Formation des images couleur et dématriçage

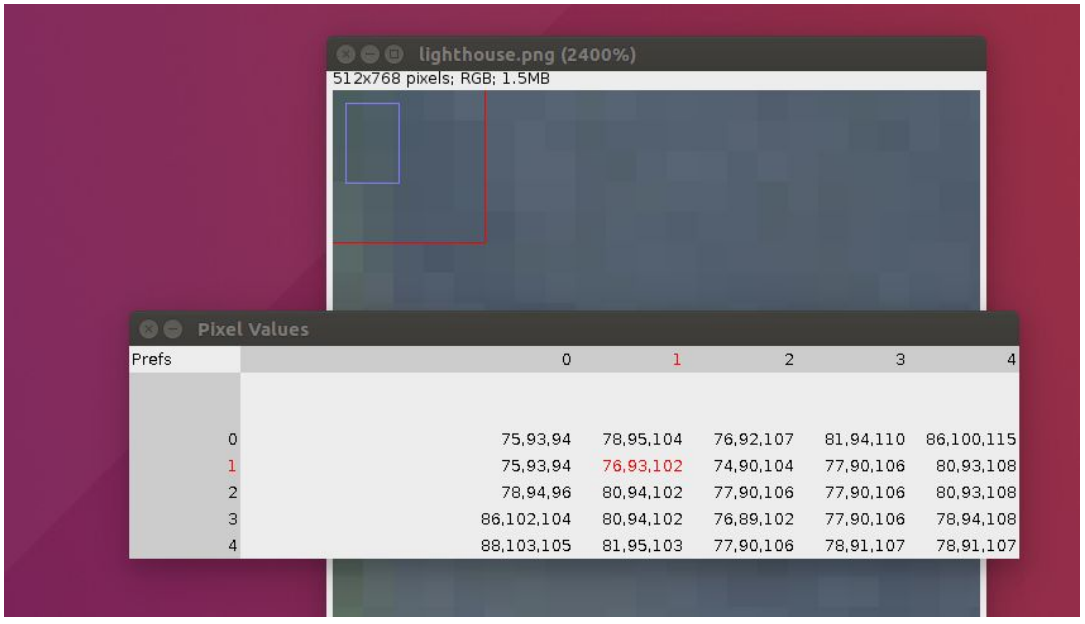
Kasperek Gautier
Sparrow Alice

1. Interprétation et simulation d'une image CFA

Interprétation d'une image CFA

Figure 1 - Image couleur lighthouse.png et son image CFA correspondante

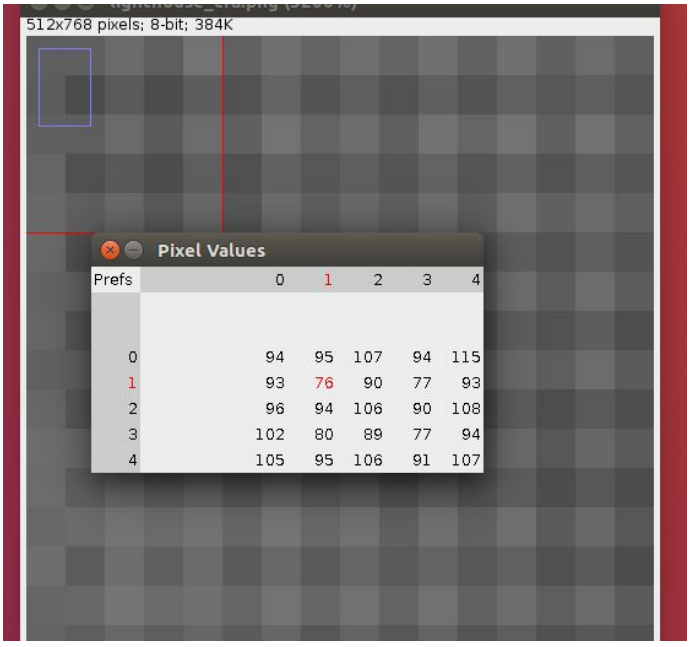
On remarque que chacun des pixels de l'image CFA (de la figure 1) ne contient qu'un seul niveau de composante couleur (soit R, soit G, soit B). La composante couleur de chaque pixel est définie par la disposition du filtre CFA. Nous allons uniquement étudier les CFA de Bayer qui comportent de filtres vert que de filtres rouge et bleu. Nous cherchons quel CFA de Bayer a été utilisé pour l'obtention de l'image CFA de la figure 1.



Prefs	0	1	2	3	4
0	75,93,94	78,95,104	76,92,107	81,94,110	86,100,115
1	75,93,94	76,93,102	74,90,104	77,90,106	80,93,108
2	78,94,96	80,94,102	77,90,106	77,90,106	80,93,108
3	86,102,104	80,94,102	76,89,102	77,90,106	78,94,108
4	88,103,105	81,95,103	77,90,106	78,91,107	78,91,107

Figure 2 - Coin haut gauche de l'image lighthouse.png

Lorsqu'on inspecte les pixels du coin en haut à gauche de l'image couleur, on remarque qu'ils ont une composante bleue importante. On prend par exemple le pixel à la position (1,1) qui a les composantes rgb suivantes : (76,93,102).



