Romulus-Mircea TEREBEȘ

TECHNIQUES AVANCEES POUR LE TRAITEMENT DES IMAGES ET DE LA VIDEO

Travaux pratiques

UTPRESS
Cluj-Napoca, 2019
ISBN 978-606-737-386-8

Romulus-Mircea TEREBEȘ

Techniques avancées pour le traitement des images et de la vidéo

Travaux pratiques

Editura UTPRESS
Cluj-Napoca, 2019
ISBN 978-606-737-386-8

Editura U.T.PRESS
Str. Observatorului nr. 34
C.P. 42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999
e-mail: utpress@biblio.utcluj.ro
http://biblioteca.utcluj.ro/editura

Director: Ing. Călin D. Câmpean

Recenzia: Prof.dr.ing. Monica Borda
Conf.dr.ing. Raul Măluţan

ISBN 978-606-737-386-8

Table des matières

TP1 Description de la plateforme de laboratoire 2

TP2 : Filtres non linéaires dans le domaine spatial..... 11

TP3 Equations aux dérivées partielles pour la restauration et amélioration des images 23

TP4 Diffusion tensorielle..... 39

TP5 La théorie de déformation des courbes fermées. La méthode fast marching:

applications pour la segmentation des images 46

TP6 Techniques d'inpainting..... 63

TP7 Débruitage par patchs. Filtres à moyennes non locales. Filtrage collaboratif 65

TP1 Description de la plateforme de laboratoire

Objectifs du TP :

- familiarisation avec la plateforme N'D dédiée au traitement d'image, du signal et des séquences vidéo ;
- implantation des quelques opérateurs classiques sous N'D.

1. Le logiciel N'D

N'D (voir Fig. 1.1) est une interface logicielle, de visualisation et de traitement d'images (2D/3D) ou de vidéos (flux enregistrés ou temps-réel) sous Windows [1]. Elle prend en charge les formats de données les plus courants.

Pour avoir une bonne description d'utilitaire, nous citons l'auteur : "Il s'agit d'une interface au caractère WIMP (Window, Icon, Menu, Pointing device) très prononcé dans laquelle chaque action est déclenchée au moyen d'une sélection de fenêtre, d'un clic, d'un choix d'une entrée d'un menu, etc. Son principe de fonctionnement est « contextuel » : un traitement est exécuté à partir d'un menu situé dans la fenêtre de l'image ou de la vidéo concernée. Les différents outils disponibles, présentés dans la suite de l'article, procurent une forte interactivité."

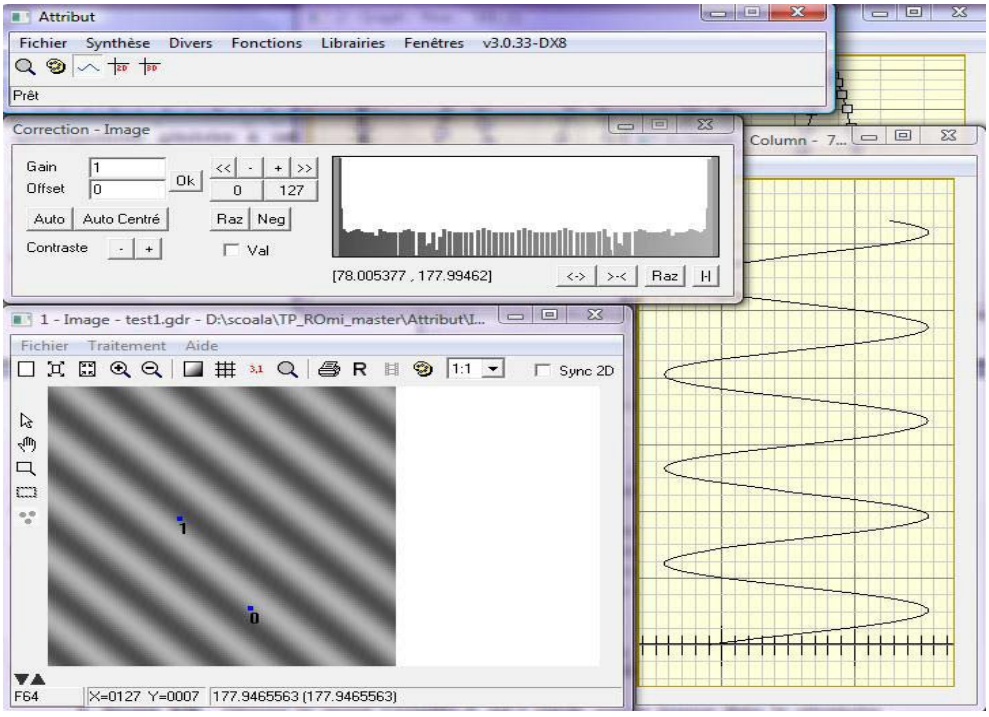


Fig. 1 Interface graphique de la plateforme N'D

N'D se compose d'un certain nombre d'outils interactifs à caractère fortement pédagogique. Il n'est naturellement pas possible dans cet TP de les décrire tous de manière exhaustive. A titre d'exemple, en dehors d'un outil de sélection de palette classique, on peut citer la fenêtre de correction d'intensité (voir figure 2) qui permet de régler précisément les paramètres d'une transformation linéaire appliquée à l'image ou de les contrôler en déplaçant des curseurs directement sur son histogramme. La loupe dans une forme numérique, adaptative ou encore 3D est un autre outil qui permet de se rendre compte de manière optimale du contenu d'une image. Enfin, un curseur 2D, allié à un déplacement synchronisé de la partie visualisée de plusieurs images, permet de confronter efficacement le résultat d'un algorithme de traitement à l'image d'origine.

Il est à noter que l'interface dispose, toujours de manière contextuelle, de traitements basiques, bien utiles, opérant sur une (valeur absolue, transformations géométriques, etc) ou plusieurs (addition, extrémum, etc) images.

N'D est modulaire. Il est possible de lui ajouter très simplement de nouveaux traitements en glissant-déplaçant, par exemple, des librairies (fichiers binaires d'extension .dll) spécifiquement générées à cet effet. Ceci permet à la fois de bénéficier de bibliothèques d'algorithmes développés par d'autres ou encore d'expérimenter ses propres algorithmes. Les modules intégrés ajoutent de nouvelles entrées dans les menus de l'interface principale ou des fenêtres spécifiques (images, vidéos, etc). Leur gestion est dynamique : les librairies sont chargeables et déchargeables à volonté sans nécessiter un arrêt de l'interface.

D'un point de vue de l'implémentation, il s'agit d'écrire, dans le langage de programmation C, le code d'une bibliothèque dynamique (d'extension .dll) qui peut être générée par un outil classique tel que Microsoft Visual Studio, Dev-Cpp, CodeBlocks, etc. Un canevas complet (fichiers source et de compilation) à compléter est fourni pour simplifier l'opération et comporte notamment un fichier qui liste toutes les fonctions reconnues par l'interface.

Le principe de fonctionnement de l'interface étant contextuel, l'ajout d'un nouveau traitement s'obtient par quelques lignes de code qui s'organisent en deux parties distinctes :

- la spécification d'une entrée supplémentaire dans le menu concerné (image, vidéo, etc)
- la définition d'une fonction exécutée lors de la sélection du traitement et qui met en œuvre l'algorithme proprement dit.

Dans la suite, nous énumérons les étapes nécessaires pour la création d'une nouvelle méthode très simple – l'ajout d'une constante aux valeurs d'une

image - dans un libraire dynamique.

1. **Ouverture du projet N'D**. Ouvrez la solution COMPILE_DLL.SLN qui se trouve dans le répertoire. ..\ND\COMPILATION. Comme simples utilisateurs d'un tel projet, nous sommes intéressés seulement aux fichiers qui devront être modifiés pour ajouter de nouvelles utilités à une bibliothèque donnée. Ces fichiers sont :

- **attlibrarymenu.h** et **attlibraryMain.c** - on édite les menus contextuels
- **attLibraryRun.c** joue le rôle de main. Dans ce fichier on édite les fonctions qui seront appelées lorsque le menu est sélectionné.

Tous les autres fichiers ne doivent pas être modifiés. Parmi ces fichiers, un grand intérêt faut accorder au **callback.h**, où on retrouve toutes les fonctions prédéfinies d'utilitaire.

2. Création d'un nouveau menu

a) Ajout de la méthode dédiée de création d'un menu (**attLibraryMenu1**) dans la fonction *GetMenuParametres* placée dans le fichier **attlibraryMain.c**

```
EXPORT_LIB void GetMenuParametres(long * nombre_menu, long *
nombre_id){
    attLibraryInitDialog();
    attLibraryMenu();
            attLibraryMenu1(); //menu ajouté
    *nombre_menu = nb_menu;
    *nombre_id = nb_id+1;
}
```

b) Ajout d'un identificateur unique pour le menu désiré dans le fichier **attlibrarymenu.h.**

```
#define IDM_ADNC            20 // identificateur unique
```

c) Modification de la fonction attLibraryMenu1 du **attlibraryMenu.c**. Par exemple, pour créer un menu pour l'ajout d'une constante à une image, vous pourriez utiliser la séquence du code suivant:

```
void attLibraryMenu1()
        {
        ATT_DESC_MENU menu1 =
                    {
                    BEGIN_MENU
                    "Ajout constante",
                    IDM_ADNC,
                    END_MENU
                    };
                    attLibraryAddMenu("MENU_2D, Operateurs de base",
                    menu1);
        }
```

MENU_2D représente l'endroit précis de notre menu parmi les menus de N'D. Ici, on parle d'un traitement 2D qui sera exécuté depuis l'image active - l'image qui subira le traitement (voir la figure suivante).

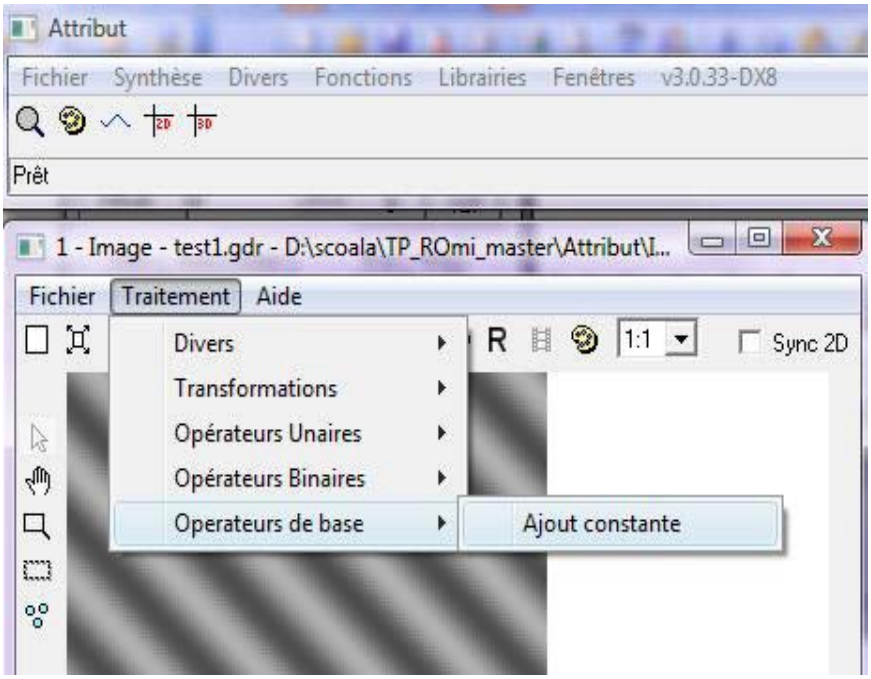


Fig. 2 L'emplacement du menu créé

Les autres menus sont des menus par default du N'D. En sélectionnant le menu « Ajout constante », la liaison entre le menu et son identificateur (ici IDM_ADNC) est automatiquement réalisée. Pour chaque identificateur, une routine associée – qui définie l'action de l'opérateur - doit être aussi décrite dans attLibraryRun.c.

3. Identificateur – Routine. Dans le fichier **attLibraryRun.c**, la routine associé au notre identificateur est ajouté dans la fonction attLibraryRun() qui joue le rôle de la fonction main d'une application en C.

```
void attLibraryRun(long numero_tache, long choix, long mode)
{
        switch (choix)
        {
                case    IDM_ADNC:
                        {
                        //…routine
                        }
                break;
         }
}
```

Pour chaque identificateur unique une routine doit être ajouté attLibraryRun().

4. Description de la nouvelle routine, à l'occurrence l'ajout d'une constante à une image. Pour appliquer un traitement à une image il faut tenir compte de quelques routines importantes fournies par le logiciel Attribut.

4.a. Chargement d'une image

```
id = attDataGetIdByCount(numero_tache, IMG_ENTREE, 1);
larg_entree = attDataGetLargeur(id);
haut_entree = attDataGetHauteur(id);
ent_f = (double *)attDataGetData(id, TYPE_DOUBLE);
```

id est l'identificateur de l'image à traiter. Dans les variables larg_entree et haut_entree on récupère les dimensions d'image, et le pointeur ent _f pointe vers la région en mémoire où les valeurs du niveau de gris pour chaque pixel de l'image d'entrée sont sauvegardées. La dimension de la zone réservée dans la mémoire est larg_entree * haut_entree. Pour le traitement d'une image couleur on a besoin d'un pointeur char à dimension 4*larg_entree* haut_entree, et du type TYPE_RGBA.

4.b Création d'une boîte à paramètres (paramètres à donner par l'utilisateur). Par exemple dans ce cas, l'utilisateur doit préciser la valeur de la constante à ajouter. La routine suivante fournit la boîte de paramètres de la figure 1.3 (la constante doit être définie comme un variable avant l'appel de la routine) :

```
long constanta;
    if ( !attBoiteParametre("Value const=%d", &constanta) ) break;
```

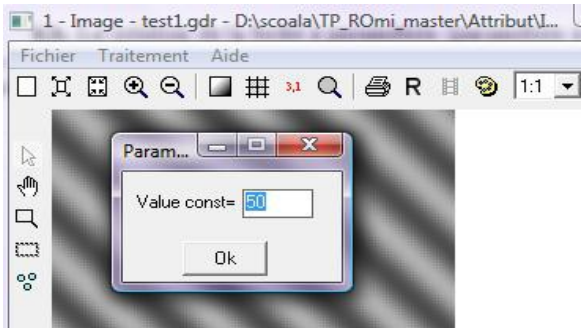


Fig. 3 Boîte à paramètres

4.c. Création, l'affichage de l'image traitée et la réservation d'une zone en mémoire.

```
attDataSetImageNombreTypeTaille(numero_tache,1,TYPE_DOUBLE,larg_entree,
haut_entree);
id = attDataGetIdByCount(numero_tache, IMG_SORTIE, 1);
sor_f = (double *)attDataGetPtr(id);
```

5. Programmation de la méthode effective. La méthode sera décrite dans un fichier de type .c et aura le prototype dans le .h. Par exemple, on appellera la méthode Ajout_const() qui aura les paramètres formelles : les pointeurs correspondants aux images d'entrée et de sortie, les dimensions d'image et les paramètres donnés par l'utilisateur :

```
Ajout_const(ent_f, sor_f, larg_entree, haut_entree, constanta);
```

Après la programmation correcte du code suivent les étapes :

6. Compilation du projet pour avoir la bibliothèque (dll).

7. Chargement si nécessaire de la librairie et des images (par drag-and-drop).

Exercices

1. En répétant les étapes décrites ci-dessus ajoutez à la libraire existante une nouvelle méthode ; par exemple la multiplication par une constante.
2. Ajoutez une nouvelle méthode qui s'applique à une image et qui remplace la valeur du pixel courant avec la valeur moyenne obtenue dans son voisinage. Le voisinage est de type carré d'une dimension impaire donnée par l'utilisateur.
3. Réalisez un filtrage linaire par un noyau Gaussien bidimensionnel en partant de la routine suivante :

```
void Convolution_Gauss(double * entree, double * sortie,long largeur,long hauteur,double Sigma, long id_sortie)
  {
      long taille=largeur*hauteur, i, j, k, taille_filtre, demi_taille;
      double pix1, pix2, somme=0.0, * interm, filtre[100];;
      taille_filtre=(int) (4 * Sigma)+1;
      if (taille_filtre %2 ==0) taille_filtre=taille_filtre+1;
      demi_taille=(int)(taille_filtre-1)/2;
      interm =(double *) calloc (taille, sizeof(double));
      // Calcul de coeficients du filtre
      for (i=0; i<=taille_filtre-1; i++)
      {
            filtre[i]=exp(-(i-demi_taille)*(i-
demi_taille)/(2*Sigma*Sigma))/sqrt(2*PI*Sigma*Sigma);
            somme=somme+filtre[i];
      }
      // Normalisation de coeficients du filtre//
      for (i=0; i<=taille_filtre-1; i++)
      {
```

```
            filtre[i]=filtre[i]/somme;
    }
    // filtrage suivant les colonnes
    for (i=demi_taille ;i<hauteur-demi_taille; i++)
            for (j=demi_taille ;j<largeur-demi_taille; j++)
            {
                    pix1=0.0;
                    for (k=i-demi_taille; k<=i+demi_taille; k++)
                    {
                            pix2=entree[k*largeur+j];
                            pix1=pix1+(filtre[k-i+demi_taille]*pix2);
                    }
                    interm[i*largeur+j]=(pix1);
            }
    // filtrage suivant les lignes
    for (i=demi_taille ;i<hauteur-demi_taille; i++)
            for (j=demi_taille ;j<largeur-demi_taille; j++)
            {
                    pix1=0.0;
                    for (k=j-demi_taille; k<=j+demi_taille; k++)
                    {
                            pix2=interm[i*largeur+k];
                            pix1=pix1+(filtre[k-j+demi_taille]*pix2);
                    }
                    sortie[i*largeur+j]=(pix1);
            }
    free(interm);
}
```

4. Traitez la même image ../Images/Lena_bruit_Gaussien.bmp par un filtre moyenneur et un filtre Gaussien de même taille (3x3 pixels et σ =0.5). Quelles sont les différences ? Augmentez la taille des filtres et notez les effets observables.
5. L'opération d'égalisation d'histogramme a pour but l'amélioration de la qualité d'une image. Elle opère sur l'histogramme de l'image traitée qui – pour une image de n x m pixels – est une fonction numérique qui associe à tout nuance de gris sa probabilité d'apparition :

   $p(l_k)=n_k/(m \times n)$

l_k – nuance de gris k

n_k – nombre des pixels ayant la valeur l_k

L'égalisation d'histogramme attribue a tout pixel une nouvelle valeur obtenue en modifiant l'histogramme selon une loi linéaire.

