Projet informatique : segmentation d'image par « découpage-fusion »

IN 101 – ENSTA

`http://www.di.ens.fr/~pointche/enseignement/ensta/projet/`

Résumé

Le but de ce projet est de découper une image en régions homogènes selon un algorithme de segmentation appelé « découpage-fusion ». Dans un premier temps, cet algorithme découpe l'image en quatre régions, puis, selon la variance de chacune des régions, réitère ce découpage sur chaque rectangle obtenu. Le processus s'arrête lorsque les régions sont, soit trop petites, soit homogènes.
Dans un deuxième temps, les régions voisines qui se ressemblent sont réunies. Ce processus est réitéré jusqu'à ce qu'il soit impossible de trouver deux régions voisines suffisamment semblables.
Ce type d'algorithme est la clé de voûte de la plupart des traitements de reconnaissance de formes utilisés en vision par ordinateur.

1 Modalités de ce projet

Ce projet peut être effectué par binôme (2 personnes). Son évaluation sera effectuée sur la base

- d'une disquette ou d'un e-mail (à `David.Pointcheval@ens.fr`) avec une archive compressée (`.tar.gz` ou `.tgz`) regroupant
  - les sources (veiller à la clarté des sources, et ne pas hésiter à commenter chaque étape) ;
  - un `Makefile` qui effectue la compilation de l'exécutable `projet` (voir la section 10.1 page 76 du polycopié, au sujet de l'utilitaire `make`, que vous avez intérêt à utiliser dès le début de vos travaux).

  Ce dernier sera testé sur les images proposées sur le site Web ;
- d'un rapport écrit, d'au plus 2 pages, détaillant vos réflexions et les problèmes rencontrés, ainsi que ce que fait effectivement votre programme (parfaitement opérationnel, certaines restrictions, ...) ;
- d'un listing du code source, ou des modules essentiels (au plus 400 lignes ou 4 pages).

Le tout doit être remis le **6 décembre**, avec le contrôle de connaissances, pour les documents physiques, par e-mail pour le reste (également le 6 décembre, 12h00, au plus tard).

Consulter la page `http://www.di.ens.fr/~pointche/enseignement/ensta/projet` pour les fichiers à charger, ainsi que toute information supplémentaire. Une liste de questions/réponses y sera proposée.

Ne pas hésiter à interroger vos enseignants (et tout particulièrement Bertrand Collin – `Bertrand.Collin@etca.fr` et David Pointcheval – `David.Pointcheval@ens.fr`), non pas pour vous faire le programme, mais pour vous éclairer en cas de problème.

2 Présentation du projet

2.1 Sommaire

Pour faire de la reconnaissance de formes à l'aide d'un système de vision, une solution courante est de structurer l'image en régions. Une image numérique est en effet avant tout un ensemble de pixels liés les uns aux autres par de simples relations de voisinage. Il est fondamental de pouvoir regrouper au sein d'une même entité (la *région*) tous les pixels voisins les uns des autres (nous parlerons de *relation de connexité*) de façon à ce que ce groupe de pixels soit homogène. Différents critères d'homogénéité existent, nous n'aborderons dans le projet que l'homogénéité des niveaux de gris.

Le but de ce projet est donc de réaliser un programme qui
- lit un fichier image au format « GIF »,
- découpe l'image de manière itérative jusqu'à obtenir un ensemble de régions rectangulaires homogènes,
- fusionne itérativement les régions rectangulaires voisines de telle manière que le résultat de la fusion soit lui aussi homogène,
- sauvegarde le résultat de cette fusion dans un fichier image en affectant à chaque pixel le niveau de gris moyen de la région à laquelle il appartient.

2.2 Les structures de données

Nous conviendrons de représenter une image comme une structure composée des champs suivants :
- `int w;` la largeur de l'image
- `int h;` la hauteur de l'image
- `unsigned char *d;` un tableau de `w*h` octets représentant les niveaux de gris des différents pixels de l'image. Le niveau de gris sera donc compris entre 0 et 255.

Ceci conduit à la définition de la structure suivante :

```
typedef struct {
  int w;
  int h;
  unsigned char *d;
} IMAGE;
```

Nous aurons besoin de conserver une image qualifiée d'« étiquettes » dans laquelle l'étiquette sera un entier (`int`). Tous les pixels avec la même étiquette sont dans la même région, à priori homogène. Ceci conduit à la structure suivante :

```
typedef struct {
  int w;
  int h;
  int *d;
} EIMAGE;
```

Dans ces structures nous conviendrons de respecter les conventions du langage `C` en faisant démarrer les indices de lignes et de colonnes à zéro. Ainsi le pixel situé en haut à gauche de l'image possédera les coordonnées `(0,0)`, celui en bas à droite les coordonnées `(w-1,h-1)`.

