COMPTE RENDU BE2

Algorithme quadripartition appliqué à l'image

I - Split

1) Lecture de la valeur d'un pixel

On définit la fonction **pixel_read**(x,y) :

- ⇨ Renvoie la valeur du pixel ciblé
- ⇨ On gère le cas où *x>=w* et/ou *y>=h* (pixel hors image)
  - o Pour cela on utilise un **raise** ValueError
  - o Si c'est le cas, on utilise un modulo (cela pourra être utile par la suite)

2) Ecriture de la valeur d'un pixel

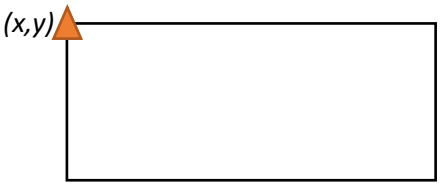
On utilise les mêmes remarques que précédemment pour définir la fonction **pixel_write**(x,y,r,g,b)

- ⇨ Fonction sans retour, affecte simplement la valeur (r,g,b) au pixel de coordonnées (x,y)

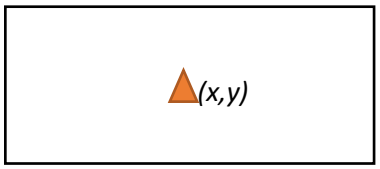
3) Ecriture d'une valeur sur un rectangle de pixel

On définit la fonction **rect_write**(x,y,w,h,r,g,b,type)

- ⇨ La variable type change le rectangle défini : (figures ci-dessous)



type = 0



type = 1

- ⇨ Le triplet r,g,b représente la valeur à affecter à l'ensemble des pixels
- ⇨ On utilise une double boucle for pour effectuer l'attribution des valeurs, en faisant appel à la fonction **pixel_write**(x,y,r,g,b) définie précédemment. Celle-ci permet d'éviter les erreurs lors de la boucle car elle contient le système de gestion d'erreur précédemment décrit

4) Vérification de l'homogénéité

On définit la fonction **homogene**(x,y,w,h,s,type) :

- ⇨ Retourne un booléen, puis le triplet de valeur d'un pixel moyen de la zone
- ⇨ Le système de sélection du rectangle procède de la même manière que précédemment
- ⇨ La variable s définit le seuil de validité pour le critère d'homogénéité
- ⇨ On calcule la moyenne et l'écart type sur chaque couleur
- ⇨ On veut que l'écart type maximum soit inférieur à la valeur seuil
- ⇨ On a défini la fonction **VarCh(T)** qui renvoie la moyenne et l'écart type d'une liste

5) Implémentation de la quadripartition (récursive)

On propose d'utiliser l'algorithme suivant :

Appel de l'algorithme de quadripartion sur toute l'image :

- On enregistre la zone en cours à sa place dans un tableau quaternaire
- Si la région est homogène, la remplacer par la couleur moyenne, et l'ajouter à la liste des régions
- Sinon, on lance quatre appels récursifs à quadripartion sur l'image coupée en 4

6) Intérêt d'un arbre quaternaire

Lors de la seule phase de split, la construction d'un arbre ne présente aucun interet. Cependant, seul la construction d'un arbre peut permettre de réaliser la phase de merge sans un cout en temps trop important. Plus particulièrement, la réalisation à la volée d'un arbre quaternaire permet de réaliser le graphe représentant l'organisation des régions de manière efficace.

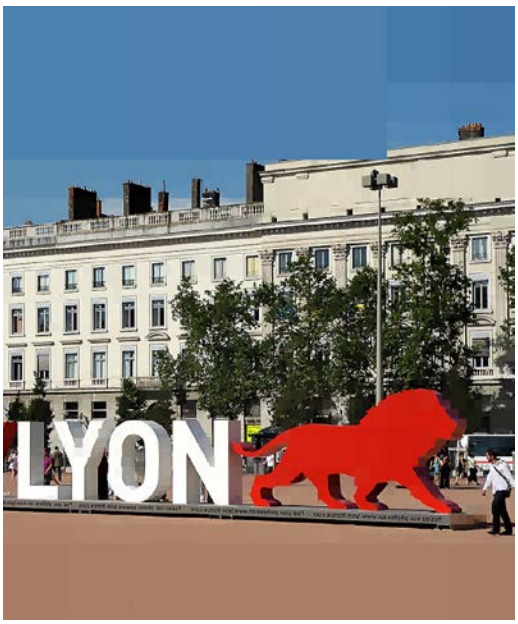
7) Implémentation de l'algorithme de quadripartition

