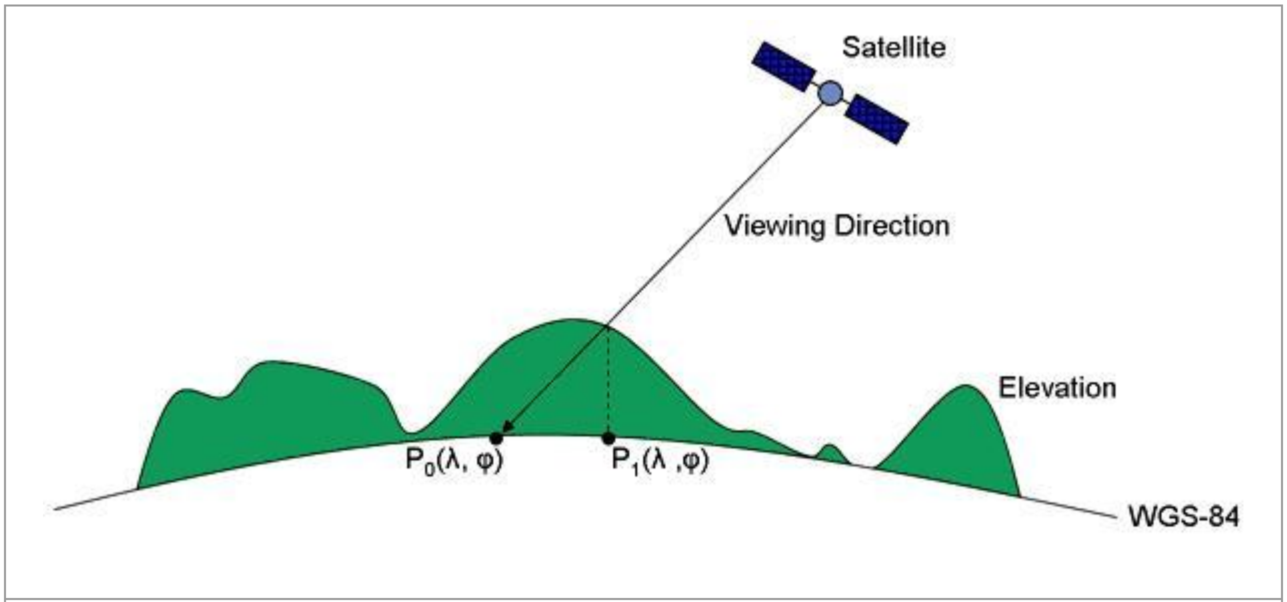
Commercial Satellite Imagery Orthorectification and Manipulation with GDAL

Version: GDAL 1.10.0 Author: Claire Porter, Polar Geospatial Center Updated: July 2014

INTRODUCTION

Orthorectification is the process of removing the effects of image perspective (tilt) and relief (terrain) for the purpose of creating a planimetrically correct image. Using an elevation model, an image can be corrected so that georeferenced pixels in the image roughly correspond to what one would see if one were flying directly over that geo-location. The resultant orthorectified image has a constant scale wherein features are represented in their *true* positions. This allows for the accurate, direct measurement of distances, angles, and areas. Image providers such as DigitalGlobe strongly recommend orthorectifying satellite images to guarantee geo-accuracy.



Effects of sensor geometry and terrain on image accuracy. Without terrain correction (orthorectification) point P would appear at P_0 instead of its correct location at P_1. Image courtesy NASA SeaDAS.

A note on accuracy: Many of the images from DigitalGlobe and Geoeye are georeferenced via a generic sensor model called RPC, rational polynomial coefficients. The RPC sensor model allows for remarkably accurate orthorectification. In fact, unless you have high-resolution DEM derived from lidar, your elevation model will introduce far more error into the resulting orthoimage than the sensor model. The RPC sensor model is almost always relative to the WGS84 ellipsoid when it comes to elevation values. Make sure your DEM's vertical coordinate system is also WGS84 or similar.