Prefs	0	1	2	3	4
0	94	95	107	94	115
1	93	76	90	77	93
2	96	94	106	90	108
3	102	80	89	77	94
4	105	95	106	91	107

Figure 3 - coin haut gauche de l'image CFA

La figure 3 ci-dessus met en évidence les valeurs des pixels du coin haut gauche de l'image CFA. On remarque que le pixel de coordonnées (1,1) que nous observions auparavant possède la valeur 76, c'est donc la composante rouge de ce pixel qui a été conservée. Les pixels au dessus, en dessous, à gauche et à droite de celui-ci ont vu conservé leur composante verte. Pour finir on constate que les diagonales du pixel positionné en (1,1) ont leur composante bleu sauvegardée. On peut conclure de ces observations que le filtre CFA utilisé est celui possédant la configuration {B,G,B}.

Figure 4 - CFA de Bayer de configuration {B,G,B}

Simulation d'une image CFA à partir d'une image couleur

```
public int setup(String arg, ImagePlus imp) {
       /*Définit la variable globale imp avec l'image courante*/
       this.imp = imp;
      // La fonction attend une image couleur
       return PlugInFilter.DOES_RGB;
}


public void run(ImageProcessor ip) {

      // Lecture des dimensions de la fenêtre
      width = imp.getWidth();
      height = imp.getHeight();

      // Dispositions possibles pour le CFA
      String[] orders = {"R-G-R", "B-G-B", "G-R-G", "G-B-G"};

      // Définition de l'interface
      GenericDialog dia = new GenericDialog("Génération de l'image
      CFA...", IJ.getInstance());
      dia.addChoice("Début de première ligne :", orders, orders[2]);
      dia.showDialog();

      // Lecture de la réponse de l'utilisateur
      if (dia.wasCanceled()) return;
      int order = dia.getNextChoiceIndex();

      // Génération de l'image CFA
      // Crée l'image cfa
      ImageProcessor im_cfa;
      // Appel à la fonction qui calcule l'image cfa
      im_cfa = cfa(1);
      // Création de l'image à afficher
      ImagePlus res_cfa = new ImagePlus("Resultat image cfa",im_cfa);
      // Affichage du resultat
      res_cfa.show();
}
```

Comme vous pouvez le voir dans le code ci-dessus, nous avons imposé l'utilisation d'une image couleur dans la méthode setup car sans cela le plugin ne fournit pas un résultat correct. Nous avons ensuite créé l'image cfa dans la fonction run par un appel à la méthode cfa. Nous affichons ensuite le résultat dans une nouvelle fenêtre.

Figure 5 - Image couleur et image CFA calculée par notre plugin

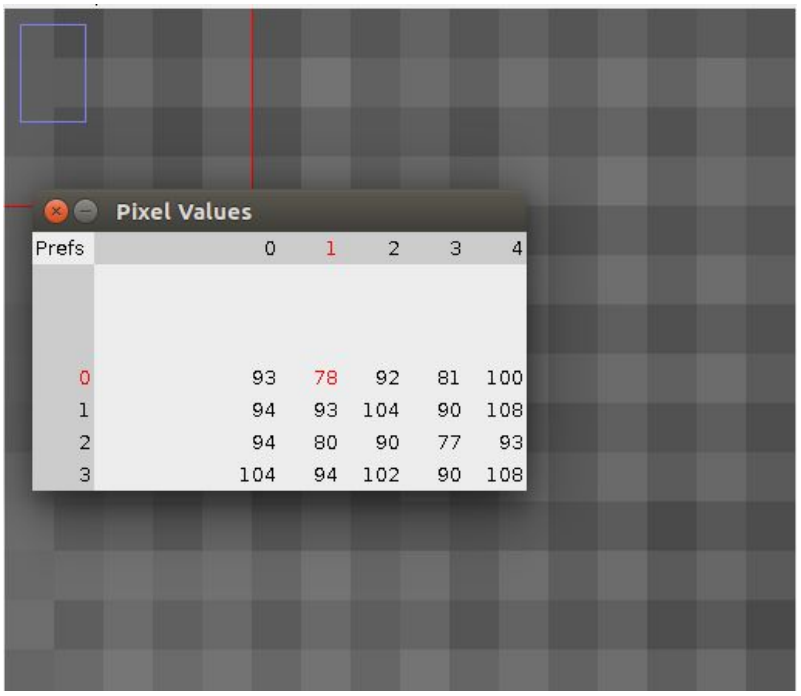


Figure 6 - Inspection des pixels de l'image cfa générée

On remarque ici que le pixel (1,1) a conservé sa composante verte, les pixels en diagonale ont fait de même. Les pixels au dessus et en dessous du pixel (1,1) ont sauvegardé leur composante rouge. Pour finir les pixels à droite et à gauche de la position (1,1) ont conservé leur composante bleue. On en conclut donc que c'est bien le CFA {G,R,G} qui a été appliqué à l'image couleur (voir figure 7).

