Les données géographiques représentées sous la forme vectorielle permettent de représenter des points, des lignes ou des polygones géométriques ayant des caractéristiques propres (leurs attributs).

1 Lire et écrire des informations depuis des fichiers Shapefile

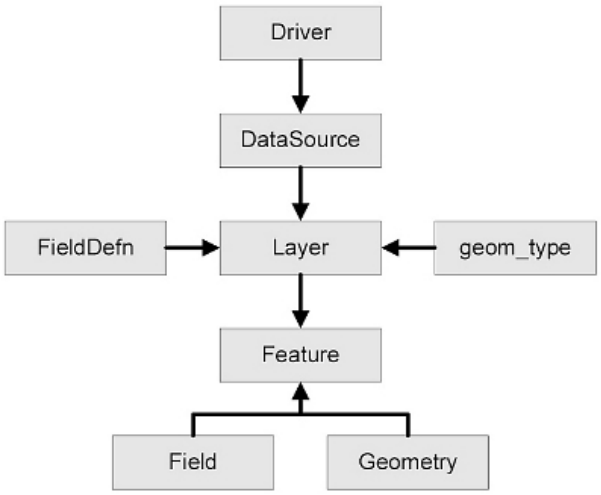

FIGURE 1 – Organisation logique des objets consituant une source de données vectorielles.

La Figure 1 illustre l'organisation logique des données vectorielles pour la librairie GDAL. Il est intéressant de noter que cette organisation est structurante pour l'ensemble de traitement qui devront être réalisés.

Dans le cas des fichiers Shapefile, cette structure est également fortement lié à l'organisation des fichiers : un fichier Shapefile ne comprend finalement que la partie droite du schéma : les formes ! Mais les autres informations concernant les attributs (*fields*) ne sont pas dans ce fichier mais dans un fichier `.dbf` qui correspond à une représentation de la table attributaire.

1.1 Les éléments de base

Lorsqu'on utilise une librairie, la première difficulté consiste à comprendre la logique de l'organisation des traitements. Dans le cas qui nous intéresse, je l'aborde par les objets.

Dans la librairie GDAL, il est important de connaître les différents On s'intéresse ici à des données issues d'un format vectoriel, typiquement un fichier Shapefile.

1.1.1 Ouverture et fermeture d'un fichier Shapefile en lecture

La lecture d'un fichier nécessite de disposer d'un outil capable d'interpréter correctement un format de fichier spécifique, c'est le rôle du **driver** (*cf.* exemple ci-dessous). Dans le cas de l'ouverture d'un Shapefile, il faut utiliser un driver initialisé avec la chaîne de caractère `'ESRI Shapefile'` (exactement).

L'exemple ci-dessous illustre l'ouverture d'un fichier Shapefile :

```
1  import os  #chargement de la librairie os
2  import sys #chargement de la librairie systeme
3  import ogr #chargement de la librairie OGR
4  from osgeo.gdalconst import * # chargement des constantes : pour GA_ReadOnly
5
6  #on va ouvrir un fichier shapefile
7  driver = ogr.GetDriverByName('ESRI Shapefile')
8
9  #se placer dans votre repertoire de travail
10 os.chdir('/home/guyet/data/ProgSIG')
11
12 #ouverture du fichier
13 datasource = driver.Open('sites.shp', GA_ReadOnly)
14 if datasource is None:
15   # cas d'erreur ( inexistence du fichier )
16   print('Could not open file')
17   sys.exit(1) #sortir du programme
```

La fonction **driver.Open**(...) permet de récupérer un objet qui représente une source de données. Dans le cadre général, ce pourrait être une base de données, un flux internet (WFS), une base de données géographique ou un fichier. Dans le cas de l'exemple proposé, nous chargeons un fichier.

— Le premier paramètre de la fonction donne le nom du fichier ! Ici, nous avons commencé par imposer de changer le répertoire de travail grâce à la ligne 10 (fonction **chdir**), il suffit alors de donner le nom du fichier sans son chemin d'accès. Une alternative aurait consister à donner son chemin "absolu", *c.-à-d.* `"/home/guyet/data/ProgSIG/sites.shp"`

— Le second paramètre permet d'indiquer le mode de lecture du fichier **GA_ReadOnly** pour ouvrir le fichier en lecture seule (pas de modification possible) et **GA_Update**, pour ouvrir le fichier avec possibilité de modification. Dans le cas où vous avez importé les constances de GDAL, vous pouvez utiliser **GA_ReadOnly** et **GA_Update**.

