LANCE Florian : n°10604720
MATHEOSSIAN Dimitri : n°10604950

Rapport MIF23: Analyse Region Growing

Sommaire

Nous avons utilisé OpenCv pour ouvrir/parcourir/modifier/afficher et enregistrer des images.
Dans le rapport, vous pourrez suivre les redirections vers d'autres parties en cliquant directement sur les numéros (exemple : « détaillé dans la partie 3.b »)

1. Les classes

Nous avons créé 4 classes, voici quelques-uns de leurs attributs :

- Graine : ses coordonnées
- Région :
  - Un index unique (comme une clé primaire)
  - Une couleur pour l'agglomération
  - Une couleur de visualisation pour l'image finale
- Accroissement :
  - La liste de toutes les régions créées (dans l'ordre)
  - Une matrice d'entier de taille N*M
- Couleur : trois composantes : R, V, B
- Histogramme (le but de cette classe est détaillé dans la partie 3.b)

On part de plusieurs Graines que l'on va répartir sur l'image, chacune d'entre elles appartiendra à une Région (au départ une région à une taille d'un pixel).

Il faut ensuite une structure qui conservera la liste des régions et qui les fera croitre et fusionner, on l'appellera Accroissement.

Et pour que ces régions puissent s'étendre correctement, il faudra stocker un ou plusieurs critères qui vont les différencier les unes des autres, comme un index et une couleur, d'où la présence de la classe Couleur.

La classe Graines :
Concrètement, on pourrait s'en passer et stocker uniquement un CvPoint dans la classe Région. Nous l'avons tout de même conservé dans le cas où nos graines auraient besoins d'informations spécifiques à leur point de départ.

La classe Région :
Elle fait partie intégrante de l'algorithme, elle permet de stocker la couleur qui va être mise à jour après chaque agglomération de point ou fusion de région.
La valeur d'index qui lui est attribué est unique, elle provient d'une variable statique qui s'incrémente à chaque nouvelle région.

La classe Accroissement :
La matrice d'entiers : elle a la même taille que l'image source à segmenter, elle permet de stocker l'appartenance de chaque pixel à une région.

Matrice « imgIndex » :

	x					
y	0	0	0	1	2	Région 0 : rouge, Région 1 : vert, Région 2 : bleu.
	0	1	1	1	2	
	0	0	1	1	2	
	0	1	1	1	2	
	0	0	2	1	1	
	2	2	2	2	2	

Carte des redirections (*map<int,int>*) : si une région est agglomérée, on saura quelle est la région qui la contient. Exemple : R0 -> R1, R1 -> R2 donc R0 -> R2.

Pile de points : elle stocke un ensemble de points à visiter pour faire évoluer les régions.

Une explication est donnée dans la partie 3.c sur le choix de ces techniques.

La classe Couleur :
Nous avons normalisé nos vecteurs de couleurs sous forme RVB et non BGR comme dans OpenCv. Nous avons dû créer un constructeur qui prend en paramètre un CvScalar, ainsi qu'une méthode pour retourner le CvScalar de nos trois composantes sous forme BGR.

Au départ nous voulions créer une couleur unique pour chaque région existante, mais on a remarqué que si le nombre de région créé était important, la génération de couleur unique et aléatoire était d'autant plus prolongée. Nous avons finalement commenté cette méthode et généré des couleurs aléatoire sans se soucier de leur duplication : 1 chance sur 16 777 216 d'obtenir la même couleur, ce qui est très mince. On remarque tout de même une limitation au niveau de la représentation visuelle si ce nombre venait à être dépassé.

2. Déroulement du programme

Voici les différentes étapes exécutées par notre programme pour segmenter une image :

Demander à l'utilisateur de choisir :

