



IMA201

Rapport final du projet: Inpainting

Auteurs:

LUZ BROCHADO, Rodrigo Cézar
BALDAS SKOK, Breno

Email:

rodrigo.luzbrochado@telecom-paris.fr
breno.baldasskok@telecom-paris.fr

October 1, 2020

Contents

1	Introduction	2
1.1	Conventions et nomenclature	2
1.2	Sélecteur de masque	3
2	“Object Removal by Exemplar-Based Inpainting”	4
2.1	La méthode	4
2.1.1	Calcule de priorité de patches	4
2.1.2	Propagation des informations de texture et de structure	6
2.1.3	Mise à jour des valeurs de confiance	6
2.2	Algorithme	7
3	“Non-Local Patch-Based Image Inpainting”	8
3.1	Minimisation du fonctionnel et Reconstitution de l'image	8
3.2	<i>Approximate Nearest neighbor Search</i>	10
3.3	<i>Approche Multi-Échelle</i>	11
3.4	Algorithme	11
4	Résultats	12
4.1	Effet de la variation de la taille du patch	12
4.1.1	Effet sur les textures	14
4.2	Itération de la Méthode	15
4.3	Limitations	17
4.3.1	Positionnement de lignes	18
4.3.2	Information continue constant	18
5	Conclusion	19

1 Introduction

Inpainting est le processus de supprimer une région de l'image et remplir le trou laissé derrière d'une manière visuellement crédible, ce qui permet la reconstitution des parties perdues ou détériorées d'images et de vidéos.

Pour obtenir la reconstitution d'une région occluse, la technique doit être capable de reproduire les informations contenues dans la zone non occultée. Ces informations se situent dans la continuité des formes géométriques, telles que des lignes ou des parties d'objets ou de textures.

Une motivation évidente pour la recherche d'algorithmes d'inpainting efficaces est que si elle est faite à la main, le processus de remplissage d'une partie inconnue d'une image est extrêmement laborieux.

Les applications de la méthode comprennent, sans s'y limiter, la reconstruction d'images endommagées, la suppression de texte dans une image, l'élimination de détails indésirables en passant par le montage vidéo en post-production. Il a également une valeur commerciale pour des logiciels comme Photoshop. Il est important de noter que l'Inpainting ne crée pas une reconstruction parfaite, mais peut être utilisé en combinaison avec d'autres algorithmes et travaux humains pour obtenir de très bons résultats.



Figure 1.1 – Un exemple de Inpainting manuel

Dans ce contexte, ce travail vise à reproduire deux algorithmes décrit dans les articles suivants: “Object Removal by Exemplar-Based Inpainting”[1] et “Non-Local Patch-Based Image Inpainting”[2].

1.1 Conventions et nomenclature

Les conventions utilisées sont les mêmes que celles de l'article “*Non-Local Patch-Based Image Inpainting*”.

- Le support d'image est indiqué par Ω
- \mathcal{H} est la zone d'occlusion
- \mathcal{D} est la région non occultée
- Une position dans le support peut être indiquée par $p = (x, y) \in \Omega$
- Le contenu de l'image elle-même est indiqué par $u : \Omega \rightarrow R^3$
- Voisinage: d'un point du support $\mathcal{N}p$, of cardinality N
- Patch: est dénoté par $W_p = (u(x_1), \dots, u(x_N))$ et correspond à un ensemble de points associés à "p".
- *Nearest Neighbor* (NN) de W_p : est un patch W_q dans un autre ensemble de patches qui minimise une certaine distance de patch $d(W_p, W_q)$
- *Shift Map*: Le plus proche voisin de $W_{p+\phi(p)}$ est $W_{p+\phi(p)}$ où $\phi : \Omega \rightarrow \mathbb{N}^2$

1.2 Sélecteur de masque

Un sélecteur de masque a été conçu pour effectuer le processus d'inpainting.

Le masque est représenté par une image en niveaux de gris de la même taille que l'image à traiter. La zone d'occlusion prend la valeur 0 alors que son complément prend la valeur 255.

Pour générer des masques il faut exécuter la classe **MaskSelector** avec les commands suivants:

```
image = cv2.imread(image address)
MS = MaskSelector(image)
MS.run(address output)
```

Pour sauvegarder une masque, il suffit de selectioner un carré et taper "S".



Figure 1.2 – Sélecteur de Masque et respective masque

2 “*Object Removal by Exemplar-Based Inpainting*”

2.1 La méthode

La méthode employée dans cet article a 3 étapes principales:

1. Le calcul de l'ordre dans lequel les patches doivent être remplis.
2. Après avoir sélectionné un patch, déterminez quel patch à l'extérieur du masque correspond le mieux à sa place.
3. La mise à jour des valeurs de priorité après la substitution d'un patch afin de trouver le nouvel ordre de remplissage.

2.1.1 Calcule de priorité de patches

Le calcul de priorité est calculé comme la multiplication de deux fonctions. La fonction Data, qui représente les informations géométriques contenues dans chaque patch, et la fonction Confiance, qui représente le niveau de confiance des informations à proximité d'un pixel. Il est à noter qu'au départ les pixels de l'image originale sont initialisés avec confiance 1 et les pixels du masque sont initialisés avec confiance 0.

$$P(p) = C(p)D(p)$$

Calcule de Confiance

Comme expliqué précédemment, les valeurs de confiance de l'image sont initialisées à 1 pour les pixels extérieurs au masque et à 0 pour les valeurs intérieures au masque. Dans toutes les itérations de l'algorithme, un contour de masque est sélectionné en prenant la différence entre le masque courant et le masque courant érodé par un élément structurant de taille 3. Ensuite, chaque pixel appartenant à la région du contour a sa valeur de confiance mise à jour avec la moyenne des valeurs de confiance des pixels appartenant au patch centré sur lui.

$$C(p) = \frac{\sum_{q \in W_p} C(q)}{|W_p|}$$

Où la notation introduit dans la introduction est utilisé.

