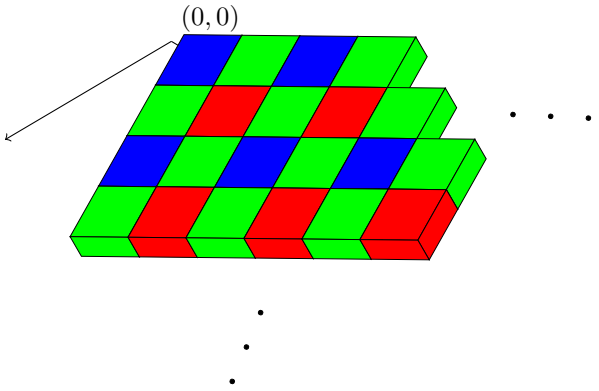
# TP : RAW

Jonathan Fabrizio
http://jo.fabrizio.free.fr/

## Objectif

L'objectif de ce TP est de pouvoir générer une image depuis les données *RAW* d'un capteur. Pour s'entraîner nous utiliserons une image issue d'une *PiCam V2*.

Le capteur de la *PiCam V2* est un CMOS : le IMX219 de Sony. Vous pouvez chercher ses spécifications sur internet. Pour restituer la couleur, le matrice de capteurs du CMOS est recouverte de filtres (respectant le motif de *Bayer* - *BGGR*). Ce CMOS est sensible au proche infrarouge mais l'optique contient un filtre qui coupe ces longueurs d'onde.




## 1 Lecture de l'image

Écrire un programme pour décoder les données *RAW* d'un capteur demande beaucoup de temps (et les spécifications ne sont pas toujours disponibles). De plus, il n'y a pas vraiment de consensus sur les formats entre les différents constructeurs. Pour simplifier le problème (car ce n'est pas l'objectif ici), je vous ai extrait juste l'image des données envoyées par le *driver* de la caméra. Il reste néanmoins un peu de travail car les valeurs sont sur 10 bits. L'encodage est le suivant :

- les lignes paires contiennent alternativement un pixel bleu, un pixel vert,
- les lignes impaires contiennent alternativement un pixel vert, un pixel rouge,
- l'image de la photo fait $3280 \times 2464$ pixels (le capteur, lui, a plus de pixels),
- les canaux sont codés sur 10bits et sont par lots de 4 pixels successifs.

Un lot de 4 pixels successifs est codé par un lot de 5 octets. Pour ces 4 pixels on a :

- 1 octet pour le premier pixel
- 1 octet pour le second pixel
- 1 octet pour le troisième pixel
- 1 octet pour le quatrième pixel
- 1 octet coupé en 4 qui encode pour chacun des 4 pixels les 2 bits de poids faible manquants (bits 0 et 1 pour le $4^{\text{ième}}$ pixel, bits 2 et 3 pour le $3^{\text{ième}}$ pixel...).

Ecrivez une fonction qui permettent de lire et décoder l'image *RAW* afin de la charger en mémoire (Attention de bien conserver les 10 bits par canal). Pour se simplifier la vie, on codera chaque canal sur un `double`.

> Note culturelle : Il existe beaucoup de formats pour enregistrer l'image *RAW* du capteur (plusieurs constructeurs ayant développé leur propre format, qu'ils ont parfois fait évoluer...). Il existe néanmoins le format DNG (*Digital Negatif*) d'Adobe qui est un peu plus ouvert. Par ailleurs, si vous avez besoin de décoder une image *RAW*, vous pouvez utiliser *libRaw*. De même, si vous voulez manipuler une image *RAW* vous pouvez utiliser *ufraw*.

