**Note : dans tous les exercices le paramètre `<factor>` est un entier.**

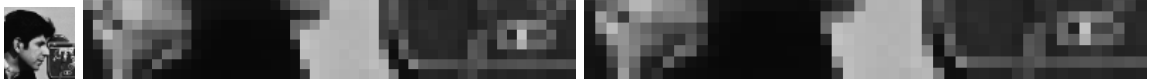
## copy <factor> <ims> <imd>

Écrire le programme `copy.c`

— le programme permet d'agrandir une image source `ims` de taille `factor` (dans les deux dimensions) par recopie de pixels et sauve le résultat dans `imd`

— le paramètre `<factor>` est considéré comme entier

— par exemple, le pixel $I(i, j)$ de l'image source est dupliqué en $I'(i, j)$, $I'(i, j + 1)$, $I'(i + 1, j)$ et $I'(i + 1, j + 1)$, si on considère que l'image $I'$ a été agrandi d'un facteur 2



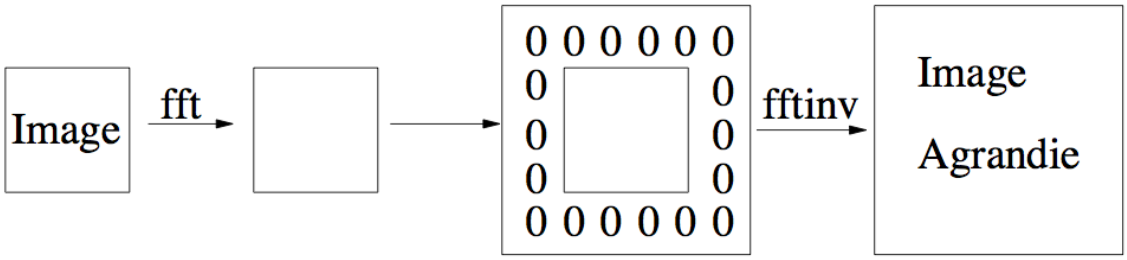
`./copy 10 ../data/cameraman.ppm a.ppm`

`./copy 15 ../data/cameraman.ppm a.ppm`

## padding <factor> <ims> <imd>

Écrire le programme `padding.c`

— le programme permet d'agrandir une image source `ims` de taille `factor` (dans les deux dimensions) en utilisant la transformée de Fourier et sauve le résultat dans `imd`. Le programme suit les étapes suivantes




— rappel : les images dans le plan de Fourier sont de type complexe $\mathbb{C}$. L'agrandissement de l'image induit une modification du facteur de normalisation (qui dépend de la taille du domaine) de l'équation de Fourier inverse (notamment pour compenser une perte de dynamique des niveaux de gris due à l'agrandissement). Une petite modification de votre module `fft.h` et `fft.c` sera peut-être nécessaire

`./padding 10 ../data/cameraman.ppm a.ppm`

`./padding 15 ../data/cameraman.ppm a.ppm`

## `filter <factor> <filter-name> <ims> <imd>`

Écrire le programme `filter.c`

— le programme permet d'agrandir une image source `ims` de taille `factor` (dans les deux dimensions) en utilisant des filtres d'interpolation (paramètre `filter-name={box, tent, bell, mitch}`) et sauve le résultat dans `imd`. Les filtres sont les suivants (**les filtres comportent des constantes, elles peuvent donc être pré-calculées**) :

— **box** :

$$h(x) = \begin{cases} 1 & \text{si } x \in [-\frac{1}{2}, \frac{1}{2}[ \\ 0 & \text{sinon} \end{cases} \tag{1}$$

— **tent** :

$$h(x) = \begin{cases} 1 - |x| & \text{si } x \in [-1, 1] \\ 0 & \text{sinon} \end{cases} \tag{2}$$

— **bell** :

$$h(x) = \begin{cases} -x^2 + \frac{3}{4} & \text{si } |x| \leq \frac{1}{2} \\ \frac{1}{2}(|x| - \frac{3}{2})^2 & \text{si } \frac{1}{2} < |x| \leq \frac{3}{2} \\ 0 & \text{sinon} \end{cases} \tag{3}$$

— **Mitchell-Netravali** :

$$h(x) = \begin{cases} \frac{7}{6}|x|^3 - 2x^2 + \frac{8}{9} & \text{si } x \in [-1, +1] \\ -\frac{7}{18}|x|^3 + 2x^2 - \frac{10}{3}|x| + \frac{16}{9} & \text{si } x \in [-2, -1] \cup [1, 2] \\ 0 & \text{sinon} \end{cases} \tag{4}$$

— L'algorithme générale en pseudo-code permettant de faire l'interpolation par convolution des pixels manquants est le suivant (**cet algorithme ne fait l'interpolation uniquement qu'en colonne**)

— `I: image source of width W`
— `I': resampled image of width W'=factor×W (factor > 1)`
— `h: filter of half size WF` (Equations from (1) to (4) ; the filter size corresponds to its domain [1]).
— `i: the ith resampled line`
— `forall j' in [0,W'[ do:`

1. `j ← j'/factor`
2. `left ← j-WF, right ← j+WF, S ← 0`
3. `forall k in [left, right] do:`
   (a) `S ← S + I[i][k]*h(k-j)`
4. `done`
5. `I'[i][j'] ← S`

`done`

1. for instance the size of the box filter (1) equals 1

./filter 15 box ../data/cameraman.ppm a.ppm

./filter 15 tent ../data/cameraman.ppm a.ppm

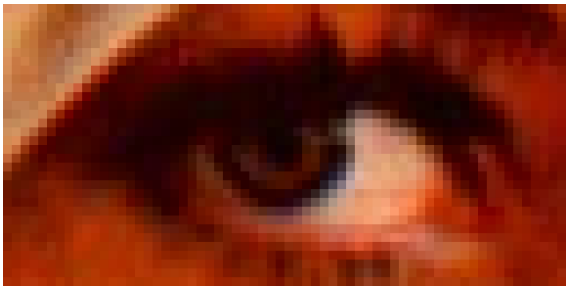
./filter 15 bell ../data/cameraman.ppm a.ppm

./filter 15 mitch ../data/cameraman.ppm a.ppm

## Extension à la couleur

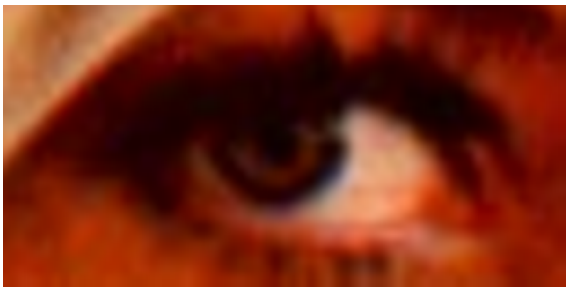
— Copier votre dossier `zoom` dans un dossier `zoom-color`
— Modifier vos programmes de prendre en charge le zoom pour de images couleurs. Les traitements se font simplement, canal par canal

`lena-small-rect.ppm`



`lena-small-rect-copy-10.ppm`

`lena-small-rect-padding-10.ppm`



`lena-small-rect-mitch-10.ppm`