

Rapport de soutenance

Etienne Lazaro, Andrew Youansamouth, Nathan Gillotin

27 février 2025



Table des matières

1	Introduction	3
1.1	Rappel du projet	3
2	Rappel concernant l'organisation du projet	4
3	Filtres	5
3.1	Les différents filtres	5
3.1.1	Filtre Médian	5
3.1.2	Filtre Moyen	5
3.1.3	Filtre Gaussien	5
3.1.4	Filtres Couleurs	6
3.2	Fonctions réalisées	7
3.2.1	Fonction median_filter	7
3.2.2	Fonction mean_filter	7
3.2.3	Fonction gaussian_filter	7
3.2.4	Fonction color_filter	7
3.2.5	Fonction grayscale_filter	7
3.2.6	Fonction sepia_filter	8
3.2.7	Fonction invert_colors_filter	8
3.3	Choix d'Implémentation	9
3.3.1	Chargement de l'image d'entrée	9
3.3.2	Application des Filtres de Couleur	9
3.3.3	Application des Filtres de Réduction de Bruit	9
3.3.4	Enregistrement des Résultats:	9
4	Redimensionnement d'image et GTK	10
4.1	Redimensionnement d'image	10
4.2	Gtk	10
4.3	Difficultées	11
4.4	Prévisions pour la suite	11
5	Segmentation d'image	12
6	Transformation d'image	13
6.1	Rotation d'image	13
6.1.1	utilité	13
6.2	Site internet	13
6.2.1	Acceuil	13
6.2.2	Projet	13
6.2.3	Téléchrgements	13
7	Annexes	14
8	Conclusion	18

1 Introduction

Dans ce rapport de soutenance nous allons expliquer en détail l'avancement de l'état de notre projet

1.1 Rappel du projet

Lorsqu'on considère les diverses opérations de modification d'images, un éventail de possibilités s'offre à nous, permettant ainsi de façonner et d'adapter les visuels en fonction des besoins spécifiques de l'utilisateur. Parmi ces opérations, la rotation d'image offre la possibilité de donner une perspective nouvelle à une scène, d'explorer des angles inattendus, et ce, sans compromettre la qualité visuelle. Ce processus, bien qu'essentiellement mathématique, a le pouvoir de métamorphoser l'aspect visuel d'une image en jouant avec les positions relatives de ses pixels.

De même, le redimensionnement d'image, un processus de modification de la taille de l'image, se révèle être une démarche incontournable pour ajuster la résolution et l'échelle d'une image selon les exigences spécifiques d'une application. Cette opération, tout en préservant les détails importants de l'image, peut influencer de manière significative la perception visuelle de la scène capturée.

La translation d'image, quant à elle, ouvre la voie à la délocalisation de la scène visuelle dans le plan bidimensionnel. C'est une opération qui permet de déplacer l'image selon un vecteur de translation défini, offrant ainsi une flexibilité accrue dans le positionnement spatial de la représentation visuelle.

Ces opérations, bien qu'elles puissent être mises en œuvre sans l'aide de bibliothèques spécialisées, bénéficient grandement de l'utilisation d'outils tels que Pillow. La bibliothèque Pillow facilite non seulement l'implémentation de ces transformations, mais elle offre également une interface conviviale pour explorer et expérimenter avec divers ajustements visuels.

C'est pour cela que nous, Xuligion, avons décidé de concevoir une application rassemblant toutes ces fonctionnalités. Nous allons proposer à l'utilisateur la possibilité d'éditer ses images à souhait et de les sauvegarder de nouveau sur leur appareil ! Il s'agit donc d'un éditeur d'image.

