

Étude et implémentation d'un algorithme d'inpainting basé sur des exemples

Sihem Abdoun et Abdelhadi Temmar

16 Janvier 2017



Figure 1: Disparition du parachutiste grâce à l'algorithme d'inpainting

1 Introduction

Dans ce projet notre intérêt est porté sur une technique qui permet la suppression d'objets dans des images par remplissage ou recouvrement automatique tout en essayant d'avoir le résultat le plus naturel et plausible possible. L'algorithme que nous avons implémenté à cet effet est basé sur le travail présenté dans [AC03]. Nous détaillerons par la suite les raisons de ce choix. Pour l'implémentation nous nous sommes basé sur les détails fournis dans l'article et nous avons, dans un premier temps, scrupuleusement suivi les étapes décrites par l'algorithme présenté. Cependant, la tâche n'étant pas triviale, ces derniers n'ont pas été suffisants pour arriver à des résultats concluants. Nous avons donc dû mettre en œuvre nos connaissances et mener beaucoup d'expériences afin de réussir à rassembler les pièces manquantes pour des résultats plus satisfaisants et comparables à ceux présentés dans l'article. Dans la suite de ce rapport, nous expliquerons donc d'une part le travail proposé dans l'article, nos ajouts et modifications puis pour finir nous présenterons les résultats obtenus et nous conclurons sur une discussion des avantages et des limites de ce qui a été fait tout au long.

2 Algorithmes pour la suppression d'objets et motivations

On peut sans grand effort imaginer des applications très utiles à de tels algorithmes, en allant de la restauration d'images ou de films détériorés à la simple suppression des yeux rouges sur les photos des personnes. Beaucoup de travaux ont été menés dans ce sens, généralement on distingue deux grandes classes d'algorithmes étudiés:

- Algorithmes de synthèse de texture: sont des algorithmes qui permettent de remplacer des régions assez vastes à partir d'exemples de textures qu'il répète avec une petite part d'aléatoire.
- Techniques d'inpainting: sont des algorithmes qui combinent des petites parties manquantes des images en exploitant des structures linéaires tels que les contours.

À l'intersection de ces deux grandes classes de méthodes se trouve l'algorithme sur lequel notre choix s'est porté pour ce projet. Les motivations derrière ce choix sont multiples. Tout d'abord, l'algorithme proposé ne fait appel à aucune autre source de données que l'image contenant l'objet

à supprimer. D'autres méthodes qui, elles, font usage d'une banque d'images pour trouver les meilleures "patterns" existent et sont très performantes mais nécessitent d'analyser une trop grande quantité d'images et par conséquent elles s'avèrent extrêmement lentes. D'autre part, le fait que le remplissage soit guidé par la texture et la structure fait que l'algorithme hallucine assez souvent, pour un background plus cohérent, des patterns(groupes de blocs de pixels). Ce qui donne très rarement une impression de discontinuité dans l'image. En effet ce travail a été réalisé au sein des laboratoires de recherche en Intelligence Artificielle de Microsoft. Selon Google Scholar, il a déjà fait objet de 836 citations.

3 Explication et implémentation de l'algorithme

Avant d'entamer l'explication des différentes parties de l'algorithme et de leur rôle par rapport au résultat escompté. Nous présentons la figure 2 qui illustre, pour plus de clarté, les différentes notations qui seront adoptées par la suite.

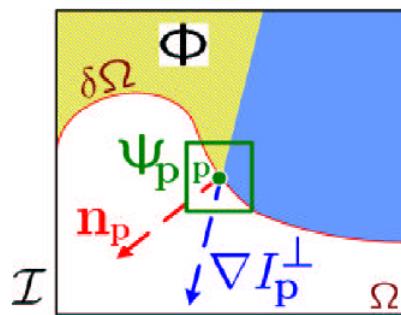


Figure 2: Notations: $\delta\Omega$ représente le contour de la région qui reste à remplir, Ω la région à remplir, n_p la normale au contour, ∇I_p^\perp représente la direction et l'intensité au point p (le gradient). I désigne l'image tout entière