Figure 7 - CFA {G,R,G}

2. Dématriçage par interpolation bilinéaire

Présentation

Afin de réaliser le dématriçage de l'image CFA nous allons utiliser des masques de convolutions permettant de calculer l'interpolation bilinéaire des valeurs des canaux rouges, verts et bleus. Pour réaliser le dématriçage nous allons utiliser la matrice suivante pour les plans rouge φR et bleu φB le masque suivant :
HB = HR = ¼ (1,2,1, 2,4,2 ,1,2,1)
Pour le plan vert φG nous utiliserons la matrice suivante :
HG = ¼ (0,1,0 ,1,4,1 0,1,0)

Pour réaliser l'interpolation bilinéaire de la valeur d'un pixel nous cherchons à calculer une moyenne des valeurs de son voisinage. Par exemple, pour une configuration {GRG}, nous calculons la valeur approché du canal bleu pour le pixel de coordonnées relatives (0,0).

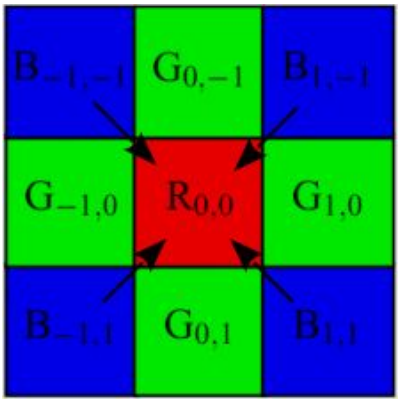


Figure 8 -

Nous choisissons de réaliser une moyenne des valeurs de bleu dans un voisinage 3*3 :

$$B(0,0) = \frac{1}{4}(B(-1,-1) + B(1,-1) + B(-1,1) + B(1,1)$$

Nous allons montrer que le calcul des différents plans φ revient à réaliser l'interpolation bilinéaire des valeurs des plans.

Tout d'abords nous divisons l'image CFA en plan incomplet comportant uniquement les valeurs fourni dans le CFA.

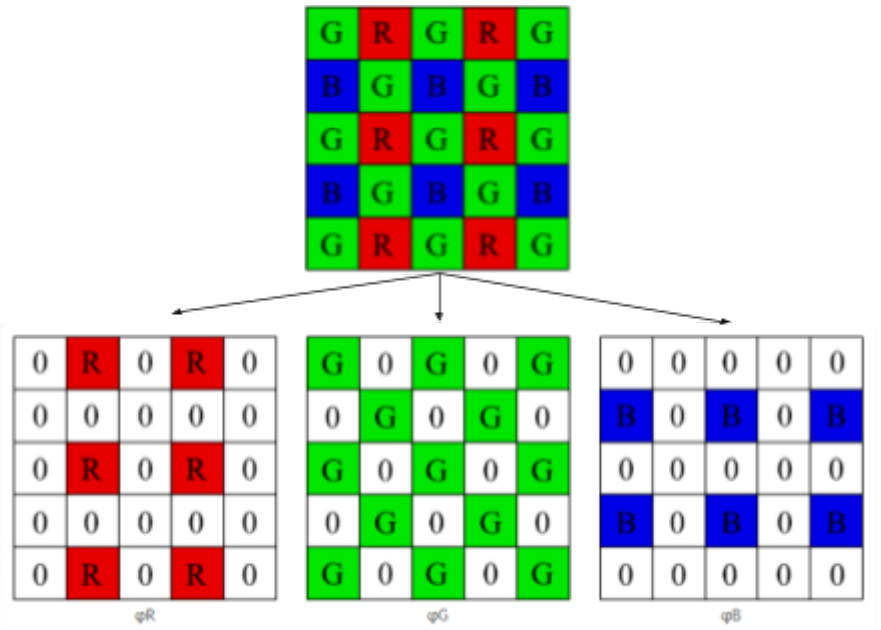


Figure 9 -

Dorénavant, l'objectif est de calculer les nouvelles valeurs de chaque valeur nulle. Considérons le pixel encadré ci-dessous :

