

Praca z danymi
wektorowymi

OGR i Python



OGR

GDAL - to zespół bibliotek operujących na danych wektorowych, umożliwiających m.in.:

- analizy przestrzenne i matematyczne na wektorowych danych geoprzestrzennych i tablicach,
- transformację geometryczną danych geometrycznych pomiędzy układami współrzędnych,
- Odczyt i zapis w wielu formatach danych wektorowych.

Licencja X/MIT – Open Source Geospatial Foundation.

Obecne wydanie 3.2.0 z października 2020.

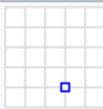
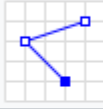
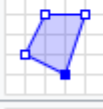
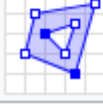
Bardzo dobrze współpracuje z Python

WKT – Well-known text

Opis geometrii lub układów współrzędnych za pomocą wyrażenia tekstowego.

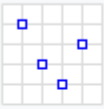
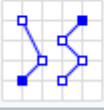
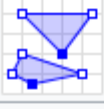

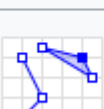
Zalety:

- interpretowalny praktycznie przez każdą bazę danych oraz program GIS
- Można go przechowywać w bazach nie wspierających geometrii a tłumaczyć podczas pobierania danych,
- Można zapisywać geometrie w plikach tekstowych i wczytywać je jak zwykłą warstwę,
- Używany często w tablicach PYTHON np. geodataframe, GDAL dataframe

Type	Examples	
Point		<code>POINT (30 10)</code>
LineString		<code>LINESTRING (30 10, 10 30, 40 40)</code>
Polygon		<code>POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))</code>
		<code>POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 35, 30 20, 20 30))</code>

Układ współrzędnych PUWG 1992 (EPSG: 2180) Zapisany w formacie WKT

```
PROJCS["ETRS89 / Poland
CS92",GEOGCS["ETRS89",DATUM["European_Terrestrial_Reference_System_1989
",SPHEROID["GRS
1980",6378137,298.257222101,AUTHORITY["EPSG","7019"]],TOWGS84[0,0,0,0,0,0,
0],AUTHORITY["EPSG","6258"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901
"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["
EPSG","4258"]],PROJECTION["Transverse_Mercator"],PARAMETER["latitude_of_ori
gin",0],PARAMETER["central_meridian",19],PARAMETER["scale_factor",0.9993],PAR
AMETER["false_easting",500000],PARAMETER["false_northing",-
5300000],UNIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Northing",NORTH],AXI
S["Easting",EAST],AUTHORITY["EPSG","2180"]]
```

Type	Examples	
MultiPoint		<code>MULTIPOINT ((10 40), (40 30), (20 20), (30 10))</code>
		<code>MULTIPOINT (10 40, 40 30, 20 20, 30 10)</code>
MultiLineString		<code>MULTILINESTRING ((10 10, 20 20, 10 40), (40 40, 30 30, 40 20, 30 10))</code>
MultiPolygon		<code>MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))</code>
		<code>MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35), (30 20, 20 15, 20 25, 30 20)))</code>
GeometryCollection		<code>GEOMETRYCOLLECTION (POINT (40 10), LINESTRING (10 10, 20 20, 10 40), POLYGON ((40 40, 20 45, 45 30, 40 40)))</code>

WKB – Well-known binary

Opis geometrii lub układów współrzędnych za pomocą wyrażenia binarnego.

Zalety:

- interpretowalny praktycznie przez każdą bazę danych oraz program GIS,
- Można go przechowywać w bazach nie wspierających geometrii a tłumaczyć podczas pobierania danych.

Format	zapis
WKT	POLYGON((205622.14219357 627910.32825126, 202494.94672709 628106.11374137, 202283.23167453 624886.41748007, 205270.92159989 624463.19496644, 205622.14219357 627910.32825126))
WKB	01030000000100000005000000e7613623b1190941958c10a88c2923410ea7e592f7b70841114f3c3a142b23410d2d78da59b108419ff2bfd5ec1123415cc36f5fb70e094127a4d2639e0e2341e7613623b1190941958c10a88c292341
GeoJson	{"type":"Polygon","coordinates":[[[205622.14219357,627910.32825126],[202494.94672709,628106.11374137],[202283.23167453,624886.41748007],[205270.92159989,624463.19496644],[205622.14219357,627910.32825126]]]}

Pierwszy bajt określa kolejność:

