Information, Cacul et Communication

Partie Programmation

Cours 9 : Manipulation d’images

17.11.2023

Patrick Wang

Au sommaire cette semaine

1. Traitement d’images
2. Informations sur le mini-projet

Au sommaire cette semaine

1. Traitement d’images
2. Informations sur le mini-projet

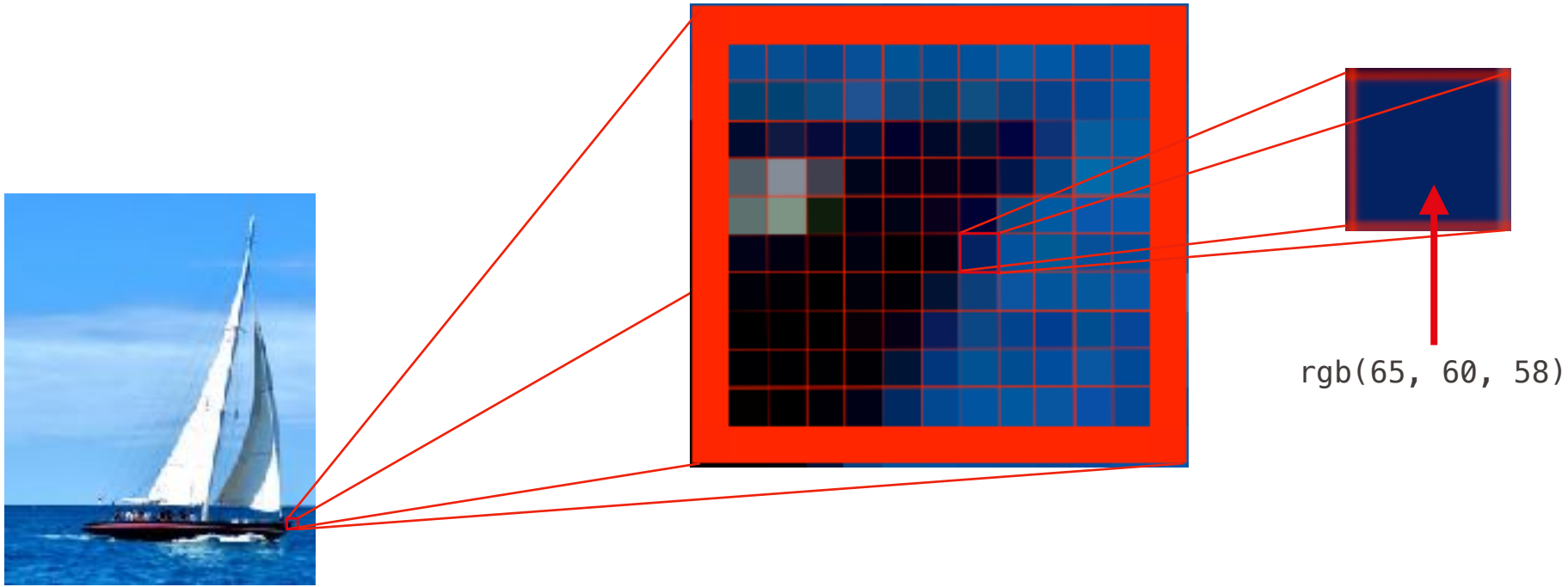
1. Traitement d'images

Images matricielles vs Images vectorielles

- Une image vectorielle est construite à partir de formes géométriques, ce qui lui permet de rester «nette» peu importe le grossissement appliqué sur l'image
- Une image matricielle est construite à partir de pixels (picture element) qui représente la plus petite unité d'information pour ce type d'images.

