Inpaiting de microtextures

4IMA01 2024 - Arthur Leclaire

Joel Cabrera - angel.cabreradechia@telecom-paris.fr
Luciana Munhos - luciana.munhos@telecom-paris.fr

Table des matières

1 Introduction du sujet

Ce projet traite de l'inpainting des microtextures, qui sont des images dont la perception visuelle n'est pas affectée par la randomisation de la phase de Fourier. Elles peuvent être synthétisées à l'aide de modèles stochastiques simples basés sur des statistiques d'ordre deux, comme le modèle ADSN.

En ce qui concerne l'inpainting, il consiste à calculer une complétion plausible des parties manquantes d'une image, à partir du contenu disponible. Dans le contexte des microtextures, nous considérons l'image comme le résultat d'une modélisation aléatoire de champ. Cette approche stochastique pour l'inpainting consiste donc à estimer, à partir de l'exemplaire masqué, un modèle de texture aléatoire qui peut ensuite être échantillonné de manière conditionnelle afin de remplir les zones manquantes.

Une procédure simple est utilisée pour estimer un modèle de texture à partir d'un exemplaire masqué, puis un échantillonnage conditionnel est effectué à l'aide d'un algorithme traditionnel. Des expériences démontrent l'efficacité de cette méthode pour combler de grands trous dans une texture et surpasse les méthodes d'inpainting existantes [1].

2 Objectifs

Ce projet a pour objectif d'apprendre à simuler un modèle de texture gaussien, puis d'adapter cette simulation au problème de l'inpainting, afin de remplir les zones manquantes d'une image de microtexture. Autrement dit, le défi consiste à compléter de manière réaliste une image dans laquelle certaines valeurs de pixels sont absentes dans une région masquée, en s'appuyant sur les parties visibles de l'image.

Nous mettons en œuvre l'algorithme d'inpainting de microtextures proposé dans l'article [1], qui repose sur un échantillonnage conditionnel précis. Nous pouvons générer ces microtextures à partir d'un échantillon en utilisant un modèle gaussien stationnaire. Au fur et à mesure des tests, nous voulons tester notre algorithme avec différentes images de textures et varier les paramètres ainsi que certaines conditions du programme afin d'observer leur impact sur l'image finale.

3 Implémentation

Avant d'aborder l'algorithme principal, il est important de mentionner que ce dernier requiert une image. Nous lui demandons ensuite de dessiner la zone à éliminer pour y appliquer l'outil d'élimination des données. Cette approche nous permet de tester notre programme avec différents masques. La fonction responsable d'obtenir l'image s'appelle ***obtain_image***.

Dans cette implémentation, nous n'avons considéré que les images carrées de taille $N \times N$ pour des images en échelle de gris, et de taille $N \times N \times 3$ pour les images en couleur.

3.1 Suppression d'une partie de l'image

Dans un premier temps, nous avons commencé par supprimer une partie de l'image afin d'obtenir une image sur laquelle nous pourrions appliquer cette méthode de suppression des éléments indésirables. Pour cela, nous avons créé une matrice indicatrice dont les valeurs sont 1 partout, sauf dans la zone indiquée pour l'utilisateur. La fonction qui prend en entrée une image et une matrice indicatrice et efface la partie indiquée est ***delete_from_img*** Cette matrice sera cruciale et facilitera les calculs dans les parties suivantes. Nous allons dire que l'image est connue partout sauf dans les zones où la matrice a la valeur 0.