Un corps possible pour une fonction C qui implante cette transformation est le suivant :

```
void Egalisation_Histo (double * intrare, double *iesire, int latime, int inaltime)
{
    int i, j,k;
    int marime=latime*inaltime;
    float suma;
    double histo_ini[256];
    double histo_modif[256];

    for (i=0; i<latime;i++)
    for (j=0; j<inaltime;j++)
            histo_ini[(int)intrare[i+j*latime]]++;

    for (i=0; i<256;i++)
            histo_ini[i]=histo_ini[i]/marime;

    for (i=0; i<256;i++)
    {
            suma=0.0;
            for (k=0;k<i;k++)
            {
                    suma=suma+histo_ini[k];
            }
            histo_modif[i]=suma;
    }
    p1=&intrare[0];
    p2=&iesire[0];
    for (i=0; i<latime;i++)
    for (j=0; j<inaltime;j++)
            iesire[i+j*latime]=histo_modif[intrare[i+j*latime]];
}
```

Identifiez le rôle de chaque instruction en langage C en insérant des commentaires dans le code.

6. Ajoutez tous les éléments nécessaires qui permettront le traitement d'une image avec la fonction précédente.
7. Traitez l'image ..\Images\Lena_contraste_faible.bmp avec l'opérateur

d’égalisation et notez l’effet de l’opération.

Références bibliographiques

[1] M. Donias – „N'D, une interface logicielle pour découvrir et expérimenter le traitement des images et de la vidéo”, J3eA, vol 7, Hors Series 1, Special Edition CETSIS2007, 2007.
[2] Support de cours http://ares.utcluj.ro/tapisv

Remerciements

Je tiens à remercier mon ancien collègue, M. Sorin Pop pour les échanges fructueux que nous avons eus ensemble et pour sa contribution à l’élaboration de cette partie du volume.

TP2 : Filtres non linéaires dans le domaine spatial

Objectifs du TP :

- implanter les principaux types des filtres non linéaires pour la restauration et l'amélioration des images dans le domaine spatial ;
- analyser les performances de ces filtres pour différents types de bruit ;
- analyser les effets secondaires de ces filtres sur les images traitées.

2.1 Filtre moyenneurs non linéaires

2.1.1 Fondements théoriques

Les filtres moyenneurs non-linéaires sont des filtres passe-bas qui, pour chaque pixel, calculent la valeur filtrée en utilisant des versions non-linéaires de la moyenne d'une suite des échantillons. Plus précisément, pour un pixel de coordonnées (x,y) et pour un voisinage de taille donnée, symétrique par rapport à (x,y), la formule de calcul est la suivante :

$$u_{filtree}(x,y) = f^{-1}\left(\frac{\sum_{(x_i,y_j)\in W(x,y)} a_{i,j} f[u(x_i,y_j)]}{\sum_{(x_i,y_j)\in W(x,y)} a_{i,j}}\right) \quad (1)$$

Si les coefficients de pondération sont tous égaux à $u(x_i,y_j)^p$ et f(x)= x, la valeur filtrée est calculée en utilisant la formule :

$$u_{filtreeCH}(x,y) = \frac{\sum_{(x_i,y_j)\in W(x,y)} [u(x_i,y_j)]^{p+1}}{\sum_{(x_i,y_j)\in W(x,y)} [u(x_i,y_j)]^{p}} \quad (2)$$

Le filtre décrit par l'équation (2) est un filtre ***moyenneur contra-harmonique*** ; des autres types de filtre peuvent être obtenus en particularisant la fonction f (voir le support de cours).

2.1.2 Exercices

1. Ouvrez la plateforme Microsoft Visual Studio et ensuite ouvrez la solution …\Compile_DLL.sln.
2. Ajoutezle code suivant :

- dans le fichier **attLibraryMenu.c** :

```
void attLibraryMenu2()
{
        ATT_DESC_MENU menu1 =
                {
                        BEGIN_MENU
                        "Moyenneur contra-harmonique",
IDM_MOY_CONTRA,
                        END_MENU
                };
                attLibraryAddMenu("MENU_2D,Filtre non lineaires", menu1);
}
```

- dans le fichier **AttLibraryMenu.h** :

```
…
#define IDM_MOY_CONTRA          18
…
```

- dans le fichier **attLibraryMain.c** dans le corps de la fonction *GetMenuParametres*:

```
…
attLibraryMenu2();
…
```

Cette séquence ajoutera une barre de menu pour pouvoir accéder à la fonction qui implémentera le filtre.

3. Ajoutez un fichier nommé TP2.c au projet courant (Project->Add to project->New-> C++ Source File).
4. Cliquez sur le fichier qui vient d'être crée.
5. Ajoutez le code suivant dans le fichier TP2.c :

```
void Moyenneur_Contraharmonique(double * entree, double * sortie, long largeur,
long hauteur, long taille, double p)
{
// entrée –image d'entrée
// sortie – image de sortie
// largeur, hauteur – dimensions de l'image
```

```
// taille du voisinage
// p paramètre
long i,j;
long u,v;
// pour chaque pixel de l'image d'entree
        for (i=taille/2; i<largeur-taille/2; i++)
        for (j=taille/2; j<hauteur-taille/2; j++){// parcourir le voisinage
                for (u=-taille/2;u<=taille/2;u++)
                for (v=-taille/2;v<=taille/2;v++){
                }
         }
}
```

6. Modifiez le code afin que la fonction implémente le filtre moyenneur contra-harmonique.
7. La fonction peut être appelée en rajoutant le code suivant dans le fichier **attLibraryRun.c** :

```
 case    IDM_MOY_CONTRA:
{
     long larg_entree, haut_entree, id;
     double * ent_f, * sor_f;
     static double p = 2.0;
     static taille=5 ;
     if ( mode == MODE_AIDE )
         {attMsg("Remplace la valeur du pixel avec la moyenne contraharmonique
des pixels voisins");              break;}
     if ( !attDataSetNombre(numero_tache, IMG_ENTREE, 1) ) break;

     if ( !attBoiteParametre("Taille du filtre=%d p=%f", &taille, &p) ) break;
     id = attDataGetIdByCount(numero_tache, IMG_ENTREE, 1);
    larg_entree = attDataGetLargeur(id);
    haut_entree = attDataGetHauteur(id);
    ent_f = (double *)attDataGetData(id, TYPE_DOUBLE);

   if ( ent_f == NULL ) break;
```

```
    attDataSetImageNombreTypeTaille(numero_tache, 1, TYPE_DOUBLE, larg_entree, haut_entree);
    id = attDataGetIdByCount(numero_tache, IMG_SORTIE, 1);
    sor_f = (double *)attDataGetPtr(id);
    Moyenneur_Contraharmonique(ent_f, sor_f, larg_entree, haut_entree, taille,p);
    attDataFree(ent_f);
    }
    break;
```

La partie représente en bleu est la façon standard en N'D d'ouvrir une image, de définir une image de sortie et de les afficher sur l'écran.

8. Compilez la bibliothèque.
9. Lancez l'exécutable et chargez la bibliothèque crée avant.
10. Ouvrez une image placée dans le répertoire ..\ImagesStandard. Filtrez l'image en utilisant des tailles du filtre de plus en plus grandes. Quel est l'effet de la taille du filtre sur les contours de l'image?
11. Ouvrez l'image maison_sel.bmp placée dans le répertoire ..\ImagesStandard. Traitez l'image avec le filtre en utilisant des paramètres *p* négatifs de plus en plus petits. Notez les effets que vous observez et expliquer les.
12. Ouvrez l'image maison_poivre.bmp placée dans le directoire le répertoire ..\ImagesStandard. Traitez l'image avec le filtre en utilisant des paramètres p négatifs de plus en plus petits. Notez les effets que vous observez et expliquer les.
13. Ouvrez l'image maison_sel_et_poivre.bmp et essayez de le traiter avec ce type de filtre. Est-ce que c'est possible d'éliminer ce type de bruit ?
14. Comparez les résultats avec celles obtenue par un filtre moyenneur classique de même taille

2.2 Filtre médians scalaires

2.2.1 Fondements théoriques

Le filtre médian est un filtre non- linéaire qui remplace la valeur du pixel courant non par la moyenne mais par la médiane de la suite des échantillons (la valeur placée au milieu de la suite des échantillons triée par ordre croissant) défini par le voisinage du pixel. Des poids peuvent être associées à chaque pixel. Soit w_1, ..., w_N une suite des poids associées à une suite des valeurs x_1, ..., x_N . Le filtre médian pondéré est défini par la formule:

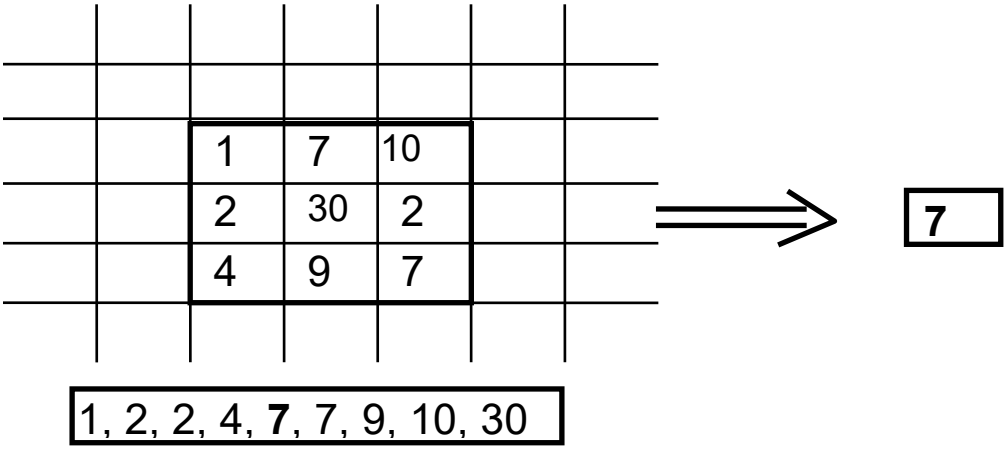
$$mediane_{ponderee}(x_1, x_2 ... x_N) = mediane(W_1 \Diamond x_1, W_2 \Diamond x_2, ... W_N \Diamond x_N) \qquad (3)$$

◊ - dénote l'opérateur de répétition

Remarques (voir le support de cours) :

- si tous les poids sont positifs l'équation (3) décrit un filtre passe-bas ;
- si N est paire la médiane d'une suite des échantillons est définie comme la moyenne de la valeur placée au milieu et la valeur suivante de la suite triée par ordre croissant ;
- si N est impaire ou si la somme des poids est impaire le filtre médian ne va pas créer des niveaux de gris ;
- en jouant sur les signes des poids le filtre peut agir comme un filtre passe bande ou passe haut ;
- le signe d'un poids commute avec la valeur de l'échantillon : (-5)◊6=5◊(-6)= -6-6-6-6-6.

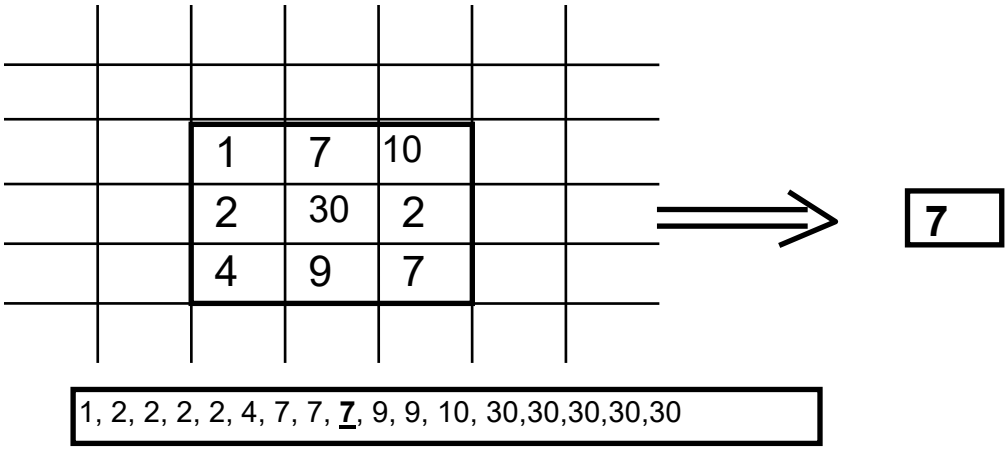
Exemple 2.2.1 Filtre médian classique [3x3] :



Exemple 2.2.2 Filtre médian pondérée [3x3]

Soit la matrice des poids :

$W = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 1 \end{bmatrix}$. Le résultat du filtre est:



2.2.3 Exercices

1. Procédez de manière similaire à 2.1.2 pour créer une fonction qui implante un filtre médian non pondéré de taille variable.

2. Ouvrez l'image sel_et_poivre.bmp et traitez la avec le filtre crée en prenant des tailles des plus en plus grandes. Notez les effets du filtre.

3. Ouvrez l'image sel_et_poivre.bmp et traitez la avec le filtre de manière itérative en prenant toujours la même taille. Comparez les résultats avec ceux obtenus avant.

4. Procédez de manière similaire à 2.1.2 pour créer une fonction qui implante un filtre médian de taille variable et qui associe un poids plus important au pixel central.

5. Procédez de manière similaire à 2.1.2 pour créer une fonction qui implante un filtre médian avec les poids : -1,-1,-1,-1,8,-1,-1,-1,-1.

6. Ouvrez une image dans le répertoire ..\ImagesStandard. Traitez l'image avec le filtre crée. Quel est le comportement du filtre ? Expliquez-le. Effectuer la somme entre l'image filtrée et l'image originale ? Quel est l'effet que vous observez ?