1. Traitement d'images

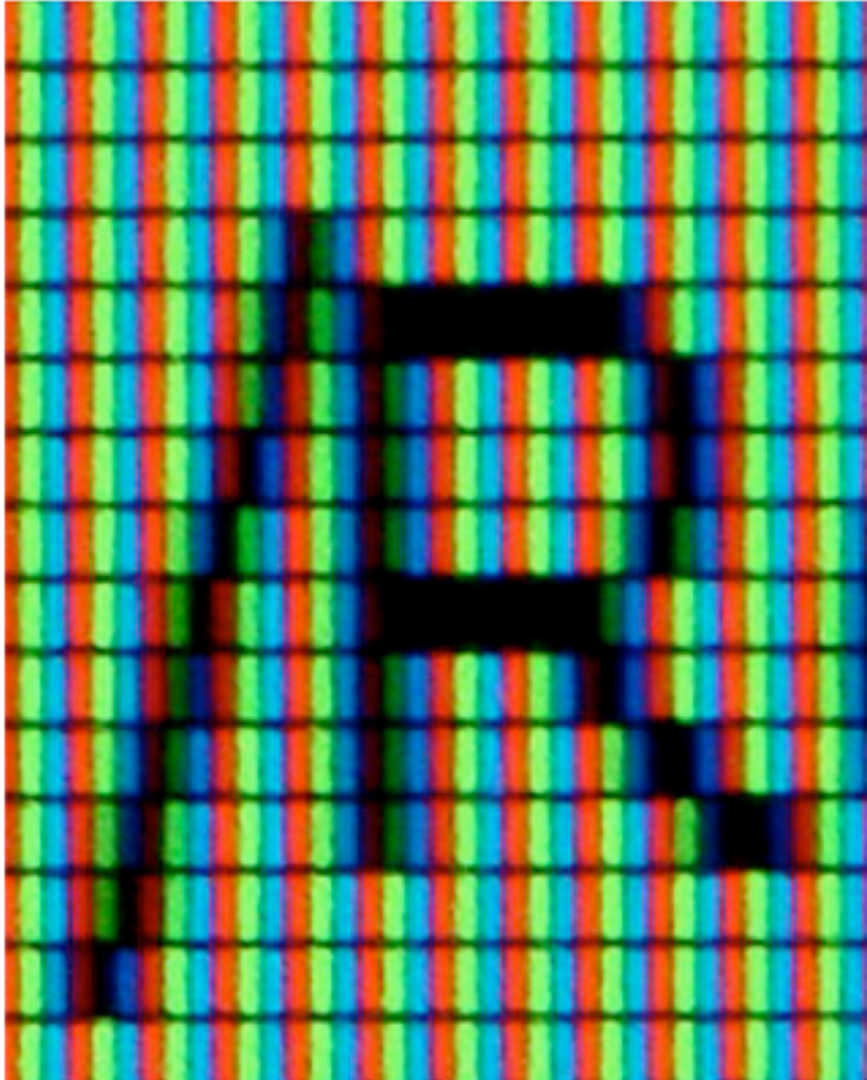
Pixels



1. Traitement d'images

Système RGB (ou RVB)

- Le système rouge-vert-bleu (RGB) est le principal système utilisé pour représenter des couleurs
- L'intensité de chaque couleur est codée sur 8 bits (0 à 255), un pixel est donc représenté par 3x8 bits
- 16'777'216 couleurs possibles



1. Traitement d'images

Le mode *grayscale*

- En nuances de gris (ou grayscale), chaque pixel est désormais représenté par 8 bits
  - 0 = noir
  - 255 = blanc
  - Au milieu : une nuance de gris

1. Traitement d'images

Quelques bibliothèques pour le traitement d'images

- Python est un langage très populaire, mais pas spécialement très performant pour des raisons qui dépassent un peu ce cours...
  - Typage dynamique et gestion de la mémoire moins précis
  - Langage «interprété» qui analyse le code à chaque exécution
- Bibliothèque numpy (Numerical Python)
  - Offre une gestion plus précise des variables numériques (uint8, uint16, etc.)
  - Va nous permettre de traiter plus efficacement des matrices d'octets
- Bibliothèque Pillow pour la gestion de fichiers images et types-Pillow pour les annotations de types

1. Traitement d'images

Création d'une image en niveaux de gris

```python
# On importe toutes les fonctions du module miniprojectutils
from miniprojectutils import *

grayscale_img = new_image_grey(10, 10)
nb_rows, nb_columns = dimensions(grayscale_img)
print(grayscale_img)
save_image(grayscale_img, "grayscale_demo_img.png")

for row in range(nb_rows):
    for column in range(nb_columns):
        grayscale_img[row, column] = row * 25
save_image(grayscale_img, "gradient-lines.png")


for row in range(nb_rows):
    for column in range(nb_columns):
        grayscale_img[row, column] = ((row+1)*(column+1))*2.55
save_image(grayscale_img, "gradient.png")
```

