Université de Genève
Faculté des sciences

Département d'informatique
Vendredi 2 juillet 2004

Travail de diplôme

Chekroun Olivier

Partitionnement d'image robuste pour l'indexation par région

Table des matières

1 Introduction

Ces dernières années on a vu la prolifération de collection d'images : appareils photos numériques et collection personnelle, collection de photos dédiées à un sujet particulier, bases de données propriétaires et bien sur le WEB avec sa facilité de diffuser des images. Malheureusement, les systèmes de recherche pour explorer ces collection n'ont pas suivi le même essor et c'est à l'heure actuelle un domaine en pleine effervescence. Pour qu'un tel système soit utilisable il faut un moyen de cataloguer ou d'indexer ces images.

Plusieurs moyens existent pour indexer ces images et permettre des recherches. On a d'abord l'approche textuelle : pour chaque image on donne quelques mots clés et/ou une description. Cette méthode est très facile à implémenter mais le travail à fournir est titanesque et la description d'une image est sujette à une subjectivité incompatible avec une recherche systématique dans une base de données. Le moteur de recherche `Google` permet de recherches des images de manières textuelles, dans ce cas l'algorithme doit vraisemblablement utiliser le contexte de la page HTML contenant l'image pour en tirer des mots clés.

Ensuite on a l'approche basée sur les caractéristiques de l'images : textures, histogrammes, etc. On peut déduire de manière systématique beaucoup de caractéristiques et donc obtenir des requêtes performantes. Cependant, un utilisateur recherche en général des objets, donc un niveau de description élevé, alors que les caractéristiques de l'image sont de bas niveau.

Finalement, l'approche la plus en vogue en ce moment est celle basée sur les régions. On essaye d'extraire des images des régions représentant des objets. On permet à l'utilisateur de comparer ces régions. Une région devrait représenter, idéalement, un objet ou une partie d'un objet. Les systèmes `BLOBWORLD` et `SIMPLIcity` rentrent dans cette catégorie.

Le problème de la recherche des régions est relié à celui de la segmentation. Tout d'abord, une remarque générale : on va utiliser le terme "segmentation" d'une manière abusive. Ce terme va sous-entendre les manières de décomposer une image en parties intéressantes qu'elle que soit la méthode utilisée.

La segmentation d'images est un domaine particulièrement étudié depuis une trentaine d'années. Mais on a découvert que relativement récemment que le problème de la segmentation n'a pas de solution optimale : une bonne segmentation dans une situation donnée est mauvaise pour une autre. Par exemple, si on a une image d'un chien, est-ce que la décomposition doit donner une région représentant le chien en entier, ou plusieurs représentant les pâtes, la tête et le corps ?

De manière informelle, "segmenter une image en" signifie séparer les pixels de l'image en régions de tel sorte que chaque pixel d'une même région appartient à un même objet, ou encore : établir une partition d'une image I en régions $R_1, \ldots, R_n$ telle que :

$$I = \bigcup_{i=1}^{n} R_i \text{ et } R_i \bigcap R_j = \emptyset \ \forall i \neq j$$

La principale difficulté réside dans le fait que les objets sont définis à un haut niveau

d'interprétation alors que l'information à disposition, couleur et position d'un pixel, est de bas niveau.

1.1 Objectifs du travail

Le but de ce travail est de fournir un programme de partitionnement d'image robuste. Le terme robuste signifie que l'on peut appliquer le programme à une série d'images sans devoir le paramétrer différemment pour chaque image. En effet, les algorithmes de segmentation sont toujours présentés sous leur meilleur jour : on montre le résultat sur une image avec des paramètres optimisés pour cette image. Or le programme demander doit servir à l'indexation d'images pour leur recherche. On désire donc un programme pouvant traiter une collection d'images sans avoir à les paramétrer une à une.

La sortie du programme doit fournir un résultat permettant l'indexation, mais malgré le titre de ce travail, on ne s'occupe que du partitionnement d'image.

Il ne s'agit pas non plus de présenter une nouvelle méthode de segmentation miracle mais d'utiliser des algorithmes déjà établis.

L'implémentation des algorithmes a été réalisée en `C++` afin d'obtenir des performances élever et permettre une bonne montée en charge sur des images de grande taille.

On favorise un ensemble de classes clairement délimitées et une implémentation propre des algorithmes plutôt que des performances absolues mais peu lisibles.

Le choix du format d'image s'est porté sur le `PNG`, car pour stocker le résultat de la segmentation on a besoin d'un format sans perte, ce qui est le cas de `PNG`.

Enfin, le programme doit implémenter 3 algorithmes de segmentation spécifiques.

Le premier algorithme retenu n'a pas de nom mais je l'appelle "MInt" ou "Basé sur un graphe". C'est un algorithme récent basé sur un graphe et l'arbre de recouvrement minimal qui utilise une heuristique originale pour segmenter une image.

Ensuite, c'est le célèbre algorithme des K-Means. Dans un premier temps on teste la possibilité d'un prétraitement par cet algorithme pour l'algorithme basé sur un graphe.

Finalement, c'est l'algorithme du "Mean Shift" (décalage de la moyenne, mais personne n'utilise ce terme) qui a été implémenté. C'est algorithme est basé sur des travaux relativement récent au niveau mathématique, 1975, remis au goût du jour il y a une dizaine d'année.

Dans le cas MInt et du Mean Shift, le programme doit produire des résultats similaires aux implémentations originales.

Ces 3 algorithmes sont à priori complètement indépendant, mais on verra qu'il y a des connexions intéressantes entre eux.

2 Algorithmes de segmentation

2.1 Petite revue des méthodes de segmentation

L'étude de la segmentation date d'une trentaine d'années ce qui est relativement vieux dans le monde informatique. Par conséquent, la segmentation d'images est un domaine vaste où l'on retrouve de très nombreuses approches et chacune de ces approches comporte elle-même des techniques très différentes. On va néanmoins résumer les grandes catégories.

Le seuillage a pour objectif de segmenter une image en plusieurs classes en n'utilisant que l'histogramme. On suppose donc qu'une région est caractérisée par sa distribution de niveaux de gris. A chaque pic de l'histogramme est associé une classe. Il existe de très nombreuses méthodes de seuillage d'un histogramme. La plupart de ces méthodes s'appliquent correctement si l'histogramme contient réellement des pics séparés. De plus, ces méthodes ont très souvent été développées pour traiter le cas particulier de la binarisation d'image et leur généralité face aux cas multi classes n'est que très rament garantie. On utilise ces méthodes dans des applications très particulières telles que le contrôle de qualité où la maîtrise du milieu ambiant et la simplicité des scènes permet d'avoir de très bonnes conditions d'éclairage

Le modèle de région exprime la segmentation vis à vis d'un critère d'homogénéité H, du type : "tous les pixels sont du même niveau de gris" ou "la variation de niveau de gris n'excède pas n niveau". On peut reformuler sous la forme : la segmentation d'une image I en regard du critère H est une partition de l'image I en n régions homogènes $X_1, \ldots, X_n$ telles que :

- $\bigcup_{i=1}^{n} X_i = I$
- $\forall i, X_i$ est connexe
- $\forall i, H[X_i]$ est vrai
- $\forall$ couple (X_i, X_j) de régions voisines, $H[X_i, X_j]$ est faux

Parmi ces méthodes de modèle région, on mentionne le "Split and Merge". Le processus est décomposé en 2 étapes. L'image initiale peut être l'image brute ou une première partition résultant d'une analyse grossière. Dans la première étape, "Split", on analyse individuellement chaque région X_i. Si celle-ci ne vérifie pas le critère d'homogénéité, alors on divise cette région en blocs et l'on itère sur chaque sous région. Dans la seconde étape, "Merge", on étudie tous les couples de régions voisines (X_k, X_i). Si l'union des 2 régions vérifie le critère d'homogénéité, alors on fusionne les régions. (On utilise souvent une structure de données Quadtree pour diviser l'image.)

On a une autre manière de construire les régions : la croissance de région. Le principe d'un algorithme de ce type est le suivant. On se fixe un point de départ dans l'image, un germe. On se donne aussi un critère d'homogénéité. Par une procédure récursive de parcourt des voisins, on inclut dans la région tous les points connexes qui vérifient le critère.

Le problème inverse de construire une région est d'en trouver les contours. Deux approches existent à nouveau : l'analyse locale versus l'analyse globale utilisant la transformée de Hough.

Pour l'analyse locale, on ne cherche pas à fixer un seuil de niveau de gris au-dessous duquel on serait dans un objet et au-dessus duquel on serait dans un autre, ce qui pourrait

peut-être toutefois s'avérer efficace pour une image d'un objet clair sur un fond sombre par exemple. Par contre, on utilise l'image dérivée qui met en évidence les variations de niveau de gris. Quand il y a une forte variation, cela est souvent dû à un changement de teinte, d'éclairement ou de texture et de façon générale, à un changement de zone. On peut utiliser la dérivée première de l'image et chercher les maximums ou prendre la dérivée seconde de l'image et chercher en quels points elle s'annule. En général, on prend l'image dérivée puis on effectue un seuillage. Pour approximer ces dérivées, on utilise des opérateurs dérivatifs.

Les contours peuvent être vus d'une manière différent : on utilise un modèle de contour déformable, nommé contour actif (snake) qui utilise des techniques d'optimisation pour trouver la meilleure approximation. Cette technique peut être vue comme un compromis entre l'analyse globale et locale et a pour avantage d'extraire des contours discontinus dans des images très bruitées. Le modèle est composé d'un groupe de fonctions d'énergie et le contour est obtenu par la minimisation de la somme pondérée de ces fonctions d'énergie. Donc, la performance du sneaker pour la détection de contour dépend en grande partie de la précision de la méthode d'optimisation.

On a aussi l'approche de la classification. En chaque pixel, on calcul un certains nombre de variables. Ensuite, on utilise des techniques d'analyses de données pour rechercher les valeurs pertinente de l'ensemble de données, par exemple la méthode des K-Means.

Les méthodes basée sur des graphes on été considérée inefficace très tôt dans "l'histoire" de la segmentation. Pourtant des avancées récentes en algorithmique les ont remises au goût du jour. Notamment. l'algorithme des NCUT à pour but de décomposer algébriquement un graphe en maintenant de bonne propriété. La recherche des régions se ramène à un calcul de valeurs propres de la matrice d'adjacence du graphe.

Un mot sur la notion de texture, bien qu'assez intuitive elle est assez délicate à formaliser et on peut en donner la définition suivante : une région de l'image possédant une organisation spatiale homogène. Le terme d'organisation fait référence à une structure identifiable ou quantifiable, à l'opposé d'une distribution anarchique.
Les textures pose très souvent un problème pour la segmentation. En effet, le critère d'homogénéité choisi peut être relativement simple mais prendre également des formes plus sophistiquées. On peut ainsi envisager de contraindre une variance local inférieure à un seuil, ce qui s'apparente à une homogénéité de la texture. La caractérisation d'une texture doit faire appel à une large expertise et ne peut se résumer à une approche simple de type seuillage ou définition d'un paramètre. On s'oriente alors vers des techniques multiparamétriques et ces paramètres sont rassemblés dans un entité unique multidimensionnelle appelée vecteur caractéristique.

On va maintenant décrire beaucoup plus précisément les algorithmes de ce travail.

2.2 Basé sur un graphe : MInt et Segment

L'algorithme décrit dans [1] est basé sur la représentation d'une image par un graphe :

– les sommets sont les pixels
– les arêtes sont des arêtes d'adjacence entre pixel voisin et leur poids la distance dans

l'espace de couleurs des pixels reliés par cet arêtes.

L'idée est d'utiliser la structure d'arbre de recouvrement minimal pour obtenir une segmentation. Un des algorithmes permettant de trouve l'arbre de recouvrement minimal d'un graphe est l'algorithme de Kruskal. Il se déroule de la manière suivante:

1. on affecte une région à chaque sommet
2. tri des arêtes par poids non décroissant
3. parcourt des arêtes triées : si les 2 extrémités de l'arête font partie de la même région on continue, sinon on fusionne les 2 régions et on ajoute l'arête en cours à la région obtenue

En pratique, lors du parcours des arêtes on a un prédicat qui décide de la fusion de 2 régions ou non. Dans le cas de l'algorithme de Kruskal, c'est l'appartenance à une même région et comme résultat on obtient une région, le graphe, et son arbre de recouvrement minimal. En utilisant d'autre prédicat, on peut construire un ensemble de régions ayant chacune leur propre arbre de recouvrement minimal.

La nouveauté de [1] est d'ajouter au prédicat de l'algorithme de Kruskal, une mesure de similarité entre les régions et si l'arête que l'on teste pour la fusion relie 2 régions trop différentes, on rejette la fusion.

Une autre nouveauté est que le prédicat est adaptatif, contrairement à la plupart des algorithmes basés sur les graphes, ce qui permet en ne prenant que des décision locales de capturés certaines propriété globales.

Si l'on regarde l'image de gauche de la figure 1, on trouve normal de dire qu'elle est composée de 3 régions : le dégradé à gauche, une partie grise à droite avec un rectangle "agité" au milieu. Pour que la région de gauche soit reconnue, on ne peut pas considérer une grande variation comme la marque d'une frontière. De plus, la variation entre la frontière de la rampe et la zone constante est plus petite que beaucoup de variation de dans la zone "agitée". On doit donc utiliser un critère non local.

Le prédicat présenté dans [1] mesure la présence d'une frontière entre 2 régions en comparant 2 quantités :

1. la variation à travers la frontière
2. la variation interne entre points voisins d'une région

On a besoin de plusieurs définitions pour établir ce prédicat et on utilisera les notations suivante : $G = (V, E)$ le graphe et $w(e)$ le poids d'une arête.

D'abord la différence interne d'une région est la valeur maximum de l'arbre de recouvrement minimal de la région :

$$Int(C) = \max_{e \in MST(C,E)} w(e)$$

Ensuite, on pose la mesure de la différence de 2 régions d'un graphe :

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w(v_i, v_j)$$

c'est le minimum de toutes les arêtes ayant une extrémité dans une région et l'autre extrémité dans l'autre région.

Le prédicat de fusion dépend du minimum des variations internes plus une valeur de tolérance, τ dépendant de la taille des régions. On obtient :

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2))$$

avec : $\tau(C) = \frac{k}{|C|}$

Le prédicat de fusion entre région est finalement :

$$D(C_1, C_2) = \begin{cases} vrai & \text{si } Dif(C_1, C_2) > MInt(C_1, C_2) \\ faux & \text{sinon} \end{cases}$$

Avec ces définitions, on peut décrire les étapes de l'algorithme.

2.2.1 Description de l'algorithme

Soit un graphe $G = (V, E)$, avec n sommets et m arrêtes.