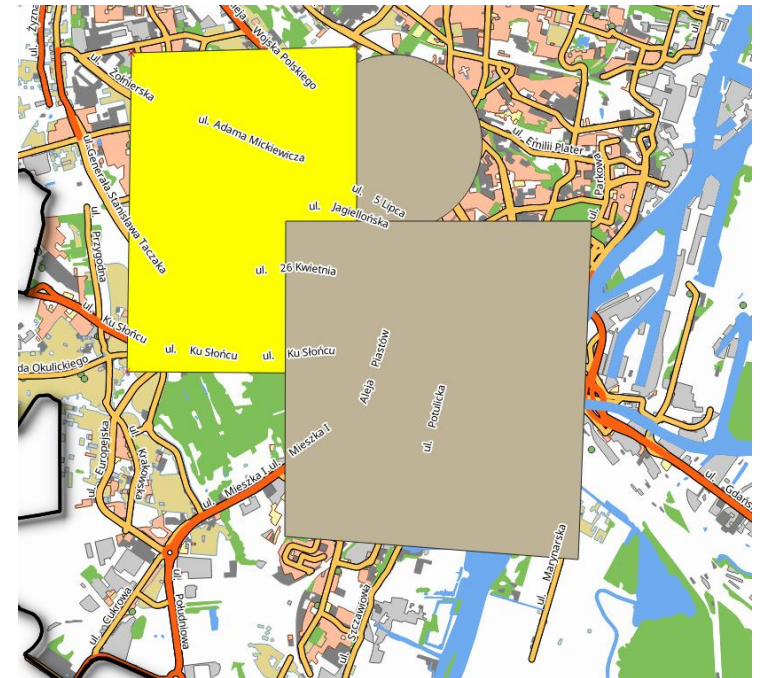
Big endian – najbardziej znaczący bajt jest na początku (00)

Little endian – najbardziej znaczący bajt jest na końcu (01)

For example, the geometry POINT (2.0 4.0) is represented as: 000000000140000000000000004010000000000000, where:

- 1-byte integer 00 or 0: big endian
- 4-byte integer 00000001 or 1: POINT (2D)
- 8-byte float 4000000000000000 or 2.0: x-coordinate
- 8-byte float 4010000000000000 or 4.0: y-coordinate

<https://en.wikipedia.org>



Definicja geometrii WKB

Konwersja do formatu WKT

Konwersja do formatu geojson

Geojson do NumPy

Numpy do geojson

GDAL przygotowanie drivera do zapisu

Zapis pliku: „plik.shp”

```
point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(14.5483, 53.4327)
```

```
wkt = point.ExportToWkt()
```

```
wkt = shapely.wkt.loads(wkt)
```

```
geojson = shapely.geometry.mapping(wkt)
```

```
array = np.array(geojson['coordinates'])
```

