Projet : Image, Rotation, Zoom et Interpolation

Notions : Allocation dynamique, fichier, structure,travail en groupe

L'objectif de ce projet est de réaliser des transformations simples d'images comme les rotations et les homothéties avec différentes méthodes d'interpolation et de les comparer.

Ce projet se réalise en binôme et demande un bonne coordination entre les 2 membres du binôme. Il faut commencer par analyser le problème, définir les structures de données, les différentes fonctions, leurs prototypes et leur rôle exact. Vous pouvez ensuite répartir le travail entre les 2 membres du binôme. Ce projet nécessite un peu de travail en dehors des séances encadrées.

1 Quelques éléments sur l'image

Une image est une fonction $I(x, y)$ à support fini (ses dimensions) et dont les valeurs sont ici des scalaires sur 8 bits : le noir correspond à une valeur nulle, le blanc à 255.

En informatique, c'est simplement un tableau d'octets non signés (`unsigned char`) à 2 dimensions.

Vous avez figure 1 deux tableaux représentant les valeurs de l'image du tangram sur un voisinage de 6x6 pixels dans le fond et sur un sommet de triangle. Les valeurs de l'intensité ne sont pas uniformes et comportent un aléa dû au bruit d'acquisition.

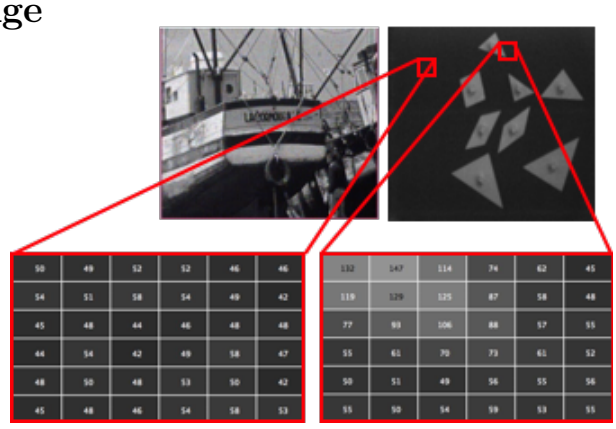

FIGURE 1 – Images, octets et valeurs numériques

2 Rotation et zoom d'image

Nous prenons ici l'exemple d'une rotation d'image. Soit $I(i, j)$ l'image d'origine et $Z(k, l)$ l'image ayant subi la rotation. Pour un angle a et un centre de rotation i_c, j_c, le pixel de l'image d'origine i, j se retrouve déplacé aux coordonnées :

$$\begin{cases} x' = i_c + (i - i_c) * cos(a) - (j - j_c) * sin(a) \\ y' = j_c + (i - i_c) * sin(a) + (j - j_c) * cos(a) \end{cases}$$

c'est à dire que la valeur de la nouvelle image au point x', y' est celle de l'ancienne image au point i, j. Comme x' et y' ne sont généralement pas des entiers, il est impossible de positionner ce pixel dans la nouvelle image, dont les coordonnées sont obligatoirement entières.

Dans ce cas, on procède en parcourant tous les pixels de l'image résultat (celle qui est tournée et dont on ne connaît pas les valeurs) en cherchant le pixel qui est son antécédent dans l'image initiale. On utilise alors la transformation inverse (rotation ou zoom inverse). Pour une rotation, le pixel de

coordonnées entières k, l de l'image transformée par la rotation est celui qui, dans l'image d'origine, avait les coordonnées suivantes :

$$\begin{cases} x = i_c + (k - i_c) * cos(a) + (l - j_c) * sin(a) \\ y = j_c - (k - i_c) * sin(a) + (l - j_c) * cos(a) \end{cases} \quad (1)$$

Pour le zoom de valeur z , c'est celui qui avait les coordonnées

$$\begin{cases} x = i_c + 1/z * (k - i_c) \\ y = j_c + 1/z * (l - j_c) \end{cases} \quad (2)$$

