Université de Bourgogne
Faculté IEM

Imagerie Couleur et Multi-spectrale : image CFA – Interpolations - Métriques

Spécialité : Traitements d'images

Travail fait par :

Amine MANSOURI. Amine_Mansouri@etu.u-bourgogne.fr
Mohammed El Amine MOKHTARI mohammed-el-amine_mokhtari@etu.u-bourgogne.fr

Année Universitaire : 2020/2021.

Table des matières

Index des figures

Introduction

Ce travail vise à développer des algorithmes de démosaîcage déjà existants utilisés dans les caméras numériques pour la reconstruction des images couleurs. La plupart des caméras numériques actuelles utilisent un seul capteur devant lequel est placée une matrice de filtres couleurs. Ce capteur échantillonne par conséquent une seule couleur par position spatiale et l'utilisation d'un algorithme d'interpolation est nécessaire pour la définition d'une image couleur avec trois composantes par position spatiale.

Objectif

à travers ce travail, nous allons :

- mettre en œuvre quelques algorithmes de démosaïcage.
- tester ces algorithmes sur plusieurs images
- comparer leurs performances.

Partie I: Mosaïcage

On va créer une fonction Matlab qui transforme une image fournie en argument en une image simulée issue d'un **CFA de Bayer.**

La fonction à créer possède l'entête suivante : ***function Image_CFA = bayer (Input_RGB)*** où **Input_RGB** est l'**image couleur** (Nb_Lignes, Nb_Colonnes, 3) et **Image _CFA** est une image de dimensions (Nb_Lignes, Nb_Colonnes) censée être récupérée à la sortie du capteur.

Le programme ce présente comme suite :

```
function Image_CFA = bayer(Input_RGB)
%transformer l'image de départ en double pour ne pas avoir de problème de calcule:
    Input_RGB = double(Input_RGB);
    %extraie la Longueur , Largeur et nombre de bandes/canales de couleur:
    [H, W, k] = size(Input_RGB);
    %créer cer variable pour ne pas ce tramper de bande dans les calcules:
    R = 1;  G=2;    B=3;
    %initier à zeros la variable qui servira d'image de stocage de
    %calcules:
    im_resultat = zeros(H, W);
    %boucle qui fait la transformation de RGB_3D vers CFA_2D:
    for i=1:H
       for j=1:W
             %%-------------------------------------------------partie: pair
            if(mod(i, 2) == 0)
                if(mod(j, 2) == 0) %%pair / pair
                   im_resultat(i, j) = Input_RGB(i, j, G);%G7
                else                %%pair / impair
                   im_resultat(i, j) = Input_RGB(i, j, B);%B8
                end
            end
                %%---------------------------------------------- partie impaire:
             if(mod(i, 2) == 1)
               if(mod(j, 2) == 0) %%impair / pair
                   im_resultat(i, j) = Input_RGB(i, j, R); %R12
               else              %%impair / impair
                   im_resultat(i, j) = Input_RGB(i, j, G); %G13
               end
            end
       end
    end
%convertir le résultat finale en uint8 pour avoir un bon affichage de l'image :
    Image_CFA = uint8(im_resultat);
end
```

Figure 1: fonction de bayer qui transforme de RGB vers image CFA

Ce programme suit le mosaicage de la figure suivante :

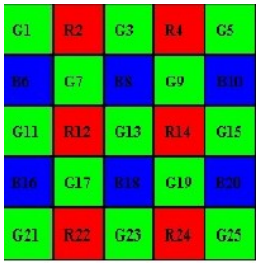


Figure 2: mosaicage de bayer

les pixels ont un emplacement pair/impair qui suit la logique suivante :

	j = 1	j = 2	…..
i = 1	Impair / impair	Impair / pair	…..
i = 2	P air / impair	Pair / pair	…..
…	…	…	…

On commence par les index ‘1’ car Matlab commence par ‘1’ et non pas ‘0’.

- On test le bon fonctionnement de la fonction avec le code suivant :

```
4
5 -  %%% Partie 1 Mosaicage:
6 -  im = imread('Demosaic5.tif');
7 -  im_res = bayer(im);
     figure, imshowpair(im, im_res, 'montage');title('image originale RGB.                Image CFA générée');
```

Figure 3: tester le bon fonctionnement de la fonction bayer()

on obtiens le résultat suivant :

Figure 4: image CFA générée à partir d'une image RGB avec bayer()

Le résultat de la fonction **bayer()** est en niveau de gris, ce qui est logique, car l'image CFA est en 2 Dimension (1 bande seulement).

Partie II : Démosaïcage

Dans cette partie, en implémente par des fonctions matlab les algorithmes suivants :

1. interpolation bilinéaire :

Définition :

L'interpolation bilinéaire est une méthode d'interpolation pour les fonctions de deux variables sur une grille régulière **(en 2D)**. Elle permet de calculer la valeur d'une fonction en un point quelconque, à partir de ses deux plus proches voisins dans chaque direction. C'est une méthode très utilisée en imagerie numérique pour le redimensionnement d'image, qui permet d'obtenir de meilleurs résultats que l'interpolation par plus proche voisin, tout en restant de complexité raisonnable.

Principe de fonctionnement :

En se basant sur la figure 2, l'algorithme se déroule de la manière suivante :

1. Interpolation des pixels verts : par **exemple** : G8 = (G3+G7+G9+G13) / 4
2. Interpolation des pixels rouge/bleu :
   - Interpolation des pixels rouge/bleu à l'endroit des pixels vert :
     La moyenne des valeurs des deux pixels adjacents de la même couleur est assignée au pixel interpolé.
     par **exemple** : B7 = (B6+B8) / 2 ; R7 = (R2+R12) / 2
   - Interpolation des pixels rouge/bleu à l'endroit des pixels bleu/rouge :
     la moyenne des valeurs des quatre pixels diagonaux adjacents est assigné au pixel interpolé.
     Par **exemple** : R8 = (R2+R4+R12+R14) / 4 ; B12 = (B6+B8+B16+B18) / 4.

Code :

En utilisant matlab, on a le script suivant de la fonction **bilineaire_interpolation() :**

```
%% 2 demosaicage:
%%% 1) interpolation bilinéaure:

function Output_RGB = bilineaire_interpolation(Image_CFA)
% si on oublie de caster l'argument input avant de l'injecter dans la fonction au debut,
%cette etape le fait. Transformer l'image de départ en double c'est pour ne pas avoir de problème de calcule:
        Image_CFA = double(Image_CFA);
        %extraie la Longueur , Largeur et nombre de bandes/canales de couleur:
        [H, W] = size(Image_CFA);
        %créer cer variable pour ne pas ce tramper de bande dans les calcules:
        R = 1;  G=2;    B=3;
        %initier à zeros la variable qui servira d'image de stocage de
        %calcules:
        im_resultat = zeros(size(Image_CFA));
        %boucle qui fait la transformation de CFA_2D vers  RGB_3D:
        for i=2:H-1
           for j=2:W-1
               %%%%%%%%%%%%% ----------------------------------------------------partie pair:
               if(mod(i, 2) == 0)
                   if(mod(j, 2) == 0)  %%%pair/pair
                       % R7 / B7
                      im_resultat(i, j, R) = ( Image_CFA(i - 1, j)+ Image_CFA(i + 1, j)) / 2;
                      im_resultat(i, j, B) = ( Image_CFA(i , j - 1)+ Image_CFA(i , j + 1)) / 2;
                      im_resultat(i, j, G) =  Image_CFA(i, j);
                   else % G8 /R8         %%pair/impair
                      im_resultat(i, j, G) = ( Image_CFA(i - 1, j)+ Image_CFA(i + 1, j) + Image_CFA(i , j-1)+ Image_CFA(i, j+1)) / 4;
                      im_resultat(i, j, R) = ( Image_CFA(i - 1, j-1)+ Image_CFA(i - 1, j+1) + Image_CFA(i+1 , j-1)+ Image_CFA(i+1, j+1)) / 4;
                      im_resultat(i, j, B) = Image_CFA(i, j);
                   end
               end
                   %%%%%%%%%%  ----------------------------------------------------partie impair:
                if(mod(i, 2) == 1)
                  if(mod(j, 2) == 0)         %%%impair/pair
                     %G12 / B12
                      im_resultat(i, j, G) = ( Image_CFA(i - 1, j)+ Image_CFA(i + 1, j) + Image_CFA(i , j-1)+ Image_CFA(i, j+1)) / 4;
                      im_resultat(i, j, B) = ( Image_CFA(i - 1, j-1)+ Image_CFA(i - 1, j+1) + Image_CFA(i+1 , j-1)+ Image_CFA(i+1, j+1)) / 4;
                      im_resultat(i, j, R) = Image_CFA(i, j);
                  else  % R13/B13             %%%impair/impair
                      im_resultat(i, j, B) = ( Image_CFA(i - 1, j)+ Image_CFA(i + 1, j)) / 2;
                      im_resultat(i, j, R) = ( Image_CFA(i , j - 1)+ Image_CFA(i , j + 1)) / 2;
                      im_resultat(i, j, G) = Image_CFA(i, j);
                  end
               end
           end
        end

        %convertir le résultat finale en uint8 pour avoir un bon affichage de l'image :
        Output_RGB = uint8(im_resultat);
end
```

on traite dans ce code les 4 cas de figure qui peuvent apparaître éventuellement.

pair / pair **pair / impair** **impair / pair** **impair / impair**

dans chaque condition on a spécifié la case et les couleurs qui seront traités, juste pour bien se situer dans la figure 2 .

