Notes de cours
Semaine 14

Cours Turing

1 Les couleurs

La couleur n'est pas une propriété physique de la matière, car elle est une interprétation du monde effectuée par notre cerveau et à l'aide de nos yeux. La rétine tapisse le fond de notre œil, elle est sensible à la lumière. Dans la rétine, on trouvera les bâtonnets et cônes. Les bâtonnets nous permettent de voir lorsque la luminosité est faible. Les cônes nous permettent de percevoir les couleurs, il en existe trois types, un nous permet de percevoir le rouge, un autre le vert et un dernier le bleu.

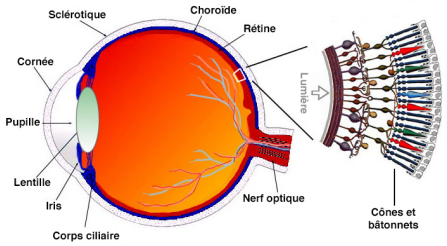

Figure 1: Les cônes se trouvent au fond de la rétine

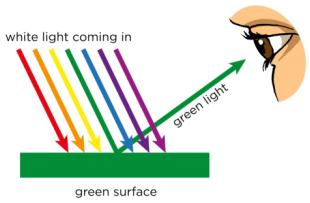

Figure 2: Une couleur est perçue en fonction des composantes réfléchies de la lumière blanche

Une surface sera perçue verte si seule la lumière verte est réfléchie. La propriété physique, qui est donc liée à la couleur, est la capacité d'une surface à absorber certaines composantes de la lumière et à renvoyer d'autres.
En fonction des espèces, la densité de chaque type de cellule dans la rétine peut changer grandement. Par exemple, les animaux chassant la nuit disposent d'une densité bien plus élevée de bâtonnets. À noter que certaines espèces possèdent un quatrième type de cône qui leur permet de percevoir des couleurs que nous ne percevons pas.

2 Système de couleurs

Dans un écran d'ordinateur, derrière chaque pixel se trouve un paquet de trois sources lumineuses, une rouge, une verte et une bleue.

Dans la Figure 3, nous pouvons voir, grâce à un zoom, la structure des pixels sur six types d'écrans. La disposition des sources lumineuses changera en fonction de la technologie d'écran et du constructeur.

L'intensité de chaque source lumineuse est modifiable et en fonction des intensités des sources rouges, vertes et bleues, différentes couleurs seront perçues.

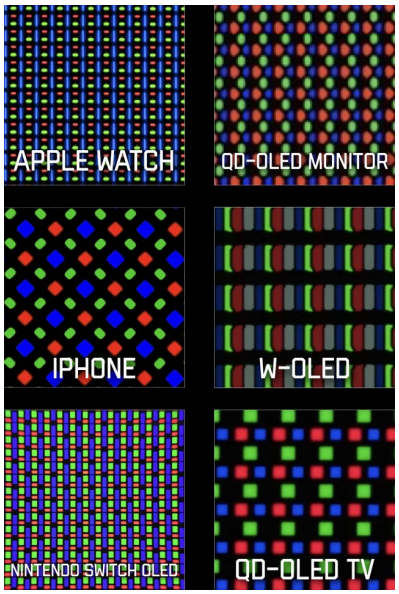

Figure 3: Structure des pixels sur 6 types d'écrans

2.1 RGB(A)

Le système le plus utilisé en informatique est le système RGB (Red, Green, Blue).

L'intensité de chaque source lumineuse est généralement encodée sur 8 bits et donc possède $2^8 = 256$ niveaux d'intensité, ce qui donne accès à $256^3 = 16777216$ de couleurs différentes. Une couleur nécessite donc $8 \cdot 3 = 24$ bits de stockage.

