[PD-05-033] GDAL/OGR and Geospatial Data IO Libraries

Abstract

Manipulating (e.g., reading, writing, and processing) geospatial data, the first step in geospatial analysis tasks, is a complicated step, especially given the diverse types and formats of geospatial data combined with diverse spatial reference systems. Geospatial data Input/Output (IO) libraries help facilitate this step by handling some technical details of the IO process. GDAL/OGR is the most widely-used, broadly-supported, and constantly-updated free library among existing geospatial data IO libraries. GDAL/OGR provides a single raster abstract data model and a single vector abstract data model for processing and analyzing raster and vector geospatial data, respectively, and it supports most, if not all, commonly-used geospatial data formats. GDAL/OGR can also perform both cartographic projections on large scales and coordinate transformation for most of the spatial reference systems used in practice. This entry provides an overview of GDAL/OGR, including why we need such a geospatial data IO library and how it can be applied to various formats of geospatial data to support geospatial analysis tasks. Alternative geospatial data IO libraries are also introduced briefly. Future directions of development for GDAL/OGR and other geospatial data IO libraries in the age of big data and cloud computing are discussed as an epilogue to this entry.

Keywords: abstract data model, coordinate transformation, format conversion, FOSS, free and open source software, geoprocessing, geospatial data, programming languages and libraries, raster data, vector data

Author & citation

Qin, C-Z. and Zhu, L-J. (2020). GDAL/OGR and Geospatial Data IO Libraries. The Geographic Information Science & Technology Body of Knowledge (4th Quarter 2020 Edition), John P. Wilson (Ed.). DOI:10.22224/gistbok/2020.4.1.

Explanation

1. Definitions
2. Why do we need geospatial data IO libraries?
3. What is GDAL/OGR?
4. Why GDAL/OGR?
5. How do we use GDAL/OGR?
6. Alternatives to GDAL/OGR: other geospatial data IO libraries
7. Future directions

1. Definitions

Geospatial data: Data with a spatial reference that identify geographic features, locations, dimensions, attributes, temporal information, etc. They can be accessed, processed, analyzed, and visualized for geospatial applications, which are often conducted

using geospatial tools (from all-purpose geographic information system (GIS) software to specific tools). Raster data and vector data are the two main types of geospatial data. Each data type may be organized and saved using various data structures and different data formats.

Raster data: A machine-readable data type that represents continuous or categorical data with a gridded structure over the geographic space, in which each pixel (or grid cell) takes on a representative value, such as an average, of the attribute of interest within the geographic space covered by the pixel. Digital images and gridded digital elevation models (DEMs) are examples of raster data.

Vector data: A machine-readable data type used to represent the location and shape of geographic features as geometrical objects (e.g., points, lines, and polygons) with explicit spatial coordinates. For each individual geometrical object, the values of its spatial and non-spatial attributes can also be recorded.

Spatial reference system (SRS): Also called a coordinate reference system (CRS), this is a fundamental component of all geospatial data, which uses coordinates to help describe where pixels or geometrical objects are located in the real world. Most SRSs can be divided into two categories: geographic and projected. A geographic SRS (also known as a geographic coordinate system) uses an ellipsoidal surface to define locations on the Earth (such as the well-known WGS84), whereas a projected SRS uses a flat Cartesian surface based on a map projection with a geographic SRS as the geodetic datum (such as the Web Mercator based on WGS84). See Map Projections and other topics in Georeferencing Systems within Data Management.

Software library: A software library provides a set of pre-written code (encapsulated as functions, classes, modules, scripts, etc.) for implementing specific, fine-grained functionalities (e.g., manipulating strings) which can be reused in code. By calling or invoking the codes in a software library (often written by others), developers can save time and effort and ensure quality during application development.

Command line utilities: A classic way to run a program by directly typing its corresponding command in the display space of an all-text command line interface (CLI; for example, bash shell on Linux and macOS systems, or cmd/PowerShell on Windows systems). One simple and frequently used example of a command line utility provided by operating systems is the command dir (or ls, depending on operating system) for listing contents in the current directory.

2. Why do we need geospatial data IO libraries?

Reading and writing geospatial data is the most fundamental procedure in geospatial analysis applications. However, it is also a complicated step when facing diverse types and formats of geospatial data, especially when combining them with diverse spatial reference systems (SRSs). Raster data and vector data are two main types of geospatial data. Each data type has various data formats, due to the diverse rationales behind their development (e.g., specific domains, vendors, applications, etc.).

