Info-F-106 : Projet d'Informatique
Rapport

Analyse et comparaison des différentes versions de ULBMP

Saou Ayman

Dimanche 12 mai 2024

Matricule : 000593526

Section : B1-INFO

Contents

1 Introduction **2**

2 Méthodes **2**
2.1 Critères de comparaison 2
2.2 Implémentation des différents critères 2
2.3 Images comparatives 3

3 Résultats **4**
3.1 Remarques 4
3.2 Taux de compression 4
3.3 Vitesse d'encodage 5
3.4 Vitesse de décodage 5

4 Discussion **6**
4.1 Taux de compression 6
4.2 Vitesse d'encodage et décodage 6

5 Conclusion **6**
5.1 Résumé 6
5.2 Meilleure version ? 7

6 Utilisation de GPT lors de la rédaction de ce rapport **7**
6.1 Pourquoi 7
6.2 Comment 7
6.3 Réflexions sur l'usage de bots GPT 7

1 Introduction

Lors de ce rapport, nous allons évoquer, analyser et comparer les différentes versions du format de compression d'images ULBMP, afin de faciliter la compréhension de ce rapport, il est préférable (mais pas obligatoire) d'être au préalable familier avec le format ULBMP et les concepts généraux de compression d'image.

Le format ULBMP est un dérivé des premières versions du format *BMP* (*BitMap Picture*) développé par Microsoft dans les années 80, c'est un format dit *"lossless"* (c'est-à-dire sans perte), le but de ce genre de format est de compresser l'image au maximum sans toutefois perdre d'information, en opposition aux formats *"lossy"* qui se permettent la perte d'information pour réduire au maximum la taille de l'image.

Tout le défi des formats comme ULBMP réside donc dans le fait de trouver des manières ingénieuses et plus efficaces de représenter l'information comme l'algorithme de compression *RLE* pour *Run-length encoding*, le format *QOI* pour *Quite Ok Image* et bien d'autres, c'est ce que nous allons aborder avec les différentes versions d'ULBMP implémentées.

2 Méthodes

2.1 Critères de comparaison

Afin de comparer les différentes versions du format ULBMP entre elles, nous utiliserons 3 critères différents

1. **Le taux de compression :** Mesure de la capacité du format à réduire le poids de l'image.

$$\text{Taux de compression} = \left(\frac{\text{Taille de l'image avant la compression}}{\text{Taille de l'image apres la compression}}\right)$$

   Un taux de compression de 2 désignant donc une image 2 fois moins lourde qu'avant compression

2. **La vitesse d'encodage :** Vitesse à laquelle l'image (l'objet de la classe Image) est encodée dans un fichier (en s).

3. **La vitesse de décodage :** Vitesse à laquelle nous récupérons les données d'une image encodée dans une certaine version du format ULBMP pour en construire l'image (en ms).

2.2 Implémentation des différents critères

Pour les comparaisons d'images non compressées, nous avons fait le choix d'utiliser le format ULBMP1 comme format standard désignant l'image avant compression, car c'est celui qui présente le moins de "stratagèmes" et qui encode l'image de la manière la plus naturelle possible.

```
# TAUX DE COMPRESSION #
size_V1 = os.path.getsize(path_V1) # Taille de l'image en format ULBMP1
image_version = Decoder.load_from(path_V1)
Encoder(image_version, version).save_to(path_version)
# Taille de l'image dans la version qu'on desire comparer
size_version = os.path.getsize(path_version)
taux_de_compression = (size_V1 / size_version)
```

Quant aux vitesses de décodage et d'encodage, nous avons utilisé deux algorithmes très simples qui se présentent sous la forme suivante :

```
# VITESSE D'ENCODAGE #
start = time.time()
Encoder(image).save_to(path, version)
end = time.time()
vitesse = (end - start)
```

```
# VITESSE DE DECODAGE #
start = time.time()
image = Decoder.load_from(path)
end = time.time()
vitesse = (end - start) * 1000 # Pour avoir le temps en ms
```

