Rapport sur le $2^{ème}$ projet informatique

Kazberuk Denis

8 mai 2024

Contents

1 Introduction

Dans le monde des bases de données, nous cherchons à pouvoir réduire la taille des fichiers, images, vidéos, etc. Pour économiser de la place, de l'argent, de la bande passante, etc. Dans ce rapport, nous nous occuperons des images. Le principe de compression de ce type de fichier sera, logiquement, de réduire le nombre d'octets à écrire. Et cela, peu importe la manière d'y arriver. La suppression totale de pixels n'est pas à exclure. Pleins d'extensions avec différentes compressions existent comme le *JPG*, *PNG*, *BMP*, etc. Dans notre cas, le format *ULBMP* est un format, inventé, qui se base sur le type *BMP*. Étant donné que ce format n'est utilisé que pour le cadre d'un projet informatique en bachelier, sa popularité n'est pas assez grande que pour l'utiliser comme extension par défaut – et c'est sûrement pour le mieux. En effet, les critères qui permettent de réduire la taille de l'image sont utiles dans des cas limités. Après recherche d'une image qui afficherait une compression exceptionnelle, nous observons que la différence reste élevée comparé aux restes. Dans sa totalité, le format ULBMP contient cinq versions. Nous ne parlerons, que de quatre d'entres-elle étant donné que la dernière ne fait pas partie du projet principal. Nous le détaillerons dans la suite de ce document. Bien évidemment, chaque variante est conçue avec une réflexion partic- ulière ; les différentes versions ne possèdent pas obligatoirement les améliorations précédentes. Pour rappel, un pixel et une combinaison de trois couleurs: le rouge, le vert et le bleu. Chaque couleur dispose d'une plage d'utilisation allant de 0_{10} à 255_{10}. En base 2, cela correspond à une suite de huit bits, soit un octet (ou byte en anglais).

2 Analyse de l'extension ULBMP

À présent, nous allons passer en revue les différentes versions que nous avons à disposition. Mais avant de rentrer dans le vif du sujet, un détail doit être avancé. Toutes les variantes possèdent un en-tête. Certain plus long que d'autre. Ce header doit contenir des informations qui apporteront une aide durant la sauvegarde ou lecture de l'image. Ces informations sont approximativement[1] cinq: Le nom de l'extension, sa version, sa longueur, la largeur et longueur de l'image dans l'ordre.

2.0.1 Première version

Initialement, cette extension n'existe pas. Il a donc fallu créer un programme permettant de lire et sauvegarder ce type. Par conséquent, la première étape consiste en la création du type. Un peu comme les fondations d'une maison. Et donc, l'optimisation sur la taille de l'image est inexistante. De fait, tous les pixels de l'image sont écrits. Cela veut dire qu'une image de 1000 pixels par 750 pixels aura $1000.750.3 = 2,25.10^6$ bits, soit 2,25 Mégabits.

2.0.2 Deuxième version

Maintenant que nous avons un programme qui lite une image en format *ULBMP*. Nous pouvons commencer à cogiter sur les possibilités à implémenter. Une nouvelles fonctionnalité est ajoutée dans cette phase. L'idée est de ne pas copier les pixels semblables qui se suivent. Évidemment, il nous faut de l'espace pour ajouter une information. Nous allons alors écrire un octet supplémentaire. Ce byte aura comme information le nombre d'occurrences de pixel derrière celui-ci. Étant donné que seuls huit bits sont ajoutés, l'occurrence maximale avant l'écriture d'un nouveau pixel est de 255_{10} . Cette technique est appelée *RLE*. Il est facile de comprendre que cet algorithme est efficace dans certaines conditions. L'image doit être plutôt homogène, en ligne de préférence.

2.0.3 Troisième version

Pour cette troisième phase, nous allons rentrer un peu plus en profondeur, dans le fonctionnement des bits. Nous le savons, dans un octet il y a huit bits. La stratégie de cette version va consister à découper l'octet.

[1] Dépendant de la version dans laquelle nous sommes.

