SEGMENTATION

Module D9

ES322 - Traitement d'Image et Vision Artificielle

Jean-Christophe Baillie

2003

Segmentation

1 Description du problème

Fondamentalement, la segmentation est un processus qui consiste à découper une image en régions connexes présentant une homogénéité selon un certain critère, comme par exemple la couleur. L'union de ces régions doit redonner l'image initiale ([8, 7, 11, 10, 14]).

La segmentation est une étape importante pour l'extraction des informations qualitatives de l'image. Elle fournit une description de haut niveau : chaque région est connectée à ses voisines dans un graphe et chaque région porte une étiquette donnant des informations qualitatives comme sa taille, sa couleur, sa forme, son orientation. L'image se réduit donc à un graphe de noeuds étiquetés qui contient presque toute l'information utile au système. Les arcs du graphe peuvent être valués précisant si les 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 fait qu'une région est au dessus d'une autre. Selon les techniques de segmentation utilisées, la construction de ce graphe peut être plus ou moins complexe.

On regroupe généralement les algorithmes de segmentation en trois grandes classes (voir [14])[1] :

1. Segmentation basée sur les pixels
2. Segmentation basée sur les régions
3. Segmentation basée sur les contours

La première catégorie travaille sur des histogrammes de l'image. Par seuillage, clustering ou clustering flou, l'algorithme construit des classes de couleurs qui sont ensuite projetées sur l'image. La segmentation est implicite puisqu'on suppose que chaque cluster de l'histogramme correspond à une région dans l'image. En pratique, ce n'est pas forcément le cas et il faut séparer les régions de l'image qui sont disjointes bien qu'appartenant au même cluster de couleur. Ces algorithmes sont assez proches des algorithmes de réduction de couleur.

[1] On peut également considérer une quatrième classe qui serait constituées des algorithmes basés sur les régions+contours, voir par exemple [2, 15, 1]

La deuxième catégorie correspond aux algorithmes d'accroissement ou de découpage de région. L'accroissement de région est une méthode bottom-up : on part d'un ensemble de petites régions uniformes dans l'image (de la taille d'un ou de quelques pixels) et on regroupe les régions adjacentes de même couleur jusqu'à ce qu'aucun regroupement ne soit plus possible. Le découpage de région est le pendant top-down des méthodes d'accroissement : on part de l'image entière que l'on va subdiviser récursivement en plus petites régions tant que ces régions ne seront pas suffisamment homogènes. Les algorithmes dit "split and merge" sont un mélange de ces deux approches et nous en décrirons deux exemples : le "split and merge" classique et l'algorithme CSC.

La troisième catégorie s'intéresse aux contours des objets dans l'image. La plupart de ces algorithmes sont locaux, c'est à dire fonctionnent au niveau du pixel. Des filtres détecteurs de contours sont appliqués à l'image. Le résultat est en général difficile à exploiter sauf pour des images très contrastées. Les contours extraits sont la plupart du temps morcelés et peu précis et il faut utiliser des techniques de reconstruction de contours par interpolation ou connaître a priori la forme de l'objet recherché. Formellement, ce type d'algorithme est proche des techniques d'accroissement de région fonctionnant au niveau du pixel. Ces techniques purement locales sont en général trop limitées pour traiter des images bruitées et complexes.

Les algorithmes que nous allons présenter et commenter sont :

1. Accroissement de région fonctionnant au niveau du pixel.
2. Split and merge classique (d'après [9]).
3. Algorithme CSC (d'après [12]) : un algorithme de "merge and split" qui donne d'excellents résultats et qui possède un certain nombre de propriétés intéressantes.

2 Accroissement de région

Les méthodes d'accroissement de région sont les méthodes de segmentation les plus simples. Le principe est basé sur une approche bottom-up : l'algorithme part de petits éléments de l'image qu'il va tenter de regrouper en éléments plus importants. Nous présentons ici la version de base de l'algorithme d'accroissement de région qui fonctionne en agrégeant des pixels.

2.1 Principe de fonctionnement

Supposons une région de couleur homogène R. Initialement, $R = 1\ pixel$. On va étendre la région R en incluant les pixels situés sur la frontière et dont la couleur est proche de celle de R (la variation de couleur est inférieure à un seuil δ, caractéristique de ce type d'algorithmes). En répétant cette procédure jusqu'à ce qu'il n'y ai plus de pixels de couleur assez proche sur la frontière, on obtient une région de couleur homogène maximale autour du pixel de départ. La région initiale "gonfle" en absorbant des pixels de la frontière, jusqu'à stabilité par rapport à la propriété d'homogénéité[2].

Cette méthode présente deux limitations sévères qui n'en font pas une méthode très efficace :

1. Les régions obtenues dépendent fortement des pixels d'amorçage choisis et de l'ordre dans lequel les pixels de la frontière sont examinés.
2. Le résultat final est très sensible à la valeur du seuil δ.

Pour illustrer le premier problème, considérons trois pixels adjacents a, b et c dont les intensités respectives sont 8,10 et 11 (par exemple, l'intensité en niveaux de gris). Le seuil est 2. La région initiale est constituée du pixel b. Deux schémas de regroupement pour les points frontière a et c sont possibles (fig 1)

Le pixel central b est l'amorce. Compte tenu du seuil $\delta = 2$, a et c sur la frontière devraient être ajoutés à l'amorce b. Si l'on commence par tenter d'aggréger a, le résultat du regroupement, noté $[ab]$, a pour intensité moyenne 9 et c s'y ajoute ensuite puisque $|11 - 9| \leq \delta$. On a donc regroupé a, b et c. Si maintenant l'algorithme commence par examiner le point frontière c, le groupement de b et c donne $[bc]$ dont l'intensité est $10,5$. Le point frontière a d'intensité 8 est trop éloigné et il est considéré comme appartenant à une autre région. On obtient donc deux groupements au lieu d'un.

Ce petit exemple illustre combien l'ordre d'examen des points sur la frontière peu influencer sur le résultat de l'algorithme. A fortiori, cet algorithme est également très sensible au choix des amorces.

Par ailleurs, comme nous allons le voir, le résultat final dépend très sensiblement du seuil. Une petite variation de ce seuil peu conduire à des modifications importantes.

[2]Cette méthode est très générale est peu être appliquée avec n'importe quel critère permettant de décider si l'on doit ajouter ou non un point de la frontière à la région.

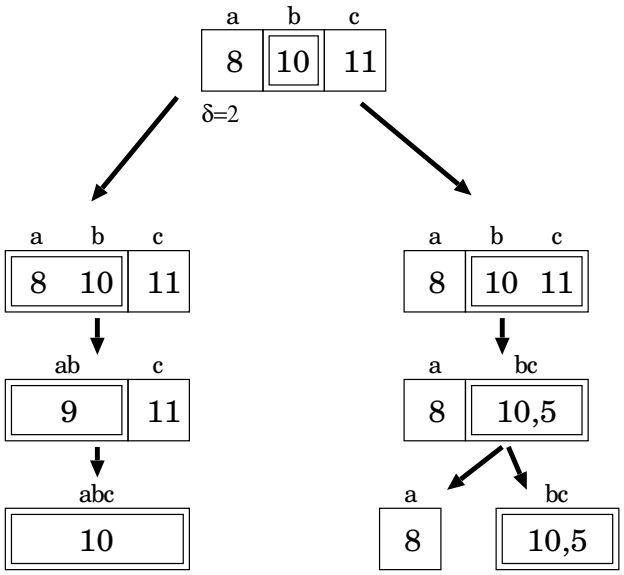


FIG. 1 – Deux schémas de regroupement pour des pixels a, b et c

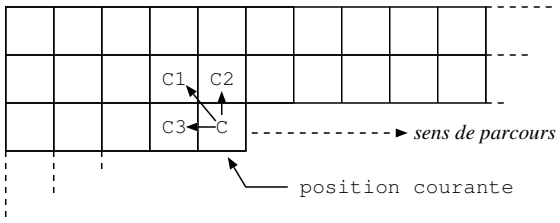
Cet algorithme fait parti de la classe d'algorithmes de segmentation dit "locaux". L'opération élémentaire consiste à manipuler des pixels adjacents et l'algorithme n'a aucune vision globale du résultat qu'il obtient. Par exemple, il est incapable de détecter que la région qu'il vient de construire est inhomogène[3], ce qui est souvent le cas.

2.2 Implémentation

Nous présentons ici une méthode classique pour implémenter cet algorithme. On associe à chaque pixel de l'image un index qui est un nombre entier. Le but de l'algorithme va être de donner à chaque pixel une valeur d'index qui corresponde à un numéro de région. Pour connaître ensuite l'étendu de la région n, il suffira d'extraire tous les pixels dont l'index vaut n.

L'index de chaque pixel est initialement fixé à -1, valeur indiquant que l'index n'a pas encore été attribué. On parcourt l'image de haut en bas et de gauche à droite. Plaçons nous en cours d'exécution : pour chaque pixel c examiné, on considère les pixels adjacents $c1, c2, c3$ qui ont déjà été examinés :

[3]L'algorithme présenté ici ne tient compte que de la valeur moyenne de la couleur. il est donc possible d'obtenir des régions avec un très fort écart type. Une amélioration possible de l'algorithme permet de tenir compte de l'écart type sur lequel est fixé un second seuil.



NB : Certaines implémentations de l'algorithme n'utilisent pas $c1$. Les régions construites peuvent alors présenter des coupures sur des points de connection articulés autour de deux pixels en diagonale.

On va ensuite choisir parmi $c1, c2, c3$ l'ensemble des pixels dont la couleur est identique à celle de c, à un facteur de tolérance δ près. Si cet ensemble est vide, on crée un nouvel index que l'on attribue au pixel courant c (il existe un compteur d'index qui permet de savoir quel est le prochain nouveau numéro d'index libre). Si cet ensemble n'est pas vide et que chaque pixel le constituant a le même index k, il suffit d'attribuer l'index k au pixel courant c. Dans le cas contraire, la situation est un peu plus complexe. Supposons que $c1$ et c ont la même couleur (à δ près) et l'index de $c1$ est 4. De même supposons que $c2$ et c ont la même couleur et que l'index de $c2$ est 7. Une possibilité est de reparcourir l'image déjà traitée et de changer tous les index de valeur 7 en index de valeur 4 et d'attribuer la valeur 4 à l'index de c. Cette solution n'est pas satisfaisante du point de vue de la complexité (algorithme en N^2, N étant la taille de l'image). Le plus efficace est de tenir à jour un chaînage qui indique les équivalences entre indexes. Par exemple ici, $4 \rightarrow 7$. Les chaînes peuvent ensuite s'allonger, par exemple : $4 \rightarrow 7 \rightarrow 11 \rightarrow 3 \rightarrow 19$. La technique algorithmique correspondante est l'utilisation d'un tableau `next`. Initialement `next[i]=i`. Quand on souhaite chaîner `p` et `q`, on procède de la façon suivante :

```
while (next[p] !=p)
   p = next[p];
next[p] = q;
```

Ce qui attache `q` à l'extrémité de la chaîne contenant `p`. Ceci établit l'équivalence entre `p` et `q`. Il suffit ensuite d'attribuer l'index `q` au pixel courant c.

La phase finale consiste à balayer à nouveau toute l'image une seule fois pour remplacer les numéros d'index par leur équivalent en extrémité de chaîne. Le résultat final est une image indexée, chaque index valant un numéro de région. La complexité globale est de l'ordre de N, la taille de l'image.

2.3 Exemples

Voici un exemple dont l'image test a été empruntée au site internet de L'Image Recognition Laboratory de l'université Koblenz-Landau (fig 2).

(`www.uni-koblenz.de/~lb/lb_images/segmentation.html`)

FIG. 2 – Image originale avant accroissement de région

Nous avons appliqué un algorithme d'accroissement de région avec un seuil de 6 puis de 7 sur le maximum de variation de RGB ($max(\Delta R, \Delta G, \Delta B) < \delta$). Les images segmentées sont présentées en fausses couleurs c'est à dire que chaque région est coloriée d'une manière aléatoire de sorte qu'on parvienne à la distinguer nettement des régions adjacentes (fig 3).