La variable de retour de cette fonction et d'ailleurs les autre fonction des prochaines méthodes, auront le même type de retour **uint8.** Ce type de variable nous donnera le bon affichage de l'image résultante qu'un type **double**.

Affichage du résultat de la fonction bilinéaire :

Pour avoir un bon aperçu du résultat, on applique le traitement sur l'image **Demosaic5.tif**, en utilisant le code suivant :

```
%% 2) interpolation bilinéaire:
%lire l'image originale:
 im_RGB = imread('Demosaic5.tif'); % figure, imshow(im);
%appliqueer l'interpolation bilinéaire sur une image CFA.
 im_OUT = bilineaire_interpolation(bayer(im_RGB));
%affichage des résultats:
 figure, imshowpair(im_RGB, im_OUT, 'montage');  title('image originale RGB.                    image générée par interpolation bilinéaire');
 figure, imshowpair(im_RGB(444:544, 317:417, :), im_OUT(444:544, 317:417, :), 'montage'); title('artefacts');
```

et on obtiens comme résultat :

Remarque :

l'interpolation bilinéaire apporte un certain nombre d'erreurs qui génèrent des artefacts, montrés ci-dessus. Aussi, l'interpolation bilinéaire introduit du **flou** sur l'image, Il est présent lorsqu'une transition de l'image d'origine n'est plus aussi franche dans l'image interpolée. On peut apercevoir ce flou sur la zone suivante :

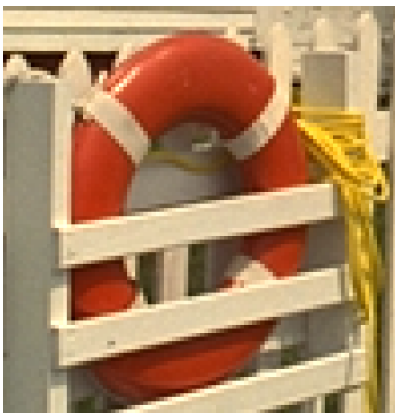


Figure 5: partie de l'image originale.

Figure 6: partie de l'image aprés interpolation

- l'**effet d'escalier** apparaît aussi sur les zone qui ne sont ni verticales ni horizontales, comme sur le **mur du phare.**

-l'effet **Aliasing** est important aussi car cette méthode fait des estimation grossière des pixels manquants

-**Définition de l'effet aliasing** : Il apparaît quand les hautes fréquences de l'image d'origine sont mal reproduites dans l'image interpolée. L'aliasing apparaît alors comme une basse fréquence parasite.

2. l'interpolation sous la contrainte de constance de la teinte :

Définition :

La propriété de corrélation spectrale montrent que les niveaux des trois composantes sont fortement corrélés dans une image couleur naturelle, et tout particulièrement dans les zones de hautes fréquences spatiales. Pour exploiter cette corrélation spectrale intra-pixel lors du dématriçage, on utilise le principe de constance de teinte.

Principe de fonctionnement :

1. Interpolation pixels vert: comme pour l'interpolation bilinéaire.

2. Considérer les valeurs des pixels bleu : trois cas de figure:

- Estimer les pixels bleu à l'endroit des pixels vert et les pixels bleu adjacents sont à gauche et à droite : $B7 = G7 / 2 * (B6 / G6 + B8 / G8)$
- Estimer les pixels bleu à l'endroit des pixels vert et les pixels bleu adjacents sont en haut et en bas : $B13 = G13 / 2 * (B8 / G8 + B18 / G18)$
- Estimer les pixels bleu à l'endroit des pixels rouge :

$$B12 = G12 / 4 * (B6 / G6 + B8 / G8 + B16 / G16 + B18 / G18)$$

Code :

En utilisant matlab, on a le script suivant de la fonction **cste_teinte_interpolation() :**

```
%%% methode d'interpolation sous constante (cste) the teinte:
function out = cste_teinte_interpolation(im_CFA_uint8, selecteur)
% si on oublie de caster l'argument input avant de l'injecter dans la fonction au debut,
%cette etape le fait. Transformer l'image de départ en double c'est pour ne pas avoir de problème de calcule:
im_cfa = double(im_CFA_uint8);
%extraie la Longueur , Largeur et nombre de bandes/canales de couleur:
[H, W, k] = size(im_CFA_uint8);
%créer cer variable pour ne pas ce tramper de bande dans les calcules:
R = 1;  G=2;    B=3;
%initier à zeros la variable qui servira d'image de stocage de
%calcules:
im_resultat_3 = zeros(H, W, k);
%generer la bande verte de l'image résultante avec la fonction
%green_gen(). l'argument selecteur détermine quel méthode choisir
green_band = green_gen(im_cfa, selecteur); %selecteur = 1 : choisir le case 1 du switch de green_ge() qui corespond a l'algo
%%
for i=2:H-1
   for j=2:W-1
%%%%%%%%%%%% pair:
       if(mod(i, 2) == 0 && mod(j, 2) == 0)  %%%pair/pair ----------------------------
                % R7 / B7
                im_resultat_3(i, j, R) = ( green_band(i , j)/2 ) * ( (im_cfa(i-1 , j) / green_band(i-1, j)) + (im_cfa(i+1 , j) / green_band(i+1, j)) );
                im_resultat_3(i, j, B) = ( green_band(i , j)/2 ) * ( (im_cfa(i , j-1) / green_band(i, j-1) ) + (im_cfa(i , j+1) / green_band(i, j+1)) );
                im_resultat_3(i, j, G) =  im_cfa(i, j);
       elseif(mod(i, 2) == 0 && mod(j, 2) == 1) %%pair/impair---
                % G8 /R8
                im_resultat_3(i, j, R) =  (green_band(i , j) / 4) * ((im_cfa(i-1 , j-1) / green_band(i-1, j-1)) + (im_cfa(i-1 , j+1) / green_band(i-1, j+1)) + ...
                                                                     (im_cfa(i+1 , j-1) / green_band(i+1, j-1)) + (im_cfa(i+1 , j+1) / green_band(i+1, j+1)) );
                im_resultat_3(i, j, B) = im_cfa(i , j);
                im_resultat_3(i, j, G) = green_band(i, j);
%%%%%%%%%%  impair:
       elseif(mod(i, 2) == 1 && mod(j, 2) == 0)     %%%impair/pair-------------------------------------
                %G12 / B12
                im_resultat_3(i, j, B) = (green_band(i , j) / 4) * ( (im_cfa(i-1 , j-1) / green_band(i-1, j-1)) + (im_cfa(i-1 , j+1) / green_band(i-1, j+1)) + ...
                                                                     (im_cfa(i+1 , j-1) / green_band(i+1, j-1)) + (im_cfa(i+1 , j+1) / green_band(i+1, j+1)) );
                im_resultat_3(i, j, R) = im_cfa(i , j);
                im_resultat_3(i, j, G) = green_band(i, j);
       else                                          %%%impair/impair --------------------------
                % R13/B13
                im_resultat_3(i, j, B) = (green_band(i , j) / 2) * ( (im_cfa(i-1 , j) / green_band(i-1, j)) + (im_cfa(i+1 , j) / green_band(i+1, j)) );
                im_resultat_3(i, j, R) = (green_band(i , j) / 2) * ( (im_cfa(i , j-1) / green_band(i, j-1)) + (im_cfa(i , j+1) / green_band(i, j+1)) );
                im_resultat_3(i, j, G) =  im_cfa(i, j);
       end
   end
end
%convertir le résultat finale en uint8 pour avoir un bon affichage de l'image :
out = uint8(im_resultat_3);
end
```

Remarque :

- Dans ce code la fonction g**reen_gen()** est utiliser pour générer une la bande verte de l'image résultante (im_resultat_3). De cette manière, on pourra visualiser et comprendre le code aisément.

-la fonction **green_gen()** est une fonction **commune** pour les **différent algorithmes** qui vont suivre.

- A l'intérieur de la fonction **green_gen(),** on retrouve une structure **switch** qui nous offre plusieurs modes de calcule de la bande verte

- on navigue dans la fonction **green_gen()** en utilisant la variable **selecteur,** par exemple :

- **selecteur = 1** => choisir le bloc qui calcule la bande verte avec une interpolation sous constante de teinte
- **selecteur = 2** => choisir le bloc qui calcule la bande verte avec une interpolation avec préservation de contour. Méthode 1.
- **selecteur = 3** => choisir le bloc qui calcule la bande verte avec une interpolation avec préservation de contour. Méthode 2 qui utilise un noyau/masque plus large.
- **selecteur = 4** => choisir le bloc qui calcule la bande verte avec une interpolation par détection de formes

le code qui se situe dans la fonction **green_gen()** est utilisé par la fonction d'interpolation sous constante de teinte :

```
%%%%% generate a green band from a CFA image input:
function out = green_gen(im_CFA_in, selection)
    im_cfa = double(im_CFA_in);

    switch selection

        case 1 %pour la fonction cste_teinte_interpolation():
                [H, W] = size(im_cfa);
                R = 1;  G=2;    B=3;
                %convertir en double pour éviter des problèmes de calculss
                green_band = double(im_cfa);

                %%% de im_cfa vers un canal vert:
                for i=2:H-1
                   for j=2:W-1
                       %%%%%%%%%%%% pair:
                       if(mod(i, 2) == 0)
                           if(mod(j, 2) == 1)  %%%pair/impair
                               %G8
                              green_band(i, j) = ( im_cfa(i - 1, j)+ im_cfa(i + 1, j) + im_cfa(i , j-1)+ im_cfa(i, j+1)) / 4;
                           end
                       end
                           %%%%%%%%%%  impair:
                      if(mod(i, 2) == 1)
                          if(mod(j, 2) == 0)         %%%impair/pair
                              %G12
                              green_band(i, j) = ( im_cfa(i - 1, j)+ im_cfa(i + 1, j) + im_cfa(i , j-1)+ im_cfa(i, j+1)) / 4;
                          end
                      end
                   end
            end
            %cette étape nous permet de méttre les cases de valeur 0 à 1.ça permet d'éliminer les problémes de division par 0.
              out = (green_band + 1); % double
```

