Semaine 11 : Acquisition d'images couleur et dématriçage

TP11

Maxime CATTEAU
Léane TEXIER

1ère partie : Interprétation et simulation d'une image CFA

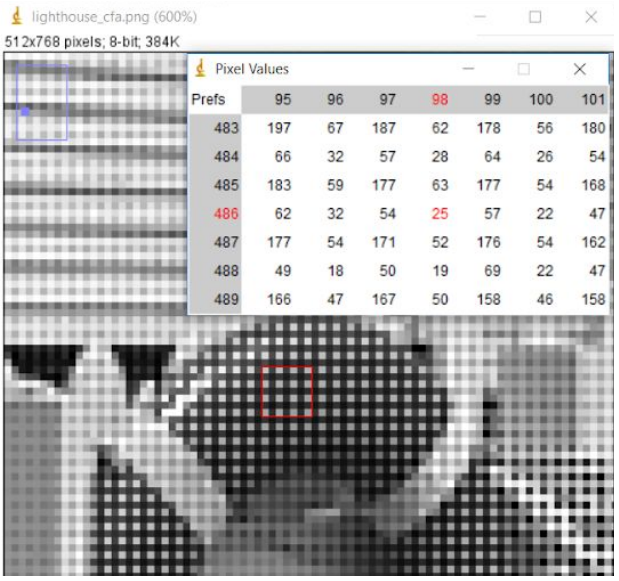
Interprétation d'une image CFA

Quelle est la disposition du filtre CFA qui a été utilisée pour générer l'image CFA ci-dessus (donner les 3x3 premiers filtres, de coordonnées $(x,y)\in\{0,1,2\}^2$) ?

Afin de trouver la disposition du filtre CFA qui a été utilisé pour générer l'image CFA donnée, nous avons tout d'abord effectué la suite du TP qui permet suivant une image de base d'obtenir l'image CFA suivant la disposition du filtre donné. Nous avons alors comparé les images obtenues par les différentes dispositions de filtres avec l'image de base. Nous avons ainsi pu remarquer que la disposition du filtre utilisé dans le cas présent est le suivant (3x3 premiers filtres de coordonnées $(x,y) \in \{0, 1, 2\}^2$) :

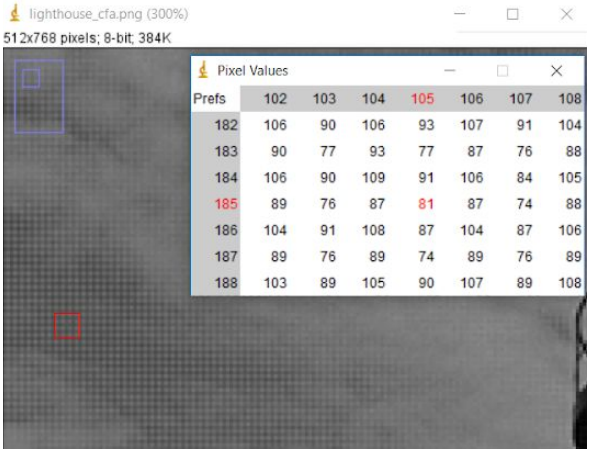
B	G	B
G	R	G
B	G	B

Une autre solution afin de trouver la disposition du filtre CFA utilisé est d'explorer la valeur des pixels grâce à l'outil ''Pixel Inspector'' proposé par ImageJ. En effet, nous pouvons, par exemple, remarquer sur l'image en couleur que la bouée est rouge. En analysant ces pixels sur l'image CFA obtenue, nous pourrons alors trouver les coordonnées des pixels où le niveau de rouge a été enregistré car la valeur du pixel sera élevée étant donné que le niveau de rouge sur l'image de base est élevé. Analysons alors ces pixels :



Analyse avec ‘’Pixel Inspector’’ des pixels située dans la zone rouge (= partie de la bouée)

Si nous regardons les pixels dont le niveau de gris est le plus élevé, nous remarquons que ce sont les pixels de la forme (2w+1, 2h+1) avec w un entier compris entre 47 et 50, et h un entier compris entre 241 et 244. Nous pouvons ainsi en déduire que ces pixels sont ceux où c’est le niveau de rouge qui a été enregistré. Grâce à cela, nous retrouvons la disposition du filtre énoncé ci-dessus (B-G-B). Nous pouvons confirmer cela en analysant d’autres pixels comme par exemple une partie du ciel qui est bleu.



Analyse avec ‘Pixel Inspector’ des pixels située dans le ciel bleu

Si on regarde les pixels dont le niveau de gris est le plus élevé, on remarque que ce sont les pixels de la forme (2w, 2h) avec w un entier compris entre 51 et 54, et h un entier compris entre 91 et 94. On peut ainsi en déduire que ces pixels sont ceux où c’est le niveau de bleu qui a été enregistré. Cela confirme donc bien la disposition du filtre CFA trouvé précédemment.

Une autre solution pour trouver la disposition du filtre est de comparer les valeurs des pixels (R, G, B) de l’image couleur avec les valeurs mêmes pixels de l’image donnée.

Prefs	0	1	2	3	4	5	6
0	75,93,94	78,95,104	76,92,107	81,94,110	86,100,115	84,97,112	84,97,112
1	75,93,94	76,93,102	74,90,104	77,90,106	80,93,108	82,96,111	86,100,115
2	78,94,96	80,94,102	77,90,106	77,90,106	80,93,108	82,96,111	88,99,115
3	86,102,104	80,94,102	76,89,102	77,90,106	78,94,108	81,94,110	87,98,114
4	88,103,105	81,95,103	77,90,106	78,91,107	78,91,107	76,89,104	83,94,110
5	89,105,104	82,97,104	78,91,107	74,87,103	77,90,103	78,92,105	81,95,107
6	87,100,100	88,100,108	80,91,107	81,94,110	77,90,106	78,91,107	81,94,110

Tableau des valeurs des pixels de l'image couleur (R, G, B)

Prefs	0	1	2	3	4	5	6
0	94	95	107	94	115	97	112
1	93	76	90	77	93	82	100
2	96	94	106	90	108	96	115
3	102	80	89	77	94	81	98
4	105	95	106	91	107	89	110
5	105	82	91	74	90	78	95
6	100	100	107	94	106	91	110

Tableau des valeurs des pixels de l'image cfa donnée

Le premier tableau nous donne les valeurs de la forme (R, G, B) des premiers 6x6 pixels de l'image originale en couleur. Le second tableau nous donne les valeurs en niveau de gris des premiers 6x6 pixels de l'image CFA donnée. Grâce à cela, on remarque que les pixels où le niveau de bleu de l'image de base ont été enregistrés dans l'image CFA sont les pixels de la forme (2w, 2h). Les pixels où le niveau de rouge de l'image de base ont été enregistrés dans l'image CFA sont les pixels de la forme (2w+1, 2h+1). Les pixels où le niveau de vert de l'image de base ont été enregistrés dans l'image CFA sont les pixels de la forme (2w+1, 2h) ou de la forme (2w,2h+1).
On pose w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Grâce à cette analyse, on peut également retrouvé la disposition du filtre appliqué (B-G-B).

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

Compléter la méthode **run** pour générer puis afficher l'image CFA, en ne considérant que la disposition G-R-G dans un premier temps. Dans le compte-rendu, décrire les lignes ajoutées et présenter le résultat obtenu en le justifiant.

Pour générer l'image CFA obtenue en considérant la disposition G-R-G et l'afficher, nous avons codé la fonction **setup** et nous avons complété la méthode **run** donnée.