La fermeture de la source de données peut se faire à l'aide de l'appel suivant :

```
datasource.Destroy()
```

Remarque 2 - Alternative à l'utilisation des constantes GDAL

Dans la plupart des cas, les programmes n'utilisent pas les constantes prédéfinies de GDAL. Celles-ci peuvent être substituées directement par leurs valeurs. **GA_ReadOnly** correspond en fait à la valeur 0 et **GA_Update** à la valeur 1.

Vous pourrez vous en rendre compte en tapant les lignes suivantes directement en ligne de commande d'une console Python :

```
from osgeo.gdalconst import *
print(GA_Update)
print(GA_ReadOnly)
```

1.1.2 La couche vectorielle

L'objet le plus générique pour de l'information géographique, est la notion de **couche** (*layer*). Lorsqu'on ouvre un fichier *shapefile*, on récupère une couche de données vectorielles.

L'accès à une couche vectorielle depuis un fichier Shapefile se fait au travers de la fonction **GetLayer**() de la source de données.

```
layer = datasource.GetLayer()
```

L'objet **layer** représente la couche de données vectorielles. Il est possible de récupérer des informations sur cette couche :

— récupération du nombre de formes (nommés *features* en anglais)

```
numFeatures = layer.GetFeatureCount()
print('Feature count: ' + str(numFeatures))
print('Feature count:', str(numFeatures))
```

— l'étendue de la couche

```
extent = layer.GetExtent()
print('Extent:', extent)
print('UL:', extent[0], extent[3]) # upper-left corner
print('LR:', extent[1], extent[2]) # lower-right corner
```

— le nom de la couche

```
print( layer.GetName() )
```

— le type de géométrie (points, lignes, polygones, etc.). On reviendra plus tard sur les différents types de géométrie. Les types de géométrie sont représentés par des entiers [1]

```
print( layer.GetGeomType() )
```

1. Des valeurs constantes permettent de rendre ces entiers plus intelligibles : **ogr.wkbPoint**, **ogr.wkbPolygon**, ... La signification de ces entiers peut être trouvée dans la norme OGC 06-103r4 "OpenGIS Implementation Standard for Geographic information – Simple feature access – Part 1 : Common architecture", v1.2.1 (voir p. 66)

Il est surtout intéressant de savoir parcourir chacune des formes (*feature*) de la couche. Pour cela, on dispose d'au moins trois manières. Dans le cas simple où vous souhaitez traiter individuellement chacune des features qui compose votre couche, le schéma le plus simple est le suivant :

```
for f in layer:
  # do something with f
```

Cette écriture très ■ pythonic ■ peut se lire ainsi : ■ pour chaque feature f dans la couche layer faire *quelque chose avec f* ■. La partie à l'intérieur du code décrira le traitement individuel à appliquer à f (*p.ex.* la transformation d'un attribut, la modification de sa géométrie, etc).

Ce schéma est le plus simple qui existe, mais il n'est malheureusement pas approprié à toutes les situations. Dans certaines situations particulières, il peut être utile de mieux maîtriser l'ordre de traitement de la boucle. Les deux solutions suivantes permettent de mieux contrôler le déroulement de la boucle :

— accès à un élément par sa position dans la liste des éléments. Dans l'exemple ci-dessous, on peut récupérer le premier élément de la couche, puis le dixième !

```
feature = layer.GetFeature(0)
feature = layer.GetFeature(10)
```

On peut alors parcourir chacune des features d'une couche en utilisant un index i qui va varier de 0 au nombre de feature total :

```
for i in range(1, layer.GetFeatureCount()):
  feature = layer.GetFeature( i )
  # do something here with feature
```

— utilisation d'un ■ iterateur interne ■ à la couche :

```
feature = layer.GetNextFeature()
while feature:
  # do something here
  feature = layer.GetNextFeature()
layer.ResetReading() #rembobine pour recommencer un parcourt !
```

La fonction GetNextFeature() passe à la ■ feature suivante ■. En fait, il faut imaginer qu'il y a un pointeur interne à la couche qui retient à quel feature la lecture en est. Lorsqu'on utilise cette fonction, le pointeur est décalé (tant que c'est possible) et elle donne le nouvelle feature.