Nous proposons aussi la structure de données relatives aux régions :

```
typedef struct {
  int type;
  int etiquette;
  int new_etiquette;
  double moyenne;
  double variance;
  int nb_pixel;
  int xbup,ybup;
  int xbdw,ybdw;
  int *voisine;
  int num_voisine;
} Region;
```

Les champs sont décrits dans le fichier de définition `projet_defs.h`, mais en voici une brève description, une région étant caractérisée par

– un type qui représente son état durant les 2 processus, celui de découpage et celui de fusion ;
– une étiquette et une future étiquette ;
– la moyenne et la variance des niveaux de gris des pixels qui la composent ;
– le nombre de pixels qu'elle contient ;
– les coordonnées, dans l'image, du coin supérieur gauche et du coin inférieur droit du rectangle qui la circonscrit ;
– un tableau contenant les étiquettes des régions qui lui sont voisines ;
– le nombre de ses voisines.

2.3 Utilitaires à votre disposition

Différents fichiers sont mis à votre disposition :

– `projet_defs.h` : ce fichier contient les définitions de structures des types `IMAGE` et `EIMAGE` et propose la définition ci-dessus de la structure `Region`, ainsi que les macros suivantes :
  – `LARG(image)` : la largeur de l'image, soit le nombre de colonnes ;
  – `HAUT(image)` : la hauteur de l'image, soit le nombre de lignes ;
  – `Pix(image,i,j)` : le niveau de gris, ou l'étiquette, du pixel situé à l'intersection de la ligne `j` et de la colonne `i`.
– `projet_io.c` : ce fichier contient les fonctions permettant de gérer les entrées/sorties ainsi que les allocations/désallocations des structures images. Leurs prototypes sont les suivants :
  – `void *AllocImage(int w, int h, int type)` : allocation d'une image de largeur `w`, de hauteur `h` et de type `type` (soit `IMAGE_TYPE`, soit `EIMAGE_TYPE`). La fonction retourne un pointeur sur la structure nouvellement allouée, donc soit de type `IMAGE *`, soit de type `EIMAGE *` (d'où le pointeur générique `void *`), selon la valeur de `type`. Attention les niveaux de gris, ou étiquettes, des pixels ne sont pas initialisés.
  – `void *FreeImage(void *s)` : libération d'une structure image (de type `IMAGE *` ou `EIMAGE *`, d'où le pointeur générique `void *`). La fonction retourne le pointeur `NULL`.
  – `IMAGE *read_file(char *name)` : lecture d'une image au format GIF. La fonction retourne un pointeur sur une structure `IMAGE` fraîchement allouée contenant les niveaux de gris des pixels contenus dans le fichier `name`.

  - `int write_file(IMAGE *inputdata,char *fpname)` : écriture de l'image pointée par `inputdata` au format GIF dans le fichier `fpname`. La fonction retourne 0 si tout se passe bien, ou une valeur non nulle en cas d'erreur. La structure image n'est pas modifiée.
- `projet_io.h` : ce fichier contient les prototypes des fonctions de lecture, écriture et allocation qui viennent d'être présentées.
- La librairie GIF est fournie sous forme d'une archive `libgif.a` (compilée pour PC/Linux), avec son fichier d'en-têtes `gif_lib.h`. Cependant, elle est peut-être déjà installée sur votre système, dans quel cas vous n'aurez rien à faire. Sinon, copiez ces deux fichiers dans le répertoire courant du projet.

2.4 Fichiers

Votre collaboration au projet pourrait être répartie dans les trois fichiers suivants :
- `split.c` les fonctions de découpage itératif ;
- `merge.c` les fonctions de fusion itérative ;
- `main.c` le fichier contenant la fonction `main(...)`.