Dans la méthode **setup**, nous initialisons notre attribut ImagePlus grâce à celui donné en paramètre et nous retournons un entier indiquant que l'image de base est une image en RGB. Voici le code de cette fonction :

```
public int setup(String arg, ImagePlus imp) {
	this.imp = imp;
	return DOES_RGB;
}
```

*Code de la fonction **setup***

Dans la méthode run, nous avons ajouté à la suite des lignes déjà présentes l'appel à la fonction *cfa* qui se charge de générer l'image CFA (dans ce cas sans prendre en compte le paramètre ordre car c'est forcément G-R-G). Puis nous créons une image de type ImagePlus suivant l'image CFA obtenue et nous l'affichons.

```
public void run(ImageProcessor ip) {
	...
	// Génération de l'image CFA
	ImageProcessor imageCFA = this.cfa(order);
	ImagePlus res = new ImagePlus("CFA result", imageCFA);
	res.show();
}
```

*Partie de code ajouté à la fonction **run***

Nous avons ensuite appelé ce plugin sur l'image de phares. Voici le résultat ainsi obtenu :

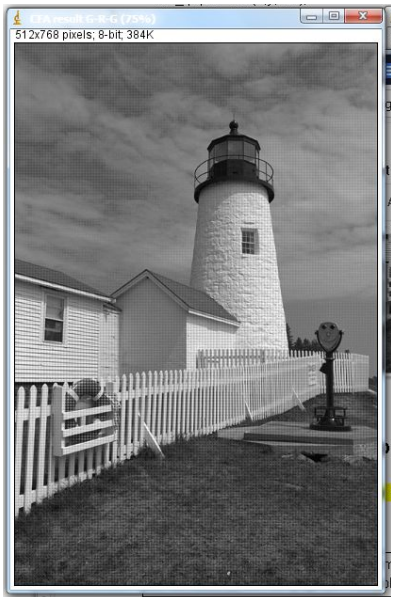
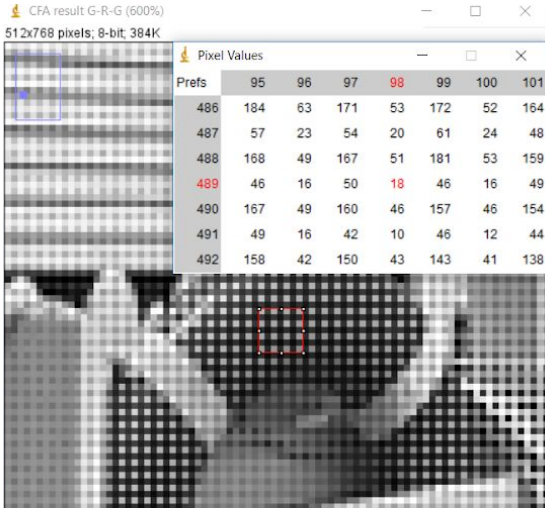


Image CFA obtenue avec la disposition G-R-G

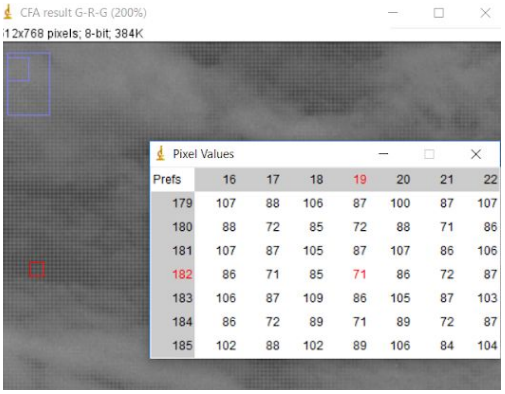
Si on examine de plus près l'image on peut remarquer que le filtre de disposition G-R-G a bien été appliqué. En effet, comme précédemment analysons les pixels de la bouée (rouge).



Prefs	95	96	97	98	99	100	101
486	184	63	171	53	172	52	164
487	57	23	54	20	61	24	48
488	168	49	167	51	181	53	159
489	46	16	50	18	46	16	49
490	167	49	160	46	157	46	154
491	49	16	42	10	46	12	44
492	158	42	150	43	143	41	138

Analyse avec 'Pixel Inspector' des pixels située dans la bouée (rouge)

Si on regarde les pixels dont le niveau de gris est le plus élevé, on remarque que ce sont les pixels de la forme (2w+1, 2h) avec w compris entre 47 et 50, et h compris entre 243 et 246. On peut ainsi en déduire que ces pixels sont ceux où c'est le niveau de rouge qui a été enregistré étant donné que c'est un endroit qui est rouge de base. Cela confirme la disposition du filtre appliqué.



Prefs	16	17	18	19	20	21	22
179	107	88	106	87	100	87	107
180	88	72	85	72	88	71	86
181	107	87	105	87	107	86	106
182	86	71	85	71	86	72	87
183	106	87	109	86	105	87	103
184	86	72	89	71	89	72	87
185	102	88	102	89	106	84	104

Analyse avec 'Pixel Inspector' des pixels située dans le ciel bleu

Si on regarde les pixels dont le niveau de gris est le plus élevé, on remarque que ce sont les pixels de la forme (2w, 2h+1) avec w compris entre 8 et 11, et h compris entre 89 et 92. On peut ainsi en déduire que ces pixels sont ceux où c'est le niveau de bleu qui a été enregistré étant donné que c'est un endroit qui est bleu de base. Cela confirme la disposition du filtre appliqué.

On peut également, comme précédemment, comparer les valeurs des pixels (R, G, B) de l'image couleur avec les valeurs mêmes pixels de l'image obtenue.

Prefs	0	1	2	3	4	5	6
0	75,93,94	78,95,104	76,92,107	81,94,110	86,100,115	84,97,112	84,97,112
1	75,93,94	76,93,102	74,90,104	77,90,106	80,93,108	82,96,111	86,100,115
2	78,94,96	80,94,102	77,90,106	77,90,106	80,93,108	82,96,111	88,99,115
3	86,102,104	80,94,102	76,89,102	77,90,106	78,94,108	81,94,110	87,98,114
4	88,103,105	81,95,103	77,90,106	78,91,107	78,91,107	76,89,104	83,94,110
5	89,105,104	82,97,104	78,91,107	74,87,103	77,90,103	78,92,105	81,95,107
6	87,100,100	88,100,108	80,91,107	81,94,110	77,90,106	78,91,107	81,94,110

*Tableau des valeurs des pixels de l'image couleur (*R, G, B*)*

Prefs	0	1	2	3	4	5	6
0	93	78	92	81	100	84	97
1	94	93	104	90	108	96	115
2	94	80	90	77	93	82	99
3	104	94	102	90	108	94	114
4	103	81	90	78	91	76	94
5	104	97	107	87	103	92	107
6	100	88	91	81	90	78	94

Tableau des valeurs des pixels de l'image cfa obtenue

Grâce à ces différents tableaux, on remarque que les pixels où le niveau de bleu de l'image de base ont été enregistrés dans l'image CFA sont les pixels de la forme (2w, 2h+1). Les pixels où le niveau de rouge de l'image de base ont été enregistrés dans l'image CFA sont les pixels de la forme (2w+1, 2h). Les pixels où le niveau de vert de l'image de base ont été enregistrés dans l'image CFA sont les pixels de la forme (2w, 2h) ou de la forme (2w+1,2h+1).
On pose w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Grâce à cette analyse, on remarque alors que notre filtre de disposition G-R-G a bien été appliqué.

Compléter la méthode **cfa** pour qu'elle prenne en compte les quatre dispositions possibles de CFA, et adapter la méthode **run** en conséquence.

