1 Manipulation des formes géométriques

1.1 Les formes de base

Toutes les formes sont des objets de la classe **geometry** représentent des formes géométriques de bases. Cette classe peut être spécialisée comme illustré par la Figure 1.

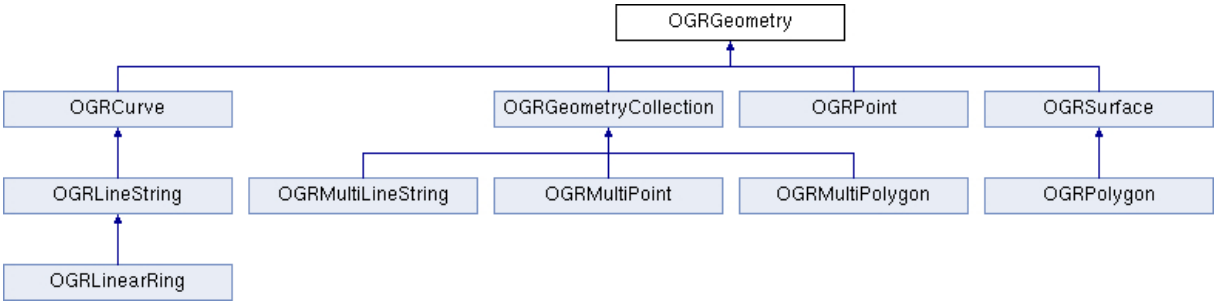

FIGURE 1 – Classes de geometry

Les différents types spécifiques de géométries sont les suivantes :
— point (`wkbPoint`) : géométriquement défini par des coordonnées (x, y),
— une polyligne (`wkbLineString`) : une succession de points liés par des lignes (droites),
— un polyligne fermée (`wkbLinearRing`) : une successions de points qui définissent un anneau (le premier point est utilisé pour "fermer" la courbe,
— un polygone (`wkbPolygon`) : défini par une polyligne fermée dite "extérieure" et un ensemble de polylignes fermées dites "intérieures" (qui représente des extrusions de la forme définie par la polyligne "extérieure".

Finalement, ces formes de bases peuvent être regroupées sous la forme de MultiLineString, MultiPoint et MultiPolygone.

! Attention ! - Shapefile

Un fichier shapefile qui contient des polygones peut contenir des polygones ou des multi-polygones ...

Remarque 2 - Pas de courbes !

Contrairement à d'autres librairies de dessin vectoriel (dans les usages différents de ceux de l'information géographique), les couches SIG vecteurs ne permettent pas de définir des formes curvilignes. Si vous souhaitez représenter un cercle ou bien une courbe, il faudra nécessairement l'approximer par des droites.

1.2 Création d'une nouvelle géométrie

Lors de la création d'une nouvelle géométrie, il faut :

1. créer un objet représentant la géométrie (utiliser la fonction **ogr.Geometry**)
2. spécifier les systèmes de coordonnées de la géométrie,
3. définir les propriétés géométriques de la géométrie (dépend de la nature de la géométrie)

Dans les exemples ci-dessous, on illustre la création de formes géométriques. On reviendra plus tard sur la définition du système de coordonnées.

Création d'un point

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

Création d'une ligne

```
line = ogr.Geometry(ogr.wkbLineString)
line.AddPoint(10,10)
line.AddPoint(20,20)
line.SetPoint(0,30,30)
```

Dans la dernière ligne, la fonction **SetPoint** transforme les coordonnées du premier point (identifié par l'indice 0)

Pour accéder aux informations d'une polyligne :

— **line.GetPointCount()** permet d'avoir le nombre de points de la ligne

— **line.getX(34)** et **line.getY(34)** permettent d'accéder aux coordonnées du 34eme points de la ligne (si il existe !)

