

Projet Inpainting (4IM01) : Rapport final

Object Removal by Exemplar-Based Inpainting

Daniel Akbarinia
Abdennour Kerboua

24 novembre 2024

1 Présentation de la méthode

L'inpainting basé sur des exemples (exemplar-based inpainting) est une technique de restauration d'images par patch qui comble les zones manquantes en recopiant des parties similaires de l'image intacte. Cette approche permet de préserver à la fois les textures et les motifs visuels, garantissant ainsi un résultat plus réaliste. L'article "Object Removal by Exemplar-Based Inpainting" (2003) propose une méthode novatrice combinant une solution de type diffusion, qui met l'accent sur la reconstruction des contours et des formes géométriques, avec une approche basée sur des exemples pour reproduire fidèlement les textures. Cette technique établit l'ordre de remplissage des pixels en fonction du gradient local, tout en sélectionnant les patchs de remplissage directement dans l'image d'origine, assurant ainsi une intégration harmonieuse des zones restaurées.

2 Implémentation de la méthode

L'article décrivant la méthode proposant un processus détaillé par étape, nous avons décidé de décomposer l'implémentation suivant cette décomposition. Voici une brève description de l'implémentation de chaque fonction étape de l'algorithme.

1. a. **Identification du front de remplissage** (noté $\delta\Omega^t$ dans l'article) : on a implémenté deux méthodes. La première consiste à maintenir à jour une matrice indiquant pour chaque pixel son état de remplissage (`True` pour rempli, `False` pour en attente de remplissage). A partir de cette matrice, on décide pour chaque pixel, s'il appartient au front de remplissage par 8-connexité (un pixel $p_{i,j}$ non rempli appartient à $\delta\Omega^t$ si et seulement si $\exists k, l \in [i-1, i+1] \times [j-1, j+1]$, tel que $p_{k,l}$ est rempli). La seconde est une variante équivalente qui convole le masque (valant 1 en les pixels remplis et 0 autre part) avec le filtre :

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

En prenant l'intersection des pixels non-remplis dans l'image de départ et des pixels strictement positifs dans l'image résultante de la convolution, on obtient le front de remplissage. Les deux méthodes sont équivalentes et entraînent en pratique des temps de calculs très proches.

- b. **Calcul des priorités** : Ce calcul implique le calcul de deux facteurs : la confidence $C(p)$ et le terme *data* $D(p)$,

- **Coefficient Confidence** : On maintient à jour une matrice `confidence` de même taille $H \times W$ que l'image qui contient pour chaque pixel $p_{i,j}$ la valeur de la confiance en ce pixel. Elle est initialisé en affectant 0 aux pixels de la zone cible et 1 à ceux de la zone source.

Au début de chaque itération du programme, la priorité des pixels du contour actuel est calculé suivant cette formule :

$$C(p) = \frac{\sum_{q \in \Psi_p \cap \Omega^c} C(q)}{|\Psi_p|} \quad (1)$$

où Ψ_p est le patch centré en p et Ω est la zone à remplir.

Ce terme permet de favoriser le remplissage en cercle concentrique.

- **Coefficient Data** : Il s'obtient à partir de cette formule : $D(p) = \frac{|\vec{\nabla}I^\perp \cdot \vec{n}_p|}{\alpha}$. Avec,
 - \vec{n}_p : le vecteur normal au contour de remplissage au pixel p , on le calcule en prenant le gradient au point p du masque (valant 0 aux pixels non-remplis et 1 ailleurs)
 - $\vec{\nabla}I^\perp$: l'orthogonal du vecteur gradient au point p . La surface manquante étant noircie, un calcul usuel de différences au point p peut donner lieu à des valeurs aberrantes. Si l'accès à l'image d'origine est possible (dans le cas d'une suppression volontaire) le calcul du gradient peut se faire sur cette dernière. Sinon, nous avons implémenté un calcul moyennant le gradient sur les pixels de la zone remplie dont le calcul ne fait pas intervenir les pixels de la zone noire (non-remplie). On effectue pour le calcul des gradients, hors le contour, une convolution de l'image avec les filtres de Sobel.