Afin de pouvoir générer l'image CFA suivant la disposition souhaitée, nous devons dans la méthode **cfa** prendre en compte le paramètre row_order donné et effectuer le traitement souhaité en fonction. Étant donné que dans la méthode **run** complétée précédemment nous appelions déjà notre méthode **cfa** en mettant en paramètre l'ordre choisi par l'utilisateur, nous ne devons alors pas modifier la fonction **run** dans cette partie mais seulement la méthode **cfa**. Pour rappel :

- ordre 0 correspond à la disposition R-G-R
- ordre 1 correspond à la disposition B-G-B
- ordre 2 correspond à la disposition G-R-G
- ordre 3 correspond à la disposition G-B-G

On peut ainsi remarquer que dans les cas où l'ordre est à 2 (= disposition G-R-G) ou à 3 (= disposition G-B-G), la disposition des G sont les mêmes. Cela signifie que dans ces deux cas, les pixels où nous enregistrons le niveau de vert de l'image initiale dans notre image CFA sont situés sur les coordonnées de la forme (2w, 2h) ou de la forme (2w+1, 2h+1) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

On peut aussi remarquer que dans les cas où l'ordre est à 0 (= disposition R-G-R) ou à 1 (= disposition B-G-B), la disposition des G sont les mêmes. Cela signifie que dans ces deux cas, les pixels où nous enregistrons le niveau de vert de l'image initiale dans notre image CFA sont situés sur les coordonnées de la forme (2w, 2h+1) ou de la forme (2w+1, 2h) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Voici le code permettant d'enregistrer le niveau de vert des pixels concernés dans notre image cfa (suivant l'ordre) :

```
if(row_order >1){
        for (int y=0; y<height; y+=2) {
                for (int x=0; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int green = (int)(pixel_value & 0x00ff00)>>8;
                        cfa_ip.putPixel(x,y,green);
                }
        }
         for (int y=1; y<height; y+=2) {
                 for (int x=1; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int green = (int)(pixel_value & 0x00ff00)>>8;
                        cfa_ip.putPixel(x,y,green);
                }
         }
}else{
         for (int y=0; y<height; y+=2) {
                 for (int x=1; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int green = (int)(pixel_value & 0x00ff00)>>8;
                        cfa_ip.putPixel(x,y,green);
                }
        }
         for (int y=1; y<height; y+=2) {
                 for (int x=0; x<width; x+=2) {
                         pixel_value = ip.getPixel(x,y);
                         int green = (int)(pixel_value & 0x00ff00)>>8;
                         cfa_ip.putPixel(x,y,green);
                }
        }

}
```

*Partie de code mettant à jour les pixels dont le niveau de vert doit être enregistrée dans l'image cfa suivant row_order ajouté à la fonction **cfa***

Pour l'enregistrement des niveaux de rouge des pixels souhaités, nous avons remarqué que chaque disposition (donc chaque ordre) enregistre le niveau de rouge de pixels différents. Nous avons alors étudié les 4 cas.

Dans le cas de l'ordre 0 (= disposition R-G-R), nous avons remarqué que les pixels où nous enregistrons le niveau de rouge de l'image initiale dans notre image CFA sont situés sur les coordonnées de la forme (2w, 2h) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Dans le cas de l'ordre 1 (= disposition B-G-B), nous avons remarqué que les pixels où nous enregistrons le niveau de rouge de l'image initiale dans notre image CFA sont situés sur les coordonnées de la forme (2w+1, 2h+1) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Dans le cas de l'ordre 2 (= disposition G-R-G), nous avons remarqué que les pixels où nous enregistrons le niveau de rouge de l'image initiale dans notre image CFA sont situés sur les coordonnées de la forme (2w+1, 2h) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Dans le cas de l'ordre 3 (= disposition G-B-G), nous avons remarqué que les pixels où nous enregistrons le niveau de rouge de l'image initiale dans notre image CFA sont situés sur les coordonnées de la forme (2w, 2h+1) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Voici le code permettant d'enregistrer le niveau de rouge des pixels concernés dans notre image cfa (suivant l'ordre) :

```
if(row_order==0){
        for (int y=0; y<height; y+=2) {
                for (int x=0; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int red = (int)(pixel_value & 0xff0000)>>16;
                        cfa_ip.putPixel(x,y,red);
                }
        }
}else if(row_order==1){
        for (int y=1; y<height; y+=2) {
                for (int x=1; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int red = (int)(pixel_value & 0xff0000)>>16;
                        cfa_ip.putPixel(x,y,red);
                }
        }
}else if(row_order==2){
        for (int y=0; y<height; y+=2) {
                for (int x=1; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int red = (int)(pixel_value & 0xff0000)>>16;
                        cfa_ip.putPixel(x,y,red);
                }
        }
}else{
        for (int y=1; y<height; y+=2) {
                for (int x=0; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int red = (int)(pixel_value & 0xff0000)>>16;
                        cfa_ip.putPixel(x,y,red);
                }
        }
}
```

*Partie de code mettant à jour les pixels dont le niveau de rouge doit être enregistrée dans l'image cfa suivant row_order ajouté à la fonction **cfa***

Pour l'enregistrement des niveaux de bleu des pixels souhaités, nous avons remarqué que chaque disposition (donc chaque ordre) enregistre le niveau de bleu de pixels différents. Nous avons alors étudié les 4 cas.

Dans le cas de l'ordre 0 (= disposition R-G-R), nous avons remarqué que les pixels où nous enregistrons le niveau de bleu de l'image initiale dans notre image CFA sont situés sur les coordonnées de la forme (2w+1, 2h+1) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Dans le cas de l'ordre 1 (= disposition B-G-B), nous avons remarqué que les pixels où nous enregistrons le niveau de bleu de l'image initiale dans notre image CFA sont situés sur les coordonnées de la forme (2w, 2h) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Dans le cas de l'ordre 2 (= disposition G-R-G), nous avons remarqué que les pixels où nous enregistrons le niveau de bleu de l'image initiale dans notre image CFA sont situés sur les coordonnées de la forme (2w, 2h+1) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Dans le cas de l'ordre 3 (= disposition G-B-G), nous avons remarqué que les pixels où nous enregistrons le niveau de bleu de l'image initiale dans notre image CFA sont situés sur les coordonnées de la forme (2w+1, 2h) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

Voici le code permettant d'enregistrer le niveau de bleu des pixels concernés dans notre image cfa (suivant l'ordre) :

```
if(row_order==0){
        for (int y=1; y<height; y+=2) {
                for (int x=1; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int blue = (int)(pixel_value & 0x0000ff);
                        cfa_ip.putPixel(x,y,blue);
                }
        }
}else if(row_order==1){
        for (int y=0; y<height; y+=2) {
                for (int x=0; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int blue = (int)(pixel_value & 0x0000ff);
                        cfa_ip.putPixel(x,y,blue);
                }
        }
}else if(row_order==2){
        for (int y=1; y<height; y+=2) {
                for (int x=0; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int blue = (int)(pixel_value & 0x0000ff);
                        cfa_ip.putPixel(x,y,blue);
                }
        }
}else {
        for (int y=0; y<height; y+=2) {
                for (int x=1; x<width; x+=2) {
                        pixel_value = ip.getPixel(x,y);
                        int blue = (int)(pixel_value & 0x0000ff);
                        cfa_ip.putPixel(x,y,blue);
                }
        }
}
```

Partie de code mettant à jour les pixels dont le niveau de bleu doit être enregistrée dans l'image cfa suivant row_order ajouté à la fonction ***cfa***

Nous avons alors appelé ce plugin sur l'image de phare avec les différentes dispositions possibles. Voici les résultats ainsi obtenu (nous n'avons pas mis le résultat avec la disposition G-R-G car c'est la même que précédemment) :

