Informatique 2:
Algorithmique sur les tableaux

4TPU148U

Université de Bordeaux

Équipe enseignante informatique 2

https://moodle1.u-bordeaux.fr/course/view.php?id=10750

2022–2023

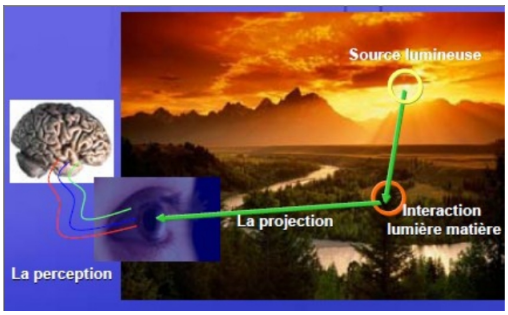
informatique

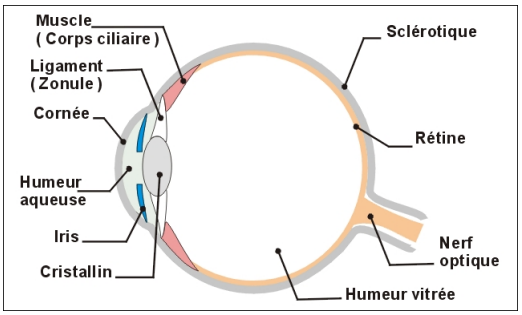
Table des matières **3**

Chapitre 1. Les images

1.1 Formation des images Rappel de physique sur la lumière

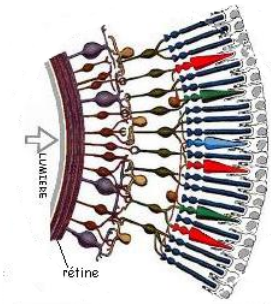
1.1.1 Image et système visuel humain

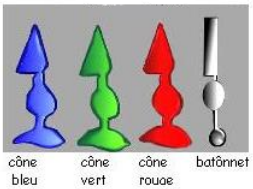




L'œil humain est un globe sphérique d'environ 25 mm. Il comporte de nombreux éléments. L' iris est la membrane colorée qui donne sa couleur à l'œil : en se contractant ou en se dilatant, elle va moduler la quantité de lumière qui traverse le trou percé en son centre appelé pupille. La lumière passe par la cornée traverse l'humeur aqueuse et le cristallin (soutenue par deux muscles) ce qui permet la formation de l'image sur la rétine. Par la suite, la lumière (formée d'ondes électromagnétiques) est convertie, par les photorécepteurs et les neurones (organisés en trois couches) de la rétine, en impulsions électriques. Celles-ci, désormais compréhensibles par le cerveau, lui sont ensuite transmises par l'intermédiaire du nerf optique.

