Projet 1 - Compte Rendu

Seam-Carving

Par Jérémie Dumas et Julien Herrmann

Sommaire

I. Manuel d'utilisation

On utilise le programme en le lançant en ligne de commande, avec les paramètres nécessaires. On peut utiliser des images en couleur (PPM v6) ou en noir et blanc (PPM v5). On peut vouloir une interface graphique (PPM v6 uniquement), visualiser la fonction d'énergie dans le fichier de sortie, etc. La liste des options disponible est la suivante :

prog.out.opt fichier opt1 opt2 …

fichier : Le nom du fichier source à utiliser. Donner le chemin absolu du fichier.
-o str : optionnel. Permet de spécifier le nom du fichier de sortie. Le fichier source est utilisé comme fichier de sortie sinon.
-m str : optionnel. Permet de spécifier la fonction d'énergie à utiliser. L'opérateur de Sobel est utilisé par défaut. Les fonctions disponibles sont "naive", "sobel" et "scharr".
-t : Permet d'afficher le temps d'exécution du redimensionnement. Ne fonctionne pas avec l'interface graphique.
-e : Permet d'enregistrer dans le fichier de sortie la fonction d'énergie plutôt que la couleur de l'image. Pratique pour avoir une idée des différences existantes entre les fonctions d'énergie.
-nx int : La nouvelle largeur de l'image. Par défaut inchangée. N'est pas nécessaire si on utilise l'interface graphique.
-ny int : La nouvelle hauteur de l'image. Par défaut inchangée. N'est pas nécessaire si on utilise l'interface graphique.
-g : Active l'interface graphique interactive.

À propos de l'interface graphique :

Affiche l'image à redimensionner dans une fenêtre graphique. Lorsque l'utilisateur charge une image, une fenêtre graphique s'ouvre et affiche l'image initiale. L'utilisateur peut cliquer sur le bord droit de l'image pour réduire sa largeur, le bord haut pour réduire sa hauteur, ou l'angle en haut à droite pour redimensionner l'image selon les deux coordonnées.
Quand l'utilisateur clique sur une des parties actives de l'image et déplace la souris, une phrase au bas de l'écran indique quelles seraient les dimensions de l'image si il relâchait le bouton gauche de la souris à cette endroit. Un cadre représente également la futur taille de l'image.

Lorsque l'utilisateur a choisi la future taille de l'image, la fonction principale la modifie puis dessine la nouvelle image ainsi obtenue, en attendant un nouveau choix de l'utilisateur.
À tout moment, l'utilisateur peut appuyer sur le bouton Q de son clavier pour enregistrer l'image redimensionnée.

II. Historique du développement

Le seam-carving, c'est une grand histoire d'amour. Outre le fait que c'était le seul projet qui avait l'air un peu intéressant dans le lot (du post-script, damned), et qu'on manque décidément toujours un peu d'imagination pour avoir un projet original, on se rend vite compte qu'il y a de quoi faire avec le seam-carving.

Au début on part donc sur une représentation de l'image par une matrice des pixels, "ça a un p'tit côté pratique" nous dira-t-on. On se met à peu près d'accord : y en a un qui s'occupe de charger les fichiers ppm et de faire l'interface graphique, l'autre qui programmera les algo' de redimensionnement. Les premières semaines on s'excite donc comme on peut sur nos images matricielles. Le calcul de l'énergie : impeccable. La prog' dyn' pour le chemin de plus faible énergie : no problem. Là où les choses se compliquent, c'est lorsqu'on veut optimiser un peu tout ça. La suppression d'un chemin dans notre matrice ne se fait pas trop en temps linéaire, et ça c'est mal. Partant de là, quand il faut mettre à jour l'énergie des voisins du chemin supprimé, considérant tous les cas particuliers et autres âneries dans l'genre, ça devient vraiment prise de tête.

On en discute un peu avec le prof (quand il est là), puis on repars à la fin du cours en se disant que "finalement ça doit pas être si compliqué que ça" ces fonctions de mise à jour. Mais rentré chez soi, on trouve que non, on avait pas pensé à tout, y a des cas où les chemins se croisent etc. qui font que ça semble pas trop trop optimisé comme truc.

Bref je reprends le problème à zéro, et décide de complètement changer de structure de donnée. Je passe donc le dimanche avant la dernière séance à tout recoder en utilisant la nouvelle structure de donnée (des listes quadruplement chaînées) décrite plus loin. C'est pas si compliqué, vu qu'il s'agit en substance de réécrire les fonctions déjà implémentées sur cette nouvelle structure. Du coup, lundi j'arrive avec un programme de seam-carving qui marche, et qui bizarrement sans les optmisations est déjà plus rapide que l'algo' fonctionnant sur les matrices. Bref la fonction de mise à jour de l'énergie, assez simple, a aussi été implémentée facilement. Pour la fonction de mise à jour du chemin de plus petite énergie, ça a été une autre paire de manche, j'y ai passé plusieurs heures. On expliquera plus loin comment elle fonctionne. En tout cas il y a plein de rebondissement, toujours de jolis bugs qui lorsqu'on les corrige cèdent la place à de nouveaux. Mais globalement on s'en sort plutôt bien, mieux même oserai-je dire qu'avec nos vieilles matrices des familles.

