# Filtre Mean Shift

par Xavier Philippeau

Date de publication : 02/05/2008

Dernière mise à jour :

Réduction d'un bruit blanc gaussien sur une image avec le filtre Mean Shift

# I - Introduction

Cet article a pour objectif de vous présenter l'algorithme "Mean Shift" et son application pour la réduction d'un bruit blanc gaussien sur une image.


*Filtre MeanShift*

Nous allons commencer par définir ce qu'est un bruit blanc gaussien, puis étudier le principe de fonctionnement du filtre, et enfin détailler l'algorithme Mean Shift.

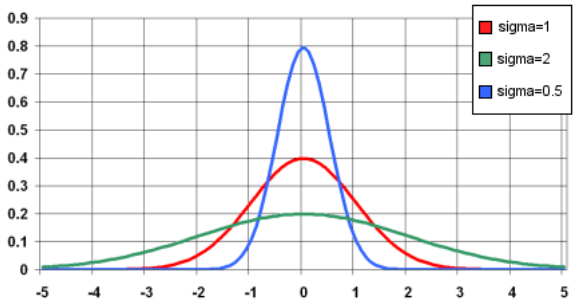
## II - Le Bruit Blanc Gaussien

On appelle bruit gaussien une perturbation dont la densité de probabilité de l'amplitude suit une loi gaussienne centrée.

Cela signifie que la probabilité pour que la perturbation soit d'une amplitude "x" est de:

$$P(x) = \frac{1}{\sigma.\sqrt{2\pi}}.exp(-\frac{x^2}{2.\sigma^2})$$

où "sigma" est un paramètre qui définit la forme de la courbe.




*Courbes Gaussiennes*

De par la forme de la courbe, nous pouvons faire les constatations suivantes :

- La probabilité d'une faible amplitude (proche de 0) est très grande
- La probabilité d'une forte amplitude est très petite
- En moyenne, l'amplitude du bruit est nulle car la courbe est symétrique

En outre, cette densité de probabilité possède des particularités mathématiques intéressantes :

- 68% de probabilité que l'amplitude soit entre -sigma et +sigma
- 95% de probabilité que l'amplitude soit entre -2*sigma et +2*sigma
- 99% de probabilité que l'amplitude soit entre -3*sigma et +3*sigma

L'ajout d'un bruit blanc gaussien sur une image modifie donc la valeur de chaque pixel suivant une loi gaussienne :

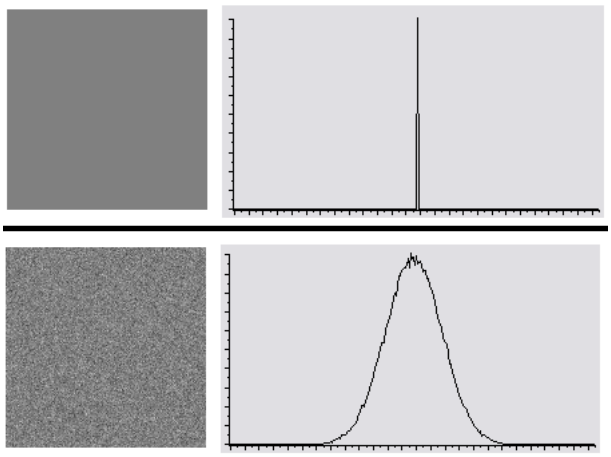

*Ajout d'un bruit gaussien sur une image*

## III - La technique de filtrage

En utilisant les particularités du bruit blanc gaussien, il est possible d'estimer la valeur originale d'un pixel en analysant l'histogramme de l'image.

Image uniforme

Prenons le cas d'une image originale uniformément grise (chaque pixel a la valeur 128).



