Vincent Delaitre et Anne-Laure Mouly

Rapport DM 1 : Seam Carving

Description de l'implémentation

Notre programme de redimensionnement d'image se sépare en plusieurs modules. Nous avons choisi de multiplier ces modules afin que les modifications soient plus faciles à effectuer.

1. Le module Types contient les types de l'énergie et des chemins à effacer. Le mettre ainsi permet plus de flexibilité dans l'ordre d'éxécution des modules : ocaml sait déjà à quoi s'attendre.

   Nous avons choisi de bien séparer les chemins horizontaux et verticaux pour pouvoir les gérer séparément.

   Ce module contient aussi les valeurs utilisées pour protéger/supprimer des zones afin de pouvoir les changer facilement.
2. Le module Pixel regroupe les fonctions définissant notre implémentation des pixels : un pixel est soit un niveau de gris représenté par un entier, soit un pixel de couleur représenté par des niveaux entiers de rouge, vert et bleu. Il contient aussi les fonctions pour convertir nos pixels afin que ocaml puisse les afficher.
3. Dans le module Matrix nous avons regroupé les fonctions permettant la manipulation des images sous forme d'un réseau : chaque pixel est défini par une étiquette et par ses quatre voisins. Cela nous permet de supprimer ou d'ajouter un chemin en temps linéaire. Il suffit de redéfinir les pointeurs "voisins" des pixels au bord du chemin.
4. Le module Energy contient les fonctions liées à au calcul de l'énergie des pixels sous forme du gradient de l'intensité. Trois versions différentes sont écrites pour calculer cette énergie. La première calcule l'énergie d'un pixel en prenant en compte ses quatres voisins haut, bas, gauche et droit alors que la deuxième prend en plus en compte les voisins dans les diagonales. La troisième sert simplement pour voir ce que donnerait la suppression des chemins en prenant simplement l'intensité comme fonction d'énergie.

   L'énergie dont le type est défini dans Types est un triplet : le gradient d'intensité puis le bonus et le malus qui lui sont associés afin de conserver ou supprimer des zones.
5. Le module Image est le coeur du programme. Il regroupe les différentes opérations faites sur les images en essayant d'être au maximum abstrait afin de faciliter les modifications.

   Le type image contient un booléen nous permettant de savoir si l'image est en couleur ou non, les dimensions de l'image et les données. Il contient ensuite des champs mutable contenant l'image ayant subie des modifications.

   Notre programme permet de lire non seulement les fichiers .ppm et .pgm mais aussi les fichiers .bmp.

   Comme nous venons de l'expliquer dans Energy : on associe de malus et des bonus pour donner plus de pouvoir à l'utilisateur dans la gestion des zones, mais aussi pour ajouter des chemins. En effet, on ne fait que dédoubler un chemin, on doit donc mettre un bonus au chemin qui vient d'être modifié pour éviter de recopier toujours le même. Ce bonus décroit au fur et à mesure qu'on ajoute de nouveaux chemins pour pouvoir éventuellement repasser à cet endroit.
6. Le module Path est constitué des fonctions calculant dynamiquement le chemin de plus faible énergie. Pour se faire, la première ligne est initialisée avec l'énergie des pixels correspondants puis

chaque case de la matrice indique la somme des énergies des pixels constituant le chemin de plus faible énergie qui arrive jusqu'à cette case. L a fonction find_path retouve alors le chemin de plus faible énergie pour l'ensemble de l'image.

7. Le module Interface regroupe les fonctions utiles à l'affichage par ocaml. On y définit les différents outils : les cases, boutons, les labels utilisés...
8. Enfin, le module Main ordonne adéquatement les fonctions données dans les autres modules

Pour plus de détails, des fichiers .mli commentés sont donné pour chaque modules. Il aussi bien sur possible de fouiller le code.

Choix et optimisations

La mise en place de ce programme n'a pas été évidente. Il y a eu beaucoup de modifications, d'optimisations et il reste bien sûr des choses à faire.

Parmi les changements qui ont été faits nous pouvons citer :

1. Le module Image n'était pas aussi abstrait au départ. Nous avons fait cette abstraction en effectuant de plus en plus de modifications.

   En particulier tout le début du DM avait été fait en utilisant une matrice de pixels comme représentation pour nos images. Nous avons changer cette représentation pour celle définie dans le module matrix quand nous nous sommes rendu compte que réécrire l'intégralité de la matrice après avoir supprimer un chemin prenait trop de temps (temps quadratique).

2. A ce propos, nous avons utilisé ocaml pour nous préciser le temps d'éxécution de chaque fonction afin de voir où notre programme prenait le plus de temps de calcul. Nous avons alors vu qu'il serait nécessaire d'optimiser la mise à jour des données après la suppression d'un chemin.

3. Au contraire, nous avons choisi dès le début de pouvoir gérer les images en couleur ainsi que le redimensionnement dans les deux directions.

   Nous avons aussi rapidement voulu faire de l'agrandissement mais cette partie n'a été complètement mise au point qu'en derniére minute. Nous nous sommes alors rendu compte qu'il ne suffisait pas de recopier le chemin de plus basse énergie. Nous avons alors mis en place le système des bonus.

   A partir de là, nous nous sommes rendu compte que permettre à l'utilisateur de protéger/supprimer des zones devenait relativement facile. Nous avons alors mis aux points ces fonctionnalités.

4. Enfin, l'interface graphique a aussi été fait très tardivement et reste à être amélioré

   Il reste á optimiser les temps de calcul ou à faire l'optimisation "plus de real time" suggérée dans le sujet.

Utilisation du programme

Tel qu'il est actuellement le programme s'éxécute en compilant le tout à l'aide du makefile puis en tapant la commande :

./resize_img image.format nvl_image.format

Les deux arguments sont respectivement le fichier d'entrée et le fichier de sortie. Les formats compatibles étant .ppm, .pgm et .bmp pour les fichiers d'entré et .ppm et .pgm pour la sortie

S'ouvre alors (n'oubliez pas d'être patient) une fenêtre comprenant l'image et des fonctionnalités pour l'utilisateur :

- des champs avec les dimensions de l'image que l'on peut régler à sa guise
- un bouton pour choisir le mode suppression ou protection, il suffit alors d'aller gribouiller l'image en rouge ou en vert respectivement
- des boutons pour lancer le redimensionnement (être encore plus patient), sauver l'image actuelle, annuler le dernier redimensionnement (être patient aussi) ou le dernier gribouilli fait et enfin pour quitter la fenêtre

Amusez-vous bien !

Juste un petit mot casé au milieu pour dire que le sujet était vraiment bien choisi, c'est beaucoup plus motivant pour travailler. Merci.

Conclusion

Nous nous y sommes pris relativement à l'avance et en travaillant beaucoup en dernière minute en plus, nous avons réussi à faire ce que nous comptions faire et même plus. N'étaient pas prévu de conserver/supprimer des zones et de plus, l'agrandissement ainsi qu'un bel interface graphique restaient en option.

Il n'a par contre pas toujours été facile de gérer la répartition des tâches surtout lorsque plein de modifications sont en place de chaque côté du binôme. Mais globalement le code a bien avancé tout au long de sa construction. Nous n'avons jamais été vraiment bloqués et les bogues ne nous ont pas pris trop de temps.

Et puis bon... ca marche :)