ABDUL WAHAB Mohammed Zishan & Adrien BONNEFON

Rapport de PSI :

Explications de Structure de données :

La structure du logiciel est assez simple :

Tout d'abord, WPF contient deux parties : la partie de mise en page (Xaml), et la partie de code (**MainWindow.xaml.cs**). MainWindow.xaml.cs agit comme le Main de C# dans WPF.

Dans Mainwindow, nous initialiserons tout le traitement d'image, donc il contient par exemple, les **fonctions de Noir et Blanc**, **Fractale**, **Détection de bord** (contours) etc…

Ces fonctions devront importer une image bitmap, et pour traiter ces images, nous utilisons une classe nommée **MyImage.cs**.

MyImage prend en paramètre l'emplacement de l'image sous forme de chaîne. Ceci est fait pour éviter d'utiliser une classe Bitmap comme demandé. Cette classe MyImage a les objectifs suivants : obtenir **la taille, la longueur et la hauteur** de l'image; obtenir **une matrice de pixels d'image sans l'Header**; Réécrivez l'image résultante sur une autre image (**sauvegarde de l'image**).

Mais dans le but de simplifier la matrice d'image RGB en une matrice plus efficace et plus simple à utiliser, nous utilisons une classe **Pixel.cs.**

Pixel prend comme argument, le rouge, le vert et le bleu de l'image. Cela nous permet de raccourcir la matrice Hauteur x (longueur * 3) en matrice Hauteur x Longueur. Le pixel serait principalement utilisé comme matrice (comme Pixel [,])

Pour le besoin d'un code plus organisé, nous avons créé une autre classe pour QR Code appelée **QRcode.cs**. (Cette classe fait partie d'une page WPF puisque le code QR est ouvert dans une autre page de WPF). La classe QRCode n'a besoin d'aucun paramètre car il s'agit d'une classe de page WPF.

Pour **Fractal**, nous avons également créé une classe **Complexe.cs**. Cette classe fait simplement les calculs nécessaires qui seront utilisés pour créer une fractale de Mandelbrot. Cette classe n'est pas nécessaire à 100%, mais nous permet d'améliorer encore notre code beaucoup plus rapidement et de créer plus de variantes de fractal dans le futur si nécessaire.

Au total, nous avons créé **5 classes**.

Explication de l'innovation

- C # a un certain problème lors de l'affichage d'une image dans WPF. Une image déjà ouverte dans l'affichage (original) ne peut pas être modifiée directement. Il nous a montré l'erreur suivante: "Le fichier image est déjà ouvert", ce qui signifie que nous ne pouvions n'effectuer aucun traitement d'image. Nous avons dû trouver une solution, nous avons donc fait ceci: d'abord l'image est choisie (de n'importe où dans le PC) par l'utilisateur, puis nous avons copié ce fichier image original dans notre dossier bin / Debug en tant que ressource. Ensuite, nous avons affiché l'image d'origine dans l'affichage et nous avons utilisé cette image copiée pour le traitement de l'image. Et surtout, nous n'avions toujours qu'un seul fichier image à copier, afin de ne pas créer d'images en double.

- Une innovation que nous avons faite a été de moderniser la mise en page WPF en supprimant des éléments Windows. Lorsque nous avons commencé à faire WPF, nous avons réalisé que WPF avait des éléments de Windows, et cela avait un aspect un peu ancien. Donc, pour résoudre ce problème, nous avons créé de nouvelles fonctions et un «Glisser» pour remplacer les fonctionnalités d'origine de Windows.
- Une autre innovation était l'animation dans WPF. L'animation dans WPF était en soi facile, mais elle manquait de certaines fonctionnalités. Nous avons donc dû trouver une solution, et après un certain temps, nous avons créé une animation fluide. Cela était principalement dû à une fonctionnalité appelée AWAIT et deux fonctions distinctes étaient nécessaires pour créer une animation fluide.