2 Rappel concernant l'organisation du projet

Interface utilisateur	
Interface utilisateur	0%
Sauvegarde de l'image	100%
Undo (Ctrl + Z)	0%
Redo (Ctrl + Shift + Z)	0%
Filtrage d'images	
Filtre Médian	100%
Filtre Moyen	100%
Filtre Gaussien	100%
Filtre de couleur	100%
Segmentation d'images	
Segmentation d'image	50%
Detection de Region d'Interet (ROI)	0%
Transformation d'images	
Rotation d'image	20%
Redimensionnement d'image	20%
Translation d'image	0%

3 Filtres

Pour cette première soutenance, le défi résidait dans la conception et la mise en œuvre de différents filtres d'image en utilisant le langage de programmation Rust. Les fonctions développées visaient à améliorer la qualité visuelle des images en appliquant diverses techniques de filtrage. Dans cette section, nous explorerons trois catégories principales de filtres: médian, moyen et gaussien, ainsi que les filtres de couleurs incluant la conversion en nuances de gris, l'effet sépia et l'inversion des couleurs.

3.1 Les différents filtres

Les filtres d'image sont des outils cruciaux pour améliorer la clarté, la netteté et l'esthétique des images. Dans notre projet, nous avons abordé trois types de filtres principaux, chacun ayant ses propres avantages et applications spécifiques.

3.1.1 Filtre Médian

Le filtre médian est une technique de filtrage qui vise à réduire le bruit et à améliorer la qualité des images en remplaçant chaque pixel par la valeur médiane des pixels voisins. Notre implémentation en Rust prend en compte les pixels dans un voisinage de 3x3 autour de chaque pixel cible, calculant ainsi la médiane et l'appliquant pour obtenir une image filtrée.

3.1.2 Filtre Moyen

Le filtre moyen, également connu sous le nom de filtre de moyenne, est une méthode de lissage qui remplace chaque pixel par la moyenne des valeurs des pixels environnants. En Rust, notre fonction moyenne_filter prend en compte un voisinage de 3x3, accumulant les valeurs des canaux de couleur et calculant la moyenne pondérée pour produire une image lissée.

3.1.3 Filtre Gaussien

Le filtre gaussien est un filtre de flou couramment utilisé pour atténuer le bruit et améliorer la netteté des images. En Rust, notre implémentation utilise une matrice de noyau gaussien générée dynamiquement en fonction de l'écart-type (sigma) fourni. La convolution de cette matrice avec l'image d'entrée produit une image filtrée avec un effet de flou contrôlé.

3.1.4 Filtres Couleurs

Les filtres de couleurs sont des transformations qui modifient l'apparence chromatique d'une image. Nous avons implémenté trois filtres de couleurs distincts: la conversion en nuances de gris, l'effet sépia et l'inversion des couleurs.

- CONVERSION EN NUANCE DE GRIS : La fonction `grayscale_filter` convertit une image en nuances de gris en ajustant les canaux RGB en fonction de leur contribution à la luminosité. Cette approche offre une représentation monochromatique de l'image originale.
- EFFET SEPIA : L'effet sépia, reproduisant l'apparence des anciennes photographies, est obtenu en ajustant les canaux RGB de chaque pixel selon des coefficients spécifiques. Notre fonction `sepia_filter` applique ces coefficients pour obtenir un rendu sépia authentique.
- INVERSION DES COULEURS L'inversion des couleurs, réalisée par la fonction `invert_colors_filter`, consiste à remplacer chaque canal de couleur par son complément afin de créer une image avec des couleurs inversées.

3.2 Fonctions réalisées

3.2.1 Fonction median_filter

La fonction median_filter prend une image dynamique en entrée et applique un filtre médian pour réduire le bruit et améliorer la qualité visuelle de l'image. Chaque pixel de l'image résultante est calculé en prenant la médiane des valeurs des pixels dans un voisinage 3x3 autour du pixel d'origine. La fonction est implémentée de manière modulaire, itérant sur tous les pixels et utilisant des opérations de tri pour trouver la médiane.

3.2.2 Fonction mean_filter

La fonction mean_filter réalise un filtrage moyen, lissant l'image en remplaçant chaque pixel par la moyenne pondérée de ses voisins dans un voisinage 3x3. Les canaux RGB de chaque pixel sont cumulés et la moyenne est calculée. Cette approche contribue à atténuer les variations de couleur indésirables dans l'image.

