Université de Lille
MASTER 1 INFORMATIQUE
Deuxième semestre

Traitement d'image

TP n°11

BARCHID Sami

CARTON Floriane

Année académique 2018-2019

Introduction

La matière vue dans ce TP traite de la formation des images couleur grâce aux méthodes de dématriçage à partir d'une image CFA. Le but, dans ce TP, est d'expérimenter et de comprendre les notions d'image CFA, de dématriçage ainsi que d'expérimenter différentes méthodes de dématriçage.

Les manipulations se feront grâce au logiciel « ImageJ » avec la programmation de plugins java. L'image couleur d'une scène suivante est fournie afin de mener nos expériences. Cette image sera nommée « phare » pour la suite de ce rapport.

Phare

Ce rapport de TP est divisé en 3 parties,

- **Image CFA :** mise en pratique et interprétations de la génération d'image CFA à partir d'une scène couleur fournie.

- **Dématriçage par interpolation bilinéaire :** expérimentations et interprétations sur la méthode de dématriçage par interpolation bilinéaire.

- **Dématriçage basé sur l'estimation locale d'un gradient :** expérimentations et interprétations sur la méthode de dématri çage basée sur l'estimation locale d'un gradient et comparaison avec la méthode par interpolation bilinéaire décrite à la partie précédente.

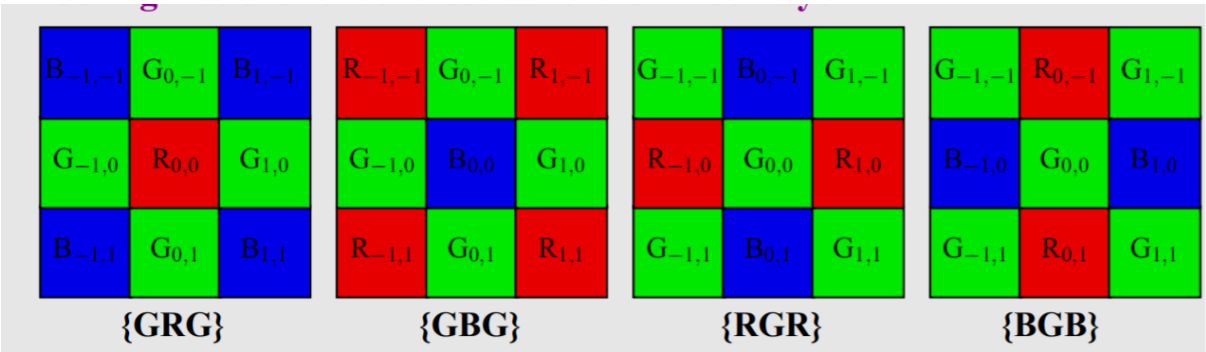
Image CFA

Le but de cette partie du TP est de comprendre la notion d'image CFA et de mettre en œuvre la construction d'une image CFA à partir d'une image couleur d'origine.

Pour la suite de ce TP, le CFA utilisé pour construire les images CFA sera le plus répandu, à savoir le CFA de Bayer, qui contient deux fois plus de filtres vert que de filtres rouge et bleu. Cela s'explique par le fait que le CFA de Bayer est s'adapte très bien à la sensibilité spectrale de l'oeil humain.

En effet, la sensibilité spectrale de l'oeil humain en vision photopique est maximale autour de $\lambda = 555 nm$ (longueur d'onde qui donne une couleur verte).

Dans un filtre CFA de Bayer, il existe 4 configurations différentes des pixels dans un voisinage 3x3 de pixels :



Interprétation d'une image CFA

Soit l'image suivante, une image CFA de la scène « phare » :