Il sera impératif d'inclure, dans chacun de ces fichiers personnels, les en-têtes de définition de types :

```
#include "projet_defs.h"
#include "projet_io.h"
```

2.5 Compilation

Vous pourrez compiler chaque module séparément (voir la section 7.3 page 58 du polycopié sur les modules et la programmation séparée) :

```
gcc -c -Wall -I. projet_io.c
gcc -c -Wall -I. split.c
gcc -c -Wall -I. merge.c
gcc -c -Wall -I. main.c
```

Enfin, vous construirez l'exécutable en éditant les liens entre les différents modules et les librairies, soit `-lm` pour la librairie mathématique et `-lgif` pour la librairie GIF. Si votre répertoire ne contient que les fichiers `.o` précédemment cités, la ligne de commande suivante peut être utilisée :

```
gcc *.o -o projet -L. -lm -lgif
```

Mais il vous est conseillé d'écrire ces lignes dans un fichier `Makefile` qui permettra la compilation automatique des différents fichiers à l'aide de la commande `make` (voir la section 10.1 page 76 du polycopié, au sujet de l'utilitaire `make`).

3 La notion de région

3.1 Notion de voisinage

En traitement d'images numériques, une région est un ensemble de pixels connexes. On appelle voisinage *8-connexe* d'un pixel, l'ensemble des pixels qui se situent à une distance eucli-

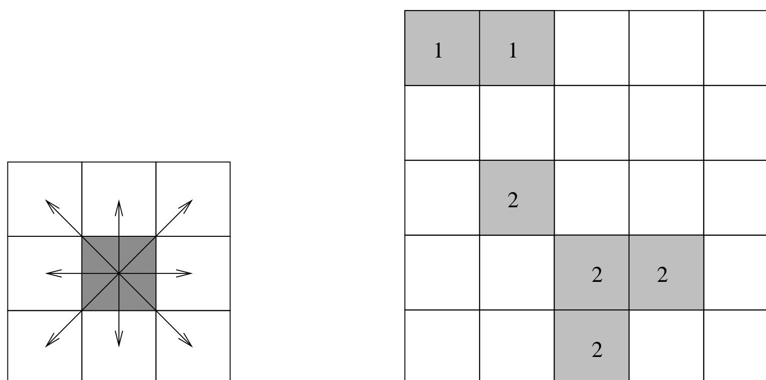


Fig. 1 – La figure de gauche montre les positions des 8 voisins connexes du pixel grisé. La figure de droite montre le résultat d'une partition des pixels grisés selon la connexité. Il est impossible de passer de la région 1 à la région 2 en suivant des voisins *8-connexes* grisés.

---

dienne inférieure ou égale à $\sqrt{2}$ de ce pixel. La figure 1 donne un exemple de partition selon la connexité.

3.2 Région à géométrie simple ou compliquée

Dans le cas de la fonction de découpage itératif, l'image est découpée tout d'abord en quatre parties égales, délimitant quatre régions. Lesquelles seront à leur tour découpées en quatre parties égales, etc... Comme on peut le constater, toutes les régions issues de ce processus sont des rectangles. Il suffit de conserver en mémoire les coordonnées du pixel situé en haut à gauche de la région, ainsi que la largeur et la hauteur de la région. Parcourir une région est alors identique à parcourir une image, du point de vue des coordonnées :

```
for(j=0;j<hauteur_region;j++)
  for(i=0;i<largeur_region;i++) {
    x_dans_image = i + x_debut_region;
    y_dans_image = j + y_debut_region;
  }
```

Pendant le processus de fusion des régions, la géométrie va se compliquer. Il devient alors impératif de se doter d'un autre moyen pour être capable de repérer les pixels d'une région. On utilise pour cela une image d'étiquettes de même taille que l'image d'origine. Cette image d'étiquettes est telle que :

> *Tous les pixels d'une région R_i possèdent l'étiquette i (dans l'image d'étiquettes) et réciproquement, tous les pixels dont l'étiquette est j (dans l'image d'étiquettes) appartiennent à la région R_j.*

3.3 L'algorithme de découpage itératif

Le découpage itératif vise à obtenir un ensemble de régions tel que chaque région repère un rectangle dans l'image et que la variance du niveau de gris des pixels de ce rectangle soit inférieure à un certain seuil qui sera donné en paramètre au programme. La figure 2 donne un exemple de découpage itératif. Le découpage s'arrête donc, soit quand toutes les régions sont