A gauche, l'image avec un seuil de 6. On constate que les fleurs sont correctement extraites mais la base en bas se confond avec le fond. Ce problème est caractéristique des algorithmes de segmentation dit "locaux" qui n'ont pas de vue d'ensemble ou de description de plus haut niveau que le pixel. Dans le cas de l'accroissement de région, entre deux points de deux régions connexes, il existe presque toujours une chaîne de pixels connexes pour laquelle l'écart de couleur entre deux pixels adjacents est inférieur au seuil. Cela se traduit par des effets de "ponts" entre zones, particulièrement sensibles dans les zones de dégradé. La conséquence est que les régions ont tendance à se mélanger au delà d'un certain seuil, et ceci même si les régions en question sont très différentes du point de vue de la couleur. Par exemple, il est facile de créer un pont de pixels entre un carré rouge et un carré bleu en utilisant un dégradé continu de rouge vers bleu. Les

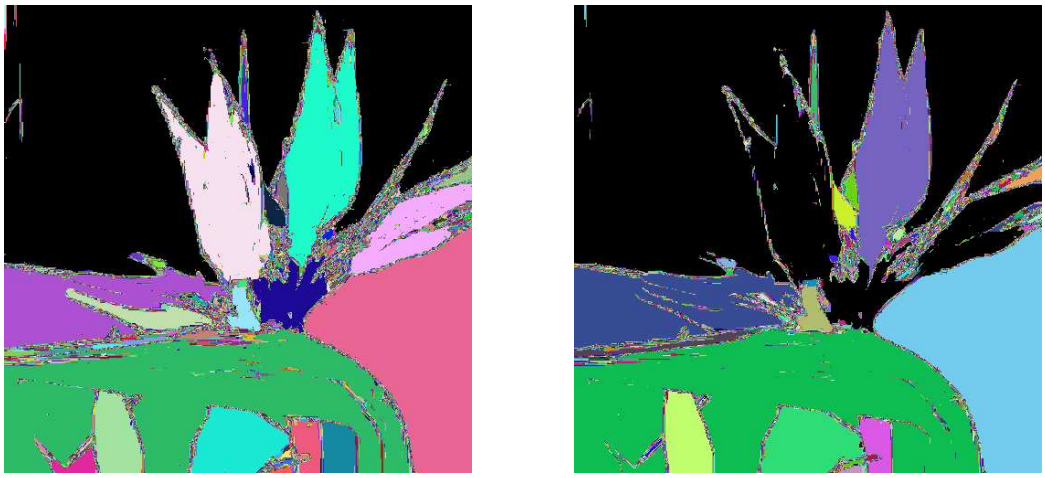


FIG. 3 – Accroissement de région : A gauche, seuil=6. A droite, seuil=7

algorithmes globaux comme le “split and merge” ou l’algorithme CSC que nous présenterons plus loin n’ont pas ce problème.

A droite, la même image est segmentée avec un seuil de 7. On constate que la fleur du milieu est confondue avec le fond (il existe visiblement un pont de pixels pour le seuil 7 entre cette région et le fond). Ceci met en lumière l’un des principaux problèmes liés à l’accroissement de région : la sensibilité au seuil. En pratique, il est presque toujours possible de trouver un seuil qui segmente bien une image donnée. Le problème vient que si l’on augmente d’une unité ce seuil, l’image est mal segmentée (fusion de régions, problème des “ponts”), si on diminue ce seuil d’une unité, l’image est sur-segmentée. Il n’existe pas de seuil qui fonctionne correctement pour toute une classe d’images, ce qui rend l’automatisation des algorithmes très difficile. A titre d’exemple, la figure 4 montre la segmentation de l’image précédente avec un seuil de 3 (à gauche) et de 4 (à droite) et un filtre pour supprimer les régions de taille inférieure à 100.

Le résultat est correct pour le seuil de 4 et devient beaucoup trop morcelé pour le seuil de 3. Le seuil optimal de cette image est donc 4. Pour une autre image, la bonne valeur de seuil sera en général différente et il faut recommencer ces estimations, ce qui rend l’algorithme peu utilisable dans un environnement bruité et complexe.

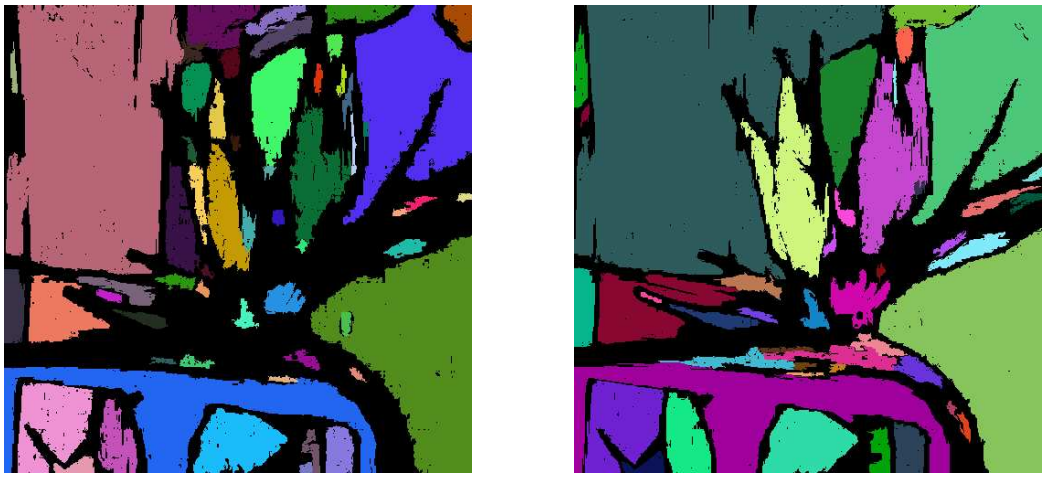


FIG. 4 – Accroissement de région : A gauche, seuil=3. A droite, seuil=4

3 Méthodes du type "split and merge"

L'algorithme "split and merge" a été présenté la première fois en 1974 par Pavlidis et Horowitz ([9]). Cet algorithme s'apparente dans son principe à l'algorithme d'accroissement de région que nous venons de présenter. La différence principale provient de la nature des régions élémentaires agrégées. Dans l'algorithme "split and merge", les régions agrégées proviennent d'une première phase (split) de traitement de l'image qui construit de manière récursive des régions carrées de taille variable mais homogènes.

3.1 Split

La méthode de découpage de l'image utilisée dans cet algorithme est basé sur la notion de "*quadtree*". Cette structure de données est un arbre quaternaire qui permet de stocker l'image à plusieurs niveaux de résolution. On part d'une région initiale qui est l'image tout entière. Si cette image vérifie un critère d'homogénéité de couleur, l'algorithme s'arrête. Sinon, on découpe cette région en quatre parties de même taille et on lance la procédure récursivement dans ces quatre parties. La région initiale va être stockée comme un noeud dans un graphe et les sous parties comme des fils de ce noeud. La figure 5 montre une image en noir et blanc 8x8 et le découpage correspondant à chaque niveau.

La structure d'arbre associée à ce découpage est illustrée figure 6.

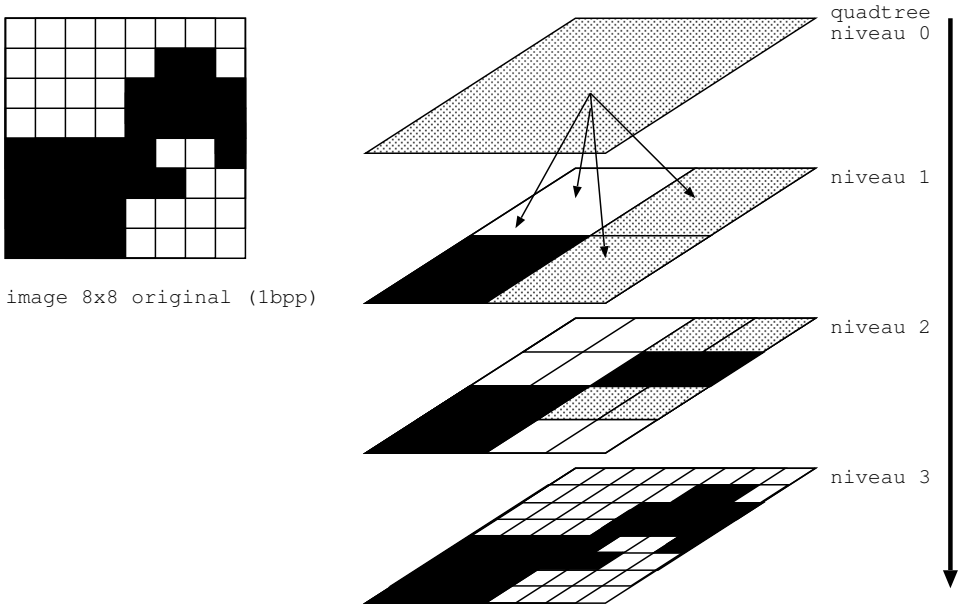


FIG. 5 – Découpage par quadtree d'une image 8x8. Traitement avec seuil=100%

Dans cet exemple, le critère d'homogénéité est absolu : une zone est dite homogène si elle ne contient que des pixels de même couleur (seuil=100%). On peut être plus tolérant et accepter qu'une zone soit déclarée homogène dès que plus de 75% d'une couleur domine. La figure 7 montre le découpage correspondant.

La structure d'arbre associée à ce découpage est plus simple (fig 8).

De manière plus générale, on va appliquer ce principe de réduction à des images colorées. Le critère d'homogénéité est fixé par un seuil sur la variance de la couleur dans la zone en cours d'examen. Au dessus de ce seuil, la zone est découpée en quatre, en dessous, elle est conservée et constitue un noeud terminal de l'arbre. On lui attribue alors la couleur de la moyenne des pixels la constituant.

3.2 Merge

La procédure de découpage décrite précédemment aboutit à un nombre de régions trop élevé. La cause fondamentale de cette sur-segmentation est que l'algorithme découpe les régions de manière arbitraire. Il se peut qu'il coupe de cette façon une zone homogène en deux ou quatre parties (fig 9).