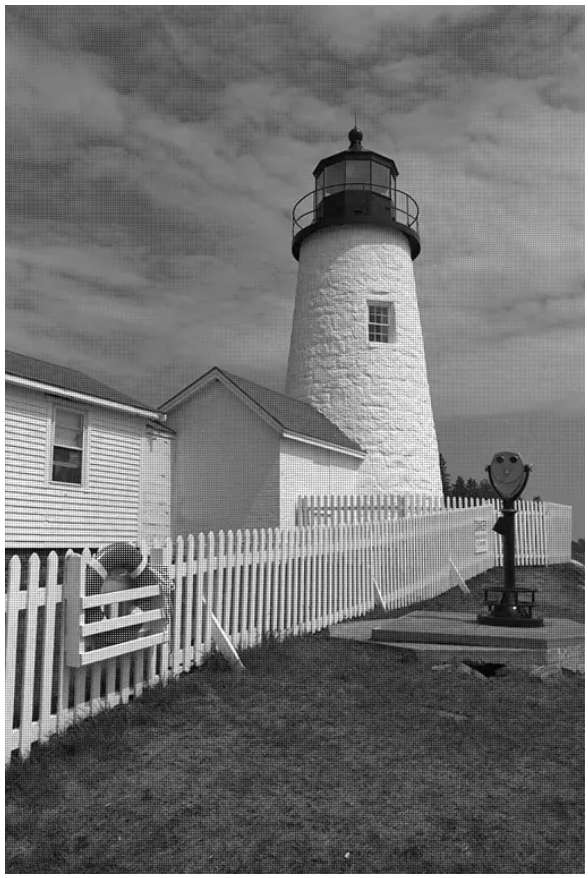


Image CFA de phare

Le but est de comprendre la composition du filtre CFA utilisé sur l'image de « phare » pour générer cette image CFA. Pour ce faire, nous allons analyser le premier voisinage 3x3 des pixels sur l'image CFA et sur l'image « phare » de référence (donc, les pixels de coordonnées (x,y)=({0,1,2},{0,1,2})) et essayer de trouver quelle est la configuration trouvée parmi celles décrites au-dessus.

	0	1	2
0	76,89,97	77,90,98	79,92,101
1	78,91,99	78,91,99	78,91,100
2	82,95,103	80,93,101	78,91,100

Image couleur de référence (format par pixel : R,V,B)

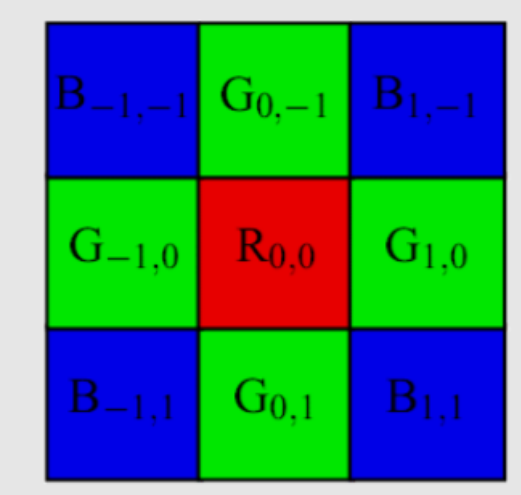
s	0	1	2
0	98	91	104
1	86	77	89
2	105	96	105

Image CFA

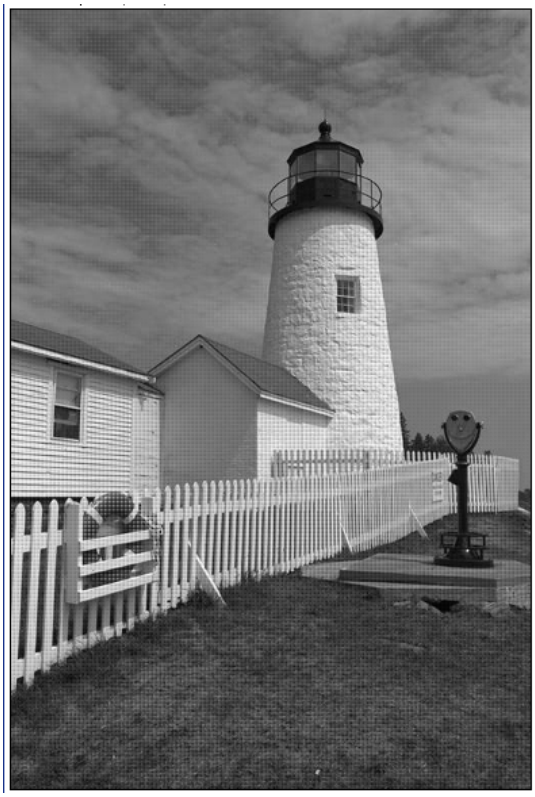
Il ne reste plus qu'à regarder, dans l'image couleur, quelle est la composante qui possède une valeur égale à la valeur trouvée dans l'image CFA à la même coorodonnée. Nous retrouverons alors la configuration du filtre CFA en ce voisinage.

