FUCHS Steve
HAHN Philippe
SCHEFFKNECHT François

Tuteur : Mme A. Deruyver

Projet :

Traitement d'images

Watershed et fusion de régions

Manuel technique

Annexe A

Structure de l'application

Dans le but premier de coordonner nos actions pendant le développement de l'application, voici une liste de tout les fichiers du programme ainsi qu'un bref descriptif sur la fonction de chacun d'eux

watershed/

StandAlone.java

Cette classe contient le programme exécutable de l'application. Elle permet de récupérer l'éventuel argument « -debug » de la ligne de commande d'exécution et d'informer au système d'exploitation de suivre chaque instruction du programme. Elle affiche ensuite l'objet MainFrame contenu dans watershed/fenetres/.

MainPanel.java

Cette classe définit le Jpanel de l'application. Elle importe la plupart des éléments graphique de la bibliothèque Swing et les éléments graphique prédéfinies par nous-mêmes tels que la barre de menus,… et les dispose correctement dans le panneau. Elle contient aussi les méthodes de gestion des fenêtres filles de l'application.

watershed/fenetres/

MainFrame.java

Cette classe étend la classe Jframe. Elle créée une fenêtre de type Swing dans laquelle sera mise un objet MainPanel définit juste au-dessus. Elle agrandit aussi cette fenêtre au maximum de l'écran.

ImageFrame.java

Cette classe étend la classe JInternalFrame. Elle permet de créer des fenêtres filles à la fenêtre principale et d'y afficher à l'intérieur une image donnée.

watershed/langages/

Const.java

Cette classe contient l'ensemble des constantes chaînes de caractères de l'application en fonction de la langue choisit. Elle contient aussi la méthode de chargement de ces variables du fichier .wlg correspondant.

Francais.wlg ; Deutsch.wlg ; English.wlg

Ces fichiers sont des fichiers textes formatés de la façon suivante : « *nom_de_la_variable_globale : valeur* ». Ceci permet sans aucun problème de créer la gestion d'une nouvelle langue.

Maker.java

En cours de construction par le sieur François.

watershed/menus/

Bar.java

Cette classe étend la classe JMenuBar et crée les différents menus de notre application avec les diverses classes contenues dans ce même répertoire et qui sont définit ci-dessous.

MEdit.java ; MFile.java ; MHelp.java ; MView.java ; MWindow.java

Ces classes mettent en œuvre les différents menus de l'application. Certains d'entre eux restent cependant dans d'autres répertoires tels que le menus des filtres.

watershed/misc/

Chargement.java

Cette classe étend la classe Windows. Elle définit la barre de progression qui s'affiche lors de, par exemple, le chargement d'une image, l'application d'un filtre, …

watershed/icones/

Ce répertoire contiendra les différentes icônes de l'application, en particulier les icônes des différents menus. Les icônes des menus sont des icônes au format GIF, de dimensions 16 x 16 pixels en 256 couleurs. Ceci permet de rendre l'interface de l'application un peu plus joviale.

watershed/workspace/

Couleur.java

Cette classe permet la gestion des couleurs. On y trouvera par exemple des fonctions comme retrouver le niveau de bleu dans une couleur, transformer cette couleur en noir et blanc, …

Data.java

Cette classe symbolise les images. Chaque image est copiée dans un tableau selon son type. Cette classe reste encore très obscure à mes yeux.

Niveaux.java

En cours de construction…

Picture.java

Cette classe m'est totalement obscure. Des méthodes telles que « zoom » me laissent penser qu'elles permettent la gestion des images affichés mais je n'en suis absolument pas sûr.

watershed/es/

LangReader.java ; LangWriter.java ; Reader.java ; Writer.java

Je comprend rien !

watershed/formats/

FileFormat.java

Cette classe est l'interface des classes définissant les formats d'images ouvrables grâce à notre application.

FileFormatChooser.java

Cette classe permet de charger l'ensemble des noms des fichiers implémentant la classe ci-dessus et de les affecter à la boîte de dialogue d'ouverture de fichier.

FileFormatFilter.java

Cette classe permet, au premier abord, la même chose que précédemment.

BMP.java ; GIF.java ; JPG.java

Ces classes définissent les types d'images ouvrables par l'application ainsi que la méthode d'ouverture de ces images.

watershed/filtres/

Filter.java

Cette classe est l'interface des classes définissant les différents filtres disponibles grâce à notre application.

FilterAction.java

Je n'arrive pas à comprendre le but de cette classe.

FilterLoader.java
Cette classe permet de détecter l'ensemble des filtres contenus dans le répertoire, c'est à dire l'ensemble des classes implémentant Filter.java.
Gradient.java ; GrayScale.java ; Moyenne.java
Ce sont les classes des actuels filtres de l'application. Chacune définit les actions des filtres qu'elles représentent.

