Rapport du projet 2 : ULBMP

LI Min-Tchun

12/05/2024

Contents

1 Introduction

Pour le deuxième projet du cours INFO-F-106, nous avons dû implémenter un compresseur d'image appelé ULBMP, basé sur le format BMP, développé par Microsoft dans les années 80.

Afin de pouvoir venir à bout de ce projet, il est demandé d'importer le module *Pyside6* afin de pouvoir créer un interface grahpique pour l'utilisateur. Il est donc important de lire la documentation de ce module afin de pouvoir l'utiliser correctement.

De plus, il est fondamental de maitriser la manipulation de fichier, de bits (notamment le shifting et le masking), ainsi que la compréhension de la représen-tation binaire et hexadécimal.

En effet, comme nous devons utiliser des pixels, il est important de comprendre que ces derniers sont composées de 3 valeurs : Rouge, Vert, Bleu (RVB ou RGB en anglais). Chacune de ces valeurs est codée sur 8 bits, ce qui signifie que chaque valeur peut prendre $2^8 = 256$ valeurs différentes.

Il était donc utile d'utiliser le programme **hexdump**, à exécuter dans le terminal afin de pouvoir visualiser les données hexadécimale.

Cependant, une question se pose : comment peut-on représenter une image en format numérique ou bien comment peut-on compresser une image ? Afin de résoudre ce problème, on va procéder comme suit, on va créer une classe *Decoder* et *Encoder*.

Le premier permettra, à partir d'un chemain d'accès du fichier, de lire chaque bytes et de retourner une *image* (on définiera plus tard plus précisemment ce que c'est).

Le deuxième permettra, à partir d'une *image*, d'écrire dans un fichier les bytes correspondants aux valeurs RGB de chaque pixel.

Il est important de notifier qu'il y a plusieurs version du compresseur d'image. Cela signifie que le header (ce qui se trouve à chaque début de fichier pour reconnaitre le format du fichier) doit être adapté à chaque version.

2 Méthodes

Comme cité dans l'introduction, nous allons devoir implémenter deux classes : *Decoder* et *Encoder*. Mais avant ça, il va falloir définir les classes *Pixel* et *Image*.

2.1 Pixel et Image

La classe *Pixel* possède les attributs suivants : r, g, b (ces derniers correspondent aux valeurs RGB d'un pixel). Le constructeur *Image* prend en paramètre la hauteur, la largeur de l'image, ainsi qu'une liste de taille $hauteur \times largeur$ comprenant des instances de la classe Pixel. Certaines fonctions spécifiques à la construction de classes doivent être créé pour faciliter la manipulation des pixels et des images. Par exemple, la fonction *getitem* de la classe *Image* permet de récupérer un pixel à partir de ses coordonnées (x, y).

```
def __getitem__(self, pos: tuple[int, int]):
    x = pos[0]
    y = pos[1]
    if not (x <= self.w and y <= self.h):
        raise IndexError("Not enough pixels.")
    return self.img[y * self.w + x]
```

Ou bien la fonction *setitem* qui permet de modifier un pixel à partir de ses coordonnées (x, y).

```
def __setitem__(self, pos: tuple[int, int], pix):
    x = pos[0]
    y = pos[1]
    if x < self.w or y < self.h:
        self.img[x + y * self.w] = pix
    else:
        raise IndexError("Invalid position.")
```

Par ailleurs, rappelons nous que chaque couleur du pixel est encodée sur 8 bits, ce qui signifie qu'il est impossible d'avoir des valeurs négatives ou supérieures à 255. De plus, une image ne peut pas posséder une quantité de pixel supérieure à la dimension de l'image. Par exemple, si une image a une hauteur de 5 et une largeur de 3, il est impossible d'accéder au pixel d'indice (6, 2) car il n'existe pas.

Voilà pourquoi les constructeurs des classes *Pixel* et *Image* sont définis comme suit :

Pour la classe Pixel :

```
def __init__(self, r, g, b):
    self.red = r
    self.green = g
    self.blue = b
    if self.red < 0 or self.green < 0 or self.blue < 0:
        raise ValueError("RGB values must be positive.")
    self.pixel = (self.red, self.green, self.blue)
```

Pour la classe Image :

```
def __init__(self, width: int, height: int, pixels):
    self.w = width
    self.h = height
    self.img = pixels
    if self.w * self.h != len(self.img):
      raise ValueError("Quantity of pixels doesn't match the image's dimension.")
    for i in pixels:
        if not isinstance(i, Pixel):
                raise ValueError("List's elements aren't Pixel instances.")
```

3 ULBMP 2.0

Ceci est le chapitre 2.

4 ULBMP 3.0

Ceci est le chapitre 3.

5 ULBMP 4.0

Ceci est le chapitre 4.

6 Conclusion

Ceci est une conclusion.