Inpainting d'images

O.Bellouti, B.Mellouki, Y.Fahim, A.J. Ndoudi-Likoho, A.Kassara, R.Laarach

June 28, 2018

Abstract

Lors de la restauration et de l'édition des images, il est courant de devoir remplir ou réparer une zone dans une image. Par exemple, lors du tournage d'un film, on doit souvent enlever des objets a posteriori en post-production, car cela coûterait trop cher de retourner la séquence. Ainsi, il est intéressant d'effectuer ce remplissage (ou "inpainting" en anglais) de manière automatique avec des méthodes du traitement de l'image.

1 Introduction

Le but de ce projet est d'explorer plusieurs approches différentes à l'inpainting d'image. Une première consiste à résoudre une équation différentielle partielle, afin de remplir la zone de manière lisse. Une deuxième emploie la notion de "patchs", c'est-à-dire des petites imagettes qui permettent d'encoder la texture et la structure des images. Enfin, on pourra explorer une approche variationnelle qui minimise une fonctionnelle d'énergie fondée sur ces patchs d'image.

2 Une approche diffusive

2.1 Un modèle simple

Dans cette première approche, nous allons utiliser plusieurs hypothèses simplificatrices: la première est de considérer que la direction de propagation est toujours orthogonale aux contours de la région qu'on veut remplir, qu'on dénote par Ω . On admet aussi que cette région a des propriétés topologiques simples: elle est compacte et convexe. Enfin, pour réduire la complexité de notre algorithme, on travaille sur des images en noir et blanc. On pose N la dimension de l'image. On rappelle qu'une image est modélisée par une fonction de $\mathbf{R^2}$.

2.1.1 Mise en équation

On pose u(x,y,t) une image.Nous considérons l'équation de diffusion à deux dimensions, avec une constante de diffusion unitaire, sur la région Ω et pour t>0.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{\partial u}{\partial t} \quad (1)$$

Nous allons discrétiser cette équation grace à la méthode des éléments finis. On choisit donc un pas de discrétisation pour chaque dimension: Δx , Δy, Δt. Pour n $\in \mathbf{N}$:

$$\frac{\partial u}{\partial t}(\mathrm{i}\Delta x, \mathrm{j}\Delta y, \mathrm{n}\Delta t) = \frac{u(i\Delta x, j\Delta y, (n+1)\Delta t) - u(i\Delta x, j\Delta y, n\Delta t)}{\Delta t} + \mathrm{O}(\Delta t) \quad (2)$$

$$\frac{\partial^2 u}{\partial x^2}(\mathrm{i}\Delta x, \mathrm{j}\Delta y, \mathrm{n}\Delta t) = \frac{u((i+1)\Delta x, j\Delta y, n\Delta t) + u((i-1)\Delta x, j\Delta y, n\Delta t) - 2u(i\Delta x, j\Delta y, n\Delta t)}{\Delta^2 x} + \mathrm{O}(\Delta^2 x) \quad (3)$$

$$\frac{\partial^2 u}{\partial y^2}(\mathrm{i}\Delta x, \mathrm{j}\Delta y, \mathrm{n}\Delta t) = \frac{u(i\Delta x, (j+1)\Delta y, n\Delta t) + u(\Delta x, (j-1)\Delta y, n\Delta t) - 2u(i\Delta x, j\Delta y, n\Delta t)}{\Delta^2 y} + \mathrm{O}(\Delta^2 y) \quad (4)$$

On verra dans la suite que le choix de Δt dépend des choix de Δx et Δy pour des raisons de stabilité du système. Pour simplifier nos calculs, on pose $\Delta x = \Delta y$=1. Il est important de préciser que ce choix est arbitraire et ne modifie pas nos résultats. C'est la relation entre le pas de la dimension temporelle, et les pas de la dimension spatiale qui détermine la stabilité du système.

On obtient donc l'équation discrétisée de (1).On cherche alors les suites $(V^n_{i,j})_{n\in\mathbf{N},(i,j)\in\Omega}$ qui vérifient l'équation récurrente suivante:

$\forall$ (i,j) $\in \Omega$ $\qquad V^0_{i,j}$ = u(0,0,0) $\quad$ si n=0 (matrice de l'image initiale)

$\forall$ (i,j) $\in \Omega$ $\qquad V^{n+1}_{i,j} = \Delta t$ ($V^n_{i+1,j} + V^n_{i-1,j}$) + Δt ($V^n_{i,j+1} + V^n_{i,j-1}$)+$(1-4\Delta t)V^n_{i,j}$ $\quad$ si n>0 $\qquad$ (5)

2.1.2 Condition de stabilité du système

Dans cette partie, on cherche à montrer que, sous une certaine condition sur Δt, la solution de l'équation (1) est contrôlée à l'infini.