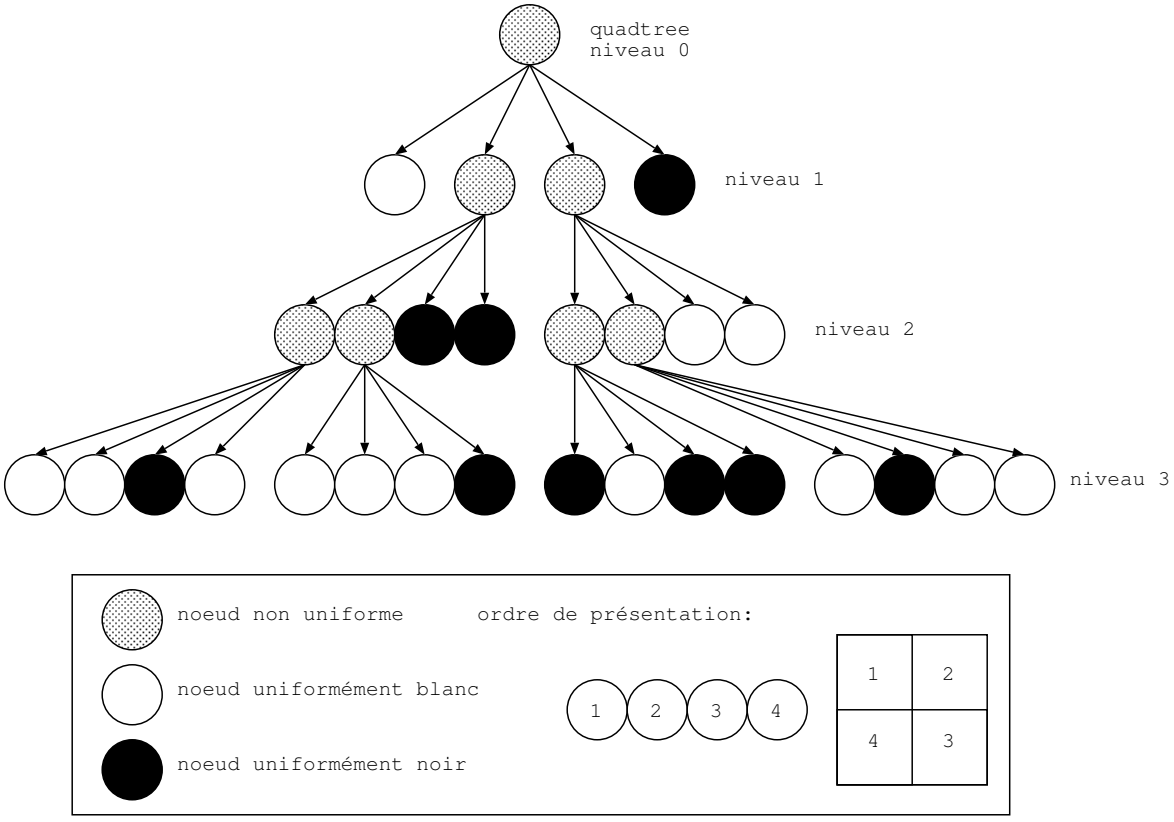


FIG. 6 – Quadtree sur une image 8x8, seuil=100%

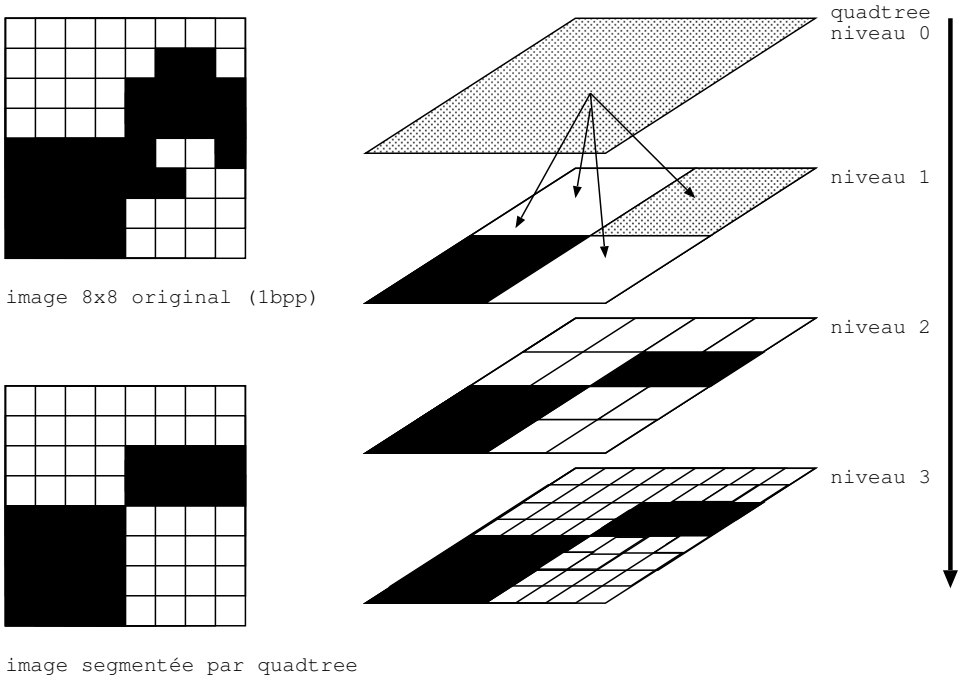


FIG. 7 – Découpage par quadtree d'une image 8x8. Traitement avec seuil=75%

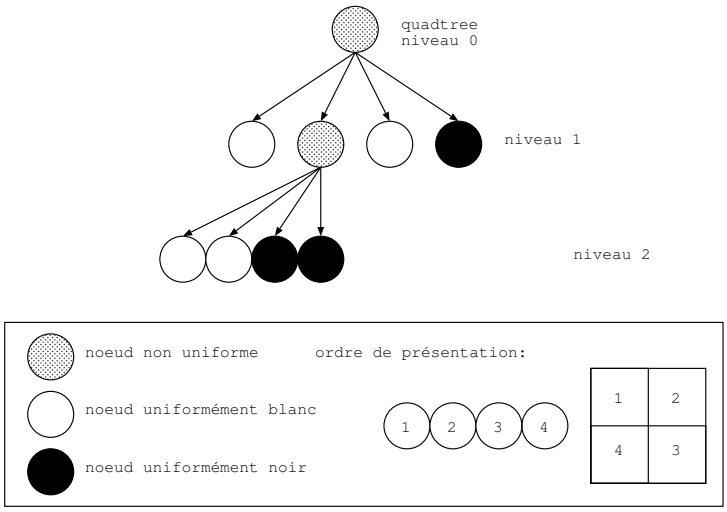


FIG. 8 – Quadtree sur une image 8x8, seuil=75%

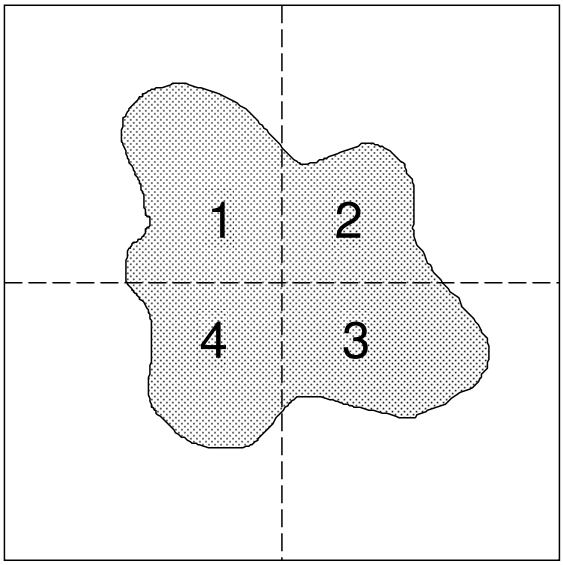


FIG. 9 – Problème de découpage arbitraire de régions dans la phase de “split”

Les parties 1,2,3 et 4 de cette image par exemple, appartiennent à des branches différentes du quadtree. Elles sont donc considérées comme des régions différentes bien que leur couleur soit identique.

La solution, qui correspond à la phase “merge” de l’algorithme, est de procéder à une fusion de régions après le découpage. L’implémentation la plus simple de cette fusion cherche tous les couples de régions adjacentes dans l’arbre issu du découpage et cherche à les fusionner si leur couleur est assez proche.

Pour réaliser cette fusion, il faut d’abord tenir à jour une liste des contacts entre régions (chaque région dispose d’un pointeur sur une liste chaînée de régions avec lesquelles elle est en contact). On obtient ainsi un graphe d’adjacence de régions ou *Region Adjacency Graph* (NB : 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 sont 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 son ajoutés à la pile des régions à comparer avec R. La région fusionnée est marquée “traitée”. Une fois la pile vide, l’algorithme choisi la prochaine région marquée “non traitée” et recommence, jusqu’à ce que toutes les régions

soient traitées[4].

Des améliorations substantielles à cet algorithme existent (entre autres, [5, 4, 3, 6]) et nous n'avons présenté ici que la version la plus élémentaire du "split and merge".

3.3 Exemple

Voici le résultat du traitement de l'image précédente (image originale fig 10) par l'algorithme de "split and merge" . Nous donnons la segmentation en fausses couleurs sur la gauche et la segmentation en vraies couleurs à droite (fig 11).

FIG. 10 – Image originale avant "split and merge"

Ces images sont empruntées au site web de l'université Koblenz-Landau. Le seuil utilisé n'est pas précisé mais on constate que le nombre de régions extraites est trop élevé. Il y a une multitude de petites régions isolées. On peut améliorer ce résultat en augmentant le seuil mais quoi qu'il en soit, le résultat final présentera toujours une forme morcelée en escaliers, caractéristique du pré-traitement en quadtree. On touche ici au principal désavantage de la méthode "split and merge" qui a beaucoup de difficultés à restituer des régions aux contours naturels. Les régions ont toujours cette apparence "carrée". De plus, cette méthode souffre du problème général des algorithmes d'accroissement de région (dont la phase "merge" est un cas particulier) : le résultat final dépend de l'ordre dans lequel sont

[4]Il existe de nombreuses variantes plus subtiles de la phase de regroupement de cet algorithme. Nous ne présentons ici que l'idée générale.

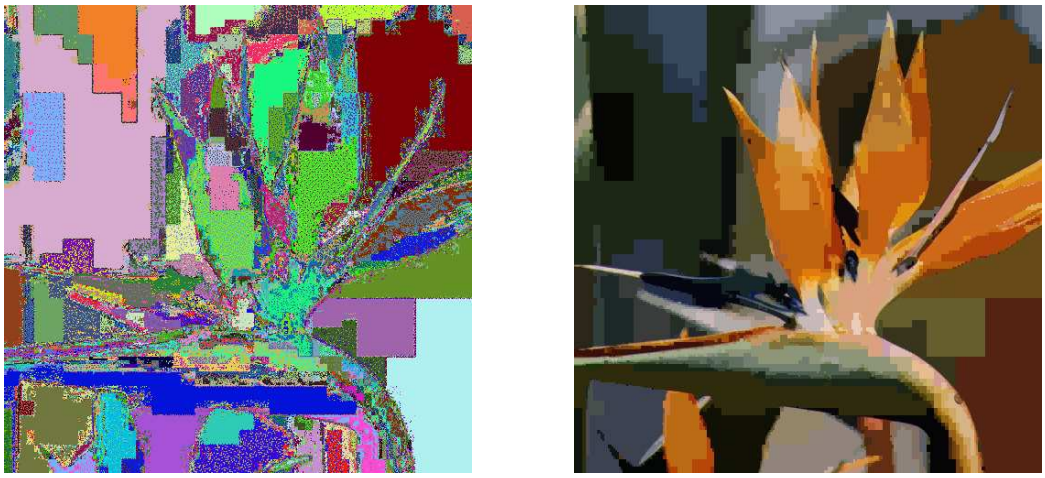


FIG. 11 – Split and merge : A gauche, la segmentation en fausses couleurs. A droite l'image avec couleurs réelles.

examinées les régions adjacentes lors du regroupement. Il s'en suit que la segmentation obtenue est peu stable d'une image sur l'autre et le processus manque de robustesse en général.

L'algorithme CSC que nous présentons à présent combine beaucoup des avantages des méthodes précédentes mais ne présente pas les limitations que nous avons soulignées.

4 Algorithme CSC

L'algorithme CSC (Color Structure Code) a été présenté en 1993 par Rehrmann et Priese [12]. Il repose sur l'utilisation d'une structure hexagonale hiérarchique de codage de l'image. Il s'agit d'un algorithme de "merge and split" particulièrement efficace et robuste. L'une des propriétés de cet algorithme est qu'il est complètement parallélisable et qu'il ne dépend pas d'un ordre de traitement de l'image comme les algorithmes d'accroissement de région.

4.1 Hiérarchie hexagonale

L'algorithme repose sur l'utilisation d'une topologie hiérarchique particulière des pixels. La topologie hiérarchique est constituée d'îlots hexagonaux disposés

à différents niveaux. Les îlots de niveau 0 sont des groupements hexagonaux de sept pixels (fig 12).

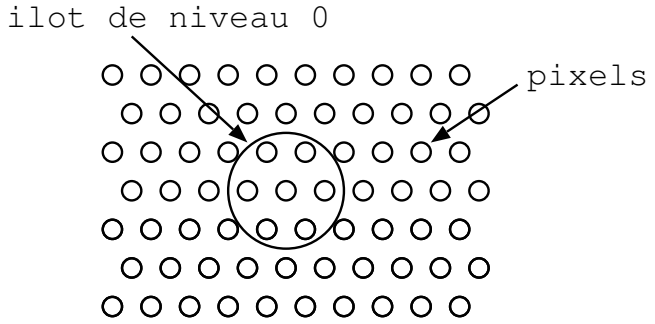


FIG. 12 – ALgorithme CSC : Exemple d'îlot de niveau 0 basé sur la topologie hexagonale

La disposition légèrement décalée des pixels de rang pair et de rang impair peut être soit interpolée soit simulée par l'intermédiaire d'une topologie légèrement différente plaquée sur le réseau orthogonal de l'image (fig 13).

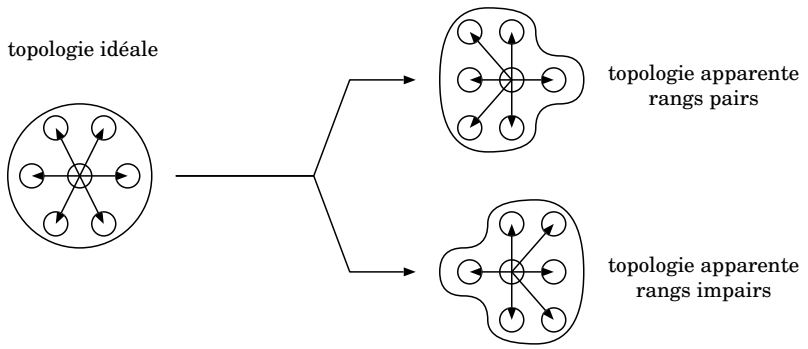


FIG. 13 – Algorithme CSC : Relations de contact d'un réseau orthogonal équivalent à une topologie hexagonale

Cette topologie est récursive, c'est à dire qu'un îlot de niveau $n+1$ est constitué de sept îlots de niveau n, disposés hexagonalement de la même manière que les pixels constituant un îlot de niveau 0 (fig 14). On remarquera que les îlots d'un niveau donné se recouvrent partiellement. Cette propriété est particulièrement importante pour l'algorithme puisqu'elle permet des simplifications et des optimisations significatives.

