INF241 - Projet, année 2013-2014
Formats et modification d'images

L'objectif de ce projet est de travailler sur le codage d'une image, de transformer ce codage d'un espace colorimétrique classique (RVB) vers un autre espace colorimétrique (YUV), et de convertir un format classique (`PPM`) en un autre format (`SF`).

1 Déroulement du projet

Pour ce projet d'INF241, vous devrez travailler de vous même sur le sujet, et vous pourrez poser vos questions aux enseignants quand vous êtes incapable de trouver la réponse. Privilégiez pour cela les séances encadrées (TD et TDe).

1.1 Planning

Le projet commence en semaine 11 (semaine du 10/03) par une séance de TD et de TDe d'introduction du présent sujet. Le projet se termine semaine 19 (semaine du 05/05) par une présentation en séance de TDe. Entre les deux, vous devez travailler en dehors des séances.

Une page web pour le projet est disponible, elle contiendra un ensemble de détails nécessaires à la réalisation de ce projet. Elle sera mise à jour régulièrement au fur et à mesure de votre progression. Pensez à la consulter régulièrement.

2 Codage d'une image

2.1 Format d'image PixMap (`PPM`)

Le format d'image PixMap, plus particulièrement le `PPM` en mode *plain* est un des formats de représentation d'image les plus simples.

Il est simplement constitué d'une entête, suivie de la description de chaque pixel de l'image codé sous la forme de trois valeurs : « niveau de rouge »(R) , « niveau de vert » (V) et « niveau de bleu » (B). Chacun de ces triplets est écrit en ASCII dans le fichier. L'exemple de la figure 1 donne un aperçu du contenu du fichier.

```
P3
3 2
255
255 255 255 255 255 255 48 53 250
 48  53 250 255 255 255 80 84 251
```

FIGURE 1 – Exemple d'image `PPM`

Dans cet exemple, on observe que l'entête commence par « P3 », ce qui indique qu'il s'agit bien d'un fichier au format `PPM`. On retrouve ensuite les dimensions en largeur (ℓ) et en hauteur (h) de l'image (ici $\ell = 3$ et $h = 2$). Pour finir, l'entête contient la profondeur maximum d'une composante (ici 255).

La suite du fichier est constituée des informations correspondantes à chacun des pixels de l'image parcourue ligne par ligne en largeur puis en hauteur. Chaque pixel est représenté par les trois valeurs : la valeur de sa composante rouge, la valeur de sa composante verte et enfin la valeur de sa composante bleue. Par exemple, le pixel de coordonnées $(0, 0)$ est de couleur blanche ($R = 255$, $V = 255$ et $B = 255$) et la couleur du pixel de $(2, 1)$ est codée par le triplet $R = 80$, $V = 84$ et $B = 251$.

Des petits exemples de fichier `PPM` sont fournis sur la page Web du projet (`http://www-verimag.imag.fr/~devismes/WWW/enseignements.html#INF241`). Utilisez-les au début du projet. Vous pourrez par la suite générer vos propres fichiers `PPM` en utilisant des outils comme gimp ou équivalent.

Avertissement

- Retirez les lignes de commentaires ajoutées par ces outils en éditant les fichiers `PPM`. Ces lignes commencent par le caractère '#'.
- Sauvegardez votre image en format de données `ASCII`.

2.2 Un format plus compact : le format `.sf`

Nous proposons le format `SF` qui permet de ne pas représenter tous les pixels. Tout d'abord, dans ce format, on ne stocke pas les pixels de couleur blanche (ainsi, la couleur par défaut sera le blanc). Ensuite, on stocke pour chaque couleur utilisée dans l'image la liste des pixels ayant cette couleur. Ainsi pour certaines images, la taille du fichier `SF` sera plus petite que celle du fichier `PPM` correspondant.

Un fichier `.sf` commence par l'entête « SF », ce qui indique qu'il s'agit bien d'un fichier au format `SF`. On retrouve ensuite les dimensions en largeur (ℓ) et en hauteur (h) de l'image. Pour finir, l'entête contient la profondeur maximum d'une composante. Ensuite, chaque ligne représente la liste des pixels d'une couleur : une ligne commence par un code couleur RVB, suivi de la liste des pixels ayant cette couleur.

Par exemple, pour l'image de la figure 1, le format `.sf` correspondant est donné dans la figure 2. La cinquième ligne nous indique que la couleur codée par $R = 80$, $V = 84$, $B = 251$ n'apparaît que sur le pixel $(2, 1)$. La sixième ligne nous indique que la couleur codée par $R = 48$, $V = 53$, $B = 250$ apparaît que sur les pixels $(0, 1)$ et $(2, 0)$. Les autres pixels sont blancs.

