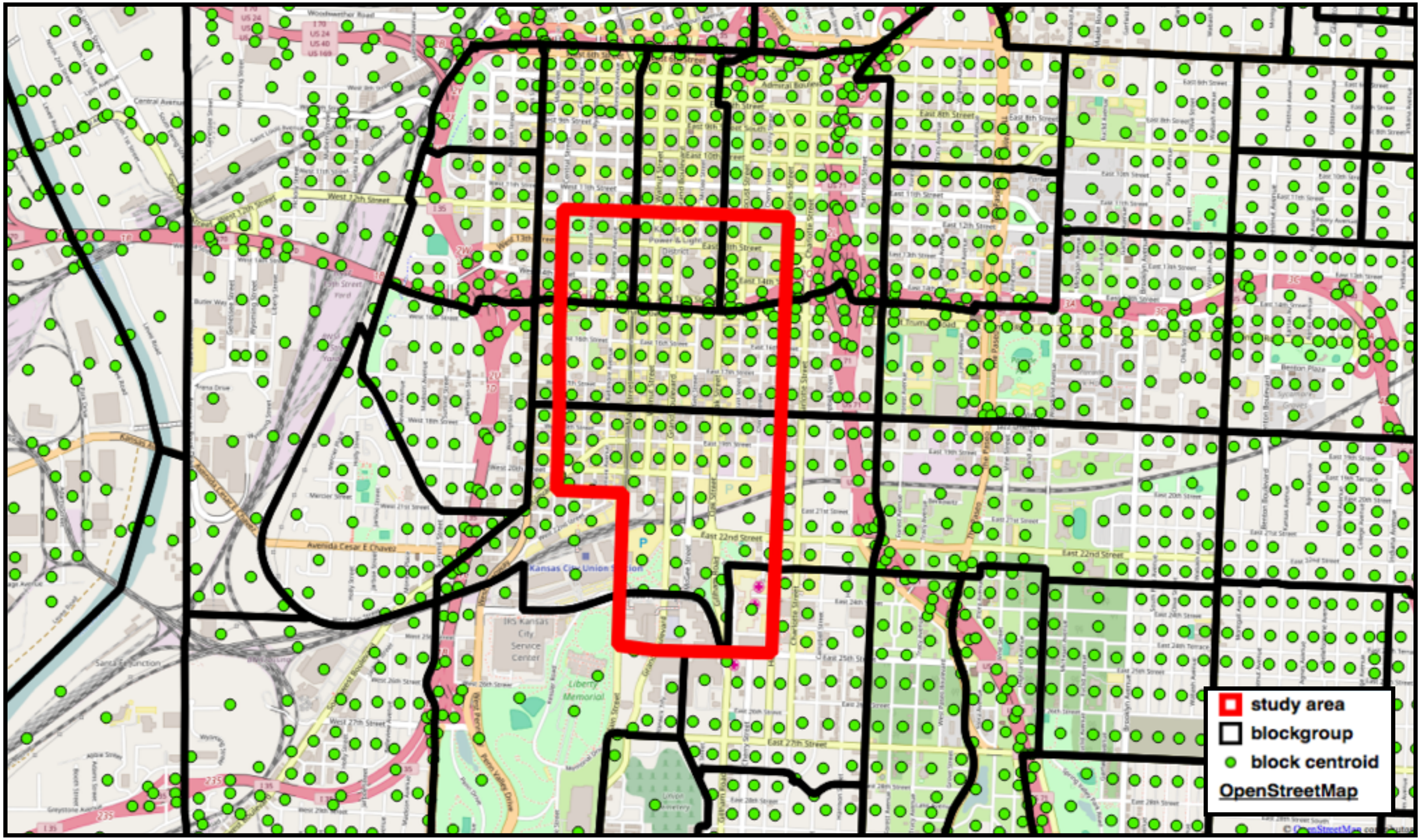
Wait, someone told me GDAL was a command line tool…

Why not use some of the higher level tools built on GDAL or something else? (fiona, Geopandas, Pyshp, PySAL)

Where to go from here

- getting familiar with QGIS is essential, go through the tutorials: http://www.qgistutorials.com/en
- GDAL and PostGIS: the 2 main fundamental (backend) tools
    - Python GDAL Cookbook https://pcjericks.github.io/py-gdalogr-cookbook/
    - *Python Geospatial Development* by Erik Westra
    - *PostGIS in Action* by Regina Obe
    - PostGIS docs http://postgis.net/docs/manual-2.2/
- FOSS4GNA talks
    - https://2015.foss4g-na.org/
    - https://www.youtube.com/playlist?list=PL_LGEn4G1x8r99HkY_RfJNlDYB9LDzMj0
- Mapbox, CartoDB, and CloudGIS blogs on their websites

Weighted Blockpoint Apportionment



Demo time!