Most off-the-shelf commercial GIS and image analysis packages (ERDAS Imagine, ENVI, ArcGIS Desktop) offer an orthorectification tool. We found that while these packages work extremely well for small projects, we could not leverage the compute resources available to us with desktop-bound software, and the licensing costs of server-based processing was out of our budget. As such, we turned to an open source alternative, the Geospatial Data Abstraction Library (GDAL). The GDAL utilities are all invoked on the command line and their API has bindings for scripting languages like Python and Ruby. In its basic form, GDAL is not as user friendly for manipulation of single images, but it can be scripted to automatically perform complicated workflows. And, because its powerful API is Linux-based, it can be ported to supercomputing infrastructure. Open source tools like GDAL have proved to be key for the type of imagery we typically handle.

GETTING THE PACKAGE

There are binaries for most platforms. More information is available at the GDAL trac page at http://trac.osgeo.org/gdal/wiki/DownloadingGdalBinaries. We suggest the OSGeo4w package for use with Windows. However you install GDAL/OGR, make sure your build includes these few crucial libraries: GEOS, PROJ4, Python bindings, and a JPEG2000 SDK such as openjepg2.

There is an open source GIS package called *Quantum GIS* that combines some of the power and accessibility of GDAL with the user-friendliness of a GUI GIS package. If you don't have access to a commercial package, QGIS may be the answer.

UNDERSTANDING THE IMAGERY

High resolution commercial imagery usually comes as GeoTiff or in the National Imagery Transmission Format (NITF). NITF is actually a metadata wrapper for a losslessly compressed JPEG2000 image. Most GIS packages can read NITF, but some still require you to purchase an additional module to enable this feature. This practice is becoming less common as these data grow in popularity.

In both cases, there may be several auxiliary files (extensions .GEO, .RPB, .EPH, .IMD, etc.) included with the image or in a tar archive. These all include important metadata that may be needed for GIS packages to accurately read the image sensor model or radiometric settings. This is especially true for GeoTiff files, which allow significantly less metadata to be stored in the image itself.

Not all commercial imagery will benefit from orthorectification. "Basic" level products (DigitalGlobe's terminology) have not yet been georectified and include the sensor model necessary to correct for terrain-related displacement. Higher level products have been orthorectified to a DEM or to a constant elevation. These products may not be able to be orthorectified again. Higher-level products are easier to work with because they have been georectified to a spatial reference system, but they generally have the drawback of being fixed products; you can't orthorectify them again to a better DEM for more accurate results. The basic level images may be unwieldy, but when you acquire a higher quality DEM, you can build a more accurate orthoimage then the one you built initially.

Basic images are georeferenced via a generic sensor model (called RPC, rational polynomial coefficients) and are not resampled to follow a coordinate system. They also have very little radiometric correction applied to convert their raw digital number (DN) values to a scale that is sensible to the human eye. As a result, the basic images behave differently than other imagery products.

Most DigitalGlobe basic level images come with their RPC sensor model relative to the WGS84 geographic coordinate system. This means that the image will load into a GIS as if it were in WGS84. While this is not a problem for most places, the images end up looking long and thin in Polar Regions. If you change the display projection of the GIS (not the image projection) to something more locally appropriate, the image should look more normal. It's important to note that the image is not actually georectified to the WGS84 projection, and that you should reproject it to a local coordinate system when orthorectifying.

ORTHORECTIFICATION WITH THE RPC MODEL

The `gdalwarp` utility can transform the basic image to a georectified image and correct the terrain-related distortion in a basic image. `gdalwarp` can also be used for reprojection. Here's the template to tell gdalwarp how to use a DEM and RPCs for orthorectification:

```
gdalwarp -rpc -to rpc_dem=<dem> -et 0.01 -resample cubic -t_srs
        <target_srs> <source_dataset> <target_dataset>
```

`-rpc` specifies that an RPC model is to be used.

`-to rpc_dem=<dem>` tells `gdalwarp` what dem to use where <dem> is the filepath of your reference DEM. Alternatively, if you do not have a DEM, you can use `-to rpc_height=<constant_elev>` to specify a constant elevation.

`-et` stands for error threshold. You will end up with fewer random orthorectification errors if you set this to 0.01.

`-resample cubic` sets the resampling method to cubic convolution. This is far more useful for visual interpretation than the default, nearest neighbor. If you want to maintain the original pixel values (no averaging), leave this argument out.

