Creating Aggregate Rasters for MapServer or GDAL

MapServer tileindex and GDAL VRT

Tileindexes

Although the gdaltindex utility meets the needs of most users, creating a tileindex shapefile is a good introduction to gdal.py. It can also be useful to have a tileindex file with more attributes for reuse in your map.

os.path

The os.path module implements functions on pathnames. Create a new text file in your working directory named `hobu.txt`. No contents are needed. We'll use this file to explore `os.path`.

The `abspath` function returns the absolute path given a relative path.

```
>>> import os.path
>>> os.path.abspath('./hobu.txt')
'P:\\OSG05\\aggregation\\hobu.txt'
>>>
```

The `basename` function returns the filename with all directories stripped from the path.

```
>>> os.path.basename('P:\OSG05\aggregation\hobu.txt')
'hobu.txt'
>>>
```

the `getctime` function returns the file creation time in seconds past the epoch

```
>>> os.path.getctime('hobu.txt')
1118386365
>>>
```

glob

Just like a shell glob. `glob.glob` returns a possibly empty list of paths that match the input pattern:

```
>>> import glob
>>> glob.glob('*.txt')
['hobu.txt']
```

```
>>>
```

Putting it together

Now we'll combine these to print information about a batch of files:

```
>>> for path in glob.glob('*.txt'):
...     print os.path.basename(path), \
...           os.path.abspath(path), \
...           os.path.getctime(path)
...
hobu.txt P:\OSG05\aggregation\hobu.txt 1118386365
>>>
```

And now we'll try this on the workshop raster data. Replace the pattern below with the path to the workshop data:

```
>>> paths = glob.glob('P:\OSG05\python-tests\data\*.tif')
>>> for path in paths:
...     print os.path.basename(path), \
...           os.path.abspath(path), \
...           os.path.getctime(path)
...
escalante30_zip.tif P:\OSG05\python-
tests\data\escalante30_zip.tif 1044213876
mtnwest_zip.tif P:\OSG05\python-tests\data\mtnwest_zip.tif
1044212332
waterpocket30_zip.tif P:\OSG05\python-
tests\data\waterpocket30_zip.tif 104421366
6
zion30_zip.tif P:\OSG05\python-tests\data\zion30_zip.tif
1044211340
cameron30_zip.tif P:\OSG05\python-
tests\data\cameron30_zip.tif 1044129070
wasatch30_zip.tif P:\OSG05\python-
tests\data\wasatch30_zip.tif 1044129100
>>>
```

gdal

OK, so we can obtain all kinds of OS info about the raster data. Now we'll get to the the important geo properties using GDAL's `gdal` Python module.

In the following steps, don't bother with typing the paths. Type the leading quotation mark, drag the file from the file explorer to the interpreter, and the close the quotes.

Let's open one of the workshop raster files in the default read-only mode:

```
>>> from gdal import gdal
>>> dataset = gdal.Open('P:\OSG05\python-
tests\data\cameron30_zip.tif')
>>> dataset
<gdal.gdal.Dataset instance at 0x008E48C8>
>>>
```

The gdal module is extensive. In this exercise we're going to limit ourselves to the following attributes of a Dataset:

```
>>> dataset.RasterCount
3
>>> dataset.RasterXSize
999
>>> dataset.RasterYSize
1586
>>> dataset.GetGeoTransform()
(-106.05969999999999, 0.00027777777777799998, 0.0,
40.842500000000001, 0.0, -0.0
0027769230769199998)
>>>
```

These are the number of bands, the number of pixels and lines, and the geo transform parameters. The elements at indexes 0 and 1 of this tuple are the upper left x value and the x pixel size. The elements at indexes 3 and 5 are the upper left y value and -1 times the y pixel size.