1. Une image
2. Un seuil
3. Une méthode de répartition des graines

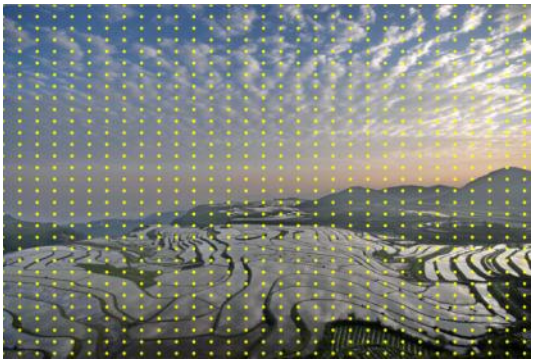
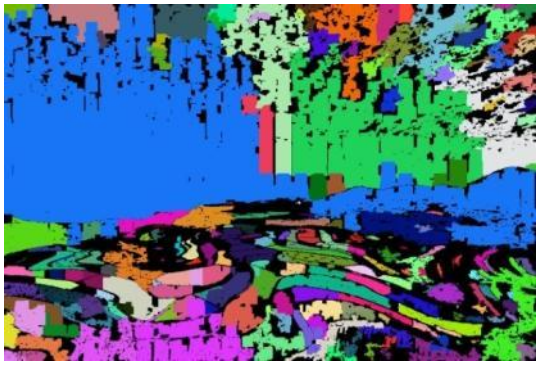
On va ensuite créer une région et lui associé une graine afin d'avoir un point de départ pour le growing. Chaque région va croître et agglomérer un maximum de pixel, et quand deux régions se rencontreront, l'une fusionnera la seconde (sous certaines conditions qui seront détaillés dans la partie 3.c.i). Une fois terminé, on va colorer l'image pour pouvoir visualiser le résultat de la segmentation.

3. Développement et choix

a. Disposition des graines

Nous avons utilisé 3 versions différentes pour disposer les graines sur l'image, la première qui a été implémentée permet à l'utilisateur de les poser où il le désir (il faut placer la souris sur l'image et appuyer sur la lettre 't'). Ce qui nous a permis de tester le growing et la fusion des régions.
Par exemple, on a pu poser deux graines assez proches les unes des autres (en terme de distance), et dans des régions relativement similaires (en terme de couleur), puis observer image par image leur accroissement.

La seconde méthode, plus générique, permet de disposer les graines automatiquement sous la forme d'une grille et selon un certain pas. Cette version est utile quand on veut comparer les régions qui s'étendent le plus sur l'image. Plus le pas est petit, plus la comparaison sera précise et les régions diversifiées.

Taille de l'image = 1280*857
Facteur de découpe = 35
PasX = largeur/35
PasY = hauteur/35

La dernière version dispose N graines aléatoirement sur l'image, elle permet de voir les différences avec la version automatique et en quelque sorte, d'automatiser la première version.

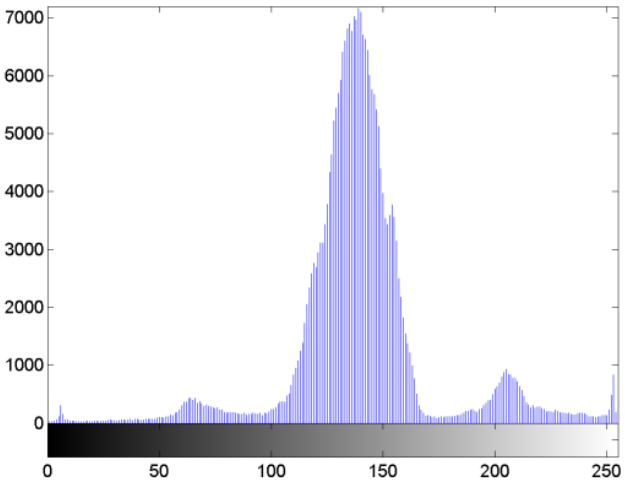
b. Choix du seuil

Le seuil est choisit par l'utilisateur avant chaque traitement de l'image (conservation ou modification de l'ancien).
Nous avons créé une classe Histogramme pour essayer de guider l'utilisateur dans son choix du seuil, elle conserve un tableau d'entier **Tab** de 255 cellules (toutes initialisées à la valeur 0), ou chaque entier de 0 à 255 représente une valeur de l'intensité lumineuse (0 Noir --> 255 Blanc).

On parcourt l'image et pour chaque pixel, on calcule sa couleur moyenne avec la formule :
I = floor((R+V+B)/3). Se qui donne l'intensité lumineuse de ce point, avec un résultat compris dans l'intervalle [0;255] (puisque R, V et B varient dans ce même intervalle). On peut alors récupérer la valeur de **Tab[I]** qui contient le nombre de fois ou cette intensité à été calculé dans l'image, et on l'incrémenter de 1 unité.

On obtient un graphique de ce genre (source : http://www.pfl-cepia.inra.fr/) :

Axe des Y : le nombre de fois où l'intensité X est apparue dans l'image.
Axe des X : l'intensité lumineuse.