Et là grande surprise, mardi soir après le TD de prog', je me mets en tête d'ajouter à notre bagage la fonction d'agrandissement. On expliquera aussi plus loin comment elle marche. Mais grosso-modo comme elle utilise la fonction de suppression précédente, elle aura aussi permis d'en révéler quelques bugs substantiels. Finalement on arrive à la faire fonctionner correctement. Avec notre super-structure de donnée ça se fait même très facilement. C'est toujours "assez rapide" on va dire. Peut-être pas encore assez rapide, mais à moins de réessayer la méthode matricielle, je ne vois pas trop comment faire. Finalement il manque peut-être encore le choix des zones à protéger/supprimer à implémenter. Je gage qu'on peut rajouter assez facilement une méthode de protection/suppression assez naïve, mais comme ça risque de ralentir les calculs dans le cas général et qu'il se fait tard, je pense qu'on va s'en passer.

III. Structure de donnée, algorithmes

Listes chaînées :

La structure utilisée est celle d'une liste chaînée, mais en plus compliquée. Chaque pixel de l'image est représentée par une cellule. Chaque cellule va avoir droit à un pointeur vers la cellule de gauche, celle de droite, ainsi que les voisins du dessus et du dessous. Une cellule sur le bord de l'image pointera vers elle-même. Avec un test d'égalité physique (==) on vérifie donc si on est au bord de l'image ou pas. Et c'est assez pratique pour la fonction de calcul de l'énergie : on considère que le voisin de gauche d'une cellule sur le rebord gauche est elle-même. Ça évite un gradient trop déséquilibré.

Les autres champs utilisés sont la donnée sur la couleur de l'image, la valeur qu'on lui attribue (luminosité dans le cas d'une image rvb par exemple), le poids (cf. partie sur l'algo' de prog' dyn'), et un pointeur vers le prédécesseur.

Prog' Dyn' :

L'algorithme utilisé pour la recherche du chemin d'énergie minimum, c'est un algo' de prog' dyn' (ça tombe bien c'est le titre du paragraphe). Comment ça marche ? On parcours les lignes de notre image, en stockant dans le champ poids de nos pixel l'énergie du chemin minimal qui part d'un pixel du haut de l'image vers le pixel courant, en utilisant le fait que le prédécesseur sur ce chemin (enregistré dans le champ "prev") est toujours soit le voisin du dessus, soit celui du dessus à droite ou à gauche. Lorsqu'on a finit de parcourir la dernière ligne, il suffit de faire une recherche de minimum sur celle-ci pour trouver le pixel d'arrivée du chemin d'énergie minimale de notre image.

Mise à jour du chemin minimal :

Il s'agit de parcourir les lignes de notre image, en ne mettant à jour que les poids nécessaires. Concrètement, on remonte le long du chemin supprimé, puis on redescend en effectuant un certain nombre d'opération. Il y a des pixels dont il faut forcément mettre à jour le poids, ce sont ceux dont un parent (haut, haut-gauche ou haut-droite) a été supprimé. Il y a les pixels dont un parent a vu son poids changer également. Pour ne les mettre à jour qu'une seule fois, on stock les cellules à mettre à jour dans une liste. Pour vérifier l'appartenance à la liste en question, on utilise un tableau (buffer), qu'on n'oublie de remettre à zéro après avoir parcouru toute la ligne. La magnifique technique dite du double buffer permet entre deux lignes de récupérer un tableau de false prêt à être utilisé pour déterminer les cellules à mettre à jour sur la ligne suivante. Bref, le code est un peu commenté, et de toutes façons vous pouvez toujours poser des questions.

Agrandissement :

Apparemment un des grands défis du seam-carving pour certains, surtout si on travaille avec des matrices. Avec notre structure de donnée ça va plutôt bien. La méthode utilisée est la suivante : on veut augmenter notre image de n pixels, on regarde donc quels sont les n prochains chemins que l'on est censé supprimer, puis on duplique ces chemins (ce sont ceux qui sont de moindre importance, donc ça tombe bien). Pour éviter de se sentir bête si on double la taille de notre image (i.e. dupliquer du contenu important), on procède en plusieurs étapes pour l'agrandissement : on itère le processus en agrandissement l'image d'au plus 50% de sa largeur courante à chaque fois. De cette façon seul le fond de "moindre importance" est dupliqué. Aussi lorsqu'on duplique un chemin, on pourrait chercher à introduire un pixel dont la couleur est la moyenne des couleurs des voisins. En pratique ça a tendance à créer des parties floues sur l'image, donc on préfère éviter.

IV. Bilan

Finalement, cette structure de donnée semble plus performante que celle utilisant des matrices. Afin de s'en convaincre, j'ai effectué quelques mesures sur l'image donnée en exemple (jointe dans l'archive), où l'on obtient les temps d'exécution suivants :

- Temps nécessaire pour enlever 50 colonnes (en secondes).

Version avec tableaux :
Sans optimisations : 49.899118
Mise à jour de l'énergie optimisée : 11.356710
Version avec listes :
Sans optimisations : 8.592537
Mise à jour de l'énergie optimisée : 3.104194
Mise à jour de l'énergie & de la recherche de chemin optimisés : 2.152134

- Agrandissement x2.

2 passes récursives terminal : 18.041128
1 passe non récursive terminal : 17.885118

PS : Attention en lisant les commentaires du code source, le fichier "Seamv.ml" est une simple copie du fichier "Seamh.ml" en inversant droite/bas et gauche/haut (en faisant attention à la fonction de duplication de chemin également). Ainsi je n'ai pas pris la peine de modifier les commentaires et les noms des variables muettes.