Calcule de Data

La fonction Data est calculée par le produit scalaire du vecteur normal à la surface du masque et le vecteur perpendiculaire au gradient de l'image originale (c'est-à-dire un vecteur idéalement parallèle aux lignes de contour de l'image).

$$D(p) = \frac{\nabla I_p^\perp \cdot n_p}{\alpha}$$

- Pour le calcul des vecteurs normaux du masque, il suffit de calculer le gradient de chaque pixel de l'image du masque (qui est une image noir et blanc) et de normaliser sa valeur.
- Pour calculer le gradient de l'image, on effectue une dilatation du masque et on obtient une région qui entoure le contour et permet de calculer le gradient sans inclure le masque, car il est caractérisé par une discontinuité. On note ce gradient dans la figure 2.1 comme ∇I_p . Une opération d'inversion de coordonnées est effectuée sur le gradient (-y, x) et le gradient des bords est mis à jour avec le gradient du pixel le plus proche du bord de dilatation.

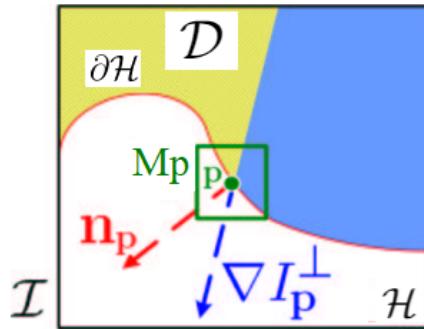


Figure 2.1 – Obtention du valeur de Data pour un patch donné. On

2.1.2 Propagation des informations de texture et de structure

Une fois qu'on a sélectionné le patch qui sera remplacé dans le masque, il est nécessaire de savoir quel patch dans l'image propage le mieux les informations au masque. La méthode utilisée est heuristique et consiste à chercher parmi les patches d'image, ce qui est le plus proche de la partie du patch sélectionné qui a une confiance non nulle. Ce calcul est effectué par une erreur quadratique moyenne.

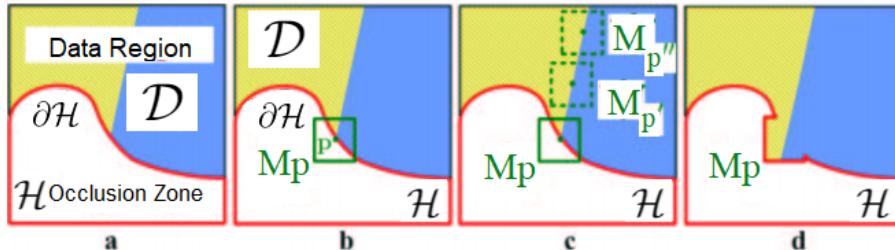


Figure 2.2 – Remplissage dans la zone d'occlusion. (a) On a le bord obtenu par érosion. (b) On obtient par la priorité un patch centré en p (c) On cherche le patch le plus proche dans la région de data. (d) On substitue le patch centré par p et on actualise le confidence de la région remplisso.

2.1.3 Mise à jour des valeurs de confiance

Après avoir rempli un patch, il est nécessaire d'assigner une mesure de la confiance de l'information contenue dans les pixels remplacés. Intuitivement, il est souhaitable qu'une perte de confiance se produise lorsque la peinture avance sur le masque pour prioriser les régions les plus proches de l'image originale. Ceci est obtenu en remplaçant la confiance de tous les pixels du patch par la valeur calculé pour le pixel central du patch.

2.2 Algorithme

Le code a été fait en python3.

Il existe une classe Criminisi_Inpaint qui possède le constructeur qui prend comme argument une image, une masque; la taille du patch (standard est 9); la taille du élément structurante carré pour la opération de érosion (standard est 3, pour obtenir un contour de taille 1); la taille du élément structurante carré pour l'opération de dilatation (standard est 5); Les images peuvent être fournies en couleur ou en niveau de gris;

Certaines des méthodes implémentées sont décrites ci-dessous:

- **apply_erosion(self)**: Crée une liste des pixels qui appartiennent au contour du masque, donné par la différence entre le masque et le masque érodée.
- **apply_dilation(self)**: Crée une liste des pixels qui appartiennent au contour extérieur au masque donné par la différence entre le masque dilatée par un élément structurant de taille 5 (pour éviter la discontinuité donnée par la masque) et le masque dilatée par un élément structurant de taille 3.
- **find_nearest_dilation(self, indexes of contour, indexes of dilation)**: Trouve pour chaque point dans le contour du masque, le point le plus proche dans le countour extérieur, pour que puisse prendre la dérivée de l'image pour le calcule de Data.
- **calculate_confidence(self)**: Calcule la valeur de Confiance pour chaque pixel dans le contour du masque.
- **calculate_data(self)**: Calcule la valeur de Data pour chaque pixel dans le contour du masque.
- **find_best_source(self, current_patch, pat_shape)**: Trouve la meilleure correspondance d'un patch du contour dans \mathcal{D} .
- **inpaint(self, SHOW_STEPS=False, MAX_ITERATIONS=10000)**: La boucle principale de l'algorithme. Le nombre maxime de itération standard est 10000 et l'utilisateur a l'option de visualiser le processus d'inpainting en temps réel, comment dans la figure 2.3.