3.2.3 Fonction gaussian_filter

La fonction gaussian_filter applique un filtre gaussien à une image dynamique en utilisant une convolution avec un noyau gaussien généré dynamiquement en fonction de l'écart-type fourni (sigma). Elle permet de créer un effet de flou contrôlé. La fonction utilise également une sous-fonction generate_gaussian_kernel pour créer le noyau gaussien et une sous-fonction convolution pour effectuer la convolution.

- SOUS FONCTION GENERATE_GAUSSIAN_KERNEL : La sous-fonction generate_gaussian_kernel calcule une matrice de noyau gaussien en fonction de l'écart-type (sigma). La matrice résultante est normalisée pour garantir une somme égale à 1, assurant ainsi un filtrage correct.
- SOUS FONCTION CONVOLUTION : La sous-fonction convolution effectue une convolution bidimensionnelle entre une image d'entrée, un noyau gaussien et produit une image de sortie résultante. Cette opération est cruciale pour appliquer le filtre gaussien avec précision.

3.2.4 Fonction color_filter

La fonction color_filter offre la possibilité d'appliquer différents filtres de couleurs à une image en fonction de la valeur n fournie. Elle agit comme un sélecteur pour les filtres de conversion en nuances de gris (grayscale_filter), sépia (sepia_filter) et inversion des couleurs (invert_colors_filter), ou renvoie simplement l'image d'origine selon le cas.

3.2.5 Fonction grayscale_filter

La fonction grayscale_filter convertit l'image en nuances de gris en ajustant les canaux RGB de chaque pixel en fonction de leur contribution à la luminosité. Elle utilise une pondération des canaux pour obtenir une représentation monochromatique de l'image originale.

3.2.6 Fonction `sepia_filter`

La fonction `sepia_filter` applique un effet sépia à l'image en ajustant les canaux RGB de chaque pixel selon des coefficients spécifiques. Elle reproduit l'apparence des anciennes photographies sépia en donnant à l'image une teinte chaude et nostalgique.

3.2.7 Fonction `invert_colors_filter`

La fonction `invert_colors_filter` inverse les couleurs de chaque pixel en remplaçant chaque canal par son complément. Cette opération crée une image avec des couleurs inversées, offrant un effet visuel distinctif.

3.3 Choix d'Implémentation

Les fonctions sont implémentées de manière modulaire, itérant sur les pixels de l'image à traiter et appliquant des opérations spécifiques à chaque fonction. Les choix d'implémentation sont guidés par la simplicité, l'efficacité et la clarté du code. Des sous-fonctions sont utilisées pour faciliter la gestion de tâches spécifiques, assurant ainsi une organisation structurée du code.

Le programme principal (main) utilise le module filters pour appliquer différentes opérations de filtrage à une image d'entrée. Actuellement, le main est temporaire et sert à démontrer le fonctionnement des fonctions de filtrage. À terme, l'utilisateur pourra interagir avec ces fonctionnalités à travers une interface graphique plus conviviale.

Voici une description des opérations effectuées par le main :

3.3.1 Chargement de l'image d'entrée

Le chemin de l'image est spécifié et elle est ouverte à l'aide de la bibliothèque image.

3.3.2 Application des Filtres de Couleur

Trois filtres de couleur sont appliqués à l'image d'entrée :

- Filtrage en nuances de gris (color_filter avec l'argument 1).
- Effet sépia (color_filter avec l'argument 2).
- Inversion des couleurs (color_filter avec l'argument 3).

Chaque image résultante est enregistrée dans un fichier PNG correspondant.

3.3.3 Application des Filtres de Réduction de Bruit

Trois filtres de réduction de bruit sont appliqués à l'image d'entrée :

- Filtre médian (median_filter).
- Filtre moyen (mean_filter).
- Filtre gaussien (gaussian_filter avec un écart-type de 1.5).

Chaque image résultante est enregistrée dans un fichier PNG correspondant.

3.3.4 Enregistrement des Résultats :

Chaque image résultante est sauvegardée dans un fichier PNG avec un nom correspondant au type de filtre appliqué.

Le main offre une démonstration pratique des fonctionnalités de filtrage. Cependant, à l'avenir, ces opérations seront intégrées à une interface graphique conviviale, permettant à l'utilisateur de choisir les filtres à appliquer et d'ajuster les paramètres selon ses préférences.

