Dérivation des images : calcul du gradient de Shen-Castan.

1• Objectif de ce travail.

Beaucoup d'algorithmes de traitement ou d'analyse d'images sont basés sur un opérateur de dérivation spatiale, ou plus exactement d'estimation du gradient d'illumination. L'extraction (ou le réhaussement) des contours, la détection de l'orientation d'une images sont deux exemples d'applications de cet algorithme. L'objet de ce travail est de réaliser un calcul de dérivation d'images basé sur l'estimateur de gradient de Shen-Castan.

2• Gradient d'une image.

Pour bien comprendre cette notion de gradient, il suffit de se représenter une image comme une carte de relief (figure 1).
Les contours des cubes correspondent à des *falaises* de l'image 3D, c'est à dire à des endroits où le relief 3D est très accidenté. Un estimateur de gradient calcule la pente en tout point de cette image 3D dans les deux directions x (lignes) et y (colonnes). La direction du vecteur obtenu est celle de la plus grande pente.

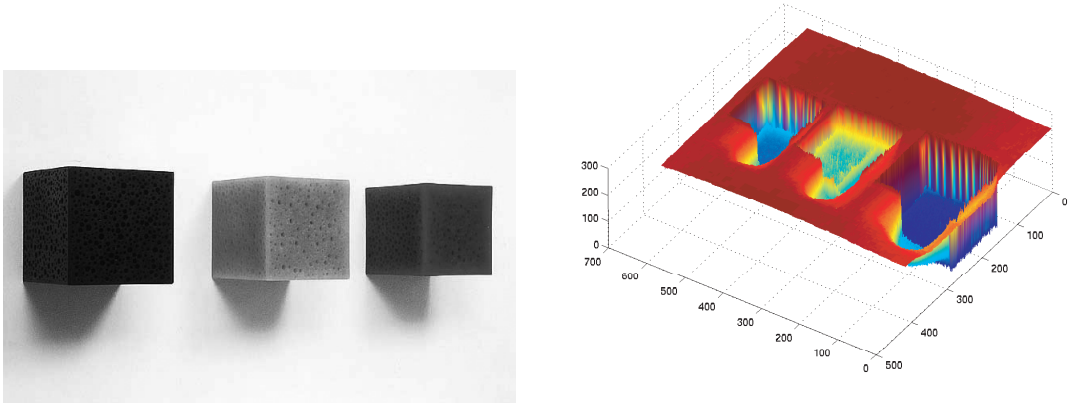


Figure 1 : image de cubes en éponge et leur visualisation en 3D.

3• Représentation 3D d'une image.

Chargez une des images qui vous sont proposées en saisissant :

```
MonImage=imread('CubesNG.jpg') ;
```

Affichez-la avec :

```
figure(1) ; image(MonImage) ;
```

puis affichez sa version 3D avec :

```
figure(2) ; surf(double(MonImage)) ; shading interp ;
```

Utilisez l'icone de rotation pour faire tourner l'image et essayer de relier ces deux représentations. Faîtes la relation entre contours et *falaises*.

4• Dérivation.

Soit $f : IR \rightarrow IR$ une fonction continue, f est dite dérivable au point $x \in IR$ si les trois

limites suivantes convergent vers la même valeur : $\lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$,

$\lim_{h \to 0} \frac{f(x)-f(x-h)}{h}$, $\lim_{h \to 0} \frac{f(x+h)-f(x-h)}{2h}$. Cette valeur notée $\frac{df}{dx}(x)$ ou aussi $f'(x)$ est la dérivée de f en x. Si f est dérivable en tout point elle est dite dérivable. Pour être dérivable, une fonction doit être continue ce qui exclut les fonction discrètes de la notion de dérivation.

En deux dimension, la notion de dérivée s'étend pour laisser place à la notion de gradient.

Soit $F : \mathbb{R}^2 \to \mathbb{R}^2$ une fonction continue, F est dite dérivable au point $(x, y) \in \mathbb{R}^2$ si F est dérivable par rapport à x, y étant fixé et dérivable par rapport à y, x étant fixé. Ces deux dérivées portent le nom de dérivées partielles de F par rapport à x (son premier argument) et y (son deuxième argument) et sont notées : $\frac{\partial F}{\partial x}(x, y)$ et $\frac{\partial F}{\partial y}(x, y)$.

