

CGDI Inpainting

Louis Béthune et Léonard Assouline

Contents

1	L'algorithme	1
1.1	Introduction	1
1.2	Approximate Nearest Neighbor	2
1.3	La reconstruction	2
1.4	La distance	2
1.5	La pyramide	3
1.6	L'algorithme final	3
2	Implémentation	4
2.1	Le programme	4
2.2	L'organisation du code	4
2.3	Problèmes connus	4
2.4	Forces et faiblesses	5
References		5
3	Annexes	5
3.1	Réussites	5
3.2	Echecs	7

L'algorithme

Introduction

Nous avons implémenté l'algorithme de [2] tel qu'il est décrit dans leur article.

Il s'agit d'un algorithme fonctionnant à base de "patchs" : ce sont des zones rectangulaires prises ailleurs dans l'image, qu'on va utiliser pour remplir l'occlusion de façon à minimiser une certaine distance afin de produire des recollements réalistes conformes à ce qui est observé ailleurs dans l'image.

Nous n'allons pas décrire dans le détail cet algorithme (on vous renvoie à l'article original pour cela) mais nous allons le résumer.

L'image complète sera désignée par Ω , la zone non efface sera décrite par \mathcal{D} et l'occlusion sera décrite par \mathcal{H} . Une position sera décrite par $p(x, y)$ et le contenu de l'image sera décrit par $u : \Omega \rightarrow \mathbb{R}^3$. Un patch est un voisinage rectangulaire d'un point : le voisinage de positions sera désigné \mathcal{N}_p pour le point p , et le patch lui-même sera désigné W_p . Notre objectif de déterminer le plus proche voisin d'un patch donné : pour une fonction de distance d le plus proche voisin de W_p est W_q avec $q = \arg \min d^2(W_p, W_q)$.

Nous allons désigner par $\phi : \Omega \rightarrow \mathbb{N}^2$ la fonction qui à une position associe l'offset, c'est à dire la différence coordonnée à coordonnée, entre un point p et son patch associé $p + \phi(p)$. Cette information sert de support à l'algorithme.

Approximate Nearest Neighbor

Trouver le plus proche voisin est un problème difficile car une recherche exhaustive mène à des temps de calcul exagérément longs. Mais il existe une heuristique popularisé dans [1] permettant d'accélérer cette recherche à condition de sacrifier des garanties sur l'optimalité du résultat : en combinant une recherche aléatoire (exploration) et la propagation de "bons" patchs d'un pixel à un autre (exploitation).

Algorithm 1: ANN search with PatchMatch.

Data: Current inpainting configuration u (height: m , width: n), ϕ , $\bar{\mathcal{H}}$
Parameters: r_{max} ($\max(m, n)$), ρ (0.5)
Result: ANN shift map ϕ

```

for  $k = 1$  to  $k_{max}$  do
    for  $i = 1$  to  $|\bar{\mathcal{H}}|$  do
        if  $k$  even then /* Propagation on even iteration (lexicographical order) */
             $p = p_i$ ;
             $a = p - (1, 0)$ ,  $b = p - (0, 1)$ ;
             $q = \arg \min_{r \in \{p, a, b\}} d(W_p^u, W_{p+\phi(r)}^u)$ ;
            if  $p + \phi(q) \in \bar{\mathcal{D}}$  then  $\phi(p) \leftarrow \phi(q)$ 
        else
            /* Propagation on odd iteration (inverted order) */
             $p = p_{|\bar{\mathcal{H}}|-i+1}$ ;
             $a = p + (1, 0)$ ,  $b = p + (0, 1)$ ;
             $q = \arg \min_{r \in \{p, a, b\}} d(W_p^u, W_{p+\phi(r)}^u)$ ;
            if  $p + \phi(q) \in \bar{\mathcal{D}}$  then  $\phi(p) \leftarrow \phi(q)$ 
        end
         $z_{max} \leftarrow \lceil -\frac{\log(\max(m, n))}{\log(\rho)} \rceil$ ;
        for  $z = 1$  to  $z_{max}$  do
             $q = p + \phi(p) + \lfloor r_{max} \rho^z \text{RandUniform}([-1, 1]^2) \rfloor$ ;
            if  $d(W_p^u, W_{p+\phi(q)}^u) < d(W_p^u, W_{p+\phi(p)}^u)$  and  $p + \phi(q) \in \bar{\mathcal{D}}$  then  $\phi(p) \leftarrow \phi(q)$ 
        end
    end
end

```

La reconstruction

Une fois que de bons patchs candidats ont été trouvés il faut reconstruire l'image. Plein de méthodes, plus ou moins naïves, existent dans la littérature. Celle de l'article donne de bons résultats : chaque pixel est colorée avec une somme pondérée des couleurs des patchs auquel il appartient. Cette pondération dépend de la distance associée à ce patch, de façon à favoriser les couleurs des patchs appropriés à la zone à inpaintée.

