TP n°2 : Interpolation et transformations géométriques

1 Installation : rectificatifs

Les instructions qui suivent corrigent la partie "Installation" du document "Utilisation de MegaWave". Ouvrez une fenêtre de type terminal, tapez

```
../mw/mw2_install
```

puis fermez la fenêtre. Cette commande se contente de rajouter une ligne dans votre fichier `.bashrc`. Votre compte est maintenant configuré pour utiliser MegaWave2. Vérifiez ceci en tapant (dans une nouvelle fenêtre terminal)

```
cview lena.img &
```

Cette commande doit afficher une image.

Cette installation ne se fait qu'une seule fois, désormais vous aurez accès aux commandes MegaWave2 à chaque fois que vous utiliserez l'un des PC de cette salle.

2 Mise en route

Contrairement à ce qui était précisé dans le document intitulé "Utilisation de MegaWave", la configuration du nouveau système installé sur ces PC n'autorise pas à changer votre shell par défaut. Cependant, nous continuerons à utiliser le shell `tcsh` pour les séances de TP. Il faudra donc commencer par taper la commande

```
tcsh
```

après l'ouverture d'un nouveau terminal où vous comptez entrer les commandes mentionnées dans le TP. Si vous ne suivez pas cette recommandation, des commandes comme `set`, `while`, `foreach`, etc. ne fonctionneront pas ou ne donneront pas les résultats escomptés.

En raison de l'installation très récente du système, la commande `tcsh` est parfois instable et se termine inopinément : typiquement, un message `Segmentation Fault` s'affiche et l'invite de commande (ce qui apparaît en début de ligne avant l'entrée d'une commande) change, vous indiquant que vous êtes revenu au shell par défaut. Il vous suffit alors de taper à nouveau `tcsh` pour rétablir la situation.

Je vous rapelle la marche à suivre pour le TP (voir "Méthode préconisée pour mener les expériences" dans "Utilisation de MegaWave") : copiez le script associé au TP dans votre répertoire principal puis éditez-le grâce (par exemple) aux commandes

```
cp ../moisan/TP/tp2 .
emacs tp2 &
```

Ensuite, allez dans le répertoire tmp grâce à la commande `cd tmp` (si ce répertoire n'existe pas, ce que vous pouvez vérifier en tapant `ls`, alors vous pouvez le créer en tapant `mkdir tmp`). Vous pouvez maintenant modifier, dupliquer, etc. les commandes du fichier tp2 et les copier-coller dans la fenêtre du terminal pour les exécuter.

Comme pour la séance précédente, vous aurez à tester un certain nombre d'algorithmes sur des images. Je vous rapelle que pour copier une image dans le fichier `src` (utilisé pour les commandes), il vous suffit d'entrer la commande

```
ccopy lena.img src
```

(ici l'image choisie est `lena.img`). La liste des images disponibles d'obtient par

```
ls $MEGAWAVE2/data/tp2
```

et vous pouvez visualiser n'importe laquelle de ces images par

```
cview <nom de l'image> &
```

Si vous avez besoin de connaître la taille d'une image, utilisez la commande

```
fsize <nom de l'image>
```

3 Transformations géométriques sans perte d'information

Dans cette partie, nous allons comparer différents interpolateurs (sinc, bicubique de Keys, splines) dans leur aptitude à réaliser les transformations euclidiennes (translation, rotation, et zoom avant). Le cas particulier du zoom arrière (réduction) sera considéré plus tard car la perte d'information qu'il requiert nécessite une approche légèrement différente.

Dans les expériences qui vont suivre, vous serez amenés à mettre en évidence essentiellement quatre types de défauts liés à l'interpolation : la pixellisation, le flou, le *ringing* et l'*aliasing*.

3.1 Translation

Si la translation d'une image d'un vecteur entier ne pose pas de problème, en revanche la translation sub-pixellique est l'exemple typique d'un processus nécessitant une interpolation de l'image.

Pour des raisons axiomatiques, l'interpolation sinc discrète (interpolation à l'aide de la transformée de Fourier discrète) est le candidat idéal pour réaliser cette transformation.