C'est la valeur de ce pixel de coordonnées x, y que l'on devrait mettre dans le pixel k, l de l'image transformée $Z(k, l)$. Là encore, les coordonnées x et y ont peu de chances d'être des entiers et les seules valeurs connues dans l'image d'origine sont les pixels de coordonnées entières. Le point x, y est un point situé entre 4 points de coordonnées entières définies par $i = \lfloor x \rfloor$, $i + 1$, $j = \lfloor y \rfloor$, $j + 1$, avec $\lfloor x \rfloor$:partie entière de x. La valeur $I(x, y)$ du pixel x, y est donc inconnue et on va utiliser une interpolation pour la calculer.

Conclusion : il faut parcourir l'image résultat pixel par pixel (ie le tableau 2D élément par élément) et calculer pour chaque pixel de coordonnées k, l de cette image résultat la valeur de son antécédent, de coordonnées x, y. Connaissant ces coordonnées, on calcule la valeur $Z(k, l)$ par interpolation.

3 Interpolation

Pour évaluer l'image d'origine en un point dont les coordonnées ne sont pas entières, on procède par interpolation. On considère la fonction image comme une fonction localement constante (ordre 0), ou affine (ordre 1), ou cubique (ordre 3).

On note k, l les coordonnées du pixel de l'image avec rotation (ou zoom) puis x, y celles du pixel dans l'image initiale. Les valeurs x, y sont obtenues à partir des valeurs k, l selon les équations 1 (ou 2).

Dans les 3 types d'interpolation présentés ci-dessous, les formules ne prennent pas en considération les problèmes des bords de l'image. Si des coordonnées sont négatives ou supérieures aux dimensions de l'image, les pixels n'existant pas, leur valeur sera considérée comme nulle. On peut aussi adapter les formules pour ces cas particuliers.

Soient x et y les coordonnées réelles du pixel dont on cherche la valeur par interpolation. Dans tous les calculs qui suivent, on définit i (respectivement j) comme la partie entière de x (resp. y) puis di (resp. dj) comme la partie fractionnaire :

$$\begin{cases} i = \lfloor x \rfloor \\ j = \lfloor y \rfloor \\ di = x - i \quad \in [0, 1[\\ dj = y - j \quad \in [0, 1[\end{cases}$$

3.1 Interpolation d'ordre 0

La solution la plus simple est de considérer que l'image a une valeur constante (polynôme d'ordre 0) entre les pixels de coordonnées entières. Par convention, le pixel en $x = i + di, y = j + dj$ (avec $0 <= di < 1$ et $0 <= dj < 1$) prend la valeur du plus proche pixel de coordonnées entières (i ou $i + 1$, j ou $j + 1$).

$$\begin{cases} Z(k, l) = I(i, j) \text{ si } 0 <= di < 0.5 \text{ et } 0 <= dj < 0.5 \\ Z(k, l) = I(i + 1, j) \text{ si } 0.5 <= di < 1 \text{ et } 0 <= dj < 0.5 \\ Z(k, l) = I(i, j + 1) \text{ si } 0 <= di < 0.5 \text{ et } 0.5 <= dj < 1 \\ Z(k, l) = I(i + 1, j + 1) \text{ si } 0.5 <= di < 1 \text{ et } 0.5 <= dj < 1 \end{cases}$$

C'est l'interpolation au plus proche voisin. Les valeurs de x et de y doivent être positives et inférieures aux dimensions de l'image.

3.2 Interpolation d'ordre 1

On approche la fonction image par une fonction linéaire (figure 2).

