My Little Photoshop

1 Consignes de rendu

A la fin de ce TP, vous devrez rendre une archive respectant l'architecture suivante :

```
rendu-tpcs8-login_x.zip
|-- rendu-tpcs8-login_x/
    |-- AUTHORS
    |-- MyLittlePhotoshop.sln
    |-- MyLittlePhotoshop
        |-- Tout sauf bin/ et obj/
```

Bien entendu, vous devez remplacer "`login_x`" par votre propre login.

N'oubliez pas de vérifier les points suivants avant de rendre :

- Le fichier `AUTHORS` doit être au format habituel (une *, une espace, votre login et un retour à la ligne).
- Pas de dossiers `bin` ou `obj` dans le projet.
- **Le code doit compiler !**

2 Introduction

2.1 Objectifs

Lors de ce TP nous allons aborder les notions suivantes :

- L'utilisation des `Bitmap` en C#
- Le traitement d'image
- Un peu de maths...

3 Cours

3.1 Le traitement d'images

> Le traitement d'images est une discipline de l'informatique et des mathématiques appliquées qui étudie les images numériques et leurs transformations, dans le but d'améliorer leur qualité ou d'en extraire de l'information.
>
> — Wikipedia

Le traitement d'image est un domaine actif depuis les années 1920 avec d'abord des problématiques de compression pour permettre la transmission d'images malgré les fortes limitation des télécommunications de l'époque, il s'est ensuite développé pour les besoins de la seconde guerre mondiale avec le radar mais c'est surtout à partir des années 1960 que les techniques modernes se sont développées avec la découverte des liens avec le traitement de signal et l'augmentation de la puissance des ordinateurs.

Aujourd'hui ce domaine est omniprésent que ce soit au travers de logiciels grand public ou de logiciels embarqués permettant par exemple à un robot de percevoir son environnement.

Le traitement s'effectue généralement par application d'opérateurs sur une image que l'on peut séparer en trois grandes familles :

Les opérateurs point à point
Ce sont les plus simples, ils consistent en l'application d'une fonction de changement de la couleur à chaque pixel de l'image sans se préoccuper de ses voisins.

Les opérateurs locaux
Ils permettent des traitements plus évolués en prenant en compte les pixels voisins du pixel en cours de traitement, ils sont nécessaires pour des traitements tels que les flous.

Les opérateurs morphologiques
Basés sur la morphologie mathématique [1], ils sont principalement utilisés pour extraire des éléments d'une image afin de les interpréter et extraire de l'information.

Par l'application et la combinaison de ces différents types d'opérateurs on peut récupérer de nombreuses informations d'une image pour des utiilisations variées telles que la reconnaissance de texte, la construction de cartes ou le suivi sur vidéo par exemple.

Le TP de cette semaine ne parlera presque pas d'extraction d'information mais plutôt des traitements d'amélioration ou de préparation pour d'autres étapes.

3.2 La classe `Bitmap`

Pour représenter une image et pouvoir modifier ses pixels en C# on peut utiliser la classe `Bitmap` qui fournit les méthodes `GetPixel(x, y)` et `SetPixel(x, y, color)`.

Pour découvrir toutes ses fonctionnalités vous pouvez visiter sa page MSDN [2] ou instancier la classe et laisser Visual Studio autocompléter la liste de ses attributs et méthodes après avoir mis un point après le nom de l'objet créé.

Pour pouvoir utiliser la classe `Bitmap` ajoutez cette ligne au début des fichiers qui en ont besoin :

```
using System.Drawing;
```

4 Exercices

4.1 Avant de commencer

Au cours de ce TP vous allez devoir rechercher une partie des informations nécessaires sur internet, pour vous guider voici quelques liens intéressants :

- `http://fr.wikipedia.org/wiki/Traitement_d%27images`
- `http://fr.wikipedia.org/wiki/Op%C3%A9ration_point_%C3%A0_point`
- `http://docs.gimp.org/fr/plug-in-convmatrix.html`
- `http://fr.wikipedia.org/wiki/Lissage_d%27images`
- `http://fr.wikipedia.org/wiki/D%C3%A9tection_de_contours`

Vous pouvez télécharger le code de base sur `http://perso.epita.fr/~acdc`.

Pour commencer créez une classe `Filters` où vous implémenterez les fonctions demandées.

1. `http://fr.wikipedia.org/wiki/Morphologie_math%C3%A9matique`
2. `http://msdn.microsoft.com/fr-fr/library/system.drawing.bitmap%28v=vs.110%29.aspx`

4.2 Exercice 0 : Point à point

Vous devez écrire la fonction `MapPixels` qui parcourt le `Bitmap` donné en paramètre et remplace chacun de ses pixels par l'application de la fonction `f` passée en second paramètre (comme on faisait en OCaml, vous en entendrez parler plus en détail plus tard).

Prototype :

```
public static Bitmap MapPixels(Bitmap img, Func<Color, Color> f)
```

4.3 Exercice 1 : Premiers filtres

Écrivez les fonctions dont les les prototypes sont détaillés ci-dessous et qui représentent les filtres point à point qui seront passés à la fonction `MapPixels`.

Greyscale
Passe l'image en niveaux de gris, pensez à chercher les coefficients à appliquer à chaque composante pour un bon résultat.

Pinkify
Rend l'image plus rose tout en restant reconnaissable, vous pouvez calculer une "nuance de rose" comme pour le filtre précédent et retourner la moyenne de cette valeur et de la couleur de base par exemple.

