Reading Raster Data with GDAL

Open Source RS/GIS Python
Week 4
GDAL

• Supports about 100 raster formats
  • ArcInfo grids, ArcSDE raster, Imagine, Idrisi, ENVI, GRASS, GeoTIFF
  • HDF4, HDF5
  • USGS DOQ, USGS DEM
  • ECW, MrSID
  • TIFF, JPEG, JPEG2000, PNG, GIF, BMP
• See http://www.gdal.org/formats_list.html
Finding available formats

• To see what formats are compiled into your version of GDAL, use this command in the FWTools shell (or terminal window on a Mac)

   gdalinfo --formats
Importing GDAL

• Need to import both gdal and gdalconst
• FWTools:

```python
import gdal, gdalconst
```

• Not FWTools:

```python
from osgeo import gdal, gdalconst
```
• All gdalconst constants start with a prefix which minimizes the possibility of conflicts with other modules
• Can import a module so you don’t have to prefix things with the module name:

```python
import gdal
from gdalconst import *
```

Or

```python
from osgeo import gdal
from osgeo.gdalconst import *
```
GDAL data drivers

- Similar to OGR data drivers
- Need to register a driver before using it
- Need to have a driver object before creating a new raster data set
- Driver names (code) are available at http://www.gdal.org/formats_list.html
• Register all drivers at once
  • Works for reading data but not for creating data sets

```python
gdal.AllRegister()
```

• Get the Imagine driver and register it
  • Works for reading and creating new Imagine files

```python
driver = gdal.GetDriverByName('HFA')
driver.Register()
```
Opening a raster data set

• Once the driver has been registered, the `Open(<filename>, <GDALAccess>)` method can be used to return a Dataset object

```python
fn = 'aster.img'

ds = gdal.Open(fn, GA_ReadOnly)
if ds is None:
    print 'Could not open ' + fn
    sys.exit(1)
```
Getting image dimensions

• Dataset objects have properties corresponding to numbers of rows, columns and bands in the data set

```
cols = ds.RasterXSize
rows = ds.RasterYSize
bands = ds.RasterCount
```

• Notice no parentheses – because they’re properties not methods
Getting georeference info

• GeoTransforms are lists of information used to georeference an image

• From the GDAL documentation:

```
adfGeoTransform[0] /* top left x */
adfGeoTransform[1] /* w-e pixel resolution */
adfGeoTransform[2] /* rotation, 0 if image is "north up" */
adfGeoTransform[3] /* top left y */
adfGeoTransform[4] /* rotation, 0 if image is "north up" */
adfGeoTransform[5] /* n-s pixel resolution */
```

• Coordinates are for top left corners of pixels (unlike Imagine, which uses centers)
• Use the `GetGeoTransform()` method on a Dataset object to get a GeoTransform

```python
g earth = ds.GetGeoTransform()
originX = geotransform[0]
originY = geotransform[3]
pixelWidth = geotransform[1]
pixelHeight = geotransform[5]
```

```python
adfGeoTransform[0]  # top left x  
adfGeoTransform[1]  # w-e pixel resolution  
adfGeoTransform[2]  # rotation, 0 if image is "north up" 
adfGeoTransform[3]  # top left y  
adfGeoTransform[4]  # rotation, 0 if image is "north up" 
adfGeoTransform[5]  # n-s pixel resolution  
```
Computing pixel offsets

- Need to get pixel offsets from the upper left corner for specific coordinates \( x, y \)

\[
\begin{align*}
x\text{Offset} &= \text{int}\left( \frac{x - \text{originX}}{\text{pixelWidth}} \right) \\
y\text{Offset} &= \text{int}\left( \frac{y - \text{originY}}{\text{pixelHeight}} \right)
\end{align*}
\]

\[(\text{originX, originY}) \quad \text{pixelWidth} \quad x \quad \text{pixelHeight} \quad y\]

\[
\begin{align*}
\frac{x - \text{originX}}{\text{pixelWidth}} &\approx 3.25 \\
\frac{y - \text{originY}}{\text{pixelHeight}} &\approx 2.5
\end{align*}
\]

Take the integer values and we get \((3,2)\)
Getting individual pixel values

• Get the Band object by passing the band index (1-based) to the Dataset’s `GetRasterBand(<index>)` method

```python
band = ds.GetRasterBand(1)
```