L'algorithme est alors donné comme suit:

- Extract the manually selected initial front $\delta\Omega^0$.
- Repeat until done:
 - 1a. Identify the fill front $\delta\Omega^t$. If $\Omega^t = \emptyset$, exit.
 - 1b. Compute priorities $P(\mathbf{p}) \quad \forall \mathbf{p} \in \delta\Omega^t$.
 - 2a. Find the patch $\Psi_{\hat{\mathbf{p}}}$ with the maximum priority,
i.e., $\Psi_{\hat{\mathbf{p}}} \mid \hat{\mathbf{p}} = \arg \max_{\mathbf{p} \in \delta\Omega^t} P(\mathbf{p})$
 - 2b. Find the exemplar $\Psi_{\hat{\mathbf{q}}} \in \Phi$ that minimizes $d(\Psi_{\hat{\mathbf{p}}}, \Psi_{\hat{\mathbf{q}}})$.
 - 2c. Copy image data from $\Psi_{\hat{\mathbf{q}}}$ to $\Psi_{\hat{\mathbf{p}}}$.
 3. Update $C(\mathbf{p}) \quad \forall \mathbf{p} \mid \mathbf{p} \in \Psi_{\hat{\mathbf{p}}} \cap \Omega$

Figure 3: Algorithme de remplissage de région

Ce qui est important à savoir dans cet algorithme, c'est qu'il ne remplit pas le trou pixel par pixel, mais bloc de pixels par bloc de pixels. Cela signifie que ψ_p est le bloc de pixels centré au point p . La taille d'un bloc est fixée comme étant la taille de la structure la plus épaisse dans l'image source (un contour par exemple). Cette taille, pour plus de souplesse et d'aisance expérimentale est plus souvent manuellement définie par l'utilisateur.

- Tout d'abord, identification les contours initiaux de l'objet (la région) à supprimer (définis par l'utilisateur).
- **Étape 1a :** On identifie le bord du trou qui reste à remplir (si il n'y a pas d'espace restant à remplir, on quitte la boucle). Pour identifier le contour du trou, nous calculons le gradient sur le masque de l'image. Le masque est simplement une matrice de la taille de l'image, qui contient "1" si le pixel fait partie du trou, 0 sinon. En faisant convoluer un masque laplacien à ce dernier, on obtiendra des valeurs nulles pour les pixels à l'intérieur et à l'extérieur du trou. En revanche, sur le contour, la valeur sera non nulle. C'est comme ça que l'on identifie le contour de la région qui reste à remplir. La figure 4 nous montre que cette méthode permet une identification précise des contours.



Figure 4: Détection du contour d'un masque par convolution avec un Lapalcien

- **Étape 1b :** On attribue à chaque patch restant à remplir une priorité de remplissage. Cette priorité se calcule de la façon suivante (la priorité d'un ψ_p est $P(p)$) :

$$P(p) = C(p)D(p) \quad (1)$$

Où $C(p)$ est appelé "*Confidence terme*" et $D(p)$ est appelé "*Data term*". Ces derniers sont calculés comme suit:

$$C(\mathbf{p}) = \frac{\sum_{\mathbf{q} \in \Psi_p \cap \bar{\Omega}} C(\mathbf{q})}{|\Psi_p|}, \quad D(\mathbf{p}) = \frac{|\nabla I_{\mathbf{p}}^{\perp} \cdot \mathbf{n}_{\mathbf{p}}|}{\alpha}$$

Ici, $|\psi_p|$ correspond à l'aire du patch (Cette aire est normalement fixe mis à part sur les bords de l'image). Il est à noter qu'à l'initialisation, $C(p)$ est égale à 0 $\forall p \in \Omega$ et à 1 pour le reste de l'image. Pour une image en niveau de gris, α est un facteur de normalisation et est égale à 255. Dans notre cas, alpha est la valeur maximale de l'image.

