Projet de Vision Algorithmique

Fast Image Inpainting Based on Coherence Transport

Vincent Delaitre

vincent.delaitre@ens-lyon.org

1 Introduction

L'inpainting sans texture consiste à interpoler des parties masquées d'une image. De nombreuses méthodes ont été développées, notamment à base d'équations aux dérivés partielles non linéaires. Celles-ci ont malheureusement du mal à passer à l'échelle car les critères de stabilité imposent d'effectuer un grand nombre d'itérations.

L'article étudié adopte un point de vue différent et décrit un algorithme qui effectue le coloriage de l'image en une seule passe. La contribution principale de cet article est de proposer une nouvelle fonction de poids pour un algorithme déjà existant et de prouver certaines bonnes propriétés de cette fonction.

Dans ce rapport, je m'attache principalement à résumer l'article et je met en lumière notamment les raisons qui ont motivé les choix des auteurs dans la construction de la fonction de poids. La première partie expose l'algorithme générique proposé par Telea et réutilisé par les auteurs. La seconde décrit la nouvelle fonction de poids. Enfin, je présente les résultats que j'ai obtenu à partir de mon propre programme dans la dernière partie.

2 État de l'art

Les méthodes d'inpainting en une seule passe ont déjà fait l'objet de recherches et Telea a par exemple proposé dans [1] un algorithme basé sur les fronts avançant. Le projet sera basé sur cet algorithme décrit section 2.1 et seule la fonction poids naïve présentée section 2.2 sera remplacée par la fonction décrite section 3.

Avant de commencer, posons quelques notations qui serviront dans le reste du rapport :
- L'ensemble des pixels de l'image est noté Ω.
- Pour tout $x \in \Omega$, on note $u(x)$ la couleur du pixel x.
- On note K la zone de l'image où la couleur des pixels est donnée.
- On note D la zone de l'image pour laquelle la couleur des pixels doit être déterminée, si bien que $K \cup D = \Omega$ et $K \cap D = \emptyset$.
- Pour tout $x \in \Omega$ on note $T(x) = dist(x, K)$.
- Les prédécesseurs d'un pixel x sont les pixels $\Omega(x) = \{y \in \Omega : T(y) < T(x)\}$.
- La boule de centre x et de rayon ϵ est notée $B_\epsilon(x) = \{y \in \Omega : |x - y| < \epsilon\}$

2.1 Algorithme générique

L'algorithme que nous allons utiliser se décompose en deux grandes étapes :

Ordonnancement des pixels de D

Le remplissage de D va se faire dans un ordre bien précis en fonction de la distance à K. Pour calculer celle-ci, on utilise la méthode des fronts avançant : pour $x \in D$ on calcule la distance $d(x, K)$ de proche en proche en commençant par les pixels ayant un voisin dans K (voir [1] pour l'algorithme et [2] pour le calcul des distances). On renvoie la liste $x_1, x_2, \cdots, x_n$ des pixels de D ordonnée par ordre croissant de la distance à K.

Remplissage de D

On parcourt la liste renvoyée ci-dessus tout en coloriant les pixels. Supposons que $x_k \in D$ soit le pixel courant : tous les $x_1, \cdots, x_{k-1}$ ont été colorés auparavant. On note $B_\epsilon^<(x_k)$ le ϵ-voisinage coloré de x_k :

$$B_\epsilon^<(x_k) = B_\epsilon(x_k) \backslash \{x_1, \cdots, x_{k-1}\} = B_\epsilon(x_k) \cap \Omega(x) \tag{1}$$

On suppose de plus que l'on a une fonction $w(x, y) \geq 0$ qui représente l'influence du pixel y sur la couleur du pixel x. On peut alors calculer la couleur de x_k par :

$$u(x_k) = \frac{\sum_{y \in B_\epsilon^<(x_k)} w(x_k, y) u(y)}{\sum_{y \in B_\epsilon^<(x_k)} w(x_k, y)} \tag{2}$$

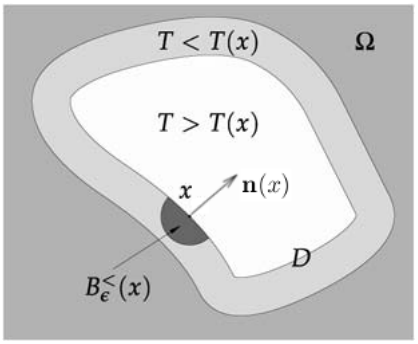
On itère ainsi jusqu'à ce que la liste des x_i soit vide. La fonction w doit être choisie avec soin et c'est tout l'objet de l'article étudié. Telea propose pour sa part une fonction de poids assez "naïve".