On définit la fonction **quadripart**(x,y,w,h,s,i) :

- s est le seuil choisi pour le test d'homogénéité
- i est la position de la région a traité dans l'arbre quaternaire. Par définition, l'indice de la région initiale (image complète) est 0. Les fils d'une région sont stockés dans les régions d'indices (4i+1, 4i+2, 4i+3 et 4i+4)

Le code de la fonction est commenté de manière détaillé pour avoir une idée de ce que fait chaque partie de la fonction.

A cet instant, les résultats sont les suivants :



Pour s=30 *Pour s=50*

Dans le graphe, chaque région est représentée de la manière suivante :[x, y, h, w, r, g, b, booléen]

Ou :

- x, y sont les coordonnées du premier point de la zone,
- w, h sont la largeur et la hauteur de la zone
- r, g, b sont les codes couleurs de la moyenne des couleurs de la zone
- le booléen représente l'homogénéité de la zone vis-à-vis du seuil (True si la zone est homogène)

8) Addendum :

Pour plus de lisibilité, on définit la fonction **Split**(s) qui réalise la phase de split de l'image avec comme seul paramètre le seuil s. (valeurs intéressantes comprises entre 20 et 50). Cette fonction affiche également l'image, et l'enregistre en explicitant dans le nom de fichier la valeur du seuil.

On avait prévu de pouvoir définir une région par le pixel en son centre. Vu les fonctionnalités développées après, cette fonctionnalité ne présente aucun intérêt. Néanmoins, vu son absence de cout sur le temps d'exécution du programme, j'ai choisi de ne pas la supprimer.

Le calcul de la moyenne et de l'écart type est réalisé par une seule fonction qui renvoie le couple de la liste fournie en paramètre. Cette fonction est dans le code, mais n'est pas commentée dans le présent rapport. Elle cherche à minimiser le nombre de calculs de moyenne.

ATTENTION : Penser à ré exécuter le programme avant chaque appel de Split(s) pour vider le graphe quadratique des valeurs prises à l'itération précédente !

II – Merge

1) Fonction Merge

On définit la fonction merge(T, i) :

- Cette fonction merge la zone définie par le tableau T à l'indice i
- On verifie si la zone comporte des sous zones
- On commence par fusionner les sous-zones
- Puis on s'occupe de fusionner avec les adjacents de la même zone

2) Fonction merge_sz(T,i)

Cette fonction est simplement un appel récursif à merge(T,i)

- On réalise quatre appels récursifs (4 sous zones de T[i])

3) Fontion merge_adjacents(T,i)

Cette fonction s'occupe de réaliser les premières fusions avec les voisins des régions adjacentes puis réalise la fusion à l'intérieur de la région

4) Fonction merge_voisins(T,i)

Cette fonction réalise 8 appels à la fonction de fusion, elle s'occupe de fusionner chaque sous zone avec les zones qui y sont adjacentes, mais situées dans les autres zones de la région mère (il s'agit donc des neveux !)

5) Fonction fusion(T,i,j)

Cette fonction réalise la fusion des zones T[i] et T[j] :

- On extrait les données propres de chaque région du graphe quaternaire qui les contient
- On effectue des tests sur l'attribut merged (booléen qui est propre à chaque région et qui caractérise le fait que la région à déjà été fusionné avec une autre)
- Si une des deux régions à déjà été fusionnée, et que c'est possible, on propage ses valeurs à l'autre région, et on change au passage son attribut merged en True
- Si ce n'est pas le cas, on réalise la fusion si elle est possible (selon un critère de seuil que l'on peut changer en fonction du résultat désiré) en effectuant la moyenne (pondérée par le poids de chaque zone en nombre de pixels) de chacune des composantes de la couleur. On bascule évidement les attributs merged de chaque zone à True.

6) Addendum

Dans cette partie, j'ai décidé d'utiliser un graphe quaternaire pour directement réaliser les fusions. Il n'y a donc pas besoin de réaliser une Région Adjacency List pour effectuer le traitement.