2.3 Affichage des images

Nous fournissons un logiciel permettant de visualiser les images au format `PPM` et `SF` (`http://www-verimag.imag.fr/~devismes/WWW/enseignements.html#INF241`).

Pour l'appeler : `./Viewer image.ppm` ou `./Viewer image.sf`.

```
SF
3
2
255
80 84 251 2 1
48 53 250 0 1 2 0
```

FIGURE 2 – Exemple d'image `SF`

3 Les espaces colorimétriques : RVB et YUV

3.1 Espace colorimétrique RVB

Comme vous avez pu le voir en cours, un des codages des couleurs utilisé sur les ordinateurs est la représentation RVB (rouge, vert et bleu). Cette représentation est basée sur les trois couleurs primaires en synthèse additive.

Les couleurs sont codées sous la forme de leur intensité pour chacune des composantes, et cette couleur est donc obtenue en additionnant ces trois composantes. Le blanc est ainsi codé par l'intensité maximum de chacune des trois composantes, alors que le noir est la somme des intensités minimum (zéro).

Lorsque l'on code une couleur sur 24 bits, il y a 8 bits pour chacune des composantes, donc 256 niveaux de rouge, 256 niveaux de vert et 256 niveaux de bleu. Voici quelques exemples de couleur classiques (*c.f.*, premier cours d'INF241) :

Couleur	R	V	B
Rouge	255	0	0
Vert	0	255	0
Bleu	0	0	255
Noir	0	0	0
Blanc	255	255	255

TABLE 1 – Exemple de représentation RVB

3.2 Espace colorimétrique YUV

L'espace colorimétrique RVB n'est pas complètement adapté à la sensibilité de l'oeil humain. On a pour cela une famille d'espaces colorimétriques plus adaptée. On compte parmi ces espaces, l'espace YUV. Y est la luminance, U et V sont des chrominances. La luminance est par exemple le signal d'une télévision noir et blanc et les chrominances sont des moyens de coder les informations de couleurs (U chrominance bleu et V chrominance rouge). L'origine de cette représentation est d'ailleurs la transmission télévisuelle.

Le passage de l'espace colorimétrique RVB vers l'espace colorimétrique YUV est un simple changement de base. Les coefficients de ce changement de base sont choisis pour refléter la sensibilité de l'oeil humain.

$$
\begin{aligned}
Y &= 0,299 \times R + 0,587 \times V + 0,114 \times B \\
U &= -0,147 \times R - 0,289 \times V + 0,436 \times B = 0,492 \times (B - Y) \\
V &= 0,615 \times R - 0,515 \times V - 0,100 \times B = 0,877 \times (R - Y)
\end{aligned}
$$

3.3 Calcul des composantes YUV

Comme vous avez pu le voir précédemment, le calcul des composantes YUV utilise des coefficients qui sont des nombres à virgule. Malheureusement, le processeur ARM que nous utilisons en TD et TDe ne sait pas faire de tels calculs. Pour permettre ce genre de calcul nous allons alors avoir recours au calcul à virgule fixe, qui est une réalisation de calcul de nombres à virgule en utilisant des entiers.

La formule de conversion d'espace colorimétrique RVB vers YUV devient alors en pseudo code :

$$
\begin{aligned}
Y &= 77 \times R + 150 \times V + 29 \times B \\
U &= -38 \times R - 74 \times V + 112 \times B \\
V &= 157 \times R - 132 \times V - 26 \times B
\end{aligned}
$$

$$
\begin{aligned}
Y &= (Y + 128) \gg 8 \\
U &= (U + 128) \gg 8 \\
V &= (V + 128) \gg 8
\end{aligned}
$$

$$
\begin{aligned}
Y &= Y + 16 \\
U &= U + 128 \\
V &= V + 128
\end{aligned}
$$

4 Passage d'un format à un autre

Ce projet est organisé en étapes de difficultés croissantes :

4.1 Première étape : changement de codage des couleurs

Il s'agit ici passer du codage des couleurs RVB au codage des couleurs YUV sans modifier la structure du codage des pixels.