1.1.2 Les cônes et la couleur



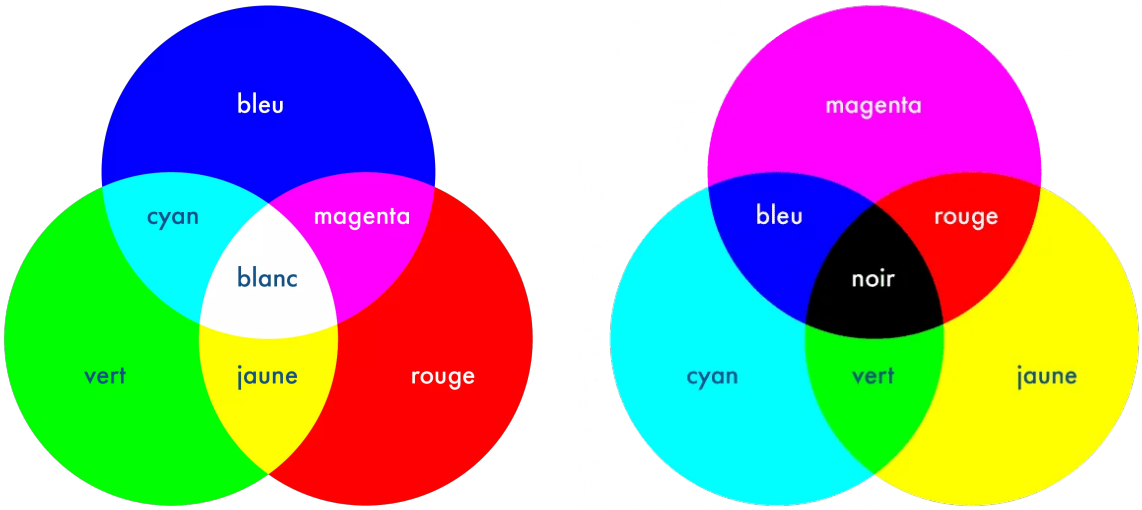


Les cônes sont des photo-récepteurs possédant plusieurs types de pigments (substances chimiques : *l'iodopsine*), caractérisant leur sensibilité à des longueurs d'onde différentes. Les cônes sont au nombre de 6 à 7 millions et leurs trois sortes de pigments permettent la vision de la couleur.

- cyanolabe 420 nm $\longrightarrow$ bleu
- chlorolabe 535 nm $\longrightarrow$ vert
- érythrolabe 570 nm $\longrightarrow$ jaune-rouge

1.1.3 La couleur

La couleur telle que nous la percevons est un phénomène complexe qui lie physique, chimie et système visuel : œil + cerveau.



(a) Synthèse additive : production de lumière (b) Synthèse soustractive : absorption de lumière

FIGURE 1.1 –

1.2 Images numériques

Une image, en informatique, est un simple tableau à deux dimensions de points colorés appelés **pixels**. Les **coordonnées** (x, y) d'un pixel expriment sa position au sein de l'image : x est son abscisse, en partant de la gauche de l'image, et y est son ordonnée, en partant du haut de l'image (attention à l'inverse de l'ordonnée mathématique). Elles partent toutes deux de 0. Le schéma ci-dessous montre un exemple d'image en gris sur fond blanc, de 7 pixels de largeur et 4 pixels de hauteur, les abscisses vont donc de 0 à 6 et les ordonnées vont de 0 à 3.

0,0	1,0	2,0	3,0	4,0	5,0	6,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3

La **couleur RGB** (r, g, b) d'un pixel est la quantité de rouge (r pour *red*), vert (g pour *green*) et de bleu (b pour *blue*) composant la couleur affichée à l'écran. Le mélange se fait comme trois lampes colorées rouge, vert et bleue, et les trois valeurs r, g, b expriment les intensités lumineuses de chacune de ces trois lampes, exprimées entre 0 et 255. Par exemple, $(0, 0, 0)$ correspond à trois lampes éteintes, et produit donc du noir. $(255, 255, 255)$ correspond à trois lampes allumées au maximum, et produit donc du blanc. $(255, 0, 0)$ correspond à seulement une lampe rouge allumée au maximum, et produit donc du rouge. $(255, 255, 0)$ correspond à une lampe rouge et une lampe verte allumées au maximum, et produit donc du jaune. Et ainsi de

suite. Cela correspond très précisément à ce qui se passe sur vos écrans ! Si vous prenez une loupe, vous verrez que l'écran est composé de petits points (appelés *sous-pixels*) verts, rouges et bleus, allumés plus ou moins fortement pour former les couleurs.

1.2.1 Notions de base sur les images Python

Pour manipuler les images avec python nous utiliserons *python Imaging Library* (PIL), qui est préinstallée sur vos machines. Utiliser l'instruction suivante pour importer PIL

```
1 from PIL.Image import *
```

On dispose alors d'un ensemble de fonctions pour manipuler des images. Voici un résumé des fonctions que nous utiliserons dans ce chapitre :

`open(nom)`	Ouvre le fichier *nom* et retourne l'image contenue dedans (par exemple `open("lion.png")`.
`save(nom)`	Sauvegarde l'image *img* dans le fichier *nom*.
`new("RGB",(large,haut))`	Retourne une image de taille $large \times haut$, initialement noire.
`show(img)`	Affiche l'image *img*.
`putpixel((x,y), (r,g,b))`	Peint le pixel (x, y) dans l'image *img* de la couleur (r, g, b)
`getpixel((x,y))`	Retourne la couleur du pixel (x, y) dans l'image *img*
`size`	Retourne un tuple (largeur, hauteur) de l'image

Les bouts de code suivants pourront être testés avec l'image `lion.jpeg` contenue dans le dossier `TD_machine/images` du github.

