# L'algorithme quadripartition appliqué pour la segmentation de l'image

INF TC1

Sujet BE2

Première partie

Version élèves

2015-16

# I Modalités

## 1. Objectifs de ce BE

- Apprentissage des algorithmes et du langage Python
- Acquisition des bases pour la suite des enseignements d'Informatique à l'ECL.

## 2. Principe de ce BE

Le principe d'un certain algorithme de segmentation d'image vous est proposé. L'objectif est d'analyser le problème posé et produire un algorithme; avant de l'implanter en langage python et d'analyser son comportement. Il vous est demandé aussi de répondre à un certain nombre de question.

## 3. Travaux à rendre

- Proposez un algorithme. Implémentez celui-ci en langage python et analysez son comportement en l'appliquant notamment sur l'image fournie et en variant le paramètre seuil.
- Répondre aux questions posées.
- Ces travaux doivent être rendus dans **un délai de 2 semaines** ouvrables après la séance.
- Créer une archive (zip, rar, etc.) contenant les codes des exercices ainsi que votre compte rendu traitant les points précédents.

# II Segmentation

## 1. Description du problème

La segmentation d'image est le processus qui consiste à découper une image en régions connexes présentant une homogénéité selon un certain critère, comme par exemple la couleur, la texture, la profondeur, le mouvement, etc. L'image initiale est retrouvée par l'union de ces régions.

La segmentation est une étape importante dans l'extraction des informations qualitatives d'une image. Elle apporte une description de haut niveau sémantique. Les régions voisines sont connectées à travers un graphe ou chaque région porte une étiquette donnant des informations qualitatives discriminantes comme sa taille, sa couleur, sa forme, son mouvement ou encore son orientation. L'image est réduite à un graphe la représentant où des nœuds étiquetés contiennent presque toutes les informations utile au système. Les arcs de ce graphe peuvent être valués précisant si deux régions connectées sont en simple contact ou si l'une est incluse dans l'autre. D'autres informations topologiques peuvent également être stockées comme

par exemple le positionnement relatif : une région est au-dessous d'une autre, etc. La construction de ce graphe peut être plus ou moins complexe et dépend des techniques de segmentation utilisées.

Les algorithmes de segmentation sont généralement groupés en trois grandes catégories :

1. Segmentation à base de pixels
2. Segmentation à base de régions
3. Segmentation à base de contours

La première catégorie utilise souvent les histogrammes de l'image. Par seuillage, les différentes couleurs représentatives de l'image sont identifiées, l'algorithme construit ainsi des classes de couleurs qui sont ensuite projetées sur l'image.

La deuxième catégorie correspond aux algorithmes d'accroissement de régions ou de partitionnement de régions. Le premier est une approche bottom-up : on part d'un ensemble de petites régions uniformes dans l'image, d'un, voire de quelques pixels, et on regroupe les régions adjacentes d'une même couleur. Ce regroupement s'arrête quand aucun regroupement n'est plus possible. Le deuxième, est une méthode top-down: on part de l'image entière que l'on va subdiviser récursivement en plus petites régions tant que ces régions ne sont pas suffisamment homogènes. Un mélange de ces deux méthodes est l'algorithme de split and merge que nous allons détailler et étudier par la suite.

Finalement, la troisième catégorie utilise l'information de contours des objets. Comme la plupart de ces algorithmes fonctionnent au niveau du pixel, ils sont considérés comme des algorithmes locaux. Ce type d'algorithme est formellement proche des techniques d'accroissement de régions fonctionnant au niveau du pixel et sont purement locales et en général limitées pour traiter des images complexes et bruitées.

# 2. Approche du type "split and merge"

L'algorithme split and merge a été proposé par Pavlidis et Horowitz en 1974. Selon cet algorithme, deux étapes de split, partionnement, et de merge, fusion, sont opérées.

## 2.1 Split

La méthode de découpage de l'image utilisée dans cet algorithme est basée sur la notion de *quadripartition*, *quadtree* en anglais. Une structure de données du type

*arbre quaternaire* permet de stocker l'image à plusieurs niveaux de résolution. On part souvent de la totalité de l'image. Si cette image vérifie un certain critère d'homogénéité de couleur, l'algorithme s'arrête. Sinon, on découpe cette région en quatre parties de la même taille et on relance, récursivement, cette procédure dans chacune des quatre parties. La région initiale est considérée comme un nœud dans un graphe et les sous parties comme les quatre fils de ce nœud.

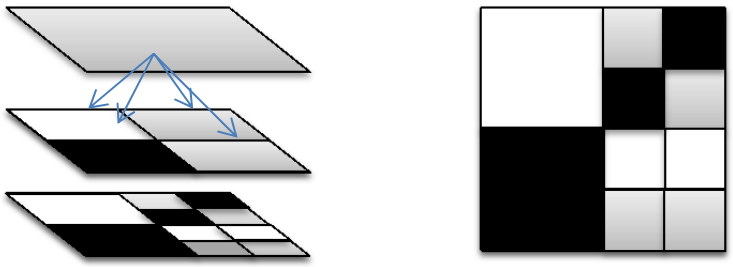

Figure 1. Découpage par quadripartition d'une image 4x4 pixels.

La Figure 1 montre une image en noir et blanc 4x4 pixels et le découpage correspondant en trois niveaux. La structure d'arbre associée à ce découpage est illustrée Figure 2.