*Un niveau de gris + bruit gaussien*

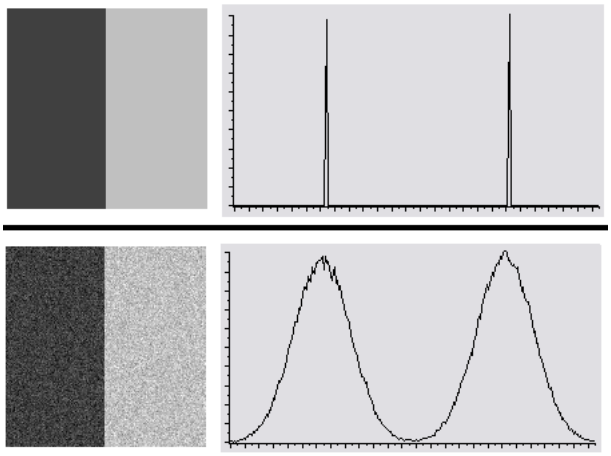
Si on ajoute un bruit blanc gaussien (sigma=10), on obtient une nouvelle image qui a les caractéristiques suivantes :

- La moyenne des valeurs des pixels vaut 128
- 99% des pixels ont une valeur entre 98 (128-30) et 158 (128+30)

Dans ce cas, pour retrouver la valeur originale d'un pixel, il suffit de prendre la moyenne des valeurs des pixels bruités.

Image non-uniforme connue à priori

Prenons maintenant le cas d'une image avec 2 zones grises de même taille (valeurs 64 ou 192).



*Deux niveaux de gris + bruit gaussien*

L'ajout du même bruit blanc gaussien (sigma=10) nous donne une image qui a les caractéristiques suivantes :

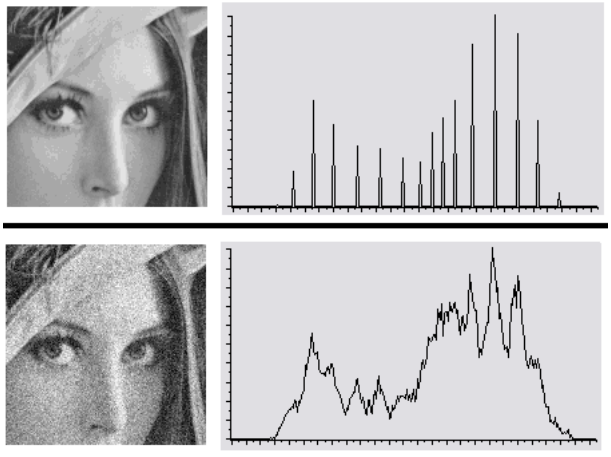
- La moyenne des valeurs des pixels vaut 128 = (64+192)/2
- 99% des pixels ont une valeur dans l'intervalle [34,94] ou [162,222]

Dans ce cas, la moyenne des valeurs ne nous permet pas de retrouver la valeur originale d'un pixel. Il est cependant possible d'estimer quel était le pixel original suivant la valeur du pixel bruité. Il suffit pour cela de trouver dans quel intervalle est situé le pixel bruité.

Si le pixel bruité est dans l'intervalle [34,94], alors il y a 99% de chance que le pixel original soit 64.

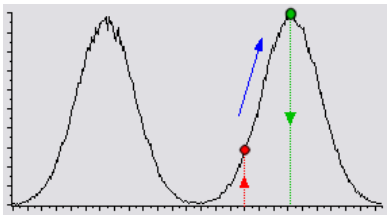
Image non-uniforme inconnue à priori

En se basant sur les 2 cas précédents, on peut remarquer que l'ajout d'un bruit blanc gaussien a pour effet de transformer chaque pic d'histogramme en une courbe gaussienne.



*Multiples niveaux de gris + bruit gaussien*

Il semble donc possible de retrouver la valeur originale d'un pixel bruité en "remontant" le long de la courbe gaussienne :



*Recherche du sommet de la gaussienne*

Dans cet exemple, le pixel bruité (rouge) a une valeur de 140. En remontant la gaussienne de droite, on trouve que la valeur maximale de l'histogramme (vert) correspond aux pixels de valeur 192.

Histogramme local

Pour faciliter le filtrage nous allons travailler sur un petit voisinage du pixel à traiter. Dans l'image originale, on assume que l'histogramme sur ce voisinage va contenir peu de pics. Par conséquence, dans l'image bruitée, l'histogramme sur ce même voisinage ne contiendra que peu de courbes gaussiennes. Il sera ainsi plus facile de déterminer quelle gaussienne il faut "remonter".