Exercice 1.2.1 Exécuter les instructions suivantes :

```
1 YELLOW = (255, 255, 0)
2 img1 = Image.new("RGB", (300, 200))
3 j = 100
4 for i in range(img1.size[1]):
5     img1.putpixel((i, j), YELLOW)
6 img1.show()
```

Expliquer ce qu'effectue chaque instruction, et observer l'image produite. Confirmer ainsi le sens dans lequel fonctionnent les coordonnées. Modifiez le code pour que la ligne jaune traverse la fenêtre.

Exercice 1.2.2 Écrire une fonction : `solidRectangle(img, x1, x2, y1, 2, color)` qui prend en paramètres une image, la position du coin supérieur gauche (x1, y1) et du coin inférieur droit (x2, y2) du rectangle, sa couleur et qui dessine un rectangle plein.

Exercice 1.2.3 Écrire une fonction `filtreRouge(img)` qui, pour chaque pixel de l'image, ne conserve que la composante rouge. Testez-la sur la photo `livre.jpeg`. Faites de même pour le vert et le bleu. Que se passe-t-il si on applique successivement un filtre rouge puis un filtre bleu sur la même photo ? Vérifiez votre hypothèse.

1.3 Niveau de gris et modification des couleurs

Il arrive souvent qu'une photo ne corresponde pas à nos souhaits. De nombreux paramètres entrent en jeu lors de la prise d'une photo. Les plus simples à corriger sont les couleurs, la luminosité et le contraste d'une image.

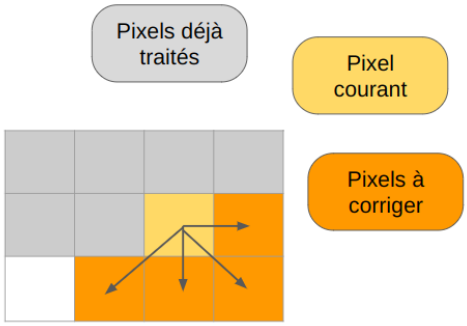
Exercice 1.3.1 Il paraît facile de convertir une photo couleur en photo en noir et blanc. Écrire une fonction `monochrome(img)` qui pour chaque pixel, calcule la moyenne $lum = \frac{r+g+b}{3}$ des composantes r, g, b, et peint le pixel de la couleur (lum, lum, lum). Tester votre fonction sur l'image de la synthèse additive (figure 1.1).

Les éléments verts semblent cependant plus foncés que sur la photo d'origine. L'œil humain est effectivement peu sensible au bleu et beaucoup plus au vert. Une conversion plus ressemblante est d'utiliser plutôt $lum = 0.3 * r + 0.59 * g + 0.11 * b$. Essayez, et constatez que les éléments verts ont une luminosité plus fidèle à la version couleur.

Exercice 1.3.2 Le seuillage est une opération qui permet de transformer une image en niveau de gris en image binaire (noir et blanc), l'image obtenue est alors appelée masque binaire. Il existe 2 méthodes pour seuiller une image, le seuillage manuel et le seuillage automatique. En pratique, le seuil optimal est considéré comme étant celui qui sépare le mieux le fond et les objets présents sur l'image. Écrire une fonction `binarisation(img)` qui calcule la moyenne des intensités des pixels d'une image en niveaux de gris et qui renvoie une image en noir et blanc avec en blanc tous les pixels dont la valeur de l'intensité est en dessous de la moyenne et en noir les pixels dont l'instensite est au dessus de la moyenne.

Exercice 1.3.3 Comme vous l'avez remarqué dans l'exercice précédent, la qualité des images produites par la fonction `binarisation` laisse à désirer. L'algorithme proposé par Floyd et Steinberg permet de limiter la perte d'information. Ecrire une fonction `floydSteinberg(img)` qui convertit une image en niveaux de gris en une image en noir et blanc. L'algorithme est décrit par les transformations d'intensité ci-dessous :

- $I(i+1, j) = I(i+1, j) + \frac{7}{16}e$
- $I(i+1, j+1) = I(i+1, j+1) + \frac{3}{16}e$
- $I(i, j+1) = I(i, j+1) + \frac{5}{16}e$
- $I(i-1, j+1) = I(i-1, j+1) + \frac{1}{16}e$
- $e = I(i, j) - 0$ si $I(i, j) <=$ seuil
- $e = I(i, j) - 1$ si $I(i, j) >$ seuil