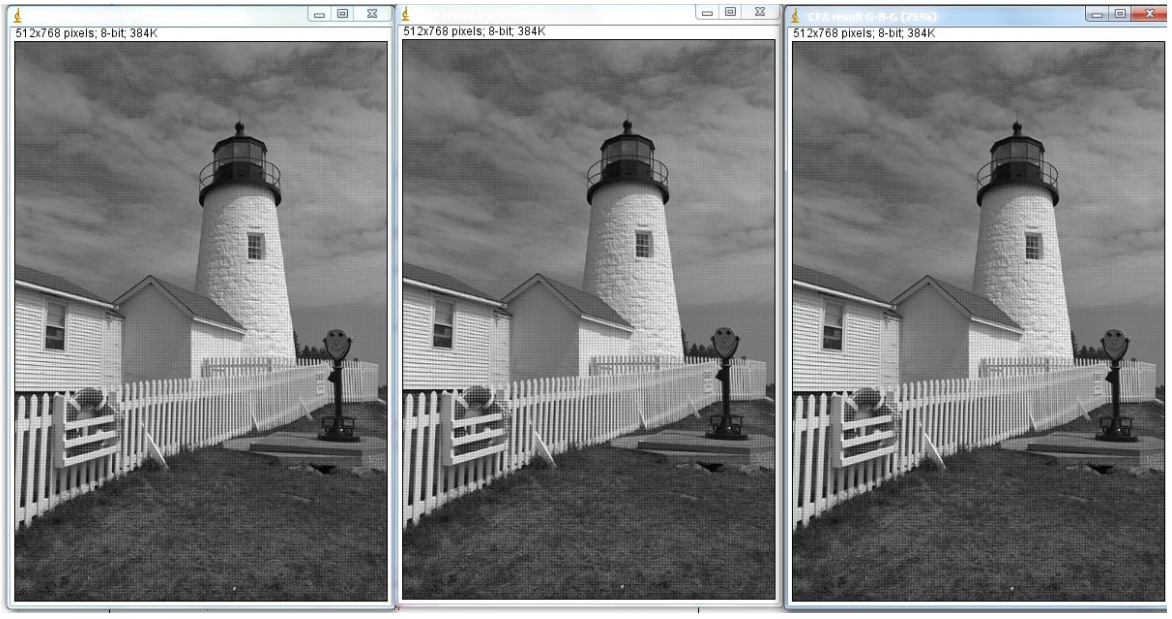


Image CFA obtenue avec les dispositions (de gauche à droite) R-G-R, B-G-B, G-B-G

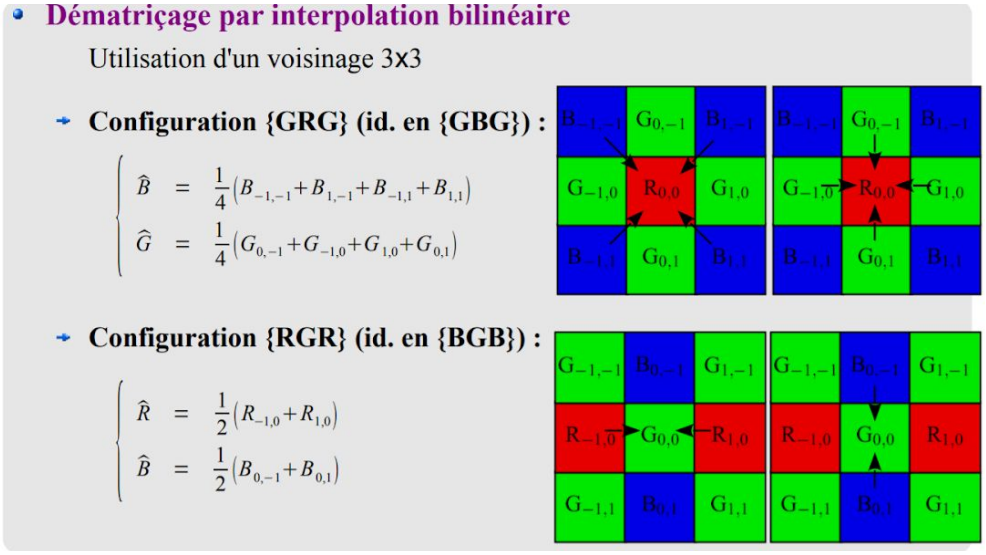
Afin de confirmer que notre code applique bien la disposition de filtre choisie par l'utilisateur, nous avons pour chaque image obtenue, effectué les mêmes analyses que précédemment (études de pixels de la bouée (rouge), études de pixels du ciel bleu et comparaisons des valeurs des 6x6 premiers pixels avec les valeurs de ces pixels de l'image originale en couleur). Cela nous a permis de vérifier la fiabilité de notre code.

2ème partie : Dématriçage par interpolation bilinéaire

Présentation

Expliquer pourquoi (et comment) ces masques de convolution permettent bien de retrouver les formules de dématriçage par interpolation bilinéaire vues en cours.

Tout d'abord, rappelons les formules de dématriçage par interpolation bilinéaire données en cours :



Considérons, tout d'abord, le plan φG :

G	0	G	0	G
0	G	0	G	0
G	0	G	0	G
0	G	0	G	0
G	0	G	0	G

Grâce à ce plan et au cours, on sait qu'il y a deux possibilités pour avoir le niveau de vert (G) soit :

- le pixel analysé contient ce niveau (configuration {RGR} ou {BGB})
- il faut faire la moyenne des 4 pixels qui sont autour de lui (cf formule du cours ci-dessus) qui contiennent le niveau de vert (configuration {GRG} ou {GBG})

Les deux dispositions possibles pour G sont respectivement :

G	0	G
0	G	0
G	0	G

0	G	0
G	0	G
0	G	0

Soit le masque suivant :

0	¼	0
¼	1	¼
0	¼	0

Si on applique la convolution par ce masque sur le pixel central du premier cas, on veut alors que la valeur retournée soit celle du pixel central i.e. $G_{0,0}$. Faisons le calcul, on a alors :

$G_{-1,-1} * 0 + G_{0,-1} * ¼ + G_{1,-1} * 0 + G_{-1,0} * ¼ + G_{0,0} * 1 + G_{1,0} * ¼ + G_{-1,1} * 0 + G_{0,1} * ¼ + G_{1,1} * 0$
$= G_{0,-1} * ¼ + G_{-1,0} * ¼ + G_{0,0} * 1 + G_{1,0} * ¼ + G_{0,1} * ¼$
$= 0 * ¼ + 0 * ¼ + G_{0,0} + 0 * ¼ + 0 * ¼$
$= G_{0,0}$

On obtient bien la valeur souhaitée. Étudions maintenant le second cas. Si on applique la convolution par ce masque sur le pixel central du second cas, on veut alors que la valeur retournée soit la moyenne des 4 pixels qui sont autour de lui (i.e. $¼ (G_{0,-1} + G_{-1,0} + G_{1,0} + G_{0,1})$ => formule du cours). Faisons le calcul, on a alors :

$G_{-1,-1} * 0 + G_{0,-1} * ¼ + G_{1,-1} * 0 + G_{-1,0} * ¼ + G_{0,0} * 1 + G_{1,0} * ¼ + G_{-1,1} * 0 + G_{0,1} * ¼ + G_{1,1} * 0$
$= G_{0,-1} * ¼ + G_{-1,0} * ¼ + G_{0,0} * 1 + G_{1,0} * ¼ + G_{0,1} * ¼$
$= G_{0,-1} * ¼ + G_{-1,0} * ¼ + 0 * 1 + G_{1,0} * ¼ + G_{0,1} * ¼$
$= G_{0,-1} * ¼ + G_{-1,0} * ¼ + G_{1,0} * ¼ + G_{0,1} * ¼$
$= ¼ (G_{0,-1} + G_{-1,0} + G_{1,0} + G_{0,1})$

On obtient bien la valeur souhaitée. On peut donc en déduire que le masque de convolution pour G permet bien de retrouver les formules de dématriçage par interpolation bilinéaire données en cours.

