Quelques filtres en photographie numérique

Un filtre permet de transformer une photographie numérique, il s'agit tout simplement d'un algorithme à appliquer. Il existe des centaines de filtres différents, permettant d'aboutir à des effets très variés.
Remarque importante : on choisira une image sans fond transparent.

I/ Squelette du programme

Ce squelette de programme est la base de toute transformation de photos :

- **Création d'une nouvelle image** ayant les mêmes dimensions que l'image de base et qui recevra les pixels transformés. Cela évite de modifier l'image de base.
  Remarque **:** il se peut que l'on doive en créer plusieurs en fonction de l'algorithme appliqué. Elles servent alors d'images auxiliaires.
- **Une double boucle** qui permet de parcourir tous les pixels de l'images.
  Remarque **:** on fera attention aux bornes lors de l'application de certains algorithmes.
- **Exploitation des données** de l'image de base avec la méthode *getpixel(coordonnées).*
- Une partie du programme dédiée à l'application de l'algorithme.
- **La mise en place des nouveaux pixels** dans la nouvelle image avec la méthode *putpixel(coordonnées, pixel).*

Remarque : s'il s'agit d'une fonction, on prendra soin de mettre en paramètre l'image de base et en valeur de retour l'image transformée.

A noter : le langage Python fonctionne avec un système « d'alias » (référence). Il ne recrée pas une image intégralement lors d'un passage en argument ou d'un renvoi dans une fonction mais seulement un « lien » vers cet image. C'est un gage d'efficacité mais attention à bien créer des nouvelles images lorsque cela est nécessaire.

Exemple : inversion du canal bleu

```
# Chargement de l'image à partir du répertoire de Jupyter
# BIEN MONTER L'IMAGE DANS LE REPERTOIRE #
imgBase = Image.open("perceval.jpg") # Lien vers l'image
width,height = imgBase.size          # Détermine la largeur et hauteur de l'image en pixels

imgNew = Image.new("RGB",(width,height))    # Création d'une image RGB vierge pour recueillir
                                            # les modifications apportées

for i in range(width):          # Double boucle permettant de parcourir
    for j in range(height):     # chaque pixel de l'image

        red,green,blue = imgBase.getpixel((i,j))   # Récupère le code RVB du pixel
```

```
################################################################################
################## Algorithme de transformation de l'image #####################
################################################################################

        n_blue = 255 - blue # Inversion du canal bleu

################################################################################

        imgNew.putpixel((i,j),(red, green, n_blue))
```

II/ Quelques transformations de base

1/ Inversions des canaux

Rappel : chaque couleur dans le code RGB est notifiée par une valeur entre 0 (aucune couleur) et 255 (couleur au maximum).

Il suffit donc d'écrire ceci pour inverser les canaux :

```
###############################################################################
################ Algorithme de transformation de l'image ######################
###############################################################################

        n_red = 255 - red     # Inversion du canal rouge
        n_green = 255 - green  # Inversion du canal vert
        n_blue = 255 - blue # Inversion du canal bleu

###############################################################################

        imgNew.putpixel((i,j),(n_red, n_green, n_blue))
```

2/ Simulation de la vue d'un daltonien

Un daltonien est une personne ne percevant un ou plusieurs canaux de couleurs dite « primaires », notamment le rouge et/ou le vert. Lien : https://www.larousse.fr/dictionnaires/francais/daltonisme/21548
Il suffit donc de mettre le/les canal(aux) concernés à zéro.

Ici un exemple d'un daltonien ne voyant pas la couleur rouge :

```
###############################################################################
################ Algorithme de transformation de l'image ######################
###############################################################################


###############################################################################

        imgNew.putpixel((i,j),(0, green, n_blue))  # Mise à 0 du canal rouge
```

Remarque : code à modifier en fonction du type de daltonisme représenté.