2.3 Filtre médians vectoriels

2.3.1 Fondements théoriques

Ce type de filtre généralise le filtre médian pour les images vectorielles, typiquement pour des images en couleurs. Si l'espace couleur employé est de type RGB chaque échantillon est un vecteur défini par trois composantes (R, G, B). Pour une suite des vecteurs des x_1, ..., x_N le filtre médian vectoriel est défini par les relations suivantes :

$$\begin{cases} MV(x_1, x_2 \ldots x_N) = x_{MV} \\ \sum_{i=1}^{N} \left\| x_{MV} - x_i \right\|_L \leq \sum_{i=1}^{N} \left\| x_j - x_i \right\|_L, j = 1..N \end{cases} \quad (4)$$

La norme la plus utilisé est L_1 ; pour un vecteur de composantes (i_1,i_2,…i_k) elle est définie par :

$$|v|_{L_1} = |i_1| + |i_2| + \ldots |i_k|$$

2.3.2 Exercices

1. Le filtre médian vectoriel opère sur les trois composantes d'une image en couleurs. Créez tout d'abord les fonctions qui vous permettront de décomposer une image RGB:

- ajoutez une barre de menu nommée « Decomposition RGB » en rajoutant la ligne :

```
…
        "Decomposition RGB",                                IDM_DECOMP_RGB,
…
```

dans le corps du menu « Filtre non lineaires » (**attLibraryMenu.c**) et attribuez une valeur unique à IDM_DECOMP_RGB en **attLibraryMenu.h**

- en TP2.c écrivez le code suivant :

```
void DecompositionCouleurs ( unsigned char * ent_f,double* rouge, double
*vert,double *bleu, double* luminance, long larg, long haut)
{
        long i, j, add;
                for (i=0;i<larg;i++)
                for (j=0;j<haut;j++)
                        {
                        add = i+j*larg;
                        rouge[add] = ent_f[4*add];
                        vert[add] = ent_f[4*add+1];
                        bleu[add] = ent_f[4*add+2];
                        luminance[add]=0.3*rouge[add]+0.59*vert[add]+0.11*bleu[add];
                        }
}
```

- ajoutez le code suivant à **attLibraryRun.c**

```
case    IDM_DECOMP_RGB:
        {
        long larg_entree, haut_entree, id, id_sorf1,id_sorf2,id_sorf3,id_sorf4;
        char * ent_f1, * sor_f1, * sor_f2, * sor_f3, * sor_f4;
        f ( attDataSetNombre(numero_tache, IMG_ENTREE, 1) == FALSE ) break;

        id = attDataGetIdByCount(numero_tache, IMG_ENTREE, 1);
        larg_entree = attDataGetLargeur(id);
        haut_entree = attDataGetHauteur(id);
        ent_f1 = (double *)attDataGetData(id, TYPE_RGBA);
        if ( ent_f1 == NULL ) break;
        attDataSetImageNombreTypeTaille(numero_tache, 4, TYPE_DOUBLE,
larg_entree, haut_entree);          id_sorf1 = attDataGetIdByCount(numero_tache,
IMG_SORTIE, 1);
        attDataSetName(id_sorf1,"Rouge");
        id_sorf2 = attDataGetIdByCount(numero_tache, IMG_SORTIE, 2);
        attDataSetName(id_sorf2,"Vert");
        id_sorf3 = attDataGetIdByCount(numero_tache, IMG_SORTIE, 3);
        attDataSetName(id_sorf3,"Bleu");
        id_sorf4 = attDataGetIdByCount(numero_tache, IMG_SORTIE, 4);
        attDataSetName(id_sorf4,"Luminance");
        sor_f1 = (double *)attDataGetPtr(id_sorf1);
        sor_f2 = (double *)attDataGetPtr(id_sorf2);
        sor_f3 = (double *)attDataGetPtr(id_sorf3);
        sor_f4 = (double *)attDataGetPtr(id_sorf4);
        DecompositionCouleurs(ent_f1,sor_f1,sor_f2,sor_f3,sor_f4, larg_entree,
haut_entree);
        attDataFree(ent_f1);
        }
break;
```

- lancez en exécution l'application et chargez la bibliothèque pour tester le code.
- écrivez maintenant le code qui vous permettra de recomposer une image à partir de ses composantes rouge, bleu et vert. En procédant de manière similaire

ajoutez une barre de menu appelée « Composition RGB » et lui attribuant un identificateur unique IDM IDM_COMP_RGB.

- le code C de la fonction est :

```
void CompositionCouleurs( unsigned char * sorRGB, double* rouge, double *vert,double *bleu, long larg, long haut)
{       long i, j, add;
            for (i=0;i<larg;i++)
            for (j=0;j<haut;j++)
            {       add = i+j*larg;
                    sorRGB[4*add]=rouge[add];
                    sorRGB[4*add+1]=vert[add];
                    sorRGB[4*add+2]=bleu[add];
            }
```

- pour appeler la fonction le code suivant devra être ajouté en **attLibraryRun.c**

```
case    IDM_DIF_COMB_COULEURS  :
        {
        long larg_entree, haut_entree, id1,id2,id3,id_sor1;
        double * ent_f1, * ent_f2, *ent_f3;
        char *sor_f;
        if ( attDataSetNombre(numero_tache, IMG_ENTREE, 3) == FALSE ) break;

        id1 = attDataGetIdByCount(numero_tache, IMG_ENTREE, 1);
        larg_entree = attDataGetLargeur(id1);
        haut_entree = attDataGetHauteur(id1);
        ent_f1 = (double *)attDataGetData(id1, TYPE_DOUBLE);
        id2 = attDataGetIdByCount(numero_tache, IMG_ENTREE, 2);
        ent_f2 = (double *)attDataGetData(id2, TYPE_DOUBLE);
        id3 = attDataGetIdByCount(numero_tache, IMG_ENTREE, 3);
        ent_f3 = (double *)attDataGetData(id3, TYPE_DOUBLE);
        if ( ent_f1 == NULL ) break;
        if ( ent_f2 == NULL ) break;
        if ( ent_f3 == NULL ) break;
        attDataSetImageNombreTypeTaille(numero_tache, 1, TYPE_RGBA, larg_entree, haut_entree);
```

```
id_sor1 = attDataGetIdByCount(numero_tache, IMG_SORTIE, 1);
        sor_f = (double *)attDataGetPtr(id_sor1);
        CompositionCouleurs( sor_f, ent_f1, ent_f2,ent_f3, larg_entree,haut_entree);
        attDataFree(ent_f1);attDataFree(ent_f2); attDataFree(ent_f3);
        }
        break;
```

2. En vous inspirant du code précédant du code précédant écrivez une fonction qui permettra le filtrage d'une image à l'aide d'une filtre médian vectoriel de taille paramétrable. La fonction devra prendre pour paramètres trois images (les composantes R,G et B) et devra retourner trois images représentant les composantes R,G et B du pixel qui minimise $x_{MV.}$ Recomposer l'image RGB en utilisant la fonction crée avant.

3. Ouvrez une image couleur placée dans le répertoire ..\ImagesStandard. Testez différentes tailles pour le filtre et comparez.

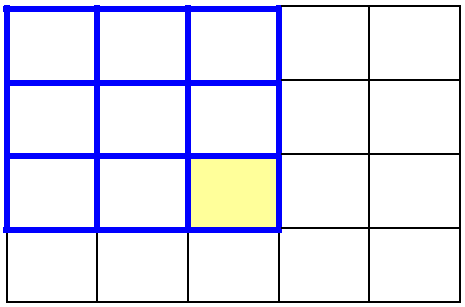
4. Ouvrez l'image en couleurs bruitée avec un bruit de type sel et poivre placée dans le répertoire ..\ImagesStandard et sel et poivre pour des imagex RGB (couleurs_sel_poivres.bmp). Testez le filtre pour une taille du voisinage de 5x5 pixels.

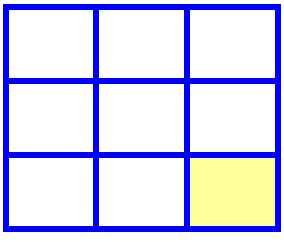
5. Répétez l'étape précédente en traitant chaque composante individuellement par un filtre médian scalaire. Comparez les résultats avec ceux obtenus précédemment. Est-ce qu'en procédant de cette manière on peut garantir que des nouveaux couleurs ne sont pas crées ?

2.4 Filtrage sélectif des régions

2.4.1 Introduction

Les filtres appartenant a cette classe (le filtre de Kuwahara, le filtre de Nagao) remplacent la valeur du pixel courant par la moyenne des valeurs calculée sur la plus homogène sous-région (l' homogénéité est exprimée en termes des variances- voir le support de cours) du voisinage et ils sont utiles pour des taches de lissage des images avec préservation des contours. Le principe derrière un filtre de type Kuwahara de taille 5x5 pixels est illustré dans la figure suivante.





- $\mu_1 = \frac{\sum_{(x_i.y_j)} u(x_i.y_j)}{9}$

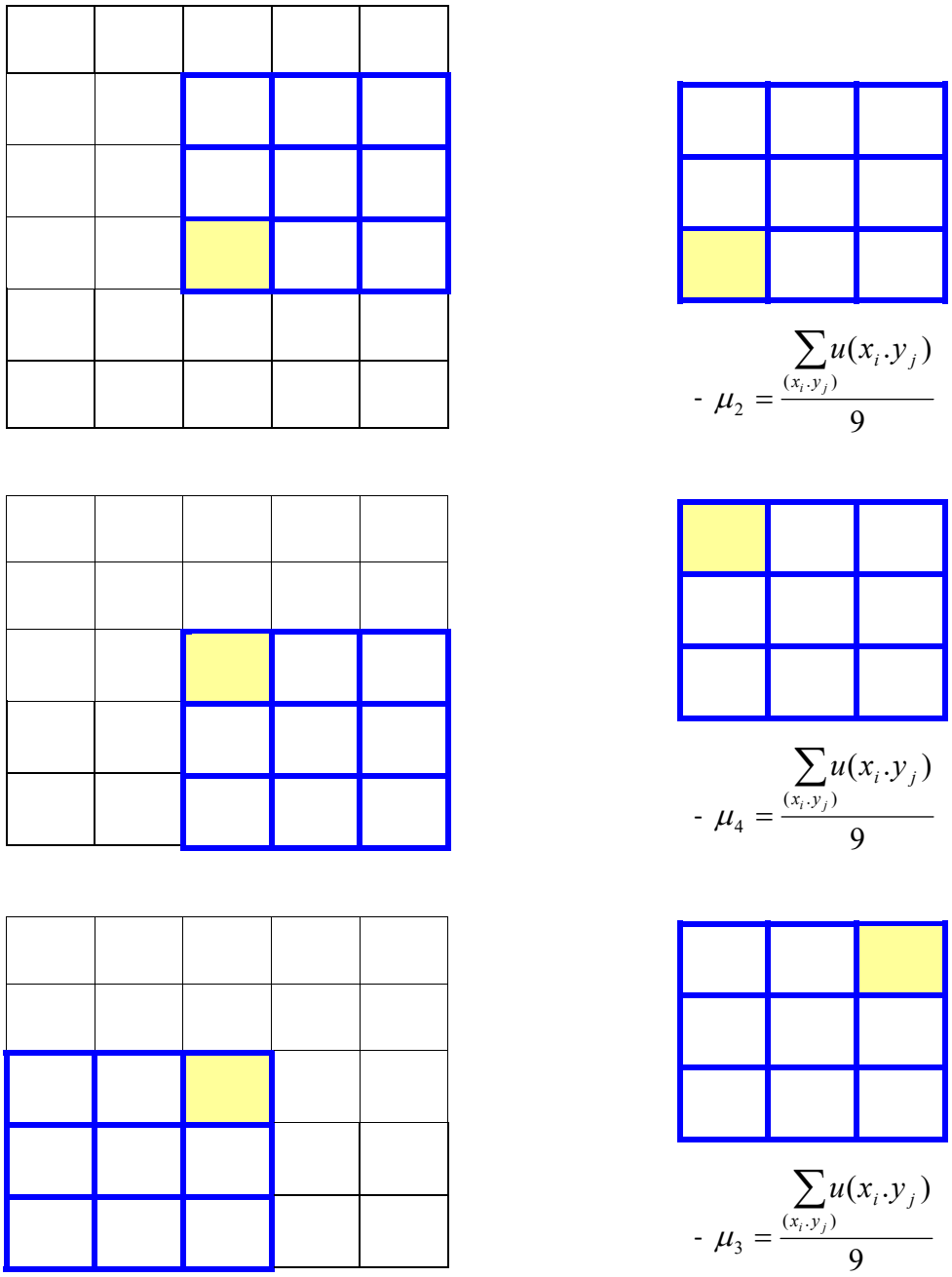


Fig.1 Filtre de Kuwahara

Le voisinage est divisée en 4 sous–régions, en chaque région la moyenne est la variance est calculée. La valeur filtrée est la moyenne qui corresponde à la zone avec la variance minimale.

2.4.2 Exercices

1. Procédez de manière similaire à 1.2 pour créer une fonction qui implante le filtrage par la méthode de Kuwahara pour une image en niveaux de gris.

2. Ouvrez l'image maison.bmp placee dans le répertoire ..\ImagesStandard. Filtrez l'image ; notez les effets du filtre.

3. Traitez la même image de manière itérative. Notez l'effet du filtre sur la qualité des contours.

4. Est-ce que ce type de filtre est efficace pour supprimer le bruit impulsif ?

Références bibliographiques

[1] S. K. Mitra; G. L. Sicuranza Nonlinear Image Processing Communications Networking And Multimedia, ISBN 10: 0125004516 ISBN 13: 9780125004510, Academic Press Inc, 2000

[2] R. Belaroussi-Traitement de l'image et de la vidéo : avec exercices pratiques en Matlab et C++, Traitement de l'image et de la vidéo. Avec exercices pratiques en Matlab et C++. IMAGES ET VIDEO, ISBN : 9782729854249, Technosup, 2010.

TP3 Equations aux dérivées partielles pour la restauration et amélioration des images

Objectifs du TP :

- implanter des modèles numériques pour les principaux types d'EDPs scalaires et tensorielles ;
- identifier les principales propriétés de chaque EDP et les possibles champs d'applications.

3.1 L'équation de diffusion isotrope

3.1.1 Fondements théoriques

L'EDP de diffusion isotrope modèle le processus de lissage d'une image par un processus similaire au celui de l'évolution de la température dans un milieu conducteur homogène. L'évolution d'une image est décrite par le système suivant des équations :

$$\begin{cases} U(x,y,0) = U_0(x,y) \\ \dfrac{\partial U}{\partial t} = div(\nabla U) = \Delta U =: \dfrac{\partial^2 U}{\partial x^2} + \dfrac{\partial^2 U}{\partial y^2} = U_{xx} + U_{yy} \end{cases} \quad (1)$$

En partant de l'image originale $U_o(x,y)$ au fur et à mesure que le temps augmente, l'EDP produit des versions de plus en plus simplifiées de l'image originale. Pour un instant *t* donné le processus est assimilable (voir le support de cours) avec une convolution de l 'image originale avec un noyaux Gaussien :

$$U(x,y,t) = G_\sigma(x,y) * U_0(x,y) \quad (2)$$

d'écart type :

$$\sigma = \sqrt{2t} \quad (3)$$

3.1.2 Résolution numérique

La résolution de l'équation (1) se fait en pratique par la méthode des différences finies, en partant d'un développement en série de Taylor d'une fonction de plusieurs variables. Si on considère :

$$U(x,y,t+dt) = U(x) + \frac{t-dt}{1!}\frac{\partial U}{\partial x}\bigg|_{x=x,y,dt} + \frac{(x-dt)^2}{2!}\frac{\partial^2 U}{\partial x^2}\bigg|_{x=x,y,dt} + \ldots \quad (4)$$

en ignorant tous les termes sauf le premier deux :

$$\frac{\partial U}{\partial t} = \frac{U(x,y,t+dt) - U(x,y,t)}{dt} \tag{5}$$

En considérant les images représentées sur des grilles régulières des points (*x=ih, y=jh*) et en simplifiant les notations :

$$\begin{cases} t = ndt \\ U(i,j,ndt) = U_{i,j}^{n}, \end{cases} \tag{6}$$

l'équation (5) devient :

$$\frac{\partial U}{\partial t} = \frac{U_{i,j}^{n+1} - U_{i,j}^{n}}{dt} = D_t^{+}(U_{i,j}) \tag{7}$$

L'opérateur D_t^{+}(.) s'appelle approximation d'ordre 1 progressive par rapport au temps.

L'EDP de diffusion isotrope s'écrit ensuite :

$$U_{i,j}^{n+1} = U_{i,j}^{n} + dt\Delta U = U_{i,j}^{n} + dt\Delta U_{i,j}^{n} \tag{8}$$

Remarques :

1) Eq. (8) décrit un processus itératif ; à chaque étape l'évolution du processus se fait en parallèle.
2) Le schéma de résolution s'appelle explicite.
3) Des contraintes doivent être imposées sur les valeurs de *dt* pour assurer la stabilité et la convergence du schéma numérique. Dans le cas 2D dt<=1/4, pour les images 3D dt<1/6.

