

Compte-rendu Image2D

Alan Métivier

07 Novembre 2019

Contents

1 Transformation de l'Image2D	1
1.1 Seuillage	1
1.2 Conversion	2
1.3 Rotation Translation	2
1.4 Opérateurs	4
1.5 Filtrage and Bruit	4
1.5.1 Salt and Pepper	4
1.5.2 Masque	4
1.5.3 Convolution	5
1.5.4 Histogramme	11
1.5.5 Filtrage Automatique	12
1.5.6 Discussion	12
2 Application gtkmm	13

1 Transformation de l'Image2D

Nous allons parler en premier des fonctions intrinsèques à la classe Image2D. Ces fonctions permettent de manipuler l'objet Image2D et de les transformer. La transformation d'images permet de retrouver des formes, de filtrer du bruit, de rajouter du bruit ou simplement de la tourner. Ce compte-rendu est la suite du dernier compte-rendu expliquant les fonctions getter et setter de Image2D ainsi que la création d'une image bmp. De nombreuses modifications ont depuis amélioré ces getter et setter.

Les images de ce compte-rendu peuvent avoir des problèmes de couleurs ou de bruit mais cela est dû à la librairie Gtkmm. Quand on les "imprime" avec la classe Image2D, elles n'ont pas de problème.

1.1 Seuillage

Seuiller une image consiste à binariser celle-ci. C'est à dire qu'on recréé une image de 0 ou 1. Au delà d'un certain seuil, on attribut la valeur maximale

(255) et 0 sinon. Pour une image en niveau de gris, il suffit de récupérer la valeur du pointeur de niveau de gris. Mais pour une image en RGB, en fonction de la couleur qu'on veut seuiller, on met la valeur maximale si la valeur de cette composante dépasse le seuil et on met la valeur des autres pointeurs à zéro sinon on met tous à zéro.

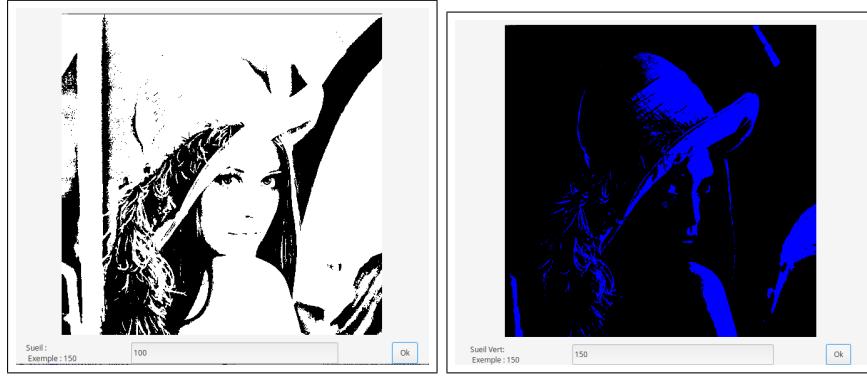


Figure 1: a - lena.bmp seuillé en grey; b - lena.bmp seuillé en RGB

1.2 Conversion

L'image se caractérise par les composantes utilisées, l'Image2D peut être en niveau de gris ou en RGB. Un attribut définit l'option de l'image, nommée "option". Cependant, il est courant de, par exemple, seuiller en niveau de gris. Il y a donc deux fonctions de conversion : RGB to Grey et Grey to RGB. A partir d'un objet Image2D, il est donc possible de modifier "option". Pour passer d'une image RGB vers une image en niveau de gris, il suffit de prendre la moyenne des trois composantes RGB. Pour passer d'une image gris en image RGB, il est pas si facile de retrouver la couleur (même s'il y a maintenant des possibilités avec l'utilisation du Deep Learning), donc on copie la valeur en niveau gris dans les 3 composantes : RGB.

1.3 Rotation Translation

Les deux fonctions suivantes ont pour but de tourner et de déplacer l'image.