3.2 Obtention du modèle ADSN

3.2.1 Génération du bruit blanc

La fonction ***get_gaussian_model*** commence par la création d'un bruit blanc, généré sous la forme d'une matrice aléatoire. L'objectif est de calculer la convolution entre le spot (t_v) et ce bruit blanc (W) afin d'obtenir le modèle ADSN, qui est donc $F = t_v * W$.

$$t_v(x) = \begin{cases} \sqrt{\frac{1}{|\omega|}}(v(x) - \bar{v}), & \text{si } x \in \omega, \\ 0, & \text{sinon.} \end{cases}$$

3.2.2 Optimisation par le théorème de la convolution

Étant donné que les matrices impliquées sont de grande taille, le calcul direct d'une convolution dans le domaine spatial nécessiterait beaucoup de temps. Pour optimiser ce calcul, nous appliquons le **théorème de la convolution**, qui stipule que la convolution dans le domaine spatial est équivalente à une multiplication dans le domaine fréquentiel. Si l'on note T_v comme la TFD de t_v, on obtient que :

$$F = \Re\{\mathcal{F}^{-1}(T_v \cdot \mathcal{F}(W))\}\ ^{1} \tag{1}$$

3.2.3 Gestion de la périodicité et ajout de padding

Lors de l'application de la transformée de Fourier discrète (DFT), la périodicité implicite des données peut introduire des artefacts indésirables au niveau des bords. Pour atténuer ces effets, il est crucial de bien gérer cette périodicité correctement.

Tout d'abord, nous utilisons un *padding* pour agrandir l'image. Cette opération permet de minimiser les artefacts liés aux bords en augmentant la taille de l'image initiale à une dimension supérieure, ici le double de sa taille initiale [2, pg. 256]. Cette opération est réalisée par la fonction *add_padding*.

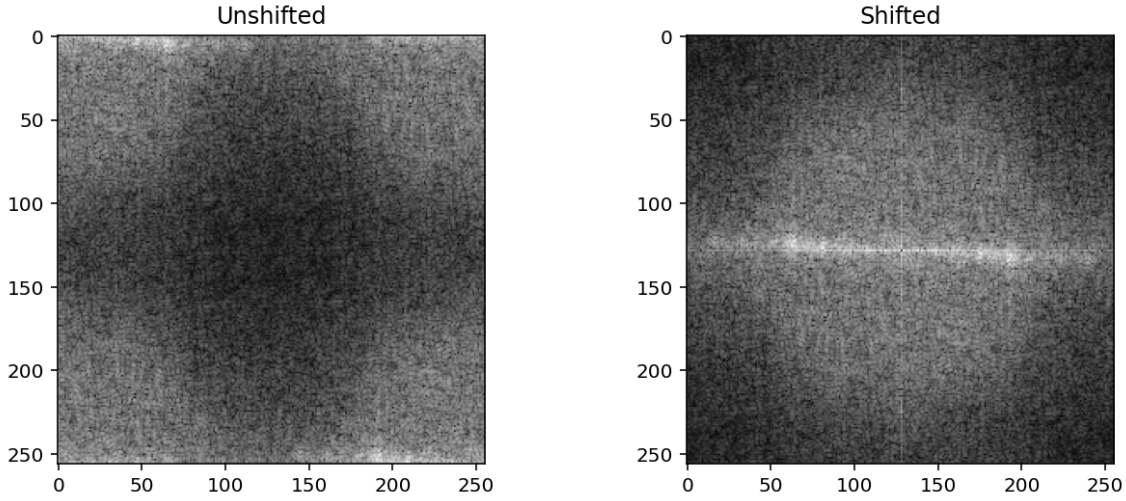
Ensuite, pour recentrer les données dans le domaine fréquentiel, chaque pixel de l'image est multiplié par $(-1)^{(i+j)}$, où i et j sont les coordonnées des pixels [2, pg. 243]. Cette étape assure

1. $\mathcal{F}$ est l'application de la TFD, et $\mathcal{F}^{-1}$ son inverse.

l'alignement de l'origine de la transformée au centre, ce qui améliore la qualité des résultats dans le domaine spectral.

3.2.4 Recentrage des données spectrales

?? Pour traiter correctement la périodicité des données lors de l'application de la transformée de Fourier discrète (DFT), il est également important de recentrer les données spectrales. Cette opération consiste à multiplier chaque pixel du domaine spatial par $(-1)^{(i+j)}$, où i et j sont les coordonnées des pixels [2, pg. 243]. Cette opération aligne l'origine de la transformée au centre dans le domaine fréquentiel (Figure 1). Le même procédé est appliqué après l'inverse de Fourier pour recentrer le résultat dans le domaine spatial. Dans la Section 4.4, on peut comparer l'ajout ou non du padding et du recentrage des données spectrales.