Figure 7: partie de la fonction green_gen() qui est utilisé par la fonction d'interpolation cste_teinte_interpolation()

Affichage des résultats de la fonction :

Pour avoir un bon aperçu du résultat, on applique le traitement sur l'image **Demosaic5.tif**, en utilisant le code suivant :

```
%% 2)  fonction interpolation sous constante de teinte:
%lecture de l'image:
im_RGB = imread('Demosaic5.tif'); % figure, imshow(im); %imread('Demosaic_4.bmp');
%choisir selecteur = 1, pour avoir la premier 'case' du switch de la fonction green_gen():
selecteur = 1 ;
im_OUT = cste_teinte_interpolation(bayer(im_RGB) , selecteur);
%affichage des résultats:
figure, imshowpair(im_RGB,im_OUT, 'montage');
title('image originale RGB.                                image générée par interpolation cste de teinte');
```

on obtiens le résultat suivant :

Remarque :

-Avec cette méthode on remarque que les artefacts s'atténuent un peut par rapport a la fonction bilinéaire. Parce que cette méthode prend en considération la corrélation spectrale entre les niveaux des trois composantes RGB. Même le flou diminue. Mais cette méthode génère des erreur d'estimation dans les zones de haute fréquence.

3. l'interpolation sous contrainte de préservation des contours :

Définition :

utilise le gradient horizontal et vertical autour du pixel manquant, calculé à partir des pixels voisins existants. L'idée est de réaliser une interpolation le long des contours et d'éviter une interpolation à travers eux.
Il est important de noter que les méthodes utilisant le gradient sont très utilisées dans le démosaïçage. Ces méthodes sont judicieuses puisqu'à chaque pixel manquant dans la matrice de Bayer, le gradient suivant deux directions est parfaitement connu à partir des pixels existants.

Principe de fonctionnement de la méthode 1 avec un noyau 3x3:

1. Interpolation des pixels vert:

Définit deux gradients, un horizontal et l'autre vertical. Exemple pour B8 (on utilise toujours la grille figure 2) :

définit les deux gradients comme ΔH=|G7-G9| and ΔV=|G3-G13|

Si ΔH<ΔV,
G8=(G7+G9)/2;
Sinon si ΔH>ΔV,
G8=(G3+G13)/2;
Sinon
G8=(G3+G7+G9+G13)/4
FIN

2. Interpolation des pixels rouge/bleu : comme pour la constance de teinte

Code :

On a le script suivant de la fonction **contour_pres_inter() :**

```
%%% methode d'interpolation preservation de contours:
function out = contour_pres_inter(im_CFA_uint8, selecteur)
% si on oublie de caster l'argument input avant de l'injecter dans la fonction au debut,
%cette etape le fait. Transformer l'image de départ en double c'est pour ne pas avoir de problème de calcule:
im_cfa = double(im_CFA_uint8);
%extraie la Longueur , Largeur et nombre de bandes/canales de couleur:
[H, W, k] = size(im_CFA_uint8);
%créer cer variable pour ne pas ce tramper de bande dans les calcules:
R = 1;  G=2;    B=3;
%initier à zeros la variable qui servira d'image de stocage de
%calcules:
im_resultat_3 = zeros(H, W, k);
%generer la bande verte de l'image résultante avec la fonction green_gen(). l'argument selecteur détermine quel méthode choisir
green_band = green_gen(im_cfa, selecteur); %selecteur = 1 : choisir le case 2 du switch de green_ge() qui corespond a l'algo
%%
for i=2:H-1
   for j=2:W-1
%%%%%%%%%%%% pair:
      if(mod(i, 2) == 0 && mod(j, 2) == 0)  %%%pair/pair ----------------------------
            % R7 / B7
            im_resultat_3(i, j, R) = ( green_band(i , j)/2 ) * ( (im_cfa(i-1 , j) / green_band(i-1, j)) + (im_cfa(i+1 , j) / green_band(i+1, j)) );
            im_resultat_3(i, j, B) = ( green_band(i , j)/2 ) * ( (im_cfa(i , j-1) / green_band(i, j-1) ) + (im_cfa(i , j+1) / green_band(i, j+1)) );
            im_resultat_3(i, j, G) =  im_cfa(i, j);
      elseif(mod(i, 2) == 0 && mod(j, 2) == 1) %%pair/impair-----------------
            % G8 /R8
            im_resultat_3(i, j, R) =  (green_band(i , j) / 4) * ((im_cfa(i-1 , j-1) / green_band(i-1, j-1)) + (im_cfa(i-1 , j+1) / green_band(i-1, j+1)) +
                                                                 (im_cfa(i+1 , j-1) / green_band(i+1, j-1)) + (im_cfa(i+1 , j+1) / green_band(i+1, j+1)) );
            im_resultat_3(i, j, B) = im_cfa(i , j);
            im_resultat_3(i, j, G) = green_band(i, j);
%%%%%%%%%%  impair:
      elseif(mod(i, 2) == 1 && mod(j, 2) == 0)     %%%impair/pair-----------------------------------
            %G12 / B12
            im_resultat_3(i, j, B) = (green_band(i , j) / 4) * ( (im_cfa(i-1 , j-1) / green_band(i-1, j-1)) + (im_cfa(i-1 , j+1) / green_band(i-1, j+1)) +
                                                                 (im_cfa(i+1 , j-1) / green_band(i+1, j-1)) + (im_cfa(i+1 , j+1) / green_band(i+1, j+1)) )
            im_resultat_3(i, j, R) = im_cfa(i , j);
            im_resultat_3(i, j, G) = green_band(i, j);
      else                                      %%%impair/impair ------------------------
            % R13/B13
            im_resultat_3(i, j, B) = (green_band(i , j) / 2) * ( (im_cfa(i-1 , j) / green_band(i-1, j)) + (im_cfa(i+1 , j) / green_band(i+1, j)) );
            im_resultat_3(i, j, R) = (green_band(i , j) / 2) * ( (im_cfa(i , j-1) / green_band(i, j-1)) + (im_cfa(i , j+1) / green_band(i, j+1)) );
            im_resultat_3(i, j, G) =  im_cfa(i, j);
      end
   end
end
    out = uint8(im_resultat_3);
end
```

-Ce code comporte les même conditions que celles des méthodes d'interpolations précédentes.
Ce qui change est le choix du selecteur qui est égale à 2 pour activer la bonne partie de la fonction **green_gen()**.
-Dans la fonction **green_gen(),** dans la variable de sortie **out,** on **ajoute un '1'** pour **initier les valeurs '0'** de la **bande verte générée** et les **mettre à '1',** afin d'éviter le problème de division sur le nombre '0' dans l'interpolation du **Rouge/Bleu. S**i on ne résoud pas ce probleme de division sur 0, il va nous généré des valeur **NaN**(Not a Number) qui causerons des trous noires dans l'image de sortie.

Affichage des résultats de la fonction :

Pour afficher les résultats, on applique le code suivant :

```
%% % 3) partie 1:
%%Interpolation sous preservation de contours avec un noyau (3x3):

%lecture de l'image:
im_RGB = imread('Demosaic5.tif');
% %choisir selecteur = 2, pour avoir le deuxième 'case' du switch de la fonction green_gen():
selecteur = 2 ;
im_OUT = contour_pres_inter(bayer(im_RGB) , selecteur);
%affichage des résultats:
figure, imshowpair(im_RGB,im_OUT, 'montage');title('image originale RGB.                    image générée par interpolation préservation contour')
figure, imshowpair(im_RGB(444:544, 317:417, :), im_OUT(444:544, 317:417, :), 'montage'); title('artefacts: Aliasing');
```

on obtiens comme affichage :

-

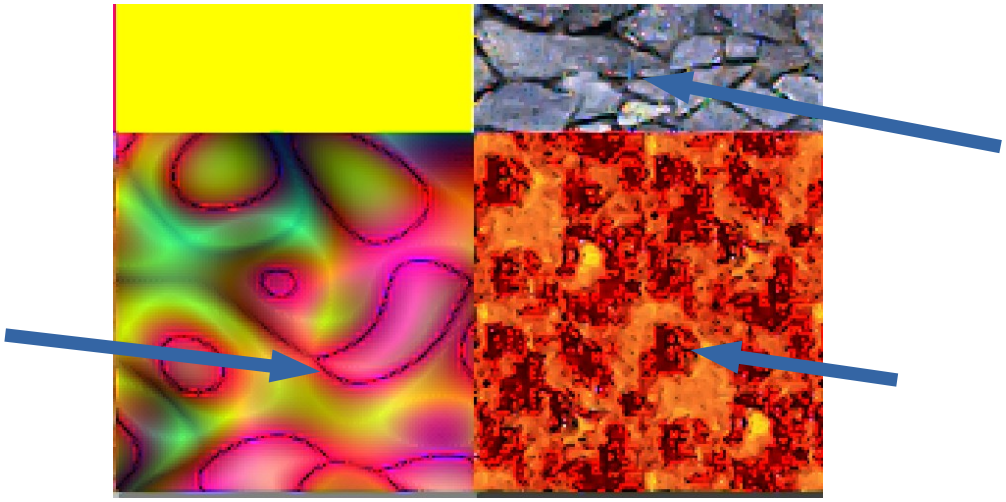
Remarque :