- Rotate : La rotation permet de tourner les pixels d'une image en fonction d'un angle donné. Le pixel déplacé est calculé par la relation suivante :

$$\begin{aligned} X &= \cos(\text{teta}) * (i - \text{centreX}) - \sin(\text{teta}) * (j - \text{centreY}) + \text{centreX} ; \\ Y &= \sin(\text{teta}) * (i - \text{centreX}) + \cos(\text{teta}) * (j - \text{centreY}) + \text{centreY} ; \end{aligned}$$



Figure 2: a - RGB to grey; b - Grey to RGB



Figure 3: Image tourné de 20 degrés

Si le pixel calculé sort de l'image, on ne l'enregistre pas. Les autres pixels sont mis à 0.

- Translate :

La translation permet de déplacer les pixels. Pour faire ça, il suffit de créer une matrice temporaire et de décaler tout les pixels. Les pixels qui sortent de l'image ne sont pas enregistrés et les nouveaux pixels sont mis à 0.

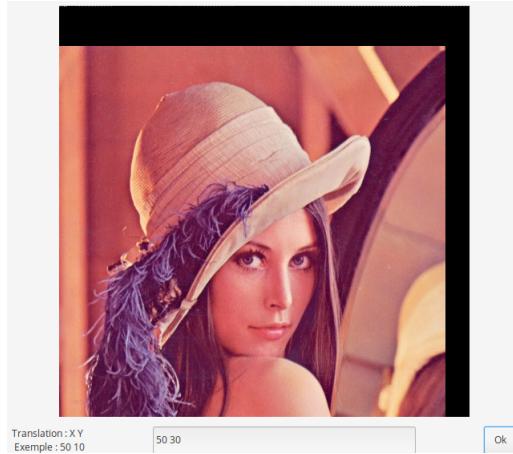


Figure 4: Image translaté de 50x et 30y

1.4 Opérateurs

Les opérateurs de surcharges peuvent donner une signification aux opérateurs. Il est utile (notamment dans l'application gtkmm) de dire qu'un objet `Image2D` est égale à une autre objet `Image2D`. C'est à dire qu'il aura les mêmes attributs. Il est possible aussi de sommer deux objets afin de sommer la valeur de leurs pixels par exemple. Par manque de temps, l'opérateur `==` et `!=` n'ont pas pu être codés bien qu'ils pourraient être très utile afin de savoir s'il existe une égalité entre les deux objets.

1.5 Filtrage and Bruit

Les fonctions de filtrage et bruit permettent de simuler un cas physique et de pouvoir trouver la solution de filtrage mais elles permettent aussi de faire la détection des coins ou des lignes rouge à 45 degré par exemple.

1.5.1 Salt and Pepper

On ajoute premièrement une fonction permettant de bruiter l'image. Le bruit Salt and Pepper apparaît généralement lors d'erreur de transmission ou alors de dysfonctionnement du capteur. Le principe pour simuler ce bruit est assez basique, il suffit de modifier `x` nombre de pixels (donné par l'utilisateur) choisis aléatoirement et de leurs attribuer aléatoirement la valeur de 255 ou de 0.

1.5.2 Masque

Le masquage d'une image permet de découper la région d'intérêt. Dans la classe `Image2D`, cette fonction demande une autre `Image2D` en argument afin

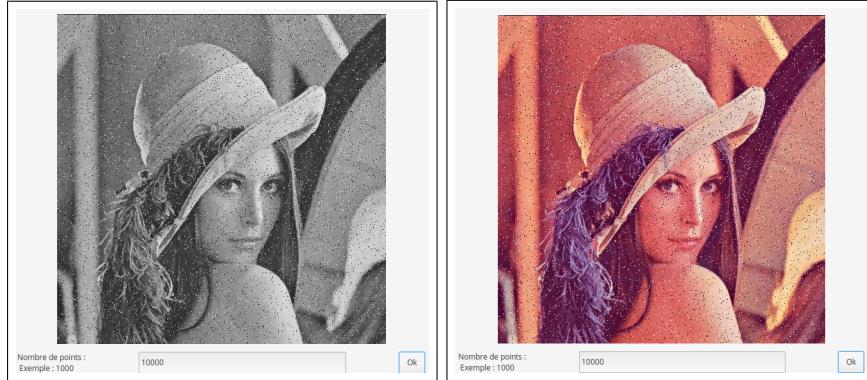


Figure 5: a - Salt and Pepper sur image en grey; b - SnP sur image en RGB

de masquer l'image courante. **Attention cette fonction ne fonctionne pas dans l'application Gtkmm.(Problème d'import d'image)**