Geospatial data Input/Output (IO) libraries are designed and made available to assist with

developing procedures for reading and writing geospatial data in a simple and unified way. These libraries hide the technical implementation details of IO from end users and developers, so as to provide concise application programming interfaces (APIs) for users to call or invoke. With such a library, developers can concentrate more on the core algorithms for geospatial analysis instead of the complicated details of opening, reading, writing, and closing geospatial data with diverse formats and SRSs. This facilitates and vastly simplifies the development of new geospatial applications.

Geospatial data IO libraries include specific libraries and universal libraries. A specific library may serve for one specific geospatial data format, for example, libgeotiff for GeoTIFF, libhdf5 for HDF5, or shapelib for ESRI Shapefile. A universal library is generally built on an abstract data model (i.e., abstract classes in programming code such as C/C++) to support diverse geospatial data formats, for example, GDAL/OGR (Geospatial Data Abstraction Library/OpenGIS Simple Features Library) for raster and vector geospatial data, or PDAL (Point Data Abstraction Library) for point cloud data.

This entry introduces the most popular and versatile universal geospatial data IO library, GDAL/OGR, as example of geospatial data IO libraries, and discusses its design, usage, and future directions.

3. What is GDAL/OGR?

GDAL/OGR is an open source geospatial data IO library with a set of command line utilities that is released under an MIT/X-style open source license. GDAL/OGR, “a translator library for raster and vector geospatial data formats” per the definition on its website, https://gdal.org/, focuses on the translation of raster and vector geospatial data with different data formats and map projections adopted in practice.

GDAL was started by Frank Warmerdam in 1998 and has been officially maintained by the Open Source Geospatial Foundation (OSGeo) since 2008. Frank Warmerdam also created OGR, a separate IO library for vector data inspired by the OpenGIS Simple Feature specification. While GDAL was specific to raster geospatial data at the outset, OGR has been officially integrated with GDAL since GDAL version 2.0 in 2015.

The latest version of GDAL, 3.1.2 released on July 2020, provides as many as 168 specific implementations (called drivers) based on a single raster abstract data model to support different raster data formats and protocols (such as JPEG, GeoTIFF, Web Map Services (WMS), and so on), whereas OGR provides as many as 99 drivers based on a single vector abstract data model for vector data formats (such as Computer Aided Design (CAD), ESRI Shapefile, Web Feature Service (WFS), etc.) (GDAL/OGR contributors, 2020).

4. Why GDAL/OGR?

For both developers of the Free and Open Source for Geospatial (FOSS4G for short) community and end users who need to handle geospatial data, GDAL/OGR provides considerable utility to reduce their burden when working with the diverse geospatial data formats used in practice (Figure 1) (Warmerdam, 2008). GDAL/OGR has become a

cornerstone of geospatial data IO in various FOSS4G software (such as GRASS GIS, QGIS, and SAGA) and proprietary software (such as FME and ArcGIS), due to 1) its high compatibility with various geospatial data formats and SRSs, 2) the flexibility of programming interfaces, and 3) efficiency for processing large-volume geospatial data.

