Rapport final de projet IMA : Inpainting d'images

--«Object Removal by Exemplar-Based Inpainting» by A. Criminisi, P. Pérez, K. Toyama

Alessandro Montaldo
Chun WU
Tuteur : Yann Gousseau

I. Introduction

Dans ce projet, nous souhaitons coder un algorithme permettant de supprimer des objets de photographies numériques et de les remplacer par des arrière-plans visuellement plausibles, qui s'agit de la technique 'Inpainting d'image'. Elle vise à reconstruire des zones inconnues dans une image en utilisant l'information connue de son voisinage. Les régions à reconstruire peuvent correspondre à des zones endommagées (par exemple, dues à des rayures ou des dégradations chimiques sur des photographies) ou des éléments gênants supprimés de la scène (logos, texte, etc.).

Dans l'état de l'art, il existe principalement deux approches d'inpainting :

- L'approche basée géométrie : les méthodes basées géométrie complètent les images en utilisant des interpolations géométriques semi-locales. Ces méthodes produisent des résultats intéressants sur la géométrie mais pas sur la texture.

- L'approche basée motif (ou patch) qui consiste à la reconstruction de régions manquantes dans des images, par copier/coller de patchs d'images en calculant la similarité des patches. Elle s'est concentré sur la synthèse de textures. Ces méthodes fonctionnent très bien pour de larges zones à remplir car elles produisent des textures cohérentes avec les données connues de l'image. En revanche, ces méthodes n'assurent pas une géométrie 'globale' de la zone à remplir.

En 2004, Criminisi, etc., a combiné les deux approches d'un nouvel algorithme (Exemplar-Based Inpainting) avec les avantages des deux approches. Une priorité de remplissage plus élevée sera donnée aux endroits où les structures linéaires sont naturellement étendues dans l'espace. Avec cet algorithme, l'espace sera rempli de textures non floues, tout en préservant la structure linéaire de la région source.

L’essentiel de notre projet est d’implémenter l'algorithme d’Exemplar-Based Inpainting en Python.

Dans ce rapport, nous expliquerons en détail l’algorithme de Criminisi, présenterons les détails de notre implémentation et analyserons les résultats.

II. Algorithme de Criminisi

L'inpainting d'image et la synthèse de texture ont leurs forces et leurs faiblesses. L'inpainting d'image étend la structure linéaire dans l'espace en utilisant les informations d'isophote des pixels dans la frontière. Les structures linéaires seront naturellement étendues dans le fossé. Cependant, étant donné que l’extension utilisant réellement les techniques de diffusion, des artefacts tels que le flou pourrait être introduits.

D'autre part, la synthèse de texture copie les pixels des parties existantes de l'image, évitant le flou. L'inconvénient de la synthèse de texture réside dans le fait qu'elle se concentre sur tout l'espace de l'image, sans accorder une priorité plus élevée aux structures linéaires autour de la frontière de la zone inconnue. De ce fait, les structures linéaires ne seront pas naturellement étendues dans l’intervalle. Le résultat aurait probablement des lignes déformées. Une observation intéressante est que, même si l'inpainting d'image et la synthèse de la texture semblent différer radicalement, elles pourraient néanmoins se compléter. Criminisi, etc. a proposé un nouvel algorithme qui fait exactement cela.

i. Vue d'ensemble de l'algorithme

L'algorithme lit une image et un masque binaire. Des pixels non nuls dans le masque indiquent qu'il y a un trou à combler. On doit d’abord sélectionner la région cible Ω manuellement et définir la taille des patchs à copier. Déterminer une bonne taille de patch est un processus très expérimental. Une fois la taille définie, localisez tous les petchs qui se trouvent complètement à l'intérieur de l'image et entièrement dans la région source.

Notez que la boucle principale effectue les opérations suivantes :

- Calculez la priorité de chaque pixel p sur la frontière (fill front) de la région cible (target region). L’ordre de reconstruction est défini par la valeur de priorité estimée pour $p \in \delta\Omega$ (où $\delta\Omega$ est le bord de Ω).

- Déterminez le pixel avec la priorité la plus élevée. Nous appellerons cela le pixel cible.
- Trouver un patch source à copier avec la distance minimale du patch cible.
- Copiez la partie correspondante du patch source dans le patch cible
- Mettez à jour le masque pour refléter le patch copié.
- Déterminez les patches d'image nouvellement valides
- Répétez jusqu'à ce que la région cible soit composée de zéro pixel.