Comme nous l'avons vu, une image n'est rien d'autre qu'une fonction 2D associant à chaque position sa valeur de luminance mesurée. Si l'image est échantillonnée (ce qui est le cas) les valeurs ne sont connues que pour des valeurs entières de x et y, elle n'est donc pas continue et donc ne peut pas être dérivée au sens classique.

5• Dérivation d'une image discrète.

5.1• Principe

La notion de dérivation pour une image discrète nécessite l'utilisation du schémas classique d'opération continues-discrètes représenté par la figure 2.

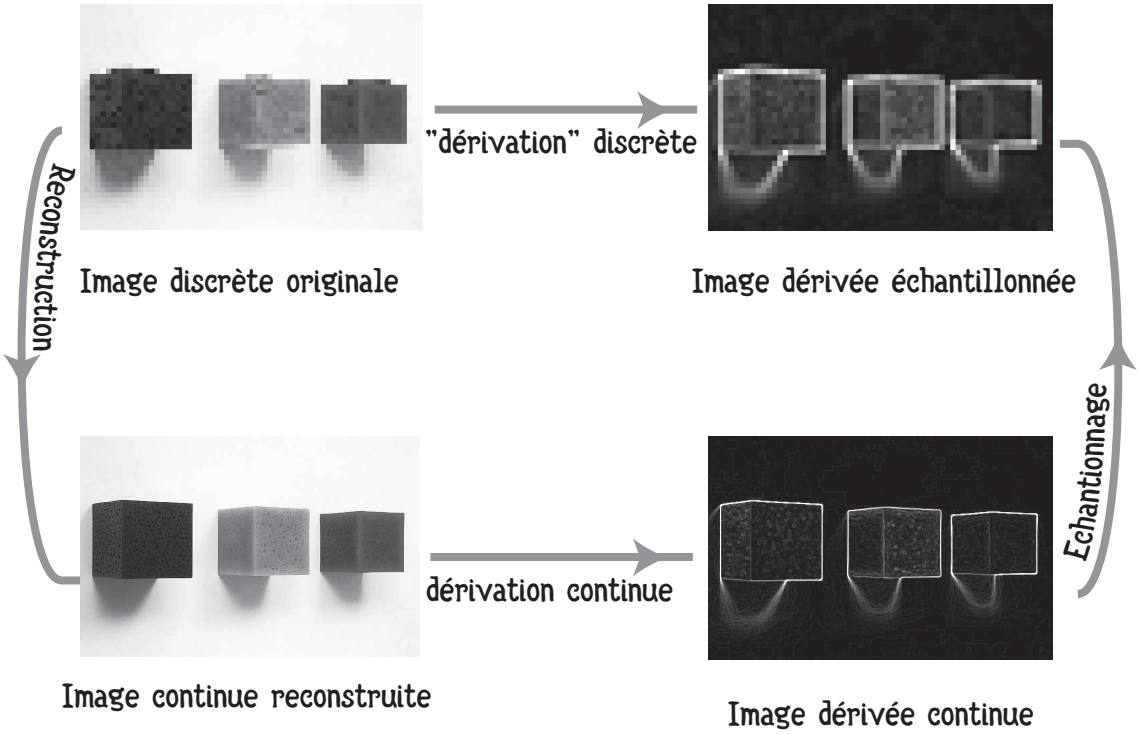


Figure 2 : dérivation continue / dérivation discrète

La "dérivation discrète" est alors réalisée par une opération de convolution discrète sur l'image discrète en échantillonnant la dérivée du noyau de reconstruction.

5.2• Un peu de théorie du filtrage.

C'est une approche simplifiée qui vous est proposée ici. Soit I une image échantillonnée. On peut reconstruire une image continue $\hat{I} : (x, y) \to \hat{I}(x, y)$ par convolution avec un noyau de recontruction $\varphi(x, y)$: $\hat{I}(x, y) = \sum_{i,j} I(i,j)\varphi(i-x, j-y) = (I \otimes \varphi)(x, y)$

($\otimes$ est l'opération de convolution).