## 2 Black point detection

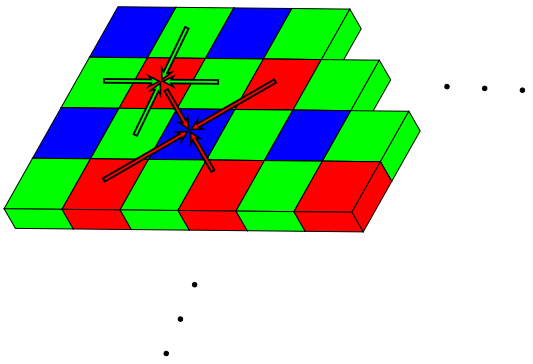
On pourrait penser qu'en sortie du capteur les zones noires de l'image sont à 0 et les zones blanches sont à 1023. Or ce n'est pas le cas. Un capteur qui n'est pas éclairé ne renvoie pas exactement la valeur 0. La première chose à faire est donc de détecter la valeur du noir. Cette valeur dépend de plein de choses (du *CCD*/*CMOS*, de la température...) si bien qu'elle n'est pas connue au moment du cliché. Ce que l'on va faire c'est d'essayer de la deviner en analysant l'image. Pour ce faire on va supposer qu'il y a des pixels qui n'ont pas reçu de lumière dans le cliché. Il suffit donc de trouver les valeurs minimum par canaux $(r_{min}, v_{min}, b_{min})$ pour essayer de déduire la valeur du noir.

Ecrivez donc une fonction qui va chercher le min par canal puis une autre qui va soustraire ces valeurs à l'ensemble des pixels de l'image (On va soustraire le $r_{min}$ à tout les pixels rouges, le $v_{min}$ à tous les pixels vert...).

> Note culturelle : Dans la pratique (et c'est le cas pour le IMX219), le CMOS ou le CCD contient une zone de capteur supplémentaire qui n'est pas exposée à la lumière (*optical black*). Cette zone sert de référence pour connaître la valeur du noir. Cela permet d'être plus précis et d'éviter de faire des suppositions quant au contenu de l'image. Cela permet aussi de gérer parfaitement les cas ou l'image ne contient que des zones claires ou blanches.

## 3 Demosaicing

Maintenant que nous avons bien les niveaux corrects, nous pouvons reconstituer l'image. Pour cela il va falloir deviner les valeurs manquantes dans l'image. Pour faire simple, pour chaque valeur manquante, vous ferez une interpolation linéaire des 2 ou 4 voisins les plus proches du même canal. Pour se faciliter la vie, vous pouvez encoder votre image dans un tableau (matrice/vecteur) de `double` avec les trois canaux $R$, $V$ et $B$.



Ecrivez une fonction qui réalise la *débayerisation* et qui revoie une nouvelle image complete ($R$, $V$ et $B$ linéaires codés sur des doubles).

> Note culturelle : Cet algorithme est bien trop naif. Dans la pratique on essaye au moins de tenir compte du gradient pour choisir la direction de l'interpolation et ne pas trop créneler les contours.

## 4 Encoding

Maintenant que vous avez une image complète, vous pouvez encoder cette image en *sRGB* 8 bits par canaux et l'enregistrer (vous pouvez utiliser les fonctions des TPs précédents pour enregistrer en TGA ou faire les votre pour écrire dans le format de votre choix).

Ecrivez une fonction pour réaliser l'encodage en *sRGB* et l'enregistrement sur disque.

Normalement, vous pouvez maintenant voir l'image prise en photo mais les couleurs ne sont pas correctes (la photo n'est pas très belle).

## 5 White Balance

Pour restituer les couleurs correctement, il faut équilibrer les trois canaux (avant l'encodage en *sRGB*). Pour cela, on applique usuellement une correction linéaire. Si nous avons un canal $C$, avec un niveau pour le noir

$C_{min}$ et un niveau pour le blanc $C_{max}$, pour tout niveau mesuré $C_m$, le niveau corrigé est : $C_c = \frac{C_m - C_{min}}{C_{max} - C_{min}}$. Comme nous avons déjà corrigé le $C_{min}$ pour le faire correspondre à 0, la correction se simplifie en : $C_c = \frac{C_m}{C_{max}}$ soit $C_c = C_m \mathrm{G}_c$.

Cela donne :

$$\begin{pmatrix} R_c \\ V_c \\ B_c \end{pmatrix} = \begin{pmatrix} \mathrm{G}_r & 0 & 0 \\ 0 & \mathrm{G}_v & 0 \\ 0 & 0 & \mathrm{G}_b \end{pmatrix} \begin{pmatrix} R_m \\ V_m \\ B_m \end{pmatrix} \quad (1)$$

Dans la pratique on normalise par le canal vert, cela donne :

$$\begin{pmatrix} R_c \\ V_c \\ B_c \end{pmatrix} = \begin{pmatrix} \mathrm{G}_r/\mathrm{G}_v & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \mathrm{G}_b/\mathrm{G}_v \end{pmatrix} \begin{pmatrix} R_m \\ V_m \\ B_m \end{pmatrix} \quad (2)$$

Ce que l'on re-ecrit en

$$\begin{pmatrix} R_c \\ V_c \\ B_c \end{pmatrix} = \begin{pmatrix} \mathrm{gain}_r & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \mathrm{gain}_b \end{pmatrix} \begin{pmatrix} R_m \\ V_m \\ B_m \end{pmatrix} \quad (3)$$

Le problème est de trouver les bons coefficients à appliquer. Pour faire simple nous rechercherons le max par canal, comme nous l'avons fait pour le noir.