Voici, le pseudocode en anglais de l'algorithme d'Exemplar-Based Inpainting défini par A. Criminisi, P. Pérez et K. Toyama dans l'article de référence :

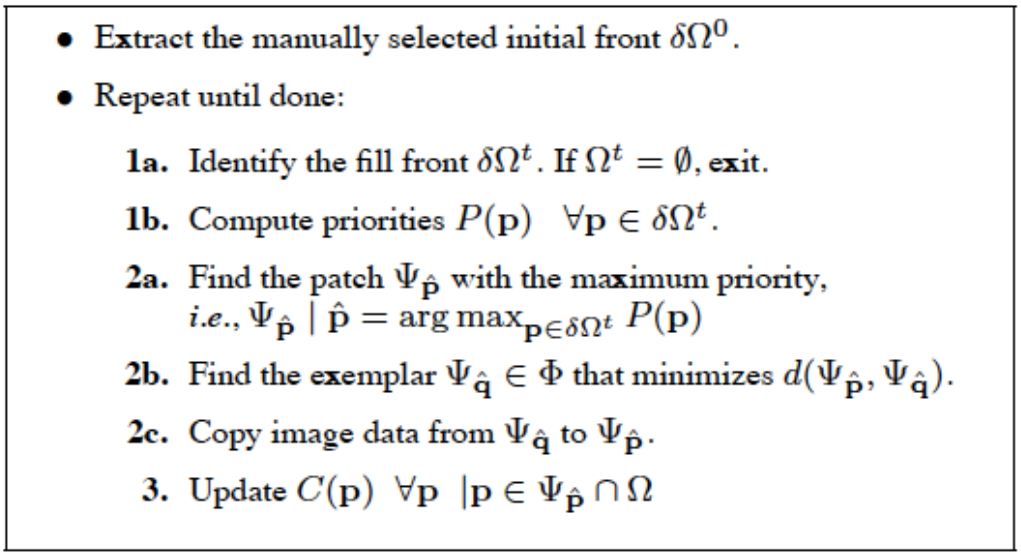


Figure 1 : Une description en pseudo-code des étapes algorithmiques – Source : "Object Removal by Exemplar-Based Inpainting" by A. Criminisi, P. Pérez, K. Toyama

On peut aussi comprendre plus facilement cet algorithme avec la figure d'illustration ci-dessous :