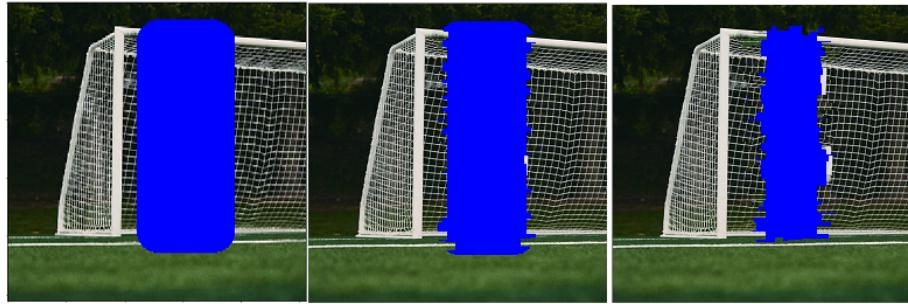


Figure 2.3 – Inpainting en temps réel

3 “*Non-Local Patch-Based Image Inpainting*”

On suivre dans cette partie du travail une approche de minimisation d'un fonctionnel, en utilisant la méthode fournie dans l'article: “Non-Local Patch-Based Image Inpainting. Alasdair Newson, Andrés Almansa, Yann Gousseau, Patrick Pérez” pour améliorer la précision du remplissage.

L'idée principale de cette méthode est de minimiser une fonction après qu'un algorithme d'inpainting a été appliqué à une certaine région d'occlusion \mathcal{H} . Ainsi, on utilise notre implémentation du article “Object Removal by Exemplar-Based Inpainting” comme méthode initiale.

Le fonctionnel que on veut minimiser est:

$$E(u, \phi) = \sum_{p \in \mathcal{H}} d^2(W_p, W_{p+\phi(p)})$$

En définissant la distance carré comme:

$$d^2(W_p, W_{p+\phi(p)}) = \sum_{q \in \mathcal{N}_p} (u(q) - u(q + \phi(p)))^2$$

On procède à la minimisation du fonctionnel:

3.1 Minimisation du fonctionnel et Reconstitution de l'image

$$E = \sum_{p \in \mathcal{H}} \sum_{q \in \mathcal{N}_p} (u(q) - u(q + \phi(p)))^2 \quad (3.1)$$

$$\frac{\partial E}{\partial u(t)} = 2 \sum_{p \in T} (u(t) - u(t + \phi(p)))$$

Où T est l'ensemble des pixels dans l'image dont le patch associé inclut t , ce qui correspond au voisinage du pixel t .

Afin de minimiser E , supposé convexe en relation à $u(t)$, on égale la dérive à zéro:

$$\frac{\partial E}{\partial u(t)} = 0$$

Donc

$$\sum_{p \in T} u(t) = \sum_{p \in T} u(t + \phi(p))$$

Et finalement

$$u(t) = \frac{\sum_{p \in T} u(t + \phi(p))}{N} \quad (3.2)$$

Où N est le nombre de pixels dans l'ensemble T .

Dans l'article, une moyenne pondérée est utilisé pour reconstruire les pixels de l'occlusion. Cependant, nous nous en tenons à la moyenne non pondérée afin de veiller à la simplicité de l'implémentation.

En quelques mots, on peut dire que $u(t)$ est reconstruit comme la moyenne des différentes valeurs candidates, provenant de tous les plus proches des voisins des patchs qui contiennent $u(t)$. Chacun de ces plus proches des voisins a sa propre version de $u(t)$, de sorte que tous sont également pris en compte dans la prédiction de la nouvelle valeur du pixel. Ainsi, le processus de reconstruction de l'image est effectué par l'attribution d'une valeur de couleur à chaque position dans la zone d'occlusion, donné par l'équation 3.2. Une image illustrant ce processus peut être trouvée sur la figure 3.1.