Il est possible d'écrire l'équation (5) sous la forme suivante, en posant V_n la matrice $(V^n_{i,j})_{(i,j)\in\Omega}$.

V_0 = u(0,0,0) $\quad$ si n=0

$V_{n+1} = \mathbf{F}(\ V^n\ )$ si n>0 $\qquad$ (3)

Avec $\mathbf{F}$ une application linéaire de $\mathbf{M}_n(\mathbf{R})$ telle que :

$\forall X \in \mathbf{M}_N(\mathbf{R})$ $\qquad \mathbf{F}(X) = AX + XB$

Avec $A = \begin{bmatrix} 1-4\Delta t & \Delta t & 0 & \dots & 0 \\ \Delta t & 1-4\Delta t & \Delta t & \ddots & \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & & & \\ & & & & \Delta t \\ 0 & \dots & 0 & \Delta t & 1-4\Delta t \end{bmatrix}$

$B = \begin{bmatrix} 0 & \Delta t & 0 & \dots & 0 \\ \Delta t & 0 & \Delta t & \ddots & \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & & & \\ & & & & \Delta t \\ 0 & \dots & 0 & \Delta t & 0 \end{bmatrix}$

F est diagonalisable sur $\mathbf{R}$ car A et B sont diagonalisables (parce que symétriques). En effet, si $(U_1, ...U_N)$ et $(V_1, ..., V_N)$ deux bases de $M_{N,1}(\mathbf{R})$ formées de vecteurs propres de A et B respectivement, alors $(U_i V_j^t)_{1\le i,j\le N}$ est une base de $M_N(\mathbf{R})$ formée de vecteurs propres de F . De plus, on a d'après [1], le spectre de F: $Sp(F) = \{\alpha + \beta, (\alpha, \beta) \in Sp(A) \times Sp(B)\}$ On a aussi d'après [2]:

$Sp(A) = \left\{2\Delta t cos(\frac{k\pi}{N+1}) + (1-4\Delta t), k \in [1, N]\right\}$ $Sp(B) = \left\{2\Delta t cos(\frac{k\pi}{N+1}), k \in [1, N]\right\}$

On en déduit que :

$Sp(F) = \left\{2\Delta t(cos(\frac{k\pi}{N+1}) + cos(\frac{k'\pi}{N+1})) + (1-4\Delta t), (k, k') \in [1, N]^2\right\}$

Ainsi on peut écrire l'équation (3) sous la forme matricielle suivante: $\forall n > 0$ $\qquad v_{n+1} = P^{-1}DP\ v_n$

Avec v_n une représentation vectorielle de la matrice V_n, et D une matrice diagonale dont les éléments diagonaux sont les éléments de l'ensemble $Sp(F)$. Après n itérations, on a :

$\forall n > 0$ $\qquad v_n = P^{-1}D^nP\ v_0$

Ainsi le système est stable si et seulement si $max_{\lambda\in Sp(F)}|\lambda| \le 1$. Or le maximum est atteint lorsque $cos(\frac{k\pi}{N+1}) = cos(\frac{k'\pi}{N+1}) = -1$, ce qui est le cas quand $k = k' = N$ et quand $N >> 1$.

Ainsi, la suite v_n converge quand:

$max_{\lambda\in Sp(F)}|\lambda| \le 1 = |1 - 8\Delta t| \le 1$ Ainsi le système est stable si et seulement si: $\qquad \boxed{\Delta t \le \frac{1}{4}}$

On prendra pour la suite : $\Delta t = \frac{1}{4}$

Remarque:
Pour $\Delta x, \Delta y$ quelconque, on obtient de la même manière l'inégalité suivante: $\Delta t \leq \frac{1}{2(\frac{1}{\Delta^2 x} + \frac{1}{\Delta^2 y})}$ (6)