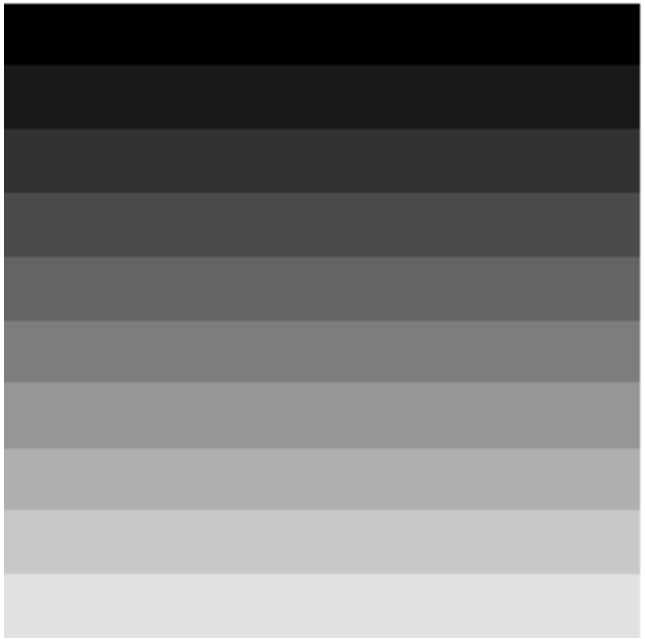
```
[[ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]]
```

Matrice 10x10

1. Traitement d'images

Création d'une image en niveaux de gris

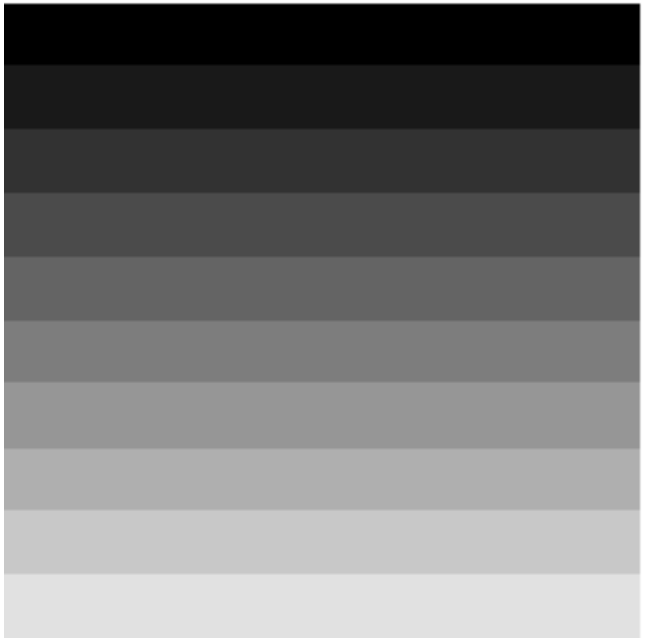
```python
# On importe toutes les fonctions du module miniprojectutils
from miniprojectutils import *

grayscale_img = new_image_grey(10, 10)
nb_rows, nb_columns = dimensions(grayscale_img)
print(grayscale_img)
save_image(grayscale_img, "grayscale_demo_img.png")

for row in range(nb_rows):
    for column in range(nb_columns):
        grayscale_img[row, column] = row * 25
save_image(grayscale_img, "gradient-lines.png")


for row in range(nb_rows):
    for column in range(nb_columns):
        grayscale_img[row, column] = ((row+1)*(column+1))*2.55
save_image(grayscale_img, "gradient.png")
```



1. Traitement d'images

Création d'une image en niveaux de gris

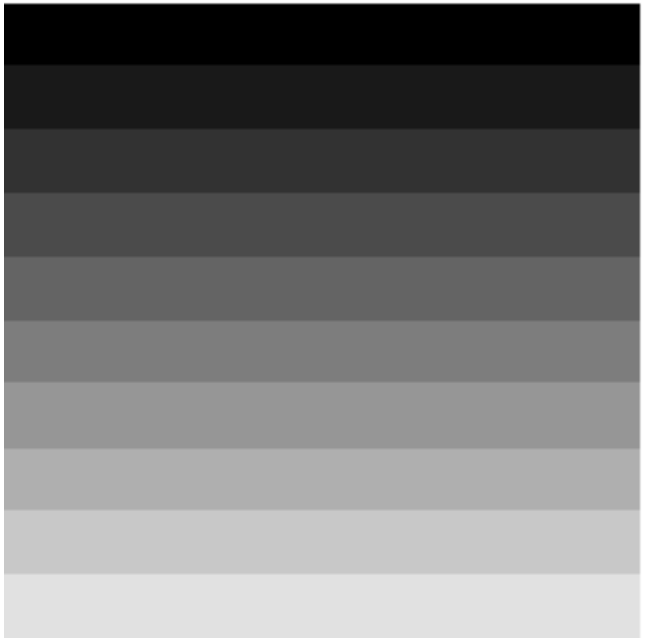
```
# On importe toutes les fonctions du module miniprojectutils
from miniprojectutils import *

grayscale_img = new_image_grey(10, 10)
nb_rows, nb_columns = dimensions(grayscale_img)
print(grayscale_img)
save_image(grayscale_img, "grayscale_demo_img.png")

for row in range(nb_rows):
    for column in range(nb_columns):
        grayscale_img[row, column] = row * 25
save_image(grayscale_img, "gradient-lines.png")


for row in range(nb_rows):
    for column in range(nb_columns):
        grayscale_img[row, column] = ((row+1)*(column+1))*2.55
save_image(grayscale_img, "gradient.png")
```