Remarque 3 - Penser à rembobiner

Cette méthode est à utiliser avec la plus grande attention ! En particulier, il faut bien penser à ■ rembobiner ■ la lecture lorsqu'on doit parcourir plusieurs fois une couche. Sinon, à la fin de la première lecture, l'itérateur reste à la fin ... Dans le code proposé, on rembobine après avoir lu, mais on pourrait très bien commencer par rembobiner !

1.2 Les *feature*

Dans un format vectorielle, une forme est associée à :

— une géométrie : elle donne les caractéristiques géométrique d'une forme, son type (point, ligne, polygone, multi-polygone) et sa description (positions spatiale du point, etc.)

— une liste de valeurs définies pour des attributs. Dans le langage des SIG, un attribut est appelé un *Field*. Les attributs sont définies à l'échelle de la couche et chaque forme a ses propres valeurs pour les attributs de la couche.

Pour accéder aux éléments de la couche, il est possible d'utiliser les fonctions suivantes :

— GetField('...'), en précisant le nom de l'attribut dont on souhaite la valeur

— GetFieldAsInteger('...') ou GetFieldAsString('...') sont des spécialisations de la fonction précédente pour être sûr de récupérer une information avec le type souhaité (si les données le permettent !)

Pour l'utilisation de ces fonctions, il est nécessaire de connaître les noms des attributs pour y accéder. Dans le cadre de ce tutoriel, on supposera que ce sera toujours le cas. Néanmoins, il est bien évidemment possible de lister l'ensemble des attributs d'une couche (avec leur noms et leurs caractéristiques) pour avoir des programmes génériques. Il faut conserver en tête que plus le programme est générique, plus il sera complexe. Ce n'est pas notre objectif que de faire des choses artificiellement complexes, cherchons déjà à traiter nos données.

Pour accéder à la géométrie d'un élément de la couche, il existe la fonction **GetGeometryRef**(). Cette fonction construit une représentation de la géométrie. En fonction de son type, la géométrie offre différentes possibilités.

Dans le cas d'un point, on peut récupérer ses positions spatiales (x, y) :

```
geometry = feature.GetGeometryRef()
x = geometry.GetX()
y = geometry.GetY()
```

Remarque 4 - Signification des valeurs X et Y

Nous reviendrons plus tard sur le gestion des systèmes de coordonnées. D'ores et déjà, il est intéressant de noter que du point de vue informatique, les valeurs enregistrées comme coordonnées géographique d'un point n'est soumis à aucune contrainte ! Il s'agit simplement de nombre ... et leur signification n'est possible que parce que la couche est associée à un SRS (Spatial Reference System).

Lorsque les traitements sur une forme sont terminés, il est utile de détruire explicitement cet objet par l'instruction **Destroy**(). Ceci permet de conserver de la mémoire pour la suite du traitement.

```
feature.Destroy()
```

1.2.1 Exemple 1 : afficher les arrêts de bus de Rennes

L'exemple ci-dessous illustre le parcours d'un fichier shapefile `tco-bus-topologie-pointsarret-td.shp` contenant l'ensemble des arrêts de bus de Rennes [2]. L'exemple affiche uniquement les arrêts de la ville de Rennes et les compte.

