

Geometry Operations: OGR and GEOS

The GEOS library

<http://geos.refractions.net>

provides the spatial predicates originally used in PostGIS, now OGR, and soon MapServer. In this exercise we'll explore unions, intersections, differences, buffers, and work our way up to the task of creating a buffered union of many features from a shapefile.

Matplotlib

Along the way we are going to use the matplotlib package for visualization of our results. This is matlab-like software that is attracting a lot of attention from Python users. If we have time at the end of the workshop, some of you may be interested in digging deeper into matplotlib.

```
>>> from matplotlib import pylab
>>> pylab.plot()
[]
>>> pylab.show()
```

This creates an output window into which we'll render geometries.

Geometries

Let's create two simple, overlapping polygons using the same string interpolation and WKT factory method as in the previous exercise:

```
>>> r1 = {'minx': -5.0, 'miny': 0.0, 'maxx': 5.0, 'maxy': 10.0}
>>> r2 = {'minx': 0.0, 'miny': -5.0, 'maxx': 10.0, 'maxy': 5.0}
>>> template = 'POLYGON ((%(minx)f %(miny)f, %(minx)f %(maxy)f, %(maxx)f %(maxy)f, %(maxx)f %(miny)f, %(minx)f %(miny)f))'
>>> w1 = template % r1
>>> w2 = template % r2
```

You could print these to verify. Next we import the ogr module and use its WKT factory to create instances of ogr.Geometry:

```
>>> from gdal import ogr
```

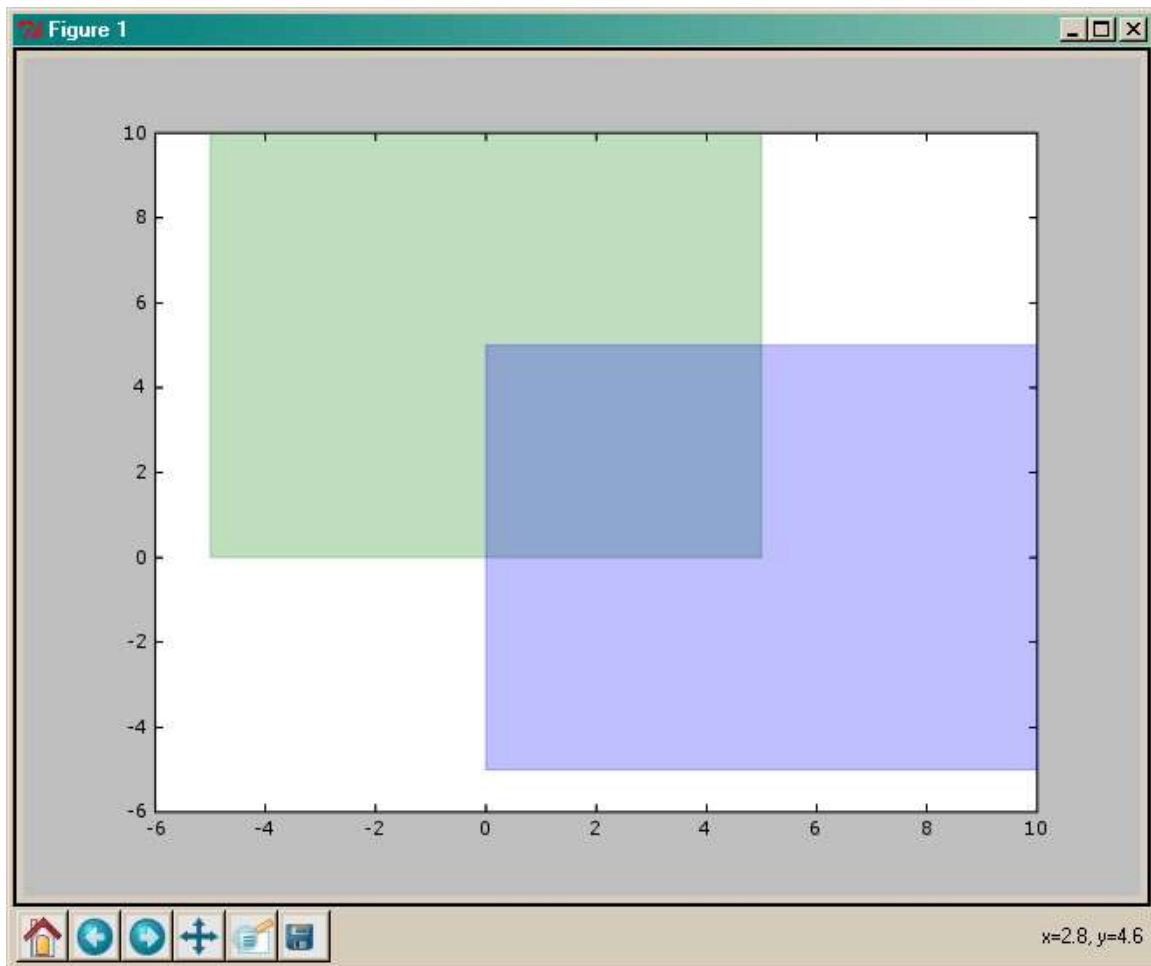
```
>>> g1 = ogr.CreateGeometryFromWkt(w1)
>>> g2 = ogr.CreateGeometryFromWkt(w2)
```