Si $\hat{I}(i, j) = I(i, j)$ pour toutes les valeurs entières (i,j) alors le noyau de recontruction φ est appelé noyau d'interpolation (plus proche voisin, bi-cubique, bi-linéaire, ...). Si φ est un noyau d'interpolation, alors l'échantillonnage de $\hat{I}$ par un noyau de Dirac ∂ redonne l'image I de départ : $\forall (i, j) \in \mathbb{N}^2, \int_{x,y} \hat{I}(x, y)\partial(i-x, j-y) = I(i, j)$.

Sinon on dit que φ est un noyau d'approximation. On utilise un noyau d'approximation quand on a du "bruit" sur l'images, ce bruit étant dû à une variation statistiques des valeurs (problèmes de mesure de flux lumineux) mais aussi à la quantification (toutes les valeurs de I sont connues à ±0,5 près).

La propriété remarquable qui est utilisée ici est la suivante : la dérivée d'un produit de convolution est (au signe près) la même chose que la convolution avec la dérivée d'un des membres du produit de convolution.
Appliqué au problème qui nous intéresse, on a :

$$\frac{\partial}{\partial x}\hat{I}(x, y) = -\sum_{i,j} I(i,j)\frac{\partial \varphi}{\partial x}(i-x, j-y) = -\left(I \otimes \frac{\partial \varphi}{\partial x}\right)(x, y).$$

Et bien sûr la même relation est vraie pour la dérivation par rapport à y.
Pour finir, en se référant au schémas de la Figure 2, on obtient l'opérateur discret de "dérivation" équivalent en faisant un échantillonnage avec un noyau d'échantillonnage μ :

$$\left(\frac{\partial}{\partial x}\hat{I} \otimes \mu\right)(i, j) = -\left(\left(I \otimes \frac{\partial \varphi}{\partial x}\right) \otimes \mu\right)(i, j) = -\left(I \otimes \left(\frac{\partial \varphi}{\partial x} \otimes \mu\right)\right)(i, j).$$

Il suffit donc de convoluer I par l'échantillonnage de la dérivée du noyau φ.

On note souvent $I_x(i, j) = \left(\frac{\partial}{\partial x}\hat{I} \otimes \mu\right)(i, j)$ la "dérivée" de I par rapport à x.

De même on a $I_y(i, j) = \left(\frac{\partial}{\partial y}\hat{I} \otimes \mu\right)(i, j) = -\left(I \otimes \left(\frac{\partial \varphi}{\partial y} \otimes \mu\right)\right)(i, j)$.

5.3• Noyau séparable.

Un noyau séparable est un noyau bidimensionnel qui s'écrit comme le produit de deux noyaux monodimensionnels : $\varphi(x, y) = \varphi_x(x)\varphi_y(y)$. La convolution avec un noyau séparable est moins gourmande en calcul qu'une convolution avec un noyau non-séparable car elle se compose d'une convolution monodimensionnelle sur x suivi d'une convolution monodimensionnelle sur y.

Dans ce cas, la dérivée de φ s'écrit : $\frac{\partial\varphi}{\partial x}(x, y) = \frac{\partial}{\partial x}(\varphi_x(x)\varphi_y(y)) = \frac{\partial\varphi_x}{\partial x}(x)\varphi_y(y)$.

Du coup, l'expression de dérivation s'écrit :

$$I_x(i,j) = -\left(I \otimes \left(\frac{\partial\varphi_x}{\partial x} \otimes \mu\right) \otimes (\varphi_y \otimes \mu)\right)(i,j)$$

Et alors la dérivée de I par rapport à x (les colonnes) se compose d'un filtrage monodimentionnel de I avec la version échantillonnée de la dérivée $\frac{\partial\varphi_x}{\partial x}$ de φ_x suivit d'un autre avec φ_y.

5.4• Algorithme récursif.

Certains noyaux aboutissent à des écritures d'algorithmes récursifs. Un algorithme récursif est un algorithme itératif utilisant les valeurs calculées à l'itération i pour calculer la valeur à l'itération i+1.

6• Filtre de Shen-Castan.

6.1• Fonction d'approximation exponentielle.

