Annexe 1 Schémas numériques utilisés

Notations :

Div $(w_1,w_2)=1/h*(I_{x,-}w_1 + I_{y,-}w_2)$

$$\nabla w = 1/h(I_{x,+}w, I_{y,+}w)$$

Avec $I_{x,+}w = w_{i+1,j} - w_{i,j} = Is$;

$I_{x,-}w = w_{i,j} - w_{i-1,j} = -In$;

$I_{y,+}w = w_{i,j+1} - w_{i,j} = Ie$;

$I_{y,-}w = w_{i,j} - w_{i,j-1} = -Iw$;

Equation de diffusion linéaire:

$$\frac{\partial u}{\partial t} = div(\nabla(u(x,y,t)) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

Le schéma numérique correspondant est:

$$\frac{u^{t+1} - u^t}{\Delta t} = \frac{u_{i+1,j}^t - 2u_{i,j}^t + u_{i-1,j}^t}{h^2} + \frac{u_{i,j+1}^t - 2u_{i,j}^t + u_{i,j-1}^t}{h^2} = \frac{I_n + I_s + I_e + I_w}{h^2}$$

Equations de diffusion non linéaire :

$\frac{\partial u}{\partial t} = div(c(|\nabla u(x,y,t)|)\nabla u(x,y,t))$

Dans chaque direction, le gradient est multiplié par une fonction qui dépend de la valeur du gradient dans cette direction.

L'expression du gradient peu ainsi s'écrire :

$$\nabla w = 1/h(c(|I_{x,+}u|) * \ I_{x,+}u, c(|I_{y,+}u|) * I_{y,+}u)$$

On applique ensuite la divergence à ce gradient :

$I_{x,-}w_1 = I_{x,-}\ [c(|I_{x,+}u|) * \ I_{x,+}u]\ = c(I_n)*I_n+c(I_s)*I_s$

D'où l'expression finale du schéma numérique:

$$\frac{u^{t+1} - u^t}{\Delta t} = \frac{c(|I_n|) * I_n + c(|I_s|) * I_s + c(|I_e|) * I_e + c(|I_w|) * I_w}{h^2}$$

Equation de dilatation :

$$\frac{\partial u}{\partial t} = +|\nabla u|$$

Le schéma numérique est celui utilisé dans [7] :

$$\frac{u^{t+1} - u^t}{\Delta t} = +\frac{\sqrt{min^2(0,-In) + max^2(0,Is) + min^2(0,-Iw) + max^2(0,Ie)}}{h^2}$$

Equation d'érosion

$$\frac{\partial u}{\partial t} = -|\nabla u|$$

$$\frac{u^{t+1} - u^t}{\Delta t} = +\frac{\sqrt{max^2(0,-In) + min^2(0,Is) + max^2(0,-Iw) + min^2(0,Ie)}}{h^2}$$

Equation de diffusion selon les lignes de niveau :

$$\frac{\partial u}{\partial t} = t|\nabla u| div\left(\frac{\nabla u}{|\nabla u|}\right)$$

Pour le calcul de $|\nabla u|$ nous utilisons le schéma utilisé pour l'érosion et la dilatation :
G = $|\nabla u| = \sqrt{max^2(0,-In) + min^2(0,Is) + max^2(0,-Iw) + min^2(0,Ie)}$

Puis :

$$\frac{u^{t+1} - u^t}{\Delta t} = \frac{1}{h^2} tG\left(\frac{In}{abs(In) + \varepsilon} + \frac{Is}{abs(Is) + \varepsilon} + \frac{Iw}{abs(Iw) + \varepsilon} + \frac{Ie}{abs(Ie) + \varepsilon}\right)$$

Le terme $+\varepsilon$ est ajouté pour éviter une division par zéro dans l'algorithme.