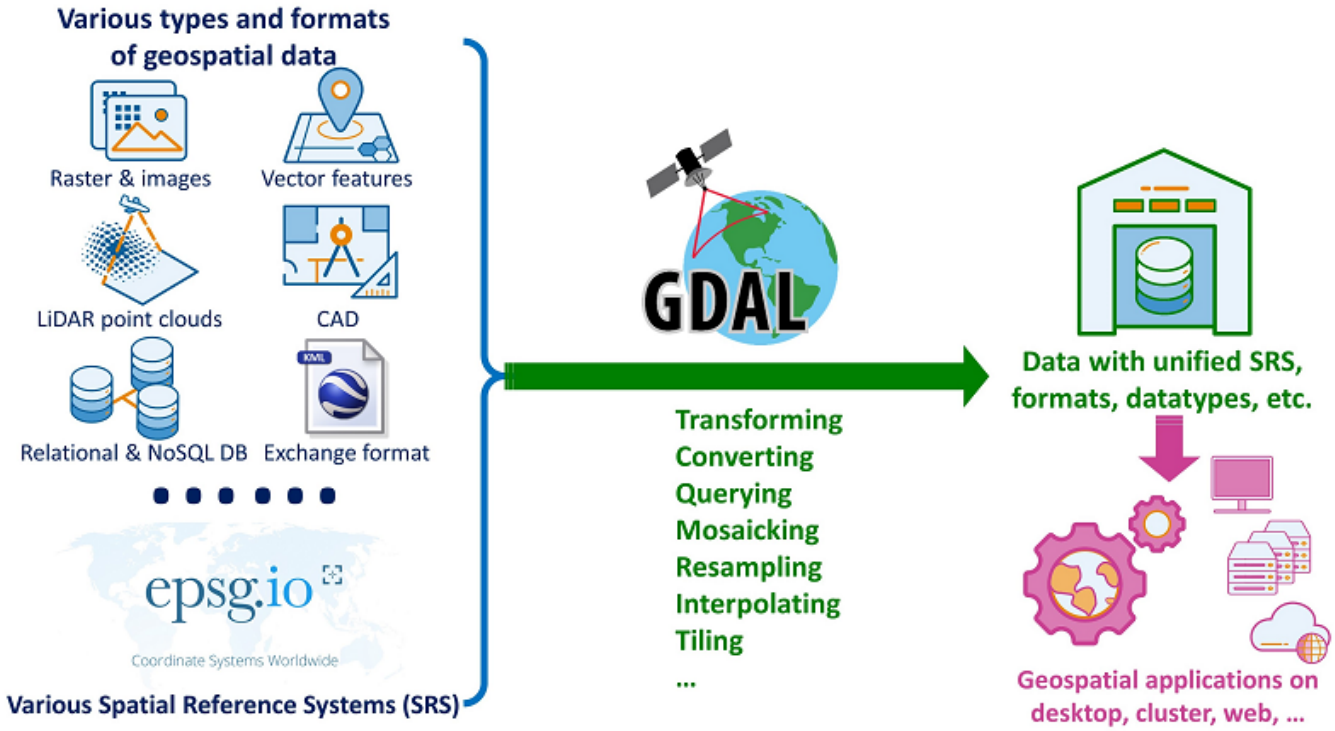


Figure 1. GDAL/OGR reduces user burden when working with diverse geospatial data formats and spatial reference systems in geospatial applications. Source: authors.

4.1 High compatibility with geospatial data formats and spatial reference systems (SRSs)

Based on the design of its spatial reference systems and single raster and vector abstract data models (Figure 2), GDAL/OGR implements a variety of data drivers to load and manage all kinds of geospatial data with the supported formats (covering most, if not all, of the geospatial data formats in practice) through unified interfaces for users.