1. Traitement d'images

Création d'une image en niveaux de gris

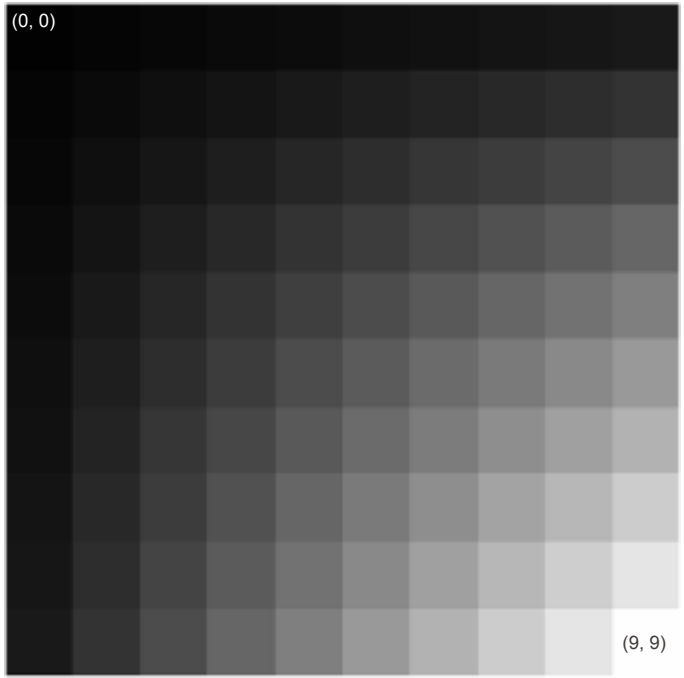
```
# On importe toutes les fonctions du module miniprojectutils
from miniprojectutils import *

grayscale_img = new_image_grey(10, 10)
nb_rows, nb_columns = dimensions(grayscale_img)
print(grayscale_img)
save_image(grayscale_img, "grayscale_demo_img.png")

for row in range(nb_rows):
    grayscale_img[row, :] = row * 25
save_image(grayscale_img, "gradient-lines.png")


for row in range(nb_rows):
    for column in range(nb_columns):
        grayscale_img[row, column] = ((row+1)*(column+1))*2.55
save_image(grayscale_img, "gradient.png")
```



1. Traitement d'images

Création d'une image en niveaux de gris

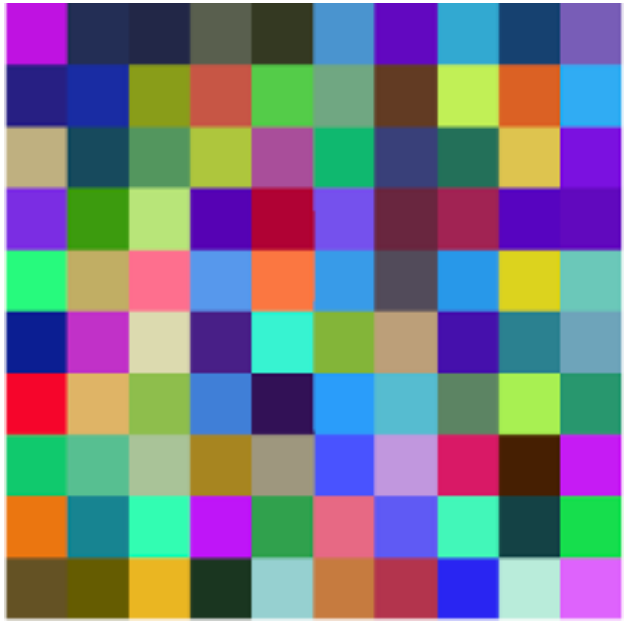
```
# On importe toutes les fonctions du module miniprojectutils
from miniprojectutils import *

grayscale_img = new_image_grey(10, 10)
nb_rows, nb_columns = dimensions(grayscale_img)
print(grayscale_img)
save_image(grayscale_img, "grayscale_demo_img.png")

for row in range(nb_rows):
    grayscale_img[row, :] = row * 25
save_image(grayscale_img, "gradient-lines.png")


for row in range(nb_rows):
    for column in range(nb_columns):
        grayscale_img[row, column] = ((row+1)*(column+1))*2.55
save_image(grayscale_img, "gradient.png")
```



