Bontron Julien bontronjulien@yahoo.fr
Clarion Jean Baptiste jbclarion@me.com
Steyer Antoine antoinesteyer@gmail.com
Equipe 1

Projet INFO 424

Synthèse et Analyse d'une image PPM

Table des matières

1 / Présentation générale du projet

Le projet consiste en la réalisation d'un synthétiseur / analyseur d'image. Il est divisé en deux programmes, codés en deux langages différents, un pour la synthèse et l'autre pour l'analyse. Les images étudiées seront au format PGM, qui est décrit plus bas, en premier lieu puis en PPM lors de l'amélioration du projet.
L'objectif est de pouvoir générer des figures simples (rectangle, triangle et cercle) et de pouvoir les identifier par la suite.

Ex : on génère grâce au synthétiseur un rectangle rouge de dimension 5 pixels par 5 pixels et avec comme point d'origine (1,1) , puis l'analyseur devra renvoyer :
« Rectangle Rouge, 5 par 5, origine (1,1) »

Pour remplir l'objectif du projet, nous devons coder plusieurs fonctionnalités dans nos programmes.

Synthétiseur :
- Génère une image à partir d'arguments passés en ligne de commande
- Encode l'image en format PGM
- Produit l'image par la sortie standard

Analyseur :
- Lit un fichier image via l'entrée standard
- Décode le fichier pour produire l'image
- Identifie les formes géométriques dans l'image

Pour le projet, nous avons à choisir un gestionnaire de version, dans notre cas Github, ainsi qu'un éditeur texte, dans notre cas Emacs sous linux.

Les fonctionnalités de base que notre projet doit avoir sont les suivantes :
- Gestion de formes géométriques simples
  - rectangle aligné avec les axes.
  - Cercle
  - triangle avec base horizontale.
- Gestion des couleurs (format PPM)

Si ces fonctionnalités sont achevées, nous pouvons ajouter des fonctionnalités supplémentaires pour optimiser notre programme.

2 / Formats pgm et ppm

2.1 / Format pgm

PGM est un format d'images bitmaps créé par Jef Poskanzer en 1988. Il découle du format PBM inventé 8 ans plus tôt.
Ce format est utilisé principalement pour des images constituées de différents niveaux de gris.
Les niveaux de gris sont codés à l'aide de valeur comprise entre 0 et x. x est la valeur maximale de l'intensité choisie.
Par définition, un pixel noir est représenté par la valeur 0 tandis qu'un pixel blanc est lui représenté par la valeur de x.

exemple du mot feep en différentes nuances de gris :

P2 (Numéro magique)
24 7 (dimension de l'image)
15 (nombres de nuances de gris)
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 15 0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11 0 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Résultat :

2.2 / Format ppm

Le format ppm est utilisé pour les images en couleur. La couleur est codée par un nombre à trois chiffre, correspondant au RVB, la distribution entre le Rouge, le Vert et le Bleu.
Il est identique au format pgm dans sa structure, seul le nombre de nuances de gris est ici remplacé par le codage des niveaux de couleur. (Remarque : Cette valeur de codage doit être inférieure à 65536)

exemple :

```
P3
# Le P3 signifie que les couleurs sont en ASCII,
# par 3 colonnes et 2 lignes,
3 2
# ayant 255 pour valeur maximum, et qu'elles sont en RVB.
255
255     0   0       0 255     0     0   0 255
255 255   0   255 255 255     0   0     0
```

Résultat :

3 / Tutoriel sur les matrices avec JAVA

1. Déclaration

En Java un tableau peut être de différents types, autant que de types de variables.

La forme générale d'une déclaration de tableau en java est la suivante :

<type du tableau> <nom du tableau> [] = { <contenu du tableau> } ;

Par exemple, pour créer un tableau d'entiers :

int tableau_Entiers[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ;

Ou encore un tableau de chaîne de caractère :

String tableau_Chaines[] = {''chaine1'', ''chaine2''} ;

Ainsi on peut créer des tableaux **bidimensionnels,** de manière très simple.

Par exemple avec un tableau à deux dimensions d'entiers :

int tableau2D_Entiers[][] = {{0, 1, 2} , {3, 4, 5} } ;

On peut aussi définir la taille de chaque « tableau » dans le tableau bidimensionnel sans avoir à l'initialiser avec des valeurs.

Int tableau2D[5][5] ;

Ceci nous donnera un tableau de taille 5 par 5, mais il est pour l'instant vide.

2. Affectation

On peut accéder à la valeur d'une case d'un tableau grâce à son indice. (Attention les indices d'un tableau en Java commencent à 0)

Par exemple, tableau_Chaines[1] correspond à ''chaine2''.

Pour un tableau bidimensionnel c'est le même principe :

tableau2D_Entier[1][2] = 6 ;

3. Initialisation

Pour initialiser toutes les valeurs du tableau bidimensionnel, il faut utiliser deux boucles ''for'' imbriquées.

Par exemple :

```
int tableau2D_Entiers[2][3] ;

for(int i = 0 ; i<2 : i++){
        for(int j = 0 ; j <3 ; j++){
                tableau2D_Entiers[i][j] = 0 ;
        }
}
```

Au final notre tableau ressemblera à : { {0, 0, 0} , {0, 0, 0} }

4. Parcours

Pour parcourir le tableau bidimensionnel il suffit, comme pour l'initialisation, d'utiliser deux boucles ''for'' imbriquées.