2.2 Fonction de poids de Telea

Reprenons la distance calculée au début de l'algorithme générique par la méthode des fronts. On peut calculer la normale au front avançant en un point $y \in D$ simplement en calculant le gradient de la distance T : $\mathbf{n}(y) = \nabla T(y)$.

L'idée de Telea est assez simple : un point y contribue d'autant plus à la couleur d'un pixel $x \in D$ qu'il en est proche et aligné avec la normale. Telea choisit donc le poids suivant :

$$w(x, y) = \frac{|\mathbf{n}(x) \cdot (x - y)|}{|x - y|^2}$$



Ce choix de w est cependant trop simpliste : il ne dépend pas des données mais seulement de la forme de la région à remplir. Les couleurs sont transportées dans la direction normale au front avançant et il en résulte un flou considérable. Les auteurs de l'article s'attachent donc à concevoir un poids plus sophistiqué prenant en compte la structure de l'image pour transporter les couleurs dans la direction adéquate.

3 Fonction de poids de Bornemann

Une fonction de poids raisonnable serait une fonction favorisant le lisage dans la direction des bords afin d'éviter les effets de flou. L'idée est d'utiliser cette direction, dite direction de cohérence, à la place du vecteur normal $\mathbf{n}$ dans le poids de Telea.

3.1 Direction de cohérence

On note G_σ la gaussienne d'écart-type σ et $\star$ le produit de convolution. Habituellement, la direction de cohérence d'une image u peut être obtenue à partir du tenseur de structure $J_{\sigma,\rho}$ définit comme suit :

$$J_{\sigma,\rho} = G_\rho \star (\nabla u_\sigma \otimes \nabla u_\sigma), \qquad \text{avec } u_\sigma = G_\sigma \star u \tag{3}$$

Le tenseur de structure peut être diagonalisé : notons $\lambda_1(x)$ et $\lambda_2(x)$ les deux valeurs propres de $J_{\sigma,\rho}(x)$ avec $\lambda_1(x) < \lambda_2(x)$. La direction de cohérence est simplement le vecteur propre $\mathbf{w_1}(x)$ associé à la plus petite valeur propre.

Il est cependant nécessaire de prendre certaines précautions. Par la suite, ces valeurs de la direction de cohérence vont être calculées au voisinage de la zone à colorier En particulier, la convolution par les gaussiennes G_σ et G_ρ ne doit pas tenir compte de cette région. Pour cela on utilise la fonction indicatrice $1_{\Omega(x)}(y) = 1$ si le pixel y appartient à $\Omega(x)$, 0 sinon.

On devrait alors plutôt définir $J_{\sigma,\rho} = G_\rho \star (1_{\Omega(x)} \nabla u_\sigma \otimes \nabla u_\sigma)$, avec $u_\sigma = G_\sigma \star (1_{\Omega(x)} u)$ mais cette formulation du tenseur pose problème. En effet, l'introduction de la fonction indicatrice a pour effet de rendre la direction de cohérence orthogonale aux fronts avançants. On retrouve alors exactement la fonction de poids de Telea. Pour compenser ce problème, il faut normaliser par la surface du voisinage porteur d'information. On définit donc le tenseur de structure modifié :

$$\hat{J}_{\sigma,\rho} = \frac{G_\rho \star (1_{\Omega(x)} \nabla u_\sigma \otimes \nabla u_\sigma)}{G_\rho \star 1_{\Omega(x)}}, \qquad \text{avec } u_\sigma = \frac{G_\sigma \star (1_{\Omega(x)} u)}{G_\sigma \star 1_{\Omega(x)}} \tag{4}$$

On utilise les mêmes notations que précédemment pour les valeurs et vecteurs propres de $\hat{J}_{\sigma,\rho}(x)$. La direction de cohérence est donc finalement :

$$\mathbf{c}(x) = \mathbf{w_1}(x)/||\mathbf{w_1}(x)|| \tag{5}$$

3.2 Force de cohérence

On définit de plus la force de cohérence comme ci-dessous. Elle mesure en quelque sorte la confiance que l'on accorde à la direction de cohérence calculée ci-dessus.