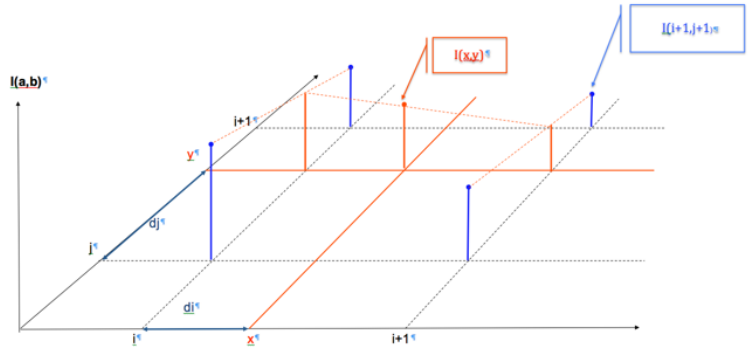

FIGURE 2 – interpolation d'ordre 1

$$\begin{aligned} Z(k,l) =& (1-di)*(1-dj)*I(i,j) + di*(1-dj)*I(i+1,j) \\ &+ (1-di)*dj*I(i,j+1) + di*dj*I(i+1,j+1) \end{aligned}$$

Remarque 1 : telle quelle, l'interpolation pour 1 pixel de l'image coûte 8 multiplications (opération coûteuse) et 7 additions. Comme une image comporte aujourd'hui plusieurs millions de pixels, faire une transformation d'image coûte de l'ordre de 100 millions de multiplications et d'additions. Pensez aussi que pour une image couleur, il faut faire subir la même opération aux 3 plans couleurs R,V,B. Il existe une amélioration efficace à cette manière d'écrire l'interpolation :

$$\begin{cases} H(a,b,c) = a + (b-a)*c \\ R_1(i,j) = H(I(i,j), I(i,j+1), dj) \\ R_2(i,j) = H(I(i+1,j), I(i+1,j+1), dj) \\ Z(k,l) = H(R_1(i,j), R_2(i,j), di) \end{cases}$$

On obtient alors la valeur de l'image en x,y avec un coût de 3 multiplications et 6 additions par pixels, en économisant de nombreuses multiplications.

Remarque 2 : il faut gérer les bords attentivement. Cette formule ne s'applique que pour x, y, i et j positifs et plus petits que l'image.

Dans le cas contraire, c'est un pixel sur le bord de l'image et on pourra utiliser :

$$\begin{cases} Z(k,l) = (1-dj)*im[i][j] + dj*im[i][j+1] \text{ si } i = nblignes - 1 \\ Z(k,l) = (1-dj)*im[0][j] + dj*im[0][j+1] \text{ si } i = -1 \\ Z(k,l) = (1-di)*im[i][j] + di*(1-dj)*im[i+1][j] \text{ si } j = nbcolonnes - 1 \\ Z(k,l) = (1-di)*im[i][0] + di*(1-dj)*im[i+1][0] \text{ si } j = -1 \\ Z(k,l) = 0 \text{ si } x <= -1 \text{ ou } y <= -1 \text{ ou } x >= nblignes \text{ ou } y >= nbcolonnes \end{cases}$$

3.3 Interpolation d'ordre 3

On peut aussi utiliser une fonction d'ordre 3 comme dans l'interpolation spline cubique :

$$\begin{cases} C_3(t) = -0.5t^3 + t^2 – 0.5t \\ C_2(t) = 1.5t^3 – 2.5t^2 + 1 \\ C_1(t) = -1.5t^3 + 2t^2 + 0.5t \\ C_0(t) = 0.5t^3 – 0.5t^2 \\ \quad R1 = I(i-1, j-1) * C_3(di) + I(i, j-1) * C_2(di) + I(i+1, j-1) * C_1(di) + I(i+2, j-1) * C_0(di) \\ \quad R2 = I(i-1, j) * C_3(di) + I(i, j) * C_2(di) + I(i+1, j) * C_1(di) + I(i+2, j) * C_0(di) \\ \quad R3 = I(i-1, j+1) * C_3(di) + I(i, j+1) * C_2(di) + I(i+1, j+1) * C_1(di) + I(i+2, j+1) * C_0(di) \\ \quad R4 = I(i-1, j+2) * C_3(di) + I(i, j+2) * C_2(di) + I(i+1, j+2) * C_1(di) + I(i+2, j+2) * C_0(di) \\ Z(k, l) = R1 * C3(dj) + R2 * C2(dj) + R3 * C1(dj) + R4 * C0(dj) \end{cases}$$

Remarque Comme pour l'interpolation bilinéaire, cette interpolation n'est valable que pour des points pour lesquels les coordonnées $i-1, i+2, j-1, j+2$ sont à l'intérieur de l'image.