• Read the data into a 2D Numeric array with `ReadAsArray(<xoff>, <yoff>, <xsize>, <ysize>)`

```python
data = band.ReadAsArray(xOffset, yOffset, 1, 1)
```
• Even though we only read one pixel value, it is in a two-dimensional array
• Since we read one pixel in each direction, the array is of size 1x1
• Need to specify both offsets, which are 0 in this case

\[
\text{value} = \text{data}[0,0]
\]
Reading an entire image at once

- Use 0 offsets and pass the numbers of rows and columns to the `ReadAsArray()` method
  
  \[
  \text{data} = \text{band.ReadAsArray}(0, 0, \text{cols}, \text{rows})
  \]

- Read individual pixels using \([yoff, xoff]\) (math matrix notation is \([\text{row}, \text{col}]\), not \([x,y]\))

- To read the pixel in the 95\textsuperscript{th} column and 43\textsuperscript{rd} row:

  \[
  \text{value} = \text{data}[42, 94]
  \]
Memory management

- Set variables to None
- Especially important if you created large arrays with `ReadAsArray()`

```python
band = None
dataset = None
```
# script to get pixel values at a set of coordinates
# by reading in one pixel at a time
# Took 0.47 seconds on my machine

import os, sys, time, gdal
from gdalconst import *

# start timing
startTime = time.time()

# coordinates to get pixel values for
xValues = [447520.0, 432524.0, 451503.0]
yValues = [4631976.0, 4608827.0, 4648114.0]

# set directory
os.chdir(r'Z:\Data\Classes\Python\data')

# register all of the drivers
gdal.AllRegister()

# open the image
ds = gdal.Open('aster.img', GA_ReadOnly)
if ds is None:
    print 'Could not open image'
sys.exit(1)
# get image size
rows = ds.RasterYSize
cols = ds.RasterXSize
bands = ds.RasterCount

# get georeference info
transform = ds.GetGeoTransform()
xOrigin = transform[0]
yOrigin = transform[3]
pixelWidth = transform[1]
pixelHeight = transform[5]

# loop through the coordinates
for i in range(3):
    # get x,y
    x = xValues[i]
y = yValues[i]

    # compute pixel offset
    xOffset = int((x - xOrigin) / pixelWidth)
yOffset = int((y - yOrigin) / pixelHeight)

    # create a string to print out
    s = str(x) + ' ' + str(y) + ' ' + str(xOffset) + ' ' + str(yOffset) + ' ' + str(band)

    # loop through the bands
    for j in range(bands):
        band = ds.GetRasterBand(j+1) # 1-based index
# read data and add the value to the string
data = band.ReadAsArray(xOffset, yOffset, 1, 1)
value = data[0,0]
s = s + str(value) + ' '  

# print out the data string
print s

# figure out how long the script took to run
endTime = time.time()
print 'The script took ' + str(endTime - startTime) + ' seconds'
Script example 2

# script to get pixel values at a set of coordinates
# by reading in entire bands
# Took 1.69 seconds on my machine

import os, sys, time, gdal
from gdalconst import *

# start timing
startTime = time.time()

# coordinates to get pixel values for
xValues = [447520.0, 432524.0, 451503.0]
yValues = [4631976.0, 4608827.0, 4648114.0]

# set directory
os.chdir(r'Z:\Data\Classes\Python\data')

# register all of the drivers
gdal.AllRegister()

# open the image
ds = gdal.Open('aster.img', GA_ReadOnly)
if ds is None:
    print 'Could not open image'
sys.exit(1)
# get image size
rows = ds.RasterYSize
cols = ds.RasterXSize
bands = ds.RasterCount

# get georeference info
transform = ds.GetGeoTransform()
xOrigin = transform[0]
yOrigin = transform[3]
pixelWidth = transform[1]
pixelHeight = transform[5]

# create a list to store band data in
bandList = []

# read in bands and store all the data in bandList
for i in range(bands):
    band = ds.GetRasterBand(i+1)
    data = band.ReadAsArray(0, 0, cols, rows)
    bandList.append(data)

# loop through the coordinates
for i in range(3):
    # get x,y
    x = xValues[i]
    y = yValues[i]
# compute pixel offset
xOffset = int((x - xOrigin) / pixelWidth)
yOffset = int((y - yOrigin) / pixelHeight)

# create a string to print out
s = str(x) + ' ' + str(y) + ' ' + str(xOffset) + ' ' + str(yOffset) + ' \

# loop through the bands and get the pixel value
for j in range(bands):
    data = bandList[j]
    value = data[yOffset, xOffset] # math matrix notation order
    s = s + str(value) + ' \