Plotting

Initially we downloaded a helper file named plot.py. It contains two functions for plotting geometries in the matplotlib window.

```
>>> from plot import plot_poly, plot_line
>>> plot_poly(g1, color='green', alpha=0.25)
>>> plot_poly(g2, color='blue', alpha=0.25)
```

The result should be something like

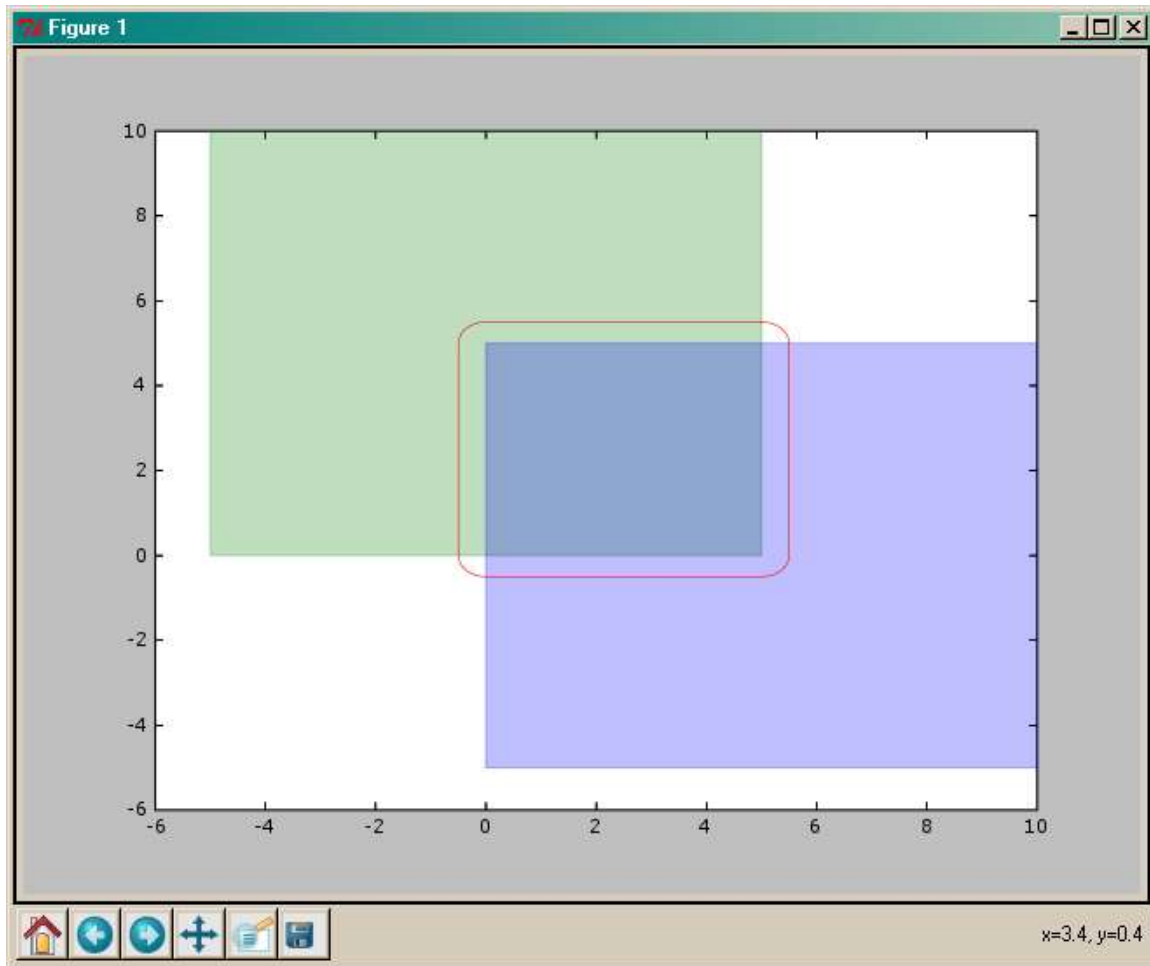


Intersection

Let's try the `Intersection()` and `Buffer()` methods of `ogr.Geometry` first.

```
>>> inter = g1.Intersection(g2)
>>> buffered_inter = inter.Buffer(0.5)
>>> plot_line(buffered_inter, color='red')
```