1. Traitement d'images

Création d'une image en RGB

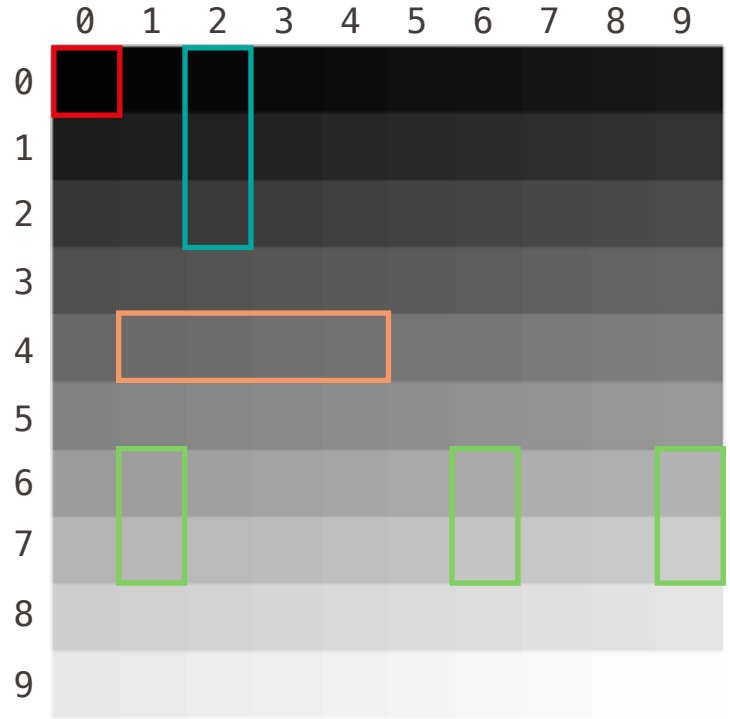
```
from miniprojectutils import *
from random import randint

image = new_image_rgb(10, 10)
for i in range(10):
    for j in range(10):
        # On utilise une "liste en compréhension"
        # On verra cela plus tard
        r, g, b = [randint(0, 255) for _ in range(3)]
        image[i, j] = [r, g, b]
save_image(image, "colors.png")
```



1. Traitement d'images

Slide récapitulative: *Subscripting* et couleurs

- Dans un numpy array, on peut récupérer :
  - Un élément : `img[0, 0]`
  - Une portion de ligne : `img[:3, 2]`
  - Une portion de colonne : `img[4, 1:5]`
  - Plusieurs portions de lignes ou de colonnes : `img[6:8, [1, 6, 9]]`
- Dans le projet, le type de `img[x, y]` sera :
  - Soit un entier (grayscale)
  - Soit une liste de trois entiers (RGB)



Au sommaire cette semaine

1. Traitement d’images
2. Informations sur le mini-projet

2. Informations sur le mini-projet

Organisation

- Compte pour 15% de la note finale (midterm: 50%, examen final: 35%)
  - Pas de question ouverte de programmation à l'examen final
  - Partie QCM en programmation sur l'ensemble du semestre
- Le mini-projet est à réaliser seul ou en binôme
  - Même note pour les membres d'un binôme
  - Annoncer les groupes sur Moodle (un fichier sera créé pour ça)
- Deux séances d'exercices consacrées à ça, et prévoir un peu de temps en plus pour finaliser ce mini-projet (profitez des séances de soutien !)
- Date limite de rendu du projet : dimanche 10 décembre 2023 23h59

2. Informations sur le mini-projet

Éléments de dissuasion...

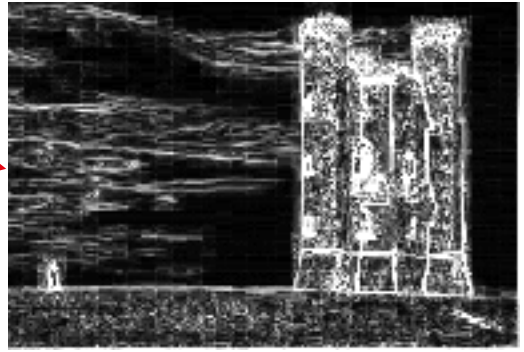
File 1	File 2	Lines Matched
ANONYMIZED		180
		74
		71
		58
		82
		79
		62
		58
		70
		44
		63
		40
		60
		56