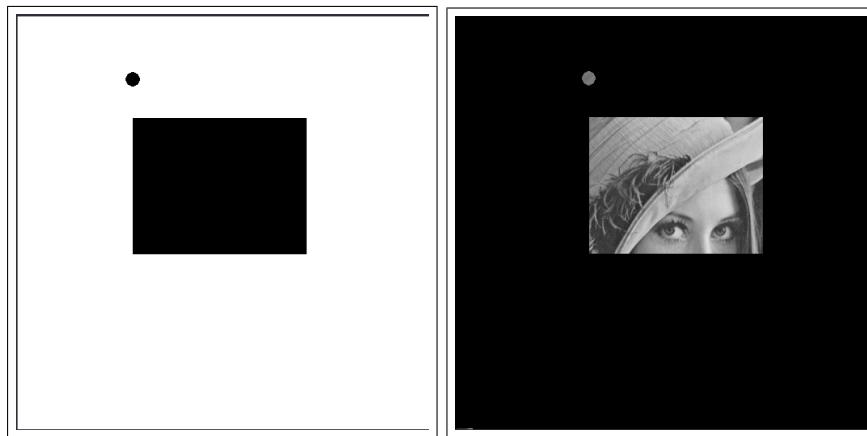


Figure 6: a - Masque; b - lena.bmp masqué

1.5.3 Convolution

La fonction Convolution de la classe Image2D est la plus utile dans le traitement d'image. La fonction étant généralisé permet l'opération de convolution avec toutes les tailles de noyaux. Nous avons donc plusieurs noyaux d'implémenter :

- Le noyau moyennant 3x3 et 5x5 : $\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$ $\begin{bmatrix} 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \end{bmatrix}$



Figure 7: Image utilisé

Ce noyau permet de lisser l'image. Il est utilisé dans le traitement d'image



Figure 8: a - Moyen 3x3; b - Moyen 5x5

pour "enlever" le bruit salt and pepper notamment.

- Le passe-haut : $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

Ce noyau permet d'atténuer l'énergie des basses fréquences de l'image. C'est à dire qu'elle fait ressortir le contraste.

- Le noyau de Sobel : $G_x \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

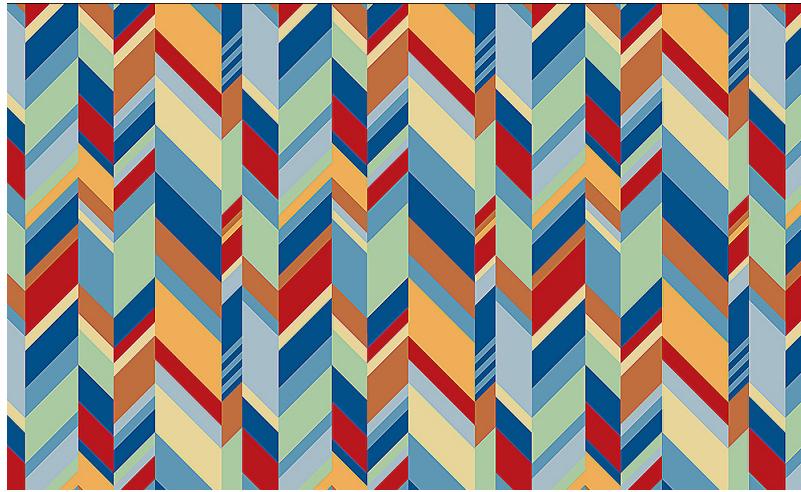


Figure 9: Passe-Haut

Ce noyau est utilisé dans la détection de contours.

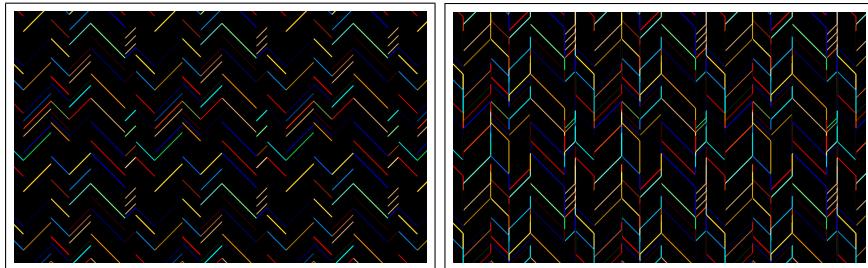


Figure 10: a - Gx; b - Gy

- Passe-Bas : $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Ce noyau est à l'inverse du Passe-haut celui qui laissera passer les basses fréquences.

