Université Claude Bernard Lyon 1

Rapport Analyse d'image

Université Claude Bernard Lyon 1

Edouard Thinot - p1909945
Théo Grillon - p1907354

Informatique graphique

Segmentation par croissance de région

---

1 Introduction

Ce projet s'inscrit dans le cadre de l'UE de master 1 informatique *Mif02 - Informatique Graphique Et Image*. Ce dernier se focalise principalement sur l'analyse d'image. Nous nous concentrons sur la segmentation d'image selon l'approche de la croissance de région (en anglais Region Growing). Ce rapport permet de présenter notre travail. Il est aussi ponctué de retour réflexif sur notre travail ainsi que la justification des choix techniques.

Voici les étapes clés du *région growing* :

— **Pose de germes** : les germes (coordonnées de pixels de l'image) sont répartis sur l'image de travail. Elles constitueront le point de départ des différentes régions.

— **Croissance conditionnelle des régions** : chaque région agglomère à tour de rôles les pixels alentour selon un prédicat. Ce dernier, détermine que les caractéristiques d'un pixel sont suffisamment semblables à la région pour y être ajouté.

— **Fusion des régions voisines semblables** : une fois les régions construites, elles peuvent être fusionnées avec leurs régions voisines si elles sont relativement semblables.

2 Pose de germes

2.1 Première approche

Une solution très simpliste consiste à poser aléatoirement des germes dans l'image à l'intérieur d'un quadrillage.

2.1.1 Quadrillage de l'image

Soit une image $I_{w,h}$:

— w = Nombre de pixels en largeur

— h = Nombre de pixels en hauteur

On calcule en combien de lignes (N_l) et de colonnes (N_c) peut-on découper l'image :

— $N_c = \lfloor \log_2(w) \rfloor$

— $N_l = \lfloor \log_2(h) \rfloor$

Comme on récupère la partie entière inférieure, cela signifie qu'on élimine des valeurs possibles lors du tirage aléatoire (deux bandes de pixels, à droite et en bas de l'image). Cependant, ces bandes sont suffisamment petites pour être ignorées sans que cela ne pose problème dans la distribution des germes (en matière de couverture de l'image).

On peut obtenir la largeur (w_{case}) et la hauteur (h_{case}) de chacune des cases du quadrillage :

— $w_{case} = w/N_c$

— $h_{case} = h/N_l$

On obtient le résultat suivant :

FIGURE 1 – Pose de 100 germes de manière aléatoire

2.1.2 Tirage aléatoire

Pour le tirage aléatoire (distribution uniforme), on tire deux valeurs i et j :

— i = rand($[0,N_c[$)

— j = rand($[0,N_l[$)

Ce couple (i, j) correspond aux coordonnées d'une case du quadrillage.

Ensuite, une fois la case du quadrillage tirée, on tire aléatoirement un pixel de coordonnées (x, y) dans cette case :

— x = rand($[w_{case} * i, w_{case} * i + w_{case}[$)

— y = rand($[h_{case} * j, h_{case} * j + h_{case}[$)

On obtient ainsi un couple d'entiers (x, y) correspondant aux coordonnées d'un pixel de l'image.

L'inconvénient de cette méthode est qu'elle demande un grand nombre de germes pour couvrir suffisamment de zone de l'image et avoir une bonne segmentation. De plus, la sélection de germe aléatoire, sans exploiter des connaissances que nous apporte une analyse primaire de l'image, n'est pas recommandé. En effet, avec cette méthode, il est possible de passer à côté de la caractéristique recherchée. D'où l'importance d'avoir un grand nombre de germes pour maximiser la chance de couvrir au mieux l'image.

2.2 Seconde approche

Notre seconde approche s'appuie principalement sur le calcule de la variance. Soit une image I_{RGB} avec trois canaux de coloration (respectivement Rouge, Vert et Bleu). On convertit l'image d'origine en format HSV (en anglais Hue, Saturation et Lightness) I_{HSV} où Var_H, Var_V, Var_S représentent les variances des canaux teintes (Hue), Saturation et valeur (Value). La moyenne des trois donne la variance d'une zone de l'image :

$$\mathrm{Var}_{\mathrm{total}} = \frac{(\mathrm{Var}_H + \mathrm{Var}_S + \mathrm{Var}_V)}{3.0}$$

La variance de chaque canal est calculé ainsi :

$$\mathrm{Var}_c = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}$$

— μ est la moyenne des intensités des pixels du canal
— On calcule ensuite la différence entre les intensités des pixels et la moyenne
— On élève au carré cette différence
— Puis on calcule la variance, moyenne des carrés des différences

La motivation ici a donc été de ne plus nous reposer sur le caractère aléatoire de la pose des germes, en exploitant les données de notre image. Cela nous a permis par le même occasion de ne plus avoir à spécifier le nombre de germes à poser pour avoir un tôt de recouvrement optimal.