1. Tri de E par poids non décroissant.
2. On commence par une segmentation S^0 où chaque sommet v_i est sa propre régions.
3. Pour $q = 1 \dots m$, on construit S^q à partir de de S^{q-1} comme suit. Soit v_i et v_j les sommets connectés par la q-ième arrête, C_i^{q-1} la région de S^{q-1} contenant v_i et C_j^{q-1} la région de S^{q-1} contenant v_j. Si $C_i^{q-1} \neq C_j^{q-1}$ et $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$, alors on fusionne C_i^{q-1} et C_j^{q-1} pour obtenirS^q, sinon $S^q = S^{q-1}$.

$w(e)$ est le poids de l'arête.

2.2.2 Problèmes rencontrés

Bien que relativement intuitif cet algorithme laisse quelque points en suspend qui sont pratiquement tous liés à la valeur de τ.

On n'a pas une définition précise du poids des arêtes. On ne sait pas en lisant l'article comment sont représentée les différences de couleurs. Est-ce que le domaine de définition

est l'intervalle $[0:1]$ ou $[0:255]$? (Pour les images couleurs l'auteur de [1] la décompose en 3 images RGB)

Que représente la valeur de τ. On a pas une définition "scientifique" de ce que représente cette valeur de tolérance, on voit simplement que sur des exemples, ça donne de bon résultats.

L'algorithme ne prend absolument pas en compte l'écart maximal entre la distance maximale entre les couleurs des pixels de l'image. Donc, suivant l'image que l'on traite, on va avoir des résultats très différents pour un même paramètre.

J'ai essayé de remédier à ces problèmes mais il reste une part de mystère dans le prédicat de fusion. L'algorithme modifié est dénommé : Segment.

Tout d'abord, les couleurs sont représentées dans l'espace **HSV** avec des valeurs dans l'intervalle $[0:1]$ et les niveaux de gris aussi.

La distance entre de 2 niveaux de gris est la valeur absolue de la différence des niveaux. La distance entre 2 couleurs et la norme dans le cône **HSV** des 2 vecteurs. L'utilisation de l'espace **HSV** permet de rendre complètement transparent le type d'image que l'on analyse ce qui n'est pas le cas dans [1], où comme dit précédemment, on décompose l'image en 3 plans et l'algorithme est répliqué pour traiter les 3 plans.

Ensuite, le prédicat a changé de la manière suivante. Doit on fusionner les composants C_1 et C_2 s'il sont reliés par l'arêtte e ?

On pose : $k' = k\Delta$ où Δ est la distance maximum de l'image

Ensuite, on calcule $k1 = \frac{k'}{|C_1|}$ et $k2 = \frac{k'}{|C_2|}$.

On calcule aussi le poids moyens de chaque région : $p_1 = \frac{1}{|C_1|}\sum_{p\in C_1}$ et $p_2 = \frac{1}{|C_2|}\sum_{p\in C_2}$.

On ajoute au poids moyens calculés une tolérance pour la fusion : $f_1 = p_1 + k_1$ et $f_2 = p_2 + k_2$.

On estime le poids de la région fusionnée en calculant la moyenne des poids de chaque région et du poids de l'arête : $estimatiion = \frac{f_1+f_2+w(e)}{3}$.

On obtient :

$$D(C_1, C_2) = \begin{cases} vrai & \text{si } w(e) <= estimation \\ faux & \text{sinon} \end{cases}$$

D'autre part, lorsque'on fusionne 2 régions on ajoute le poids de l'arrête au poids du de la nouvelle région $w' = w + w(e)$, donc le poids d'une région est la somme sur l'arbre minimale : $\sum_{e\in MST(C,E)w(e)}$

FIG. 1 – Exemple de l'algorithme basé sur un graphe

Le fait d'utiliser une moyenne pondérée est non le min fait perdre la propriété suivante : avec $MInt$, lorsqu'une fusion est rejetée on n'est sur qu'elle n'aura pas lieu plus tard, alors qu'avec ce prédicat modifié on ne sait pas.

Contrairement au prédicat initial, on essaye d'évaluer ce que donne le résultat de la fusion de 2 région.

Il semble que l'utilisation de :

– la moyenne pondérée à la place du minimum évite de rejeter des fusions
– la normalisation par le facteur Δ

rendent l'algorithme plus robuste, dans le sens que pour un paramètre donné, l'algorithme partitionne des images différentes d'une manière similaire.

Remarque : dans [**?**], l'auteur démontre plusieurs propriétés sur le graphe obtenu. On dirait qu'il s'est un peu "arrangé" pour que son prédicat lui permette de démontrer ces propriétés.

C'est algorithme est relativement simple à comprendre et à mettre en place. De plus, c'est un algorithme glouton : on ne revient jamais sur une décision, ce qui donne un flot de donnée lui aussi très simple. Finalement, hormis une certaine gourmandise pour créer le graphe initial et en prenant quelques précautions, il est très efficace : la partie la plus longue est le tri des arêtes !

Dans la suite, on parlera de "basé sur un graphe", "MInt" ou "Segment" indifféremment pour désigner cette méthode de parcourt de graphe pour en extraire des régions.

2.2.3 Exemple

Exemple de segmentation basée sur un graphe à la figure 1 avec comme paramètre 200. On obtient les résultats suivants :

	Composants	arêtes
Grid graphe	9408	37046
Fusion	3	2

On observe que les 3 régions attendues : le dégradé, le rectangle de droite et le rectangle "agité" sont trouvée.

2.3 K-Means

L'algorithme des K-Means n'est pas spécialement conçu pour la segmentation d'image. C'est en effet un algorithme de classification beaucoup employé en intelligence artificielle et en datamining car il est très simple. L'idée de l'algorithme des K-Means est de répartir l'ensemble des points d'un espace autour de k points représentatifs appelés centroïdes.

Dans le cas de la segmentation d'image, l'espace à considérer est l'espace des couleurs des pixels. La distance entre les points est la distance dans l'espace des couleurs. La recherche des centres correspond à une "palettisation" de l'image. En effet, les k centre de l'ensemble des données sont les couleurs représentatives de l'ensemble de l'image.

2.3.1 Description de l'algorithme

Soient des points p_j, k un nombre de centres et L_i l'étiquette du i-ème pixel après application de l'algorithme.

1. On choisit $K_{i=1...k}$ centres
2. On répète jusqu'à convergence.
   - Affectation chaque p_j au centre le plus proche K_i.
   - Calcul de la moyenne de chaque centre $m_i = \frac{1}{|K_i|} \sum_{p \in K_i}$
   - Affectation des nouveaux centre $K_i = m_i$
3. Pour chaque $p_{j=1...n}$, on affecte l'étiquette : $L_i = \{p \in K_i\}$

Lorsque l'on calcule la moyenne, on peut obtenir un point qui ne fait pas partie de l'ensemble de points initial. Cela peut donner des résultats un peu curieux dans l'espace des couleurs. On effectue donc une étape supplémentaire après le calcul de la moyenne : on recherche le point $p \in K_i$ le plus proche de cette moyenne.

La convergence est atteinte lorsque les points ne changent plus de centre d'une itération à l'autre. Contrairement à d'autres domaines d'application de cette méthode, on n'est pas obligé d'atteindre une converge totale. En effet, si quelques pixels ne sont pas affectés exactement à leur centre cela n'a pas d'impact visuel.

Comme dit précédemment, la méthode converge mathématiquement mais le résultat dépend fortement des valeurs initiales, donc des couleurs initiales. Pour des choix différents de centres initiaux on peut obtenir des résultats complètement différents (ce qui peut parfois être souhaité).

2.3.2 Initialisation des centres

On a choisi d'initialiser les k centres de la manière suivantes :

- On calcule l'histogramme cumulé de l'image
- On divise l'axe des cardinaux en k classes
- On choisit un point dans l'image inverse des cardinaux

dans le but de répartir les couleurs de manière symétrique, voir figure 2.

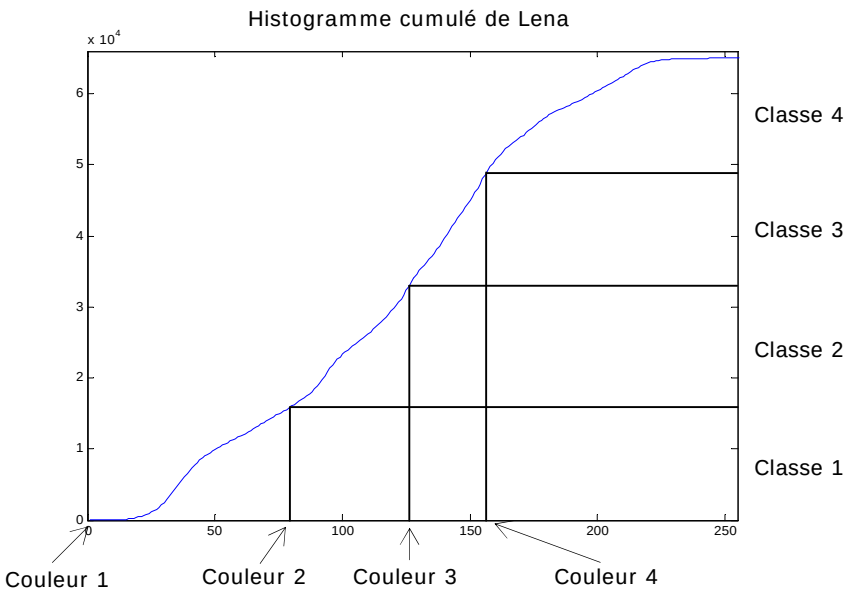


FIG. 2 – Choix des couleurs initiales

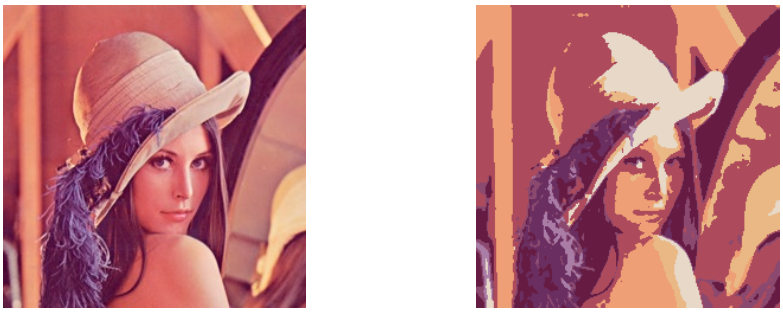


FIG. 3 – Exemple de l'algorithme des K-Means

On peut encore noter à propos de cette algorithme que les régions trouver ne sont pas forcément connexes.

On voulait utiliser l'algorithme des K-Means pour effectuer un prétraitement à l'algorithme basé sur les graphes. Dans ce but, il faut fixer un nombre k élever pour obtenir de petites régions. Gros problème : la complexité de l'algorithme explose avec k, et l'on a pas trop exploré cette voie.

Si cet algorithme peut servir à segmenter une image, le résultat obtenu n'est pas intéressant car on obtient beaucoup trop de région. C'est une méthode à utiliser conjointement avec d'autres.

2.3.3 Exemple

Exemple de segmentation par la méthode des K-Means à la figure 3 avec comme paramètre 8 (nombre de centres). On obtient : 888 composants et 2310 arêtes.

2.4 Mean Shift

La procédure du Mean Shift est une méthode récente d'analyse de données pour en trouver les modes. Ce n'est donc pas une méthode établie à première vue pour le traitement d'image, mais on peut facilement dériver des méthodes de segmentation.

La méthode est de calculé le gradient d'une fenêtre d'exploration des données centrée en un point. Ce vecteur indique la direction d'un mode locale. Il se trouve qu'on peut facilement approximer ce vecteur en calculant la moyenne des points de la fenêtre. Ensuite, on déplace le centre de la fenêtre au point calculé et en itérant ce calcul sur le nouveau point calculé on converge rapidement vers un mode locale.

2.4.1 Rappel mathématique du Mean Shift

Soit $\{X_i\}_{i=1,\ldots,n}$ un ensemble de points dans un espace euclidien de dimension d : $\mathbb{R}^d$. L'estimateur de densité multivarié de noyau avec le noyau K et rayon de fenêtre h est définé par :

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)$$

et le noyau K doit satisfaire certaines conditions.

Le noyau d'Epanechnikov est un noyau optimal :

$$K_e(x) = \begin{cases} \frac{1}{2} c_d^{-1}(d+2)(1- \parallel x \parallel) & \text{si } \parallel x \parallel < 1 \\ 0 & \text{sinon} \end{cases}$$

où c_d est le volume de la sphère unité de dimension d.

On obtient l'estimation du gradient de densité du noyau d'Epanechnikov :

$$\hat{\nabla} f(x) \equiv \nabla \hat{f}(x) = \frac{1}{nh_d} \sum_{i=1}^{n} \nabla K\left(\frac{x - X_i}{h}\right)$$

ou encore :

$$\hat{\nabla} f(x) = \frac{n_x}{n(h^d c_d)} \frac{d+2}{h^2} \left(\frac{1}{n_x} \sum_{X_i \in S_h(x)} [X_i - x] \right)$$

où $S_h(x)$ est une hypersphère de rayon h, de volume $h^d c_d$, centrée en x et contenant n_x points.

On définit le vecteur Mean Shift par :

$$M_h(x) \equiv \frac{1}{n_x} \sum_{X_i \in S_h(x)} [X_i - x] = \frac{1}{n_x} \sum_{X_i \in S_h(x)} X_i - x$$

et finalement :

$$M_h(x) \equiv \frac{h^2}{d+2} \frac{\hat{\nabla} f(x)}{f(x)}$$

Dans [5] on trouve une démonstration complète de la méthode. C'est dans ce même article qu'est décrite la segmentation par le Mean Shift utilisé dans ce travail.

On va très largement simplifier la méthode décrite. Ici, la fenêtre est un "hyper rectangle" dans l'espace des données, et le calcul de la moyenne est la moyenne des vecteurs de cet "hyper rectangle".

Pour l'analyse d'image on considère l'espace conjoint des données : spatial et attribut, avec comme attribut d'un point sa couleur.

Un point d'une image noir et blanc a 3 coordonnées (x, y, g) avec g le niveau de gris et x, y les coordonnées du point, alors qu'un point d'une image couleur a 5 coordonnées (x, y, h, s, v) avec x, y les coordonnées du point et h, s, v les composants de la couleurs dans l'espace **HSV**.

L'utilisation de l'espace **HSV** permet de ne considérer qu'un seul cas : celui des images couleurs. En effet, pour une image à niveau de gris, les coordonnées h, s sont égales à 0 et n'influent pas pour le calcul de la moyenne.

La fenêtre de recherche dépend de 3 paramètres :

1. le centre de la fenêtre
2. un rayon dans le domaine spatiale
3. un rayon dans le domaine des attributs

L'estimation des rayons dans le cas général de la recherche des modes possède un sens physique et on peut donc les établir à priori. Néanmoins, dans le cas de l'analyse d'image on a pas méthode pour fixer ces paramètre à priori. D'ailleurs, la plupart des article des auteurs de [5] depuis leur découverte consiste à l'améliorer ou à trouver des méthodes permettant d'estimer les paramètres.

