Creating Geometries and Handling Projections with OGR

Open Source RS/GIS Python
Week 2

Creating a new geometry

1. Create an empty Geometry object with **`ogr.Geometry(<OGRwkbGeometryType>)`**

2. Define what the geometry is in a different way for each type (point, line, polygon, etc.)

Creating points

- Use **`AddPoint( <x>, <y>, [<z>])`** to set coordinates
- The height value <z> is optional and defaults to 0

```
point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(10,20)
```

Creating lines

- Add new vertices to the line with **`AddPoint(<x>, <y>, [<z>])`**
- Change the coordinates of a vertex with **`SetPoint(<index>, <x>, <y>, [<z>])`** where <index> is the index of the vertex to change

```
line = ogr.Geometry(ogr.wkbLineString)
line.AddPoint(10,10)
line.AddPoint(20,20)
line.SetPoint(0,30,30) #(10,10) -> (30,30)
```

- To get the number of vertices in a line use **`GetPointCount()`**

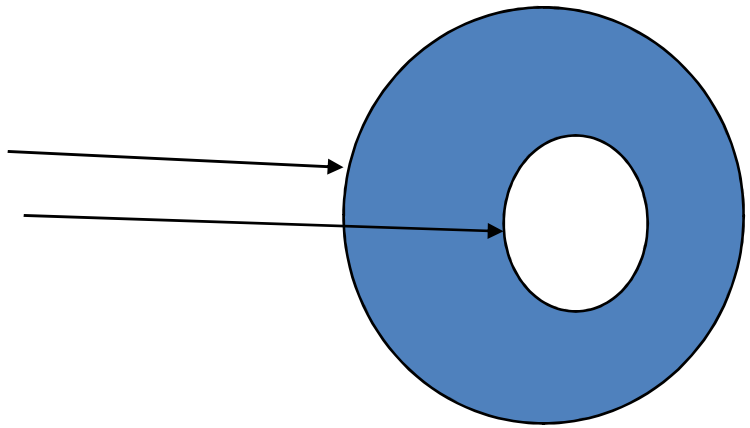
```
print line.GetPointCount()
```

- To get the x,y coordinates for a specific vertex use **`GetX(<vertex_index>)`** and **`GetY(<vertex_index>)`**

```
print line.GetX(0)
print line.GetY(0)
```

Creating polygons

- Must create rings first and then add them to the polygon later
- For example, a polygon with one hole in it will have two rings



- To make a ring, create an empty ring Geometry and then use AddPoint(<x>, <y>) to add vertices

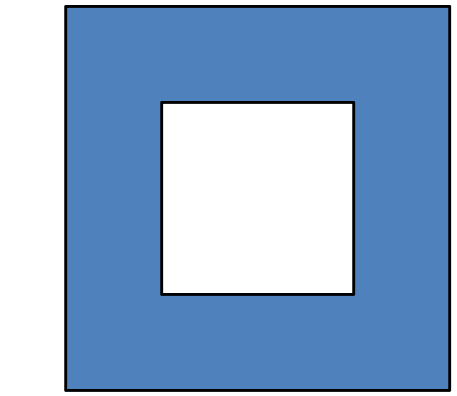
```
ring = ogr.Geometry(ogr.wkbLinearRing)
ring.AddPoint(0,0)
ring.AddPoint(100,0)
ring.AddPoint(100,100)
ring.AddPoint(0,100)
```

- To close the ring, use **`CloseRings()`** or make the last vertex the same as the first

```
ring.CloseRings()
ring.AddPoint(0,0)
```

```
outring = ogr.Geometry(ogr.wkbLinearRing)
outring.AddPoint(0,0)
outring.AddPoint(100,0)
outring.AddPoint(100,100)
outring.AddPoint(0,100)
outring.AddPoint(0,0)

inring = ogr.Geometry(ogr.wkbLinearRing)
inring.AddPoint(25,25)
inring.AddPoint(75,25)
inring.AddPoint(75,75)
inring.AddPoint(25,75)
inring.CloseRings()
```



- Now make the polygon and add the rings with **`AddGeometry(<geometry>)`**

```
polygon = ogr.Geometry(ogr.wkbPolygon)
polygon.AddGeometry(outring)
polygon.AddGeometry(inring)
```

- Get the number of rings in a polygon with **`GetGeometryCount()`**

```
print polygon.GetGeometryCount()
```