# print out the data string
print s

# figure out how long the script took to run
endTime = time.time()
print 'The script took ' + str(endTime - startTime) + ' seconds'
Assignment 4a

• Read pixel values from an image
  • Print out the pixel values for all three bands of aster.img at the points contained in sites.shp
  • Use any method of reading the raster data that you want, but I would suggest one pixel at a time (fastest in this case since we don't need much data)
  • Turn in your code and the output (right-click in the Crimson Editor output window to copy all output)
Reading raster data efficiently

• Reading one pixel at a time is about as inefficient as you can get (DON’T DO IT unless you just need a few pixel values here and there)

• Reading the entire image at once is pretty efficient, but not the best
  • Plus, you might not have enough RAM to hold it all or process it
• Anyone seen the Block Size information in Erdas Imagine?
• Has to do with how the values are stored on disk
• Most efficient way to access raster data is by blocks
• Unfortunately, don’t always know block size
Getting block size

- This week’s data has a module called utils
- Can use it to get block size like this:

```python
import utils
blockSize = utils.GetBlockSize(band)
xBlockSize = blockSize[0]
yBlockSize = blockSize[1]
```
Tiled images

- Some file types, like most GeoTIFFs, are not tiled
  - A block is a row
- By default Erdas Imagine files are tiled into blocks that are 64x64 pixels
- This example has 5x5 tiles
Reading one row at a time

- Loop through the rows and read all pixels in that row during each iteration

```python
for i in range(rows):
    data = band.ReadAsArray(0, i, cols, 1)
    # do something with the data here, before # reading the next row
```

- The built-in `range(n)` function creates a list of numbers from 0 to n-1

```python
>>> print range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```
Reading a row of blocks

- It’s almost as easy to read in a row of blocks.
- Need to check that we can get a whole block in the y direction – get an error if request more data than exists in the file.
• Use `range(start, stop, step)` to loop through each group of blocks