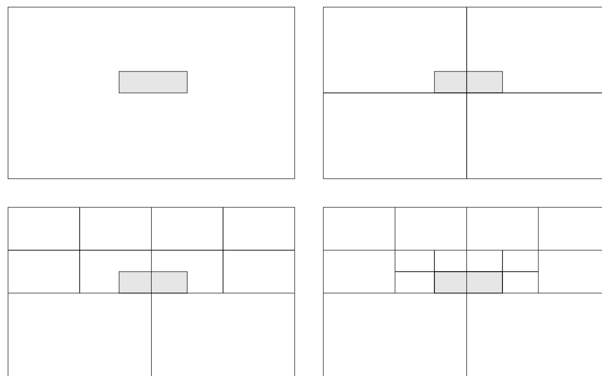


Fig. 2 – La position du rectangle noir dans l'image fait que le découpage doit être itéré plusieurs fois jusqu'à ce que la dichotomie coïncide avec les bords du rectangle noir. Trois étapes de découpage sont donc nécessaires pour obtenir des régions homogènes.

---

homogènes (variance inférieure au seuil), soit quand il n'est plus possible de diviser les régions non homogènes parce qu'elles sont trop petites.

Afin de réaliser cette fonction, il est possible de balayer un tableau de régions et en fonction des qualités de chaque région, de ranger la région analysée dans un nouveau tableau si celle-ci est homogène, ou de ranger 4 régions fabriquées en découpant la région analysée si celle-ci n'est pas homogène. L'ancien tableau est détruit et on renvoie le nouveau tableau. On évitera d'analyser à nouveau des régions dites homogènes en conservant dans la structure tableau un drapeau « d'homogénéité », le champ `type` de la structure `Region`, qui permettra aussi de différencier les régions devant être découpées, celles devant être gardées et celles dont on ignore tout. On pourra se servir des constantes suivantes (également définies, entre autres, dans le fichier `projet_defs.h`) :

- `#define REGION_JESAISPAS 0`
- `#define REGION_AGARDER   1`
- `#define REGION_ADECOUPER 2`

3.4 L'algorithme de fusion itératif

Comme on le remarque dans la figure 2, la fonction de découpage conduit bien souvent à un découpage abusif de l'image. Il faut donc procéder au regroupement familial des régions voisines telles que leur fusion conduise à la création d'une région toujours homogène. Différentes techniques existent, nous allons en programmer une assez simple sur le plan algorithmique, malheureusement pas très rapide. Une version plus efficace est proposée section 4. Vous pourrez d'emblée implémenter la version plus efficace, pour les programmeurs avertis, ou dans un deuxième temps comme amélioration si vous le souhaitez. Dans tous les cas, vous avez intérêt à lire la description suivante.

Nous devrons impérativement respecter le principe de réciprocité qui suppose que la $i$ème région du tableau des régions possède l'étiquette i et donc que chaque pixel d'étiquette i dans l'image étiquettes appartient à la $i$ème région. La fusion peut être programmée de la façon suivante :

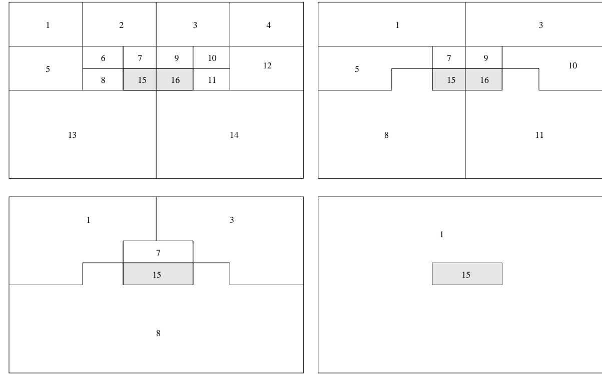


FIG. 3 – Les régions sont fusionnées les unes après les autres. On conserve à chaque fois une seule des deux étiquettes. On remarque la nécessité de conserver une image d'étiquettes car les régions ne sont plus rectangulaires et ne peuvent donc plus être décrites par une primitive géométrique simple.

```
(1)   etq = Creer_Image_Etiquette(image_source,
                                   region, nb_region);
(2)   Construire_Tableau_Voisines(etq,region,nb_region);
(3)   Tant que c'est possible
(4)     Pour i allant de 0 et i < nb_region {
(5)       region = Fusionner_Voisine(region,i,etq,
                                     &nb_region,&nb_fusion);
(6)       si nb_fusion != 0
            Construire_Tableau_Voisines(etq,region,nb_region);
(7)     }
```

Nous allons expliciter en détail chaque étape.

