Intro to OGR

Intro to OGR

Overview

1. What is OGR?
2. Caveats and Limitations
3. Basic Object Model
4. Getting Started with OGR

What is OGR?

What is OGR?

- GDAL is the Geospatial Data Abstraction Library, a library of geospatial functions written in C/C++
- OGR is the part of GDAL for interfacing with features and geometries
- Think of GDAL as an io engine: it allows getting data into and out of your program
- OGR also has geometry objects which uses the GEOS library for computational gemoetry (allows basic vector analysis)
- Additional OSR side of GDAL handles projections and transformations
    - Uses the Proj.4 library (if you are concerned about projections and maintaining your data through transformations, please see the Proj.4 documentation for more information)

Caveats and Limitations

Caveats and Limitations

- More verbose than arcpy
    - Code is C++ wrapped in python to generate python bindings
    - Interface is like C++, so not pythonic at all
- Less abstract than arcpy
    - Working with data at a lower-level
    - Requires a better understanding of the file/data structures and operations
    - Basic things become difficult
- Does not have a "toolbox" from which one can pull highly-abstracted tools
    - Again, basic things become difficult

Caveats and Limitations

- Error messages can be cryptic, especially for python programmers
    - Often messages are C++ focused

```
Traceback (most recent call last):
  File "\\vmware-host\Shared Folders\Documents\Dropbox\School_Dropbox\Teaching\G
EOG410\Labs\Code\Lab5\lab5.py", line 337, in <module>
    sys.exit(main(**parse_arguments(sys.argv[1:])))
  File "\\vmware-host\Shared Folders\Documents\Dropbox\School_Dropbox\Teaching\G
EOG410\Labs\Code\Lab5\lab5.py", line 279, in main
    write_json_features_to_shp(selected_points, points_layer)
  File "\\vmware-host\Shared Folders\Documents\Dropbox\School_Dropbox\Teaching\G
EOG410\Labs\Code\Lab5\lab5.py", line 207, in write_json_features_to_shp
    feature['properties'][fieldname])
  File "C:\Anaconda\lib\site-packages\osgeo\ogr.py", line 2702, in SetField
    return _ogr.Feature_SetField(self, *args)
NotImplementedError: Wrong number or type of arguments for overloaded function '
Feature_SetField'.
  Possible C/C++ prototypes are:
    OGRFeatureShadow::SetField(int,char const *)
    OGRFeatureShadow::SetField(char const *,char const *)
    OGRFeatureShadow::SetField(int,int)
    OGRFeatureShadow::SetField(char const *,int)
    OGRFeatureShadow::SetField(int,double)
    OGRFeatureShadow::SetField(char const *,double)
    OGRFeatureShadow::SetField(int,int,int,int,int,int,int,int)
    OGRFeatureShadow::SetField(char const *,int,int,int,int,int,int,int)
```

Basic Object Model

Basic Object Model

- OGR is highly class-based
    - Requires an understanding of the class hierarchy
    - Classes use composition to a large degree
        - A layer "has-a" feature, a feature "has-a" field, etc
- Classes in C++ typically do not have accessible properties
    - This design affects how the python code works
    - Want to know the area of a polygon geometry object? Must use `GetArea()` method of the polygon

Basic Object Model

- Vector can be thought of via the following model:
    - Data source
        - Layer
            - Spatial Reference
            - Features
                - Attribute(s)
                - Geometry/Geometries
                    - Sub geometries…

Basic Object Model

- A Point geometry contains points
- A LineString geometry contains points
- A Polygon geometry contains LinearRing geometries
    - A LinearRing contains points
    - For shapefiles outer rings are clockwise and inner rings are counter-clockwise
    - Other applications may require rings be counter-clockwise: this can be a pretty big deal in computational geometry **BE AWARE**
- All of these can be "Multi": MultiPoint, MultiLineString, MultiPolygon
- A Geometry Collection has multiple geometries of any type

Basic Object Model

- The docs are a good guide
  - Harder to read than ArcGIS docs, but have the info
  - Relationships between objects are linked so traversing the hierarchy is typically easy
    - Do not need to memorize every little thing
  - Sometimes I cannot find something, so I use Google and it can

Getting Started with OGR

Gettign Started with OGR

- Use patterns! Look for examples and emulate what they do to fit your application.
    - GDAL Python Cookbook
    - Geoprocessing with python using Open Source GIS
- Read the python Gotchas
    - Many examples do not follow the advice of the Gotchas, so read it and internalize what is there so if you see bad examples, you can fix them when you use them in your own code