(a) Domaine spectral de l'image sans centrage.

(b) Domaine spectral de l'image centré en multipliant par $(-1)^{(x+y)}$.

FIGURE 1 – Effet du centrage sur le domaine spectral de l'image.

3.2.5 Traitement des images en couleur

Dans le cas des images en couleur, il faut appliquer ce filtre de manière identique à chaque canal (effectuer une convolution avec le même W pour chaque canal).

3.3 Covariance de la texture

La covariance mesure la dépendance linéaire entre deux variables aléatoires. Pour un **champ stationnaire** Z de moyenne nulle, elle se définit comme :

$$C(z) = E(Z(0)Z(z)),$$

exprimant la corrélation moyenne entre deux points séparés par z.

Dans cette implémentation, la microtexture est modélisée comme un champ stationnaire de moyenne nulle avec une covariance qui dépend uniquement de la différence entre les coordonnées x et y, ce qui simplifie son estimation. Dans ce contexte, nous estimons la covariance du modèle ADSN par :

$$c_v(x,y) = E(t_v * W(x) t_v * W(y)) = t_v * \tilde{t_v^T}(x-y) \tag{2}$$

Nous allons maintenant démontrer ce résultat :

$$c_v(x,y) = E(t_v * W(x) t_v * W(y)) = E\left(\sum_z t_v(z) \cdot W(x-z) \sum_w t_v(w) \cdot W(y-w)\right) \tag{3}$$

$$c_v(x,y) = E\left(\sum_{z,w} t_v(z) t_v(w) \cdot W(x-z) W(y-w)\right) \tag{4}$$

Étant donné que W est un bruit gaussien, nous avons que les valeurs mesurées en des points différents sont indépendantes :

$$\forall x \neq y, E(W(x) \cdot W(y)) = 0$$

et donc :

$$E(W(x-z) \cdot W(y-w)) = \begin{cases} 1 & \text{si } x - z = y - w \\ 0 & \text{sinon} \end{cases} \tag{5}$$

Ainsi, nous trouvons :

$$c_v(x,y) = E(t_v(z) \cdot t_v((y-x)+z)) = E(t_v(z) \cdot \tilde{t_v}((x-y)-z)) \tag{6}$$

En conséquence, nous obtenons la relation suivante :

$$c_v(x,y) = t_v * \tilde{t_v^T}(x-y)$$

En appliquant la transformée de Fourier discrète, nous pouvons obtenir c_v efficacement. Si $\tilde{t_v}(t) = t_v(-t)$, grâce aux propriétés de symétrie de la DFT [2, pg. 246] et en notant $T^*(w)$ la conjuguée de $T_v(w)$, il s'ensuit que :

$$t_v(t) \iff T_v^*(-w)\,^2$$
$$\tilde{t_v}(t) \iff T_v^*(w)$$

et donc :

2. Nous notons $t_v \iff T_v$ pour indiquer que T_v est la DFT de t_v.

$$c_v = \Re\{\mathcal{F}^{-1}\{T_v \cdot \tilde{T}_v\}\} = \Re\{\mathcal{F}^{-1}\{T_v \cdot T_v^*\}\} \tag{7}$$

qui peut être calculée rapidement.

Dans le cas des images en couleur, t_v est généralement de dimension $N \times N \times 3$. Pour simplifier, nous aplatissons cette matrice en une matrice de dimension $N \times (3 \cdot N)$. Le calcul de c_v suit ensuite exactement le même procédé.