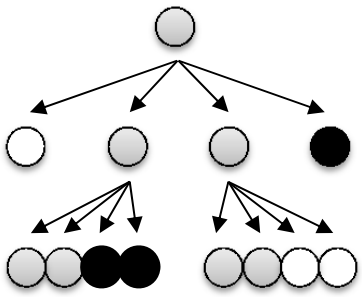

Figure 2. Arbre quaternaire à partir de l'image de la Figure 1.

Dans cet exemple, le critère d'homogénéité est absolu. Une zone est dite homogène si elle ne contient que des pixels de la même couleur (seuil d'homogénéité = 100%). En pratique on est plus tolérant et considère qu'une zone est homogène dès que plus de 75% d'une couleur domine.

De manière plus générale, on va appliquer ce principe de réduction à des images colorées. Chaque pixel d'une image en couleur est représenté par trois intensités en

rouge, vert et bleu. Chaque intensité est codée sur un octet, sa valeur varie de 0 à 255. Le critère d'homogénéité est fixé par un seuil. En dessous de ce seuil, la région est conservée et constitue une feuille, nœud terminal, de l'arbre. On lui attribue alors la couleur de la moyenne des pixels la constituant. Au-dessus de ce seuil, la région est découpée en quatre.

## 2.2 Merge

La procédure de découpage décrite précédemment aboutit à un nombre de régions parfois trop élevé. Il se peut qu'elle coupe de cette façon une zone homogène en deux ou quatre parties.

La solution, qui correspond à la phase fusion, merge, de l'algorithme, est de procéder à une fusion de régions après le découpage. L'implémentation la plus simple de cette fusion consiste à rassembler, fusionner, les couples de régions adjacentes repérées, de couleur proche, dans l'arbre issu de la phase split.

Pour réaliser cette fusion, il faut d'abord tenir à jour une liste des contacts entre régions (chaque région dispose d'une liste de régions avec lesquelles elle est en contact). On obtient ainsi un graphe d'adjacence de régions ou *Region Adjacency Graph* (Ce graphe de contact doit se construire en même temps que l'arbre de découpage). Ensuite, l'algorithme va marquer toutes les régions comme non-traitées, et choisir la première région R non traitée disponible.

Les régions en contact avec R sont empilées et examinées les unes après les autres pour savoir si elles doivent fusionner avec R. Si c'est le cas, la couleur moyenne de R est mise à jour et les régions en contact avec la région fusionnée sont ajoutées à la pile des régions à comparer avec R. La région fusionnée est marquée comme traitée. Une fois la pile vide, l'algorithme choisi la prochaine région marquée comme non traitée et recommence, jusqu'à ce que toutes les régions soient traitées.

## 2.3 Exemple

Un exemple d'image traitée par l'algorithme *quadripartition* est illustré par la Figure 3. L'image originale est traitée avec des valeurs de seuil de plus en plus petites augmentant le nombre de régions détectées.


Figure 3. Image de Lyon et les résultats obtenus avec différentes valeurs de seuil. En haut à gauche, l'image originale non segmentée. Les suivantes avec des valeurs de seuil de plus en plus basses augmentant le nombre de régions.

# 3. Travail à réaliser

On souhaite réaliser l'algorithme de split qui utilise la stratégie *quadripartition*.

Ceci nécessite l'utilisation de la librairie PILLOW fourni avec Anaconda. Nous avons notamment besoin de lire une image à partir de son nom dans le répertoire courant :

```
im = Image.open("Image8.bmp").
```

Importer les données des pixels sous forme d'une matrice px :

```
px = im.load() #Importation des pixels de l'image
```

Obtenir la taille de cette image :

```
w,h=im.size
```

Utiliser, et donc importer, la fonction mathématique racine carrée, sqrt, qui se trouve dans la librairie math :

```
from math import sqrt
```

Vous devez donc exécuter la séquence suivante au début de votre programme.

```
from PIL import Image #Importation de la librairie d'image PIL
im = Image.open("Image8.bmp") #Ouverture du fichier d'image
px = im.load() #Importation des pixels de l'image

from math import sqrt #Importation de la fonction sqrt de la librairie math

w,h=im.size
```

Par la suite on peut accéder au pixel px[x,y] de coordonnées x, y par la commande python suivante :

```
p=px[x,y]
```

Pour affecter une couleur r,g,b au pixel p[x,y], on utilisera la commande :

```
px[x,y] = r,g,b
```

où r,g et b sont des variables.

Si les commandes de lecture d'image et d'accès aux pixels ne sont pas disponibles, c'est que la librairie PILLOW n'est pas installée. Pour

l'installer vous devez exécuter la commande suivante dans une fenêtre de commande Windows.

```
Microsoft Windows [version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.


C:\Users\Ardabilian>pip install Pillow
Downloading/unpacking Pillow
Installing collected packages: Pillow
Successfully installed Pillow
Cleaning up...

C:\Users\Ardabilian>
```

a) Ecrire une fonction permettant de lire la valeur d'un pixel.

b) Ecrire une fonction permettant d'affecter une couleur à un pixel.

c) Ecrire une fonction permettant d'affecter une couleur à une région rectangulaire de l'image.

d) Proposez une fonction qui estime l'homogénéité des pixels d'une région rectangulaire de l'image.

e) Proposez un algorithme récursif permettant de reproduire la stratégie *quadripartition*.

f) Selon vous, la réalisation récursive d'un *arbre quaternaire*, est-elle indispensable pour l'étape de split ? Qu'en est-il pour les deux étapes split et merge ?

g) Ecrire l'algorithme de split en langage python.