Histogramme du niveau de gris dans une image

Une fois l'histogramme calculé, on recherche l'intensité en X ou le Y est le plus élevé (le pic maximal), et on va proposer à l'utilisateur d'utiliser cette valeur (si elle est supérieur à 0).

Après plusieurs teste sur de grandes images (beaucoup de couleurs différentes), on remarque que le pic d'intensité se situe souvent vers des valeurs entre 130 et 180, ce qui est très élevé pour segmenter une image (on peut obtenir une région qui recouvre toute l'image).

Dans d'autres cas ou les couleurs d'une image ne sont pas trop différentes, l'histogramme peut nous fournir une valeur exploitable.

Exemple : ici de nuit avec l'opéra de Lyon, on a un pic d'intensité en 7.

On obtient une image occupé à 59% par des régions (nombre de pixels agglomérés / nombre de pixels total.

c. Critères d'agglomération et de fusion

i. Ce qui est implémenté

Contexte au démarrage :

- aucune région n'est en contacte avec une autres car elles font toute un pixel (à moins que chaque pixel de l'image correspond à une région, ce qui n'a aucun intérêt).
- chaque région porte la même couleur que sa graine

Pour agglomérer un point nous avons établie un seuil que l'utilisateur peut faire varier avant chaque exécution du programme. La condition suivante est utilisée :

Soit R1 la couleur moyenne de la Région 1, et P2 la couleur moyenne de l'image au point P2. Si $\| R1 - P2 \| \leq seuil$, alors P2 est ajouté dans la Région 1.

Quand les régions sont suffisamment grandes, certaines vont se rencontrer et vont devoir fusionner. Comme précédemment, nous avons établies des critères :

- Si $\|$*Couleur moyenne de la région R1 - couleur moyenne de la région R2*$\| \leq seuil$ - Et si la région R1 contient d'avantage de pixel que la région R2, alors R1 agglomère R2

En plus du seuil, nous avons essayé d'ajouter la condition complémentaire :

Si la région R2 contient d'avantage de pixel que la région R1, alors R2 agglomère R1.

Mais après plusieurs testes dans les mêmes conditions, nous avons remarqué une légère augmentation du temps d'exécution (les valeurs sont en seconde) :

Nom de l'image (taille en pixels)	1 condition	2 conditions	% d'augmentation
deuxcarre.pgm (256*256)	0,234	0,25	6,84%
opera-lyon.jpg (540*360)	0,577	0,624	8,15%
paysage-colore.jpg (1024*768)	3,978	4,119	3,54%
NewYork.jpg (1468*1101)	6,224	6,255	0,50%

Après avoir vérifié que l'évaluation de cette nouvelle condition n'était pas la cause de l'augmentation, nous l'avons supprimé. Car à un moment ou à un autre du programme, si R2 > R1 alors un des pixels à la frontière de R2 se chargera de vérifier et d'agglomérer R1.

ii. Autres versions envisagées

Le seuil pourrait varier en fonction du nombre de point restant à agglomérer et le temps T passé à agglomérer. Le souci de cette méthode est qu'une région peut s'arrêtée de grandir quand le seuil est trop bas, puis recommencer au vu du nouveau seuil, alors que les autres régions ont continuées de croitre pendant le laps de temps. Il faudrait donc conserver chacun des points frontières où le programme c'est arrêté, de façon à relancer le growing.