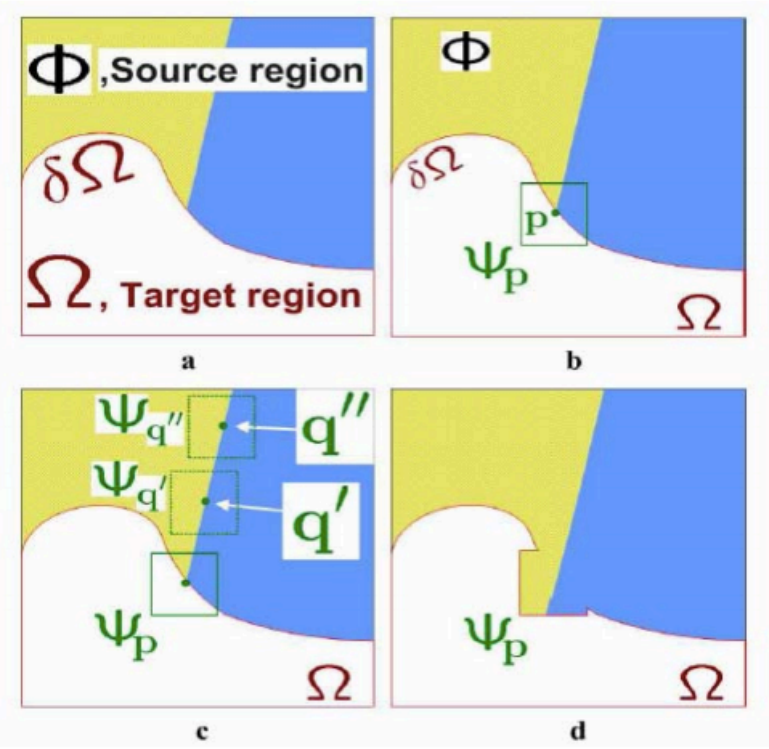


Figure 2 : Propagation de la structure par synthèse de texture basée sur des exemples(patches). – Source : "Object Removal by Exemplar-Based Inpainting"by A. Criminisi, P. Pérez, K. Toyama. L'utilisateur sera invité à sélectionner une région cible, Ω, manuellement. (a) La région source est notée Φ ; la région cible est notée Ω ; le contour de la région cible est noté δΩ . (b) Pour chaque pixel p du contour δΩ, un patch Ψp est construit, avec p au centre du patch. Une priorité est calculée en fonction de la quantité d'informations fiables autour du pixel, ainsi que de l'isophote à ce point. (c) Le patch avec la priorité la plus haute serait la cible à remplir. Une recherche globale est effectuée sur toute l'image pour trouver un patch source, Ψq qui présente le plus de similarité avec Ψp. (d) La dernière étape serait de copier les pixels de Ψq pour remplir Ψp. Avec un nouveau contour, le prochain tour de recherche du patch source continue jusqu'à ce que tous les trous soient remplis.

ii. Détails de l'algorithme

Deux parties de l'algorithme méritent une discussion approfondie :

1. Comment choisit-on quel pixel sur la frontière (fill front) a la priorité la plus haute ?
2. Comment décidons-nous quel patch source à copier dans un patch cible spécifié ?

- Calculer la priorité de patch

Criminisi et etc. ont proposé la fonction de priorité P(p) définie comme le produit du terme de confiance C (p) et du terme de données D (p) :

$$P(\mathbf{p})=C(\mathbf{p})D(\mathbf{p})$$

Où p est le pixel central d'un patch.

- Terme de confiance C(**p**)

Remplir des morceaux près de la frontière du trou devrait être intuitivement plus facile que de les remplir à l'intérieur du trou. Cette technique tire son nom du fait qu'avec un trou de forme régulière, l'algorithme rongera tout l'extérieur du trou avant de s'aller plus à l'intérieur, un ordre qui ressemble à un pelage d'oignon. Pour appliquer ce comportement, une image de confiance est conservée. Initialement, la confiance à l'extérieur du trou est 1 et la confiance à l'intérieur du trou est 0. Vous pouvez considérer la confiance comme une mesure de la quantité d'informations fiables entourant le pixel. Dans la méthode de Criminisi, la confiance d'un pixel est définie comme suit :

$$\mathrm{C(p)} = \frac{\sum \text{Confidences des pixels du patch dans la région source}}{nombre\ des\ pixels\ dans\ un\ patch}$$

$$\mathrm{C(p)} = \frac{\sum_{\mathbf{q}\in\psi_p \cap \Phi} C(q)}{|\psi_p|}$$

Où les valeurs initiales de C (p) sont définies comme

$$\mathrm{C(p)} = \begin{cases} 0, \forall \mathbf{p} \in \Omega \\ 1, \forall \mathbf{p} \in \Phi \end{cases}$$

La confiance mesure la quantité d'informations de texture d'un patch cible. Après plusieurs itérations du processus de remplissage, le terme de confiance se réduit car les valeurs de confiance deviennent proches de zéro.

- Terme de données D (p)

Le terme de données D (p) est utilisé pour encourager la propagation de la structure linéaire dans la région cible. Il est défini comme :

$$\mathrm{D}(\mathbf{p}) = \frac{|\ \nabla I_{\mathrm{p}}^{\perp} \cdot \boldsymbol{n_p}\ |}{\alpha}$$

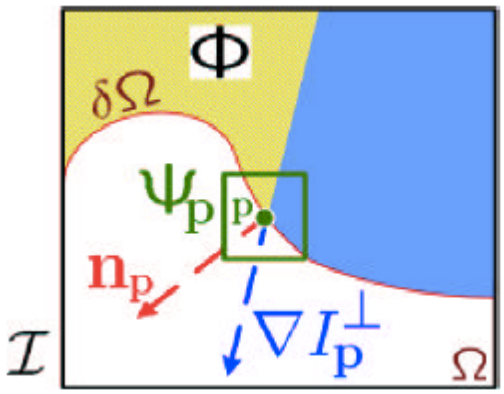


Figure 3 : Diagramme de notation. – Source : “Object Removal by Exemplar-Based Inpainting”by A. Criminisi, P. Pérez, K. Toyama. L'image entière est notée avec I.