2.1.3 Majoration de l'erreur

Dans cette partie, on cherche à montrer que $R_n = U_n - V_n$ converge bien vers 0 lorsque $(\Delta t, \Delta x, \Delta y)$ tend vers $(0,0,0)$, avec $U_n = (u(i\Delta x, j\Delta y, n\Delta t))_{1 \leq i,j \leq N}$ où u est la solution de l'équation (1) à conditions initiales fixées. En utilisant les relations (2), (3) et (4) et l'équation de diffusion discrétisée, on a pour $\Delta x, \Delta y, \Delta t > 0$:

$\forall (i,j) \in \Omega \ \ \forall n > 0 \qquad R_{i,j}^{n+1} = \frac{\Delta t}{\Delta^2 x}\ (R_{i+1,j}^n + R_{i-1,j}^n) + \frac{\Delta t}{\Delta^2 y}(R_{i,j+1}^n + R_{i,j-1}^n) + (1 - 2\Delta t(\frac{1}{\Delta^2 x} + \frac{1}{\Delta^2 y}))R_{i,j}^n + o(\Delta t, \Delta^2 x, \Delta^2 y)$

On considère la norme $||A||_\infty = max_{1 \leq i,j \leq N}|a_{i,j}|$ sur $M_n(\mathbf{R})$

Alors on a, par inégalité triangulaire et par l'inégalité (6) :

$\forall n > 0 \ \ ||R_n||_\infty \leq 4\Delta t(\frac{1}{\Delta^2 x} + \frac{1}{\Delta^2 y})||R_{n-1}||_\infty + ||R_{n-1}||_\infty + o(\Delta t, \Delta^2 x, \Delta^2 y) \leq 3||R_{n-1}||_\infty + o(\Delta t, \Delta^2 x, \Delta^2 y)$
En réitérant cette opération n-1 fois, on obtient:

$\forall n > 0 \ \ ||R_n||_\infty \leq 3^n||R_0||_\infty + o(\Delta t, \Delta^2 x, \Delta^2 y)$
Or $R_0 = U_0 - V_0 = 0$. On en déduit que $\forall n > 0 \ \ ||R_n||_\infty \leq o(\Delta t, \Delta^2 x, \Delta^2 y)$, d'où le résultat.

2.2 Algorithme:

Le langage utilisé est Matlab.

Algorithm 1 Résout l'équation (5)

Require: $eps > 0, \Delta t, \Delta x, \Delta y, x_0, y_0, x_1, y_1$
  $Vf \leftarrow trou(x0, y0, x1, y1, image)$ *trou est une fonction qu'on a implémenté séparemment et qui permet de remplacer un rectangle de image par des pixels noirs .*
  $N = size(Vf)$
  $N \leftarrow -n$
  $R \leftarrow 1$
  while $R > eps$ **do**
    $V \leftarrow V_f$
    for $i = x_0 : x_1$ **do**

      for $j = y_0 : y_1$ **do**
        $V_f(i,j) = \Delta t * (V_f(i+1,j) + V_f(i-1,j)) + \Delta t * (V_f(i,j+1) + V_f(i,j-1)) + (1 - 4 * \Delta t) * V_f(i,j)$
      end for
    end for
    $R \leftarrow norm(V - Vf)$ *La norme de* $(V_n + 1 - V_n)$ *est la condition d'arret de la boucle*
  end while
  imshow(V_f)

2.3 Résultats:

Figure 1: L'écriture en dessous du papillon a été enlevée

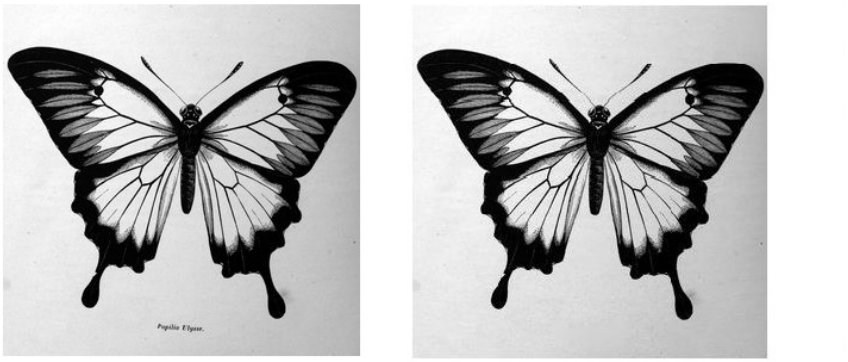


Figure 2: L'algorithme est moins efficace quand il s'agit d'une texture complexe

Figure 3: Quand l'arrière plan est constant, le résultat est très satisfaisant

3 Au-delà de l'Inpainting:

Dans cette partie, on va explorer une autre facette de la restauration d'Images, basée sur l'équation de Poisson [3] avec condition de Dirichlet aux bords : $\Delta f = div\ v$

3.1 Mise en équation:

Le problème est le suivant: on considère une fonction f^* , image destination, qui n'est pas connue sur une certaine région Ω, dont les propriétés topologiques sont similaires au cas précedent, on cherche alors à interpoler f^* sur Ω en utilisant un champ de guidage $v = \vec{grad}\ g$ avec g l'image source.