Considérons, maintenant, le plan φR :

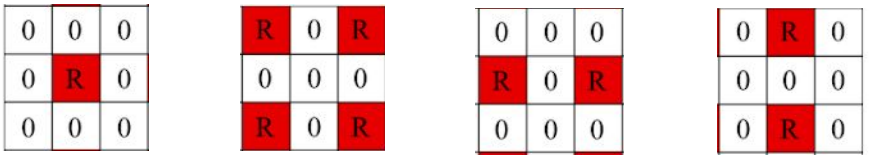
0	R	0	R	0
0	0	0	0	0
0	R	0	R	0
0	0	0	0	0
0	R	0	R	0

Grâce à ce plan et au cours, on sait qu'il y a quatre possibilités pour avoir le niveau de rouge (R) soit :

- on est dans la configuration {GRG} et dans ce cas le pixel analysé contient ce niveau
- on est dans la configuration {GBG} et dans ce cas on doit faire la moyenne des 4 pixels diagonaux (cf formule du cours ci-dessus) qui contiennent le niveau de rouge

- on est dans la configuration {RGR} et dans ce cas on doit faire la moyenne des 2 pixels horizontaux voisins (cf formule du cours ci-dessus) qui contiennent le niveau de rouge
- on est dans la configuration {BGB} et dans ce cas on doit faire la moyenne des 2 pixels verticaux voisins (cf formule du cours ci-dessus) qui contiennent le niveau de rouge

Les quatres dispositions possibles pour R sont respectivement :



Soit le masque suivant :

¼	½	¼
½	1	½
¼	½	¼

Si on applique la convolution par ce masque sur le pixel central du premier cas, on veut alors que la valeur retournée soit celle du pixel central i.e. $R_{0,0}$. Faisons le calcul, on a alors :

$R_{-1,-1} * ¼ + R_{0,-1} * ½ + R_{1,-1} * ¼ + R_{-1,0} * ½ + R_{0,0} * 1 + R_{1,0} * ½ + R_{-1,1} * ¼ + R_{0,1} * ½ + R_{1,1} * ¼$
$= 0 * ¼ + 0 * ½ + 0 * ¼ + 0 * ½ + R_{0,0} * 1 + 0 * ½ + 0 * ¼ + 0 * ½ + 0 * ¼$
$= R_{0,0}$

On obtient bien la valeur souhaitée. Étudions maintenant le deuxième cas. Si on applique la convolution par ce masque sur le pixel central du deuxième cas, on veut alors que la valeur retournée soit la moyenne des 4 pixels diagonaux (i.e. ¼ ($R_{-1,-1} + R_{1,-1} + R_{-1,1} + R_{1,1}$) => formule du cours). Faisons le calcul, on a alors :

$R_{-1,-1} * ¼ + R_{0,-1} * ½ + R_{1,-1} * ¼ + R_{-1,0} * ½ + R_{0,0} * 1 + R_{1,0} * ½ + R_{-1,1} * ¼ + R_{0,1} * ½ + R_{1,1} * ¼$
$= R_{-1,-1} * ¼ + 0 * ½ + R_{1,-1} * ¼ + 0 * ½ + 0 * 1 + 0 * ½ + R_{-1,1} * ¼ + 0 * ½ + R_{1,1} * ¼$
$= R_{-1,-1} * ¼ + R_{1,-1} * ¼ + R_{-1,1} * ¼ + R_{1,1} * ¼$
$= ¼ (R_{-1,-1} + R_{1,-1} + R_{-1,1} + R_{1,1})$

On obtient bien la valeur souhaitée. Étudions maintenant le troisième cas. Si on applique la convolution par ce masque sur le pixel central du troisième cas, on veut alors que la valeur retournée soit la moyenne des 2 pixels horizontaux voisins (i.e. ½ ($R_{-1,0} + R_{1,0}$) => formule du cours). Faisons le calcul, on a alors :

$R_{-1,-1} * ¼ + R_{0,-1} * ½ + R_{1,-1} * ¼ + R_{-1,0} * ½ + R_{0,0} * 1 + R_{1,0} * ½ + R_{-1,1} * ¼ + R_{0,1} * ½ + R_{1,1} * ¼$
$= 0 * ¼ + 0 * ½ + 0 * ¼ + R_{-1,0} * ½ + 0 * 1 + R_{1,0} * ½ + 0 * ¼ + 0 * ½ + 0 * ¼$
$= R_{-1,0} * ½ + R_{1,0} * ½$
$= ½ (R_{-1,0} + R_{1,0})$

On obtient bien la valeur souhaitée. Étudions maintenant le quatrième et dernier cas. Si on applique la convolution par ce masque sur le pixel central du quatrième cas, on veut alors que la valeur retournée soit la moyenne des 2 pixels verticaux voisins (i.e. $½ (R_{0,-1} + R_{0,1})$ => formule du cours). Faisons le calcul, on a alors :

$R_{-1,-1} * ¼ + R_{0,-1} * ½ + R_{1,-1} * ¼ + R_{-1,0} * ½ + R_{0,0} * 1 + R_{1,0} * ½ + R_{-1,1} * ¼ + R_{0,1} * ½ + R_{1,1} * ¼$
$= 0 * ¼ + R_{0,-1} * ½ + 0 * ¼ + 0 * ½ + 0 * 1 + 0 * ½ + 0 * ¼ + R_{0,1} * ½ + 0 * ¼$
$= R_{0,-1} * ½ + R_{0,1} * ½$
$= ½ (R_{0,-1} + R_{0,1})$

On obtient bien la valeur souhaitée. On peut donc en déduire que le masque de convolution pour R permet bien de retrouver les formules de dématriçage par interpolation bilinéaire données en cours.

Considérons, maintenant, le plan φB :

0	0	0	0	0
B	0	B	0	B
0	0	0	0	0
B	0	B	0	B
0	0	0	0	0

Grâce à ce plan et au cours, on sait qu'il y a quatre possibilités pour avoir le niveau de bleu (B) comme c'est le cas pour le plan rouge. La démonstration est similaire à celle du plan rouge étant donné que le masque est également le même. On peut donc en déduire que le masque de convolution pour B permet bien de retrouver les formules de dématriçage par interpolation bilinéaire données en cours.

On a donc prouvé que les masques de convolution donnés permettent bien de retrouver les formules de dématriçage par interpolation bilinéaire vues en cours.

Programmation du dématriçage

Sur le modèle du plugin écrit au premier exercice, écrire les méthodes **setup** et **run** d'un nouveau plugin **sample_cfa.java** qui, à partir d'une image CFA en niveaux de gris sur 8 bits, génère les 3 plans φk sous la forme de 3 images également sur 8 bits.

Dans cette partie, nous souhaitons à partir d'une image CFA (obtenue grâce à la disposition G-R-G) en niveaux de gris sur 8 bits, générer les 3 plans φk sous la forme de 3 images également sur 8 bits. Afin de faire cela, nous avons créé un nouveau plugin sur le modèle du plugin précédent. Nous avons tout d'abord écrit la fonction **setup** qui comme précédemment initialise notre attribut ImagePlus grâce à celui donné en paramètre et nous

retourne un entier indiquant que l'image de base est une image en niveaux de gris sur 8 bits (= DOES_8G). Voici le code de cette fonction :

```
public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
        return DOES_8G;
}
```

*Code de la fonction **setup***

Nous avons ensuite codé la méthode **run**. Dans cette fonction, nous mettons tout d'abord à jour les attributs *width* et *height* de notre classe en fonction de la largeur et de la hauteur de l'attribut représentant notre ImagePlus. Puis, nous avons mis les lignes de code qui nous sont données qui permettent de calculer les échantillons de chaque composante de l'image CFA et de créer l'image résultat. Nous avons ajouté une dernière ligne qui affiche alors cette image. Voici le code de cette fonction :

```
public void run(ImageProcessor ip) {
        // Lecture des dimensions de la fenêtre
        width = imp.getWidth();
        height = imp.getHeight();

        // 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);

        // Affiche l'image résulat
        cfa_samples_imp.show();
}
```

*Code de la fonction **run***

En vous inspirant de la méthode **cfa** du premier exercice, écrire la méthode:

ImageProcessor cfa_samples(ImageProcessor cfa_ip, int k)

qui retourne, à partir d'une image CFA **cfa_ip**, l'image en niveaux de gris sur 8 bits (instance de **ByteProcessor**) correspondant au plan φk (φR pour k=0, φG pour k=1, et φB pour k=2).