Ecrivez une fonction qui réalise la balance des blancs. Cette fonction doit prendre en entrée l'image ainsi que les coefficients $\mathrm{gain}_r$ et $\mathrm{gain}_b$ et réalise la balance des blancs.

> Note : Le capteur renvoie dans le EXIF les valeurs qu'il a utilisées. Pour information, je vous copie le EXIF ici :$gain_r = 1.164$ et $gain_b = 2.328$

Essayez votre fonction avec les valeurs fournies par le driver, encodez à nouveau votre image et comparer les résultats - avec et sans égalisation des blancs.

Ecrivez une seconde fonction qui essaye de trouver ces coefficients. Pour cela, votre fonction cherchera le max par canal et en déduira $\mathrm{gain}_r$ et $\mathrm{gain}_b$.

Appliquez les coefficients obtenus, encodez votre image et comparer vos résultats avec le résultat obtenu avec les valeurs du driver. Comparer aussi les résultats obtenus par GIMP (`Colors->Auto->White Balance`).

> Note culturelle : Les coefficients de la matrice de correction dépendent de la température de la couleur de la source lumineuse. Il est donc possible d'imposer une balance des blancs pour compenser la couleur d'une source si celle-ci est connue (Soleil, ampoule au tungsten...)

# 6 Tone Mapping

Pour rendre l'image plus belle ou pour obtenir un rendu particulier (pour se rapprocher d'un rendu argentique d'une pellicule particulière par exemple, ou pour réaliser un filtre comme le sépia...), les fabricants de caméras ou d'appareils photos appliquent des corrections, juste avant l'encodage. La plupart du temps le fabricant ne communique pas sur les corrections qu'il applique.

Vous pouvez télécharger l'image associée au RAW que vous avez traité. Comparer votre résultat avec le résultat du fabricant.

Ajouter dans votre chaine, une étape de *tone mapping*, juste avant l'encodage pour essayer de s'approcher au mieux de l'image du fabricant (Ce n'est vraiment pas facile et vous avez trop peu d'information pour y arriver vraiment).

# 7 Dead Pixels (Optionnel)

Dès la sortie de l'usine, les capteurs ont souvent des pixels morts. Il est donc nécessaire de faire un traitement pour éviter d'avoir des valeurs aberrantes dans l'image. Téléchager la carte des pixels morts.

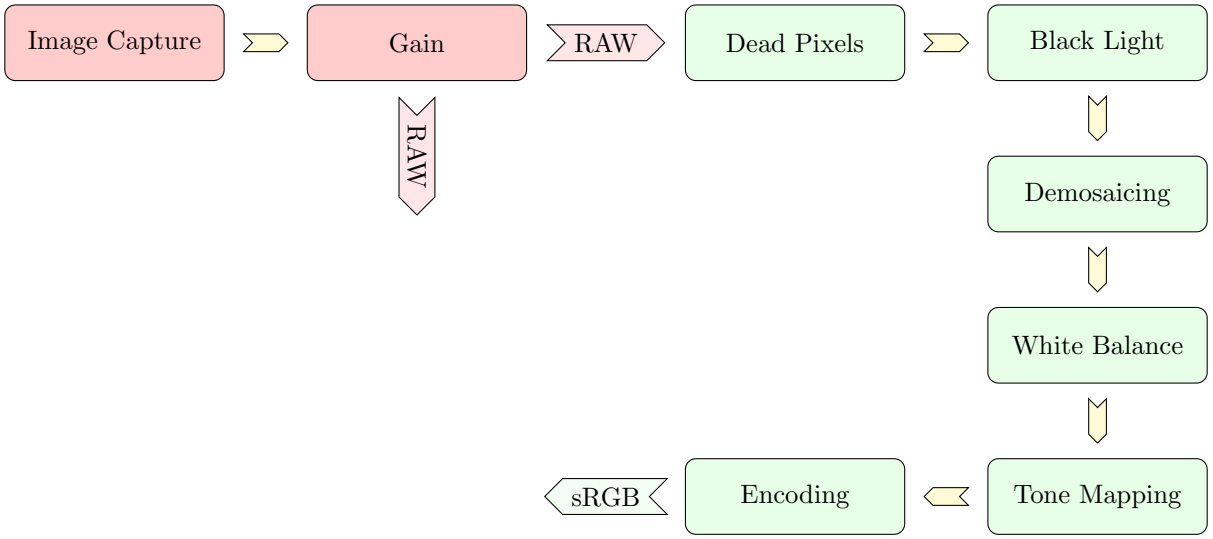
> Note culturelle : J'ai créé cette carte spécifiquement pour ce capteur en prenant une image dans le noir complet et en cherchant les valeurs aberrantes.