La solution la plus intuitive à ce problème est une fonction f qui minimise la différence entre le gradient de f à l'intérieur de Ω et champ de guidage afin d'obtenir le moins d'écart possible entre $f_{|\Omega}$ et g , tout en respectant la condition de Dirichlet sur la frontière $\delta\Omega$, ce qui revient à résoudre le problème de minimisation suivant:

$$min_f \int\int_\Omega |\vec{grad}\ f - v|^2 \text{ avec } f_{|\delta\Omega} = f^*_{|\delta\Omega} \qquad (7)$$

D'après le théorème d'Euler Lagrange, la solution f de (7) doit vérifier l'équation de Poisson suivante:

$$\Delta f = div\ v \text{ sur } \Omega \text{ avec } f_{|\delta\Omega} = f^*_{|\delta\Omega} \qquad (8)$$

On discrétise alors l'équation (7), et la condition imposée par le théorème d'Euler Lagrange (8). En posant f_p la valeurs de f au pixel p de Ω, N_p l'ensemble des pixels voisins de p. Alors la suite $(f_p)_{p\in\Omega}$ vérifie le système suivant(on considère notamment le cas où Ω n'a pas d'intersection avec les bords de l'image):

$$\forall p \in \Omega \quad |N_p| f_p - \sum_{q\in N_p\cap\Omega} f_q = \sum_{q\in N_p\cap\delta\Omega} f^*_q + \sum_{q\in N_p} g_p - g_q \qquad (9)$$

Ce système est linéaire, ce qui nous permet de l'écrire sous la forme matricielle suivante, avec $n = |\Omega|$ et A une matrice par bloc de $M_{n^2}(\mathbf{R})$

$$Af = b \qquad (10) \text{ où } f = \begin{bmatrix} f_{1,1} \\ f_{1,2} \\ \vdots \\ f1,n \\ f2,1 \\ \vdots \\ f2,n \\ \vdots \\ fn,n \end{bmatrix}$$

$$A = \begin{bmatrix} M & -I_n & 0 & \dots & 0 \\ -I_n & M & -I_n & \ddots & \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & & & \\ & & & & -I_n \\ 0 & \dots & 0 & -I_n & M \end{bmatrix} \quad \text{Où} \quad M = \begin{bmatrix} 4 & 1 & 0 & \dots & 0 \\ 1 & 4 & 1 & \ddots & \\ & \ddots & \ddots & \ddots & \\ \vdots & & & & \\ & & & & 1 \\ 0 & \dots & 0 & 1 & 4 \end{bmatrix}$$

Et b un vecteur de $\mathbf{R^{n^2}}$ (il y'a n^2 pixels), exprimant les conditions aux bords et la nature du champ de guidage:

$$b = \begin{bmatrix} \sum_{q\in N_1\cap\delta\Omega} f^*_q + \sum_{q\in N_1} g_1 - g_q \\ \vdots \\ \sum_{q\in N_{n^2}\cap\delta\Omega} f^*_q + \sum_{q\in N_1} g_{n^2} - g_q \end{bmatrix}$$

3.2 Résolution du système: Méthode de Jacobi

3.2.1 Principe

La méthode de Jacobi consiste à construire une suite récurrente de la forme $x_{k+1} = F(x_k)$ de manière a ce que la suite de vecteur $(x_n)_{n\in\mathbf{N}}$ converge vers la solution de l'équation$Ax = b$.

Le principe est le suivant: il est toujours possible d'écrire $A = D - E - F$ avec D matrice diagonale, E matrice triangulaire inférieure et F matrice triangulaire supérieure .

On a alors $x = D^{-1}(E + F)x + D^{-1}b = F(x)$, la solution du système linéaire est alors un point fixe de F , et tous les points fixes de F sont des solutions du système linéaire. La convergence est assurée sous certaines conditions sur A expliquées dans [4].