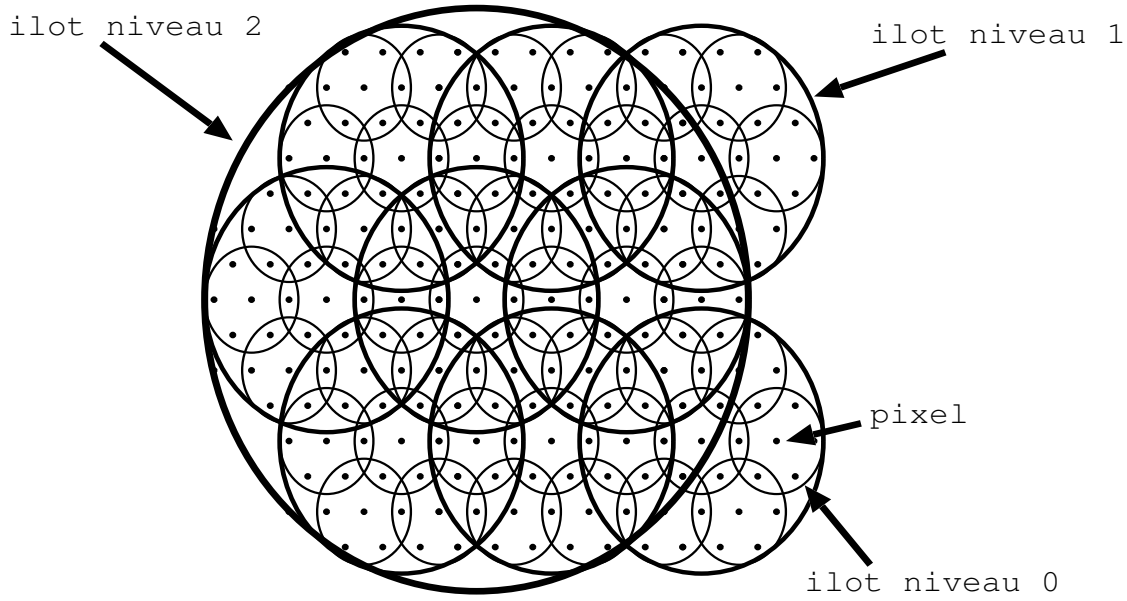


FIG. 14 – Algorithme CSC : Structure hexagonale hiérarchique

La construction récursive des îlots s'arrête lorsqu'un îlot recouvre toute l'image. Le nombre d'îlots décroît d'un facteur 4 d'un niveau à l'autre.

4.2 Initialisation

La première phase de l'algorithme consiste à appliquer un simple algorithme d'accroissement de région localement dans chaque îlot de niveau 0. Cette première étape construit entre 1 et 7 régions de niveau 0, contenues dans chaque îlot. Nous appellerons ces régions des *éléments* (fig 15). L'étape de construction de ces éléments de niveau 0 peut s'exécuter en parallèle sur plusieurs processeurs puisque le traitement effectué dans un îlot ignore complètement celui de l'îlot voisin, bien qu'ils partagent un pixel un commun.

La méthode utilisée pour comparer la couleur des pixels est plus complexe qu'une simple distance euclidienne entre les composantes rouges, vertes et bleues. Nous présenterons cette technique un peu plus loin.

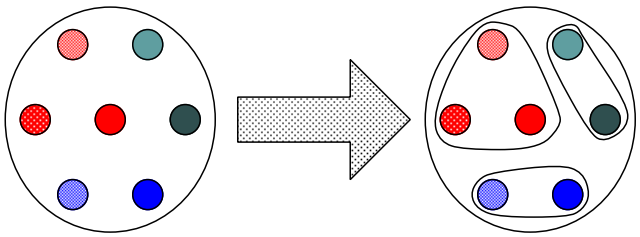


FIG. 15 – Algorithme CSC : Phase d'initialisation

4.3 Phase de groupement

Dans la phase de regroupement, des éléments de niveau n sont regroupés en éléments de niveau $n+1$. Cette opération s'effectue à l'intérieur d'un îlot de niveau $n+1$. Les éléments de niveau n sont regroupés si ils sont connectés et de couleur proche. Le test de connectivité est particulièrement efficace et utilise le résultat suivant :

> **Propriété :** Deux éléments de niveau n sont connectés s'ils ont au moins un élément de niveau $n-1$ en commun.

Au niveau 0, cette propriété signifie simplement que les deux éléments ont un pixel en commun. On notera que deux éléments de niveau n connectés appartiennent forcément à deux îlots adjacents de niveau n et leurs sous-éléments communs se trouvent dans l'îlot de niveau $n-1$ commun.

L'opération est répétée de niveau en niveau jusqu'au niveau recouvrant toute l'image. Le résultat est un arbre et le groupement de k éléments de niveau n en un élément de niveau $n+1$ se fait uniquement par l'utilisation de pointeurs. Lorsqu'un élément de niveau n ne trouve aucun autre élément auquel se lier, il devient la racine d'un arbre correspondant à une région terminée de l'image et peut être stocké. L'exemple suivant illustre le processus de regroupement. On considère les sept îlots de niveau n constituant l'îlot de niveau $n+1$ en cours de traitement (fig 16).

Chaque îlot de niveau n contient un ensemble d'éléments de niveau n, issus du traitement récursif à l'étape précédente (fig 17). La figure 18 montre les éléments de niveau n dans une disposition "éclatée", tels qu'ils apparaissent localement pour chaque îlot de niveau n.

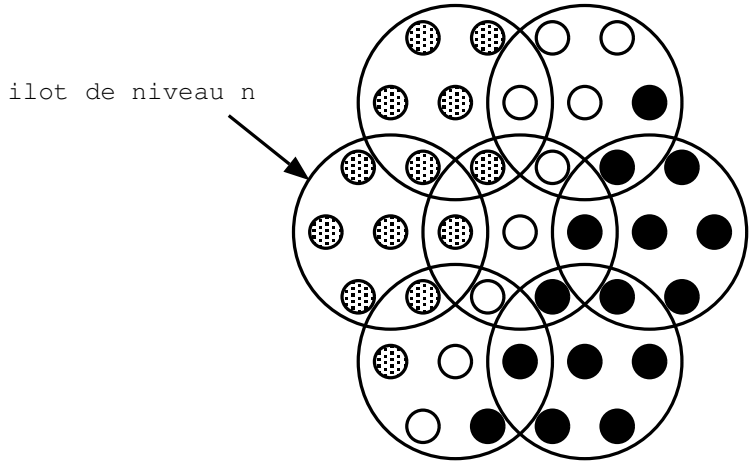


FIG. 16 – Example d'îlots de niveau n regroupés dans un îlot de niveau $n+1$

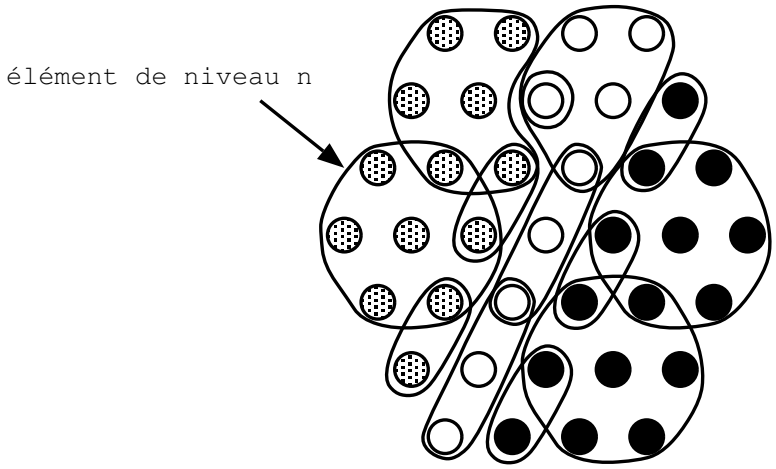


FIG. 17 – Example d'éléments de niveau n

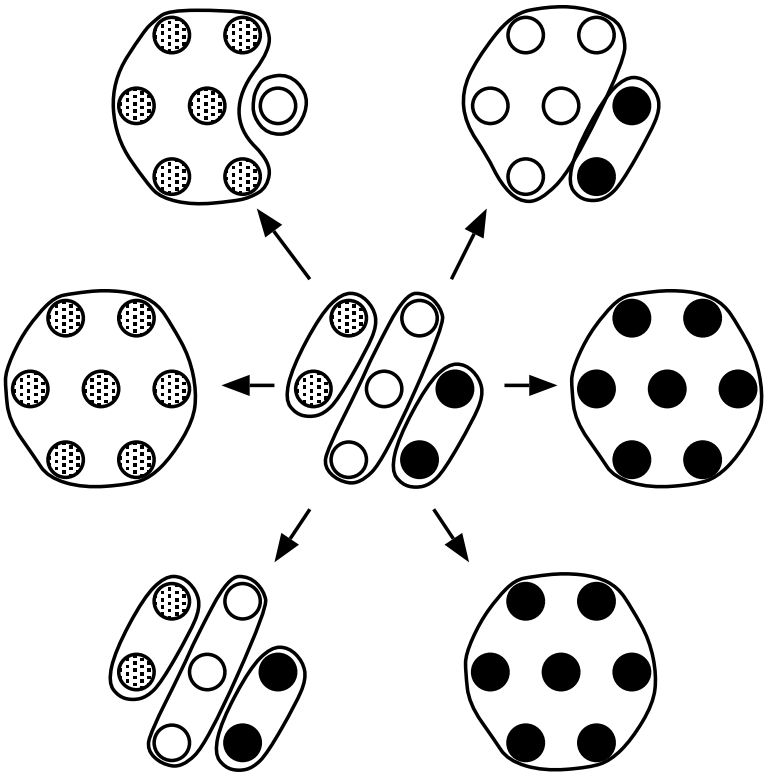


FIG. 18 – Example d'éléments de niveau n (disposition éclatée)

Les éléments de niveau n sont ensuite considérés par l'îlot de niveau $n+1$ auquel ils appartiennent (fig 19).

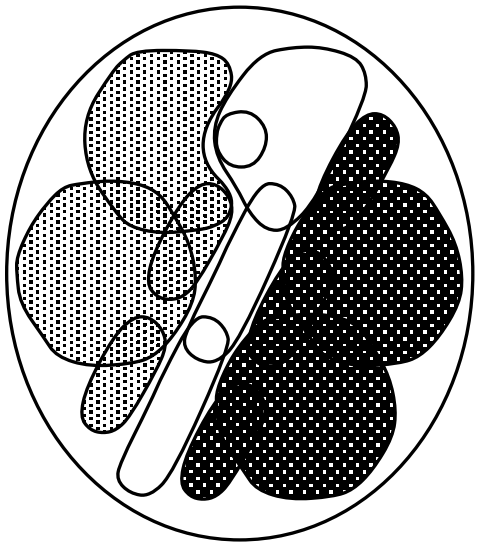


FIG. 19 – Example d'éléments de niveau n vus au niveau $n+1$

Finalement, les éléments de niveau n sont regroupés au sein d'éléments de niveau $n+1$ (fig 20).

La figure 21 illustre la structure d'arbre résultant du regroupement.

4.4 Phase de découpage

L'erreur caractéristique des algorithmes d'accroissement de région est de connecter deux régions de couleurs différentes parce qu'il existe un "pont" ou chaîne de pixels de la première région vers la deuxième (cf 2.3). Le long de cette chaîne, chaque pixel a une couleur proche de celle de son voisin, ce qui fait que l'algorithme ne "voit" pas la dérive qui mène d'une couleur à l'autre. Ces algorithmes utilisent uniquement de l'information locale et ne sont pas capables de détecter des contours au contraste trop faible. L'algorithme CSC utilise à la fois des informations locales (phase de regroupement) et globales (phase de découpage).

