TP8 – Réalité diminuée

On désigne par *réalité diminuée* le processus inverse de la *réalité augmentée*. Cette tâche, qui consiste à supprimer un ou plusieurs objets d'une scène, est parfois utilisée pour de mauvaises raisons, par exemple pour éliminer une personne indésirable sur une photographie, mais elle sert également à retoucher une image, par exemple pour « gommer » le mobilier dans un appartement afin de faciliter sa visite virtuelle.

Limites de l'*inpainting* par diffusion

La méthode d'*inpainting* vue dans le TP7 s'appelle l'*inpainting par diffusion*. Elle permet de restaurer des zones de faible épaisseur, comme les rayures sur une photographie ancienne, mais ne permet pas de répondre aux besoins de la réalité diminuée. Faites une copie du script `exercice_2.m` du TP7, de nom `exercice_0.m`, et apportez les modifications suivantes : fixez les paramètres `sigma_bruit` à 0 et `lambda` à 1 ; remplacez les fichiers `fleur.png` et `fleur_masque.png` par `randonneur.jpg` et `randonneur_masque.png`, respectivement. Le résultat n'est pas satisfaisant. La réalité diminuée nécessite d'utiliser des méthodes d'*inpainting* qui, contrairement à l'*inpainting* par diffusion, ne découlent pas d'une approche variationnelle.

Principe de l'*inpainting* par rapiéçage

Si Ω désigne l'ensemble des pixels de l'image, et D le domaine à restaurer, l'*inpainting* « par rapiéçage » (*patch-based inpainting*) consiste à rechercher, en tout pixel $\mathbf{p}$ de la frontière ∂D de D, le pixel $\widehat{\mathbf{q}} \in \overline{D} = \Omega \backslash D$ (complémentaire de D à Ω) dont le voisinage $V(\widehat{\mathbf{q}})$ « ressemble le plus » au voisinage $V(\mathbf{p})$ de $\mathbf{p}$. En pratique, le voisinage est une fenêtre centrée de taille $(2\,t+1) \times (2\,t+1)$, $t > 0$. Si $\mathbf{p} = (i_\mathbf{p}, j_\mathbf{p})$ est un pixel de ∂D, nous notons $R(\mathbf{p})$ l'ensemble des *indices relatifs* des pixels voisins $\mathbf{p}' = (i_\mathbf{p} + i, j_\mathbf{p} + j)$ déjà remplis :

$$R(\mathbf{p}) = \{(i,j) \in [-t,t]^2 \,/\, (i_\mathbf{p} + i, j_\mathbf{p} + j) \in \overline{D}\} \tag{1}$$

Il apparaît que $\mathrm{Card}(R(\mathbf{p})) > 0$, puisqu'un voisin de $\mathbf{p}$ au moins appartient à $\overline{D}$. Pour une image en niveaux de gris I, la *dissemblance* $d(\mathbf{p},\mathbf{q})$ entre le voisinage d'un pixel $\mathbf{p} \in \partial D$ et le voisinage d'un pixel $\mathbf{q} \in \overline{D}$ est définie comme suit (cette définition est facile à étendre aux images en couleur) :

$$d(\mathbf{p},\mathbf{q}) = \frac{1}{\mathrm{Card}(R(\mathbf{p}))} \sum_{(i,j) \in R(\mathbf{p})} [I(i_\mathbf{p} + i, j_\mathbf{p} + j) - I((i_\mathbf{q} + i, j_\mathbf{q} + j)]^2 \tag{2}$$