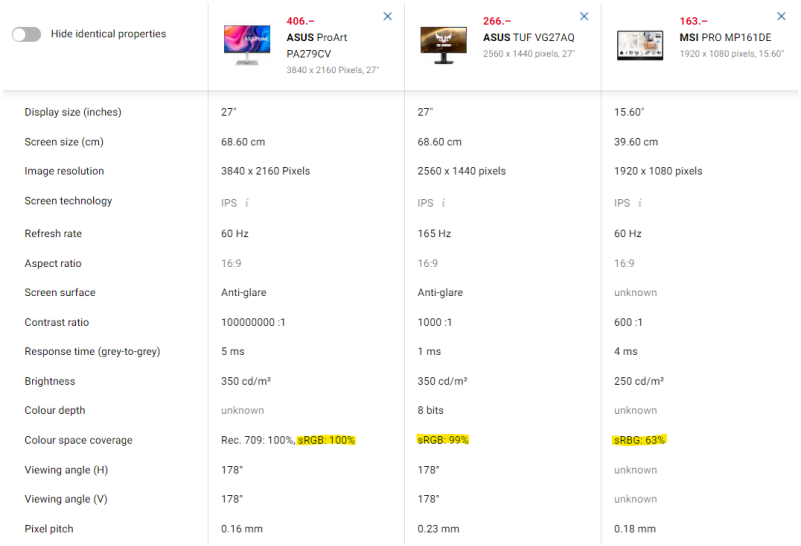

Figure 4: Représentation de la couleur rouge en RGB

En pratique, les écrans ne reproduisent pas les couleurs parfaitement, il existe des métriques pour évaluer la fidélité de reproduction des couleurs. Dans la figure 4, nous pouvons voir les métriques de reproduction des couleurs pour trois écrans.

Name	Color	Code	RGB
white		#ffffff or #fff	rgb(255,255,255)
silver		#c0c0c0	rgb(192,192,192)
gray		#808080	rgb(128,128,128)
black		#000000 or #000	rgb(0,0,0)
maroon		#800000	rgb(128,0,0)
red		#ff0000 or #f00	rgb(255,0,0)
orange		#ffa500	rgb(255,165,0)
yellow		#ffff00 or #ff0	rgb(255,255,0)
olive		#808000	rgb(128,128,0)
lime		#00ff00 or #0f0	rgb(0,255,0)
green		#008000	rgb(0,128,0)
aqua		#00ffff or #0ff	rgb(0,255,255)
blue		#0000ff or #00f	rgb(0,0,255)
navy		#000080	rgb(0,0,128)
teal		#008080	rgb(0,128,128)
fuchsia		#ff00ff or #f0f	rgb(255,0,255)
purple		#800080	rgb(128,0,128)

Figure 5: Couleurs et leurs représentations en RGB

Figure 6: Une image RGBA avec un dégradé de transparence

Dans la Figure 16, nous voyons des exemples de couleurs et de leur encodages RGB. Certaines extensions d'images, comme PNG (Portable Network Graphics), permettent la transparence, la couleur sera donc représentée par une quatrième valeur codée sur 8 bits contrôlant la transparence. On parle de RGBA (Red, Green, Blue, Alpha).

2.2 Autres systèmes de couleurs

En informatique, nous utilisons le système RGB, mais il existe de nombreux autres systèmes de couleurs, par exemple :

- HSV (Hue, Saturation, Value)
- HSL (Hue, Saturation, Lightness)
- YUV (Luminance, Chrominance)

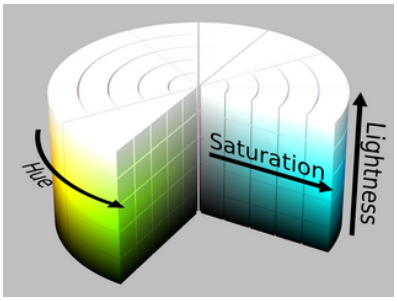

Figure 7: Représentation des couleurs dans le système HSL

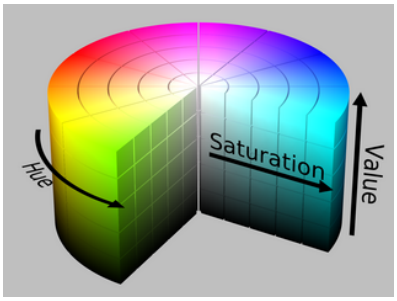

Figure 8: Représentation des couleurs dans le système HSV

Il existe de nombreux autres systèmes de couleurs qui ne sont pas forcément liés aux écrans, comme le CMYK (Cyan, Magenta, Yellow, Key) utilisé pour l'impression.