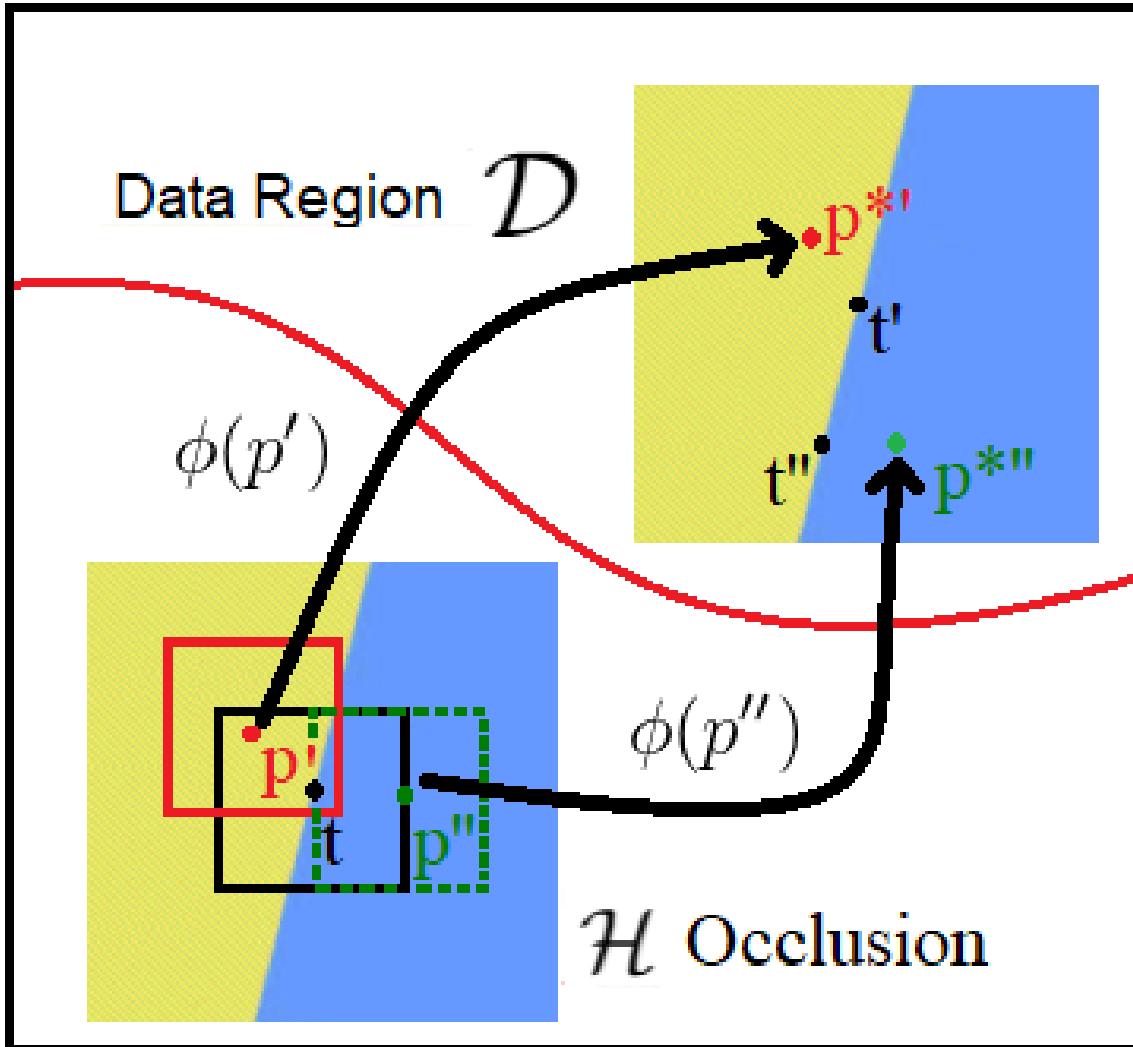


Figure 3.1 – p' et p'' sont des pixels centrales des patches utilisés pour trouver les NN centrés sur p^{**} et $p^{**''}$. On trouve avec son position les points t' et t'' qui seront utilisés dans la moyenne pour trouver un nouveau valeur pour $u(t)$

3.2 Approximate Nearest neighbor Search

La recherche d'un "Nearest Neighbour" (NN) est coûteuse si on veut des temps d'exécution plausibles. On doit donc utiliser une approximation, le "Approximate Nearest Neighbour" (ANN) au lieu des plus proches voisins exacts. Pour cela, l'algorithme *Patch Match*[3] est utilisé pour rechercher les voisins les plus proches. Cet algorithme s'appuie sur l'idée que les bonnes "shift maps" ont tendance à être constantes par morceaux. En d'autres termes, un décalage relatif qui conduit à un

bon NN pour un patch W_p a une bonne chance d'aboutir à un bon NN pour les patchs situés autour de W_p .

3.3 Approche Multi-Échelle

Pour qu'on puisse éviter les minima locaux du fonctionnel (eq. 3.1), on utilise une approche multi-échelle. Ainsi, l'image et la masque sont sous-échantillonnage avec un facteur $\frac{1}{2}$ et on utilise un filtre gaussien 3x3 avec écart type de 1.5 pixels pour éviter l'aliasing. Pour chaque niveaux de résolution, on itère l'algorithme pour améliorer la solution du inpainting et après on passe au niveau suivante (plus grande résolution).

Quant à la façon dont les solutions calculées pour l'inpainting sont transmises aux niveaux de résolution plus élevés de la pyramide, cela se fait par interpolation 2D par plus proche voisins du *shift map* ϕ . Le choix d'interpoler le *shift map* au lieu de l'image résultante a été fait pour éviter le flou qui en résulterait. Ensuite, la solution initiale est obtenue en reconstruisant l'image à la nouvelle résolution avec l'équation 3.2.

3.4 Algorithme

Le code a été fait en python3.

Il existe une classe `Newson_Inpaint` qui possède le constructeur qui prend comme argument une image, une masque et la taille du patch pour l'inpainting initial et celui de la méthode (standard est 9).

Certaines des méthodes implémentées sont décrites ci-dessous:

- **`initialization(, SHOW_STEPS=True)`**: Crée les plusieurs échelles de résolution de l'image et la masque et effectue l'inpainting initial.
- **`upScale(self)`**: Effectue l'interpolation du *shift map* avec le plus proche voisin.
- **`get_shift_map(self, scale, upscale = False)`**: calcule le *shift map* avec la fonction `patch_match` qui nous a été fourni.
- **`get_coords(self, scale)`**: calcule des coordonnes des plus proches voisins de chaque pixel dans le masque.
- **`get_new_values(self, scale, SHOW_STEPS = False)`**: implemente l'équation 3.2.
- **`inpaint (self, SHOW_STEPS=False, N_scale = 4, N_it = 10, err_tol = 0.1)`**: La boucle principale de l'algorithme. L'utilisateur peut choisir le

nombre d'échelles et d'iterations par échelle. Il a aussi l'option de visualiser le résultat d'inpainting de chaque itération.