4 Mesure de qualité

Pour quantifier la différence entre 2 images I_1 et I_2, on peut utiliser le Peak Signal-to-noise-Ratio ($PSNR$). On calcule d'abord l'erreur quadratique moyenne MSE entre les deux images par :

$$\begin{cases} MSE = \dfrac{1}{nbl * nbc} \displaystyle\sum_{i=0}^{nbl-1} \sum_{j=0}^{nbc-1} \left(I_1(i,j) - I_2(i,j)\right)^2 \\ PSNR = 20 * log_{10}\left(\dfrac{255}{\sqrt{MSE}}\right) \end{cases}$$

Le PSNR s'exprime en dB. Plus la valeur est grande, plus la différence entre les 2 images est faible. C'est une mesure souvent utilisée pour comparer les compresseurs d'images entre eux, mais elle ne permet pas de prendre en compte la perception humaine de l'image, qui est plus sensible à certaines caractéristiques qu'à d'autres.

5 Format PGM

Les fichiers images fournis sont au format PGM P5. Un fichier PGM P5 comporte deux parties distinctes :

- un entête contenant des informations en mode texte, que l'on peut lire facilement avec les fonctions fscanf, fgets.
  - Une ligne contenant le magic number c'est à dire la chaîne de caractères "P5".
  - Une ligne de commentaire commençant par '#' : La norme précise qu'il peut y avoir un nombre quelconque de lignes de commentaire commençant par '#', n'importe où dans l'en tête. Les fichiers fournis n'ont toutefois qu'une et une seule ligne de commentaire, juste après la ligne contenant le magic number "P5".
  - Une ligne contenant le nombre de colonnes puis le nombre de lignes
  - Une ligne contenant la plus grande valeur possible d'intensité
- Les valeurs de tous les pixels en binaire. Ils sont rangés dans l'ordre d'apparition sur l'image par un balayage vidéo de gauche à droite, de haut en bas. Comme il s'agit de données binaires, elles doivent être lues avec la fonction fread. Elles ont été écrites avec la fonction fwrite.

6 Travail à réaliser

6.1 Allocation/libération d'image

— Ecrire une fonction de création d'image sous forme d'un tableau à deux dimensions, alloué dynamiquement. La zone de données sera contiguë.
L'entête de la fonction sera `unsigned char** alloueImage(int nlignes, int ncol)`

— Ecrire une fonction qui libère la mémoire allouée pour une image.
L'entête de la fonction sera `void libereImage(unsigned char** im)`

6.2 Lecture/sauvegarde d'images

Vous allez écrire une fonction pour effectuer le chargement de l'image en mémoire dans un tableau et une fonction pour sauvegarder une image dans un fichier au format PGM.

6.2.1 Lecture d'images

Ecrire une fonction `unsigned char ** lectureImagePgmBinaire(char* fic, int* pnbl, int* pnbc)` qui effectue la lecture d'un fichier image PGM et retourne l'image allouée dynamiquement. Elle met aussi à jour les paramètres passés par adresse (nombre de lignes et nombre de colonnes). Elle réalise les actions suivantes :

1. Ouvre le fichier (la chaîne de caractères `fic` contient le nom de l'image).
2. Retourne NULL si le fichier ne peut pas être ouvert
3. Lit la chaîne de caractères Magic Number. Retourne NULL si ce n'est pas la bonne chaîne, i.e. "P5".
4. Saute la ligne de commentaire (utiliser fgets()).
5. Lit le nombre de colonnes et de lignes. Attention à l'ordre.
6. Met à jour les variables pointées par pnbl et pnbc qui doivent contenir les nombres de lignes et de colonnes de l'image
7. Lit la valeur maximale des intensités. Si la valeur est supérieure à 255, quitte la fonction en retournant la valeur NULL.
8. Crée une nouvelle image aux bonnes dimensions à l'aide de votre fonction de création d'image.
9. Lit les valeurs des pixels (fonction fread) et les stocke dans l'image précédemment créée. Quitte la fonction avec la valeur NULL si la valeur d'un pixel ne peut être lue.