$$\mu(x) = \begin{cases} 1 & \text{si } \lambda_1(x) = \lambda_2(x) \\ 1 + \kappa \exp(\frac{-\delta^4}{(\lambda_1(x) - \lambda_2(x))^2}) & \text{sinon} \end{cases} \tag{6}$$

avec δ une constante (paramètre de l'algorithme). La force de cohérence va servir à rendre la nouvelle fonction de poids ci-dessous plus ou moins tolérante avec les pixels tels que le vecteur $\mathbf{x} - \mathbf{y}$ n'est pas aligné avec $\mathbf{c}(\mathbf{x})$.

3.3 Fonction de poids

Comme énoncé plus haut, l'idée était d'utiliser le vecteur de direction de cohérence $\mathbf{c}(x)$ plutôt que la normale dans le poids de Telea. Cela mène à la fonction de poids suivante :

$$w(x, y) = \frac{|\mathbf{c}(x) \cdot (x - y)|}{|x - y|^2} \tag{7}$$

Malheureusement, la dépendance directionnelle est linéaire et n'est pas assez prononcée. On utilise donc la fonction exponentielle pour accélérer la décroissance du poids dès que l'on s'écarte de la direction $\mathbf{c}(\mathbf{x})$. L'angle caractéristique dépend lui de la valeur de $\mu(x)$. Le nouveau poids est donc défini de la manière suivante :

$$w(x, y) = \sqrt{\frac{\pi}{2}} \frac{\mu(x)}{|x - y|} \exp\left(-\frac{\mu(x)^2}{2\epsilon^2} |\mathbf{c}(x)^\perp \cdot (x - y)|^2 \right) \tag{8}$$

4 Résultats

J'ai implémenté l'algorithme générique de Telea ainsi que la fonction de poids de Bornemann. Je ne rentre pas dans les détails d'implémentation car celle-ci ne pose pas de grande difficulté. L'algorithme est résumé à la fin de la section 5 de l'article et les détails d'implémentation sont donnés section 6, notamment pour la gestion des images en couleurs.

Mon projet peut être compilé sous Linux simplement en tapant `make`. Il se lance en indiquant l'image à charger et la couleur à remplacer :

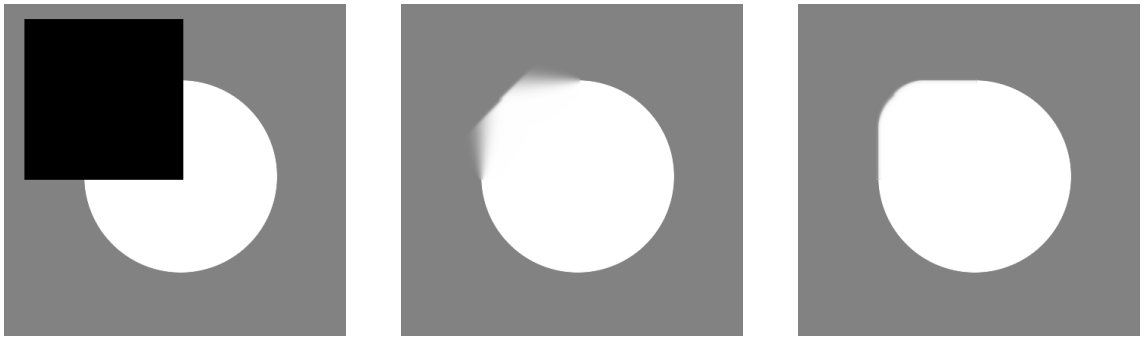
```
./inpaint filename R V B
```

Je présente ci-dessous les résultats de mon projet pour quatre problèmes assez différents :

– Figure 1 : une zone à remplir très large et des contours très nets.

– Figure 2 : une zone à remplir fine et peu de texture.
– Figure 3 : une zone à remplir très fine mais recouvrant la majeure partie de l'image.
– Figure 4 : une zone à remplir assez large et une texture avec de nombreux détails.

Mes résultats diffèrent légèrement de ceux présentés dans l'article d'origine. Les calculs ont été effectué sur un processeur 2GHz Intel CoreTM 2 Duo avec 4GB de mémoire RAM sous Ubuntu Linux v.9.04.