2.2.1 Découpage de l'image

Pour la découpe, une zone de l'image est représenté par deux points. Chaque point possède deux valeurs correspondant à un emplacement dans l'image. Ainsi une région de découpage est représentée par deux points et la variance associés à cette zone. Notre image initiale est donc la zone couverte par le rectangle, tracé entre le point d'origine (un haut à gauche) de coordonnées *(0,0)*, et l'autre extrémité de l'image (en bas à droite) ($NbPixel_{longeur}$,$NbPixel_{largeur}$).

Les critères de découpe sont les suivants :
— Taille minimale d'une région suite à la découpe (30 pixels)
— Limite de division successive (paramétrable, cependant 5 semble optimal)
— La variance de cette région (40.0)

Si la zone de découpe n'est pas trop petite, que le nombre de découpes successives / la variance ne dépasse pas leur limites, alors la zone est partagée en quatre sous zones égales. Le découpage utilise donc quatre threads. L'image initiale est découpée en quatre. Chaque thread poursuit l'opération de découpe de manière récursive tant que tous les critères sont respectés.

2.2.2 Positionnement des germes

Le découpage de l'image nous fournit un ensemble de zones (*SegmentedRegion*) sous forme d'une liste. La pose de germes consiste à parcourir cette dernière et placée un germe au centre de chaque zone.

Utilisé la variance comment critère de positionnement des germes favorise ainsi une couverture maximale pour la segmentation de notre image initiale. De plus, cette deuxième version améliore la modularité de notre programme. Comme l'illustre la figure 2, la pose automatique de 794 germes couvre **93%** de l'image.

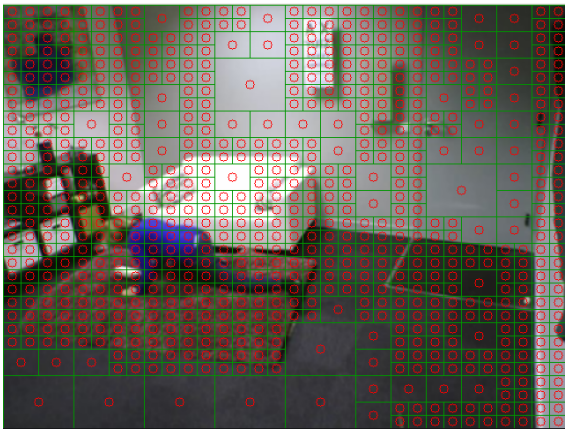

FIGURE 2 – Deuxième version du positionnement des germes. Les rectangles verts représentent les régions de découpe. Les cercles rouges sont centrés sur les coordonnées du germe.

3 Prédicat

Pour effectuer la croissance des régions, on a besoin de définir un prédicat d'homogénéité. Il va permettre de définir, lors de l'itération sur les pixels voisins d'un germe, s'ils doivent être ou non ajoutés à la région courante.

3.1 Première approche - Espace RGB

Notre tout premier prédicat était relativement simple, il réalisait la différence entre le germe et le pixel actuel pour chaque canal de couleurs (RGB).

$$Diff_{c\in[0,1,2]} = |\text{seedPixel}[c] - \text{actualPixel}[c]|$$

- c = valeurs numériques des canaux RGB respectivement 2, 1 et 0.
- seedPixel[c] = valeur du canal c du germe pour ce domaine.
- actualPixel[c] = valeur du canal d'un des pixels voisins au domaine.

Ces différences étaient ensuite comparées au seuil. Si elle était toutes inférieure à ce dernier, alors le pixel en cours de traitement était propice à intégrer cette région.

$$Diff_R < \text{threshold} \wedge Diff_G < \text{threshold} \wedge Diff_B < \text{threshold}$$

Nous avons ensuite défini un prédicat qui se basait sur des valeurs de hachage supposées uniques pour chaque combinaison RGB. L'idée était de réduire la valeur RGB en un entier afin de pouvoir facilement réaliser une comparaison. Le prédicat réalise un ratio entre deux hashs. Cela nous donne un pourcentage supposé nous dire si elles sont proches ou non.