Problèmes rencontrés

Nous avons certainement traversé des problèmes et des difficultés. Le principal problème était qu'au début, nous n'utilisions pas de classe PIXEL. Cela a rendu le traitement d'image un peu plus long à chaque fois. Mais finalement, nous avons créé une classe Pixel, ce qui a rendu la matrice d'entrée beaucoup plus facile à manipuler.

Un autre problème était le cahier des charges. On comprend qu'un projet peut évoluer souvent et que les détails du cahier des charges peuvent évoluer. Mais nous avons trouvé que l'énoncé avait parfois des informations erronées ou était trompeuse (en particulier dans la partie code QR). Heureusement, cela a été modifié dans les dernières versions du cahier des charges.

Auto critique

Même si nous avons été très actifs dans ce projet, nous avons été un peu lents, car nous étions trop fixés sur les détails. Nous avons toujours voulu terminer une fonction de traitement d'image, et parfois cela nous a fait prendre du retard. Mais heureusement, nous avons pu rattraper le retard et terminer la plupart des fonctionnalités requises bien avant la date limite.

La compression d’images :

La compression d’image a pour but de réduire le poids des données d’une image avec ou sans altération de la qualité, accélérant de facto la rapidité de leur transmission et facilitant leur stockage. Dans notre monde actuel où l’information et son stockage sont des problématiques centrales tant en politique, que socialement, économiquement et au regard de l’environnement, l’optimisation est plus que jamais un aspect fondamental du traitement des données.

Le choix de la compression dépend de l’utilisation finale de l’image (priorité à la qualité pour un tableau ou une œuvre d’art, priorité au stockage et à l’optimisation pour des transmissions dans l’espace par exemple)

La compression d’image est permise par le fait qu’il existe des corrélations au sein des pixels constituant l’image et il existe de nombreuses méthodes de compressions qui les utilisent et qui se divisent en 2 genres : avec ou sans perte

Parmi les méthodes sans perte on compte :

Ce type de compression utilise la redondance de l’information (le fait que plusieurs pixels soient de la même couleur) et l’entropie des données, un principe thermodynamique appliqué à la théorie de l’information par

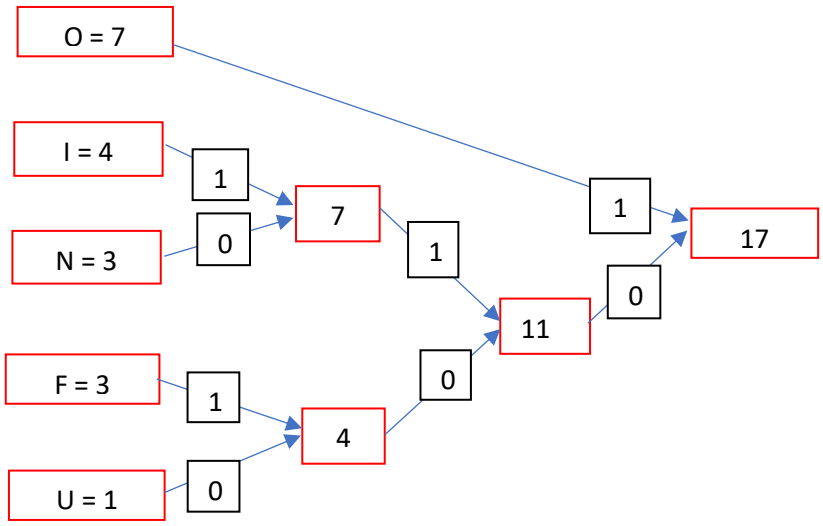
Claude Shannon en 1948. L'entropie augmente avec le nombre d'évènements équiprobables et avec l'équilibre des probabilités, elle permet de mesurer la quantité d'information (un signal peu informatif est redondant, un signal très informatif est diversifié et peu prédictible). La compression sans perte est généralement faite par 2 procédés : le modèle statistique ou le codage par dictionnaire.

