More GDAL

More GDAL

Overview

1. Review Reading Raster Data
2. Writing Raster Data
3. Processing Data

Reading Raster Data

Reading Raster Data

- Open a dataset
- Get desired band
- Read band as array
    - whole or part

```python
from osgeo import gdal

dataset = gdal.Open(r"C:\data\rasters\landcover.img")

# Open() will return None if fails to open data at path
if not dataset:
    raise Exception("Failed to read specified dataset {}".format(dataset))

band = dataset.GetRasterBand(1)
# could check to see if band is None to make sure this worked

# we can read the whole band into an array
array = band.ReadAsArray(0, 0, dataset.RasterXSize, dataset.RasterYSize)
```

Reading Raster Data

- For many applications reading whole band can be effective
    - But it is memory intensive!

Reading Raster Data

- Best general-case solution is to read in image by blocks
    - We can get block size from the band before reading

```
# get the blocksize from the band
# returns a list of len 2, [0] is # of cols in block and [1] is # of rows
blocksize = band.GetBlockSize()

rows = dataset.RasterYSize
cols = dataset.RasterXSize

# iterate through rows of blocks
for row in xrange(0, rows, blocksize[1]):
    if row + blocksize[1] < rows:
        row_read_size = blocksize[1]
    else:
        row_read_size = rows - row

    # iterate through columns of blocks
    for col in xrange(0, cols, blocksize[0]):
        if col + blocksize[0] < cols:
            col_read_size = blocksize[0]
        else:
            col_read_size = cols - col

        data = band.ReadAsArray(col, row, col_read_size, row_read_size)
        # do something with the block data here
```

Writing Raster Data

Writing Raster Data

- To write data we need a writable output raster

    1. We can open a raster with update = True, providing write access

```
# open a raster with update
writable_dataset = gdal.Open(r"C:\data\rasters\landcover.img", gdal.GA_Update)
# could also just do gdal.Open(path, True)
```

Writing Raster Data

- To write data we need a writable output raster

  1. We can open a raster with update = True, providing write access

  2. We can create a brand new raster

```
# first need to explicitly register a driver so GDAL knows the format to use
driver = gdal.GetDriverByName("HDA")  # HDA is ERDAS Imagine .img format
driver.Register()

# note: we could also get the driver from an existing opened dataset
indriver = dataset.GetDriver()

# now we can create a dataset
cols, rows, bands = 1000, 500, 1
datatype = gdal.GDT_UInt32
created = driver.Create(r"C:\data\rasters\new.img", cols, rows, bands, datatype)

# our created dataset exists but is not georeferenced
# we need to use the SetGeoTransform(gt) and
# SetProjection(osr.SpatialReference) methods
```

Writing Raster Data

- GDAL supports the follwing datatypes (but some drivers may not):
    - GDT_Byte = 1
    - GDT_CFloat32 = 10
    - GDT_CFloat64 = 11
    - GDT_CInt16 = 8
    - GDT_CInt32 = 9
    - GDT_Float32 = 6
    - GDT_Float64 = 7
    - GDT_Int16 = 3
    - GDT_Int32 = 5
    - GDT_TypeCount = 12
    - GDT_UInt16 = 2
    - GDT_UInt32 = 4
    - GDT_Unknown = 0

Writing Raster Data

- Not all drivers support writing or copying data
    - See the driver list
- We can programmatically check a driver for support:

```
drivername = "MEM"  # "MEM" is the driver to create a dataset in memory
driver = gdal.GetDriverByName(format)

# get the driver metadata
metadata = driver.GetMetadata()

# see if driver has Create() support
if metadata.has_key(gdal.DCAP_CREATE) and metadata[gdal.DCAP_CREATE] == 'YES':
    print 'Driver {} supports Create() method.'.format(drivername)

# see if driver has CreateCopy() support
if metadata.has_key(gdal.DCAP_CREATECOPY) and \
                metadata[gdal.DCAP_CREATECOPY] == 'YES':
    print 'Driver {} supports CreateCopy() method.'.format(drivername)
```

Writing Raster Data

- To write data we need a writable output raster

  1. We can open a raster with update = True, providing write access

  2. We can create a brand new raster

  3. We can make a copy of an existing raster

```
# again, need to get and register a driver
driver = gdal.GetDriverByName("ENVI")
driver.Register()

# now we can copy an existing opened dataset
strict = True
copied_dataset = driver.CreateCopy(dataset, r"C:\data\rasters\copy.tif", strict)
```