3.4.1 Créer l'image d'étiquettes (ligne (1))

Dans un premier temps nous allons créer l'image d'étiquettes qui correspond à la répartition spatiale des régions issues de la phase de découpage. Cette création est assez simple puisque, rappelons-le, à l'issue du découpage, toutes les régions sont des rectangles plus ou moins grands dont nous connaissons les coordonnées du point de départ dans l'image (`xbup,ybup` dans la structure d'une région) et les coordonnées du point final (`xbdw,ybdw` dans la structure d'une région). La création se résume donc, pour chaque région, à remplir un rectangle de l'image d'étiquettes avec l'indice de la région dans le tableau d'étiquettes :

```
Pour i allant de 0 et i < nb_region
  Pour y = ybup_i et tant que y < ybdw_i
    Pour x = xbup_i et tant que x < xbdw_i
      Placer i dans etiquette en x,y
```

Nous verrons, par la suite (comme le montre la figure 3), que la géométrie des régions n'étant plus rectangulaire, c'est l'image étiquettes qui nous permettra de repérer les régions dans l'image.

Nous passons alors à la deuxième étape, la fonction qui associe à chaque région un tableau donnant l'indice de ses voisines.

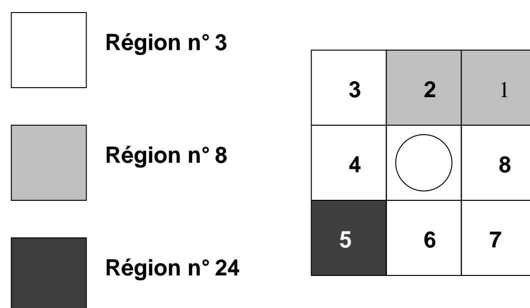


FIG. 4 – L'analyse porte sur le pixel cerclé qui appartient à la région d'indice 3 du tableau des régions. Les pixels voisins sont numérotés de (1) à (8). Le voisin (1) ne possède pas le même niveau de gris que le pixel cerclé, la région à laquelle il appartient (région d'indice 8 dans le tableau des régions) est donc une voisine de la région d'indice 3. On ajoute donc 8 dans le tableau de voisines de la région 3 (champ `voisine` de la structure `Region`) et réciproquement, on ajoute 3 dans le tableau de voisines de la région 8. On procède de la même façon avec le voisin (2), à ceci-près qu'appartenant lui aussi à la région 8, aucune modification ne sera apportée aux tableaux de voisines des régions 3 et 8.

3.4.2 Recherche des régions adjacentes (lignes (2))

Dans la pratique une telle recherche se réalise par la création d'un graphe d'adjacence. Nous allons procéder plus simplement, mais au détriment du temps de calcul. Il faudra donc éviter de tester votre programme sur des images 1600×1200 (d'où l'intérêt du cours d'algorithmique qui va suivre). La version améliorée présentée section 4 s'inspire de cette notion de liste d'adjacence.

Il s'agit d'analyser l'image étiquettes et de repérer les régions qui sont voisines. Pour cela, on se place en un point (i, j) de l'image d'étiquettes et on analyse les points voisins, au nombre de huit (voir à ce sujet le paragraphe 3.1). Nous savons que le point de coordonnées (i, j) de l'image d'étiquettes désigne un point de la région `R[Pix(etq,i,j)]` (correspondance entre étiquette et indice de la région dans le tableau des régions). Par conséquent, si un des voisins possède un niveau différent, ce voisin désigne une région voisine de `R[Pix(etq,i,j)]`. La figure 4 donne un exemple d'analyse.

3.4.3 Fusionner les régions (ligne (5))

C'est de loin la partie la plus importante du mécanisme de fusion et celle à laquelle il faut apporter le plus grand soin sous peine de `SIGSEGV` (erreur de segmentation) ou de résultats fantaisistes !

On analyse le tableau de voisines de la région d'indice i et on regarde si la valeur absolue de la différence des moyennes entre R_i et chacune de ses voisines est plus faible qu'un certain seuil. Si c'est le cas, on place dans un tableau le fait que la voisine est candidate à la fusion.

Une fois toutes les voisines examinées, on procède à la fusion entre la région courante et les voisines candidates. Pour cela, on doit :

(a) placer, dans les différents champs de R_i, le résultat de la fusion, que ce soit pour le nombre de pixels, la moyenne ou la variance. Dans les champs `xbup`, `ybup`, `xbdw` et `ybdw`, on place le MIN ou le MAX (selon le cas) des champs des 2 régions ;

(b) changer le type des voisines qui fusionnent de manière à ce qu'elles deviennent des `REGION_ADEGAGER`

(c) préparer le travail pour le changement des étiquettes, puisque les étiquettes relatives aux voisines qui fusionnent doivent changer de valeur. On utilise pour cela le champ `new_etiquette` de la région. Une voisine qui fusionne aura donc une nouvelle étiquette correspondant à l'étiquette de R_i, soit i.