2.3 Images comparatives

Pour nos comparaisons, nous avons décidé d'utiliser quatre images, dont *checkers.ulbmp* fourni initialement, faute d'images avec uniquement du noir et du blanc trouvé sur internet, pour avoir une image composé d'uniquement deux couleurs uniques (cela s'avèrera pertinent lorsque que nous essaierons d'interpréter les résultats obtenus lors des comparaisons), le reste des images proviennent toutes du site `https://www.hlevkin.com/hlevkin/02imageprocC.htm`.

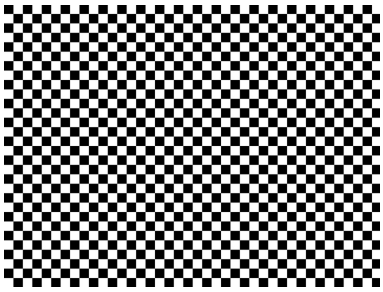

Figure 1: checkers.ulbmp

Figure 2: airplane.ulbmp

Figure 3: monkey.ulbmp

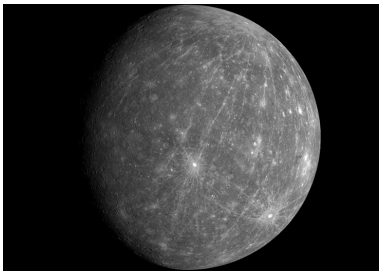

Figure 4: mercure.ulbmp

Concernant le choix des images, nous avons essayé d'avoir un panel d'images avec des caractéristiques différentes pour couvrir tous les avantages et inconvénients de chaque version du format ULBMP, *checkers* pour une image avec un pattern récurrent composé de deux couleurs uniques, *airplane* pour une image avec un grand nombre de pixels et de couleurs uniques, *monkey* pour une image avec un nombre de couleurs uniques $> 2^8$, pareillement pour *mercure*, mais avec énormément de pixels et des couleurs redondantes.

3 Résultats

3.1 Remarques

Pour des soucis de concision et de pertinence, nous utiliserons uniquement la profondeur 8 avec et sans RLE pour la version 3, nous n'allons pas évoquer les profondeurs 1, 2, 4 et 24 de la version 3 du format ULBMP, car la profondeur 24 correspond à une compression utilisant la version 1 ou 2 du format selon le RLE et que nous nous sommes également rendu compte lors des tests que La taille d'un fichier en version 3 du format ULBMP est proportionnelle à sa profondeur. Un fichier enregistré avec une profondeur $d = 1$ est deux fois plus petit qu'avec une profondeur $d = 2$ et quatre fois plus petit qu'avec une profondeur $d = 4$. Dans le cas où le choix est permis ($2^d \geq$ couleurs uniques pour d = profondeur) la compression la plus efficace est celle avec la plus petite profondeur. Le même principe s'applique pour les vitesses d'encodage, cependant les vitesses de décodage sont similaires pour les profondeurs 1, 2 et 4.

Quant aux tests portant sur la vitesse d'encodage et de décodage, les résultats utilisés seront des moyennes de plusieurs tests afin d'évaluer la tendance générale. Les résultats sont tout de même à prendre avec des pincettes, car dépendants de nombreux facteurs et de l'optimisation de mon implémentation. Qui plus est, notre implémentation de la version 4 est très *très* mal optimisée, ce qui fausse en partie les résultats, nous en tiendrons donc compte pour la conclusion en revoyant à la haute les performances de cette dernière.