Création d'un polygone

L'opération de création d'un polygone est plus complexe, elle nécessite de créer un anneau extérieur puis de faire les anneaux intérieurs.

```
#creation d'un anneau exterieur
ring = ogr.Geometry(ogr.wkbLinearRing)
ring.AddPoint(0,0)
ring.AddPoint(100,0)
ring.AddPoint(100,100)
ring.AddPoint(0,100)
ring.CloseRings()

#creation d'un anneau interieur
inring = ogr.Geometry(ogr.wkbLinearRing)
inring.AddPoint(25,25)
inring.AddPoint(75,25)
inring.AddPoint(75,75)
inring.AddPoint(25,75)
inring.CloseRings()

#creation du polygone
polygon = ogr.Geometry(ogr.wkbPolygon)
polygon.AddGeometry(ring)
polygon.AddGeometry(inring)
```

1.3 Destruction des géométries

À chaque fois que vous créer une géométrie, celle-ci est retenue en mémoire de l'ordinateur. Il est préférable d'indiquer explicitement à l'ordinateur qu'une géométrie ne sera plus utile dans la suite du programme pour qu'il libère de la mémoire [1]. Ceci est possible grâce à l'instruction **Destroy()**, par exemple :

```
polygon.Destroy()
```

1.4 Exemple de création d'un fichier Shapefile à partir de rien ...

L'exemple ci-dessous permet de créer des points dans un Shapefile à partir de saisies clavier de l'utilisateur.

L'intérêt du programme est de donner la possibilité, en l'adaptant de générer automatiquement des formes complexes pour, par exemple, mener des analyses régionalisées.

Le programme peut être téléchargé ici.

```
# recuperation d'un driver
driver = ogr.GetDriverByName('ESRI Shapefile')

# creation d'un nouveau fichier
fin='test_generation.shp'
if os.path.exists(fin):
  driver.DeleteDataSource(fin)
ds = driver.CreateDataSource(fin)
if ds is None:
  print('Could not create file ', fin)
  sys.exit(1)
```

1. Pour l'exécution d'un programme la mémoire importante est la mémoire RAM. Celle-ci est en quantité limitée. Les programmes qui ne libère pas la mémoire qu'ils utilisent peuvent non-seulement s'arrêter, mais par la même provoquer des erreurs sur tout le fonctionnement de l'ordinateur. On parle alors de fuite mémoire.

```
layer = ds.CreateLayer('test', geom_type=ogr.wkbPoint)

# ajoute d'un attribut aux points
fieldDefn = ogr.FieldDefn('id', ogr.OFTInteger)
layer.CreateField(fieldDefn)

while 1:
  val1=raw_input("Donnez une valeur de X: ")
  val1= int(val1)
  val2=raw_input("Donnez une valeur de Y (-1 pour quitter): ")
  val2= int(val2)
  if val2==-1:
    break

  # creation d'un nouveau point
  point = ogr.Geometry(ogr.wkbPoint)
  point.AddPoint(val1, val2)

  # creation de la feature avec ses attributs associe a la forme
  featureDefn = layer.GetLayerDefn()
  feature = ogr.Feature(featureDefn)
  feature.SetGeometry(point)
  feature.SetField('id', id)
  id = id+1

  #ajout a la couche
  layer.CreateFeature(feature)

  point.Destroy()
  feature.Destroy()

#fermeture du fichier
ds.Destroy()
```

1.5 Exercices

Exercice 1 (Marche aléatoire)

Question a) Écrire un script Python créant un fichier `output.shp` qui contiendra une polyligne qui passe par tous les points du fichier `sites.shp`. On ne s'intéressera pas à l'ordre dans lequel sont pris en compte les points.