Pour trouver $\widehat{\mathbf{q}}$, il est inutile de calculer $d(\mathbf{p},\mathbf{q})$ en chaque pixel $\mathbf{q} \in \overline{D}$, car une image est généralement constituée de régions de texture homogène. Nous nous contentons de rechercher $\widehat{\mathbf{q}}$ dans une fenêtre $F(\mathbf{p})$ centrée en $\mathbf{p}$, de taille $(2\,T+1) \times (2\,T+1)$, $T > t$. Il suffit même de mener cette recherche pour l'ensemble des pixels $\mathbf{q} \in F'(\mathbf{p})$ tels que $V(\mathbf{q}) \subset \big(F(\mathbf{p}) \cap \overline{D}\big)$, comme le montre la figure 1. Le pixel $\widehat{\mathbf{q}}$ associé à un pixel $\mathbf{p} \in \partial D$ est donc :

$$\widehat{\mathbf{q}} = \underset{\mathbf{q} \in F'(\mathbf{p})}{\mathrm{argmin}} \{d(\mathbf{p},\mathbf{q})\} \tag{3}$$

Le rapiéçage consiste à remplacer les pixels manquants de $V(\mathbf{p})$ par les pixels de mêmes positions dans $V(\widehat{\mathbf{q}})$.

La première méthode d'inpainting par rapiéçage a été décrite en 2003 par Criminisi, Pérez et Toyama dans un article intitulé *Region filling and object removal by exemplar-based image inpainting*. Bien sûr, de nombreuses améliorations ont été proposées depuis. Une version commerciale de l'inpainting par rapiéçage est proposée dans le logiciel *PaintShop Pro*. Par ailleurs, le greffon (*plugin*) de *GIMP* de nom *PatchMatch*, disponible dans la boîte à outils *G'MIC* développée à l'Université de Caen, constitue une version libre de cette méthode d'*inpainting*.

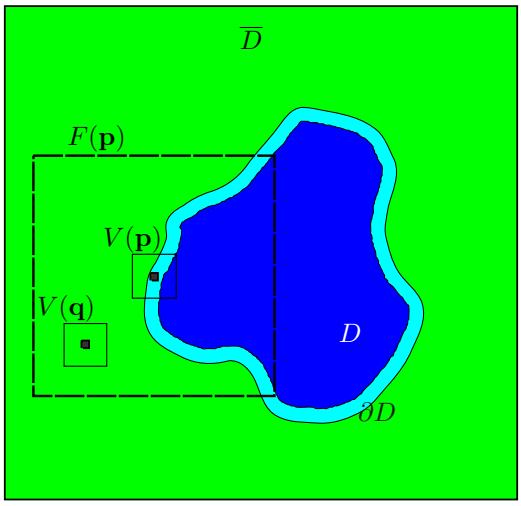


FIGURE 1 – Principe de l'*inpainting* par rapiéçage.

Exercice 1 : traitement des pixels de ∂D par tirage aléatoire

Une première façon de coder la méthode d'*inpainting* par rapiéçage consiste à répéter la boucle suivante, tant que le domaine D n'est pas vide :

1. Choisir un pixel $\mathbf{p}$ de la frontière ∂D par tirage aléatoire.
2. Rechercher le pixel $\widehat{\mathbf{q}}$ défini en (3) en testant tous les pixels $\mathbf{q} \in F'(\mathbf{p})$. Pour chaque $\mathbf{q}$:
   - Calculer la dissemblance $d(\mathbf{p}, \mathbf{q})$ définie par (2).
   - Si $d(\mathbf{p}, \mathbf{q}) < \widehat{d}$, alors $\widehat{\mathbf{q}} \leftarrow \mathbf{q}$ et $\widehat{d} \leftarrow d(\mathbf{p}, \mathbf{q})$.
3. Utiliser $V(\widehat{\mathbf{q}})$ pour compléter les pixels manquants de $V(\mathbf{p})$ par rapiéçage.
4. Mettre à jour D et ∂D.

Complétez le script `exercice_1.m` qui, en appelant les fonctions `d_min` et `rapiecage_1` déjà écrites, doit permettre de reproduire cet algorithme. En le testant sur l'image `randonneur.jpg`, vous constatez que `exercice_1.m` fournit des résultats beaucoup plus réalistes que `exercice_0.m`.