La première étape de la fusion ayant été réalisée, on doit maintenant reconstruire le tableau de régions (certaines régions disparaissent) et mettre à jour l'image d'étiquettes pour prendre en compte les changements. Il faudra veiller lors de cette étape à obtenir un nouveau tableau et des étiquettes qui respectent le principe de réciprocité. On n'oubliera pas non plus de changer le nombre de régions, cette valeur contrôlant bon nombre d'opérations.

3.4.4 Dernière étape (ligne (6))

Une fois les changements avalisés, il ne reste plus qu'à reconstruire les tableaux de voisines. Une seule chose, veiller à libérer la mémoire convenablement, car une allocation non libérée a peu de conséquences sur un tableau de 5 éléments, mais dans le cas du traitement d'images, la moindre image conduit très vite à des tableaux de 10.000 régions.

3.5 Et pour finir

Une fois les deux étapes du processus de segmentation réalisées, on pourra créer une image de type `IMAGE` dans laquelle chaque pixel possédera un niveau de gris égal au niveau de gris moyen de la région à laquelle il appartient. En faisant varier le seuil d'homogénéité, on pourra constater que la représentation de l'image change. Ce type de segmentation peut conduire à des procédés de compression.

4 Objectifs et/ou extensions : fusion itérative améliorée

Le but de ce projet est de vous familiariser avec le langage C, il est donc licite de procéder par « petits bouts » ! On pourra dans un premier temps réaliser un seul découpage, et vérifier qu'il est correct avant de mettre en place la procédure itérative. Par la suite on pourra réaliser une animation en effectuant une sauvegarde d'images intermédiaires traduisant le niveau de gris moyen des régions en cours de construction. On pourra aussi chercher d'autres caractéristiques d'homogénéité.

Mais tout d'abord, on pourra tenter d'améliorer l'efficacité de l'algorithme, notamment la phase de fusion. En effet, le processus de fusion est bien souvent très long. Il effectue de nombreuses mises à jour et passe plus de temps à reconstruire les tableaux de voisinage qu'à procéder à la fusion de régions ! Nous proposons ici une solution plus rapide pour réaliser la fusion.

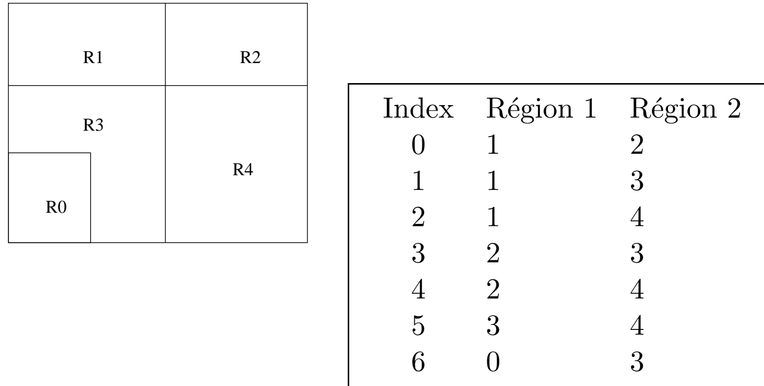
4.1 Présentation abrégée

A l'issue du processus de découpage, on procède toujours à la création de l'image d'étiquettes tel que décrit dans le paragraphe 3.4.1 ainsi qu'à la création des tableaux de voisinage de chaque région (paragraphe 3.4.2).

On introduit ensuite une nouvelle structure qui permet de conserver toutes les relations de voisinage qui existent entre les différentes régions. Ceci pourrait être réalisé au moyen d'une matrice M dans laquelle on ne trouverait que des 0 ou des 1, un 0 en $M(i,j)$ signifiant que les régions i et j ne sont pas voisines alors que la valeur 1 en $M(k,l)$ signifie que les régions sont voisines. La relation « être voisin » étant symétrique, la matrice serait donc symétrique et la moitié de cette matrice ne servirait à rien. C'est pour cela que l'on choisit de conserver les relations de voisinage dans un tableau de structures.