3.4 Obtention des points conditionnels

Pour l'obtention des points conditionnels, la fonction ***get_conditional_points*** a besoin de la matrice indicatrice et de la largeur w du bord. Cette dernière peut être calculée très facilement en dilatant la masque M par un carré de 3×3, nous allons dire que le résultat de cette dilatation est M_D. Finalement, on obtient la masque des points conditionnels à partir de $M_D - M$. Nous pouvons voir cette approche dans la Figure 2.

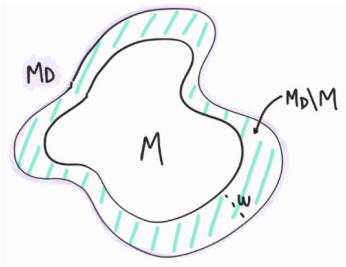


FIGURE 2 – Obtention de la masque des points conditionnels avec un largeur w

3.5 Calcul de la matrice de covariance sur les points conditionnels

Ce qui nous intéresse ici est la covariance au niveau des points conditionnels. Comme nous l'avons mentionné, nous travaillons sur un champ stationnaire, donc la matrice se calcule de la manière suivante :

$$\Gamma_{|C\times C}(c,d) = c_v(c-d), \forall c,d \in C$$

dans le cas d'une image en niveaux de gris.

Dans le cas d'une image en couleur, le calcul est un peu plus complexe, car il faut également prendre en compte les interactions entre les trois canaux de couleur pour chaque pixel. Ainsi, la matrice de covariance devient :

$$\Gamma_{|C\times C}(c,d) = \begin{bmatrix} c_v^{(R,R)}(c-d) & c_v^{(R,G)}(c-d) & c_v^{(R,B)}(c-d) \\ c_v^{(G,R)}(c-d) & c_v^{(G,G)}(c-d) & c_v^{(G,B)}(c-d) \\ c_v^{(B,R)}(c-d) & c_v^{(B,G)}(c-d) & c_v^{(B,B)}(c-d) \end{bmatrix}, \quad \forall c,d \in C,$$

où $c_v^{(k,l)}(c-d)$ représente la covariance entre le canal k de c et le canal l de d, avec $k, l \in \{R, G, B\}$.

Il est important de noter que les indices sont traités différemment dans le cas de l'indexation pour le modèle de couleur, par rapport à la version précédente. Chaque couleur est associée à un triplet d'indices, ce qui implique une gestion particulière des indices pour chaque composant (R, G, B) de l'image. Cette approche permet de prendre en compte la corrélation entre les différentes composantes de couleur tout en maintenant la structure de la matrice de covariance.

3.6 Estimation du Kriging avec CGD

Pour cette partie, nous avons appliqué l'algorithme CGD afin d'obtenir $\psi_1 = \Gamma^{-1}_{|C\times C}(u_{|C} - \hat{v})$ et $\psi_2 = \Gamma^{-1}_{|C\times C}F_{|C}$. Avec $F = t_v \cdot W$ correspondant au modèle gaussien, $u_{|C}$ les valeurs de l'image aux points conditionnels et $\hat{v}$ la moyenne de l'image.

Il n'y a pas eu de différences significatives par rapport au document de référence. Cependant, dans le cas des images en couleur, il faut prêter attention au fait que F et $(u_{|C} - \hat{v})$ appartiennent à $R^{128\times128\times3}$. Par conséquent, nous devons aplatir ces matrices de la même manière que nous avons aplati c_v.

Après avoir appliqué l'algorithme CGD, nous obtenons les vecteurs ψ_1 et ψ_2. Il ne reste alors qu'une dernière étape : calculer les composantes de krigeage $((u - \hat{v})^*)$ et d'innovation (F^*). Selon le document, ces composantes peuvent être exprimées comme suit :

$$(u - \hat{v})^* = c_v * \Psi_1$$

$$F^* = c_v * \Psi_2$$

Pour calculer Ψ_i, il est nécessaire d'étendre ψ_i en ajoutant des zéros afin d'obtenir une nouvelle image de la même dimension que l'image originale, contenant les valeurs de ψ_i aux positions des points conditionnels.