Exercice 1.3.4 Le filtre moyenneur est une opération de traitement d'images utilisée pour réduire le bruit dans une image et/ou flouter une image. Le principe est très simple : un pixel est remplacé par la moyenne de lui-même et de ses voisins. C'est dans la définition du voisinage que tout réside. Concrètement, avec un filtre moyenneur de largeur 3, pour calculer la nouvelle valeur du pixel rouge de l'image originale de gauche, on calcule la valeur moyenne (pixel traité compris) des pixels situés dans un carré de dimension 3x3 centré sur ce pixel. Cela donne la nouvelle valeur du pixel sur l'image transformée :

127	127	127	127
127	127	127	127
127	127	127	127

127	183	96	169
162	93	180	85
104	180	93	181

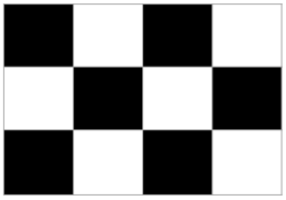


FIGURE 1.2 – transformation de Floyd-Steinberg : [0, 127] —> noir et [128, 255] —> blanc

24	32	234	255	123
44	122	34	200	12
5	167	121	221	202
240	232	128	155	98
124	132	58	167	107

		153		

$$\frac{122 + 34 + 200 + 167 + 121 + 221 + 232 + 128 + 155}{9} = 153$$

Écrire une fonction `blur(img)` qui pour chaque pixel n'étant pas sur les bords, le peint de la couleur moyenne des pixels voisins. Que pensez-vous de votre flou ? Proposer une idée d'amélioration et l'implémenter

1.3.1 Ajustement linéaire de la luminosité et du contraste

On peut ajuster la valeur de la luminosité d'une image en niveau de gris en ajoutant une valeur b à l'intensité du pixel. Si on repère dans le plan la position d'un pixel par ses coordonnées (x, y) on a :

$$J(x, y) = I(x, y) + b$$

- si $b > 0$ la luminosité augmente
- si $b < 0$ la luminosité diminue

(a) Lion trop sombre (b) on ajoute 100 à l'intensité de chaque pixel

FIGURE 1.3 –

La valeur b est ajoutée à chaque pixel de l'image. On augmente dans ce cas l'intensité du niveaux de gris sur tous les points de l'image quelque soit la valeur de départ.

De manière similaire, on peut ajuster le contraste d'une image à l'aide d'un coefficient a sur l'intensité des pixels.

$$g(x, y) = a \times f(x, y)$$

Contrairement à la luminosité tous les pixels ne seront pas affectés de la même façon. Un pixel dont la valeur initiale en intensité de gris est de 0 aura une valeur finale de 0. En fait plus la valeur d'intensité de départ est grande et plus l'écart d'intensité sera grand. En combinant contraste et luminosité on obtient :

$$J(x, y) = a \times I(x, y) + b$$

(a) Image de départ (b) On multiplie par $1,6$ l'intensité de chaque pixel

FIGURE 1.4 –

Exercice 1.3.5 Écrire une fonction `ajustementLinéaire(img, a, b)` qui prend en paramètres une image un coefficient b de luminosité, un coefficient a de contraste et qui renvoie une image ayant subi les modifications de luminosité et/ou de contraste.

Histogramme d'une image

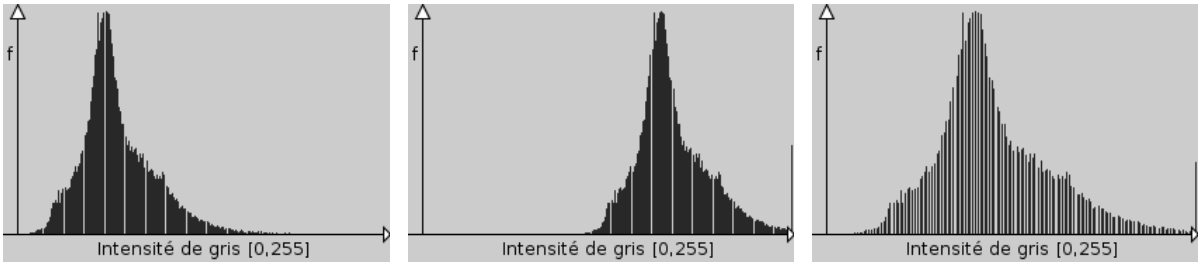
Dans une image en niveaux de gris l'histogramme comptabilise le nombre d'occurence (en ordonnées) de chacune des valeurs d'intensité de 0 à 255 (en abscisses) quand l'image est codée sur 8 bits. Un histogramme permet d'obtenir des informations sur la répartition des intensités comme par exemple la valeur moyenne. Par contre un histogramme ne fournit aucune information de répartion spatiale. Deux images peuvent posséder le même histogramme et être totalement différentes. Certaines modifications d'histogrammes permettent d'améliorer le contraste global d'une image ou encore d'améliorer le contraste pour certaines plages de niveaux de gris. On part d'une image $I(x, y)$ codée sur 8 bits dont on peut calculer un histogramme $h(n)$ où n correspond au niveau de gris allant de 0 à 255 (figure 1.6). On peut également calculer l'histogramme cumulé :

