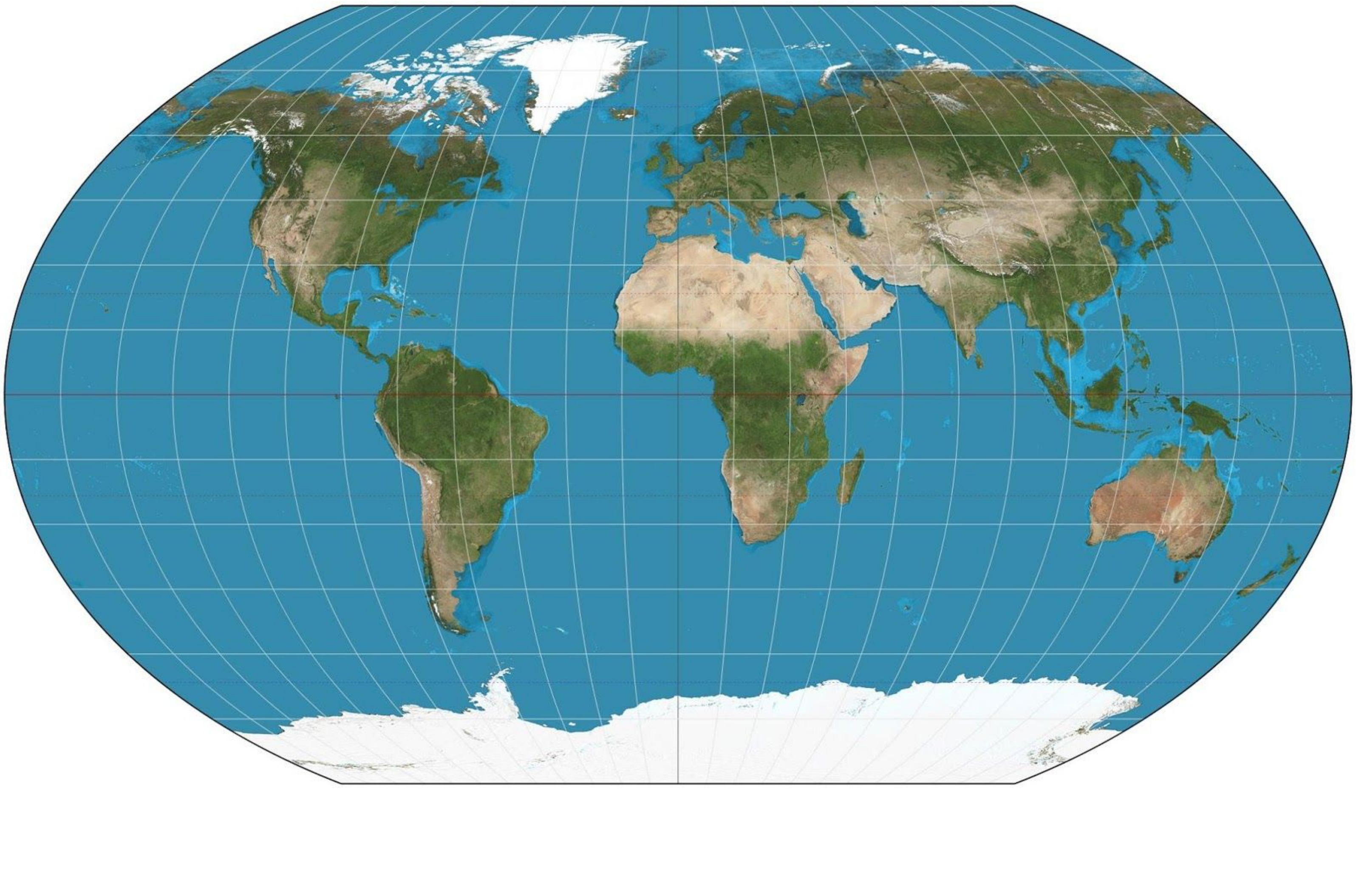
GIS: An ArchGDAL Overview

By Soumitra Shewale



1.1 What are GIS databases?

GIS stands for Geographic Information System. GIS databases are databases that contain geographical data. For instance, a GIS database could contain location information about crimes and could help analyze previous crimes and prevent them in the future. Another example could be visualizing the outbreak of a disease, recognizing patterns in the data and preventing or limiting the spread of the disease.

GIS databases also contain layers that can be accessed to collect different types of information for a given place. For example, we can visualize rainfall distribution, wind speeds, and the average temperature at a place and that could yield important meteorological results. All these different parameters would exist in different layers since they represent different types of data.

GIS can give us deeper insights about issues that challenge the world, predict future events, help us understand trends and so much more. It truly is a versatile form of data that has a lot of potential.

1.2 A deeper look

GIS databases are of two types: vector and raster. Vector databases contain information about discrete objects such as lines, points, areas, labels, etc. Raster databases, on the other hand, contain continuous information such as heatmaps, distribution charts, imagery, etc.

Vector databases are generally used to map and label things that have fixed boundaries and can be mapped discretely. This includes streets, states, buildings, etc.

Raster databases are generally used to visualize distributions, satellite imagery, elevations, weather data, etc.

1.3 Where does ArchGDAL come in?

GDAL stands for Geospatial Data Abstraction Library, which is an open-source library that can process GIS data. Currently, the project is being maintained by OSGeo and is being updated frequently. It is released under the X/MIT license and is mainly written in

C and C++. However, many translation libraries exist that can interface between a given programming language and GDAL. One example is ArchGDAL.jl which interfaces between GDAL and Julia. It is, in fact, a high-level API for GDAL and builds upon GDAL.jl.