The result:

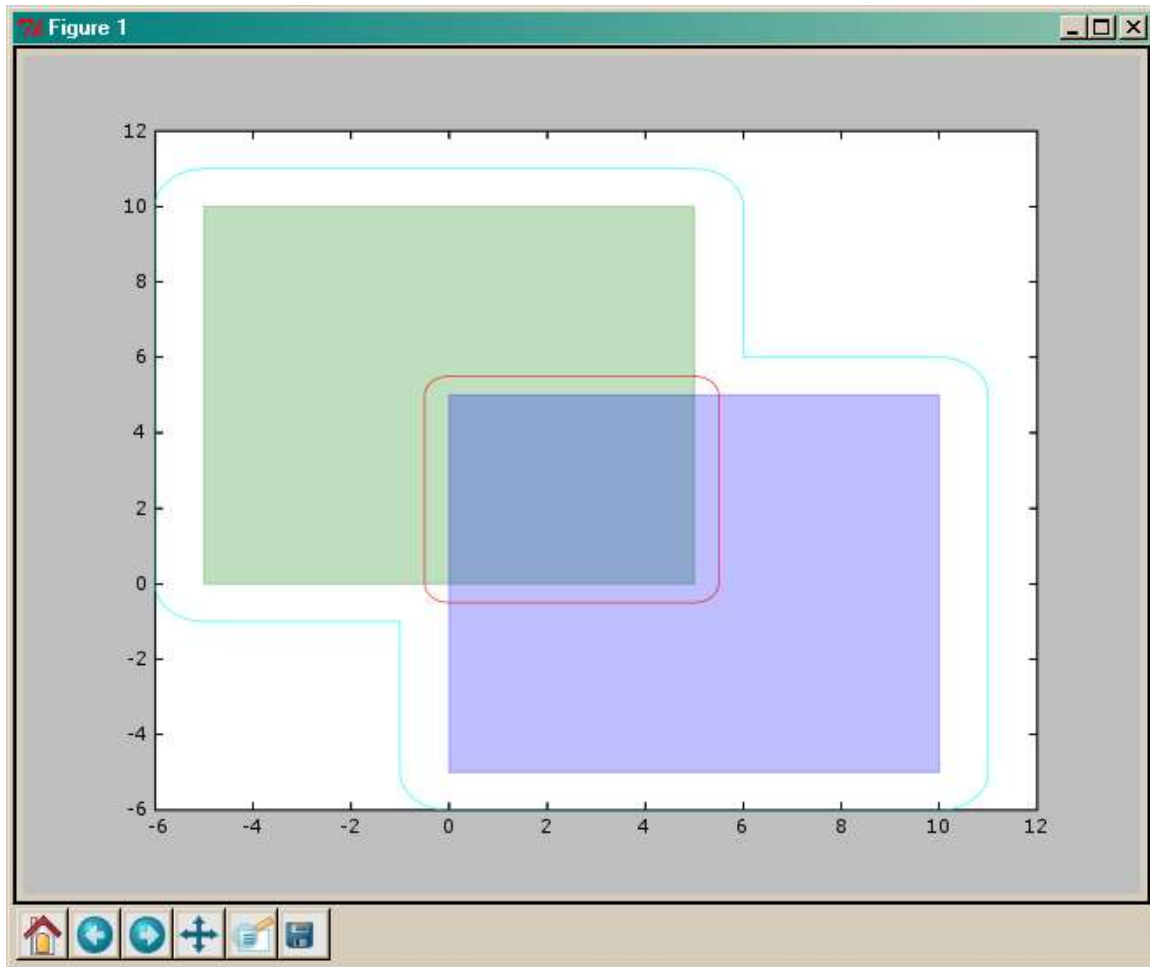


Union

Now the `Union()` method.

```
>>> union = g1.Union(g2)
>>> buffered_union = union.Buffer(1.0)
>>> plot_line(buffered_union, color='cyan')
```

and the results



Lifelike Geometries

Let's close up that output window and move on to less artificial geometries. At `c:\ms4w\python\data\world_borders.shp` is a world borders shapefile derived from VMAP0 by Schuyler Erle, Rich Gibson, and Jo Walsh. We'll use the `OGRFeatureIterator` class from the `fiter.py` helper module to select several of the features from this shapefile:

```
>>> from fiter import OGRFeatureIterator
>>> filename = r'c:\ms4w\python\data\world_borders.shp'
```