-Avec cette interpolation qui utilise le gradient verticale et horizontale, on remarque que l'effet aliasing et le flou , diminuent par rapport aux méthode précédente. Car le gradient prend en considération son voisinage de pixel

-Mais, d'autre part, on remarque aussi qu il y a apparition de contours indésirable dans certaine zones de transition de couleurs, du coup la transition n'est pas lisse,

Cet effet se surnom **ringing**. Ce phénomène aussi appelé écho, est présent le long des contours. Il est souvent dû à l'**overshoot** (ou l'**undershoot**) introduit par les rebonds de la fonction d'interpolation.

comme montré sur la figure :



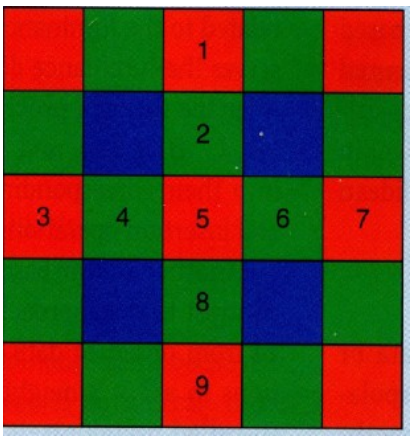
Principe de fonctionnement de la méthode 2 avec noyau 5x5:

Cette méthode est connue aussi pour le nom Interpolation directionnelle par le gradient corrigé du Laplacien .

Cette méthode proposent d'utiliser d'autres estimateurs pour la classification et l'interpolation des pixels. Dans cette implémentation, on utilise un calcul de gradient sur les pixels verts et de Laplaciens sur les pixels rouges ou bleus. La méthode pour l'interpolation du pixel vert *G5* est basée sur l'algorithme suivant :

1. Interpolation des pixels vert:

    1. Calculer le gradient horizontal
        $\Delta H = |G4-G6| + |R5-R3+R5-R7|$
    2. Calculer le gradient vertical
        $\Delta V = |G2-G8| + |R5-R1+R5-R9|$
    3. Si $\Delta H > \Delta V$
        $G5 = G2+G8/2 + (R5-R1+R5-R9)/4$
    Sinon si $\Delta H < \Delta V$
        $G5 = G4+G6/2 + (R5-R3+R5-R7)/4$
    Sinon
        $G5 = (G4+G6+G2+G8)/4 + \ldots$
        $(R5-R1+R5-R9+R5-R3+R5-R7)/8$



2. Interpolation pixels rouge/bleu : comme pour la constance de teinte

Code :

Concernant le code de cette algorithme, il suffit juste de mettre la variable selecteur = 3. pour activer cette partie de la fonction commune **green_gen()**

```
case 3 %pour la fonction countour_pres_inter() avec un noyau plus large (5x5):
    [H, W] = size(im_cfa);
    R = 1;  G=2;    B=3;
    %convertir en double pour éviter des problèmes de calculss
    green_band = double(im_cfa);

    %%% de im_cfa vers un canal vert:
    for i=4:H-3  %we reduce the image because the kernel is wider (5x5)
       for j=4:W-3
           %%%%%%%%%%%% pair:
           if( (mod(i, 2) == 1 && mod(j, 2) == 0) || (mod(i, 2) == 0 && mod(j, 2) == 1) )  %%% si: pair/impair OR impair/pair
               %G14 OR G10
               DH = abs(im_cfa(i , j-1) - im_cfa(i, j+1)) + abs(im_cfa(i, j) - im_cfa(i, j-2) + im_cfa(i, j) - im_cfa(i, j+2));
               DV = abs(im_cfa(i-1 , j) - im_cfa(i+1, j)) + abs(im_cfa(i, j) - im_cfa(i-2, j) + im_cfa(i, j) - im_cfa(i+2, j));
               if(DH < DV)
                 green_band(i, j) =  ((im_cfa(i , j-1) + im_cfa(i, j+1)) / 2) +  ((im_cfa(i, j) - im_cfa(i, j-2) + im_cfa(i, j) - im_cfa(i, j+2)) / 4);
               elseif(DH > DV)
                 green_band(i, j) =  ((im_cfa(i-1 , j) + im_cfa(i+1, j)) / 2) +  ((im_cfa(i, j) - im_cfa(i-2, j) + im_cfa(i, j) - im_cfa(i+2, j)) / 4);
               else
                 green_band(i, j) = ( (im_cfa(i - 1, j)+ im_cfa(i + 1, j) + im_cfa(i , j-1)+ im_cfa(i, j+1)) / 4) + ...
                                    (((im_cfa(i, j)-im_cfa(i-2,j)+im_cfa(i, j)-im_cfa(i+2, j))+(im_cfa(i,j)-im_cfa(i,j-2)+im_cfa(i,j)-im_cfa(i,j+2)))/8);
               end
           end
       end
    end
    %on ajoute le 1, pour éviter la division sur 0 et avoir des cases NaN(Not a Number)
    out = (green_band + 1); % double
```

Figure 8: partie de la fonction green_gen() utilisée dans l'interpolation preservation de contours méthode 2

Affichage des résultats de la fonction :

On utilise le code suivant :

```
%% % 3) partie 2:
%%Interpolation sous preservation de contours avec un noyau (5x5):

%lecture de l'image:
im_RGB = imread('Demosaic5.tif');
% %choisir selecteur = 2, pour avoir le deuxième 'case' du switch de la fonction green_gen():
selecteur = 3 ;
im_OUT = contour_pres_inter(bayer(im_RGB) , selecteur);
%affichage des résultats:
figure, imshowpair(im_RGB,im_OUT, 'montage');title('image originale RGB.                          image générée par interpolation préservation contour
figure, imshowpair(im_RGB(444:544, 317:417, :), im_OUT(444:544, 317:417, :), 'montage'); title('artefacts: Aliasing');
```

on obtiens l’affichage suivant :

on remarque que l'artefact aliasing diminue considérablement quand on augmente la taille du noyau. Car la relations entre les pixels voisins est plus étroite et estime mieux les pixels manquants quand cherche à remplir.

4. l'interpolation bilinéaire en utilisant un masque de convolution:

Définition :

Comme le pas d'échantillonnage de chacune des couleurs est régulier dans le CFA de Bayer, il est possible d'appliquer l'interpolation bilinéaire à l'aide de filtres de convolution. Par exemple le filtre $\mathbf{F_G}$ proposé dans la figure 9 ci-dessous permet l'interpolation des pixels verts manquants par convolution de ce noyau de filtre avec la matrice des pixels verts dans laquelle les pixels manquants sont remplis par des zéros. De même, le noyau du filtre de convolution $\mathbf{F_{RB}}$ permet l'interpolation bilinéaire des plans rouge et vert.

$$\mathbf{F_G} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} /4 \qquad \mathbf{F_{RB}} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} /4$$

Figure 9: noyaux de convolution

Principe de fonctionnement :

1. on parcours toute l'image CFA, et selon la condition (eg : pair/impair) on affecte le bon pixel à la bonde de couleur appropriée.

2. après avoir former les 3 canaux de couleur de l'image de résultat, on fait la convolution de chaque bande de couleur avec le noyau spécifié, en utilisant la fonction prédéfinie de Matlab **conv2(),** en argument de la fonction on injecte la bande et son noyau respectif. La fonction **conv2()** nous facilite la tache en appliquant la convolution sur toute la bande 2D de l'image.

Code :

Le code de la fonction **bilineaire_conv_interpolation()** qui fait la convolution entre l'image CFA et les noyaux de convolution, se présente sous la forme suivante :

```
function out = bilineaire_conv_interpolation(im_CFA_uint8)
% si on oublie de caster l'argument input avant de l'injecter dans la fonction au debut,
%cette etape le fait. Transformer l'image de départ en double c'est pour ne pas avoir de problème de calcule:
    im_cfa = double(im_CFA_uint8);
    %extraie la Longueur , Largeur et nombre de bandes/canales de couleur:
    [H, W, k] = size(im_CFA_uint8);
    %créer cer variable pour ne pas ce tramper de bande dans les calcules:
    R = 1;  G=2;    B=3;
    %initier à zeros la variable qui servira d'image de stocage de
    %calcules:
    im_resultat_3 = zeros(H, W, k);
    %% créer les noyaux de convolutions:
    kernel_G  = [0, 1, 0;  1, 4, 1;  0, 1, 0] ./4;
    kernel_RB = [1, 2, 1;  2, 4, 2;  1, 2, 1] ./4;
    %% boucles qui génèrent les 3 bandes de l'image resultante séparémment:
    for ii=2:H-1
        for jj=2:W-1
        %%%%%%%%%%%% pair:-------------------------------------------------------------------
           if(mod(ii, 2) == 0 && mod(jj, 2) == 0)         %%%pair/pair ----------------------
                  % R7 / B7
                 im_resultat_3(ii, jj, G) =  im_cfa(ii, jj);
           elseif(mod(ii, 2) == 0 && mod(jj, 2) == 1)     %%pair/impair-----------------
                  % G8 /R8
                 im_resultat_3(ii, jj, B) = im_cfa(ii, jj);
        %%%%%%%%%%  impair:-------------------------------------------------------------------
           elseif(mod(ii, 2) == 1 && mod(jj, 2) == 0)     %%%impair/pair----------------------------------
                 %G12 / B12
                 im_resultat_3(ii, jj, R) = im_cfa(ii, jj);
           else                                           %%%impair/impair ------------------------
                 % R13/B13
                 im_resultat_3(ii, jj, G) =  im_cfa(ii, jj);
           end
        end
    end
    %% calcule de convolution aprés obtention des bandes de couleurs:
    im_resultat_3(:, :, R) = conv2(im_resultat_3(:, :, R), kernel_RB, 'same');
    im_resultat_3(:, :, G) = conv2(im_resultat_3(:, :, G), kernel_G, 'same');
    im_resultat_3(:, :, B) = conv2(im_resultat_3(:, :, B), kernel_RB, 'same');
    %% image de sortie en uint8 pour avoir le bon affichage:
    out = uint8(im_resultat_3);
end
```

on remarque dans le code que la fonction conv2() applique la convolution sur toute la bande c'est pour cette raison qu'elle est utilisée après les boucles imbriquées