La phase de découpage a lieu simultanément à la phase de regroupement. Lorsque deux éléments de niveau n sont connectés et de couleur similaire, ils sont regroupés dans un élément de niveau $n+1$. Dans le cas où ils sont connectés

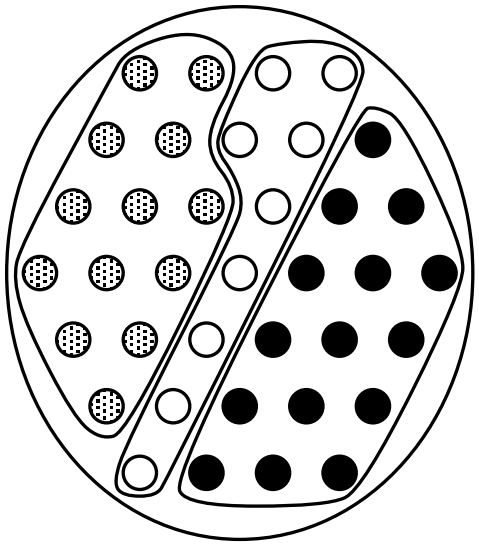


FIG. 20 – Example d'éléments de niveau $n+1$

mais de couleur incompatible ils ne vont pas être regroupés, bien que tous les éléments de niveau $n-1$ les constituant soient de couleurs homogènes et proches. L'analyse au niveau n est une analyse globale des propriétés colorimétriques des éléments de niveau $n-1$. Grâce à ce traitement, le problème des chaînes de pixels est supprimé.

Lorsque deux éléments connectés ne sont pas regroupés parce que leurs couleurs sont trop éloignées, ils partagent cependant un ensemble de sous éléments puisqu'ils sont connectés (ceci est dû à la structure partiellement recouvrante des îlots). Il va donc falloir les séparer pour obtenir deux éléments sans recouvrement et partager la région commune. Cette opération peut être réalisée de manière économique et élégante grâce à la structure hexagonale hiérarchique qui a été mise en place.

Considérons le cas général de la figure 22.

C_1 et C_2 sont deux éléments de niveau $n+1$. Ils sont constitués d'un ensemble d'éléments de niveau n. Les îlots de niveau n contenant ces éléments sont marqués d'un cercle (voir le haut de la figure 22). Puisque C_1 et C_2 sont connectés, ils partagent (au moins) un élément de niveau n, que nous noterons s. La couleur de s est alors comparée à celle de C_1 et de C_2 et s est alors lié à l'élément (C_1 ou C_2) dont la couleur est la plus proche. Supposons que ce soit C_2, s doit donc être effacé de la liste des fils de C_1 (il s'agit d'une simple suppression de pointeur).

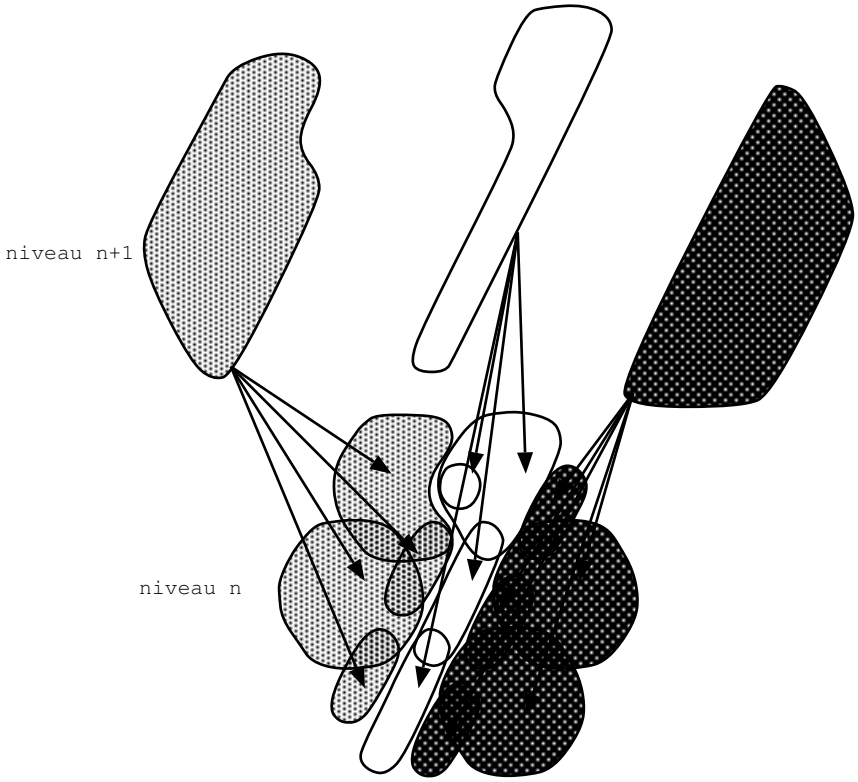


FIG. 21 – Algorithme CSC : Structure arborescente résultant de la phase de groupement

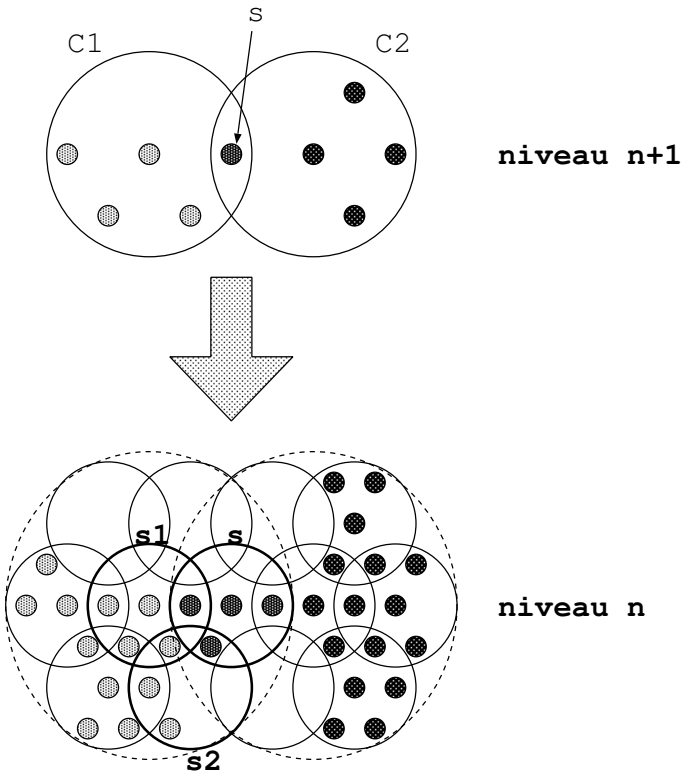


FIG. 22 – Résultat de la phase de découpage récursif de l'algorithme CSC

s n'est pas pour autant intégralement attribué à C_2. Bien qu'il ait été supprimé de C_1, il reste encore attaché à cet élément par les connections qui l'attachent avec les éléments $s1$ et $s2$ de niveau n dans C_1 (c'est une conséquence directe de la structure en recouvrement). Il s'agit donc de séparer s de $s1$ et $s2$, ce qui est réalisé de manière récursive en reprenant la même procédure que celle que nous venons de décrire pour la séparation de C_1 et C_2. La seule différence est que les couleurs utilisées pour déterminer de quel côté bascule la région commune sont les couleurs de C_1 et de C_2, qui représentent l'information la plus globale.

NB : Lors de la subdivision récursive de s, il est tout à fait possible que la région commune à s et à $s1$ soit assignée à $s1$. Ce qui indique que, bien que s ait été attribuée à C_2, cela ne signifie pas que tous les pixels constituant s seront finalement attribués à C_2. s peut perdre des sous régions dans la descente récursive.

Cette procédure simple et élégante (il s'agit de simples délétions de pointeurs) conduit à des frontières très précises entre régions.

Remarque : L'algorithme présenté peut poser problème dans certains cas très rares où la phase de découpage peut conduire à des éléments non connexes, c'est à dire à des régions disjointes. Les méthodes possibles pour prendre en compte ce problème sont complexes et coûteuses. En pratique, la situation ne se présentant presque jamais, il est tout à fait acceptable d'ignorer le problème.

4.5 Taille et couleur moyenne des éléments

Une question délicate que nous n'avons pas abordé est celle du calcul de la couleur moyenne d'un élément. Compte tenu de la structure en recouvrement de la hiérarchie hexagonale, certains pixels communs à deux éléments regroupés seront comptés en double si l'on effectue une simple moyenne. Pour simplifier la question, on peut ramener le calcul de la couleur moyenne à celui du nombre de pixels effectivement couverts par un élément donné. Lorsqu'on regroupe deux éléments (qui partagent donc un sous élément commun), le plus simple semble être de sommer le nombre de pixels de chaque élément et de soustraire le nombre de pixels du sous élément commun compté en double. On peut reprendre les notations de la figure 22, en supposant à présent que l'on cherche à regrouper C_1 et C_2. Le problème lorsque l'on fait le calcul $\Sigma C_1 + \Sigma C_2 - \Sigma s$ est que l'on soustrait abusivement des sous éléments communs à s et $s1$ d'une part et s et $s2$ d'autre part. Il faut donc ajouter ces sous éléments et poursuivre récursivement le calcul

vers les sous-sous-éléments de $s1$ et $s2$, en alternant somme et soustraction. Cette opération est similaire en terme de complexité à l'opération de découpage que nous venons de présenter. Le coût en temps est considérable car l'opération de regroupement est très fréquente (le découpage est un événement rare, comme nous le verrons plus loin, ce qui rend son coût global négligeable). Le calcul exact de la taille des éléments est donc trop lourd et il est important d'utiliser une approximation.

Considérons la situation de la figure 23.

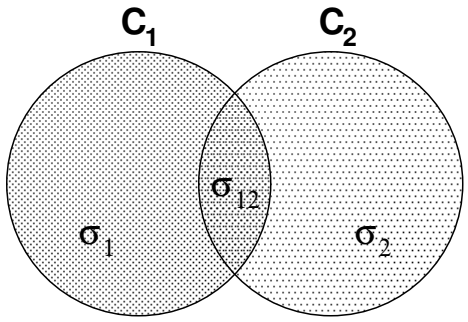


FIG. 23 – Algorithme CSC : problème du mélange des couleurs C_1 et C_2

Deux régions R_1 et R_2 de taille σ_1 et σ_2 et de couleur moyenne C_1 et C_2 doivent être regroupées. La zone commune est de taille σ_{12}. Comme on ne dispose pas d'informations autres que les informations statistiques concernant la couleur moyenne C_1 et C_2 des deux zones, on ne peut évaluer la couleur moyenne de l'intersection au mieux que comme étant égale à $(C_1 + C_2)/2$. Étant donné que les deux couleurs sont proches puisque les deux régions sont amenées à être regroupées, l'erreur commise est faible. On pose a priori que :

$$\max(\frac{|C_2 - C_1|}{C_2}, \frac{|C_2 - C_1|}{C_1}) = \frac{|C_2 - C_1|}{\min(C_1, C_2)} \leqslant \delta$$

δ est donc le pourcentage de variation de couleur maximum autorisé. L'erreur commise en prenant $(C_1 + C_2)/2$ comme couleur pour l'intersection est donc au plus de $1/2 \times \delta \min(C_1, C_2)$. Dans la suite, nous noterons $C = \min(C_1, C_2)$.

