TP1 : Convolution : Lissage et détection de contours

Un système d'enregistrement d'image ne restitue pas l'image de manière parfaite : des informations parasites apparaissent et viennent s'ajouter de manière aléatoire aux détails de la scène d'origine. Cet effet est désigné sous le terme général de "bruit". Pour un appareil photo numérique, le bruit est principalement dû à la physique des capteurs et aux composants électroniques qui exploitent le signal provenant des capteurs. Le lissage a pour but de réduire l'influence du bruit, afin d'obtenir une meilleure restitution de l'image pour son analyse. La détection de contours cherche à déterminer quels sont les pixels qui forment les bords des objets d'une scène.

Ce premier tp introduit les filtres linéaires. Un filtre F(x) est linéaire lorsqu'il répond à la condition suivante : $F(\alpha x) = \alpha F(x)$. Il est implanté par une convolution.

L'objectif du tp est d'implanter et étudier 2 techniques de convolution (spatiale dans l'espace image et fréquentielle dans l'espace de Fourrier) soit pour lisser une image (première partie), soit pour effectuer une détection de contours (deuxième partie). La première partie sur le lissage gaussien teste et d'établit les règles d'utilisation de ces 2 méthodes d'implémentation de la convolution tandis que la deuxième partie compare la détection de contours par gradient et par laplacien.

1 Lissage linéaire

Cette catégorie de filtres correspond à des filtres passe-bas, éliminant les hautes fréquences qui représentent généralement le bruit. Le problème est de déterminer la fréquence de coupure exacte du filtre car les contours des objets sont eux aussi représentés par des fréquences relativement élevées, qu'il faut conserver. Le rôle d'un filtre de lissage peut être résumé en quatre actions :

- Tous les pixels d'une zone relativement homogène doivent être à une même valeur : cela revient à réduire la variance de cette zone homogène.
- Tous les pixels proches d'un contour doivent appartenir à une zone homogène s'ils n'appartiennent pas à ce contour. Cela revient à ne pas créer de nouvelles zones ou valeurs à proximité des points de contours.
- Aucun point de contour ne doit disparaître : le filtre ne doit pas entraîner de perte de résolution sur l'image, en particulier sur les zones où les contours ont de 1 ou 2 pixels d'épaisseurs (Images de caractères ou d'empreintes digitales).
- Le filtre ne doit pas entraîner de déformations de ces contours (en particulier pour les coins et les intersections de contours) et ne doit pas déplacer ces contours (en particulier lorsque des mesures métriques sont envisagées).

Les filtres linéaires permettent une réduction du bruit dans les images, mais ont l'inconvénient de traiter de manière identique le bruit et les signaux utiles (contour), et donc de dégrader l'information pertinente dans le cas d'une détection de contours. Par contre, les filtres linéaires prennent en compte de la hauteur des bruits qu'ils ont à traiter. Généralement, leur action est proportionnelle à la valeur du bruit à éliminer.

1.1 FFT et filtrage fréquentiel

La transformée de Fourier discrète d'une image de taille $N \times M$ s'écrit :

$$\mathrm{TF}(I)(u,v) = R(u,v) = \frac{1}{\sqrt{N.M}} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} I(x,y) e^{-2.i.\pi\left(\frac{u.x}{N} + \frac{y.v}{M}\right)}$$

Lors du passage du domaine continu au discret, si le signal continu est discrétisé à la fréquence $1/T_e$ pour un signal de N échantillons, la FFT périodise le signal ainsi que le résultat, qui est échantillonné à la fréquence $1/(N.T_e)$. L'algorithme calcule la FFT pour des valeurs positives $0 <= u <= N$, correspondant aux fréquences $u/(N.T_e)$, alors que les représentations continues classiques visualisent le résultat entre les indices $-N/2$ et $N/2$. On revient à cette représentation grâce à l'aspect périodique de la TF avec la relation $R(u) = R(u+N)$ pour retrouver les valeurs aux indices $-N/2 <= u <= 0$.

En 2D, la disposition des fréquences obtenue par les algorithmes de FFT est celle de la figure 1a, avec les hautes fréquences centrées en $N/2, M/2$ et les basses fréquences situées dans les angles. En replaçant les basses fréquences au centre de l'image, on obtient une représentation fréquentielle plus conforme (figure 1b), avec l'origine des coordonnées fréquentielles en ($N/2$, $M/2$) obtenue en utilisant la fonction fftshift, en utilisant la périodicité des 2 axes de l'image.

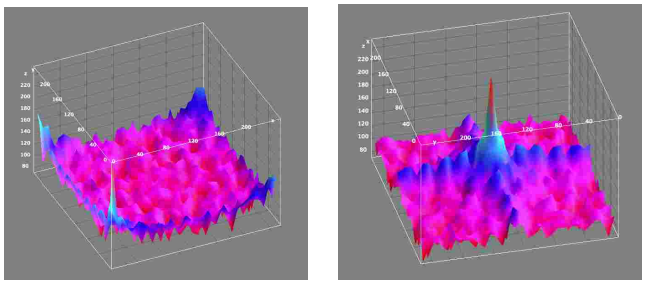

FIGURE 1 – a) : a gauche, représentation directe : les fréquences hautes sont au centre b) a droite, représentation shiftée : les fréquences basses sont au centre