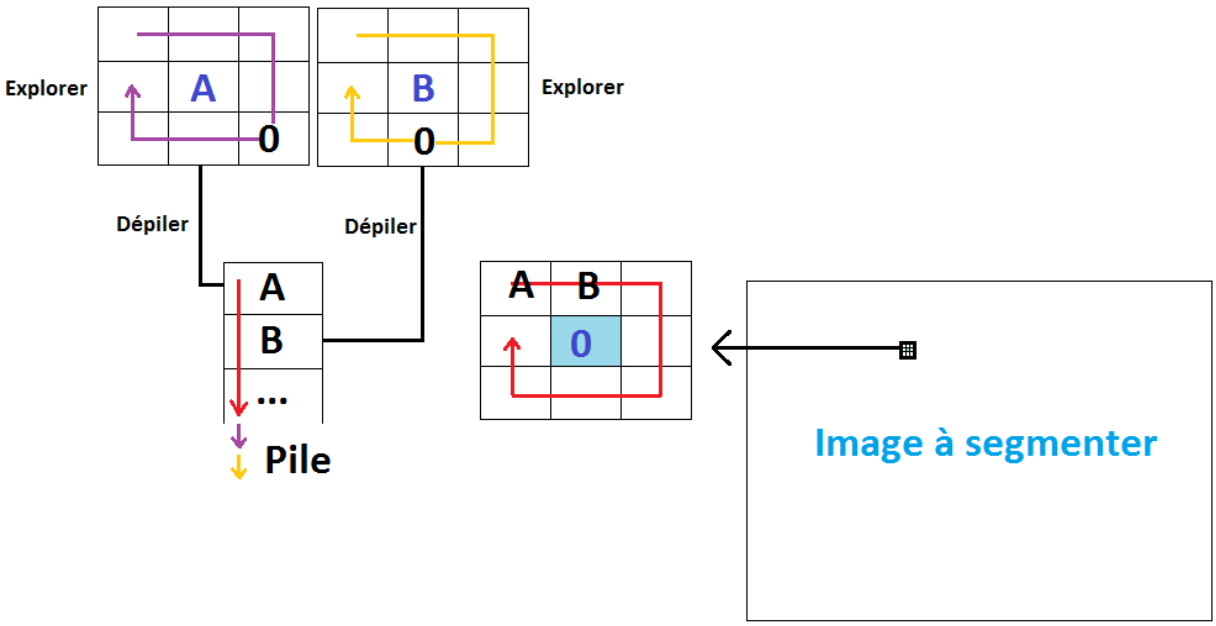
Une seconde version serait d'attendre la fin du growing, puis tant que l'image n'est pas occupée à N%, on augmente le seuil et on relance le processus vers les frontières de chacune des régions.

d. Accroissement et fusion des régions

Chaque région doit grandir autour de sa graine, pour cela, on va regarder les 8 voisins qui l'entourent et pour ceux qui n'appartiennent à aucune région, on va vérifier que leur couleur est assez proche de celle de la région. Si c'est le cas, on va les fusionner et explorer leur voisin à leur tour, sinon on ne s'en occupe pas.

Nous étions partie sur une méthode récursive puis avons dérivé vers une forme itérative avec une pile (équivalent, sauf en nombre d'appel de fonctions et en création de variables). Pour que chaque région croisse en même temps (sans utiliser de thread), nous avons placé toutes les graines dans la pile au début du programme.

Voici un exemple de parcours : on part du point 0, en empile les 8 voisins et on les teste un par un.



Les flèches rouge, jaune et violette représentent le parcours des pixels voisin et l'ordre dans lequel les points sont ajoutés dans la pile. On ne va pas ajouter tous les points voisins au point 0, mais seulement ceux qui respectent les critères définies dans la partie 3.c.i ci-dessus.

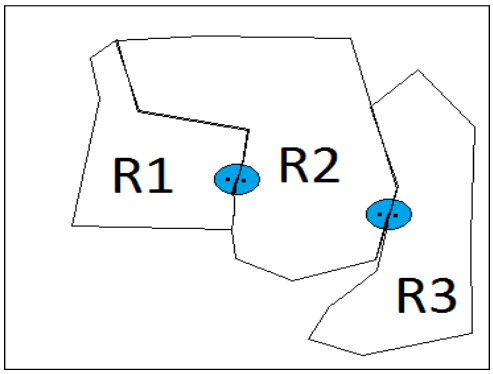
Notre programme ne se divise pas en deux parties telles que : Growing puis Fusion. Nous effectuons la fusion des régions pendant le growing quand au moins un point d'une région est voisin d'un autre point d'une région différente.

Notre fusion de régions a subit divers modifications, voici 3 versions (dans l'ordre chronologique) :

Version 1 :
Chaque région grossit et quand elle rencontre un pixel d'une autre région elle l'agrège (si le seuil est respecté).

Exemple :
P1 (resp. P2) appartient à la région R1 (resp. R2) de couleur moyenne C1 (resp. C2).
Si |C1 - C2| <= seuil et que R1 à une taille supérieur à R2, alors R1 agglomère le point P2.