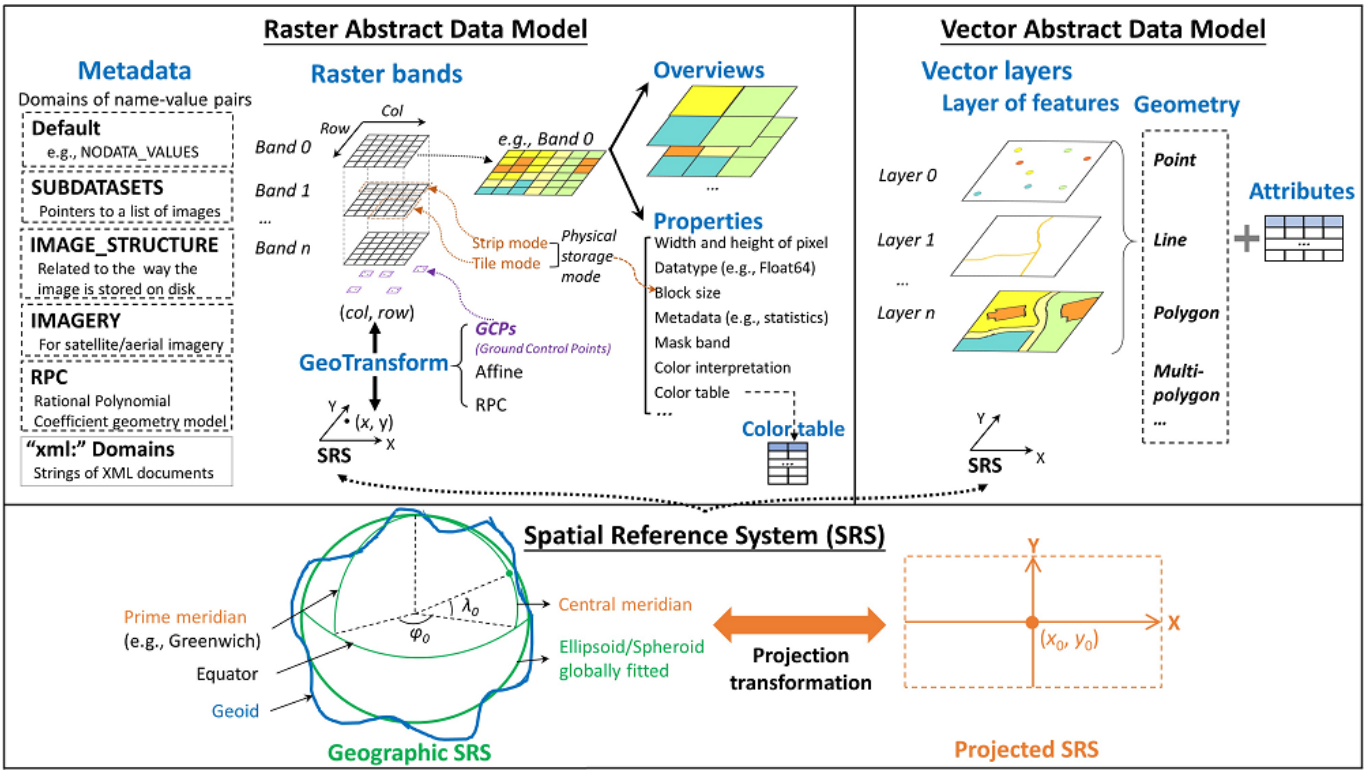


Figure 2. Schematic diagram of the raster and vector abstract data models, as well as the spatial reference system (SRS) in GDAL/OGR. SRS is the fundamental component of all geospatial data and contains two major types: geographic and projected. The raster abstract data model contains two key components: raster bands and metadata. Each raster band has a set of properties and may have zero to multiple overviews. The vector abstract data model mainly includes vector layers of features. Each feature is comprised of one type of geometry and the associated attributes. Source: authors.

4.1.1 Spatial reference systems in GDAL/OGR

The spatial reference system (SRS) or coordinate reference system (CRS) is the fundamental component of all geospatial data. Geospatial data covering the same geographical area but derived from different sources may have distinct SRSs, according to SRS types (e.g., geographic SRS and projected SRS), the cartographic projection method and its parameter settings, and geodetic datum based on an Earth reference ellipsoid, etc. (Figure 2; interested readers may search SRSs by location via https://epsg.org/search/map). Transformations from one SRS to another are vitally important when handling geospatial data and can be highly complicated considering the conversion precision (PROJ contributors, 2020).

In GDAL/OGR, SRSs are represented in the OpenGIS Well Known Text (WKT) format to be compatible with European Petroleum Survey Group (EPSG) codes for predefined SRSs, ESRI WKT format, etc. GDAL/OGR not only supports all SRSs defined in the EPSG and ESRI databases, but also allows users to define new SRSs by setting key parameters (including ellipsoid/spheroid name with semi-major axis and inverse flattening, prime meridian name and offset from Greenwich, projection method and parameters (e.g., central meridian), unit name and conversion factors to meters or radians, and so on). It is worth mentioning that common spatial databases supported by GDAL/OGR generally have three ways to handle

SRSs: (1) use GeoJSON to store geospatial data and thus only support the SRS of WGS 84 (i.e., EPSG 4326), in the manner adopted for MongoDB; (2) use a specific SRS syntax and accept user-defined SRSs, in the manner adopted for DB2 Spatial Extender; and (3) pre-define SRSs based on EPSG database and then adopt OpenGIS WKT representation or variants, as is done in Oracle Spatial, PostGIS, and MySQL.

GDAL/OGR adopts a powerful generic coordinate transformation engine (the PROJ library; PROJ contributors, 2020) as the built-in engine for conducting projections and transformations between different SRSs. This allows GDAL/OGR to perform both cartographic projections on large scales and coordinate transformation with high geodetic precision for most of the SRSs used in practice. GDAL/OGR keeps almost synchronous updates with PROJ. As of writing this entry, the latest release of PROJ 7.1.0 on July 1st, 2020 has been supported by the latest GDAL/OGR 3.1.2 released on July 7th, 2020.

4.1.2 GDAL raster abstract data model

The GDAL raster abstract data model has high compatibility with most (if not all) raster data formats and protocols in practice, including georeferenced images (e.g., GeoTIFF), relational databases (e.g., Oracle Spatial and PostGIS), portable databases (e.g., Rasterlite), and so on (Figure 1).

Besides SRS, two key components are included in the GDAL raster abstract data model: raster bands and metadata (Figure 2). Raster bands are composed of a group of individual raster bands with the same range of image coordinates (i.e., columns and rows). Each individual raster band logically represents a single band, channel, or layer of the raster data as a two-dimensional image using columns and rows. A set of properties is associated with each raster band, e.g., the width and height of each cell, block size related to physical storage mode, statistics stored in metadata, etc. Each individual cell position (col, row) in a raster can be transformed to georeferenced coordinates (x, y) or (longitude, latitude) via: (1) ground control points (GCPs), (2) affine transformation based on the coordinates of the top left corner of the raster along with the width and height of the cell, and (3) the rational polynomial coefficient geometry model (RPC), if available. In addition, each raster band may have zero or multiple overviews that have different sizes (in columns and rows) but cover the same geographic region (Section 4.3).