2. a. On obtient alors le terme de priorité pour chaque pixel du contour et on accède facilement au pixel de plus grande priorité. On sauvegarde alors le patch cible Ψ_p de taille fixée en début de programme (le plus souvent 9×9)
 - b. La fonction **distance** est implémentée de façon à renvoyer la SSD entre un patch cible Ψ_t et un patch source Ψ_s par des calculs élémentaires tout en évitant de prendre en compte les pixels non-remplis dans le calcul de la SSD. On trouve alors le meilleur patch en parcourant tous les patchs de la région source et en retenant le patch Ψ_s tel que la valeur $\text{distance}(\Psi_p, \Psi_s)$ est minimale. La recherche est l'étape la plus chronophage de l'algorithme. On limite ainsi la recherche du patch à une zone circonscrite autour de la zone cible Ω dont le rayon est modulable.
 - c. Etape triviale copiant les pixels du patch complet idéal vers les pixels non-remplis du patch cible.
3. On met à jour la matrice **priority** avec la formule : $C(\mathbf{q}) = C(\hat{\mathbf{p}}) \quad \forall \mathbf{q} \in \Psi_{\hat{\mathbf{p}}} \cap \Omega$.

3 Critique de la méthode

3.1 Utilité du terme priorité $P(p)$

3.1.1 Coefficient Data

On peut observer la participation du coefficient au remplissage de la zone manquante.

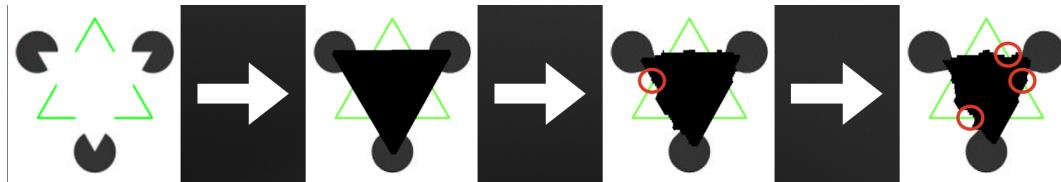


FIGURE 1 – Participation du coefficient $D(p)$

L'influence du coefficient *Data* sur le résultat visuel semble être très minime. Les auteurs de la méthode prétendent que sa présence dans l'expression du coefficient priorité $P(p)$ définissant l'ordre de traitement

permet la reproduction des contours et formes de l'image. On applique la méthode sur des images de formes synthétisées en incluant ou non le coefficient *Data* et les résultats sont très similaires :

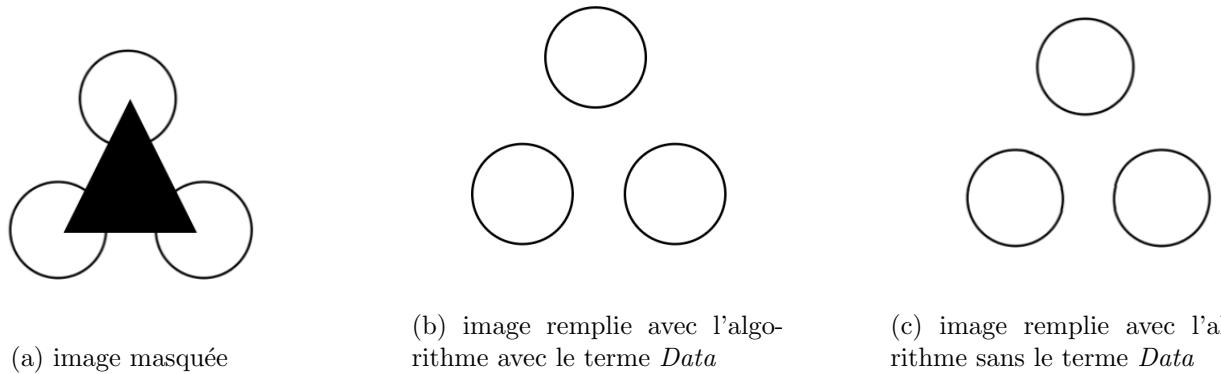


FIGURE 2 – Résultats avec et sans le coefficient *Data* avec une taille de patch 11×11 pixels