Affichage des résultats de la fonction :

On affiche le résultat du traitement en utilisant le code suivant :

```
%% 4) interpolation bi-lineaire avec un masque:

%lecture de l'image:
im_RGB = imread('Demosaic5.tif');
%générer l'image CFA:
im_CFA = bayer(im_RGB);
%obtention de l'image interpollée:
im_OUT = bilineaire_conv_interpolation(im_CFA); % l'image qui entre en argument pour la fonction bi_inter , il faut un type double;
%affichage des résultats:
figure, imshowpair(im_RGB,im_OUT, 'montage');title('image originale RGB. / image générée par interpolation bilinéaire avec convolution');
figure, imshowpair(im_RGB(444:544, 317:417, :), im_OUT(444:544, 317:417, :), 'montage'); title('artefacts: Aliasing');
```

on obtiens le l'affichage suivant :

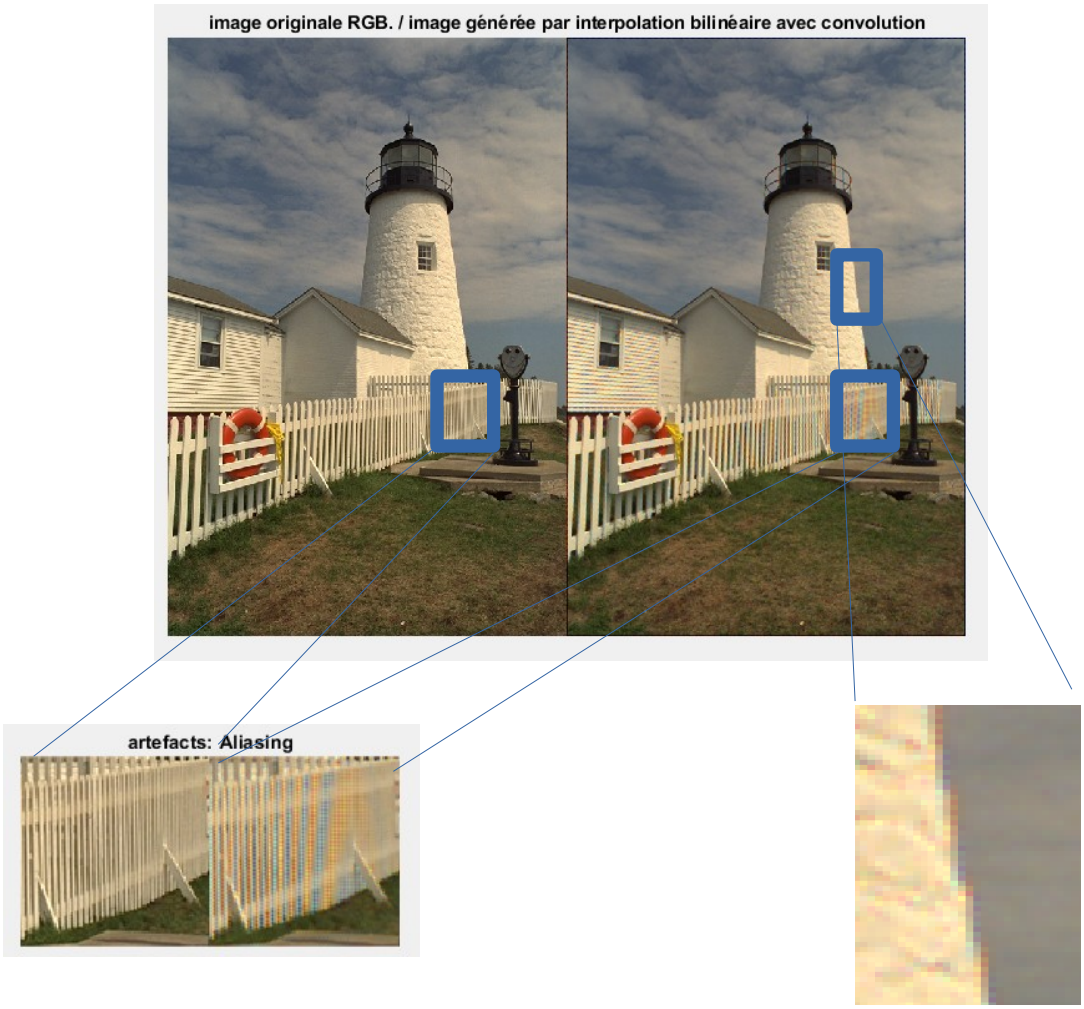
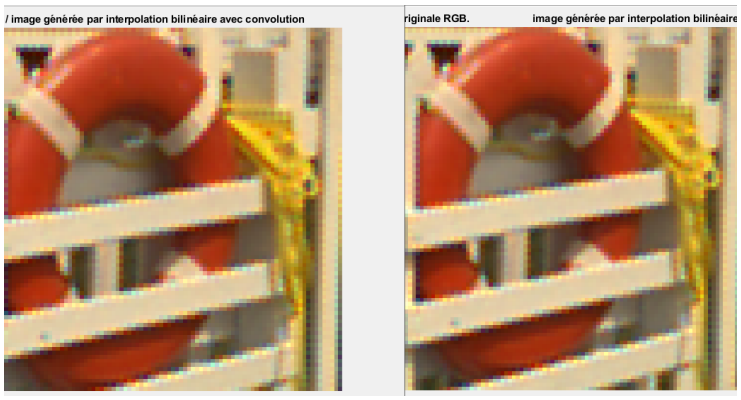


Figure 10: effet escalier

- On remarque que l'effet aliasing (ou zipper) et le même que sur une interpolation bilinéaire sans convolution , le flou aussi est présent, comme le montre la figure suivante :



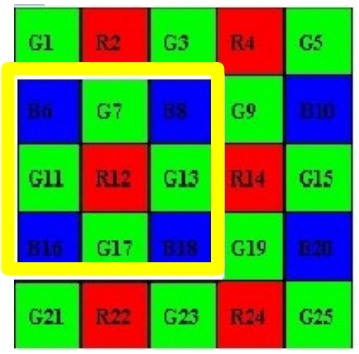
5. l'interpolation dédiée à la reconnaissance des formes:

Définition :

En parallèle de la méthode d'interpolation de teinte, Cok propose une méthode d'interpolation basée sur la reconnaissance des formes. En effet, l'hypothèse de constance de teinte n'est plus valide à la frontière entre les objets. Ce problème est particulièrement important et a connu de nombreuses propositions de solution. L'objectif de cette nouvelle méthode est de classifier plusieurs types de formes de voisinage et de réaliser une interpolation différente suivant le type de voisinage. D'après lui, cette méthode permet de repousser les erreurs d'interpolation dans les zones texturées tel que l'observateur humain ne peut les voire. Ceci étant, cette méthode est avant-gardiste de nombreuses autres, tâchant d'améliorer l'interpolation des pixels verts supposées responsables de l'échec partiel de la méthode de constance des teintes. L'ensemble de ces méthodes pour lesquelles l'algorithme s'adapte aux propriétés locales de l'image sont appelés adaptatifs.

Principe de fonctionnement :

L'algorithme suit la logique suivante :



1. Interpolation pixels vert :

Chaque pixel à interpoler pour la couleur vert est entouré de 4 pixels vert

	p1	
p2	X	p3
	p4	

- On définit **m = (p1+p2+p3+p4) / 4**
  Si **pi<m ==> L**
  Si **pi>m ==> H**
  Si **pi=m ==> E**

3 formes sont distinguées:

Contour

	H	
H	**X**	**L**
	H	

Bande

	L	
H	**X**	**H**
	L	

Coin

	H	
L	**X**	**H**
	L	

1.1 forme Contour :

G12 = **median**{G7, G11, G13, G17},
C-à-d, si on trie {G7, G11, G13, G17} comme A>B>C>D
alors
G12 = **median**{G7, G11, G13, G17} = (B + C) / 2

1.2 forme Bande :

un voisinage plus large est requis, comme le montre la figure suivante :

	A		**A**	
A		**L**		**A**
	H	**X**	**H**	
A		**L**		**A**
	A		**A**	

on cherche la valeur G du pixel centrale **X**. La valeur est calculée comme suite :

$$G_x = Clip_c^B(2M-S)$$

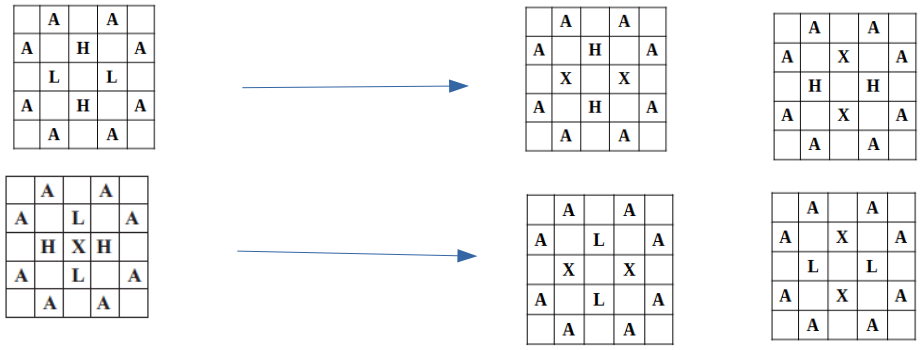
où

M = **median**{G7, G11, G13, G17}, , S = **SOMME**(A) / 8

Et '**clip**' est une fonction qui contraint la prédiction à être entre les valeurs B et C (de la median()).

