Programmation Orientée Objet pour le Calcul Scientifique
Master MIPS - Année 2014-2015

TD/TP 3

Objectifs

Utilisation de la programmation orientée objet en situation réelle : un algorithme de compression d'une image à partir d'une technique de maillage.
Vous devrez employer les mécanismes vus en cours permettant d'introduire des arguments de qualité logicielle (jusqu'à l'utilisation de la bibliothèque STL).

Un compte-rendu de TP individuel devra être produit (uniquement ce TD/TP 3). Les différents éléments du compte-rendu (code source, rapport au format PDF) devront être envoyés à l'adresse pascal.have@ifpenergiesnouvelles.fr pour le mercredi 14 janvier 2015 minuit.

Compression d'image : méthode du maillage

Considérons une image comme une structure 2D de pixels portant chacun une couleur (niveau de gris) entre 0 (noir) et 255 (blanc). Chaque pixel a une position définie par des coordonnées entières dans les bornes de l'image (hauteur x largeur).

L'idée est de construire un maillage de triangles à partir d'une image où chaque triangle portera une couleur définie à partir des couleurs portées par les sommets du triangle (ex : moyenne des couleurs aux sommets).

Figure 1 : Triangulation et affichage de l'image compressée

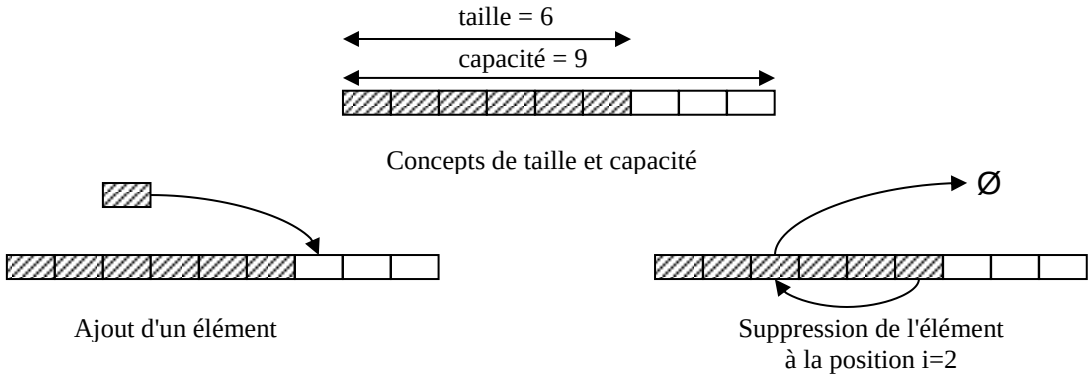
Sous quelles conditions peut-on dire qu'une telle construire peut produire une compression de l'image initiale ?

Outil : Tableau générique

Réalisez une classe `Tableau` générique représentant un conteneur permettant de contenir des éléments d'un type uniforme `T` avec les capacités suivantes :

- Ajout d'un élément en fin de conteneur (méthode `void Tableau::ajoute(const T&)`). Lors d'un ajout, le tableau devra éventuellement s'agrandir pour offrir la place nécessaire au nouvel élément. Même si le tableau s'agrandit les valeurs contenues ne sont pas *perdues*.
- Suppression de l'élément à la position i (méthode `void Tableau::supprime(int i)`). Les éléments d'indice supérieur à i devront être décalés d'une position. Les *pointeurs* sur une position d'index supérieur à i sont invalidés.
- Séparation du concept de `taille` (nombre d'éléments *actifs* du conteneur) et de la `capacité` (nombre d'éléments stockage avant nouvel agrandissement du conteneur). Des accesseurs associés à ces deux grandeurs seront fournis.

- Dimensionnement de la taille (méthode `void Tableau::retaille(int)`) et de la capacité (méthode `void Tableau::reserve(int)`).
- Accès direct à l'élément à la position i en lecteur **et** écriture (*cf.* `operator[](int)`). Vous ajouterez un contrôle de validité des positions demandées (*cf.* `assert`)
- Copie de tableau (par constructeur **et** opérateur).



Concepts de taille et capacité

Ajout d'un élément

Suppression de l'élément
à la position i=2

Pour réaliser cette classe `Tableau` générique, vous pourrez commencer par un tableau d'un type `T` donné par un `typedef` puis appliquer la méthode de *transformation* évoquée durant le cours (finaliser cette classe avec ses tests avant de passer à la *transformation*).