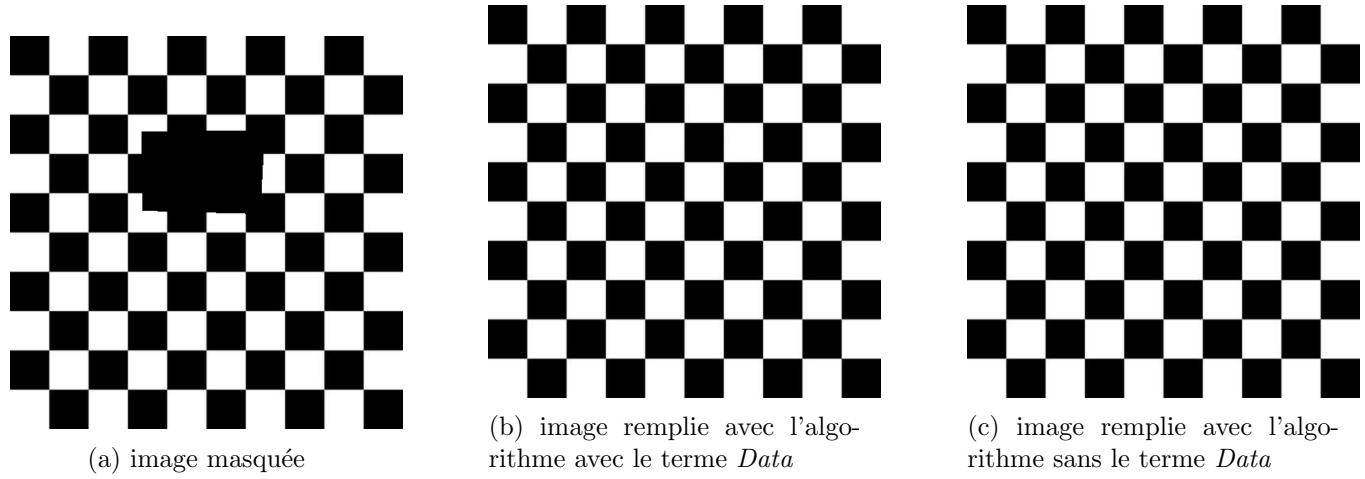


FIGURE 3 – Résultats avec et sans le coefficient *Data* avec une taille de patch 11×11 pixels

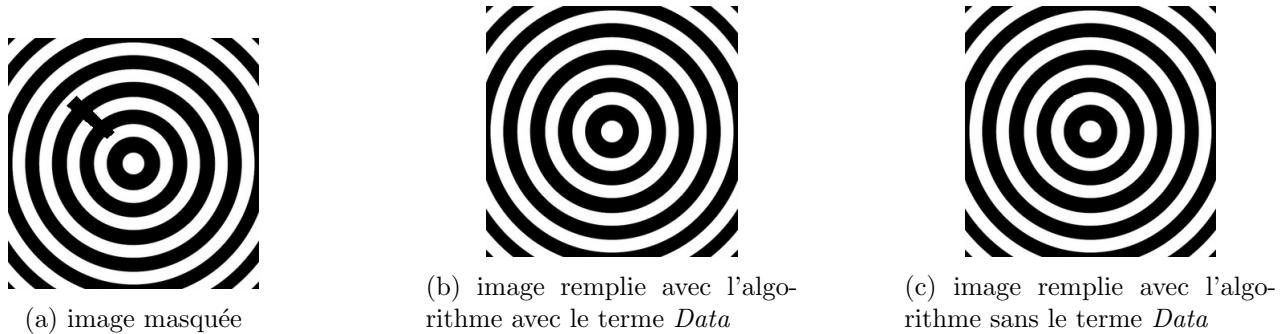
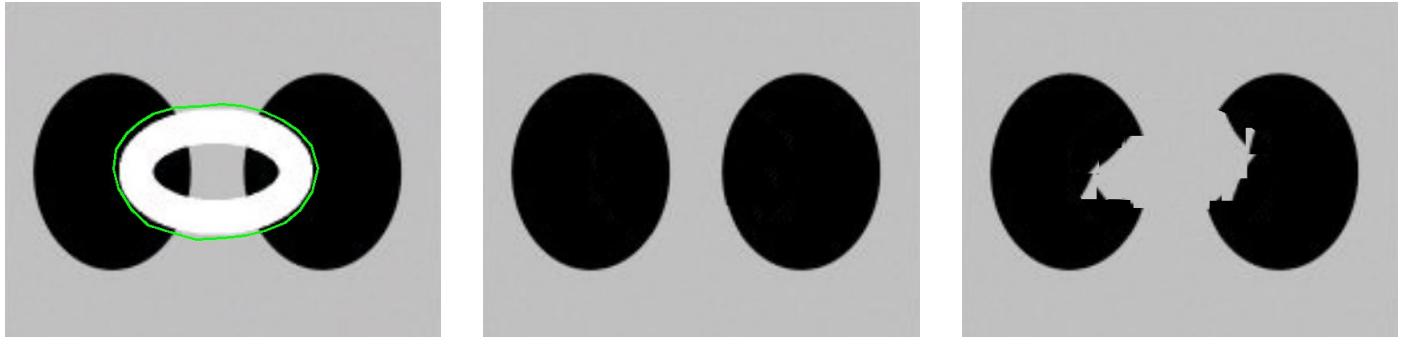