2. Informations sur le mini-projet

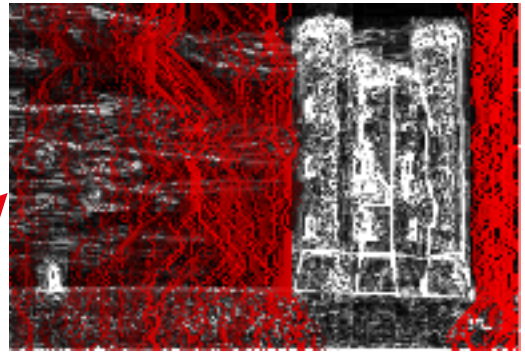
Principe du *seam carving*

Selon une idée et projet original de Jamila Sam et Barbara Jobstmann

Pour chaque pixel, calcul de l'information portée par rapport à ses voisins



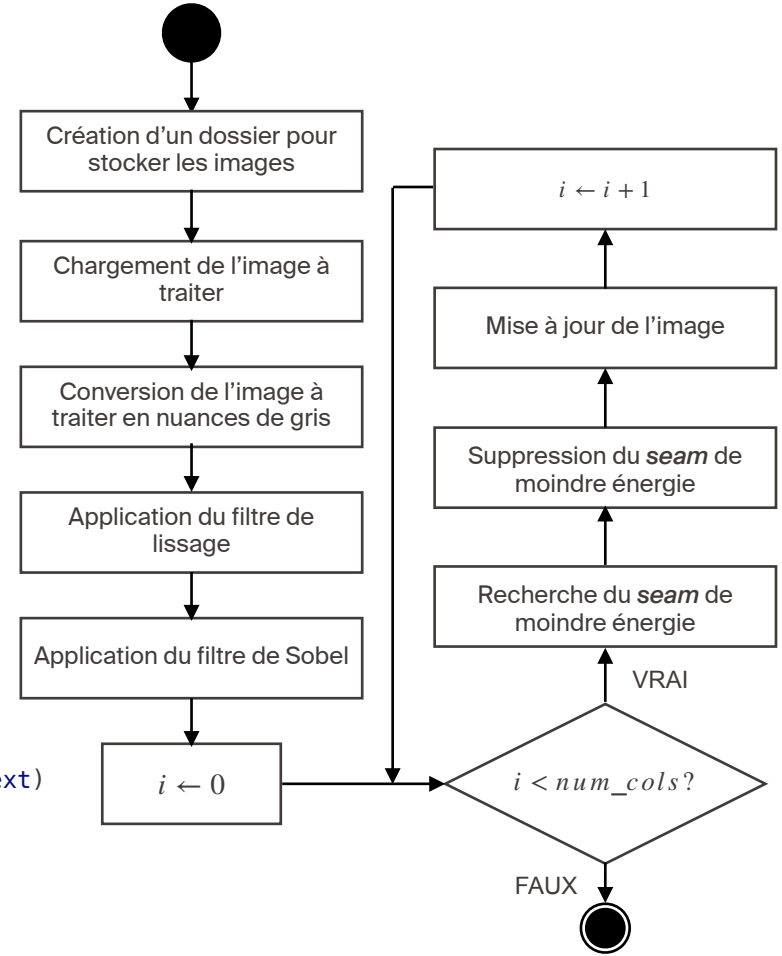
Recherche de «lignes verticales » de basse information



Suppressions successives de ces lignes dans l'image originale

2. Informations sur le mini-projet

```python
def seam_carving(img_path: str, num_cols: int) -> None:
    name, ext = split_name_ext(img_path)
    folder = name + os.path.sep
    os.makedirs(folder, exist_ok=True)

    img = load_image(img_path)

    img_grey = to_grayscale(img)
    save_image(img_grey, folder + "grey" + ext)

    img_grey = smoothen(img_grey)
    save_image(img_grey, folder + "smooth" + ext)

    img_grey = sobel(img_grey)
    save_image(img_grey, folder + "sobel" + ext)
    img_grey = np.uint8(img_grey)

    for i in range(num_cols):
        seam = find_seam(img_grey)

        img_highlight = highlight_seam(img, seam)
        save_image(img_highlight, folder + f"highlight_{i}" + ext)
        img_grey_highlight = highlight_seam(img_grey, seam)
        save_image(img_grey_highlight, folder + f"highlight_{i}_grey" + ext)

        img = remove_seam(img, seam)
        save_image(img, folder + f"step_{i}" + ext)
        img_grey = remove_seam(img_grey, seam)
```