Nous avons alors remarqué qu'il nous était nécessaire de faire une méthode **cfa_samples** car elle est appelée dans les lignes de code qui nous sont données pour la méthode **run**. Cette fonction permet de générer le plan φk suivant l'image donnée et le nombre nous indiquant la composante souhaitée :

- 0 génère alors le plan φR (rouge)
- 1 génère alors le plan φG (vert)
- 2 génère alors le plan φB (bleu)

Étant donné que l'image CFA à traiter a été obtenue grâce à la disposition G-R-G, nous savons alors que les pixels contenant les niveaux de rouge de l'image de base sont les pixels de la forme (2w+1, 2h), les pixels contenant les niveaux de vert de l'image de base sont les pixels de la forme (2w, 2h) ou de la forme (2w+1, 2h+1), les pixels contenant les niveaux de bleu de l'image de base sont les pixels de la forme (2w, 2h+1) avec w un entier compris entre 0 et width/2 et h un entier compris entre 0 et height/2.

L'image crée par cette fonction est alors une image qui est de base noire et dont les pixels où le niveau de la composante donnée en paramètre est enregistré dans l'image CFA sont mis à jour grâce au niveau de gris correspondant. Voici le code de cette fonction :

```
public ImageProcessor cfa_samples(ImageProcessor ip, int comp){
	// Image couleur de référence et ses dimensions
	width = imp.getWidth();
	height = imp.getHeight();
	ImageProcessor ipRes = ip.createProcessor(width, height);

	int pixel_value = 0;    // Valeur du pixel source

	if(comp == 0){
		for (int y=0; y<height; y+=2) {
			for (int x=1; x<width; x+=2) {
				pixel_value = ip.getPixel(x,y);
				ipRes.putPixel(x,y,pixel_value);
			}
		}
	} else if(comp == 1){
		for (int y=0; y<height; y+=2) {
			for (int x=0; x<width; x+=2) {
				pixel_value = ip.getPixel(x,y);
				ipRes.putPixel(x,y,pixel_value);
			}
		}
		for (int y=1; y<height; y+=2) {
			for (int x=1; x<width; x+=2) {
				pixel_value = ip.getPixel(x,y);
				ipRes.putPixel(x,y,pixel_value);
			}
		}

	} else if(comp == 2){
		for (int y=1; y<height; y+=2) {
			for (int x=0; x<width; x+=2) {
				pixel_value = ip.getPixel(x,y);
				ipRes.putPixel(x,y,pixel_value);
			}
		}
	}
	return ipRes;
}
```

*Code de la fonction **cfa_samples***

Écrire une macro ImageJ **demat_bilin.txt** qui, partant d'une image CFA,

1. appelle le plugin **sample_cfa** pour générer les plans φk dans une pile ;
2. réalise la convolution de chacun de ces plans φk par son masque respectif Hk (*cf.* menu *Process*/*Filters*/*Convolve...*), en veillant à *ne pas normaliser* le masque (les coefficients devront donc être saisis sous forme de nombres réels) ;
3. combine les 3 plans filtrés de la pile dans une image couleur unique (*cf.* menu *Image*/*Color*/*Stack to RGB*) pour former l'image estimée par interpolation bilinéaire.

Afin de réaliser la macro demandée (qui forme l'image estimée par interpolation bilinéaire), nous avons utilisé l'outil **Recorder** d'ImageJ afin d'enregistrer les différentes étapes données qu'on a alors effectué manuellement. Nous avons, par la suite, ajouté les lignes de sélection des plans (*setSlice(x)*) car cela n'est pas enregistré par l'outil.

La première ligne du code appelle le plugin **sample_cfa** pour générer les plans φk dans une pile. Nous sélectionnons ensuite cette pile. Puis, nous sélectionnons chacun des plans un par un en réalisant la convolution de chacun par son masque respectif. Pour finir, nous combinons les 3 plans filtrés pour former l'image RGB estimée par interpolation bilinéaire. Voici le code de cette macro :

```
run("Compile and Run...", "compile=[C:/Program Files/ImageJ/plugins/sample_cfa.java]");
selectWindow("Échantillons couleur CFA");
setSlice(1);
run("Convolve...", "text1=[0.25 0.5 0.25\n0.5 1 0.5\n0.25 0.5 0.25\n] slice");
setSlice(3);
run("Convolve...", "text1=[0.25 0.5 0.25\n0.5 1 0.5\n0.25 0.5 0.25\n] slice");
setSlice(2);
run("Convolve...", "text1=[0 0.25 0\n0.25 1 0.25\n0 0.25 0\n] slice");
run("Stack to RGB");
```

*Code de la macro **demat_bilin***

3ème partie : Dématriçage basé sur l'estimation locale d'un gradient

Implémentation de la méthode

Sur le modèle du plugin écrit au premier exercice, écrire les méthodes **setup** et **run** d'un nouveau plugin **demat_ha.java**. Ce plugin accepte en entrée une image CFA en niveaux de gris sur 8 bits et, dans sa méthode **run**, il génère l'image estimée en stockant ses 3 plans dans une pile d'images 8 bits.

Pour cette question, nous devons nous inspirer du travail réalisé dans les parties précédentes du TP. Nous devons donc reprendre la méthode **setup** de la partie 2 puisque notre image d'entrée sera une image CFA en niveaux de gris sur 8 bits. Voici donc le code de la méthode **setup** :

```
public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
        return DOES_8G;
}
```

*Code de la méthode **setup** dans le plugin **demat_ha.java***

Ensuite, pour la méthode **run**, nous devons nous inspirer du premier exercice en ce qui concerne la création d'un nouvel *ImageProcessor* et les instructions permettant d'afficher une image comme la méthode ***show()*** mais il faudra surtout se servir de la 2ème partie afin de pouvoir stocker les 3 plans de l'image CFA dans une pile d'image 8 bits. Il faut également penser à procéder par convolution plutôt que d'effectuer cette manipulation via une macro comme nous avons pu le faire précédemment. Voici donc le code contenu dans notre méthode **run** afin de réaliser les instructions énoncées dans la consigne :

```
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};
	float[] kernelG = {0,1,0 , 1,4,1 , 0,1,0};
	for (int i=0;i<kernel.length;i++) {
		kernel[i]=kernel[i]/4;
	}

	for (int i=0;i<kernelG.length;i++) {
		kernelG[i]=kernelG[i]/4;
	}

	ImageProcessor red = cfa_samples(ip,0);
	ImageProcessor green = est_G_hamilton(ip);
	ImageProcessor blue = cfa_samples(ip,2);

	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);
	// Composante G estimée par la méthode Hamilton & Adams
	conv.convolve(green, kernelG, 3, 3);
	// Composante B estimée par interpolation bilinéaire grâce à la convolution
	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("Échantillons couleur CFA", samples_stack);
	cfa_samples_imp.show();
}
```

*Code de la méthode **run** dans le plugin **demat_ha.java***

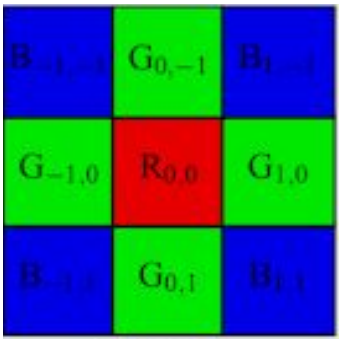
Le code permettant de créer les images est prêt, il ne nous reste plus qu'à créer la méthode ***est_G_hamilton(ImageProcessor cfa_ip)*** qui nous permettra donc d'estimer le plan G par la méthode d'Hamilton & Adams que nous avons pu découvrir dans le cours.

Écrire la méthode :

ImageProcessor est_G_hamilton(ImageProcessor cfa_ip) ;

qui estime le plan G par la méthode de Hamilton & Adams (toujours en considérant le CFA G-R-G).