Though this software is highly robust, there are a few gotchas that can make it fail, especially near 180 degrees longitude. For best results, look at the `gdalwarp` documentaion at http://www.gdal.org/gdalwarp.html. Search the web for your error; there are great archives of GDAL's help listserv. You can join the gdal-dev email list: gdal-dev@lists.osgeo.org and ask the developers if you are still having trouble. Always feel free to ask PGC staff as well. It's very likely we have seen the error before.

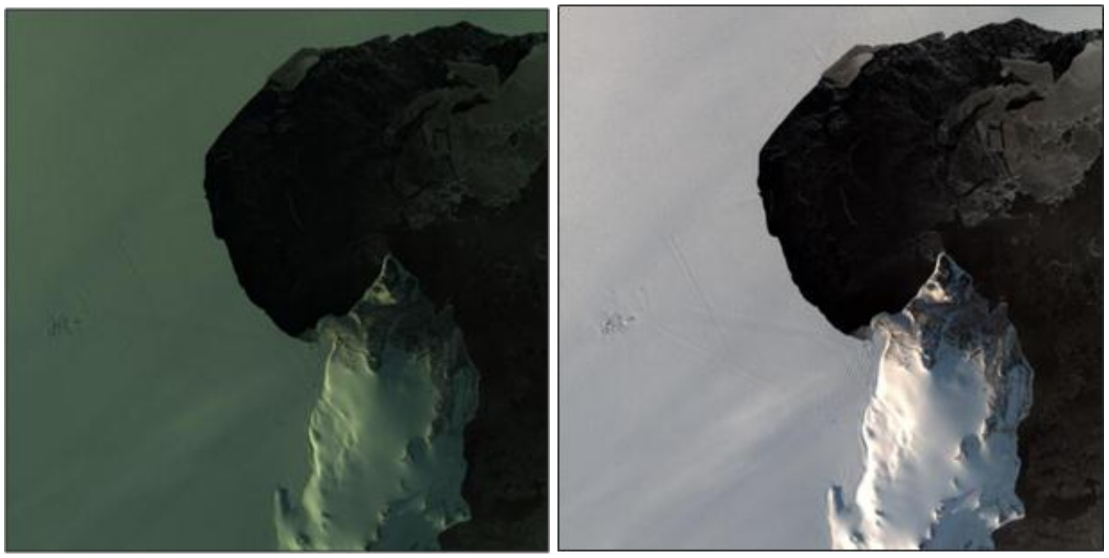
PROJECTIONS

In `gdalwarp`, `-t_srs <target_srs>` specifies the target image projection. Use the EPSG code or PROJ4 string. For example `-t_srs EPSG:3031` or `-t_srs '+proj=stere +lon_0=0 +lat_0=-90 +lat_ts=-71 +ellps=WGS84 +datum=WGS84'` tells gdalwarp to convert to Antarctic Polar Stereographic -71. While EPSG codes are easier to work with, we have seen errors with projections using NAD83 not being recorded properly. Specifically, ArcGIS and other GIS packages could locate the images in the correct place, but could not accurately perform any further transformations. As a result, we use PROJ4 strings whenever possible.

You can look up EPSG codes at http://spatialreference.org/ref/epsg/ or http://www.epsg-registry.org/. Change any EPSG code or WKT projections (well known text) into a PROJ4 string using the GDAL utility `gdalsrsinfo`.

IMAGE COLOR CORRECTION

Almost all commercial imagery is 11 bit, meaning that it has values ranging from 0 to 2,047 for each band. The imagery is collected with a certain exposure level, much like a handheld camera, in order to maximize the information captured. This makes it look unlike what the human eye expects (i.e. completely green or yellow in the multispectral). For basic viewing in a GIS, you can use a standard deviation stretch to make the image look "normal." Remember to use the correct band combinations: RGB is 3,2,1 in 4-band images and 5,4,2 for 8-band.



Two images illustrating the benefits of radiometric correction on a QuickBird-2 Scene of McMurdo Station, Antarctica. The left image shows the raw DN values with a global min-max stretch. The right shows top-of-atmosphere reflectance values calculated using band-specific calibration factors.