L'exemple ci-dessous illustre comment construire une liste de points (listes de tuples) ordonnées par les X :

```
shapeData = osgeo.ogr.Open('sites.shp')
layer = shapeData.GetLayer()
points = []
for index in xrange(layer.GetFeatureCount()):
  feature = layer.GetFeature(index)
  geometry = feature.GetGeometryRef()
  points.append( geometry )
```

L'exemple suivant illustre l'utilisation des nombres aléatoires :

```
import random
x=random.randint(1,10) #genere un nombre entier aleatoire entre 1 et 10 compris,
y=random.uniform(1,10) # genere un nombre reel aleatoire, selon une lois uniforme, entre 1 et 10.
```

Question b) Écrire un script Python créant un fichier `output.shp` qui génère une marche aléatoire de longueur l à partir des points de `sites.shp`.

Une marche aléatoire est une polyligne qui passe d'un point à un autre de manière aléatoire. Pour réaliser ce script, vous commencerez par construire un vecteur de points à partir des points localisant les sites donnés dans le fichier `sites.shp`. Ensuite, vous générerez une polyligne de l points en tirant aléatoirement des points dans ce vecteur. Cette polyligne sera ensuite enregistrée dans le Shapefile.

Exercice 2 (Décomposition de lignes) *Lors de la manipulation de couche de lignes comme des routes ou des cours d'eau, il n'est pas rare d'avoir l'intégralité d'un cours d'eau (et ses affluents) ou du réseau routier dans un seul objet ligne. La manipulation de cet objet peut se montrer difficile. On rappelle que dans les données vectorielles, une ligne est décrite par une succession de segments (ligne droite entre deux points).*

Dans cet exercice, l'objectif est de décomposer une couche de longues lignes en une couche de lignes équivalentes, mais pour laquelle tous les objets sont des segments (une ligne droite entre deux points).

Question a) Décomposition d'une ligne *Écrire une fonction* **decompose(line)** *qui prend en paramètre une géométrie de lignes et qui retourne une liste de géométries de ligne correspondant à tous les segments de la ligne en entrée.*

Question b) Décomposition d'une couche de lignes *Écrire une fonction* **decompose_file(fname)** *qui prend en paramètre un nom de fichier shapefile (lignes) et qui créé un nouveau fichier shapefile de lignes contenant le résultat de la décomposition des lignes du fichier par la fonction précédente. Tous les attributs des lignes seront recopiés.*

2 Opérations géométriques

2.1 Présentation des opérations géométriques

La librairie OGR/GDAL offre des possibilités élémentaires de manipulation des objets géométriques. Si on considère deux géométries (par défaut, des polygones) **g** et **h**, la librairie propose les opérations suivantes :

- **geom** = **g**.**Intersection(h)** : construit la géométrie **geom** par l'intersection des deux géométries **g** et **h**,
- **geom** = **g**.**Union(h)** : construit la géométrie **geom** par l'union des deux géométries **g** et **h**,
- **geom** = **g**.**Difference(h)** : construit la géométrie **geom** par le découpage de la géométrie **g** de sa partie commune avec **h**,
- **geom** = **g**.**Buffer(34)** : construit la géométrie **geom** en ajoutant un buffer d'une distance de 34 à la géométrie **g**,
- **geom** = **g**.**GetEnvelope()** : construit la géométrie **geom** définissant le rectangle englobant de **g**,
- **geom** = **g**.**ConvexHull()** : construit l'enveloppe convexe de **g**.

S'ajoutent à ces fonctions, un ensemble de fonctions qui peuvent servir à identifier des relations particulières entre deux géométries. Considérons de nouveau deux géométries **g** et **h**.

- **g**.**Contains(h)** : teste si la forme **h** est incluse dans la forme **g**, cette relation peu, en particulier être utilisée pour savoir si un point se trouve à l'intérêt d'une forme.
- **g**.**Touches(h)** : teste si la forme **g** touche la forme **h**, c'est-à-dire s'ils ont un bord commun,
- **g**.**Overlaps(h)** : teste si les formes **h** et **g** se superpose (partiellement). Il faut préférer l'usage de cette fonction à l'usage de la fonction **Intersects** qui ne regarde que les enveloppes.
- **g**. **Disjoint** (**h**) : teste si les formes **h** et **g** sont disjointes (c'est l'inverse de *overlaps*)
- **g**.**Distance(h)** : calcule la distance entre les formes **h** et **g**. La distance entre deux polygones est la distance la plus courte entre deux points des polygones respectifs.