4 Résultats

Bien que nous l'ayons implémenté, nous n'avons pas pu résoudre une erreur dans le code se référant à l'article “Non-Local Patch-Based Image Inpainting”[2] (la valeur du voisin le plus proche trouvée ne correspondait pas à une bonne approximation, ce qui a détérioré le résultat de la méthode), ce qui a rendu impossible la génération de résultats. Ainsi, les résultats affichés ci-dessous et leur analyse ne concernent que l'article “Object Removal by Exemplar-Based Inpainting”[1].

4.1 Effet de la variation de la taille du patch

On veut effacer la personne dans la plage et analyser l'effet de la taille du patch:

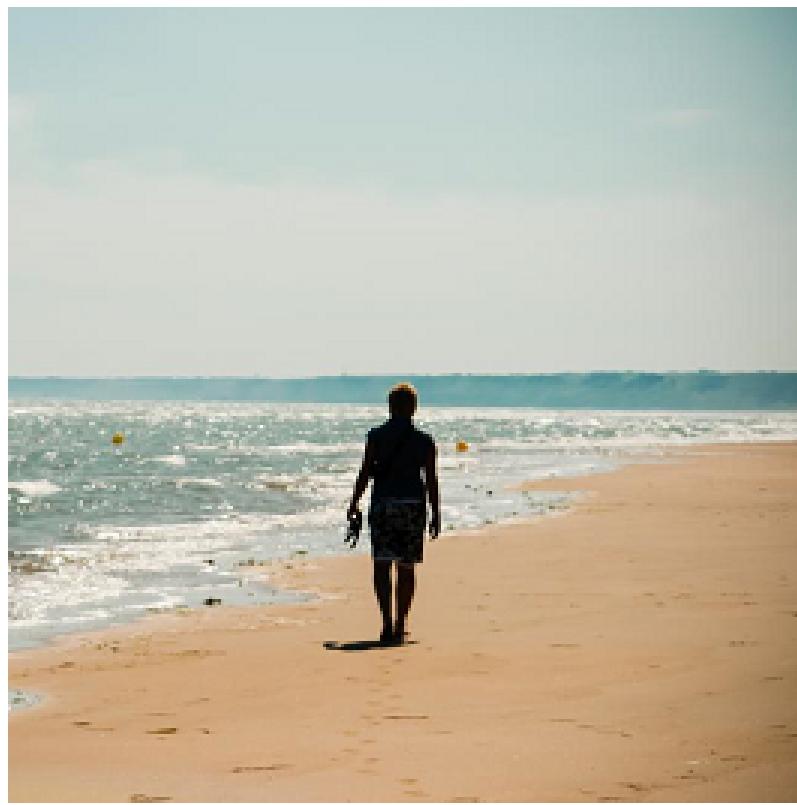
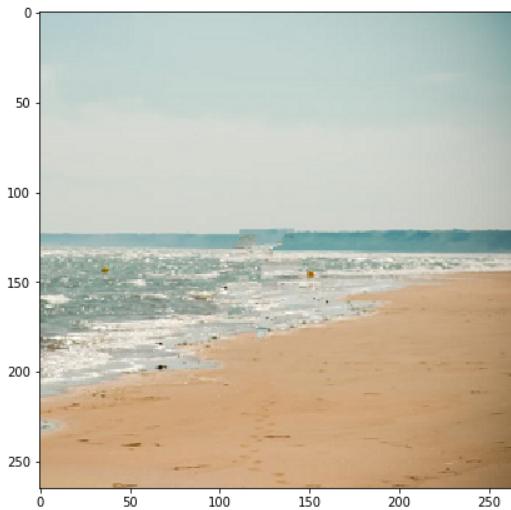
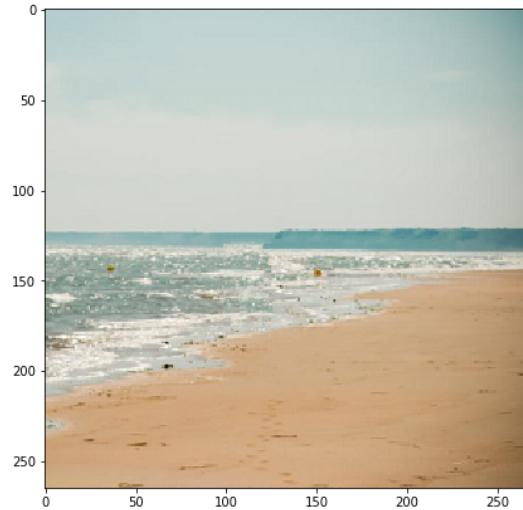


Figure 4.1 – beach.png

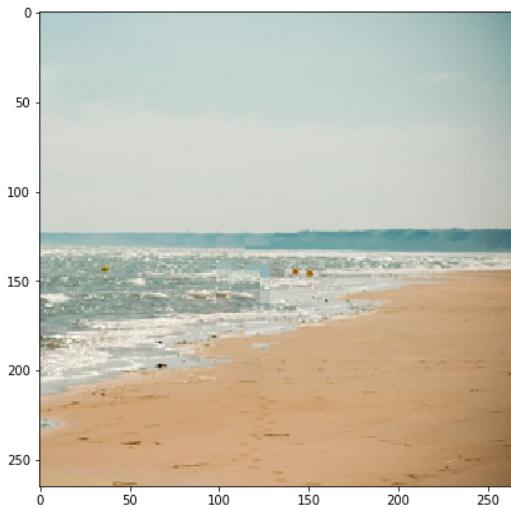
Les résultats sont montrés ci dessous:



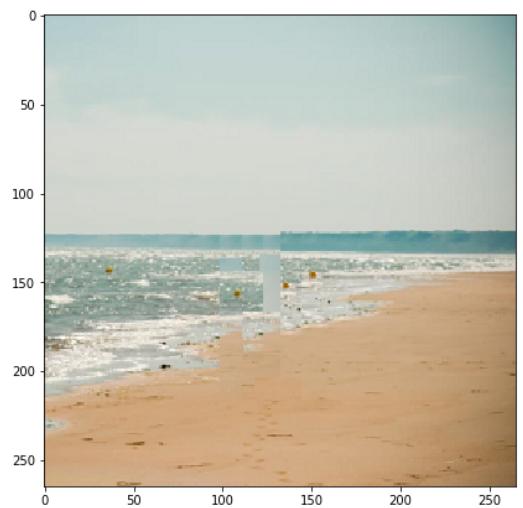
(a) taille de patch = 5; Itérations : 430



(b) taille de patch = 7; Itérations : 235



(c) taille de patch = 9; Itérations : 156



(d) taille de patch = 23; Itérations : 28

Nous pouvons voir qu'à mesure que le patch s'agrandit, la discontinuité entre deux régions remplis devient plus évidente. D'autre part, à mesure que les patches deviennent plus petites, en plus d'augmenter les coûts de calcul, les défauts et le bruit commencent à être répliqués en tant qu'informatisons.

Dans le cas de cette image, un patch de taille 7 semble reproduire de bons résultats.

4.1.1 Effet sur les textures

On travaillera maintenant avec l'image d'un tableau de échec. On mesure la taille moyenne de chaque carré comme 32×25 pixels.



Figure 4.3 – chess.png

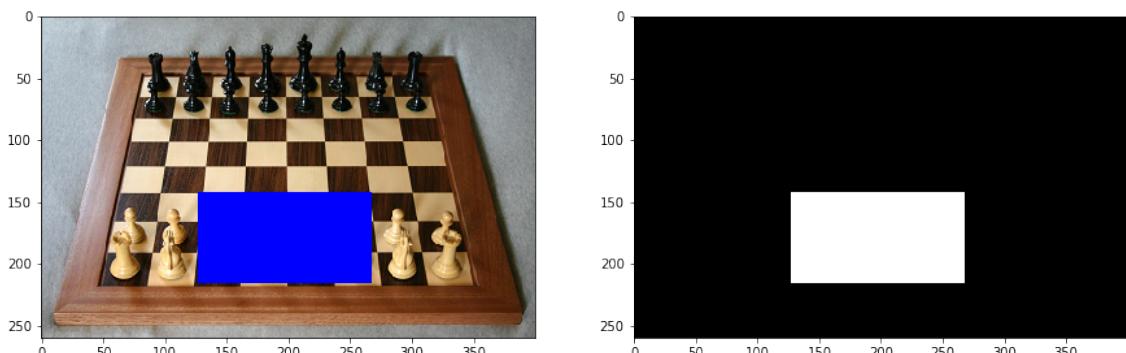


Figure 4.4 – Masque appliquée

Les résultats sont montrés ci dessous:



(a) taille de patch = 11



(b) taille de patch = 21



(c) taille de patch = 25



(d) taille de patch = 33;

Lorsque la taille du patch est inférieure à la taille du motif de la table de échec, une autre motif de taille plus petite semble apparaître et les pièces sont répliques. Il est possible de réaliser que, en général, l'augmentation de la taille jusqu'à au moins la taille de la texture principale de l'image génère des résultats plus fiables.

4.2 Itération de la Méthode

Il est possible d'itérer la méthode avec des masques en sélectionnant des régions où l'inpainting n'a pas fait une bonne reconstruction pour obtenir de meilleurs résultats. Les résultats ont été obtenus avec la taille du patch 23 (plus grand que la texture derrière du gardien de but)