La méthode de segmentation par le Mean Shift se déroule en 2 temps : le filtre Mean Shift et la fusion Mean Shift.

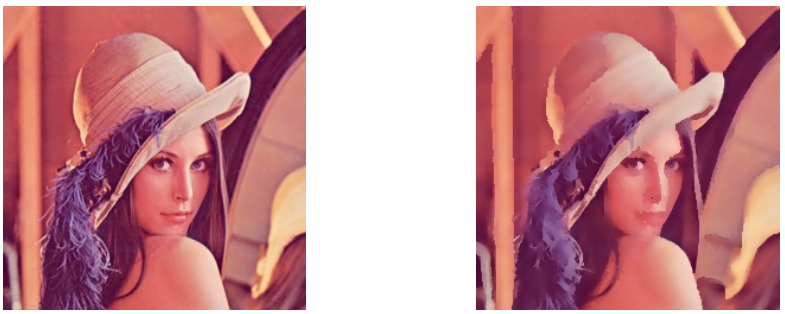


FIG. 4 – Exemple de filtre Mean Shift

2.4.2 Filtre Mean Shift

Soient $\{x_j\}_{j=1...n}$ et $\{z_j\}_{j=1...n}$ les points originaux et filtrés de l'image dans le domaine joint spatial-attribut. Les indices s et a représentent respectivement les coordonnées spatiales et d'attributs.

Pour $j = 1 \dots n$

1. Initialisation : $k = 1$ et $y_k = x_j$
2. Calcul de y_{k+1} par le mean shift jusqu'à convergence : y_c
3. $z_j = (x_j^s, y_c^a)$

Pour chaque point on calcule son mode ce qui dans le cas d'une image donne une grande quantité de calculs. Cependant, en utilisant des techniques de bassins d'attraction, on peut accélérer le calcule des modes, mais en introduisant une légère imprécision. 2 méthodes sont possibles :

1. lors de chaque itération, on teste si le point calculé est proche d'un point de l'image et si c'est le cas on associe à ce point le mode en cours de recherche
2. comme la première méthode mais au lieu d'ajouter un seul point on ajoute aussi un voisinage

La première méthode a été implémentée.

2.4.3 Segmentation Mean Shift

Soient $\{x_j\}_{j=1...n}$ et $\{z_j\}_{j=1...n}$ les points originaux et filtrés de l'image dans le domaine joint spatial-attribut, et L_i l'étiquette du i-ème pixel dans l'image segmentée.

1. Application du filtre Mean Shift à l'image, mais $z_i = y_c$
2. Trouver dans le domaine joint les régions $\{C_p\}_{p=...m}$ en groupant tous les z_i qui sont plus proche de h_s dans le domaine spatial et h_a dans le domaine des attributs
3. Pour chaque $j = 1 \dots n$, on affecte l'étiquette : $L_i = \{p \mid z_i \in C_p$
4. Option : éliminer les régions qui moins de M pixels

FIG. 5 – Exemple de segmentation Mean Shift

On peut souligner qu'aussi bien les fondements mathématiques de cette méthode, que la méthode elle même sont relativement récents. C'est pour cela que l'on trouve de grandes différences dans la segmentation par le Mean Shift suivant les articles auxquels on se réfère.

Ici on utilise tous les modes calculés, dans d'autres méthodes on choisit des modes aléatoirement en espérant qu'il soit représentatif. Dans [9], on décompose l'image par l'algorithme SVD ce qui donne des points singulier dont on calcule les modes. En bref, la segmentation Mean Shift n'est pas une méthode bien délimitée : n'importe quelle utilisation de la procédure Mean Shift pour segmenter peut rentrer dans cette catégorie.

2.4.4 Exemple

Exemple de filtre et segmentation Mean Shift figure 4 et figure 5 avec comme paramètres de la fenêtre 3 et 0.2 et un élagage des régions plus petites que 500 pixels. On obtient les résultats suivants :

	Composants	arêtes
Filtre	55522	215036
Fusion	187	317
Elagage	19	37

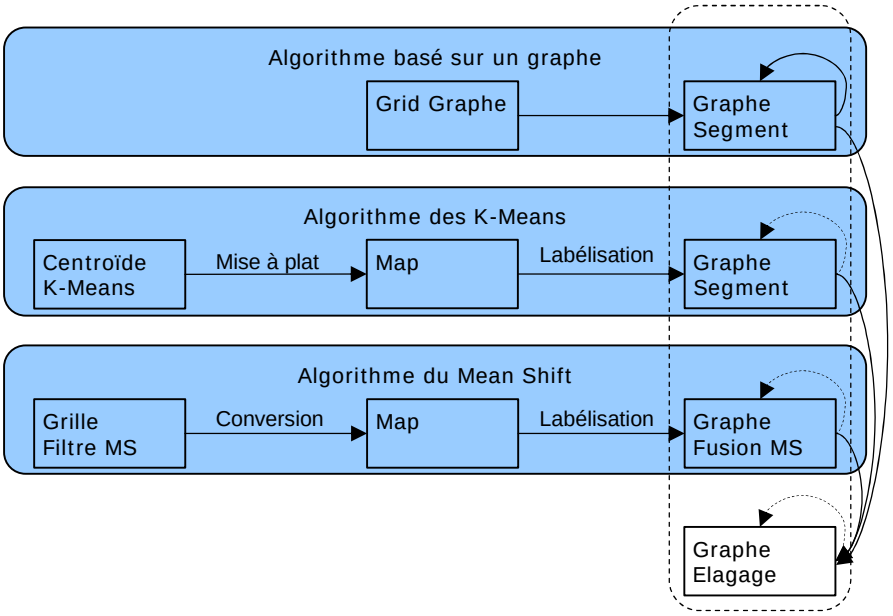


FIG. 6 – Workflow

3 Mise en oeuvre : programme `segment`

L'objectif de travail est fournir un programme en ligne de commandes : `segment`. Il doit permettre d'utiliser les algorithmes précédents et offrir certaines combinaisons entre eux. Le programme traite des images au format `PNG` couleurs ou noir et blanc. En sortie, il fournit des images au format `PNG` du résultats de la segmentation. La détection du type d'images est automatique.

La figure 6 montre les différents algorithmes employés et les rapports entre eux. Les boites sont séparées en 3 catégories selon leur relation avec les graphes d'adjacence :

1. indépendant des graphes
2. entrée quelconque, traitement et sortie d'un graphe
3. entrée un graphe , opérations sur les graphes et sortie d'un autre graphe

En particulier, pour la dernière catégorie, la sortie peut être réutiliser comme entrée !

On voit donc qu'on a besoin d'une structure de graphe et d'un mécanisme permettant d'effectuer efficacement des opérations sur ces graphes. Le fait d'utiliser en entrée un graphe et de produire un graphe impose que les graphes soient hiérarchiques.

3.1 Structure de graphe hiérarchique

L'idée de départ est d'utiliser l'algorithme MInt ou Segment pour :

– appliquer d'abord l'algorithme avec une faible valeur du paramètre ce qui donne un graphe de région

Structure de graphe hiérarchique

FIG. 7 – Structure de graphe hiérachique

– on applique à nouveau l'algorithme à ce graphe ce qui en donne un autre mais avec moins de composants et d'arêtes mais un niveau hiérarchique en plus

La figure 7 montre qu'on obtient une "pyramide de segmentation" : on a une structure hiérarchique des regroupements de points de l'image. On peut noter que le premier étage de la pyramide est en général coûteux en temps de calcul. **Par contre, les étages suivants sont quasiment gratuits !**

On trouve beaucoup de bibliothèque implémentant le type de donnée graphe comme structure mathématique. Cependant, elles sont en générale très complète et très lourdes. Or les graphes que l'on manipule peuvent être très conséquent et on a besoin d'une structure légère.

Concrètement, on a 2 type de noeuds : les feuilles, qui correspondent aux pixels et les noeuds qui regroupent des feuilles ou des noeuds. Les opérations supportées par cette structure sont uniquement celles dont on a besoin et ne correspondant pas du tout au type de graphe abstrait admis en général. Le graphe lui-même est conçu sous forme de liste d'adjacence.

Si c'est naturellement pour implémenter l'algorithme basé sur un graphe et obtenir la pyramide de segmentation que cette structure a été développée c'est plutôt avec surprise que qu'on a pu la réutiliser dans la segmentation Mean Shift !

Contrairement à l'implémentation effectuée par les auteurs de [1] et [5], ici on a une réelle structure de graphe hiérarchique et pas seulement une image du résultat. Ce qui veut dire qu'on peut appliquer n'importe quel algorithme de graphe sur le résultat obtenu. D'autre type de noeud peuvent être introduit en cas de besoin.

3.2 Labellisation et recherche de frontières

Pour différentes opérations on a besoin de passer d'une grille à plat des composants tenant compte de la position spatial des pixels à une représentation sous formes de graphes. Inversement pour trouver les bords des composants il faut passer de la structure de graphe à une grille des données. La figure 8 montre le rôle de conversion de la structure que j'ai appelée "Pixel Map".

Les algorithmes des K-Means et du filtre Mean-Shift ne produisent pas une décomposition de l'image d'entrée sous formes de composantes de connexes. Il faut observer le résultat de ces algorithmes et effectuer une labellisation. Celle-ci consiste à mettre sous forme

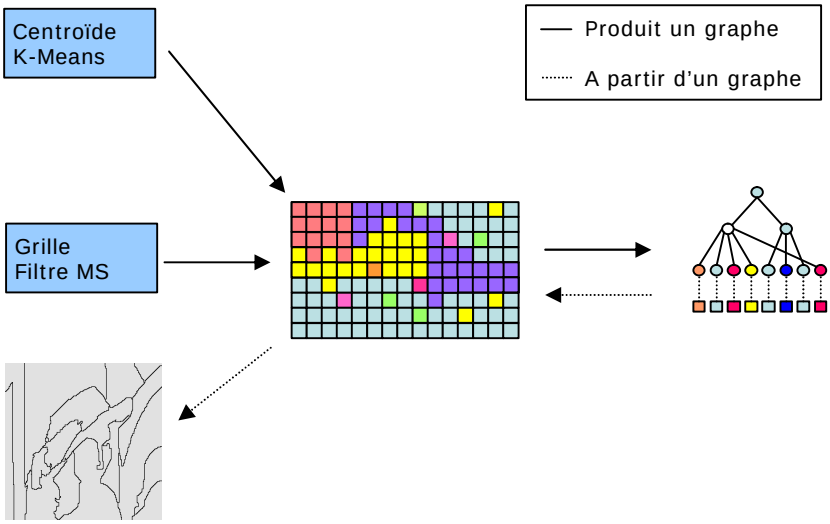


FIG. 8 – "Pixel Map"

rectangulaire le résultat de chaque algorithme en produisant une image associant à chaque pixel de l'image de départ le résultat de l'algorithme : centre pour les K-Means et mode pour le Mean Shift.

Ensuite, il faut déterminer les composantes connexes de l'image. Ceci ce fait de la manière suivante : on parcourt l'image de gauche à droite et de haut en bas en largeur d'abord. Soit un pixel est affecté à une composante soit il ne l'est pas. S'il est affecté on recherche tous ses voisins ayant la même couleur, sinon c'est une nouvelle composante connexe. On effectue ensuite un deuxième parcourt de l'image afin de déterminer les régions adjacentes et les arêtes qui les relient.

Pour produire les bords des composants, on a le problème inverse : les pixel sont associés à une région mais on ne sait pas si 2 pixels sont voisins. Dans ce cas, on aplatit le graphe et on parcourt l'image de gauche à droite et de haut en base. On regarde si un pixel a la même étiquette que ces voisin : si oui ce n'est pas un point du bord d'une région et sinon c'est un point du bord.

C'est cette structure qui a le rôle de plateforme entres les différents algorithmes.

3.3 Ensembles disjoints

Les algorithmes présentés utilisent des régions disjointes. Pour les représenter, on a la notion d'ensemble disjoint dynamiques. Dynamique car on effectue des opérations sur ces ensembles :

1. recherche de l'ensemble contenant un élément
2. réunion de 2 ensembles disjoints

Il faut donc une structure de données pour ces ensembles, ainsi que ces 2 opérations.

On peut à première vue croire qu'une implémentation directe sous forme de liste peut suffire, mais la complexité des opérations est exponentielle. Donc, à partir du moment où l'on manipule un grand nombre d'ensemble disjoints et où le nombre d'opérations que l'on effectue est lui aussi élevé, il faut une structure dédiés. Particulièrement, pour les algorithmes décrits, le volume de données est très importants et les traitements que l'on effectue aussi.

Une structure de données "ensembles disjoints" maintient une collection $S = \{s_1, s_2, \ldots, s_k\}$ d'ensemble disjoints dynamiques.

Chaque ensemble est identifié par un représentant, qui est un élément de l'ensemble. On veut implémenter les opérations suivantes :

– `find(y)` trouve le représentant de l'ensemble auquel appartient `y`
– `merge(x, y)` réunir les 2 ensemble auquel appartiennent `x` et `y`

On représente chaque ensemble disjoint par un arbre, la partition est alors une forêt. La racine de chaque arbre est le représentant. Dans chaque noeud, on considère juste le pointeur vers la racine (on n'a pas besoin de "descendre" dans l'arbre). Avant d'utiliser la structure d'ensembles disjoints, on indexe ceux-ci à l'aide d'un tableau initialisé a -1. Lors de la réunion de 2 ensemble on met à jour l'index de chaque ensemble :

– l'ensemble "absorbé" pointe sur la racine de l'arbre : permet facilement de remonter à la racine de l'arbre.
– l'ensemble "absorbant" a un index égal au cardinal de l'ensemble précédé du signe moins : permet de distinguer une racine et de connaître la taille de l'arbre

Les opérations deviennent alors :

– `find(y)` parcourir l'arbre du noeud `y` jusqu'à la racine de l'arbre
– `merge(x, y)` raccrocher l'arbre de racine `y` à l'arbre de racine `x`

Le coût global des 2 opérations peut être plus faible si on s'arrange pour que la hauteur des arbres ne croissent pas trop. On diminue la hauteur de l'arbre concaténé si on raccroche l'arbre de plus petite hauteur à l'arbre de plus grande hauteur : c'est la réunion pondérée.

La compression de chemin consiste à mettre à jour, lors de la recherche de la racine de l'arbre les indexes visités. On obtient ainsi la racine de l'arbre en 1 étape : c'est la compression de chemin.

La structure d'arbre permet d'utiliser les 2 techniques d'optimisations simultanément. On peut montrer, mais c'est difficile, que l'on obtient un algorithme linéaire en la taille des donnés.

Cette structure seule est très simple à mettre en oeuvre. C'est été un peu plus compliqué dans le cas des graphes hiérarchiques car il faut maintenir la cohérence entre les différents étages du graphe.