4 Redimensionnement d'image et GTK

4.1 Redimensionnement d'image

Pour cette première soutenance je me suis occupé de commencer à planifier l'interface graphique et j'ai codé la fonction de redimensionnement d'image. Pour ce qu'il s'agit du redimensionnement d'image, la fonction quand exécutée va charger une image depuis un path, qui est pour le moment n'est pas modifiable mais qui le deviendra avec la méthode "readline". Une fois l'image trouvée la fonction va demander à l'utilisateur d'entrer des dimensions x et y, pour ce faire j'utilise la méthode "readline" qu'on a appris pendant les tp, ces deux valeurs représenteront les nouvelles coordonnées à appliquer à l'image, la fonction va ensuite utiliser la méthode "resize" du module "DynamicImage" pour appliquer ces nouvelles coordonnées à une copie de l'image. Une fois que le main a récupéré cette copie il va la sauvegarder avec la méthode "save" sur un path choisi de nouveau arbitrairement, comme pour le premier path, avec l'implémentation de l'interface graphique l'utilisateur pourra choisir ou sauvegarder son image.

J'ai aussi prévu de rajouter une option "undo" au logiciel, même si rien de concret ne fonctionne actuellement. Une pile sera initialisée quand le logiciel sera ouvert et à chaque modification de l'image, le logiciel va insérer l'image avant modification au sommet d'une pile. Appeler le undo va pop le sommet de cette pile et donc ramener l'image avant la dernière modification.

4.2 Gtk

Cela conclut ma partie sur les options fondamentales du logiciel, mais j'ai également commencé à travailler sur à quoi ressemblerait l'interface graphique du logiciel. Elle sera faite sur gtk car c'est la bibliothèque graphique avec laquelle je suis le plus familier, quand au squelette de l'interface, il sera fait sur glade et va inclure un bouton pour chaque option, avec des champs de texte pour entrer des informations numériques et des boutons pour charger des fichiers en fonctions des options du logiciel.

Les difficultés de cette interface vont venir de gtk en rust, déjà car je vais devoir comprendre comment appeler toutes les méthodes gtk que je connais en rust ce qui est normal mais également car après quelques tests j'ai compris que gtk fonctionne d'une manière spécifique avec le système d'ownership de rust, je n'ai pas pleinement compris le fonctionnement mais je pense que certaines méthodes peuvent attribuer l'ownership de certaines valeurs à des sous fonctions et que par conséquence elle ne deviennent plus utilisables dans le cas général, je vais donc devoir trouver comment remédier à ce problème.

4.3 Difficultées

La difficulté principale était la compréhension et l'organisation du projet sous rust, on s'y attendait et c'est pour cela qu'on a visé des objectifs assez simples pour cette première soutenance, afin de ne pas foncer contre un mur et plutôt rentrer doucement dans le bain du langage avec des applications simples du projet. Cette étape est importante car maintenant que je suis plus à l'aise sur l'environnement du projet je peut commencer à réfléchir aux parties plus complexes du projet avec une crainte atténuer et une idée de résolution plus claire.

4.4 Prévisions pour la suite

Comme expliqué ma priorité sera de faire fonctionner l'interface graphique pour le projet et également apporter mon aide à d'autres membres du groupe sur leurs parties si nécessaire.

Je vais déjà coder la fonction undo, puis je vais commencer ma documentation sur gtk en rust, il faut que je comprenne les différences avec le C et comment appeler les méthodes de la librairie en rust (exemple, les signaux des boutons du logiciel), ensuite je vais faire l'interface graphique sur glade.

5 Segmentation d'image

La segmentation d'image est un processus fondamental en traitement d'images consistant à diviser une image en régions ou segments, souvent basées sur des caractéristiques telles que la couleur, la texture, le contraste, etc. Cette technique est largement utilisée dans de nombreux domaines, notamment la vision par ordinateur, la reconnaissance de formes, la médecine, la cartographie, etc.