- Laplacien : $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Ce noyau est utilisé aussi dans la détection de contours. Les zéros dans

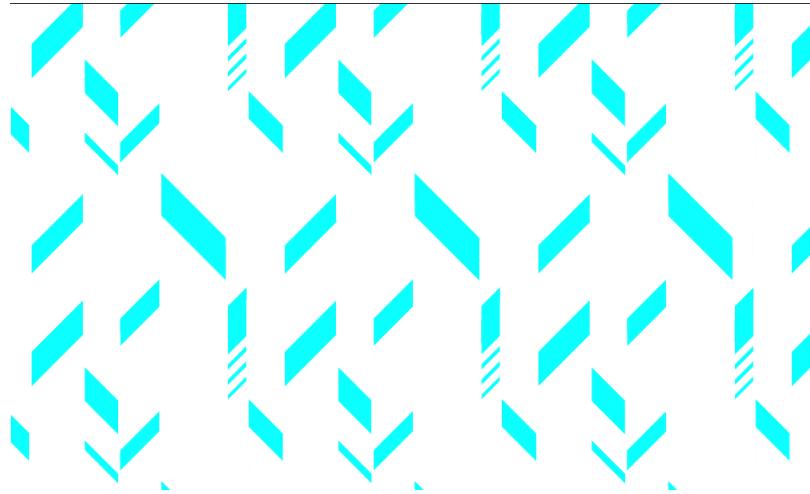


Figure 11: Passe-Bas

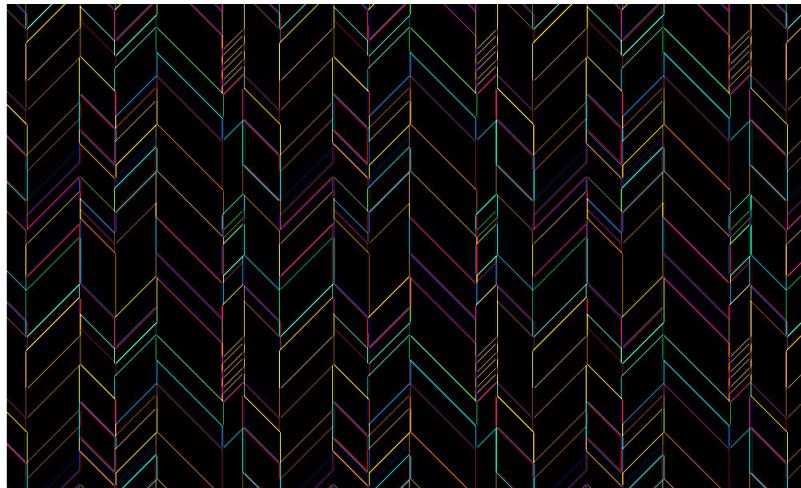


Figure 12: Laplacien

la matrice permettent de ressortir le contour.

- Prewitt : $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ $M_z = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

Ce noyau permet aussi la détection de contour, il permet de calculer une approximation du gradient d'intensité lumineuse.