Metadata in the GDAL raster abstract data model provide an auxiliary data structure for storing application-specific textual data as a list of name/value pairs, which are split into several groups (called domains; Figure 2). These domains include the “Default” domain for recording items with well-defined semantics (e.g., NODATA values), the “SUBDATASETS” domain for recording the pointers to a list of images within a single multi-dimensional raster (e.g., HDF5 (Hierarchical Data Format 5), a popular format in scientific community for storing multi-object data structures such as raster images and multidimensional arrays), the “IMAGE_STRUCTURE” domain for recording the method of storing the raster image on disk (e.g., compression type), the “IMAGERY” domain for remote sensing data, if available, and the “xml:” domains for storing XML documents.

4.1.3 OGR vector abstract data model

Through a unified interface for users, the OGR vector abstract data model supports many commonly-used vector data formats, including GIS-specific formats (e.g., ESRI Shapefile and Geodatabase, and Mapinfo), CAD (e.g., DXF), RDBMS (e.g., PostGIS, Oracle), NoSQL databases (e.g., MongoDB), portable databases (e.g., Spatialite), exchange formats (e.g., KML, and GeoJSON), WebService (e.g., WFS, and CartoDB), and so on (Figure 1).

The OGR vector abstract data model mainly includes vector layers and their SRS (Figure 2). Vector layers represent a set of geometrical objects (called features) with one or more layers. Each layer can have multiple features which commonly consist of one type of geometry class (e.g., polygons for representing administrative districts) as well as a set of attributes (e.g., the population of each individual administrative district). As the core elements of the OGR vector abstract data model, the geometry classes encapsulate the OpenGIS model vector data and represent various kinds of vector geometry (including point, line, polygon, multi-point, multi-line, multi-polygon, etc.).

The ability of vector data to represent spatial relationships between geometry objects, especially topological relationships (e.g., connectivity between lines, adjacency between polygons, and enclosure among nested polygons), is one significant advantage when performing geospatial analysis (e.g. network analysis). Some existing vector data formats, such as TopoJSON and GRASS GIS vector format, support this ability by explicitly declaring and storing topological relationships. However, the OGR vector abstract data model does not natively support the creation or preservation of topological relationship data, although this functionality had been proposed as one potential future direction of GDAL/OGR development (Rouault, 2017). Therefore, OGR can currently read vector data from those formats which store topological relationships, but cannot write the topological relationship data into those formats.

OGR natively adopts SQL (Structured Query Language) to handle geometry objects in vector data, such as selecting and updating objects by attributes or geometry types. Alternatively, it is possible to use the SQLite dialect, and thus benefit from spatial functions provided by the spatial extensions of SQLite—Spatialite, e.g., querying geometry objects that intersect with a given rectangle area. In addition, for vector data that store layer attributes in a database, OGR can achieve richer functionality by passing the SQL commands to the underlying database driver.

4.2 High Flexibility of Programming Interfaces

Warmerdam (2008) stated, “libraries should make interfaces such as error handling, and file system IO hookable for maximum application control,” and this directive is implemented in the GDAL/OGR library. GDAL/OGR provides developers plenty of fine-grained APIs to precisely control the general IO workflow, such as opening the data file (or database), fetching a raster band or vector layer, reading data, creating the output file, and releasing resources. Developers can not only invoke these APIs to manipulate geospatial data with the supported formats, but also create their own drivers for application-specific data formats (interested readers may refer to the GDAL/OGR official tutorials of implementing raster driver and vector driver).

Since it is written in C/C++ language, GDAL/OGR is inherently cross-platform compatible and portable; thus, it is able to run on almost any mainstream system (including Linux,

Windows, macOS, Andriod, etc.). GDAL/OGR APIs can be used directly from C/C++ and can also be “wrapped” for use with other prevailing programming languages (Python, Perl, VB, C#, Java, and so on).

4.3 High efficiency of processing large-volume geospatial data

As geospatial applications must handle increasingly large volumes of data, GDAL provides three main techniques to more effectively access or browse a raster band with a vast amount of data: overviews, tiles, and pyramids. An overview is optionally generated by downsampling the raster band to a coarser resolution and can be stored in an external file (e.g., a file with the suffix “.ovr” for GeoTIFF) or included within the raster data (Figure 2). The overview is useful for rapidly displaying the underlying raster band with a coarser resolution. Tiles are designed as subsets of the raster band at the same resolution and saved as separate files that can be accessed more efficiently for subareas of interest within the raster band, which is especially useful in the context of web service applications. Pyramids are combinations of tiles and overviews at different levels. At each level, the raster band with a coarsened resolution (i.e., the overview at that level) is divided into a number of tiles. All tiles have the same size, regardless of level (e.g., 256 × 256 cells). Pyramids are useful for accelerating zoom in/out operations in GIS applications such as ArcMap and Google Earth.

