

The Fast Bilateral Solver

Introduction à l'imagerie numérique

Clément RIU
Tong ZHAO

4 mai 2019

Table des matières

1	Introduction	2
2	Principe du Bilateral Grid	2
3	Principe du Fast Bilateral Solver	3
4	Expériences réalisées	4
4.1	Carte de disparité	4
4.2	Colorisation d'image	7
4.3	Temps de calcul	10
5	Discussion et conclusion	10
	Références	11

1 Introduction

Le principe de la plupart des méthodes de lissage d'image est de remplacer la valeur d'un pixel par la moyenne pondérée de ses voisins. Cette opération lisse l'image sur les régions uniformes, ce qui est bien le résultat attendu, mais elle brouille les bords qui séparent les différentes régions de l'image, ce qui ajoute du flou à l'image. Une solution est d'appliquer un filtre préservant les contours. Étant un algorithme flexible et efficace en termes de calcul, le filtre bilatéral joue un rôle très important dans ce genre d'applications. Néanmoins il n'est pas suffisant pour certaines tâches plus complexes.

L'article [Barron and Poole, 2016] présente un nouvel algorithme de lissage sensible aux bords. Le but de cet algorithme est de proposer une méthode rapide qui effectue le lissage par domaine et qui puisse facilement être intégrée dans un algorithme d'apprentissage profond. L'idée principale repose sur la résolution d'un problème d'optimisation dans l'espace bilatéral. L'intérêt de cet algorithme réside donc dans une méthode à la fois précise, car sensible aux bords et rapide.

Cette article fait souvent référence à de précédent travaux de l'auteur, notamment [Barron et al., 2015].

2 Principe du Bilateral Grid

Le filtre bilatéral est défini comme :

$$I^{filtered}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

où $I^{filtered}$ est l'image filtrée, I est l'image originale, Ω est le domaine des voisins de x , f_r est le noyau pour lisser les différences d'intensités, et g_s est le noyau pour lisser les différences de coordonnées. Le filtre prend en compte en même temps l'intensité lumineuse et la distance spatiale entre les voisins, et formule donc une opération préservant les contours. L'inconvénient de cette méthode est l'absence d'implémentation efficace.

Un schéma largement utilisé pour accélérer le filtre bilatéral est proposé dans l'article [Chen et al., 2007] : la grille bilatérale. Cette nouvelle structure de donnée projette une image en 2D dans une grille en 3D en combinant le domaine spatial et le domaine d'intensité. La grille est ensuite convoluée par un noyau en dimension 3 et l'image de sortie est calculée par une interpolation.

Il y a trois opérations principales : *splat*, *blur* et *slice*.

- *Splat* : Projeter les valeurs de pixel dans une grille en haute dimension, i.e. espace bilatéral. Pour une image en niveau du gris, la dimension de la grille est 3. Pour une image RGB, on la transforme dans l'espace YUV et la dimension de la grille est 5 (2 coordonnées spatiales plus 3 canaux de couleur). La densité de l'espace YUV rend possible les opérations suivantes.
- *Blur* : Appliquer comme une opération de flou sur une image, cette étape est simplement une convolution en haute dimension.
- *Slice* : On retrouve les valeurs filtrées du pixel par une interpolation dans l'espace bilatéral. On peut donc considérer le *splat* et le *slice* comme une paire d'opération inverse l'une de l'autre.

Avec ces trois opération, on obtient une factorisation efficace pour la matrice de poids W :

$$W = S^T \bar{B} S \tag{1}$$

où S^T est la matrice *splat*, S la matrice *slice* et \bar{B} la matrice *blur*. En pratique S est une matrice dont les colonnes représentent les pixels de l'image, nulles partout sauf en une ligne encodée par les informations spatiales, de luma et de chroma de l'image. Et comme son nom l'indique, la matrice \bar{B} est une matrice de flou qui « étale » les points de la grille.

3 Principe du Fast Bilateral Solver

Le Fast Bilateral Solver nécessite une image cible, figure 1, qui constitue l'image à lisser et une confiance par pixel 2. L'algorithme va alors retrouver une image lisse par morceaux qui ressemblera à l'image cible lorsque la confiance par pixel est élevée.