3.2 Taux de compression

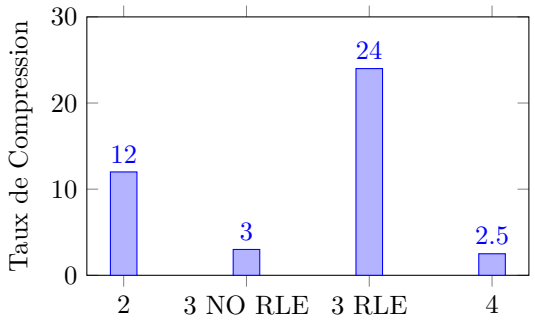

Figure 5: checkers

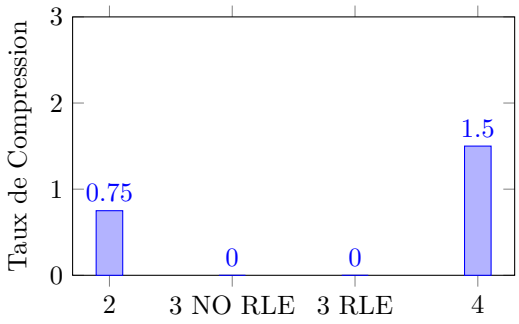

Figure 6: airplane

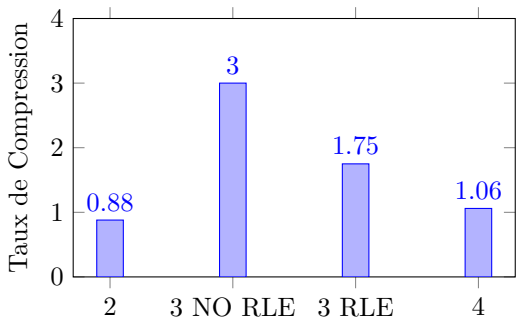

Figure 7: monkey

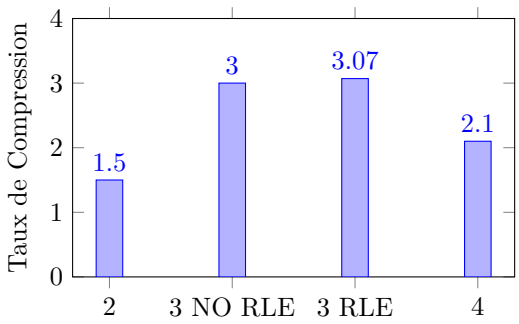

Figure 8: mercure

On observe que la version 3 est celle qui obtient le plus haut taux de compression en comparaison avec les autres versions (lorsqu'elle est possible), la version 2 quant à elle obtient des résultats plutôt médiocres et est dans certains cas moins efficaces que la version 1 pour les images *monkey* et *airplane* avec un taux de compression < 1, cependant, elle se démarque lors de la compression de *checkers* avec un taux de compression

de 12, finalement, la version 4 est la plus stable avec un taux de 1.8 en moyenne.