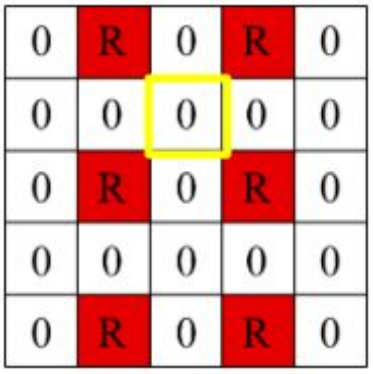


Figure 10a - Plan rouge avant dématriçage

D'une part nous calculons la valeur du pixel de coordonnées relative (0,0) encadré par interpolation bilinéaire, nous obtenons :

$$R(0,0) = \frac{1}{4}(R(-1,-1) + R(1,-1) + R(-1,1) + R(1,1))$$

D'autre part, nous appliquons le masque de convolution HR au voisinage 3*3 du pixel courant :

$$R(0,0) = \frac{1}{4}((1*R,\ 2*0,\ 1*R)\,(2*0,\ 4*0,\ 2*0)\,(1*R,\ 2*0,\ 1*R))$$
$$\Leftrightarrow R(0,0) = \frac{1}{4}(R + R + R + R)$$

Nous remarquons que nous retombons sur le calcul et la méthode de l'interpolation linéaire.

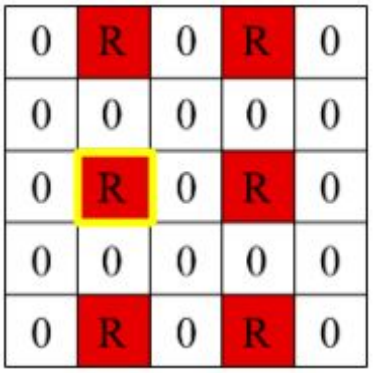


Figure 10b - Plan rouge avant dématriçage

Ce calcul est vérifiable pour tout autre pixel choisi dans cette configuration, par exemple si le pixel courant est un pixel contenant une valeur non nulle. On applique la matrice à l'entourage du pixel :

$$R(0,0) = \frac{1}{4}((1*,\ 2*0,\ 1*)\,(2*0,\ 4*R,\ 2*0)\,(1*0,\ 2*0,\ 1*0))$$
$$\Leftrightarrow R(0,0) = \frac{1}{4}(4R) = R$$

Nous remarquons alors que ceci revient à garder la valeur du pixel courant, comme lorsqu'il n'est pas nécessaire de réaliser une interpolation bilinéaire. Ces calculs sont vérifiables pour toutes les configurations du plan rouge ainsi que celle du plan bleu.

Nous effectuons un calcul similaire avec l'interpolation bilinéaire et l'application de la matrice HG dans le plan vert pour un pixel encadré en jaune ci-dessous de valeur nulle.

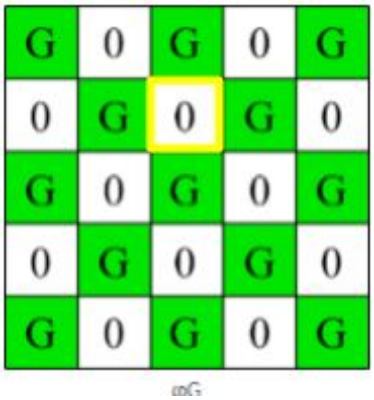


Figure 11a - Plan vert avant dématriçage

D'une part, nous effectuons une interpolation bilinéaire sur les valeurs adjacentes horizontalement et verticalement au pixel courant de coordonnées relatives (0,0) soit :

$$\hat{G}(0,0) = \frac{1}{4}(G(-1,0) + G(1,0) + G(0,1) + G(0,-1))$$

D’autre part, nous appliquons le masque HG = ¼ (0,1,0 ,1,4,1 0,1,0) :

$$G(0,0) = \frac{1}{4}((0*0,\ 1*G(0,-1),0*0)(1*G(-1,0),\ 4*0,\ *\ G(1,0))(0*0,\ 1*G(0,-1),\ 0*0))$$
$$\Leftrightarrow \hat{G}(0,0) = \frac{1}{4}(G(-1,0) + G(1,0) + G(0,1) + G(0,-1))$$

Nous remarquons que les deux façons de calculer sont équivalentes pour cette configuration dans le cas du plan vert. De la même manière pour le pixel courant (0,0) dans la configuration ci-dessous, la matrice HG possède des coefficients nuls dans ses coin qui permet d’annuler les valeurs des verts voisins, et les valeurs zéros du plan annule les coefficients à 1 de la matrice. Enfin, la matrice assigne donc au pixel courant ¼ * 4*G(0,0) soit le pixel courant.

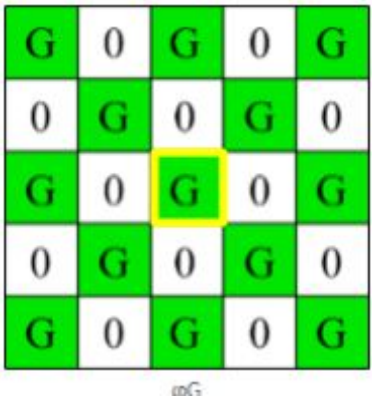


Figure 11b - Plan vert avant dématriçage

En appliquant les masques de convolutions sur les différents plans, nous pouvons réaliser un dématriçage à la manière d’une interpolation bilinéaire des valeurs.