(a) Image originale (b) Telea : CPU 0.7s (c) Bornemann : CPU 17.9s ($\epsilon = 6$, $\kappa = 25$, $\sigma = 12$, $\rho = 18$)

Fig. 1. Image artificielle : rond blanc sur fond gris : 452x440px

5 Conclusion

La méthode proposée par l'article donne des résultats de très bonne qualité même si les performances pourraient éventuellement être améliorées pour les images possédant des textures de fréquence élevée (cf les plumes du perroquet figure 4).

Je n'ai pas comparé avec les algorithmes basés sur des équations différentielles mais il semble que le programme s'exécute en un temps très raisonnable (de l'ordre de 15s pour une image 500x500).

On pourrait cependant reprocher à cette méthode de dépendre de paramètres tels que ϵ, σ ou ρ qui nécessitent d'être adaptés pour chaque image. Il pourrait être intéressant d'essayer de s'affranchir de cette contrainte.

Références

1. Telea A. An image inpainting method based on the fast marching method. In *J. Graph. Tools 9(1)*, pages 23–34, 2004.
2. C. Chalons. La méthode fast-marching pour la propagation de fronts. `http://people.math.jussieu.fr/~chalons/Propagation_Fronts/slides_cours_fast_marching.pdf`.
3. T. März F. Bornemann. Fast image inpainting based on coherence transport. In *J. Graph. Tools 9(1)*, pages 259–278, 2007.

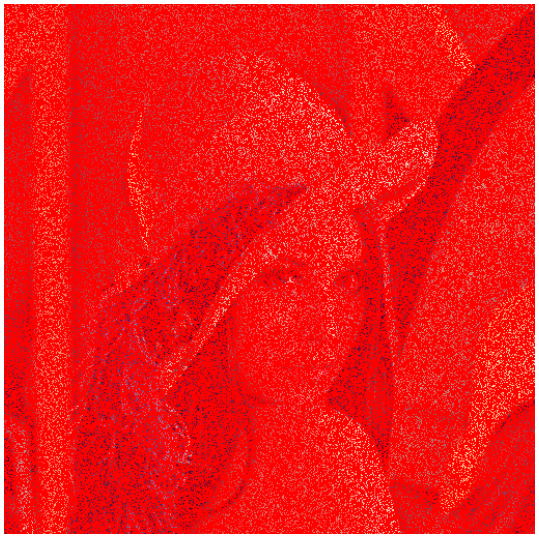
(a) Image originale

(b) Telea : CPU 0.3s

(c) Bornemann : CPU 1.7s ($\epsilon = 4$, $\kappa = 25$, $\sigma = 2$, $\rho = 3$)

Fig. 2. Remplissage de texte : 437x296px

(a) Image originale



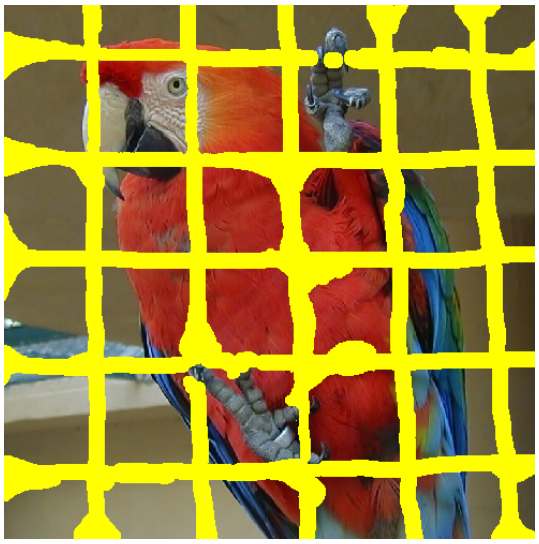
(b) Image masquée à 80%

(c) Telea : CPU 3.8s

(d) Bornemann ($\epsilon = 6$, $\kappa = 25$, $\sigma = 2$, $\rho = 4$) : CPU 14.8s

Fig. 3. Immortal Digital Lena : 512x512px

(a) Image originale



(b) Image masquée

(c) Telea : CPU 7.5s

(d) Bornemann ($\epsilon = 20$, $\kappa = 25$, $\sigma = 4$, $\rho = 7$) : CPU 18.3s

Fig. 4. Perroquet en cage : 495x498px