Avec cette structure et celle de graphe hiérarchique on peut implémenter : MInt, Segment, la fusion Mean Shift et l'élagage. En effet, on parcourt les arêtes d'adjacence du graphe, et on décide de fusionner 2 régions en utilisant le prédicat de fusion de l'algorithme. Elle permet aussi d'avoir une trace de la construction du graphe.

3.4 Classe d'interface et compteur de références en C++

Plusieurs classes réalisées sont structurées d'une manière décrite dans [15], [13] et [14]. Il s'agit de gérer une hiérarchie d'objet de manière transparente pour l'utilisateur du point de vue de la gestion de la mémoire. De plus, on optimise l'accès au données en cachant les classes à l'utilisateur et en utilisant un compteur de référence : les données ne sont représentées qu'une fois en mémoire.

Pour illustrer le fonctionnement de cette méthode on va décrire les classes `Color`, `Color_Base`, `Color_Color` et `Color_Gray` représentée à la figure 9.

Les 3 classes `Color_Base`, `Color_Color` et `Color_Gray` forment une hiérarchie d'objets. Dans ce cas `Color_Base` est abstraite mais ce n'est pas nécessaire. Comme dans n'importe quelle hiérarchie les objets dérivés peuvent redéfinir des méthodes. Cependant, on n'utilise pas directement cette hiérarchie.

C'est la classe `Color`, qui ne fait pas partie de la hiérarchie qui sert d'interface. Cette classe possède un pointeur sur `Color_Base` et donc, lorsqu'elle invoque une méthode elle profite du polymorphisme.

La classe `Color_Base` possède un compteur de références. C'est la classe d'interface `Color` qui met à jour ce compteur et qui, le cas échéant, libère la mémoire.

Pour l'utilisateur, la classe `Color` possède 2 constructeur :

```
Color (float)
```

pour un niveau de gris et :

```
Color (float, float, float, bool)
```

pour une couleur en mode `RGB` ou `HSV`

C'est dans ces 2 constructeurs que l'on crée dynamiquement une nouvelle instance soit de la classe `Color_Color` soit `Color_Gray` et que l'on initialise le compteur de référence.

La mise à jour du compteur de référence se fait dans 3 méthodes clés reproduites et expliquées ci-dessous. Tout d'abord, le constructeur de copie :

```
Color::Color(const Color& color){
    p = color.p;
    ++p->use;
}
```

on affecte à la copie le même pointeur que l'objet copier et il y a une référence de plus, la copie, sur l'objet.

Ensuite l'opérateur d'affectation :

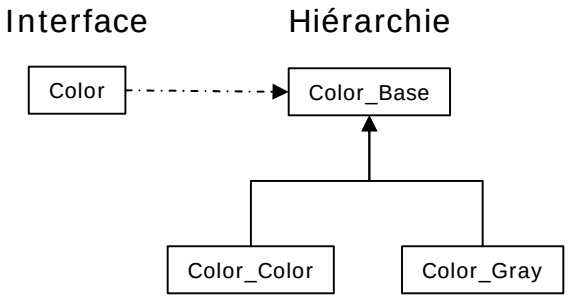


FIG. 9 – `Color`

```
Color& Color::operator=(const Color& rhs){
    rhs.p->use++;
    if(--p->use == 0){
        delete p;
    }
    p = rhs.p;
    return *this;
}
```

l'objet que l'on affecte a une référence de plus, celui que l'on que l'on pointait à une référence de moins. Si il a plus aucune référence on le libère. Finalement on affecte au pointeur l'objet pointé par l'objet affecté.

Et finalement le destructeur :

```
Color::~Color(){
    if(--p->use == 0){
        delete p;
    }
}
```

on décrémente le compteur et si l'on était le dernier objet à référencer l'objet pointé on le libère.

Pour que le tout soit opérationnel et utilisable, il faut encore que la classe d'interface ait une méthode correspondant à chaque méthode de la classe de base de la hiérarchie. Ainsi, on appelle la méthode sur l'interface qui redirige l'appel vers une instance réelle de la hiérarchie.

Bien qu'un peu lourde à mettre en place, cette manière de faire est très pratique car :

– gestion automatique de la mémoire, donc pas d'erreur
– efficacité : en réalité on manipule un pointeur
– lisibilité : à l'usage on a l'impression de manipuler des type de bases du langage

En plus de `Color`, j'ai utilisé cette méthode pour les types suivants : `Edge`, `FVector`, `Node`, `Pixel` et `PNG`:

3.5 Format `PNG`

Le format PNG (Portable Network Graphics, ou format Ping) est un format de fichier graphique bitmap (raster). Il a été mis au point en 1995 afin de fournir une alternative

libre au format GIF, format propriétaire dont les droits sont détenus par la société Unisys (propriétaire de l'algorithme de compression LZW), ce qui oblige chaque éditeur de logiciel manipulant ce type de format à leur verser des royalties. Ainsi PNG est également un acronyme récursif pour PNG's Not Gif.

Le format PNG permet de stocker des images en noir et blanc (jusqu'à 16 bits par pixels de profondeur de codage), en couleurs réelles (True color, jusqu'à 48 bits par pixels de profondeur de codage) ainsi que des images indexées, faisant usage d'une palette de 256 couleurs. De plus il supporte la transparence par couche alpha, c'est-à-dire la possibilité de définir 256 niveaux de transparence, tandis que le format GIF ne permet de définir qu'une seule couleur de la palette comme transparente. Il possède également une fonction d'entrelacement permettant d'afficher l'image progressivement.

La compression proposé, `zlib`, par ce format est une compression sans perte 5 à 25 % meilleure que la compression GIF. Enfin PNG embarque des informations sur le gamma de l'image, ce qui rend possible une correction gamma permet une indépendance vis-à-vis des périphériques d'affichage. Des mécanismes de correction d'erreurs sont également embarquées dans le fichier afin de garantir son intégrité.

Cependant, le brevet d'Unisys a pris fin en juin 2003 aux Etats-Unis (2004 pour certains pays d'Europe, d'Asie et le Canada), justifiant moins l'existence du PNG. Pourtant, selon le W3C, celui-ci est aujourd'hui parfaitement supporté par plus de 60 navigateurs, 180 visualiseurs d'images et plus de 100 éditeurs graphiques. Le PNG se retrouve donc aujourd'hui à armes égales avec ses concurrents pour s'imposer sur le Web.

On est donc dans une situation paradoxale où l'on parle de ce format comme si c'était de la musique d'avenir et qu'il n'arriverait pas à concurrencer GIF alors que concrètement on peut l'utiliser sur n'importe quel plateforme sans aucun problème ! Par contre, il existe très peu d'outil permettant leur manipulation en `C/C++`.

PNG est une spécification de format. La librairie quasi officielle qui essaye d'appliquer ces spécifications est la `libpng` et utilise la libraire de compression `zlib`. En tant que telle la libpng est une librairie de bas niveau : pour ouvrir un fichier on doit passer par toute une phase d'initialisation de structure, allocation de mémoire, etc. avant d'accéder au contenu de l'image, et l'utilisation en générale est assez ardue. On n'a pas d'interface de "haut niveau" dans cette librairie simplifiant les opérations.

On peut trouver des environnements telles que Java ou .NET, ainsi que les librairies Devil ou ImageMagick offrant de telles classes de haut niveau. Cependant, elles sont conçus dans une optique beaucoup plus large que le format `PNG`.

C'est pour ces raisons, que j'ai implémenté une hiérarchie de classes permettant soit de lire un fichier PNG, soit de créer un fichier au format PNG. A nouveau, ces classes n'ont comme fonctionnalité que ce qui est nécessaire pour produire les algorithmes de ce travail. D'autre part, la plupart des fonctions avancées ne sont pas supportée. Les classes implémentées permettent de gérer les modes niveaux de gris et "True color".

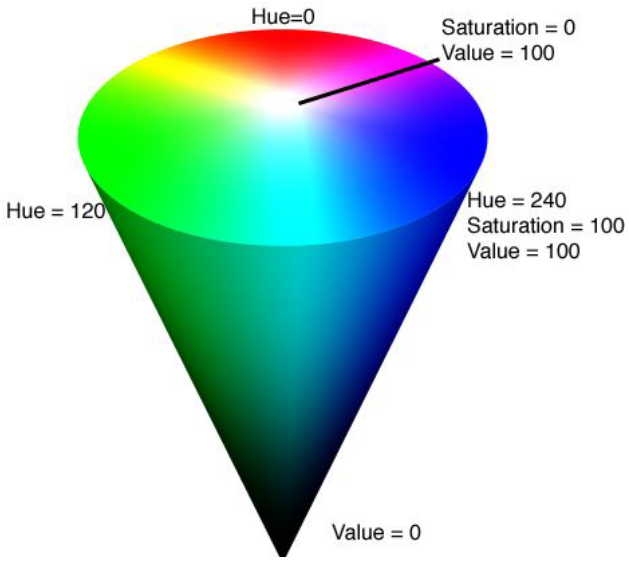


FIG. 10 – Cône HSV

3.6 Espaces de couleurs

Lorsque l'on analyse des images en "noir et blanc" ou niveau de gris, les différents algorithmes présentés ne posent pas de problèmes : les manipulations algébriques sur les niveaux de gris correspondent parfaitement à ce que l'on attend. Par contre, quand on utilise des images couleurs un problème se pose : qu'est-ce que la distance entre 2 couleurs ?

On rappelle brièvement que l'on obtient très facilement les composantes RGB d'une image pour des raisons historiques liées au développement du matériel. Cependant, l'approche d'un utilisateur est plutôt de décrire une couleur par : sa teinte, sa pureté et son éclat. C'est ainsi qu'on été inventé les espaces de couleurs perceptuelles.

Dans un de ces espaces, les couleurs sont représentées de manière compréhensible par un non technicien. Il existe beaucoup de tels espaces et en particulier : "PANTONE Color System", "Munsell Color System" et HSV (Hue, Saturation, Value). HSV est parfois appelé : HSI (Hue, Saturation, Intensity) ou HSB (Hue, Saturation, Brightness). Contrairement à ce qu'on peut croire on utilise beaucoup ce type d'espace : les "Color Picker" sont souvent basé sur eux !

Pour ce travail, on choisit l'espace HSV. Conceptuellement, cet espace est un cône (voir figure 10). Du côté circulaire du cône sont représentées les couleurs séparées par un angle pris de ligne 0 qui est le rouge. La saturation est la distance au centre du cercle : les couleurs hautement saturée sont à l'extérieure du cercle alors les tons gris sont proches du centres. Finalement, l'éclat est déterminé par la position verticale.

Cet espace résout le problème de distance précédemment cité. En effet, la distance entre 2 couleurs correspond à la perception visuelle de distance contrairement à l'espace RGB.

Un autre avantage, utilisé dans le cadre du Mean Shift, est qu'une image en niveau de gris est un cas particulier d'une image couleur, où seul la composante VALUE à un rôle et correspond au niveau de gris.

Les classes gérant les couleurs décrites précédemment permettent de passer d'une manière transparente d'une représentation à l'autre.

3.7 Utilisation du programme segment

Ci-dessous on a le "help" du programme. On y retrouve tous les algorithmes décrits dans ce travail. Pour chaque traitement le programme fournit une image de sortie ainsi qu'un fichier texte décrivant le déroulement de l'algorithme.

D'abord on a la liste des options possibles. Ensuite, pour chaque option on a la description des paramètres. Finalement, on quelques exemples d'utilisation.

```
$ ./segment.exe -help
segment.exe -input:<file>
            [-output:<file>]
            [-smooth:size,mu,sigma]

            [-kmean:<k>
            |-msfiltre:h,r]

            [-segment:<param1[,param2,..]>
            |-mint:<param1[,param2,..]>
            |-msfusion:h,r>]
            [-prune:<s>]


Paramètres :
==========
smooth : filtre gaussien
         size  : taille du masque (défaut 5)
         mu    : (défaut 0.0)
         sigma : (défaut 0.8)

kmean : algorithme des K-Means
        k : nombre de centre du K-Means

msfiltre : filtre Mean Shift
           h : rayon spatial de la fenêtre
           r : rayon dana l'espace des attributs de la fenêtre

segment : algorithme modifié
          tau 1 : constante de segmentation étape 1
          tau 2 : constante de segmentation étape 2
          ...

mint : algorithme original
       tau 1 : constante de segmentation étape 1
       tau 2 : constante de segmentation étape 2
       ...

msfusion : fusion Mean Shift
           h : rayon spatial de la fenêtre
           r : rayon dana l'espace des attributs de la fenêtre

prune : élagage
        s : taille minimale d'une région
```


FIG. 11 – Fausses couleur (palette "cool" de Matlab)

```
Remarque :
========
Si on ne donne pas l'option 'kmean' ou 'msfiltre', on commence
la segmentation avec un grid graphe.


Exemple :
=======

Mean Shift
segment.exe -input:Images/lena1.png -msfiltre:3,0.2 -msfusion:3,0.2 -prune:500
Image : Images/lena1.png
Filtre Mean Shift : paramètre spatial 3 et paramètre de couleurs 0.2
Fusion Mean Shift : paramètre spatial 3 et paramètre de couleurs 0.2
Elagage : régions de moins de 500 pixels

Segment
segment.exe -input:Images/lena1.png -segment:50
Image : Images/lena1.png
Segment : paramètre 50

MInt
segment.exe -input:Images/lena1.png -mint:10
Image : Images/lena1.png
MInt : paramètre 50

Filtre Mean Shift et Segment
segment.exe -input:Images/lena1.png -msfiltre:3,0.2 -segment:50
Image : Images/lena1.png
Filtre Mean Shift : paramètre spatial 3 et paramètre de couleurs 0.2
Segment : paramètre 50
```

3.8 Images de sortie

La structure de graphe décrite précédemment est représentable graphiquement : on parcourt la pyramide obtenue à partir d'un niveau souhaité. Voici les différents types d'images produites.

La figure 11 représente les différents composants trouver lors de la segmentation dans une palette de fausses couleurs. Le nombre de couleurs étant limité 2 régions qui n'ont aucun rapport peuvent avoir la même couleurs.

La figure 12 représente aussi les différents composants trouver lors de la segmentation.

FIG. 12 – Couleurs ”moyennes”

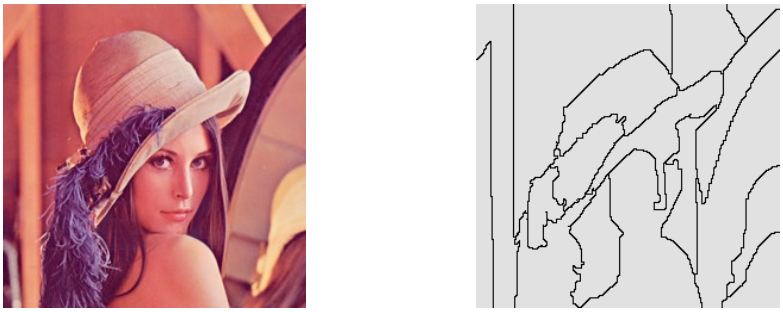


FIG. 13 – Bords des régions

Cependant, on utilise une couleur calculée lors de la fusion de 2 régions (en général la moyenne des couleurs pondérée par la taille des régions).

