Ditherpunk: retour au monochrome

Florent Becker

Dans ce TP, vous allez réaliser quelques manipulations d'images à l'aide de la bibliothèque rust `image`. L'énoncé de ce TP est largement inspiré de la page https://surma.dev/things/ditherpunk/ que vous consulterez pour des explications plus détaillées.

L'énoncé détaille quelques traitements visant à transformer une image en une version utilisant peu de couleurs: soit monochrome (noir et blanc, sans gris), soit vers une palette avec peu de couleurs.

Vous ferez un rendu de votre TP sous forme d'une application en ligne de commande permettant d'appliquer les traitements que vous avez implémentés. Votre application permettra:
- de sélectionner l'image en entrée
- de sélectionner un nom pour le fichier de sortie (par défaut, `out.png`)
- de choisir le traitement à appliquer et ses options.

Cette sélection se fera via la ligne de commande, en utilisant la bibliothèque `argh` (https://crates.io/crates/argh). Les questions du TP servent de guide, c'est votre application qui sera évaluée.

Vous travaillerez par binôme, en implémentant en priorité les parties 1 à 4 et 7. Les parties 5 et 6 vous permettent de passer d'un niveau satisfaisant à formidable.

Ce travail est à rendre pour le 23 janvier 23h59.

1 La bibliothèque `image`

La bibliothèque `image` (https://crates.io/crates/image) permet d'ouvrir des images depuis un programme Rust. Une fois l'image ouverte, on accède à un tableau de pixels, dont le type dépend du type de fichier. La dernière version de la bibliothèque `image`, la 0.25 est incompatible avec le compilateur Rust installé sur les machines de l'IUT. Nous installerons donc la version 0.24. Sa documentation est disponible là: https://docs.rs/image/0.24.9/image/index.html .

Question 1

Créer un nouveau projet Cargo, avec une dépendance sur la bibliothèque `image`, version 0.24.

Question 2

Pour ouvrir une image depuis un fichier, on utilise `ImageReader::open("myimage.png")?.decode()?;` On obtient un `DynamicImage`, à quoi correspond ce type? Comment obtenir une image en mode rbg8 à partir de ce DynamicImage?

Indiquer les réponses dans votre README.

Question 3

Sauver l'image obtenue au format `png`. Que se passe-t-il si l'image de départ avait un canal alpha?

Expliquer dans le README de votre rendu ce qui se passe ici.

Question 4

Afficher dans le terminal la couleur du pixel $(32, 52)$ de l'image de votre choix.

On peut réaliser une boucle sur les pixels d'une image avec les méthodes `enumerate_pixels` et `enumerate_pixels_mut`, selon que l'on ait besoin de les modifier ou non.

Question 5

Passer un pixel sur deux d'une image en blanc. Est-ce que l'image obtenue est reconnaissable?

2 Passage en monochrome par seuillage

Le premier traitement à implémenter est le passage direct en monchrome: pour chaque pixel, si sa luminosité est supérieure à 50%, on le remplace par du blanc; inférieure, on le remplace par du noir.

Figure 1: Passage en monochrome par seuilllage

Question 6

Comment récupérer la luminosité d'un pixel?

Question 7

Implémenter le traitement

Question 8

Permettre à l'utilisateurice de remplacer "noir" et "blanc" par une paire de couleurs au choix.

3 Passage à une palette

Figure 2: Passage du département à une palette 8 couleurs: noir, blanc, rouge, vert, bleu, jaune, magenta, cyan

On peut également indexer l'image sur une palette, c'est-à-dire une liste de couleurs. Pour celà, on remplace chaque pixel par la couleur de la palette qui est la plus proche de lui.

Question 9

Comment calculer la distance entre deux couleurs? Indiquer dans le README la méthode de calcul choisie.

Question 10

Implémenter le traitement

Question 11

Votre application doit se comporter correctement si on donne une palette vide. Vous expliquerez dans votre README le choix que vous avez fait dans ce cas.