Le terme $C(p)$ permet de privilégier le remplissage des blocs ayant des pixels dans le voisinage qui ont déjà été remplis ou faisant partie de l'image originale. En effet, c'est un terme de confiance. En d'autres termes, nous accordons plus de confiance aux pixels ayant dans leur voisinage des pixels provenant de l'image originale. On verra à l'étape 3, que l'on fait en sorte que les pixels remplis par l'algorithme aient un poids plus faible.

Le terme $D(p)$ permet de privilégier les patchs qui contiennent une continuation de contour ou de structure. Nous ajoutons à ce terme une constante que l'on fixe à 0.001 pour éviter d'avoir des valeurs nulles.

- **Étape 2a :** On choisit le patch dont le pixel centrale se trouve sur le bord du trou et ayant la priorité maximale.
- **Étape 2b :** On trouve le bloc de pixels ψ_q qui va minimiser la distance $d(\psi_{\hat{p}}, \psi_q)$. On recherche donc $\psi_{\hat{q}} = \underset{\psi_q \in \Phi}{\operatorname{argmin}} d(\psi_{\hat{p}}, \psi_q)$. La distance d est définie comme la distance *ssd* des pixels qui ont déjà été remplis. Cela n'est pas précisé dans l'article, mais nous faisons en

sorte que les patchs trouvés n'ait pas de pixels appartenant à Ω

- **Étape 2c :** On copie le bloc de pixels qui a minimisé la distance. On va donc remplir tous les pixels $p \in \psi_{\hat{p}} \cap \Omega$ par leur position correspondante dans le patch $\psi_{\hat{q}}$
- **Étape 3 :** On met à jour $C(p)$ pour tous les pixels qui appartiennent au patch qui vient d'être rempli et qui appartiennent aussi à Ω^t . La mise à jour se fait de cette façon :

$$C(p) = C(\hat{p}) \quad \forall p \in \psi_{\hat{q}} \cap \Omega^t$$

Cette étape permet de réduire le terme de confiance pour les pixels qui sont entourés par des patchs remplis par l'algorithme. Cela permet de privilégier (et donc donner plus de confiance) le remplissage des blocs qui sont entourés par des pixels de l'image originale.

- On recommence ces étapes jusqu'à ce que Ω^t (la partie qui reste à combler) soit vide.

4 Choix d'implémentation

4.1 Langage de programmation utilisé

Pour réaliser ce projet, nous avons décidé d'utiliser Python. Notre motivation derrière ce choix est simple; python est un langage utilisable librement qui fourni des librairies telles que: *scipy*, *matplotlib*, *skimage* et *numpy*. Ces dernières permettent une manipulation simple et efficace des images.

4.2 Estimation de la normale

Pour estimer n_p au contour, on calcul le gradient suivant x et suivant y du masque. Pour avoir un vecteur unitaire, on normalise par la norme du gradient. En faisant ça, nous estimons bien la normale car le gradient est perpendiculaire au contour.

4.3 Sélection de la zone à supprimer

Pour la sélection de la zone à supprimer, l'idéal aurait été de permettre à l'utilisateur de définir les contours de l'objet de la façon la plus précise qui soit. Par soucis de simplification nous avons choisi de permettre la sélection de la zone sous forme de rectangle pour ne récupérer que les coordonnées de deux pixels pour la définition de la zone. Ce point constitue une de nos principales perspectives d'amélioration sur l'algorithme: permettre une sélection plus précise.

4.4 Calcul de ΔI_p

ΔI_p est estimé comme la valeur maximale du gradient $\in \psi_p \cap I$. Avant de calculer le gradient, nous appliquons un filtre gaussien avec un faible écart type (qui peut être paramétré par l'utilisateur). Même si cela n'a pas eu de grands effets sur les images de test que nous avons utilisé, nous pensons que ça peut utile si l'image présente un faible bruit gaussien.

4.5 Système de couleurs utilisé

Nous avons décidé de réaliser tous nos calculs en utilisant le système de couleur LAB, car les couleurs sont plus uniformes. C'est à dire que les 3 canaux de couleurs ont la même importance contrairement au système RGB. En effet, l'œil humain perçoit beaucoup mieux le vert que le bleu.