Cet algorithme est donné par les formules suivantes :

$$u(p) = \frac{\sum_{q \in \mathcal{N}_p} s_q^p u(p + \phi(q))}{\sum_{q \in \mathcal{N}_p} s_q^p}$$

Avec :

$$s_q^p = \exp \frac{-d^2(W_q, W_{q+\phi(q)})}{2\sigma^2}$$

Avec σ choisi comme le 75ème percentile de toutes les distances. Cet algorithme tend à produire des images floues, ce qui n'est pas un problème lors des étapes intermédiaires. En revanche, sur l'étape de reconstruction finale, on n'effectue pas de moyenne pondérée : on garde la couleur du pixel dont la distance associée est la meilleure. On attend donc le dernier moment pour recoller des morceaux de patch entiers. Si on avait effectué cette opération plus tôt on aurait piégé l'algorithme dans un mauvais minimum local. Là, on s'assure de reproduire un résultat net et précis.

La distance

La distance est rarement discutée dans les articles du domaine. Une mesure évidente est la norme deux (prise au carré pour éviter les opérations coûteuses de calcul de racine), de la différence pixel à pixel de

chaque patch sur chaque canal **RGB**. Mais cette mesure présente des défauts : elle est susceptible de confondre deux texture dont la couleur moyenne est semblable mais dont les "motifs" (comprendre : la structure) sont différents. Dans l'article original, on propose pour solution d'utiliser une *texture feature map* T pour comme terme supplémentaire pour régler ce problème : il s'agit d'un tableau contenant des valeurs calculées à partir des dérivées de l'image originale dans les directions x et y . Il apparaît de façon heuristique que ce nouveau terme permet de lever l'ambiguïté sur les textures semblables, et augmente la qualité du résultat.

$$d^2(W_p, W_q) = \frac{1}{N} \sum_{r \in \mathcal{N}_p} (\|u(r) - u(r - p + q)\|_2^2 + \lambda \|T(r) - T(r - p + q)\|_2^2)$$

Avec :

$$T(p) = \frac{1}{\text{card}(\nu)} \sum_{q \in \nu} (|I_x(q)|, |I_y(q)|)$$

Avec I_x et I_y les dérivées dans les directions x et y respectivement.

Nous avons également testé d'autres distances, comme par exemple l'angle entre les deux patchs (chaque patch vu comme un vecteur). Cela n'a pas donné de résultats satisfaisants.

La pyramide

Cet algorithme utilise une pyramide gaussienne : c'est à dire qu'il construit une suite d'images par *downsampling* successifs, complétés d'une interpolation gaussienne. Cette pyramide a une hauteur telle que dans l'image de résolution minimale, la taille de l'occlusion est du même ordre de grandeur que le patchsize. Cela permet de voir l'inpainting comme un procédé itératif : on commence par inpainter les images de faible résolution pour reconstruire de larges zones de l'image de façon cohérente, puis progressivement on *upsample* le résultat obtenu pour affiner sa qualité et inpainter des détails de plus en plus précis (dans les images de haute résolution le patchsize est souvent petit devant la taille de l'occlusion). Se référer à l'article original pour plus de détails.

L'algorithme final

Algorithm 2: Complete proposed inpainting algorithm.

Data: Input image u , occlusion \mathcal{H}

Result: Inpainted image

```

 $\{u^\ell\}_{\ell=1}^L \leftarrow \text{ImagePyramid}(u);$  // Equation (6)
 $\{T^\ell\}_{\ell=1}^L \leftarrow \text{TextureFeaturePyramid}(u);$ 
 $\{\mathcal{H}^\ell\}_{\ell=1}^L \leftarrow \text{OcclusionPyramid}(\mathcal{H});$ 
 $\phi^L \leftarrow \text{Random};$ 
 $(u^L, T^L, \phi^L) \leftarrow \text{Initialization}(u^L, T^L, \phi^L, \mathcal{H}^L);$  // Equation (11)
for  $\ell = L$  to 1 do
     $k = 0, e = 1;$ 
    while  $e > 0.1$  and  $k < 10$  do
         $v = u^\ell;$ 
         $\phi^\ell \leftarrow \text{ANNsearch}(u^\ell, T^\ell, \phi^\ell, \mathcal{H}^\ell);$  // Algorithm 1
         $u^\ell \leftarrow \text{Reconstruction}(u^\ell, \phi^\ell, \mathcal{H}^\ell);$  // Equation (3)
         $T^\ell \leftarrow \text{Reconstruction}(T^\ell, \phi^\ell, \mathcal{H}^\ell);$ 
         $e = \frac{1}{3|\mathcal{H}^\ell|} \|u_{\mathcal{H}^\ell}^\ell - v_{\mathcal{H}^\ell}\|_1;$ 
         $k \leftarrow k + 1;$ 
    end
    if  $\ell = 1$  then
         $u \leftarrow \text{FinalReconstruction}(u^1, \phi^1, \mathcal{H});$  // Equation (5)
    else
         $\phi^{\ell-1} \leftarrow \text{UpSample}(\phi^\ell, 2);$  // Section 3.5
         $u^{\ell-1} \leftarrow \text{Reconstruction}(u^{\ell-1}, \phi^{\ell-1}, \mathcal{H}^{\ell-1});$ 
         $T^{\ell-1} \leftarrow \text{Reconstruction}(T^{\ell-1}, \phi^{\ell-1}, \mathcal{H}^{\ell-1});$ 
    end
end