3/ Mise en dégradé de gris d'une photo en couleurs

Rappel : dans le code RGB, une teinte de gris a les mêmes valeurs pour les composantes rouge, vert, bleu.
Voici deux programmes à tester :

Une moyenne basique des valeurs des trois canaux

```
###############################################################################
################ Algorithme de transformation de l'image ######################
###############################################################################

         # Division euclidienne, assurant que `gray`est un entier
        gray = (red + green + blue) // 3
###############################################################################

        imgNew.putpixel((i,j),(gray, gray, gray))  # Chaque canal a la même valeur
```

Une formule empirique tenant compte de la sensibilité de l'œil humain au vert

```
################################################################################
################## Algorithme de transformation de l'image #####################
################################################################################

        # Conversion en entier par la méthode int(valeur)
        gray = int(rouge*0.299 + vert*0.587 + bleu*0.144)
################################################################################

        imgNew.putpixel((i,j),(gray, gray, gray))  # Chaque canal a la même valeur
```

4/ Ajout de transparence

Il suffit de créer une nouvelle image ayant le canal « alpha » gérant la transparence. Les images en format .png la gèrent notamment. Les valeurs de ce canal sont également entre 0 et 255.

Voici le programme :

```
# Attention : Image (à choisir, ici perceval) à monter sur Jupyter
imgBase = Image.open("perceval.png")   # Charge l'image en question
widht,height = imgBase.size    # Récupère les dimensions de l'image

imgNew = Image.open("RGBA",(width,height))  # RGBA assure le canal alpha

for i in range(width) :
    for j in range(height) :

        red, green, blue = imgBase.getpixel(i,j)

################################################################################
################## Algorithme de transformation de l'image #####################
################################################################################

        alpha = 50   # Valeur à choisir entre 0 (invisible) et 255 (aucune transparence)
################################################################################

        imgNew.putpixel((i,j),(red, green, blue, alpha))  # Canal alpha ajouté
```

5/ Effet miroir

L'effet miroir est créé en inversant les pixels selon la longueur *(width)* de l'image.
Voilà un exemple de programme :

```
for i in range(width) :
    for j in range(height) :

        red, green, blue = imgBase.getpixel(i,j)

################################################################################
################## Algorithme de transformation de l'image #####################
################################################################################

################################################################################

        # Attention, la variable i varie entre 0 et width-1
        imgNew.putpixel((width-1-i,j),(red, green, blue))
```

6/ Inversion de l'image par symétrie centrale

On souhaite obtenir ceci :

Le principe est le même que précédemment mais il faut aussi inverser les pixels selon leur hauteur *(height)*.

```
for i in range(width) :
    for j in range(height) :

        red, green, blue = imgBase.getpixel(i,j)

################################################################################
################# Algorithme de transformation de l'image ######################
################################################################################

################################################################################

        # Attention, la variable `i` varie entre 0 et width-1. Même chose pour la variable `j`
        imgNew.putpixel((width-1-i,height-1-j),(red, green, blue))
```

III/ Aller plus loin (*)

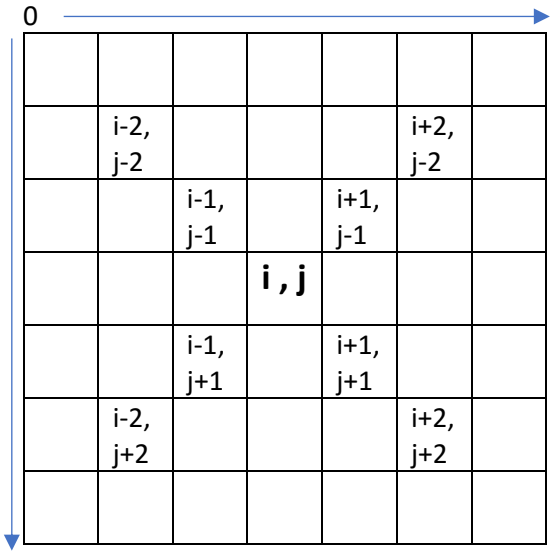
1/ Floutage d'une photo

On souhaite obtenir ceci :

Une idée est -pour chaque pixel- **faire la moyenne** des pixels environnants sur leur diagonale. La photo ci-dessus a été réalisée à l'aide des huit plus proches pixels selon leurs diagonales.

