Compte Rendu TP3 POOCS

Compression d'une Image par Maillage

JOHN GLIKSBERG

M1 IHPS @ UVSQ

Introduction

Le but de ce TD/TP est de fournir un outil qui génère une maille de triangles à partir d'une image pour réduire la quantité d'information stockée d'une image.

Le maillage consistera en un ensemble de triangles dont les points sont des pixels de l'image. Chaque pixel retiendra la couleur de l'image en ce point. On pourra retrouver un affichage de l'image d'origine en dessinant les triangles remplis de la couleur moyenne de celles de ses points.

La "compression" se fait sur la formation de triangles qu'on privilégiera dans les zone avec des grandes differences de couleur.

On n'alignera *pas* les côtés des triangles avec les lignes qu'on peut distinguer dans l'image mais on construira toujours de nouveaux triangles dans la zone qui diffère le plus de l'image (sauf lorsque l'on a atteint une résolution trop fine).

C'est ainsi un processus itératif car on ne prévoit pas où sera le prochain découpage.

On décide d'avance qu'on ne travaillera que sur des images en niveaux de gris. Pour la modélisation de notre outil on décide de se limiter à des fichiers en entrée au format `.pgm` et un enregistrement du maillage au format `.vtk`.

Définitions: Dans un maillage M donné, pour un triangle T dans M donné, si C_1, C_2 et C_3 sont les couleurs des points de T, on définit C la couleur moyenne du triangle, et l'*écart* en T correspond à la différence absolue entre C et la couleur de l'image au barycentre des points du triangle.

Ceci nous permet d'une itération à l'autre de choisir le triangle au plus grand écart dans notre maillage. On travaille maintenant sur le barycentre P de ce triangle, c'est le point à partir duquel on va rendre notre maillage plus précis.

Le processus précis est décrit dans l'énoncé et consiste à vider une cavité de triangles autour de P et reconstruire de nouveaux triangles plus petits.

Cet algorithme de maillage privilegiera souvent trop certaines régions de l'image, mais en limitant la granularité à celle du pixel, on répartit l'importance donnée aux zones de l'image.

Mise en place

On a implémenté les différentes parties de notre outil avec pour chacune une classe décrite entièrement dans un fichier `.hh` et un fichier `.cc` si ce n'est pas une classe `template` (dans quel cas on construit un objet `.o` pour cette classe).

Voici un graphe d'inclusion qui représente ces classes selon l'inclusion de leur *header*.

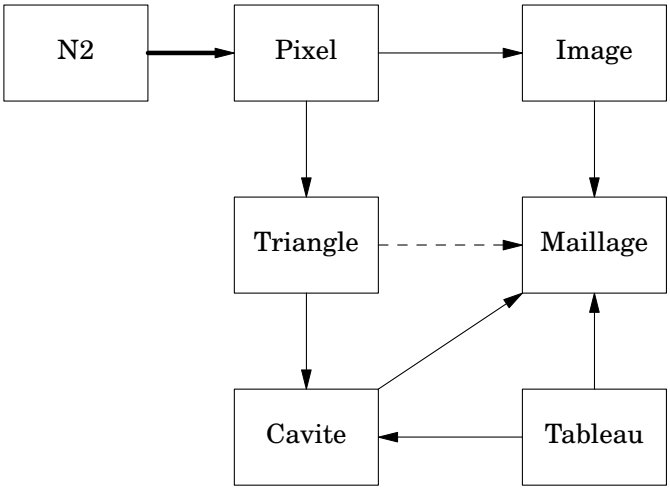

Tableau (`Tableau<T>`)

Cette classe `template` nous a fait un exercice sur les classes génériques: on peut accéder à ou modifier une liste d'élèments générique. On réduit le nombre d'allocations nécessaires au passage à l'échelle en séparant les concepts de *taille* et de *capacité*.

La méthode `retaille` peut faire appel à `reserve` si on demande une taille qui excède la capacité (on choisit de ne pas *réallouer vers le bas*). Comme les objets stockés sont génériques, on ne fait pas de réallocation à la `realloc`, mais on fait un appel à `new` et on boucle avec des appels à `operator=`. De même lorsqu'on supprime un élèment on ne fait que le remplacer par le dernier; on fait encore un appel à `operator=` et on appelle le constructeur générique pour le dernier élèment déplacé. C'est relativement lent par rapport à des fonctions type `memcpy` mais on préfère préserver l'usage *POO*.

N2

On implémente ici une classe très simple reprèsentant des coordonnées entières à deux dimensions, et quelques opérations de base.

Par exemple, `operator==` rendait aisé de tester si un point était présent dans un maillage sans avoir à implémenter `bool *_pixels_presents`. C'est par contre une méthode plus lente qu'avec ce tableau.

Pixel

Cette classe hérite de `N2` et lui ajoute une valeur de couleur *typedef'd* à un `int`. Sachant qu'on travaille avec le format `.pgm`, on aurait pû stocker la couleur dans un `char`, comme ce sera le cas dans une `Image`, mais pour pouvoir manipuler les `Pixel`s avec aise on a fait ce choix.

On a ainsi prévu que pour le calcul d'un barycentre on voudra pouvoir écrire :

```
Pixel p = (p1 + p2 + p3) / 3;
```

Sans se soucier d'*overflow* sur les `Pixel`s intermédiaires.

Triangle

Cette classe représente un rassemblement de trois `Pixel`s et c'est à ce niveau qu'on implémente en particulier la méthode `bool cercleCirconscritContient(N2)`.