La méthode d'Hamilton & Adams est une technique de dématriçage exploitant la corrélation spatiale. Dans cette partie du TP, elle estimera le plan G en considérant le CFA G-R-G. Pour rappel, le CFA G-R-G est construit tel que :

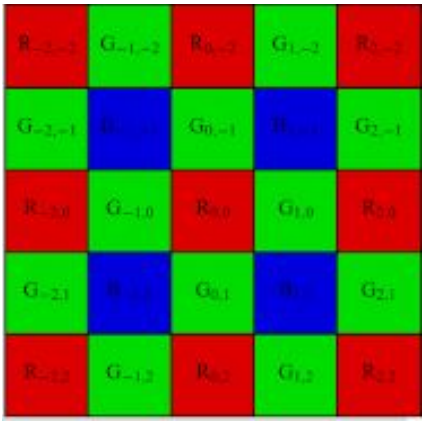


Configuration G-R-G dans le CFA de Bayer

La première partie de l'algorithme consiste à calculer les gradients horizontaux Δ^x et les gradients verticaux Δ^y avec les formules suivantes :

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

Cela étend donc la configuration G-R-G qui sera plus lisible comme telle :



Configuration G-R-G étendue dans le CFA de Bayer

Concernant l'algorithme, nous étudions chaque pixel de la forme (*2x, 2y*) et chaque pixel de la forme (*2x + 1, 2y + 1*) afin de calculer le gradient horizontal et le gradient vertical. Pour l'étude de chaque pixel, il est nécessaire d'effectuer un masquage pour éliminer le bit de signe. Ainsi, la récupération d'un pixel sera sous la forme ***getPixel(x,y)&0xff***. Voici le

morceau de code de la méthode ***ImageProcessor est_G_hamilto(ImageProcessor cfa_ip)*** permettant de calculer les gradients horizontaux et verticaux de chaque pixel étudié :

```
float gradXLeftPart = cfa_ip.getPixel(x-1,y) - cfa_ip.getPixel(x+1,y);
float gradXRightPart = 2*cfa_ip.getPixel(x,y) - cfa_ip.getPixel(x-2,y) - cfa_ip.getPixel(x+2, y);
float gradX =  Math.abs(gradXLeftPart) + Math.abs(gradXRightPart);

float gradYLeftPart = cfa_ip.getPixel(x,y-1) - cfa_ip.getPixel(x,y+1);
float gradYRightPart = 2*cfa_ip.getPixel(x,y) - cfa_ip.getPixel(x,y-2) - cfa_ip.getPixel(x,y+2);
float gradY =  Math.abs(gradYLeftPart) + Math.abs(gradYRightPart);
```

Morceau de code permettant de déterminer le gradient horizontal et vertical d'un pixel (x, y)

Une fois les gradients déterminés, il nous suffit de les comparer afin d'attribuer la nouvelle valeur au pixel étudié. Nous sommes donc face à 3 cas d'études différents :

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

Soit l'un des gradient est plus important que l'autre, soit les deux sont égaux. Nous représentons ces 3 cas dans l'algorithme de la façon suivante :

```
if(gradX < gradY){
        pixel_value = ((cfa_ip.getPixel(x-1,y) + cfa_ip.getPixel(x+1,y))/2) +
                (((2*cfa_ip.getPixel(x,y)) - cfa_ip.getPixel(x-2,y) - cfa_ip.getPixel(x+2,y))/4);
      } else if(gradX > gradY){
        pixel_value = ((cfa_ip.getPixel(x,y-1) + cfa_ip.getPixel(x,y+1))/2) +
                (((2*cfa_ip.getPixel(x,y)) - cfa_ip.getPixel(x,y-2) - cfa_ip.getPixel(x,y+2))/4);
      } else {
        // gradX == gradY
        pixel_value = ((cfa_ip.getPixel(x,y-1)+cfa_ip.getPixel(x,y+1)+
                  cfa_ip.getPixel(x-1,y)+cfa_ip.getPixel(x+1,y))/4) +
                (((4*cfa_ip.getPixel(x,y))-cfa_ip.getPixel(x,y-2) -
                  cfa_ip.getPixel(x,y+2)-cfa_ip.getPixel(x-2,y) -
                  cfa_ip.getPixel(x+2,y))/8);
      }
}
```

Morceau de code permettant d'affecter la bonne valeur au pixel étudié en fonction de la valeur des gradients

Vous pourrez trouver en **Annexe 1** le code complet de la méthode ***ImageProcessor est_G_hamilton(ImageProcessor cfa_ip)***.

Nous obtenons donc le résultat suivant, visible avant et après l'opération de convolution :

Sur la gauche, le résultat de la méthode Hamilton & Adams non convoluée pour un filtre G-R-G, sur la droite, le même résultat après la convolution

D'un point de vue visuel, nous remarquons très peu de différences pour ce résultat en comparaison avec l'interpolation bilinéaire de la partie précédente. Nous allons voir par la suite qu'il y a toutefois quelques différences.

Évaluation de la qualité de l'image estimée

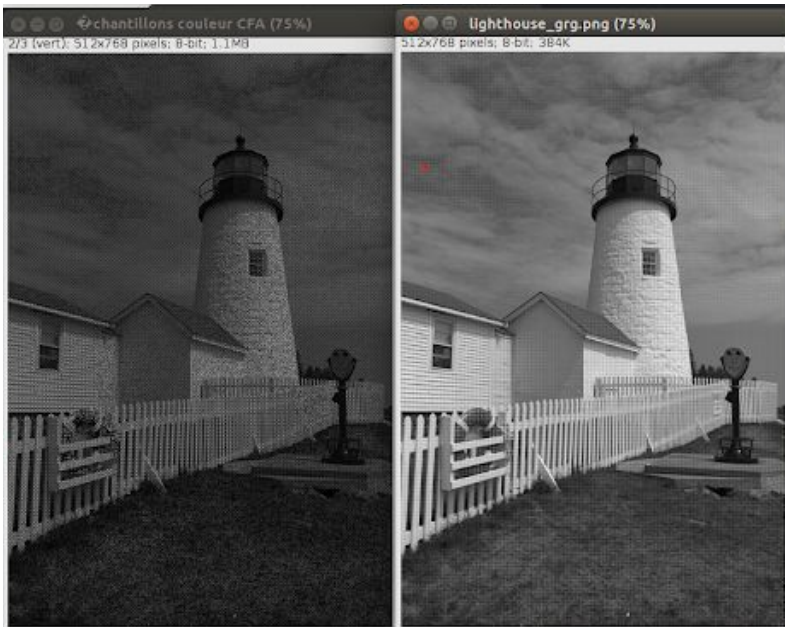
Comparer le plan vert ainsi estimé à celui obtenu par interpolation bilinéaire. Cette comparaison peut être uniquement visuelle ou se baser sur le calcul du rapport signal-sur-bruit pic-à-pic (PSNR) du plan vert.

Le rapport signal-sur-bruit pic-à-pic (en anglais **P**eak **S**ignal to **N**oise **R**atio, **PSNR**) mesure la qualité d'une image Î par rapport à l'image de référence I, tout cela en évaluant l'erreur quadratique moyenne (en anglais **M**ean **S**quare **E**rror, **MSE**). Cette mesure s'exprime en décibels **dB** grâce à la formule suivante :

$$PSNR(\boldsymbol{I}, \hat{\boldsymbol{I}}) = 10 \cdot \log_{10}\left(\frac{d^2}{MSE}\right) \quad \text{avec} \quad MSE = \frac{1}{3wh} \sum_{k=R,G,B} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \left(I_{x,y}^{k} - \hat{I}_{x,y}^{k}\right)^2$$