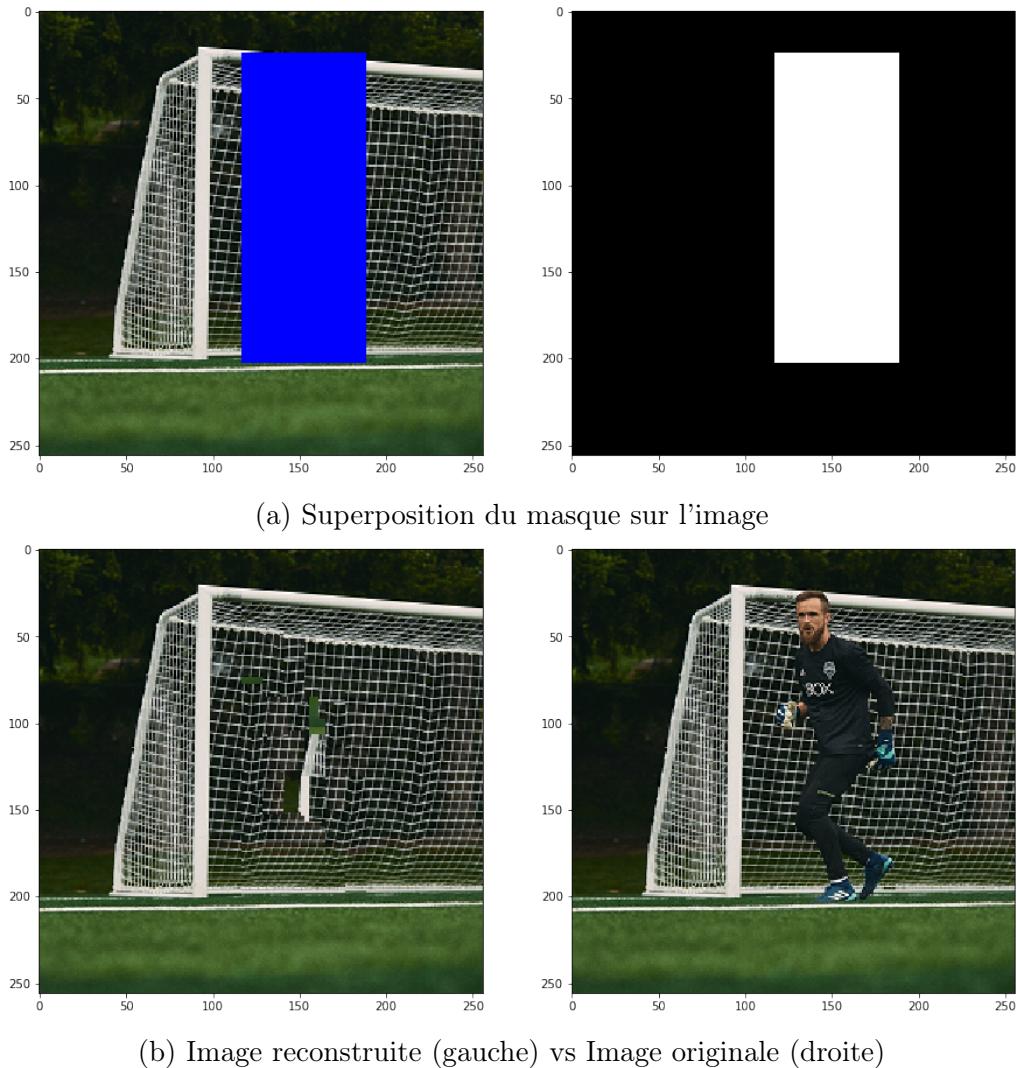


Figure 4.6 – Résultats de la méthode sur une image complexe.

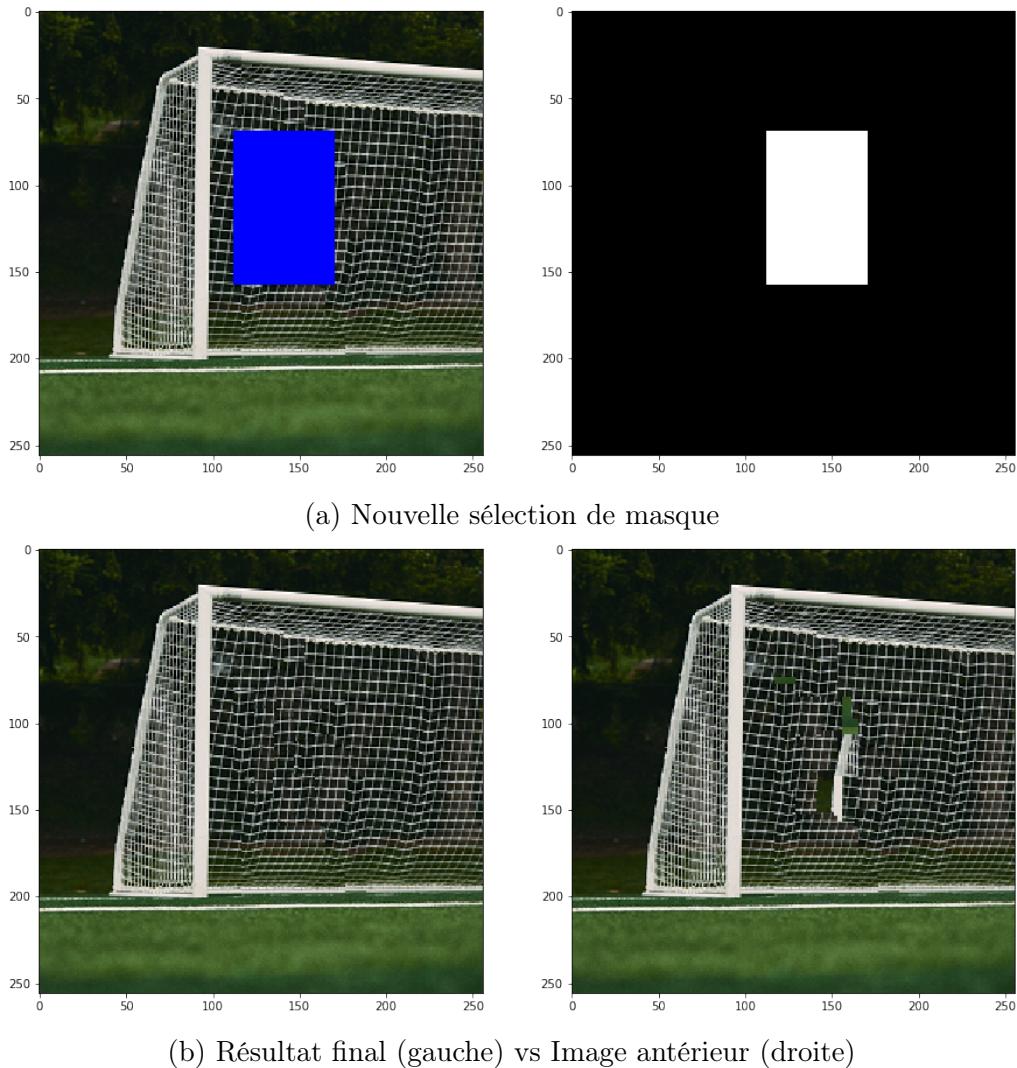


Figure 4.7 – Deuxième itération du méthode de inpainting.