FIGURE 4 – Résultats avec et sans le coefficient *Data* taille de patch : 11×11 pixels

Dans un seul des cas que nous avons testé, le coefficient *Data* impacte la qualité du résultat. Dans l'image suivante en favorisant le remplissage par les bords des deux ellipses on évite un résultat non naturel. Voir figure 5.

Ces résultats sont indépendants de la méthode de calcul du gradient. On observe le même impact du



(a) image avec contour sélectionné

(b) image remplie avec l'algorithme avec le terme *Data*

(c) image remplie avec l'algorithme sans le terme *Data*

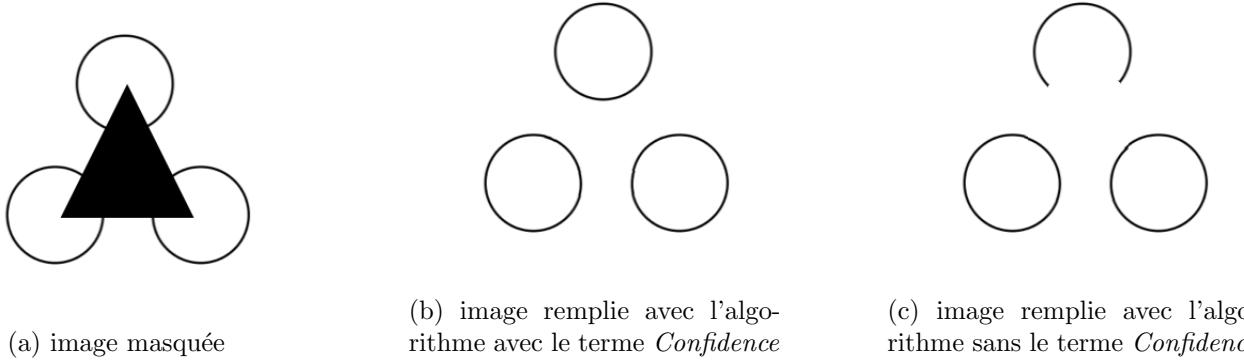
FIGURE 5 – Résultats avec et sans le coefficient *Data* taille de patch : 63×63 pixels

coefficient *Data* peu importe si on le calcule à partir de la méthode décrite à l'étape **1.** **b.** section 2 ou en calculant le gradient directement sur la figure d'origine, y compris dans les résultats présentés figure 5.

3.1.2 Coefficient *Confidence* : remplissage en cercle concentrique

Le coefficient *Confidence* (confiance) noté $C(p)$ évalue la confiance attribué à la connaissance d'un pixel p . On aura d'autant plus confiance en un pixel qu'il est proche de la source initiale (où $C(p)$ est maximal égal à 1). On évalue alors les pixels par coefficient de confiance décroissant. L'influence de ce coefficient varie selon les images inpaintées.

Dans certain cas, la symétrie de l'image semble nécessiter de procéder au remplissage par cercles concentriques, ce qui est favorisé par l'utilisation du terme *Confidence*. On le voit dans le cas figure 6.



