Recommendations for
Improved Elevation (Vertical) Data Support in GDAL

Author(s): Ray Gardener, Daylon Graphics Ltd. (rayg@daylongraphics.com)
Last updated: April 28, 2006, 4th draft

Introduction

Ah, to live on a flat world. Life would indeed be simpler. But even then, people would want to measure buildings and know how high aircraft were flying. It seems that our third dimension is, for better or worse, inescapable.

The Geospatial Data Abstraction Library simplifies working with geographic coordinates, and by natural extension, geographically positioned objects. While it excels at handling locations along the ground, support for locations along the vertical axis (altitudes/elevations) has been mixed. We need to explore how we got here, what the issues currently are, and where we can go from here.

Note: we use the word "vertical" here to mean an axis perpendicular to the groundplane, and not to the Y axis of a bitmap or DEM, which is usually the north-south axis. The terms "breadth" and "breadthwise" are used when referring to the latter.

References to OGR types are made, but are considered references to GDAL proper.

Contributor Background

The basis for this text stems from using GDAL to implement raster elevation I/O for my company's Leveller software product. The experiences gained in developing a wrapper around GDAL (and other internal and external code modules such as plug-ins) have provided useful insights into the issues and recommendations. It is preferable to have the solutions at GDAL's level (and/or perhaps lower, e.g., PROJ.4) to standardize vertical data I/O and 3D coordinate transforms, as GDAL already enjoys an extensive driver set for DEM datasets.

Frank Warmerdam, GDAL's principal maintainer, has been helping guide the discussion of this subject from its earliest times, and has been instrumental in exploring the space of ideas for the best scaleable solution with minimal impact on the existing codebase.

How We Got Here

Among other things, GDAL abstracts the representation of georeferenced data. Although I'm personally primarily concerned with raster data, the issues can pertain equally to vector data (insofar as a vertical axis is used in one's vector data).

A core GDAL service is converting between different map projections, and indeed, GDAL uses the popular PROJ.4 library to do so. Related to the issue of converting between different projections is defining precisely coordinates in latitude/longitude, which use datums (approximating ellipsoids of the

Earth's shape) so that a place measured at one time can be known in relation to another place even if defined at a later time or with a different referent. Without such corrections for latitude/longitude, one's input data into a conversion would be faulty, by up to a few kilometers.

Elevations can't even be determined until one knows where one is on the ground, and with the ground coordinate issue resolved, the remaining problem of elevation uncertainty was sufficiently minor for most people. We usually use mean sea level as a vertical referent, and since it has a relatively small difference from the geoid, most people are happy. On top of this, much source data comes with an implicit elevation referent to MSL, so one's hands tend to be tied anyway. Finally, our knowledge of the geoid hasn't really been accurate until recently (the mid-90s), and even with a good geoid, it takes time for people to understand, use, and build the corresponding infrastructure of algorithms, and then for software to use and popularize them.

GDAL's design, especially with raster data, is to provide helpful access to the raw pixels in terms of formalizing block/tile I/O but to remain efficient by letting library users interpret the pixels as they best see fit. Essentially, GDAL provides a “raster filesystem” protocol.

While GDAL supports the obvious usage of color in bitmap files, other uses -- elevations included -- are left more at the user's interpretation. Compounding the problem is that some raster files are explicitly for the storage of color, some for elevations, and some are for the former but have been de facto “pressed into service” for elevations (e.g., heightfields for POV-Ray and other modeling packages). Naturally, this last case cannot say what units the elevations are in, since a color file has no idea about elevations. It's logically difficult to formalize elevation support when the standards and practices are, well, a bit hazy.

What are the Problems?

There are three main issues:

- No formal way to identify a raster band* as being elevation data,
- An inability to define an explicit vertical referent for such a dataset or band, and
- No formal way to accurately write raster* elevation data. In fact, this is a more general problem of not being able to accurately write any kind of data which is out-of-range of the physical pixel type.

* For vector shapes, since they are geometrical objects by nature, elevation aspects can be implicitly asumed (e.g., a Z coordinate member).

Fortunately, the referent problems can be treated independantly. We can continue to use MSL as an implicit referent while treating the identity and write issues.

What are the effects of these problems? Without an explicit vertical referent, one may experience vertical registration errors (elevation shift) when merging DEMs (which is more likely to happen with data obtained from different epochs). The error can be up to 100 meters. A less worrisome (although potentially long-term) problem is that implicitly using mean sea level as the referent will invalidate datasets if (or, as some feel, when) sea level changes.