Où $\boldsymbol{n_p}$ est la normale au contour δΩ de la région cible Ω et $\nabla I_{\mathrm{p}}^{\perp}$ est l’isophote au point p. Un isophote est simplement un vecteur de gradient pivoté de 90 degrés. Il indique la direction de la continuité linéaire plutôt que la direction de la différence maximale, comme l'indique le gradient. Et un facteur de normalisation α = 255 est choisie pour les images 8 bits. Le terme D (p) favorise la reconstruction des structures linéaires locales orthogonales à Ω en p.

Quand les priorités P(**p**) sont calculées, on peut facilement choisir le pixel cible p de plus haute priorité.

- Choisir le meilleur patch source

Une fois qu'un pixel cible est sélectionné, nous devons trouver le meilleur patch source à copier. Criminisi a proposé la comparaison d'un patch source et d'un patch cible en calculant la somme normalisée des différences au carré entre chaque pixel se trouvant dans la région source du patch cible et les pixels correspondants dans chaque patch source.

Nous trouvons le pixel p avec la priorité maximale et donc le patch source le plus similaire $\psi_{\hat{q}}$, où $\hat{q}$ est le pixel central du patch. Nous cherchons dans la région source candidate pour trouver un patch avec la distance minimale par rapport au patch $\psi_{\hat{p}}$, c'est-à-dire,

$$\psi_{\hat{q}} = \arg \min_{\psi_q \in \Phi} d\left(\psi_{\hat{p}}, \psi_q\right)$$

La distance $d\left(\psi_{\hat{p}}, \psi_q\right)$ est définie comme la somme des différences au carré (SSD) des pixels déjà remplis(connues) dans les deux patchs, exprimée par

$$d(\psi_{\hat{p}}, \psi_q) = \sqrt{\frac{1}{\boldsymbol{N_p}}} \sqrt{\sum_{\hat{p}_i \in \psi_{\hat{p}} \cap \Phi} (\|G_{\hat{p}_i} - G_{q_i}\|^2)}$$

Où $\boldsymbol{N_p}$ est le nombre de pixels qui sont dans le patch cible et aussi dans la région source, **G** est le vecteur de couleur de pixel, q_i est les pixels correspondants des $\hat{p}_i$ dans chaque patch source.

- Mettre à jour la valeur de confiance

Une fois que le patch $\psi_{\hat{p}}$ a été rempli avec de nouvelles valeurs des pixels, la confiance C (p) est mise à jour dans la zone délimitée par $\psi_{\hat{p}}$ comme suit :

$$C(q) = \mathrm{C}\,(\hat{p}) \;\; \forall q \in \psi_{\hat{p} \cup \Omega}$$

Le processus se répète jusqu'à $\delta\Omega = \varnothing$.