Nous allons montrer qu'une bonne approximation consiste à ignorer la région commune et à considérer R_1 et R_2 comme disjointes, c'est à dire (en notant $H(R)$ la couleur d'une région) :

$$H(R_1 \cup R_2) = \frac{\sigma_1 C_1 + \sigma_2 C_2}{\sigma_1 + \sigma_2} = H_{approx}$$

Une propriété importante de l'algorithme CSC est que la variance sur la taille des éléments d'un niveau donné est assez faible. Cela signifie que les tailles de R_1 et de R_2 sont proches. Cette propriété est facilement vérifiable expérimentalement et nous avons constaté sa validité en pratique. Pour la suite, nous poserons donc $\sigma_2 = \sigma_1(1+\varepsilon)$. Nous poserons également que $\sigma_{12} \approx \alpha\sigma_2$ et, symétriquement $\sigma_{12} \approx \alpha\sigma_1$, ce qui mène à $\sigma_{12} \approx \alpha\frac{\sigma_1+\sigma_2}{2}$.En pratique, α est de l'ordre de $1/7$ et représente la proportion de surface recouvrante de deux îlots adjacents dans la topologie. Dans ce cas, le calcul exact du mélange des couleurs donne :

$$\begin{aligned}
H(R_1 \cup R_2) &= \frac{(\sigma_1 - \sigma_{12})C_1 + (\sigma_2 - \sigma_{12})C_2 + 1/2 \cdot \sigma_{12}(C_1 + C_2)}{\sigma_1 + \sigma_2 - \sigma_{12}} \\
&= \frac{\sigma_1 C_1(1-\alpha) + \sigma_2 C_2(1-\alpha) + 1/4 \cdot \alpha(\sigma_1+\sigma_2)(C_1+C_2)}{(\sigma_1+\sigma_2)(1-\alpha/2)} \\
&= \frac{(\sigma_1 C_1 + \sigma_2 C_2)(1-\alpha)}{(\sigma_1+\sigma_2)(1-\alpha/2)} \\
&\quad + \frac{1/4 \cdot \alpha(\sigma_1 C_1 + \sigma_2 C_2 + \sigma_1 C_2 + \sigma_2 C_1)}{(\sigma_1+\sigma_2)(1-\alpha/2)} \\
&= \frac{(\sigma_1 C_1 + \sigma_2 C_2)(1-\alpha)}{(\sigma_1+\sigma_2)(1-\alpha/2)} \\
&\quad + \frac{1/4 \cdot \alpha\left[2(\sigma_1 C_1 + \sigma_2 C_2) - (\sigma_1 C_1 + \sigma_2 C_2) + \sigma_1 C_2 + \sigma_2 C_1\right]}{(\sigma_1+\sigma_2)(1-\alpha/2)} \\
&= \frac{(\sigma_1 C_1 + \sigma_2 C_2)(1-\alpha+\alpha/2)}{(\sigma_1+\sigma_2)(1-\alpha/2)} \\
&\quad + \frac{1/4 \cdot \alpha\left[(\sigma_1 C_2 + \sigma_2 C_1) - (\sigma_1 C_1 + \sigma_2 C_2)\right]}{(\sigma_1+\sigma_2)(1-\alpha/2)} \\
&= H_{approx} + \alpha\frac{(\sigma_1 C_2 + \sigma_2 C_1) - (\sigma_1 C_1 + \sigma_2 C_2)}{4(\sigma_1+\sigma_2)(1-\alpha/2)} \\
&= H_{approx} + \alpha\frac{\varepsilon(C_1 - C_2)}{4(1-\alpha/2)(2+\varepsilon)}
\end{aligned}$$

Ce qui peut encore s'écrire après quelques manipulations :

$$\begin{aligned}
H(R_1 \cup R_2) &= H_{approx}\left(1 + \frac{\alpha\varepsilon(C_1 - C_2)}{4(1-\alpha/2)(C_1 + C_2 + \varepsilon C_2)}\right) \\
&= H_{approx}(1+\eta)
\end{aligned}$$

Avec :

$$\begin{aligned} |\eta| &\leqslant \frac{\alpha\varepsilon|C_1 - C_2|}{4(1-\alpha/2)(C_1 + C_2 + \varepsilon C_2)} \\ &\leqslant \frac{\alpha\varepsilon|C_1 - C_2|}{4(1-\alpha/2)(\min(C_1, C_2) + \min(C_1, C_2) + \varepsilon \min(C_1, C_2))} \\ &\leqslant \frac{\alpha\varepsilon|C_1 - C_2|}{4(1-\alpha/2)(1+\varepsilon)C} \\ &\leqslant \frac{\alpha\varepsilon\delta}{4(1-\alpha/2)(1+\varepsilon)} \end{aligned}$$

L'erreur relative commise est donc très faible. Avec $\alpha = 1/7$, $\delta = 20\%$ et $\varepsilon = 4$ (valeur très pessimiste puisque, en pratique on observe plutôt $\varepsilon \approx 0$), on obtient :

$$|\eta| \leqslant 6,15 \cdot 10^{-3}$$

Ce qui valide complètement l'approximation réalisée. On utilise donc les formules approchées suivantes :

$$\sigma(R_1 \cup R_2) = (1 - \frac{\alpha}{2}) \cdot (\sigma(R_1) + \sigma(R_2)) \tag{1}$$

$$H(R_1 \cup R_2) = \frac{\sigma(R_1)C(R_1) + \sigma(R_2)C(R_2)}{\sigma(R_1) + \sigma(R_2)} \tag{2}$$

4.6 Pré-traitement : filtrage SNN

Pour améliorer les performances de l'algorithme, il est intéressant d'utiliser un filtre réducteur de bruit en entrée. Plusieurs méthodes classiques ont été essayées : filtre gaussien, filtre médian, filtre $k$nn (k-Nearest-Neighbor). Le plus efficace dans le cadre de l'algorithme CSC est le filtre *snn* (Symmetric Nearest Neighbor, [11]), comme le prouvent les tests de reconnaissance de forme effectués par Rehrmann ([13]). C'est également le plus rapide puisqu'il implique le plus faible de nombre de calculs.

Le fonctionnement du filtre *snn* est simple. Chaque pixel est examiné dans le cadre de la topologie hexagonale mise en place pour l'algorithme CSC (voir figure 24).

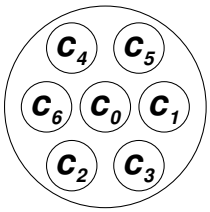


FIG. 24 – Disposition des pixels dans le réseau hexagonal

On considère les trois paires de pixels symétriques autour de c_0 : (c_1, c_6),(c_2, c_5) et (c_3, c_4). Dans chaque paire, on va choisir le pixel dont la couleur est la plus proche de celle de c_0. On obtient trois pixels $c_{i_1}, c_{i_2}, c_{i_3}$.La couleur de c_0 est alors remplacée par la moyenne des couleurs de ces trois pixels.

Ce filtre a l'avantage de renforcer les contours de l'image (pas de flou) et supprime le bruit de manière efficace.

4.7 Similarité de couleurs

Un des aspects importants de l'algorithme CSC est la mesure de similarité de couleurs utilisée. La plupart des méthodes classiques utilisent une distance euclidienne sur les composantes de couleurs dans un espace comme RGB. Le désavantage de ces méthodes est qu'elles sont en général assez éloignées des mesures de similarité de l'oeil humain. La segmentation qui en résulte est de moins bonne qualité.

La méthode proposée ici repose sur l'utilisation d'un prédicat de couleur D. Étant données deux couleurs c_1 et c_2, on pose :

$$D(c_1, c_2) = \begin{cases} vrai & : c_1 \text{et } c_2 \text{ sont de couleur similaire} \\ faux & : \text{sinon} \end{cases}$$

L'espace de couleur utilisé ici est HSV (Hue, Saturation, Value). L'avantage de cet espace est qu'il permet de séparer clairement l'information colorimétrique de l'information de luminance ou de saturation. En particulier,

$$\begin{aligned} hue(r, g, b) &= hue(\alpha r, \alpha g, \alpha b) \\ &= hue(r + \beta, g + \beta, b + \beta) \end{aligned}$$

Idéalement donc, pour comparer des couleurs en ignorant les variations dues à l'illumination (V) ou à la saturation (S), il suffit de mesurer l'écart de valeur de H. Le problème de la mesure de couleur dans l'espace HSV est lié à l'existence d'une singularité en $(0, 0, 0)$. Lorsque S ou V deviennent trop faibles, la valeur de H est extrêmement instable (à la limite, quand S=0 ou V=0, H n'est pas défini). Dans ce cas, une mesure de distance euclidienne sur S et V devient bien plus efficace pour comparer les couleurs. La mesure D proposée ici tient compte de ces domaines de validité en utilisant des seuils variables pour les tolérances d'écart sur H, S et V. La valeur des trois seuils associés à ces trois composantes est fonction de S et de V. Ainsi, le seuil de H va devenir très tolérant quand S ou V diminue, ce qui revient à ignorer H. Plus précisément :

$$D(h_1, s_1, v_1, h_2, s_2, v_2) = vrai \quad \Leftrightarrow \quad \left\{ \begin{array}{l} h_1 \ominus h_2 \leqslant Hseuil \\ |s_1 - s_2| \leqslant Sseuil \\ |v_1 - v_2| \leqslant Vseuil \end{array} \right.$$

avec

$$\begin{array}{rcl} Hseuil & = & Htab[\min(\overline{s_1}, \overline{s_2})][\max(\overline{v_1}, \overline{v_2})] \\ Sseuil & = & Stab[\min(\overline{s_1}, \overline{s_2})][\max(\overline{v_1}, \overline{v_2})] \\ Vseuil & = & Vtab[\min(\overline{s_1}, \overline{s_2})][\max(\overline{v_1}, \overline{v_2})] \end{array}$$

L'opérateur $\ominus$ représente la différence modulo 2π, c'est à dire :

$$h_1 \ominus h_2 = \left\{ \begin{array}{rcl} |h_1 - h_2| & : & si\ |h_1 - h_2| \leqslant \pi \\ 2\pi - |h_1 - h_2| & : & sinon \end{array} \right.$$

Les tableaux $Hseuil$, $Sseuil$ et $Vseuil$ sont des tableaux 16x16. Les barres au dessus des composantes ($\overline{s_1}$,$\overline{v_2}$,...) signifient que l'on prend la valeur quantifiée de la composante entre 0 et 15. Pour une composante k dont la valeur varie entre 0 et 255, on a $\overline{k} = k\ div\ 16$.

Les tableaux $Hseuil$, $Sseuil$ et $Vseuil$ ont été déterminés expérimentalement. Nous en donnons ici une table est une représentation graphique (fig 25).

$Hseuil$

S \ V	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	360	360	360	360	360	360	360	360	360	360	360	360	360	360	360	360
1	360	360	360	360	360	360	360	360	360	360	360	360	360	360	360	360
2	360	360	360	125	125	125	125	125	125	125	125	125	125	125	125	125
3	360	360	100	35	10	10	10	10	10	10	10	10	10	10	10	10
4	360	360	35	10	10	10	10	10	10	10	10	10	10	10	10	10
5	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10
6	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10
7	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10
8	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10
9	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10
11	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10
12	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10
13	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10
14	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10
15	360	360	10	10	10	10	10	10	10	10	10	10	10	10	10	10

$Sseuil$

S \ V	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	56	40	32	24	24	24	24	24	24	24	24	24	24	24	24	24
1	56	40	32	24	24	24	24	24	24	24	24	24	24	24	24	24
2	56	40	32	24	24	24	24	24	24	24	24	24	24	24	24	24
3	56	49	41	34	34	34	34	34	34	34	34	34	34	34	34	34
4	56	49	41	34	34	34	34	34	34	34	34	34	34	34	34	34
5	64	49	41	34	34	34	34	34	34	34	34	34	34	34	34	34
6	64	49	41	34	34	34	34	34	34	34	34	34	34	34	34	34
7	64	49	41	41	41	41	41	41	41	41	41	41	41	41	41	41
8	64	49	41	41	41	41	41	41	41	41	41	41	41	41	41	41
9	64	49	49	49	49	49	49	49	49	49	49	49	49	49	49	49
10	64	49	49	49	49	49	49	49	49	49	49	49	49	49	49	49
11	64	56	56	56	56	56	56	56	56	56	56	56	56	56	56	56
12	64	56	56	56	56	56	56	56	56	56	56	56	56	56	56	56
13	64	56	56	56	56	56	56	56	56	56	56	56	56	56	56	56
14	64	56	56	56	56	56	56	56	56	56	56	56	56	56	56	56
15	64	56	56	56	56	56	56	56	56	56	56	56	56	56	56	56

$Vseuil$

S \ V	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
1	16	16	17	18	18	18	18	18	18	18	18	18	18	18	18	18
2	16	17	18	19	20	20	20	20	20	20	20	20	20	20	20	20
3	16	18	19	20	21	22	22	22	22	22	22	22	22	22	22	22
4	16	20	22	24	25	25	25	25	25	25	25	25	25	25	25	25
5	16	20	24	28	30	30	30	30	30	30	30	30	30	30	30	30
6	16	20	28	32	35	35	35	35	35	35	35	35	35	35	35	35
7	16	20	32	36	40	40	40	40	40	40	40	40	40	40	40	40
8	16	20	32	36	40	40	40	40	44	44	44	44	44	44	44	44
9	16	20	32	36	40	44	44	44	48	48	48	48	48	48	48	48
10	16	20	32	36	40	44	48	52	52	52	52	52	52	52	52	52
11	16	20	32	36	40	44	48	52	56	56	56	56	56	56	56	56
12	16	20	32	36	40	46	52	56	56	56	56	56	56	56	56	56
13	16	20	34	38	42	46	52	56	56	56	56	56	56	56	56	56
14	16	20	34	38	42	46	52	56	56	56	56	56	56	56	56	56
15	16	20	34	38	42	46	52	56	56	56	56	56	56	56	56	56

On constate qu'à faible intensité, les seuils de H et de S sont très élevés (ce qui revient à les ignorer dans l'évaluation du prédicat D). Lorsque la saturation est faible, il y a peu de colorisation donc le seuil de H est très élevé. Cette méthode donne de très bons résultats et permet d'atténuer fortement l'effet des variations d'illumination (ombres, reflets). En pratique, il est intéressant de rendre la valeur des seuils variable d'un niveau à l'autre dans le fonctionnement de l'algorithme.