Programmation du dématriçage

Dans la partie précédente nous avons prouvé que l’application des filtres HG, HB et HR permettaient le dématriçage respectif des plans vert, bleu et rouge à la manière d’un dématriçage par interpolation bilinéaire. Dans cette partie nous implémentons un plugin et une macro ImageJ réalisant ce dématriçage permettant ensuite d’obtenir une image couleur à partir du cfa d’une image.

```
import ij.*;
```

```
import ij.plugin.filter.*;
import ij.process.*;
import ij.gui.*;
import ij.plugin.filter.Convolver;

public class sample_cfa implements PlugInFilter {
```

```
      ImagePlus imp;    // Fenêtre contenant l'image de référence
      int width;         // Largeur de la fenêtre
      int height;

public int setup(String arg, ImagePlus imp) {
      /*Définit la variable globale imp avec l'image courante*/
      this.imp = imp;
      // La fonction attend une image 8 bits
      return PlugInFilter.DOES_8G;
      }

public void run(ImageProcessor ip) {



    // Calcul des échantillons de chaque composante de l'image CFA
    ImageStack samples_stack = imp.createEmptyStack();
    samples_stack.addSlice("rouge", cfa_samples(ip,0));  // Composante R
    samples_stack.addSlice("vert", cfa_samples(ip,1));   // Composante G
    samples_stack.addSlice("bleu", cfa_samples(ip,2));   // Composante B

    // Création de l'image résultat
    ImagePlus cfa_samples_imp = imp.createImagePlus();
    cfa_samples_imp.setStack("Échantillons couleur CFA", samples_stack);
    cfa_samples_imp.show();

}

public ImageProcessor cfa_samples(ImageProcessor cfa_ip, int k){
    width = imp.getWidth();
      height = imp.getHeight();

    ByteProcessor im_rouge_cfa = new ByteProcessor(width, height);
    ByteProcessor im_vert_cfa = new ByteProcessor(width, height);
    ByteProcessor im_bleu_cfa = new ByteProcessor(width, height);
    for(int i = 0; i < width; i++) {
             for(int j = 0; j < height; j ++) {
             //Cas des pixels verts
             if( ( (i%2 == 0) && (j%2 == 0) ) || ( (i%2 != 0) && (j%2 !=
              0) )){
                   im_vert_cfa.set(i,j,cfa_ip.getPixel(i,j));
             }
             //Cas des pixels rouges
             if( (i%2 != 0) && (j%2 == 0) ){
```

```
                    im_rouge_cfa.set(i,j,cfa_ip.getPixel(i,j));
                }
                //Cas des pixels bleus
                if( (i%2 == 0) && (j%2 != 0) ){
                    im_bleu_cfa.set(i,j,cfa_ip.getPixel(i,j));
                }
            }
        }
        if(k == 0){
            return im_rouge_cfa;
        }
        if(k == 1){
            return im_vert_cfa;
        }
        else{
            return im_bleu_cfa;
        }

    }

}
```

```
run("Compile and Run...",
"compile=/home/m1/sparrow/.imagej/plugins/sample_cfa.java");
selectWindow("Échantillons couleur CFA");
run("Convolve...", "text1=[0.25 0.5 0.25\n0.5 1 0.5\n0.25 0.5 0.25]
slice");
run("Next Slice [>]");
run("Convolve...", "text1=[0 0.25 0\n0.25 1 0.25\n0 0.25 0] slice");
run("Next Slice [>]");
run("Convolve...", "text1=[0.25 0.5 0.25\n0.5 1 0.5\n0.25 0.5 0.25]
slice");

run("Stack to RGB");
```

Voici le résultat que nous avons obtenu. On remarque sur la figure 12 que l'image du phare est globalement bien restaurée. On constate cependant des "fausses couleurs" sur la barrière blanche devant le phare.

Figure 12 - Dématriçage par interpolation bilinéaire

Figure 13 - Couleurs aberrantes

Nous observons des fausses couleurs sur les barrières devant le phare, l'estimation des niveaux de couleurs est incorrecte. Cela est principalement due à une interpolation au travers d'un contour, en effet on utilise les pixels voisins pour l'interpolation. Si les voisins n'appartiennent à la même zone que celui dont on calcule la valeur le résultat est faussé.

3. Dématriçage basé sur l'estimation locale d'un gradient

Dans cette partie nous allons utiliser l'algorithme de dématriçage proposé par Hamilton & Adams exposé en cours. Nous allons nous en servir pour estimer le plan vert, puis estimer les plans rouge et bleu par interpolation bilinéaire. Nous comparerons enfin le résultat obtenu avec celui fourni par l'interpolation bilinéaire pour les trois plans.