FIGURE 1 – Image cible

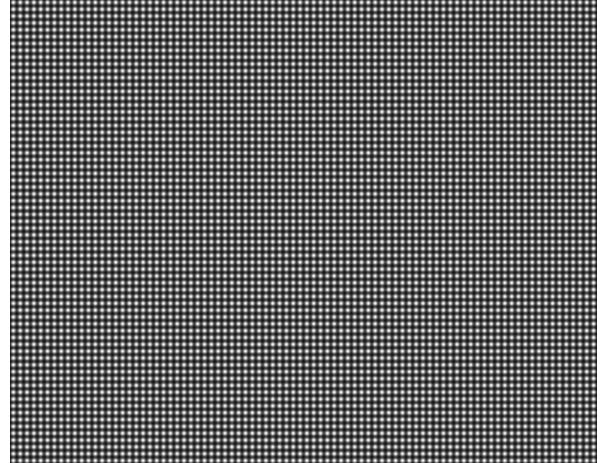


FIGURE 2 – Confiance par pixel.

Les auteurs présentent le problème sous la forme de la minimisation suivante :

$$\min_x \frac{\lambda}{2} \sum_{i,j} \hat{W}_{i,j} (x_i - x_j)^2 + \sum_i c_i (x_i - t_i)^2 \quad (2)$$

Où : c est la matrice de confiance, t est l'image cible, i et j indexent les pixels de l'image et W est une matrice d'affinité définie comme suit :

$$W_{i,j} = \exp \left(-\frac{(d_{spatial})_{i,j}^2}{2\sigma_{spatial}^2} - \frac{(d_{luma})_{i,j}^2}{2\sigma_{luma}^2} - \frac{(d_{chroma})_{i,j}^2}{2\sigma_{chroma}^2} \right) \quad (3)$$

Où les distances spatiale, en luma et en chroma sont calculées sur l'image d'origine dans le système YUV. On note néanmoins que la matrice utilisée dans la fonction objectif est \hat{W} , la bistrochastisation de la matrice W calculée par la grille bilatérale. La façon de calculer cette matrice sera explicitée plus bas.

La fonction objectif est donc un compromis, contrôlé par λ , entre le lissage de l'image, la première somme, et l'attache au donnée lorsque la confiance est élevée, la deuxième somme. On comprend donc pourquoi, si λ , $\sigma_{spatial}$, σ_{luma} et σ_{chroma} sont bien choisis, la solution x au problème de minimisation donnera une image satisfaisante.

La bistrochastisation de W : Afin de faciliter le calcul de la solution, on écrit :

$$\hat{W} = S^T D_{\mathbf{m}}^{-1} D_{\mathbf{n}} \bar{B} D_{\mathbf{n}} D_{\mathbf{m}}^- 1S \text{ avec } SS^T = D_{\mathbf{m}} \quad (4)$$

Pour calculer les matrices $D_{\mathbf{n}}$ et $D_{\mathbf{m}}$, on utilise l'algorithme suivant (reproduit de [Barron et al., 2015]) :

Le changement de variable : Afin de réduire la complexité de l'optimisation, on applique un changement de variable qui transforme une matrice de grande taille en une matrice de petite taille dans l'étape *splat* :

Algorithm 1 Bistochastisation de W :

```

m ←  $S \times \mathbf{1}$ 
n ←  $\mathbf{1}$ 
while L'algorithme n'a pas convergé ou le nombre d'itération est inférieur au nombre d'itération maximum do
     $n \leftarrow \sqrt{(\mathbf{n} \circ \mathbf{m}) / (\bar{B}\mathbf{n})}$ 
end while
 $D_{\mathbf{n}} \leftarrow \text{diag}(\mathbf{n})$ 
 $D_{\mathbf{m}} \leftarrow \text{diag}(\mathbf{m})$ 

```

$$\mathbf{x} = S^T \mathbf{y} \quad (5)$$

En pratique, la plupart des opérations sur la grille nécessitent uniquement une résolution grossière, où le nombre de voxels de la grille est beaucoup plus petit que le nombre de pixels de l'image.