Let's use these properties and methods to compute the bounding boxes for our raster data files:

```
>>> paths = glob.glob('P:\OSG05\python-tests\data\*.tif'):
>>> for path in paths:
...     ds = gdal.Open(path)
...     geo = ds.GetGeoTransform()
...     pixels = ds.RasterXSize
...     lines = ds.RasterYSize
...     minx = geo[0]
...     maxx = minx + pixels * geo[1]
...     maxy = geo[3]
...     miny = maxy + lines * geo[5]
...     print os.path.basename(path), (minx, miny, maxx,
maxy)
...
escalante30_zip.tif (-111.705, 37.686388888056207,
-111.22944443999971, 38.06583
3333055551)
mtnwest_zip.tif (-115.5, 36.50000000000022,
-103.50000000000048, 42.0)
waterpocket30_zip.tif (-111.28472222194445,
37.298055554999578, -110.72666665999
982, 38.340000000000003)
zion30_zip.tif (-113.21111111, 37.106111111111382,
-112.74444443999984, 37.63166
6666111109)
cameron30_zip.tif (-106.05969999999999,
40.402080000000488, -105.78219999999978,
 40.842500000000001)
wasatch30_zip.tif (-111.85889999999999, 40.38999999999951,
-111.40639999999964,
40.77028)
>>>
```

There's no `close` method for a GDAL dataset. The dataset is closed at the end of the interior block above when Python's garbage collection sweeps out the local `ds` object. You might want to be explicit about it, appending

```
...     del ds
```

to the end of the block.

ogr

That's all we need from `gdal.py` in order to create our raster tileindex. Now we'll need to learn to create an output vector dataset and push features into it. Here, in a nutshell, is creation and saving of a polygon type shapefile using GDAL's `ogr.py` module:

```
>>> from gdal import ogr
>>> driver = ogr.GetDriverByName('ESRI Shapefile')
>>> tileindex_shp = driver.CreateDataSource
('tileindex.shp')
>>> tileindex = tileindex_shp.CreateLayer('tileindex',
geom_type=ogr.wkbPolygon)
>>> tileindex_shp.Destroy()
>>>
```

The Destroy method is more bark than bite. It doesn't delete the file on disk, just closes the output stream and releases allocated memory. Look in your working directory and you will find a shapefile – a rather pointless shapefile with no records, no fields.

Shapefile fields

Let's address that now. Delete the three shapefile components, and repeat the following lines. Try using your interpreter's command history.

```
>>> tileindex_shp = driver.CreateDataSource
('tileindex.shp')
>>> tileindex = tileindex_shp.CreateLayer('tileindex',
geom_type=ogr.wkbPolygon)
```

Next we'll define a string type field named 'location' and set its width to 200 characters:

```
>>> field = ogr.FieldDefn('location', ogr.OFTString)
>>> field.SetWidth(200)
```

and add this field to the layer

```
>>> tileindex.CreateField(field)
0
```

we'll leave the data source open.

Adding Features

A record in our shapefile layer is represented by ogr's `Feature` class. The constructor requires a `FieldDefn` argument, and we obtain one from the layer itself. The value of our single 'location' field is set using the feature's SetField method. Note the return of the `abspath` function and our `hobu.txt` file.

```
>>> feature = ogr.Feature(tileindex.GetLayerDefn())
>>> feature.SetField(0, os.path.abspath('hobu.txt'))
```

A complete feature needs a geometry. We won't dive too deep into ogr.Geometry yet, but will use Python's string interpolation to hack a WKT (well-known text) string and exploit ogr's WKT geometry factory. This time we are using a Python mapping as the object of the interpolation operator instead of a tuple as we did earlier:

```
>>> wkt = 'POLYGON ((%(minx)f %(miny)f, %(minx)f %(maxy)f,
%(maxx)f %(maxy)f, %(maxx)f %(miny)f, %(minx)f %(miny)f))'
>>> wkt = wkt % {'minx': -10, 'miny': -10, 'maxx': 10,
'maxy': 10}
>>> wkt
'POLYGON ((-10.000000 -10.000000, -10.000000 10.000000,
10.000000 10.000000, 10.000000 -10.000000, -10.000000
-10.000000))'
```

Next we create an ogr.Geometry from this string and set the feature's geometry from it:

```
>>> geom = ogr.CreateGeometryFromWkt(wkt)
>>> feature.SetGeometryDirectly(geom)
0
```

create a new feature in our layer based upon this one, and close the data source.

```
>>> tileindex.CreateFeature(feature)
0
>>> tileindex_shp.Destroy()
```

Open the shapefile in OpenEV to see the results.