Designed for high-volume, network-hosted raster data, GDAL virtual file systems support the direct access of data stored on a network rather than downloading the entire dataset to a local disk, which is important for effectively utilizing the open geospatial data that are increasingly available on the web. Since version 3.1, GDAL has supported Cloud Optimized GeoTIFF (COG), which is a regular GeoTIFF with an additional external file storing necessary metadata, overviews, and tiling information to effectively locate the data required for geospatial processing without a specialized server.

As for vector data, the OGR supports spatial indexing based on quadtree to accelerate access (i.e., querying) for spatially filtered data. For example, the OGR Shapefile driver uses a spatial indexing file (.qix), which has the same format as that in MapServer.

Besides the highly efficient accessing and exploration of geospatial data introduced above, two important mechanisms were designed for GDAL/OGR to efficiently process large-volume geospatial data (e.g., conducting raster algebras): block caching and virtual memory mapping (the latter is currently only implemented for Linux). In the mechanism of block caching, a block corresponds to the successive physical storage size of raster band data on the disk, which can be strips of one or more scanlines or tiles with uniform height and width (Figure 2). By exposing the block size to GDAL, large raster data can be efficiently read and processed block-by-block with minimal overhead. When users want to perform global, zonal, and focal calculations on large raster data, the regular practice of loading the whole data into memory may fail if the RAM capacity of the system is exceeded. To solve this issue, the virtual memory mapping mechanism allows for an array to be created transparently without reading any cell values, then data corresponding to areas of interest and neighboring cells are fetched as necessary. The fetched data are kept in a RAM cache (10 MB by default) with a least-recently-used strategy.

5. How Do We Use GDAL/OGR?

The three direct methods of using GDAL/OGR are (1) the programmatic method, i.e., in which GDAL/OGR APIs are invoked to access geospatial data; (2) the command-line method, in which specific GDAL/OGR utilities are executed to process geospatial data in a command-line environment; and (3) the GUI method, in which GDAL/OGR utilities are encapsulated as user interfaces in GIS software (e.g., QGIS) in order to assemble the required commands and execute them in the background. Figure 3 depicts a simple example of calculating statistics for raster bands in GeoTIFF data using each of these three methods. Using the programmatic method, three main APIs are used in the example Python code snippet, i.e., open data file, obtain single raster band data, and obtain/calculate statistics for this raster band (Figure 3). Using the command-line method, a GDAL utility command (gdalinfo) prints out detailed statistics for each raster band as well as other metadata of the GeoTIFF data (Figure 3). The GUI method provides a user-friendly guide to fulfilling the command-line method but without error-prone manual parameter-setting (Figure 3).