3 Image

Nous allons nous concentrer sur les images dites matricielles, avec lesquelles nous interagissons en permanence. Il existe d'autres types d'images comme les images vectorielles : il ne s'agit pas d'une extension différente, mais bien d'une approche, un paradigme, entièrement à part.

Figure 9: À gauche, une image matricielle, à droite, une conversion en image vectorielle

Dans la Figure 9, nous voyons qu'il y a une perte de détails lors de la conversion en image vectorielle, cependant une image vectorielle être redimensionnée sans perte de qualité contrairement à une image matricielle.

Les images matricielles peuvent être vues comme des tableaux en deux dimensions où chaque case contient une couleur.

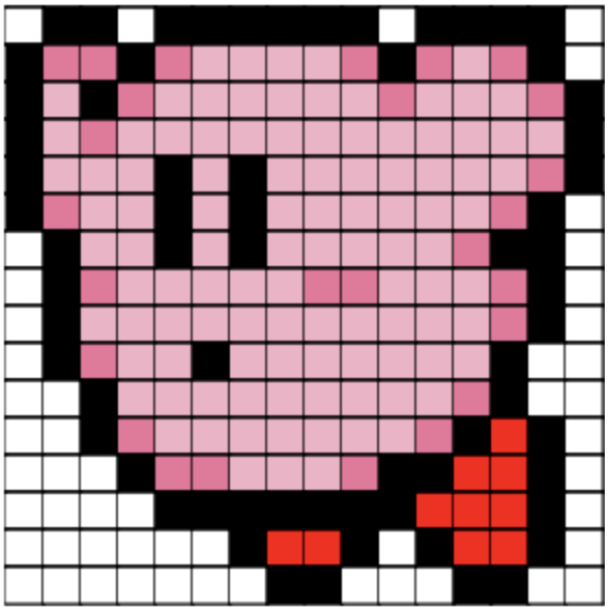

Figure 10: Une image matricielle avec 16 pixels de large et 16 pixels de haut.

4 Extensions d'images

Il existe de nombreux formats d'images, chacun ayant ses avantages et ses inconvénients.

- Images matricielles:
  - BMP
  - PNG
  - JPG
  - GIF
  - ...
- Images vectorielles:
  - SVG
  - eps
  - ...

4.1 Poids théorique d'une image

Une image utilisant le système de couleur RGB avec des intensités encodées sur 8 bits mesurant 1920 pixels de large et 1080 pixels de haut nécessite $1920 \cdot 1080 \cdot 3 \cdot 8 = 49766400$ bits $=$ 6220800 octets $\approx$ 6.22 mégaoctets de mémoire.

D'après Netflix, il faut 5 mégabits par seconde pour regarder un film en HD et les vidéos ont au minimum 24 images par seconde [1].

Une image envoyée par Netflix pèse donc $\frac{5}{24} = 0.208\overline{3}$ mégabits = $0.208\overline{8} \approx 0.026041\overline{6}$ mégaoctets.
C'est donc environ $\frac{6.22}{0.026041\overline{6}} = 238.848$ moins lourd que le poids théorique d'une image.

