Report
on

Extracting Bands using GDAL in Python
16-Jan-2019

Submitted by

Love Prakash

DronaMaps Private Limited

Contents

1 GDAL - Geospatial Data Abstraction Library:

GDAL is a translator library for raster and vector geospatial data formats that is released under an X/MIT style Open Source license by the Open Source [1]Geospatial Foundation. As a library, it presents a single raster abstract data model and single vector abstract data model to the calling application for all supported formats. It also comes with a variety of useful command line utilities for data translation and processing. The NEWS page describes the December 2018 GDAL/OGR 2.4.0 release.

2 Reading Raster Data with Python and gdal:

I am trying to learn Python for Geoprocessing. Here are some very basic notes on "playing" with Python/gdal/ogr. The online documentation is, I must say, rather confusing, so I try it step by step at the command line as below.

3 Opening the File:

Before opening a GDAL supported raster datastore it is necessary to register drivers. There is a driver for each supported format. Normally this is accomplished with the GDALAllRegister() function which attempts to register all known drivers, including those auto-loaded from .so files using GDALDriverManager::AutoLoadDrivers(). If for some applications it is necessary to limit the set of drivers it may be helpful to review the code from gdalallregister.cpp. Python automatically calls GDALAllRegister() when the gdal module is imported.

Once the drivers are[1] registered, the application should call the free standing GDALOpen() function to open a dataset, passing the name of the dataset and the access desired (GA_ReadOnly or GA_Update).

3.1 In Python:

```
from osgeo import gdal
dataset = gdal.Open(filename, gdal.GA_ReadOnly)
if not dataset:
...
```

4 If unable to load file:

If GDALOpen() returns NULL it means the open failed, and that an error messages will already have been emitted via CPLError(). If you want to control how errors are reported to the user review the CPLError() documentation. Generally speaking all of GDAL uses CPLError() for error reporting. Also, note that pszFilename need not actually be[1] the name of a physical file (though it usually is). It's interpretation is driver dependent, and it might be an URL, a filename with additional parameters added at the end controlling

the open or almost anything. Please try not to limit GDAL file selection dialogs to only selecting physical files.

5 Getting Dataset Information:

As described in the GDAL Data Model, a GDALDataset contains a list of raster bands, all pertaining to the same area, and having the same resolution. It also has metadata, a coordinate system, a georeferencing transform, size of raster and various other information.

In the particular, but common, case of a "north up" image without any rotation or shearing, the georeferencing transform takes the following form :

```
adfGeoTransform[0] /* top left x */
adfGeoTransform[1] /* w-e pixel resolution */
adfGeoTransform[2] /* 0 */
adfGeoTransform[3] /* top left y */
adfGeoTransform[4] /* 0 */
adfGeoTransform[5] /* n-s pixel resolution (negative value) */
```

6 Fetching a Raster Band:

At this time access to raster data via GDAL is done one band at a time. Also, there is metadata, block sizes, color tables, and various other information available on a band by band basis. The following codes fetches a GDALRasterBand object from the dataset (numbered 1 through GetRasterCount()) and displays a little information about it.

6.1 In Python (note several bindings are missing):

```
band = dataset.GetRasterBand(1)
print("Band Type={}".format(gdal.GetDataTypeName(band.DataType)))

min = band.GetMinimum()
max = band.GetMaximum()
if not min or not max:
(min,max) = band.ComputeRasterMinMax(True)
print("Min={:.3f}, Max={:.3f}".format(min,max))

if band.GetOverviewCount() > 0:
print("Band has {} overviews".format(band.GetOverviewCount()))

if band.GetRasterColorTable():
print("Band has a color table with {} entries".format(band.GetRasterColorTa
```

7 Reading Raster Data

: There are a few ways to read raster data, but the most common is via the GDALRasterBand::RasterIO() method. This method will automatically take care of data type conversion, up/down sampling and windowing. The following code will read the first scanline of data into a similarly sized buffer, converting it to floating point as part of the operation.