4 / Création d'un tableau bidimensionnel en ada

1. Déclaration

La déclaration se fait très facilement de la manière suivante :

type Nom_matrice is array(1..N,1..M) of type_élément ;

Le premier intervalle correspond au nombre de lignes, le second au nombre de colonnes.

2. Initialisation

L'initialisation se fait à l'aide de deux boucles FOR imbriquées l'une dans l'autre.
Exemple d'une initialisation à 0:

```
for i in 1..N loop
 for j in 1..M loop
    T(i,j) := 0 ;
 end loop ;
end loop ;
```

3. Parcours

Pour parcourir la matrice, on utilise également l'imbrication des deux boucles FOR. On peut également indiquer le début et la fin du parcours afin de ne pas parcourir entièrement la matrice.

4. Affectation

Pour changer la valeur d'une case dans la matrice, on utilise :

Nom_matrice (x,y):= a

Avec : x l'abscisse de la case
y : l'ordonnée de la case
a : la valeur à mettre dans la case (x,y)

Pour changer la valeur de plusieurs cases qui ont même abscisse ou ordonnée, on utilise

```
For i in a..b loop
        Nom_matrice(c, i):= e   (ou Nom_matrice(i,d):=e)
end loop
```

Avec : a..b l'intervalle dans lequel on veut changer les valeurs des cases
c la ligne où on veut modifier les cases
d : la colonne où on veut modifier les cases
e la valeur à mettre dans cette case

5 / Utilisation des logiciels sur le terminal

Synthétiseur (Ada) :

Compilation
tapez make dans le terminal

Exécution
tapez : »./ synthese --Taille nombre_ligne nombre_colonne –-nom_figure dimensions –Couleur
voir dans le Readme pour les commandes exactes

Analyseur (Java) :

Compilation

javac Analyseur_Image.java Matrice.java

Exécution

java -cp . Analyseur_Image < image.ppm

6 / Récapitulatif des séances de TP

Séance 7 :

- Julien et Jean-Baptiste : Avancement du programme de Synthèse (Ada)
- Antoine : Avancement du programme d'Analyse (Java) et réorganisation du compte rendu.

Séance 8 :

- Julien et Jean-Baptiste : Avancement du programme de Synthèse (Ada), remplissage du triangle, implantation des commandes du terminal.
- Antoine : Avancement du programme d'Analyse (Java), reconnaissance d'un rectangle en pgm, et mise à jour du compte rendu.

Séance 9 :

- Julien et Jean-Baptiste : Avancement du programme de synthèse (Ada), problème avec la synthèse du triangle, passage d'une base verticale à une base horizontale.
- Antoine : Avancement du programme d'analyse (Java), reconnaissance d'un cercle en pgm, et mise à jour du compte rendu.

Séance 10 :

- Julien et Jean-Baptiste : Avancement du programme de synthèse (Ada), problème avec la synthèse du triangle.
- Antoine : Avancement du programme d'analyse (Java), reconnaissance d'un triangle en pgm, et mise à jour du compte rendu.

7 / Fonctionnalités supplémentaires

Dans le programme de synthèse d'image, nous avons implanté trois fonctionnalités supplémentaires.

La première consiste à pouvoir donner une couleur au fond de l'image pour qu'elle ne soit pas blanche. Pour cela, nous avons ajouté trois variables pour le fond de l'image et nous mettons ces valeurs lorsque l'on initialise l'image. Afin de l'utiliser, il suffit de taper « --Fond » suivit de la couleur voulu à la fin de la séquence d'instruction dans le terminal.
Ex : ./synthese --Cercle 50 50 20 --Fond Rouge

La deuxième consiste à avoir une nouvelle forme que l'on puisse dessiner. Nous avons donc choisi d'ajouter une procédure pour former une droite en utilisant l'algorithme de bresenham. Pour l'utiliser il suffit de taper « --Droite » suivit des deux points de son extrémité.
Ex : ./synthese --Droite 10 10 100 100

La troisième a pour but de pouvoir dessiner plusieurs dessins sur la même image. Pour cela, il a fallu doubler les variables dont nous avons besoin pour tracer une deuxième figure. Nous pouvons donc maintenant avoir deux figures différentes ou identiques sur la même figure.
Ex : ./synthese --Rectangle 10 10 30 40 --Bleu ---Triangle 5 5 25 5 15 15 ---Jaune

8 / Documentation

Liens utiles Projet :

- Wikipedia Info424 (lama) :

http://lama.univ-savoie.fr/mediawiki/index.php/INFO424_:_Projet_en_informatique

Liens utiles JAVA :

- Description format .pgm : https://www.enseignement.polytechnique.fr/informatique/profs/Philippe.Chassignet/PGM/index.html
- Lecture de .pgm : https://www.enseignement.polytechnique.fr/informatique/profs/Philippe.Chassignet/PGM/pgm_java.html
- Documentation java : http://docs.oracle.com/javase/7/docs/api/

Liens utiles ADA

- Documentation : http://www.adacore.com/developers/documentation
  http://beru.univ-brest.fr/~singhoff/DOC/LANG/ADA/adadistilled.pdf