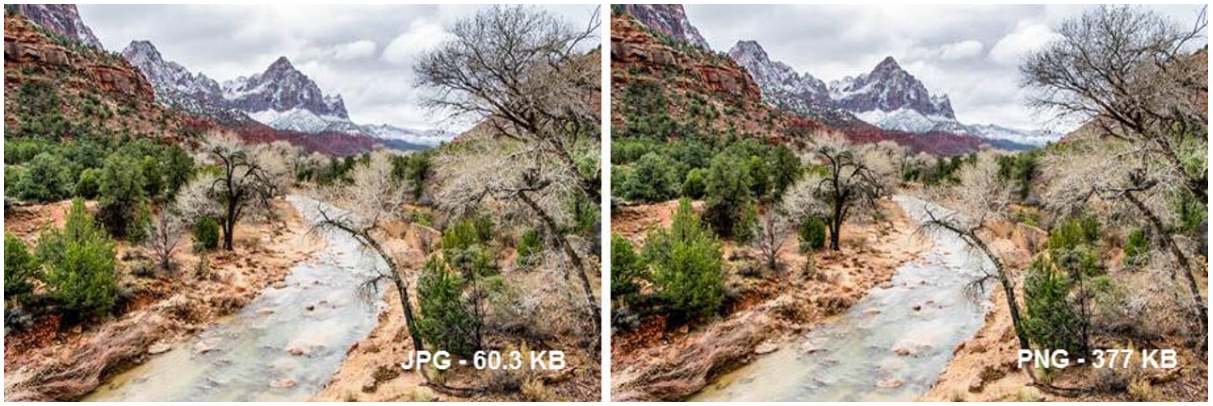

Figure 11: La même image en JPG et en PNG, la différence de poids est due à la technique de compression

Il existe de nombreuses techniques de compression d'images, qui permettent de réduire le poids d'une image avec ou sans perte de qualité. Dans la figure 11, la même image est représentée avec deux extensions très utilisées :

- JPG (Joint Photographic Experts Group) : compression avec perte de qualité
- PNG (Portable Network Graphics) : compression sans perte de qualité

[1] https://netflixtechblog.com/native-frame-rate-playback-6c87836a948

Figure 12: Une image RGBA avec un dégradé de transparence

Normalement, pour des photos, on utilise JPG, car la perte de qualité n'impacte que très peu l'expérience visuelle. Pour des images avec des aplats de couleurs, on utilise PNG, car la perte de qualité est visible comme c'est le cas dans la figure 12.

Il est important d'utiliser le bon format d'image pour le bon usage.

5 Numpy

La librairie **numpy** permet de créer des tableaux multidimensionnels et de les modifier efficacement. Cette librairie sera utilisée conjointement avec `Pillow`.

```
import numpy as np

# Création d'un tableau à partir d'une liste Python
a = np.array([5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])

# Création d'un tableau rempli de 0 (entiers signés sur 8 bits).
b = np.zeros((1920, 1080, 3), dtype=np.uint8)

# Création d'un tableau rempli de 1 (nombres à virgule flottante sur 32 bits).
c = np.ones((1920, 1080, 3), dtype=np.float32)
```

Pillow nous permettra de lire et d'écrire des images (I/O, Input/Ouput, Entrée/Sortie) et **numpy** de les manipuler.

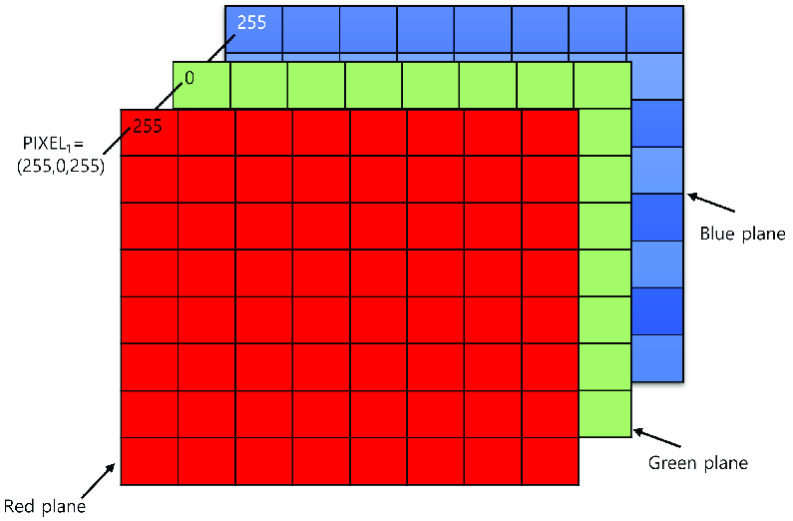

Figure 13: aa

Les images sont souvent perçues comme des tableaux à deux dimensions, mais dans ce chapitre, nous allons les considérer comme des tableaux à trois dimensions :

1. La première dimension correspond à la hauteur de l'image.
2. La deuxième dimension correspond à la largeur de l'image.
3. La troisième dimension correspond aux composantes de couleurs de l'image.