If you need to convert the values into absolute radiance or top-of-atmosphere reflectance, either for spectral analysis or just so the images match each other well, you'll need to perform some math using the calibration factors in the metadata. For DigitalGlobe imagery, you can get this information from their white paper on the subject at https://www.digitalglobe.com/sites/default/files/Radiometric_Use_of_WorldView-2_Imagery%20(1).pdf. Note that while this article is specific to WorldView-2, the techniques apply to all DG data. Ask PGC staff for information related to GeoEye formatted data.

GDAL can rescale the imagery according the calibration factors you derive in two different ways: 1) `gdal_translate` with the `-scale` option or 2) building an intermediate dataset in the GDAL Virtual Format.

`gdal_translate` has an option, `-scale`, that allows you to set the min and max input and output values:

```
    gdal_translate -scale 0 2047 0 255 <source_ds> <target_ds>
```

This scaling applies to all bands in the raster. So, if you have a multispectral raster where each bacd should have a different scale range, you must run each band separately:

```
gdal_translate -b 1 -scale 0 2047 0 300 <source_ds> <target_ds_band1>
gdal_translate -b 2 -scale 0 2047 0 245 <source_ds> <target_ds_band2>
gdal_translate -b 3 -scale 0 2047 0 273 <source_ds> <target_ds_band3>
gdal_translate -b 4 -scale 0 2047 0 280 <source_ds> <target_ds_band4>
```

and merge them back together afterward:

```
  gdal_merge.py -o <target_ds_merged> -separate <target_ds_band1>
       <target_ds_band2> <target_ds_band3> <target_ds_band4>
```

Note: as of gdal 1.11, `gdal_translate` has the option to specify the scale range for each band, obviating the need to run each band separately and merge them afterwards.

We choose to use GDAL's Virtual Format, VRT, to build our corrected imagery. The VRT format is both flexible and lightweight. It allows you to do a lot of image manipulation without writing an excessive amount of intermediate files. The format itself is simply a markup language in a text file. Learn more in the tutorial here: http://www.gdal.org/gdal_vrttut.html.

You can make simple VRTs with the `gdalbuildvrt` utility. For more advanced uses, you can write the VRT files with the GDAL API and Python/C/C++ or you can write them using text editing programs or scripts.

BIT DEPTH CHANGES

When correcting the imagery, you may find it wise to change the datatype of the pixels. The imagery comes in unsigned 16-bit integer type, which is not much help if you want to store reflectance values that range from 0-1. Alternately, we often make unsigned 8-bit products because they are easier for laypeople to use in a GIS.

`gdal_translate` has an option `-ot,` for output type, that allows you to specify the data type. Common options are Byte, UInt16, Float. For more information, see the `gdal_translate` documentation at http://www.gdal.org/gdal_translate.html.

OUTPUT FILE FORMATS

GDAL can read and write many image formats. More can be added by including format-specific SDKs in your build. Type `gdal_translate --formats` to see a list of the formats supported by your build.

Each format has creation options. You invoke them using the `-co` option. They can be extremely helpful, so it's worth your while to look them over. For example, the GeoTiff format driver lets you set the compression method so your output files take up less space:

```
 gdal_translate -of GTiff --co COMPRESS=LZW <source_ds> <target_ds>
```

Here's GDAL's raster format list: http://www.gdal.org/formats_list.html

GDAL RUNTIME CONFIGURATION OPTIONS

GDAL has some generic configuration options that can make or break your workflow. For example, we initially got a Segmentation Fault every time we tried to run `gdalwarp` on a large image. But, when we set the `GDAL_CACHEMAX` option to 2GB, the memory error disappeared:

```
  gdalwarp --config GDAL_CACHEMAX 2048 -t_srs EPSG:3031 <source_ds>
                           <target_ds>
```

Similarly, we were having trouble with images that cross 180 degrees. Including the `CENTER_LONG` option helped solve the problem:

```
gdalwarp --config CENTER_LONG 179.45 -t_srs EPSG:3031 <source_ds>
                          <target_ds>
```

All the config options can be found here: http://trac.osgeo.org/gdal/wiki/ConfigOptions.

BUILDING PYRAMIDS

For ease of use, we always build pyramids for our processed imagery. Use the `gdaladdo` utility to add pyramid layers. They are stored internally by default, but can be built in an external file by using the `-ro` option.

```
gdaladdo <target_ds> 2 4 8 16
```