Compte rendu #4

Interface graphique & organisation

CANHOTO Mickaël, BELDJILALI Maxime
Master 1 Imagine, groupe #2.2

I - Introduction

Dans le cadre du projet de création d'un programme d'image mosaïque, nous avons entrepris le développement d'une interface utilisateur en utilisant la bibliothèque TKinter de Python. En parallèle, nous avons restructuré le code existant en adoptant une approche orientée objet, notamment en regroupant les fonctionnalités liées à la création de mosaïques d'images au sein d'une classe dédiée, nommée "Mosaic". Cette réorganisation vise à améliorer la lisibilité, la maintenabilité et la modularité du code, tout en facilitant l'ajout de nouvelles fonctionnalités à l'avenir. Dans ce compte rendu, nous détaillerons les étapes de développement effectuées, ainsi que les choix techniques. Finalement en conclusion nous parlerons des futurs implémentations que nous appliquerons

II - Interface graphique

Figure 1 : Interface graphique

Nous nous donnons ici la possibilité de prendre une image en entrée, une image en sortie et le dossier menant à notre dataset. Cela reste rudimentaire mais nous améliorerons au fil des ajouts à notre implémentation, notamment avec la possibilité de changer la résolution en blocs de l'image en sortie, ainsi qu'avec les dimensions des tuiles.

Une fois que l'algorithme a fini la création de l'image, celle-ci est affiché :

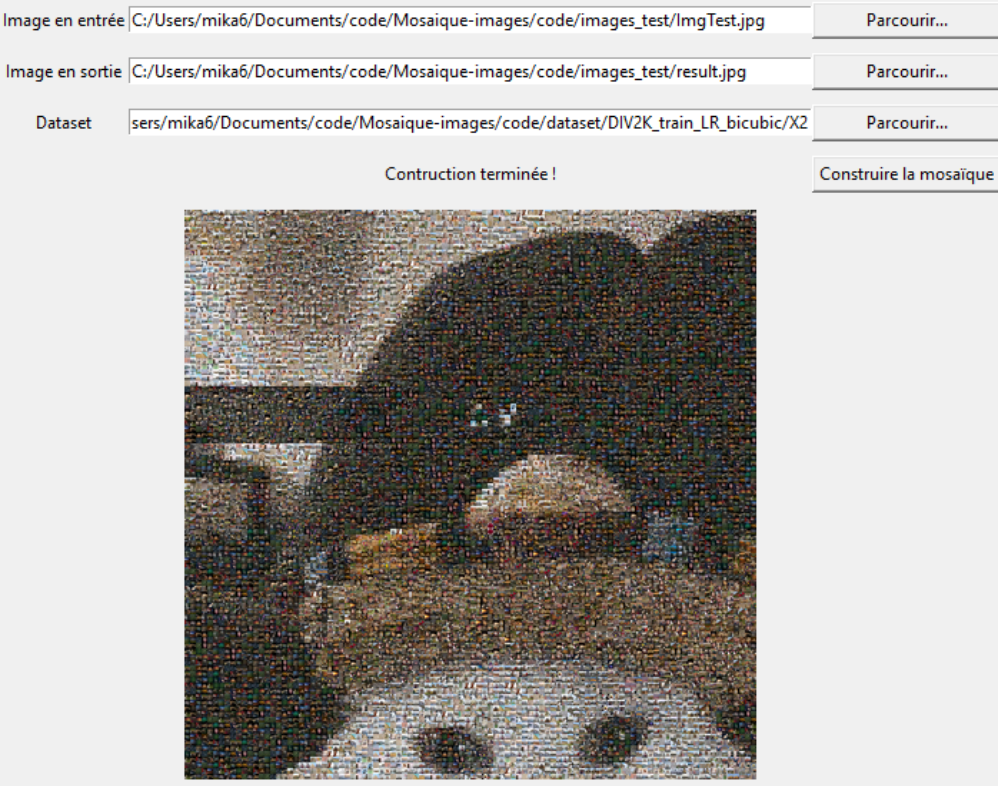

Figure 3 : Interface final

III - Organisation

Pour la partie réorganisation, nous avons créé une classe Mosaic. Cela nous a permis de clairement séparer les étapes de notre algorithme de photo mosaïque et de les faire correspondre plus précisément à celles détaillées précédemment.

Nous avons la classe init qui va permettre l'initialisation de la classe et des ses variables.

```
def __init__(self, image_in_location: str, image_out_location: str, \
        dataset_location: str, target_res=(100, 100), mosaic_size=(32, 32)):
        """Initialize all parameters for mosaic building.
        image_in_location -> Path to the image in input.
        image_out_location -> Path to the mosaic in output.
        dataset_location -> Path to the dataset.
        target_res -> Number of blocks of the mosaic.
        mosaic_size -> Size of each tile.
        """
```

Ensuite nous avons 2 méthodes principales, process_dataset() qui traite la base de données et crée le KD Tree pour l'assignation des imagettes puis build_mosaic() qui va construire l'image mosaic final.

```
  def process_dataset(self):
        """Compute criteria for every image from the dataset
        Store thoses values in a KDTree
        """

        ## Load dataset
        self.load_dataset()

(...)

 ## Create a KDTree for the image values
 self.tree = spatial.KDTree(image_values)


    def build_mosaic(self):
        """Build mosaic"""
 (...)

        ## Write image in output
        self.image_out.save(self.image_out_location)
```

Une fois ces deux fonctions appelées, nous obtenons le résultat final.

IV - Conclusion

En résumé, la mise en œuvre d'une interface graphique avec TKinter et la réorganisation du code en utilisant une approche orientée objet ont permis d'améliorer significativement le projet de création d'un programme d'image mosaic. Cette démarche a favorisé une meilleure structuration du code, rendant le programme plus modulaire et extensible. Cela nous permettra à terme l'ajout de fonctionnalités tel que des structures d'accélération et la création de vidéos mosaïques. Dans les semaines à venir, notre objectif principal serait d'obtenir un programme efficace et robuste afin d'explorer ensuite d'autres fonctionnalités.