Implémentation de la méthode

Ce plugin accepte en entrée une image CFA en niveaux de gris sur 8 bits et, dans sa méthode run génère l'image estimée en stockant ses 3 plans dans une pile d'images 8 bits.

Nous obtenons le plan rouge et le plan bleu par interpolation bilinéaire à partir des échantillons de l'image CFA nous réutilisons le plugin réalisé plus haut et nous y ajoutons le calcul de la convolution.
Pour calculer le plan vert nous procédons de la manière suivante :

1. Nous calculons les gradients horizontal et vertical. Le résultat de ce calcul nous permet de savoir quelle formule utiliser pour l'estimation du plan vert
2. Nous interpolons la valeur du niveau de vert :
    - Si la valeur du gradient horizontal est inférieure à la valeur du gradient vertical

      $$\hat{G} = \frac{(G_{-1,0} + G_{1,0})}{2} + \frac{(2R - R_{-2,0} - R_{2,0})}{4}$$

      $G_{-1,0}$ est le niveau de vert du pixel situé à gauche du pixel que nous traitons. Cette notion de voisinage est visible sur la figure 13.

    - Si la valeur du gradient horizontal est supérieure à la valeur du gradient vertical

      $$\hat{G} = \frac{(G_{0,-1} + G_{0,1})}{2} + \frac{(2R - R_{0,-2} - R_{0,2})}{4}$$

    - Si la valeur du gradient horizontal est égale à la valeur du gradient vertical

      $$\hat{G} = \frac{(G_{0,-1} + G_{-1,0} + G_{1,0} + G_{0,1})}{4} + \frac{(4R - R_{0,-2} - R_{-2,0} - R_{2,0} - R_{0,2})}{8}$$