Pour affiner encore le filtrage, on peut pondérer l'occurrence de la valeur d'un pixel suivant sa distance au centre du voisinage. Par exemple, pour un pixel P de valeur V, au lieu d'ajouter 1 au compteur d'occurrences de V, on y ajoute $1/(1+\text{distance}^2)$

## IV - L'algorithme Mean Shift

Dans le chapitre précédent, nous avons utilisé comme exemples des images en niveaux de gris. Cela présentait l'avantage d'avoir des histogrammes à 1 dimension :

- Histogramme(valeur de gris)=Nombre d'occurrences de la valeur

Cependant, pour filtrer une image couleur, nous allons devoir utiliser des histogrammes à 3 dimensions :

- Histogramme(valeur de rouge,valeur de vert,valeur de bleu)=Nombre d'occurrences de la couleur

Si la recherche du maximum d'une courbe gaussienne était relativement facile, la recherche du maxima d'une hypersurface gaussienne est plus compliquée. C'est ici qu'intervient l'algorithme Mean Shift.

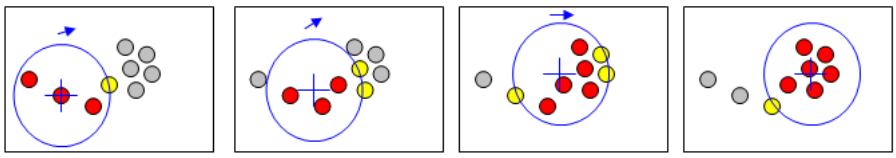
L'algorithme

Le Mean Shift est un algorithme itératif qui a pour objectif de faire converger un point vers le maximum local le plus proche :

1 On commence par choisir un point de départ P.
2 On cherche l'ensemble E des points qui sont dans le voisinage de P.
3 On déplace P vers l'isobarycentre de E.
4 On réitère depuis l'étape 2 jusqu'à convergence.

Les déplacements successifs vers l'isobarycentre font converger le point P vers les zones de fortes densités (*).

Exemple du déroulement de l'algorithme pour des points en dimension 2 :



*Mean Shift 1 Dimension*

L'algorithme Mean Shift est applicable quelle que soit la dimension de l'espace des points.

*(*) On peut également utiliser des pondérations particulières du barycentre pour faire converger le point P vers d'autres zones.*

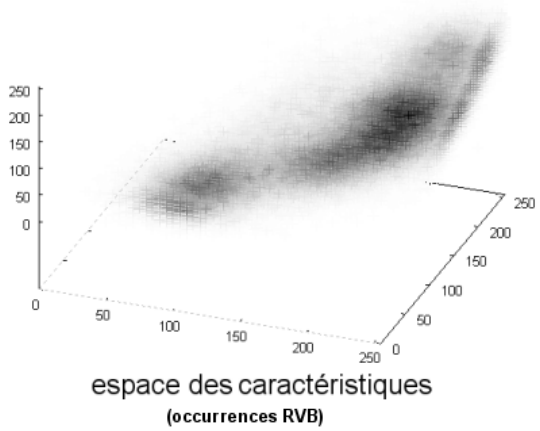
## V - Espace des caractéristiques (Feature Space)

L'espace des caractéristiques représente l'espace des points sur lesquels nous allons appliquer l'algorithme Mean Shift.

Dans notre cas, cet espace devra contenir l'histogramme RVB de l'image bruitée, c'est donc un espace à 3 dimensions :

E = { ensemble des points P, avec P=(rouge,vert,bleu) }

Chaque point P aura une valeur qui représente le nombre d'occurrences de la couleur (rouge,vert,bleu) dans le voisinage considéré.




*Occurences RVB*

Il est bien sur possible d'utiliser d'autres espaces des caractéristiques, par exemple en choisissant une autre représentation des couleurs (HSL, YUV, ...), ou en ajoutant d'autres informations (spatiales, statistiques, ...)

## VI - Implémentation du filtre (java)

Nous allons maintenant détailler l'implémentation du filtre de bruit blanc gaussien sur une image couleur.