Les dérivées partielles spatiales sont calculées en utilisant les approximations d'ordre 1 progressives et régressives suivant les deux directions :

$$D_x^{-}(U_{i,j}^{n}) = U_{i,j}^{n} - U_{i-1,j}^{n}\ D_x^{+}(U_{i,j}^{n}) = U_{i+1,j}^{n} - U_{i,j}^{n} \tag{9}$$

$$D_y^{+}(U_{i,j}^{n}) = U_{i,j+1}^{n} - U_{i,j}^{n},\ D_y^{-}(U_{i,j}^{n}) = U_{i,j}^{n} - U_{i,j-1}^{n}$$

$$\begin{aligned} U_{xx} &= D_x^{+}[D_x^{-}(U_{i,j}^{n}) = D_x^{+}[U_{i,j}^{n} - U_{i-1,j}^{n}] = U_{i+1,j}^{n} - U_{i,j}^{n} - U_{i,j}^{n} + U_{i-1,j}^{n} = \\ &= U_{i+1,j}^{n} - 2U_{i-1,j}^{n} + U_{i-1,j}^{n} \end{aligned} \tag{10.a}$$

$$U_{yy} = D_y^+[D_y^-(U_{i,j}^n)] = D_y^+[U_{i,j}^n - U_{i,j-1}^n] = U_{i,j+1}^n - U_{i,j}^n - U_{i,j}^n + U_{i,j-1}^n = \\ = U_{i,j+1}^n - 2U_{i,j}^n + U_{i,j-1}^n \quad (10.b)$$

Le modèle numérique complet de l'EDP de diffusion isotrope est :

$$U_{i,j}^{n+1} = U_{i,j}^n + dt\Delta U = U_{i,j}^n + dt(U_{i+1,j}^n - 2U_{i-1,j}^n + U_{i-1,j}^n + U_{i,j+1}^n - 2U_{i,j}^n + U_{i,j-1}^n) \quad (11)$$

3.1.3 Exercices

1. Ouvrez la plateforme Microsoft Visual Studio et ensuite ouvrez la solution ..\Compile_DLL.sln
2. Ajoutez le code suivant :

- dans le fichier **attLibraryMenu.c** :

```
void attLibraryMenu3()
{
        ATT_DESC_MENU menu3 =
                {
                        BEGIN_MENU
                        "Diffusion isotrope",           IDM_DIF_ISOTROPE,
                        END_MENU
                };
                attLibraryAddMenu("MENU_2D,Filtres de type EDP", menu3);
}
```

- dans le fichier **AttLibraryMenu.h** :

```
…
#define IDM_DIF_ISOTROPE            30
…
```

- dans le fichier **attLibraryMain.c** dans le corps de la fonction *GetMenuParametres*:

```
…
attLibraryMenu3();
…
```

Cette séquence ajoutera une barre de menu pour pouvoir accéder à la fonction qui implémentera le filtre.

3. Ajoutez un fichier nommé TP3.c au projet courrant (Project->Add to project->New-> C++ Source File).
4. Cliquez sur le fichier qui vient d'être crée
5. Ajoutez le code suivant dans le fichier TP3.c :

```
void Diffusion_Isotrope(double * entree, double * sortie, long largeur, long hauteur, long taille, long no_iterations)
{
```

```
// entrée –image d'entrée
// sortie – image de sortie
// largeur, hauteur – dimensions de l'image
// taille du voisinage
// arret  - le temp d' arret
long i,j,n, k;
double uxx, uyy ;
// pour un certain nombre d' iterations
for (n=0 ;n<no_iterations ;n++)
    {
     for (k=0 ;k<largeur*hauteur ;k++)
         sortie[k]=entree[k] ;
     for (i=taille/2; i<largeur-taille/2; i++)
     for (j=taille/2; j<hauteur-taille/2; j++)
         {
         }
    }
}
```

6. Ajoutez le code nécessaire afin que le modèle numérique de l' EDP de diffusion isotrope soit complet.
7. La fonction peut être appelée en rajoutant le code suivant dans le fichier **attLibraryRun.c** :

```
 case    IDM_DIF_ISOTROPE:
{
     long larg_entree, haut_entree, id;
     double * ent_f, * sor_f;
     static nbiter=5 ;
     if ( mode == MODE_AIDE )
          {attMsg("Diffusion isotrope");                    break;}
     if ( !attDataSetNombre(numero_tache, IMG_ENTREE, 1) ) break;

     if ( !attBoiteParametre("Nombre d'iterations =%d ", &nbiter) ) break;
     id = attDataGetIdByCount(numero_tache, IMG_ENTREE, 1);
    larg_entree = attDataGetLargeur(id);
    haut_entree = attDataGetHauteur(id);
    ent_f = (double *)attDataGetData(id, TYPE_DOUBLE);
```

```
    if ( ent_f == NULL ) break;
     attDataSetImageNombreTypeTaille(numero_tache, 1, TYPE_DOUBLE,
larg_entree, haut_entree);
     id = attDataGetIdByCount(numero_tache, IMG_SORTIE, 1);
      sor_f = (double *)attDataGetPtr(id);
      Diffusion_Isotrope (ent_f, sor_f, larg_entree, haut_entree, nbiter);
      attDataFree(ent_f);
       }
       break;
```

La partie représente en bleu est la façon standard en Attribut d'ouvrir une image, de définir une image de sortie et de les afficher sur l'écran.

8. Compilez la bibliothèque.
9. Lancez l'exécutable et chargez la bibliothèque crée avant.
10. Ouvrez l'image Lena.bmp placée dans le répertoire ..\ImagesStandard. Filtrez l'image en utilisant des temps d'arrêts de plus en plus grands. Quel est l'effet du nombre d'itérations sur le contenu de l'image?
11. Ouvrez l'image Lena_bruitee.bmp placée dans le répertoire ..\ImagesStandard. Quel est l'effet du nombre d'itérations sur le bruit présent dans l'image?
12. En pratique l'implémentation de l 'EDP de diffusion isotrope peut se faire à travers son interprétation c.a.d. une convolution avec un noyaux Gaussien.
13. En procédant de manière similaire a 2) ajoutez une barre de menu nomme « Convolution gaussienne » et attribuer une valeur unique. Les contenus des fichiers **attLibraryMenu.c** et **attLibraryMenu.h** devront être :

```
void attLibraryMenu3()
{
        ATT_DESC_MENU menu3 =
                {
                        BEGIN_MENU
                        "Diffusion isotrope",
IDM_DIF_ISOTROPE,
                          "Convolution Gaussiene",
IDM_CONV_GAUSS,
                        END_MENU
                };
                attLibraryAddMenu("MENU_2D,Filtres de type EDP", menu3);
}
```

- AttLibraryMenu.h :

```
...
#define IDM_DIF_ISOTROPE            30
```

```
#define IDM_CONV_GAUSS            31
…
```

14. Ajoutez le code suivant dans le fichier TP3.c :

```
void Convolution_Gauss(double * entree, double * sortie,long largeur,long
hauteur,double Sigma)
{
        long taille=largeur*hauteur, i, j, k, taille_filtre, demi_taille;
        double pix1, pix2;
        double * interm;
        double filtre[100];
        double  somme;
        double test;
        taille_filtre=(int) (4 * Sigma)+1;
        if (taille_filtre  %2 ==0)
                taille_filtre=taille_filtre+1;
        else
                ;
        demi_taille=(int)(taille_filtre-1)/2;
        interm =(double *) calloc (taille, sizeof(double));
        somme=0.0;
        // Calcul de coefficients du filtre
        for (i=0; i<=taille_filtre-1; i++)
        {
                test=(i-demi_taille)*(i-demi_taille);
                test=test/(2*Sigma*Sigma);
                test=exp(-test);
                filtre[i]=exp(-(i-demi_taille)*(i-
demi_taille)/(2*Sigma*Sigma))/(2*PI*Sigma*Sigma);
                somme=somme+filtre[i];
        }
        // Normalisation de coefficients du filtre//
        for (i=0; i<=taille_filtre-1; i++)
        {
                filtre[i]=filtre[i]/somme;
        }
        // filtrage suivant les colonnes
        for (i=demi_taille ;i<hauteur-demi_taille; i++)
```

```
                for (j=demi_taille ;j<largeur-demi_taille; j++)
                {
                        pix1=0.0;
                        for (k=i-demi_taille; k<=i+demi_taille; k++)
                        {
                                pix2=entree[k*largeur+j];
                                pix1=pix1+(filtre[k-i+demi_taille]*pix2);
                        }
                        interm[i*largeur+j]=(pix1);
                }
        // filtrage suivant les lignes
        for (i=demi_taille ;i<hauteur-demi_taille; i++)
                for (j=demi_taille ;j<largeur-demi_taille; j++)
                {
                        pix1=0.0;
                        for (k=j-demi_taille; k<=j+demi_taille; k++)
                        {
                                pix2=interm[i*largeur+k];
                                pix1=pix1+(filtre[k-j+demi_taille]*pix2);
                        }
                        sortie[i*largeur+j]=(pix1);
                }
        free(interm);
}
```

15. La fonction peut être appelée en rajoutant le code suivant dans le fichier **attLibraryRun.c :**

```
 case    IDM_CONV_GAUSS:
{
    long larg_entree, haut_entree, id;
    double * ent_f, * sor_f;
    double sigma;
    if ( mode == MODE_AIDE )
        {attMsg("Convolution gaussiene");              break;}
    if ( !attDataSetNombre(numero_tache, IMG_ENTREE, 1) ) break;

    if ( !attBoiteParametre("Ecart type =%f ", &sigma) ) break;
```

```
 id = attDataGetIdByCount(numero_tache, IMG_ENTREE, 1);
 larg_entree = attDataGetLargeur(id);
 haut_entree = attDataGetHauteur(id);
 ent_f = (double *)attDataGetData(id, TYPE_DOUBLE);
if ( ent_f == NULL ) break;
 attDataSetImageNombreTypeTaille(numero_tache, 1, TYPE_DOUBLE,
 larg_entree, haut_entree);
 id = attDataGetIdByCount(numero_tache, IMG_SORTIE, 1);
 sor_f = (double *)attDataGetPtr(id);
 Convolution_Gauss  (ent_f, sor_f, larg_entree, haut_entree, sigma);
 attDataFree(ent_f);
 }
 break;
```

16. Ouvrez une image et traitez la avec un filtre de type EDP isotrope en considérant un nombre de 5 itérations. Traitez la même image avec un filtre Gaussien d'écart type σ=1.0. Analysez les résultats obtenus.
17. Expliquez le comportement de l'EDP de diffusion isotrope à travers son interprétation

3.2 L'EDP de diffusion anisotrope

3.2.1 Fondements théoriques

L'EDP de diffusion anisotrope a été introduite par Perona et Malik afin d'éliminer les inconvénients de l 'EDP de diffusion isotrope : le déplacement des contours et l'élimination des petits détails. L'EDP de diffusion anisotrope est :

$$\begin{cases} U(x,y,0) = U_0(x,y) \\ \dfrac{\partial U}{\partial t} = div[c(x,y,t)\nabla U(x,y,t)] \end{cases} \quad (12)$$

La diffusivité *c(x,y,t)* est construite de manière à favoriser le lissage intra-régions et à pénaliser la diffusion inter-régions à travers l'utilisation d'une fonction décroissante, dépendante de la norme du vecteur gradient :

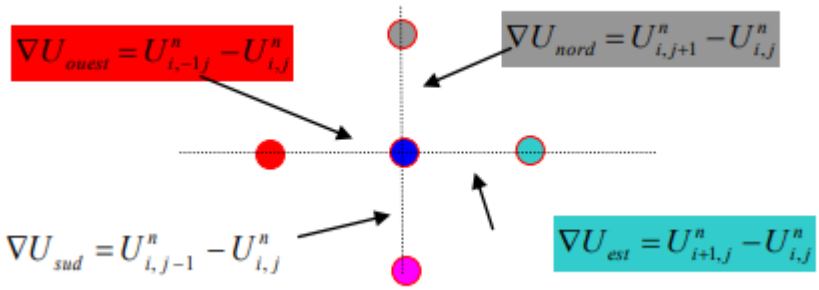
$$\begin{cases} c(x,y,t) = g(|\nabla U|) \\ g[|\nabla U|] = \dfrac{1}{1+\left[\dfrac{|\nabla U|}{K}\right]^2} \end{cases} \quad (13)$$

3.2.2 Résolution numérique

La résolution de l'équation (12) se fait en utilisant le schéma suivant (voir le support de cours) :

$$\nabla U_{sud} = U^n_{i,j-1} - U^n_{i,j}\, D_x^-\{g[|D_x^+(U^n_{i,j})|]D_x^+(U^n_{i,j})\} + D_y^-\{g[|D_y^+(U^n_{i,j})|]D_y^+(U^n_{i,j})\}$$

En considérant l'image représentée sur une grille sur une grille régulière des points des développements simples conduisent au modèle suivant : $U^{n+1}_{i,j} = U^n_{i,j} + dt\{\nabla U_{nord}\, g(\nabla U_{nord}) + \nabla U_{sud}\, g(\nabla U_{sud}) + \nabla U_{ouest}\, g(\nabla U_{ouest}) + \nabla U_{est}\, g(\nabla U_{est})\}$ dont l'interprétation est donnée dans la figure suivante :



Un pas de discrétisation en temps inferieur à 0.25 assure la stabilité du schéma numérique.

3.2.3 Exercices

1. Ajoutez une barre de menu et attribuez la une valeur unique. Les contenus des fichiers **attLibraryMenu.c** et **attLibraryMenu.h** devront être :

```
void attLibraryMenu3()
{
        ATT_DESC_MENU menu3 =
                {
                        BEGIN_MENU
                        "Diffusion isotrope",  IDM_DIF_ISOTROPE,
                         "Convolution Gaussiene", IDM_CONV_GAUSS,
                        "Diffusion anisotrope",IDM_DIF_ANISOTROPE,
                           END_MENU
                };
                attLibraryAddMenu("MENU_2D,Filtres de type EDP", menu3);
}
```

- fichier **AttLibraryMenu.h** :

```
…
#define IDM_DIF_ISOTROPE          30
#define IDM_CONV_GAUSS            31
#define IDM_DIF_ANISOTROPE                32
…
```

2. Ajoutez le code suivant dans le fichier TP3.c :

```
void Diffusion_Anisotrope(double * entree, double * sortie, long largeur, long hauteur, long taille, long no_iterations, double barriere)
{
// entrée –image d'entrée
// sortie – image de sortie
// largeur, hauteur – dimensions de l'image
// taille du voisinage
// arret  - le temp d' arret
long i,j,n, k;
double uxx, uyy ;
// pour un certain nombre d' iterations
for (n=0 ;n<no_iterations ;n++)
    {
     for (k=0 ;k<largeur*hauteur ;k++)
         sortie[k]=entree[k] ;
     for (i=taille/2; i<largeur-taille/2; i++)
     for (j=taille/2; j<hauteur-taille/2; j++)
         {
         }
     }
}
```

3. Ajoutez le code nécessaire afin que le modèle numérique de l' EDP de diffusion anisotrope soit complet.
4. La fonction peut être appelée en rajoutant le code suivant dans le fichier **attLibraryRun.c** :

```
 case    IDM_DIF_ANISOTROPE:
{
     long larg_entree, haut_entree, id;
     double * ent_f, * sor_f;
     static long nbiter=10 ;
     static double K=10 ;
     if ( mode == MODE_AIDE )
         {attMsg("Diffusion isotrope");                    break;}
```

```
    if ( !attDataSetNombre(numero_tache, IMG_ENTREE, 1) ) break;

    if ( !attBoiteParametre("Nombre d'iterations =%d Barriere de diffusion",
&nbiter, &K) ) break;
    id = attDataGetIdByCount(numero_tache, IMG_ENTREE, 1);
    larg_entree = attDataGetLargeur(id);
    haut_entree = attDataGetHauteur(id);
    ent_f = (double *)attDataGetData(id, TYPE_DOUBLE);

   if ( ent_f == NULL ) break;
    attDataSetImageNombreTypeTaille(numero_tache,      1,      TYPE_DOUBLE,
larg_entree, haut_entree);
    id = attDataGetIdByCount(numero_tache, IMG_SORTIE, 1);
    sor_f = (double *)attDataGetPtr(id);
    Diffusion_Anisotrope (ent_f, sor_f, larg_entree, haut_entree, nbiter,K);
    attDataFree(ent_f);
     }
     break;
```

5. Compilez la bibliothèque.
6. Lancez l'exécutable et chargez la bibliothèque crée avant.
7. Ouvrez l'image Lena.bmp placée dans le répertoire ..\ImagesStandard. Filtrez l'image en utilisant le même nombre d'itérations mais des barrières de plus en plus grandes. Quel est l'effet sur l'image ? Expliquez-le.
8. Ouvrez une image placée dans le répertoire ..\ImagesStandard. Filtrez l'image en utilisant l'EDP de diffusion anisotrope (K=15, nbiter=50) et l'EDP de diffusion isotrope (nbiter=50). Expliquez les différences.
9. Ouvrez l'image bruit_gauss.bmp placée dans le répertoire ..\ImagesStandard. Trouvez le choix des paramètres qui permet l'élimination du bruit et apporte les moindres modifications aux objets présents dans l'image. Expliquez le choix des paramètres.
10. Modifiez la fonction de diffusion afin qu'elle permette le filtrage des images avec la fonction de You :

$$\begin{cases} g(s) = \frac{1}{K} + p(K+\varepsilon)^{p-1}, si \;\; s < K \\ g(s) = \frac{1}{s} + p(s+\varepsilon)^{p-1}, si \;\; s \geq K \end{cases}$$

11. Traitez la même image avec l'EDP anisotrope et la fonction de You en prenant comme paramètres K=10, p=0.25. Analysez et commenter les résultats obtenus.