(a) image masquée

(b) image remplie avec l'algorithme avec le terme *Confidence*

(c) image remplie avec l'algorithme sans le terme *Confidence*

FIGURE 6 – Résultats avec et sans le coefficient *Confidence* taille de patch 11×11 pixels

Lors de l'exécution du programme sans le coefficient $C(p)$, le remplissage commence par le bas du triangle noir. Une fois le triangle rempli de blanc du bas jusqu'au début du cercle supérieur, la partie blanche remplie du triangle influence le calcul de distance entre les patchs sur le contour supposé du cercle et le cercle n'est pas complété.

Cependant, dans le cas figure 7 le remplissage en cercle concentriques peut altérer la performance de l'algorithme. En effet, le remplissage par le bas de la partie manquante permet d'éviter l'apparition d'artefacts propagés par le second panneau.

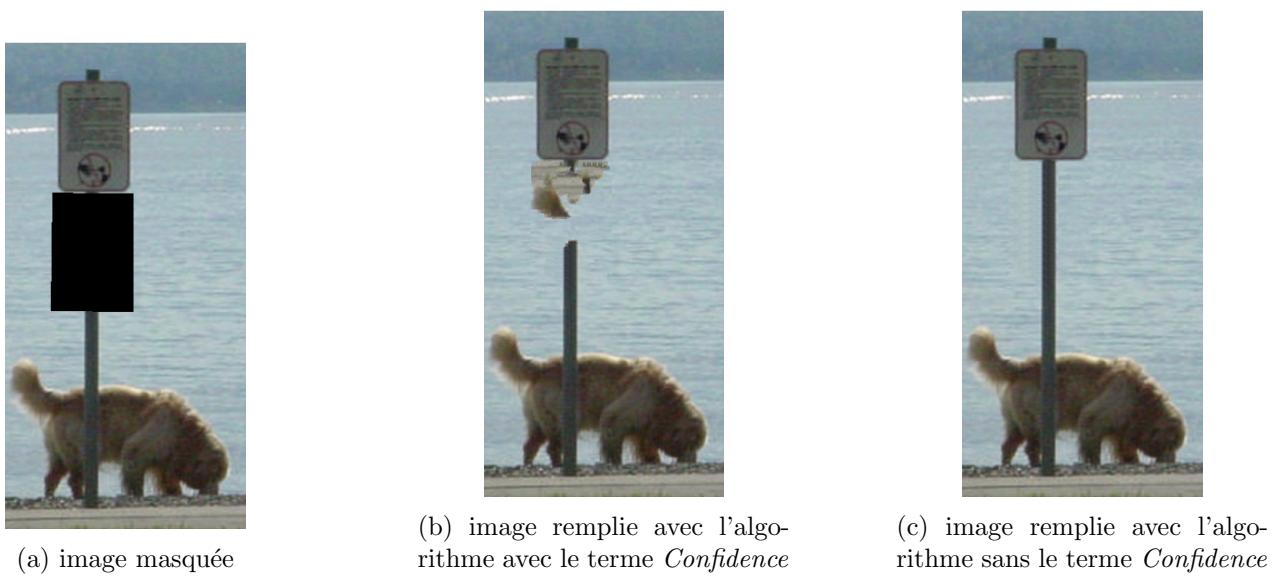


FIGURE 7 – Résultats avec et sans le coefficient *Confidence* taille de patch 11×11 pixels