1.4 The Arch principles

ArchGDAL also follows the Arch principles while developing and maintaining their library. The principles are:

- Simplicity: without unnecessary additions or modifications. (i) Preserves GDAL Data Model, and makes available GDAL/OGR methods without trying to mask them from the user. (ii) minimal dependencies
- Modernity: ArchGDAL strives to maintain the latest stable release versions of GDAL as long as systemic package breakage can be reasonably avoided.
- Pragmatism: The principles here are only useful guidelines. Ultimately, design decisions are made on a case-by-case basis through developer consensus. Evidence-based technical analysis and debate are what matter, not politics or popular opinion.
- User-Centrality: Whereas other libraries attempt to be more user-friendly, ArchGDAL shall be user-centric. It is intended to fill the needs of those contributing to it, rather than trying to appeal to as many users as possible.
- Versatility: ArchGDAL will strive to remain small in its assumptions about the range of user needs, and to make it easy for users to build their own extensions/conveniences.

These principles are adapted from the original Arch Linux principles which can be found here.

1.5 Installation

To install ArchGDAL, just type

```
import Pkg

Pkg.add("ArchGDAL")
```

To test your installation, type

```
Pkg.test("ArchGDAL")
```

1.6 Getting Started

Let us get started with working with databases in ArchGDAL.

Obviously, we need to import ArchGDAL first to work with it:

```julia
using ArchGDAL
```

To initialize ArchGDAL, we must register our drivers. To do this, simply type:

```julia
ArchGDAL.registerdrivers() do

    # your code here

end
```

Or, you could do this in 2 lines:

```julia
GDAL.allregister()

# your code here

GDAL.destroydrivermanager()
```

However, I prefer the former and will use that method throughout this document.

1.7 Working with GIS databases

To start working with a database, we must read it into our script first.

So, after we add the `read()` function, our code will be:

```julia
ArchGDAL.registerdrivers() do

    ArchGDAL.read(filepath) do dataset

        print(dataset)

    end

end
```

Where filepath is the path to our database file.

Now, it is important to note that vector databases are of type `.geojson` while Raster databases are of type `.tif`. It is also important to note that there are many types of file types other than geojson and tif that can be used with ArchGDAL such as .shp, .shx, but the example files on the examples repo for ArchGDAL only have geojson and tif files.

Now, let us import a vector dataset. On the ArchGDAL.jl repo, an example dataset already exists; It can be found here. I have placed the example dataset in the same folder as my Jupyter Notebook. To point to the dataset, I just typed the following above the `do` block:

```
filepath = "point.geojson"
```

Now, let us print some basic information about the database:

```
using ArchGDAL

filepath = "point.geojson"

ArchGDAL.registerdrivers() do

    ArchGDAL.read(filepath) do dataset

        print(dataset)

    end

end
```

We get:

```
GDAL Dataset (Driver: GeoJSON/GeoJSON)

File(s):

  point.geojson

Number of feature layers: 1
  Layer 0: point (wkbPoint)
```

We see that we get the type of the database, the files in our database, the number of layers in our database, and information about each layer. The programmatical way to access this information can be found in the docs.

Similarly, when we import the example raster database from here and run the same script on it, we get the following output:

```
GDAL Dataset (Driver: GTiff/GeoTIFF)

File(s):

  world.tif


Dataset (width x height): 2048 x 1024 (pixels)

Number of raster bands: 3

  [GA_ReadOnly] Band 1 (Red): 2048 x 1024 (UInt8)

  [GA_ReadOnly] Band 2 (Green): 2048 x 1024 (UInt8)
  [GA_ReadOnly] Band 3 (Blue): 2048 x 1024 (UInt8)
```

We got the type of file, the files in our database, the resolution of our database, and the raster bands or layers of our database.

Again, you can access this data programmatically and the instructions to do so are here.

1.8 Working with files

ArchGDAL allows the following functions to work with files:

- createcopy()
- update()
- create()
- read()

`createcopy()`: Creates a copy of the database passed as parameter.

`update()`: Opens a database with write permissions.

`create()`: Creates a new empty database.

Note: Sequential write formats (Like PNG and JPEG) do not support `create()`, but do implement `createcopy()`.

Also, some drivers only implement `create()`, but not `createcopy()`. In that case, `createcopy()` will be used as a mechanism when calling `create()`.

`read()`: Opens the database in read-only mode.

1.9 Accessing Data from Databases

To access individual fields and features (such as lines, points, labels, etc.) in vector databases, we have some methods:

`ArchGDAL.getgeomfield(feature, i)`: Gets the geometric field in the feature at index i.

`ArchGDAL.getfield(feature, i)`: Gets the field in the feature at index i

`ArchGDAL.ngeomfield(feature)`: Gets the number of geometric fields in the feature.

`ArchGDAL.nfield(feature)`: Gets the number of fields in the feature.

In raster databases, we have many methods that get the attributes of the database. Going over all these methods is out of the scope of this introduction, and the docs for these methods can be found here.

In conclusion, ArchGDAL is a very powerful tool and can be used for a variety of applications. This was only a brief exploration of the tool, and there is much more to this tool than just this. Check the repo out here, and the docs for this tool here.

1.10 References

Cover Image belongs to Strebe from Wikimedia Commons and is published under the CC-BY-SA 3.0 license. The link to it is here: https://commons.wikimedia.org/wiki/File:Kavraiskiy_VII_projection_SW.jpg

Introduction to GIS information is from ESRI: https://www.esri.com/en-us/what-is-gis/overview