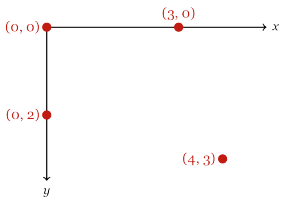

Figure 14: Système de coordonnées de **Pillow**

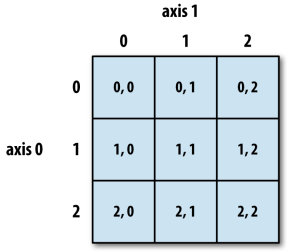

Figure 15: Système de coordonnées de **numpy**

numpy et **Pillow** n'utilisent pas le même ordre des dimensions comme montré dans les figures 14 et 15.

Le pixel (0, 2) dans **Pillow** correspond au pixel (2, 0) dans **numpy**.

5.1 Débordement de capacité

Les intensités lumineuses sont stockées sous forme d'entiers positifs (non signés) sur 8 bits. Il y aura débordement de capacité pour les valeurs plus grandes que 255, car ces valeurs ne sont pas représentables.

Dans le code ci-dessous, **c** contiendra la valeur 144 au lieu de 400, car la représentation binaire de 400 est 110010000 qui nécessite 9 bits, le bit de poids fort est perdu et la valeur est tronquée à 8 bits.
$400 = 256 + 144$.

```
import numpy as np
a = np.array(200, dtype=np.uint8)
b = np.array(2, dtype=np.uint8)
c = a * b # c contiendra 144 au lieu de 400
```

Cela peut causer des effets et/ou inattendus comme montré dans la figure 17.

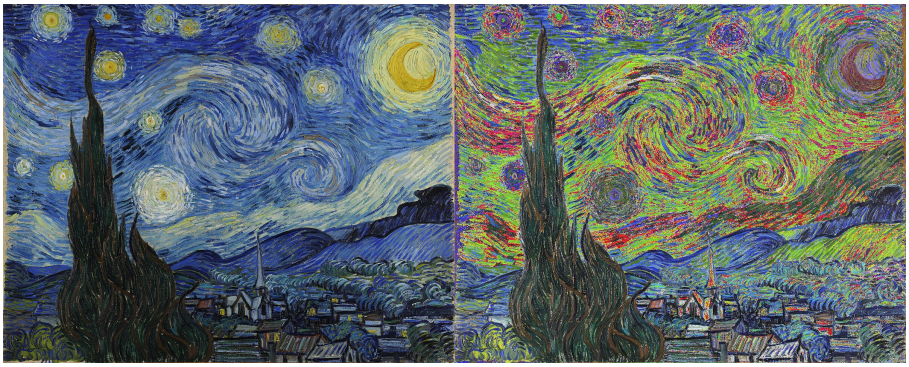

Figure 16: Résultat d'un débordement de capacité après avoir multiplié toutes les valeurs de l'image par 2

5.2 Type de données

```
1 from PIL import Image
2 import numpy as np
3 image_pil = Image.open('input/starry.jpg')
4 image_np = np.array(image_pil)
5 image_np = np.array(image_np) / 2
6
7 im = Image.fromarray(image_np).save('ex0.jpg')
```

! La ligne 5 va modifier le type de données de `image_np` de `uint8` à `float64`.
Pillow ne peut pas sauvegarder ce type de fichier.

Il est possible de spécifier le type de données d'un tableau numpy en utilisant le paramètre `dtype` lors de la création du tableau.

```
image_np = np.zeros((1920, 1080, 3), dtype=np.uint8)
```

Il est également possible de modifier le type de données d'un tableau numpy existant en utilisant la méthode `astype`.

```
image_np = image_np.astype(np.uint8)
```

5.3 Accèder aux valeurs d'un tableau

Pour accéder à une valeur d'un tableau, il faudra spécifier les coordonnées de la valeur à accéder dans des crochets droits et séparer chaque dimension par une virgule.

```
print(image_np[50, 100, 2])
```

Ce code affiche la valeur de la composante bleue du pixel sur la 51ème ligne et la 101ème colonne de l'image.

```
print(image_np[50, 100])
```

Ce code affiche les valeurs RGB de la 51ème ligne et la 101ème colonne de l'image.

```
print(image_np[50])
```

Ce code affiche les valeurs RGB de la 51ème ligne.