Coordonnées entières

Réalisez une classe `N2` représentant des coordonnées entières 2D ayant des capacités numériques telles que (liste non exhaustive à compléter suivant les besoins algorithmiques qui suivent) :

- Constructeur d'un `N2` par défaut (valeurs à 0)
- Constructeur d'un `N2` à partir de 2 entiers (constructeur `N2::N2(int x, int y)`)
- Somme et différence de deux `N2`
- Division par un scalaire (entier)
- Accès aux composantes (méthodes `int N2::x()` et `int N2::y()`)
- Copie de `N2` (constructeur par copie et opérateur de copie)
- Écriture d'un `N2` sur un *flux* (*cf.* `operator<<`)

Pixel et Image

Un `Pixel` est une position (de type `N2`) portant une couleur. Faites en sorte, avec un minimum de code que l'on puisse effectuer des opérations numériques similaires à `N2` dessus. En complément, fournissez un accès à la propriété `couleur` (lecture seule) et un constructeur à partir d'une couleur et d'une position de type N2 ou représentée par un couple d'entiers (X,Y).

À partir du lecteur d'image au format PGM fourni construisez une classe `Image` représentant les pixels d'une image en niveau de gris.
La classe `Image` doit fournir les services suivants :

- Pixel à partir d'une position de type `N2` (méthode `Pixel Image::pixel(const N2& X)`)
- Pixel à partir d'une position (x,y) (méthode `Pixel Image::pixel(const int& x, const int& y)`)
- Nombre de lignes de l'image (méthode `int Image::nbLignes()`)
- Nombre de colonnes de l'image (méthode `int Image::nbColonnes()`)

Facultatif : Séparer les concepts d'image et de lecteur à partir d'un fichier afin de faciliter l'introduction de nouveaux lecteurs pour d'autres formats sans impact sur le reste de l'algorithme.

Triangle et maillage

Réalisez une classe `Triangle` représentant un triangle à coordonnées entières avec les fonctionnalités suivantes :

- Construction d'un triangle à partir de trois `Pixel`s.
- Calcul du barycentre du triangle (position et couleur décrit via un `Pixel`).
- Accès aux Pixels formant les sommets du triangle (y ajouter un contrôle d'accès)
  `const Pixel& Triangle::operator()(const int& i) const` (pour `i`=0, 1 ou 2)
- Test d'appartenant d'un point à coordonnées entières au cercle circonscrit à un triangle (méthode `bool Triangle::cercleCirconscritContient(const N2& d) const`, voir ci-dessous).

Un `Maillage` est ici un ensemble de triangles discrétisant une image fournie au moment de sa construction (et enregistrée dans l'attribut `m_image`). Un `Maillage` ne possède que deux méthodes publiques :

- `void Maillage::sauvegarde(const char * filename) const`
  qui sauve la triangulation dans un fichier de nom `filename` sous un format Medit[1] (.mesh) ou Paraview[2] (unstructured .vtk)
- `void Maillage::ajoute(const int n, const int precision)`
  qui itère jusqu'à `n` fois la procédure d'insertion d'un point dans un maillage de Delaunay tant que l'erreur de couleur sur le point ajouté (écart entre les couleurs de référence de l'image et du point ajouté) est supérieure à `precision`.

Algorithme de Delaunay

Une triangulation de Delaunay est un maillage de qui vérifie la propriété de la boule vide. C'est-à-dire que le cercle circonscrit à tout triangle ne contient aucun autre triangle. En fait, il suffit que toutes les paires de triangles adjacents par une arête vérifient cette propriété pour qu'elle soit vérifiée globalement. De plus, si aucune paire de triangles adjacents ne forme un trapèze, la triangulation de Delaunay est unique. Enfin, la propriété qui nous intéresse le plus pour ce projet est que partant d'une triangulation de Delaunay, pour insérer un sommet *P*, il suffit de supprimer les triangles dont les cercles circonscrits contiennent *P* puis d'étoiler la cavité obtenue à partir de *P*.
L'algorithme de compression d'image est le suivant. On choisit comme maillage initial le rectangle défini par les 4 coins de l'image et on le coupe par une de ses diagonales. Ensuite, on regarde la différence entre la valeur interpolée en certains points des triangles (ex : barycentre du triangle) et la couleur du pixel dans l'image originale. Si l'écart est jugé trop important dans l'un de ces triangles, on ajoute alors un sommet à la triangulation (ex : le barycentre du triangle choisi). On répète ce processus de manière itérative, jusqu'à obtenir satisfaction (nombre de triangles ou erreur d'approximation).