Note : la valeur n'est pas 100 % égale entre l'image couleur et l'image CFA dû à de légères erreurs d'estimation.

La configuration du filtre trouvée ici est la configuration {GRG} (ou B-G-B) suivante :



Les images CFA trouvées pour chaque configuration sont les suivantes :



R-G-R

B-G-B

G-R-G

G-B-G

Note importante : Dans ce TP, les calculs de dématriçage qui seront présentés dans les parties suivantes seront opérés avec l'image CFA de Bayer obtenue grâce à la configuration **G-R-G**.

Ainsi, pour s'assurer de la justesse des résultats obtenus, nous prenons et comparons l'image CFA à la configuration G-R-G. Comme pour la vérification avec l'image CFA du phare fournie dans l'énoncé, il suffit de comparer les valeurs des niveaux de gris de l'image CFA de Bayer G-R-G avec les niveaux RGB de l'image « phare » dans le premier voisinage 3x3. Nous obtenons les analyses suivantes :

	0	1	2
0	76,89,97	77,90,98	79,92,101
1	78,91,99	78,91,99	78,91,100
2	82,95,103	80,93,101	78,91,100

Image couleur de référence (format par pixel : R,V,B)

s	0	1	2
0	89	77	92
1	99	91	100
2	95	80	91

Image CFA de Bayer (configuration G-R-G)

Nous obtenons donc bel et bien ici, sur la première et dernière ligne, des niveaux de gris équivalents aux couleurs rouge – verte – rouge, nous sommes donc bien dans la configuration **G-R-G**.

Dématriçage par interpolation linéaire

Le but, dans cette partie du TP, est de comprendre et expérimenter l'algorithme de dématriçage par interpolation linéaire grâce à l'image CFA de Bayer (G-R-G).

Ce dématriçage est réalisable par convolution en considérant les plans $\varphi_R, \varphi_G et\, \varphi_B$ respectivement définis par les seuls niveaux où la composante R, G ou B est disponible dans l'image CFA (les autres niveaux étant fixés à 0).

Nous devons donc élaborer un plugin ImageJ qui prend en entrée l'image CFA G-R-G du phare et qui effectue les opérations suivantes :

- Générer les plans $\varphi_R, \varphi_G et\, \varphi_B$, obtenus en ne conservant que les valeurs de rouge, de vert ou de bleu de l'image CFA en entrée de l'algorithme

- Filtrer les plans par le masque de convolution adéquat. C'est ce masque de convolution suivant le plan qui va effectuer l'interpolation linéaire.

- Combiner les trois plans filtrés en une image couleur unique, qui sera l'image obtenue grâce à l'opération de dématriçage.