Pour pouvoir calculer la dérivée d'une image numérique, il faut donc disposer d'un filtre de reconstruction dérivable. De même, il faut avoir un algorithme qui ne soit pas trop gourmand en temps de calcul.

Shen et Castan ont proposé d'utiliser un filtre exponentiel comme noyau d'approximation (ce n'est pas un noyau d'interpolation) et un échantillonnage de Dirac. Pourquoi pas une interpolation ? Parce que la dérivation est très sensible aux effets d'escalier induits conjointement par la quantification des valeur de l'image numérique et l'échantillonnage.

Ce filtre a pour expression :

$$\varphi(x, y) = \frac{1}{2 - e^{-\alpha}} e^{-\alpha(|x| + |y|)} \quad \text{avec } \alpha > 0.$$

Vous pouvez facilement visulaliser ce filtre :

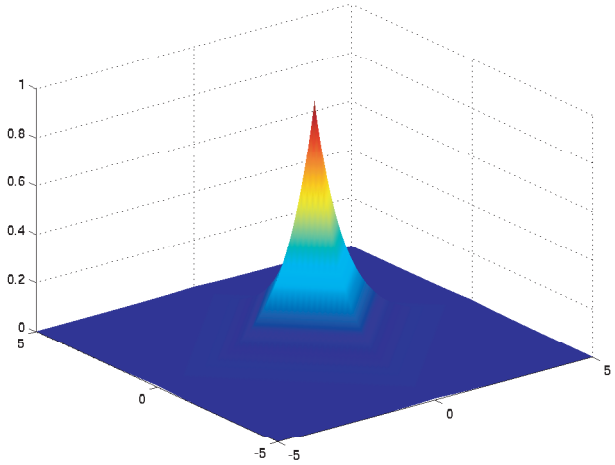


Figure 2 : Filtre d'approximation de Shen-Castan.

```
alpha = 2 ; x = -5:0.01:5 ; y = x ;
Filtre = exp(-alpha*abs(x)') * exp(-alpha*abs(x)) ;
figure(6) ; surf(x,y,Filtre) ; shading interp ;
```

Ce filtre a trois qualités importantes : son utilisation permet un calcul récursif, il est séparable et il est dérivable presque partout (sauf en 0, mais on s'en débrouille avec la théorie des distributions).

6.2• Algorithme de Shen-Castan.

Tout calcul fait, l'algorithme récursif de Shen-Castan permet de calculer une estimation du gradient de l'image en tout point de l'image discrète en utilisant les propriétés de séparabilité du filtre. Je donne ici la façon de calculer le gradient en x (colonne) que l'on va noter Gradient_x. Vous essayerez d'en déduire l'algorithme de calcul de Gradient_y la dérivée partielle de la fonction d'illumination suivant les lignes (y).
Pour calculer le gradient suivant les colonnes, il faut convoluer l'image avec un filtre qui lisse suivant les lignes puis convoluer le résultat de ce lissage avec la dérivée du filtre suivant les colonnes.