Pour chaque point différent de 0 vous savez que vous ne pouvez pas faire confiance au capteur. Pour ces points là, vous remplacerez dans votre image la valeur de ces pixels par une interpolation des valeurs données par les capteurs de même couleur les plus proches, comme vous l'avez fait pour la *débayérisation*.

Note culturelle : Ce n'est pas la seule correction que fait l'appareil photo. En plus d'éliminer les pixels morts (détéctés en usine), l'appareil connait aussi l'optique utilisée et ses défauts (les déformations qu'elle induit). L'appareil applique donc des corrections spatiales lorsqu'il reconnait l'optique associée. Idem, les images sont en général bruitées en sortie de capteur, il y a donc des étapes de débruitages. Enfin, il est souvent possible de gérer la sensibilité (ISO) du capteur au moment de la prise d'image. Il y a un amplificateur en sortie pour changer la luminosité de l'image. Cet étage d'amplification a tendance à augmenter le bruit de l'image (d'où la nécessité de l'étape de débruitage). D'un fabricant à l'autre, il est difficile de savoir dans quel ordre précisément ces étapes sont réalisées et surtout quels sont les étapes qui sont réalisées avant la génération du *RAW* (le *RAW* n'étant pas si brut que ça...).

# 8 Résumé

Un résumé de la chaine que nous venons de voir. Cette chaine est incomplète (débruitage...) et surtout les parties réalisées par le driver, i.e. avant la génération du *RAW* sont supposées. Attention dans la pratique cette chaine peut varier d'un appareil à l'autre (tant dans l'ordre que dans le contenu).




Note culturelle : Nous avons utilisé le modèle *RGB* linéaire tout au long du TP mais certains constructeurs utilisent le modèle *LMS* et CIE *XYZ*.