4 Tramage aléatoire (dithering)

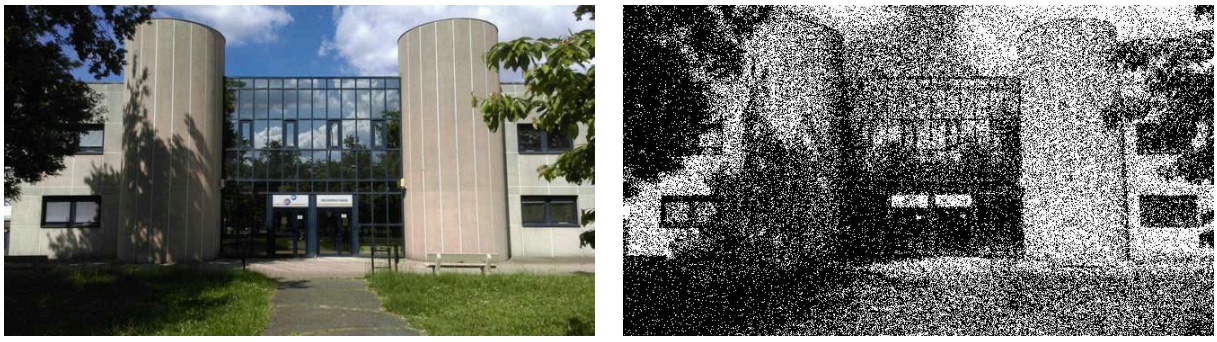

Figure 3: Passage en monochrome par dithering (tramage aléatoire)

Les deux traitements précédents conduisent généralement à de grands aplats, soit de noir et de blanc, soit d'une couleur de la palette. Pour retrouver un peu de nuance, on peut recourir à un bruit aléatoire. On appelle ce traitement *tramage* ou *dithering*. Le plus simple à implémenter est le tramage aléatoire. Dans un tramage aléatoire, pour chaque pixel de l'image, on tire au hasard un *seuil* entre 0 et 1, et on met le pixel en blanc si sa luminosité est supérieure au seuil.

Question 12

Implémenter le tramage aléatoire des images.

5 Utilisation de la matrice de Bayer comme trame

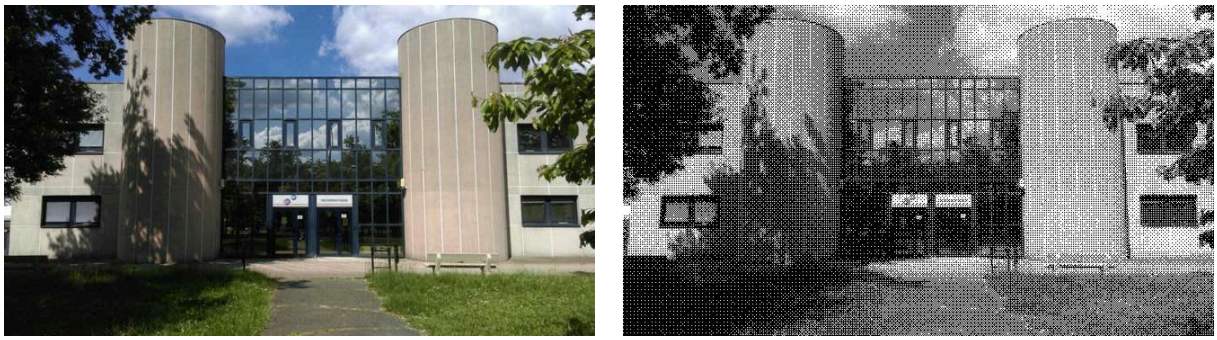

Figure 4: Passage en monochrome par matrice de Bayer (*ordered dithering*), ici d'ordre 4

Au lieu d'utiliser une valeur aléatoire par pixel, on peut se donner une trame, c'est à dire une matrice donnant pour chaque pixel la valeur de seuil à utiliser. Quand la matrice est plus petite que l'image à traiter, on la répète (comme une mosaïque). Cette méthode s'appelle "ordered dithering".