7.1 In Python:

```
scanline = band.ReadRaster(xoff=0, yoff=0,
xsize=band.XSize, ysize=1,
buf_xsize=band.XSize, buf_ysize=1,
buf_type=gdal.GDT_Float32)
```

7.2 The RasterIO call takes the following arguments.

```
CPLErr GDALRasterBand::RasterIO( GDALRWFlag eRWFlag,
int nXOff, int nYOff, int nXSize, int nYSize,
void * pData, int nBufXSize, int nBufYSize,
GDALDataType eBufType,
int nPixelSpace,
int nLineSpace )
```

Note that the returned scanline is of type string, and contains xsize*4 bytes of raw binary floating point data. This can be converted to Python values using the struct module from the standard library:

```
import struct
tuple_of_floats = struct.unpack('f' * b2.XSize, scanline)
```

Note that the same RasterIO() call is used to read, or write based on the setting of eRWFlag (either GF_Read or GF_Write). The nXOff, nYOff, nXSize, nYSize argument describe the window of raster data on disk to read (or write). It doesn't have to fall on tile boundaries though access may be more efficient if it does.

The pData is the memory buffer the data is read into, or written from. It's real type must be whatever is passed as eBufType, such as GDT_Float32, or GDT_Byte. The RasterIO() call will take care of converting between the buffer's data type and the data type of the band. Note that when converting floating point data to integer RasterIO() rounds down, and when converting source values outside the legal range of the output the nearest legal value is used. This implies, for instance, that 16bit data read into a GDT_Byte buffer will map all values greater than 255 to 255, the data is not scaled!

The nBufXSize and nBufYSize values describe the size of the buffer. When loading data at full resolution this would be the same as the window size. However, to load a reduced resolution overview this could be set to smaller than the window on disk. In this case the RasterIO() will utilize overviews to do the IO more efficiently if the overviews are suitable.

The nPixelSpace, and nLineSpace are normally zero indicating that default values should be used. However, they can be used to control access to the memory data buffer, allowing reading into a buffer containing other pixel interleaved data for instance.

8 Closing the Dataset:

Please keep in mind that GDALRasterBand objects are owned by their dataset, and they should never be destroyed with the C++ delete operator. GDALDataset's can be closed by calling GDALClose() (it is NOT recommended to use the delete operator on a GDALDataset for Windows users because of known issues when allocating and freeing memory across module boundaries. See the relevant topic on the FAQ). Calling GDALClose will result in proper cleanup, and flushing of any pending writes. Forgetting to call GDALClose on a dataset opened in update mode in a popular format like GTiff will likely result in being unable to open it afterwards.

9 Script in Python to extact bands of a tiff file:

```
In [1]: import gdal

In [2]: from gdalconst import *

In [3]: from osgeo import gdal

In [4]: filename = 'xyz.tiff'

In [5]: dataset = gdal.Open(filename, GA_ReadOnly)

In [6]: dataset
Out[6]: <osgeo.gdal.Dataset; proxy of <Swig Object of type 'GDALDatasetShadow *' at 0x0000020AC7273840>

In [7]: cols = dataset.RasterXSize

In [8]: rows = dataset.RasterYSize

In [9]: bands = dataset.RasterCount

In [10]: driver = dataset.GetDriver().LongName

In [11]: cols
Out[11]: 1001

In [12]: rows
Out[12]: 801
```

Figure 1: Python code

```
In [13]: bands
Out[13]: 3

In [14]: driver
Out[14]: 'GeoTIFF'

In [15]: geotransform = dataset.GetGeoTransform()

In [16]: geotransform
Out[16]: (0.0, 1.0, 0.0, 0.0, 0.0, 1.0)

In [18]: originX = geotransform[0]

In [19]: originY = geotransform[3]

In [20]: pixelWidth = geotransform[1]

In [21]: pixelHeight = geotransform[5]

In [22]: originX
Out[22]: 0.0

In [23]: originY
Out[23]: 0.0

In [24]: pixelWidth
Out[24]: 1.0
```