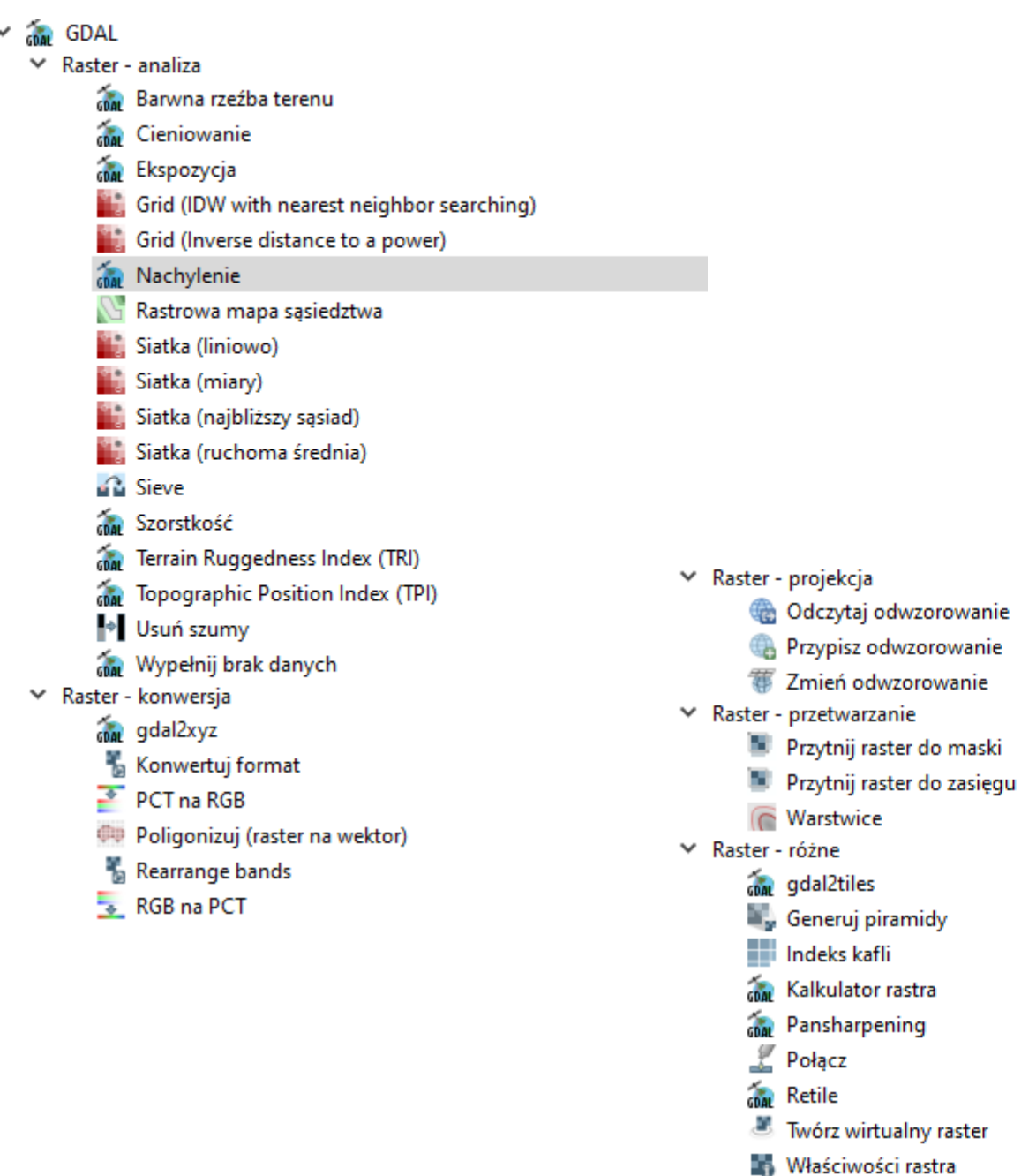
```
geojson['coordinates'] = b.tolist()
```

```
geojson2 = json.dumps(geojson)
```

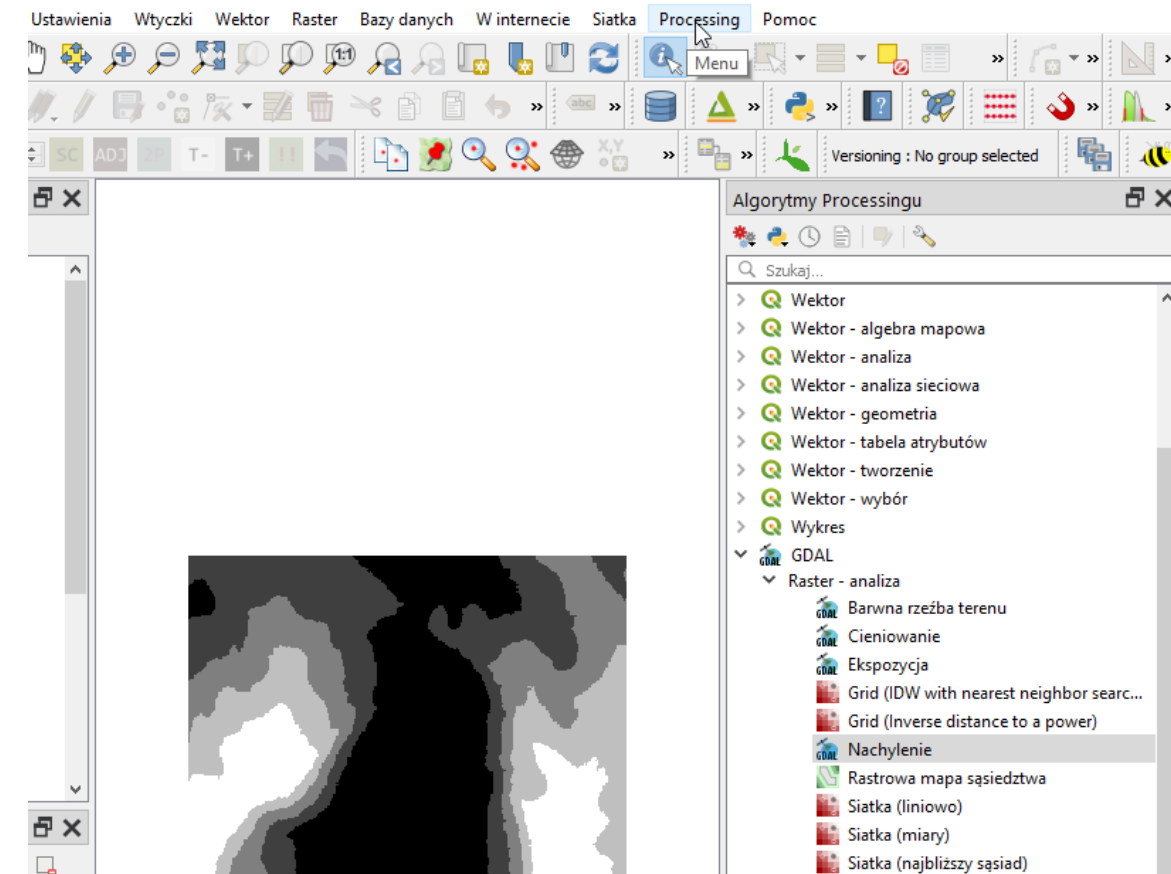
```
outDriver = ogr.GetDriverByName('ESRI Shapefile')
```

```
outShapefile = 'plik.shp'
```


















```
outDriver.CreateDataSource(outShapefile)
```



