Introduction à la Programmation

—

TP #6

—

G1: monasse*(at)*imagine.enpc.fr
G2: nicolas.audebert*(at)*onera.fr
G3: alexandre.boulch*(at)*onera.fr
G4: maxime.ferrera*(at)*onera.fr
G5: laurent.bulteau*(at)*u-pem.fr
G6: pierre-alain.langlois*(at)*eleves.enpc.fr

1 Images

FIGURE 1 – Deux images et différents traitements de la deuxième (négatif, flou, relief, déformation, contraste et contours).

Dans ce TP, nous allons jouer avec les tableaux bidimensionnels statiques (mais stockés dans des tableaux 1D) puis dynamiques. Pour changer de nos passionnantes matrices, nous travaillerons avec des images (figure 1).

1.1 Allocation

1. *Récupérer le projet :*
   Télécharger le fichier `Tp7_Initial.zip` sur la page habituelle, le décompresser et lancer Visual C++.
2. *Saturer la mémoire :*
   Rien à voir avec ce qu'on va faire après mais il faut l'avoir fait une fois...Faire, dans une boucle infinie, des allocations de 1000000 entiers sans désallouer et regarder la taille du process grandir. (Utiliser `Ctrl+Shift+Echap` pour accéder au gestionnaire de tâches). Compiler en mode Release pour utiliser la "vraie" gestion du tas (Le mode Debug utilise une gestion spécifique qui aide à trouver les bugs et se comporte différemment...)

1.2 Tableaux statiques

3. *Niveaux de gris :*
Une image noir et blanc est représentée par un tableau de pixels de dimensions constantes `W=300` et `H=200`. Chaque pixel `(i,j)` est un `byte` (entier de 0 à 255) allant de 0 pour le noir à 255 pour le blanc. L'origine est en haut à gauche, `i` est l'horizontale et `j` la verticale. Dans un tableau de `byte` mono-dimensionnel `t` de taille `W*H` mémorisant le pixel `(i,j)` en `t[i+W*j]` :
— Stocker une image noire et l'afficher avec `putGreyImage(0,0,t,W,H)`.
— Idem avec une image blanche.
— Idem avec un dégradé du noir au blanc (attention aux conversions entre `byte` et double).
— Idem avec $t(i,j) = 128 + 128\sin(4\pi i/W)\sin(4\pi j/H)$ (cf figure 1). Utiliser

```
#define _USE_MATH_DEFINES
#include <cmath>
```

pour avoir les fonctions **et les constantes** mathématiques : `M_PI` vaut π.

4. *Couleurs :*
Afficher, avec `putColorImage(0,0,r,g,b,W,H)`, une image en couleur stockée dans trois tableaux `r`, `g` et `b` (rouge, vert, bleu). Utiliser la fonction `click()` pour attendre que l'utilisateur clique avec la souris entre l'affichage précédent et ce nouvel affichage.

1.3 Tableaux dynamiques

5. *Dimensions au clavier :*
Modifier le programme précédent pour que `W` et `H` ne soient plus des constantes mais des valeurs entrées au clavier. Ne pas oublier de désallouer.

1.4 Charger un fichier

6. *Image couleur :*
La fonction `loadColorImage(srcPath("ppd.jpg"),r,g,b,W,H);` charge le fichier `"ppd.jpg"` qui est dans le répertoire contenant les sources (`srcPath`), alloue elle-même les tableaux `r,g,b`, les remplit avec les pixels de l'image, et affecte aussi `W` et `H` en conséquence. Attention : ne pas oublier de désallouer les tableaux `r,g,b` avec delete [] après usage.
— Charger cette image et l'afficher. Ne pas oublier les désallocations.
7. *Image noir et blanc :*
La fonction `loadGreyImage(srcPath("ppd.jpg"),t,W,H)` fait la même chose mais convertit l'image en noir et blanc. Afficher l'image en noir et blanc...

1.5 Fonctions

8. *Découper le travail :*
On ne garde plus que la partie noir et blanc du programme. Faire des fonctions pour allouer, détruire, afficher et charger les images :

```
byte* AlloueImage(int W,int H);
void DetruitImage(byte *I);
void AfficheImage(byte* I,int W,int H);
byte* ChargeImage(char* name,int &W,int &H);
```

9. *Fichiers :*
Créer un `image.cpp` et un `image.h` en conséquence...

1.6 Structure

10. *Principe :*
Modifier le programme précédent pour utiliser une structure :

```
struct Image {
  byte* t;
  int w,h;
};
```

`AllocImage()` et `ChargeImage()` pourront retourner des `Image`.

11. *Indépendance :*
Pour ne plus avoir à savoir comment les pixels sont stockés, rajouter :

```
byte Get(Image I,int i,int j);
void Set(Image I,int i,int j,byte g);
```

12. *Traitements :*
Ajouter dans `main.cpp` différentes fonctions de modification des images

```
Image Negatif(Image I);
Image Flou(Image I);
Image Relief(Image I);
Image Contours(Image I,double seuil);
Image Deforme(Image I);
```

et les utiliser :

(a) `Negatif` : changer le noir en blanc et vise-versa par une transformation affine.

(b) `Flou` : chaque pixel devient la moyenne de lui-même et de ses 8 voisins. Attention aux pixels du bords qui n'ont pas tous leurs voisins (on pourra ne pas moyenner ceux-là et en profiter pour utiliser l'instruction `continue` !).

(c) `Relief` : la dérivée suivant une diagonale donne une impression d'ombres projetées par une lumière rasante.
— Approcher cette dérivée par différence finie : elle est proportionnelle à $I(i+1,j+1)-I(i-1,j-1)$.
— S'arranger pour en faire une image allant de 0 à 255.

(d) `Contours` : calculer par différences finies la dérivée horizontale $d_x = (I(i+1,j) - I(i-1,j))/2$ et la dérivée verticale d_y, puis la norme du gradient $|\nabla I| = \sqrt{d_x^2 + d_y^2}$ et afficher en blanc les points où cette norme est supérieure à un seuil.

(e) `Deforme` : Construire une nouvelle image sur le principe $J(i,j) = I(f(i,j))$ avec f bien choisie. On pourra utiliser un sinus pour aller de `0` à `W-1` et de 0 à `H-1` de façon non linéaire.

1.7 Suite et fin

13. S'il reste du temps, s'amuser :
— Rétrécir une image.
— Au lieu du négatif, on peut par exemple changer le contraste. Comment ?