# 9 Annexe

## 9.1 EXIF de l'image

```
Format: JPEG (Joint Photographic Experts Group JFIF format)
 Mime type: image/jpeg
 Class: DirectClass
 Geometry: 3280x2464+0+0
 Resolution: 72x72
 Print size: 45.5556x34.2222
 Units: PixelsPerInch
 Type: TrueColor
 Endianess: Undefined
 Colorspace: sRGB
 Depth: 8-bit
 Channel depth:
   red: 8-bit
   green: 8-bit
   blue: 8-bit
 Channel statistics:
   Pixels: 8081920
   Red:
     min: 1 (0.00392157)
     max: 255 (1)
     mean: 145.629 (0.571094)
     standard deviation: 66.2979 (0.259992)
     kurtosis: -0.883431
     skewness: -0.259015
```

```
    entropy: 0.937018
  Green:
    min: 0 (0)
    max: 255 (1)
    mean: 139.814 (0.548291)
    standard deviation: 66.7991 (0.261957)
    kurtosis: -0.863373
    skewness: 0.00688072
    entropy: 0.937883
  Blue:
    min: 1 (0.00392157)
    max: 255 (1)
    mean: 142.377 (0.558341)
    standard deviation: 67.7066 (0.265516)
    kurtosis: -0.940359
    skewness: 0.00677727
    entropy: 0.92979
Image statistics:
  Overall:
    min: 0 (0)
    max: 255 (1)
    mean: 142.607 (0.559242)
    standard deviation: 66.9371 (0.262498)
    kurtosis: -0.90409
    skewness: -0.0800305
    entropy: 0.934897
Rendering intent: Perceptual
Gamma: 0.454545
Chromaticity:
  red primary: (0.64,0.33)
  green primary: (0.3,0.6)
  blue primary: (0.15,0.06)
  white point: (0.3127,0.329)
Background color: white
Border color: srgb(223,223,223)
Matte color: grey74
Transparent color: black
Interlace: None
Intensity: Undefined
Compose: Over
Page geometry: 3280x2464+0+0
Dispose: Undefined
Iterations: 0
Compression: JPEG
Quality: 100
Orientation: Undefined
Properties:
  date:create: 2019-03-29T10:09:45+01:00
  date:modify: 2018-06-01T14:57:42+02:00
  exif:ApertureValue: 20000/10000
  exif:BrightnessValue: 285/100
  exif:ColorSpace: 1
  exif:ComponentsConfiguration: 1, 2, 3, 0
  exif:DateTime: 2018:05:03 15:15:01
  exif:DateTimeDigitized: 2018:05:03 15:15:01
  exif:DateTimeOriginal: 2018:05:03 15:15:01
  exif:ExifImageLength: 2464
  exif:ExifImageWidth: 3280
  exif:ExifOffset: 192
  exif:ExifVersion: 48, 50, 50, 48
  exif:ExposureMode: 0
  exif:ExposureProgram: 3
  exif:ExposureTime: 9443/1000000
  exif:Flash: 0
  exif:FlashPixVersion: 48, 49, 48, 48
  exif:FNumber: 20000/10000
  exif:FocalLength: 30390/10000
  exif:ImageLength: 2464
  exif:ImageWidth: 3280
  exif:InteroperabilityOffset: 904
  exif:ISOSpeedRatings: 100
  exif:Make: RaspberryPi
  exif:MakerNote: 101, 118, 61, 45, 49, 32, 109, 108, 117, 120, 61, 45, 49, 32, 101, 120, 112, 61, 57, 52, 52, 51, 32, 97, 103
  exif:MaxApertureValue: 20000/10000
  exif:MeteringMode: 2
```

```
    exif:Model: RP_imx219
    exif:ResolutionUnit: 2
    exif:ShutterSpeedValue: 6726539/1000000
    exif:thumbnail:Compression: 6
    exif:thumbnail:ImageLength: 48
    exif:thumbnail:ImageWidth: 64
    exif:thumbnail:InteroperabilityIndex: R98
    exif:thumbnail:JPEGInterchangeFormat: 1040
    exif:thumbnail:JPEGInterchangeFormatLength: 24576
    exif:thumbnail:ResolutionUnit: 2
    exif:thumbnail:XResolution: 72/1
    exif:thumbnail:YResolution: 72/1
    exif:WhiteBalance: 0
    exif:XResolution: 72/1
    exif:YCbCrPositioning: 1
    exif:YResolution: 72/1
    jpeg:colorspace: 2
    jpeg:sampling-factor: 2x2,1x1,1x1
    signature: 749e75c2a8ce66921d549d95106d3191b02d119da5123c399c69991bf14a864e
  Profiles:
    Profile-exif: 25622 bytes
  Artifacts:
    filename: image_000001.jpg
    verbose: true
  Tainted: False
  Filesize: 14.82MB
  Number pixels: 8.082M
  Pixels per second: 73.47MB
  User time: 0.100u
  Elapsed time: 0:01.110
  Version: ImageMagick 6.9.7-4 Q16 x86_64 20170114 http://www.imagemagick.org
```

## 9.2 EXIF de l'image : décodage de "*MakerNote*"

```
ev=-1
mlux=-1
exp=9443
ag=469
focus=255
gain_r=1.164
gain_b=2.328
greenness=-11
ccm=6022,-2314,394,-936,4728,310,300,-4324,8126,0,0,0
md=0
tg=233
233
oth=0
0
b=0
f=233
233
fi=0
ISP
Build
Date:
Mar
23
2018,
16:34:44
VC_BUILD_ID_VERSION:
d92349b3d22c277e44d508a2aae6af2b1e5bb224
(clean)
VC_BUILD_ID_USER:
dc4
VC_BUILD_ID_BRANCH:
master
```