La matrice à utiliser s'appelle matrice de Bayer. Elle est définie par récurrence, en fonction d'un entier n appelé son ordre. La matrice B_n d'ordre n est définie ainsi:

- la matrice de Bayer d'ordre 0 $B_0 = (0)$ est une matrice nulle de taille 1×1
- la matrice de Bayer d'ordre $n+1$ est obtenue par blocs: $\frac{1}{4}\begin{pmatrix} 4.B_n & 4.B_n+2U_n \\ 4.B_n+3U_n & B_n+U_n \end{pmatrix}$, où U_n est une matrice de taille $2^n \times 2^n$ dont tous les coefficients valent 1.

On a donc $B_0 = (0)$, $B_1 = \frac{1}{4}.\begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix}$, $B_2 = \frac{1}{16}.\begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$...

Question 13

Déterminer B_3.

Question 14

Quel type de données utiliser pour représenter la matrice de Bayer? Comment créer une matrice de Bayer d'ordre arbitraire?

Question 15

Implémenter le tramage par matrice de Bayer.

6 Diffusion d'erreur

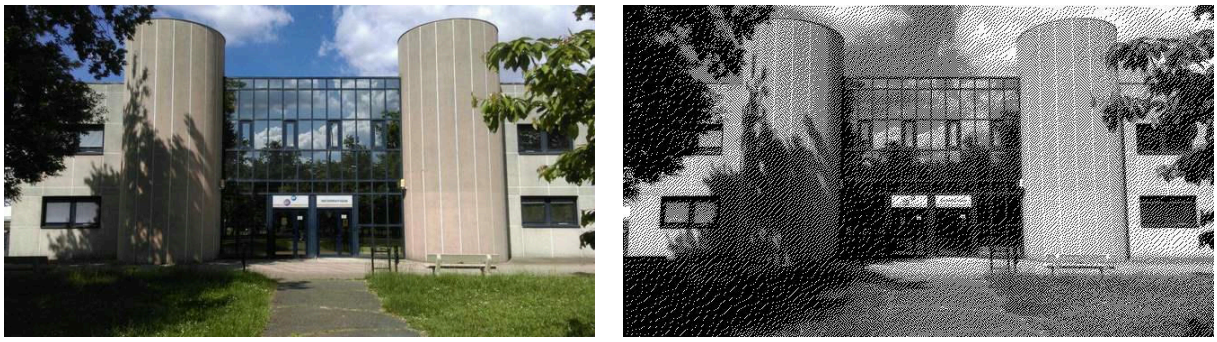

Figure 5: Passage en monochrome avec diffusion d'erreur sur 2 voisins

Plutôt que d'utiliser une trame, il existe aussi une technique de diffusion d'erreurs, documentée par exemple sur https://fr.wikipedia.org/wiki/Diffusion_d%27erreur#Diffusion_d'erreurs. Dans cette technique, après avoir calculé la couleur qui va remplacer chaque pixel, on calcule l'erreur que l'on a commise en faisant cette approximation, c'est à dire la différence entre la couleur choisie et la couleur réelle du pixel.

Quand on passe une image en monochrome, on est dans l'espace à une dimension des luminosités. L'erreur commise pour un pixel est donc égale à sa luminosité l si le pixel est remplacé par du noir, et $1 - l$ s'il est remplacé par du blanc. Si on passe l'image dans une palette de couleurs, l'erreur est un vecteur à 3 composantes (rouge, vert, bleu).

Pour chaque pixel, cette erreur est répartie entre ses voisins qui n'ont pas encore été testés. Ainsi, en mode monochrome, si un pixel est remplacé par du noir, les pixels voisins seront éclaircis avant d'être traités, tandis que s'il est remplacé par du blanc, ils seront rendus plus foncés. L'erreur est répartie entre ces voisins par une matrice avec un coefficient $*$ qui représente la position du pixel courant. Ainsi, si cette matrice est $\begin{pmatrix} * & 0.6 \\ 0.4 & 0 \end{pmatrix}$, 60% de l'erreur du pixel (x, y) est envoyée sur le voisin de droite $(x + 1, y)$ et 40% sur le voisin du dessous $(x, y + 1)$.