Un des intérêt de la transformée de Fourier pour le filtrage est que celle ci transforme la convolution en produit dans l'espace fréquentiel, ce qui donne une complexité en $o(N \times M)$:

$$TF(I * H) = TF(I) \times TF(H)$$

On peut donc appliquer un filtre linéaire H en multipliant la transformée de Fourier d'une image par la transformée de Fourier du filtre, puis en effectuant la transformée de Fourier inverse du résultat.

$$I * H = TF^{-1}(TF(I) \times TF(H))$$

1.1.1 Filtrage gaussien par FFT

Le filtrage gaussien est une filtrage linéaire assez efficace en terme de débruitage et trouve un écho dans le système visuel humain : c'est en effet le type de filtrage réalisé par les premières cellules du processus visuel humain situées après les cellules photo-réceptrices de l'œil, avec plusieurs tailles de filtres sur des canaux différents. La gaussienne bidimensionnelle s'écrit :

$$G(x,y) = \frac{1}{\sigma_x\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma_x^2}} . \frac{1}{\sigma_y\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma_y^2}}$$

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \text{si} \quad \sigma_x = \sigma_y = \sigma$$

La FFT de cette gaussienne :

$$TF\left(G(x,y)\right) = \widetilde{G}(u,v) = e^{-2\pi^2\sigma^2(u^2+v^2)}$$

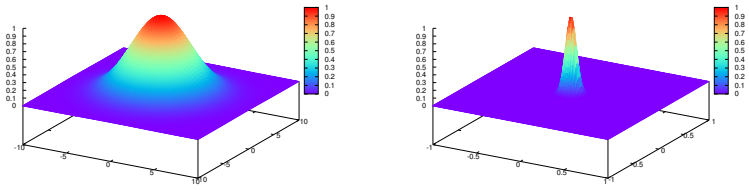

FIGURE 2 – Gaussienne à gauche et sa FFT à droite (supports différents)

On peut donc facilement réaliser un filtrage gaussien en définissant directement le filtre dans l'espace de Fourrier (sans faire de FFT). Il suffit ensuite de faire le produit de la TF de l'image avec ce filtre puis de prendre la TF^{-1} pour réaliser un filtrage gaussien. Le paramètre σ est lié à la taille des objets présents dans l'image. Les avantages sont :

- Le filtrage est presque exact, car les coefficients sont rapidement presque nuls
- La taille de la gaussienne n'intervient pas dans le design du filtre
- La complexité et le temps de calcul sont constants pour des σ différents
- Il existe des bibliothèques particulièrement optimisées pour le calcul de la fft (voir FFTW).

1.1.2 Remarques importantes pour la mise en œuvre :

- Le filtre est centré sur le milieu de l'image. Il faut donc penser à "shifter" la TF de l'image avant de faire le produit, puis penser à "shifter" le produit avant de faire la TF^{-1}.
- Les fréquences sont normalisées. Dans le domaine discret, la transformée de Fourrier de la gaussienne s'exprime par :

$$TF(G(u,v)) = e^{-2\pi^2\sigma^2\left(\left(\frac{u-N/2}{N}\right)^2+\left(\frac{v-M/2}{M}\right)^2\right)}$$
$$\text{avec } 0 \leq u < N$$
$$\text{et } 0 \leq v < M$$

- Attention à la mise en oeuvre en C/C++ des divisions entières

1.1.3 Comparaison

Si l'image non bruitée est connue, une mesure de comparaison est le Peak Signal Noise Ratio (PSNR) basé sur l'erreur quadratique entre l'image non bruitée connue que l'on cherche et l'image lissée qui approxime cette image. Plus l'image lissée ressemble à l'image recherchée, plus est le PSNR. Le PSNR est donné par :

$$PSNR(im_1, im_2) = 10.log\left(\frac{255^2.N.M}{\sum_{i=0}^{N}\sum_{j=0}^{M}(im_1(i,j) - im_2(i,j))^2}\right)$$

1.1.4 Travail à réaliser

L'objectif est de comparer l'effet du paramètre σ sur quelques images. Les images formes1bbxx.pgm sont des images synthétiques qui simulent un flou (défaut de mise au point par exemple) auxquelles on

a ajouté un bruit blanc. L'image formes1sp.pgm simule un bruit multiplicatif type speckle. Les images formes1pets1.pgm, formes1pets5.pgm, formes1pets10.pgm contiennent un bruit de type poivre et sel. L'image initiale étant connue, on peut calculer le PSNR entre l'image lissée et l'image parfaite pour quantifier la qualité du lissage.

1. Appliquer le filtre gaussien en utilisant la FFT. Le code de la FFT et un exemple d'utilisation sont disponibles sur le kiosk. Attention lors de la multiplication des transformées de Fourier à bien considérer les mêmes coordonnées fréquentielles : il est recommandé de replacer les basses fréquences au centre en utilisant fftshift.
2. Comparer sur les images bruitées disponibles sur le kiosk, qualitativement et quantitativement (plus le PSNR est élévé, plus la qualité du lissage est bonne) les résultats obtenus pour les différents types de bruit (gaussien, speeckle, poivre et sel) en fonction de la valeur de σ. Pour quel type de bruit le filtre gaussien est il efficace ?