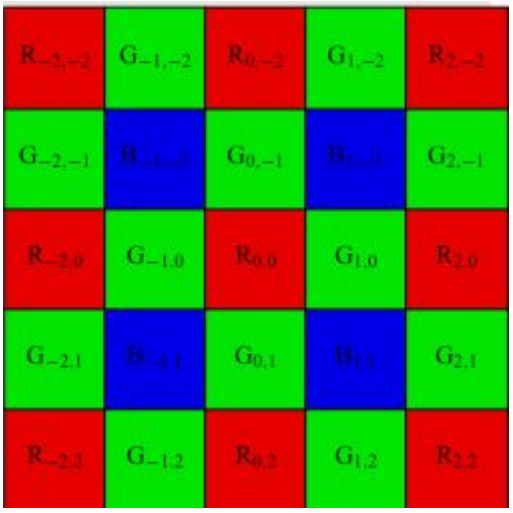


Figure 14 - Voisinage du pixel courant

```
public void run(ImageProcessor ip) {

    // Lecture des dimensions de la fenêtre
    width = imp.getWidth();
    height = imp.getHeight();

    // Déclaration d'un noyau et d'un objet Convolver pour la
    convolution
    float[] kernel = {1,2,1 , 2,4,2 , 1,2,1};
    for (int i=0;i<kernel.length;i++) {
        kernel[i]=kernel[i]/4;
    }
    ImageProcessor red = cfa_samples(ip,0);
    ImageProcessor blue = cfa_samples(ip,2);
    ImageProcessor green = est_G_hamilton(ip);

    Convolver conv = new Convolver();
    conv.setNormalize(false);     // SANS normalisation (par défaut,
    convolve() normalise)
    // Composante R estimée par interpolation bilinéaire grâce à la
    convolution
    conv.convolve(red,kernel,3,3);
    conv.convolve(blue,kernel,3,3);

    // Calcul des échantillons de chaque composante de l'image CFA
    ImageStack samples_stack = imp.createEmptyStack();
    samples_stack.addSlice("rouge", red);    // Composante R
    samples_stack.addSlice("vert", green);// Composante G
    samples_stack.addSlice("bleu", blue);    // Composante B
```

```
    // Création de l'image résultat
    ImagePlus cfa_samples_imp = imp.createImagePlus();
    cfa_samples_imp.setStack("Echantillons couleur CFA",
    samples_stack);
    cfa_samples_imp.show();
}

ImageProcessor est_G_hamilton(ImageProcessor cfa_ip) {
  ImageProcessor est_ip = cfa_ip.duplicate();
  width = est_ip.getWidth();
  height = est_ip.getHeight();

  for(int x = 0; x < width; x++){
    for(int y = 0; y < height; y++){
      if(((x%2 != 0) && (y%2 == 0)) || ((x%2 == 0) && (y%2 != 1))){
        int centre = cfa_ip.getPixel(x,y)&0xff;

        int vert_gauche = cfa_ip.getPixel(x-1,y)&0xff;
        int vert_droite = cfa_ip.getPixel(x+1,y)&0xff;
        int vert_haut = cfa_ip.getPixel(x,y-1)&0xff;
        int vert_bas = cfa_ip.getPixel(x,y+1)&0xff;

        int rb_gauche = cfa_ip.getPixel(x-2,y)&0xff;
        int rb_droite = cfa_ip.getPixel(x+2,y)&0xff;
        int rb_haut = cfa_ip.getPixel(x,y-2)&0xff;
        int rb_bas = cfa_ip.getPixel(x,y+2)&0xff;

        int gradientX = Math.abs(vert_gauche - vert_droite) +
        Math.abs(2*centre - rb_gauche - rb_droite);
        int gradientY = Math.abs(vert_haut - vert_bas) +
        Math.abs(2*centre - rb_haut - rb_bas);

        if(gradientX < gradientY){
            est_ip.putPixel(x,y,(vert_gauche + vert_droite)/2 +
            (2*centre - rb_gauche - rb_droite)/4);
        }else if(gradientX > gradientY){
            est_ip.putPixel(x,y,(vert_haut + vert_bas)/2 + (2*centre -
            rb_haut - rb_bas)/4);
        }else{
            est_ip.putPixel(x,y,(vert_gauche + vert_haut + vert_gauche
            + vert_bas)/4 + (4*centre - rb_haut - rb_gauche -
            rb_droite - rb_bas)/8);
        }

    }
```

```
    }
  }


  return (est_ip);
}

public ImageProcessor cfa_samples(ImageProcessor cfa_ip, int k){
    width = imp.getWidth();
    height = imp.getHeight();

    ByteProcessor im_rouge_cfa = new ByteProcessor(width, height);
    ByteProcessor im_vert_cfa = new ByteProcessor(width, height);
    ByteProcessor im_bleu_cfa = new ByteProcessor(width, height);
    for(int i = 0; i < width; i++) {
           for(int j = 0; j < height; j ++) {
           //Cas des pixels verts
           if( ( (i%2 == 0) && (j%2 == 0) ) || ( (i%2 != 0) && (j%2 !=
           0) )){
                 im_vert_cfa.set(i,j,cfa_ip.getPixel(i,j));
           }
           //Cas des pixels rouges
           if( (i%2 != 0) && (j%2 == 0) ){
                 im_rouge_cfa.set(i,j,cfa_ip.getPixel(i,j));
           }
           //Cas des pixels bleus
           if( (i%2 == 0) && (j%2 != 0) ){
                 im_bleu_cfa.set(i,j,cfa_ip.getPixel(i,j));
           }
     }
    }
    if(k == 0){
      return im_rouge_cfa;
    }
    if(k == 1){
      return im_vert_cfa;
    }
    else{
      return im_bleu_cfa;
    }

}
```

Figure 15 - Résultat du dématriçage avec Hamilton & Adams

La figure 15 ci-dessus est le résultat du dématriçage par la méthode de Hamilton & Adams. On remarque que cette image est de meilleure qualité que celle obtenue par la méthode d'interpolation bilinéaire. On constate quelques fausses couleurs dans les zones de détails fins comme la barrière devant le phare. Cela peut être dû à des mauvais choix dans la direction d'interpolation ou à des incohérences entre direction d'interpolation pour le rouge et le bleu.