*"Yeah, when I said this mountain is 10,000 feet tall,
I meant, of course, relative to the old sea level."*

Being unable to identify a band as elevation data is problematic when programming in a generic manner. Currently, one must tabulate the drivers in use and build a private list of which ones store elevations. This is not difficult in a static environment, but makes implementing drivers as user-loadable (or even user-definable) "plug-ins" tricky.

Where We Can Go from Here

Band Identity

Solving the band identity issue is a logical first step. With bands clearly identified as holding elevation data, it makes it easier for GDAL to maintain an elevation concept invariant for a band/dataset. We could have the functions `band::SetType` and `band::GetType`. An enumerated value or string (with some strings predefined, e.g., "color", "elevation", etc.) would provide the distinction. The string approach is less space efficient but provides flexibility for those implementing drivers that store other data types in raster form (along with a namespace collision avoidance protocol).

Color formats such as PNG/BMP, etc. should have `GetType` return "color" even though they have been usurped for elevation storage. An exception are color datasets that have an associated metadata file (or files) providing georeferencing data and an elevation storage intent. Formalizing the metadata protocol for particular color formats is not addressed here (although storing WKT strings in .prj files appears popular, in which case, a `VERT_CS` node in the WKT string would suggest elevations).

A "raw" type might be useful for those who want to bypass interpretations, e.g., write elevations directly instead of using logical pixel values (see "Writing Ranged Data" below).

As GDAL is implemented in C++, it might be worthwhile to take the concept of type to its logical conclusion and have the following type hierarchy:

```
GDALRasterBand
      GDALColorRasterBand
      GDALElevationRasterBand
```

This way, protocols specific to elevations and colors could be logically organized and kept apart where warranted. `GetType` would then implement an RTTI functionality.

"Type explosion" can occur if we wish to define explicit types for every conceivable georeferenced raster, such as temperature, pressure, all manner of optical and radar signals, etc. However, elevations may be worth formalizing since they directly relate to 3D coordinate system support; i.e., a natural extension of the GDAL data model to three dimensions. A band that knows it contains elevations, for instance, could be efficiently reprojected between both horizontal and vertical datums.

Vertical Referent Support

Vertical referent support requires:

- Being able to indicate the referent (e.g., a vertical datum EPSG code)
- Saying what units elevations are in (e.g., `band::SetUnitType`)
- Algorithms to convert elevations between referents.
- Protocols for drivers to set and get vertical referents.

Of GDAL's current drivers, only (?) the GeoTIFF format has explicit vertical datum indication (i.e., `VerticalCSTypeGeoKey`, which uses an EPSG code). Other formats may convert elevations to MSL, store a WKT string containing a `VERT_CS` node, etc., but the particulars of course depend on the driver. GeoTIFF, however, appears to be missing the newest vertical datums, such as EGM96.

Since OGRSpatialReference already has an `importWkt` method, using the `VERT_CS` node protocol would be an easy extension. This implies supporting `COMPD_CS` as a root node, however, since `VERT_CS` is sibling to the (normally root) `PROJCS/GEOCS/LOCALCS` nodes.

Converting elevations between referents is actually a matter of converting a 3D point, since a ground location is always required. I have some hesitation about having PROJ.4 do this, even though the `pj_transform` function takes 3D vectors, since PROJ.4's problem domain is lat/long coordinate conversion and vertical data is not a mandatory part of terrestrial locations. Or, if PROJ.4 were to be extended, that its problem domain be explictly widened to include vertical coordinate systems. However, most people who see PROJ.4 as "handling projections" may find the vertical parts of the interface a distraction. The advantage, however, is a centralized solution for terrestrial coordinate transformations. If not at the PROJ.4 level, however, then definitely in GDAL (i.e., OGRSpatialReference/OGRCoordTransform).

Geoid-based vertical datums comprised of detailed gravipotential maps can be large, and could easily be overkill for casual users. Ellipsoid approximations may suffice in these cases. Having an error term in the `VERT_CS` node would be handy to let users know the accuracy of an approximation, but this is a protocol change outside GDAL.