Le programme peut être téléchargé ici.

```
import ogr, os, sys
from osgeo.gdalconst import * # chargement des constantes : pour GA_ReadOnly

os.chdir('./LiveData/data/vecteurs/bus_star/')
driver = ogr.GetDriverByName('ESRI Shapefile')

# ouverture du fichier d'entree
arretsBusDS = driver.Open('tco-bus-topologie-pointsarret-td.shp', GA_ReadOnly)
if arretsBusDS is None:
  print('Could not open file')
  sys.exit(1)
arretsBusLayer = arretsBusDS.GetLayer()

#Pour toutes les formes de la couche d'entree, faire
cnt = 0
for arretBusFeature in arretsBusLayer:
    if arretBusFeature.GetField('nomcommune')=="Rennes":
        geom = arretBusFeature.GetGeometryRef()
        print("Arret '" + arretBusFeature.GetField('nom') +"', localisation: ("+ \
         str(geom.GetX()) +","+ str(geom.GetY()) +").")
        cnt += 1

print("Il y a " + str(cnt) + " arrets de bus dans Rennes.")

# Fermeture des sources de donnees
arretsBusDS.Destroy()
```

Vous pourrez tester et modifier le programme pour n'afficher, par exemple, que les arrêts de la ville de Betton, ou entre d'afficher le numéro de l'arrêt (regarder la couche en utilisant QGis pour connaître le nom des attributs ou de leurs valeurs).

1.2.2 Lister les attributs d'une couche

Le code ci-dessous permet de lister les attributs d'une couche. La fonction **GetLayerDefn**() permet d'obtenir une "liste" des attributs de la couche. Il ne s'agit pas d'une liste Python, mais d'un objet spécifique. Pour accéder à un élément de cette liste, il est nécessaire d'utiliser une fonction **GetFieldDefn(i)** où **i** est l'index de l'attribut. Finalement, les quatre propriétés des attributs sont accessibles via des fonctions dédiées (voir l'exemple ci-dessous).

2. Fichier disponible à l'url suivante : `https://data.rennesmetropole.fr/`.

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

dataSource = driver.Open("tco-bus-topologie-pointsarret-td.shp", )
layer = dataSource.GetLayer()

layerDefinition = layer.GetLayerDefn()
print "Name - Type Width Precision"
for i in range(layerDefinition.GetFieldCount()):
    fieldName = layerDefinition.GetFieldDefn(i).GetName()
    fieldTypeCode = layerDefinition.GetFieldDefn(i).GetType()
    fieldType = layerDefinition.GetFieldDefn(i).GetFieldTypeName(fieldTypeCode)
    fieldWidth = layerDefinition.GetFieldDefn(i).GetWidth()
    GetPrecision = layerDefinition.GetFieldDefn(i).GetPrecision()

    print fieldName + " - " + fieldType+ " " + str(fieldWidth) + " " + str(GetPrecision)
```

1.3 Écriture d'un fichier shapefile

La création d'un fichier *shapefile* reprend la même logique que la lecture d'un fichier. Pour la création d'une couche éditable, il faudra définir un *driver* et une source de données, puis chaque feature qui doit composer la couche sera ajoutée une à une.

1.3.1 Création d'une couche éditable

La création d'un nouveau fichier suit le même principe que l'ouverture d'un fichier : il est nécessaire d'avoir un driver qui saura comment enregistrer le fichier :

```
fout = '/home/guyet/data/ProgSIG/output.shp'
dsout = driver.CreateDataSource(fout)
```

dsout est alors une source de données correspondant au fichier **fout** dans lequel on va pouvoir écrire des données. Ici, on utilise un chemin absolu, mais si le répertoire de travail a été défini au préalable, il est possible de se limiter à doner le nom du fichier.

Pour la création d'une couche, il est nécessaire d'indiquer quel va être le type de forme que la couche contiendra (points, lignes, etc.). La fonction d'ajout d'une couche à une source de données est la suivante :

```
layerout = dsout.CreateLayer('Nouvelle couche', geom_type=ogr.wkbPoint)
```

Notez que ceci ne peut être fait que sur une source de données qui a été créée pour enregistrer des données.

Une fois que la couche a été définie, il est possible (et souvent nécessaire) de lui ajouter les attributs des formes de la couche. Je rappelle ici qu'il s'agit de définir uniquement les entêtes de la table des attributs. On donnera une valeur à ces attributs pour chaque forme !

Dans l'exemple ci-dessous, on définit un nouvel attribut nommé `id` et on indique qu'il s'agira de nombre entier. On ajoute ensuite cet attribut à la couche **layerout**. Puis, on ajoute un second attribut `descr` qui est une chaîne de caractères de longueur 40 au plus.