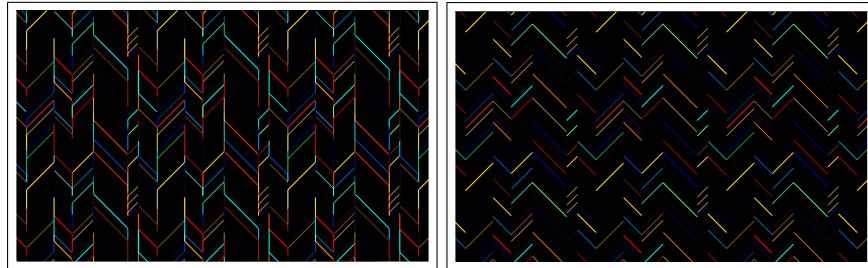


Figure 13: a - My; b - Mz

- Gaußien :
$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

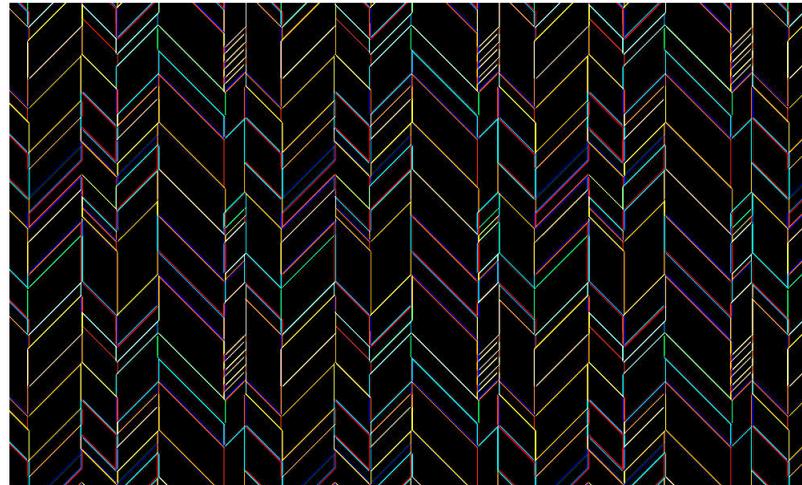


Figure 14: Gaußien

Ce noyau utilise la courbe Gaußienne afin de détecter les bords.

- Masque horizontal :
$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$$

Ce noyau permet de retrouver les lignes horizontales.

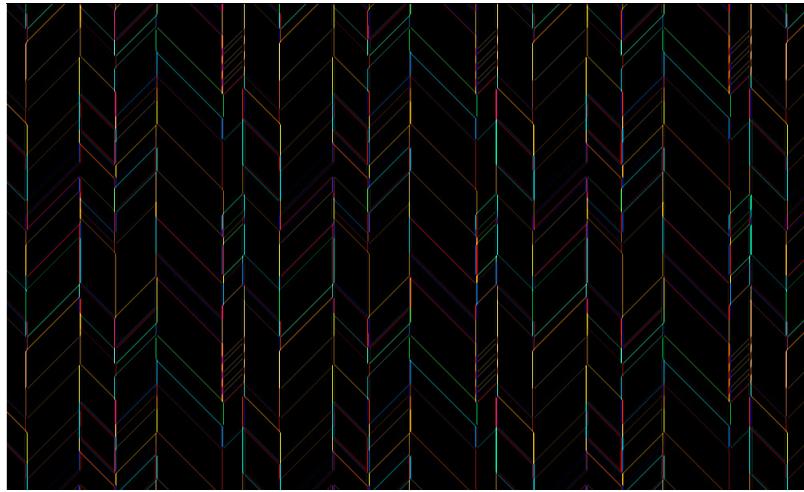


Figure 15: Horizontal mask

- Masque vertical : $\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$

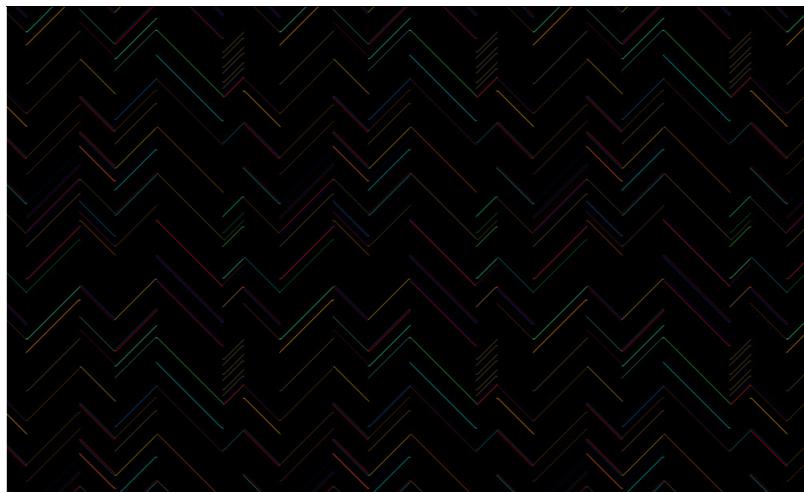


Figure 16: Vertical mask

Ce noyau permet de retrouver les lignes verticales.