Annexe 2 Code Matlab (diffusion Linéaire, Malik et Perona)

```
function Jd=diffusion(I,method,N,dt,K,sigma2)

% Simulates N iterations of diffusion, parameters:
% J =  source image (2D gray-level matrix) for diffusion
% method =  'lin':  Linear diffusion (constant c=1).
%           'pm1': perona-malik, c=exp{-(|grad(J)|/K)^2} [PM90]
%           'pm2': perona-malik, c=1/{1+(|grad(J)|/K)^2} [PM90]
%
% K    edge threshold parameter
% N    number of iterations
% dt   time increment (0 < dt <= 0.25, default 0.2)
% sigma2 - if present, calculates gradients of diffusion coefficient
%          convolved with gaussian of var sigma2 (Catte et al [CLMC92])

if ~exist('N')
   N=1;
end
if ~exist('K')
   K=1;
end
if ~exist('dt')
   dt=0.2;
end
if ~exist('sigma2')
   sigma2=0;
end

[Ny,Nx]=size(I);
J=cell(N);
J{1}=I;

if (nargin<3)
   error('not enough arguments (at least 3 should be given)');
end

for i=1:N;
   % gaussian filter with kernel 5x5 (Catte et al)
   if (sigma2>0)
      Jo = J{i};   % save J original
      J{i}=gauss(J{i},5,sigma2);
   end

    % calculate gradient in all directions (N,S,E,W)
    In=[J{i}(1,:); J{i}(1:Ny-1,:)]-J{i};     % -I_(x,-) w=w_(i,j)-w_(i-1,j)
    Is=[J{i}(2:Ny,:); J{i}(Ny,:)]-J{i};      %I_(x,+) w=w_(i+1,j)-w_(i,j)
    Ie=[J{i}(:,2:Nx) J{i}(:,Nx)]-J{i};       %I_(y,+) w=w_(i,j+1)-w_(i,j)
    Iw=[J{i}(:,1) J{i}(:,1:Nx-1)]-J{i};       %-I_(y,-) w=w_(i,j)-w_(i,j-1)


    % calculate diffusion coefficients in all directions according to
method
   if (method == 'lin')
        Cn=K; Cs=K; Ce=K; Cw=K;
   elseif (method == 'pm1')
        Cn=exp(-(abs(In)/K).^2);
        Cs=exp(-(abs(Is)/K).^2);
```

```
        Ce=exp(-(abs(Ie)/K).^2);
        Cw=exp(-(abs(Iw)/K).^2);
   elseif (method == 'pm2')
        Cn=1./(1+(abs(In)/K).^2);
        Cs=1./(1+(abs(Is)/K).^2);
        Ce=1./(1+(abs(Ie)/K).^2);
        Cw=1./(1+(abs(Iw)/K).^2);
   else
        error(['Unknown method "' method '"']);
   end
   if (sigma2>0)   % calculate real gradients (not smoothed)
        In=[Jo(1,:); Jo(1:Ny-1,:)]-Jo;
        Is=[Jo(2:Ny,:); Jo(Ny,:)]-Jo;
        Ie=[Jo(:,2:Nx) Jo(:,Nx)]-Jo;
        Iw=[Jo(:,1) Jo(:,1:Nx-1)]-Jo;
      J{i}=Jo;
    end
   % Next Image J
   J{i+1}=J{i}+dt*(Cn.*In + Cs.*Is + Ce.*Ie + Cw.*Iw);
end;
Jd = J;
```

```
function Ig=gauss(I,ks,sigma2)
% Ig=gauss(I,ks,sigma2)
% ks - kernel size (odd number)
% sigma2 - variance of Gaussian
[Ny,Nx]=size(I);
hks=(ks-1)/2;  % half kernel size
if (Ny<ks)   % 1d convolution
    x=(-hks:hks);
    flt=exp(-(x.^2)/(2*sigma2));  % 1D gaussian
    flt=flt/sum(sum(flt));  % normalize
   % expand
   x0=mean(I(:,1:hks)); xn=mean(I(:,Nx-hks+1:Nx));
    eI=[x0*ones(Ny,ks) I xn*ones(Ny,ks)];
    Ig=conv(eI,flt);
    Ig=Ig(:,ks+hks+1:Nx+ks+hks);  % truncate tails of convolution
else
   %% 2-d convolution
    x=ones(ks,1)*(-hks:hks); y=x';
    flt=exp(-(x.^2+y.^2)/(2*sigma2));  % 2D gaussian
    flt=flt/sum(sum(flt));  % normalize
   % expand
   if (hks>1)
      xL=mean(I(:,1:hks)')'; xR=mean(I(:,Nx-hks+1:Nx)')';
   else
      xL=I(:,1); xR=I(:,Nx);
   end
   eI=[xL*ones(1,hks) I xR*ones(1,hks)];
   if (hks>1)
      xU=mean(eI(1:hks,:)); xD=mean(eI(Ny-hks+1:Ny,:));
   else
    xU=eI(1,:); xD=eI(Ny,:);
   end
    eI=[ones(hks,1)*xU; eI; ones(hks,1)*xD];
    Ig=conv2(eI,flt,'valid');
end
```

Annexe 3 Code Matlab (Dilatation et Erosion)

```
%Fonction qui calcul le résultat de l'EDP de dilatation
% pour 'iter' itérations avec un pas de temps 'dt'
function J=dilatation2(I,iter,dt)

J=double(I);
[Ny,Nx]=size(J);

   for i=1:iter;
      % Calcul du gradient dans toutes les directions (N,S,E,W)
      In=[J(1,:); J(1:Ny-1,:)]-J;
      Is=[J(2:Ny,:); J(Ny,:)]-J;
      Ie=[J(:,2:Nx) J(:,Nx)]-J;
      Iw=[J(:,1) J(:,1:Nx-1)]-J;

      %Calcul de la norme du gradient
      G=sqrt(power(min(0,-In),2)+power(max(0,Is),2)+power(min(0,-Iw),2)+power(max(0,Ie),2));

      J=J+dt*G;
      J=(J<256).*J;
   end

end
```

```
%Fonction qui calcul le résultat de l'EDP d'érosion
%pour 'iter' itérations avec un pas de temps 'dt'
function J=erosion2(I,iter,dt)

J=double(I);
[Ny,Nx]=size(J);

   for i=1:iter;
      % Calcul du gradient dans toutes les directions (N,S,E,W)
      In=[J(1,:); J(1:Ny-1,:)]-J;    % -I_(x,-) w=w_(i,j)-w_(i-1,j)
      Is=[J(2:Ny,:); J(Ny,:)]-J;     %I_(x,+) w=w_(i+1,j)-w_(i,j)
      Ie=[J(:,2:Nx) J(:,Nx)]-J;      %I_(y,+) w=w_(i,j+1)-w_(i,j)
      Iw=[J(:,1) J(:,1:Nx-1)]-J;      %-I_(y,-) w=w_(i,j)-w_(i,j-1)

      %Calcul de l'opposé de la norme du gradient
      G=sqrt(power(max(0,-In),2)+power(min(0,Is),2)+power(max(0,-Iw),2)+power(min(0,Ie),2));

      J=J-dt*G;
      J=(J>0).*J;
   end

end
```