Q1. Justifiez l'assertion suivante : toute famille d'opérateurs (T_z) $(z \in \mathbb{R}^2)$ agissant sur les images discrètes $M \times N$ périodiques, continue par rapport à z, vérifiant la propriété de semi-groupe $(T_{z+w} = T_z \circ T_w)$ et coïncidant avec les translations entières (ie $T_z u(x) = u(x+z)$ pour tout z de $\mathbb{Z}^2$) est obtenue par interpolation sinc **discrète. A-t-on unicité ?**

En choisissant successivement plusieurs images, testez la translation de vecteur $(1/2, 1/2)$ à l'aide de la commande

```
fftrot -x 0.5 -y 0.5 src trans
```

et comparez le résultat (`trans`) à l'original à l'aide de la commande `cview`.

Q2. Quelles différences observez-vous ? Voyez-vous un moyen d'y remédier partiellement ?

3.2 Rotation

La rotation d'une image est un transformation intéressante car elle met bien en évidence la capacité d'un interpolateur à préserver l'information contenue dans l'image. Commençons par "préparer" l'image source en la limitant à un disque

```
fmaskrot src src_disc
```

puis faisons lui subir 36 rotations de 10 degrés

```
cp src_disc mov_0
set n = 0
while ($n < 35)
set old = $n
@ n ++
fzrt -o 0 mov_$old mov_$n 1 10 0 0
end
mkcmovie mov 0 35
cmview -l mov &
```

Le résultat est catastrophique mais ce n'est pas très étonnant compte tenu du fait que nous avons effecté la rotation au moyen d'une interpolation au plus proche voisin (option `-o 0` de `fzrt`). Nous pouvons maintenant faire varier la méthode d'interpolation en changeant cette valeur ($o = 1, -3, 3, 5$, taper `fzrt` seul pour obtenir la signification de ces nombres). Pour générer directement le résultat après 36 rotations (sans le film), vous pouvez utiliser les commandes

```
cp src_disc rotated
repeat 36 fzrt -o 0 rotated rotated 1 10 0 0
cview rotated &
```

Vous pouvez aussi bien sûr changer le nombre de rotations et l'angle correspondant.

Q3. Comparez et commentez les résultats obtenus pour chaque méthode d'interpolation ($o = -3, 1, 3, 5$).

A priori, une rotation par interpolation de Fourier demanderait $O(N^3 \log N)$ opérations, puisqu'il faudrait calculer pour chaque point de la nouvelle grille une valeur obtenue par transformée de Fourier inverse. Il est assez surprenant que l'on puisse en fait réaliser cette opération en $O(N^3 \log N)$ opérations, en décomposant la rotation en produit de "glissements" : c'est la méthode de Yaroslavsky. Testez cette méthode en remplaçant le précédent `repeat ...` par

```
repeat 36 fftrot -a 10 rotated rotated
```

et observez le résultats obtenu.

Q4. Que concluez-vous ? Pourquoi n'y a-t-il (quasiment) aucune perte d'information ? Quel est le prix à payer pour ce résultat ?

Indication : vous pouvez faire afficher le temps de calcul lors de l'utilisation d'une commande en la faisant précéder de `time` (exemple: `time repeat 36 ...`); le premier nombre affiché (suivi d'un u minuscule) est alors le temps consacré au calcul (en secondes).

3.3 Zoom

S'il est clair que l'interpolation de Fourier (interpolation sinc discrète) est d'une efficacité remarquable pour la translation et la rotation, en revanche la situation est plus discutable pour le zoom avant (agrandissement). En effet, comme nous avons pu l'expérimenter dans le TP précédent, le zoom par *zero-padding* (mise à zéro des fréquences manquantes) introduit en général un effet de Gibbs désagréable. Prenez ue image de petite taille (256x256) et comparez le zoom x4 par zero padding

```
fftzoom -z 4 src zoom1
```

avec un zoom par interpolation spatiale

```
cview -z 4 -o ... src &
```

avec différents ordres d'interpolation ($o = 0, 1, -3, 3, 5, 7, 9$). Vous pouvez aussi utiliser la commande `llview` (que nous emploierons plus spécifiquement dans une séance ultérieure) en tapant

```
llview -d 0 src
```

puis en utilisant le bouton de droite dans la fenêtre pour zoomer (taper 'h' pour une liste complète des commandes) et en cliquant dans la fenêtre de commandes pour changer la méthode d'interpolation.

