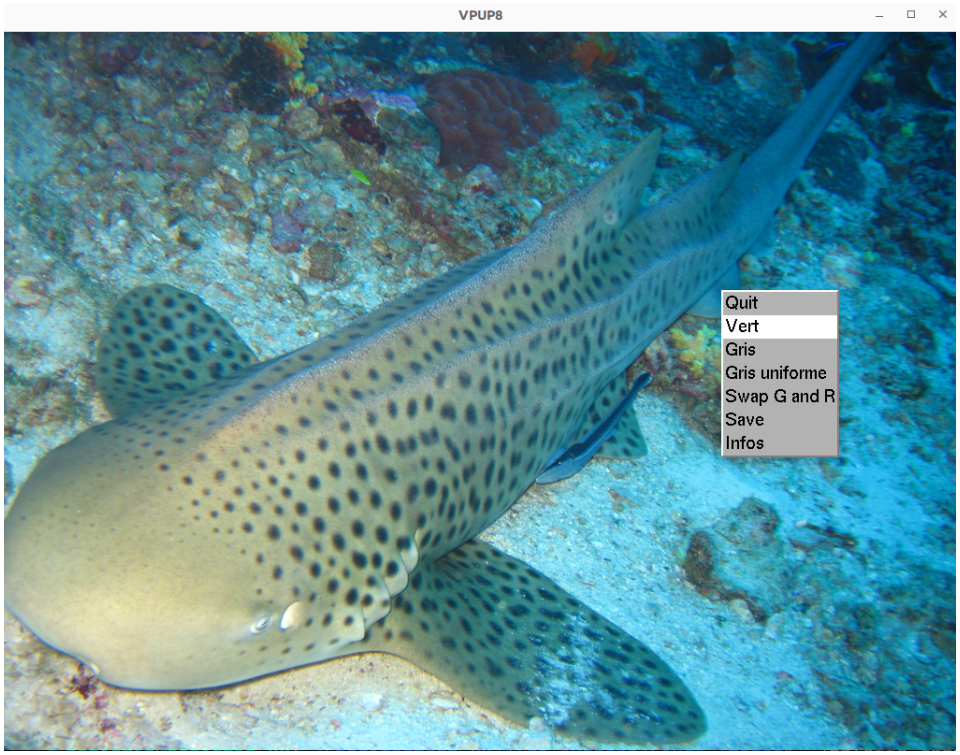
2023-2024 Université Paris 8
Algorithmique avancée Licence 3 Informatique

TP n°4 – Images : modification et compression.



Vous trouverez dans l'archive *Images.zip* sur Moodle plusieurs fichiers C dont :

- *ima.h*, le header, qui contient la structure d'image, les prototypes et les directives de préprocesseur permettant l'inclusion des bibliothèques graphiques notamment.
- *ppm.c* qui contient des fonctions de gestion d'images PPM,
- *modif.c*, qui contient des fonctions de modification d'image PPM.
- *main.c* qui contient des fonctions de gestion de la fenêtre graphique et du menu (voir l'image ci-dessus).
- un *makefile* pour compiler tout ça, qu'il faudra peut-être réadapter à votre configuration. L'exécution sera alors réalisée par

```
./palette Chemin_vers_image
```

0 Initialisation

Récupérer les fichiers de l'archive *Images.zip*, et vérifiez que vous pouvez lire et afficher des images. Sinon, il vous faudra peut-être modifier l'inclusion des librairies dans *ima.h* ou installer les librairies nécessaires si vous utilisez votre machine.

1 Modification d'images

En vous inspirant de la fonction *vert* qui modifie une image en mettant dans chaque pixel une teinte de vert à 255 et une teinte de rouge et de bleu à 0, écrire des fonctions :

1. *vert_moy* qui met dans chaque pixel une teinte de vert qui correspond à la moyenne des teintes de rouge, vert et bleu du pixel.

2. *vert_uniform* qui met dans chaque pixel une teinte de vert qui correspond à la moyenne pondérée des teintes de rouge (poids 2), vert (poids 4) et bleu (poids 1) du pixel.

3. *rouge_origin* qui met dans chaque pixel une teinte de rouge qui est la même que celle de l'image d'origine, et met toutes les teintes de vert et de bleu à 0.

4. *gris_uniform* qui met dans chaque pixel une teinte de gris qui est la moyenne pondérée des teintes de rouge (poids 2), vert (poids 4) et bleu (poids 1) du pixel. On rappelle que pour obtenir du gris, on mettra la même valeur dans R,G et B.

5. *swap_green_red*, qui échange dans chaque pixel les teintes de rouge et de vert, et qui met toutes les teintes de bleu à 0.

2 Algorithme de compression sans perte

L'objectif de cette partie est d'implémenter l'algorithme de compression sans perte *Run Length Encoding* (RLE).

1. Écrire une fonction qui permet de trier les 3 couleurs d'une image : on stockera tous les *GLubyte* de rouge, de vert et de bleu dans un grand tableau.

2. Écrire une fonction qui permet de compresser un tableau de *GLubyte* selon le principe de l'algorithme RLE naïf.

3. Écrire une fonction qui décompresse un tableau de *GLubyte*, puis une fonction qui reconstitue l'image à partir du tableau trié de couleurs.

4. Écrire une version améliorée de cet algorithme en ajoutant dans la compression l'amélioration vue en cours avec les multiplicateurs négatifs.

3 Color Look Up Tables et réduction de couleurs

L'objectif de cette Section est de déterminer la table d'indirection de couleurs (C-LUT) afin de pouvoir réduire le nombre de couleurs de notre image.

1. Écrire une fonction qui permet de construire la C-LUT : on utilisera une structure `Color` pour construire un tableau de couleurs.

2. Écrire une fonction qui permet de déterminer pour chaque couleur dans la C-LUT sa fréquence d'apparition, afin de ne garder au final que les couleurs prédominantes. On pourra commencer par trier la C-LUT via une fonction qui à une couleur associe un entier, par exemple celle vue en cours. Pour ce faire, on pourra utiliser la fonction `qsort` de *stdlib.h*, ou réimplémenter par exemple un tri à bulles.

3. Écrire une fonction qui sélectionne `k` couleurs de la C-LUT (les `k` plus fréquentes), et qui remplace chaque pixel de l'image par un pixel de la couleur la plus proche parmi celles sélectionnées. On testera le résultat produit pour différentes valeurs de `k`.