La reformulation du problème : Les deux équations permettent de ré-écrire le problème sous la forme suivante :

$$\min_y \frac{1}{2} \mathbf{y}^T A \mathbf{y} - \mathbf{b}^T \mathbf{y} + \mathbf{c} \quad (6)$$

Où $A = \lambda(D_{\mathbf{m}} - D_{\mathbf{n}} \bar{B} D_{\mathbf{n}}) + \text{diag}(Sc)$, $\mathbf{b} = S(c \circ t)$ et $\mathbf{c} = \frac{1}{2}(c \circ t)^T t$. La minimisation de ce problème donne rapidement $A\mathbf{y} = b$ dans l'espace bilatéral et donc, dans l'espace des pixels $x = S^T(A^{-1}\mathbf{b})$

Le préconditionnement : Afin de résoudre ce système linéaire creux en grande dimension, un préconditionnement est nécessaire afin que la méthode itérative puisse trouver une bonne solution. Un préconditionneur M est une matrice non singulières qui approche A . Le système linéaire devient donc :

$$M^{-1} A \mathbf{x} = M^{-1} b$$

Il est numériquement stable et plus facile à calculer. Dans notre implémentation, on choisit simplement le préconditionneur de Jacobi :

$$M = \text{diag}(A)$$

Le solveur : La méthode du gradient conjugué est un algorithme pour résoudre des systèmes d'équations linéaires dont la matrice est symétrique définie positive, ce qui correspond parfaitement notre problème. On utilise cette méthode préconditionnée par M et initialisée par $W = S^T \bar{B} S$ pour résoudre le système $A\mathbf{y} = b$.

L'implémentation : C'est donc cet algorithme que nous avons implémenté. Le détail de l'implémentation peut se trouver sur : <https://github.com/Tong-ZHAO/Fast-Bilateral-Filter>.

4 Expériences réalisées

4.1 Carte de disparité

Une première application proposée par l'article est l'utilisation du Fast Bilateral Solver afin d'améliorer le résultat d'un calcul de carte de disparité. Voici quelques exemples issus du jeux de don-

nées Middlebury 2005 [Scharstein and Szeliski, 2003]. La carte de disparité est calculée par MC-CNN [Zbontar and LeCun, 2015]¹.

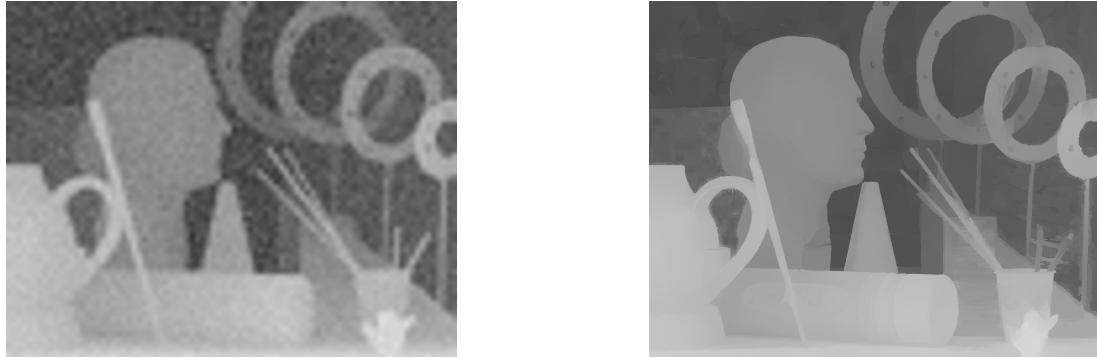


FIGURE 3 – Image **Art** - à gauche l'image cible, à droite, la sortie du Fast Bilateral Solver.

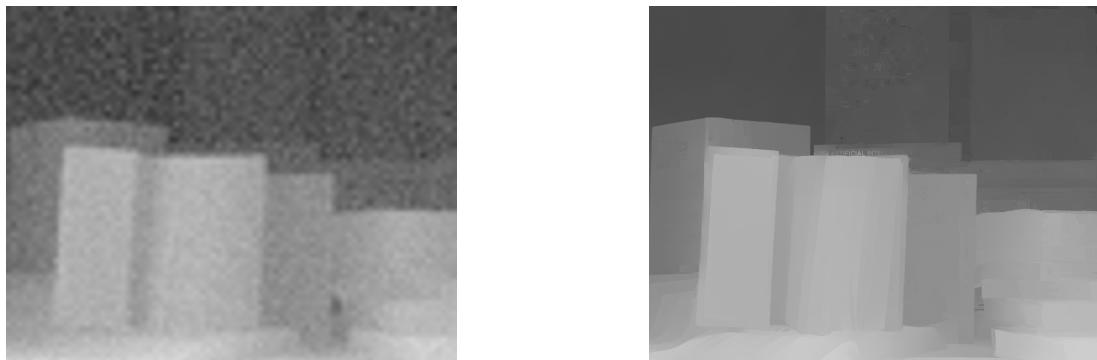


FIGURE 4 – Image **Books** - à gauche l'image cible, à droite, la sortie du Fast Bilateral Solver.