Q5. Comparez les différentes méthodes de zoom sur quelques images, et indiquez à chaque fois (en précisant le nom de l'image) la méthode qui vous semble réaliser le meilleur compromis. Quels sont les défauts mis en concurrence ?

4 Comportement fréquentiel

Pour visualiser l'atténuation fréquentielle réalisée par l'interpolation, nous pouvons procéder comme suit :

1. générer une image à spectre plat et à phases aléatoires, soit u telle que $\hat{u}(\xi) = e^{i\varphi(\xi)}$ où les $\varphi(\xi)$ sont une réalisation d'un champ de variables aléatoires indépendantes (à la symétrisation près qui assure que u est réelle) uniformément distribuées sur $[0, 2\pi[$.

   ```
   fconst u 1 512 512
   fft2d -A u u
   frandphase u u
   ```

2. translater cette image de 0.5 pixel en x (par exemple) selon la méthode d'interpolation choisie

   ```
   fzrt -o 5 u v 1 0 0.5 0
   ```

3. visualiser son spectre radial, c'est-à-dire la fonction

   $$S(r) = \text{moyenne}_{|\xi|=r} |\hat{u}(\xi)|.$$

   ```
   fft2drad -l v s
   splot s &
   ```

Q6. Appliquez cette opération pour les différents interpolateurs rencontrés (plus proche voisin, bilinéaire, bicubique Keys, splines, sinc **discret) et expliquez les différences obtenues.**

5 Réduction d'une image

La réduction d'une image ne peut se faire correctement par sous-échantillonnage direct car apparaît en général un phénomène d'*aliasing* dû au repliement spectral des fréquences qui ne peuvent plus être exprimées compte tenu de la résolution de la nouvelle image. Une coupure fréquentielle préalable est aussi délicate car si elle est réalisée brutalement, il y a apparition du phénomène de Gibbs. Nous avons déjà constaté ces deux phénomènes lors de la dernière séance, mais vous pouvez les reproduire simplement en choisissant un image de grande taille (minimum 512x512) et en la dézoomant d'un facteur 4 au moyen de

```
fftzoom -i -z 4 src reduced1
fsample src reduced2 4
```

Une possibilité est alors d'effectuer une coupure fréquentielle plus douce en convoluant l'image de départ avec une Gaussienne

```
fsepconvol -g 1 src tmp
fsample tmp reduced3 4
```

ou avec une prolate

```
prolate 5 0.25 kernel
fconvol src kernel tmp
fsample tmp reduced4 4
```

avant d'effectuer l'échantillonnage. Cependant, ces opérations ne sont pas des projections : si l'image de départ est déjà sous-échantillonable (i.e. si son spectre est déjà nul pour toutes les fréquences qui seront perdues lors du changement de résolution), alors alors il y aura tout de même une perte d'information lors de la convolution (Gaussienne ou prolate). Ceci n'était pas le cas avec un zoom Fourier inverse par coupure fréquentielle (`fftzoom`), qui, en revanche, est bien la projection L^2 associée à l'interpolation sinc.

L'échantillonnage par splines tombe à point pour trancher ce dilemme : plutôt que de projeter dans l'espace des fonctions sinc, nous pouvons projeter dans une base de splines d'ordre donné (n). L'image réduite obtenue sera alors celle qui minimisera la norme L^2 entre l'interpolée de l'image originale et l'interpolée de l'image réduite, les deux interpolations étant entendues au sens des splines d'ordre n.

Cette opération est réalisée au moyen de la commande `funzoom` :

```
funzoom -z 4 -o 0 src reduced5
```

A noter que pour l'ordre 0, la projection revient à intégrer l'image de départ (interpolée à l'ordre 0) sur une matrice de pixels carrés jointifs. Ce dispositif reproduit donc assez fidèlement le fonctionnement d'une matrice CCD formé de capteurs jointifs.

Q7. Commentez l'influence de l'ordre de la base de splines utilisée (option `-o`, de 0 à 5) sur le résultat. Quel vous semble être le choix le plus adapté en fonction de l'image utilisée ?

Q8. A la lumière du cours et des expériences réalisées sur l'échantillonnage, expliquez en quelques phrases l'intérêt et les limites du théorème de l'échantillonnage pour les images numériques.