Afin que cela puisse nous servir à quelque chose. De plus, nous allons créer une palette de couleur. Celle-ci va se trouver en entier dans le header. Ici, tous les pixels dont nous aurons besoin se trouvent dans l'en-tête à une certaine position. Par conséquent, nous n'écrirons plus les couleurs dans le corps du fichier (vu qu'ils sont déjà dans l'en-tête), mais seulement la position à laquelle le pixel se trouve. La technique du *RLE*peut s'ajouter. Non seulement l'efficacité de cette technique se trouve dans la possibilité d'écrire plusieurs pixels avec un octet. Mais aussi d'écrire les couleurs de l'image une unique fois. Prenons un exemple, si nous avons une image avec douze couleurs différentes. Alors elle a une profondeur de 4, car: $2^4 = 16$ et $12 < 16$. Par conséquent, un octet sera coupé en deux par le milieu, afin d'avoir deux paquets de quatre bits. Et ces bits donneront la position du pixel dans le header qu'il faudra écrire.

2.0.4 Quatrième version

La quatrième et dernière version sur laquelle nous nous concentrerons. Comparée aux phases déjà créées, celle-ci veut se lier aux cas pratiques. La réflexion qui a été faite pour cette mise à jour est de travailler sur la différence entre les couleurs. Dans une image que nous prendrions dans la vie de tous les jours, il y a souvent beaucoup de dégradé. Une coupure nette est souvent moins agréable à regarder. C'est pour cela que cette version est apparue. À présent, au lieu d'avoir un byte qui est un pixel ou qui y fait référence, l'octet va être la différence avec la couleur adjacente. E revanche, il arrive que les pixels aient une trop grande différence, dans quel cas il faudra complètement créer un nouveau pixel. On peut rapidement s'imaginer que l'efficacité de cette version est grande quand il y a de long dégradé.

3 Méthodes

Pour implémenter ces différentes versions du format *ULBMP*, nous sommes passés par plusieurs classes séparées en fichiers.

3.1 Les basiques

Ce projet parle d'image, c'est évident que nous aurons besoin d'un objet qui représente une image, mais aussi un pixel. À noter que les deux codes sont disposés dans deux fichiers différents. La classe Image possède des attributs cruciaux pour le bon déroulement du programme. Comme par exemple la largeur, la longueur, le nombre de pixels total et tous les pixels de l'image. De plus, des méthodes sont ajoutées à la classe: des getters, un setter, une méthode spéciale de comparaison, etc. Autres que les méthodes classiques, une méthode et fonction de vérification sont comprises. La méthode analyse la correctitude de l'indice pour le pixel demandé. La fonction vérifie la concordance entre le nombre de pixels reçu et la taille de l'image. En ce qui concerne la classe Pixel, elle aussi possède les attributs et méthodes classiques. Soit, la valeur du R, G et B ainsi que les getters, setters, comparaisons. Comme pour la classe Image, une fonction de vérification est présente qui n'accepte que des valeurs entre 0_{10} à 255_{10}.

3.2 Le vif du sujet

Cette section du document portera sur les classes Encode et Decode. Le premier sert à sauvegarder une image, l'autre à la lire. Ils sont groupés dans un même fichier, celui-ci est long de 742 lignes. C'est pourquoi, toutes les fonctions/méthodes ne seront pas décrites. Seulement les plus importantes vont être résumées ; vous êtes invités à lire le code[2] par vous même si vous le désirez.

3.2.1 La lecture

Comme expliquer plus haut, la lecture d'une image se fait avec la classe Decoder. La méthode de travail consiste en la lecture du fichier complet *ULBMP*, et de déposer ces valeurs dans une liste. Ensuite, chaque

[2]Lien vers le Github

version possèdera son propre parcours au sein du code, pour arriver à sa fin. Bien évidemment, nous avons essayé de créer des fonctions le plus générique, afin de grouper un maximum de fonction entre-elle. La classe Decoder n'a d'attribut important qui avantage le déplacement des fonctions dans la classe.

3.2.2 La sauvegarde

La classe Encode possède la même logique générale que celle pour lire. En revanche, maintenant, nous avons des attributs et méthodes classiques. C'est pourquoi, quand l'occasion se présente, les fonctions, qui servent à l'écriture dans un fichier, sont mises dans la classe. L'avantage, bien connu, d'utiliser les attributs directement dans la fonction, sans le passage des paramètres effectifs → formels.

3.2.3 Analyse des performances

4 Résultats

1ère ligne	test [Dirac, 1981]
2ème ligne	test 2

5 Discussion

6 Conclusion

References

[Dirac, 1981] Dirac, P. A. M. (1981). *The Principles of Quantum Mechanics*. International series of monographs on physics. Clarendon Press.