3.1.3 Conclusion

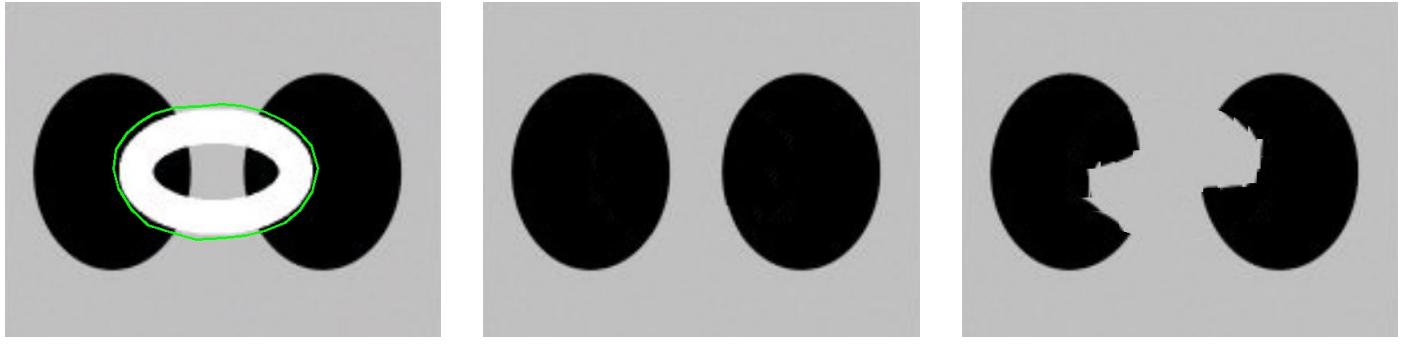
Dans la plupart des cas, le terme de priorité n'affecte que très peu le résultat de l'inpainting, mais en général il améliore la régularité, la reproduction des formes et l'uniformité des textures. En tout cas, il faut savoir adapter leur utilisation aux spécificités de chaque image pour optimiser la qualité du rendu.

3.2 Influence de la taille du patch

La taille du patch a différents impacts sur l'exécution du programme. Premièrement, plus le patch est petit plus le temps de remplissage augmente. En effet, la recherche considère un nombre plus élevé de patch et réduit la surface de zone cible remplie à chaque itération. D'autre part, la taille du patch a une influence sur la qualité de l'image. Plus le patch est petit plus il permet de capturer des détails fins, mais peut introduire des incohérences globales (des formes non naturelles), surtout si la zone à remplir est grande. À l'inverse, plus le patch est grand plus il favorise la continuité et la cohérence globale, mais peut manquer de précision dans les détails ou introduire des artefacts si le contexte local n'est pas bien adapté. Le choix de la taille dépend donc d'un compromis entre la préservation des détails et la cohérence de la structure globale de l'image, en gardant en tête l'impact sur le temps de calcul. Un exemple sur la qualité de l'image est donné figure 9. Le temps d'exécution peut varier entre 2-5 min pour une taille de patch autour de 20×20 pixels à plusieurs dizaines de minutes pour des tailles de patch autour de 5×5 pixels.

3.3 Limites de la méthode décrite

La méthode décrite incluant le calcul des termes $C(p)$ et $D(p)$ présente certaines limites quand il s'agit de remplir des zones manquantes dans des images avec plusieurs formes complexes. Voici des résultats sur l'exemple du triangle de "Kanizsa" abordé dans l'article :

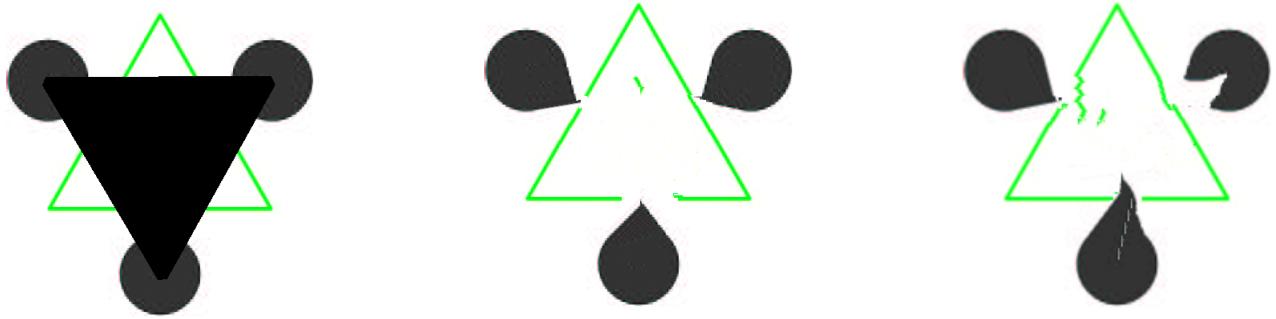


(a) image avec le contour sélectionné

(b) image remplie avec l'algorithme taille patch 63×63

(c) image remplie avec l'algorithme taille patch 35×35

FIGURE 8 – Impact de la taille du patch



(a) image avec le contour sélectionné

(b) image remplie avec l'algorithme taille patch 7×7

(c) image remplie avec l'algorithme taille patch 11×11

FIGURE 9 – Remplissage du triangle de Kaniza

4 Autres résultats

On présente ici des résultats sur des images naturelles en couleur dont la qualité du rendu est tout à fait satisfaisante. Les textures sont bien reproduites et dans le cas de la figure 12, la structure de la petite maisonnette est bien reproduite et racordée.