$$h_c(n) = \sum_{i=0}^{n} h(n) \quad \forall \in [0, 255]$$

Exercice 1.3.6 Écrire une fonction `histogramme(img)` qui prend en paramètre une image et qui renvoie l'histogramme de cette image sous la forme d'un tableau.

(a) Image initiale (b) Luminosité modifiée (c) Contraste modifié

FIGURE 1.5 –



(a) Image initiale (b) Luminosité modifiée (c) Contraste modifié

FIGURE 1.6 –

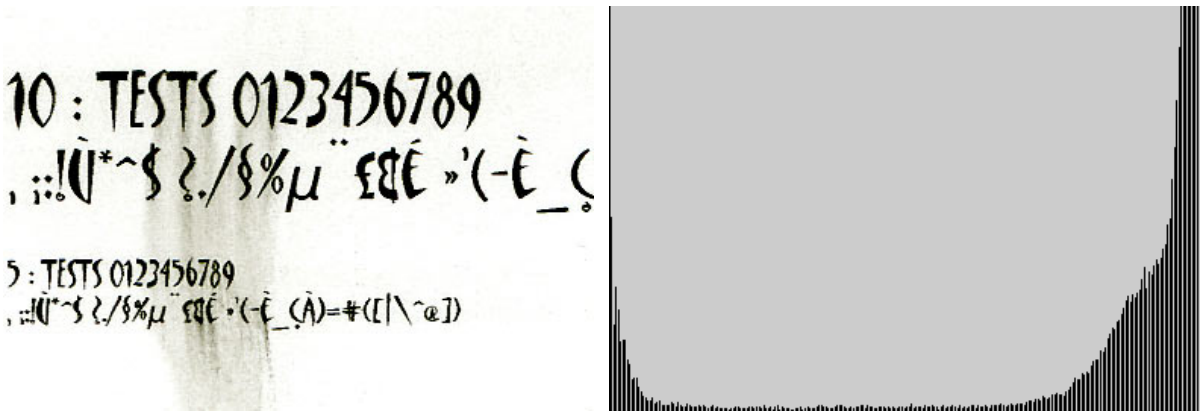
Exercice 1.3.7 La normalisation ou étirement d'histogramme consiste à exploiter les 256 valeurs de niveau d'intensité des canaux (r,g,b). Cela revient à une simple normalisation qui consiste à ramener la valeur max à 255 et la valeur min à 0. Écrire une fonction `normalisationSimple(img)` qui prend en paramètre une image et renvoie une image normalisé.

Exercice 1.3.8 La normalisation de l'exercice précédent est très sensible au bruit. On lui préfère une normalisation qui exploite les statistiques globales de l'image tel que la moyenne d'intensité μ et l'écart-type σ qui sera ensuite suivie d'un nouvel étirement pour ramener les valeurs entre 0 et 255. Écrire une fonction `normalisationGlobale(img)` qui prend en paramètre une image et renvoie une image normalisé.

$$J(x,y) = \frac{I(x,y) - \mu}{\sigma}$$

Exercice 1.3.9 Un exemple d'application est la suppression d'un fond indésirable. Sur l'image de la figure 1.7, l'histogramme permet de mettre en évidence deux zones de niveaux de gris bien distinctes, le texte (niveau proche de 0) et la bavure laissée à la photocopieuse (niveau proche de 255). Écrire une fonction `nettoyerFondIndesirable(img)` L'histogramme permet d'en déduire une valeur seuil de niveau de gris pour supprimer la bavure.

Exercice 1.3.10 La soustraction d'images est utilisée pour mettre en évidence certains détails caractéristiques qui peuvent évoluer au cours du temps (figure 1.8). On effectue alors la différence $g(x,y) = f_2(x,y) - f_1(x,y)$ qui permet d'extraire les détails intéressants. Écrire une fonction `soustraire(img1, img2)` qui met en évidence les différences entre deux images.