1.2 Convolution spatiale

Dans le domaine continu, une convolution spatiale entre une image I et un filtre H est définie par :

$$R(x,y) = (I * H)(x,y) = (H * I)(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} I(x-x', y-y').H(x',y')dx'dy'$$

Dans le domaine discret, une convolution entre une image I et un filtre H est définie par :

$$R(x,y) = \sum_{i=-n}^{+n}\sum_{j=-m}^{+m} I(x+i, y+j).H(i,j)$$

1.2.1 Filtrage linéaire gaussien

Les coefficients du filtre discret H sont les valeurs d'une gaussienne à deux dimensions, normalisés de telle sorte que leur somme soit égale à 1. Ce type de filtre est efficace et trouve un écho dans le système visuel humain : c'est en effet le type de filtrage réalisé par les premières cellules du processus visuel humain situées après les cellules photo-réceptrices de l'œil, avec plusieurs tailles de filtres sur des canaux différents. La gaussienne bidimensionnelle s'écrit :

$$\begin{aligned} H(x,y) = G(x,y) &= \frac{1}{\sigma_x\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma_x^2}} . \frac{1}{\sigma_y\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma_y^2}} \\ &= \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \text{si} \quad \sigma_x = \sigma_y = \sigma \end{aligned}$$

Dans le domaine discret, la convolution s'écrit :

$$\begin{aligned} R(x,y) &= \frac{1}{\sigma_x\sqrt{2\pi}} . \frac{1}{\sigma_y\sqrt{2\pi}} \sum_{i=-n}^{+n}\sum_{j=-m}^{+m} e^{-\frac{i^2}{2\sigma_x^2}} . e^{-\frac{j^2}{2\sigma_y^2}} . I(x+i, y+j) \\ &= \frac{1}{2\pi\sigma^2} \sum_{i=-n}^{+n}\sum_{j=-m}^{+m} e^{-\frac{(i+j)^2}{2\sigma^2}} . I(x+i, y+j) \quad \text{si} \quad \sigma_x = \sigma_y = \sigma \end{aligned}$$

Remarque sur la mise en oeuvre : En pratique, les coefficients $e^{-\frac{(i+j)^2}{2\sigma^2}}$ sont précalculés et stockés dans un tableau de taille WxW réels. Pour un filtre isotrope en x,y, les bornes n et m sont finies, identiques et égales à W. W est la taille du tableau contenant les coefficients précalculés, normés à 1. Cette valeur W dépend de σ et doit permettre d'approximer ce filtre dans le domaine discret. Les plus petits coefficients

du filtre sont pratiquement nuls : $e^{-\frac{(W(\sigma)+W(\sigma))^2}{2\sigma^2}} \approx 0$.

$$H(x,y) = \frac{e^{-\frac{(x+y)^2}{2\sigma^2}}}{\sum_{i=-W(\sigma)}^{+W(\sigma)} \sum_{j=-W(\sigma)}^{+W(\sigma} e^{-\frac{(i+j)^2}{2\sigma^2}}} \quad avec\ x,y \in \mathbb{Z}\times\mathbb{Z}$$
$$et\ \ -W(\sigma) <= x <= W(\sigma)$$
$$et\ \ -W(\sigma) <= y <= W(\sigma)$$

1.2.2 Filtrage linéaire séparable

Un filtre séparable est un filtre tel que $H(x,y) = H_1(x).H_2(y)$. Dans ce cas, la convolution 2D s'écrit comme une convolution 1D par H_1, suivi par une convolution 1D par H_2.

$$\begin{aligned} R(x,y) = (I*H)(x,y) &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} I(x-x',y-y').H(x',y').dx'.dy' \\ &= \int_{-\infty}^{+\infty}\left[\int_{-\infty}^{+\infty} I(x-x',y-y').H_1(x').dx'\right].H_2(y').dy' \\ &= ((I*H_1)*H_2)(x,y) \end{aligned}$$

La fonction gaussienne G(x,y) étant séparable, la convolution bidimensionnelle se transforme en 2 convolutions monodimensionnelles, une selon l'axe horizontal suivie d'une autre selon l'axe vertical.

$$\begin{aligned} R(x,y) = (I*G)(x,y) &= \frac{1}{2\pi\sigma^2}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} I(x-x',y-y').e^{-\frac{x'^2+y'^2}{2\sigma^2}}.dx'.dy' \\ &= \frac{1}{2\pi\sigma^2}.\int_{-\infty}^{+\infty}\left[\int_{-\infty}^{+\infty} I(x-x',y-y').e^{-\frac{x'^2}{2\sigma^2}}.dx'\right]e^{-\frac{y'^2}{2\sigma^2}}.dy' \end{aligned}$$

Dans le domaine discret :

$$\begin{aligned} R(x,y) = (I*G)(x,y) &= \frac{1}{2\pi\sigma^2}\sum_{i=-n}^{+n}\left[\sum_{j=-m}^{+m} e^{-\frac{j^2}{2\sigma^2}}.I(x+i,y+j)\right].e^{-\frac{i^2}{2\sigma^2}} \\ &= ((I*G_x)*G_y)(x,y) \\ \text{avec}\quad G_x(i) = G_y(i) &= \frac{e^{-\frac{i^2}{2\sigma^2}}}{\sum_{i=-n}^{+n} e^{-\frac{i^2}{2\sigma^2}}} \end{aligned}$$

Avec le lissage séparable, le nombre d'opération est considérablement réduit.