Figure 2: Python code

```
In [25]: pixelHeight
Out[25]: 1.0

In [26]: band = dataset.GetRasterBand(1)

In [27]: bandtype = gdal.GetDataTypeName(band.DataType)

In [28]: bandtype
Out[28]: 'Byte'

In [29]: scanline = band.ReadRaster( 0, 0, band.XSize, 1,band.XSize, 1, band.DataType)

In [30]: scanline
Out[30]: b'yxz{|z{}|}||{}\x80}}\x7f|}}~~}}~}\x81\x80\x7f\x81\x81\x80~\x81\x80\x82\x81\x80\x80\x7f}\x82\x82\x83
\x80~~\x83\x82\x82\x83\x82\x82\x80{uo\x85\x8b\x9d\x95\xa4\x96\x9f\x94\x91v\x7f\xd9\xca\xc7\xc5\xc4\xc
\x8e\xd3\xbe\xad\xa5\x8faLB?E>LDHE<EFA1:88>IJH:EEJSPZfmdary{oh\\TU]{z}xyuvs{}z\x8d\x8f\x8f\x8f\x89\x9
\xde\xdb\xc9\xca\xdc\xd8\xb6\xba\xbf\xc5BHMk\xe0\xdf\xe0\xdd\xdb\xd7\xd8\xd9\xd9\xdc\xda\xa6\x93}:?D7
8c\x95\xa4\xa5\xa3\xa6\xa2\xa2\xa6\xa6\x94\x9a\x9e\xa0\x92\x9b\xa0\x9d\xa8\xab\xab\xad\xb2n`\xb5\xbb\
\xad\xae\xac\xae\xaf\xad\xac\xae\xb9\xa3hVfhklbdXWQK<:;62?B=A=7AED:CA==:Fa?DGF:FLKJEJRJ:=:E;G5@5E3;7.
6SE?A77\x80E5+4471\x9diLZUP_zk|xZV\\cerg\x80\xaa\xaa\xad\xa5\x9e\x8c\x88\x80\x8a\x89\x86\x8e\x8f\x9d\
\x83fpo\x7foercYV[]c_fUTrdYcYWed^jcx\x9b\x8f\x8d\x97\x94\x8f\x89~\x83udhpEAB@??OKR[ADGMPIFD@MMKAIg\x8
a\x81\x8c\xa6u\x8c\xab\xbb\xa4\x99\xa0\x8a\x7f\x89\x90\x8f\x80\x80p\x86y~\x84\x93\x8b\x91\x9a{^alcWSD
5\xa5\xb5\xb5\xb6\xb4\xb6\xb9\xb4\xb4\xb1\xb5\xbe\xb1\xb2\xae\xaekDHGi^`VZ^Z`]\\X]XXR\\\\u\x83\x95\x8
\xd3\xcc\xcb\xcb\xcc\xd3\xd3\xd4\xd5\xd6\xd5\xae\xaa\xa9\xa9\xa7\xa5\xa3\xa3\xa3\xa3\xa5_LNNSU\\TUYYX
8l\x86o\xaa\xae\xcc\x90\x8b\x8a\x82p\x82\x96\xa9\xa8\xb0\x82\xba\xc6\xbf\xb4\xa2\x92\x86\xae\x83\xa5\
[\\`dkqqzy\x89fk\x82gti\xc3n}\x7fy~\x8ft}}\x83\x98\x99\x99\xa2\xc1\xbc\xc1\xcc\xb1Z`ji}ct\x80\x86\x83
\x90\x8c\x7f\x84\x85f\x82j\x81\xbarK=s}fb\xc3\xc7\xc4\xc5\xc6\xc1\xd2\xc9\xc6\xc4\xc3\xc4\xc4\xcb\xc6
o\x84\x82p_zdotYLX\\Z]QMGDSKdYNSUYNFb^=TMd\x9eRQKTZPt|\x8c\x8e\x92\x80FfY`=Z@BPXI@RURijp]XXBhSXdev\x8
9\xb1\xb2\xb2\xb7\xab\xb2\xb3\x9agz\x85\x85seZ^ZRZQRz_bnLTSUSDO?LORNT=kL<POZgbESZqOjPA:UHNFBzjQZDHFES
VuYPS[KOUt\\NjZCFdriKXJ\x84xUZ\x7fYppMPYRSNUleJb]Z_]UJKIQ_ha`d`agWPZQTTSf'
```

Figure 3: Python code

```
In [31]: import struct

In [36]: data = band.ReadAsArray(0, 0, cols, rows)

In [38]: value = data[800,900]

In [39]: value
Out[39]: 110

In [40]: import numpy

In [41]: data = band.ReadAsArray(0, 0, dataset.RasterXSize, dataset.RasterYSize).astype(numpy.float)

In [42]: value = data[800,900]

In [43]: value
Out[43]: 110.0

In [46]: src_ds = gdal.Open("xyz.tiff")
         if src_ds is not None:
             print ("band count: " + str(src_ds.RasterCount))
         band count: 3

In [ ]:
```

Figure 4: Python code

References

[1] Daniel Clewley, Peter Bunting, James Shepherd, Sam Gillingham, Neil Flood, John Dymond, Richard Lucas, John Armston, and Mahta Moghaddam. A python-based open source system for geographic object-based image analysis (geobia) utilizing raster attribute tables. *Remote Sensing*, 6(7):6111–6135, 2014.