Python GDAL/OGR programozás

Készítette: dr. Siki Zoltán

Egy mintapéldán keresztül ismerkedjünk az OGR Python programozással. A feladat megoldásához a gépre telepíteni kell a GDAL könyvtárat, Python 2.4 vagy újabb verzióját, a GDAL_Python-t.

Windows
operációs rendszer esetén legegyszerűbben az OSGeo4W telepítővel telepítheti a fenti könyvtárakat (lásd: http://trac.osgeo.org/osgeo4w/), a QGIS telepítésével minden szükséges komponens felkerül a gépére.

Fedora Linux
operációs rendszer esetén a következő parancssal telepítheti a szükséges összetevőket:

```
yum install python gdal gdal-python
```

A fenti parancshoz rendszergazda jogokkal kell rendelkeznie.

Próbáljuk ki, hogy minden rendben van a telepítéssel.

Windows
operációs rendszer esetén a programok menüből indítsa el az OSGeo4W (OSGeo4W for Windows command shell) parancsot. Ez egy parancs ablakot nyit meg, melyben a szükséges környezetet beállítja.

Fedora Linux
operációs rendszer esetén nyisson meg egy shell (terminál) ablakot.

A parancs ablakban adja ki az alábbi parancsot (az ön által beírandó parancsot dőlt betűkkel szedjük)

```
gdalinfo --formats
```

Erre egy hosszú listát kap a támogatott vektoros formátumokról, ha a GDAL telepítése sikeres volt:

```
Supported Formats:
  -> "ESRI Shapefile" (read/write)
  -> "MapInfo File" (read/write)
  -> "UK .NTF" (readonly)
  -> "SDTS" (readonly)
  -> "TIGER" (read/write)
  -> "S57" (read/write)
  -> "DGN" (read/write)
  -> "VRT" (readonly)
  -> "REC" (readonly)
... itt több sor következik még
```

Ezután nézzük meg, hogy a Python telepítés rendben van-e. Adja ki az alábbi parancsot:

```
python
```

Erre a következő választ kell kapnia, ha a Python telepítés sikerült:

```
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

A „>>>” a Python prompja, ez jelzi, hogy kész parancsok fogadására. A következőkben a parancsok előtt megjelenik a Python promt, de ezt nem kell beírnia. Előfordulhat, hogy más Python verzió van a gépén, 2.4 vagy újabb verzió megfelelő.
Végül ellenőrizzük a GDAL Python telepítését:

```
>>>import ogr
```

Ha minden rendben van akkor a Python prompt jelenik meg ezután.

Az ellenőrzés befejezése után lépjünk ki a Python értelmezőből:

```
>>>exit()
```

Kezdjünk hozzá egy konkrét probléma megoldásához és azon keresztül ismerkedjünk meg az OGR Python programozás alapjaival. A megoldandó probléma legyen a következő. Egy felület típusú Shape fájl alapján készítsünk egy a felületek centrálisát tartalmazó pont réteget. A felületek attribútumait is vigyük át a centrálisokat tartalmazó rétegbe. Tervezzük meg a programunkat, az alábbi lépéseket kell végrehajtanunk:

1. Nyissuk meg az input felület Shape fájlt
2. Hozzuk létre az eredmény pont réteget
3. Másoljuk át az attribútum definíciókat az új pont rétegbe
4. A felület réteg minden elemére
   határozzuk meg a centrális koordinátáit
   hozzunk létre egy új pont elemet a megfelelő felület attribútumaival
5. Zárjuk le az állományokat

A fenti problémát több asztali térinformatikai rendszerrel is megoldhatjuk, például a QGIS is tartalmaz ilyen funkciót. Miért van értelme Python programot készíteni a megoldásra? Egy-két rétegre jól használható a programok grafikus felhasználói felülete (GUI), de ha sok Shape fájlra kell megismételnünk a műveletet, akkor a parancssori használat előnyösebb. Parancsfájlok segítségével más programokkal is együtt használhatjuk ezt a programot. Hogy más programokba be tudjuk illeszteni a programunkat, nem lehet interaktív. Az input felületeket tartalmazó input Shape és az eredmény pont Shape nevét a parancssorban adjuk meg.

A program teljes listája a dokumentum végén található meg. Az egyes részleteket értelmezzük a következőkben, a könnyebb tájékozódás érdekében sorszámokat használunk. Igyekszünk hibatűrő programot készíteni, sokat foglalkozunk a hibák kizárásával.