3.2.2 Algorithme

Le langage utilisé est Matlab.

Algorithm 2 Résout l'équation (10)

Require: $eps > 0, A, b$
  $L \leftarrow size(A)$
  $N \leftarrow L(1)$
  $D \leftarrow diag(diag(A))$
  $G \leftarrow D * E$
  $H \leftarrow D * B$
  $R \leftarrow 10$
  $X \leftarrow zeros(N, 1)$
  while $R > eps$ **do**
    $Y \leftarrow X$
    $X \leftarrow G * X + H$
    $R \leftarrow norm(X - Y)$
  end while

3.3 Résultats:

Figure 4: Les conditions de Dirichlet imposent la continuité sur les bords: Le visage de Ronaldo a été remplacé par celui de Kroos sans modifier le reste de l'image

4 Méthode Inpainting par patchs

La deuxième méthode que nous avons mise en place est la méthode d'inpainting par patchs. Le but de cette méthode est de remplacer progressivement notre masque par des patchs qui appartiennent à l'image connue. Un patch correspond à une petite matrice de taille L (généralement L=3 ou L=5) qui est extraite de l'image d'origine. Le choix de la taille L doit être effectué de manière judicieuse afin d'optimiser la qualité du rendu final. Ce choix sera expliqué dans la suite du rapport. Le remplacement d'un patch du masque doit être fait par son plus proche voisin dans l'image connue. Au premier abord, cette méthode semble être plus efficace que la méthode basée sur l'équation de la diffusion. En effet, cette dernière méthode ne permet pas de représenter les textures de l'image. Ce problème pourrait être résolu grâce à la méthode par patchs car on se sert des patchs de l'image connue pour remplir le trou.
Afin d'être clair sur les notations, nous allons définir Oméga comme étant l'occlusion que nous cherchons à remplir progressivement. Soit Φ la zone de l'image connue, qui correspond donc à l'image privée de l'occlusion Ω. On nomme W_p comme étant le patch situé au pixel p.
Le but est donc de combler les patchs noirs par des patchs de Φ en minimisant la distance entre ces patchs. Soit W_p, un patch situé à la frontière du masque, on va chercher son plus proche voisin W_q dans Φ. Cette recherche s'effectuera en essayant de minimiser la distance d(W_p, W_q) entre ces deux patchs.
On définit la distance entre 2 patchs W_p et W_q par :

$$d(W_p,W_q)=\sum[\ W_p[i,\ j] - W_q[i,\ j]\ ]*[\ W_p[i,\ j] - W_q[i,\ j]\ ] \text{ pour } (i,j) \text{ appartenant à } [0,L]$$

où L est la taille du patch que l'on a défini.

Pour un patch donné W_p situé à la frontière, on va parcourir tous les patchs W_q dans Φ en faisant bien attention qu'il n'y ait pas intersection entre W_q et Ω. En effet, imaginons que l'intersection entre W_q et Ω ne soit pas vide. Ainsi, notre patch W_q contiendra des pixels noirs provenant du masque. W_p sera donc remplacé par un patch W_q qui contient des pixels noirs, ce qui bloque notre avancement dans le processus. Ceci est donc à bannir. Une fois cette précaution prise, on remplace W_p par son plus proche voisin, et ainsi tous les pixels noirs du patch W_p seront désormais remplacés par des pixels connus.
L'objectif de l'étude est d'itérer ce processus sur tous les patchs du masque afin d'obtenir l'image finale qui ne contiendrait plus le masque.

Notre première idée était de choisir pour le patch initial celui situé tout en haut à gauche de notre image. On itère ensuite l'algorithme en remplissant la première ligne puis la seconde, puis la troisième et ainsi de suite jusqu'à remplir totalement notre masque. On montrera ensuite à partir des résultats obtenus que cette méthode ne donne pas le résultat optimal bien que le résultat final soit satisfaisant.