La figure 13 montre le bords des régions.

Ces 3 images sont produites par le programme lors du traitement d'une image. Elles permettent chacune d'apprécier le résultats sous un angle légèrement et ce qui n'apparaît pas forcément dans une est visible dans une autre.

La figure 14 montre finalement l'ensemble de la structure de graphe au format `Post-script`. Chaque sommet du graphe est représenté par un petit disque de couleur. Cette couleur peut-être celle de la palette ou calculée. De plus, les arêtes d'adjacence sont représentés par un segment de droite entre sommets.

Ce type d'image est plutôt destiné à la mise au point et à la compréhension des algorithmes. En le modifiant un peu, on peut faire apparaître l'arbre de recouvrement minimal de chaque région.

FIG. 14 – Graphe d'adjacence au format `Postscript`

4 Résultats

Pour l'évaluation des résultats aucune procédure n'a été établie. C'est donc sur un petit nombre d'image traitées systématiquement et les essais menés au cours du travail qu'on tire des conclusions.

Du point de vue des performances, on demande que le programme soit relativement efficace et supporte la montée en charge, ce qui est le cas. C'est l'utilisation du langage `C++` qui le permet. On profite également de la portabilité de ce langage pour obtenir soit un exécutable Windows soit un exécutable Unix.

Si on veut de meilleures des performances en temps on peut trouver des implémentations sur les sites des auteurs des algorithmes, mais je trouve qu'ils sont d'une facilité de lecture très douteuse et absolument pas évolutif : les algorithmes sont complètement figés dans différentes partie de code.

En ce qui concerne l'algorithme MInt et segmentation Mean Shift, les résultats présentés reproduisent ceux que l'on obtient avec les programmes originaux. Il faut quand même faire attention aux paramètres qui n'ont pas le même domaine de définition. On peut considérer que cet objectif est atteint.

On rappelle que l'objectif du programme est de fournir une segmentation robuste : lorsque les paramètres sont fixés, on veut obtenir des segmentations "semblables" pour toutes les images. Dans ce domaine, je trouve que l'algorithme Segment, la version modifiée du MInt donne les meilleures résultats.

On donne aussi les résultats des K-Means et du filtre Mean Shift. Ce ne sont pas des segmentations mais cela montre le fonctionnement de ces algorithmes qui ont leur propre signification indépendamment du partitionnement d'images.

Les résultats ont été obtenus en traitant systématiquement les images suivantes :

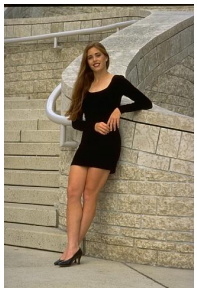
- image d'une jeune femme appuyé contre une rampe d'escalier, provient du benchmark de Berkeley
- image un peu sexy (couleur et niveau de gris) d'une jeune argentine, c'est la plus grande image de la série
- la fameuse photo de Lena (couleur et niveau de gris)
- image (couleur et niveau de gris) d'une camionnette sur une route
- image synthétique avec le dégradé et un rectangle contenant un rectangle "agité"

4.1 Segment

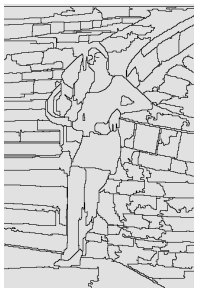
L'image toute à gauche est l'original. On applique l'algorithme plusieurs fois indépendamment les une des autres avec différents paramètres ce qui donne les images de la première ligne. Ensuite, on applique l'algorithme avec différents paramètres mais en se basant sur le résultat précédent : utilisation de la pyramide de segmentation, ce qui donne les images de la seconde lignes.

On remarque tout de suite que la l'utilisation de la pyramide donne plus de régions.

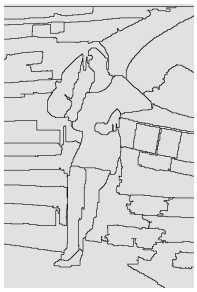
Si on regarde une série de résultats en colonnes, on voit que la segmentation est régulière. Ce qui veut dire qu'on a effectivement un algorithme robuste. Dans ce domaine ce sont vraiment les meilleures résultats.