Génération des plans

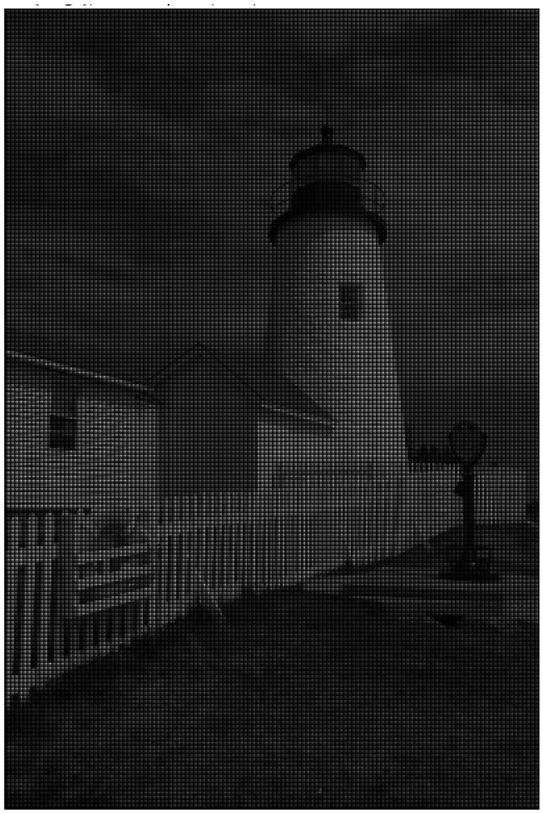
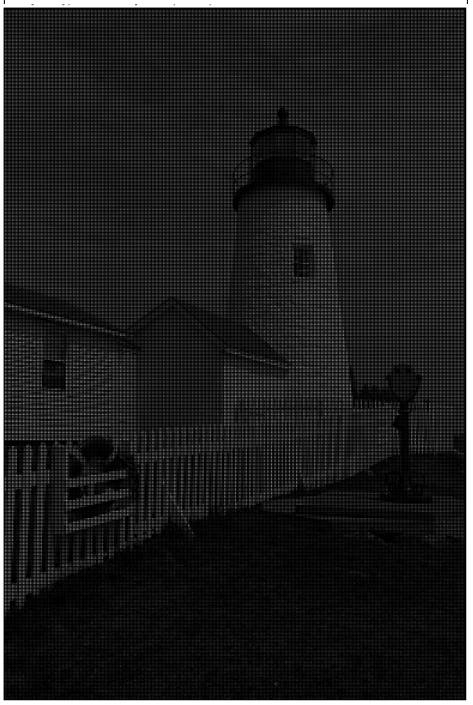
La génération des plans $\varphi_R, \varphi_G et \varphi_B$ est implémentée en Java dans le plugin ImageJ :

```
/**
 * Retourne, à partir de l'image CFA en paramètre "cfa_ip", l'image en niveaux
 * de gris sur 8 bits correspondant au plan ϕrouge, ϕvert ou ϕbleu suivant la
 * valeur de k
 *
 * @param cfa_ip l'image CFA utilisée pour le calcul
 * @param k      un entier représentant la composante (rouge, vert ou bleu)
 *               utilisée pour calculer le bon ϕk
 * @return l'image ϕk
 */
public ImageProcessor cfa_samples(ImageProcessor cfa_ip, int k) {
    width = cfa_ip.getWidth();
    height = cfa_ip.getHeight();

    // Création de l'image à retourner
    ImageProcessor phi = new ByteProcessor(width, height);

    // SI ϕR
    if (k == 0) {
        // Ajout des niveaux de rouge
        for (int x = 1; x < width; x = x + 2) {
            for (int y = 0; y < height; y = y + 2) {
                phi.putPixel(x, y, cfa_ip.getPixel(x, y));
            }
        }
    }
    // SI ϕV
    else if (k == 1) {
        // Ajout des niveaux de rouge
        for (int x = 0; x < width; x = x + 2) {
            for (int y = 0; y < height; y = y + 2) {

                phi.putPixel(x, y, cfa_ip.getPixel(x, y));
            }
        }

        for (int x = 1; x < width; x = x + 2) {
            for (int y = 1; y < height; y = y + 2) {
                phi.putPixel(x, y, cfa_ip.getPixel(x, y));
            }
        }
    }
    // SI ϕB
```

```
        // SI ϕB
        else {
            for (int x = 0; x < width; x = x + 2) {
                for (int y = 1; y < height; y = y + 2) {
                    phi.putPixel(x, y, cfa_ip.getPixel(x, y));
                }
            }
        }

        return phi;
    }
```

L'algorithme implémenté se contente de générer une image du plan recherché en récupérant les niveaux de gris de l'image CFA qui correspondent à la composante rouge vert ou bleu (suivant le paramètre k).

Les images des plans ainsi générées sont les suivantes :

φ_R	φ_G	φ_B
		

On remarque que les images des plans montrent des résultats logiques selon plusieurs observations triviales :

- Dans le plan φ_R , la bouée (qui est complètement rouge dans l'image de référence) est quasiment blanche ici, montrant un haut niveau de rouge.

- Le plan φ_G est le plus clair, dû au fait que la sensibilité spectrale de l'oeil humain en vision photopique est plus grande dans les longueurs d'onde du vert

- Et de même, le plan φ_B paraît être le plus foncé car la sensibilité spectrale de l'oeil humain est moindre dans les longueurs d'onde proche du bleu.

