# Algorithmique des images TD n° 2

***Transformation d'images***

## Exercice 1. Transformation d'images

Le but de ce TD est d'implanter des fonctions permettant d'appliquer les transformations géométriques de base d'une image.

Dans ce TD on suppose que l'on utilisera la structure `ppm_t` définie dans le TD précédent. L'ordre d'interpolation sera défini comme une macro dans notre code :

```
#define N 1 // Interpolation d'ordre 1
```

On rappelle ci dessous les expressions des fonctions de poids nécessaires à l'interpolation :

- Ordre 0 :

$$.\beta_0(x) = \begin{cases} 0 & \texttt{si } |x| > 0.5 \\ 1 & \texttt{si } |x| < 0.5 \\ 0.5 & \texttt{si } |x| = 0.5 \end{cases}$$

- Ordre 1:

$$.\beta_1(x) = \begin{cases} 0 & \texttt{si } |x| > 1 \\ x+1 & \texttt{si } -1 \leq x \leq 0 \\ 1-x & \texttt{si } 0 \leq x \leq 1 \end{cases}$$

- Ordre 2:

$$.\beta_2(x) = \begin{cases} 0 & \texttt{si } |x| > 1.5 \\ 0.5(x+1.4)^2 & \texttt{si } -1.5 \leq x \leq -0.5 \\ 0.75 - x^2 & \texttt{si } -0.5 \leq x \leq 0.5 \\ 0.5(x-1.5)^2 & \texttt{si } 0.5 \leq x \leq 1.5 \end{cases}$$

- Ordre 3:

$$.\beta_3(x) = \begin{cases} 0 & \texttt{si } |x| > 2 \\ 1/2|x|^3 - x^2 + 2/3 & \texttt{si } 0 \leq |x| \leq 1 \\ 1/6(2-|x|)^3 & \texttt{si } 1 \leq |x| \leq 2 \end{cases}$$

### Q- 1.1 Fonction d'interpolation

Créer des fonctions `double B0(double x)`, `double B1(double x)`, `double B2(double x)` et `double B3(double x)` qui renvoient les valeurs respectives de $\beta_0(x)$, $\beta_1(x)$, $\beta_2(x)$ et $\beta_3(x)$.

On définit en variable globale un tableau de pointeurs sur fonctions `B` de sorte à pouvoir appeler la bonne fonction $\omega$ selon la valeur de $N$.

```
double (*B[4])(double) = {B0, B1, B2, B3};
```

Ainsi en appelant `B[N]` on appellera la fonction de poids pour l'interpolation à l'ordre `N`.

### Q- 1.2 Interpolation en niveau de gris

Créer une fonction `unsigned char interpolation_pgm(pgm_t *image, double x, double y)` qui renvoie la valeur interpolée du pixel en nuance de gris de coordonnées `(x,y)` dans l'image `image`.

### Q- 1.3 Interpolation en couleur

Créer une fonction `rgb interpolation_ppm(ppm_t *image, double x, double y)` qui renvoie la valeur interpolée du pixel en rgb de coordonées `(x,y)` dans l'image `image`.

### Q- 1.4 Rotations

Créer des fonctions `pgm_t *rotation_pgm(pgm_t *image, double theta, int x0, int y0)` et `ppm_t *rotation_ppm(ppm_t *image, double theta, int x0, int y0)` qui calculent les rotations d'angle `theta` et de centre le point de coordonnées `(x0,y0)` des images au format `pgm` et `ppm` passées en paramètres.

**Attention** : les fonctions `cos` et `sin` de la bibliothèque `math.h` prennent les angles en paramètre en radiant.

### Q- 1.5 Zoom et cisaillement

En suivant le modèle précédent, créer les fonctions :

- `ppm_t *zoom(ppm_t *image, double lambda, int x0, int y0, int Dx, int Dy)` qui crée une image de taille `Dx x Dy` correspondant au zoom de facteur `lambda` autour du point de coordonnées `(x0,y0)` de l'image passée en paramètre.
- `ppm_t *shear(ppm_t *image, double cx, double cy, int Dx, int Dy)` qui crée une image de taille `Dx x Dy` correspondant au cisaillement de facteur `cx`, `cy` de l'image passée en paramètre.

## Exercice 2. Transformations affines

On considère maintenant la structure suivante permettant de stocker les coordonnées d'un pixels :

```c
struct point
{
  int x,y;
};
typedef struct point point_t;
```

### Q- 2.1 Transformations

Créer une fonction `double *get_affine_transformation(point X_start[3], point X_end[3])` qui prend en paramètres trois couples de coordonnées de pixels et renvoie les six coefficients de la transformation affine qui permet de passer de l'une à l'autre.

### Q- 2.2 Généralisation

Généraliser les questions précédentes en créant une fonction `ppm_t *affine_transformation(ppm_t *image, double *coeff_transformation)` qui renvoie l'image correspondante à la transformation affine donnée par les coefficients `coeff_transformation` de l'image `image` passée en paramètre.

**Remarque** : on n'oubliera pas de calculer l'inverse de la transformation passée en paramètre afin de pouvoir calculer la transformée de l'image.

## Exercice 3. Programme de test

### Q- 3.1 Main

Créer un programme qui prendra en paramètre une image et le nom de la transformation à appliquer le programme générera l'image transformée dans un fichier par défaut.