2. Informations sur le mini-projet

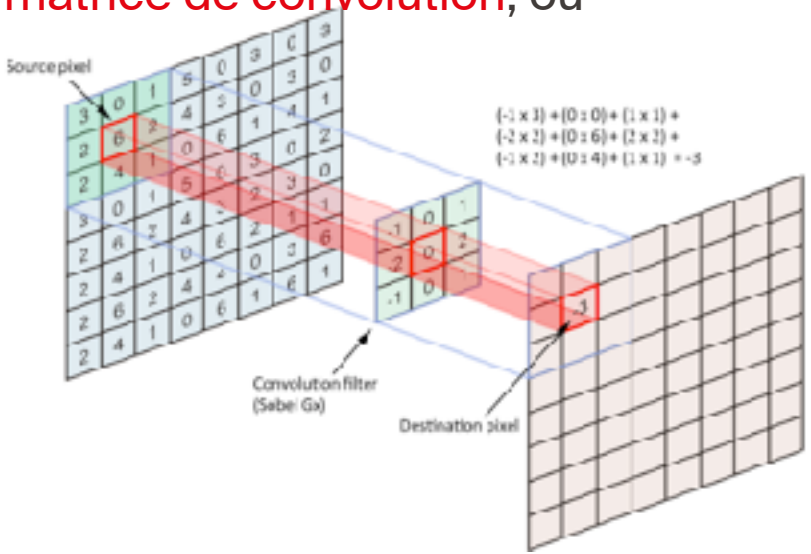
Votre travail

```python
def rgb_to_grey(r: int, g: int, b: int) -> int:
    """Convert an RGB color to a greyscale value."""
    return ...  # TODO

def to_grayscale(img: Image) -> Image:
    """Convert the given image to grayscale."""
    print("   Converting to grayscale...")
    return ...  # TODO

def clamp_index(index: int, length: int) -> int:
    """Return the index, clamped to the range [0, length-1]."""
    return ...  # TODO

def apply_kernel(img_grey: Image, kernel: Kernel) -> Image:
    """Apply a kernel to an image."""
    return ...  # TODO

def find_seam(img_grey: Image) -> Seam:
    """Find the seam with the lowest energy."""
    print("   Finding seam...")
    return ... # TODO
```

2. Informations sur le mini-projet

Convolutions et `apply_kernel()`

- La convolution d'une image consiste à créer une nouvelle image où chaque pixel est le résultat d'une somme pondérée des pixels voisins
- Le calcul est réalisé en définissant une matrice de convolution, ou kernel
- Dans le mini projet :
  - Lissage
  - Filtre de Sobel



https://entiretydotai.github.io/blogs/deep%20neural%20network/convolutions/computer-vision/2020/07/06/Convolutions.html

2. Informations sur le mini-projet

Matrices de convolution pour le lissage et filtre de Sobel

$$\begin{bmatrix} \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{2}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \end{bmatrix}$$

Matrice pour le lissage

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Filtre de Sobel en X

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

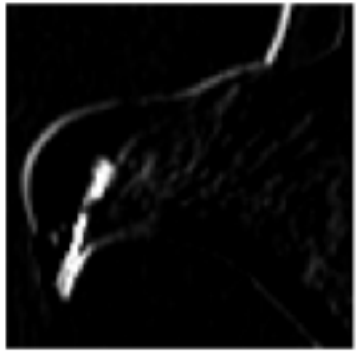
Filtre de Sobel en Y

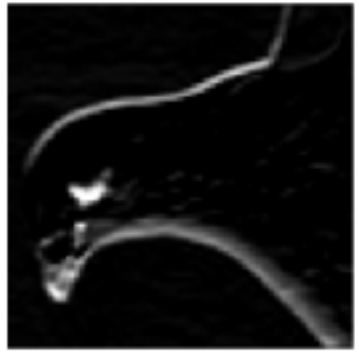
Calcul de la variation en X et en Y des valeurs d'intensité des pixels voisins.

Image originale

Moyenne des pixels voisins, avec un poids double pour le pixel central

Une valeur proche de 0 indique que les pixels au voisinage sont semblables. Une valeur éloignée de 0 indique que les pixels au voisinage sont différents.