III. Détails d'implémentation

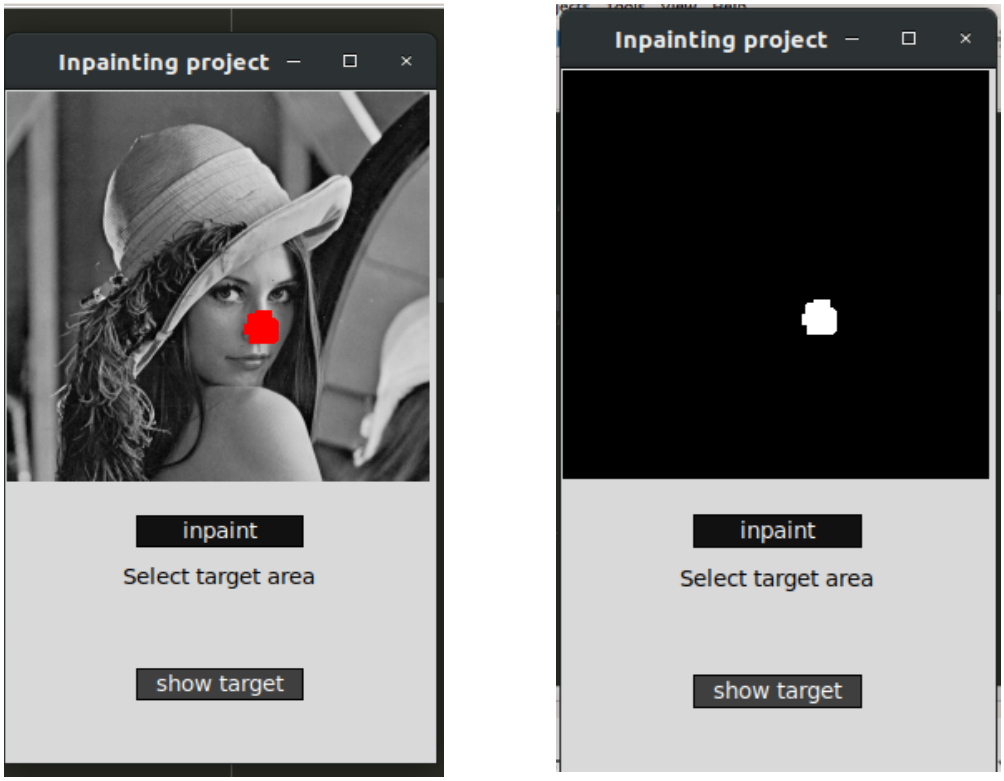
i. Interface GUI

Déboguer un algorithme comme celui-ci peut être très difficile. Même lorsque tout fonctionne correctement, l'inspection de chaque étape peut donner lieu à des informations précieuses. Nous avons créé une interface graphique GUI pour activer ce type d'inspection. Ses fonctionnalités comprennent :

- On peut importer une image originale à inpainter ;
- On peut dessiner la région cible en glissant le curseur ;
- On peut cliquer sur le bouton 'inpaint' pour lancer le programme ;
- On peut cliquer sur le bouton 'show target' pour visualiser le masque en noir/blanc.

Afin de créer cette interface, on importe la bibliothèque (library) 'Tkinter' de Python.

Voici des captures d'écran de l'interface GUI :



a) b)

Figure 4 : Captures d'écran de l'interface GUI ; a) Le région cible est dessiné en rouge à l'aide d'un curseur ; b) Le masque qu'on a dessiné est présenté en blanc.

ii. Implémentation de la fonction 'findFrontierPoints'

Idées principales : On exécute deux fois la recherche des points sur la frontière horizontalement et après verticalement. On met tous les pixels dans le masque égaux à 1 et les autres égaux à 0. Dans chaque itération, on compare la valeur des deux pixels voisins. Une fois on entre dans la région cible et détecte la différence de valeur de pixel de 0 vers 1, on ajoute le dernier pixel dans la liste 'frontierPoints' ; une fois on sort de la région cible et détecte la différence de valeur de pixel de 1 vers 0, on ajoute le premier pixel dans la liste. Et finalement, la fonction donne la liste des points comme le résultat.

iii. Implémentation de la fonction pour calculer la priorité de pixel

Idées principales : On code la fonction du terme de confiance C(p) juste comme l'indique théoriquement la partie II. Mais pour le terme de données D(p), on ne sait pas trop comment coder pour calculer $\boldsymbol{n_p}$, la normale au contour δΩ de la région cible Ω. Donc dans un premier temps, on n'a pas codé le terme de donnée de sorte que la priorité de pixel était définie comme P(p)=C(p). Dans la seconde version de notre programme, on rajoute le terme de données D(p) dans l'expression de P(p).

iv. Implémentation de la fonction 'minimiser la distance'

Pour implémenter la fonction 'minimiser la distance', on la code en suivant exactement l'équation suivante :

$$d(\psi_{\hat{p}}, \psi_q) = \frac{1}{\boldsymbol{N_p}} \sum_{\hat{p}_i \in \psi_{\hat{p}} \cap \Phi} |G_{\hat{p}_i} - G_{q_i}|$$

Où $\boldsymbol{N_p}$ est le nombre de pixels qui sont dans le patch cible et aussi dans la région source, **G** est les niveaux de gris d'un pixel, q_i est les pixels correspondants des $\hat{p}_i$ dans chaque patch source. Pour une image en couleur, on code la distance comme :

$$d(\psi_{\hat{p}}, \psi_q) = \frac{1}{\boldsymbol{N_p}} \sum_{\hat{p}_i \in \psi_{\hat{p}} \cap \Phi} \sum_{b=0}^{B} |G_{\hat{p}_{b_i}} - G_{q_{b_i}}|$$

Où G_{b_i} est la valeur de couleur dans chaque bande de fréquence et B est le nombre des bandes de fréquence, par exemple pour le système RGB, B=3.