Voici comment le hash est calculé :

$$(\sum_{k=0}^{2} \text{B} \cdot 10^{2k} + \text{G} \cdot 10^{2k+1} + \text{R} \cdot 10^{2k+2}) - (B - G * R)$$

Cependant, ça ne fonctionne pas puisque que le calcul des hashs est incorrect et ne permet pas d'avoir une valeur pertinente dans le sens où deux valeurs RGB proches l'une de l'autre n'ont pas la garantie d'avoir des valeurs de hachage proche. Par exemple, en prenant A=(0, 0, 255) et B=(5, 0, 255), comme le poids de la composante R dans le calcul du hash est très élevé, quand bien même sa valeur est petite, la somme totale et donc le hash de A devient beaucoup plus grand que celui de B.

3.2 Seconde approche - Espace HSV

Nous cherchions un moyen de pouvoir déterminer facilement la proximité entre deux couleurs de pixel. Pour cela, l'utilisation de l'espace HSV nous a parus être une bonne idée, car il offre une représentation plus intuitive des couleurs que d'autres espaces de couleur tels que l'espace RGB. Cette représentation permet de manipuler plus facilement des caractéristiques telles que la teinte, la saturation et la luminosité, ce qui facilite la mise en place du prédicat d'homogénéité.

3.2.1 Qu'est-ce que HSV ?

L'espace de couleur HSV (Hue, Saturation, Value) est un modèle colorimétrique qui représente les cou-

leurs en trois composantes principales :

- **Teinte (Hue)** : représente la tonalité de la couleur, généralement exprimée en degrés sur un cercle chromatique. La teinte spécifie la couleur dominante, comme le rouge, le vert ou le bleu.
- **Saturation (Saturation)** : indique l'intensité de la couleur. Une saturation élevée signifie des couleurs vives et pures, tandis qu'une saturation faible se traduit par des teintes plus fades ou des nuances de gris.
- **Valeur (Value)** : représente la luminosité ou l'éclat de la couleur. Une valeur élevée indique une couleur plus claire, tandis qu'une valeur faible correspond à une couleur plus sombre.

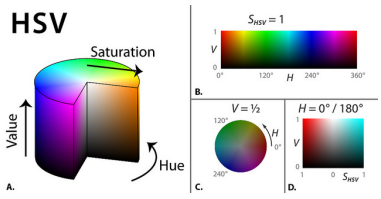

FIGURE 3 – Espace de couleur HSV

3.2.2 Comment l'utiliser pour le prédicat ?

Pour le prédicat, on définit un intervalle de valeur HSV. Cet intervalle est défini pour chaque région, avant l'étape de croissance, en fonction de la valeur HSV du germe initial. Pour déterminer la borne inférieure et supérieure, on a défini quatre cas de valeur possible : la valeur HSV peut décrire du blanc, du gris, du noir ou bien une couleur (rouge, vert, etc...).

Le premier channel qui nous intéresse est celui de la luminosité (V). Si sa valeur est inférieure à 15%, cela signifie que nous sommes dans le cas du noir, peu importent sa valeur de teinte (H) ou de saturation (S). Sinon, on s'intéresse à la valeur d'intensité ou de saturation (S). Plus la saturation est proche de zéro ($< 30\%$), plus cela signifie que le pixel est en niveau de gris. Pour déterminer si nous sommes dans le cas du gris ou du blanc, on regarde la composante V (gris si $<70\%$, blanc sinon). Enfin, si les précédents ne sont pas vérifiés, nous sommes dans le cas d'une couleur dans laquelle la teinte (H) va nous intéresser.

Soit *value* la valeur HSV testée, *lowerb* et *upperb* les bornes de l'intervalle. Le prédicat d'homogénéité correspond à la vérification de cette assertion :

$$\text{lowerb} \leq \text{value} \leq \text{upperb}$$

4 Croissance de région

4.1 Processus de croissance

Comme mentionner dans la partie sur les prédicats d'homogénéité, notre première version de l'algorithme d'accroissement de région était relativement simple et non optimal. La coloration des régions ainsi que la calcule d'appartenance des voisins se faisait en même temps.

Cette partie permet donc d'exposer les détaille de notre algorithme de croissance de région.