GDAL W QGIS



OGR w QGIS

- ▼  GDAL
 - > Raster - analiza
 - > Raster - konwersja
 - > Raster - projekcja
 - > Raster - przetwarzanie
 - > Raster - różne
 - ▼ Vector miscellaneous
 -  Eksportuj do PostgreSQL (nowe połączenie)
 -  Eksportuj do PostgreSQL (zapisane połączenia)
 -  Informacje o warstwie wektorowej
 -  Utwórz wirtualną warstwę wektorową
 -  Wykonaj zapytanie SQL
 - ▼ Wektor - geometria
 -  Agreguj
 -  Bufor jednostronny
 -  Bufory wektorowe
 -  Przesuń krzywą
 -  Przytnij wektor do maski
 -  Przytnij wektor do zasięgu
 -  Punkty wzdłuż linii
 - ▼ Wektor - konwersja
 -  Konwertuj format
 -  Rasterize (overwrite with attribute)
 -  Rasterize (overwrite with fixed value)
 -  Rasteryzuj (wektor na raster)

GDAL – Geospatial Data Abstraction Library

GDAL - to zespół bibliotek umożliwiających m.in.:

- analizy przestrzenne i matematyczne na danych geoprzestrzennych,
- transformację geometryczną
- Odczyt i zapis w wielu formatach danych rastrowych i wektorowych.

Licencja X/MIT – Open Source Geospatial Foundation.

Obecne wydanie 3.1.0 z maja 2020.

Bardzo dobrze współpracuje z Python

Dwa typy bibliotek:

GDAL – do analiz na rastrach

OGR – do analiz na wektorach

OGR 3.2.0

Biblioteka python do współpracy z narzędziami GDAL.

Wymaga doinstalowania:

- Libgdal w wersji co najmniej 3.1.0
- Numpy w wersji co najmniej 1.0
- Shapely.wkt

Do wizualizacji wyników można używać np. PyPLOT oraz Matplotlib

Instalacja w środowisku Python:

Pip install GDAL

Conda install -c conda-forge gdal

Główne moduły powiązane z GDAL

from osgeo import gdal #dane rastrowe

from osgeo import ogr #dane wektorowe

from osgeo import osr #układy współrzędnych

from osgeo import gdal_array #tablica gdal

Raster (1 lub n kanałów)

Wczytanie przez GDAL

Wyodrębnienie kanału do analizy

Tablica NumPy

Przetwarzanie tablicy

GDAL przygotowanie drivera do zapisu

Zapis pliku: „plik.tif”

Raster: plik.tif (1 kanał)

```
gdal.GetDriverByName('AAIGrid')
```

```
nmt=gdal.Open(plik)
```

```
nmt1 = nmt.GetRasterBand(1)
```

```
array = nmt1.ReadAsArray()
```

```
[cols, rows] = array.shape
```

```
driver = gdal.GetDriverByName("GTiff")  
outdata = driver.Create(plik, rows, cols, 1,  
gdal.GDT_Float64)
```

```
outdata.GetRasterBand(1).WriteArray(array)
```