```python
>>> print range(0, 13, 5)
[0, 5, 10]
```

```python
bSize = 5
for i in range(0, rows, bSize):
    if i + bSize < rows:
        size = bSize
    else:
        size = rows - i
data = band.ReadAsArray(0, i, cols, size) # do something with the data here, before reading the next set of blocks
```
rows = 13
bSize = 5
for i in range(0, rows, bSize):
    if i + bSize < rows:
        size = bSize
    else:
        size = rows - i
    data = band.ReadAsArray(0, i, cols, size)

i = [0, 5, 10]
0 + 5 < 13, so size = 5
ReadAsArray(0, 0, 11, 5)
rows = 13
bSize = 5
for i in range(0, rows, bSize):
    if i + bSize < rows:
        size = bSize
    else:
        size = rows - i
    data = band.ReadAsArray(0, i, cols, size)

i = [0, 5, 10]

5 + 5 < 13, so size = 5
ReadAsArray(0, 5, 11, 5)
rows = 13
bSize = 5
for i in range(0, rows, bSize):
    if i + bSize < rows:
        size = bSize
    else:
        size = rows - i
    data = band.ReadAsArray(0, i, cols, size)

i = [0, 5, 10]

10 + 5 > 13, so size = 13 - 10 = 3
ReadAsArray(0, 10, 11, 3)
Reading block by block

- The most efficient way to read data
- Use one loop for the rows and one for the columns
- Need to check that there is an entire block in both directions
rows = 13, cols = 11
range(0,13,5) & range(0,11,5) both return [0, 5, 10]

xBSize = 5
yBSize = 5
for i in range(0, rows, yBSize):
    if i + yBSize < rows:
        numRows = yBSize
    else:
        numRows = rows - i
for j in range(0, cols, xBSize):
    if j + xBSize < cols:
        numCols = xBSize
    else:
        numCols = cols - j
    data = band.ReadAsArray(j, i, numCols, numRows)
# do something with the data here, before
# reading the next block
rows = 13, cols = 11, xBSize = 5, yBSize = 5
for i in range(0, rows, yBSize):
    if i + yBSize < rows:
        numRows = yBSize
    else:
        numRows = rows - i
for j in range(0, cols, xBSize):
    if j + xBSize < cols:
        numCols = xBSize
    else:
        numCols = cols - j
    data = band.ReadAsArray(j, i, numCols, numRows)

i = [0, 5, 10]
0 + 5 < 13, so numRows = 5

j = [0, 5, 10]
0 + 5 < 11, so numCols = 5

ReadAsArray(0, 0, 5, 5)
rows = 13, cols = 11, xBSize = 5, yBSize = 5
for i in range(0, rows, yBSize):
    if i + yBSize < rows:
        numRows = yBSize
    else:
        numRows = rows - i
for j in range(0, cols, xBSize):
    if j + xBSize < cols:
        numCols = xBSize
    else:
        numCols = cols - j
data = band.ReadAsArray(j, i, numCols, numRows)

i = [0, 5, 10]
0 + 5 < 13, so numRows = 5

j = [0, 5, 10]
5 + 5 < 11, so numCols = 5

ReadAsArray(5, 0, 5, 5)
rows = 13, cols = 11, xBSize = 5, yBSize = 5
for i in range(0, rows, yBSize):
    if i + yBSize < rows:
        numRows = yBSize
    else:
        numRows = rows - i
for j in range(0, cols, xBSize):
    if j + xBSize < cols:
        numCols = xBSize
    else:
        numCols = cols - j
data = band.ReadAsArray(j, i, numCols, numRows)

i = [0, 5, 10]
0 + 5 < 13, so numRows = 5

j = [0, 5, 10]
10 + 5 > 11, so numCols = 11 - 10 = 1

ReadAsArray(10, 0, 1, 5)
rows = 13, cols = 11, xBSize = 5, yBSize = 5
for i in range(0, rows, yBSize):
    if i + yBSize < rows:
        numRows = yBSize
    else:
        numRows = rows - i
for j in range(0, cols, xBSize):
    if j + xBSize < cols:
        numCols = xBSize
    else:
        numCols = cols - j
    data = band.ReadAsArray(j, i, numCols, numRows)

i = [0, 5, 10]
5 + 5 < 13, so numRows = 5

j = [0, 5, 10]
0 + 5 < 11, so numCols = 5

ReadAsArray(0, 5, 5, 5)
```
rows = 13, cols = 11, xBSize = 5, yBSize = 5
for i in range(0, rows, yBSize):
    if i + yBSize < rows:
        numRows = yBSize
    else:
        numRows = rows - i
for j in range(0, cols, xBSize):
    if j + xBSize < cols:
        numCols = xBSize
    else:
        numCols = cols - j
    data = band.ReadAsArray(j, i, numCols, numRows)

i = [0, 5, 10]
5 + 5 < 13, so numRows = 5

j = [0, 5, 10]
5 + 5 < 11, so numCols = 5

ReadAsArray(5, 5, 5, 5)
```
rows = 13, cols = 11, xBSIZE = 5, yBSIZE = 5
for i in range(0, rows, yBSIZE):
    if i + yBSIZE < rows:
        numRows = yBSIZE
    else:
        numRows = rows - i
for j in range(0, cols, xBSIZE):
    if j + xBSIZE < cols:
        numCols = xBSIZE
    else:
        numCols = cols - j
data = band.ReadAsArray(j, i, numCols, numRows)

i = [0, 5, 10]
5 + 5 < 13, so numRows = 5

j = [0, 5, 10]
10 + 5 > 11, so numCols = 11 - 10 = 1

ReadAsArray(10, 5, 1, 5)
rows = 13, cols = 11, xBSize = 5, yBSize = 5
for i in range(0, rows, yBSize):
    if i + yBSize < rows:
        numRows = yBSize
    else:
        numRows = rows - i
for j in range(0, cols, xBSize):
    if j + xBSize < cols:
        numCols = xBSize
    else:
        numCols = cols - j
    data = band.ReadAsArray(j, i, numCols, numRows)

i = [0, 5, 10]
10 + 5 > 13, so numRows = 13 - 10 = 3

j = [0, 5, 10]
0 + 5 < 11, so numCols = 5

ReadAsArray(0, 10, 5, 3)
rows = 13, cols = 11, yBSize = 5, xBSize = 5
for i in range(0, rows, yBSize):
    if i + yBSize < rows:
        numRows = yBSize
    else:
        numRows = rows - i
for j in range(0, cols, xBSize):
    if j + xBSize < cols:
        numCols = xBSize
    else:
        numCols = cols - j
data = band.ReadAsArray(j, i, numCols, numRows)

i = [0, 5, 10]
10 + 5 > 13, so numRows = 13 - 10 = 3

j = [0, 5, 10]
5 + 5 < 11, so numCols = 5

ReadAsArray(5, 10, 5, 3)
rows = 13, cols = 11, xBSize = 5, yBSize = 5
for i in range(0, rows, yBSize):
    if i + yBSize < rows:
        numRows = yBSize
    else:
        numRows = rows - i
for j in range(0, cols, xBSize):
    if j + xBSize < cols:
        numCols = xBSize
    else:
        numCols = cols - j
    data = band.ReadAsArray(j, i, numCols, numRows)

i = [0, 5, 10]
10 + 5 > 13, so numRows = 13 - 10 = 3

j = [0, 5, 10]
10 + 5 > 11, so numCols = 11 - 10 = 1

ReadAsArray(10, 10, 1, 3)
Numeric & numpy

• Python modules for processing large arrays
• We’ll talk more about it next week
• Use Numeric with FWTools and numpy otherwise