4.1.1 Qu'est-ce qu'une région ?

On définit les régions comme un ensemble de pixels respectant le même prédicat d'homogénéité. Pour effectuer les étapes de croissance et de fusion, on a besoin des informations suivantes pour chaque région :

- les coordonnées des pixels contenus dans la région
- la borne inférieure et supérieure HSV
  - utilisée pour le prédicat d'homogénéité
- la moyenne des valeurs HSV des pixels contenue dans la région
  - utilisée pour la fusion

Toutes les régions sont stockées dans une table de hachage. La clé correspond à la valeur hexadécimale, stockée sur un entier, du code RGB du germe initial. Cette structure de données permet facilement l'accès et la suppression de région, en temps constant dans le meilleur des cas (c'est-à-dire sans collisions).

4.1.2 Que fait l'algorithme de croissance ?

Le but de ce processus est de remplir une matrice de même dimension que l'image source. Cette matrice représente le masque sur lequel vont être affichées les régions.

L'algorithme effectue des comparaisons et manipule des valeurs de pixel en permanence. Pour simplifier tout ça, notre masque ne comporte pas directement les valeurs sous le format RGB mais contient à la place des entiers représentants leur valeur hexadécimale. Ces entiers sont utilisés en tant que clé de région.

Chaque germe, avant de commencer leur croissance, initialise sa valeur dans le masque et de défini son intervalle HSV. On place les coordonnées du germe dans une pile puis on lance le parcours sur ses voisins.

4.1.3 Comment déterminer l'ajout d'un voisin ?

Lors du parcours d'un des pixels voisins, on regarde sa `valeur` dans le masque. Il existe plusieurs cas possibles :

- **Si `valeur` = 0 :**
  - Le masque étant initialisé à 0, et le noir ne faisant pas partie des valeurs de clé assignables, cela signifie que le pixel n'appartient à aucune région. On lui assigne pour valeur la clé de la région courante et on ajoute ses coordonnées à la pile.
- **Sinon si `valeur` $\neq$ 0 :**
  - Et que `valeur` $\neq$ clé région courante
    - le pixel voisin appartient à une autre région. On vérifie si une fusion est possible. Si tel est le cas, on fusionne les deux régions, sinon on ne fait rien.

L'algorithme de croissance s'arrête lorsque la pile est vide.

4.1.4 Comment visualiser le résultat ?

Comme le masque est une matrice d'entier, si l'on souhaite visualiser le résultat de la segmentation, par exemple à l'aide de la fonction `cv::imshow(...)` sous openCV, on applique un traitement supplémentaire qui convertit les valeurs hexadécimales du masque en valeur RGB.

4.1.5 Quelle en est la complexité temporelle ?

Sans prendre en compte l'étape de traitement nécessaire pour l'affichage du résultat, puisque ça ne fait pas partie intégrante du processus de croissance, la complexité temporelle de cet algorithme est de $\Omega(n)$, où n représente le nombre de pixels contenus dans la région, et de $O(n \cdot \omega_{max})$, où ω_{max} représente la taille maximale de ω parmi toutes les fusions.

4.2 Processus de fusion

4.2.1 Quels critères pour la fusion ?

On rappelle que chaque région possède un intervalle correspondant aux valeurs HSV accepté par le prédicat d'homogénéité correspondant. Un pixel n'est ajouté à une région, que si sa valeur HSV est incluse dans cet intervalle. C'est la même chose pour les régions. Pour savoir si deux régions adjacentes doivent être fusionnées, on va vérifier si leur valeur moyenne respective est contenue dans l'intervalle de l'autre région. Autrement dit :

Soit $\overline{\text{HSV}}_{R1}$ la valeur HSV moyenne de la région $R1$,
$\overline{\text{HSV}}_{R2}$ la valeur HSV moyenne de la région $R2$,
et $[\text{lowerb}, \text{upperb}]$ l'intervalle HSV de la région.

Alors, la condition de fusion entre $R1$ et $R2$ peut-être exprimée par l'assertion suivante :

$$\text{si } \overline{\text{HSV}}_{R1} \in [\text{lowerb}_{R2}, \text{upperb}_{R2}]$$
$$\text{et}$$
$$\text{si } \overline{\text{HSV}}_{R2} \in [\text{lowerb}_{R1}, \text{upperb}_{R1}],$$
$$\text{alors } R1 \text{ et } R2 \text{ peuvent fusionner.}$$

4.2.2 Quelles sont les étapes ?

La fusion est une union entre deux régions. Notre première étape est de déterminer laquelle des deux régions va être incluse dans l'autre. Plus concrètement, on cherche la région contenant le moins de pixels.