Évaluation de la qualité de l'image estimée

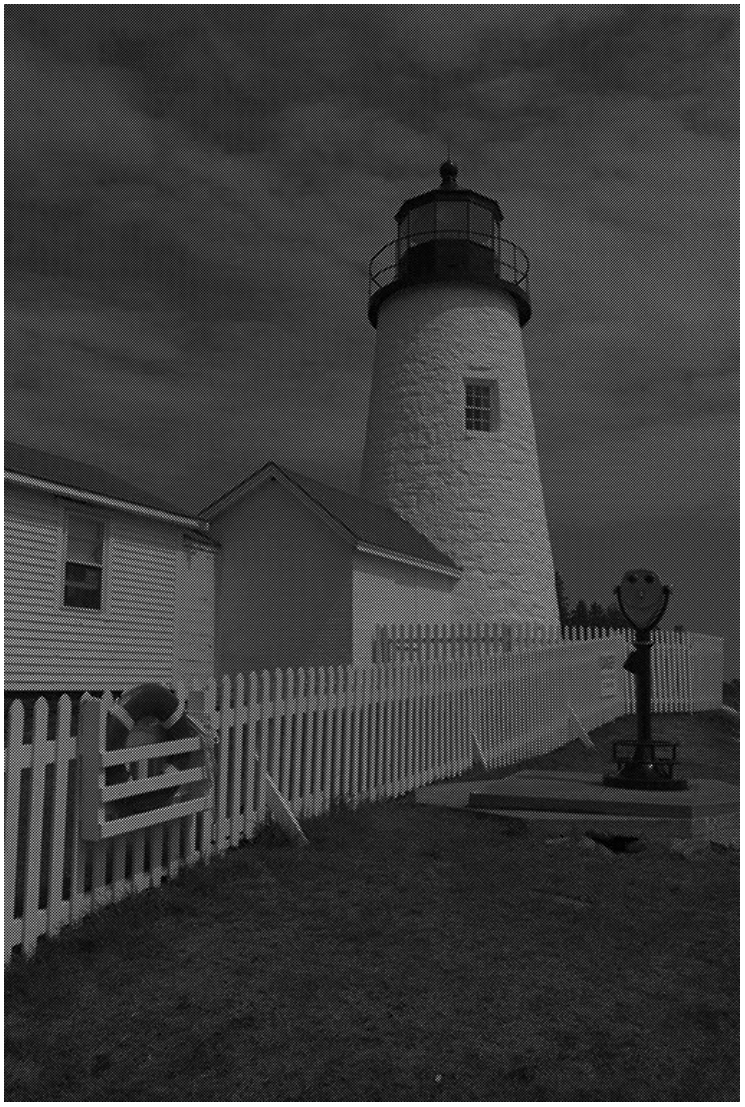


Figure 16 - Plan vert obtenu par interpolation bilinéaire

La figure 16 ci-dessus correspond au plan vert calculé par interpolation bilinéaire. La figure 17 ci-dessous correspond elle au plan vert obtenu avec l'algorithme d'Hamilton & Adams. On peut dans un premier temps comparer visuellement ces deux images on voit clairement que le plan de la figure 16 est plus sombre que celui de la figure 17. Dans un second temps nous avons calculé le PSNR entre les deux images. Le PSNR permet de mesurer la similarité entre deux images, en effet plus le celui-ci est élevé plus les deux images se ressemblent.

PSNR(vert,vert-1) = 9.7598dB

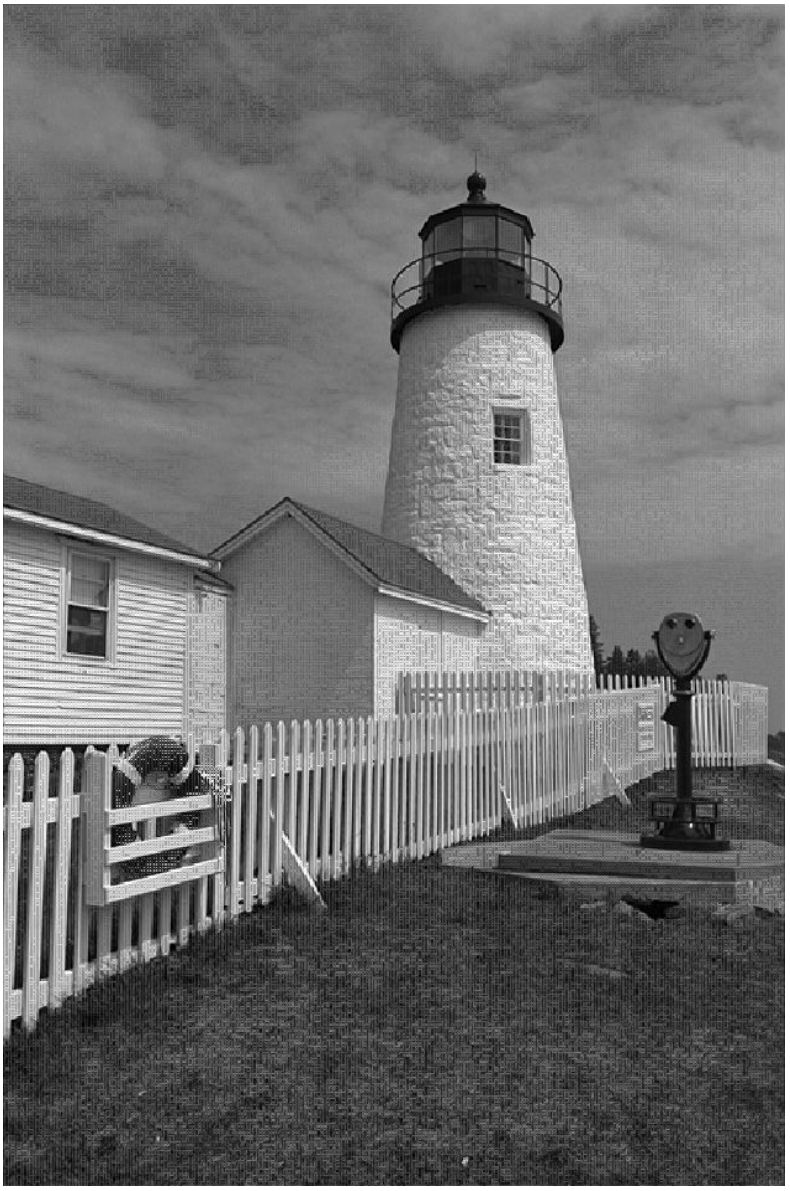


Figure 17 - Plan vert obtenu par la méthode de Hamilton & Adams