Question 16

Implémenter un mécanisme de diffusion d'erreur suivant la matrice $\begin{pmatrix} * & 0.5 \\ 0.5 & 0 \end{pmatrix}$ pour les images en noir et blanc.

Question 17

Pour une palette de couleurs comme dans la partie 3, expliquer dans votre README comment vous représentez l'erreur commise à chaque pixel, comment vous la diffusez.

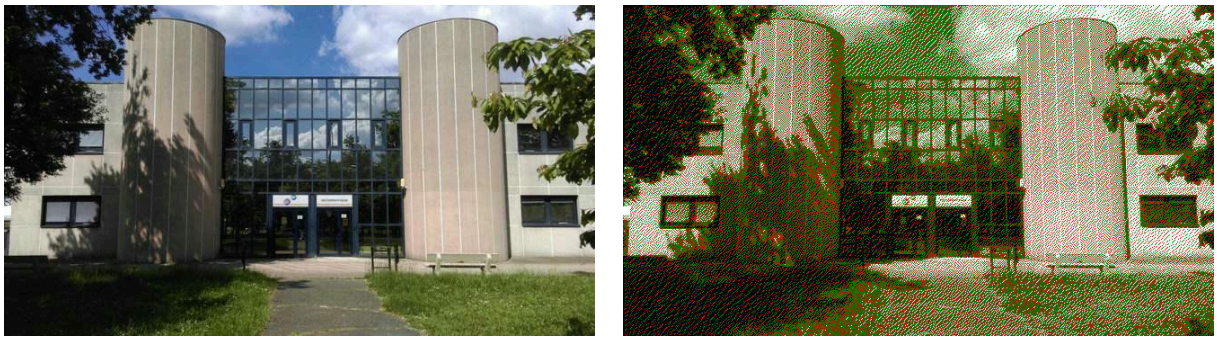

Figure 6: Passage en 5 couleurs (noir, blanc, rouge, bleu, vert) avec diffusion d'erreur sur 2 voisins

Question 18

Implémenter la diffusion d'erreur pour la palettisation d'images.

Question 19

Implémenter la diffusion d'erreur pour la matrice de Floyd-Steinberg $\frac{1}{16}\begin{pmatrix} 0 & * & 7 \\ 3 & 5 & 1 \end{pmatrix}$

Question 20

Comment représenter une matrice de diffusion d'erreur arbitraire? Permettre de changer de matrice de diffusion d'erreurs, et tester les matrices de diffusion de Jarvis-Judice-Ninke $\frac{1}{48}\begin{pmatrix} 0 & 0 & * & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{pmatrix}$ et Atkinson $\frac{1}{8}\begin{pmatrix} 0 & * & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$.

7 La bibliothèque `argh`

Pour l'implémentation de l'interface en ligne de commande de votre application, vous allez utiliser la bibliothèque `argh`. Pour utiliser cette bibliothèque, vous devez:

- spécifier votre interface
- transformer cette spécification en un type Rust
- utiliser les fonctions d'`argh` pour récupérer une instance de votre type à partir des options fournies à l'invocation de votre programme.

Quand on utilise `cargo run` pour lancer un programme Rust en cours de développement, on peut l'appeler ainsi:

```
$ cargo run --option_cargo1 --option_cargo2 -- option_prog1 option_prog2
```

Les options avant le "`--`" sont passées à cargo, celles qui le suivent sont passées au programme. L'appel donné au-dessus est donc essentiellement équivalent à:

```
$ cargo build --option_cargo1 --option_cargo2
$ mon_programme option_prog1 option_prog2
```

Question 21

Donner une spécification de votre interface sous forme d'un projet d'écran d'aide, tel que celui qui sera obtenu par `cargo run -- --help`.

Question 22

Déterminer le type Rust correspondant à une sélection d'options fournies par l'utilisateur.

Question 23

Implémenter votre interface en ligne de commande à l'aide de la directive `#[derive(FromArgs)]` sur votre type, suivant la documentation à https://docs.rs/argh/0.1.13/argh/ .

Pour les différentes opérations correspondant aux sous-parties ci-dessus, vous pouvez utiliser des sous-commandes.