12. Traitez l'image image_medicale.bmp avec l'EDP anisotrope et l'EDP isotrope en prenant pour paramètres K=3, nbiter=75. Analysez et commenter les résultats obtenus.
13. Analysez les propriétés de l'EDP de diffusion anisotrope de filtrage du bruit en prenant comme image d'entrée Lena_bruitee.bmp et des valeurs de plus en plus grandes pour le paramètre K.

3.3 Le modèle de lissage sélectif de Catte et al.

3.3.1 Fondements théoriques

Le modèle adresse les inconvénients théoriques et pratiques du filtre Perona-Malik (voir le support de cours). En introduisant une pré convolution avec un noyau Gaussien dans l'estimation de la fonction de diffusion les auteurs montrent que le nouveau filtre admet une solution unique et a des meilleures propriétés d'élimination du bruit. L'EDP est :

$$U(x,y,0) = U_0(x,y)$$

$$\frac{\partial U}{\partial t} = div[g(|\nabla G_\sigma * U|)\nabla U] = 0 \qquad (14)$$

3.3.2 Résolution numérique

Le modèle numérique de Catte et al. est le suivant (voir le support de cours) :

$$\frac{\partial U_{i,j}^n}{\partial t} = \frac{1}{h}\{\frac{[g(|\nabla U_\sigma|)|_{i+1,j} + g(|\nabla U_\sigma|)|_{i,j}]}{2} D_x^+ U_{i,j}^n - \frac{[g(|\nabla U_\sigma|)|_{i-1,j} + g(|\nabla U_\sigma|)|_{i,j}]}{2} D_x^- U_{i,j}^n\} +$$
$$+\frac{1}{h}\{\frac{[g(|\nabla U_\sigma|)|_{i,j+1} + g(|\nabla U_\sigma|)|_{i,j}]}{2} D_y^+ U_{i,j}^n - \frac{[g(|\nabla U_\sigma|)|_{i,j-1} + g(|\nabla U_\sigma|)|_{i,j}]}{2} D_y^- U_{i,j}^n\}$$

avec :

$$|\nabla U_\sigma|_{i,j} \cong \sqrt{D_x^0(U_{\sigma i,j})^2 + D_y^0(U_{\sigma i,j})^2}$$

et

$$D_x^0(U_{i.j}^n) = \frac{U_{i+1,j}^n - U_{i-1,j}^n}{2}, \quad D_y^0(U_{i.j}^n) = \frac{U_{,j+1}^n - U_{i,j}^n}{2}$$

3.3.3 Exercices

1. Ajoutez une barre de menu et attribuez la une valeur unique. Les contenus des fichiers **attLibraryMenu.c** et **attLibraryMenu.h** devront être :

```
void attLibraryMenu3()
```

```
{
        ATT_DESC_MENU menu3 =
                {
                        BEGIN_MENU
                        "Diffusion isotrope", IDM_DIF_ISOTROPE,
                        "Convolution Gaussiene", IDM_CONV_GAUSS,
                        "Diffusion anisotrope",    IDM_DIF_ANISOTROPE,
                        "Diffusion Catte et al.",  IDM_DIF_CATTE,
                           END_MENU
                };
                attLibraryAddMenu("MENU_2D,Filtres de type EDP", menu3);
}
```

- fichier AttLibraryMenu.h :

```
…
#define IDM_DIF_ISOTROPE          30
#define IDM_CONV_GAUSS            31
#define IDM_DIF_ANISOTROPE        32
#define IDM_DIF_CATTE             33
…
```

2. Ajoutez le code suivant dans le fichier TP3.c :

```
void Diffusion_Catte(double * entree, double * sortie, long largeur, long hauteur,
long taille, long no_iterations, double barriere, double sigma)
{
// entrée –image d'entrée
// sortie – image de sortie
// largeur, hauteur – dimensions de l'image
// taille du voisinage
// arret  - le temp d' arret
long i,j,n, k;
double uxx, uyy ;
double * interm ;
interm=(double*) calloc (largeur*hauteur, sizeof(double) );
// pour un certain nombre d' iterations
for (n=0 ;n<no_iterations ;n++)
    {
     for (k=0 ;k<largeur*hauteur ;k++)
         sortie[k]=entree[k] ;
    Convolution_Gauss(entree, interm, largeur, hauteur, sigma) ;
```

```
    for (i=taille/2; i<largeur-taille/2; i++)
    for (j=taille/2; j<hauteur-taille/2; j++)
        {
        }
    }
free(interm) ;
}
```

3. Ajoutez le code nécessaire afin que le modèle numérique de l' EDP de Catte et al. soit complet.
4. La fonction peut être appelée en rajoutant le code suivant dans le fichier **attLibraryRun.c :**

```
 case   IDM_DIF_ANISOTROPE:
{
    long larg_entree, haut_entree, id;
    double * ent_f, * sor_f;
    static long nbiter=10 ;
    static double K=10 ;
    static double sigma=1.0 ;

    if ( mode == MODE_AIDE )
        {attMsg("Diffusion isotrope");                    break;}
    if ( !attDataSetNombre(numero_tache, IMG_ENTREE, 1) ) break;

    if ( !attBoiteParametre("Nombre d'iterations =%d Barriere de diffusion %f,
Sigma %f", &nbiter, &K,  &sigma) ) break;
    id = attDataGetIdByCount(numero_tache, IMG_ENTREE, 1);
   larg_entree = attDataGetLargeur(id);
   haut_entree = attDataGetHauteur(id);
   ent_f = (double *)attDataGetData(id, TYPE_DOUBLE);

  if ( ent_f == NULL ) break;
   attDataSetImageNombreTypeTaille(numero_tache, 1, TYPE_DOUBLE,
larg_entree, haut_entree);
   id = attDataGetIdByCount(numero_tache, IMG_SORTIE, 1);
    sor_f = (double *)attDataGetPtr(id);
    Diffusion_Catte (ent_f, sor_f, larg_entree, haut_entree, nbiter,K, sigma);
    attDataFree(ent_f);
     }
     break;
```

3.4 Filtres de choc

3.4.1 Fondements théoriques

Les filtres de choc adressent un problème classique en restauration d'images : l'élimination du flou. Le processus de restauration est modelé par une équation inverse de propagation de la chaleur dont l'évolution en temps conduit à une accentuation des objets d 'une image. Le processus est engendré par l' EDP (Osher et Rudin) :

$$\frac{\partial U}{\partial t} = -signe(U_{\eta\eta})\left|\nabla U\right| \tag{15}$$

La stabilité de l'EDP ne peut pas être assurée que dans le domaine numérique par un modèle numérique élaboré :

$$\left|\nabla U\right|_{i,j}^{n} = \sqrt{\{m[D_x^+(U_{i,j}^n), D_x^-(U_{i,j}^n)]\}^2 + \{m[D_y^+(U_{i,j}^n), D_y^-(U_{i,j}^n)]\}^2}$$

$$\begin{cases} U_{\eta\eta} = \nabla U_\eta \cdot \vec{\eta} = \dfrac{1}{\left|\nabla U\right|}[\dfrac{\partial}{\partial x}(\left|\nabla U\right|)U_x + \dfrac{\partial}{\partial y}(\left|\nabla U\right|)U_y] \\ \quad = \dfrac{(U_{xx}U_x + U_{yx}U_y)U_x + (U_{xy}U_x + U_{yy}U_y)U_y}{\left|\nabla U\right|^2} = \\ \quad = \dfrac{U_{xx}U_x^2 + 2U_{xy}U_xU_y + U_{yy}U_y^2}{\left|\nabla U\right|^2} \end{cases}$$

$$U_{yy} = U_{i,j+1}^n - 2U_{i,j}^n + U_{i,j-1}^n$$

$$U_{xx} = U_{i+1,j}^n - 2U_{i,j}^n + U_{i-1,j}^n$$

$$\frac{\partial^2 U_{i,j}^n}{\partial x \partial y} = \frac{U_{i+1,j+1}^n + U_{i-1,j-1}^n - U_{i+1,j-1}^n - U_{i-1,j+1}^n}{4h^2}$$

La fonction *m(.)* s'appelle *minmod* et elle sélectionne en tout pixel la direction appropriée de diffusion afin que les points d'inflexion de l'image soient préservés :

$$m(x,y) = [signe(x) + signe(y)]\min(\left|x\right|, \left|y\right|)/2 \tag{16}$$

La méthode de résolution est explicite, un pas de discrétisation en temps dt=0.1 garanti la stabilité du schéma numérique

3.4.2 Exercices

1. Ajoutez une barre de menu et écrivez le code qui implante l'EDP des filtres de choc.

2. En utilisant la fonction crée traitez l'image Lena_floue.bmp et analysez les résultats.

3. Répétez l'étape 3) pour l'image Lena_bruitee.bmp.

Références bibliographiques

1. R. Terebes - Diffusion directionnelle. Applications à la restauration et à l'amélioration d'images de documents anciens. Thèse de doctorat, Université Bordeaux 1 http://grenet.drimm.ubordeaux1.fr/pdf/2004/TEREBES_ROMULUS_MIRCEA_2004.pdf
2. D. Tscumperlé – Régularisation d'Images Multivaluées par EDP et Applications. Thèse de doctorat, Université de Nice Sophia-Antipolis -http://tel.ccsd.cnrs.fr/documents/archives0/00/00/23/96/index.html
3. S. Osher, L. Rudin - "Feature-oriented image enhancement with shock filters", S1IAM Journal on Numerical Analysis, vol.27, no.3, pp. 919-940, 1990.
4. P. Perona, J. Malik -"Scale space and edge detection using anisotropic diffusion", IEEE Transactions on Pattern Analysis an Machnie 1ntellngence, vol.12, no.7, pp.629-639, 1990.
5. Support de cours http://ares.utcluj.ro/tapisv.

TP4 Diffusion tensorielle

Objectifs du TP :

- implanter les modèles numériques pour les principaux types d'EDPs tensorielles ;
- identifier les principales propriétés de chaque EDP implantée.

4.1 Filtre de diffusion pour le rehaussement des contours (Edge Enhancing Diffusion-EED)

4.1.1 Fondements théoriques

Le filtre de type EED est un filtre tensoriel de diffusion dont la diffusivité scalaire est remplacée par une matrice de diffusion :

$$\frac{\partial U}{\partial t} = div(D\nabla U) \tag{1}$$

La matrice de diffusion est définie en partant d'une image prélissée par une convolution avec un noyau Gaussien d'écart type σ $U_\sigma = G_\sigma * U_0(x,y)$. Ensuite la matrice de diffusion est définie par :

$$D = \frac{1}{(U_{\sigma x})^2 + (U_{\sigma y})^2} \begin{pmatrix} \lambda_1 (U_{\sigma x})^2 + \lambda_2 (U_{\sigma y})^2 & (\lambda_2 - \lambda_1) U_{\sigma x} U_{\sigma y} \\ (\lambda_2 - \lambda_1) U_{\sigma x} U_{\sigma y} & \lambda_1 (U_{\sigma y})^2 + \lambda_2 (U_{\sigma x})^2 \end{pmatrix} \tag{2}$$

dont les deux valeurs λ_1 et λ_2 sont choisies de manière à diffuser au long de contours et arrêter le processus de lissage à travers les contours :

$$\begin{cases} \lambda_1 = \begin{cases} 1 & si \ |\nabla U_\sigma| = 0 \\ 1 - \exp(-1/(|\nabla U_\sigma| / K)^{2m}) & si \ |\nabla U_\sigma| > 0 \end{cases} \\ \lambda_2 = 1 \end{cases} \tag{3}$$

4.1.2 Résolution numérique

Le modèle numérique correspondant à l'EED est (voir le support de cours) :

$$U_{i,j}^{n+1} = U_{i,j}^n + dtL * U_{i,j}^n$$

avec

$D = \begin{pmatrix} a & b \\ c & a \end{pmatrix}$ et :

L =

$-(b_{i-1,j} + b_{i,j-1})/4h^2$	$(c_{i,j-1} + c_{i,j})/2h^2$	$(b_{i+1,j} + b_{i,j-1})/4h^2$
$(a_{i-1,j} + a_{i,j})/2h^2$	$-(a_{i-1,j} + 2\,a_{i,j} + a_{i+1,j})/2h^2$ $-(c_{i,j-1} + 2\,c_{i,j} + c_{i,j+1})/2h^2$	$(a_{i+1,j} + a_{i,j})/2h^2$
$(b_{i-1,j} + b_{i,j+1})/4h^2$	$(c_{i,j+1} + c_{i,j})/2h^2$	$-(b_{i+1,j} + b_{i,j+1})/4h^2$

4.1.3 Exercices

1. Ouvrez la plateforme Microsoft Visual Studio et ensuite ouvrez la solution …\Compile_DLL.sln
2. Ajoutez le code suivant :

- dans le fichier attLibraryMenu.c :

```
void attLibraryMenu3()
{
        ATT_DESC_MENU menu4 =
                {
                        BEGIN_MENU
                        "Diffusion EED", IDM_DIF_EED,
                        END_MENU
                };
                attLibraryAddMenu("MENU_2D,Filtres de type tensorielles", menu4);
}
```

- dans le fichier **aAttLibraryMenu.h** :

```
…
#define IDM_DIF_EED          40
…
```

- dans le fichier **attLibraryMain.c** , dans le corps de la fonction *GetMenuParametres*:

```
…
attLibraryMenu4();
…
```

Cette séquence ajoutera une barre de menu pour pouvoir accéder à la fonction qui implémentera le filtre.