(a) Bavure de photocopieuse (b) Histogramme

FIGURE 1.7 –

(a) Image initiale (b) Image initiale avec deux voitures (c) Détection des deux voitures

FIGURE 1.8 –

1.3.2 Exercices complémentaires

Exercice 1.3.11 Une définition de la **droite discrète arithmétique** a été donnée en 1991 par J-P REVEILLES.

Un ensemble de pixels $\mathcal{E}$ appartient à la droite discrète arithmétique(figure 1.9) de pente $\frac{a}{b}$, de borne inférieure μ et d'épaisseur ω (avec a, b, μ et ω dans $\mathbb{Z}, b \neq 0$ et $pgcd(a, b) = 1$) si et seulement si tous les pixels (x, y) de $\mathcal{E}$ vérifient :

$$\mu \leq ax - by < \mu + \omega$$

Cette droite se note $\mathcal{D}(a, b, \mu, \omega)$

1.4 Transformations géométriques 2D

1.4.1 Transformations euclidiennes

Une transformation géométrique permet de déplacer un pixel d'une position (x, y) vers une nouvelle position (x', y') à l'aide d'équations d'écrivant la transormation.

$$\begin{cases} x' & = T_x(x, y) \\ y' & = T_y(x, y) \end{cases}$$

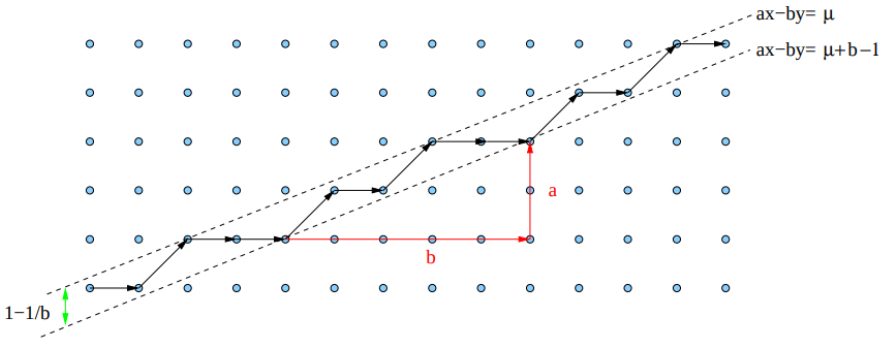


FIGURE 1.9 – Droite de Reveilles de paramètres $\mathcal{D}(2, 5, -1, 5)$

Parmi les transformations géométriques, on peut citer la translation et la rotation. Elles permettent par exemple de stabiliser les images d'une séquences vidéo. Lorsque vous tenez votre caméra à la main de faibles translations et rotations peut donner le mal de mer à quiconque regarde le film non corrigé. Pour corriger ces mouvements parasites, on effectue un ensemble de transformations permettant d'éliminer les mouvements parasites.

Exercice 1.4.1 On souhaite ouvrir la porte gauche d'un placard. Écrire une fonction
`def translatePixelToLeft(img, xi, yi, xf, yf, value)`
qui prend en paramètre une image, le cadre à décaler avec le pixel en haut à gauche (xi, yi) et le pixel en bas à droite (xf, yf) du cadre ainsi que la valeur de décalage. Pour la porte de gauche les coordonnées de ces deux pixels sont (71, 4) à (131, 186)) et le décalage est de 64 pixels. Vous pourrez remplir l'espace libéré avec de la couleur, ou avec l'image `fantome.png`.

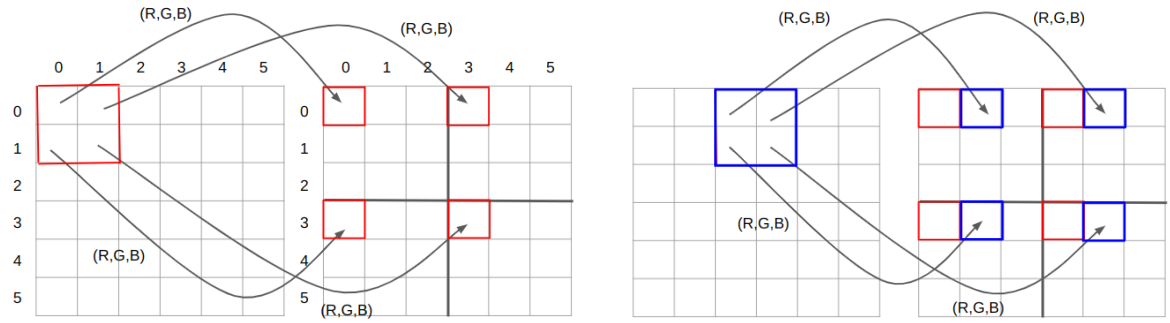
Exercice 1.4.2 Même question en entrouvrant la porte de droite, des pixels $(135, 4)$ à $(196, 186)$, en la décalant de seulement 32 pixels vers la droite.