5.4 Modification des valeurs d'un tableau

Pour modifier une valeur d'un tableau, il faudra spécifier les coordonnées de la valeur à modifier et utiliser l'opérateur d'affectation =.

```
image_np[50, 100, 2] = 0
```

Ce code modifie la valeur de la composante bleue du pixel sur la 51ème ligne et la 101ème colonne de l'image.

5.5 Slicing

Il est possible d'accéder à une sous-partie de l'image utilisant le *slicing*, en spécifiant des intervalles.
Un intervalle est de la forme $début : fin : pas$ où $début$ est inclus et fin est exclu. Les valeurs sont indexées à partir de 0.

Supposons le tableau monodimensionnel suivant :

```
a = np.array([5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
```

```
# Intervalles où le début est spécifié
print(a[2:9:1]) # Affichera : 7 8 9 10 11 12 13
print(a[2:9:2]) # Affichera : 7 9 11 13
print(a[2:9])   # Affichera : 7 8 9 10 11 12 13
print(a[2:])    # Affichera : 7 8 9 10 11 12 13 14 15

# Intervalles où le début n'est pas spécifié
print(a[:9:1])  # Affichera : 5 6 7 8 9 10 11 12 13
print(a[:9:2])  # Affichera : 5 7 9 11 13

# Intervalles avec des valeurs négatives
print(a[-8:-1:1]) # Affichera : 8 9 10 11 12 13 14
print(a[-8:-1:2]) # Affichera : 8 10 12 14
print(a[-8:-1])   # Affichera : 8 9 10 11 12 13 14
print(a[-8:])     # Affichera : 8 9 10 11 12 13 14 15
print(a[::2])     # Affichera : 5 7 9 11 13 15
print(a[::1])     # Affichera : 5 6 7 8 9 10 11 12 13 14 15
```

Si certaines valeurs ne sont pas spécifiées, elles seront remplacées par les valeurs par défaut suivantes :

- `début` = 0
- `fin` = Longueur du tableau
- `pas` = 1

```
print(np.array_equal(a[2:9], a[2:9:1])) # Affichera True
print(np.array_equal(a[2:], a[2:11:1])) # Affichera True
print(np.array_equal(a[:9], a[0:9:1])) # Affichera True
```

Il est également possible d'utiliser un pas négatif : dans ce cas, les éléments seront parcourus en sens inverse.

```
print(a[9:2:-1]) # Affichera 14 13 12 11 10 9 8
print(a[9:2:-2]) # Affichera 14 12 10 8
```

5.6 Opérations sur les tableaux

Il existe de nombreuses opérations sur les tableaux numpy, notamment :

- `np.sum` : Somme des valeurs du tableau
- `np.mean` : Moyenne des valeurs du tableau
- `np.std` : Écart-type des valeurs du tableau
- `np.min` : Minimum des valeurs du tableau
- `np.max` : Maximum des valeurs du tableau

Ces opérations ne retournent pas un tableau, mais un nombre.

Il existe également des opérations qui retournent un tableau (et appliquent leur opération sur chaque élément du tableau original) :

- `np.abs` : Valeur absolue des valeurs du tableau
- `np.sqrt` : Racine carrée des valeurs du tableau
- `np.square` : Carré des valeurs du tableau
- `np.sin` : Sinus des valeurs du tableau
- `np.cos` : Cosinus des valeurs du tableau
- `np.tan` : Tangente des valeurs du tableau
- ...

Finalement, il existe des fonctions qui prennent en paramètre deux tableaux et retournent un tableau :

- `np.add` : Addition des valeurs des tableaux
- `np.subtract` : Soustraction des valeurs des tableaux
- `np.multiply` : Multiplication des valeurs des tableaux
- `np.divide` : Division des valeurs des tableaux
- `np.logical_and` : ET logique des valeurs des tableaux
- `np.logical_or` : OU logique des valeurs des tableaux
- `np.logical_xor` : OU exclusif logique des valeurs des tableaux
- `np.logical_not` : NON logique des valeurs des tableaux

La documentation complète de **numpy** est disponible ici.