Exercice 3 *L'objectif de cet exercice est de transformer un couche de polygônes en utilisant la fonction* **Buffer***() sur toute les géométries d'une couche. Vous pourrez utiliser la couche RPG disponible dans le répertoire* `data/vecteurs/RPG/`.

Question a) Commencer par faire un programme qui recopie intégralement un fichier shapefile dans un autre fichier shapefile (vous vous inspirerez de l'exercice sur la recopie de la couche de points, sans recopier les attributs).

Question b) Modifier votre code pour que, à la place de la géométrie originale – appelons cette variable **geom** – vous construisiez une nouvelle géométrie à l'aide de la fonction **geom**.**Buffer(50)**. Cette nouvelle géométrie sera celle effectivement utilisée pour la second couche. La valeur `50` est indicative (vous pourrez la modifier pour voir les effets, y compris avec des valeurs négatives).

Télécharger la correction.

2.2 Exemple avancé : fusion des sous-bassins versants des Grands Lacs

On dispose d'une couche vecteur de polygones délimitant les sous-bassins versants dans la région des Grands Lacs (`glwsheds.shp`) On cherche à construire un nouveau fichier `greatlakesBV.shp` regroupant chaque bassin versant dans un minimum de géométries. Par rapport aux données disponibles, il faut fusionner les formes géométriques qui se touchent et qui appartiennent aux même bassin versant.

On notera que chaque sous-bassin versant est caractérisé par des attributs dont l'attribut `LAKEBASIN` qui indique à quel bassin versant appartient un sous-bassin.

La tâche semble assez simple (et doit se faire assez simplement (?) avec GRASS ou autres outils de manipulation de données SIG), mais il y a une difficulté algorithmique pour faire la fusion de plusieurs formes géométriques ...

Bien ! Discutons maintenant stratégie. Pour s'attaque à ce problème, nous allons commencer par récupérer toutes les géométries en les classant par BV (on utilisera pour cela l'attribut `LAKEBASIN`). Nous utiliserons pour cela une structure de données de type dictionnaire qui va associer un BV à une liste de forme géométriques.

Une fois que nous avons les listes de formes géométriques pour chaque BV, il faut fusionner deux à deux les formes qui se touchent. On construit une nouvelle liste de formes géométriques fusionnées **outputgeoms**, et ajoute successivement chaque élément `e` de la liste de géométrie initiale.

L'algorithme élémentaire pour cet ajout est le suivant :

```
FOR h in outputgeoms
   SI h touche e ALORS
      fusionner e dans h
      break
```

Néanmoins, en fonction de l'ordre de traitement et de l'organisation spatiale des formes, cette méthode ne conduit pas à assurer la bonne fusion de toutes le formes. La solution ci-dessous est meilleure : on introduit une variable `modifie` qui indique si la liste a subit des modifications par l'ajout de `e` (ie la fusion de `e` avec un élément de la liste)). Si tel est le cas, on parcourt de nouveau cette liste pour savoir si l'élément fusionné n'a pas lui même de nouvelles connexion avec d'autres éléments.

```
modifie=true
TANT QUE modifie
   modifie = false
      FOR h in outputgeoms
         SI h touche e ALORS
            fusionner e dans h
            e <- h
            modifie = true
```

Les nouvelles listes de formes géométriques seront elles aussi organisées dans un dictionnaire, et on se servira de ce dictionnaire de formes géométrique pour générer le fichier de sortie.