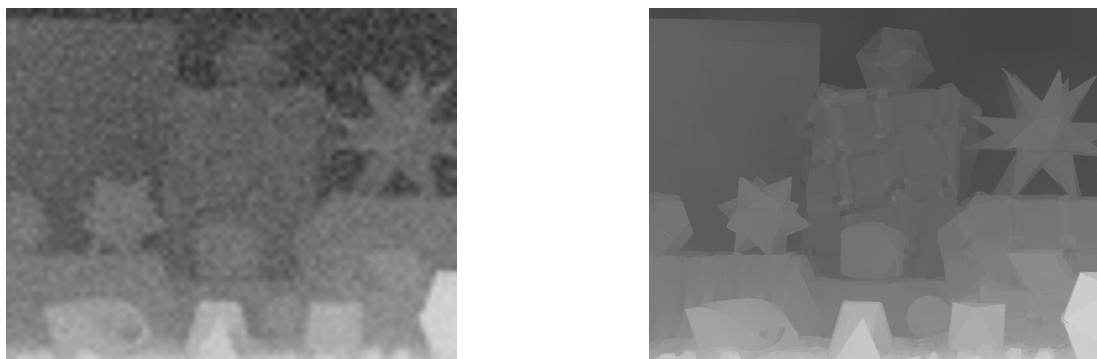


FIGURE 5 – Image **Mobius** - à gauche l'image cible, à droite, la sortie du Fast Bilateral Solver.

Le modèle a quatre paramètres importants : $\sigma_{spatial}$, σ_{luma} , σ_{chroma} et λ définissant respectivement la discréétisation spatiale dans l'espace de la grille bilatérale, la discréétisation en luminosité, la discréétisation en couleur et l'importance du caractère lisse contre l'attache au donnée. Pour comparer l'effet des paramètres nous utilisons une mesure simple, la distance à l'image *ground truth* du jeu de données Middlebury. On observe l'évolution de l'erreur en fonction du paramètre sur les trois images présentées ci-dessus :

1. Les cartes de disparités ont été calculées par les auteurs de l'article [Barron and Poole, 2016], les améliorations par notre implémentation.