Pour ce travail nous vous suggérons les étapes suivantes :

1. Lire un fichier `PPM` et réécrire son contenu à l'écran.
   Pour simplifier les choses, vous lirez le fichier d'entrée comme s'il était tapé au clavier (lecture sur l'entrée standard). Vous utiliserez donc pour cela les fonctions que vous

avez déjà vues en TDe, à savoir celles qui sont dans le fichier es.s (décrites dans l'annexe 3 du poly des sujet de TD et TDe). Vous utiliserez alors votre programme en redirigeant l'entrée standard et en utilisant le fichier qui vous intéresse à la place de la manière suivate : `arm-eabi-run monProg < fichierTest.ppm`

2. Programmer une fonction qui étant donné trois valeurs R, V, B produit les valeurs Y, U, V. La règle de calcul est donnée dans le paragraphe 3.3.
3. Lire un fichier `PPM` et produire un fichier au même format mais en produisant les composantes YUV en lieu et place des composantes RVB du fichier initial. Ce fichier sera produit sur la sortie standard (avec les fonction de es.s). Vous devrez alors rediriger la sortie standard vers un fichier de la manière suivante : `arm-eabi-run monProg > fichierSortie.ppm`

4.1.1 Mise au point de votre programme

Avec les deux redirections (entrée et sortie standard), il peut sembler compliqué de mettre au point un programme avec `gdb`. Il n'en est rien, il suffit de placer les redirections au bon endroit, à savoir sur la commande `run` comme suit :

```
$ arm-eabi-gdb monProg
[...]
> run < fichierTest.ppm > fichierSortie.ppm
```

4.2 Deuxième étape : changement de structure du fichier

Il s'agit ici de transformer une image au format `PPM` en image au format `SF`. L'idée est d'économiser des pixels et en particulier de ne pas représenter les pixels de couleur blanche (`255 255 255`). Nous allons représenter l'ensemble des pixels « non-blancs » par une liste de listes : nous avons une liste de couleurs et pour chaque couleur une liste de points (coordonnées des points) ayant cette couleur.

Nous pouvons décrire le format de la façon suivante :

```
Couleur : structure {
    couleur_suivante : adresse de la Couleur suivante
    liste_pixels : adresse du premier Point
    R : octet
    V : octet
    B : octet
}
```

```
Point : structure {
    point_suivant : adresse du Point suivant
    x : entier
    y : entier
}
```

La figure 3 montre comment le contenu du fichier `SF` fourni dans la figure 2 doit être représenté en mémoire centrale.

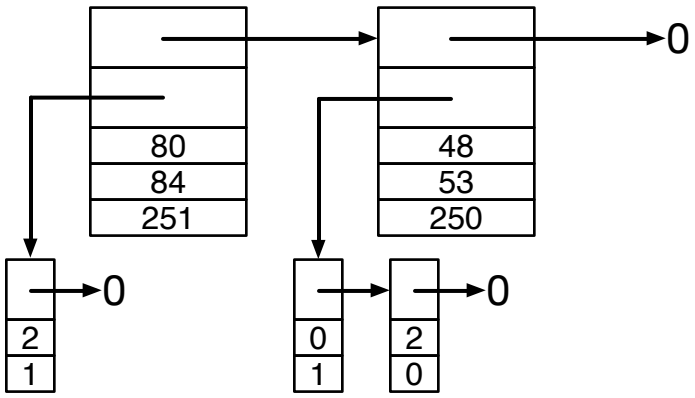

FIGURE 3 – Structure représentant le fichier `SF` de la figure 2

Pour cela nous vous suggérons les étapes intermédiaires suivantes :

1. Lire une séquence d'entiers de longueur paire, générer une liste de couples correspondante, puis l'afficher. Pour cela utiliser les fonctions définies dans le module `listeCouples.s` disponible sur le site du projet.
2. Générer une structure de données en mémoire centrale représentant en mémoire une image au format `SF`. Vous vous baserez sur les algorithmes donnés dans la section 5.3.
3. Lire un fichier `PPM` et le traduire en un fichier `SF`. Pour cela, vous devez lire le fichier `PPM` et remplir au fur et à mesure votre structure en mémoire centrale. Ensuite, vous afficherez votre structure sur la sortie standard en respectant le format `SF`. Ainsi, la conversion sera opérée via la commande suivante : `arm-eabi-run monProg < fichierTest.ppm > fichierSortie.sf`

Nous vous fournissons les modules `allocation.s` et `listeCouples.s` qui permettent respectivement d'allouer de la mémoire de façon dynamique et de gérer une liste de couples d'entiers (`http://www-verimag.imag.fr/~devismes/WWW/enseignements.html#INF241`).

5 Annexes

5.1 Module `allocation.s`

Ce module contient la fonction `MonMalloc` de profil `fonction MonAlloc (T : un entier) : adresse` qui alloue une zone de mémoire de `T` octets et rend l'adresse de cette zone.

```
@ Fonction MonMalloc
@ parametre : r0 = taille en octet T
@ retour : r0 = AD, ou AD est l'adresse d'un bloc de taille T
```

5.2 Module `listeCouples.s`

Une liste est représentée par l'adresse de son premier élément. Une liste vide est représenté par l'adresse 0.