Aside for mapscript users

The `mapscript.pointObj` and `mapscript.rectObj` classes each have magic methods to support Python's built in `str()` function. Give these a quick try:

```
>>> from mapscript import mapscript
```

```
>>> p = mapscript.pointObj(1, 2)
>>> str(p)
"{ 'x': 1 , 'y': 2, 'z': 0 }"
>>> r = mapscript.rectObj(-10,-10,10,10)
>>> str(r)
"{ 'minx': -10 , 'miny': -10 , 'maxx': 10 , 'maxy': 10 }"
>>>
```

Hey, what do you know? Looks a lot like a Python dict, and with the help of the built in `eval()` function, we can turn it into a dict and interpolate the values into a WKT string:

```
>>> wkt = 'POLYGON ((%(minx)f %(miny)f, %(minx)f %(maxy)f,
%(maxx)f %(maxy)f, %(maxx)f %(miny)f, %(minx)f %(miny)f))'
>>> wkt = wkt % eval(str(r))
>>> wkt
'POLYGON ((-10.000000 -10.000000, -10.000000 10.000000,
10.000000 10.000000, 10.000000 -10.000000, -10.000000
-10.000000))'
>>>
```

Complete tileindex script

A complete tileindexing script is included in the workshop at c:/ms4w/apps/python/aggregation/aggtindex.py and can be run using the accompanying aggregation.bat file. Aim it at the workshop raster files in c:/ms4w/apps/python/python/data and check the results again in OpenEV.

Virtual Datasets

GDAL's virtual dataset, or VRT, driver is a means of (among other things) aggregating raster data. The document at http://www.gdal.org/gdal_vrttut.html describes how to express a virtual dataset using XML. We're going to create a VRT that aggregates the workshop raster files, allowing them to be visualized or processed as if they were a single dataset.

XML and Elementtree

Python has a standard XML library, and a great range of other available libraries for parsing and writing XML. The `elementtree` package

http://effbot.org/zone/element-index.htm

is a good match for VRT's lightweight XML.

Here's a very simple example that's easy to type in the interpreter:

```
>>> from elementtree.ElementTree import Element,
SubElement
>>> html = Element('html')
>>> body = SubElement(html, 'body')
>>> heading = SubElement(body, 'h1')
>>> heading.text = 'Introducing ElementTree'
>>> para = SubElement(body, 'p')
>>> para.text = 'Package for manipulating hierarchical
data'
```

Now let's import the tostring function so that we can see how this is encoded:

```
>>> from elementtree.ElementTree import tostring
>>> tostring(html)
'<html><body><h1>Introducing ElementTree</h1><p>Package
for manipulating hierarchical data</p></body></html>'
```

On second thought, let's add some CSS to demonstrate element attributes:

```
>>> head = SubElement(html, 'head')
>>> style = SubElement(head, 'style')
>>> style.attrib['type'] = 'text/css'
>>> style.text = 'H1{color:red} P{color:blue}'
>>> from elementtree.ElementTree import tostring
>>> tostring(html)
'<html><body><h1>Introducing ElementTree</h1><p>Package
for manipulating hierarchical data</p></body><head><style
type="text/css">H1{color:red} P{color:blue}
</style></head></html>'
```

and then use the `ElementTree` class to write this to a file

```
>>> from elementtree.ElementTree import ElementTree
>>> tree = ElementTree(html)
>>> tree.write('example.html')
```

Open example.html in a web browser. Minus the standard preamble, it's XHTML, and easy to generate using `elementtree`.

