








TP21 – Templates

Objectifs du TP


L'objectif du TP est de reprendre la classe matrice du TP 14 et de la rendre générique. C'est-à-dire que l'on veut que cette classe soit apte à traiter des éléments de n'importe quels types (**double**, **float**, **int**, **CBaleine**, etc...). Ce TP se découpe en deux parties. La première est destinée à la modification de la classe du TP 14, la seconde permet d'utiliser cette classe en situation réelle dans une application de traitement d'images.

Exercice 1 Modification de la classe `CMatrice`.

 Pour cet exercice, vous utiliserez la solution **TP 21 – Test.sln**. Une version interactive de l'énoncé de cet exercice est disponible dans le fichier **doc-test\html\index.html**, qui est aussi le point d'entrée de l'aide en ligne du code déjà réalisé.

-  Copiez les **contenus** des fichiers **Matrice.h** et **Matrice.cpp** du TP 14 dans leurs homologues de ce TP.
-  En utilisant la technique des **templates**, modifiez la classe **CMatrice** de sorte que les éléments stockés soient d'un type générique.
 -  Que devient le fichier **Matrice.cpp** ?
-  Exécutez le programme de test de façon qu'il passe tous les tests...
 -  Vous devrez dans un premier temps faire en sorte qu'il compile en ajoutant à la classe **CMatrice** les membres adéquats
 -  Adaptez le template **CMatrice** au besoin.

Exercice 2 Utilisation de la classe `CMatrice` dans un cas réel

 Pour cet exercice, vous utiliserez la solution **TP 21 – Graphique.sln**. Une version interactive de l'énoncé de cet exercice est disponible dans le fichier **doc-graphique\html\index.html**, qui est aussi le point d'entrée de l'aide en ligne du code déjà réalisé.

2.1 Objectif du programme.

Il s'agit d'un programme qui applique une transformation géométrique à une image. La transformation est représentée par une matrice 3x3 dont le rôle est de calculer les nouvelles coordonnées des pixels de l'image originale.

2.1.1 Un peu de maths...

2.1.1.1 Coordonnées homogènes et transformations géométriques

Un point du plan est localisé à l'aide de son abscisse x et de son ordonnée y . On définit les coordonnées homogènes de ce point comme étant tous les triplets de valeurs proportionnels à $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$.

C'est-à-dire que le même point est aussi bien repéré par $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ que par $\begin{pmatrix} 42x \\ 42y \\ 42 \end{pmatrix}$ ou encore $\begin{pmatrix} \alpha x \\ \alpha y \\ \alpha \end{pmatrix}$ avec α non nul.

Ainsi, en utilisant les coordonnées homogènes, on peut représenter sous forme de matrices toutes les transformations homographiques (toutes les transformations qui conservent l'alignement des points). Par exemple, une translation de vecteur $\begin{pmatrix} t_x \\ t_y \end{pmatrix}$ (un cas particulier des homographies) est représentée par la matrice $\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$. Pour appliquer une transformation géométrique, il suffit de calculer le produit matriciel entre la matrice de transformation et les coordonnées homogènes du point à transformer. Ainsi, l'application de la translation précédente au point d'abscisse x et d'ordonnée y se traduit par : $\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$.

2.1.2 Les différentes transformations considérées par l'application

L'application pourra appliquer des rotations (matrice R), agrandissements (matrice S) et déchirements (matrice T), ainsi que toutes les combinaisons de ces transformations. L'ensemble de ces combinaisons s'appellent les transformations affines (matrice A) (toutes les transformations qui conservent le parallélisme – un sous-ensemble des homographies). Pour combiner des transformations, il suffit de multiplier leurs matrices ensembles. Nous appliquerons dans l'ordre la rotation, la mise à l'échelle puis le déchirement, ainsi nous avons $A = T S R$.

2.1.2.1 Matrices de rotation

Une matrice de rotation d'angle θ autour de l'origine s'écrit : $R = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

2.1.2.2 Matrices d'agrandissement / réduction

Une matrice de mise à l'échelle de facteurs α_x selon les abscisses et α_y selon les ordonnées s'écrit : $S = \begin{pmatrix} \alpha_x & 0 & 0 \\ 0 & \alpha_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

2.1.2.3 Matrices de déchirement

Une matrice de déchirement de facteurs β_x selon les abscisses et β_y selon les ordonnées s'écrit : $T = \begin{pmatrix} 1 & \beta_x & 0 \\ \beta_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

2.1.3 Transformer une image

Pour transformer une image, il suffit de calculer la nouvelle position des pixels. Soit $p = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ les coordonnées d'un pixel de l'image originale, sa position dans l'image transformée sera alors $p' = A p$. Or les coordonnées des pixels doivent être entières, et avec le calcul $A p$ il y a peu de chance de tomber sur un résultat entier. Donc si l'on souhaite appliquer l'algorithme dans ce sens direct, il faudra rendre entière les coordonnées et l'on risque alors de rater des pixels de l'image d'arrivée. Le plus sûr est alors de déterminer la position d'un pixel dans l'image originale à partir de sa position dans l'image

transformée, c'est-à-dire calculer p à partir de p' . L'algorithme consiste alors à parcourir tous les pixels p' de l'image transformée et de calculer leur position $p = A^{-1}p'$ dans l'image de départ pour connaître leur couleur. Si le calcul ne tombe pas sur une position de pixel entière, alors on prendra le pixel le plus proche.

2.2 Présentation de la structure du programme

2.2.1 La classe CTP (fichiers TP.h et TP.cpp)






Le programme s'articule toujours autour d'un objet de type **CTP** dont les méthodes sont appelées par le programme principal en fonction des actions de l'utilisateur. Cette classe gèrera (une fois que vous l'aurez fait dans la partie 2.3) les transformations de l'image.

2.2.2 La classe CBitmap

La classe **CBitmap** gère un tableau de pixel. La fonction **CBitmap::Transform** implémente l'algorithme précédent pour transformer l'image.

2.3 Travail à réaliser







2.3.1 Classe CTP

-  Ajouter à la classe **CTP** 3 matrices membres privés. Ces matrices 3x3 doivent travailler sur des **double**.
 -  Une matrice **m_matRot** pour la matrice de rotation.
 -  Une matrice **m_matScale** pour la matrice de mise à l'échelle.
 -  Une matrice **m_matTear** pour la matrice de déchirement.
-  Dans le fichier **CTP**, référez-vous aux commentaires « **\todo** » pour modifier le code.

2.3.2 Classe CBitmap

-  Modifiez la fonction **CBitmap::Transform** en vous référant aux commentaires « **\todo** ».

Exercice 3 Méta programmation

-  Créez une nouvelle solution Visual Studio
-  En vous inspirant du programme de mesure de performances vu en cours et du TD 07, évaluez les performances de calculs de $12!$ par une méthode itérative, une méthode récursive et une méthode en méta programmation.
 -  Qu'en concluez-vous ?
-  Calculez par les trois méthodes $13!$
 -  Vérifiez les résultats des calculs.
 -  S'ils sont incorrects, comment feriez-vous pour les rendre correct ?