# CGDI

Florent Guépin et Stéphane Pouget

April 2018

## Introduction

Notre travail sur le projet peut se distinguer en deux étapes : dans un premier temps nous avons travaillé sur l'article [1]. Cela n'a pas fonctionné : des problèmes de définitions implicites dans l'article ne nous permettaient pas de conclure durant la phase d'implémentation. Puis, nous nous sommes interessés à l'article [2]. Pourquoi cet article ? Il nous a semblé plus clair que les autres et présentant de solides possibilités d'implémentation. Nous avons utilisé Python3 pour implémenter notre inpainter afin de privilégier une approche simple au niveau langage de programmation. Cela nous est cependant légérement préjudiciable quand nous faisons touner notre algorithme : les image-résultats peuvent mettre du temps à apparaître. Pour pallier à ce problème et dans un souci de visibilité, nous générons les image-résultats intermédiaires au fur et à mesure.

## Le Fonctionnement de l'Algorithme

L'algorithme se déroule en trois étapes.

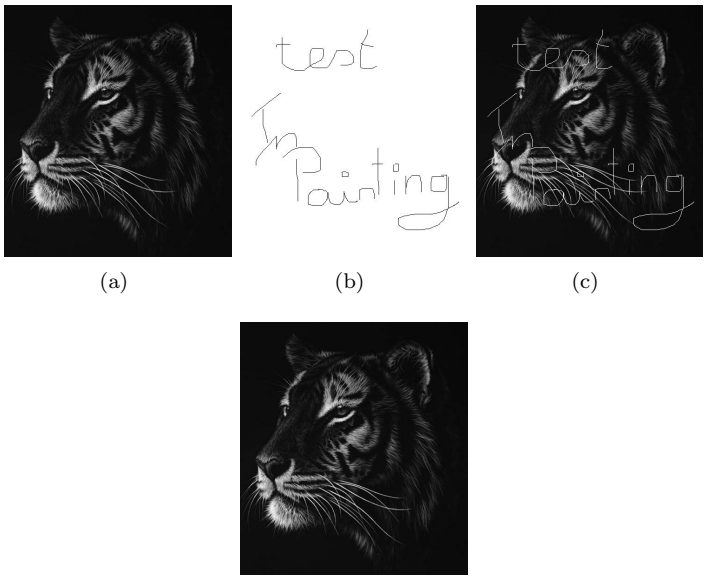
- Pour chaque point $p \in \delta\Omega$ i.e. à la frontière entre l'image et la zone à compléter on crée un patch d'une taille que l'on peut donner en paramètre et l'on calcule une priorité $P(p)$ qui est le produit d'un terme de confidence et d'un terme de données. Le terme de confidence est une mesure de la quantité d'informations fiables entourant le pixel $p$ et le terme de données permet de garder la structure intacte.
- Dans un deuxième temps on cherche un patch de la même taille dans l'image qui permettra de remplir notre nouveau patch. Pour cela on cherche le meilleur patch qui minimise la distance entre les deux.
- Enfin on met à jour les informations et on recommence l'algorithme tant que notre zone n'est pas totalement remplie.

## L'implémentation

La phase d'implémentation est d'abord passée par une compréhension du sujet traité : que devions nous implémenter pour obtenir un résultat cohérent ? Puis, après avoir compris, nous avons implémenté notre inpainter comme présenté dans l'article : les étapes 1a, 1b etc sont autant de correspondances avec nos fichiers Python.

## Les Résultats

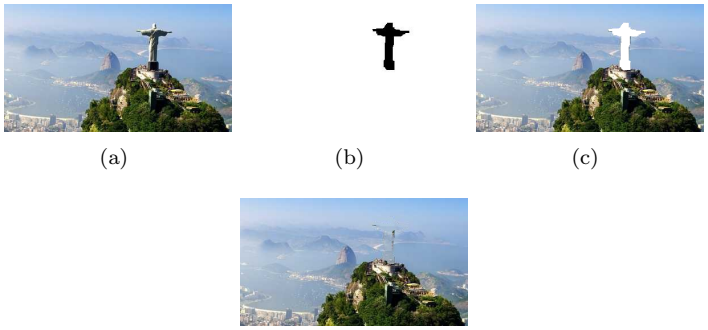
Voici des exemples qui illustrent le fonctionnement de l'algorithme :




(a) (b) (c)

(d)

Figure 1: Résultat sur une image Noir et blanc



(a) (b) (c)

(d)

Figure 2: Résultat sur une image Couleur

Nous vous avons fourni plus d'exemples dans le fichier tests de l'archive.

## Les Points Positifs

Notre algorithme est tout à fait capable de réussir son but : inpainter les images. Il est performant sur des images en noir et blanc, sur des masques fins, tel que de l'écriture ou des rayures. Il est aussi capable d'"effacer" des peronnages/lieu d'un décor à partir du moment où le décor est suffisament riche en information. Notre algorithme est facile d'utilisation, et nous sommes capable d'évaluer son fonctionnement durant sa progression. De plus, nous pouvons gérer le problème de vitesse en augmentant la taille de la fenêtre de regard de l'algorithme, avec cependant une perte significative d'information et donc, de cohérence dans l'image finale.

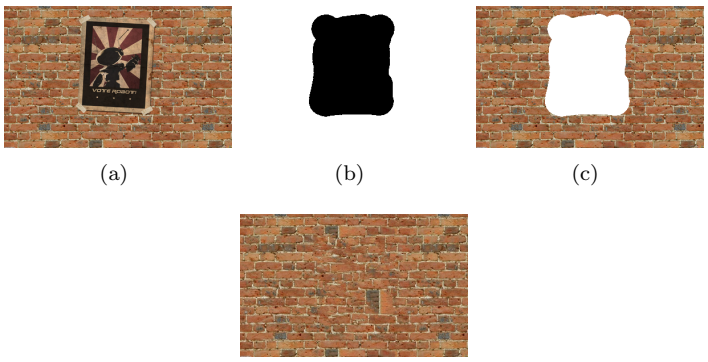
## Les Points Négatifs

Cependant, notre algorithme présente quelques faiblesses lorsqu'il doit inpainter des détails d'une image, en se basant sur ce qui l'entoure (exemple de la corbeille de fruits) chose qu'un autre algorithme d'inpainting, tel qu'un algorithme basé sur un réseau de neurones, serait capable de faire. De plus, nous devons patienter un certain temps pendant le déroulement de l'algorithme : sur une fenêtre de 3 unités notre algorithme peut mettre une soirée à calculer par exemple, l'image du mur (fichier tests/mur). De plus, notre algorithme a du mal à "effacer" lorsque l'image dans laquelle l'élément doit être effacé ne présente pas de détail particulier : par exemple, dans notre fichier tests, nous avons un dossier Opencv, le logo est très simple, sans détail. Alors, cela trompe le principe de confiance de notre algorithme, la conséquence : des taches vertes ou blanches apparaissent au lieu d'être purement et simplement supprimées en transformant le fond en noir.

## Discussion sur l'Approche

Nous avons choisi cet article car il nous semble être un des meilleurs dans le domaine. Il gère trés bien la texture et assez bien les structures. Dans certains cas il ne fonctionne pas efficacement comme pour les fruits, mais c'est un cas extrème où l'utilisateur n'a aucun interêt à employer l'algorithme.

## Un Résultat Probant



(a) (b) (c)

(d)

Figure 3: Résultat sur l'exemple du sujet

## References


[1] Marcelo Bertalmío, Vicent Caselles, Simon Masnou, and Guillermo Sapiro. Inpainting. In *Computer Vision, A Reference Guide*, pages 401–416. 2014.

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