Remarques sur la mise en oeuvre

1. Ce type de filtre, surtout dans sa version séparable, impose des calculs en réels : en calcul entier, la troncature effectuée lors du 1er passage provoque des erreurs importantes lors du 2ième passage.
2. Problème des bords : dans le domaine numérique, l'image est de taille finie. La convolution des pixels aux bord de l'image (i<W/2) fait appel à des pixels situés en dehors de l'image. Ces pixels peuvent prendre les valeurs suivantes :
   — Zéro
   — La valeur du pixels du bord : on prolonge l'image par continuité
   — La valeur des pixels du bord opposé de l'image : on périodise l'image. C'est cette version qui sera utilisée ici pour permettre une comparaison avec la lissage par FFT. Cela s'écrit facilement par $im[(i+N)\%N][(j+M)\%M]$ en C/C++.

1.2.3 Travail à réaliser

Implémenter la convolution gaussienne d'une image par un masque séparable, en périodisant l'image pour gérer les effets de bords. La taille du masque et la valeur du lissage doivent être paramétrables. Tracer l'écart entre la convolution par FFT et celle obtenue par un masque en fonction de la taille du masque (pour des valeurs fixées de $\sigma = 0.5, 1, 5, 10$ par exemple). Déterminer la loi empirique $W(\sigma)$ pour laquelle les résultats sont identiques par FFT et par un masque ?

1.3 Complexité et comparaison des 2 méthodes

1.3.1 Complexité

Pour le filtrage fréquentiel d'une image de taille $N \times M$ et un masque de taille $n \times m$, la multiplication nécessite $N \times M$ opérations, et l'algorithme FFT a une complexité en O($(N \times M).ln(N \times M$)), ce qui donne une complexité totale en O($(N \times M).ln(N \times M$)).

Pour un filtrage spatial d'une image de taille $N \times M$ et un masque de taille $n \times m$ la complexité est en O($(n + m).N.M$) si le filtre est séparable et de taille m, et en O($n.m.N.M$) si le filtre n'est pas séparable et de taille m.

1.3.2 Travail à réaliser

L'objectif est de comparer la convolution spatiale et la convolution par FFT du point de vue du temps de calcul pour une valeur de σ donnée. La largeur W du masque est la plus petite taille de filtre pour laquelle les résultats de filtrage sont identiques entre la version fréquentielle et la version spatiale.

En utilisant une taille de masque W convenable en fonction de σ, comparer les temps de calcul des deux implémentations pour des gaussiennes de variances croissantes. Pour quelles tailles faut il utiliser la version fréquentielle plutôt que la version spatiale ?

Remarque : pour mesurer le temps

```
#include <time.h>
main() { clock_t debut, fin ;
  debut=clock();
  fonction_dont_on_veut_mesurer_le_temps();
  fin=clock();
  printf("duree=\%f\n",((double)fin-debut)/CLOCKS_PER_SEC);
}
```

2 Détection de contours

Les frontières qui séparent les régions homogènes occupent une part importante dans le système visuel humain. On peut définir un contour comme une variation brutale de niveau de gris, plus ou moins entachée de bruit. Les neuro-physiologistes pensent que le système visuel humain commence par effectuer une détection de contour, certaines zones neuronales se spécialisant dans la détection dans des directions précises avec plusieurs lissages différents

D'un point de vue formel, en 2D, un contour est une rupture entre 2 surfaces à peu près constante : frontière entre 2 objets réels, entre 2 couleurs par exemple. Les modèles de contours classiques sont

présentés figure 3.

Les contours peuvent être détectés par un examen des grandeurs étudiées (niveau de gris, distance, couleur, ...) soit :

— par des opérateurs locaux de différentiation du premier ordre (figure 4)
— par des opérateurs locaux de différentiation du second ordre (figure 4)
— par des opérations ensemblistes (Morphologie mathématique).

$$\frac{\partial I}{\partial \vec{u}} = \lim_{h->0} \frac{I(\vec{x}+h.\vec{u}) - I(\vec{x})}{h} = \left\| \vec{\nabla} I \right\| .\vec{u} = \vec{G} = \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix}$$

$$\text{avec } G = \left\| \vec{\nabla} I \right\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \text{ et } \vec{u} = \frac{\vec{\nabla} I}{\left\| \vec{\nabla} I \right\|}$$

De part sa définition, un contour présente un maxima local dans une direction de l'espace. La fonction $g(\phi) = \vec{\nabla} I.\vec{u} = \frac{\partial I}{\partial x} cos(\phi) + \frac{\partial I}{\partial y} sin(\phi)$ possède donc un maxima local et sa dérivée selon ϕ s'annule :

$$\frac{\partial g}{\partial \phi} = \frac{\partial \vec{u}.\vec{\nabla} I}{\partial \phi} = \frac{\partial I}{\partial x}.sin\phi - \frac{\partial I}{\partial y}.cos\phi = 0 \Rightarrow \phi = arctg \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix}$$

Donc, un contour d'orientation ϕ au point de coordonnées (x, y) est détecté par un maximum de la dérivée première de la fonction image I(x,y), dans la direction ϕ du gradient, normale à la tangente au contour en ce point. Cette direction est celle de la plus grande pente de la fonction continue I(x,y). Le module de gradient est la variation locale d'intensité de l'image.