Filtre par convolution

Le masque de convolution utilisé pour filtrer chaque plan est différent selon le plan utilisé.

Le résultat de ce filtre par convolution et la combinaison des plans en une image couleur sont implémentés par la macro ImageJ suivante :

```
// Appeler le plugin "SampleCfa"
run("SampleCfa ");


// Récupérer composante rouge
setSlice(1);
run("Convolve...", "text1=[0.25 0.5 0.25\n0.5 1 0.5\n0.25 0.5 0.25\n] slice");

// Récupérer composante vert
setSlice(2);
run("Convolve...", "text1=[0 0.25 0\n0.25 1 0.25\n0 0.25 0\n] slice");

// Récupérer composante bleu
setSlice(3);
run("Convolve...", "text1=[0.25 0.5 0.25\n0.5 1 0.5\n0.25 0.5 0.25\n] slice");

// Combiner les trois slices en une image couleur
run("Stack to RGB");
```

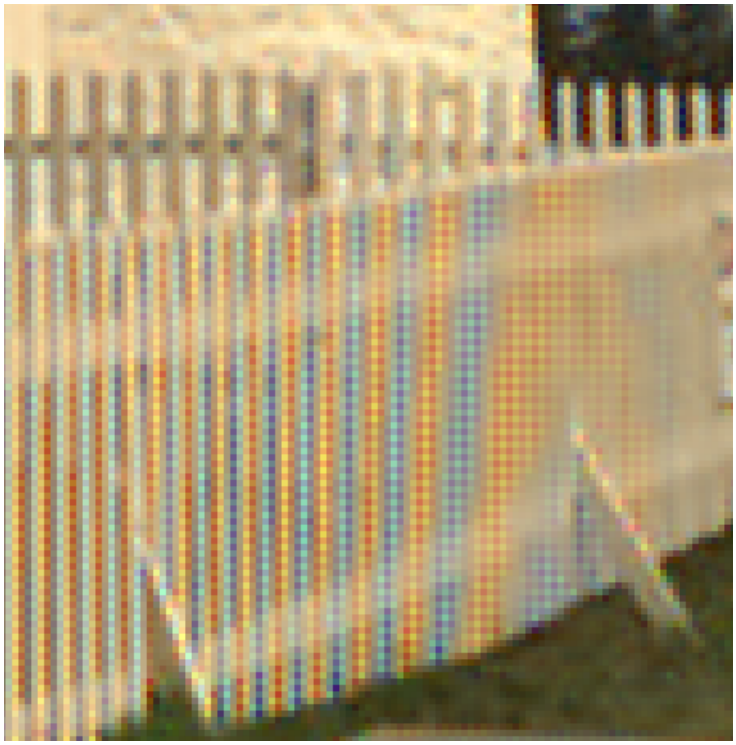
L'exécution de cette macro fournit, alors, le résultat suivant :

Image couleur par dématriçage par interpolation bilinéaire

Nous pouvons remarquer plusieurs choses en analysant l'image obtenue :

- Nous remarquons l'apparition de couleurs aberrantes dans les zones de fortes variations de couleurs. En effet, lorsque, dans un même voisinage 3x3 de pixels appartiennent à deux zones homogènes distinctes, nous obtenons des couleurs fausses. Ceci est dû au fait que, comme nous calculons la valeur des pixels selon une simple interpolation sans s'assurer que nous restons dans la même zone homogène, nous obtenons des couleurs correspondantes à la « moyenne » entre deux zones homogènes distinctes. Cet effet est facilement remarquable sur des interpolations au travers d'un contour.
  Un exemple de ces couleurs aberrantes peut être vu dans les

clotures du phare, où il y a une grande alternance entre des zones correspondantes aux clotures blanches et des zones correspondantes à la pelouse (foncée).



Exemple de couleur aberrante dans l'image

- La disposition en quinconce des niveaux de gris dans l'image CFA originale crée un crénelage au niveau des contours verticaux et horizontaux. Cela donne un effet de contour beaucoup moins nets. Cet effet est appelé le zipper effect. Un exemple de cet « effet fermeture éclaire » peut être vu au niveau du phare :

Exemple de zipper effect

On peut conclure sur le fait que, en générale, le résultat est dans l'ensemble du dématriçage par interpolation linéaire fournit un résultat compréhensible et relativement proche de l'image de référence mais contient des défauts non négligeables pour l'utilisateur final.