- Get a ring object from a polygon with **`GetGeometryRef(<ring_index>)`**; indices are the same order you added them to the polygon

```
outring = polygon.GetGeometryRef(0)
inring = polygon.GetGeometryRef(1)
```

- Get the number of vertices in a ring and the coordinates of those vertices the same way as for lines (slide 5)

Multi Geometries

- MultiPoint, MultiLineString, MultiPolygon…
- Create a geometry and then add it to the multi-version with **`AddGeometry(<geom>)`**

```
multipoint = ogr.Geometry(ogr.wkbMultiPoint)
point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(10,10)
multipoint.AddGeometry(point)
point.AddPoint(20,20)
multipoint.AddGeometry(point)
```

- Get sub-geometries the same way as getting rings from a polygon

When to destroy geometries

- Do not destroy geometries that come from an existing feature – Python will crash!
- Do destroy geometries that are created during script execution (even if they are used to create new features)
  - Created manually (today)
  - Created from geoprocessing functions (next week)

```
Polygon.Destroy()
```

```
# script to add a point to a new shapefile

# import modules and set the working directory
import ogr, os, sys
os.chdir('f:/data/classes/python/data')

# get the driver
driver = ogr.GetDriverByName('ESRI Shapefile')

# create a new data source and layer
if os.path.exists('test.shp'):
  driver.DeleteDataSource('test.shp')
ds = driver.CreateDataSource('test.shp')
if ds is None:
  print 'Could not create file'
  sys.exit(1)
layer = ds.CreateLayer('test', geom_type=ogr.wkbPoint)

# add an id field to the output
fieldDefn = ogr.FieldDefn('id', ogr.OFTInteger)
layer.CreateField(fieldDefn)

# create a new point object
point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(150, 75)
```

```
# get the FeatureDefn for the output layer
featureDefn = layer.GetLayerDefn()

# create a new feature
feature = ogr.Feature(featureDefn)
feature.SetGeometry(point)
feature.SetField('id', 1)

# add the feature to the output layer
layer.CreateFeature(feature)

# destroy the geometry and feature and close the data source
point.Destroy()
feature.Destroy()
ds.Destroy()
```

Review: Splitting strings

- To split the string 'A1,23,56' into a list with 3 entries use the built-in string method **split(<separator>)**

```
info = 'A1,23,56'.split(',')
id = info[0]
x = info[1]
y = info[2]
```

- If <separator> is not provided, it splits on whitespace

- Homework hint on splitting strings:

  The input lines look like this:

  county_name:x_1 y_1,x_2 y_2,…,x_n y_n

  1. Split on colons
  2. Get the county name and list of coordinates
  3. Split the list of coordinates on commas
  4. Loop through this new list (one entry for each xy pair)
     1. Split the xy pair on spaces to get x and y
     2. Use x and y to add a vertex to your geometry

```
## county_name:x1 y1,x2 y2,…,xn yn
# for each line in the text file:
    ring = ogr.Geometry(ogr.wkbLinearRing)
    tmp = line.split(':')
    name = tmp[0]
    coords = tmp[1]
    coordlist = coords.split(',')
    for coord in coordlist:
        xy = coord.split()
        x = float(xy[0])
        y = float(xy[1])
        ring.AddPoint(x, y)
    poly = ogr.Geometry(ogr.wkbPolygon)
    poly.AddGeometry(ring)
    # create a new feature and set geometry and
    #   county name
    # destroy poly and your new feature
```

Review: Type conversions

- **`bool(<x>)`** – convert to boolean (0 and empty strings are False)
- **`float(<x>)`** – convert floating point
- **`int(<x>)`** – convert to an integer
- **`long(<x>)`** – convert to a long integer
- **`str(<x>)`** – convert to a string

Assignment 2a

- Write polygons to a shapefile
  - Create a new polygon shapefile and populate it with the polygons defined in ut_counties.txt
  - Each line in the file looks like this:

county_name:x_1 y_1,x_2 y_2,…,x_n y_n

where $x_n = x_1$ and $y_n = y_1$

  - Turn in your code and a screenshot of the new shapefile being displayed

Projections

- Use SpatialReference objects
- Lots of different ways to specify projections
  - Well Known Text (WKT): http://en.wikipedia.org/wiki/Well-known_text
  - PROJ.4: www.remotesensing.org/geotiff/proj_list/
  - EPSG (European Petroleum Survey Group): see epsg file in your FWTools2.x.x/proj_lib directory
  - USGS: see importFromUSGS() description at www.gdal.org/ogr/classOGRSpatialReference.html
  - ESRI .prj (import only), PCI software, XML