Et le laplacien passe par 0. Dans le repère de Frenet $\vec{t}, \vec{n}$, nous avons :

$$\Delta I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} = \frac{\partial^2 I}{\partial n^2} + \frac{\partial^2 I}{\partial t^2}$$

$$\text{si x,y est un contour, alors} \frac{\partial^2 I}{\partial t^2} >> \frac{\partial^2 I}{\partial n^2} \text{ et } \frac{\partial^2 I}{\partial n^2} = 0 \Rightarrow \Delta I = \frac{\partial^2 I}{\partial n^2} + \frac{\partial^2 I}{\partial t^2} = \frac{\partial^2 I}{\partial t^2} = 0$$

Donc, pour détecter les contours, nous avons 2 solutions :

— Calculer le gradient et vérifier si le point est maxima dans la direction du vecteur gradient.
— Calculer le laplacien et vérifier s'il y a un passage par zéro dans son voisinage.

En traitement d'image, on utilise couramment les différences finies pour calculer les opérateurs différentiels. A la fois pour obtenir une fonction continue différentiable et pour réduire le bruit, l'image est habituellement lissée avant ou simultanément aux calculs de gradient ou laplacien.

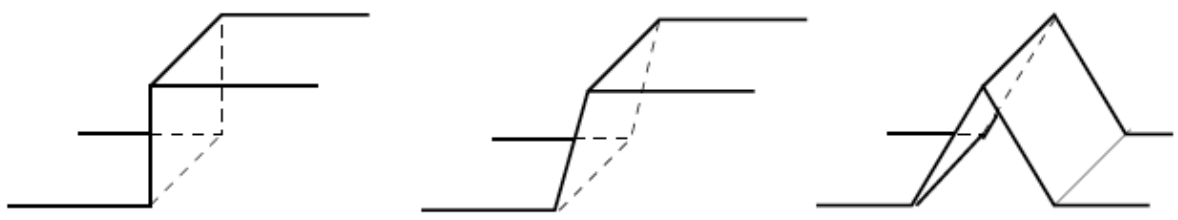

FIGURE 3 – Contours idéalisés

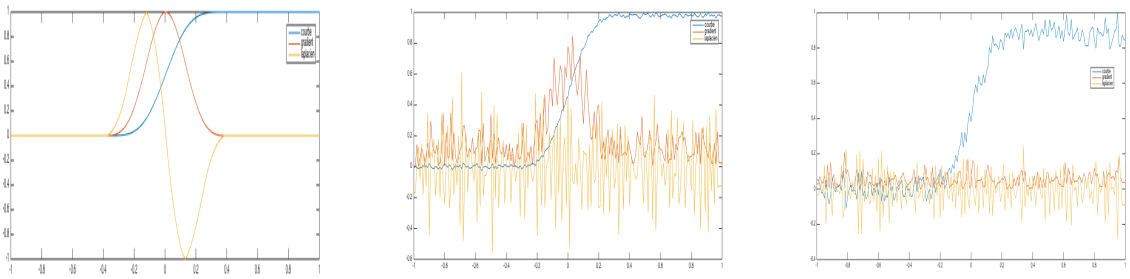

FIGURE 4 – Détections par opérateurs différentiels : en bleu, l'intensité, en rouge, le gradient, en jaune le laplacien. A gauche : sans bruit, au milieu : bruit faible, à droite : bruit fort

2.1 Opérateurs différentiels du premier ordre

2.1.1 Gradient discret

Plusieurs approximations discrètes permettent de calculer le gradient d'une image. Elles sont basées sur les développements limités :

$f(x+h) = f(x) + hf'(x) \quad et \quad f(x-h) = f(x) - hf'(x)$

d'ou $\frac{\partial f}{\partial x}(x) = \frac{(f(x+h)-f(x))}{h}$ *ou* $\frac{\partial f}{\partial x}(x) = \frac{(f(x+h)-f(x-h))}{2.h}$

Implantés sous forme de masques, ces approximations deviennent des convolutions avec des masques [-1 1] ou [-1 0 1] pour la direction x et leurs transposées pour la direction y.

Notons que les opérations successives de lissage et de gradient par convolution peuvent être obtenues par une seule opération de convolution du fait de la propriété des convolutions : $Lissage*(Gradient*I) = (Lissage * Gradient) * I$.

Par exemple, les gradients de Sobel et de Prewitt sont obtenus de la manière suivante :

	lissage	*	gradient	=	masque M1
Prewitt	$\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$	*	$\begin{bmatrix}0&0&0\\-1&0&1\\0&0&0\end{bmatrix}$	=	$\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}$
Sobel	$\begin{bmatrix}1&1&1\\1&2&1\\1&1&1\end{bmatrix}$	*	$\begin{bmatrix}0&0&0\\-1&0&1\\0&0&0\end{bmatrix}$	=	$\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix}$

En 2D, deux masques M_1 et $M_2 = M_1^t$ permettant le calcul des deux composantes G_x et G_y :

$$G_x(i,j) = M_1 * I(i,j) \quad \text{et} \quad G_y(i,j) = M_2 * I(i,j)$$

Le module et l'orientation de ce vecteur gradient $\vec{G}$ sont alors obtenus par :

$$\left\|\vec{G}\right\| = \sqrt{(G_x^2 + G_y^2)} \quad \text{et} \quad \phi = arctg\left(\frac{G_y}{G_x}\right)$$

2.1.2 Extraction des contours à partir du gradient

Pour extraire les contours, il faut ensuite prendre les pixels dont le module de gradient est maximum localement. En pratique, il convient de supprimer les points qui ne sont pas des maxima locaux dans la direction du gradiant donné par les valeurs du vecteur $G_x(i,j)$ et $G_y(i,j)$.