La deuxième étape correspond à la mise à jour des bornes de l'intervalle et la moyenne HSV. Pour mettre

à jour la moyenne R1, on ne reparcours pas tous les pixels de la région en sommant leurs valeurs. À la place, on effectue un calcul de moyenne pondérée. On peut résumer cette étape par ces instructions :

Soit |R1| et |R2| le nombre de pixel de chaque région, En supposant $|\text{R2}| \leq |\text{R1}|$:

- $\text{lowerb}_{R1} = \min(\text{lowerb}_{R1}, \text{upperb}_{R2})$
- $\text{upperb}_{R1} = \max(\text{upperb}_{R1}, \text{upperb}_{R2})$
- $\overline{\text{HSV}}_{R1} = \dfrac{|\text{R1}| \cdot \overline{\text{HSV}}_{R1} + |\text{R2}| \cdot \overline{\text{HSV}}_{R2}}{|\text{R1}| + |\text{R2}|}$

La dernière étape fait l'inclusion des pixels de R2 dans R1. La structure de données utilisée pour stocker les coordonnées des pixels est une liste doublement chaînée. Cela permet de réaliser la concaténation de deux listes en temps constant. Pour terminer, la valeur des pixels contenus dans R2 est mise à jour. Notre algorithme de fusion à une complexité temporelle $\Theta(\omega)$, où $\omega = \min(|\text{R1}|, |\text{R2}|)$.

5 Conclusion

À travers le travaille de recherche d'information et d'expérimentation, nous avons pu réaliser un projet à la hauteur de nos attentes. Nous ne nous sommes pas contentés des premières réalisations. Merci à nos professeurs.es et à leur vision critique de nous avoir permis d'aller plus loin dans nos réflexions. Cela nous a permis de nous focaliser sur l'aspect technique du région growing en m'étant en place des solutions justifiée, en adéquation avec le cahier des charges. Bien que notre code mériterait d'être homogénéisé, nous avons implémenté les fonctionnalités centrales du projet en tâchant de respecter une philosophie d'économie. Pour cela, nos structures de données / algorithmes ont été réfléchies en gardant en tête la notion de complexité et d'espace mémoire.

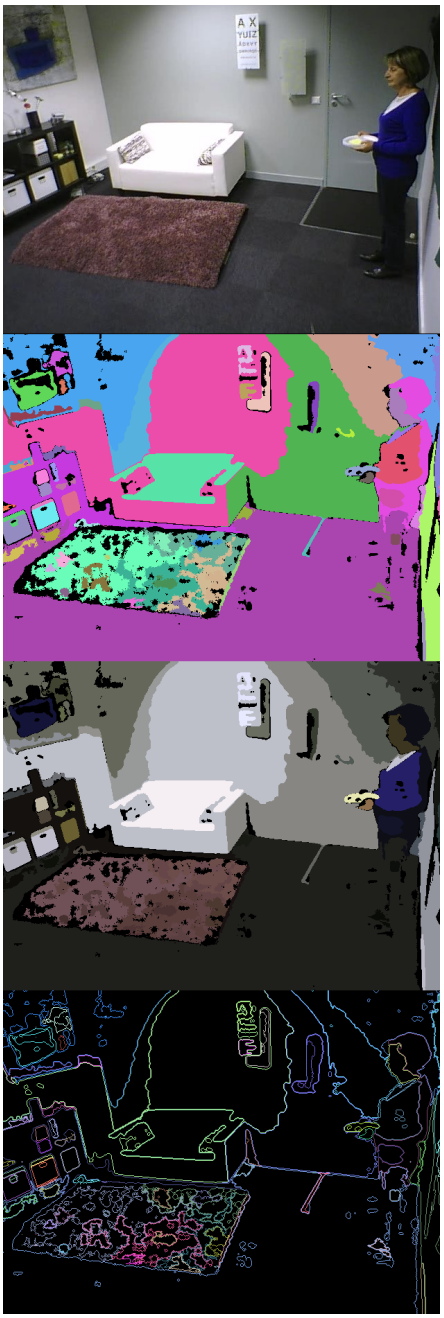

FIGURE 4 – Résultats finals obtenus dans l'ordre : l'image d'origine, la segmentation avec deux colorations différentes (paramètre avec 5 comme indice de division maximal, pour une couverture de 94%) et l'affichage des frontières.