```

Implémentation

Le programme

Le programme a été écrit en **C++14** et utilise les fonctionnalités d'**OpenCV** (version **3.1** et supérieur). Il y a une dépendance avec `Boost.Options`, et les options disponibles peuvent être affichées en tapant `-h` ou `--help`.

On peut fournir un masque avec l'option `-m` mais c'est optionnel. Si aucun masque n'est fourni une fenêtre va s'ouvrir et permet à l'utilisateur de créer son propre masque en cliquant dans l'image, régler la taille du pinceau avec les touches `+` ou `-`. Le masque créé apparaît alors en négatif. Si l'option `-c` est activée le masque sera enregistré sur le disque avec le nom fourni en argument.

L'option `-p` contrôle la taille des patchs. Il vaut mieux utiliser des entiers impairs. L'option `-l` permet de modifier λ , c'est à dire le tradeoff entre les distances pixel à pixel et la *feature map*.

Le temps d'exécution dépend de plusieurs facteurs, parmi lesquels la taille de l'image, la taille de l'occlusion, et la taille des patchs (ce dernier paramètre peut, pour une même image grand format, faire varier le temps d'exécution de quelques secondes à presque une minute). L'option `-a` qui contrôle le nombre d'itérations de ANN peut servir à directement modifier le temps d'exécution, au prix d'une recherche de plus proches voisins plus ou moins efficace.

Durant la production de l'image l'image va être successivement affichées en plusieurs dimensions (une pour chaque niveau de la pyramide) pour qu'on puisse suivre en temps réel l'inpainting. Aussi, il ne faut considérer l'inpainting comme terminé que lorsque "done" s'affiche dans la console.

L'organisation du code

Le fichier `cvui.h` est une librairie extérieure que nous n'avons pas codé. Les fichiers `gui_skeleton` contiennent le code de l'éditeur de masque interactif. Les fichiers `inpainting` contiennent le squelette général de l'algorithme. Les fichier `ANNsearch` contiennent l'implémentation de l'algorithme ANN ainsi que quelques utilitaires pour manipuler des patchs ou des distances. Pour finir les fichiers `reconstruction` contiennent les codes de reconstruction et d'initialisation de l'image. Le total fait à peu près 600 lignes.

Problèmes connus

Cet algorithme est assez difficile à implémenter : il y a des ambiguïtés sur beaucoup d'aspects, et il est de plus très volumineux. Il est toujours difficile de déboguer des algorithmes d'image car c'est rarement possible de distinguer un résultat qui est mauvais à cause d'un bug d'un résultat qui est mauvais parce qu'il s'agit d'un défaut inhérent à l'algorithme.

Aussi, je soupçonne plusieurs bugs d'être présents.

Les flottants sont affreux Une propriété très perturbante des flottants : lorsque x est "trop" négatif (disons, $x = -500$) alors $\exp x$ peut retourner 0. C'est mathématiquement invalide mais c'est une erreur d'arrondi qui ne manque pas d'arriver dans les équations impliquant la fonction \exp . Elle mène alors à des divisions par 0 puis par de terribles propagations de *NaN* qui résultent en de gros artefacts. J'ai pseudo-réolu ce problème en arrondissant chaque 0 à $\epsilon_{machine}$. Ce n'est pas entièrement satisfaisant mais ça limite les dégâts.

Ambiguïtés partout L'article ne précise pas si on travaille sur des couleurs avec un domaine "classique" (de 0 à 255) ou normalisé (de 0 à 1) or cela a une influence directe sur : l'architecture du code, et la valeur des constantes (λ etc...).