```
1.  #!/usr/bin/python
```

Az 1. sornak csak Linux/Unix operációs rendszer esetén van speciális jelentése, azt mondja meg a buroknak, hogy melyik program hajtsa végre a szkriptünket. A szkript fájlunknak futtathatónak kell lennie ehhez, azaz az „x” bitet be kell állítani a fájlra. Ezt az alábbi paranccsal tehetjük futtathatóvá a szkriptünket:

```
chmod +x centroid.py
```

Ezután nem csak a

```
python centroid.py
```

paranccsal, hanem a

```
./centroid.py
```

paranccsal is elindíthatjuk a szkriptünket.

```
2.  """
3.  This Python script creates a centroid point shape for a polygon shape
4.  the shape file is the first command line parameter
5.  the output is written to the second comand line parameter
6.  """
```

A 2-6 sorokban több soros megjegyzésben röviden leírjuk a program funkcióját.

```
7.  import ogr, os, sys
8.
```

A hetedik sorban a felhasznált Python modulokat adjuk meg. Az ogr modul tartalmazza a vektoros adatok kezeléséhez szükséges objektumokat, függvényeket, az os és a sys az operációs rendszer szolgáltatásait teszi elérhetővé. Az `import ogr` formát ma már nem javasolják, helyette a `from gdal import ogr` használjuk. A kódban az `ogr.`, `os.` vagy `sys.` Résszel kezdődő utasítások a megfelelő modul elemeire hivatkoznak.

```
9.  # check parameters
10. if (len(sys.argv) != 3):
11.   print 'Usage: python centroid.py <polygon_shape> <output_point_shape>'
12.   sys.exit(1)
13.
```

A 9-13 sorokban ellenőrizzük, hogy a szkript futtatásához szükséges számú paramétert megkaptuk-e. A Python a `sys` modul `argv` listájába helyezi el automatikusan a parancssorban megadott paramétereket. A 0. elem mindig a futtatott szkript neve, ezért három elemet várunk, az 1. elem az input felület Shape, a 2. elem pedig az eredmény Shape neve. Ha a programunk háromnál kevesebb vagy több paramétert kap, a használatra vonatkozó üzenetet ír ki és kilép. Az `exit` függvény paraméterét a hívó parancsfájl megkapja, általános gyakorlat, hogy a 0 érték jelenti a szkript hibátlan lefutását, az attól eltérő érték valamilyen hibára utal. Az is általános gyakorlat, hogy a szkript a használatára vonatkozó üzenetet ír vissza, ha nem a megfelelő számú paramétert kapta meg.

```
14. # input polygon shape file
15. sname = sys.argv[1]
16. if not os.path.exists(sname):
17.   print sname + ' shape not found'
18.   sys.exit(1)
19.
```

A 14-19 sorokban az sname változóba átvesszük az input shape fájl nevét és az `os.path` objektum `exists` metódusával megvizsgáljuk, hogy a megadott Shape fájl létezik-e, ha nem hibaüzenetet ír ki a szkript és kilép.

```
20. # get the shapefile driver
21. driver = ogr.GetDriverByName('ESRI Shapefile')
22.
23. # open the data source
24. datasource = driver.Open(sname, 0)
25. if datasource is None:
26.   print 'Could not open file ' + sname
27.   sys.exit(1)
28.
```

A 20-28 sorokban beszerzünk egy hivatkozást az `ogr` modul Shape fájlok olvasására szolgáló meghajtójára (`driver` változó), majd a meghajtó segítségével megnyitjuk az adatforrást. Amennyiben a meghajtó nem tudta megnyitni az adatforrást, akkor a `None` értéket adja vissza. Ez a helyzet akkor fordulhat elő, ha a Shape fájl tartalma sérült vagy az shx vagy dbf fájl nem található.