Soient une triangulation T_n et un point P à y ajouter.
L'insertion de ce point passe par plusieurs phases :

1. Recherche des triangles dont le cercle circonscrit contient le point P de coordonnées D. Pour cela, nous utilisons la méthode du déterminant. Soit un triangle de sommet A, B et C. Considérons le déterminant suivant :

$$\begin{vmatrix} A_x & A_y & A_x^2+A_y^2 & 1 \\ B_x & B_y & B_x^2+B_y^2 & 1 \\ C_x & C_y & C_x^2+C_y^2 & 1 \\ D_x & D_y & D_x^2+D_y^2 & 1 \end{vmatrix} = \begin{vmatrix} A_x-D_x & A_y-D_y & (A_x^2-D_x^2)+(A_y^2-D_y^2) \\ B_x-D_x & B_y-D_y & (B_x^2-D_x^2)+(B_y^2-D_y^2) \\ C_x-D_x & C_y-D_y & (C_x^2-D_x^2)+(C_y^2-D_y^2) \end{vmatrix} > 0$$

   En supposant que A, B et C sont placés dans le sens anti-horaire, ce déterminant est positif si et seulement si D se trouve dans le cercle circonscrit.

[1] http://www.ann.jussieu.fr/~frey/software.html (disponible sous Linux x86 / MacOSX / Windows)
[2] http://www.paraview.org (disponible sous Linux x86 / MacOSX / Windows)

2. Définition du bord de la *cavité*[3] formée la suppression des triangles de la phase 1. Pour des raisons pratiques, nous ordonnerons la liste des points formant la cavité suivant leur angle relativement au point P (utiliser la fonction `atan2`).
3. Étoilement de la cavité par le point P

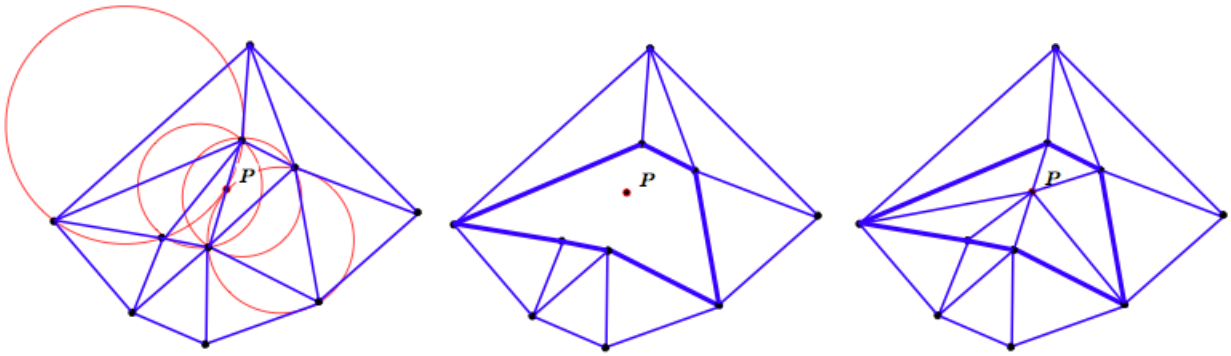


Figure 2 : Etape d'insertion d'un point

Remarques

- Penser à toujours bien orienter vos triangles de manière uniforme afin d'uniformiser le traitement sur ceux-ci (l'orientation directe étant la plus simple pour la plupart des calculs).
- Si vous avez fait `Tableau`, c'est pour l'utiliser.
- Pour simplifier certains calculs, il est conseillé de prévoir la bijection `Image::numero` qui transforme un point `N2` d'une image en un entier de l'intervalle [0: largeur x hauteur[(et réciproquement).
- Attention, avec les coordonnées entières le barycentre peut être un sommet du triangle ou un point déjà utilisé dans la triangulation, il convient alors de ne pas l'insérer de nouveau.
- Ajouter tous les qualificatifs `const` nécessaires (les précédents prototypes de méthodes peuvent être en ce sens incomplet)
- Pour éviter tout désagréable débordement d'entier `int` lors du calcul du déterminant utilisez un type `long long int`.
- Pour créer des images au format PGM vous pouvez utiliser la commande suivante: `convert image.jpg image.pgm` (convert fait partie du package ImageMagick)

Figure 3 : Exemple trivial de format VTK

http://www.vtk.org/VTK/img/file-formats.pdf

```
# vtk DataFile Version 3.0
Commentaire sur ce maillage
ASCII
DATASET UNSTRUCTURED_GRID

POINTS 4 float
0 0 0
639 0 0
0 -479 0
639 -479 0

CELLS 2 8
3 0 1 2
3 1 3 2

CELL_TYPES 2
5
5

CELL_DATA 2
SCALARS data int 1
LOOKUP_TABLE default
189
161

POINT_DATA 4
SCALARS node_data int 1
LOOKUP_TABLE default
1
2
3
4
```

[3] La classe `Cavite` du fichier `cavite.hpp` fourni implémente en grande partie les phases 1 et 2 et peut être directement réutilisé dès lors que les différents prototypes de fonctions demandées sont respectés.