Ouverture des fichiers On commence notre script de manière "habituelle" en ouvrant les fichiers qui nous seront utiles :

— le fichier contenant la description des sous-bassins versants,
— le fichier de sortie près à l'écriture.

```
import ogr, os, sys

os.chdir('/home/tguyet/Enseignements/2012-2013/TASE-GAPE/ProgSIG/Tutoriel/data/greatlakes/')

 filelakes  = "glwsheds.shp"
 fileout  = "output.shp"
 driver  = ogr.GetDriverByName('ESRI Shapefile')

# Ouverture de la couche des sous-BV des lacs
sBV = driver.Open(filelakes, 0)
 if sBV is None:
    print 'Could not open file ' + filelakes
    sys.exit(1)
sBVLayer = sBV.GetLayer()

# creation de la couche de sortie
```

```
if os.path.exists(fileout):
    driver.DeleteDataSource(fileout)
outDS = driver.CreateDataSource(fileout)
if outDS is None:
    print 'Could not create file'
    sys.exit(1)
outLayer = outDS.CreateLayer("BV", srs=sBVLayer.GetSpatialRef(), geom_type=ogr.wkbPolygon)


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

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

Construction du dictionnaire des sous-bassins versants. Pour chaque forme du fichier d'entrée, on regarde la valeur de son attribut et on ajoute sa géométrie à la liste adéquate

```
sousBVdict={}

sfeat = sBVLayer.GetNextFeature()
while sfeat:
    #on recupere la geometrie d'un site
    geom = sfeat.GetGeometryRef().Clone() #attention ici a bien cloner la geometrie !

  #on recupere l'attribut indiquant le BV d'appartenance
    lakeBV = sfeat.GetField('LAKEBASIN')

    # si c'est le premier element pour un BV, on cree une liste vide
    if not sousBVdict.has_key( lakeBV ) :
        sousBVdict[lakeBV]=list()

    #on ajoute la geometrie dans la liste des ssBV correspondant
    sousBVdict[lakeBV].append( geom )

    sfeat.Destroy() #on peut detruire car on a clone la geometrie !
    sfeat = sBVLayer.GetNextFeature()
```

Fusion des géométries qui se touchent. On traite séparément chacun des bassins versants

```
BVdict={}
for lake in sousBVdict: #pour chaque bassin versant
    outputgeoms = list() #liste des geometries fusionnees

    for geom in sousBVdict[lake]: # pour chaque geometrie
        modified=True
        while modified:
            modified=False
            for h in outputgeoms:
                if geom.Touches(h): # test sur la propriete geometrique
                    geom=geom.Union(h) #fusion des formes
                    outputgeoms.remove(h)
                    modified=True
        outputgeoms.append(geom)

    #on associe la nouvelle liste au bassin-versant 'lake'
    BVdict[lake]=outputgeoms
```

Enregistrement du fichier de sortie. On enregistre chacune des géométries on mentionne dans ses attributs sont BV d'appartenance.