Si l'image est en couleur, il faut également aplatir la matrice Ψ_i afin qu'elle corresponde à la dimension de c_v. Enfin, pour obtenir chaque composante, nous appliquons la TFD à c_v et à chaque Ψ_i, effectuons la multiplication dans le domaine fréquentiel, puis appliquons l'inverse de la TFD. Tout ce processus est visible dans les fonctions ***apply_cgd*** et ***get_kring_innov_components***.

4 Résultats

Pour évaluer les résultats de cette mise en œuvre, nous avons défini six cas d'utilisation. Les cinq premières sont des micro-textures simples, tandis que la dernière est un peu plus complexe. Les paramètres qui ont été modifiés sont la quantité d'itérations pour l'algorithme CGD (k_max) et la largeur du bord pour l'obtention des points conditionnels ($\partial_w M$).

4.1 Les cas RGB simples

La Figure 3 montre les résultats pour les trois premiers cas d'utilisation. Dans ces trois cas, nous avons testé l'algorithme avec un masque simple et uniforme (cercle), plusieurs masques différents dans une même image, ainsi que des masques touchant le bord de l'image. Nous pouvons observer de bons résultats, où c'est difficile de identifier la zone manquante.

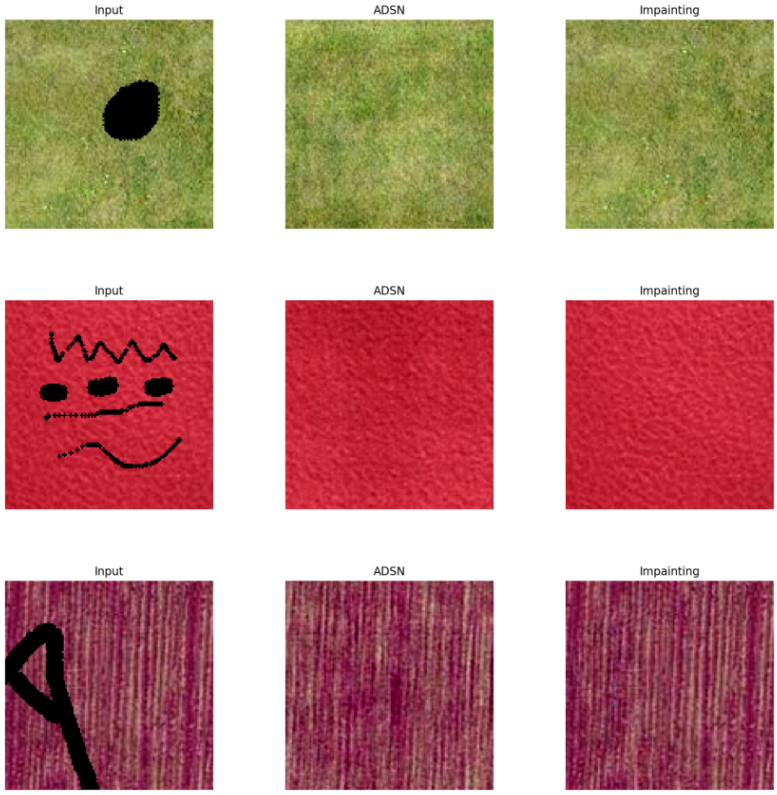


FIGURE 3 – Résultats des trois premières cas avec $C = \partial_3 M$ et $k\_max = 1000$

Côté complexité, avec $w = 3$ ils ont obtenu respectivement 486, 3.043 et 1.137 points conditionnels, avec un temps de calcul de la matrice $\Gamma_{|C \times C}$ de 1,22, 51,2 et 6,92 secondes. Nous pouvons constater que le temps de calcul de cette matrice n'est pas linéaire et dépend de chaque micro-texture. En revanche, le calcul des composantes de krigeage et d'innovation est constant, avec un temps de calcul de 0,01 secondes.

Des tests avec $w = 1$ ont été réalisés et nous pouvons constater que pour ces micro-textures simples nous obtenons également de très bons résultats pour de petites zones (Figure 4).