3. Ajoutez un fichier nommé TP4.c au projet courrant (Project->Add to project->New-> C++ Source File)
4. Cliquez sur le fichier qui vient d'être crée.
5. Ajoutez le code suivant dans le fichier TP4.c :

```
void Diffusion_EED(double * entree, double * sortie, long largeur, long hauteur, long taille, long no_iterations, double barriere, double sigma, double m)
{
// entrée –image d'entrée
// sortie – image de sortie
// largeur, hauteur – dimensions de l'image
// taille du voisinage
// arret  - le temp d' arret
long i,j,n, k;
double uxx, uyy ;
double * interm ;
interm=(double*) calloc (largeur*hauteur, sizeof(double) );
// pour un certain nombre d' iterations
 for (k=0 ;k<largeur*hauteur ;k++)
        sortie[k]=entree[k] ;

for (n=0 ;n<no_iterations ;n++)
    {
     for (k=0 ;k<largeur*hauteur ;k++)
        entree[k]=sortie[k] ;
     Convolution_Gauss(entree, interm, largeur, hauteur, sigma) ;
     for (i=taille/2; i<largeur-taille/2; i++)
     for (j=taille/2; j<hauteur-taille/2; j++)
         {
         }
     }
free(interm) ;
}
```

5. Ajoutez le code nécessaire afin que le modèle numérique de l' EDP de Weickert soit complet.

6. La fonction peut être appelée en rajoutant le code suivant dans le fichier **attLibraryRun.c :**

```
 case   IDM_DIF_EED:
{
     long larg_entree, haut_entree, id;
     double * ent_f, * sor_f;
     static long nbiter=10 ;
     static double K=10 ;
     static double sigma=1.0 ;
     static double m=1.0 ;
     if ( mode == MODE_AIDE )
         {attMsg("Diffusion isotrope");                    break;}
     if ( !attDataSetNombre(numero_tache, IMG_ENTREE, 1) ) break;

     if ( !attBoiteParametre("Nombre d'iterations =%d Barriere de diffusion=%f%n,
Sigma=%f,  Exp=%f ", &nbiter, &K,  &sigma, &m) ) break;
     id = attDataGetIdByCount(numero_tache, IMG_ENTREE, 1);
    larg_entree = attDataGetLargeur(id);
    haut_entree = attDataGetHauteur(id);
    ent_f = (double *)attDataGetData(id, TYPE_DOUBLE);

   if ( ent_f == NULL ) break;
    attDataSetImageNombreTypeTaille(numero_tache,      1,      TYPE_DOUBLE,
larg_entree, haut_entree);
     id = attDataGetIdByCount(numero_tache, IMG_SORTIE, 1);
     sor_f = (double *)attDataGetPtr(id);
     Diffusion_EED (ent_f, sor_f, larg_entree, haut_entree, nbiter,K, sigma, m);
     attDataFree(ent_f);
      }
      break;
```

7. En utilisant la fonction crée, traitez l'image geom.bmp avec K=4, nbiter=50, m=2, sigma=0.75. Comparez les résultats avec ceux obtenus par un filtre de type Catté et al. correspondant au même choix des paramètres et avec ceux obtenus en utilisant une EDP de diffusion isotrope. Expliquez les différences.
8. Répétez 7) pour l'image maison_gauss.bmp

9. Répétez 8) en choisissant sigma=2. Expliquez les différences.

4.2 Diffusion pour l'augmentation de la cohérence (Coherence Enhancing Diffusion-CED)

4.2.1 Fondements théoriques

Une autre méthode a été proposée par Weickert pour l'augmentation de la cohérence des textures 1D. L'équation de diffusion est toujours (1). Ce qui diffère dans cette approche est la manière de construire la matrice de diffusion *D*. Un rôle central est joué par le tenseur de structure (voir le support de cours) :

$$J_\rho(\nabla U_\sigma) = G_\rho * J_0(\nabla U_\sigma) = \begin{pmatrix} G_\rho * (\frac{\partial U_\sigma}{\partial x})^2 & G_\rho * \frac{\partial U_\sigma}{\partial x}\frac{\partial U_\sigma}{\partial y} \\ G_\rho * \frac{\partial U_\sigma}{\partial x}\frac{\partial U_\sigma}{\partial y} & G_\rho * (\frac{\partial U_\sigma}{\partial y})^2 \end{pmatrix} \quad (4)$$

Avec la notation $J = \begin{pmatrix} j_{11} & j_{12} \\ j_{12} & j_{22} \end{pmatrix}$, les valeurs propres sont choisies de manière à induire une diffusion prépondérante dans la direction moyenne des structures :

$$\lambda_1 =: \alpha$$

$$\lambda_2 =: \begin{cases} \alpha & si \;\; j_{11} = j_{22} \; et \; j_{12} = 0 \\ \alpha + (1-\alpha)\exp[-\frac{C}{(j_{11}-j_{22})^2 + 4j_{12}^2}] & sinon \end{cases} \quad (5)$$

La matrice de diffusion est construite ensuite :

$$D = \begin{pmatrix} \vec{v_1} \mid \vec{v_2} \end{pmatrix} \begin{pmatrix} \mu_1 & 0 \\ 0 & \mu_2 \end{pmatrix} \begin{pmatrix} \vec{v_1}^T \\ \vec{v_2}^T \end{pmatrix} = \begin{pmatrix} a & b \\ b & c \end{pmatrix} \quad (6)$$

ou les composantes des vecteurs sont :

$$\vec{v_1} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} \frac{2j_{12}}{\sqrt{\left(j_{22} - j_{11} + \sqrt{(j_{11}-j_{22})^2 + 4j_{12}^2}\right)^2 + 4j_{12}^2}} \\ \frac{j_{22} - j_{11} + \sqrt{(j_{11}-j_{22})^2 + 4j_{12}^2}}{\sqrt{\left(j_{22} - j_{11} + \sqrt{(j_{11}-j_{22})^2 + 4j_{12}^2}\right)^2 + 4j_{12}^2}} \end{pmatrix} \qquad \vec{v_2} = \begin{pmatrix} -v_y \\ v_x \end{pmatrix} \quad (7)$$

Les valeurs *a,b,c* sont :

$$\begin{cases} a = \lambda_1 v_x^2 + \lambda_2 v_y^2 \\ b = (\lambda_1 - \lambda_2) v_x v_y \\ c = \lambda_1 v_y^2 + \lambda_2 v_x^2 \end{cases} , \quad (8)$$

4.2.3 Exercices

1. Ajoutez une barre de menu et écrivez le code qui implante le filtre CED.

2. Traitez les images geom.bmp et maison_gauss.bmp et analysez les résultats obtenus.

2. En utilisant la fonction crée traitez l'image empreinte.bmp avec α=0.01, nbiter=100, sigma=2.0, ro=4.0. Comparez les résultats avec ceux obtenus par un filtre EED correspondant au choix indiqué en 1.8.

3. Répétez 2) en choisissant α=-0.01.

4.3 L'EDP de diffusion de Tschumperlé et Deriche

Notons aussi l'existence de l'approche récente de régularisation tensorielle de Tschumperlé et Deriche. En partant de (1) l'auteur propose une EDP :

$$\frac{\partial U}{\partial t} = trace\left[\begin{pmatrix} \rightarrow\rightarrow^T \\ v_1\, v_1 \end{pmatrix} \frac{1}{\lambda_1+\lambda_2+1} + \begin{pmatrix} \rightarrow\rightarrow^T \\ v_2\, v_2 \end{pmatrix} \frac{1}{\sqrt{\lambda_1+\lambda_2+1}} \begin{pmatrix} U_{xx} & U_{xy} \\ U_{xy} & U_{yy} \end{pmatrix}\right] \tag{9}$$

dont l'interprétation directionnelle est :

$$\frac{\partial U}{\partial t} = \frac{1}{\lambda_1+\lambda_2+1} U_{v_1 v_1} + \frac{1}{\sqrt{\lambda_1+\lambda_2+1}} U_{v_2 v_2} \tag{10}$$

et les valeurs propres sont celles du tenseur de structure:

$$\lambda_{1,2} = (j_{11}+j_{22} \pm \sqrt{(j_{11}-j_{22})^2+4j_{12}^2}) \tag{11}$$

En utilisant l'équation (6.a) et les approximations d'ordre 2 :

$$\begin{cases} \dfrac{\partial^2 U_{i,j}^n}{\partial x^2} = \dfrac{U_{i+1,j}^n + U_{i-1,j}^n - 2U_{i,j}^n}{h^2} \\ \dfrac{\partial^2 U_{i,j}^n}{\partial y^2} = \dfrac{U_{i,j+1}^n + U_{i,j-1}^n - 2U_{i,j}^n}{h^2} \\ \dfrac{\partial^2 U_{i,j}^n}{\partial x \partial y} = \dfrac{U_{i+1,j+1}^n + U_{i-1,j-1}^n - U_{i+1,j-1}^n - U_{i-1,j+1}^n}{4h^2} \end{cases}$$

implantez ce nouveau modèle et comparez-le avec les deux autres modèles tensoriels.

Références bibliographiques

1. R. Terebes - Diffusion directionnelle. Applications à la restauration et à l'amélioration d'images de documents anciens. Thèse de doctorat, Université Bordeaux 1
http://grenet.drimm.ubordeaux1.fr/pdf/2004/TEREBES_ROMULUS_MIRCEA_2004.pdf

2. D. Tschumperlé – Régularisation d'Images Multivaluées par EDP et Applications. Thèse de doctorat, Université de Nice Sophia-Antipolis -http://tel.ccsd.cnrs.fr/documents/archives0/00/00/23/96/index.html
3. G. Aubert,P Kornprobst- Mathematical Problems in Image Processing, Partial Differential Equations and the Calculus of Variations, ISBN 978-0-387-44588-5, Springer, 2006.
4. J. Weickert- Anisotropic - Diffusion in Image Processing, B.G. Teubner Stuttgart, 1998
5. Support de cours http://ares.utcluj.ro/tapisv

TP5 La théorie de déformation des courbes fermées. La méthode fast marching: applications pour la segmentation des images

Objectifs du TP :

- etudier et implémenter des méthodes numériques pour la déformation des contours fermées.

5.1 Les méthodes "level set" et "fast marching"

5.1.1 La méthode level set

La méthode «level set » a été développe par Stanley Osher et James Sethian. Dans cette méthode, une courbe 2D *C(x,y)* (séparant par exemple deux régions) évolue suivant sa normale avec une vitesse F. Ce front *C* est considéré comme le niveau zéro d'une fonction ϕ 3D. L'idée de cette méthode "level set" est alors de déduire la propagation de *C(x,y)* a partir de la propagation de ϕ.

Soit un contour initial *C(x,y)*, courbe fermée de R^2 et *F* une fonction qui donne la vitesse de *C(x,y)* dans la direction normale. La méthode "level set" consiste a considérer *C* comme le niveau zéro d'une fonction ϕ appelée fonction "level set" construite d'habitude comme la fonction de distance signée d'une pixel a C(x,y) :

- si (x,y) $\in$C ϕ(x,y,t=0)=0

- sinon ϕ(x,y,t=0)=$\pm$d où d est la distance de x à C[1]

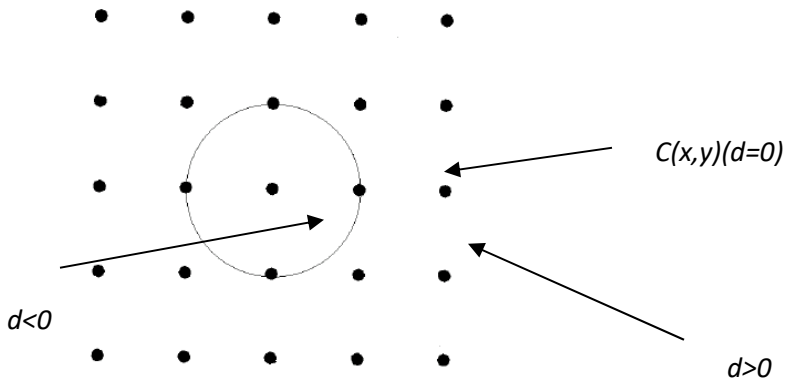


Fig.1. Courbe 2D et distance signée

[1] Le signe positif (négatif) est choisi si le point x est a l'extérieur (intérieur de la courbe)

Au lieu de faire évoluer en temps la courbe *C(x,y)*, les auteurs proposent de faire évoluer la fonction level set associée :

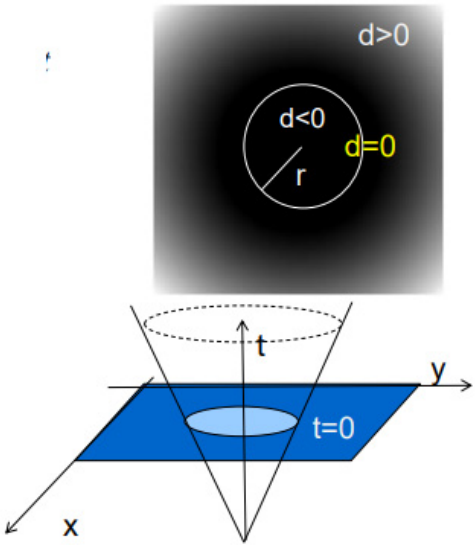


Fig.2. Courbe 2D et fonction level set associée

et ensuite de prendre la section transversale de ϕ pour déduire la position du contour dans le plan 2D. Sethian et Osher ont montré qu'une telle approche présente les avantages qu 'elle approxime de manière efficace la courbe même si elle subit de modifications topologiques.

L'équation d'évolution de la fonction level-set est facilement déduite on considérant l'ensemble des points de la courbe *(x(t, y(t)* a un certain instant t décrits par l'ensemble des vecteurs de position $\vec{C}(t) = x(t)\,\vec{i} + y(t)\,\vec{j}$:

$$\phi[\vec{C}(t), t] = 0$$

$$\frac{\partial}{\partial t}(\phi[\vec{C}(t), t] = 0) -> \frac{\partial \phi}{\partial t} + \nabla\phi \cdot \frac{\partial \vec{C}(t)}{\partial t} = 0$$

En considérant que la forme se déforme dans la direction de la normale $\vec{N}(t)$:

$$\frac{\partial \vec{C}(t)}{\partial t} = F\,\vec{N} = F\,\frac{\nabla\phi}{|\nabla\phi|}$$

on obtient l'équation de déformation de la fonction level set :

$$\frac{\partial \phi}{\partial t} + |\nabla \phi| F = 0$$

5.1.2 La méthode fast marching

Dans le cas particulier quand la vitesse F est soit positive soit négative pour tous les points de la courbe on obtient un cas particulier de la méthode level set nommée « fast marching ». Au lieu de déformer le contours initial on calcule le temps d'arrivée de ce front *T(x.y)* dans le pixel des coordonnées *(x.y)* :

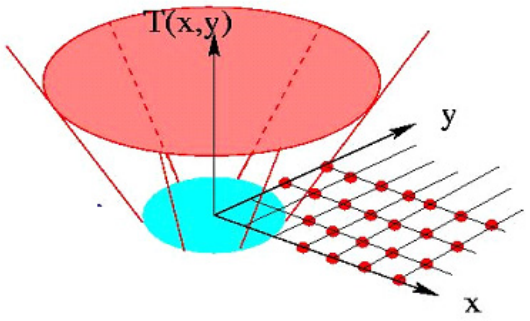


Fig.3. Le principe de la méthode fast marching (*source [5]*)

Pour tous les points du contour initial *T(x,y)*=0. La fonction *T(x,y)* doit également respecter la condition :

$$F|\nabla T| = 1$$

Une fois les valeurs de T calculées la courbe a un instant t est déduite immédiatement en posant :

$$C(t) = \{(x, y) | T(x, y) = t\}$$

5.1.3 Schéma numérique pour la méthode fast marching

Une approximation judicieuse du gradient de *T(x,y)* est nécessaire pour résoudre le problème. Le schéma proposé par Sethian consiste en :

$$\max[D_x^+(T_{ij}), -D_x^-(T_{i,j})]^2 + \max[D_y^+(T_{ij}), -D_y^-(T_{i,j})]^2 = \frac{1}{F_{ij}^{\ 2}}$$

Une classification des pixels est introduite pour permettre le calcul de $T_{i,j}$:

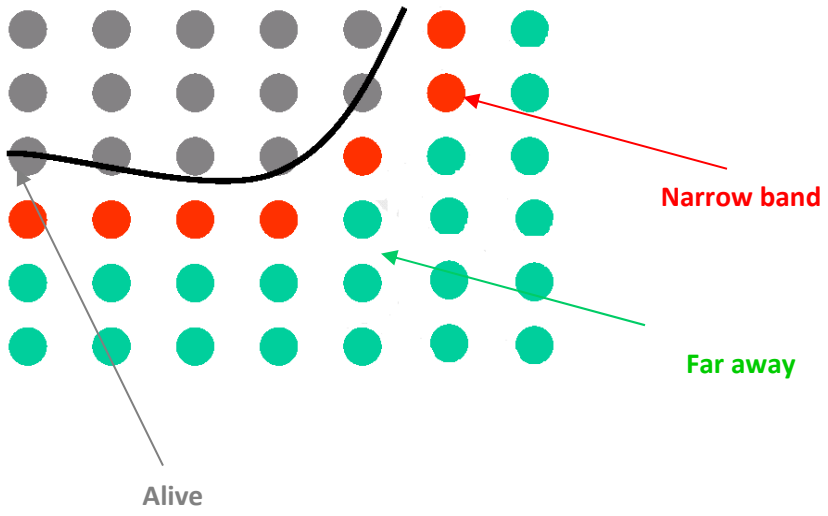


Fig.4. Classification des pixels dans la méthode fast marching

Les pixels dit ***alive*** sont des pixels dont on connaît la valeur de $T_{i,j}$. Les pixels dits ***narrow band*** sont des pixels dont on dispose une estimation de $T_{i,j}$ mais ces valeurs peuvent changer au fur et a mesure que la courbe évolue en temps. Pour les pixels ***far away*** on ne dispose pas d'une estimation pour $T_{i,j}$.

L'algorithme de fast marching est une procédure itérative qui consiste dans les étapes suivantes :

1) Initialisation:

- tous les pixels du contour sont libellés alive

- tous les pixels voisins au pixels placés sur le contour sont libellés narrow band ($T_{i,j}=(dx^2+ dy^2)^{1/2}/F_{i,j}$)

- tous les autres pixels sont libellés far away

2) Soit (i_{min},j_{min}) *le point narrow band avec T de valeur minimale*

3) (i_{min},j_{min}) *est libellé alive et supprimé de la liste narrow band*

4) Tous les voisins de (i_{min},j_{min}):(i_{min}-1,j_{min}), (i_{min}+1,j_{min}), (i_{min},j_{min}-1), (i_{min},j_{min}+1) *qui sont far away sont placées dans la liste narrow band et leurs valeurs T sont calculées*

5) La procédure est itérée en partant de 2 jusq'au moment que la liste narrow band est vide.