4.3 Limitations

Les deux exemples suivants montrent certaines des limites de la méthode.

4.3.1 Positionnement de lignes

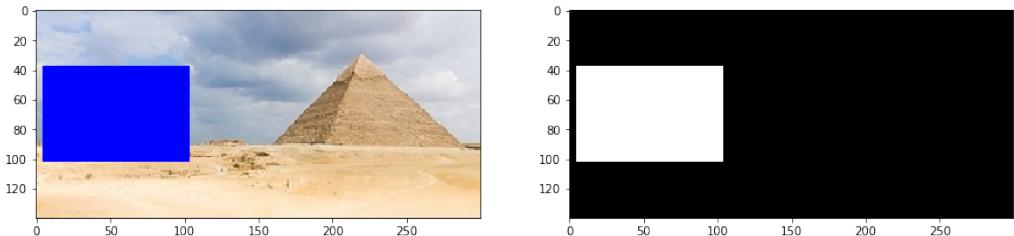


Figure 4.8 – Masque appliquée

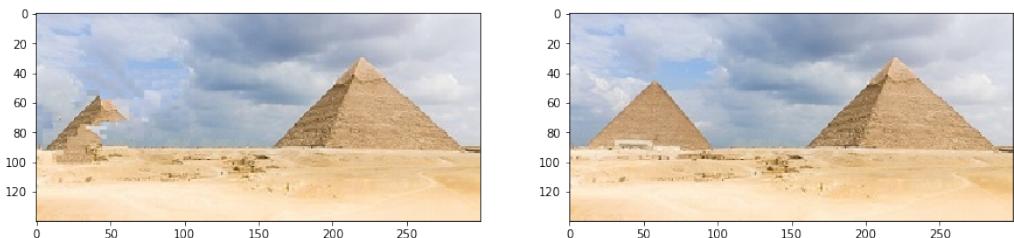


Figure 4.9 – On a essaye de enlever la pyramide à gauche de l'image a gauche.

Dans le cas ci-dessus, un patch correspondant au côté gauche du masque commence à faire apparaître une nouvelle pyramide. L'utilisation de la fonction DATA permet de poursuivre ce remplissage. Cependant, lorsque l'algorithme traitera le côté droit du masque, il décide de continuer avec la ligne d'horizon. Cela va nécessairement générer une reconstruction irréalistique.

Ce problème se produit lorsqu'il y a différentes lignes d'informations entrant dans la zone du masque et qu'il n'est pas possible de le résoudre simplement avec la méthode utilisée.

4.3.2 Information continue constant

Selon la théorie de l'information, on peut penser qu'une image est un ensemble de variables aléatoires. Si X représente l'ensemble des positions dans la "Data Region", et la zone remplie après l'algorithme est une fonction de X , c'est à dire, $f(X)$, on a que l'entropie de X sachant $f(x)$ est toujours $H(x)$

$$H(X|f(X)) = H(X)$$

et l'information mutuelle entre la zone remplie et l'image est donné par

$$I(X, f(X)) = H(X) - H(X|f(X)) = 0$$

Donc, il est impossible de s'attendre à ce qu'elle génère de nouvelles informations. Dans le cas ci-dessus, la trompe de l'éléphant ne pourrait pas être produite sans une base de connaissances.

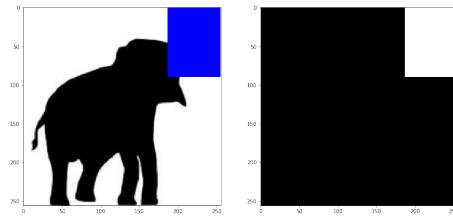


Figure 4.10 – Masque appliquée

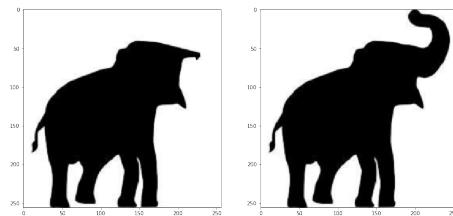


Figure 4.11 – Le masque était sur la trompe de l'éléphant, ce qui a généré une erreur de remplissage

5 Conclusion

Les méthodes d'inpainting analysées et implémentées dans ce projet sont des outils très puissants, et témoignent de l'ingéniosité qui existe dans le traitement de l'image. Comme on l'a vu, il est possible de combiner l'algorithme de différentes manières, comme en réalisant de différentes itérations pour obtenir une meilleur résultat ou l'utilisant comme base pour créer des méthodes encore plus efficaces.

Comme on pouvait s'y attendre, leur application comporte des limites, mais ils restent utiles pour un grand nombre de cas.

References

- [1] A. Criminisi, P. Perez K. Toyama. *Object Removal by Exemplar-Based Inpainting*. 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.
- [2] Alasdair Newson , Andrés Almansa , Yann Gousseau , Patrick Pérez. *Non-Local Patch-Based Image Inpainting*. Published in Image Processing On Line on 2017-12-13.
- [3] Connelly Barnes, Eli Shechtman, Adam Finkelstein, Dan B Goldman, *PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing* Published in ACM Transactions on Graphics (Proc. SIGGRAPH), August 2009
- [4] Y. Wexler, E. Shechtman, and M. Irani
Space-time video completion n Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, 2004, pp. 120–127.