La première partie de l'algorithme est :

```
MonImage = imread('CubesNG.jpg') ; figure(1) ; clf ; image(MonImage) ;
axis image ; colormap(gray(255)) ;
alpha = 2 ; % donnez une valeur a alpha
a0 = exp(-alpha) ; a1 = 1 - a0 ;
[Nlin, Ncol] = size(MonImage) ;
MonImageFiltree = zeros(Nlin, Ncol) ;
MonImageDerivee = zeros(Nlin, Ncol) ;

% filtrage suivant lignes (ce qui veux dire qu'on filtre chaque colonne)
for col=1:Ncol
    Colonne = zeros(1,Nlin) ;
    Colonne(1:Nlin) = MonImage(1:Nlin,col) ;
    Colonne1 = Colonne ;
    Colonne2 = Colonne ;
    % filtrage dans le sens progressif
    for lin=2:Nlin
        Colonne1(lin) = ( a1 * Colonne1(lin-1) ) + ( a0 * Colonne(lin) ) ;
    end
    % filtrage dans le sens régressif
    for lin=Nlin-1:-1:1
        Colonne2(lin) = ( a1 * Colonne2(lin+1) ) + ( a0 * Colonne(lin) ) ;
    end
   MonImageFiltree(1:Nlin,col) = Colonne1(1:Nlin) + Colonne2(1:Nlin) - a0*Co-
lonne(1:Nlin) ;
end

% dérivation suivant les colonnes (ce qui veut dire qu'on filtre
% chaque lignes avec le filtre dérivateur)
for lin=1:Nlin
    Ligne = zeros(1,Ncol) ;
    Ligne(1:Ncol) = MonImageFiltree(lin,1:Ncol) ;
    Ligne1 = Ligne ;
    Ligne2 = Ligne ;
    for col=2:Ncol
        Ligne1(col) = ( a1 * Ligne1(col-1) ) + ( a0 * Ligne(col) ) ;
    end
    for col=Ncol-1:-1:1
        Ligne2(col) = ( a1 * Ligne2(col+1) ) + ( a0 * Ligne(col) ) ;
    end
    MonImageDerivee(lin,1:Ncol) = Ligne1(1:Ncol) - Ligne2(1:Ncol) ;
end

Gradient_X = -MonImageDerivee ;

% Visualisation
cmax = max(max(MonImageDerivee)) ;
cmin = min(min(MonImageDerivee)) ;
MonImageDeriveeVisu = 255 * ( MonImageDerivee - cmin ) / ( cmax - cmin ) ;
figure(3) ; image(uint8(floor(MonImageDeriveeVisu))) ;
axis image ; colormap(gray(255)) ;
```

Essayez cet algorithme pour différentes valeurs de alpha. Que constatez-vous ? Comment interprétez vous les images que vous voyez ?

Astuce pour faire le même algorithme suivant les lignes : vous pouvez transposer l'image (échanger les lignes et les colonnes), calculer la derivée suivant les colonnes puis re-transposer le résultat.

6.3• Norme du gradient

Un pixel de contour est un pixel pour lequel le vecteur gradient à une norme importante. Nous vous proposons ici de calculer et visualiser la norme d(u grandient de l'image. Cela s'obtient très simplement. Si $I_x(i,j)$ est l'image du grandient en x au point (i,j) et $I_y(i,j)$ est le gradient en y au même point (i,j), la norme du gradient en (i,j) est :

$$G(i,j) = \sqrt{(I_x(i,j))^2 + (I_y(i,j))^2}.$$

Pour nous, en numérique, on écrira :

```
NormeGradient = sqrt( Gradient_X.^2 + Gradient_Y.^2 ) ;
```

Affichez cette norme en niveaux de gris (attention à bien normaliser votre image). Calculez la norme du gradient pour différentes valeurs de alpha. Que remarquez-vous ? Essayez d'inverser l'image obtenue :

```
figure(7) ; image(255 - ImageDeNorme) ;
axis image ; colormap(gray(255)) ;
```

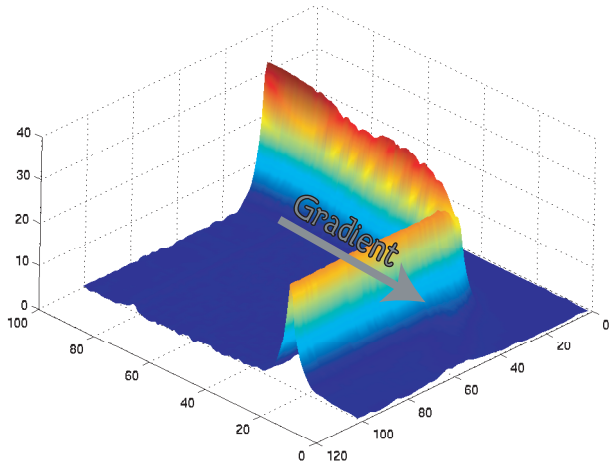
Que pensez vous du résultat ? Essayez la même chose avec une autre image (à niveau de gris).

6.4• Seuillage des contours.

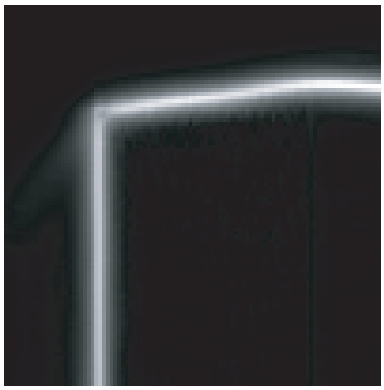
Seuillez l'image des contours en utilisant son histogramme. Que remarquez-vous ?