Ensuite nous rentrons dans le processus de fusion qui va examiner le tableau de voisinage, chercher les deux régions voisines les plus proches au sens de l'homogénéité et les fusionner en prenant soin de mettre à jour :

- la nouvelle région obtenue par fusion,
- le tableau de voisinage,



Index	Région 1	Région 2
0	1	2
1	1	3
2	1	4
3	2	3
4	2	4
5	3	4
6	0	3

FIG. 5 – Un découpage très simple de l'image donne un tableau de relations peu complexe. Nous remarquons que les étiquettes du champ `reg1` sont toujours plus petites que les étiquettes du champ `reg2` ce qui permet d'éviter que les entrées du type $R_k \longrightarrow R_l$ et $R_l \longrightarrow R_k$ soient toutes les deux présentes.

– les listes de voisinage des régions.

Nous allons examiner ces étapes plus en détail.

4.2 Le tableau de relations de voisinage

La structure proposée est la suivante :

```
typedef struct {
   int valid;
   int reg1;
   int reg2;
} Voisinage;
```

Le champ `valid` permet de savoir si la relation est encore valide ou non car pendant le processus de fusion certaines régions disparaissent en s'agrégeant.

Les champs `reg1` et `reg2` identifient les étiquettes (ou les indices dans le tableaux de régions puisqu'il y a équivalence) des deux régions voisines.

Prenons le cas d'école où l'image est découpée en cinq régions, `R0`, `R1`, `R2`, `R3` et `R4`. La figure 5 décrit l'image ainsi que le tableau de relations qui en résulte. Lors de la création des tableaux de voisines dans chaque région, nous pouvons calculer combien de relations de voisinage de type $R_k \longrightarrow R_l$ avec $k < l$ ont été créées. Nous pouvons ainsi allouer le tableau de structures `Voisinage`. On remplit alors le tableau de la manière suivante :

```
indice_relation = 0;
Pour chaque région d'indice K faire
   Pour chaque voisine faire
      Si K<indice_voisine
         voisinage[indice_relation].valid = 1;
         voisinage[indice_relation].reg1  = K;
         voisinage[indice_relation].reg2  = indice_voisine;
         indice_relation += 1;
```

4.3 Le processus de fusion

La fusion devient moins difficile à gérer, il suffit en effet de parcourir le tableau des relations et de chercher parmi les relations qui sont valides, celles pour lesquelles on obtient le meilleur score d'homogénéité (par exemple la valeur absolue de la différence des moyennes des deux régions est la plus faible). On procède alors à la fusion des deux régions qui sont décrites dans la relation. Cette fusion doit impérativement être menée comme suit. On suppose que la région `R1` et la région `R2` fusionnent :

- on place dans `R1` les paramètres de la nouvelle région (moyenne, variance, etc ...) ;
- on remplace les relations de voisinage de type « `R2` voisine de `RXX` » en relation de type « `R1` voisine de `RXX` » si `RXX` n'est pas déjà dans le tableau des voisines de `R1`. Si `RXX` appartient déjà au tableau de voisines de `R1`, on invalide cette relation (champ `valid`) ;
- on remplace les relations de voisinage de type « `RXX` voisine de `R2` » en relation de type « `RXX` voisine de `R1` ». Si `R1` appartient déjà au tableau de voisines de `RXX` on invalide la relation ;
- on indique que la région `R2` n'existe plus (champ `type` à une valeur particulière) ;
- on prépare la mise à jour des étiquettes en plaçant la valeur du champ `new_etiquette` de `R1` dans le champ `new_etiquette` de `R2` ;
- on rend invalide la relation `R1` voisine de `R2` dans le tableau de relations ;
- et enfin on enlève `R2` du tableau des voisines de `R1`.

Cela peut paraître long, mais en fait, chacune de ces opérations est assez rapide et au total on gagne beaucoup de temps car on effectue de simples mises à jour plutôt que de reconstruire les relations de voisinage.