Dans ce code Rust, nous avons utilisé la bibliothèque `image`, qui fournit des fonctionnalités de base pour travailler avec des images. Tout d'abord, nous avons chargé une image à partir d'un fichier en utilisant la fonction `image::open`, qui retourne une structure `DynamicImage` représentant l'image. Ensuite, pour pouvoir manipuler les pixels individuels de l'image, nous avons converti cette `DynamicImage` en une image `RGBA` en utilisant la méthode `to_rgba8`.

Ensuite, nous avons parcouru chaque pixel de l'image d'entrée en utilisant deux boucles `for` imbriquées. Pour chaque pixel, nous avons extrait sa couleur en utilisant la méthode `get_pixel` et appliqué une technique simple de segmentation par seuillage. Dans cette approche, nous avons choisi un seuil fixe (dans ce cas, 128) et comparé la valeur de la composante rouge du pixel à ce seuil. Si la valeur de la composante rouge est supérieure au seuil, nous avons attribué la couleur blanche au pixel dans l'image segmentée, sinon nous lui avons attribué la couleur noire.

Enfin, une fois que tous les pixels ont été segmentés, nous avons sauvegardé l'image segmentée dans un fichier en utilisant la méthode `save`. L'image segmentée est sauvegardée au format `PNG` avec le nom de fichier "segmented_image.png". Cette approche simple de segmentation par seuillage constitue une introduction élémentaire à la segmentation d'image, offrant une base pour des techniques plus avancées telles que la segmentation basée sur la région, la segmentation par contour, ou même des méthodes plus sophistiquées telles que la segmentation par apprentissage profond.

6 Transformation d'image

6.1 Rotation d'image

La rotation d'une image implique de faire pivoter tous les pixels autour d'un point central. Cela peut être effectué sans l'utilisation de bibliothèques spécifiques de traitement d'images en utilisant les principes mathématiques de la rotation. Nous avons souhaiter utiliser la bibliothèque imageproc afin de simplifier les calculs et gagner en optimisation.

6.1.1 utilité

Ces trois fonctions trouveront leur pleine utilité une fois implémenté dans une interface graphique. comme vu précédament, et surtout pour la translation d'image, certaines fonctions ne peuvent pas donner de résultats interessant, puisque peu utilisable tel quel pour n'importe quel objectif, mais leur utilité deviendront très importante une fois qu'ils peuvent s'utiliser avec aisance, comme lorsque utiliser dans une interface graphique.

6.2 Site internet

6.2.1 Acceuil

Le site d'accueil de la page interet montre le logo d'épita et de notre groupe.

6.2.2 Projet

cette page est la page principale de notre site. elle regroupe l'avancée du projet, une présentation du projet, et surtout une présentation de chacun des membres.

6.2.3 Téléchargements

depuis cette page, nous pouvons imprimer les deux rapport ainsi qu'un executable.

7 Annexes

```
extern crate image;
use image::{DynamicImage, GenericImageView, ImageOutputFormat, Rgba, RgbaImage};

fn main() {
    let img: DynamicImage = image::open("example.jpg").expect("Failed to open image file");
    let img_rgba: RgbaImage = img.to_rgba8();
    let (width, height) = img_rgba.dimensions();
    let mut segmented_img = RgbaImage::new(width, height);

    for y in 0..height {
        for x in 0..width {
            let pixel_color = img_rgba.get_pixel(x, y);
            let threshold: u8 = 128;
            let new_pixel_color = if pixel_color[0] > threshold {
                Rgba([155, 255, 255, 255])
            } else {
                Rgba([0, 0, 0, 255])
            };
            segmented_img.put_pixel(x, y, new_pixel_color);
        }
    }
    let segmented_dynamic_img: DynamicImage = DynamicImage::ImageRgba8(segmented_img);

    segmented_dynamic_img
        .save("segmented_image.png")
        .expect("Failed to save segmented image file");
}
```