La complexité calculatoire peut être réduite si on remarque que l'information se propage toujours en partant des plus petites valeurs de *T* aux plus grandes.

Des implémentations pratiques de l'algorithme utilisent une structure de type « min-heap » pour les pixels classifiées *narrow band* (une arbre binaire ayant la

propriété que la valeur associée a un nœud parent n'est plus grande que celles associées au nœuds enfants) :

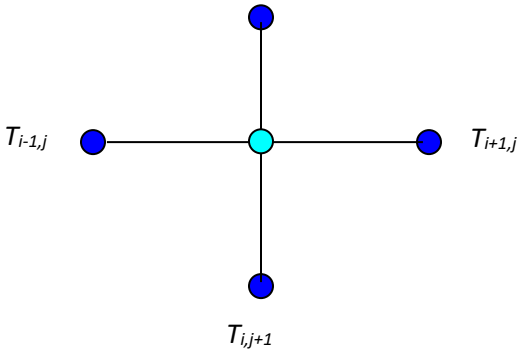
Si on dénote par ***trial*** la liste associée au pixels *narrow band*, en termes de pseudo-code, la description de l'algorithme fast-marching est :

Initialisation des toutes les valeurs au infini
alive ← *tous les pixels dont les valeurs sont connues*
trial ← *tous les pixels adjacents au pixels alive*
pendant que *(la liste narrow band n'est pas vide)*
nœud A ← *nœud racine de la structure min heap trial*
ajouter A a alive
pour *tous les voisins B de A*
si *T(B) est infini*
ajouter B a la liste trial
sortez la liste trial
T(B)=SolutionEquation(B)
fin si
fin pour
fin pendant

En pratique les points *alive* sont des pixels décrivant une courbe initiale qui se déformera en temps ou sont des germes (seeds en anglais) placées dans l'image originale. En ce qui concerne la résolution de l'équation qui décrit le modèle numérique, il faut remarquer tout d'abord qu'elle ne fait intervenir que les pixels voisins (4-voisinage) du pixel courrant :

$$\max[D_x^-(T_{ij}),-D_x^+(T_{i,j})]^2+\max[D_y^-(T_{ij}),-D_y^+(T_{i,j})]^2=$$

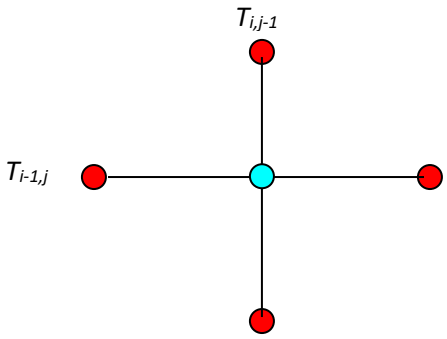
$$\max[T_{i,j}-T_{i-1,j},T_{i,j}-T_{i+1,j}]^2+\max[T_{i,j}-T_{i,j-1},T_{i,j}-T_{i,j+1}]^2=\frac{1}{F_{i,j}^2}$$



Le problème revient donc a calculer la valeur inconnue $T_{i,j}$ en partant des valeurs des voisins connues ou inconnues, en fonction de la classification des pixels voisins.

Exemple :

- considérons que tous les pixels sont narrow-band est que les valeurs calculées sont ordonnées de la manière suivante $T_{i-1,j} < T_{i+1,j}$ *et* $T_{i,j+1} < T_{i,j-1}$ **:**



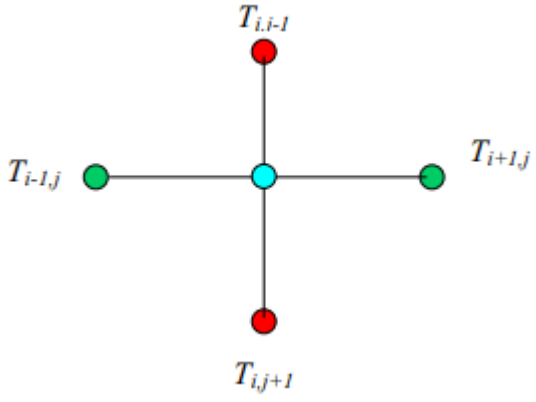
$$\max[D_x^-(T_{ij}),-D_x^+(T_{i,j})]^2 + \max[D_y^-(T_{ij}),-D_y^+(T_{i,j})]^2 = \frac{1}{F_{i,j}^2} \Rightarrow$$

$$(T_{i,j} - T_{i-1,j})^2 + (T_{i,j} - T_{i,j+1})^2 = \frac{1}{F_{i,j}^2}$$

La valeur de $T_{i,j}$ correspondante est la plus grande solution de l'équation quadratique précédente.

Exemple :

- considérons que tous les pixels horizontaux sont far away et que les voisins verticaux sont narrow band.



Car le premier terme s'annule, l'équation s'écrit :

$$\max(T_{i,j} - T_{i,j+1}, T_{i,j} - T_{i,j-1})^2 = \frac{1}{F_{i,j}^2}$$

et la solution est :

$$T_{i,j} = \min(T_{i,j+1}, T_{i,j-1}) + \frac{1}{F_{i,j}}$$

Une solution similaire existe pour des pixels horizontaux de type narrow band et pixels verticaux de type far away ; elle est donnée par :

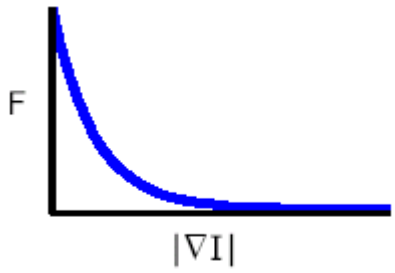
$$T_{i,j} = \min(T_{i-1,j}, T_{i+1,j}) + \frac{1}{F_{i,j}}$$

5.2 Application de la méthode fast marching pour la segmentation d' images

5.2.1 Introduction

Pour des applications de type segmentation des images la vitesse de déplacement en chaque pixel est établie comme la sortie d'une fonction de type Perona – Malik jouant le rôle d'un détecteur de contours :

$$F(x)_{i,j} = \exp^{-\alpha|\nabla G_\sigma * U|_{i,j}}$$



En partant d'un ensemble des germes prédéfinis par l'utilisateur ($T_{i,j}=0$) la méthode favorisera le déplacement a l 'intérieur de régions avec une vitesse quasi-constante ($F \cong 1$) et pénalisera les déformations a travers les contours ($F \rightarrow 0$).

En comparaison avec des autres approches la méthode permet l'extraction des objets singuliers (le cas d'un seul germe) ou multiples (pour plusieurs germes).

5.2.2 Implémentation de l'algorithme

1. Ouvrez la plateforme Microsoft Visual Studio et ensuite ouvrez la solution …\Compile_DLL.sln
2. Ajoutez le code suivant :

- dans le fichier attLibraryMenu.c :

```
void attLibraryMenu4()
{
        ATT_DESC_MENU menu4 =
                {
                        BEGIN_MENU
                        "Fast marching",                                IDM_FM,
                        END_MENU
                };
                attLibraryAddMenu("MENU_2D,Segmentation", menu4);
}
```

- dans le fichier AttLibraryMenu.h :

```
…
#define IDM_FM                  31 //valeur unique
…
```

- dans le fichier attLibraryMain.c dans le corps de la fonction GetMenuParametres:

```
…
attLibraryMenu4();
…
```

3. Ajoutez un fichier nommé TP5.c au projet courrant (Project->Add to project->New-> C++ Source File)
4. Cliquez sur le fichier qui vient d'être crée et ajoutez le code suivant dans le fichier TP5.c :

```
void fast_marching( double *entree, double *lis, double * sortie, long larg, long haut, int xclick, int yclick, int xclick1, int yclick1)
{
        int ux,uy;
        double * tij,* tijtemp, *tijlis;
        int py=yclick, nx=xclick;
        double somme=0;
        long m_cx;
```

```
long m_cy;
long coord_min;
long * classif;
int i=0;
long cont;
long * trial, * contour;
int taille=2, taille_cont=0;
double dist;
long coord_chemin;
double min1, min2;
tij=(double *)calloc((larg)*(haut), sizeof (double));
tijlis=(double *)calloc((larg)*(haut), sizeof (double));
tijtemp=(double *)calloc((larg)*(haut), sizeof (double));
classif=(long *)calloc((larg+1)*(haut+1), sizeof (long));
 /*classification des pixels 0- alive, 1 narrow band, 2 far –away*/
trial=(long *)calloc((larg)*(haut), sizeof (long));

for (uy=0;uy<haut;uy++)
for (ux=0;ux<larg;ux++){
        tijlis[i]=lis[i];
        classif[i]=2;
        i++;
        }
for (uy=0;uy<haut;uy++)
        for (ux=0;ux<larg;ux++)
                tij[ux+uy*larg]=10000000;

trial[0]=nx+py*larg;
trial[1]=xclick1+yclick1*larg;
init(tij,tijlis,classif,larg,haut, nx, py);
init(tij,tijlis,classif,larg,haut, xclick1, yclick1);
do
```

```
{
        taille--;
        coord_min=trial[0];
        trial++;
        classif[coord_min]=0;
                m_cy=coord_min/larg;
        m_cx=coord_min-m_cy*larg;
        classif[m_cx+m_cy*larg]=0;
        if (m_cx>1 && m_cy>1 && m_cy<haut-1 && m_cx <larg-1 )
{
if (classif[m_cx-1+m_cy*larg]!=0)
{
        if (classif[m_cx-1+m_cy*larg]==2){
                classif[m_cx-1+m_cy*larg]=1;
                trial[taille++]=m_cx-1+m_cy*larg;
                calcul (tij,classif,tijlis,larg,haut,m_cx-1,m_cy);
                triere(trial,taille,tij,m_cx-1+m_cy*larg);
     }
}
else
        ;
if (classif[m_cx+1+m_cy*larg]!=0)
{
        if (classif[m_cx+1+m_cy*larg]==2){
                classif[m_cx+1+m_cy*larg]=1;
                trial[taille++]=m_cx+1+m_cy*larg;
                calcul (tij,classif,tijlis,larg,haut,m_cx+1,m_cy);
                triere(trial,taille,tij,m_cx+1+m_cy*larg);
}
}
else
;
```

```
if (classif[m_cx+(m_cy-1)*larg]!=0)
        {
                if (classif[m_cx+(m_cy-1)*larg]==2)
                {
                        classif[m_cx+(m_cy-1)*larg]=1;
                        trial[taille++]=m_cx+(m_cy-1)*larg;
                        calcul (tij,classif,tijlis,larg,haut,m_cx,m_cy-1);
                        triere(trial,taille,tij,m_cx+(m_cy-1)*larg);
                }
        }
        else
                ;
        if (classif[m_cx+(m_cy+1)*larg]!=0)
        {
                if (classif[m_cx+(m_cy+1)*larg]==2)
                {
                        classif[m_cx+(m_cy+1)*larg]=1;
                        trial[taille++]=m_cx+(m_cy+1)*larg;
                if (m_cx !=0 && m_cx !=larg && m_cy !=0 && m_cy+1 !=haut)
                        calcul (tij,classif,tijlis,larg,haut,m_cx,m_cy+1);
                        triere(trial,taille,tij,m_cx+(m_cy+1)*larg);
                }
        }
        }
        else
                ;
        }while(taille>0);
    for( i=0; i<larg*haut; i++)
        {
                sortie[i]=tij[i];
        }
}
```

```
void init(double *im1,double * im2, long * im3, long dimx,long dimy, long nx, long py)
{
        int ux, uy;
        int midx, midy;
        midx=nx;
        midy=py;
        im1[midx-1+midy*dimx]=pow(exp(0.25* im2[midx-1+midy*dimx]),1);
        im1[midx+1+midy*dimx]=pow(exp(0.25 *im2[midx+1+midy*dimx]),1);
        im1[midx+(midy+1)*dimx]=pow(exp(0.25*im2[midx+(midy+1)*dimx]),1);
        im1[midx+(midy-1)*dimx]=pow(exp(0.25 *im2[midx+(midy-1)*dimx]),1);
        im1[midx+midy*dimx]=0;
        im3[midx-1+midy*dimx]=2;
        im3[midx+1+midy*dimx]=2;
        im3[midx+(midy+1)*dimx]=2;
        im3[midx+(midy-1)*dimx]=2;
        im3[midx+(midy)*dimx]=0;
}
```

```
void calcul (double * tij,long * classif,double * lis, long dimx,long dimy,long m_x, long m_y)
{
        double mini,minj;
        double delta, potel, t1, t2;
                if (m_x-1>0 && m_x+1<dimx && m_y>0 && m_y<dimy)
                {
                t1=tij[m_x-1+m_y*dimx];
                t2=tij[m_x+1+m_y*dimx];
                mini=_min(t1,t2);
                }
                else
                        mini=10000000;
                if (m_y-1>0 && m_y+1<dimy && m_x>0 && m_x<dimx)
```

```
                {
                t1=tij[m_x+(m_y+1)*dimx];
                t2=tij[m_x+(m_y-1)*dimx];
                minj=_min(t1,t2);
                }
                else
                        minj=10000000;
                potel=pow(exp(0.25*lis[m_x+dimx*m_y]),1);
                if (potel <=fabs(mini-minj))
                        tij[m_x+(m_y)*dimx]=potel+_min(mini,minj);
                else
                {
                delta=2.0*potel*potel-(mini-minj)*(mini-minj);
                tij[m_x+(m_y)*dimx]=((mini+minj)+sqrt(delta))/2.0;
                }
}
```

```
double _min (double u, double v)
{
        if (u<v)
                return u;
        else
                return v;
}
```

```
void triere (long *suite,long longuer, double * valeur, int coord)
{
        int i=longuer-1, echange;
        if (coord>0)
        {
        while (suite[i]!=coord) i--;
        while (valeur [suite[i]]<valeur[suite[i-1]] && i-1>=0)
```

```
        {
        echange=suite[i];
        suite[i]=suite[i-1];
        suite[i-1]=echange;
        i--;
        }
        }
}
```

5. Ajoutez le code suivant dans le fichier attLibraryRun.c :

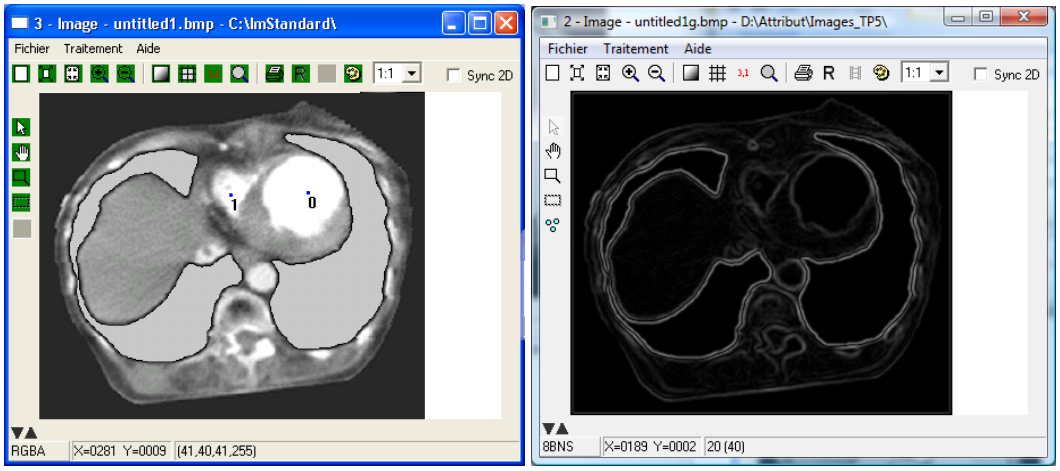
```
case    IDM_FM :
        {
          long  larg_entree,  haut_entree,  larg_sortie,  haut_sortie,  *  souris_x,  *
souris_y, nb_points_souris, id_entree, id_sortie;
        double * ent_f,* ent_lis, * sor_f;
         if ( mode == MODE_AIDE )
                {         break;  }
        if ( attDataSetNombre(numero_tache, IMG_ENTREE, 2) == FALSE ) break;
        id_entree = attDataGetIdByCount(numero_tache, IMG_ENTREE, 1);
        // Obtention du nombre de points sélectionnés à la souris
        nb_points_souris = attDataGetClickNumber(id_entree);
        if ( nb_points_souris )
               {
                 souris_x = (long *)attDataGetClickX(id_entree);
               souris_y = (long *)attDataGetClickY(id_entree);
               larg_entree = attDataGetLargeur(id_entree);
               haut_entree = attDataGetHauteur(id_entree);
               ent_f = (double  *)attDataGetData(id_entree, TYPE_DOUBLE);
               id_entree = attDataGetIdByCount(numero_tache, IMG_ENTREE, 2);
               ent_lis = (double  *)attDataGetData(id_entree, TYPE_DOUBLE);
               if ( ent_f == NULL ) break;
               attDataSetImageNombreTypeTaille(numero_tache, 1,
TYPE_DOUBLE, larg_entree, haut_entree);
```

```
                larg_sortie=larg_entree;
                haut_sortie=haut_entree;
                attDataSetImageNombreTypeTaille(numero_tache, 1,
TYPE_DOUBLE, larg_sortie, haut_sortie);
                id_sortie = attDataGetIdByCount(numero_tache, IMG_SORTIE, 1);
                sor_f = (double  *)attDataGetPtr(id_sortie);
                fast_marching( ent_f,ent_lis, sor_f, larg_entree, haut_entree,
souris_x[0], souris_y[0],souris_x[1], souris_y[1]);
                attDataFree(souris_x); attDataFree(souris_y);
                attDataFree(ent_f);attDataFree(ent_lis);
                }
        }
                break;
```

