ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

ÉCOLE
D'INGÉNIEURS
PARIS-LA DÉFENSE

Rapport Projet Scientifique Informatique

Héloïse **Fourdain** – Ilies **Gourri**
2E ANNÉE CYCLE PRÉPARATOIRE

Gilles Nocture – Avril 2021

Réalisation d'un logiciel de traitement d'image

IMAGE ET QRCODE

Après avoir choisi tous les deux l'option Informatique pour le semestre 4, et constitué notre équipe, il nous a été fourni un document afin de nous expliquer ce que l'on devait accomplir durant ce semestre. Nous devions réaliser un programme de traitement d'image, avec une très grande autonomie qui nécessitait l'ensemble des connaissances acquises au cours de ces trois derniers semestres.

STRUCTURE

Le Programme est constitué de 3 classes principales : Pixel, MyImage et QRcode

La classe Pixel est composée de 4 attributs, chacun décrivant un pixel, puis un tableau de trois entiers, qui permet de créer un pixel en définissant le taux du pixel rouge, vert et bleu. Elle possède aussi deux méthodes, une permettant de transformer en nuance de gris un pixel, puis une permettant de transformer en noir ou en blanc un pixel (si sa nuance de gris est inférieure à 128 il sera noir sinon blanc).

La classe MyImage est la pièce centrale du projet. En effet, c'est dans celle-ci qu'on créé une image au format bitmap, qu'on la traite ou encore qu'on l'affiche.

Elle possède 16 attributs, qui sont indispensables pour décrire une image (nom, typeImage, tailleFichier,tailleImage,...).
Cette classe possède 2 constructeurs. L'un d'entre eux permet de lire une image déjà existante et de l'instancier au type MyImage. Ainsi il possède une entrée de type string qui sera le nom du fichier à lire. Pour finir l'autre permet de créer des images de type QRcode.

La méthode **From_Image_To_File()** permet d'écrire les données d'un objet de type MyImage au format BMP dans le dossier DEBUG. Elle prend donc en entrée le nom du fichier et est de type void. Le constructeur MyImage à partir d'une image existante fait tous les deux appels aux méthodes : **Convert_Int_To_Endian et Convert_Endian_To_Int** qui sont donc nécessaires pour convertir les tableaux d'octets en int et vice-versa en respectant la lecture du little-endian.

Les méthodes qui ont nécessité le plus de travail sont la **Rotation()** et **DecrypteImage().** La rotation nous a posé problème pour les cas les plus difficiles (hors 90°,180° et 270°). En effet, il reste quelques défauts sur l'image après son traitement pour ces cas. Bien que nous ayons réussi sans trop de problèmes la fonction **ImageDansImage(),** nous avons eu plus de soucis pour le décryptage. Le résultat de ses traitements n'est pas concluant.

Concernant de l'effet miroir, nous avons hésité entre doubler l'image ou un effet miroir classique, cependant il nous a été confirmé de faire un effet miroir classique.

La méthode **Histogramme()** elle permet d'obtenir l'histogramme d'une image BMP. Ainsi on peut avoir plus facilement les Histogrammes de la couleur Rouge, Vert, Bleu ainsi que celui de la moyenne (en noir). Ces trois valeurs sont des tableaux de int. La classe possède un constructeur spécifique de la classe Pixel qui permet d'instancier un Histogramme à partir d'une matrice de Pixel.
Les méthodes concernant le QRcode sont juste des méthodes pour remplir une image au format QR.

La classe QRcode a été nécessaire pour pouvoir créer une image de type QRcode. Comme vu dans le cahier des charges, le QRcode possède différents bits de données pour pouvoir être affiché. Le premier attribut est logiquement la chaîne de caractères alphanumérique en string appelé "chain". On trouve aussi des informations sur sa version et qrcodeChaine qui contiendra le code final du QR code.
Tout d'abord nous avons plusieurs méthodes qui permettent la conversion d'une chaîne de caractères en entier, puis en binaire. Ainsi nous pouvons réaliser la méthode qui renvoie le code final en binaire du QR code selon la chaine de caractère mis en paramètre. Le code est déjà corrigé par ReedSolomonAlgorithm (fournit par le professeur).
Ensuite nous réalisons des méthodes de remplissage pour avoir les motifs carrés en haut à gauche, en bas à gauche et en haut à droite. Puis nous avons une

fonction pour placer le module sombre, les motifs de synchronisation, les séparateurs et les bits de données. Pour pouvoir placer correctement les octets de la bonne manière, nous avons fait une méthode de remplissage pour la montée et la descente utilisée différemment selon la version choisie.
On applique enfin le masquage 0 (1110111 11000100).

Pour finir on peut noter que l'on peut rentrer des textes en minuscules qui seront automatiquement retranscrits en majuscule.

La classe Complexe permet quant à elle de réaliser des opérations sur les complexes lors de la création des fractales de MandelBrot.