On voit que ce n'est pas une solution car les points d'une région peuvent être agglomérés en même temps par deux régions :



R1 agglomère un point de R2
R3 agglomère un point de R2

Version 2 :
Pour cette nouvelle version, nous avons choisit d'agglomérer toute une région au lieu d'un seul point. Chaque région dispose d'un *vecteur<CvPoint>* pour stocker l'ensemble des points qu'elle a aggloméré.

Quand une région R2 est agglomérée par une région R1, on passe l'ensemble des points de R2 dans R1. Le tableau d'index "imgIndex" (voir la partie 1) présent dans Accroissement se met aussi à jour.

Après plusieurs testes on a remarqué que si le nombre de région agglomérées était élevé, le temps d'exécution se voyait augmenté, on est donc passé à la 3ème version.

Version 3 (la dernière) :
Le principe reste le même, agglomérer une région entière au lieu d'un seul point.
Le *vecteur<CvPoint>* dans la classe Région à été supprimé et remplacé par une variable qui conserve la taille de la région (incrémenté de un à chaque nouveau point ajouté).
Quand une région R1 agglomère une région R2, on ajoute une redirection dans une map nommé "carteRedirection" (présent dans la classe Accroissement).

Exemple :
R1 agglomère R2 :
On met la taille de R2 = 0 et celle de R1 = R1 + R2
On ajoute dans notre map : "R2" --> "R1"
Ensuite R3 agglomère R1 :
On met la taille de R1 = 0 et celle de R3 = R3 + R1
On ajoute dans notre map : "R1" --> "R3"
etc ...

Les redirections sont mises à jour à chaque fois que l'on doit accéder à une région :
"R2" pointe directement sur "R3", ce qui évite de repasser par toutes les redirections.

Dans la continuité de l'exemple précédant :
Soit le tableau d'index "imgIndex" stocké dans la classe Accroissement pour une image 5*4 :

Y\X	0	1	2	3	4
0	0	0	1	1	2
1	3	0	1	2	2
2	3	0	1	1	2
3	3	3	3	2	2

On accède à imgIndex[4][0] qui retourne 2 (l'index de la région 2), puis on récupère la dernière redirection de la région 2 avec la méthode : indexRedirection(imgIndex [4][0]) (dans Accroissement), qui retourne un numéro d'index. C'est a ce moment que la map "carteRedirection" se met à jour, on va parcourir toutes les redirections de R2 jusqu'à R3, puis on va mettre à jour la clé "R2" dans la map, "R2" --> "R3". Au prochain accès, "R2" pointera directement vers "R3".

4. Estimation de la complexité

Les méthodes dont on va parler se situent toutes dans la classe Accroissement :

- deposerGraines : crée une nouvelle région avec les bons attributs
- changerProprietaireRegion : permet d'agglomérer une région dans une autre, elle s'occupe de la redirrection.
- indexRedirection : permet de récupérer la dernière redirection au niveau des région, et met à jour la map
- contamination : effectue plusieurs testes pour effectuer l'agglomération d'un point ou d'une région

Posons N*M la quantité de pixels de l'image à segmenter.

Le cas le plus lourd qui puisse se présenter, c'est qu'il y ait autant de graines que de pixels.
On a N*M graines donc 6*N*M instructions pour la méthode deposerGraines.

Dans ce cas, s'il y a uniquement le cas où les pixels s'étendent en absorbant les autres dans la procédure contamination, changerProprietaireRegion serai appelé à chaque fois. Dans le pire des cas, on a le nombre de régions qui se divise par 2 et une taille de région qui double à chaque tour (nombre de fois où la taille de la région double par rapport à sa valeur précédente : 1 – 2 – 4 – 8 ...).

Dans contamination, l'appel à la méthode indexRedirection contient une boucle While qui redirige vers la bonne région, si le nombre de régions est divisé par deux à chaque tour, alors le nombre de redirection augmente au fil du temps (elles sont mises à jour au fur et à mesure ce qui évite d'avoir (N*M)-1 redirections pour (N*M)-1 régions dans le cas ou le seuil serait à 255 → version 3 du growing). Ce qui donne une complexité de (N*M)-1 redirections au total.

On a alors: 6*N*M + (4 + (nombreTour*3))*((N*M)-1)/nombreTour, avec :

- nombreTour le nombre de tour nécessaire pour effectuer la segmentation
- 3 le nombre d'instructions de la boucle d'indexRedirection
- 4 le nombre d'instructions hors boucle d'indexRedirection.