(a) Image de départ



(b) image remplie avec l'algorithme

FIGURE 10 – Image d'un lion en couleur



(a) Image de départ



(b) Image remplie avec l'algorithme

FIGURE 11 – Image d'un bateau en couleur



(a) Image de départ



(b) Image remplie avec l'algorithme

FIGURE 12 – Image d'un saut en couleur

5 Conclusion et Amélioration apportée

En conclusion, à ce point de l'implémentation notre algorithme donne de très bons résultats sur des images en noir et blanc et des images de formes basiques et en couleurs aux textures et formes simples.

Une amélioration apportée à la méthode décrite a été la circonscription de la recherche du patch source optimale à une zone entourant la source à remplir. Cela permet d'une part de diminuer grandement le temps de calcul mais aussi dans une grande majorité des cas améliorer le rendu visuel du résultat. Par exemple la figure 9b est obtenu en restreignant la recherche à un carré 50 pixels autour de la zone de recherche et la figure 9c en effectuant la recherche sur l'entièreté de l'image.

Cela dit, des méthodes d'inpainting plus récentes offrent désormais des résultats équivalents, voire supérieurs, tout en réduisant significativement les temps d'exécution. Dans cette perspective, nous proposons d'explorer et d'implémenter une approche basée sur Patch-Match, un algorithme rapide et efficace pour la recherche de correspondances entre patchs, qui a été largement adopté dans les travaux récents d'inpainting. Cette méthode repose sur une recherche approximative des patchs similaires et ouvre la voie à des restaurations plus rapides et robustes, tout en conservant une grande qualité visuelle.

6 Complément : Inpainting par PatchMatch

Cet algorithme, introduit par Barnes, Shechtman, Finkelstein et Goldman dans leur article *PatchMatch : A randomized correspondence algorithm for structural image editing* (2009), permet d'établir une carte de correspondances entre les pixels de la région à compléter, notée Ω , et ceux de la région extérieure, notée Ω^c .

Dans ce qui suit, nous commencerons par expliquer le fonctionnement général de l'algorithme PatchMatch, en considérant le cas de deux images distinctes. Ensuite, nous détaillerons son application spécifique au problème de l'inpainting.

6.1 Algorithme de base

L'algorithme PatchMatch a pour but de calculer une carte de correspondance (Nearest Neighbor Field, **NNF** en anglais) entre deux images I_1 et I_2 qui associe à chaque patch de I_1 son plus proche voisin dans I_2 selon une fonction de similarité comme la SSD. On choisit arbitrairement une taille de patch comme 9x9.

L'algorithme PatchMatch est divisé en 3 étapes :

- **Initialisation :** Nous initialisons la carte de correspondance NNF de manière aléatoire. A chaque patch valable de la région cible, nous associons un patch valable de la région source de manière aléatoire. Nous implémentons la carte de correspondance NNF sous la forme d'un tableau numpy, qui prend les coordonnées d'un patch cible valide et renvoie les coordonnées d'un patch source valide ainsi que la SSD entre ces deux patch.
- **Propagation :** On effectue une étape de propagation dans la région cible qui consiste en deux passes de haut en bas et de gauche à droite puis dans le sens inverse. Pour la première passe, si on note p un pixel constituant un patch valide de la région cible et $NNF[p]$ le patch source vers lequel p mène alors si $p + \delta$ avec $\delta = (-1, 0)$ ou $(0, -1)$ mène vers un patch source avec un SSD meilleur que $NNF[p]$, on met à jour $NNF[p]$ avec le même patch que $NNF[p + \delta]$ sinon, on ne fait rien. On fait la même chose pour la deuxième passe avec $\delta = (1, 0)$ ou $(0, 1)$.
- **Recherche aléatoire :** Pour chaque pixel p de la région cible formant un patch valide, on affecte un décalage aléatoire à $NNF[p]$ toujours dans la région source et on regarde si le patch associé à ce décalage donne une SSD plus faible avec le patch cible que $NNF[p]$. Si c'est le cas, $NNF[p]$ devient $NNF[p]$ plus ce décalage aléatoire, sinon on ne fait rien. Dans notre implémentation, on effectue successivement un nombre arbitraire de décalages dont les amplitudes sont à chaque fois divisées par 2.