Pour plus de précision, il faut interpoler les valeurs dans cette direction. Sans interpolation, on teste les valeurs des pixels dans la direction (0,45,90,135 degrés) la plus proche de l'orientation du gradient. Pour simplifier, dans ce TP, on pourra éventuellement utiliser un simple seuillage qui sélectionne les valeurs fortes du gradient.

2.1.3 Travail à réaliser

Implanter une des méthodes de calcul du gradient précédente ainsi que la détection par seulllage du module du gradient des points de contours.

2.2 Opérateurs différentiels du deuxième ordre

2.2.1 Laplacien discret

Le laplacien est l'opérateur de second ordre le plus fréquemment utilisé à cause de sa simplicité. Il peut être approximé de la même façon que le gradient par des convolutions discrètes entre l'image et un des masques suivant :

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 2 & -4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Comme il est tres sensible au bruit, il est souvent précédé par un lissage gaussien et devient le laplacien de gaussienne ou filtre LoG (Laplacian of Gaussian) :

$$\Delta G_\sigma(x,y) = \frac{1}{\pi\sigma^4}\left(\frac{x^2+y^2}{2\sigma^2} - 1\right) e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

2.2.2 DOG :Différence Of Gaussian

Le lissage d'une image suivi par un opérateur laplacien (laplacien de gaussienne) peut être approximé par la différence entre deux lissages $H_1(x,y)$ (lissage fort) et $H_2(x,y)$ (lissage faible) de la même image (figure 5). Cet opérateur est connu sous le nom de Difference Of Gaussian. Le rapport entre les 2 écarts types des filtres de lissage est généralement pris à 1,6. On peut implanter efficacement ce filtre avec la FFT.

2.2.3 Opérateurs laplacien, lissage et FFT

Le laplacien est un filtre passe haut, qui conserve donc les hautes fréquences en éliminant les composantes basses fréquences. En supposant que le bruit est une haute fréquence, les zones constantes des basses fréquences et les contours des fréquences moyennes, il est possible d'isoler les contours et de faire un filtrage simultanément dans le plan fréquentiel en utilisant un filtre passe bande.

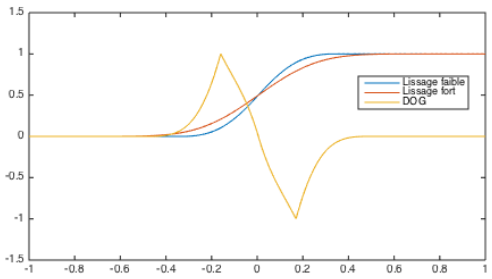

FIGURE 5 – Détections par DOG : en bleu, lissage faible, en rouge, en jaune le laplacien

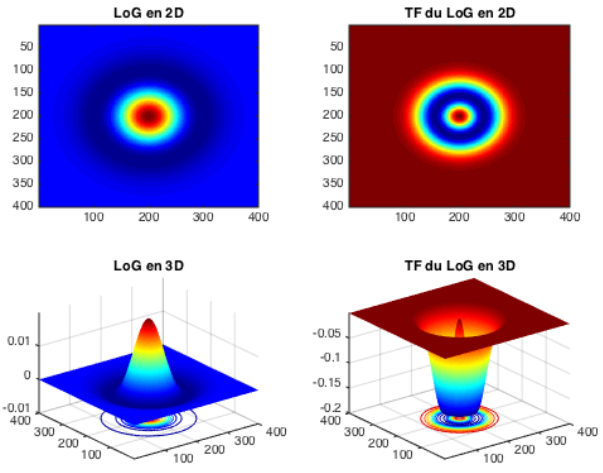

FIGURE 6 – Filtre LOG et sa FFT

En posant $r = \sqrt{x^2 + y^2}$, $\Delta G_\sigma(r) = \frac{1}{\pi\sigma^4}\left(\frac{r^2}{2\sigma^2} - 1\right) e^{-\frac{r^2}{2\sigma^2}}$

et la FFT du LOG est $\mathcal{F}(\Delta G_\sigma)(u,v) = -4\pi^2(u^2+v^2)\mathcal{F}(G_\sigma) = -4\pi^2(u^2+v^2)e^{-2\pi^2\sigma^2(u^2+v^2)}$

On constate effectivement que dans le domaine fréquentiel, le LoG est non nul sur une couronne (figure 6 à droite)

Comme dans le cas du lissage, il suffit donc de multiplier la FFT de l'image par un filtre passe bande, soit idéal (valeur 0 ou 1) soit sous la forme $\mathcal{F}(\Delta G_\sigma)(u,v)$.

Dans ce TP, le filtre idéal est une couronne de valeur 1 comprise entre les rayons R1 et R2, de valeur nulle ailleurs. La FFT inverse du produit filtre idéal par la FFT de l'image nous donne donc l'image du laplacien. Attention, ces images de laplacien contiennent des valeurs négatives.