- Si la prédiction est supérieur à B, elle est égale à B;
- Si la prédiction est inférieure à C, elle est égale à C.

avec la formes bande, on a 6 condition qui peuvent être remplis :

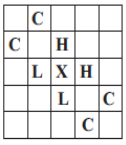


X : peut prende X=L ou X=E

X : peut prende X=H ou X=E

1.3 forme Coin:

un voisinage plus large est requis, comme montré sur la figure suivante :

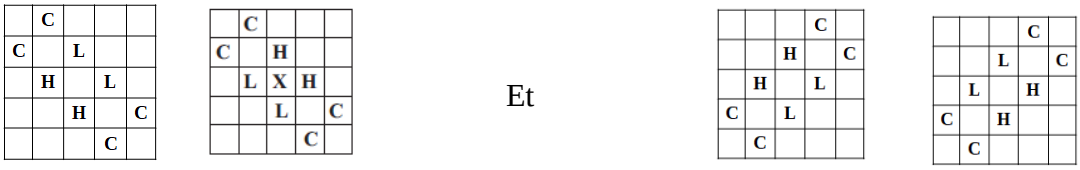


$$G_x = Clip_c^B(M - (S-M)/4)$$

où

M = **median**{H, L,} S = **SOMME**(C) / 4

dans cette forme on a 4 conditions à prendre en charge pour ces dispositions (patterns) :



Et

2. Interpolation des pixels rouge/bleu : toute méthode décrite avant peut faire l'affaire, (interpolation bilinéaire ou constance de la teinte)

Code :

On a le programme suivant qui execute la fonction de reconnaissance de formes :

```
%%% methode d'interpolation sous reconnaissance de formes:
function out = rec_form_inter(im_CFA_uint8, selecteur)
% si on oublie de caster l'argument input avant de l'injecter dans la fonction au debut,
%cette etape le fait. Transformer l'image de départ en double c'est pour ne pas avoir de problème de calcule:
    im_cfa = double(im_CFA_uint8);
    %extraie la Longueur , Largeur et nombre de bandes/canales de couleur:
    [H, W, k] = size(im_CFA_uint8);
    %créer cer variable pour ne pas ce tramper de bande dans les calcules:
    R = 1;  G=2;    B=3;
    %initier à zeros la variable qui servira d'image de stocage de
    %calcules:
    im_resultat_3 = zeros(H, W, k);
    %selecteur = 4: pour le seleceur afin de  choisir le case 4 du switch de green_ge()
    green_band = green_gen(im_cfa, selecteur);
    %%
    for i=2:H-1
       for j=2:W-1

    %%%%%%%%%%%%% pair:
           if(mod(i, 2) == 0 && mod(j, 2) == 0)  %%%pair/pair ----------------------------
                   % R7 / B7
                  im_resultat_3(i, j, R) = ( im_cfa(i - 1, j)+ im_cfa(i + 1, j)) / 2;
                  im_resultat_3(i, j, B) = ( im_cfa(i , j - 1)+ im_cfa(i , j + 1)) / 2;
                  im_resultat_3(i, j, G) =  im_cfa(i, j);
           elseif(mod(i, 2) == 0 && mod(j, 2) == 1) %%pair/impair---
                   % G8 /R8
                  im_resultat_3(i, j, R) = ( im_cfa(i - 1, j-1)+ im_cfa(i - 1, j+1) + im_cfa(i+1 , j-1)+ im_cfa(i+1, j+1)) / 4;
                  im_resultat_3(i, j, B) = im_cfa(i , j);
                  im_resultat_3(i, j, G) = green_band(i, j);

    %%%%%%%%%%  impair:
           elseif(mod(i, 2) == 1 && mod(j, 2) == 0)     %%%impair/pair-----------------------------------
                  %G12 / B12
                  im_resultat_3(i, j, B) = ( im_cfa(i - 1, j-1)+ im_cfa(i - 1, j+1) + im_cfa(i+1 , j-1)+ im_cfa(i+1, j+1)) / 4;
                  im_resultat_3(i, j, G) = green_band(i, j);
           else                                          %%%impair/impair ------------------------Pb B: reglé
                 % R13/B13
                  im_resultat_3(i, j, B) = ( im_cfa(i - 1, j)+ im_cfa(i + 1, j)) / 2;
                  im_resultat_3(i, j, R) = ( im_cfa(i , j - 1)+ im_cfa(i , j + 1)) / 2;
                  im_resultat_3(i, j, G) =  im_cfa(i, j);

           end
       end
    end
        out = uint8(im_resultat_3);
end
```

La partie réservée dans la fonction **green_gen()** pour générer la bande de pixels verts , se désigne comme suite :

```
case 4
        [H, W] = size(im_cfa);
        %initier la bande verte:
        green_band = double(im_cfa);
        for i=3:H-2
           for j=3:W-2
               %%%%%%%%%%%% %%% si: pair/impair OR impair/pair
               if( (mod(i, 2) == 0 && mod(j, 2) == 1) || (mod(i, 2) == 1 && mod(j, 2) == 0) )
                       %G8  / G12
                        kernel = [im_cfa(i-1 , j), im_cfa(i , j-1), im_cfa(i, j+1), im_cfa(i + 1, j)];
                        m = mean(kernel);
                        H = kernel > m;
                        L = kernel < m;
                        E = kernel == m;
                        count_H = sum(double(H));
                        count_L = sum(double(L));
                        count_E = sum(double(E));

                        %Si on a 3xH ou 3xL :     %cas: contour ----------------------------------------
                        if((count_H == 3) || (count_L == 3))
                           green_band(i, j) = median(kernel);

                        %Si on a 2xH ou 2xL opposés:  %cas: bande ----------------------------------------
                        elseif(( (H(1) == 1 && H(4) == 1) || (H(2) == 1 && H(3) == 1)) ||( (L(1) == 1 && L(4) == 1) || (L(2) == 1 && L(3) == 1)))
                            M = median(kernel); %(sum(kernel) / 4);
                            S = mean([im_cfa(i-2, j+1),  im_cfa(i-1, j+2),  im_cfa(i+1, j-2),  im_cfa(i+2, j-1), ...
                                  im_cfa(i-2, j-1),  im_cfa(i-1, j-2),  im_cfa(i+1, j+2),  im_cfa(i+2, j+1)]) ;
                            sorted_kernel = sort(kernel, "descend");
                            green_band(i, j) = clip((2*M - S), sorted_kernel(3), sorted_kernel(2)) ;

                            if(count_E == 1 || count_E == 2)
                                green_band(i, j) = median(kernel);
                            end

                        %Si on a 2xH et 2xL :     %cas: coin  ----------------------------------------
                        elseif(count_H == 2 && count_L == 2)

                            %première disposition:
                            if(   (H(3) == 1 && H(4) == 1) ||  (H(1) == 1 && H(2)) )
                                M = median(kernel);
                                S = mean([im_cfa(i-2, j+1),  im_cfa(i-1, j+2),  im_cfa(i+1, j-2),  im_cfa(i+2, j-1)]);
                                sorted_kernel = sort(kernel, "descend");
                                green_band(i, j) = clip((M - ((S - M)/4)), sorted_kernel(3), sorted_kernel(2));

                            %deuxième disposition:
                            elseif((H(1) == 1 && H(3) == 1) ||  (H(2) == 1 && H(4)) )
                                M = median(kernel);
                                S = mean([im_cfa(i-2, j-1),  im_cfa(i-1, j-2),  im_cfa(i+1, j+2),  im_cfa(i+2, j+1)]);
                                sorted_kernel = sort(kernel, "descend");
                                green_band(i, j) = clip((M - ((S - M)/4)), sorted_kernel(3), sorted_kernel(2));
                            end

                        else %disposition par defaut:
                            green_band(i, j) =  mean([im_cfa(i - 1, j), im_cfa(i + 1, j), im_cfa(i , j-1), im_cfa(i, j+1)]); %[green_band(i, j)]
                        end
               end
           end
        end
    %variable de sortie:
    out = green_band;
```

on retrouve dans cette partie du code les différente condition qu'on a traités lors de la partie **Principe de fonctionnement.** On ajoute une dernière condition qui sera traité si les dispositions précédentes n'aboutissent pas, cette condition est la dernière condition dans le code ci-dessus.