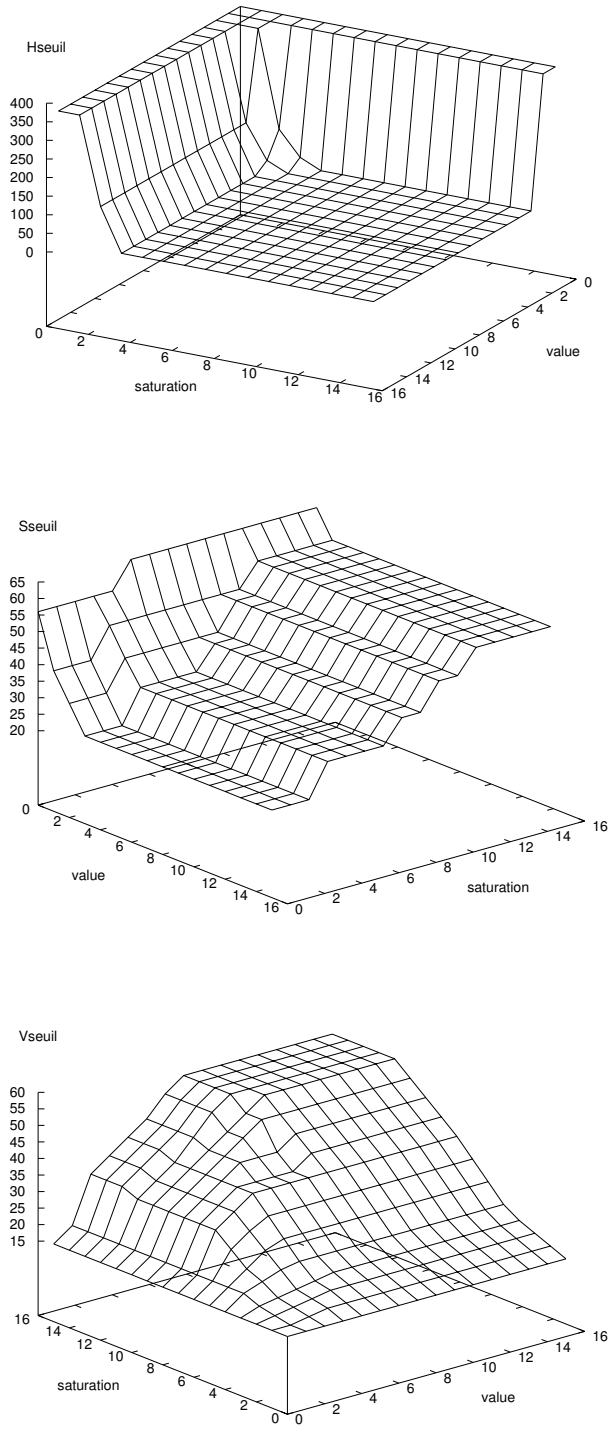


FIG. 25 – Représentation graphique des tableaux de seuil $Hseuil$, $Sseuil$ et $Vseuil$

Les éléments de niveau n seront examinés avec plus de tolérance que ceux de niveau $n+1$. Pour réaliser cette variation, on modifie la valeur du palier le plus bas du tableau Htab et des bordures extrêmes des tableaux Stab et Vtab.

4.8 Complexité

La complexité de l'algorithme CSC est linéaire en le nombre de pixels. C'est un très bon résultat pour un algorithme à la fois global et local.

Notons N le nombre de pixels de l'image. La phase de pré-traitement (filtrage *snn*) est linéaire en N. Les deux principales opérations des phases d'initialisation et de regroupement sont l'évaluation du prédicat D pour le calcul de la similarité de couleur et le test de connectivité entre deux éléments. Dans la phase d'initialisation, tous les pixels voisins doivent être comparés. Il y a donc $\frac{N \cdot 6}{2} = 3N$ comparaisons. Dans cette phase qui se situe au niveau des pixels, les tests de connectivité sont implicitement donnés par la structure hexagonale. Le nombre de comparaisons de couleurs dans la phase de regroupement dépend du nombre d'éléments dans chaque îlot, qui varie d'une image à l'autre. Comme le nombre d'îlots décroît d'un facteur 4 d'un niveau à l'autre, il est raisonnable de considérer que le nombre d'éléments décroît également d'un facteur 4 en moyenne. Cette situation est exactement vérifiée dans le cas d'une image parfaitement homogène ou chaque îlot contient exactement un élément qui le recouvre entièrement. Le nombre d'éléments différents aux niveaux les plus bas est directement fonction du nombre de régions de couleurs différentes existant au niveau des pixels. Plus ce nombre est élevé, plus ces régions seront de petite taille en moyenne et moins il faudra monter dans la hiérarchie pour trouver un élément les couvrant totalement. Donc, dans ce cas, il y aura moins d'éléments dans les plus hauts niveaux. Ces considérations se vérifient expérimentalement et on constate que le nombre d'éléments à chaque niveau correspond à peu près au nombre d'îlots plus une constante (petite). On peut donc estimer le nombre total de comparaisons de couleur, phase d'initialisation comprise, pour une image de 512x512 (moins de 8 niveaux) par :

$$3 \cdot N \cdot \sum_{n=0}^{8} \left(\frac{1}{4}\right)^n \approx 3 \cdot N \cdot \frac{4}{3} = 4 \cdot N$$

Les éléments appartenant à des îlots voisins ne sont pas nécessairement connectés. Il faut donc vérifier cette connectivité. Par le même raisonnement que celui que nous venons de détailler, on aboutit au fait que le nombre de tests de connectivité dans la phase de regroupement est de l'ordre de N. Deux éléments sont connectés s'ils partagent un même sous-élément. Le test correspondant se fait en temps

constant, en vérifiant si les deux éléments possèdent un pointeur commun dans leur liste de sous-éléments[5].

En général, le temps nécessaire pour les phases de pré-traitement, d'initialisation et de regroupement est presque constant et ne dépend pas de l'image. Ce n'est pas le cas de la phase de découpage. Le coût d'un appel à la procédure de découpage récursif dépend du niveau où il est effectué et de la taille de la frontière entre les deux éléments à séparer. Le nombre de découpages dépend de la nature de l'image mais en pratique, le temps consacré à ces opérations est de l'ordre de 5% du temps total de calcul. Il peut donc être considéré comme négligeable. Ceci est dû au fait que le nombre de découpages nécessaires est faible.

4.9 Sensibilité aux paramètres

Une propriété très importante de l'algorithme CSC, et qui en fait un très bon algorithme de segmentation, est sa faible sensibilité aux variations de paramètres. Le traitement effectué est donc robuste et ne nécessite pas l'utilisation d'une valeur *ad hoc* de paramètre comme c'est souvent le cas pour les algorithmes d'accroissement de régions.

Nous avons voulu illustrer cette sensibilité à l'aide d'images concrètes. Reprenons l'exemple déjà utilisé pour l'accroissement de région et l'algorithme "split and merge" (fig 26).

Les deux paramètres que nous allons faire varier sont :

- h : La valeur plancher de Hseuil (c'est la valeur de seuil sur H correspondant à la zone de validité de H où S et V ne sont pas trop petits)
- Δh : la variation du plancher h d'un niveau à l'autre. Une valeur négative va avoir tendance à sur-segmenter l'image. Une valeur positive va regrouper plus facilement les régions. NB : la variation considérée ici est une variation arithmétique. On peut considérer une variation géométrique mais les résultats sont moins intéressants.

On fixe $\Delta s = \Delta v = 0$, ce qui signifie que les tableaux Sseuil et Vseuil restent inchangés d'un niveau à l'autre. Les images suivantes montrent différentes valeurs :

Pour les valeurs raisonnables (et conseillées) de h et Δh qui sont respectivement autour de 20 et de 0, il est quasiment impossible de distinguer les différences

[5]Ce test efficace n'est possible que grâce à la structure hexagonale hiérarchique avec recouvrement. Dans une structure sans recouvrement, ce test doit être effectué au niveau du pixel, ce qui est extrêmement coûteux.

FIG. 26 – Image originale avant algorithme CSC

FIG. 27 – Segmentation CSC avec $h = 20$ et $\Delta h = 0$

FIG. 28 – Segmentation CSC avec $h = 20$ et $\Delta h = 2$

FIG. 29 – Segmentation CSC avec $h = 20$ et $\Delta h = -2$

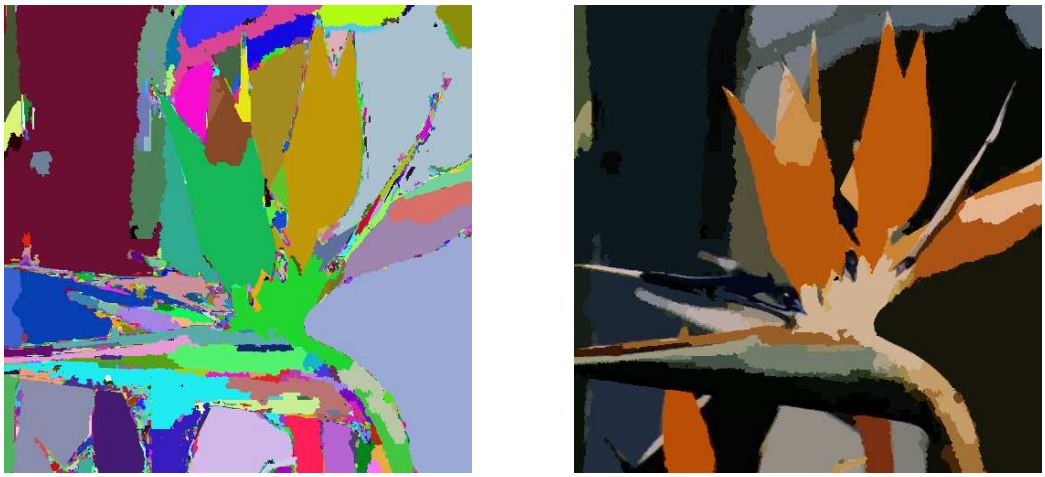


FIG. 30 – Segmentation CSC avec $h = 30$ et $\Delta h = 2$

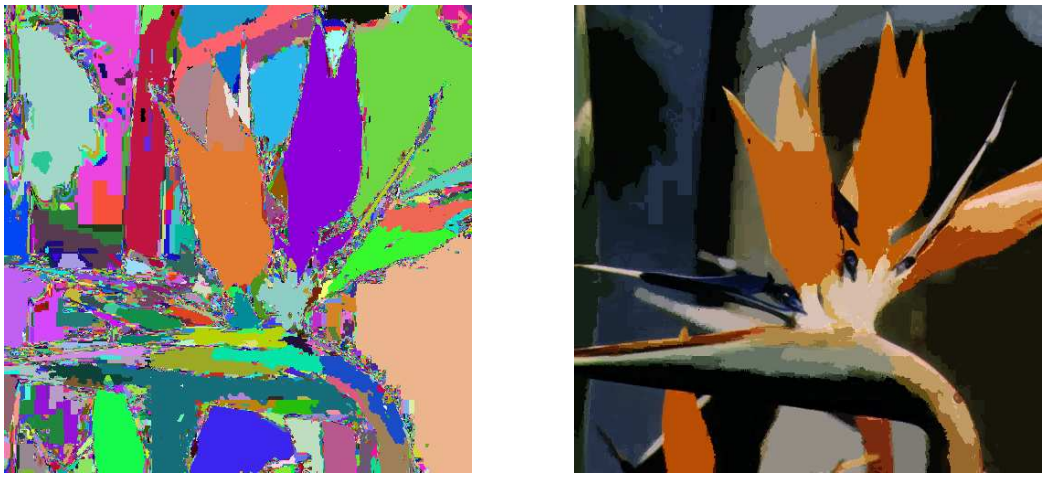


FIG. 31 – Segmentation CSC avec $h = 1$ et $\Delta h = 2$

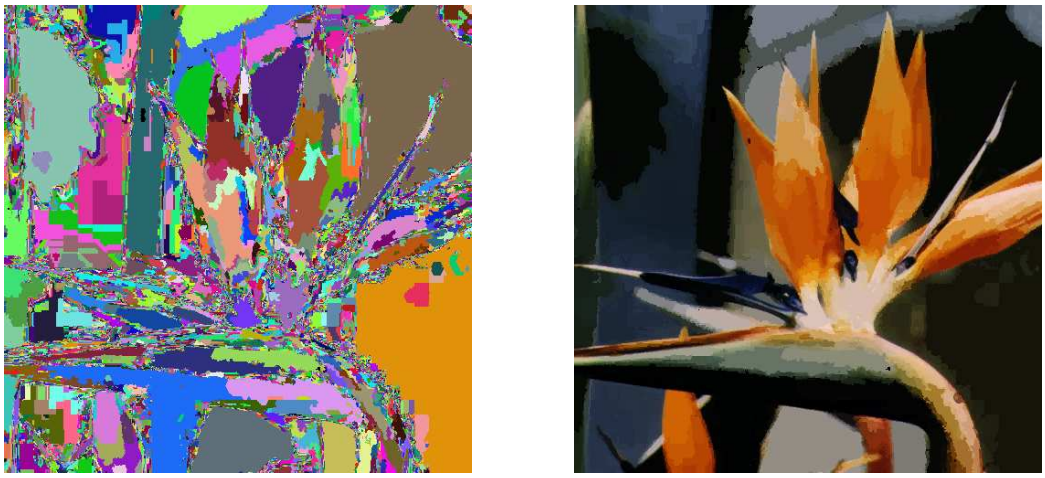


FIG. 32 – Segmentation CSC avec $h = 1$ et $\Delta h = 0$

entre deux images. Elles existent cependant, et on peut les révéler en effectuant une différence et en amplifiant le résultat. Ce n'est que près de valeurs exagérées comme $h = 1$ (c'est à dire une quasi réduction du prédicat D à l'identité) que l'on observe des différences sensibles et une perte de qualité. D'autres exemples sur d'autres images sont présentés au paragraphe 4.11. Dans tous les cas, l'algorithme montre la même robustesse. Une fois les régions de petite taille filtrées, on obtient généralement un découpage en régions stable et relativement consistant même sur une image qui bouge légèrement ou pour laquelle la luminosité varie.