Dématriçage basé sur l'estimation locale d'un gradient

Le but de cette partie du TP est de comprendre et expérimenter l'algorithme de dématriçage par l'estimation d'un gradient proposé par Hamilton et Adams (HA97).

Pour ce faire, nous mettrons au point un plugin ImageJ implémentant cet algorithme de dématriçage en réalisant uniquement l'estimation du plan φ_G .

Ce plugin prend en entrée l'image CFA de Bayer (G-R-G) et effectue les opérations suivantes :

- Estimer $\varphi_R et \varphi_B$ par interpolation bilinéaire comme vue à la partie précédente.
- Estimer φ_G par la méthode de Hamiltion & Adams.

Nous réaliserons alors, grâce au langage macro d'ImageJ, la combinaison des trois plans comme vu à la partie précédente.

Estimation de φ_G par la méthode de Hamilton & Adams

Il s'agit d'implémenter, dans notre plugin, les formules suivantes basées sur les calculs des gradients horizontaux et verticaux :

❶ Calcul des gradients horizontal Δ^x et vertical Δ^y :

$$\Delta^x = |G_{-1,0} - G_{1,0}| + |2R - R_{-2,0} - R_{2,0}| \ ,$$
$$\Delta^y = |G_{0,-1} - G_{0,1}| + |2R - R_{0,-2} - R_{0,2}| \ .$$

❷ Interpolation du niveau de vert :

$$\hat{G} = \begin{cases} \left(G_{-1,0} + G_{1,0}\right)/2 + \left(2R - R_{-2,0} - R_{2,0}\right)/4 & \text{si } \Delta^x < \Delta^y, \\ \left(G_{0,-1} + G_{0,1}\right)/2 + \left(2R - R_{0,-2} - R_{0,2}\right)/4 & \text{si } \Delta^x > \Delta^y, \\ \left(G_{0,-1} + G_{-1,0} + G_{1,0} + G_{0,1}\right)/4 & \\ \quad + \left(4R - R_{0,-2} - R_{-2,0} - R_{2,0} - R_{0,2}\right)/8 & \text{si } \Delta^x = \Delta^y. \end{cases}$$