Tentez ensuite d'effacer le bateau rouge de l'image `regate.jpg`. Le résultat est décevant, car le remplissage est effectué sans tenir compte ni de la forme de D, ni de la structure de l'image au voisinage immédiat de D.

Exercice 2 : traitement des pixels de ∂D selon un ordre de priorité

Pour améliorer les résultats du script `exercice_1.m`, l'idée de Criminisi, Pérez et Toyama consiste à déterminer le prochain pixel $\mathbf{p} \in \partial D$ à traiter selon un ordre de priorité.

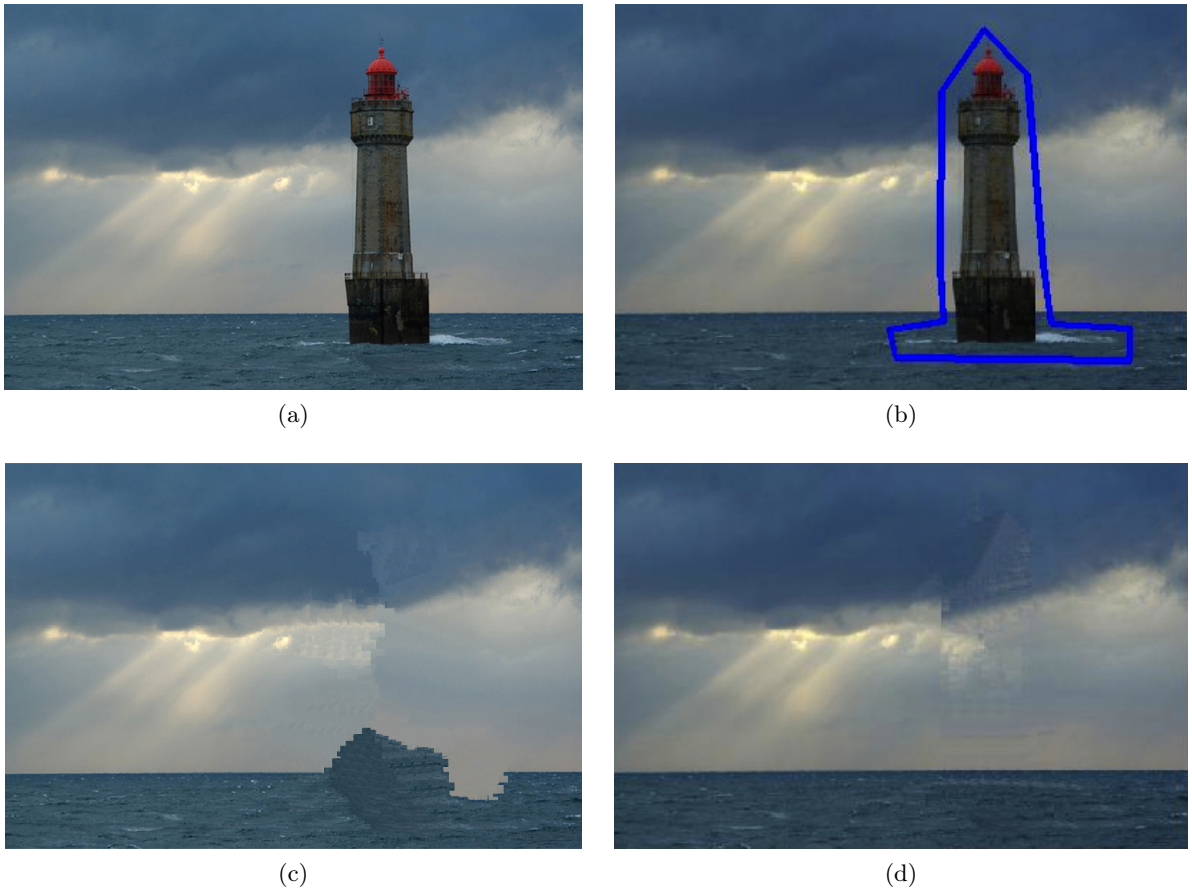
D'une part, il semble pertinent de traiter en premier les pixels situés sur une partie convexe de ∂D, car le nombre de voisins à compléter y est moindre. D'autre part, pour éviter le défaut du résultat de la figure 2-c, il convient de donner priorité aux pixels du contour où le gradient de l'image est à la fois élevé et tangent au contour. La priorité $P(\mathbf{p})$ d'un pixel $\mathbf{p} \in \partial D$ est donc égale au produit de deux coefficients $C(\mathbf{p})$ et $A(\mathbf{p})$:

- À l'initialisation, la *confiance* vaut $C(\mathbf{p}) = 1$ si $\mathbf{p} \in \overline{D}$ (pixel déjà rempli, donc considéré comme fiable), et $C(\mathbf{p}) = 0$ sinon. À chaque rapiéçage, les pixels de $V(\mathbf{p})$ ayant une confiance nulle reçoivent comme nouvelle valeur de la confiance $C(\mathbf{p})$ la confiance moyenne calculée sur $V(\mathbf{p})$.
- L'*attache aux données* $A(\mathbf{p})$ est égale à $|\mathbf{t}(\mathbf{p}) \cdot \nabla u(\mathbf{p})|$, où $\mathbf{t}(\mathbf{p})$ désigne un vecteur *de norme unitaire*, localement tangent au contour de D, et $\nabla u(\mathbf{p})$ le gradient en $\mathbf{p}$ de l'image u en cours de complétion.

Écrivez la fonction `priorites`, d'en-tête `function [P,C_nouv] = priorites(u,D,C,delta_D,t)`, qui calcule les priorités et met à jour la confiance. La fonction `gradient` de Matlab peut être utilisée pour calculer les vecteurs $\mathbf{t}$ et ∇u. Il est conseillé d'utiliser le canal L (« luminance ») de l'image convertie au format *CIE LAB* (fonction `rgb2lab`) pour calculer ∇u. Enfin, n'oubliez pas de convertir l'argument d'entrée `D` au format `double`.

Faites une copie de la fonction `rapiecage_1`, de nom `rapiecage_2`, et une copie du script `exercice_1.m`, de nom `exercice_2.m`, que vous modifierez de manière à implémenter cette nouvelle version de l'*inpainting* par rapiéçage avec un ordre de priorité. Une différence notable entre `exercice_1.m` et `exercice_2.m` est que, lorsque l'ensemble $F'(\mathbf{p})$ est vide, c'est-à-dire si aucun pixel $\mathbf{q}$ n'est tel que $V(\mathbf{q}) \subset \left(F(\mathbf{p}) \cap \overline{D}\right)$, il ne suffit pas de sauter les étapes 3 et 4, comme le fait `exercice_1.m`, car cela provoquerait une boucle infinie. Le prochain pixel à traiter dans un tel cas doit être celui de plus forte priorité pour lequel $F'(\mathbf{p})$ n'est pas vide.

Le script `exercice_2.m` peut fournir de bons résultats, comme cela est illustré sur l'exemple de la figure 2. Vous pouvez maintenant essayer de reproduire le trucage de l'image `photo_truquee.jpg` où Nikolaï Iejov, après sa disgrâce, fut effacé de la photographie `photo_originale.jpg` pour ne plus apparaître aux côtés de Staline !



(a) (b)

(c) (d)

FIGURE 2 – (a) Image originale. (b) Sélection du domaine D. (c) Résultat sans ordre de priorité (exercice 1), qui comporte un défaut flagrant. (d) Résultat avec ordre de priorité (exercice 2).

En guise de conclusion, vous noterez que cette nouvelle version de l'*inpainting* par rapiéçage ne suffit pas toujours à répondre aux besoins de la réalité diminuée. Pour vous en convaincre, tentez d'effacer le fauteuil de l'image `mur.jpg`.