Initialisation problématique La qualité du résultat final dépend beaucoup de la fonction **Onion-PeelInitialization**. Or je la soupçonne de ne pas fonctionner correctement en raison des phénomènes sus-nommés. Elle produit des grandes zones uniformément colorés. Cela piège l'algorithme dans de terribles minimums locaux où l'occlusion est inpaintée avec une solution indépendante des conditions au bord (pourtant la base d'un inpainting réussi).

Forces et faiblesses

Cet algorithme éprouve beaucoup de difficultés lorsque l'occlusion chevauche des zones avec une structure importante et beaucoup de couleurs différentes. L'exemple de l'homme devant des lampions est l'exception qui confirme la règle. Dans les annexes vous pouvez constater plusieurs échecs. De plus cet algorithme produit un résultat différent à chaque exécution : il peut être nécessaire de le lancer plusieurs fois avec les mêmes paramètres avant de trouver un résultat satisfaisant. Cela peut-être un avantage (diversité) ou un défaut (manque de garanties, temps d'exécution total plus important que celui d'une unique exécution).

Il possède l'avantage d'être assez rapide : on peut régler les paramètres de façon à le rendre très rapide sans trop la qualité du résultat. Les expérimentations ci-dessous ont été réalisées avec les valeurs par défaut pour λ et pour le nombre d'itérations de **ANN**.

References

- [1] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics-TOG*, 28(3):24, 2009.
- [2] Alasdair Newson, Andrés Almansa, Yann Gousseau, and Patrick Pérez. Non-local patch-based image inpainting. *Image Processing On Line*, 7:373–385, 2017.

Annexes

Réussites



Figure 1: Autoroute

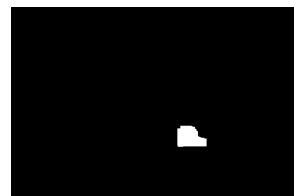


Figure 2: Masque appliqué



Figure 3: patchsize=17



Figure 4: Vélos



Figure 5: Masque appliqué



Figure 6: patchsize=17

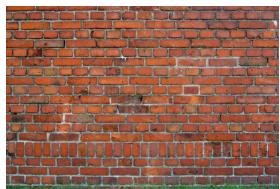


Figure 7: Mur

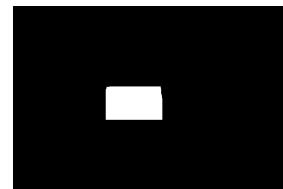


Figure 8: Masque appliqué

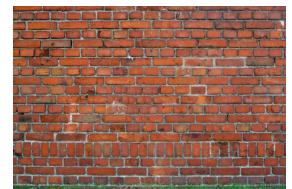


Figure 9: patchsize=29



Figure 10: Buildings



Figure 11: Masque appliqué



Figure 12: patchsize=17



Figure 13: Femme devant la mer



Figure 14: Masque appliqué



Figure 15: patchsize=27



Figure 16: Vaches

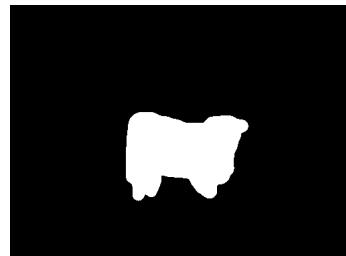


Figure 17: Masque appliqué



Figure 18: patchsize=7



Figure 19: Homme devant des lampions



Figure 20: Masque appliqué



Figure 21: patchsize=29



Figure 22: Homme devant des lampions



Figure 23: Masque appliqué

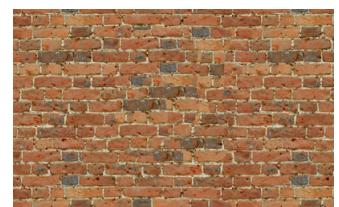


Figure 24: patchsize=17

Echecs



Figure 25: Femme devant la fontaine

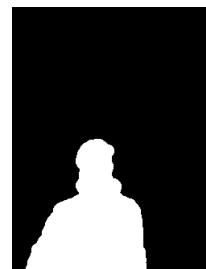


Figure 26: Masque appliqué



Figure 27: patchsize=19



Figure 28: Bateau

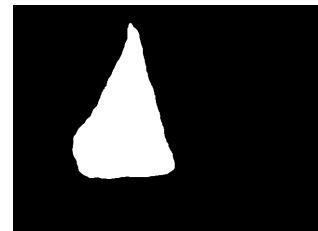


Figure 29: Masque appliqué



Figure 30: patchsize=9



Figure 31: Saut



Figure 32: Masque appliqué



Figure 33: patchsize=17



Figure 34: Saut

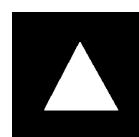


Figure 35: Masque appliqué



Figure 36: patchsize=9