```
fieldDefn = ogr.FieldDefn('id', ogr.OFTinteger)
layerout.CreateField(fieldDefn)
fieldDefn = ogr.FieldDefn('descr', ogr.OFTString)
fieldDefn.SetWidth(40)
layerout.CreateField(fieldDefn)
```

Dans ce code, la fonction **FieldDefn** permet de créer un nouvel attribut à partir de deux informations : le nom de l'attribut et son type (entier, texte, ...). Ensuite, la fonction **CreateField** d'une couche de données ajoute cet attribut à la couche. Dans certains cas, comme pour l'attribut `descr`, l'attribut peut être configurer plus finement. Ici, on donne la taille maximum de l'attribut.

1.3.2 Création de features

La création de nouvelles formes/features ne peut se faire qu'une fois qu'une couche a été définie ainsi que ses attributs. En particulier, il est nécessaire de connaître la taille de chaque attribut pour savoir comment les enregistrer. Avant tout chose, on a besoin de récupérer la description de tous les attributs :

```
featureDefn = layerout.GetLayerDefn()
```

La création d'une nouvelle forme se fait ensuite en quatre étapes :

1. On créé effectivement la nouvelle forme avec les bons attributs pour la couche

```
feature = ogr.Feature(featureDefn)
```

2. On donne une géométrie à la forme

```
point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(10,20)
feature.SetGeometry(point)
```

3. Ajouter les valeurs pour chaque attribut

```
feature.SetField('id', 23)
```

4. Ajouter la forme à la couche

```
layerout.CreateFeature(feature)
```

1.3.3 Exemple 2 : recopie d'un fichier shp

L'exemple ci-dessous illustre la recopie d'un fichier, *feature* par *feature*.

```
import ogr, os, sys
from osgeo.gdalconst import *

os.chdir('/home/guyet/data')
driver = ogr.GetDriverByName('ESRI Shapefile')

# ouverture du fichier d'entree
inDS = driver.Open('sites.shp', GA_ReadOnly)
if inDS is None:
  print('Could not open file')
  sys.exit(1)
inLayer = inDS.GetLayer()

# creation du fichier de sortie : couche de points
if os.path.exists('test.shp'):
  driver.DeleteDataSource('test.shp')
outDS = driver.CreateDataSource('test.shp')
if outDS is None:
  print('Could not create file')
  sys.exit(1)

outLayer = outDS.CreateLayer('test', geom_type=ogr.wkbPoint)

# definition des attributs de la couche de sortie par recopie
fieldDefn = inLayer.GetFeature(0).GetFieldDefnRef('id')
outLayer.CreateField(fieldDefn)
fieldDefn = inLayer.GetFeature(0).GetFieldDefnRef('cover')
outLayer.CreateField(fieldDefn)

# featureDefn decrits les attributs de la couche outLayer
featureDefn = outLayer.GetLayerDefn()

#Pour toutes les formes de la couche d'entree, faire
for inFeature in inLayer:
  # Creation d'une forme par recopie
  outFeature = ogr.Feature(featureDefn)
  outFeature.SetGeometry(inFeature.GetGeometryRef())
  outFeature.SetField('id', inFeature.GetField('id'))
  #NB: seul l'attribut 'id' est recopie ici !

  # Ajouter de la forme a la couche de sortie
  outLayer.CreateFeature(outFeature)

  # destruction des formes
  inFeature.Destroy()
  outFeature.Destroy()
```

```
# Fermeture des sources de donnees
inDS.Destroy()
outDS.Destroy()
```

Le programme peut être téléchargé ici.

On peut noter qu'à la fin de cet exemple, deux instructions viennent conclure l'exécution du programme appelant les fonctions de destruction des sources de données. Ces instructions ont pour effet de forcer à ■ fermer ■ le fichier. Cette étape est souvent indispensable après l'écriture d'information. C'est parfois la seule trace d'une modification.

1.4 Un exemple plus complexe : combien autour ?

Dans cette exemple, on cherche à savoir pour chaque point de la couche `sites.shp` combien il y a d'objets autour d'un objet donné.

Les deux choses qu'on propose de regarder ici sont :

— la décomposition de la tâche à l'aide d'une fonction

— l'utilisation de structure de données classiques (dictionnaire et liste) pour collecter les informations