Now, define a spatial bounding box and an OGR attribute filter to constrain features. The `GEOS Union()` operation is very slow, and we don't want to wait for too many polygons.

```
>>> bounds = (-10.0, 30.0, 20.0, 60.0)
>>> attrfilter = "fips_cntry = 'UK'"
```

Next, we create a list to hold selected features, and declare the name `u`, for our union geometry, to begin with the value `None`.

```
>>> geoms = []
>>> u = None
```

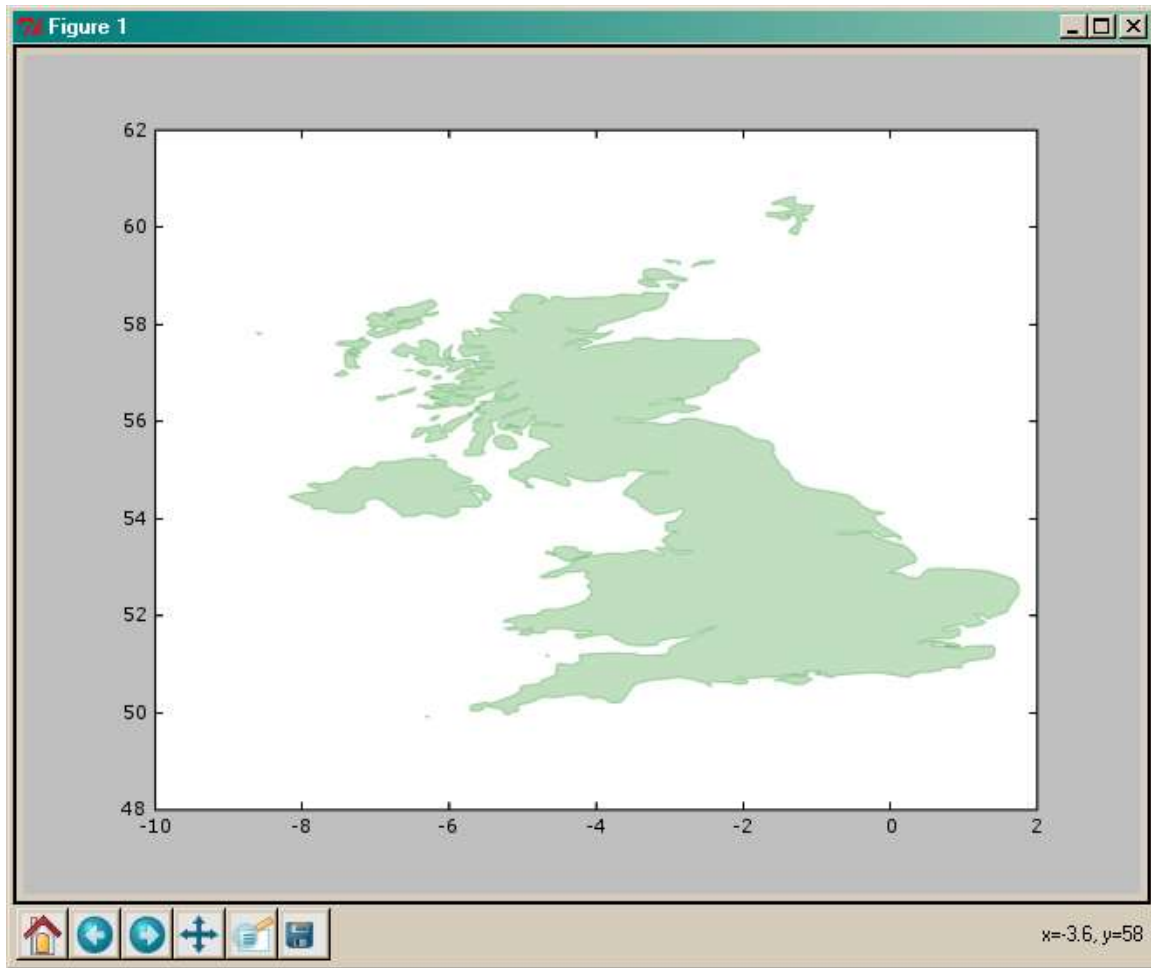
The following iteration appends each selected geometry `g` and builds up the union of all selected geometries. Iterators are a very common construct, and a big component of Python flavor. The `if/else` blocks below ensure that we begin our union geometry as the clone of a selected geometry, and clone only once.

```
>>> for g in OGRFeatureIterator(filename, bounds,
attrfilter):
...     geoms.append(g)
...     if u:
...         u = u.Union(g)
...     else:
...         u = g.Clone()
...
>>>
```