La figure ci-dessous est un exemple-jouet de la compléion par PatchMatch d'une zone cible à partir des patch de la zone source. L'avantage de cette approche est que l'execuition prend moins de 1 seconde alors qu'il faudrait plusieurs minutes pour compléter via des approches gloutonnes comme celle vue plus tôt. La zone cible apparaît floue à cause du système de reconstruction par vote des patch adopté dans notre implémentation.



(a) Image de départ



(b) Image remplie avec PatchMatch

FIGURE 13 – Image d'un lac : Le carré noir est la zone cible

6.2 Inpainting par PatchMatch

Soit I une image et Ω un masque définissant une zone à compléter. L'idée principale consiste à considérer $I|_{\Omega}$ comme la région cible et $I|_{\Omega^c}$ comme la région source. À chaque pixel de Ω est attribué un pixel correspondant de Ω^c , de manière à ce que les patchs centrés sur ces deux pixels minimisent la SSD.

Cependant, $I|_{\Omega}$ ne contient que des 0 initialement. Une première étape consiste donc à initialiser la zone masquée afin de calculer une estimation initiale de la carte de correspondance NNF. Pour ce faire, nous appliquons sur Ω une méthode d'inpainting basée sur la diffusion, ce qui permet d'obtenir des données globalement valides à basse résolution, facilitant la comparaison avec les zones situées à l'extérieur de Ω . Dans notre implémentation, nous avons utilisé une diffusion qui successivement, pour chaque pixel de la region à remplir fait la moyenne des pixels remplis autour, s'il y en a, jusqu'à remplir toute la zone cible. Nous construisons ensuite une pyramide multi-résolution de N niveaux à partir de I . L'image I_k représente l'image à la k -ième résolution de la pyramide, où $k = 0$ correspond à la résolution d'origine, et chaque niveau supérieur réduit la taille de l'image d'un facteur de 2. Ainsi, l'image I_k aura des dimensions réduites d'un facteur 2^k par rapport à I_0 . De même, la taille des patchs T est divisée par 2^k à chaque étage. Nous construisons les pyramides à l'aides des fonctions `pyrDown` et `pyrUp` de OpenCv qui sont adaptées à cette utilisation.

À l'étage N , qui est la résolution la plus basse, la carte de correspondance NNF est initialisée aléatoirement, puis mise à jour à l'aide de l'algorithme PatchMatch. Cette étape est rendue possible grâce à l'estimation valide des données dans Ω obtenue lors de l'étape de diffusion. Ensuite, pour chaque étage k de la pyramide, la carte de correspondance et la zone cible sont propagées depuis l'étage supérieur $k + 1$ avec un zoom. Puis, on applique à nouveau PatchMatch pour affiner le NNF et on met à jour l'image de l'étage k .

Au moment de l'écriture du rapport, notre implémentation de PatchMatch pour l'inpainting ne produit pas encore des résultats satisfaisants car la diffusion initiale de l'image que nous utilisions et sur laquelle repose l'efficacité de l'algorithme n'est pas la bonne et l'article original n'explique pas laquelle il faut utiliser.



(a) Image de départ dont la région cible a subi une diffusion



(b) Image remplie avec PatchMatch

FIGURE 14 – Image d'un lac



FIGURE 15 – Image de départ à laquelle on enlève le buisson



(a) image de niveau 1 de la pyramide



(b) image de niveau 2 de la pyramide



(c) image de niveau 3 de la pyramide

FIGURE 16 – Pyramide multi-résolution PatchMatch