- Masque Oblique 1 : $\begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$



Figure 17: Oblique mask

Ce noyau permet de retrouver les lignes obliques dans le sens ↗ .

- Masque Oblique 2 : $\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$

Ce noyau permet de retrouver les lignes obliques dans le sens ↘ .

On remarque que plusieurs noyaux se ressemblent et ressortent les mêmes choses dans l'image mais de manière différentes. L'implémentation de tout ces noyaux de détection de contours permet de visualiser la différence entre chaque méthode. Certain noyau sont trop petit afin de bien sélectionner les lignes recherchées.

1.5.4 Histogramme

L'histogramme d'une image représente la distribution des intensités de l'image. Il est possible d'analyser celui-ci afin de pouvoir ensuite le traiter. Le filtrage de l'histogramme permet de récupérer les pixels proches.

L'histogramme a été ajouté aux arguments. L'histogramme de l'image si elle est grise est un tableau de 255 éléments, 1 élément peut contenir un maximum du nombre de pixel de l'image. Il existe 3 histogrammes pour chaque composantes si l'image est en RGB. Ce n'est pas la meilleure option, il aurait fallut créer un

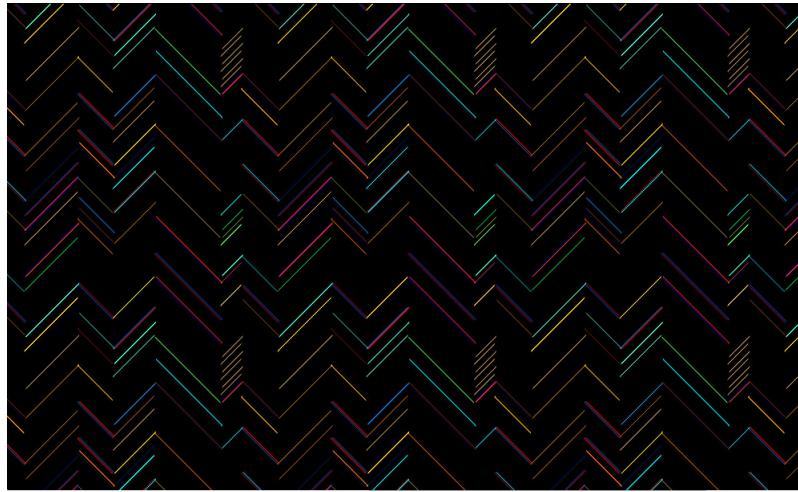


Figure 18: Oblique mask

histogramme de toutes les possibilités (ex: 0 R; 0 B; 1 G ...). Afin de visualiser l'histogramme, la fonction `printhistogramme` crée une `Image2D` de 255 x 100, puis on ramène le nombre de pixels à un pourcentage mais elle ne fonctionne pas bien.

1.5.5 Filtrage Automatique

Le but du filtrage automatique étant de filtrer l'image en fonction d'une couleur et de la valeur la plus récurrente. La fonction prend donc en argument la couleur souhaité et la variance autour de la valeur. On repère ensuite là où il y a le plus de pixel coloré dans l'histogramme et on garde seulement les valeurs avec la variance autorisées. Cette méthode n'est pas optimale, il faudrait utiliser la détection de la variance automatiquement et le faire sur l'histogramme complet et non sur chaque composantes.

1.5.6 Discussion

La classe `Image2D` créée à partir de rien possède plusieurs erreurs et est incomplète. Cependant, le workflow était très instructif et donne envie de continuer la manipulation d'images. Afin de visualiser les résultats que peut donner la classe `Image2D`, une application graphique a été créé.