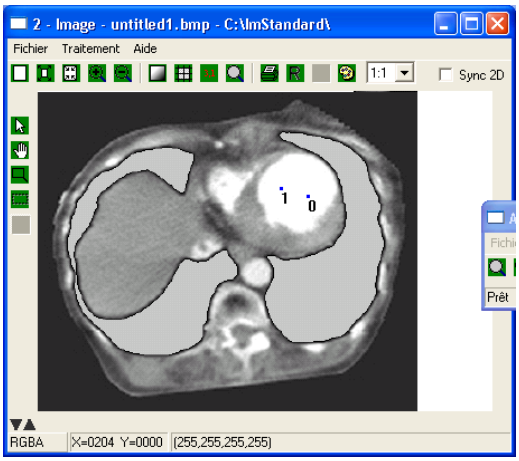
5.2.3 Exercices

1. Analysez le code et expliquez le rôle de chaque fonction

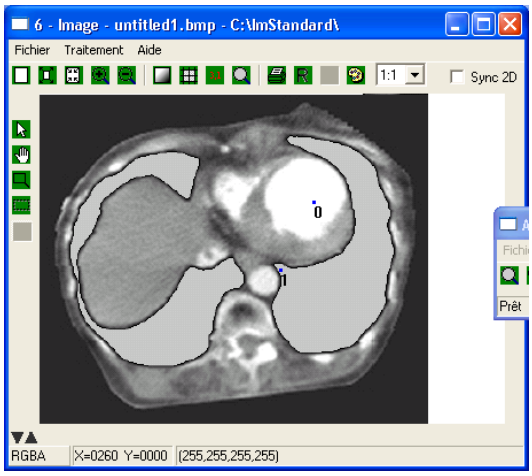
2. Ouvrez les images *untitled1.bmp* et *untitled1g.bmp* placées dans le répertoire …\ImagesStandard. Choisissez deux germes comme dans la figure suivante :



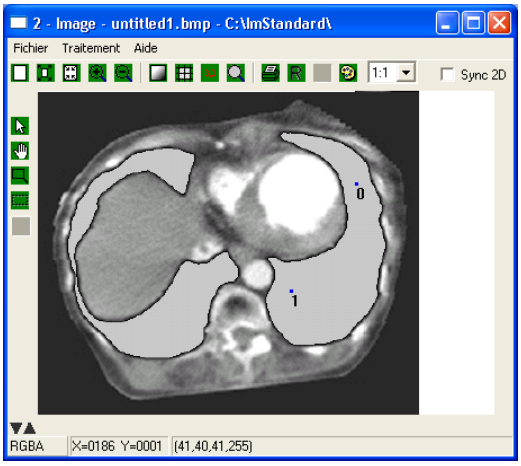
1. En vous basant sur les explications théoriques identifiez le rôle de la deuxième image.
2. Lancez l'application est observez les résultats
3. Chargez la bibliothèque « outils.dll » seuillez l'image de sortie en utilisant un seuil bas de 0 et un seuil haut de 100. Observez les résultats.
4. Répétez 2,3 et 4 en choisissant les deux germes à l'intérieur d'un seul objet :



5. Quel est le nombre des objets segmentés ?
6. Choisissez les deux germes comme dans la figure suivante :



7. Segmentez l'image en utilisant l'algorithme fast marching et seuillez le résultat de manière similaire a 4. Analysez le résultat et commentez-le.
8. Répétez 8 et plaçant les deux germes comme dans la figure suivante :



9. Modifier le code afin qu'il permette de placer plusieurs germes.
10. Placez les germes afin qu'ils permettent la segmentation de tous les objets de l'image.

Références bibliographiques

1. G. Aubert,P Kornprobst, Mathematical Problems in Image Processing, Partial Differential Equations and the Calculus of Variations, ISBN 978-0-387-44588-5, Springer, 2006.
2. S. Osher, N. Paragios - Geometric Level Set Methods in Imaging Vision and Graphics, Springer Science Bussiness. 2006.
4. J.A. Sethian Level set methods and Fast marching methods - https://math.berkeley.edu/~sethian/2006/level_set.html [01.06.2019]
5. Support de cours http://ares.utcluj.ro/tapisv.

TP6 Techniques d'inpainting

Objectifs du TP :

- implanter des modèles numériques pour les principales approches d'inpainting ;
- identifier les propriétés, avantages et désavantages de chaque approche .

6.1 Introduction

Une technique d'inpainting est essentiellement une technique d'interpolation intelligente qui a pour but de déterminer, de manière automatique, le niveau de gris ou la couleur des pixels considérés comme manquants dans une image, c'est à dire, dont on ne connait pas les valeurs.

Par rapport aux approches de restauration d'images, la reconstruction d'une image en utilisant des techniques d'inpainting s'avère plus difficile car l'information décrivant, localement, un objet, un contour, un détail ou une région homogène est inexistante.

Le point commun pour toutes les méthodes d'inpainting est qu'elles nécessitent l'intervention de l'utilisateur pour la définition d'un masque couvrant les régions à inpainter. L'initialisation de ce masque se fait avec une valeur prédéfinie et la reconstruction à lieu seulement dans l'intérieur de cette région.

Même dans l'absence d'une cadre théorique unificateur, la plupart des techniques sont fondées sur la continuation des lignes isophotes (ligne de même niveau) à l'intérieur du masque tout en assurant la continuité de la dérivée première pour un aspect plus naturel.

6.2 Exercices

1. Téléchargez le fichier tp6.c placé sur la page web du cours (http://ares.utcluj.ro/tapisv.html)
2. Le fichier propose des implantations numériques pour 2 approches présentées au cours : l'approche d'Oliveira et l'approche de Do.
   Les 2 fonctions qui implantent ces méthodes ont le prototype C :

   ```
   void Inpainting_Oliveira( double * entree, double * masque, double * sortie, long nbiter, long largeur, long hauteur)

   void Inpainting_Do(double * entree, double *masque, double * sortie,double sigma, double ro, long largeur,long hauteur,double dt, long nb_iter, long id_temp)
   ```

   Les 2 fonctions prennent pour paramètres:

entrée – l'image d'entrée dans laquelle la masque d'inpainting a été en préalable initialisée en marquant les régions a inpainter avec la valeur réservée 255

masque – une copie de l'image d'entrée utilisée dans le code pour limiter l'action sur les régions marquées

sortie - l'image de sortie

Les autres paramètres sont particuliers pour chaque méthode et ils imposent le comportement désiré.

En vous appuyant sur les aspects théoriques présentés en cours, introduisez des commentaires dans le code pour expliquer le rôle de chaque séquence de code décrivant les deux fonctions.

3. Ouvrez la plateforme N'D et rajoutez tous les éléments nécessaires qui permettront le traitement paramétré d'une image avec chaque méthode.
4. Téléchargez l'image *peppers_inpaint.bmp* et traitez-la avec la méthode d'Oliveira pour 50 itérations. Quels sont les effets que vous observez dans l'image ?
5. Traitez la même image avec la méthode de Do pour 2500 itérations. Analysez de manière comparative les résultats obtenus.
6. Choisissez une image et définissez votre propre masque d'inpainting et ensuite traitez l'image avec les deux approches. Quelle est l'influence de la taille du masque sur la qualité des résultats obtenus ?

Références bibliographiques

1. M.M. Oliveira, B. Bowen, R. McKenna et Y-S Chang – Fast Digital Inpainting, Proceedings of the IASTED International Conference on Visualization, Imaging and Image Processing (VIIP 2001), Marbella, Spain, September 3-5, 2001.

2. V.Do, G.Lebrun, L. Malapert,, C. Smet, D. Tschumperle- Inpainting d'Images Couleurs par Lissage Anisotrope et Synthèse de Textures. Reconnaissance des Formes et Intelligence Artificielle (RFIA'06), 2006, Tours, France, pp.35–42

5. Support de cours http://ares.utcluj.ro/tapisv.

TP7 Débruitage par patchs. Filtres à moyennes non locales. Filtrage collaboratif

Objectifs du TP :

- étude des méthodes récentes de filtrage par patch (les filtres a moyennes non-locales et le filtrage collaboratif).

7.1 Filtres à moyennes non locales

D'autres approches existent pour supprimer l'effet du bruit tout en gardant l'information présente sur les contours. La plupart de ces approches utilisent la même idée: faire des moyennes sur les pixels similaires. Les approches différent par la modalité dont cette similarité est définie.

Les filtres à moyennes non-locales (Non-Local Means en anglais – NLM) ont été proposés la première fois par Buades en 2005 [1]. Dans cette classe d'approches la similarité entre pixels est quantifiée en considérant aussi les pixels voisins les entourant (fig.1).

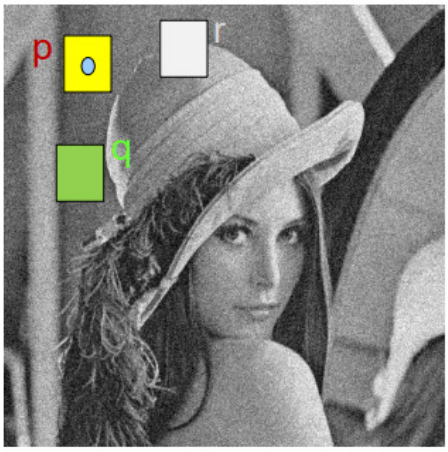


Fig. 1. Patchs centrés

La valeur calculée dans le pixel p prend en compte la similarité entre son voisinage (patch – en jaune dans la figure) et les patch associés à d'autres pixels. Dans l'exemple donnée précédemment, la contribution du pixel q dans le calcul de la valeur filtrée du pixel p est beaucoup plus importante que celle du pixel r. L'algorithme complet NLM est le suivant :

$$NL(p)(i) = \sum_{j \in I} w(i,j)v(j), \qquad (1)$$

dont les poids $\{w(i,j)\}$ dépendent de la similitude entre les pixels i et j et sont usuellement calculées en utilisant la fonction de distance euclidienne entre les patch

comprenant les pixels i et j $v(N_i) = (v(j), j \in N_i)$:

$$w(i,j) = \frac{1}{Z(i)} e^{-\frac{\left\|v(N_i)-v(N_j)\right\|^2_{2,a}}{h^2}} \quad (2)$$

avec

$$Z(i) = \sum_j e^{-\frac{\left\|v(N_i)-v(N_j)\right\|^2_{2,a}}{h^2}} \quad (3)$$

un facteur de normalisation.

Les résultats en filtrage obtenus par la méthode NLM sont remarquables et la méthode peut être testée en ligne :
http://demo.ipol.im/demo/bcm_non_local_means_denoising/

Malgré ces résultats, la méthode présente quelques problèmes dans le zone de forte contraste ou pour des images fortement bruitées. Dans ces situations la méthode ne peut pas détecter des patch voisins avec des similitudes suffisamment grandes pour permettre un lissage efficace. Plusiuers auteurs ont adressé ces aspect en proposant des approches utilisant des fonctions de quantification des similitudes optimisées ou des patchs non carrés, orientés avec agrégation des résultats ([2], [3]).

7.2 Filtrage collaboratif

Par rapport aux approches de type NLM, les filtres qui rentrent dans cette catégorie permettent meilleure exploitation de la redondance de l'information, permettant la collaboration entre les patchs pour le de bruitage. L'idée a été introduite par Dabov [4] et opère à partir d'un regroupement en 3D des patchs 2D similaires sur lequel une méthode de filtrage 3D utilisant l'information commune est appliquée. Le principe est illustré dans fig.2.

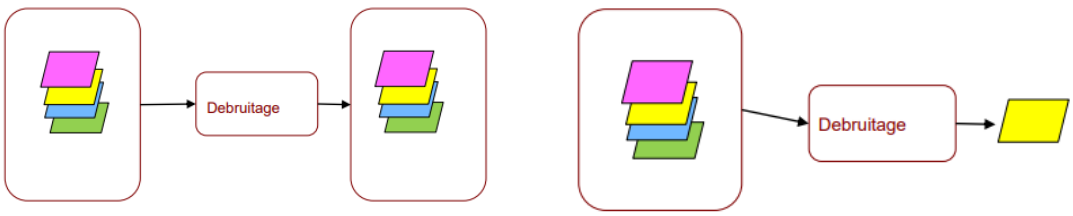


Fig. 2. Filtrage NLM (gauche) et BM3D (droite).

L'operateur résultant (**BM3D – Block Matching 3D**) utilise un chaine de traitement impliquant plusieurs opérations pour le calcul de la valeur de la fonction luminance

associée a un pixel : **regroupement** (les patch similaires d'une image 2D sont empilés dans un tableau 3D), **debruitage collaboratif** (chaque bloc d'un groupe collabore pour le filtrage de tous les autres du groupe et vice versa), **agrégation** (les patch peuvent se chevaucher donc plusieurs estimations pour chaque pixel existent; ces valeurs sont combinées par une moyenne pondéré ayant des poids inverse proportionnelle avec la variance du patch ayant produit l'estimation). La procédure produit une version initiale débruitée de l'image. Celle-ci est utilisée dans une deuxième itération pour le calcul de la version finale. Nous nous referons a la publication initiale pour l'algorithme complet. L'algorithme ne peut pas être testé en ligne ; néanmoins les auteurs incluent sur une page web dédiée une implémentation de référence.

7.3 Exercices

1. Téléchargez les implémentations Matlab des approches de type NLM optimisées proposées par B. Goossens et al. (https://quasar.ugent.be/bgoossen/download_nlmeans/) et, respectivement, C. Deledalle et al. (http://josephsalmon.eu/code/index_codes.php?page=NLMSAP).
2. Construisez un tableau pour noter les mesures quantitatives de type PSNR et SSIM obtenues utilisant la méthode NLM de Buades et celles indiquées à 1) pour le lissage de trois images standard (*Lena, Barbara* et *Cameraman)*, bruitées avec plusieurs niveau de bruit Gaussien (écart type σ=10, 30 et 50).
3. Complétez le tableau tout en sauvegardant aussi les images filtrées. Quels-sont les effets que vous observez au fur et à mesure que le niveau de bruit augmente ?
4. Analysez les résultats visuelles et indiquez quel est l'aspect théorique qui le justifie.
5. Téléchargez l'implémentation Matlab de l'approche BM3D (http://www.cs.tut.fi/~foi/GCF-BM3D/)
6. Répétez l'analyse effectuée à 3) et 4).

Références bibliographiques

[1] A. Buades, B. Coll, J.- M. Morel, A non-local algorithm for image denoising IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2(7):60- 65 vol. 2, 2005.

[2] C. -A. Deledalle, V. Duval, J. Salmon, Non-Local Methods with Shape-Adaptive Patches (NLM-SAP). Journal of Mathematical Imaging and Vision, Springer Verlag, 2012, 43 (2), pp.103-120.

[3] B. Goossens, H. Q. Luong, A. Pižurica, W. Philips, An improved Non-Local Means algorithm for image denoising, 2008 International Workshop on Local and Non-Local Approximation in Image Processing (LNLA2008), pp. 14 pages, 2008.

[4] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, Image denoising by sparse 3D transform-domain collaborative filtering," IEEE Trans. Image Process., vol. 16, no. 8, pp. 2080-2095, August 2007.

[5] Support de cours http://ares.utcluj.ro/tapisv