Exercice 1.4.3 L'objectif de cet exercice est d'écrire le code de deux fonctions Python permettant d'effectuer la transformation miroir (symétrie axiale) d'une image.

1. Écrire la fonction `miroirVertical(img1, img2)` qui place dans `img2` l'image correspondant au miroir vertical de `img1` (`img1` et `img2` sont supposées de même taille).

2. Écrire la fonction `miroirHorizontal(img1, img2)` qui crée et retourne une nouvelle image correspondant au miroir horizontal de `img`.

Exercice 1.4.4 On appelle transformation bijective d'image la transformation d'une image finie de $n \times m$ pixels sur elle-même : chaque pixel est donc déplacé et aucun pixel n'est perdu, ce qu'on appelle en mathématiques une permutation de l'ensemble des pixels. Par exemple, la transformation de l'image $(n \times m)$ qui déplace le pixel (i, j) en $((i + 1)\%n, j)$, correspond à un décalage d'un pixel vers la droite de l'image.
Une transformation connue est celle du **photomaton**. Le principe est le suivant : l'image de départ est découpée en blocs de 4 pixels. Le premier bloc en haut à gauche comporte les pixels encadrés en rouge (figure 1.10). Le pixel en haut à gauche $(0, 0)$ est recopié dans la même position $(0, 0)$, le pixel $(1, 0) \longrightarrow (3, 0)$, le pixel $(0, 1) \longrightarrow (0, 3)$ et le pixel $(1, 1) \longrightarrow (3, 3)$. Puis on recommence avec le bloc de pixels suivants (en bleu sur la figure 1.10), et ainsi de suite jusqu'au dernier bloc en bas à droite de l'image de départ. Écrire une fonction `photomaton(img)` qui prend en paramètre une image et qui renvoie une image *photomaton*



(a) Transformation bijective du bloc rouge

(b) Transformation bijective du bloc bleu

FIGURE 1.10 – Transformation bijective photomaton, l'image doit comporter un nombre pair de lignes et de colonnes

Exercice 1.4.5 Écrire une fonction `rotateV1(img, angle)` qui prend chaque pixel de l'image de départ et qui calcule par rapport à l'origine la position du pixel sur l'image d'arrivée. Cette fonction prend en argument l'image de départ, l'angle de rotation et renvoie l'image après transformation

$$\begin{cases} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{cases}$$

Exercice 1.4.6 Écrire une fonction `rotateV2(img, angle)` qui prend chaque pixel de l'image d'arrivée et qui calcule le pixel correspondant sur l'image de départ. Cette fonction prend en argument l'image de départ, l'angle de rotation et renvoie l'image après transformation.

$$\begin{cases} x &= x'\cos\theta + y'\sin\theta \\ y &= -x'\sin\theta + y'\cos\theta \end{cases}$$

Exercice 1.4.7 De manière générale une rotation est déterminée par un centre $C(x_c, y_c)$ et un angle θ. Si un point (x, y) subit une rotation d'angle θ et de centre C, sa nouvealle position est :

$$\begin{cases} x' &= x_c + (x - x_c)\cos\theta - (y - y_c)\sin\theta \\ y' &= y_c + (x - x_c)\sin\theta + (y - y_c)\cos\theta \end{cases}$$

En déduire la vesion `rotateV3(img, angle, xc, yc)` ou les paramètres `xc` et yc représentent les coordonnées du centre de rotation.

1.5 La segmentation couleur

La segmentation va consister à regrouper les pixels de l'image en régions (composantes connexes) qui vérifient un critère d'homogénéité (par exemple sur la couleur) afin d'obtenir une description de l'image en régions homogènes. Une application simple à réaliser est l'incrustation d'image.

(a) Poisson sur fond vert

(b) Incrustation du poisson

FIGURE 1.11 –