2.2.4 Extraction des contours

Pour extraire les contours après avoir calculer le laplacien, il faut ensuite détecter les passages par zéro du laplacien. Du fait de la discrétisation, la valeur nulle n'est presque jamais présente et il faut donc détecter les passages du positif au négatif (et réciproquement) entre 2 points d'un voisinage 2×2 : on teste le passage par zéro entre les pixels de coordonnées i, j et $i+1, j$ et le passage par zero entre les

pixels de coordonnées i, j et $i, j+1$.

2.2.5 Travail à réaliser

Implanter une détection de contour soit par convolution avec un masque soit par DOG soit par FFT ainsi que l'extraction de contours par passage par zéro.

2.3 Comparaison

Comparer les détections/Extraction de contours par gradient et par laplacien d'un point de vue qualitatif. Quelles sont les propriétés de chacune des méthodes.

3 Principales fonctions d'entrées sorties, de création et conversion

3.1 Principales fonctions d'entrées sorties, de création et conversion

— Lecture d'image 8bits dans un fichier pgm : unsigned char ** lectureimagepgm(char *name, int* prows, int* pcols)
Lecture dans le fichier "filename" de format pgm d'une image en niveau de gris de *prows lignes et *pcols colonnes. Le nombre de ligne et de colonnes est lu dans le fichier "filename". L'image est créée dynamiquement, les pixels remplis puis l'image est retournée. Retourne NULL en cas d'echec

— Sauvegarde d'image 8 bits dans un fichier pgm : void ecritureimagepgm(char *name, unsigned char **im, int rows, int cols)
Sauvegarde dans le fichier "filename" de format pgm d'une image en niveau de gris de rows lignes et cols colonnes. Les nombres de lignes et colonnes doivent être connus. L'image est créée dynamiquement, les pixels remplis puis l'image retournée.

— Lecture d'image 8bits dans un fichier pgm : double** lectureimagedoubleraw(char *filename, int rows, int cols)
Lecture dans le fichier "filename" de format raw d'une image de réels de rows lignes et cols colonnes. Le nombre de ligne et de colonnes est lu dans le fichier "filename". L'image est cree dynamiquement, les pixels remplis puis l'image est retournée. Retourne NULL en cas d'echec

— Sauvegarde d'image de réels dans un fichier raw : void ecritureimagedoubleraw(char *filename, double **im, int rows, int cols)
Sauvegarde dans le fichier "filename" de format raw (ie seulement les données images, pas d'entete) d'une image de réels double de rows lignes et cols colonnes.

— Creation dynamique d'image d'octets :unsigned char **alloue_image(int nl, int nc)
Allocation dynamique d'un tableau 2D d'octets de nl lignes et nc colonnes. Retourne l'image ainsi créée. La zone de données image est allouée en une seule allocation ce qui permet d'utiliser indifféremment les notations im[i][j] avec 0<=i<n let 0<=j<nc ou (*im)[k] avec 0<=k<nl*nc pour parcourir l'image.

— libération d'image :void libere_image(unsigned char** im)
libere la mémoire allouée pour l'image im.

— Creation dynamique d'image de réels double précision : double ** alloue_image_double(int nl, int nc)
idem précédemment mais pour une image contenant des réels double précision

— Conversion d'image d'octets en image de réels double précision : double** imuchar2double(unsigned char **im, int nl, int nc)
Crée une image de double et recopie l'image d'octets

— Conversion d'image de réels double précision en image d'octets :unsigned char**imdouble2uchar(double** im,int nl, int nc)
Crée une image d'octets et recopie l'image de réels SANS mise à l'echelle. Attention aux dépassements de capacité des octets.
— Recopie d'une partie d'image : unsigned char** crop(unsigned char **im, int oi, int oj, int fi, int fj)
Recopie la partie comprise entre les indices (oi,oj) et (fi,fj) dans une nouvelle image allouée dynamiquement. Retourne cette nouvelle image.

3.2 Principales fonctions liées à la FFT

— Puissance de 2 :int nextpow2(double num)
Détermine la puissance de 2 juste supérieure ou égale à num
— Puissance de 2 : int ispowerof2(double num)
vrai si num est une puissance de 2
— Transformée de fourrier rapide et inverse : int fft(double** ims_reel, double** ims_imag, double** imd_reel, double** imd_imag , int dimx, int dimy)
Calcul la FFT de l'image complexe (partie réelle ims_reel et imaginaire ims_imag). Le résultat complexe est donné par les images partie réelle imd_reel et imaginaire imd_imag Les dimensions dimx et dimy doivent être des puissances de 2.
— Transformée de fourrier rapide et inverse : int ifft(double** ims_reel, double** ims_imag, double** imd_reel, double** imd_imag , int dimx, int dimy)
idem, mais transformée inverse
— Echange des quadrants de la fft : void fftshift(double** imsr, double** imsi, double** imdr, double** imdi, int nl, int nc)
Centre la transformée de fourrier au milieu de l'image
— Cree une image de reels dont les dimensions sont des puissances de 2 : double** padimdforfft(double** im, int* pnl, int* pnc)
Si les dimensions *pnl, *pnc ne sont pas des puissances de 2, crée une image dynamique de taille 2k, copie l'image initiale de réels double précision et complète par des zéros. Change les valeurs *pnl, *pnc. Retourne la nouvelle image créée.
— Cree une image d'entiers dont les dimensions sont des puissances de 2 : double** padimucforfft(unsigned char** im, int* pnl, int* pnc)
Idem, mais pour une image d'octets.