La fonction `AjoutListeCouple` permet d'ajouter un couple d'entiers dans une liste. Son profil est : `AjoutListeCouple(L : liste, x : entier, y : entier) : liste`. Elle prend ainsi trois paramètres : l'adresse de la liste dans laquelle on veut ajouter le couple formé des entiers `x` et `y` : `(x,y)`.

Les paramètres seront passés par la pile. Les paramètres seront empilés de gauche à droite (ordre inverse de la convention utilisée par gcc) : le paramètre de droite (y) occupant le sommet (adresse la plus basse) du bloc de paramètres dans la pile.

La fonction empilera son résultat au-dessus des paramètres. La libération de l'espace de stockage du résultat et des paramètres est du ressort de l'appelante. Ainsi, la fonction `AjoutListeCouple` rend en sommet de pile l'adresse de la nouvelle liste.

```
@ Fonction AjoutListeCouple
@ parametres : une adresse et deux entiers sur la pile dans cet ordre
@ retour : une adresse en sommet de pile
```

La procédure `AfficheListeCouple` permet d'afficher le contenu d'une liste de couples d'entiers séparés par des espaces. L'adresse de la liste doit être passée en paramètre dans la pile.

```
@ Fonction AfficheListeCouple
@ parametres : une adresse sur la pile
```

5.3 Algorithmes à mettre en œuvre

Nous utilisons ici les structures de données définies dans le paragraphe 4.2. Dans les algorithmes proposés ci-dessous : si `L` est différent de 0 alors `L` est l'adresse du premier élément d'une liste de `Couleur` sinon la liste L est vide !

```
@ affichage de la liste L
procedure Affiche(L : liste)

     si L != 0 alors
```

```
        @ Ecrire sans passage à la ligne
        Ecrire mem[L].R
        Ecrire ' '
        Ecrire mem[L].V
        Ecrire ' '
        Ecrire mem[L].B
        AfficheListeCouple(mem[L].liste_pixels)
        Ecrire '\n'
        Affiche(mem[L].couleur_suivante)
     fin si
fin

@ recherche d'une Couleur R,V,B dans la liste L
fonction Recherche(L : liste, R : octet, V : octet, B : octet) : liste

     si L = 0 oualors (mem[L].R = R et mem[L].V = V et mem[L].B = B) alors
        renvoyer L
     sinon
        renvoyer Recherche(mem[L].couleur_suivante)
     fin si
fin

@ Ajout d'une couleur a la liste L, avec une liste de Point vide
fonction Nouveau(L : liste, R : octet, V : octet, B : octet) : liste

     AD:= MonMalloc(taille d'un élément Couleur)

     mem[AD].couleur_suivante := L
     mem[AD].liste_pixels := 0
     mem[AD].R := R
     mem[AD].V := V
     mem[AD].B := B

     renvoyer AD
fin

@ Ajout dans la liste L d'un point de coordonnees (x,y),d e couleur R,V,B
fonction Ajout(L : liste, R : octet, V : octet, B : octet, x : entier, y : entier) : liste

     AD:=Recherche(L,R,V,B)
     ret := L

     if AD = 0 alors
        AD:=Nouveau(L,R,V,B)
        ret := AD
     fin si

     mem[AD].liste_pixels := AjoutListeCouple(mem[AD].liste_pixels,x,y)

     renvoyer ret

fin
```

5.4 Pour aller plus loin : conversion YUV vers RVB

Si vous souhaitez jouer avec la représentation YUV et observer les modifications des couleurs engendrées, il vous faudra repasser dans l'espace colorimétrique RVB. Pour cette conversion inverse, on utilise alors les formules en virgule fixe suivantes :

$$\begin{aligned} C &= Y - 16 \\ D &= U - 128 \\ E &= V - 128 \end{aligned}$$

$$\begin{aligned} R &= 298 \times C + 409 \times E \\ V &= 298 \times C - 100 \times D - 208 \times E \\ B &= 298 \times C + 516 \times D \end{aligned}$$

$$\begin{aligned} R &= \text{sat}((R + 128) \gg 8) \\ V &= \text{sat}((V + 128) \gg 8) \\ B &= \text{sat}((B + 128) \gg 8) \end{aligned}$$

L'opération `sat`, est une opération de saturation. Si la valeur est supérieure au maximum représentable on la fixe au maximum (ici 255), si elle est inférieure au minimum on la fixe au minimum (ici 0).