Voici un exemple de programme considérant les quatre plus proches pixels selon les diagonales :

Coordonnées des pixels (i ,j)



Exemple de programme

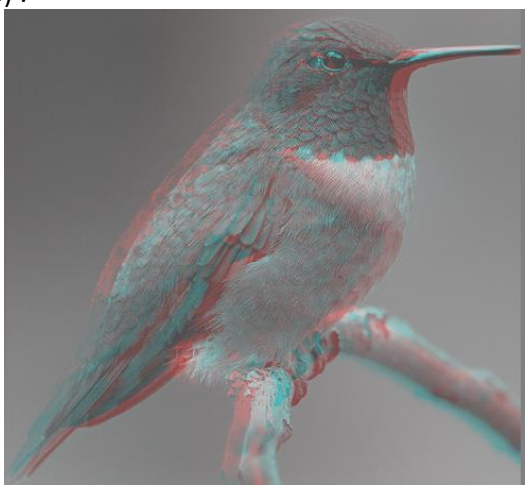
```
imgBase = Image.open("perceval.png")
width,height = imgNew.size

imgNew = Image.new("RGB",(width,height))

# Attention aux indices !
    for j in range(2,imgBase.size[1]-2):
        for i in range(2,imgBase.size[0]-2):

            rouge, vert, bleu = imgBase.getpixel((i,j))

            # Récupération des 4 pixels les plus proches (diagonales)
            rouge1,vert1,bleu1=imgBase.getpixel((i-1,j-1))
            rouge2,vert2,bleu2=imgBase.getpixel((i-1,j+1))
            rouge3,vert3,bleu3=imgBase.getpixel((i+1,j-1))
            rouge4,vert4,bleu4=imgBase.getpixel((i+1,j+1))

            # Moyenne des 9 pixels
            moy_rouge = int((rouge+rouge1+rouge2+rouge3+rouge4)/5)
            moy_vert = int((vert+vert1+vert2+vert3+vert4)/5)
            moy_bleu = int((bleu+bleu1+bleu2+bleu3+bleu4)/5)

            imgNew.putpixel((i,j),(moy_rouge,moy_vert,moy_bleu))
```

A noter : ce type de modifications d'images peut être effectué à l'aide de matrices avec la bibliothèque `*numpy*` du langage python. Ne pas hésiter à s'y intéresser.

Remarque : pour augmenter le flou, il suffit de faire la moyenne entre plus de pixels selon les diagonales. On fera attention aux indices.

2/ Principe de l'anaglyphe

On souhaite obtenir ceci (ici avec un éclaircissement de 150) :



C'est une image créée pour être vue en relief, à l'aide de deux filtres de couleurs différentes complémentaires (rouge et cyan par exemple posées sur des lunettes 3D) disposés devant chacun des yeux de l'observateur. Ce principe est fondé sur la notion de stéréoscopie qui permet à notre cerveau d'utiliser le décalage entre nos deux yeux pour percevoir le relief.

Une paire de lunettes 3D rouge/cyan

Source : Wikipédia

Plus d'informations ici : http://photo.stereo.free.fr/stereoscopie/stereoscopie-principe.php

Le couple de couleurs (rouge / cyan) est adapté pour voir les teintes vertes (photos de la nature par exemple) mais pas pour teintes rouges !
On prendra donc soin de mettre **les photos colorées en teintes de gris** pour éviter ce problème. On **éclaircira** également la photo pour qu'elle ne soit pas trop sombre.