```
import ogr, os, sys
os.chdir('/home/guyet/data')

def compter_autour(geom, layer, distance):
    nb=0
    for i in range(1, layer.GetFeatureCount()):
        feature = layer.GetFeature(i)
        fg = feature.GetGeometryRef()
        fx= fg.GetX()
        fy= fg.GetY()
        d2 = (fx-geom.GetX())**2 + (fy-geom.GetY())**2
        #print(d2)
        if d2 < distance**2:
            nb += 1
    return nb-1

# ouverture du fichier d'entree
driver = ogr.GetDriverByName('ESRI Shapefile')
inDS = driver.Open('sites.shp', 0)
if inDS is None:
  print('Could not open file')
  sys.exit(1)
inLayer = inDS.GetLayer()

L=[]
D={}
#Pour toutes les formes de la couche d'entree, faire
for inFeature in inLayer:
    fid = inFeature.GetField('id')
    nb = compter_autour(inFeature.GetGeometryRef(), inLayer, 20000)
    print("Nb voisins de ", fid,":",str(nb))
    L.append( [fid, nb] )
    D[fid] = nb
    # destruction des formes
    inFeature.Destroy()
# Fermeture des sources de donnees
inDS.Destroy()

print(L)
print(D)
```

Le programme peut être téléchargé ici.

1.5 Exercices

Exercice 1 (Lecture d'un fichier Shapefile)

Question a) Commencez par explorer les données avec QGis

Question b) Créer un script Python pour déterminer le nombre de features de la couche *Pour cette première question, il s'agit de mettre en place correctement votre script : chargement des librairies, définition du répertoire de travail et ouverture d'une couche (inutile de parcourir les features)*

Question c) Compléter ensuite votre script pour afficher le type de forme que comporte la couche (vous comparer la valeur obtenu aux constantes **ogr.wkbPoint**, **ogr.wkbLineString** et **ogr.wkbPolygon**), l'étendue de la couche ainsi que la liste de ses attributs.

Question d) Compléter votre script Python pour afficher en ligne (**print**) les `ID`, `COVER` et les positions `x y` de chaque point du fichier `sites.shp`.

Exercice 2 (Lecture d'un fichier Shapefile) *Dans cet exercice, vous utiliserez la couche vectorielle des stations de vélo de Rennes (`supports-velos.shp`)*

Question a) À l'aide d'un programme Python, répondre aux questions suivantes :
- *Combien y a-t-il de stations de vélo à rennes ?*
- *lister toutes les rues qui ont un support de vélo (doublons autorisés) dans la ville de Rennes ?*
- *Combien y a-t-il d'emplacements de vélo à rennes ?*
- *Combien y a-t-il de stations de vélo qui ont strictement plus de 10 supports ?*

Question b) En utilisant l'instruction **input**() illustrée ci-dessous, faire un programme qui demande à l'utilisateur l'identifiant d'une station et affiche ses coordonnées spatiales et le nombre de supports de vélo.