Pour chaque pixel P de l'image bruité, nous devons effectuer les étapes suivantes :

### Etape 1 : Calcul de l'espace local des caractéristiques

On crée un nouvel espace des caractéristiques vide, c'est à dire un tableau 3D :

```
int localfeaturespace[][][] = new int[256][256][256];
```

on parcourt les pixels dans un voisinage n*n autour du pixel P à traiter, et on augmente les compteurs d'occurrences

```
for(int yk=y-n/2;yk<=y+n/2;yk++) {
  for(int xk=x-n/2;xk<=x+n/2;xk++) {
   Color color = image.getRGBColor(xk,yk);
   localfeaturespace[color.red][color.green][color.blue]+=1;
  }
 }
```

Comme nous l'avons dit précédemment, on peut pondérer l'occurrence par la distance du pixel au centre du voisinage (P)

```
for(int yk=y-n/2;yk<=y+n/2;yk++) {
  for(int xk=x-n/2;xk<=x+n/2;xk++) {
   Color color = image.getRGBColor(xk,yk);
   double d = distance( x-xk, y,yk );
   double weight = 1/(1+d*d);
   localfeaturespace[color.red][color.green][color.blue]+=1*weight;
  }
 }
```

### Etape 2 : Recherche du maximum de densité

On applique l'algorithme du Mean Shift dans l'espace des caractéristiques en partant de la valeur pixel du P

```
Color meanshift(Color color, int[][][] fspace) {

  // initialize meanshift variables
  Vector3D P = new Vector3D(color.red, color.green, color.blue);
  Vector3D move = null;

  // first move
  move = MSVector(P,fspace);

  // move until convergence (or loop limit)
  int loop=0;
```

```
    while( move.getNorm()>EPSILON && loop<MAXLOOP) {
     P.add(move);
     move = MSVector(P,fspace);
     loop++;
    }

    // return corresponding color
    return new Color( (int)P.x, (int)P.y, (int)P.z );
   }
```

Le calcul du vecteur déplacement est effectué en calculant le barycentre d'un voisinage de P dans l'espace des caractéristiques :

```
  Vector3D MSVector(Vector3D v, int[][][] fspace) {

    // initialize centroid variables
    Vector3D centroid = new Vector3D(0,0,0);
    double total=0;

    // weighted sum all points in the sampling area
    for(int x=(int)(v.x-RADIUS);x<=(int)(v.x+RADIUS);x++) {
     for(int y=(int)(v.y-RADIUS);y<=(int)(v.y+RADIUS);y++) {
      for(int z=(int)(v.z-RADIUS);z<=(int)(v.z+RADIUS);z++) {

       if (OutOfBound(x,y,z)) continue;

       double density = fspace[x][y][z];
       centroid.x += density*x;
       centroid.y += density*y;
       centroid.z += density*z;
       total+=density;
      }
     }
    }

    // normalize the result
    if (total>0) centroid.multiplyByScalar(1.0/total);

    // return the displacement
    return Vector3D.substract(centroid,v);;
   }
```

## VII - Conclusion

Pour chaque pixel, nous avons trouvé la couleur ayant le maximum d'occurrences dans le voisinage spatial 2D (étape 1) et dans le voisinage RGB 3D (étape 2).

Le résultat du filtre dépend principalement de la taille du voisinage RGB. Une taille trop petite va laisser des pixels bruités (effet de grain). Une taille trop grande va uniformiser les couleurs (effet de flou).


*Effets de la taille du voisinage RGB*

La taille la plus pertinente se situe aux alentours de "sigma" (l'ecart-type du bruit gaussien). Cette taille permet de débruiter environ 70% des pixels. Le bruit restant peut alors être considéré comme un bruit impulsionnel et être traité avec des filtres spécifiques (médian, despeckle, ...).

Plugin pour ImageJ

Vous pouvez **télécharger ici** une implémentation Java (jar) de ce filtre sous forme de plugin pour le logiciel ImageJ. Les sources sont contenues dans l'archive Java.

Références

Pour plus d'informations sur l'algorithme Mean Shift et ses utilisations dans le domaine de l'image, je vous conseille la lecture des publications de Dorin Comaniciu et Peter Meer:


- D. Comaniciu, P. Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pp. 603-619, May, 2002*
- D. Comaniciu, P. Meer, "Robust analysis of feature spaces: color image segmentation," *cvpr, p. 750, 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97), 1997*
- Dorin Comaniciu, Peter Meer, "Mean Shift Analysis and Applications," *iccv, p. 1197, Seventh International Conference on Computer Vision (ICCV'99) - Volume 2, 1999*