Plan de l'algorithme

- Choisir une photo de base.
- La transformer en teintes grisées.
- Créer trois images vierges : une pour la photo rouge, l'autre la cyan et la troisième qui sera la moyenne des deux premières et sera la photo finale.
- Photo rouge :
  - Décaler à gauche chaque pixel de *d_pixels* (valeur autour de 10 à préciser) si possible.
  - Chaque pixel a pour couple (gris, light, light), *gris* est la valeur du pixel de la photo en teintes grisée et *light* une valeur entre 0 et 255 pour régler la luminosité de la photo.
- Photo cyan :
  - Décaler à droite chaque pixel (valeur autour de 10 à préciser) si possible.
  - Chaque pixel a pour couple (light, gris, gris), *gris* est la valeur du pixel de la photo en teintes grisée et *light* une valeur entre 0 et 255 pour régler la luminosité de la photo.
- Photo finale :
  - Prendre largeur_photo_base – 2*d_pixels comme largeur de photo.
  - Chaque pixel est la moyenne des pixels des photos rouge et cyan.
- Afficher le résultat.

Voici un exemple de programme :

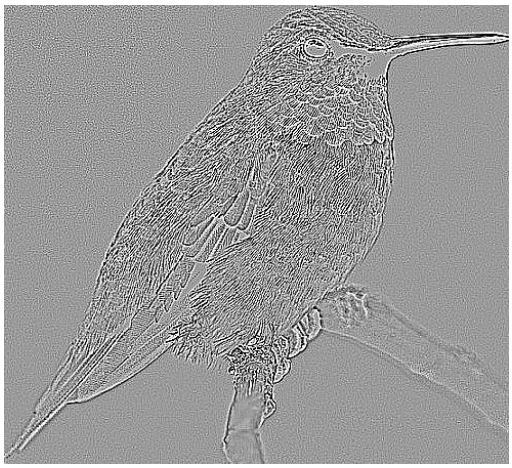
```
# Décalage en pixels pour provoquer l'effet anaglyphique, à tester
img_decal = 8

# Pour éclaicir l'image, à  tester pour la valeur
# entre 0 et 255
img_light = 150

width,height = img1.size

# Création des images à  composante rouge, cyan et la future image en anaglyphe
img_rouge = Image.new("RGB",img1.size)
img_cyan = Image.new("RGB",img1.size)
img2 = Image.new("RGB",(width-2*img_decal,height))
```

```
for i in range(width):
    for j in range(height):
        rouge,vert,bleu=img1.getpixel((i,j))

        # Teinte de gris entre 0 et 255
        gris = int(0.299*rouge + 0.587*vert + 0.144*bleu) # Formule empirique pour griser

        # Décaler à gauche de `img_decal` pixels et contrôle d'index
        if i-img_decal > 0 :
            img_rouge.putpixel((i-img_decal,j),(gris,img_light,img_light))

        # Décaler à droite de `img_decal` pixels et contrôle d'index
        if i+img_decal < width :
            img_cyan.putpixel((i+img_decal,j),(img_light,gris,gris))


# Remplissage de l'image pour l'anaglyphe
for i in range(width-2*img_decal):
    for j in range(height):
        r_img_rouge,v_img_rouge,b_img_rouge = img_rouge.getpixel((i+img_decal,j))
        r_img_cyan,v_img_cyan,b_img_cyan = img_cyan.getpixel((i+img_decal,j))

        # Composition finale des deux images red-cyan
        r_img2,v_img2,b_img2 = int(0.5*(r_img_rouge+r_img_cyan)),int(0.5*(v_img_rouge+v_img_cyan)),int(0.5*(b_img_rouge+b_img_cyan))

        img2.putpixel((i,j),(r_img2,v_img2,b_img2))
```

3/ Détection de contours

L'idée est de repérer les ruptures de couleurs c'est-à-dire les différences de valeurs des canaux entre les pixels voisins. On travaillera également en teintes grisées ici.
On peut proposer que si elles sont importantes, les ruptures de couleurs seront mises en valeur par une du noir du blanc sinon il s'agira de gris « moyen »

On souhaite obtenir ceci :



Une idée de de calculer la différence entre la valeur du pixel et celles des 8 pixels environnants, on multipliera par 8 la valeur du pixel central.