```
a = input("donner moi une valeurs stp") # bloque le programme
print (a)
```

Question c) Pour une station donnée (par exemple, à la place de la gare, support vélo avec l'**objectid** 258) :
- *quelle est la station la plus proche (à vol d'oiseaux) ?*
- *combien y a-t-il de stations à moins de 500m ?*

Question d) Créer une liste dans laquelle vous mettrez toutes les distances (à vol d'oiseau) entre pairs de station de vélos ? quelle est la moyenne des distances ?

Exercice 3 (Lecture avancée d'un fichier Shapefile) *On cherche à modifier le script de la section 1.4 pour avoir des nombres d'objet par type d'objet (rocks, shrubs, etc)*

Question a) Nettoyer le programme pour qu'il n'utilise plus que la liste (sans affichage et sans dictionaire)

Question b) Modifier la fonction pour qu'elle compte uniquement les éléments de type *rocks* et tester

Question c) Modifier la fonction pour lui ajouter un paramètre **type** et qu'elle compte uniquement les éléments de type **type**

Question d) Modifier le programme principal pour qu'il ajoute à la liste la valeur pour les différents types (*rocks*, *shrubs*, ...)

Exercice 4 (Décomposition d'un fichier Shapefile)

Question a) Écrire un script Python créant un fichier `output.shp` qui ne contiendra que les points correspondant à la valeur `trees` de l'attribut `COVER` du fichier `sites.shp`.

Le fichier ne conservera que l'attribut `ID` du fichier d'origine, l'autre attribut étant inutile.

Question b) Transformer votre script en **une fonction** qui prendra en paramètre le nom du fichier d'entrée, le nom du fichier de sortie et la valeur de l'attribut dont on conserve les points.

Question c) Écrire un script Python utilisant votre fonction pour créer un fichier pour chacune des modalités de l'attribut `COVER` *Vous pourrez récupérer la liste des modalités de l'attribut en allant visualiser les données sous QGis.*

Exercice 5 (Fusion de deux fichiers Shapefile)

Question a) Écrire un script qui construit un fichier Shapefile unique à partir de deux fichiers Shapefile de points.

Vous supposerez que les deux fichiers utilisent le même système de projection.

Question b) Tester votre script en reconstruisant un fichier `site_rebuilt.shp` à partir des décompositions obtenues lors de l'exercice précédent.

Question c) Modifier votre script pour ajouter un nouvel attribut dans le fichier fusion qui contiendra l'information sur l'origine de chaque forme. Vous utiliserez un attribut au format texte (**ogr.OFTString**) de longueur 10.

1.6 Utilisation des filtres

Revenons maintenant quelques instants sur l'exemple des l'affichage des bus de Rennes ou de la sélection des sites correspondant à un type d'objet. Cette opération qui consiste à parcourir une couche selon une sélection d'objets est assez usuel, et plutôt que de le faire par de la programmation, il est possible d'utiliser des filtres.

Les filtres vont sélectionner un sous-ensemble de features de votre couche selon un critère que vous aller définir (par exemple, les features d'arrêt correspondant à Rennes), une fois ce filtrage appliqué, le parcours de la couche se fera en tenant compte de votre sélection.

L'exemple ci-dessous reprend le parcours des arrêts de bus de Rennes uniquement.

```
import ogr, os, sys
from osgeo.gdalconst import * # chargement des constantes : pour GA_ReadOnly

driver = ogr.GetDriverByName('ESRI Shapefile')

# ouverture du fichier d'entree
arretsBusDS = driver.Open('tco-bus-topologie-pointsarret-td.shp', GA_ReadOnly)
if arretsBusDS is None:
  print('Could not open file')
  sys.exit(1)
arretsBusLayer = arretsBusDS.GetLayer()

arretsBusLayer.SetAttributeFilter("nomcommune=='Rennes'")

for arretBusFeature in layer:
        geom = arretBusFeature.GetGeometryRef()
        print("Arret '" + arretBusFeature.GetField('nom') +"', localisation: ("+ \
         str(geom.GetX()) +","+ str(geom.GetY()) +").")

# Fermeture des sources de donnees
arretsBusDS.Destroy()
```

Un autre type de filtre similaire permet de faire une sélection des features à parcourir par une restriction spatiale. Un objet représentant une couche offre deux fonctionnalités principales :

- la fonction **SetSpatialFilter (geom)** qui prend en paramètre une géométrie et qui applique un filtre en utilisant une géométrie (de type polygone ou multi-polygone a priori)
- la fonction **SetSpatialFilterRect (xm, ym, xM, yM)** qui permet une sélection à partir des coordonnées d'un rectangle. Cette fonction se charge simplement de créer la forme rectangulaire pour vous ...

L'exemple ci-dessous illustre l'ajout d'un filtre spatial défini par un polygone créé "à la main" (ici, construit par une séquence WKT, c'est-à-dire une succession de points).