Now, let's plot the selected geometries using the previously imported `plot_poly()` function.

```
>>> for g in geoms:
...     plot_poly(g, color='green', alpha=0.25)
...
>>>
```

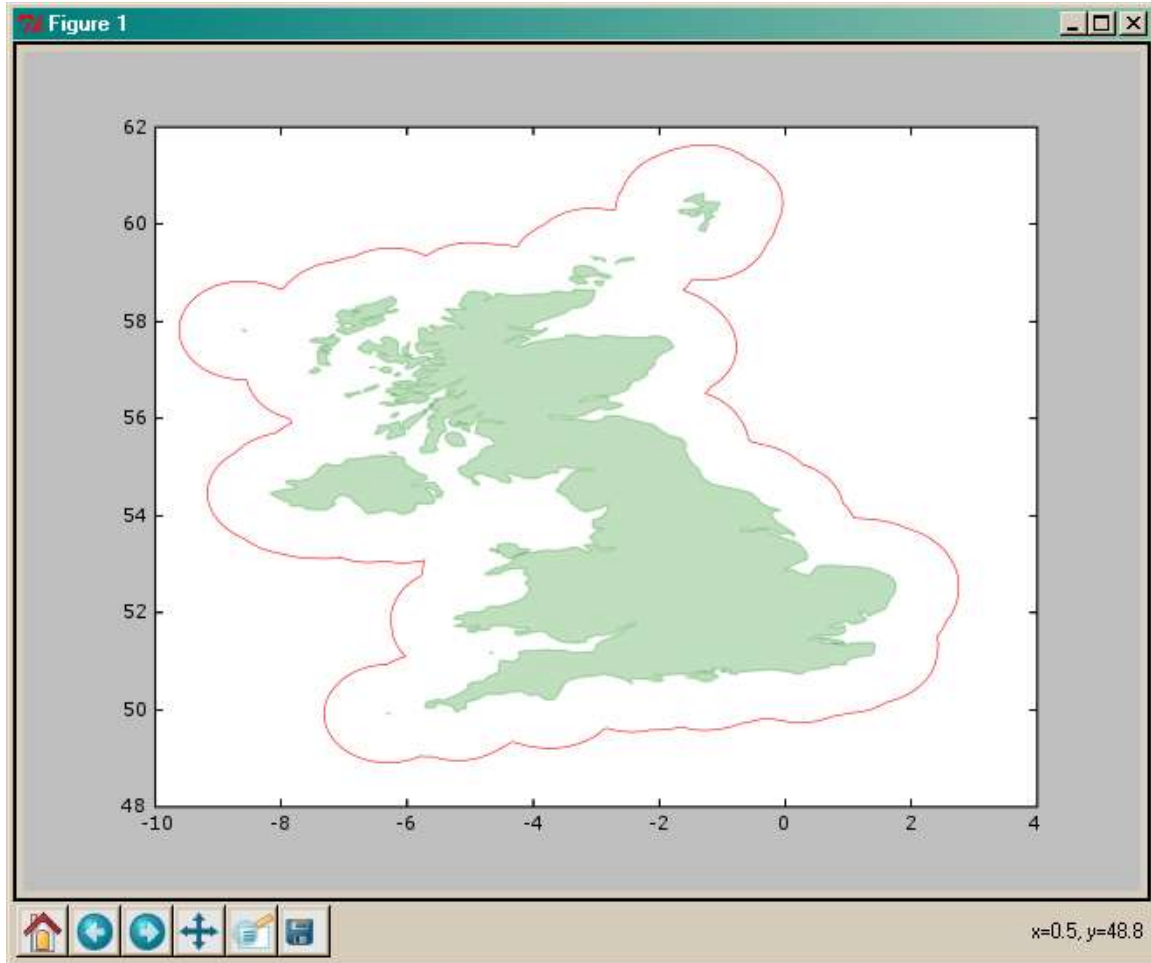
the result



Now, buffer the union and plot it. This is a fairly lengthy operation ...

```
>>> buffer = u.Buffer(1.0)
>>> plot_line(buffer, color='red')
>>>
```

The results



Continuation

In the workshop's extra time, some of you may want to try saving these geometries to a file using ogr.py as we did in the previous tileindex exercise, and display them in OpenEV. Some may be interested in grabbing some features via WFS and plotting them in the same window with the buffered UK features.