Elevation units are actually already supported with the `band::SetUnitType/GetUnitType` functions. However, the protocol could be firmer. Currently, unit definitions "m", "ft", and "sft" are de facto supported, but it's ultimately left up to however the driver wants to interpret the string. Which is not necessarily a bad thing; again, a driver may want to implement a custom unit type for special situations or for bands that are not elevations. When there is an explicit band type for elevations, however, I recommend the following unit label strings be canonized (based on a union of PROJ.4 units, units_of_measure.csv, etc. Measure authorities are mostly Wikipedia):

Label	**Description**	**Measure (meters)**	**EPSG code**
"ft"	foot	0.3048	9002
"sft"	survey foot	0.304800609601219	9003
"yd"	yard	0.9144	
"syd"	survey yard	0.914401828803658 (PROJ.4)	
"m"	meter	1.0	9001
"fath"	fathom	1.8288	9014
"rd"	rod	5.0292	

Other units that are less common or not really used but might be good to include for ground measures:

Label	**Description**	**Measure (meters)**	**EPSG code**
"li"	link	0.201168	
"sli"	survey link	0.201168402336805	9034
"sp"	span	0.2286	
"dam"	decameter	10.0	
"dkm"	dekameter	10.0	
"ch"	chain	20.1168	
"sch"	survey chain	20.1168402336805	9033
"hm"	hectometer	100.0	
"f"	furlong	201.168	
"km"	kilometer	1000.0	9036
"mi"	mile	1609.344	9093
"smi"	survey mile	1609.34721869444	9035
"nmi"	nautical mile	1852.0	9030
"kmi"	nautical mile	1852.0 (PROJ.4)	9030

There are of course countless other units, but, as well noted in PROJ.4, they are historical artifacts and the benefit of their inclusion would almost certainly outweigh the inconvenience.

Writing Ranged Data

Writing elevations is problematic in that the library client (e.g., application) must do the encoding down to the raw physical pixel format used by the driver. The driver indicates the type of pixels, but this is not enough information to safely do the encoding. Symptoms include truncating, improper rounding to integers, and inability for the driver to encode required header information. The usual solution is for the client to prebuild a list of driver info and explicitly encode pixels based on the characteristics of each. However, this does not solve the problem of header data, which only the driver can really know (and the client should not). Driver write support (e.g., implementations of `driver::Create`) isn't as strong as read support, and this may be part of the reason why.

This problem is not limited to elevations. In fact, it needs to be generalized to apply to any type of data that is logically out-of-range of the physical pixel type (and a scale+offset mapping won't suffice).

Recommendations include:

- Provide pixel data to drivers in logical form, i.e., as a block of floating-point values (Leveller, e.g., uses float64 tiles).
- Provide the driver with the logical pixel value range (e.g., symmetrify `band::GetMinimum/Maximum`).
- Have `band::SetNoDataValue` indicate logical void values.
- Have `band::SetOffset` and `band::SetScale` be moot for writing.

Some of these points were addressed successfully in the Terragen™ driver, but required using nonstandard metadata and asymmetric pixel types (i.e., reading int16, writing float32). The asymmetry is not avoided even with the recommendations, of course, but formalizing it makes it possible for roundtripping code to succeed.

Another possibility for the first item is to have drivers override a virtual `band::LogicalToPhysical` method. Here, data provision occurs as before, except the client calls `LogicalToPhysical` to map logical pixel values to physical ones.

Research Notes/References

GDAL library
http://www.remotesensing.org/gdal/

OGR library
http://www.remotesensing.org/gdal/ogr/

PROJ.4 library
http://www.remotesensing.org/proj/

Wikipedia, articles on SI units, linear measures, etc.
http://en.wikipedia.org/wiki/SI

The NASA GSFC and NIMA Joint Geopotential Model
http://cddis.nasa.gov/926/egm96/egm96.html

WKT (Well-Known Text) format
http://geoapi.sourceforge.net/snapshot/javadoc/org/opengis/referencing/doc-files/WKT.html

Vertical Datums, Elevations, and Heights
http://www.usna.edu/Users/oceano/pguth/website/so432web/e-text/NIMA_datum/VERTICAL%20DATUMS.htm

GeoTIFF specification
http://www.remotesensing.org/geotiff/spec/geotiffhome.html

POV-Ray raytracing renderer
http://www.povray.org

Terragen GDAL driver
http://www.daylongraphics.com/products/leveller/download/gdal_driver_terragen.zip