ICC/Programmation — Semaine 9, Miniprojet 1

GC/MX, 17 novembre 2023

Le miniprojet est à faire par groupes de deux ou individuellement. Les groupes peuvent s'échanger des idées ou des approches générales, mais pas du code directement.

Préparation

Ouvrez Visual Studio Code et depuis le menu Terminal, choisissez New Terminal. Tapez-y cette ligne pour installer si nécessaire les modules de traitement d'image:

```
python3 -m pip install numpy Pillow types-Pillow
```

Si cette étape échoue, appelez un·e assistant·e.

Téléchargez depuis Moodle le fichier **`miniproject-start.zip`** et décompressez-le. Déplacez ensuite à la racine de votre workspace les deux fichiers **`miniprojectutils.py`** et **`miniproject.py`**, ainsi que le dossier **`imgs`**, qui contient quelques images exemples.

- **`miniprojectutils.py`** contient les fonctions que vous pouvez appeler sans avoir besoin de forcément tout comprendre. Vous pouvez consulter leur implémentation et voir comment elles utilisent les bibliothèques **`numpy`** et **`Pillow`** pour faire leur travail. Merci de ne **pas** modifier ce fichier.
- **`miniproject.py`** est le fichier que vous devrez petit à petit compléter. Il contient déjà le code de base décrit au cours, avec un certain nombre de commentaires pour expliquer son fonctionnement.

Lancez le fichier **`miniproject.py`**. Il devrait afficher deux lignes sur la console en disant qu'il commence le travail et qu'il charge une image, puis s'arrêter juste après. Si cela ne fonctionne pas, appelez un·e assistant·e.

Partie 1. Conversion en niveaux de gris

(a) Commencez par implémenter la fonction **`rgb_to_grey()`**, dont le but est de convertir une couleur RGB en un niveau de gris. Cette fonction reçoit donc trois arguments, **`r`**, **`g`**, et **`b`**, tous entre 0 et 255 (compris), et doit retourner un **`int`** entre 0 et 255 aussi.

On pourrait se dire qu'une bonne méthode de conversion en niveau de gris serait de faire la moyenne entre les trois composantes RGB. Mais cela fonctionne mal: notre œil perçoit les contributions des composantes à la clarté d'une couleur de manière différente. Basez-vous plutôt sur cette formule pour faire une somme pondérée, dont vous voyez qu'elle donne une importance prépondérante à la composante verte:

$$grey = 0.2126 \cdot r + 0.7152 \cdot g + 0.0722 \cdot b$$

Rajoutez en bas du fichier (mais *dans* la condition **`if __name__ == "__main__"`** et *avant* l'appel à **`seam_carving()`**) quelques lignes pour vérifier que votre fonction retourne toujours une valeur entière bien comprise entre 0 et 255.

(b) Implémentez la fonction **`to_grayscale()`**. Elle reçoit en paramètre une image couleur et doit retourner une nouvelle image en niveaux de gris de la même taille, obtenue en convertissant chaque pixel avec la fonction **`rgb_to_grey`**.

Pour tester votre code, décommentez dans la fonction **`seam_carving()`** le code lié à l'étape 1. Si tout se passe bien, un fichier image est généré et ajouté à votre workspace à l'emplacement suivant: **`imgs/americascup/grey.jpg`**. Ouvrez-le et vérifiez que l'image obtenue est celle en haut de la page suivante.

Si vous avez des difficultés, essayez avec une petite image de 10×10 pixels que vous générez vous-même avec du code comme montré dans le cours plutôt que directement avec la grande image. C'est plus simple de voir les valeurs intermédiaires (en faisant **`print()`** ou via le *debugger*) sur de petites images.

Partie 2. Convolution

(a) Réglons d'abord le problème des «pixels de bord» comme décrit au cours. Nous allons pour cela utiliser et implémenter la fonction **`clamp_index()`**. L'idée de cette fonction est qu'elle «valide» un indice sur une certaine plage entre 0 et un maximum défini (non compris). Par exemple, si on lui demande de valider un indice **`i`** entre 0 et 10, elle doit retourner **`i`** s'il vaut 0, 1, 2, 3, 4, 5, 6, 7, 8 ou 9, car ce sont toutes des valeurs valides, mais doit retourner 0 si **`i`** < 0 et 9 si **`i`** ≥ 10.

Testez votre fonction en l'appelant avec quelques valeurs.

Cela s'utilisera ensuite ainsi: on validera un indice de colonne (disons **`col`**) avec
`col = clamp_index(col, width)`,
si **`width`** est la largeur de l'image. De manière similaire, on validera un indice de ligne **`row`** avec
`row = clamp_index(row, height)`, si **`height`** est la hauteur de l'image.

(b) Implémentez la fonction **`apply_kernel()`** pour qu'elle crée une nouvelle image en niveaux de gris qui est le résultat de l'application du kernel passé en second paramètre à l'image en niveaux de gris passée en premier paramètre, selon les slides du cours. Utilisez la fonction **`clamp_index()`** pour valider les index.

Testez votre code en décommentant les étapes 2 et 3 dans la fonction **`seam_carving()`**. Ceci devrait vous générer une image lissée dans **`imgs/americascup/smooth.jpg`** et l'image représentant le résultat du filtre de Sobel dans **`imgs/americascup/sobel.jpg`**, qui sont montrées ci-dessous. L'image lissée est peu différente de l'image grise précédente; comparez celles que vous avez générées pour vous assurer que la deuxième est bel et bien légèrement plus lisse (ou plus floue) que la première.

Vous pouvez modifier l'appel terminal à **`seam_carving`** pour tester votre code avec les autres images exemples: **`cats.jpg`**, **`doves.jpg`** et **`tower.jpg`**.