ESRI .prj

```
PROJCS["WGS_1984_UTM_Zone_12N",GEOGCS["GCS_WGS_1984",DATUM["D_
WGS_1984",SPHEROID["WGS_1984",6378137,298.257223563]],PRIMEM["
Greenwich",0],UNIT["Degree",0.0174532925199433]],PROJECTION[
"Transverse_Mercator"],PARAMETER["False_Easting",500000],PARAM
ETER["False_Northing",0],PARAMETER["Central_Meridian",-
111],PARAMETER["Scale_Factor",0.9996],PARAMETER["Latitude_Of_O
rigin",0],UNIT["Meter",1]]
```

WKT

```
PROJCS["UTM Zone 12, Northern
Hemisphere",GEOGCS["WGS_1984",DATUM["WGS_1984",SPHEROID["WGS
84",6378137,298.2572235630016],TOWGS84[0,0,0,0,0,0,0]],PRIMEM[
"Greenwich",0],UNIT["degree",0.0174532925199433],AUTHORITY["EP
SG","4326"]],PROJECTION["Transverse_Mercator"],PARAMETER["lati
tude_of_origin",0],PARAMETER["central_meridian",-
111],PARAMETER["scale_factor",0.9996],PARAMETER["false_easting
",500000],PARAMETER["false_northing",0],UNIT["Meter",1],AUTHOR
ITY["EPSG","32612"]]
```

Pretty WKT

```
PROJCS["UTM Zone 12, Northern Hemisphere",
    GEOGCS["WGS_1984",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.2572235630016],
            TOWGS84[0,0,0,0,0,0,0]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.01745329251994433],
        AUTHORITY["EPSG","4326"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-111],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["Meter",1],
    AUTHORITY["EPSG","32612"]]
```

EPSG

```
32612
```

Proj.4

```
+proj=utm +zone=12 +ellps=WGS84 +datum=WGS84 +units=m +no_defs
```

USGS

```
(1, 12, (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0), 12)
```

PCI

```
('UTM    12   D000', 'METRE', (0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0))
```

XML

```
<gml:ProjectedCRS gml:id="ogrcrs13">
  <gml:srsName>UTM Zone 12, Northern Hemisphere</gml:srsName>
  <gml:srsID>
    <gml:name gml:codeSpace="urn:ogc:def:crs:EPSG::">32612</gml:name>
  </gml:srsID>
  <gml:baseCRS>
    <gml:GeographicCRS gml:id="ogrcrs14">
      <gml:srsName>WGS_1984</gml:srsName>
      <gml:srsID>
        <gml:name gml:codeSpace="urn:ogc:def:crs:EPSG::">4326</gml:name>
      </gml:srsID>
      <gml:usesEllipsoidalCS>
        <gml:EllipsoidalCS gml:id="ogrcrs15">
          <gml:csName>ellipsoidal</gml:csName>
          <gml:csID>
            <gml:name gml:codeSpace="urn:ogc:def:cs:EPSG::">6402</gml:name>
          </gml:csID>
          <gml:usesAxis>
            <gml:CoordinateSystemAxis gml:id="ogrcrs16" gml:uom="urn:ogc:def:uom:EPSG::9102">
              <gml:name>Geodetic latitude</gml:name>
              <gml:axisID>
                <gml:name gml:codeSpace="urn:ogc:def:axis:EPSG::">9901</gml:name>
              </gml:axisID>
              <gml:axisAbbrev>Lat</gml:axisAbbrev>
              <gml:axisDirection>north</gml:axisDirection>
            </gml:CoordinateSystemAxis>
          </gml:usesAxis>
          <gml:usesAxis>
            <gml:CoordinateSystemAxis gml:id="ogrcrs17" gml:uom="urn:ogc:def:uom:EPSG::9102">
              <gml:name>Geodetic longitude</gml:name>
              <gml:axisID>
                <gml:name gml:codeSpace="urn:ogc:def:axis:EPSG::">9902</gml:name>
              </gml:axisID>
              <gml:axisAbbrev>Lon</gml:axisAbbrev>
              <gml:axisDirection>east</gml:axisDirection>
            </gml:CoordinateSystemAxis>
          </gml:usesAxis>
        </gml:EllipsoidalCS>
      </gml:usesEllipsoidalCS>
      ... goes on and on and on ...
```