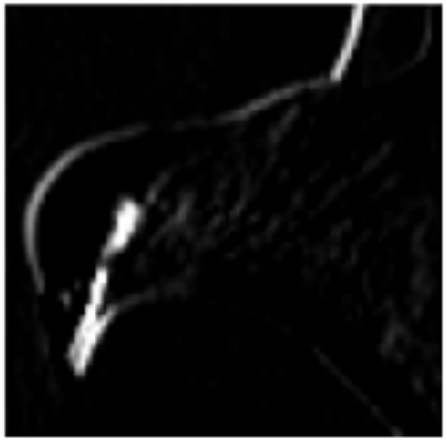




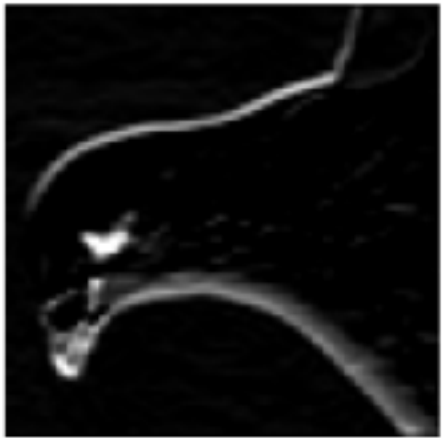
2. Informations sur le mini-projet

Filtre de Sobel et détection de contours – suite

- Pour chaque pixel, on va appliquer le filtre de Sobel en X et en Y, puis calculer la «norme» de ces deux valeurs



x, Filtre de Sobel en X



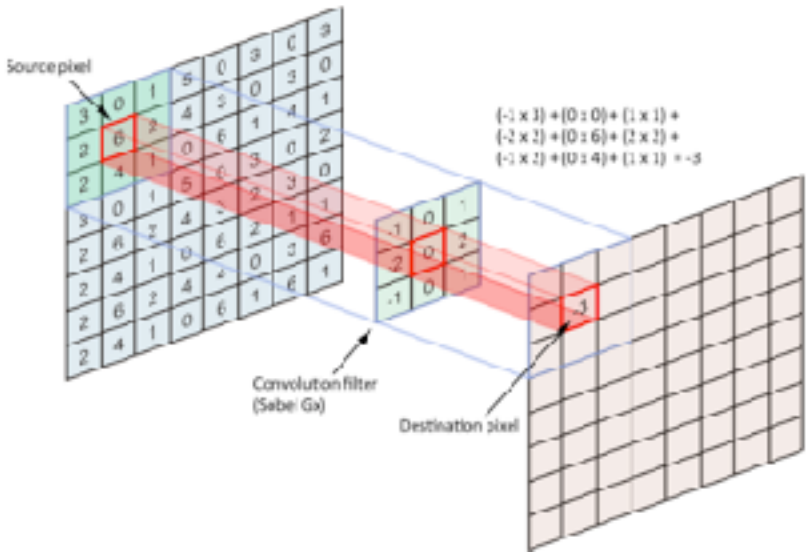
y, Filtre de Sobel en Y

$\sqrt{x^2 + y^2}$

2. Informations sur le mini-projet

Que faire sur les bords ?

- Que faire sur les coins ? sur les bords ?



https://entiretydotai.github.io/blogs/deep%20neural%20network/convolutions/computer-vision/2020/07/06/Convolutions.html

Si $x < 0$, on prend la valeur pour $x = 0$.
Si $y < 0$, on prend la valeur pour $y = 0$.

	3	3	0	1
3	3	0	1
2	2	6	2
2	2	4	1

Il faut aussi faire attention aux valeurs «trop grandes» de x et y.

2. Informations sur le mini-projet

numpy et les opérations optimisées

```python
def smoothen(img_grey: Image) -> Image:
    """Smooth the image using a 3x3 kernel."""
    print("   Smoothening image...")
    kernel_smooth = np.array([
        [1, 1, 1],
        [1, 2, 1],
        [1, 1, 1],
    ]) / 10

    return apply_kernel(img_grey, kernel_smooth)
```

```python
def sobel(img_grey: Image) -> Image:
    """Apply the Sobel filter to the image."""
    print("   Sobel...")
    kernel_sobel_x = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1],
    ])
    sobel_x = apply_kernel(img_grey, kernel_sobel_x)
    kernel_sobel_y = np.array([
        [-1, -2, -1],
        [ 0,  0,  0],
        [ 1,  2,  1],
    ])
    sobel_y = apply_kernel(img_grey, kernel_sobel_y)
    result = np.sqrt(sobel_x * sobel_x + sobel_y * sobel_y)
    return result
```

Bilan de la séance

- Traitement et manipulation d'images :
  - Passer par les pixels
  - Faire attention aux systèmes de représentation des couleurs
- Introduction du miniprojet et de sa structure globale
- Travail à faire :
  - Quelques fonctions de pre-processing
  - Fonction d'application d'un kernel