```
29. # get the source layer
30. layer = datasource.GetLayer()
31.
```

A 29-31 sorokban az input adatforrás első (és egyben egyetlen) rétegét szerezzük be. Itt nem vizsgáltuk

a sikeres végrehajtást. A `GetLayer` metódusnak paraméterként megadhatjuk a réteg sorszámát, azokban az esetekben, amikor az adatforrás több réteget tartalmazhat. Az `ogrinfo` programmal ellenőrizhetjük, hogy egy adatforrás hány réteget tartalmaz.

```
32. # check geometry type
33. feature = layer.GetNextFeature()
34. geom = feature.GetGeometryRef()
35. if geom.GetGeometryName() != 'POLYGON' and geom.GetGeometryName() !=
      'MULTIPOLYGON':
36.   print 'Not a polygon shape'
37.   sys.exit(1)
38.
```

A 32-38 sorokban beszerezzük az input réteg első elemét (`GetNextFeature`) és megvizsgáljuk, hogy megfelelő típusú-e. Nem megfelelő típus esetén kilépünk a szkripből és üzenetet küldünk a hiba okáról.

```
39. # open the output point shape
40. pname = sys.argv[2]
41. centroid = driver.CreateDataSource(pname)
42. if centroid is None:
43.   print 'Could not create file ' + pname
44.   sys.exit(1)
45.
```

A 29-35 sorokban létrehozzuk az eredmény Shape fájlt a már előzőleg használt meghajtó segítségével. Itt is vizsgáljuk, hogy sikerült-e a Shape létrehozása. A sikertelenséget az írási jog hiánya okozhatja.

```
46. # create output layer
47. lyr = centroid.CreateLayer('0', None, ogr.wkbPoint)
48. if lyr is None:
49.   print 'Layer creation failed.'
50.   sys.exit(1)
51.
```

A 46-51 sorokban egy pont típusú réteget hozunk létre az eredmény Shape fájlban. Miért van erre szükség? A Shape fájlban csak egy réteg adatait tartalmazhatja. Viszont más a GDAL/OGR könyvtár által kezelt állományban több réteg is lehet, ezért az azonos kezelés érdekében a Shape fájlban is rétegekkel kell dolgoznunk. Az `ogr.wkbPoint` egy konstans, mely a létrehozott réteg típusát adja meg. További konstansok állnak rendelkezésre különböző típusú rétegek létrehozásához (`wkbLineString`, `wkbPolygon`, `wkbMultiPoint`, `wkbMultiLineString`, `wkbMultiPolygon`, ...). A réteg létrehozásánál is vizsgáljuk a sikeres végrehajtást, az előzőekhez hasonlóan.

```
52. # create attribute definitions
53. for i in range(feature.GetFieldCount()):
54.   lyr.CreateField(feature.GetFieldDefnRef(i))
55.
```

Az 52-55 sorokban létrehozzuk az eredmény Shape attribútum definícióit az input réteg attribútumai alapján.

```
56. # loop through the features in the layer
57. while feature:
```

A szkriptünk lényegi része az 56 sortól kezdődik. A `while feature` ciklus egyesével végig meg az input Shape elemein. Ehhez kapcsolódik a 70-71 sor, ahol felszabadítjuk az elem által elfoglalt memória területet (`Destroy`) és átlépünk a következő elemre (`GetNextFeature`).

```
58.    fod = layer.GetLayerDefn()
59.    fo = ogr.Feature(fod)
```

Az 58-59 sorban egy új pont típusú elemet hozunk létre. A réteg definícióból derül ki, hogy milyen típusú geometriáról van szó és milyen attribútumok tartoznak hozzá.

```
60.    # get the x,y coordinates of centroid
61.    geom = feature.GetGeometryRef()
62.    p = geom.Centroid()
63.    fo.SetGeometry(p)
```

A 60-63 sorokban először beszerezzük az input réteg aktuális elemének geometriáját. Ebből létrehozunk a centrálist, melyet hozzárendelünk az előzőleg létrehozott pont elemhez. Az így létrehozott centrális nem esik feltétlenül a felületen belül. Amennyiben a geos könyvtárral fordított ogr könyvtárat használunk, akkor a

```
geom.PointOnSurface(p)
```

utasítással biztosan a felületbe eső belső pontot kapunk.