4.10 Implémentation et calcul parallèle

Nous avons réalisé une implémentation de l'algorithme CSC en C++ sur un PC Athlon 1.2GHz, 256Mo RAM. Un certain nombre d'optimisations ont été mises en place : pré-calcul des transformations RGB→HSV, pré-calcul de la topologie hexagonale, réduction au minimum des appels de fonction (fonction `inline`), pré-allocation des zones mémoires pour le stockage des arbres d'éléments, remplacement systématique des listes chaînées par des tableaux dynamiques,... Une dernière optimisation reste à mettre en place qui consiste à obtenir l'image de la caméra directement en coordonnées HSV. Actuellement, la caméra est utilisée en mode RGB et le temps de conversion pourrait être économisé.

Nous arrivons à une vitesse de traitement de l'ordre de 10Hz pour des images 160x120 en provenance d'une caméra RGB. Nous n'avons pas cherché à travailler

avec des images de plus haute résolution puisque dans notre application (tracking d'objets colorés) les détails ne sont pas importants. Il est intéressant de noter que le résultat de l'algorithme sur une image de résolution réduite et quasi identique à celui obtenu sur l'image originale (fig 33). C'est une des propriétés de l'algorithme CSC de n'être pas sensible à l'échelle, ce qui permet d'augmenter la rapidité de l'algorithme simplement en réduisant la résolution. Ce n'est pas le cas de l'algorithme d'accroissement de région dont les performances passent par un optimum à une échelle donnée.

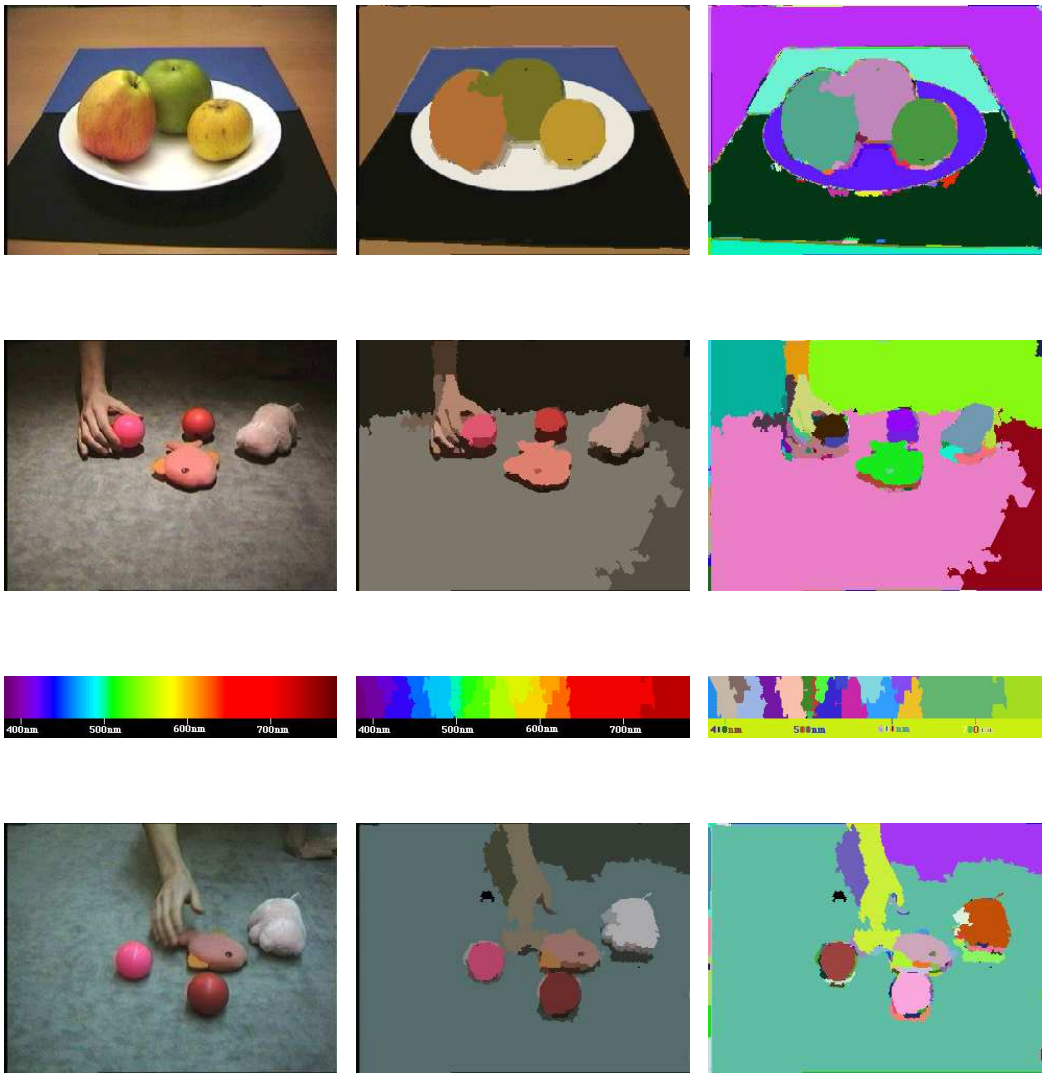
FIG. 33 – Segmentation CSC avec $h = 20$ et $\Delta h = 0$. En haut, 512x512, en bas 100x100

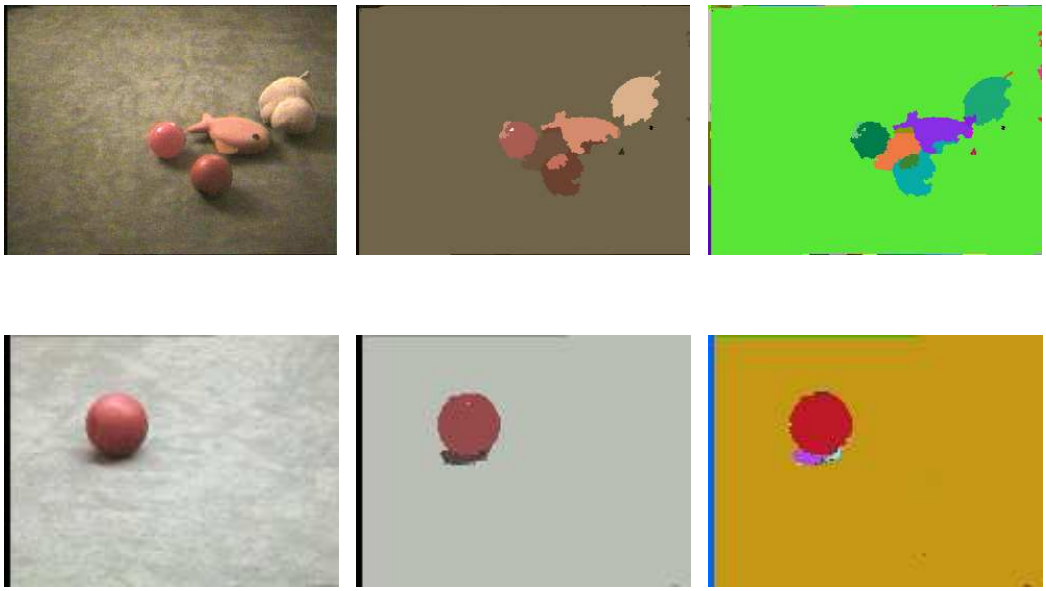
Nous projetons d'améliorer la vitesse d'exécution en implémentant une version parallélisée de l'algorithme. Comme chaque traitement à chaque niveau s'ef-

fectue localement dans un îlot, l'algorithme est naturellement parallèle. L'ensemble des îlots peut être divisé en deux ou quatre parties attribuées chacune à un processeur. La gestion des priorités d'accès aux données est relativement simple dans ce cas.

4.11 Exemples

Les exemples ci-dessous illustrent le comportement de l'algorithme sur des images difficiles. De gauche à droite sont disposées l'image originale, la segmentation en vraies couleurs et la segmentation en fausses couleurs :





Références

[1] Barba D. Benois J. Image segmentation by region-contour cooperation for image coding. In *ICPR92, La Haye*, pages vol C :pp 331–334, 1992.

[2] Largarde-Menet S. Bonin P. Segmentation coopérative région/contour basée sur des critères d'analyse texturelle en vue d'une application à la stereovision. In *RFIA89*, pages pp 645–660, 1989.

[3] L. Brun and J. Domenger. A new split and merge algorithm with topological maps and inter-pixel boundaries. *To be published in WSCG'97*, 1996.

[4] F. Cheevasuvut, H. Maitre, and D. Vidal-Madjar. A robust method for picture segmentation based on a split and merge procedure. *CVGIP, 34 :268–281, 1986.*, 1986.

[5] P. Chen and T. Pavlidis. Segmentation by texture using a co-occurence matrix and a split and merge algorithm. *Computer Graphics, and Image Processing, 10 :172–182.*, 1979.

[6] S. Chen, W. Lin, and C. Chen. Split-and-merge image segmentation based on localized feature analysis and statistical tests. *CVGIP : Graph. Models Image Process., vol. 53, pp. 457–475*, 1991.

[7] A.R. Hanson and E.M. Riseman. Segmentation of natural scenes. In *Computer Vision Systems, A. R. Hanson, and E. M. Riseman (Eds.), New York : Academic Press*, pages 129–163, 1978.

[8] Haralick and Shapiro. Image segmentation techniques. *Applications of Artificial Intelligence II*, 1985 April 9-11.

[9] S.L. Horowitz and T. Pavlidis. Picture segmentation by a directed split and merge procedure. In *CMetImAly77*, pages 101–11, 1977.

[10] Y. Ohta. Knowledge-based interpretation of outdoor natural color scenes , pitman. *LONDON*, 85 :1985, 1985.

[11] M. Pietikainen and D. Harwood. Segmentation of color images using edge-preserving filters. *In V. Cappellini and R. Marconi, editors, Advances in Image Processing and Pattern Recognition, pages 94–99. North-Holland.*, 1986.

[12] L. Priese and V. Rehrmann. A fast hybrid color segmentation method. *In S. J. Poppl and H. Handels, editors, Mustererkennung 1993, pages 297–304. Springer Verlag, 1993. 15. DAGM-Symposium, Lubeck, 27.-29.*, 1993.

[13] V. Rehrmann. echtzeitf ahige farbbildauswertung. *Volker Rehrmann, Stabile, echtzeitf ahige Farbbildauswertung, Verlag Folbach.*, 1994.

[14] Wladyslaw Skarbek and Andreas Koschan. Colour image segmentation – a survey –. Technical report, Institute for Technical Informatics, Technical University of Berlin, October 1994.

[15] Monga O. Wrobel B. Segmentation d'images naturelles : coopération entre un détecteur-contour et un détecteur-région. In *GRETSI87*, pages pp 539–542, 1987.