FIGURE 4 – Résultat pour une image en couleur avec $C = \partial_1 M$, $k\_max = 1000$

4.2 Les cas gris avec de grandes zones manquantes

Les résultats pour les cas où la zone manquante est plus grande sont illustrés dans la Figure 5. Les images sont en échelle de gris, ce qui permet de réduire les temps de calcul tout en conservant les caractéristiques essentielles des textures analysées. Alors que les temps de calcul ne diffèrent pas beaucoup d'un cas à l'autre, nous observons toutefois que les résultats sont très différents.

Dans le premier cas (rangée supérieure), nous observons une micro-texture relativement lisse et homogène. Le modèle ADSN produit une estimation qui maintient cette cohérence, et le résultat de l'inpainting respecte les structures globales de la texture.

En revanche, dans le deuxième cas (rangée inférieure), la texture présente des discontinuités plus marquées et une structure plus complexe. Ces caractéristiques posent un défi au modèle ADSN, qui introduit des artefacts dans l'estimation. Par conséquent, le résultat final de l'inpainting présente des erreurs notables, en particulier au niveau des raccords entre la zone traitée et le reste de l'image. Ce comportement met en évidence les limites du modèle pour traiter des textures plus hétérogènes et discontinuées.

4.3 Textures plus complexes

Nous allons ensuite analyser les résultats obtenus pour des micro-textures plus complexes en échelle de gris avec des variations de paramètres w et k_max. Cela nous permet d'évaluer les limites et les performances de l'algorithme dans des scénarios plus exigeants.

4.3.1 Variation de w

Dans cette partie, nous avons exploré l'impact de la largeur du bord (le paramètre w) sur les résultats de l'inpainting. Comme le montre la Figure 6, les tests ont été effectués avec $w = 3, w = 6$ et $w = 12$. Une augmentation de w entraîne une croissance du nombre de points conditionnels (c'est-à-dire C).

Dans cette expérience, les résultats montrent que ces améliorations sont minimes et ne justifient pas l'augmentation exponentielle des temps de calcul ($7,37$ secondes, $40,24$ secondes et $2,7$ minutes pour $w = 3, 6$ et 12 respectivement). Le temps de calcul devient beaucoup plus long pour une qualité d'image finale qui reste presque inchangée. Ainsi, selon les exigences de l'application, il n'est souvent

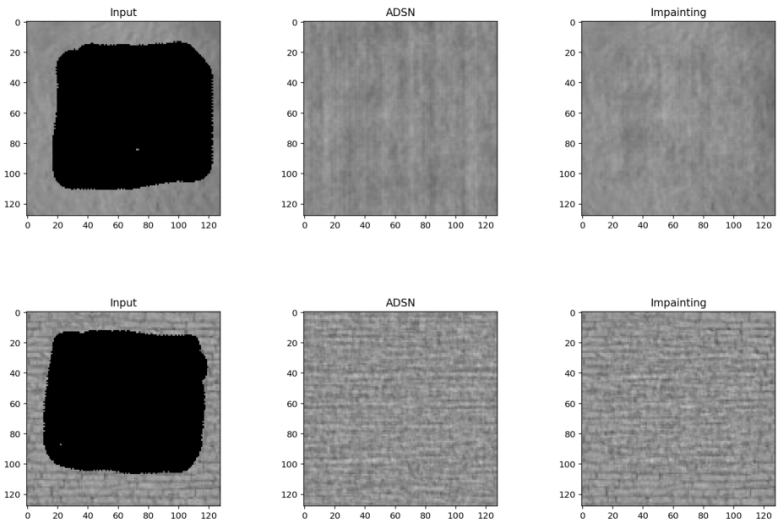


FIGURE 5 – Résultats pour des grands zones en images en échelle de gris. Dans ces cas-là $C = \partial_3 M$ et $k\_max = 1000$