Writing Raster Data

- With a writeable dataset, writing data is simply a matter of getting a band and writing an array via the method `WriteArray(array, xoff=0, yoff=0)`

```
import numpy

rows, cols = 354, 455

# create an array
array = numpy.zeros((rows, cols)), dtype=numpy.uint32)

# register a driver
driver = driver.GetDriverByName("MEM")
driver.Register()

# create output raster dataset
output_ds = driver.Create("", cols, rows, 1, gdal.GDT_UInt32)

# get band
band = output_ds.GetRasterBand(1)

# write array to band and deallocate array memory
band.WriteArray(array)
array = None

# close everything
band = None
output_ds = None
```

Writing Raster Data

- Set a NoData value on a band using `SetNoDataValue(value)`
    - Read the NoData value with `GetNoDataValue()`
- Caluculate statistics on a band via `ComputeStatistics(approx_ok)`
    - returns list of min, max, mean, and stdev
- Get just min or max from `GetMinimum()` and `GetMaximum()`
- Get histogram using `GetHistogram(min=-0.5, max=255.5, buckets=256)`
- Add pyramids to a band with `outDataset.BuildOverviews(overviewlist=[2,4, 8,16,32,64,128])`
- HOWEVER, if you want to do something with a band after running `WriteArray()`, you must use `FlushCache()` to flush the data to disk
- Note we could also have done the previous example using `Fill(value)`
- See Band class reference for more methods and details

Writing Raster Data

What about writing data in blocks?

```
rows, cols = in_dataset.RasterYSize, in_dataset.RasterXSize

inband = in_dataset.GetRasterBand(1)
outband = new_dataset.GetRasterBand(1)

blocksize = inband.GetBlockSize()

for row in xrange(0, rows, blocksize[1]):
    if row + blocksize[1] < rows:
        row_read_size = blocksize[1]
    else:
        row_read_size = rows

    for col in xrange(0, cols, blocksize[0]):
        if col + blocksize[0] < cols:
            col_read_size = blocksize[0]
        else:
            col_read_size = cols - col

        data = band.ReadAsArray(col, row, col_read_size, row_read_size)

        # do something with data to generate out_array

        outband.WriteArray(out_array, col, row)
```

Processing Data

Processing Data

- Once data is read into an array, we can use numpy functions or other packages that support numpy arrays to process the data
- numpy arrays make conditional statements/comparisons, mathematical operations, and logic easy
    - Works just like map algebra in arcpy (perhaps a bit more picky about parentheses)

Processing Data

- Like `arcpy.sa.Con()`? Try `numpy.where(condition, if_true, if_false)`
- `where()` without if_true and if_false will return a tuple of indicies for all cells meeting the condition

```
>>> array = numpy.arange(10)

>>> print array
[0 1 2 3 4 5 6 7 8 9]

>>> print numpy.where(array % 2 == 0, 50, 1)
[50  1 50  1 50  1 50  1 50  1]

>>> print numpy.where(array ** 2 < 50)
(array([0, 1, 2, 3, 4, 5, 6, 7]),)
```

Processing Data

- Another key syntax is conditional indexing

```
>>> array = numpy.arange(10)

>>> array[array > 3] = 12

>>> print array
[ 0  1  2  3 12 12 12 12 12 12]

# conditional array can be different array
>>> other_array = numpy.zeros_like(array)

>>> other_array[array == 12] = -99

>>> print other_array
[  0   0   0   0 -99 -99 -99 -99 -99 -99]

# conditional array can be smaller than the array on which it is operating
# and can be larger ONLY IF the condition does not provide an index out of
# bounds on the array on which it is operating
```

Processing Data

- Want to find all cells matching a range of values? Use `numpy.in1d()`

```
>>> array = numpy.array([[1, 6, 7, 4, 5],
...                      [3, 5, 6, 6, 4],
...                      [7, 5, 5, 1, 2]])

>>> values = [1, 4, 7]

>>> print numpy.in1d(array.ravel(), values).reshape(array.shape)
[[ True False  True  True False]
 [False False False False  True]
 [ True False False  True False]]
```

Processing Data

- Have a multiband raster? Want to keep the bands together? Make a 3-dimensional array!

```
dataset = gdal.Open(r"C:\data\rasters\bgrimage.img")

array = None
for band in xrange(dataset.RasterCount):
    if not array:
        array = dataset.GetRasterBand(band + 1).ReadAsArray()
    else:
        array = array.dstack((array, dataset.GetRasterBand(band + 1).ReadAsArray()))

# now by using subscripting we can access a pixel in array
# get upper left pixel
array[0, 0]

# want to use subscripting to get a band instead of a pixel?
# need to transpose
array = array.transpose(2, 0, 1)

# get band 3
array[2]

# now switch back?
```