Getting Started with OGR

- OGR can be imported from the osgeo package:

```
from osgeo import ogr

# Note that you can import ogr directly, but that is depricated
```

Getting Started with OGR

- Drivers are used for opening and writing data in the formats supported by OGR (see the driver list)

- Need to get a driver to read or write data:

```
driver = ogr.GetDriverByName('ESRI Shapefile')

# open a data source as read-only
readonly = driver.Open(datapath)

# you can also open a data source for writing by providing True
# as a second argument to Open:
writable = driver.Open(datapath, True)

# if open fails, it will return None, so always
# check for that condition before proceeding
```

Getting Started with OGR

- Data sources always have a layer
    - Do shapefiles have layers?

Getting Started with OGR

- Data sources always have a layer
    - Do shapefiles have layers? Yes! Just one.

```
layer = readonly.GetLayer()  # or .GetLayer(0)


# a folder containing shapefiles can also be opened
# each shapefile within is considered a layer
workspace = driver.Open(path_to_data_dir)
a_shapefile_as_layer = workspace.GetLayer(2)

# but what index is each layer?
# instead use GetLayerByName()
# notice name does not have extension
a_shapefile_as_layer = workspace.GetLayerByName("points")
```

Getting Started with OGR

- Getting features in a layer

```
feature = a_layer.GetFeature(feature_index)
next_feature = a_layer.GetNextFeature()

# what if you want to do something to all features?
for feature_index in xrange(a_layer.GetFeatureCount()):
    a_layer.GetFeature(feature_index)

# could also use a while loop, getting next features each time
# iterating until gotten feature is None
```

Getting Started with OGR

- Features have fields:

```
field = feature.GetField(field_index)
# or
field = feature.GetField(field_name)

# fields of a feature are defined by a layer
# via a field definition object
layer_defn = a_layer.GetFieldDefn()

# need to iterate through fields?
# use the definition object
for field_index in xrange(layer_defn.GetFieldCount()):
    # we can get the field name from the field definition
    fieldname = layer_defn.GetFieldDefn(field_index).GetName()

    # we can also get the values of fields
    feature.GetFieldAsString(field_index)
    feature.GetFieldAsInteger(field_name)
```

Getting Started with OGR

- Want the geometry of a feature?

```
geom = feature.GetGeometryRef()
```

Getting Started with OGR

- To properly close something, need to set it to None
- This also writes any changes to the data source
- Examples commonly have `object.Destroy()`, but this is not good practice and can crash python if used improperly

```
feature = None
a_layer = None
workspace = None
```

Getting Started with OGR

- Writing data can be done with an existing file or a new file

```
# edit existing data
writable = driver.Open(path_to_existing_dataset, True)

# create new data
new_file = driver.CreateDataSource(path_to_nonexistant_data)
# need to create a new layer
new_layer = new_file.CreateLayer(layer_name,
                                 geom_type=ogr.wkbPoint,
                                 srs=spatial_reference)

# before creating a data souce, can check for existance:
if os.path.exists(path_to_shapefile):
    # this will delete all the parts of the file for you
    driver.DeleteDataSource(path_to_shapefile)
```

Getting Started with OGR

- Add fields to a layer (note: fields cannot be added to a layer that already contains features)

```
# create a new field
field_defn = ogr.FieldDefn(field_name, ogr.OFTString)

# string fields have a width
field_defn.SetWidth(25)

# add field to layer
layer.CreateField(field_defn)

# want to copy a field def from an existing layer?
layer_defn = original_layer.GetLayerDefn()
field_defn = layer_defn.GetFieldDefn(field_name_or_index)
new_layer.CreateField(field_defn)
```

Getting Started with OGR

- Now you probably want to create features:

```
# need a layer definition after all fields
# have beeen added to layer
layer_defn = layer.GetLayerDefn()

# create a new feature instance using the layer defn
feature = ogr.Feature(layer_defn)

# use an existing geometry for feature geom
feature.SetGeometry(point_geom)

# set all your fields
feature.SetField(field_name_or_id, value)

# write the feature to the layer
layer.CreateFeature(feature)

# be sure to set everything to None to write the changes
# order is important!
feature = None
layer = None
new_file = None
```