5 Difficultés rencontrées

Comme nous l'avons signalé dans l'introduction de ce rapport, beaucoup de choses ne sont pas renseignées dans l'article, par exemple le calcul de n_p , ΔI_p et du contour. Une autre difficulté est le choix d'images pertinentes et de la région du trou sur ces dernières. En effet, pour l'algorithme, certaines images peuvent être plus "faciles" que d'autres à réussir.

6 Résultats et Discussion

Dans cette section, nous présentons les expériences les plus pertinentes que nous avons menées. Les premières expériences visaient à mesurer l'efficacité de l'algorithme dans sa globalité. Dans les images présentées par la figure 5 nous avons tenté de supprimer une statue assez imposante au centre de l'image. Le résultat est assez surprenant car mis à part une légère faute sur l'un des deux poteaux le résultat retourné par l'algorithme semble très encourageant.

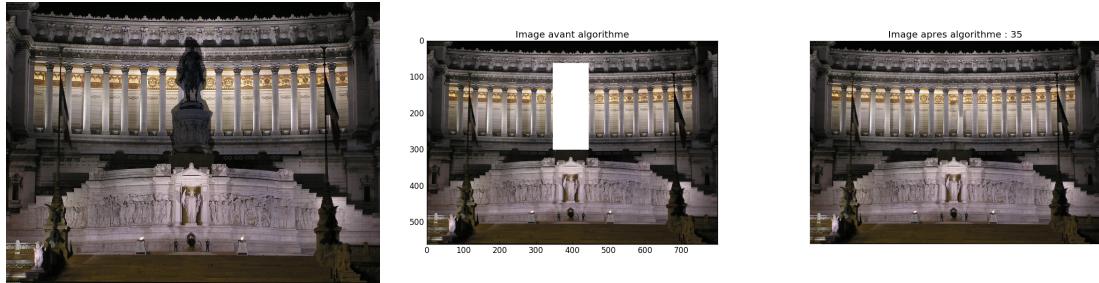


Figure 5: A gauche: image originale, Milieu : Zone a supprimer, a droite : Résultat produit par l'algorithme

Ensuite, nous avons mené des expériences pour mesurer l'influence de la taille du patch sur le résultat final. Le choix de la taille du patch, conformément à nos attentes, s'avère être cruciale dans cet algorithme. En effet, si l'on choisit une taille de patch trop petite alors l'algorithme ne verra qu'un voisinage très petit et ne sera pas en mesure de garder les structures de l'image. Nous pouvons le voir sur la figure 6. À l'inverse, si la taille du patch est trop grande, l'algorithme peinera à créer une grande diversité dans le remplissage et les répétitions de patch en seront moins discrètes.



Figure 6: Taille de patch 11 et 3

Pour montrer simplement que l'algorithme réussit à compléter les structures de façon efficace. Nous avons tenté comme montré par la figure 7 de supprimer un morceau stratégique d'un triangle et d'appliquer l'algorithme pour voir si ce dernier réussit à reconstituer la forme géométrique.

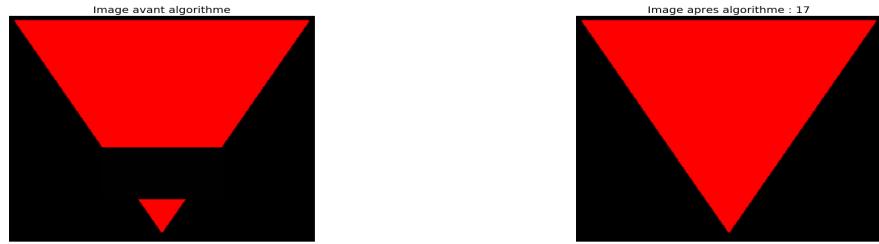


Figure 7: Reconstitution de structure

Nos dernières expériences ont porté sur l'analyse du comportement de l'algorithme sur les images texturées. Ce dernier semble fonctionner aussi bien sur les images texturées que sur les images non texturées comme nous le montre la figure 6. Cependant, sur des textures complexes des artefact peuvent se voir. Les figures 8 et 9 montrent ce phénomène. En effet, sur la diagonale (là où les pixels se rejoignent à la fin) on voit un phénomène apparent de discontinuité.