321 × 481



Paramètre : 10
Composants : 145



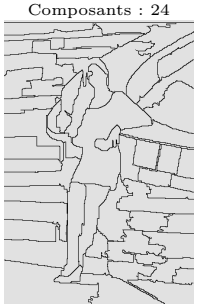
Paramètre : 50
Composants : 43

Paramètre : 100
Composants : 24

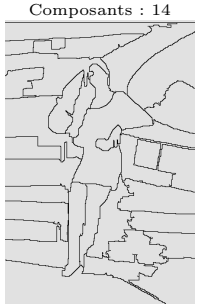
Paramètre : 200
Composants : 14

Paramètre : 10
Composants : 145

Paramètre : 50
Composants : 101

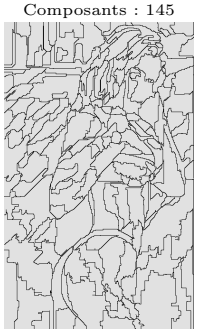


Paramètre : 100
Composants : 60

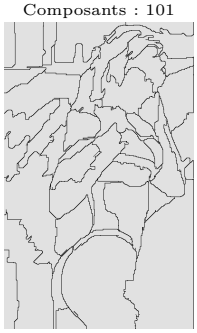


Paramètre : 200
Composants : 35

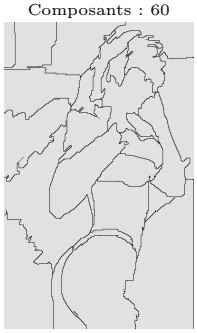
400 × 647



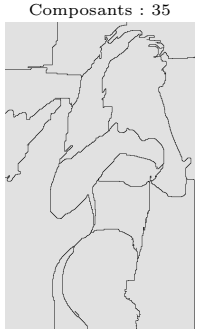
Paramètre : 10
Composants : 200



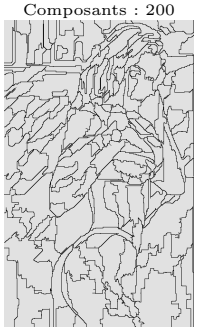
Paramètre : 50
Composants : 49



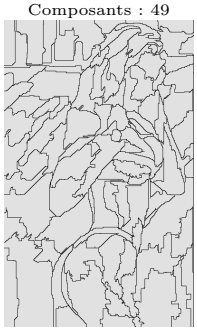
Paramètre : 100
Composants : 27



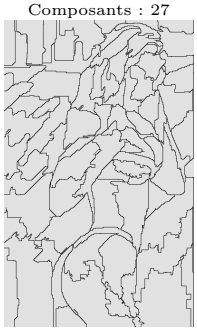
Paramètre : 200
Composants : 15



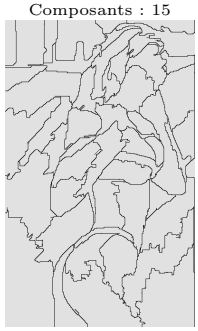
Paramètre : 10
Composants : 200



Paramètre : 50
Composants : 111

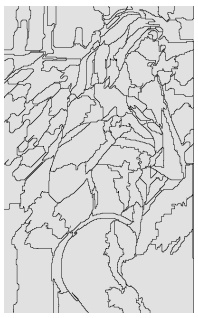


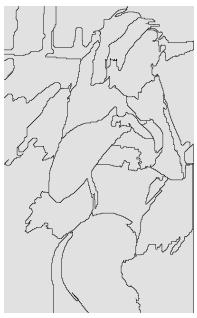
Paramètre : 100
Composants : 84

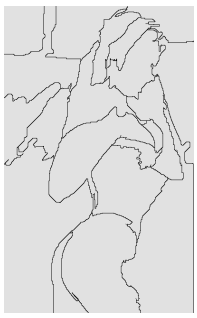


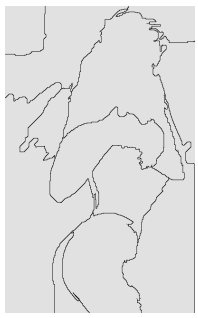
Paramètre : 200
Composants : 52

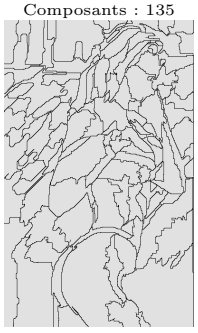

400 × 647

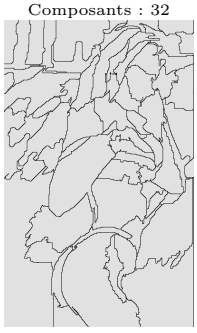

Paramètre : 10
Composants : 135

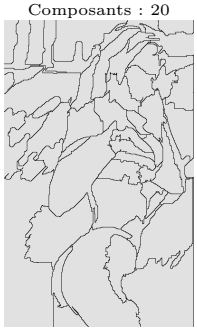

Paramètre : 50
Composants : 32

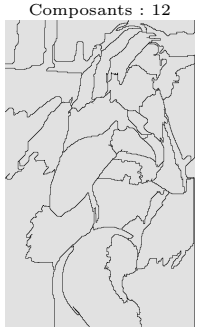

Paramètre : 100
Composants : 20


Paramètre : 200
Composants : 12

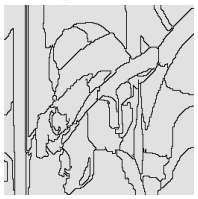

Paramètre : 10
Composants : 135

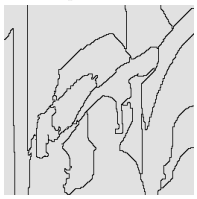

Paramètre : 50
Composants : 69

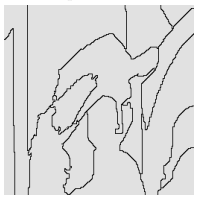

Paramètre : 100
Composants : 59

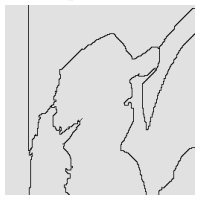

Paramètre : 200
Composants : 37

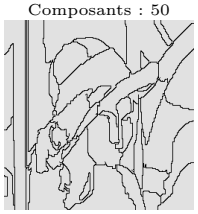

255 × 255

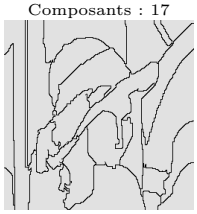

Paramètre : 10
Composants : 50

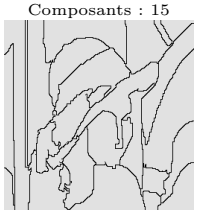

Paramètre : 50
Composants : 17

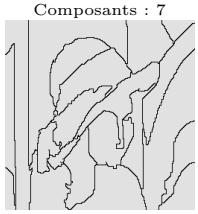

Paramètre : 100
Composants : 15


Paramètre : 200
Composants : 7

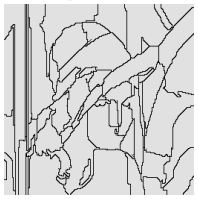

Paramètre : 10
Composants : 50

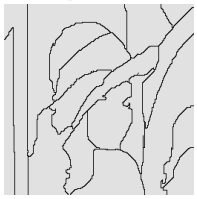

Paramètre : 50
Composants : 28

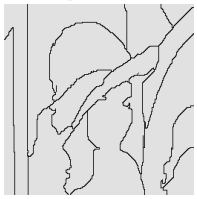

Paramètre : 100
Composants : 28

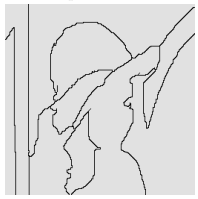

Paramètre : 200
Composants : 21

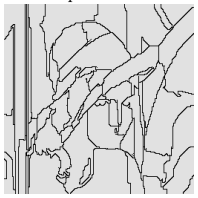

256 × 256

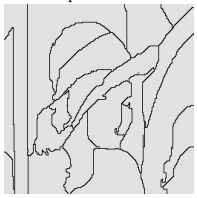

Paramètre : 10
Composants : 66

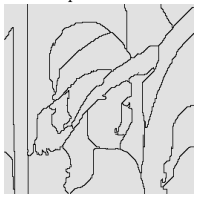

Paramètre : 50
Composants : 18

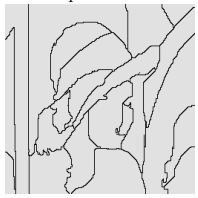

Paramètre : 100
Composants : 16

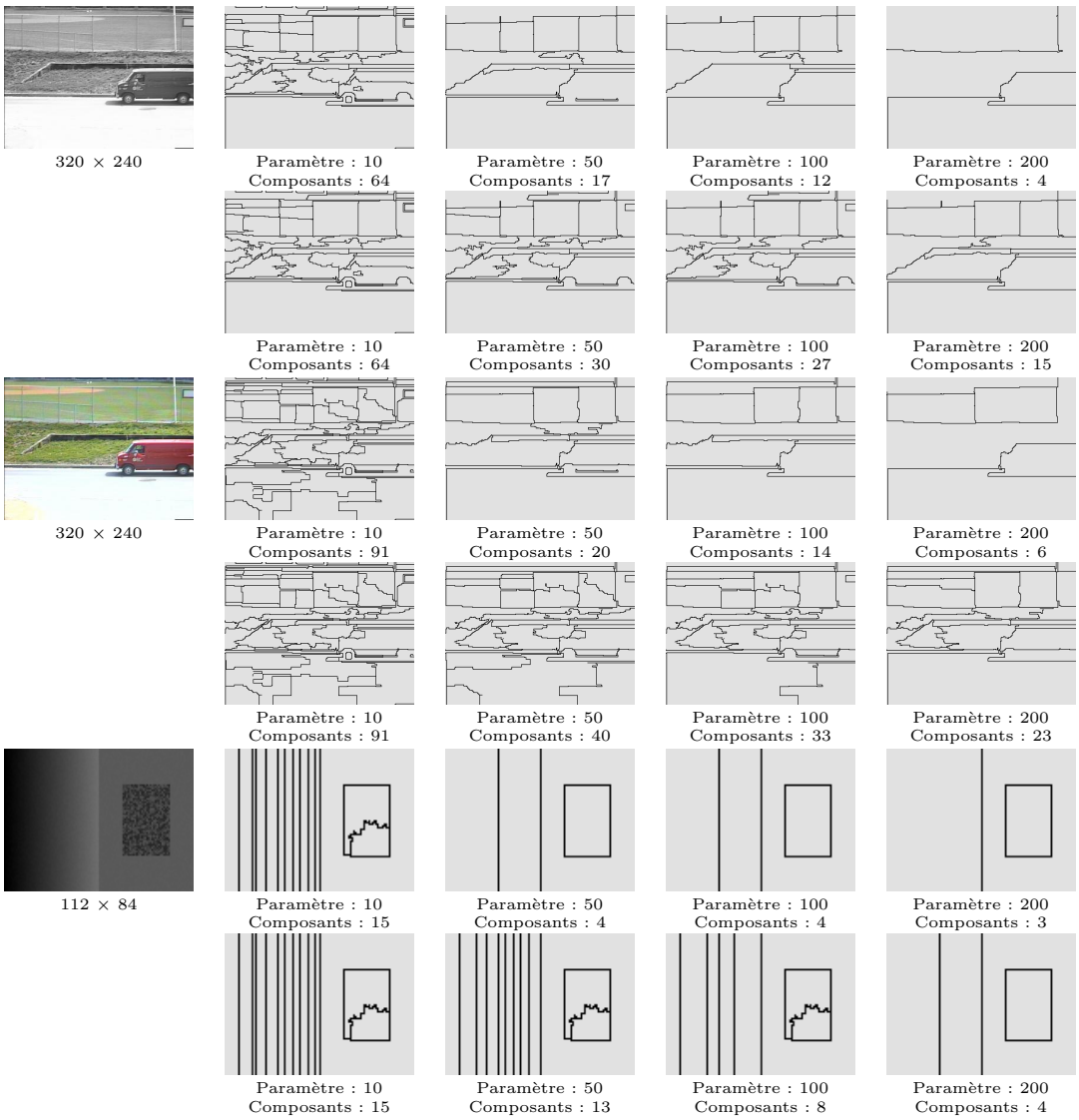

Paramètre : 200
Composants : 10


Paramètre : 10
Composants : 66


Paramètre : 50
Composants : 22


Paramètre : 100
Composants : 22


Paramètre : 200
Composants : 22

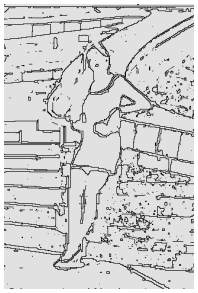


4.2 MInt

Cette méthode qui peut aussi être utilisée de manière pyramidale reprend la même disposition que Segment.

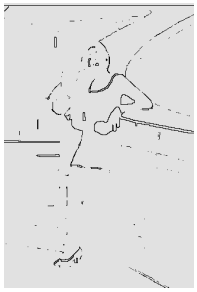
Cette fois, les résultats ont l'air beaucoup moins bon : lorsqu'on augmente la valeur du paramètre des régions entières disparaissent. En fait, si on paramétre chaque image une à une, on peut aussi obtenir de très bon résultats. Cette méthode n'est par robuste dans le sens que l'on souhaite.

321 × 481

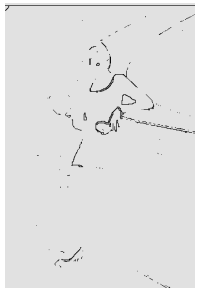


Paramètre : 1
Composants : 889

Paramètre : 5
Composants : 312



Paramètre : 10
Composants : 165



Paramètre : 20
Composants : 123

Paramètre : 1
Composants : 889

Paramètre : 5
Composants : 362

Paramètre : 10
Composants : 307

Paramètre : 20
Composants : 270

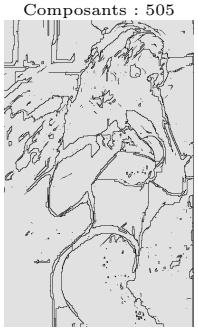
400 × 647

Paramètre : 1
Composants : 505

Paramètre : 10
Composants : 149

Paramètre : 20
Composants : 130

Paramètre : 50
Composants : 42



Paramètre : 1
Composants : 505

Paramètre : 10
Composants : 286

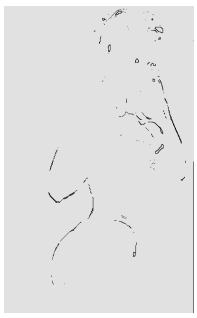
Paramètre : 20
Composants : 259

Paramètre : 50
Composants : 245

400 × 647

Paramètre : 1
Composants : 564

Paramètre : 10
Composants : 195



Paramètre : 20
Composants : 112

Paramètre : 50
Composants : 56

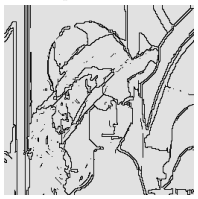
Paramètre : 1
Composants : 564

Paramètre : 10
Composants : 356

Paramètre : 20
Composants : 326

Paramètre : 50
Composants : 321

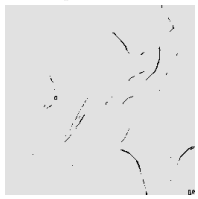
255 × 255



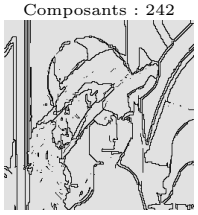
Paramètre : 1
Composants : 242

Paramètre : 5
Composants : 110

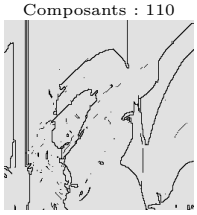
Paramètre : 10
Composants : 74



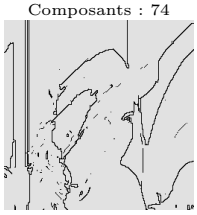
Paramètre : 20
Composants : 62



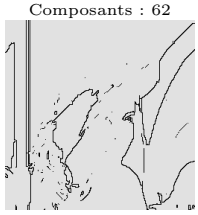
Paramètre : 1
Composants : 242



Paramètre : 5
Composants : 132



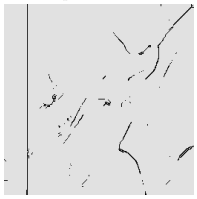
Paramètre : 10
Composants : 123



Paramètre : 20
Composants : 111

256 × 256

Paramètre : 1
Composants : 200

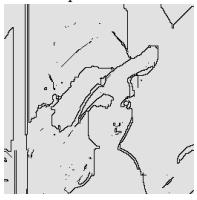


Paramètre : 10
Composants : 69

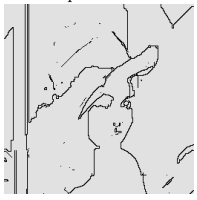
Paramètre : 20
Composants : 42

Paramètre : 50
Composants : 32

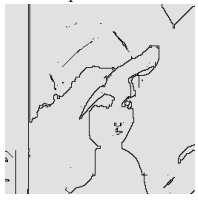
Paramètre : 1
Composants : 200



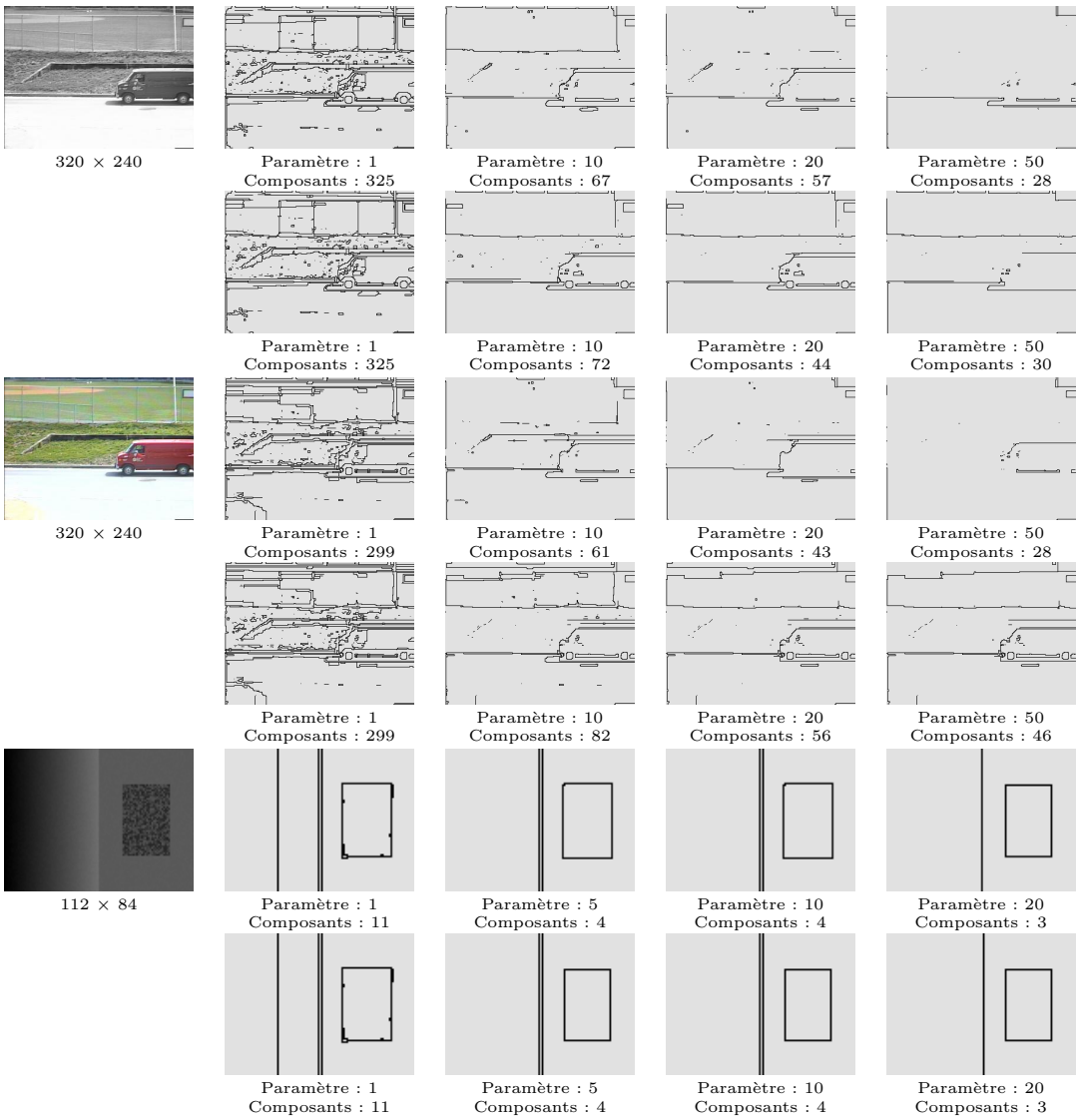
Paramètre : 10
Composants : 84



Paramètre : 20
Composants : 67



Paramètre : 50
Composants : 54



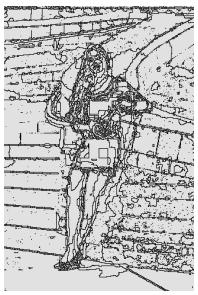
4.3 Segmentation Mean Shift

Pour cette méthode, on reprend la même disposition, mais en colonnes on fait varier le paramètre de la fenêtre de recherche selon les attributs et en ligne on fait varier le paramètre spatiales.

On remarque que l'algorithme est beaucoup plus sensible à la variation de premier paramètres que du second.

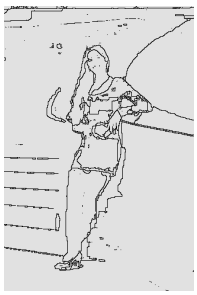
Sinon, on peut tirer les mêmes conclusions que pour le MInt : les résultats ont l'air moyen et très variables. A nouveau, si on paramétre chaque image on peut obtenir de très bons résultats, mais ça n'est pas robuste.