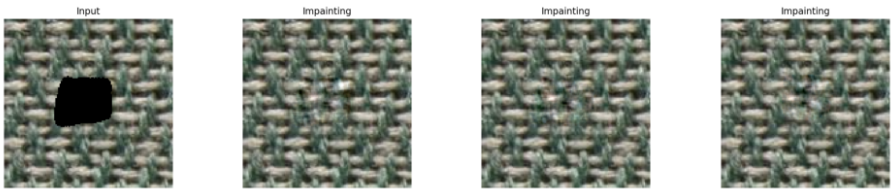


FIGURE 6 – Résultats pour $w = 3, 6, 12$

pas pertinent d'augmenter considérablement la valeur de w au-delà d'un certain seuil, car le rapport entre le coût computationnel et les bénéfices obtenus devient défavorable.

4.3.2 Variation de k_max

Nous avons également étudié l'effet du paramètre k_max, qui correspond au nombre maximal d'itérations pour l'algorithme CGD, sur les résultats et le temps de calcul. La Figure 7 présente les résultats obtenus pour $k\_max = 1000$ et $k\_max = 100000$, en fixant $w = 6$. Une valeur plus élevée de k_max permet notamment d'améliorer la précision des résultats pour des textures très complexes, mais cela nécessite beaucoup plus de temps de calcul. Ce résultat confirme qu'une valeur modérée de k_max est suffisante pour garantir des résultats visuellement convaincants pour des micro-textures moins complexes.

Malgré la complexité de la texture, l'exécution de l'algorithme avec une valeur de $k\_max = 100000$ a donné un résultat très satisfaisant, avec une zone manquante à peine perceptible. En revanche, le cas avec moins d'itérations ne semble pas constituer un bon inpainting, car les différences

y sont plus visibles.

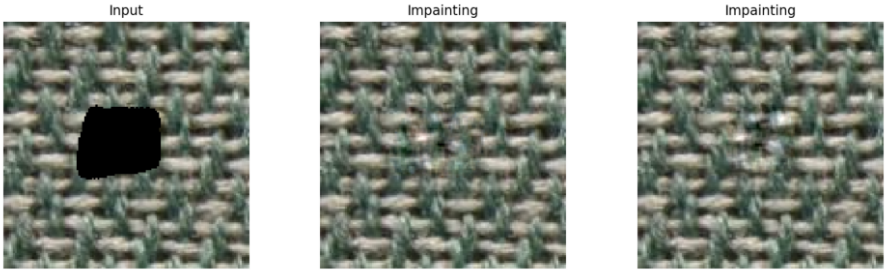


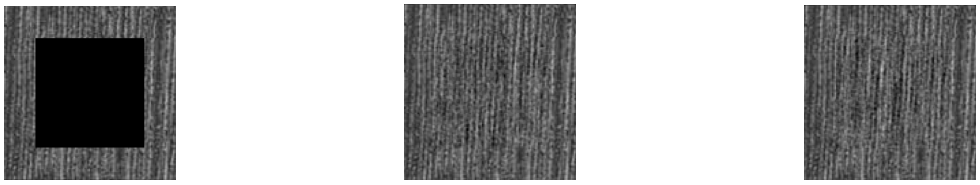
FIGURE 7 – Résultats pour $w = 6$ avec $k\_max = 1000$ (temps de calcul total de 52,4 secondes) et $k\_max = 100000$ (temps de calcul total de 5012,6 secondes)

4.4 Effet du padding et du centrage sur la convolution

Dans la Section **??**, nous avons mentionné l'importance d'ajouter un padding avant de réaliser la convolution et de centrer les données spectrales. Dans cette sous-section, nous allons comparer ces deux approches.

4.4.1 Textures simples

La Figure 8 montre les deux approches : l'une qui inclut le padding et le centrage des données, et l'autre qui ne le fait pas. À l'œil nu, il est difficile de les distinguer, mais il serait possible de définir une mesure pour les comparer. Cette mesure consisterait à prendre les pixels du bord de la zone ajoutée et à calculer la différence entre ces pixels et les pixels adjacents appartenant à l'image originale. Pour le résultat de la Figure 8b, la différence était en moyenne de 120.4, tandis que pour le résultat de la Figure 8c, elle était de 108.7. Cela semble donc indiquer que la deuxième approche est légèrement meilleure.