Figure 8: Apparition d'un artefact _ Taille de patch 11



Figure 9: Apparition d'un artefact _ Taille de patch 5

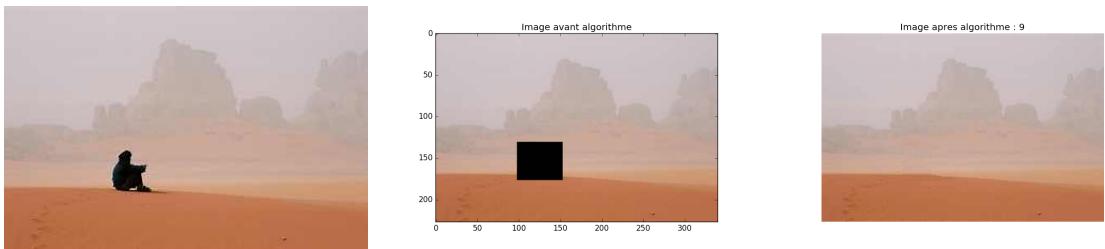


Figure 10: Suppression d'individu dans un décor

7 Critiques et améliorations possibles

Le principal désavantage de cet algorithme est sa lenteur d'exécution. L'étape qui prend le plus de temps dans l'algorithme est l'étape de recherche du meilleur patch ψ_q . En effet, pour faire cela il doit rechercher tout les patchs de l'images ce qui demande beaucoup de puissance. Néanmoins, nous pensons que si nous avions réalisé ce projet en C++, l'algorithme aurait été beaucoup plus rapide et fluide. Pour accélérer le processus, nous avons pensé à ne faire la recherche que dans un certain voisinage du pixel \hat{p} , mais dans les images texturées, cette méthode diminue clairement la qualité du résultat.

Un autre désavantage à cette algorithme est le fait de devoir définir la taille d'un bloc manuellement. En effet, nous avons vu que cette algorithme est très sensible à la taille du patch. Une amélioration possible serait donc d'utiliser un algorithme pour calculer l'épaisseur de la plus grosse structure dans l'image. Un algorithme de segmentation peut permettre de faire cela. Aussi, suivant les régions, il serait intéressant de faire en sorte que la taille du patch varie. Ainsi, dans des régions comportant des structures épaisses on utiliserai des tailles assez grandes, et inversement lorsque qu'il n'y a pas de structure.

Nous avons aussi remarqué que l'algorithme avait des difficultés sur les images bruitées. Nous pensons que la meilleure solution serait d'appliquer un filtre bilatérale à l'image en préalable, mais cela ajouterai évidemment du temps de calcul, et des paramètres en plus à trouver. Nous avons essayé le filtre gaussien, mais celui-ci peut dégrader la qualité des contours et par conséquent l'algorithme devient moins performant.

8 Conclusion

Dans ce travail nous avons étudié un algorithme pour la suppression d'objets sur des images et réussi à l'implémenter tout en comblant les détails manquants de l'article où il a été proposé. Nous nous sommes pour cela basé sur nos connaissances et un ensemble d'expériences qui nous ont permis d'observer le comportement de l'algorithme et de détecter les anomalies. Aux termes de ce travail, il s'avère que l'algorithme que nous avons implémenté est particulièrement efficace pour la suppression de petites, moyennes et grandes régions et produit des résultats comparables à ceux présents dans l'article. Cependant, ce dernier a des faiblesses, notamment: sa sensibilité au bruit et à la taille du patch qui doit être défini manuellement.

References

- [AC03] Kentaro Toyama Antonio Criminisi, Patrick Perez. Object removal by exemplar-based inpainting. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, June 2003.