```
for lake in BVdict:
    for geom in BVdict[lake]:
        outFeature = ogr.Feature(featureDefn)
        outFeature.SetGeometry(geom)
        outFeature.SetField('LAKEBASIN', lake)
        outLayer.CreateFeature(outFeature)
        geom.Destroy() #si on n'en a plus besoin par la suite !
        outFeature.Destroy()
outDS.Destroy()
```

2.3 Exercices

Exercice 4 (Reconstructions des Grands Lacs) *On dispose d'une couche vecteur de polygone délimitant les régions administratives autour des Grands Lacs (`glpolit_gen.shp`). On souhaite construire des nouvelles couches à partir de ces données. On utilise pour cela des opérations géométriques entre les géométries.*

Question a) Construire les limites du lac supérieur *On souhaite maintenant extraire le limites du lac supérieur.*

1. *Commencer par identifier à la main (dans QGis) les limites approximatives de la boîte englobante du lac supérieur, et notez ses coordonnées,*
2. *Construire un programme dont le principe sera :*
   - *construire un polygone correspondant à la boîte englobante identifiée manuellement,*
   - *extruder le polygone avec chacun des polygones de la couche des limites administratives.*

Question b) Identification des berges du Lac Supérieur *On cherche à obtenir uniquement les berges du lac (de distance 0.03 unités). À la suite du programme précédent, utiliser la fonction* **Buffer** *uniquement à partir de la géométrie du Lac.*

Question c) Faire de même pour obtenir la surface du Lac à plus de 0.03 unités de la terre.

Exercice 5 (Voisinage d'une parcelle) *Dans le cadre des actions sanitaires de la DRAAF, il peut être utile d'identifier facilement (et rapidement) un ensemble de parcelles sur lesquels appliquer des traitements curatifs ou préventifs à la suite de l'identification d'un problème sanitaire sur une parcelle agricole.*

Le jeu de donnée `intern_pern_polygon.shp` est un extrait du RPG (Registre Parcellaire Graphique) reprenant les parcelles déclarées par les agriculteurs.

On souhaite mettre en place un programme pour identifier facilement un ensemble de parcelles à traiter. On s'intéressera particulièrement aux numéros d'identifiant de parcelles (attribut `IDPARCEL`) pour identifier une parcelle (NB : plusieurs polygones peuvent avoir le même identifiant de parcelle, c'est normal). Pour le début de cet exercice, vous vous intéresserez au voisinage de la parcelle `046001946_002`.

Question a) Identification des parcelles dans le voisinage de la parcelle `046001946_002` *Dans cette question, il s'agit de s'intéresser aux parcelles qui dont le centroïde est à une distance inférieure à* 300 *unités du centroïde de la parcelle `046001946_002`. Vous* ***afficherez à l'écran le numéro des parcelles*** *qui sont dans le voisinage de la parcelle cible.*

Vous commencerez par réfléchir à la stratégie générale pour arriver à cette solution, et passerez ensuite à l'implémentation.

Aide : *Quelques fonctions qui seront utiles :*

- **ResetReading**() *: fonction pour recommencer à zéro le parcours des features d'une couche*
- *pour récupérer la position le centroide d'un polygone, vous utiliserez la fonction* **Centroid**() *sur une géométrie. Cette fonction retourne un* **OGRPoint** *sur lequel vous pourrez récupérer les coordonnées grâce aux fonctions* **GetX**() *et* **GetY**()
- *le calcul de la distance entre deux points* (x, y) *et* (lx, ly) *peut se faire au moyen du code ci-dessous :*

```
import math
d = math.sqrt( (x-lx)**2 + (y-ly)**2 )
```

Question b) Identification des parcelles dans une zone tampon de la parcelle `046001946_002` *La stratégie est ici globalement la même que précédemment. On cherche sélectionner les parcelles qui se trouvent dans une région tampon autour de la parcelle cible. On utilise pour cela un buffer de taille* 300 *autour du polygone. Toutes les parcelles qui intersectent sont alors considérées comme appartenant à la zone tampon.*

Vous ***afficherez à l'écran le numéro des parcelles*** *qui sont dans le voisinage de la parcelle cible.*

Question c) Demander à l'utilisateur le numéro de la parcelle *Modifier l'un des programmes précédent pour que l'utilisateur saisisse le numéro de la parcelle cible à rechercher. L'exemple ci-dessous permet de récupérer une valeur auprès de l'utilisateur (saisie clavier). Vous ferez les modifications nécessaire pour qu'il n'y ait pas d'erreur lorsque la saisie de l'utilisateur ne correspond à aucune parcelle existante (vous pourrez même le signaler à l'utilisateur par un message à l'écran).*

```
val = raw_input("saisir le numero de la parcelle cible")
```

Question d) Gestion de deux zones *On souhaite maintenant considérer deux seuils : les parcelles qui se trouvent à moins de* 300 *unités (en distance ou en tampon) doivent subir un traitement curatif, les parcelles entre* 300 *et* 600 *unités doivent subir un traitement préventif.*

Un numéro de parcelle ne peut être que dans une seule liste. Si une parcelle est dans les deux listes, elle doit disparaître de liste des "préventifs" (elle subit le traitement plus plus contraignant)

Aide : plutôt que d'afficher les résultats, vous utiliserez des listes pour retenir les résultats.

Pour faire des différences de listes, vous pourrez utiliser l'exemple ci-dessous (notez que vous perdrez également tous les doublons) :