Voici un exemple de programme :

```
# Création d'une nouvelle image ayant les mêmes
# dimensions que celle chargée
img2 = Image.new("RGB",img1.size)

for j in range(1,img1.size[1]-1):
    for i in range(1,img1.size[0]-1):

        l_colors = []

        l_colors.append( img1.getpixel((i,j)) )
        l_colors.append( img1.getpixel((i-1,j-1)) )
        l_colors.append( img1.getpixel((i,j-1)) )
        l_colors.append( img1.getpixel((i+1,j-1)) )
        l_colors.append( img1.getpixel((i-1,j)) )
        l_colors.append( img1.getpixel((i+1,j)) )
        l_colors.append( img1.getpixel((i-1,j+1)) )
        l_colors.append( img1.getpixel((i,j+1)) )
        l_colors.append( img1.getpixel((i+1,j+1)) )
        # Teintes de gris
        l_gris = []
        for colors in l_colors :
            l_gris.append(int(0.299*colors[0] + 0.587*colors[1] + 0.144*colors[2]))

        # On calcule la différence de teinte entre le pixel et les 8 plus proches voisins
        col_contours = 8*l_gris[0]

        for count in range(1,len(l_gris)) :
            col_contours -= l_gris[count]

        # Valeur à tester (entre 50 et 200)
        # pour mettre en valeurs les ruptures de couleurs
        col_add_valeur = 120

        col_contours += col_add_valeur

        # Si peu de variations de teintes, mettre du grisé (dépend de color_add_valeur)
        col_contours = 255-col_contours

        img2.putpixel((i,j),(col_contours,col_contours,col_contours))
```

4/ Photomaton

Il s'agit de proposer des formats « photos à cartes d'identité ».
Le principe est le suivant :

- Choix d'une photo, si possible des photos dont les mesures sont des multiples de deux (256, 512 notamment).
- Construction de 4 petites photos de dimension de moitié selon ces critères :
    - Photo 1 : contient les pixels de lignes et colonnes paires. A afficher en haut / gauche.
    - Photo 2 : contient les pixels des lignes paires et colonnes impaires. A afficher en haut / droite.
    - Photo 3 : contient les pixels des lignes impaires et colonnes paires. A afficher en bas / gauche.
    - Photo 4 : contient les pixels des lignes et colonnes impaires. A afficher en bas / droite.

Remarque : les 4 petites photos sont **différentes** et permettent de reconstituer la photo de départ.

On souhaite obtenir ceci :

Voici un exemple de programme pour obtenir 4 petites photos :

```
# Création d'une nouvelle image ayant les mêmes
# dimensions que celle chargée
img2 = Image.new("RGB",img1.size)

# Récupération des dimensions de l'image
img_width = img1.size[0]
img_height = img1.size[1]

for j in range(img_height) :
    for i in range(img_width) :

        rouge, vert, bleu = img1.getpixel((i,j))

        # Si `i` et `j` sont pairs
        if not i%2 and not j%2 :
            img2.putpixel( (i//2 , j//2 ) , (rouge,vert,bleu) )
        # Si `i` est pair et `j` impair
        elif not i%2 and j%2 :
            img2.putpixel( (i//2 , j//2  + img_height//2) , (rouge,vert,bleu) )
        # Si `i` est impair et `j` pair
        elif i%2 and not j%2 :
            img2.putpixel( (i//2 + img_width//2, j//2 ), (rouge,vert,bleu) )
        # Si `i` et `j` sont impairs
        elif i%2 and j%2 :
            img2.putpixel( (i//2 + img_width//2, j//2 + img_height//2) , (rouge,vert,bleu) )
```

Autres possibilités :

- Ne pas hésiter à aller chercher sur Internet d'autres transformations possibles de photos.
- Combiner plusieurs transformations.
- Appliquer les transformations sur une partie de l'image (transparence par exemple).

Bon courage à tou(te)s et surtout, faites-vous plaisir avec les manipulations de photos ☺