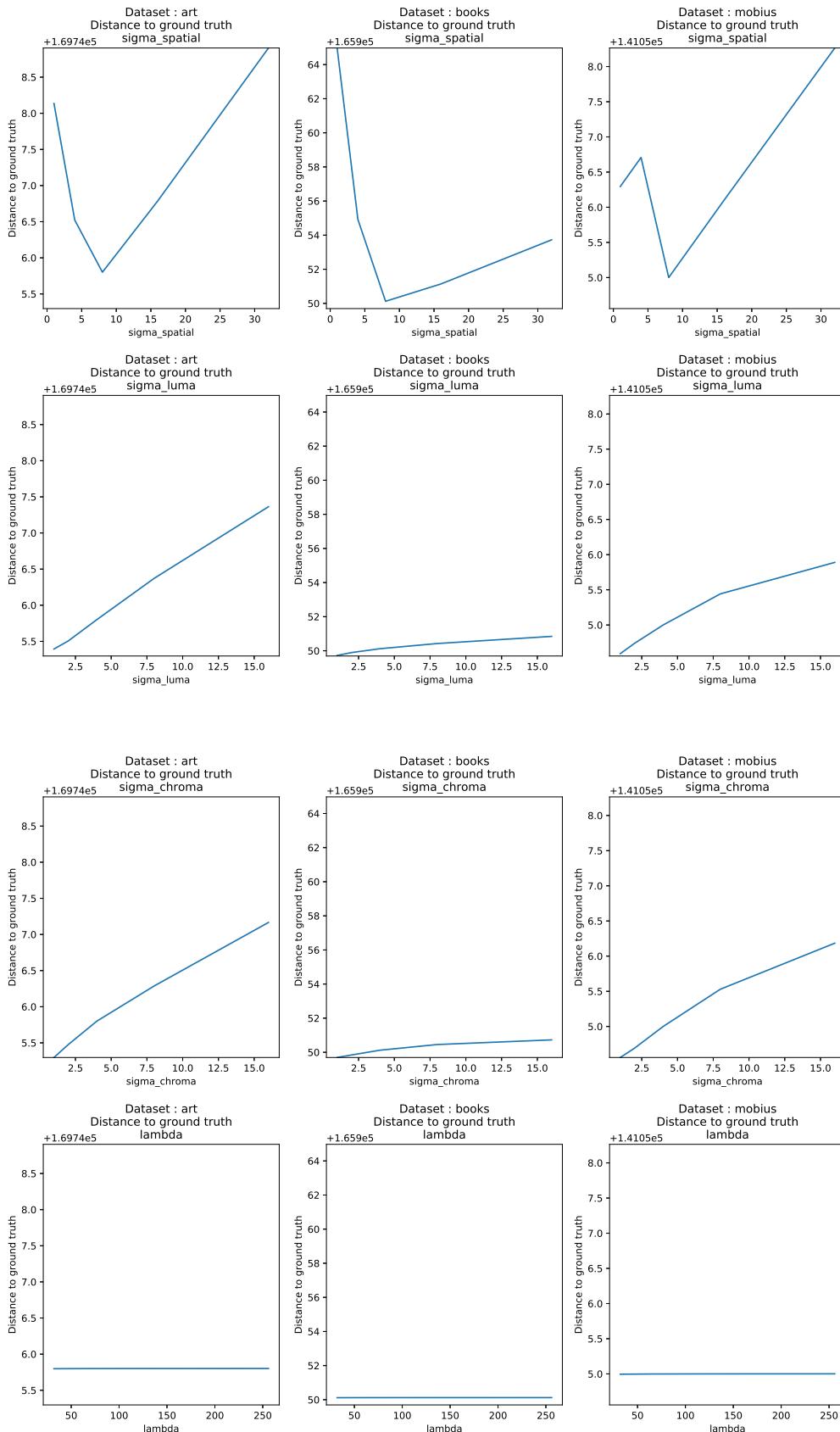


FIGURE 6 – Évolution de la distance à la *ground truth* en fonction des paramètres $\sigma_{spatial}$, σ_{luma} et λ , de haut en bas, sur les images Art, Books, Mobius de gauche à droite.

On remarque que le paramètre de discréétisation spatiale a un impact important sur la distance à la *ground truth*. Par exemple, pour l'image **mobius**, en faisant varier le paramètre $\sigma_{spatial}$ et en gardant

fixe les autres paramètres² on obtient les résultats suivants :



FIGURE 7 – Résultat du Fast Bilateral Solver pour les valeurs de $\sigma_{spatial}$ suivante : 1, 4, 8, 16, 32.

Il est étonnant de remarquer la faible influence du paramètre λ sur le résultat. Même en faisant varier ce paramètre de 10^{-2} à 10^3 , la valeur de la distance varie beaucoup mais, visuellement, le résultat ne varie que peu. Pour analyser plus en détail l'effet de λ sur le résultat, il faudrait peut être utiliser des mesures plus précises de la qualité du résultat, comme la mesure *bad 1%*, ou des images plus complexes, avec plus d'occlusions par exemple. On note également que nous ne nous sommes pas intéressé à la matrice c de confiance. Son influence pourrait expliquer les observations faites ici.

4.2 Colorisation d'image

Le but de la colorisation est d'introduire de la couleur dans une image en niveaux de gris grâce à des pixels colorés par l'utilisateur. Le principe du Fast Bilateral Solver Fast Bilateral Solver s'applique donc parfaitement au problème.

Il est formulé comme suivant : l'image de référence est une image en niveaux de gris, l'image cible est l'image en niveaux de gris partiellement colorée par l'utilisateur, la confiance est une matrice de valeur 0 partout sauf les pixels colorés, dont la valeur est 1. La sortie du solveur est donc une image RGB colorée en respectant la colorisation de l'utilisateur. L'algorithme se décompose comme suit :

Algorithm 2 Colorisation de l'image L

Input : Image de référence L , image cible L_t , confiance L_c

Output : Image colorée L_o

procedure COLORISATION

```

 $L_t^{yuv} \leftarrow \text{rgb2yuv}(L_t)$ 
 $L^{yuv} \leftarrow \text{rgb2yuv}(L)$ 
 $L_t^u \leftarrow L_t^u - 128$ 
 $L_t^v \leftarrow L_t^v - 128$ 
 $U \leftarrow \text{bilateral-solver}(L, L_t^u, L_c)$ 
 $V \leftarrow \text{bilateral-solver}(L, L_t^v, L_c)$ 
 $L_o \leftarrow \text{yuv2rgb}(L^y, U + 128, V + 128)$ 
 $L_o \leftarrow \frac{L_o - L_o^{\min}}{L_o^{\max}} \times 255$ 

```

end procedure

La base de données qu'on a choisi est celui de l'article [Levin et al., 2004], qui contient 6 images en niveaux de gris et leur images colorées à la main. On visualise trois d'entre elles. Les résultat sont réalistes et ils respectent bien leurs images cible.³.