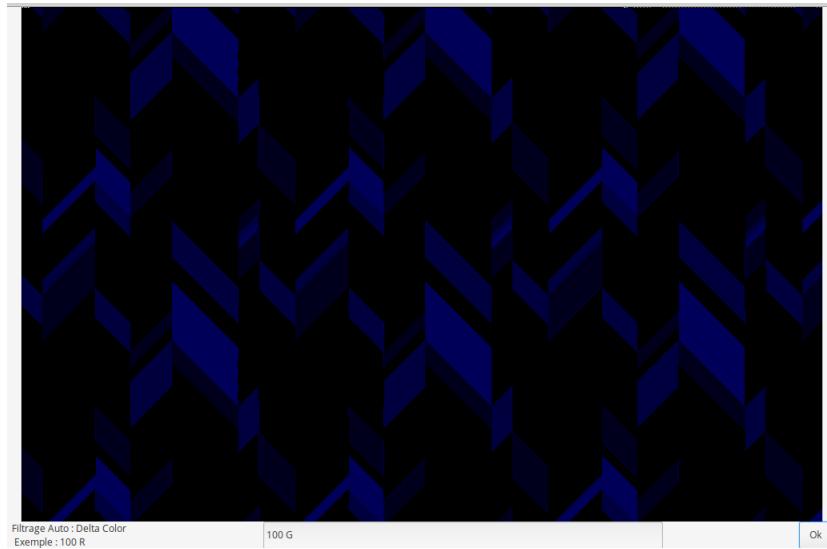


Figure 19: Filtrage Auto

2 Application gtkmm

Utilisant la librairie Gtkmm C++ ainsi que la classe Image2D, l'application Gtkmm du même mot a été créé pour faciliter la manipulation de la classe Image2D. Un menu comportant plusieurs sous-menu laisse l'utilisateur choisir la transformation voulue. Une fois que la transformation est choisie, il suffit de rentrer le paramètre nécessaire tel que le seuil et de cliquer sur Ok pour lancer le traitement. Sur Linux (Ubuntu kernel)

```
$ sudo apt install g++
```

Pour lancer la compilation dans le dossier contenant main.cpp, (-g pour debug) :

```
$ g++ -g Fenetre.cpp main.cpp Image2D.cpp -o Image2Dapp `pkg-config  
gtkmm-3.0 --cflags --libs`
```

Pour lancer l'application compilée :

```
$ ./Image2Dapp
```

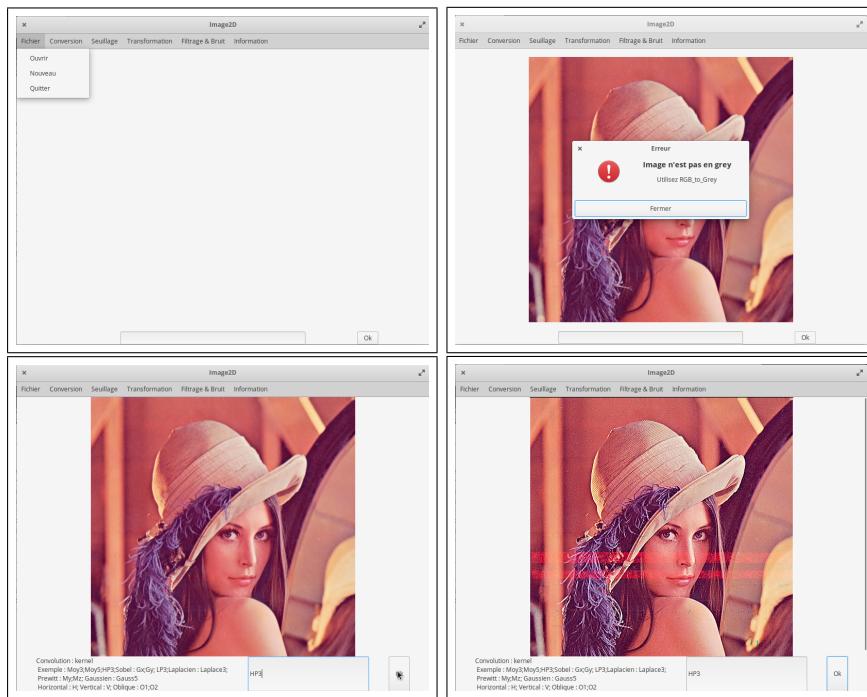


Figure 20: a - Fenetre intiale; b - Erreur de choix de la fonction; c - Fenetre choix d'une fonction; d - Filtre HP3 appliqué