Il est conseillé de vérifier chaque donnée lue dans le fichier, avec printf().

6.2.2 Sauvegarde d'images

Ecrire une fonction `void sauvegardeImagePgmBinaire(char* fic, unsigned char** im, int nbl, int nbc)` qui effectue la sauvegarde de l'image im de nbl lignes et nbc colonnes dans le fichier image PGM dont le nom est donné par fic. Il faut simplement écrire les informations dans l'ordre voulu, soit avec des fprintf pour la partie entête, soit des fwrite pour les valeurs de pixels. Ne pas oublier le Magic Number ("P5") en début de fichier.

6.3 Interpolations

1. Faire une fonction qui calcule et retourne la valeur d'une image I au point de coordonnées réelles x et y en utilisant l'interpolation du plus proche voisin.
Son entête sera `unsigned char interpol0 (unsigned char** im1, int nbl, int nbc, double x, double y)`
2. Faire une fonction qui calcule et retourne la valeur d'une image I au point de coordonnées réelles x et y en utilisant l'interpolation bilinéaire.
Son entête sera `unsigned char interpol1 (unsigned char** im1, int nbl, int nbc, double x, double y)`

3. Faire une fonction qui calcule et retourne la valeur d'une image I au point de coordonnées réelles x et y en utilisant l'interpolation spline.
   Son entête sera `unsigned char interpol2 (unsigned char** im1, int nbl, int nbc, double x, double y)`

6.4 Rotation

6.4.1 Fonctions

Faire une fonction qui effectue la rotation d'angle `a` d'une image de `nbl` lignes et `nbc` colonnes et qui retourne la nouvelle image créée. On utilisera l'allocation dynamique. Pour simplifier, le centre de rotation sera le centre de l'image. La dimension de l'image tournée sera la même que celle de l'image initiale. Cette fonction utilisera soit l'interpolation d'ordre 0, d'ordre 1 ou d'ordre 3 en fonction du paramètre `ordre`.

Son entête sera `unsigned char ** rotationImage (unsigned char** im, double a, int ordre, int nbl, int nbc)`

6.4.2 Programme de rotation

Faire un programme qui lit une image dans un fichier, affiche cette image à l'écran, effectue une rotation et affiche le résultat à l'écran.

Vous pouvez vous inspirez du programme suivant qui affiche une image, effectue une rotation et l'affiche dans la même fenêtre.

```
#include <stdio.h>
#include <math.h>
#include <SDL2/SDL_phelma.h>

int main(int ar, char** ag) { int i,j,nbligne,nbcol;
  unsigned char** im=NULL;
  unsigned char** im2=NULL;
  SDL_PHWindow* f=NULL;
  char* fichier = "palmier.pgm";

  /*
    Lecture de l'image au format pgm avec votre fonction
    nbligne contient le nombre de lignes
    nbcol contient le nombre de colonnes
    im contient les donnees de l'image
  */
  im= lectureImagePgmBinaire(fichier,&nbligne,&nbcol);
  if (im==NULL) { printf("Erreur chargement image %s\n",fichier); exit(EXIT_FAILURE) ; }


  /* Creation d'une fenetre graphique de 2*nbligne lignes et nbcol colonnes */
  if( ( f=SDL_PH_CreateWindow(nbcol,2*nbligne)) == NULL) {
    fprintf(stderr,"Creation de fenetre impossible : %s\n", SDL_GetError());
    exit(EXIT_FAILURE);
  }

  /* Affichage de l'image a partir du coin haut gauche (0,0)  dans la fenetre */
  SDL_PH_DrawIm8(f,im,nbligne,nbcol,0,0);

  /* Et maintenant, on attend */
  puts("Taper pour continuer"); getchar();

  im2=rotationImage(im,20.0/180*3.14,1,nbligne,nbcol);

        /* On affiche cette nouvelle image sur l'ecran, en dessous de la premiere, en nbligne,0 */
  SDL_PH_DrawIm8(f,im2,nbligne,nbcol,0,nbligne);

  /* Et maintenant, on attend */
  puts("Taper pour continuer"); getchar();

      /* Sauvegarde de l'image dans le fichier rotation.pgm */
```

```
  im= sauvegardeImagePgmBinaire(fichier,im,nbligne,nbcol);

  /* Liberation memoire de l'image 2D */
 libereImage(im);

  /* Destruction de la fenetre graphique SDL Phelma */
  SDL_PH_DestroyWindow(f);
  SDL_Quit();
  exit(EXIT_SUCCESS);
}
```