321 × 481

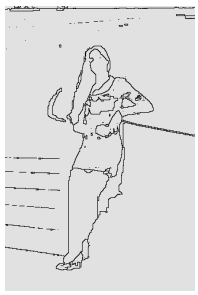


Paramètre : 3,0.05
Composants : 6694

Paramètre : 3,0.1
Composants : 1561



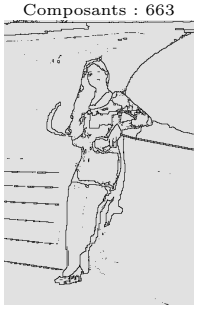
Paramètre : 3,0.15
Composants : 663



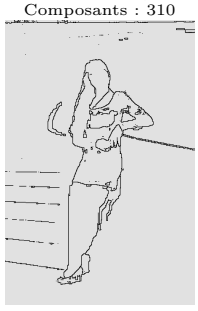
Paramètre : 3,0.2
Composants : 310

Paramètre : 5,0.05
Composants : 6860

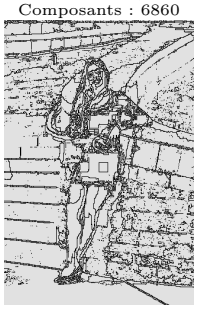
Paramètre : 5,0.1
Composants : 1646



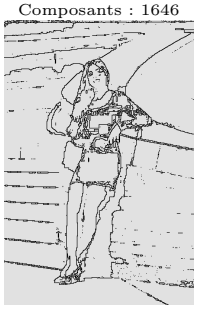
Paramètre : 5,0.15
Composants : 692



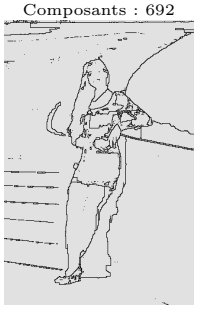
Paramètre : 5,0.2
Composants : 313



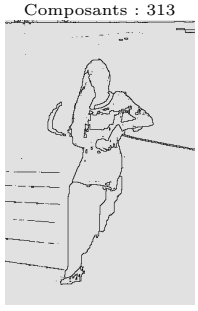
Paramètre : 7,0.05
Composants : 6951



Paramètre : 7,0.1
Composants : 1648



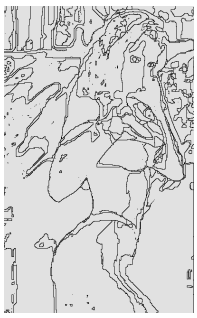
Paramètre : 7,0.15
Composants : 679



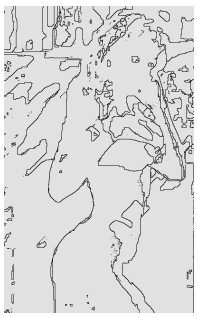
Paramètre : 7,0.2
Composants : 319

400 × 647

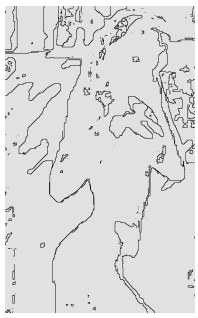
Paramètre : 3,0.05
Composants : 4301



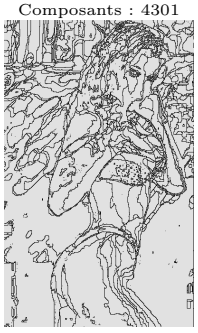
Paramètre : 3,0.1
Composants : 1066



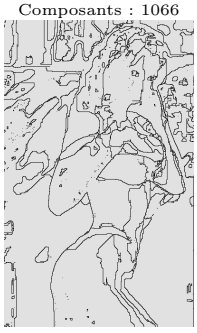
Paramètre : 3,0.15
Composants : 602



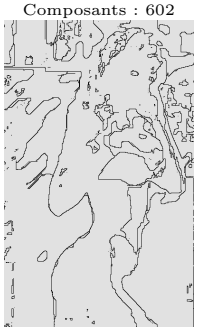
Paramètre : 3,0.2
Composants : 273



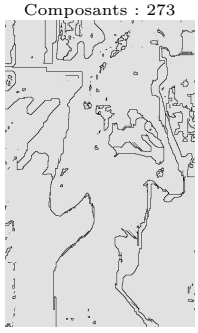
Paramètre : 5,0.05
Composants : 4317



Paramètre : 5,0.1
Composants : 1191

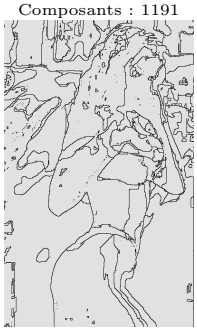


Paramètre : 5,0.15
Composants : 598

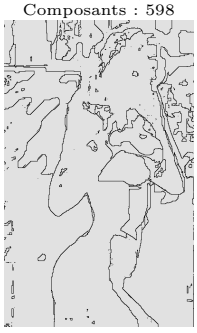


Paramètre : 5,0.2
Composants : 251

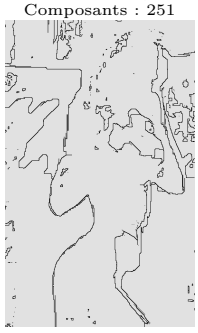
Paramètre : 7,0.05
Composants : 4417



Paramètre : 7,0.15
Composants : 617



Paramètre : 7,0.2
Composants : 245



400 × 647

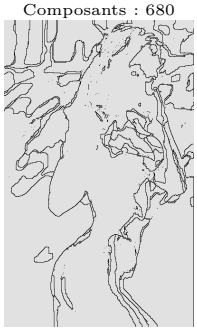
Paramètre : 3,0.05
Composants : 2654

Paramètre : 3,0.1
Composants : 680

Paramètre : 3,0.15
Composants : 290

Paramètre : 3,0.2
Composants : 93

Paramètre : 5,0.05
Composants : 2977

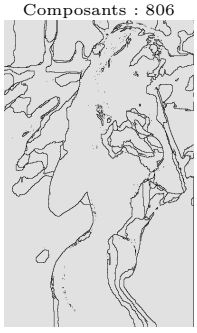


Paramètre : 5,0.1
Composants : 806

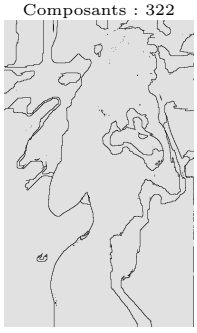
Paramètre : 5,0.15
Composants : 322

Paramètre : 5,0.2
Composants : 40

Paramètre : 7,0.05
Composants : 3151



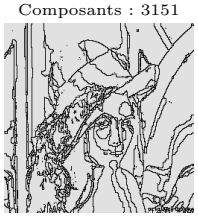
Paramètre : 7,0.1
Composants : 879



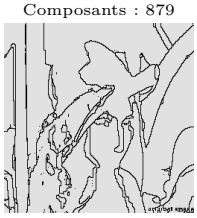
Paramètre : 7,0.15
Composants : 323

Paramètre : 7,0.2
Composants : 78

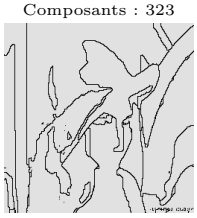
255 × 255



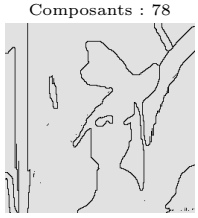
Paramètre : 3,0.05
Composants : 1918



Paramètre : 3,0.1
Composants : 420

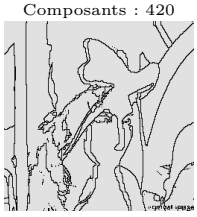


Paramètre : 3,0.15
Composants : 147

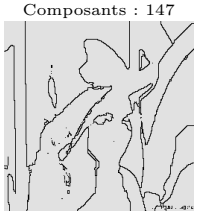


Paramètre : 3,0.2
Composants : 37

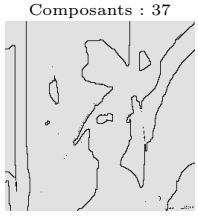
Paramètre : 5,0.05
Composants : 2059



Paramètre : 5,0.1
Composants : 465

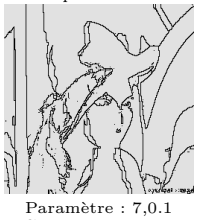


Paramètre : 5,0.15
Composants : 148

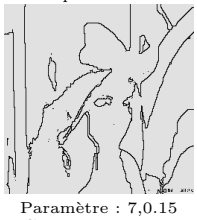


Paramètre : 5,0.2
Composants : 49

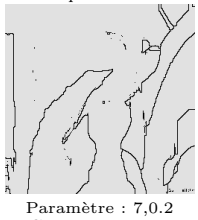
Paramètre : 7,0.05
Composants : 2167



Paramètre : 7,0.1
Composants : 469

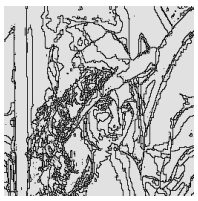


Paramètre : 7,0.15
Composants : 154

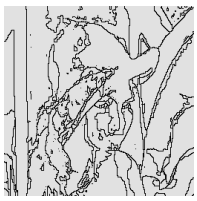


Paramètre : 7,0.2
Composants : 72

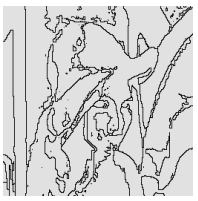
256 × 256



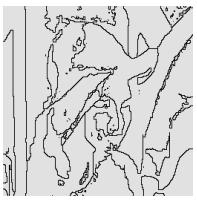
Paramètre : 3,0.05
Composants : 2628



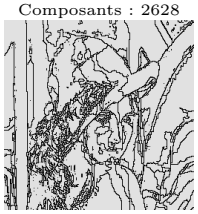
Paramètre : 3,0.1
Composants : 611



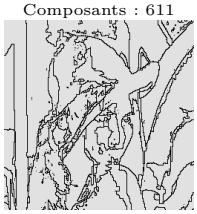
Paramètre : 3,0.15
Composants : 226



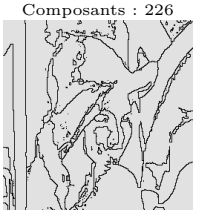
Paramètre : 3,0.2
Composants : 136



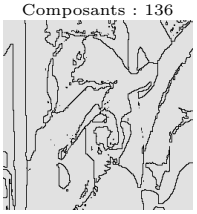
Paramètre : 5,0.05
Composants : 2652



Paramètre : 5,0.1
Composants : 632

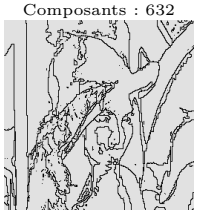


Paramètre : 5,0.15
Composants : 213

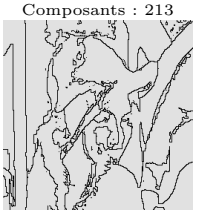


Paramètre : 5,0.2
Composants : 137

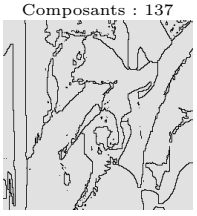
Paramètre : 7,0.05
Composants : 2628



Paramètre : 7,0.1
Composants : 618

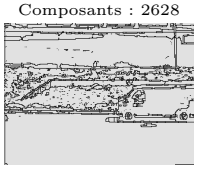


Paramètre : 7,0.15
Composants : 221

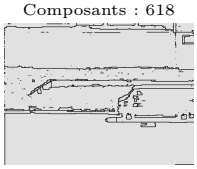


Paramètre : 7,0.2
Composants : 138

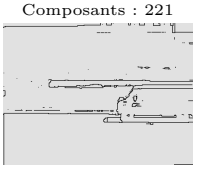
320 × 240



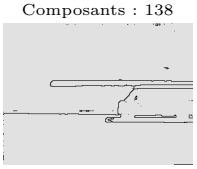
Paramètre : 3,0.05
Composants : 1452



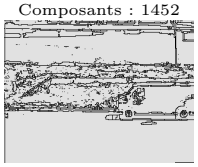
Paramètre : 3,0.1
Composants : 344



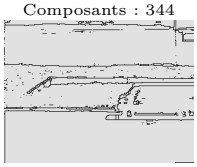
Paramètre : 3,0.15
Composants : 129



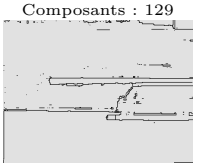
Paramètre : 3,0.2
Composants : 64



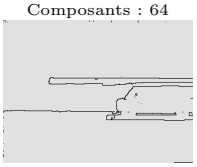
Paramètre : 5,0.05
Composants : 1591



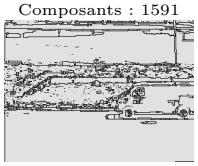
Paramètre : 5,0.1
Composants : 401



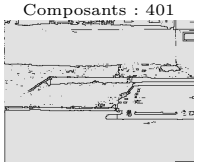
Paramètre : 5,0.15
Composants : 153



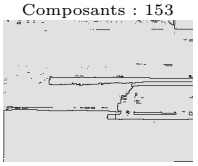
Paramètre : 5,0.2
Composants : 49



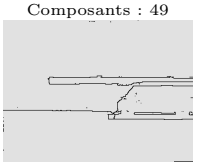
Paramètre : 7,0.05
Composants : 1700



Paramètre : 7,0.1
Composants : 463

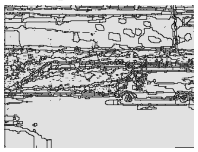


Paramètre : 7,0.15
Composants : 174

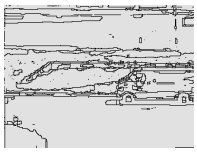


Paramètre : 7,0.2
Composants : 52

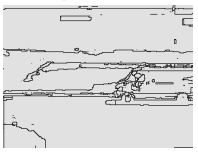
320 × 240



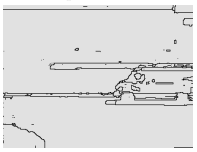
Paramètre : 3,0.05
Composants : 2932



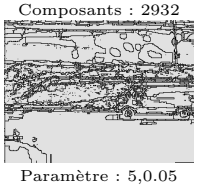
Paramètre : 3,0.1
Composants : 706



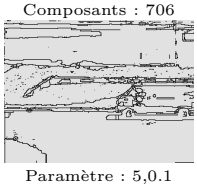
Paramètre : 3,0.15
Composants : 270



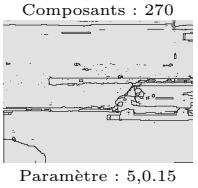
Paramètre : 3,0.2
Composants : 176



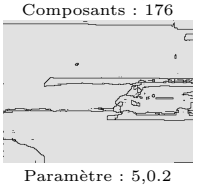
Paramètre : 5,0.05
Composants : 3031



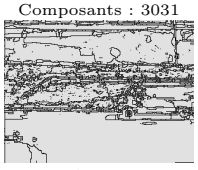
Paramètre : 5,0.1
Composants : 779



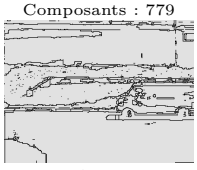
Paramètre : 5,0.15
Composants : 323



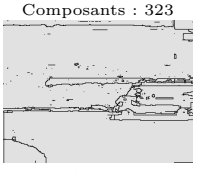
Paramètre : 5,0.2
Composants : 158



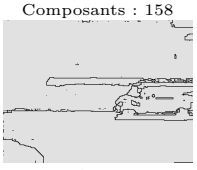
Paramètre : 7,0.05
Composants : 2983



Paramètre : 7,0.1
Composants : 772

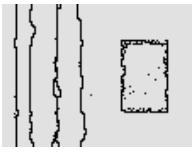


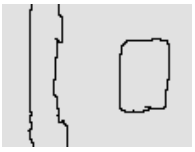
Paramètre : 7,0.15
Composants : 326



Paramètre : 7,0.2
Composants : 187

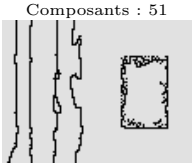

112 × 84

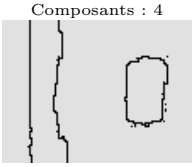

Paramètre : 3,0.05
Composants : 51

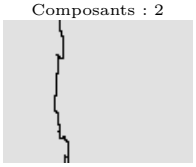

Paramètre : 3,0.1
Composants : 4

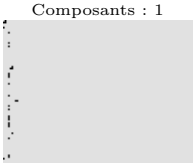

Paramètre : 3,0.15
Composants : 2

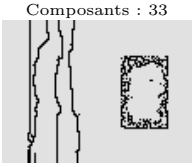
Paramètre : 3,0.2
Composants : 1

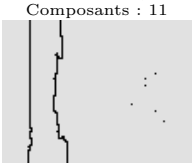

Paramètre : 5,0.05
Composants : 33

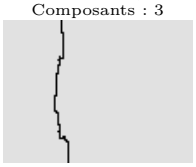

Paramètre : 5,0.1
Composants : 11

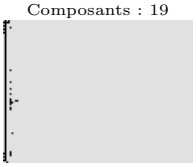

Paramètre : 5,0.15
Composants : 3


Paramètre : 5,0.2
Composants : 19


Paramètre : 7,0.05
Composants : 36


Paramètre : 7,0.1
Composants : 9

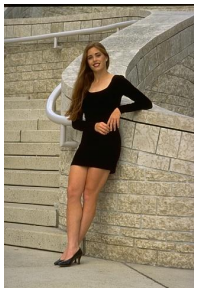

Paramètre : 7,0.15
Composants : 2


Paramètre : 7,0.2
Composants : 21

4.4 KMean

L'image de gauche est l'originale et les suivantes donnent le résultat de l'application de l'algorithme pour : 2, 4, 8 et 16 centres.

Pour $k = 2$ on a une sorte de binarisation de l'images. Pour $k = 16$, bien que dégradée l'image ressemble fortement à l'original : on palettise l'image en 16 couleurs.