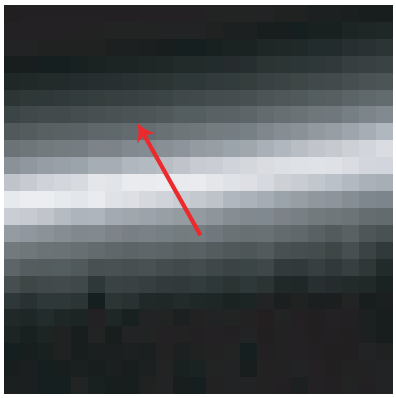
6.5• Extraction des maxima locaux de la norme du gradient.

Cette partie n'est à faire que si vous êtes très avancé.

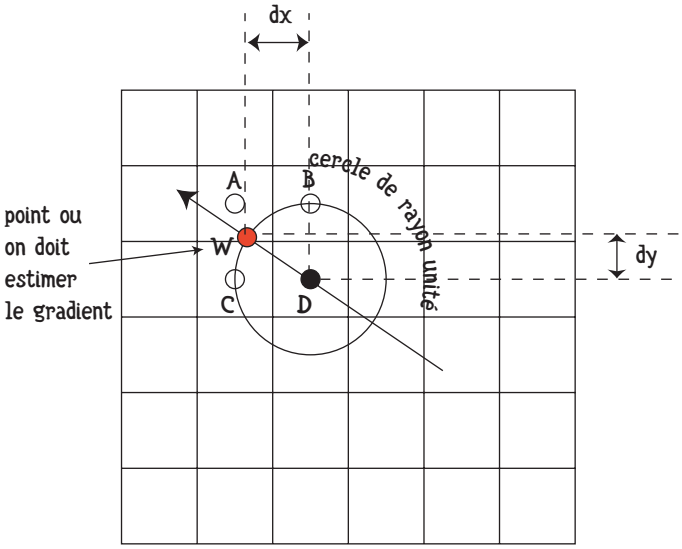


Comme on peut l'intuiter sur l'illustration ci-contre représentant une partie de la norme du gradient de l'image des cubes, le contour subjectif des objets de l'image correspond à la ligne de crête de la norme du gradient. Cette ligne de crête est perpendiculaire au vecteur gradient calculé précédemment. L'idée sous-jascente à l'extraction de la ligne de crête consiste donc tout simplement à comparer la norme du gradient en chaque pixel avec la norme du gradient des deux points situés de part et d'autre du pixel considéré dans la direction du gradient.





Sur la figure ci-contre on peut voir l'image en niveau de gris de la partie concernée de la norme du gradient et un détail de cette partie. Dans la direction du gradient, il n'y a aucune raison qu'il y ait un pixel entier. Il faut donc estimer la norme du gradient dans la direction du vecteur gradient à une distance unitaire du pixel considéré.
Cette méthode peut être résumée dans le schémas suivant :



On doit créer une interpolation au point W en ne prenant en compte que ces plus proches voisins (A, B, C et D). Une façon très simple de faire cette interpolation est d'utiliser une pondération linéaire.

La valeur $\hat{G}(W)$ du gradient en W est donnée par :

$$\hat{G}(W) = \frac{\alpha G(A) + \beta G(B) + \gamma G(C) + \delta G(D)}{\alpha + \beta + \gamma + \delta}$$

Par exemple la valeur de δ, le coefficient de G(D) est donné par :

$$\delta = \min(\delta_x, \delta_y) \text{ avec } \delta_x = 1 - dx \text{ et } \delta_y = 1 - dy$$

On peut vérifier facilement que $\hat{G}(A) = G(A)$, $\hat{G}(B) = G(B)$, ...

Il vous faut donc maintenant trouver une méthode permettant de calculer la position des deux points dont on doit estimer le gradient, puis utiliser cette procédure pour transformer votre image de gradient en une image binaire ou chaque pixel est labélisé 1 s'il est un point de contour, et 0 sinon.
Quand vous avez réussi à effectuer ce travail, essayez votre nouvelle extraction de contour sur d'autres images.