Affichage des résultats de la fonction :

On utilise cette partie du code pour verifier que notre programme fonctionne :

```
%% 5) interpolation par reconncaissance de formes:

%lecture de l'image:
im_RGB = imread('Demosaic5.tif');  %imread('vert.png');%
im_CFA = bayer(im_RGB);
%choisir selecteur = 2, pour avoir le deuxième 'case' du switch de la fonction green_gen():
selecteur = 4;
%appel de la fonction d'interpolation:
im_OUT = rec_form_inter(im_CFA , selecteur);
%affichage résusltat:
figure, imshowpair(im_RGB,im_OUT, 'montage');
```

on obtiens les résultats suivants :

on remarque que l'effet escalier diminue juste un peu par rapport à la methode bilinéaire. Le flou persiste toujours comme le montre la figure suivante :

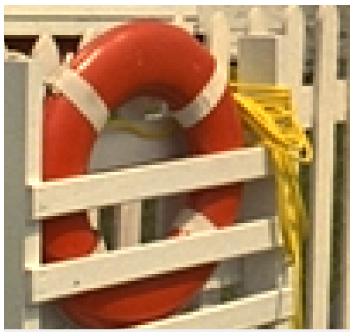


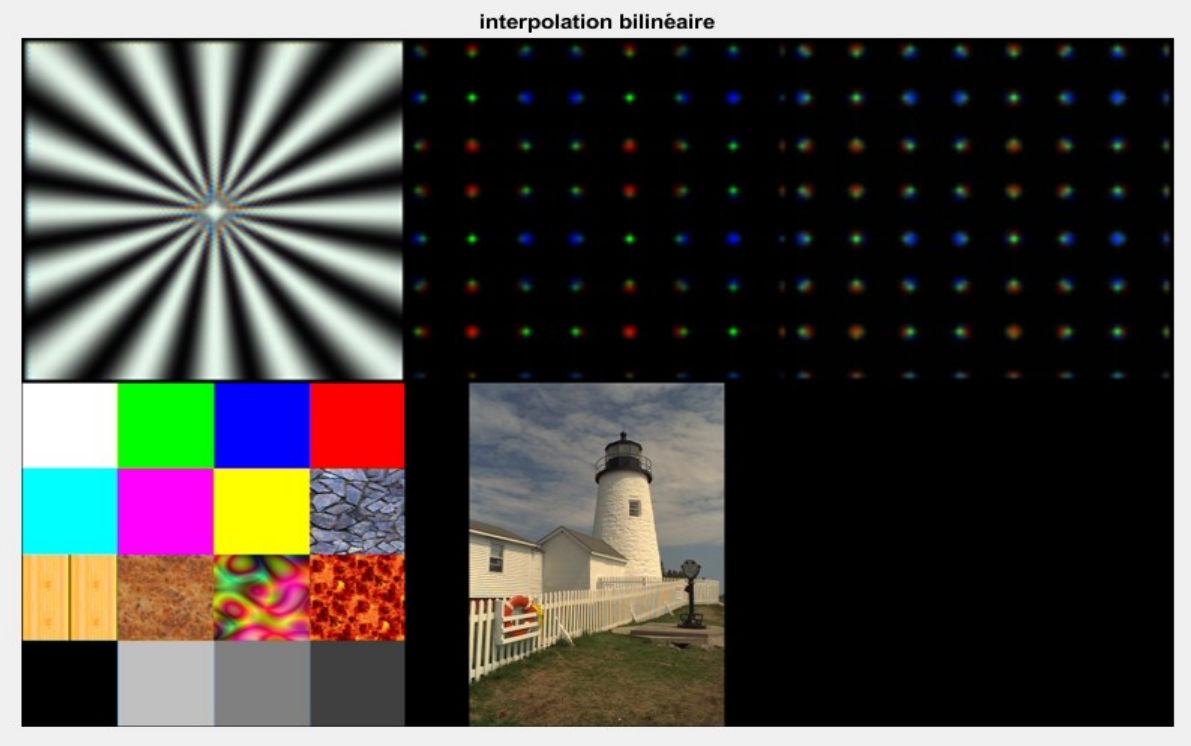
Figure 12: image originale

Figure 11: interpolation reconnaissance de formes

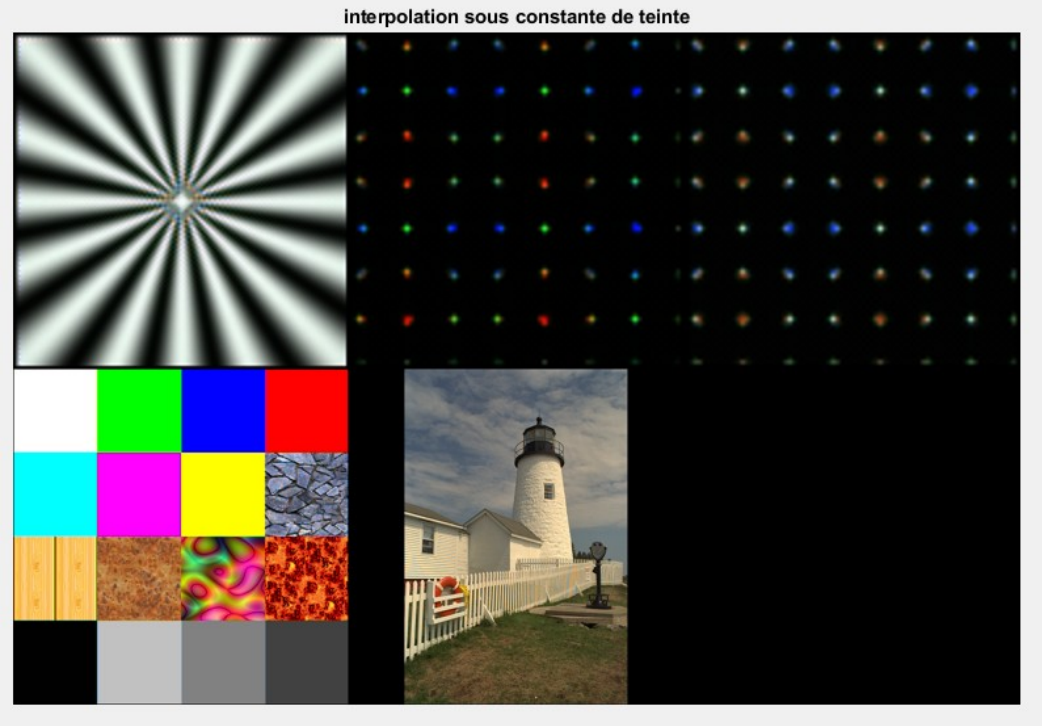
Partie III : Application à la reconstruction d'images

Dans cette partie on va testez nos algorithmes sur chacune des images test du répertoire *\Images* pour en générer les images couleur après démosaïcage.

1. bilinéaire interpolation :

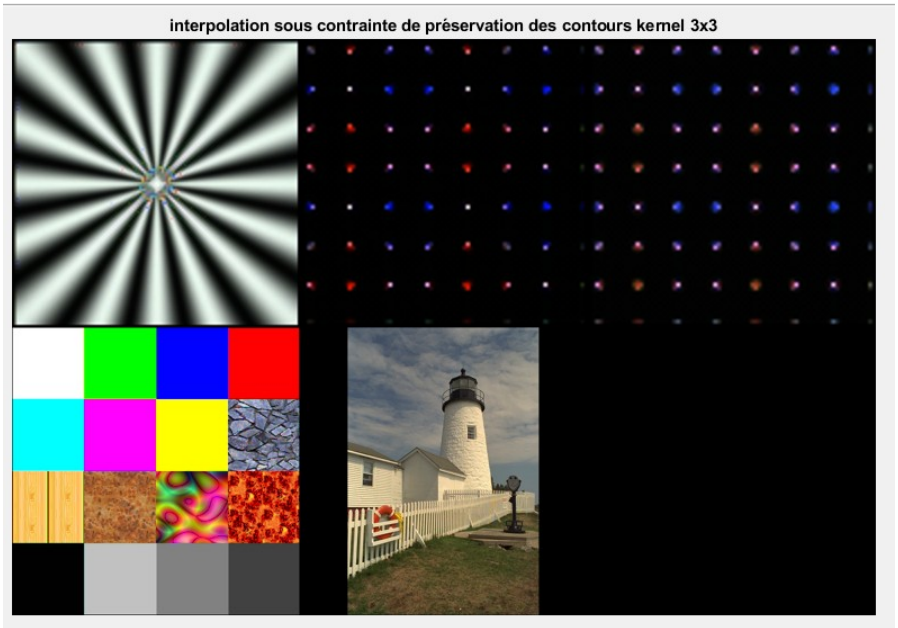


2. l'interpolation sous la contrainte de constance de la teinte :

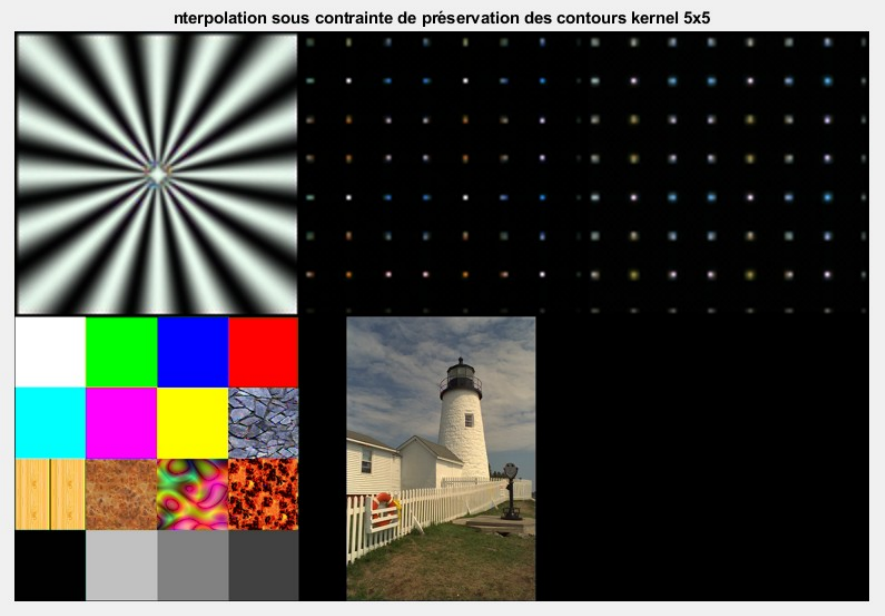


3. l'interpolation sous contrainte de préservation des contours :