IV. Results and Discussions

i. Effet de taille de patch

Dans nos tests, nous avons joué sur quelques paramètres de l'algorithme Criminisi. Un paramètre important de l'algorithme est la taille de patch. La taille de patch va influencer le temps d'exécution et la qualité d'inpainting. Criminisi a déclaré dans son article que le patch devrait être légèrement plus grand que le plus grand élément de texture pouvant être distingué. Il a donné 9x9 comme défaut dans son papier.

- Image « Église »

a)

b)

c) d) e)

Figure 6 : Résultats avec tailles de patch différentes ; a) Image originale ; b) Image avec la région cible sélectionnée en rouge ; c) Résultats avec taille de patch de 3x3 ; d) Résultat avec taille de patch de 5x5 ; e) Résultat avec taille de patch 7x7

Taille de patch	Nombre d'itérations	Temps d'exécution
3x3	364	574s
5x5	166	614s
7x7	92	620s

- Image « Chien »

a) b)

c) d) e)

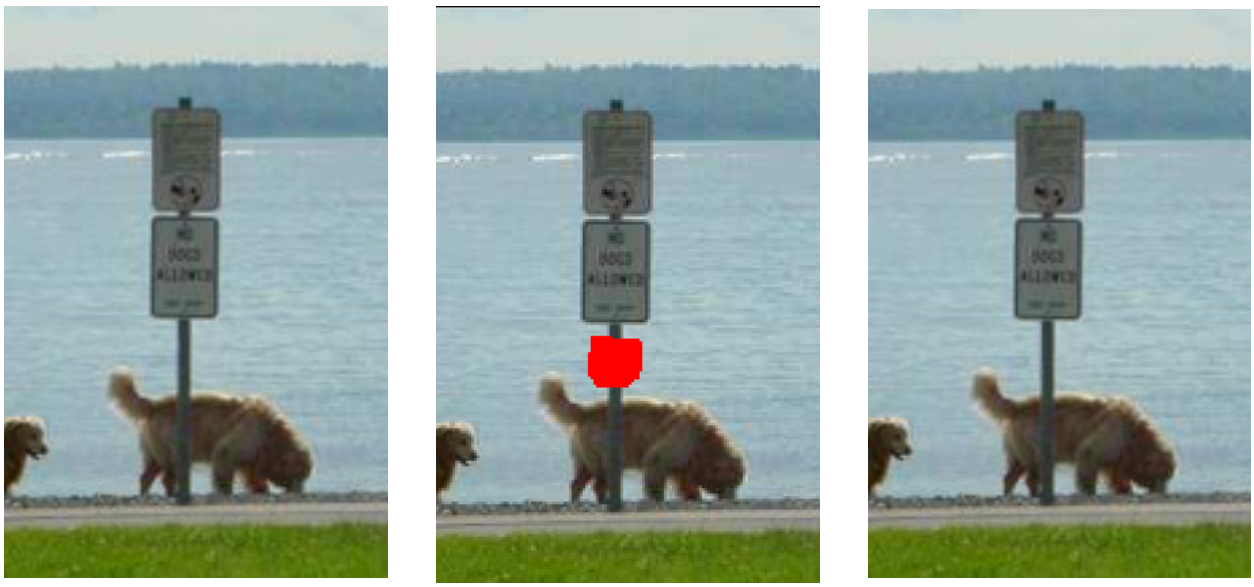
Figure 5 : Résultats avec tailles de patch différentes ; a) Image originale ; b) Image avec la région cible sélectionnée en rouge ; c) Résultats avec taille de patch de 3x3 ; d) Résultat avec taille de patch de 5x5 ; e) Résultat avec taille de patch 7x7

Taille de patch	Nombre d'itérations	Temps d'exécution
3x3	480	616s
5x5	208	639s
7x7	122	698s