1) Le codage entropique (le cas du codage Huffman) :

Cette méthode de compression a pour but de transformer la représentation d'une source de données pour sa transmission. Dans le cas ici présent d'une image la matrice de pixels de l'image suivant le nombre d'occurrence d'un pixel particulier, pour illustrer le principe de cette compression il est plus simple de voir comment le codage Huffman (qu'on appelle aussi arborescence de Huffman) fonctionne pour encoder un texte.

Mettons que l'on veuille coder le message suivant : « info info infoooo oui! »

On compte le nombre d'occurrence de chacune des lettres :



Ce qui nous donne pour chacune des lettres le code suivant :

O = 1

I = 011

N = 010

F = 001

U = 000

Le O est codé sur 1 bit le reste des lettres sur 3 le message est donc codé sur :

7*1 + 3*(4 + 3 + 3 + 1) = 40 bits

Une économie conséquente par rapport à l'espace qu'il aurait fallut pour stocké ce même message en prenant leur index dans l'alphabet par exemple

La même logique s'applique au pixel on les classifie par occurrence et on utilise un système similaire pour les codé en binaire, de cette façon une image dont chaque pixel est codé par 24 bits (une image décrite par une matrice RVB pouvant donc prendre 2^{24} couleurs distinctes mais la probabilité de tombé sur une image composée uniquement de pixels tous différents est extrêmement faible. On réalise donc la même opération que pour le message en répartissant les valeurs et leurs probabilités en arbre et en leur associant un chiffre en binaire suivant leur disposition dans l'arbre ainsi on réduit ainsi la taille de l'image sans aucune perte de données puisque chacune des couleurs de l'image possède un code propre.

Cette forme de compression est notamment utilisée pour le format d'image JPEG

2) Le codage de type dictionnaire (le cas de l'algorithme LWZ) :

Le codage par dictionnaire consiste à prendre les occurrences les plus fortes et à les remplacer par un code plus court stocké dans une structure de données qu'on appelle un dictionnaire

Exemple : « Si ton tonton tond ton tonton, ton tonton tondu sera »

Devient « Si @ @@ @d @ @@, @ @@ @du sera »

Ici l'algorithme trouve que la suite de lettre ''ton'' revient fréquemment il est donc plus optimisé de remplacer cette séquence par un seul et même caractère et d'enregistrer la signification dans le caractère dans le dictionnaire

On peut transposer cette logique aux pixels d'une image afin de compresser l'image sans perte d'information

Un algorithme de codage universel conçu par Lempel et Ziv en 1977 amélioré par Welsh en 1984 d'où le nom d'algorithme LWZ utilise ce principe en encodant automatiquement une suite de symbole (dans notre cas des pixels) issu d'une source (une image), il est aujourd'hui utilisé pour le format GIF

Il existe différentes techniques de compression avec pertes :

1) Le sous-échantillonnage :

L'œil humain est plus sensible aux variations de luminance (souvent rapporté à la luminosité, elle est définie comme l'intensité lumineuse par unité de surface perpendiculairement à la direction d'origine de l'émission) qu'à la chrominance (la couleur), on peut ainsi supprimer une quantité d'information relative à la couleur de l'image relativement importante sans en affecter fortement la perception du rendu final

2) Le codage par transformation :

C'est la méthode la plus populaire, la transformée en cosinus discrète (notée TCD les algorithmes de calcul transformées TCD se basent sur la possibilité de décomposer la matrice de définition sous forme d'un produit de matrices dont le calcul est plus simple) et la transformation par ondelettes (décomposition d'une image en de nombreuses sous-bandes, c'est-à-dire des images de résolutions inférieures, généralement suivie d'une quantification puis d'un codage entropique)

3) La quantification

C'est une étape très importante de la compression des données, c'est celle ou l'on peut agir pour atteindre un débit déterminé, en compromettant sur la qualité de l'image résultante

Conclusion :

La compression d'image est un processus complexe et les moyens d'y parvenir sont divers et nécessite réflexion et recherche pour déterminer quels procédés sont plus adaptées telle ou telle utilisation finale suivant la rapidité d'exécution, le coefficient de compression souhaité et le degré de netteté attendu.