```python
import Numeric # FWTools
import numpy # otherwise (ie on Macs)
```
Converting array data types

• If reading byte data (which we are at this point) then the arrays returned by `ReadAsArray()` will also by byte

• Sometimes we need the data as a different type

```python
data = band.ReadAsArray(j, i, nCols, nRows)
data = data.astype(Numeric.Float)  # Numeric
data = data.astype(numpy.float)  # numpy
```

• Can do it in one step:

```python
data = band.ReadAsArray(j, i, nCols, nRows).astype(Numeric.Float)
```
Creating a mask

• Say we want to do some processing on all pixels with a value greater than 0
• Syntax is the same for numpy

```python
mask = Numeric.greater(data, 0)

>>> a = Numeric.array([0, 4, 6, 0, 2])
>>> print a
[0 4 6 0 2]
>>> mask = Numeric.greater(a, 0)
>>> print mask
[0 1 1 0 1]
```
Summing values in an array

- Use `Numeric.sum(<array>)` or `numpy.sum(<array>)`

```python
>>> a = Numeric.array([0, 4, 6, 0, 2])
>>> print a
[0 4 6 0 2]
>>> print Numeric.sum(a)
12
```
• If array is 2D then `sum()` returns an array

```python
>>> b = Numeric.array([[a, [5, 10, 0, 3, 0]]])
>>> print b
[[ 0  4  6  0  2]
 [ 5 10  0  3  0]]
>>> print Numeric.sum(b)
[ 5 14  6  3  2]
```

• To get one total sum

```python
>>> print Numeric.sum(Numeric.sum(b))
30
```
Counting pixels where value > 0

• Create a mask and sum the values

```python
>>> print a
[0 4 6 0 2]
>>> mask = Numeric.greater(a, 0)
>>> print mask
[0 1 1 0 1]
>>> print Numeric.sum(mask)
3
```
# script to count the number of non-zero pixels in the first band
import os, sys, ogr, gdal, utils, Numeric
from gdalconst import *

os.chdir(r'Z:\Data\Classes\Python\data')

# register all of the GDAL drivers
gdal.AllRegister()

# open the image
ds = gdal.Open('aster.img', GA_ReadOnly)
if ds is None:
    print 'Could not open aster.img'
    sys.exit(1)

# get image size
rows = ds.RasterYSize
cols = ds.RasterXSize
bands = ds.RasterCount

# get the band and block sizes
band = ds.GetRasterBand(1)
blockSizes = utils.GetBlockSize(band)
xBlockSize = blockSizes[0]
yBlockSize = blockSizes[1]

# initialize variable
count = 0
# loop through the rows
for i in range(0, rows, yBlockSize):
    if i + yBlockSize < rows:
        numRows = yBlockSize
    else:
        numRows = rows - i

# loop through the columns
for j in range(0, cols, xBlockSize):
    if j + xBlockSize < cols:
        numCols = xBlockSize
    else:
        numCols = cols - j

# read the data and do the calculations
data = band.ReadAsArray(j, i, numCols, numRows).astype(Numeric.Float)
mask = Numeric.greater(data, 0)
count = count + Numeric.sum(Numeric.sum(mask))

# print results
print 'Number of non-zero pixels:', count
Assignment 4b

- Write a script to calculate the average pixel value for the first band in aster.img
- Read in the data one block at a time
- Do the calculation two ways
  - Including zeros in the calculation
  - Ignoring zeros in the calculation
- Turn in your code and the output