(a) Input

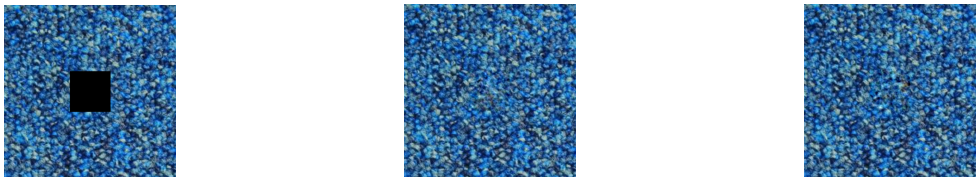
(b) $w = 1$, faire la convolution sans tenir compte du padding ni centrer les données spectrales.

(c) $w = 1$, faire la convolution en tenant compte du padding et en centrant les données spectrales.

FIGURE 8 – Effet du padding et du centrage sur la convolution dans une image simple

4.4.2 Textures complexes

En revanche, si nous essayons avec une autre image, nous observons un comportement différent. Par exemple, dans la Figure 9, visuellement, la deuxième approche semble offrir de meilleurs résultats pour la zone centrale, avec une transition plus fluide et une meilleure préservation des détails fins de la texture. Cette observation est confirmée par notre mesure : la première approche a donné un score de 136, tandis que la seconde a obtenu un score de 124, ce qui indique que la deuxième approche permet effectivement une meilleure gestion de la texture dans cette zone spécifique.



(a) Input

(b) Faire la convolution sans prendre en compte le padding et sans centrer les données spectrales.

(c) Faire la convolution en prenant en compte le padding et en centrant les données spectrales.

FIGURE 9 – Effet du padding et du centrage sur la convolution dans une image plus complexe

5 Conclusions

Nous pouvons donc conclure que les objectifs fixés au début de ce projet ont été atteints. Nous avons appris à simuler un modèle de texture gaussien stationnaire et à adapter cette simulation au problème de l'inpainting, ce qui nous a permis de remplir de manière réaliste les zones masquées d'images de microtextures. L'algorithme d'inpainting mis en œuvre, basé sur un échantillonnage conditionnel précis tel que proposé dans l'article [1], s'est montré efficace dans différents cas de test.

De plus, nous avons testé notre algorithme sur une variété d'images de microtextures, exploré l'impact de paramètres tels que w et k_max, et observé comment ces variations influencent la qualité des résultats et le temps de calcul. Ces expérimentations nous ont permis de mieux comprendre les limites et les performances de l'approche adoptée. Ainsi, les résultats obtenus confirment la capacité de cet algorithme à répondre aux défis posés par des scénarios plus complexes.

En ce qui concerne le padding et le centrage spectral, ces techniques ont permis d'améliorer les transitions pour certaines textures tout en maintenant des résultats naturels pour des textures complexes. Toutefois, l'absence de padding pourrait, dans certains cas, provoquer des effets d'aliasing, susceptibles d'altérer la qualité sur des images spécifiques.

Bien que les résultats soient prometteurs, des tests sur des bases d'images plus variées permettraient de renforcer la robustesse de l'algorithme face à des scénarios encore plus diversifiés.

Références

[1] Bruno Galerne et Arthur Leclaire. "Texture Inpainting Using Efficient Gaussian Conditional Simulation". In : *SIAM Journal on Imaging Sciences* 10.3 (sept. 2017), p. 1446-1474. doi : 10.1137/16M1109047. url : https://hal.science/hal-01428428.

[2] Rafael C. Gonzalez et Richard E. Woods. *Digital Image Processing*. 4th. Harlow, Essex, England : Pearson Education Limited, 2018. isbn : 978-1-292-22304-9.