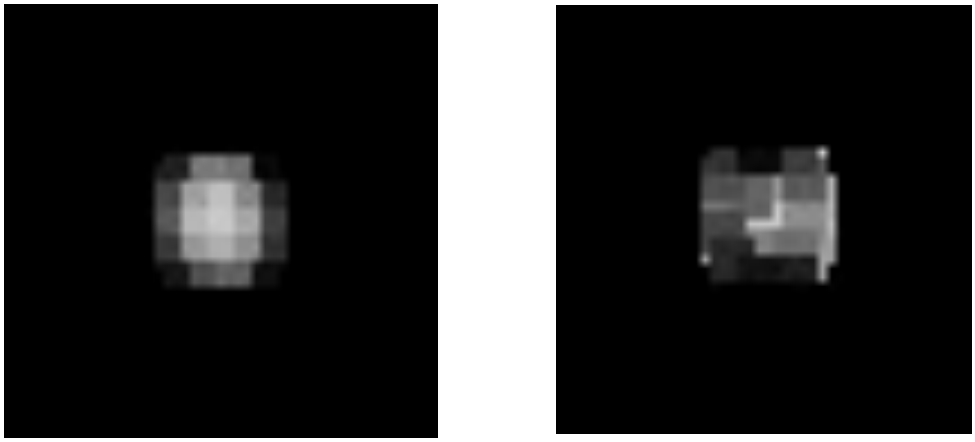
Nous trouvons que le patch de taille 3x3 donne le meilleur résultat.

ii. Effet d'isophoto

La prochaine expérience que nous avons faite est de voir comment l'isophote affecterait les résultats. On utilise par défaut 5 x 5 comme la taille de patch dans cette expérience.



a) b) c)



e) f)

Figure 6 : Effet d'isophote ; a) Image originale ; b) Image avec la région cible sélectionnée en rouge ; c) Résultat obtenu sans/avec isophote (parail pour les deux cas) ; e) Ordre de remplissage (de pixels foncés aux pixels clairs) sans isophote ; f) Ordre de remplissage (de pixels foncés aux pixels clairs) avec isophote

Dans cet exemple, il est possible de voir l'impact de la contribution D(p) dans l'ordre de sélection des pixels avec la plus haute priorité sur les frontières. Notre programme construit une image d'illustration qui montre l'ordre de remplissage allant du pixel plus sombre vers celui plus clair. Sans la contribution de D(p), l'ordre de remplissage ne dépend que du terme de confiance sur la figure 6 e), cependant en introduisant la contribution des isophotes, la direction de remplissage suit la structure géométrique (la barre) de la région source, comme l'indique la figure 6 f).

V. Conclusion

L'inpainting d'image remplit la région manquante ou endommagée dans une image en utilisant les informations spatiales de la région voisine. L'inpainting basée sur des exemples synthétise de manière itérative la région cible, par le patch le plus similaire dans la région source et selon l'ordre de remplissage. Cette méthode est une approche efficace pour reconstruire de grandes régions cibles.

Généralement, les méthodes d'inpainting basées sur des exemples (Exemplar-Based Inpainting) utilisent une taille de patch fixe et effectuent une recherche dans toute la région de la région source. Cependant, l'utilisation d'un patch de taille fixe peut présenter certains inconvénients, comme 'Exemplar-Based Inpainting' suppose que les motifs de texture de la région source peuvent être distingués avec une taille de patch appropriée. Si la taille de patch ne convient pas pour remplir une partie de la région cible, les informations de structure et de texture ne peuvent pas être affectées correctement. Par exemple, si la taille de patch est trop grande, la structure peut être reconstruite de manière incorrecte. Au contraire, si le patch est trop petit, la synthèse d'une grande région présentant des motifs de texture similaires prend trop de temps. Au futur, on pourra essayer d'utiliser une taille de patch adaptative afin d'améliorer la qualité des résultats de remplissage.

Et bien sûr, on devra améliorer l'implémentation du terme de donnéés D(p) en considérant pas seulement $\nabla I_{\mathrm{p}}^{\perp}$ qui est l'isophote au point p, mais aussi $\boldsymbol{n_p}$, la normale au contour $\delta\Omega$ de la région cible Ω. En outre on pourra optimiser l'implémentation de l'algorithme afin de réduire le temps d'exécution.

Références :

[1] A. Criminisi, P. Perez, K. Toyama. *Region filling and object removal by exemplar-based inpainting.* In 2004 IEEE Transactions on Image Processing 9 1200-1212.

[2] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. *Image inpainting.* In Proc. ACM Conf. Comp. Graphics (SIGGRAPH), pp. 417–424, New Orleans, LU, Jul 2000.

[3] Jino Lee, Dong-Kyu Lee, and Rae-Hong Park. *Robust exemplar-based inpainting algorithm using region segmentation*. In IEEE Transactions on Cosumer Electronics (Volume:58, Issue:2, May 2012), pp. 553-561, 05 July 2012.