Le code du plugin implémentant cette formule est le suivant :

```
ImageProcessor est_G_hamilton(ImageProcessor cfa_ip) {
    width = cfa_ip.getWidth();
    height = cfa_ip.getHeight();
        ImageProcessor est_ip = cfa_ip.duplicate();
        for(int j=0;j<height;j=j+2){
            for(int i=1;i<width;i=i+2){

                int pCourant = cfa_ip.getPixel(i,j) & 0xff;

                int pXGauche = cfa_ip.getPixel(i-1,j) & 0xff;
                int pXDroite = cfa_ip.getPixel(i+1,j) & 0xff;
                int pXGaucheG = cfa_ip.getPixel(i-2,j) & 0xff;
                int pXDroiteD = cfa_ip.getPixel(i+2,j) & 0xff;

                int pYHaut = cfa_ip.getPixel(i,j-1) & 0xff;
                int pYBas = cfa_ip.getPixel(i,j+1) & 0xff;
                int pYHautH = cfa_ip.getPixel(i,j-2) & 0xff;
                int pYBasB = cfa_ip.getPixel(i,j+2) & 0xff;

                int gradX= Math.abs(pXGauche-pXDroite)+Math.abs(2*pCourant-pXGaucheG-pXDroiteD);
                int gradY= Math.abs(pYHaut-pYBas)+Math.abs(2*pCourant-pYHautH-pYBasB);

                // SI Δx < Δy
                if(gradX<gradY){

                    est_ip.putPixel(i,j,((pXGauche+pXDroite)/2+(2*pCourant-pXGaucheG-pXDroiteD)/4));
                }
                // SI Δx > Δy
                else if(gradX>gradY){
                    est_ip.putPixel(i,j,((pYHaut+pYBas)/2+(2*pCourant-pYHautH-pYBasB)/4));
                }
                // SI Δx = Δy
                else{
                    est_ip.putPixel(i,j,((pYHaut+pXGauche+pXDroite+pYBas)/4+(4*pCourant-pYHautH-pXGaucheG-pXDroiteD-pYBasB)/8));
                }
            }

        for(int j=1;j<height;j=j+2){
            for(int i=0;i<width;i=i+2){

                int pCourant = cfa_ip.getPixel(i,j) & 0xff;

                int pXGauche = cfa_ip.getPixel(i-1,j) & 0xff;
                int pXDroite = cfa_ip.getPixel(i+1,j) & 0xff;
                int pXGaucheG = cfa_ip.getPixel(i-2,j) & 0xff;
                int pXDroiteD = cfa_ip.getPixel(i+2,j) & 0xff;

                int pYHaut = cfa_ip.getPixel(i,j-1) & 0xff;
                int pYBas = cfa_ip.getPixel(i,j+1) & 0xff;
                int pYHautH = cfa_ip.getPixel(i,j-2) & 0xff;
                int pYBasB = cfa_ip.getPixel(i,j+2) & 0xff;

                int gradX= Math.abs(pXGauche-pXDroite)+Math.abs(2*pCourant-pXGaucheG-pXDroiteD);
                int gradY= Math.abs(pYHaut-pYBas)+Math.abs(2*pCourant-pYHautH-pYBasB);
                if(gradX<gradY){

                    est_ip.putPixel(i,j,((pXGauche+pXDroite)/2+(2*pCourant-pXGaucheG-pXDroiteD)/4));
                }
                else if(gradX>gradY){
                    est_ip.putPixel(i,j,((pYHaut+pYBas)/2+(2*pCourant-pYHautH-pYBasB)/4));
                }
                else{
                    est_ip.putPixel(i,j,((pYHaut+pXGauche+pXDroite+pYBas)/4+(4*pCourant-pYHautH-pXGaucheG-pXDroiteD-pYBasB)/8));
                }
            }
        }

        return (est_ip);
}
```

Après exécution du plugin et combinaison des images en une image couleur, nous obtenons le résultat suivant :

Résultat par dématriçage basé sur HA97

Nous pouvons faire plusieurs observations :

- Nous retrouvons, comme pour l'image obtenue via dématriçage par interpolation linéaire, des couleurs aberrantes, pour la raison que,

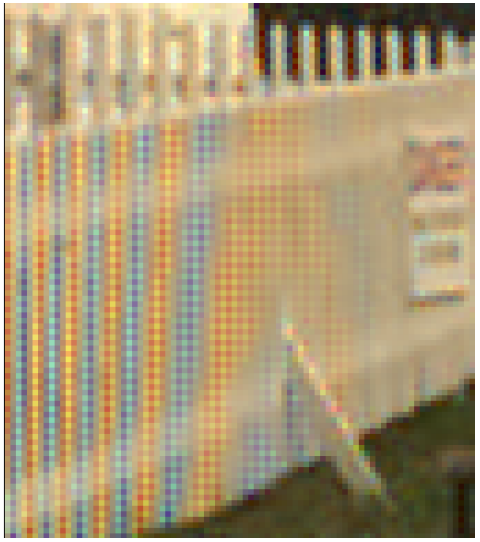
dans des zones de détails fins comme vers les bordures, il y a des mauvais choix dans la direction d'interpolation, ce qui peut donner des mauvaises couleurs.

Nous avons ensuite recherché à comparer les deux images obtenues par dématriçage avec l'image de référence :

Image	Forte variation de zones homogènes (zoom clôtures)	Contours du phare
 Référence		

HA97

Interpolation bilinéaire



Nous remarquons ici plusieurs choses :

- Des couleurs aberrantes apparaissent pour HA97 comme pour l'interpolation linéaire, dû aux explications données auparavant.

- L'algorithme de HA97 permet, contrairement à l'interpolation linéaire, de limiter le « zipper effect » sur les contours horizontaux et verticaux. Ceci s'explique du fait que l'estimation des gradients verticaux et horizontaux permettent de prévoir ces problèmes d'interpolation aux contours.

- A la vue naturelle, l'image obtenue par HA97 reste meilleure que celle obtenue par interpolation linéaire dû au fait que le zipper effect est diminué.