```
>>> l1=[45, 56, 67]
>>> l2=[12, 56]
>>> list(set(l1) - set(l2))
[67, 45]
```

Télécharger les corrections Question a, Question b.

Exercice 6 (Récupération de l'information par localisation) *L'objectif de cet exercice est de faire la fusion d'information entre couche en utilisant la localisation des polygones. Cet exercice provient d'un problème pratique de récupération d'informations issue d'une couche d'ilôts de culture*[2] *pour l'ajouter à une couche de parcelles.*

Normalement, il existe un outil dans QGis capable de faire ce genre de transformation ... sauf qu'il ne marchait pas (sur les nouvelles versions, j'espère qu'il a été modifié) et dans tous les cas, je ne savais pas exactement ce qu'il faisait ! Donc, la seule solution était de refaire un petit programme Python pour faire la même tâche.

Le principe de la méthode est le suivant : on dispose de deux couches, les ilôts et les parcelles. Pour chaque parcelle, on va son calculer isobarycentre P. Ensuite, pour chacun des ilôts I, on vérifie si P se trouve dans le polygone i. Si c'est bien le cas, on ajoute à la parcelle l'information utile, et on passe à la parcelle suivante.

Question a) Écrire en pseudo-code l'algorithme de fusion des informations par localisation. (solution données plus bas)

Dans les données proposées se trouvent dans les fichiers `Parcels.shp` *et* `Ilots.shp`.

On récupèrera l'information `PAE_ID_EXP`*, c'est-à-dire l'identifiant de l'agriculteur, ainsi que l'attribut* `COMMUNE_IL`*. Ces deux attributs sont des nombres entiers. Il n'est pas impossible qu'une parcelle ne corresponde à aucun ilôt. Dans ce cas, vous n'ajouterez aucune information à la parcelle pour ces attributs.*

Question b) Créer une fonction pour la fusion des informations par localisation en supposant que les deux fichiers ont les mêmes systèmes de coordonnées (en pratique, les fichiers proposés sont en Lambert-II)

Question c) (⋆⋆) Modifier votre programme pour faire une(des) fonction(s) de votre programme. Vous pourrez prendre en paramètre de votre fonction les noms des fichiers ainsi que deux listes d'attributs (1 pour chaque couche).

Question d) (⋆)Même question que la question 6 en utilisant le fichier `Ilots_RGB93.shp` qui reprend les mêmes ilôts mais en coordonnées Lambert 93.

! Attention ! - Temps de calcul

L'algorithme proposé ici est un algorithme basique, mais il est très peu efficace ! La complexité de l'algorithme sera de l'ordre de $I \times P$, c.-à-d. le nombre d'ilôts fois le nombre de parcelles. Pour de grandes couches de données, le temps de calcul qui en découle peut être très important. Il faudrait alors améliorer l'algorithme pour réaliser la même tâche, mais plus efficacement.

Remarque 3 - Aide : algorithmes

Le principe de l'algorithme est celui d'une recopie de la couches de parcelles, à la seule différence que lorsqu'on recopie d'une forme, il faut ajouter des attributs provenant des attributs d'un ilot couche

```
- ouvrir la couche de parcelles LP
- ouvrir la couche d'ilots LI
- creer une nouvelle couche de parcelles avec les attributs de LP et les attributs de la couche LI a rec

Pour toute parcelle P faire
    creer P' par recopie de P
    creer un point C, barycentre de P

    #recherche d'un ilot
    Pour tout ilot I faire
        Si C est dans le polygone de I
            ajouter les attributs de I à P'
            arreter la recherche des ilots

- fermer tous les fichiers
```