Ce nombre de tour serait égal à la valeur minimale de i telle que 2^i > N*M

i * ln 2 > N*M
i > (N*M) / ln 2

6*(N*M) + 4*(N*M) / i + 3*(N*M)

i étant très grand même pour des images de petites taille, 4*(N*M)/ i devient négligeable.
On aurait donc une complexité 9 * (N*M) à laquelle on rajoute la coloration des pixels modifiés (dans le cas le plus lourd) qui demanderait 2*(N*M) instructions.

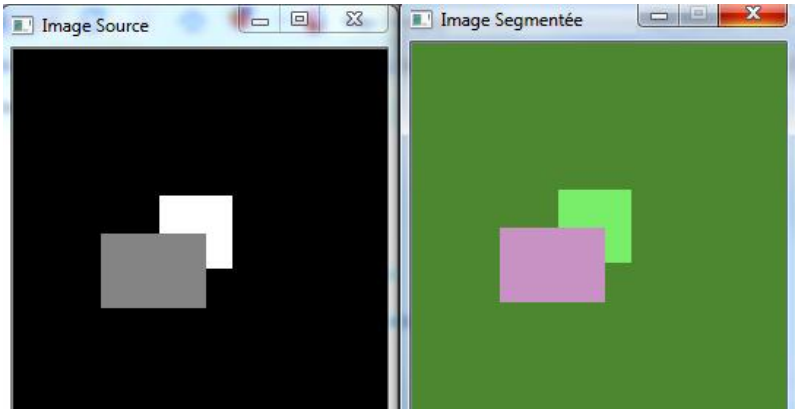
Au final 11 * (N*M) instructions seraient nécessaires.

5. Les critères qui influent sur la segmentation

A près divers essais, on en a déduit que les résultats dépendent fortement :

- du nombre de graines et de leur disposition
- du seuil choisit et de la technique de seuillage
- de l'ordre d'agglomération des points et régions
- le type d'image à segmenter (couleur ou en niveau de gris)

6. Quelques exemples



Pour un seuil = 10 (disposition automatique) :

Graines = 4 0.234 secondes	Graines = 16 0.262 secondes	Graines = 576 0.421 secondes

Teste avec une image de New York de largeur 1468 par 1101, avec un seuil de 10 pour la suite :

Graines = 10605

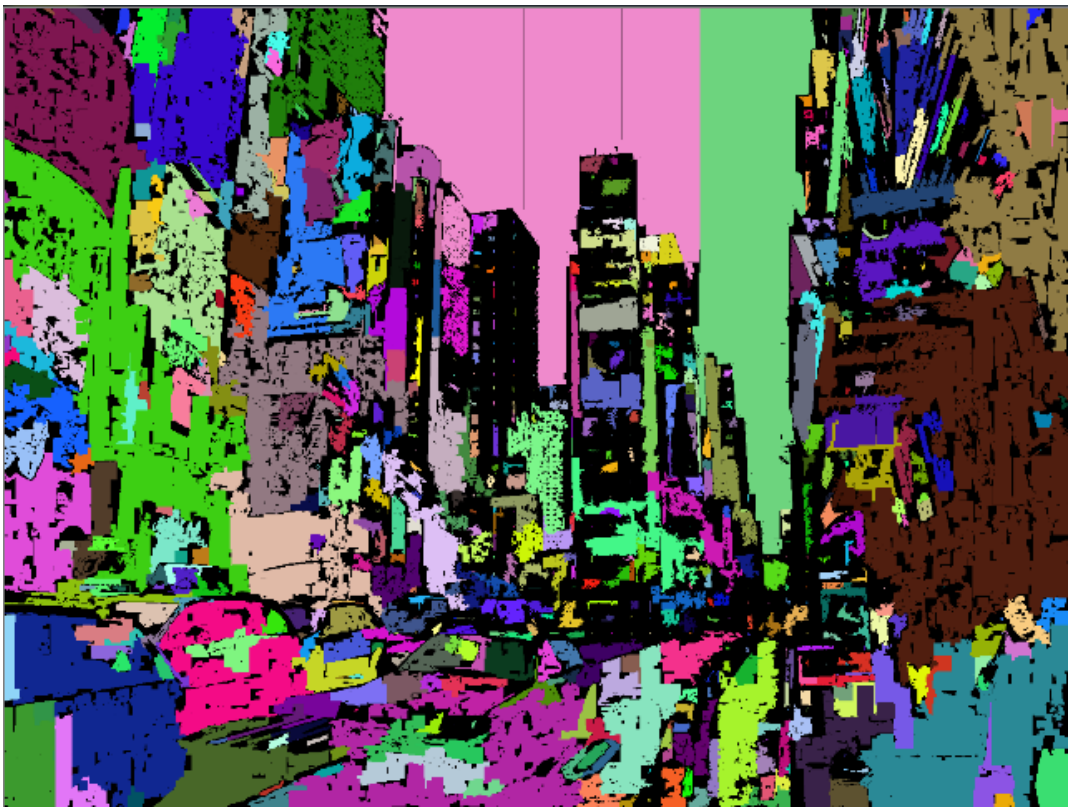
10 secondes pour : la disposition des graines + growing + fusion.