Annexe B

Implémentation de l'algorithme de Watershed

I] Etape de préparation

L'image devra être, avant tout, convertie en noir & blanc et elle aura été, cela est toutefois facultatif quoique bénéfique, soumis soit à un filtre médian, soit à un filtre gradient, soit aux deux.

Dans le cadre de l'application de l'algorithme de Watershed, a chaque point de l'image, on doit associer un label. Il nous faut donc recopier l'image dans un tableau de structure définit comme suit :

- Le tableau doit avoir pour hauteur le nombre de pixels de hauteur de l'image,
- Le tableau doit avoir pour largeur le nombre de pixels de largeur de l'image,
- Le tableau doit contenir des objets définit comme suit :

```
class WPoint
{
      int x;
      int y;
      int nGris;
      long label;
}
```

Chaque pixel de l'image de coordonnées (x, y) sera converti en un objet de type WPoint qui sera placé dans le tableau à l'emplacement [x] [y].

Il faudra, pour cela :

1. Recopier l'intensité de gris (de 0 à 255) du pixel dans la variable nGris,
2. Initialiser label à 0 (non affecté).

Les variables x et y de l'objet sont facultatif, elles représenteront les coordonnées du WPoint dans le tableau et leurs présence dépendra de leurs utilité.

Ensuite, il nous faudra initialiser une liste (ou un tableau) définie comme suit :

- La liste contiendra des éléments de type File <WPoint>,
- Elle contiendra autant de files que de niveau de gris dans l'image,
- Les files seront ordonnées du plus petit niveau de gris au plus grand,
- Chaque file sera initialisée correctement (vide au départ).

Ces files contiendront en fait des WPoints référençant des WPoints du tableau.

Nous voilà donc avec tout les éléments nécessaires au bon déroulement des opérations…

II] Recherche des minima et initialisation de la liste de files

La première étape de l'algorithme de lignes de partage des eaux est d'initialiser correctement les files de la liste.

En effet, les files doivent être initialisées avec les minima de l'image, c'est à dire les points de niveau de gris les plus bas.

Comment les trouver ? Deux méthodes s'offrent à nous.

1) Recherche des minima sans intervention de l'utilisateur

Cette technique consiste à balayer l'image dans son intégralité et de labelliser les points considérés comme des minima.

On utilise pour cela l'algorithme suivant :

```
pour
   x <- 0
   xMax <- largeur en pixels de l'image
   label <- 1
tant que x < xMax répéter
   pour
      y <- 0
      yMax <- hauteur en pixels de l'image
   tant que y < yMax répéter
      si WPoint(x, y) a un niveau de gris < à celui de ses 8 voisins
         alors WPoint(x, y).setLabel(label)
               label <- label + 1
         sinon rien
      fsi
      y <- y + 1
   fpour
   x <- x + 1
fpour
```

Les minima de l'image seront alors, à la fin de ce traitement, les WPoints dont le label sera supérieur à 1.

2) Recherche des minima avec des marqueurs

Cette solution est la plus simple d'un point de vue programmation car elle ne nécessite pas de fonction de recherche des minima, c'est l'utilisateur, grâce à une sorte de pinceau virtuel qui colorera les zones qu'il considère comme des minima.

Les points à insérer dans la liste de files sont les points frontières des zones marqués.

Cette méthode comporte toutefois un hic, il ne faut pas tomber sur un utilisateur fainéant…

III] Déroulement de l'algorithme de Watershed

L'algorithme est le suivant :

```
pour
   rien
tant que chacune des files n'est pas vide répéter
   soit x un WPoint
   soit a, b, c, d les WPoints de x // on ne considère que 4 voisins
   x <- extraire()
   si a.label = 0
      alors  a.label = x.label
             insérer(a)
      sinon  rien
   fsi
   si b.label = 0
      alors  b.label = x.label
             insérer(b)
      sinon  rien
   fsi
   si c.label = 0
      alors  c.label = x.label
             insérer(c)
      sinon  rien
   fsi
   si d.label = 0
      alors  d.label = x.label
             insérer(d)
      sinon  rien
   fsi
fpour
```

```
fonction extraire()

pour
   x <- 0
tant que liste(x).longueur() = 0 répéter
   x <- x + 1
fpour

renvoyer liste(x).prendre()
```

```
fonction insérer(WPoint x)

nPriorité <- x.nGris
liste(nPriorité).rajouter(x)
```

A la fin de cette algorithme, tout les points de l'image sont labellisés ; il ne reste alors plus qu'à afficher le résultat.