4.4 Les prototypes des fonctions

Nous indiquons ci-après les prototypes des fonctions qu'il serait bon de définir pour réaliser la fusion « améliorée » :

- Ajouter l'indice d'une région dans le tableau des voisines. Cette fonction prend le tableau des régions ainsi que l'indice `val1` de la région sur laquelle on travaille et l'indice `val2` de la région que l'on désire ajouter dans le tableau de voisines. Si l'ajout est possible alors la fonction retourne `1`. Dans le cas contraire (la région d'indice `val2` appartient déjà aux voisines de la région d'indice `val1`) la fonction retourne `0`.

```
int ajouter_voisine( Region *region,
                     int val1,
                     int val2,
                     int nombre_region )
```

- Enlever l'indice d'une région dans le tableau des voisines. Cette fonction prend le tableau des régions ainsi que l'indice `val1` de la région sur laquelle on travaille et l'indice `val2` de la région que l'on désire enlever du tableau de voisins. Si l'éviction est possible alors la fonction retourne `1`. Dans le cas contraire (la région d'indice `val2` n'appartient pas aux voisins de la région d'indice `val1`) la fonction retourne `0`.

```
int enlever_voisine( Region *region,
                     int val1,
                     int val2,
                     int nombre_region )
```

- Initialisation du tableau des relations : cette fonction alloue et instancie le tableau des relations. Elle place le nombre de relations instanciées dans `nombre_relation` passé par adresse (en pointeur).

```
Voisinage *initialisation_tableau_voisinage(
                  Region *region,
                  int nombre_region,
                  int *nombre_relation )
```

- Recherche de la meilleure fusion : cette fonction doit parcourir le tableau de relations et retourne l'indice de la relation qui est la meilleure candidate à la fusion.

```
int chercher_meilleure_fusion(
                  Region *reg,
                  int nombre_region,
                  Voisinage *relation,
                  int nombre_relation,
                  double seuil )
```

- Fusionner deux régions : cette fonction essaie de fusionner les deux régions décrites dans la relation d'indice `index_fus`, à savoir la région `relation[index_fus].reg1` et la région `relation[index_fus].reg2`. La fonction remet à jour les tableaux de voisins ainsi que le tableau de relations.

```
int fusionner_deux_regions( Region *reg,
                            int nombre_region,
                            Voisinage *relation,
                            int nombre_relation,
                            int index_fus)
```

5 Rappels

5.1 Un peu de statistiques

Soit une région R composée de N pixels. On rappelle que la moyenne et la variance des niveaux de gris de la région R sont données par :

$$\mu = \frac{1}{N}\sum_{(i,j)\in R} Pix(im,i,j)$$

$$\sigma = \frac{1}{N}\sum_{(i,j)\in R} (Pix(im,i,j))^2 - {\mu_i}^2$$

Ces deux quantités peuvent donc être calculées au sein d'une seule boucle par : Ceci conduit à la définition de la structure suivante :

```
mu = 0.0;
sigma = 0.0;
for(j=0;j<hauteur_region;j++) {
  for(i=0;i<largeur_region;i++) {
    x_dans_image = i + x_debut_region;
    y_dans_image = j + y_debut_region;
    mu += (v=Pix(im,x_dans_image,y_dans_image));
    sigma += v*v;
  }
}
mu /= hauteur_region * largeur_region;
sigma /= hauteur_region * largeur_region;
sigma -= mu*mu;
```

On rappelle enfin que connaissant la région R_1 (N_1 pixels) de moyenne et variance μ_1, σ_1 et la région R_2 (N_2 pixels) de moyenne et variance μ_2, σ_2, la moyenne et la variance de la fusion des deux régions en R_f donne une moyenne et une variance de :

$$N_f = N_1 + N_2$$

$$\mu_f = \frac{N_1.\mu_1 + N_2.\mu_2}{N_f}$$

$$\sigma_f = \frac{N_1.(\sigma_1 + {\mu_1}^2)}{N_f} + \frac{N_2.(\sigma_2 + {\mu_2}^2)}{N_f} - {\mu_f}^2$$

5.2 Maintenir les étiquettes

Lors de la fusion des régions, certaines régions disparaissent et donnent leurs pixels à une région, d'autres régions changent d'indice dans le tableau des régions reconstruit. Afin de gérer facilement l'image d'étiquettes et pour maintenir le principe de réciprocité, on rappelle que la relation « avoir l'étiquette de » est transitive. Supposons qu'une région R_k disparaisse au profit de la région R_j et que la région R_j occupe, après reconstruction, la place m dans le tableau des régions, nous avons donc les relations :

$$R_k \longrightarrow R_j \longrightarrow m.$$

Si l'on prend soin de placer la valeur j dans le champ `new_etiquette` de la région R_k et la valeur m dans celui de la région R_j, alors tous les pixels d'étiquette k se verront affecter l'étiquette :

```
region[region[k].new_etiquette].new_etiquette
```