Redimensionnement d'image par Seam Carving

Implémentation par DAG et plus court chemin

Introduction

Le redimensionnement d'images par seam carving repose sur une idée simple : **réduire une image en supprimant progressivement des chemins de pixels de faible importance visuelle, plutôt qu'en appliquant une transformation géométrique uniforme.**
Dans ce projet, cette idée est traduite de manière rigoureuse à l'aide des outils de la théorie des graphes vus en cours, en particulier la modélisation par graphe orienté acyclique (DAG) et le calcul de plus courts chemins par programmation dynamique en ordre topologique.

L'ensemble de la démarche est implémenté en Python dans un fichier unique, de façon volontairement simple et lisible, afin de rester au plus proche des algorithmes étudiés en cours.

Chargement et représentation de l'image

L'image d'entrée est chargée sous forme de tableau NumPy de dimension $(H, W, 3)$, correspondant aux trois canaux RGB. Cette étape est implémentée dans la fonction charger_image_rgb, qui encapsule simplement la lecture du fichier et la conversion dans un format manipulable par la suite.

De la même manière, la fonction « sauvegarder_image_rgb » permet d'enregistrer les images intermédiaires et finales. Ces fonctions ne contiennent aucune logique algorithmique complexe : elles servent uniquement d'interface propre entre les données images et les traitements algorithmiques, ce qui rend le reste du code plus lisible.

Calcul de la carte d'énergie

La carte d'énergie constitue le point de départ de l'algorithme de seam carving. Elle permet d'associer à chaque pixel un coût représentant son importance visuelle.
Dans le code, cette étape est réalisée par la fonction « calculer_energie ».

Concrètement, l'image est d'abord convertie en niveaux de gris via la fonction « _rgb_vers_gris ». Cette conversion est volontairement simple et repose sur une combinaison linéaire standard des canaux RGB, suffisante dans le cadre du projet.

Le gradient de l'image est ensuite calculé à l'aide d'un opérateur de Sobel implémenté manuellement dans la fonction « _sobel ». Cette implémentation "maison" permet de garder un contrôle total sur le calcul et d'éviter toute abstraction excessive. L'énergie finale d'un pixel est définie comme la somme des valeurs absolues des gradients horizontal et vertical.

Comme vu en cours, cette carte d'énergie joue exactement le rôle d'une matrice de coûts, qui sera utilisée dans le calcul du plus court chemin.

Modélisation en graphe orienté acyclique

Une fois la carte d'énergie calculée, le problème central consiste à déterminer une couture verticale de coût minimal.
Dans le code, cette étape est directement liée à la fonction « trouver_couture_minimale ».

La modélisation suit exactement le cadre théorique vu en cours :

- chaque pixel est considéré comme un sommet,
- les transitions possibles entre pixels définissent des arêtes orientées,
- l'orientation est strictement descendante (de la ligne $r$vers la ligne $r + 1$).

Cette orientation garantit que le graphe est un DAG, ce qui est une hypothèse fondamentale pour l'algorithme utilisé.
Dans l'implémentation, il n'est pas nécessaire de construire explicitement le graphe : la structure du DAG est implicite dans les indices du tableau et dans les transitions autorisées.

Plus court chemin et programmation dynamique

Le calcul de la couture minimale est réalisé par programmation dynamique, ce qui correspond exactement à l'algorithme du **plus court chemin sur DAG** vu en cours.

Dans la fonction « trouver_couture_minimale », le tableau dp joue le rôle du tableau des distances minimales, tandis que le tableau parent permet de mémoriser les prédécesseurs afin de reconstruire le chemin optimal.

L'ordre topologique est ici naturel : il correspond à l'ordre des lignes de l'image. Le remplissage du tableau dp se fait donc ligne par ligne, ce qui revient à appliquer une relaxation dans l'ordre topologique, comme présenté dans le cours sur les DAGs.