Explication des méthodes en détail

NuanceGris()

Cette fonction, prend l'image de départ, et prend pour chaque pixel, la moyenne entre les trois valeurs, rouge, vert et bleu. Elle change la valeur du Pixel de départ pour la remplacer par cette moyenne. Une fois cette opération effectuée sur chaque Pixel, l'image qui en sort est en nuances de gris.

Rotation()

Cette fonction permet grâce à des opérations mathématiques de retourner une image peu importe l'angle. Tout d'abord on convertit l'angle en radian puis nous déterminons les nouvelles largeurs et hauteurs grâce aux opérations. Enfin grâce à un index défini selon les cas, on effectue le remplissage de l'image et du fond blanc.

Fractale()

La fonction Fractale permet se dessiner la fractale de Mandelbrot. On commence par définir la taille de l'image pour connaître le nombre de points du plan complexe à étudier. On définit le nombre d'itérations qui décide si la suite ($z = z^*z + c$) créée par les nombres z et c converge en ce point. On utilise les calculs sur partie réelle et partie imaginaire pour ne pas avoir besoin de classe nombre imaginaire. Si la suite dépasse 2 on déduit qu'elle diverge. Au bout d'un nombre d'itérations choisi on considère que la suite converge.

Histogramme()

Cette fonction permet de prendre chaque pixel un à un, récolte chaque valeur de ce pixel (la valeur rouge, la valeur verte et la valeur bleue). On somme ensuite toutes ses valeurs pour avoir la somme pour chaque couleur. On les divise par 255 pour avoir une échelle que nous pouvons comparer et nous regroupons ces valeurs dans un tableau. Ensuite nous allons créer l'histogramme dans une boucle for qui parcoure la taille de l'image. Nous y ajouterons des boucles if qui permettront de créer la courbe noire.

ImageDansImage(Pixel[,] image2)

La fonction ImageDansImage permet de cacher une image (image2) dans une autre image avec laquelle on travaille actuellement. Elle commence par réduire le nombre de couleurs en réduisant de 8 bits à 4 bits chaque pixel pour notre image initiale : une division par 16 appliquée à la deuxième image. On insère alors chaque pixel de la deuxième image comme résidu, sur les pixels de la première image. (Note : la taille finale est fixée sur la première image, si la deuxième image est plus grande alors une partie de la data est perdue.)

DecrypteImage ()

La méthode DecrypteImage () permet de réaliser l'étape inverse, par des divisions euclidiennes on retrouve pour chaque pixel les résidus potentiels d'une image cachée. En les multipliant par 16 on est censé retrouver une image proche de celle qui a été cachée précédemment, parfois décalée ou tronquée. La méthode retourne une matrice de pixels.

ImageCryptage(Pixel[,] imageF, string fichier)

Cette fonction prend au départ la matrice de Pixel que toutes les autres fonctions précédentes ont créé, et un fichier qui correspond à l'image de départ. Elle va créer une nouvelle image à partir de ça. Pour cela elle crée un fichier de taille 54 (pour le Header) + la taille de la matrice de Pixel qui correspond à hauteurF*largeurF*3.
Ensuite, elle remplit le header à partir de l'image de départ, puis modifie les différentes valeurs en lien avec la taille. Puis, il ne reste plus qu'à remplir l'image en elle-même. Une fois fini, le programme ouvre le ficher.

Innovation

Négatif()

Cette fonction, prend l'image de départ, et prend pour chaque pixel, sa valeur, et elle soustrait cette même valeur à 255. Elle change la valeur du Pixel de départ pour la remplacer par cette nouvelle valeur calculée. Une fois cette opération faite sur chaque Pixel, l'image qui en sort est en négatif, donc chaque couleur est inversée dans l'image. Cette fonction fait partie des fonctions supplémentaires qui nous est demandé au TD5.

Sepia()

La fonction permet de transformer un pixel en nuance de Sépia. Sachant que : rouge = 162, vert = 128 et bleu = 101 correspondent aux valeurs Sépia traditionnelles, nous effectuons tout simplement une moyenne entre la nuance de gris et ces valeurs de base. Ce qui nous donne une image avec une nuance sépia.

Compression

Lors de cette recherche bibliographique sur la compression nous nous sommes penchés sur l'Algorithme de Huffman. Il permet de compresser des données en exploitant leur redondance. Les caractères les plus fréquents sont remplacés par des codes binaires courts et les moins fréquents par des codes longs. On parle ici de codage à longueur variable préfixé car aucun des codes binaires n'est préfixe d'un autre.
Au final, la suite finale de mots sera en moyenne plus petite qu'avec un codage normal.

Conclusion

Avec plus de temps, nous aurions pu améliorer quelques aspects comme l'affichage du Qrcode. En effet, agrandir l'image qui renvoie le QRcode aurait été pertinent.
En somme, nous sommes tous les deux très fiers de ce que l'on a accompli durant ce module. Nous avons réussi à faire tout ce qui a été demandé, avec certains éléments qui nous paraissaient complexes dans un premier temps, et qui se sont révélés comme un succès. Nous ressortons de ce projet avec un sentiment d'accomplissement et de nouvelles compétences en programmation et en mathématiques.