2. $\sigma_{luma} = \sigma_{chroma} = 4$, $\lambda = 128$.

3. $\sigma_{spatial} = 5$, $\sigma_{luma} = \sigma_{chroma} = 4$, $\lambda = 128$

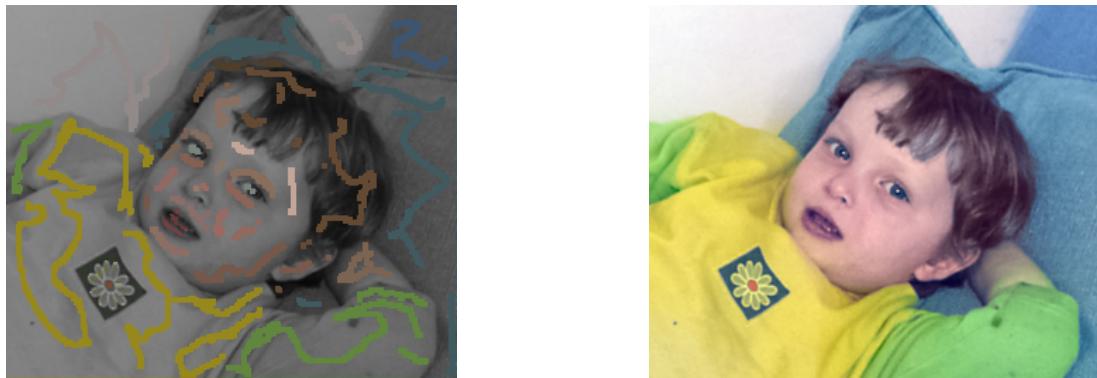


FIGURE 8 – Image Boy - à gauche l'image colorée à la main, à droite, la sortie du Fast Bilateral Solver.



FIGURE 9 – Image Hair - à gauche l'image colorée à la main, à droite, la sortie du Fast Bilateral Solver.

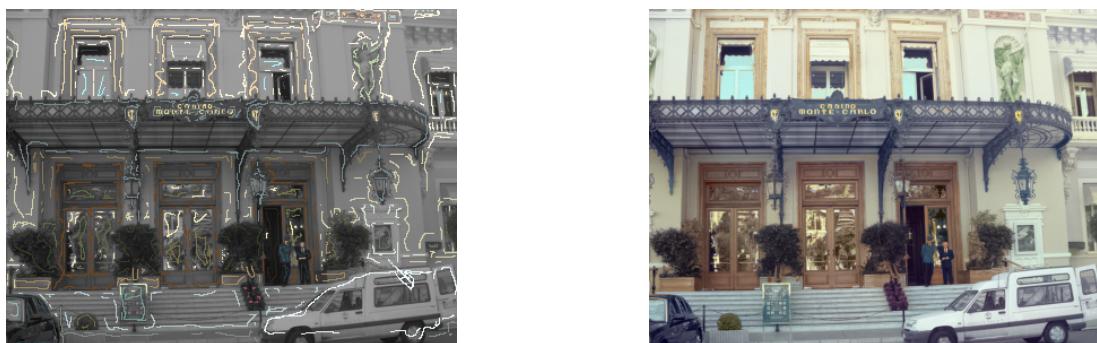


FIGURE 10 – Image Monaco - à gauche l'image colorée à la main, à droite, la sortie du Fast Bilateral Solver.

Afin de découvrir l'influence de la région colorée, on applique sur une image de référence trois cibles différentes et on visualise les images de sortie.



FIGURE 11 – Les trois cibles (a, b, c) sur l'image Odeya



FIGURE 12 – Les sorties du Fast Bilateral Solver (a, b, c) sur l'image Odeya

On ne voit de grande différences entre l'image *a* et *b*, qui nous indique que un petit ensemble de pixels labelisés est capable de produire un résultat satisfaisant. Par contre l'image *c* traite mieux les détails, e.g. le bord de la main et le couleur de la robe.