La compilation se fera à l'aide d'un fichier `Makefile` avec la commande `make rotation` ou `make zoom` par exemple. Si votre programme comporte les fichiers `rotation.c` (qui peut contenir la fonction main) et les fichiers `f1.c f2.c` qui contiennent les différentes fonctions écrites par chaque membre du binôme, le fichier `Makefile` a l'allure suivante :

```
# Pour les machines de phelma
DIRSDL=/users/prog1a/C/librairie/SDL2_Phelma

# Pour le compilateur
CFLAGS= -c -g  -I$(DIRSDL)/include  -Wall
LDFLAGS= -L$(DIRSDL)/lib -lSDL2 -lSDL2_gfx -lSDL2_ttf -lSDL2_image -lSDL2_phelma -lm -lSDL_sound

# tous les fichiers executables de ce projet
EXE= rotation zoom

# Les regles de construction des executables

rotation: rotation.o f1.o f2.o
  clang -o $@ $^  $(LDFLAGS)

zoom: zoom.o f1.o f2.o
  clang -o $@ $^  $(LDFLAGS)

all: $(EXE)

# Les regles de compilation
%.o: %.c
  clang $(CFLAGS) $<

# Pour faire le menage
clean :
  rm -f $(EXE) *.o
```

6.5 Zoom

6.5.1 Fonctions

Faire une fonction qui effectue le zoom de valeur `val` d'une image `im` de `nbl` lignes et `nbc` colonnes. Cette fonction utilisera l'interpolation d'ordre 0, 1 ou 3, en fonction du paramètre `ordre`.

Son entête sera `unsigned char ** zoomImage(unsigned char** im, double val, int ordre, int nbl, int nbc)`

6.5.2 Programme de zoom

Faire un programme qui lit une image dans un fichier, affiche cette image à l'écran, effectue un zoom de cette image et affiche cette image à l'écran.

FIGURE 3 – Image, rotation de 20 degrés, rotation inverse : on ne retrouve pas complètement l'image

6.6 PSNR et qualité

6.6.1 Fonctions

Lors d'une rotation, une partie de l'image n'est pas calculable : les points proches des angles n'ont pas d'antécédents dans l'image I (voir figure 3).

Faire une fonction qui calcule le PSNR entre deux images `im1` et `im2`, à l'intérieur d'un rectangle défini par les points de coordonnées `i1,j1` et `i2,j2`. Son entête sera `double ** psnrImage (unsigned char** im1, unsigned char** im2, int nbl, int nbc, int i1, int j1, int i2, int j2)`

6.6.2 Programme

Faire un programme qui calcule le PSNR entre une image initiale et une image qui a subi N rotations de 1 degré horaire puis N rotations de 1 degré anti-horaire dans les 3 cas d'interpolations. Limitez le calcul aux parties qui ont été calculables (ne pas prendre en compte les bandes noires sur les coins qui n'ont pas pu être calculées : voir figure 3)