Easy VRT

For a first example, we're going to quickly create a VRT that simply proxies a single band of one of our workshop rasters much like in the first example on the VRT tutorial page.

```
>>> from gdal import gdal
>>> ds = gdal.Open
(r'c:\ms4w\python\data\wasatch30_zip.tif')
>>> geo = ds.GetGeoTransform()
>>> pixels = ds.RasterXSize
>>> lines = ds.RasterYSize
```

You could print the values of these if you wanted. That's all we need from `gdal`, and now we begin by creating our top level element:

```
>>> vrt_elem = Element('VRTDataset',
                       rasterXSize=str(pixels),
                       rasterYSize=str(lines))
```

Note that all `Element` attributes must be strings. Next we add a GeoTransform `SubElement` and set its text node to a string representation of the raster dataset's geotransform.

```
>>> geo_elem = SubElement(vrt_elem, 'GeoTransform')
>>> geo_elem.text = '%f, %f, %f, %f, %f, %f' % (geo)
```

Next we'll add a band element to the root

```
>>> band_elem = SubElement(vrt_elem, 'VRTRasterBand',
dataType='Byte', band='1')
```

and then take a preview of our VRT under construction

```
>>> tostring(vrt_elem)
'<VRTDataset rasterXSize="1629"
rasterYSize="1369"><GeoTransform>-111.858900, 0.000278,
0.000000, 40.770280, 0.000000,
-0.000278<GeoTransform><VRTRasterBand band="1"
dataType="Byte" /></VRTDataset>'
```

Only thing left to do is to define the source data for the band. This involves several new levels of sub elements. Take care that they are subbed from the proper parent element. If you mistakenly insert an element into another, you can take advantage of the fact that all Elements are list-like and delete the sub element at a certain index.

```
>>> source_elem = SubElement(band_elem, 'SimpleSource')
```

```
>>> filename_elem = SubElement(source_elem,
'SourceFilename', relativeToVRT='0')
>>> filename_elem.text =
r'c:\ms4w\python\data\wasatch30_zip.tif'
>>> sband_elem = SubElement(source_elem, 'SourceBand')
>>> sband_elem.text = '1'
>>> srect_elem = SubElement(source_elem, 'SrcRect',
xOff='0', yOff='0', xSize=str(pixels), ySize=str(lines))
>>> drect_elem = SubElement(source_elem, 'DstRect',
xOff='0', yOff='0', xSize=str(pixels), ySize=str(lines))
```

Now let's wrap this up in an ElementTree and write it to disk.

```
>>> vrttree = ElementTree(vrt_elem)
>>> vrttree.write('first.vrt')
```

This first.vrt file can be opened in OpenEV. You should see a gray scale image of the Wasatch Range centered roughly on the Alta ski area at the head of Little Cottonwood Canyon.

Further VRT Element Hacking

A handy feature is that our elements are entirely mutable. Set the source band to "2" and write to a new file

```
>>> sband_elem.text = '2'
>>> vrttree.write('second.vrt')
```

repeat for the third band

```
>>> sband_elem.text = '3'
>>> vrttree.write('third.vrt')
```

Raster hackers might find this a good way to tweak pixel scaling, color tables, or even filter kernels. See http://www.gdal.org/gdal_vrttut.html for more VRT options.

Complete VRT Script

Finally, we return to the objective: a VRT that aggregates source rasters of a single class (same band count, same projection, and pixel resolution). It's not much more involved than our previous example. The VRT raster size and extents are expanded as each input raster is read, and the individual raster data is mapped into the aggregate output by calculating the appropriate destination rectangle.

The completed script is at c:/ms4w/apps/python/aggregation/aggvrt.py and can be run using the accompanying aggregation.bat file. Aim it at the 5 workshop raster files matching the pattern c:/ms4w/apps/python/python/data/*30*.tif , redirect the output to a .vrt file and check the results again in OpenEV. You should see results like this