Binarize
Rend l'image binaire (blanc et noir uniquement) par une méthode de seuil (d'autres méthodes sont bien plus efficaces mais ne s'appliquent pas en point à point).

Invert
Transformez l'image en son négatif (le noir devient blanc et inversement par exemple).

Prototypes :

```
public static Color Greyscale(Color c)
public static Color Pinkify(Color c)
public static Color Binarize(Color c)
public static Color Invert(Color c)
```

4.4 Exercice 2 : Miroir, mon beau miroir

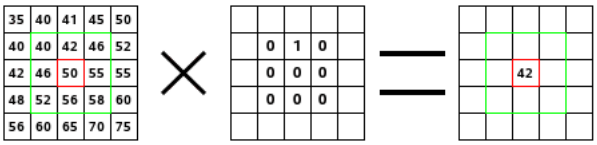
Écrivez les fonctions `MirrorH` et `MirrorV` qui effectuent une symétrie des pixels de l'image selon un plan respectivement vertical et horizontal.

Prototypes :

```
public static Bitmap MirrorH(Bitmap img)
public static Bitmap MirrorV(Bitmap img)
```

4.5 Exercice 3 : Convolution

Écrivez la fonction `Convolution` qui applique une matrice de convolution carrée (même largeur et hauteur) à une image.
Pour ce faire il faut parcourir l'image et remplacer chaque composante de la couleur du pixel en cours par la somme des produits de la composante équivalente d'un pixel voisin avec son coefficient équivalent dans la matrice [3].
Voici un schéma pour illustrer l'algorithme :



Vous pouvez considérer que les pixels en dehors de l'image sont noirs (tout à 0).
PROTIP : Vous pouvez faire une fonction `IsValid(int x, int y, Size s)` pour tester si des coordonnées sont bien dans l'image.
PROTIP 2 : Pour éviter les débordements liés aux erreurs de matrices ou d'approximation de flottants il est conseillé de vérifier que les composantes restent dans l'intervalle $[0; 255]$.
PROTIP 3 : Vous pouvez récupérer la largeur de la matrice avec `mat.getLength(0)` (ou 1 pour la hauteur mais ce n'est pas nécessaire ici).
PROTIP 4 : N'oubliez pas de travailler sur une copie de l'image pour ne pas interférer avec votre propre traitement.

Prototype :

```
public static Bitmap Convolution(Bitmap img, float[,] mat)
```

4.6 Exercice 4 : Entrez dans la Matrice

Définissez les matrices `AverageMat`, `GaussMat` et `EdgeMat` qui seront utilisées par la fonction `Convolution`, une taille de 3x3 est suffisante pour ce TP et ne devrait pas être trop lente :

AverageMat
Matrice de flou par moyenne.

GaussMat
Matrice de flou gaussien, pensez bien à normaliser (somme des cases de la matrice = 1) pour ne pas dépasser lors de la convolution.

EdgeMat
Matrice de détection de contour, renseignez vous sur les méthodes de Roberts, Prewitt, Sobel et Canny (par ordre de difficulté).

Pour définir une matrice en C# vous pouvez faire comme ceci :

```
public static float[,] MyLittleMatrix = new float[,]
                                                {
                                                    {0, 0, 0},
                                                    {0, 1, 0},
                                                    {0, 0, 0}
                                                };
```

3. Je clair Luc, ne pas ? — *La stratégie de l'échec*

4.7 Bonus

Voici quelques idées de bonus triés par difficulté croissante :

Facile
- Filtre d'éclaircissement (log)
- Filtre d'assombrissement (exp)

Moyen
- Autres matrices de convolution
- Rotations

Difficile
- Filtres de morphologie (érosion, dilatation, ouverture, fermeture)

Over 9000
- Détection de blobs (segmentation puis union-find)
- Utilisation de transformées de Fourrier (FFT) pour accélérer beaucoup
- Utilisation de `ClGlInterop`[4] pour accélérer encore plus

5 Résultats attendus

FIGURE 1 – Image de base

FIGURE 2 – Greyscale

FIGURE 3 – Pinkify

FIGURE 4 – Binarize

4. `http://www.cmsoft.com.br/index.php?option=com_content&view=category&layout=blog&id=137&Itemid=197`

FIGURE 5 – Invert

FIGURE 6 – MirrorV

FIGURE 7 – MirrorH

FIGURE 8 – AverageMat

FIGURE 9 – GaussMat

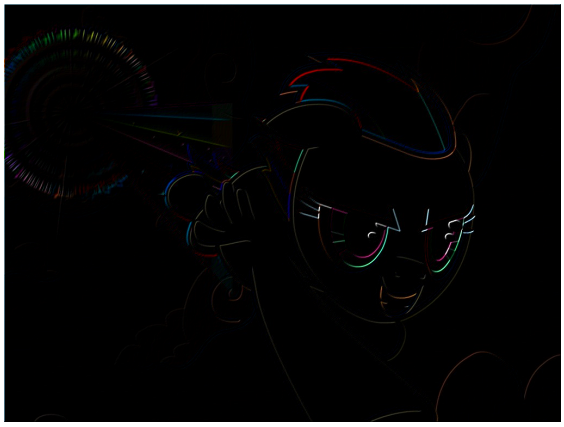


FIGURE 10 – EdgeMat

It's dangerous to code alone !