- **méthode 1 avec un noyau 3x3 :**

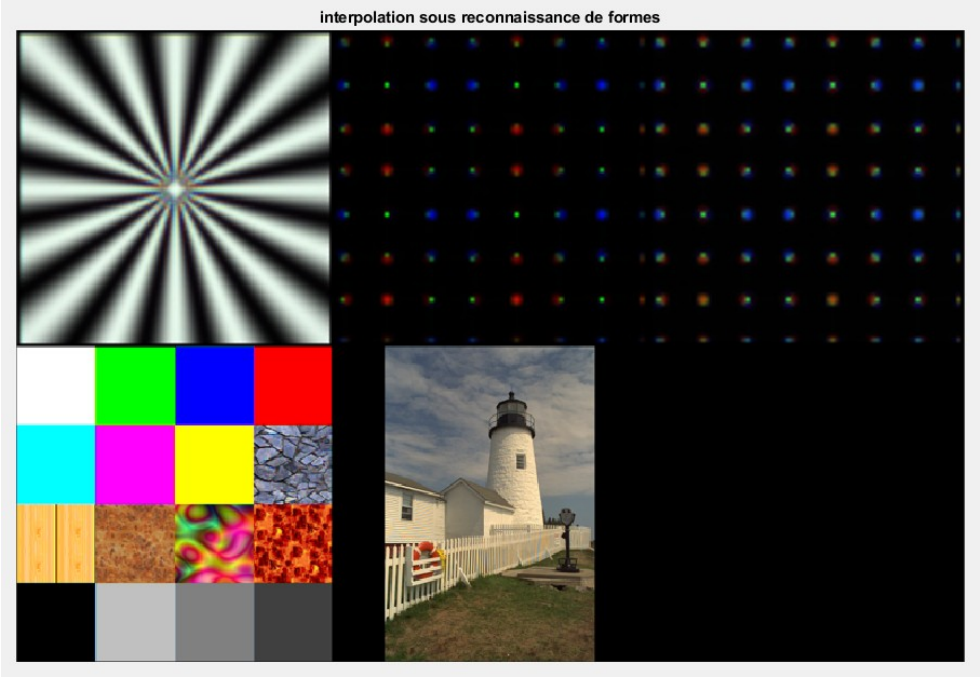


- **méthode 1 avec un noyau 5x5 :**



4. l'interpolation bilinéaire en utilisant un masque de convolution :

5. l'interpolation dédiée à la reconnaissance des formes:



Partie IV : évaluation quantitative

Les differentes métriques :

1. MSE Globale (Mean Square Error) :

Lors de cette metrique on cherche à calculer l'erreur totale qui est générée par nos fonctions d'interpolations, afin de mieux estimer la qualité de nos fonctions d'interpolation.

La fonction MSE globale se présente comme suite :

```
%% global MSE (mean squared error):
function Valeur = mse_global(im_org, im_gen)
    %%%convert images from uint8 to double, to get the calculus below right
    im_org = double(im_org);
    im_gen = double(im_gen);
    %déterminer les dimensions des images, qui sont les mêmes
    [H, W, k] = size(im_org);
    %calculer la différence mise au carrée entre les 3 bandes des images:
    diff_squared  = (im_org - im_gen).^2;
    %sommet toutes les erreurs des pixels et diviser par les dimensions:
    Valeur = sum(sum(sum(diff_squared))) / (W * H * 3); % somme de toutes les bandes
end
```

il faut noté qu'il existe une fonction prédéfinie de Matlab qui calcule la MSE , la fonction en question est **immse()**. Nos résultats avec notre fonction **mse_global()** donne le même résultat.

2. MSE Locale :

Dans cette fonction on fait appel à la fonction **mse_global()** réalisée précédemment, car le calcule est le même, il suffit de désigner la région d'intérêt (**ROI** : Region Of Interest) que nous voulons traiter.

Si l'utilisateur veut afficher cette région **ROI,** il suffit de mettre la variable booléenne **show_fig** égal à **1**.

```
%% MSE local: cette fonction fait appel à la fonction mse_global/
function Valeur =  mse_local(im_org, im_out, roi, show_fig)
    %Si nargin(Number of Arguments) est égale a 3, donc sans affichage:
    if(nargin == 3)
        %choisir la region qui nous interesse(ROI):
        croped_org = im_org(roi(2):(roi(2)+roi(4)), roi(1):(roi(1)+roi(3)), :);
        croped_gen = im_out(roi(2):(roi(2)+roi(4)), roi(1):(roi(1)+roi(3)), :);
        %utiliser la fonction calculée précédemment:
        Valeur = mse_global(croped_org , croped_gen);
   %Si on veut affiché l'image coupée, on met show_fig = true:
    elseif(nargin == 4)
        %choisir la region qui nous interesse(ROI):
        croped_org = im_org(roi(2):(roi(2)+roi(4)), roi(1):(roi(1)+roi(3)), :);
        croped_gen = im_out(roi(2):(roi(2)+roi(4)), roi(1):(roi(1)+roi(3)), :);
        %utiliser la fonction calculée précédemment:
        Valeur = mse_global(croped_org , croped_gen);

        if(show_fig == true)
            figure, imshowpair(croped_org, croped_gen, 'montage');
        end
    end
end
```

3. ΔE (Distance Euclidienne):

Distance Euclidienne proportionnelle à la perception.
La différence entre (L_1, a_1 , b_1) et (L_2, a_2 , b_2) est donnée par :

$$\Delta E^*_{ab} = \sqrt{(L^*_2 - L^*_1)^2 + (a^*_2 - a^*_1)^2 + (b^*_2 - b^*_1)^2}$$

la fonction se code comme suite :

```
function Valeur = delta_E_lab(im_org, im_gen)
    %initier les bande de l'espace Lab, pour mieu visualiser le code:
    L = 1;  a = 2; b = 3;
    %convertir de l'espace RGB vers Lab:
    im_org_lab = rgb2lab(im_org);
    im_gen_lab = rgb2lab(im_gen);
    %calcule de la distance chromatique:
    delta_E_mat = sqrt( (im_org_lab(:, :, L) -  im_gen_lab(:, :, L)).^2 +...
                        (im_org_lab(:, :, a) -  im_gen_lab(:, :, a)).^2 +...
                        (im_org_lab(:, :, b) -  im_gen_lab(:, :, b)).^2 );
    %sortie de la fonction
    Valeur = mean(mean(delta_E_mat));
end
```

4. Temps consacré au calculs:

Pour cette métrique on peut utiliser la fonction prédéfinie de Matlab **timeit()** avec en argument une **fonction handle**

pour la simplicité, on va utiliser dans notre cas la fonction **tic** et **toc.**

Application des métriques :

critère Méthode d'interpolation	subjectif	Métriques : MSE Global MSE Local ΔE	Temps écoulé [sec]
bilinéaire	Apparition des artefacts : flou, effet escalier, aliasing	189,33 / 529,34 / 5,00	0,23
Constante de teinte	-Les artefacts diminue far rapport a la bilinéaire -ringing plus important.	141,83 / 298,83 / 3,96	0,4
Préservation de contours noyau 3x3	-Artefact ondulant près des arêtes vives (effet Rining) causé par la perte d'info dans les hautes fréquences -Effet aliasing et flou diminue considérablement par rapport au autres méthodes	122,83 / 144,78 /3,49	0,4
Préservation de contours noyau 5x5	Même remarque que sur un noyau 3x3, avec l'aliasing et le flou diminuant encore.	112,10 / 67,18 / 2,97	0,46
Bilinéaire avec convolution	-On améliore un peu l'aliasing par rapport à la bilinéaire, mais le flou reste.	199,30 / 529,34 / 5,10	0,19
Reconnaissance de formes	-Même résultat que la bilinéaire en matiére de flou et aliasing -Améliore l'effet escalier et pas d'effet de ringing.	182,63 / 485,94 / 5,24	4,4

légende du tableau :

- rouge = valeur maximum par rapport à toutes les méthodes
- vert = valeur minimum par rapport à toutes les méthodes

Partie V : Discussion des résultats et conclusion:

Pour conclure, nous avons illustré avec nos exemples l'intaérét de prendre en compte des voisinages importants lors de l'interpolation. Le surplus de calcul engendré s'il est acceptable, permet d'obtenir une qualité d'interpolation bien meilleure que les interpolations bilinéaire. Ce gain de qualité est `a relativiser puisque comme les voisinages utilisés ne sont pas infinis comme le voudrait l'interpolation théoriquement idéale, l'artefact de ringing est introduit. Notons cependant que de nombreux algorithmes parviennent a posteriori retirer efficacement cet artefact, et que de nombreux post-traitement de ce genre sont présents dans les systémes d'interpolation industriels. De manière générale, toutes les méthodes présentées ici engendrent des artefacts plus ou moins prononcés. Selon le type d'images à interpoler et le domaine d'application, certains artefacts seront plus gênants que d'autres. Le choix de l'algorithme est donc un compromis entre le temps qu'il est possible d'allouer à l'interpolation et les artefacts qui vont être introduits.