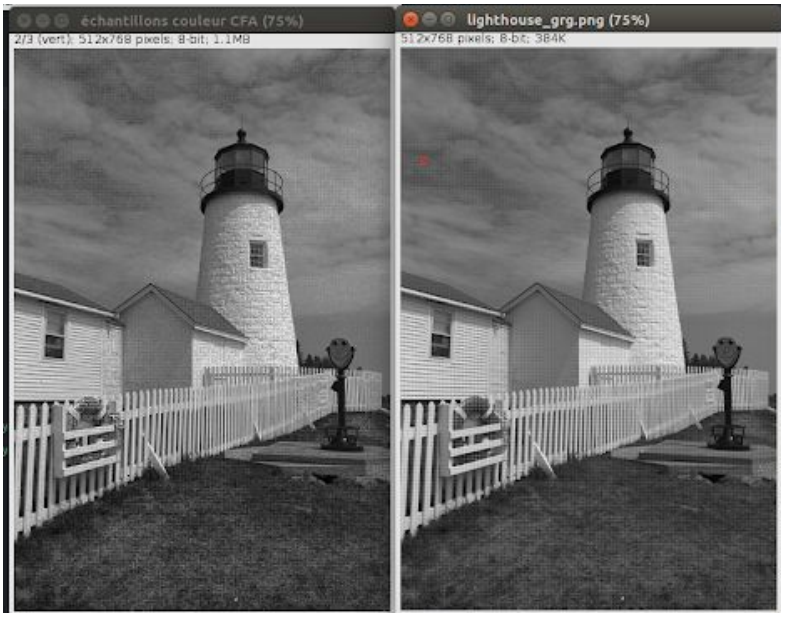
Plus la valeur du PSNR est élevée, plus l'image estimée est "proche" de l'image référence. En effet, le PSNR évalue la proximité entre deux images et non la qualité visuelle de l'image estimée Î.

Pour cette question, nous mesurons le **PSNR** entre le plan vert estimé par interpolation bilinéaire et l'image CFA ainsi que le **PSNR** entre le plan vert estimé avec la méthode d'Hamilton & Adams et l'image CFA.



Sur la gauche, l'estimation du plan vert avec la méthode de l'interpolation bilinéaire, sur la droite l'image CFA de base

Nous obtenons sur cette comparaison un PSNR d'à peu près 9,5 ce qui est très peu. Observons le PSNR de la comparaison entre l'estimation du plan vert avec la méthode d'Hamilton & Adams :



Sur la gauche, l'estimation du plan vert avec la méthode d'Hamilton & Adams, sur la droite l'image CFA de base

Pour cette comparaison, nous avons un PSNR d'environ 24,5 ce qui est bien mieux qu'avec la méthode précédente.

Nous observons que le PSNR le plus élevé est celui obtenu avec l'estimation du plan vert par la méthode d'Hamilton & Adams, ce qui nous indique que l'estimation effectuée par cette même méthode est plus proche de l'image CFA que l'estimation effectuée par l'interpolation bilinéaire. Cela se voit également à l'oeil nu étant donné que l'image CFA ressemble plus à l'image obtenue par la méthode d'Hamilton & Adams que celle obtenue par interpolation bilinéaire.

Comparer l'image couleur ainsi estimée à celle obtenue (à l'exercice 2) par interpolation bilinéaire des trois composantes couleur. L'estimation du seul plan vert par cette méthode permet-elle d'obtenir une image couleur satisfaisante ?

L'image par interpolation bilinéaire présente quelques défauts bien visibles notamment au niveaux des barrières. Voici le résultat obtenu pour l'estimation du seul plan vert par la méthode d'Hamilton & Adams :

Image estimée du seul plan vert avec la méthode d'Hamilton & Adams

Nous remarquons que l'herbe n'est pas colorée, c'est pourquoi une estimation du seul plan vert n'est pas suffisante, il faut effectuer une estimation des plans rouge et bleus également qui se fait de manière similaire.

Annexe 1

```
public ImageProcessor est_G_hamilton(ImageProcessor cfa_ip) {
   int width = imp.getWidth();
           int height = imp.getHeight();

   ImageProcessor est_ip = cfa_ip.duplicate();

   int pixel_value = 0;
   // G 1
   for(int y = 0; y < height; y=y+2){
     for(int x = 0; x < width; x=x+2){
       float gradXLeftPart = cfa_ip.getPixel(x-1,y) - cfa_ip.getPixel(x+1,y);
       float gradXRightPart = 2*cfa_ip.getPixel(x,y) - cfa_ip.getPixel(x-2,y) - cfa_ip.getPixel(x+2, y);
       float gradX =  Math.abs(gradXLeftPart) + Math.abs(gradXRightPart);

       float gradYLeftPart = cfa_ip.getPixel(x,y-1) - cfa_ip.getPixel(x,y+1);
       float gradYRightPart = 2*cfa_ip.getPixel(x,y) - cfa_ip.getPixel(x,y-2) - cfa_ip.getPixel(x,y+2);
       float gradY =  Math.abs(gradYLeftPart) + Math.abs(gradYRightPart);

       if(gradX < gradY){
         pixel_value = ((cfa_ip.getPixel(x-1,y) + cfa_ip.getPixel(x+1,y))/2) +
                   (((2*cfa_ip.getPixel(x,y)) - cfa_ip.getPixel(x-2,y) - cfa_ip.getPixel(x+2,y))/4);
       } else if(gradX > gradY){
         pixel_value = ((cfa_ip.getPixel(x,y-1) + cfa_ip.getPixel(x,y+1))/2) +
                   (((2*cfa_ip.getPixel(x,y)) - cfa_ip.getPixel(x,y-2) - cfa_ip.getPixel(x,y+2))/4);
       } else {
         // gradX == gradY
         pixel_value = ((cfa_ip.getPixel(x,y-1)+cfa_ip.getPixel(x,y+1)+
                     cfa_ip.getPixel(x-1,y)+cfa_ip.getPixel(x+1,y))/4) +
                   (((4*cfa_ip.getPixel(x,y))-cfa_ip.getPixel(x,y-2) -
                     cfa_ip.getPixel(x,y+2)-cfa_ip.getPixel(x-2,y) -
                     cfa_ip.getPixel(x+2,y))/8);
       }
       est_ip.putPixel(x,y,pixel_value);
     }
   }

   //G 2
   for(int y = 1; y < height; y=y+2){
     for(int x = 1; x < width; x=x+2){
       float gradXLeftPart = cfa_ip.getPixel(x-1,y) - cfa_ip.getPixel(x+1,y);
       float gradXRightPart = 2*cfa_ip.getPixel(x,y) - cfa_ip.getPixel(x-2,y) - cfa_ip.getPixel(x+2, y);
       float gradX =  Math.abs(gradXLeftPart) + Math.abs(gradXRightPart);

       float gradYLeftPart = cfa_ip.getPixel(x,y-1) - cfa_ip.getPixel(x,y+1);
       float gradYRightPart = 2*cfa_ip.getPixel(x,y) - cfa_ip.getPixel(x,y-2) - cfa_ip.getPixel(x,y+2);
       float gradY =  Math.abs(gradYLeftPart) + Math.abs(gradYRightPart);
```

```
    if(gradX < gradY){
      pixel_value = ((cfa_ip.getPixel(x-1,y) + cfa_ip.getPixel(x+1,y))/2) +
                (((2*cfa_ip.getPixel(x,y)) - cfa_ip.getPixel(x-2,y) - cfa_ip.getPixel(x+2,y))/4);
    } else if(gradX > gradY){
      pixel_value = ((cfa_ip.getPixel(x,y-1) + cfa_ip.getPixel(x,y+1))/2) +
                (((2*cfa_ip.getPixel(x,y)) - cfa_ip.getPixel(x,y-2) - cfa_ip.getPixel(x,y+2))/4);
    } else {
      // gradX == gradY
      pixel_value = ((cfa_ip.getPixel(x,y-1)+cfa_ip.getPixel(x,y+1)+
                    cfa_ip.getPixel(x-1,y)+cfa_ip.getPixel(x+1,y))/4) +
                   (((4*cfa_ip.getPixel(x,y))-cfa_ip.getPixel(x,y-2) -
                    cfa_ip.getPixel(x,y+2)-cfa_ip.getPixel(x-2,y) -
                    cfa_ip.getPixel(x+2,y))/8);
    }


    est_ip.putPixel(x,y,pixel_value);
   }
  }

  return (est_ip);
}
```

*Code complet de la méthode **est_G_hamilton***