FIG. 1 – *Image test filtres*

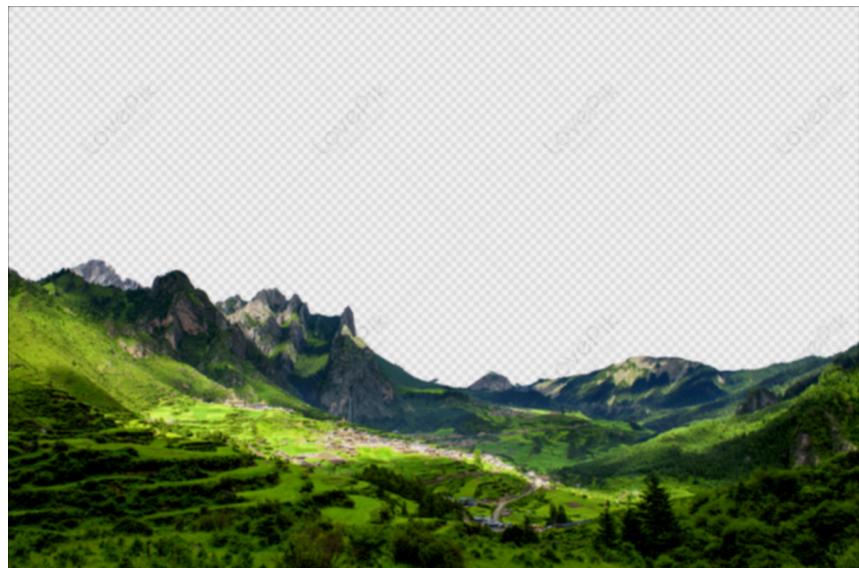


FIG. 2 – *Filtre Gaussien*



FIG. 3 – *Filtre Moyen*



FIG. 4 – *Filtre Median*



FIG. 5 – *Filtre de couleur : Grayscale*



FIG. 6 – *Filtre de couleur : Sepia*

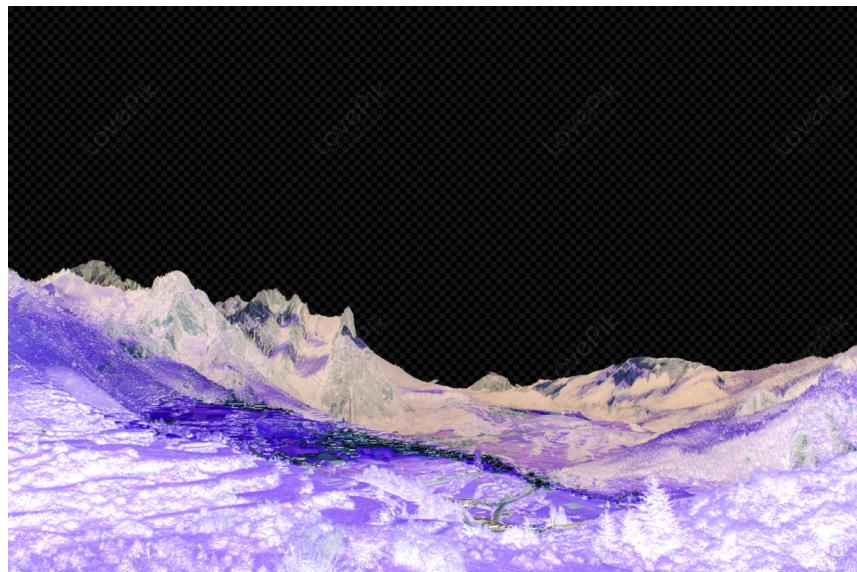


FIG. 7 – *Filtre de couleur : Inverse*

8 Conclusion

Lors de la décision du projet nous n'étions pas familiers avec le langage rust, nous avions donc choisi un sujet qui reprenait des éléments du projet s3 afin d'avoir une bonne idée de nos objectifs et comment les accomplir dès le départ, la difficulté étant donc de les appliquer en rust avec les librairies propres au langage. Notre priorité pour cette première soutenance était donc de nous familiariser avec le langage et d'avoir un plan concret sur comment appliquer nos plans, c'est également pour cette raison que nous nous sommes focalisés sur des applications des méthodes basiques du projet pendant cette première soutenance, car pour nous, essayer de concrétiser le projet alors que nous n'étions pas familiers avec notre environnement de travail n'allait amener nulle part.