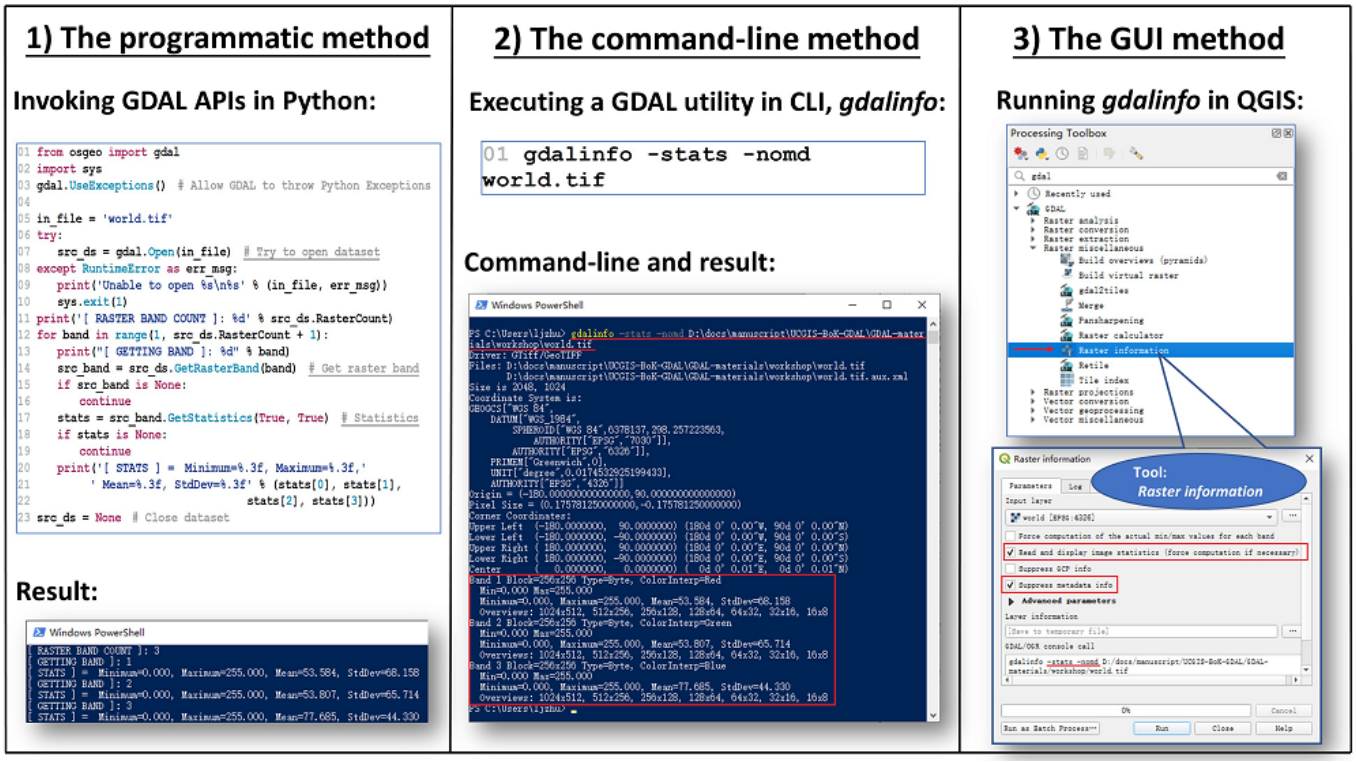


Figure 3. Three direct methods for using GDAL/OGR: an example of calculating statistics of raster bands in GeoTIFF data. Source: authors.

In general, the programmatic method allows developers using GDAL/OGR APIs to control geospatial data in their problem-specific applications in a precise, but perhaps tedious, manner. In the command-line method, GDAL/OGR utilities efficiently and concisely process geospatial data within their limited functionalities. The GUI method can effectively relieve users (especially users unfamiliar with command-line interfaces) of the burden of learning how to directly utilize GDAL/OGR utilities and their corresponding parameters.

Table 1 briefly lists four main categories of current GDAL/OGR utilities for manipulating geospatial data, including discovering data, unifying data formats and spatial reference

systems, DEM processing, and optimizing access to geospatial data. Interested readers can find a full list on https://gdal.org/programs/index.html.

Table 1. Four Main Categories of GDAL/OGR Utilities for Manipulating Geospatial Data

Category	Examples of Primary Application Context (and Related Utilities)
Discovering Data	• Listing information (metadata, statistics, field lists, etc.) of raster (gdalinfo) or vector (ogrinfo) geospatial data. • Querying the information of one cell (e.g., location and values) in raster data (gdallocationinfo) or the geographic features of a vector data selected by bounding box or SQL expression (ogrinfo). • Extracting raster data in a sub-window (gdal_translate) or vector data through SQL expression (ogr2ogr).
Unifying Data Format and Spatial Reference System	• Converting data formats (gdal_translate and ogr2ogr). • Georeferencing, reprojection, and transformation for a consistent SRS (gdalwarp, gdaltransform, and ogr2ogr). • Polygonizing from raster (gdal_polygonize) or rasterizing from vector geometries (gdal_rasterize). • Resampling and mosaicking of raster data (gdalwarp, gdal_merge, and gdalbuildvrt) or merging vector data (ogrmerge).
DEM Processing	• Creating a DEM from scatter data (gdal_grid) or contour lines (gdal_rasterize + gdal_fillnodata.py, demo data and commands can be found from this link). • Filling voids (or NODATA cells) in a raster data as DEM (gdal_fillnodata). • Deriving contour lines (gdal_contour) or common topographic attribute datasets (e.g., hillshade, slope, aspect, and viewshed) (gdaldem and gdal_viewshed) from a DEM.
Optimizing Access to Geospatial Data	• Using a specific physical storage mode (i.e., tile mode or strip mode; see Figure 2) to storage raster data (gdal_translate). • Generating tiles of raster (gdal2tiles) and vector (ogrtindex) data for WebService. • Creating overviews for raster data (gdaladdo).

6. Alternatives to GDAL/OGR: Other Geospatial Data IO Libraries

From the perspective of supporting various geospatial data formats, some of the power of GDAL/OGR can also be attributed to the fact that it is coupled or integrated with other existing credible geospatial data IO libraries for specific data formats. Among these are three libraries which have been major contributors to GDAL since its early stages: libtiff, libgeotiff, and shapelib. These libraries can be substituted for GDAL/OGR in programming cases where only their specific formats need be considered, such as the libhdf5 library for the HDF5 format, libnetcdf for NetCDF (Network Common Data Form), and libspatialite for Spatilite.