```
driver = ogr.GetDriverByName("ESRI Shapefile")
dataSource = driver.Open("tco-bus-topologie-pointsarret-td.shp", GA_ReadOnly)
layer = dataSource.GetLayer()

wkt = "POLYGON ((-103.81402655265633 50.253951270672125,-102.94583419409656
     51.535568561879401,-100.34125711841725 51.328856095555651,-100.34125711841725
     51.328856095555651,-93.437060743203844 50.460663736995883,-93.767800689321859
     46.450441890315041,-94.635993047881612 41.613370178339181,-100.75468205106476
     41.365315218750681,-106.12920617548238 42.564247523428456,-105.96383620242338
     47.277291755610058,-103.81402655265633 50.253951270672125))"

layer.SetSpatialFilter( ogr.CreateGeometryFromWkt(wkt) )

for feature in layer:
    print feature.GetField('nom')
```

Dans l'exemple ci-dessous, j'utilise les fonctionnalités de filtrage pour classer les arrêts de bus de la STAR par commune en croisant les informations relatives aux limites des communes de la métropole (couche de polygones) avec les données de la couche des arrêts de bus.

```
import ogr, os, sys
from osgeo.gdalconst import * # chargement des constantes : pour GA_ReadOnly

os.chdir('../../LiveData/data/vecteurs/bus_star/')
driver = ogr.GetDriverByName('ESRI Shapefile')

# ouverture du fichier d'entree
arretsBusDS = driver.Open('tco-bus-topologie-pointsarret-td.shp', GA_ReadOnly)
if arretsBusDS is None:
  print ('Could not open file')
  sys.exit(1)
arretsBusLayer = arretsBusDS.GetLayer()


# ouverture du fichier d'entree
communesDS = driver.Open('limites-communales-referentielles-de-rennes-metropole-polygones.shp',
      GA_ReadOnly)
if communesDS is None:
  print ('Could not open file')
  sys.exit(1)
communesLayer = communesDS.GetLayer()

# Pour toutes les communes, faire:
for Comm in communesLayer:
    print ("====" + Comm.GetField("nom") + " ====")

    # Selection spatiale des arrets
    arretsBusLayer. SetSpatialFilter ( Comm.GetGeometryRef() )
    # Pour chaque arret de la selection , faire :
    for arretBusFeature in arretsBusLayer:
        geom = arretBusFeature.GetGeometryRef()
        print ("Arret '" + arretBusFeature.GetField('nom') + "'")
    arretsBusLayer.ResetReading()

# Fermeture des sources de donnees
arretsBusDS.Destroy()
communesDS.Destroy()
```

Le programme peut être téléchargé ici.

1.7 Utilisation de la librairie `shapefile`

Lorsque vous ne traitez que des fichiers Shapefile, une alternative possible à l'utilisation de la librairie GDAL/OGR est l'utilisation de la librairie Python nommée `pyshp` (nom du *package*) ou `shapefile` (nom de l'*import*).

Cette librairie a été conçue pour mettre à disposition, peut être plus facilement, des fonctionnalités usuelles de manipulation des shapefile.

```
os.chdir('./LiveData/data/vecteurs/bus_star/')

import shapefile

# ouverture du fichier d'entree
r = shapefile .Reader("tco-bus-topologie-pointsarret-td.shp")


#### acces aux features ####
shapes = r.shapes()
len(shapes)

# type de premiere geometrie
print ( shapes[1].shapeType )

#### attributs de la couche ####
# lister les attributs de la couche
print ( r. fields )

# acces aux valeurs des attributs du 2e element (meme ordre que pour shapes)
print ( r.record(2) )

### Cas d'une couche de polygones ####
```

```
r = shapefile.Reader("limites-communales-referentielles-de-rennes-metropole-polygones.shp")

pg_shapes = r.shapes()
print ( shapes[1].shapeType )

# description du polygone
print ( len(shapes[1].points) ) # nb de points qui le compose
```

Le programme peut être téléchargé ici.

La librairie `shapefile` peut être utilisée également pour créer (resp. modifier) des shapefiles grâce à l'utilisation d'objet **Writer** (resp. **Editor**). Leur utilisation est plus technique et le lecteur intéressé pourra consulter l'aide en ligne [3] sur cette librairie.

3. `https://pypi.python.org/pypi/pyshp`