```
64. for i in range(fod.GetFieldCount()):
65.     fo.SetField(fod.GetFieldDefn(i).GetNameRef(), feature.GetField(i))
66.
67.    lyr.CreateFeature(fo)
68.
```

A 64-65 sorokban a felület attribútumait átmásoljuk a centrálisba, majd a 67. sorban az elemet hozzáadjuk az eredmény réteghez. Ezzel a tényleges tevékenységet befejeztük.

```
69. # destroy the feature and get a new one
70.    fo.Destroy()
71.    feature.Destroy()
72.    feature = layer.GetNextFeature()
73.
```

A 69-73 sorokban az aktuális az elemek által elfoglalt területet szabadítjuk fel, végül tovább lépünk a következő felületre az input rétegen. A `GetNextFeature` a `None` értéket adja vissza, ha már nincs több elem a rétegben, ami a logikai kifejezésben a hamis értéket jelenti a `while` ciklusban.

```
74. # close the data sources
75. datasource.Destroy()
76. centroid.Destroy()
```

Az összes elem feldolgozása után a 74-76 sorokban lezárjuk a nyitott adatforrásokat és felszabadítjuk a lefoglaltt erőforrásokat.

A szkriptünk teljes listája:

```
1.  #!/usr/bin/python
2.  """
3.  This Python script creates a centroid point shape for a polygon shape
```

```
4.  the shape file is the first command line parameter
5.  the output is written to the second comand line parameter
6.  """
7.  import ogr, os, sys
8.
9.  # check parameters
10. if (len(sys.argv) < 3):
11.   print 'Usage: python centroid.py <polygon_shape> <output_point_shape>'
12.   sys.exit(0)
13.
14. # input polygon shape file
15. sname = sys.argv[1]
16. if not os.path.exists(sname):
17.   print sname + ' shape not found'
18.   sys.exit(1)
19.
20. # get the shapefile driver
21. driver = ogr.GetDriverByName('ESRI Shapefile')
22.
23. # open the data source
24. datasource = driver.Open(sname, 0)
25. if datasource is None:
26.   print 'Could not open file ' + sname
27.   sys.exit(1)
28.
29. # get the source layer
30. layer = datasource.GetLayer()
31.
32. # check geometry type
33. feature = layer.GetNextFeature()
34. geom = feature.GetGeometryRef()
35. if geom.GetGeometryName() != 'POLYGON' and geom.GetGeometryName() !=
      'MULTIPOLYGON':
36.   print 'Not a polygon shape'
37.   sys.exit(1)
38.
39. # open the output point shape
40. pname = sys.argv[2]
41. centroid = driver.CreateDataSource(pname)
42. if centroid is None:
43.   print 'Could not create file ' + pname
44.   sys.exit(1)
45.
46. # create output layer
47. lyr = centroid.CreateLayer('0', None, ogr.wkbPoint)
48. if lyr is None:
49.   print 'Layer creation failed.'
50.   sys.exit(1)
51.
52. # create attribute definitions
53. for i in range(feature.GetFieldCount()):
54.   lyr.CreateField(feature.GetFieldDefnRef(i))
55.
56. # loop through the features in the layer
57. while feature:
58.   fod = layer.GetLayerDefn()
59.   fo = ogr.Feature(fod)
60.   # get the x,y coordinates of centroid
61.   geom = feature.GetGeometryRef()
```

```
62.    p = geom.Centroid()
63.    fo.SetGeometry(p)
64.    for i in range(fod.GetFieldCount()):
65.      fo.SetField(fod.GetFieldDefn(i).GetNameRef(), feature.GetField(i))
66.
67.    lyr.CreateFeature(fo)
68.
69.    # destroy the feature and get a new one
70.    fo.Destroy()
71.    feature.Destroy()
72.    feature = layer.GetNextFeature()
73.
74. # close the data sources
75. datasource.Destroy()
76. centroid.Destroy()
```

Budapest, 2013. december 7.