Some geospatial data IO libraries that might be more suitable for beginners have also been developed based on GDAL/OGR, especially for developers using other languages (e.g., Python). GDAL/OGR APIs that have been wrapped for other languages (e.g., Python) expose as many detailed interfaces as the original C/C++ APIs by directly invoking the corresponding C functions. Because the wrapped APIs provide very limited abstraction for their C APIs, the invoking code may be tedious and nonintuitive. For example, the procedure for obtaining a count of features in a Shapefile with GDAL/OGR is reminiscent of

a C++ coding style, rather than a style build around manipulating Python file-like objects (e.g., using with statement). In addition, developers have to consider memory management issues induced by the use of the C language (e.g., explicitly releasing resources after being used). This situation is unfriendly to developers not familiar with C/C++. Therefore, some libraries based on GDAL/OGR have been constructed, such as Python libraries Rasterio and Fiona, which focus on raster and vector data, respectively. They use more idiomatic Python data types, functions, and classes and thus might be friendlier to beginners.

7. Future Directions

Recent development and usage trends for GDAL/OGR have been driven by the increasing volume of geospatial data (especially those large volumes of data available on the web) rather than growth in the number of available data formats, and this is likely to continue into the future. The resource-intensive data IO processes needed to handle these large data sets have been and will continue to be a bottleneck for geospatial analysis applications. In the age of big data and cloud computing, high performance should be a primary concern, so that geospatial analysis applications can take full advantage of cyberinfrastructure and make complex geospatial analysis tasks easier for users.

Parallel data IO is one direct and long-standing solution for the bottleneck problem. Parallel data IO allows multiple process units, which can vary between different parallel computing models (e.g., threads in the open multiprocessing (OpenMP) model available for multiple-core processors, or processes in the message passing interface (MPI) model available for a computer or cluster), to read and write their data concurrently. Since one dataset within GDAL/OGR cannot be used by multiple threads at the same time (i.e., GDAL/OGR is thread-unsafe), some efforts towards achieving parallel data IO at the process level (Qin et al., 2014; Tarboton, 2016; Miao et al., 2017) have been made in recent years to implement MPI-based parallel data IO in GeoTIFF format, which is one of the mostly used geospatial raster data formats. Qin et al. (2014) showed that when conducting column-wise or block-wise domain decomposition, the parallel data IO using GDAL is highly inefficient and cannot produce a correct output. They proposed a two-phase IO strategy to overcome this problem by mapping and distributing data between multiple processes before the writing operations start (Qin et al., 2014). This strategy has only been tested on raster data stored in strip mode and is only applicable for static load balancing. Miao et al. (2017) proposed a dynamic scheduling strategy through storage mapping and data arrangement techniques that can synchronize data IO for raster data stored using strip or tile mode.

Future directions for GDAL/OGR development may include: (1) resolving the thread-safe problems of GDAL datasets for as many drivers of common geospatial data formats as possible; and (2) natively supporting parallel data IO on different kinds of computing machines, especially the distributed cloud computing environment, to promote high-performance cloud-based geospatial analysis applications and allow for on-the-fly geospatial processing. This latter direction should be of particular focus for GDAL/OGR as well as other potential IO libraries.

References

GDAL/OGR contributors (2020). GDAL/OGR Geospatial Data Abstraction software Library.

Open Source Geospatial Foundation.

Miao, J., Guan, Q., & Hu, S. (2017). pRPL + pGTIOL: The marriage of a parallel processing library and a parallel I/O library for big raster data. Environmental Modelling & Software, 96, 347–360.

PROJ Contributors. (2020). PROJ coordinate transformation software library. Open Source Geospatial Foundation.

Qin, C.-Z., Zhan, L.-J., & Zhu, A.-X. (2014). How to Apply the Geospatial Data Abstraction Library (GDAL) Properly to Parallel Geospatial Raster I/O? Transactions in GIS, 18(6), 950–957.

Rouault, E. (2017). GDAL 2.2 – What's new? FOSS4G EUROPE 2017, July 18-22, Paris.

Tarboton, D.G. (2016). Terrain Analysis Using Digital Elevation Models (TauDEM 5.3.7). Accessed 19 October 2016.

Warmerdam, F. (2008). The Geospatial Data Abstraction Library. In G. B. Hall & M. G. Leahy (Eds.), Open Source Approaches in Spatial Data Handling (pp. 87–104). Springer.