3.3 Vitesse d'encodage

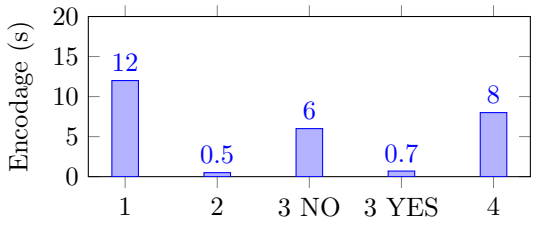

Figure 9: checkers

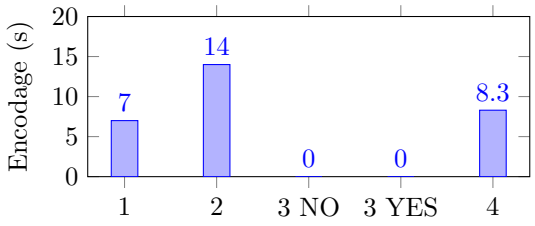

Figure 10: airplane

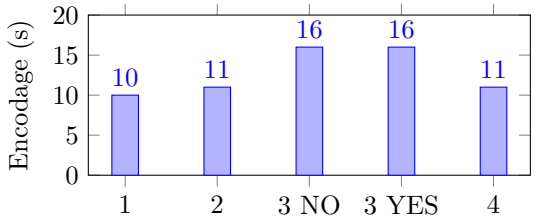

Figure 11: monkey

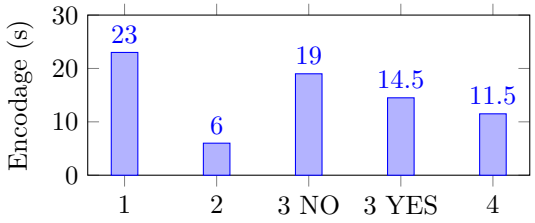

Figure 12: mercure

On observe encore une fois que la version 4 est la plus stable, la version 2 à d'excellents résultats pour checkers, mais médiocre pour le reste des images et les versions 1 et 3 sont très inégales.

3.4 Vitesse de décodage

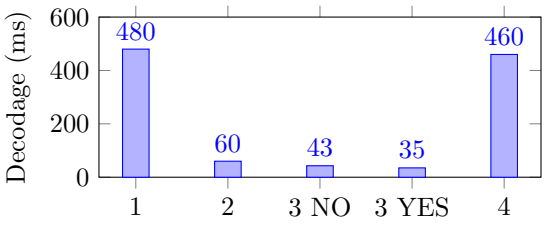

Figure 13: checkers

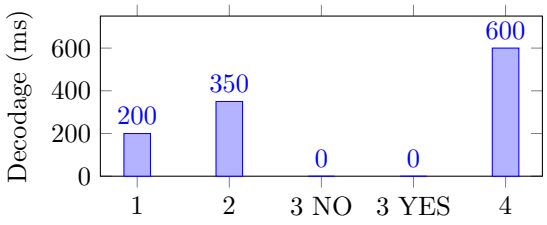

Figure 14: airplane

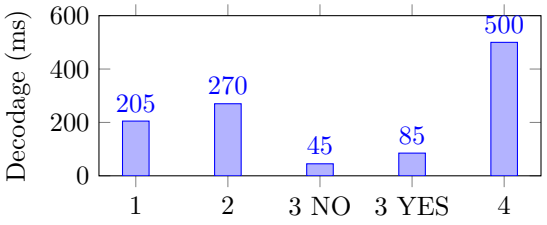

Figure 15: monkey

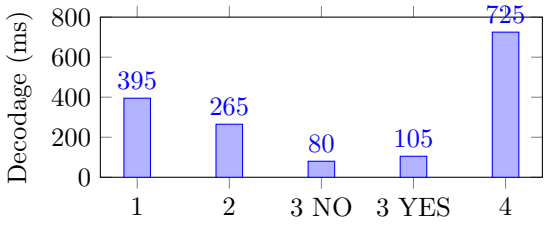

Figure 16: mercure

La version 4 est encore une fois stable, mais peu performante, la version 3 obtient les meilleurs résultats, les versions 1 et 2 inégales.

4 Discussion

Tout d'abord concernant le point 3.1 la proportionnalité entre la taille des images et leur profondeur en version 3 s'explique par le fait que la profondeur représente le nombre de bits utilisés pour représenter un pixel, il est évident que représenter 4 pixels sur un byte prendra deux fois moins de place que de n'en représenter 2.

4.1 Taux de compression

La version 3 obtient les meilleurs résultats, car elle utilise seulement un byte pour représenter un pixel, voir moins si le RLE est activé et que l'image contient plusieurs pixels identiques qui se suivent, ceci explique aussi pourquoi la version 2 pour checkers obtient un bien meilleur résultat que la version 3 sans RLE, car le RLE est extrêmement efficace pour les images comme checkers avec un pattern redondant, c'est pour la même raison que la version 2 (qui utilise le RLE) obtient de très mauvais résultats Excepté pour la compression de checkers. La version 4 quant à elle est plutôt régulière, efficace (taux > 2) pour les images avec peu de différences entre les pixels comme *checkers* et *mercure*, mais beaucoup moins pour *airplane* et *monkey* ou les pixels sont très différents les uns des autres, dû au fait que cette version utilise un encodage avec lequel chaque pixel utilise un nombre de bytes proportionnel à la différence entre lui-même et le pixel qui le précède.