3.3 Fonctions utilitaires

— PSNR :double psnr(unsigned char **im1, unsigned char **im2, int nl, int nc) ;
Détermine le PSNR de deux images de type unsigned char
— PSNR :double psnr(double **im1, unsigned char **im2, int nl, int nc) ;
Détermine le PSNR de deux images de type double
— EQM : double eqm(unsigned char **im1, unsigned char **im2, int nl, int nc) ;
Détermine l'ecart quadratique moyen entre deux images de type unsigned char
— Norme :double** norme(double** real, double** imag, int nl, int nc)
Détermine le module d'une image de complexes
— Phase :double** phase(double** real, double** imag, int nl, int nc)
Détermine la phase d'une image de complexes
— Hamming :double** hamming_double(double** im, double** res,int nl, int nc)
Détermine le produit de l'image de double avec une fenêtre de Hamming. Si le paramètre res est NULL, crée une nouvelle image.
— Hamming :double** hamming_uc(unsigned char ** im, double** res,int nl, int nc)
Détermine le produit de l'image de double avec une fenêtre de Hamming.

3.4 Exemple de code d'ES

```
#include "pgm.h"
        /*
                Calcul de l'inverse video d'une image.
                Si le parametre sortie est NULL, on cree une nouvelle image.
                On parcourt toute l'image ligen par ligne,
                colonne par colonne, on calcule son inverse video
                et on retourne l'image ainsi modifiee
        */
unsigned char** inverse( unsigned char** sortie,  unsigned char** entree, int nl, int nc) { int i,j;
  if (sortie==NULL) sortie=alloue_image(nl,nc);
  for(i=0; i<nl; i++)
    for(j=0; j<nc; j++)
      sortie[i][j]=255-entree[i][j];
  return sortie;
}

        /*
                Exemple de code avec Entrees Sortie et transformations d'images
                S'utilise sous la forme  "exemple tangram.pgm res.pgm"
        */
main (int ac, char **av) {
        /* av[1] contient le nom de l'image, av[2] le nom du resultat . */
  int nb,nl,nc, oldnl,oldnc;
  unsigned char **im2=NULL,** im1=NULL;

  if (ac < 2) {printf("Usage : %s entree sortie \n",av[0]); exit(1); }
        // Lecture d'une image pgm dont le nom est passe sur la ligne de commande
  im1=lectureimagepgm(av[1],&nl,&nc);
  if (im1==NULL)  { puts("Lecture image impossible"); exit(1); }
        /* Calcul de son inverse video */
  im2=inverse(NULL,im1,nl,nc);
        /* Sauvegarde dans un fichier dont le nom est passe sur la ligne de
           commande */
  ecritureimagepgm(av[2],im2,nl,nc);
}
```

3.5 Exemple de code avec FFT

```
#include "pgm.h"
        /*
                Exemple de code avec FFT
                S'utilise sous la forme  "exemple tangram.pgm res.pgm"
        */
main (int ac, char **av) {
        /* av[1] contient le nom de l'image, av[2] le nom du resultat . */
  int nb,nl,nc, oldnl,oldnc;
  unsigned char **im2=NULL,** im1=NULL;
  double** im4,** im5, ** im6, ** im7, **im8, **im9,**im10;

  if (ac < 3) {printf("Usage : %s entree sortie \n",av[0]); exit(1); }
        /* Lecture d'une image pgm dont le nom est passe sur la ligne de commande
           */
  im1=lectureimagepgm(av[1],&nl,&nc);
  if (im1==NULL)  { puts("Lecture image impossible"); exit(1); }
        /* Passage en reels  */
```

```
17    double**im3=imuchar2double(im1,nl,nc);
18    oldnl=nl; oldnc=nc;
19          /*  la fft demande des puissances de 2. On padde avec des 0, mais les
                dimensions nl et nc changent */
20    im7=padimdforfft(im3,&nl,&nc);
21    /*    On peut aussi directement utiliser
22          im7=padimucforfft(im1,&nl,&nc);
23          sans convertir im1 en image de reels
24    */
25          // Creation des images pour les parties reelles et imagianires des fft
26    im4=alloue_image_double(nl,nc);
27    im5=alloue_image_double(nl,nc); i
28    im6=alloue_image_double(nl,nc);
29          /* Clacul de la fft de im7,im4 */
30    fft(im7,im4,im5,im6,nl,nc);
31          /* Creation des  parties reelles et imagianires des fft inverses */
32    im9=alloue_image_double(nl,nc);
33    im10=alloue_image_double(nl,nc);
34          /* Clacul de la fft inverse de im5,im6 */
35    ifft(im5,im6,im9,im10,nl,nc);
36          /* Conversion en entier8bits de la partie reelle de la fftinverse,
37             Suppresion des 0 qui ont servi a completer avec la fonction crop
38             Sauvegarde au format pgm de cette image  identique a 'linverse video
39             car on a realise la suite fftinv(fft(image))*/
40    ecritureimagepgm(av[2],crop(imdouble2uchar(im9,nl,nc),0,0,oldnl,oldnc),oldnl,
          oldnc);
41  }
```