Getting a layer's projection

- If a data set has projection information stored with it (for example, a .prj file with a shapefile)

```
spatialRef = layer.GetSpatialRef()
```

- If no projection information with the data, then **`GetSpatialRef()`** returns **`None`**
- Can also get it from a geometry object:

```
spatialRef = geom.GetSpatialReference()
```

Creating a new projection

1. Import osr
2. Create an empty SpatialReference object with **`osr.SpatialReference()`**
3. Use one of the import methods (next slide) to import projection information into the SpatialReference object
4. See the osr documentation for ways to set parts of the projection individually

- **ImportFromWkt(<wkt>)**
- **ImportFromEPSG(<epsg>)**
- **ImportFromProj4(<proj4>)**
- **ImportFromESRI(<proj_lines>)**
- **ImportFromPCI(<proj>, <units>, <parms>)**
- **ImportFromUSGS(<proj_code>, <zone>)**
- **ImportFromXML(<xml>)**

```
import osr
```

or

```
from osgeo import osr

spatialRef = osr.SpatialReference()


spatialRef.ImportFromEPSG(32612)
```

or

```
spatialRef.ImportFromProj4('+proj=utm
  +zone=12 +ellps=WGS84 +datum=WGS84
  +units=m +no_defs')
```

Exporting a projection

- These methods will return strings
- **`ExportToWkt()`**
- **`ExportToPrettyWkt()`**
- **`ExportToProj4()`**
- **`ExportToPCI()`**
- **`ExportToUSGS()`**
- **`ExportToXML()`**

Projecting a geometry

1. Create a CoordinateTransformation
   1. Get the SpatialReference for the source
   2. Get the SpatialReference for the target
   3. Create the CoordinateTransformation with **`osr.CoordinateTransformation( <sourceSpatialRef>, <targetSpatialRef>)`**
2. Use **`Transform(<CoordTransform> )`** on the geometry object

```
sourceSR = osr.SpatialReference()
sourceSR.ImportFromEPSG(32612) #UTM 12N WGS84
targetSR = osr.SpatialReference()
targetSR.ImportFromEPSG(4326) #Geo WGS84
coordTrans =
  osr.CoordinateTransformation(sourceSR,
  targetSR)
geom.Transform(coordTrans)
```

More on projecting

- Modifies the geometry in place, which means you do not Destroy the geometry after transforming it
- In order to project an entire DataSource, you need to project one geometry at a time

```
>>> import ogr, osr, os
>>> os.chdir('d:/data/classes/python/os4')

>>> driver = ogr.GetDriverByName('ESRI Shapefile')
>>> dataset = driver.Open('ut_cities.shp')
>>> layer = dataset.GetLayer()
>>> feature = layer.GetNextFeature()
>>> geom = feature.GetGeometryRef()
>>> print geom.GetX(), geom.GetY()
-111.835875243 41.7399703435

>>> geoSR = osr.SpatialReference()
>>> geoSR.ImportFromEPSG(4326) # unprojected WGS84
0
>>> utmSR = osr.SpatialReference()
>>> utmSR.ImportFromEPSG(32612) # UTM 12N WGS84
0
>>> coordTrans = osr.CoordinateTransformation(geoSR, utmSR)
>>> geom.Transform(coordTrans)
0
>>> print geom.GetX(), geom.GetY()
430493.402455 4621243.63986
```

Creating an ESRI .prj file

1. Use **`MorphToESRI()`** on your output SpatialReference object (this modifies it in place)
2. Open a text file for writing – make sure it has the same name as the shapefile but with a .prj extension
3. Write out the string you get from using **`ExportToWkt()`** on the SpatialReference
4. Close the text file

- Assuming the shapefile is called test.shp:

```
targetSR.MorphToESRI()
file = open('test.prj', 'w')
file.write(targetSR.ExportToWkt())
file.close()
```

Assignment 2b

- Reproject a shapefile
  - Create a new polygon shapefile
  - Loop through the polygons in ut_counties.shp, reproject each one, and write it out to the new shapefile
  - Go from EPSG 4269 (unprojected NAD83) to EPSG 26912 (UTM 12N NAD83)
  - Turn in your code and a screenshot of the new shapefile being displayed