Détails de quelques étapes de l'algorithme :

- On convertit en niveaux de gris notre image que l'on souhaite remplir
- On rentre en paramètre : les paramètres de notre masque à savoir mMasque et nMasque qui correspondent respectivement à la longueur et à la largeur du masque
- On crée notre image obstruée qui correspond à notre image initiale avec la présence du trou
- L'objectif de l'algorithme est donc de compléter ce trou en utilisant la méthode par patchs expliquée au paragraphe précèdent
- - On crée notre matrice masque qui correspond à la matrice du masque où la matrice aux coordonnées (i, j) vaut 1 si (i,j) appartient au trou et 0 sinon. Cette matrice masque sera mise à jour dynamiquement au fur et à mesure que l'on remplit notre trou. En effet, après la première étape de notre algorithme, lorsque le premier patch du trou Wp est remplacé par son plus proche voisin, il faut mettre à jours notre masque car le nouveau trou correspond maintenant au trou précèdent privé de W_p

Un détail important qui a été pris en compte dans l'algorithme est de modifier uniquement les pixels présents dans le trou et de ne jamais modifier les pixels connus présents autour du trou. En effet, cela pourrait biaiser l'évolution de notre processus.

Dans notre algorithme, nous avons implémenté deux manières de définir le patch. Dans une première implémentation, nous avons défini le patch Wp comme étant le patch débutant au pixel p de cordonnées (x, y).

Puis dans une deuxième implémentation, nous avons défini le patch Wp comme étant le patch centré au pixel p de coordonnées (x, y). Dans le deuxième cas, nous avons moins d'informations connues dans notre patch initial, c'est pourquoi il est nécessaire de considérer un patch W_p de taille L=5. Ceci n'était pas nécessaire dans le premier cas car, on arrivait à obtenir un maximum d'informations connues en plaçant de manière judicieuse notre patch initial. Plus le patch est de taille petite, plus le rendu final sera optimal. Cependant le patch ne doit pas être trop petit, car sinon nous avons très peu d'informations connues. Il y'a donc un compromis à faire, c'est pourquoi le choix de la taille du patch sera fixé à L=3 dans la première représentation du patch et fixé à L=5 pour le patch centré.

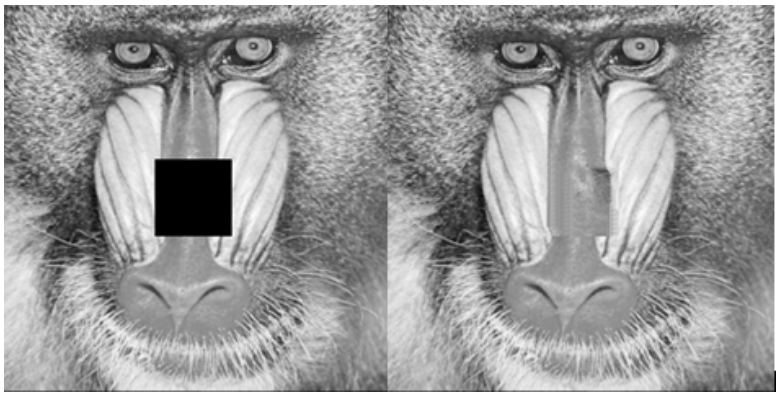
Notre deuxième méthode pour le choix du pixel sur lequel on itère, est d'introduire un critère de confiance qui décidera du choix du pixel sur lequel on itère. L'objectif est de remplir successivement les contours du patch jusqu'à atteindre le centre du trou. Ceci donnera de meilleurs résultats que la première méthode car on vient d'abord rogner les contours du trou.

Étapes supplémentaires par rapport à la première méthode :

En plus de mettre à jour la matrice masque, on introduit une nouvelle matrice qui correspond à la matrice confiance. Cette matrice confiance est aussi dynamique car elle est mise à jour à chaque fois que le trou évolue.

4.1 Présentation des résultats:

Résultats avec la première méthode, où l'on remplit successivement chaque ligne de trou:

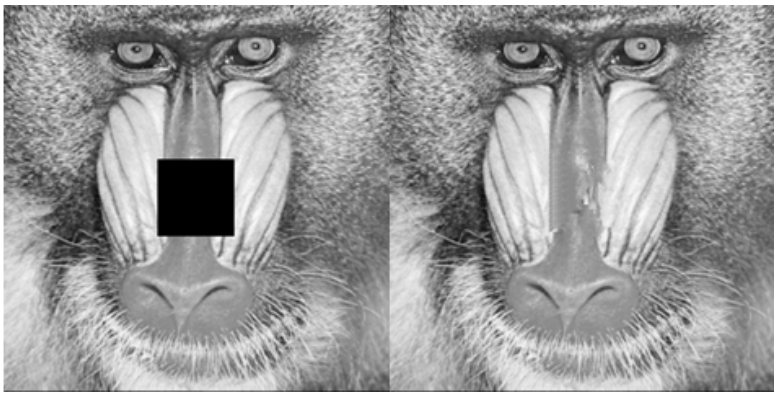


Dans le cas du babouin, on remarque que le nez est assez bien reconstruit. En effet l'algorithme a essayé de reconstruire fidèlement les couleurs propres à chaque zone. On remarque tout de même que les frontières sont assez difficiles à reconstruire. En effet, la frontière du côté gauche du nez, est brouillée et dépasse légèrement sur la partie blanche. On peut vite s'apercevoir, à partir du résultat, que le trou est traité ligne par ligne car on peut remarquer une certaine périodicité dans les lignes (surtout aux débuts des lignes). On remarque aussi qu'en fin de ligne, l'erreur se propage dans les lignes suivantes. Ceci peut être corrigé en traitant d'abord tout le contour, d'où la deuxième méthode mise en place. Dans le cas des poivrons, la reconstruction est moins

satisfaisante. Ceci est dû à l'emplacement du masque qui est dans une zone très difficile à reconstruire. On peut

émettre les mêmes remarques que dans l'image du babouin car l'erreur en fin de ligne se propage sur toutes les autres lignes. De plus, les frontières sont assez bien reconstruites à gauche du masque mais plus du tout à partir du milieu du masque.

Résultats avec la deuxième méthode, où l'on remplit d'abord les contours grâce au critère de confiance:



On remarque par rapport à la première méthode que l'erreur occasionnée dans une ligne, ne se propage plus dans la ligne suivante. Ceci permet de mieux représenter les frontières, notamment dans la partie droite du nez du babouin. Le résultat est beaucoup plus satisfaisant car la reconstruction respecte mieux l'emplacement des couleurs de l'image originale. La nouvelle méthode permet d'améliorer grandement le rendu final sur les

poivrons. En effet, la frontière entre les deux poivrons est presque réussie, ce qui était loin d'être le cas dans notre première méthode. On relève quelques mauvais choix de pixels tout de même à droite de l'image reconstruite. Mais globalement, le résultat est tout de même satisfaisant car la zone est assez difficile à reconstruire.

4.2 Minimisation d'une fonctionnelle d'énergie

Dans cette section, nous allons essayer de minimiser une fonctionnelle d'énergie obtenue suite à l'implémentation de la méthode des patchs.

On considère la fonction ϕ: $\Omega \to \mathbf{N}^2$ qui donne le plus proche voisin d'un pixel p appartenant à Ω.

La fonctionnelle qu'on souhaite minimiser prend la forme suivante :

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

avec:

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

La solution de ce problème de minimisation est obtenue en prenant pour tout p appartenant à Ω :

$$u(p) = \sum_{q \in N_p} \frac{u(p+\phi(q))}{|N_p|}.$$

Démonstration: La formule à minimiser est la suivante:

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

En remplaçant les termes de la somme par leurs expressions:

$$E(u,\phi) = \sum_{p\in\Omega} \sum_{q\in N_p} (u(q) - u(q+\phi(p)))^2$$

Cette somme est égale à la somme suivante :

$$E(u,\phi) = \sum_{p\in\Omega} \sum_{q\in N_p} (u(p) - u(p+\phi(q)))^2$$

Tous les termes de la somme sont positifs. Le problème de minimisation revient donc à minimiser le terme $\sum_{q\in N_p} (u(p) - u(p+\phi(q)))^2$ pour chaque pixel. La solution de ce problème de minimisation est obtenue grâce à la méthode des moindres carrés. Le minimum est donc atteint si pour tout pixel p $u(p) = \sum_{q\in N_p} \frac{u(p+\phi(q))}{|N_p|}$.

References

[1] A basic proof of a theorem involving eigenvalues, William Schlieper, 2012. http://www.math.ucla.edu/ was3/ax-xb.pdf

[2] http://mp.cpgedupuydelome.fr/pdf/Réduction - Calcul de polynômes caractéristiques.pdf (Exercice 2)

[3] P. Pérez, M. Gangnet, A. Blake. Poisson image editing. ACM Transactions on Graphics (SIGGRAPH'03), 22(3):313-318, 2003.

[4]http://www.math-info.univ-paris5.fr/ pastre/meth-num/MN/A-Jacobi-GaussSeidel-gradients/cours-jacobi-gaussseidel-gradients.pdf