Une fois la dernière ligne atteinte, le pixel de coût minimal est sélectionné, puis la couture est reconstruite en remontant les prédécesseurs. Cette reconstruction correspond exactement à la méthode utilisée dans les algorithmes de plus court chemin classiques.

Suppression de la couture minimale

Après avoir identifié la couture d'énergie minimale, celle-ci est supprimée de l'image. Cette étape est implémentée dans la fonction « supprimer_couture ».

Pour chaque ligne, le pixel correspondant à la couture est retiré, et les pixels restants sont conservés dans un nouveau tableau de largeur réduite. Cette opération est volontairement simple et locale, afin de rester fidèle à la définition théorique d'une couture comme un chemin valide dans le DAG.

La suppression modifie nécessairement la structure de l'image, ce qui justifie le recalcul de la carte d'énergie à l'itération suivante.

Boucle itérative et réduction progressive

La fonction « seam_carving_reduire_largeur » implémente la boucle principale de l'algorithme. Elle applique successivement les étapes suivantes :

- calcul de la carte d'énergie,

- calcul de la couture minimale,
- suppression de la couture.

Cette boucle est exécutée $k$fois, où $k$est fixé par l'utilisateur (par exemple la moitié de la largeur initiale).
Comme vu en cours dans les algorithmes itératifs sur graphes, chaque suppression modifie le problème initial, ce qui rend indispensable le recalcul de l'énergie à chaque itération.

Les images intermédiaires peuvent être sauvegardées afin de visualiser l'évolution progressive de la réduction et de mieux comprendre l'impact de la suppression successive des coutures. Cette possibilité permet également de valider expérimentalement le bon fonctionnement de l'algorithme. Dans le cadre du rendu final, seule l'image résultante après l'ensemble des itérations est conservée, afin de simplifier la présentation des résultats.

Le résultat obtenu est cohérent avec le principe du seam carving : les zones de faible énergie, principalement situées dans le fond de l'image, sont progressivement supprimées, tandis que les structures importantes, comme le visage, sont préservées.

Liens entre le code et le cours

Chaque partie du code correspond directement à une notion du cours :

- la carte d'énergie correspond à la définition des coûts,
- la fonction « trouver_couture_minimale » implémente le plus court chemin sur un DAG,
- l'ordre des lignes joue le rôle d'un ordre topologique,
- le tableau parent permet la reconstruction du chemin optimal,
- la boucle principale correspond à une application itérative d'un algorithme de graphe sur une structure évolutive.

Ce parallélisme volontaire entre le code et le cadre théorique facilite la compréhension et garantit la correction de l'approche.

Conclusion

L'implémentation proposée illustre de manière concrète comment des concepts fondamentaux de la théorie des graphes, tels que les graphes orientés acycliques, l'ordre topologique et le calcul de plus courts chemins, vus en cours, peuvent être mobilisés pour résoudre un problème réel de traitement d'images. Loin de rester purement théoriques, ces notions trouvent ici une application directe et intuitive, montrant leur pertinence dans un contexte pratique.

Le code a été volontairement conçu de façon simple, structurée et progressive, afin de rester au plus proche des algorithmes étudiés en cours. Chaque étape de l'algorithme correspond clairement à une idée théorique précise, ce qui facilite à la fois la compréhension du fonctionnement global et l'analyse de sa correction. Les commentaires et la décomposition en fonctions permettent également de mettre en évidence le lien étroit entre la modélisation en graphe et son implémentation effective.

Le seam carving apparaît ainsi comme un exemple particulièrement parlant de l'interaction entre modélisation théorique et implémentation pratique. Il montre comment une bonne formulation algorithmique, basée sur des hypothèses structurelles simples (en l'occurrence

l’absence de cycles et l’existence d’un ordre topologique), permet d’obtenir une solution à la fois élégante, efficace et visuellement pertinente. Ce projet met en évidence l’intérêt d’aborder des problèmes concrets avec des outils issus de la théorie des graphes, et souligne la valeur ajoutée d’un raisonnement algorithmique rigoureux dans des domaines appliqués comme le traitement d’images.