Pour étudier l'influence des paramètres, on colorise l'image Boy en variant les paramètres.

Le choix du $\sigma_{spatial}$ On remarque que le choix de $\sigma_{spatial}$ est important. Si il est trop petit (dans notre cas quand $\sigma_{spatial} \in \{1, 2\}$), les couleurs ne diffusent pas bien. Si il est trop grand (dans notre cas quand $\sigma_{spatial} \in \{8, 16\}$), les couleurs sont mélangées.



FIGURE 13 – Résultat du Fast Bilateral Solver pour les valeurs de $\sigma_{spatial}$ suivante : 1, 2, 4, 8, 16.

Le choix du σ_{luma} On remarque que le choix de σ_{luma} a une influence sur la propagation des couleurs. Ce choix est important pour que les petites régions soient réalistes. Par exemple, le petit point rouge sur le logo du pull semble artificiel quand $\sigma_{luma} \in \{1, 2, 4\}$. Si il est trop grand (dans notre cas quand $\sigma_{luma} = 32$), il y a des halos autour des bords.



FIGURE 14 – Résultat du Fast Bilateral Solver pour les valeurs de σ_{luma} suivante : 1, 2, 4, 8, 32.

Le choix du σ_{chroma} On observe que ce paramètre n'a pas une influence forte sur le résultat donc on peut choisir un grand σ_{chroma} pour accélérer l'algorithme.



FIGURE 15 – Résultat du Fast Bilateral Solver pour les valeurs de σ_{chroma} suivante : 1, 4, 8, 16, 32.

Le choix du λ Ce paramètre contrôle l'équilibre entre le niveau de régularisation par la grille bilatéral et celui par la carte de confiance. On voit que $\lambda = 128$ est un bon compromis.



FIGURE 16 – Résultat du Fast Bilateral Solver pour les valeurs de λ suivante : 1, 10, 50, 128, 500.

4.3 Temps de calcul

On compare notre implémentation à celle fourni par l'auteur (https://github.com/poolio/bilateral_solver), une version optimisée. On calcule la carte de disparité sur l'image `Art` avec les mêmes paramètres. Notre implémentation prend 13s pour le Fast Bilateral Solver et l'implémentation de l'auteur prend seulement 1.69s pour le calculer. Le calcul le plus lourd est la construction de la grille, qui consume 12.2s.

5 Discussion et conclusion

L'article présente donc un nouvel algorithme, améliorant le filtre bilatéral. Nous avons vu son fonctionnement et testé une implémentation sur deux cas d'applications : le lissage de carte de disparité et la colorisation d'image. Dans les deux cas nous remarquons des résultats très satisfaisant.

Néanmoins, l'algorithme semble assez sensible au paramétrage choisi, en particulier le paramètre de discréétisation spatiale et la présence de quatre paramètres majeurs peut rendre compliqué le choix de ces paramètres sans une étape de cross-validation.

Les auteurs de l'article présente comme principal intérêt sa rapidité. Nous n'avons pas implémenté notre propre filtre bilatéral pour le comparé à notre Fast Bilateral Solver, néanmoins les gains de temps mentionnés en table [1] de l'article donnent le Fast Bilateral Solver comme environ 4 fois plus rapide.

Notons que nous ne nous sommes pas intéressés à l'aspect différentiable du filtre, à savoir les possibilités d'ajout à un réseau de neurones.

Références

- [Barron et al., 2015] Barron, J. T., Adams, A., Shih, Y., and Hernández, C. (2015). Fast bilateral-space stereo for synthetic defocus. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4466–4474.
- [Barron and Poole, 2016] Barron, J. T. and Poole, B. (2016). The fast bilateral solver. In *European Conference on Computer Vision*, pages 617–632. Springer.
- [Chen et al., 2007] Chen, J., Paris, S., and Durand, F. (2007). Real-time edge-aware image processing with the bilateral grid. In *ACM Transactions on Graphics (TOG)*, volume 26, page 103. ACM.
- [Levin et al., 2004] Levin, A., Lischinski, D., and Weiss, Y. (2004). Colorization using optimization. In *ACM transactions on graphics (tog)*, volume 23, pages 689–694. ACM.
- [Scharstein and Szeliski, 2003] Scharstein, D. and Szeliski, R. (2003). High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE.
- [Zbontar and LeCun, 2015] Zbontar, J. and LeCun, Y. (2015). Computing the stereo matching cost with a convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1592–1599.