321 × 481


Paramètre : 2


Paramètre : 4


Paramètre : 8

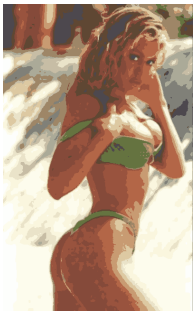

Paramètre : 16

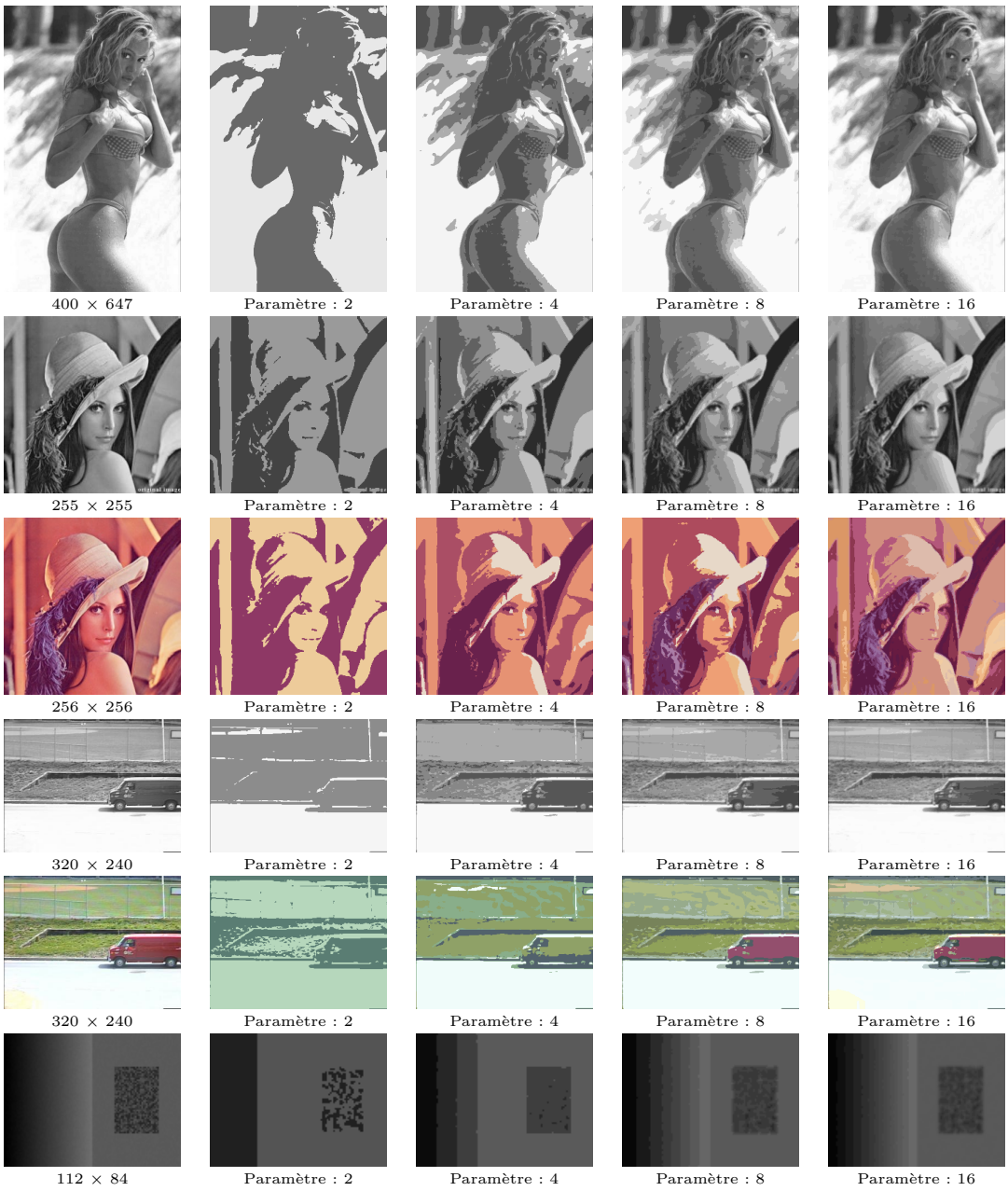

400 × 647


Paramètre : 2


Paramètre : 4


Paramètre : 8

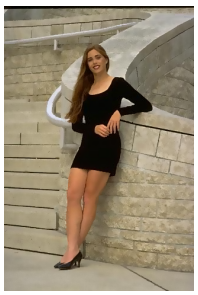

Paramètre : 16

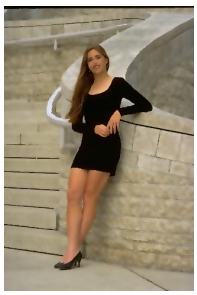


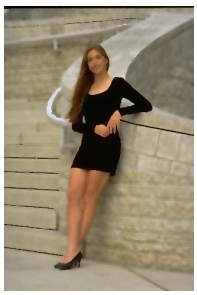
4.5 Filtre Mean Shift

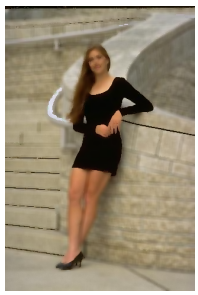
Ces résultats reprennent la disposition de la segmentation Mean Shift (ils les précédent dans l'algorithme). On voit que filtre lisse l'image tout en préservant les arrêtes. Lorsque l'on augment le paramètre des couleurs le lissage est plus important et un flou s'introduit. Si le paramètre spatial augment on a plus de flou.


321 × 481


Paramètre : 3,0.1


Paramètre : 3,0.2

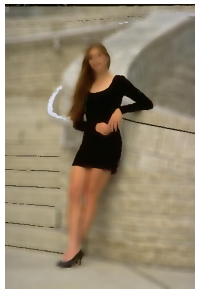

Paramètre : 3,0.3


Paramètre : 3,0.4


Paramètre : 5,0.1


Paramètre : 5,0.2

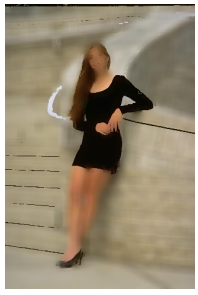

Paramètre : 5,0.3


Paramètre : 5,0.4


Paramètre : 7,0.1


Paramètre : 7,0.2


Paramètre : 7,0.3


Paramètre : 7,0.4

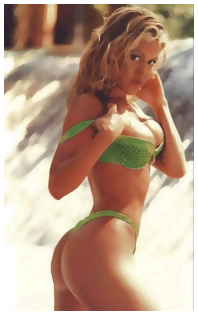

400 × 647


Paramètre : 3,0.1


Paramètre : 3,0.2

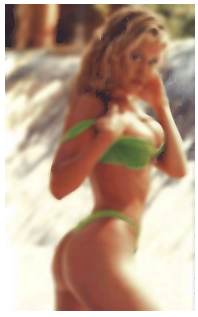

Paramètre : 3,0.3

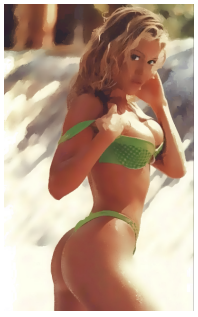

Paramètre : 3,0.4

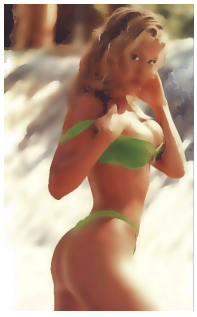

Paramètre : 5,0.1


Paramètre : 5,0.2

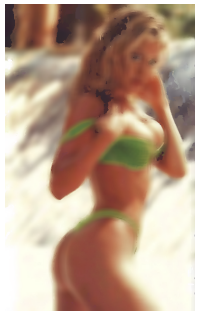

Paramètre : 5,0.3


Paramètre : 5,0.4


Paramètre : 7,0.1


Paramètre : 7,0.3


Paramètre : 7,0.4




400 × 647


Paramètre : 3,0.1


Paramètre : 3,0.2


Paramètre : 3,0.3


Paramètre : 3,0.4


Paramètre : 5,0.1


Paramètre : 5,0.2


Paramètre : 5,0.3


Paramètre : 5,0.4


Paramètre : 7,0.1


Paramètre : 7,0.2


Paramètre : 7,0.3


Paramètre : 7,0.4


255 × 255


Paramètre : 3,0.1


Paramètre : 3,0.2


Paramètre : 3,0.3


Paramètre : 3,0.4


Paramètre : 5,0.1


Paramètre : 5,0.2

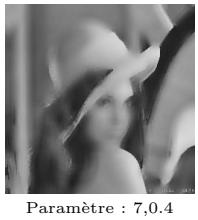

Paramètre : 5,0.3


Paramètre : 5,0.4


Paramètre : 7,0.1

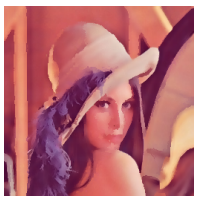

Paramètre : 7,0.2

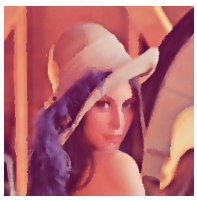

Paramètre : 7,0.3

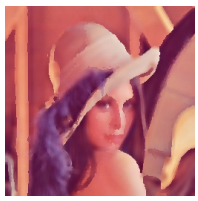

Paramètre : 7,0.4

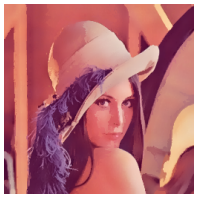

256 × 256

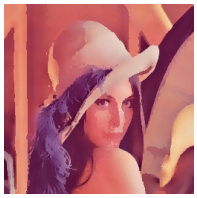

Paramètre : 3,0.1

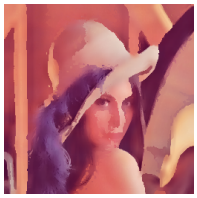

Paramètre : 3,0.2

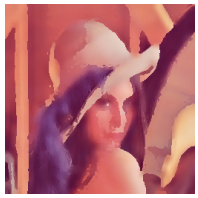

Paramètre : 3,0.3

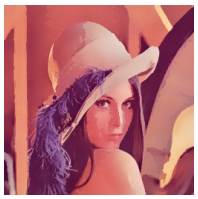

Paramètre : 3,0.4

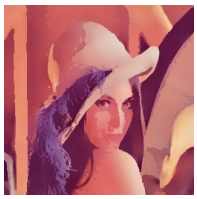

Paramètre : 5,0.1

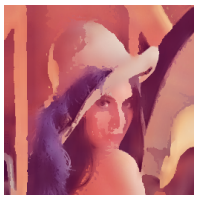

Paramètre : 5,0.2

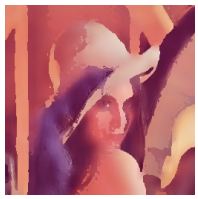

Paramètre : 5,0.3


Paramètre : 5,0.4


Paramètre : 7,0.1


Paramètre : 7,0.2


Paramètre : 7,0.3


Paramètre : 7,0.4


320 × 240


Paramètre : 3,0.1


Paramètre : 3,0.2


Paramètre : 3,0.3


Paramètre : 3,0.4


Paramètre : 5,0.1


Paramètre : 5,0.2


Paramètre : 5,0.3


Paramètre : 5,0.4


Paramètre : 7,0.1


Paramètre : 7,0.2


Paramètre : 7,0.3


Paramètre : 7,0.4


320 × 240


Paramètre : 3,0.1


Paramètre : 3,0.2


Paramètre : 3,0.3


Paramètre : 3,0.4


Paramètre : 5,0.1


Paramètre : 5,0.2


Paramètre : 5,0.3


Paramètre : 5,0.4


Paramètre : 7,0.1

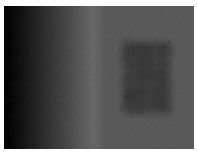

Paramètre : 7,0.2

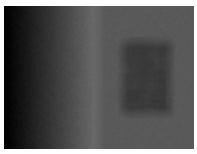

Paramètre : 7,0.3


Paramètre : 7,0.4


112 × 84

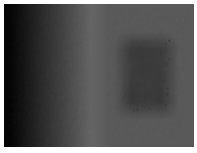

Paramètre : 3,0.1


Paramètre : 3,0.2


Paramètre : 3,0.3


Paramètre : 3,0.4


Paramètre : 5,0.1


Paramètre : 5,0.2


Paramètre : 5,0.3


Paramètre : 5,0.4


Paramètre : 7,0.1


Paramètre : 7,0.2


Paramètre : 7,0.3


Paramètre : 7,0.4

4.6 Environnement

Ce projet a été réalise dans l'environnement suivant :
- Système d'exploitations : `Windows 2000 Advanced Server Ver 5.0 Build 2195 Service Pack 3`
- L'environnement d'émulation `Unix` : `Cygwin` version `CYGWIN_NT-5.0`
- L'environnement de développement `Eclipse` version `2.1.0`
- Le plugin pour `Eclipse` de développement en `C/C++` : `CDT` version `1.1.0`
- Le compilateur `C++` : `gcc` version `3.2-3`
- Le débogueur : `gdb` version `20030303-1`

Il a aussi été testé sur les machines linux du CUI ainsi que sur VIPER.

Références

[1] Pedro F. Felzenszwalb, Daniel P. Huttenlocher
Efficient Graph-Based Image Segmentation

[2] Pedro F. Felzenszwalb, Daniel P. Huttenlocher
Image Segmentation Using Local Variation

[3] Ian H. Witten, Eibe Frank
Data Mining Morgan Kaufmann, 2000

[4] Abderrahim Labbi, Christian Pellegrini
Cours d'intelligence artificielle

[5] Dorin Comaniciu, Peter Meer
Mean Shift : A Robust Approach Toward Feature Space Analysis
IEEE, Vol 24 No 5 May 2002

[6] Dorin Comaniciu, Peter Meer
Mean Shift Analysis and Applications

[7] Dorin Comaniciu, Peter Meer
Distribution Free Decomposition of Multivariate Data

[8] Dorin Comaniciu, Peter Meer
Robust Analysis of Feature Spaces : Color Image Segmentation

[9] Yakov Keselman, Evangelia Micheli-Tzanakou
Extraction and characterisation of regions of interest in biomedical images

[10] Jianbo Shie, Jitendra Malik
Normalized Cuts and Image Segmentation

[11] Chad Carson, Serge Belongie, Hayit Greenspan, Jitendra Malik
Blobworld: Image segmentation using Expectation-Maximization and its application to image querying

[12] Nino Silverio
Langage C++
Eyrolles, 1996

[13] Andrew Koenig, Barbara E. Moo
Ruminations on C++
Addison-Wesley, 1997

[14] Andrew Koenig, Barbara E. Moo
Accelerated C++
Addison-Wesley, 2000

[15] Bjarne Stroustrup
Le langage C++
Addison-Wesley, 1992