1,37 secondes pour la coloration de l'image.

Graines = 2915

6,43 secondes pour : la disposition des graines + growing + fusion.

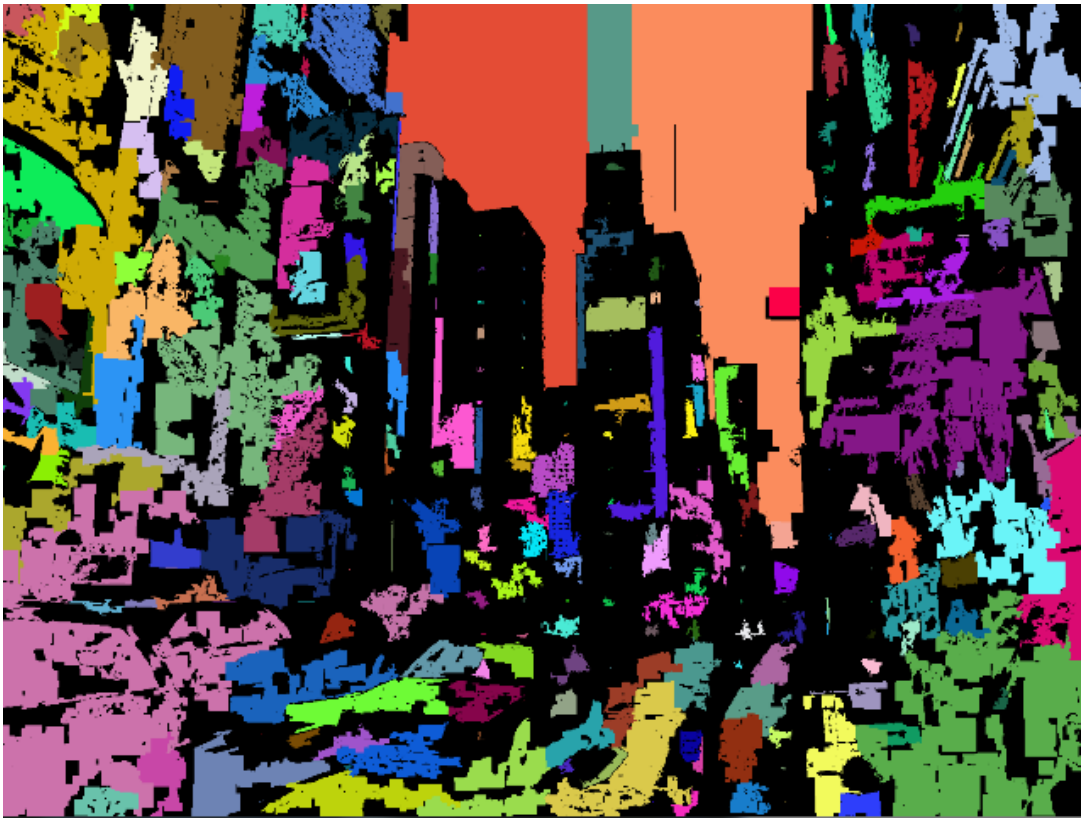
1 secondes pour la coloration de l'image.



Graines = 625

4,13 secondes pour : la disposition des graines + growing + fusion.

0,64 secondes pour la coloration de l'image.



On voit que clairement que le nombre de graine au départ influence nettement les résultats.

Si on utilise le seuil (= 14) donnée par l'histogramme (10605 graines), on obtient :

On voit ici que certaines régions (à gauche et à droite de l'image) sont d'avantage étendues.