4.2 Vitesse d'encodage et décodage

La version 1 qui encode les pixels un par un obtient les pires résultats en moyenne, la version 2 obtient un excellent résultat pour checkers mais très médiocre pour le reste des images (pour les mêmes raisons qu'évoqué précédemment), la version 3 obtient un très bon résultat pour checkers avec le RLE (pour les mêmes raisons que la version 2 encore une fois), cependant nous obtenons de mauvaises performances pour le reste des images et pour la version sans RLE, en effet pour les images avec peu voir aucun pixels identiques consécutifs, cette version doit ajouter un byte pour l'occurrence sans grand intérêt à chaque pixel, ce qui occupe plus d'espace que ce qui était censé en gagner, enfin la version 4, elle n'obtient pas les pires résultats, mais pas les meilleurs non plus, étant donné qu'elle fut conçue pour "s'adapter" au type d'images (image avec pattern de pixels récurrents et semblables comme *checkers* et *mercure* ou image ou les pixels très différents se suivent comme *airplane* et *monkey*), cela peut se voir dans l'implémentation : la version 4 utilise au grand maximum 4 bytes pour représenter un pixel dans le pire des cas et 1 byte dans le meilleur des cas, et 2 ou 3 pour les cas intermédiaires.

Les mêmes explications et analyses faites pour l'encodage s'applique pour le décodage, tout en prenant en compte le fait que notre implémentation de la version 4 est très mal optimisée (*cf.* le point 3.1).

5 Conclusion

5.1 Résumé

Pour conclure, la version 1 du format est globalement la pire et dans chaque cas d'utilisation, utiliser une autre version sera plus efficace, la version 2 est très efficace pour les images contenant de longues suites de pixels identiques, la version 3 est très dépendante de l'image et des paramètres utilisés (RLE, profondeur), globalement les profondeurs 1, 2, 4 et 8 sont efficaces, mais contraignantes (*cf.* 3.1), pour finir, la version 4, tout comme la version 1, aura toujours une version plus efficace selon l'image, mais reste cependant efficace quelle que soit l'image, c'est la plus régulière et stable.

5.2 Meilleure version ?

Dans le cadre d'un usage généralisé, la version 4 est la meilleure du format ULBMP, sinon pour les cas spécifiques, chacune se démarque des autres selon la situation.

6 Utilisation de GPT lors de la rédaction de ce rapport

6.1 Pourquoi

Ceci étant ma première utilisation du format LaTeX, la prise en main ne fut pas facile et j'ai eu recours à un bot GPT uniquement pour tout ce qui est relatif à l'écriture en LaTeX, toute la rédaction et réflexion est personnelle et écrite par moi-même.

6.2 Comment

Étant une personne qui aime et qui a plus facile à apprendre avec des exemples, j'ai fait usage de GPT comme d'un moteur de recherche, mais qui me donne immédiatement ce que je lui demande, exemple : un template d'une page de garde en LaTeXpour en comprendre la structure et pouvoir la modifier selon ma vision personnelle, j'ai réalisé ce processus pour l'inclusion de code (*cf.* 2.2), d'images et de graphiques. J'ai fait usage du bot GPT gratuit *ChatGPT 3.5* le 10/05/2024 et 11/05/2024.

6.3 Réflexions sur l'usage de bots GPT

Je trouve que ces nouveaux outils sont formidables lorsque l'on sait les utiliser à bon escient, mais qu'il est facile de tomber dans la fainéantise et de céder à la tentation de leur déléguer toutes nos taches jusqu'à la réflexion même, l'usage de ces bots doit rester à but éducatif, sans toutefois prendre tout comme acquis et en vérifiant toujours les informations avec de sources tangibles.