Traitement numérique des images

Synthèse de textures par rééchantillonnage de patchs

Aurélien Boffy - Vincent Michel

janvier 2007

Introduction

Nous présentons dans ce rapport un algorithme récent pour le problème du remplissage automatique d'une région d'une image.

Une solution à ce problème a des nombreuses applications, de la restauration d'images qui ont été endommagées (tâches, déchirures, etc.), à la correction d'images (par exemple pour enlever des objets ou des personnes indésirables sur des photos de vacances, ou encore retoucher des visages pour supprimer des rides ou des imperfections), en passant par la manipulation d'images pour réaliser des photos montages.

C'est un problème connu et de nombreuses méthodes ont déjà été essayées, le plus souvent en tentant de synthétiser la texture de la zone manquante en essayant de prolonger vers l'intérieur les contours qui sont connus (cette approche est connue sous le nom d'« *inpainting* »). Cependant, lorsque la région endommagée est trop grosse ou les textures à synthétiser trop complexes, les résultats ne sont pas exploitables.

Le principe du travail que nous présentons dans ce rapport est différent : la reconstruction de l'image considérée est réalisée en remplaçant les zones manquantes ou endommagées par des copier/coller des zones connues et en les assemblant comme pour un puzzle. De tels algorithmes sont apparus en 1999 et nous présentons ici une variante qui a été introduite par Patrick Pérez *et al.*, dans un rapport de recherche de *Microsoft Research* de 2004, intitulé *PatchWorks : Example-Based Region Tiling for Image Editing*. Une des spécificités de la méthode est de synthétiser la zone manquante patch par patch et non pixel par pixel comme c'était généralement le cas auparavant. Ainsi, les temps de calcul sont significativement réduits et les résultats sont généralement meilleurs lorsque la texture à synthétiser est complexe.

Dans un premier temps, nous allons faire une brève présentation de l'algorithme de l'article, puis, dans une seconde partie, nous présenterons de nombreux exemples afin de tenter d'expliquer le rôle des différents paramètres, et de détailler les avantages et inconvénients de la méthode. Enfin, dans une dernière partie, les fonctionnalités du programme que nous avons implémenté seront brièvement décrites.

1 L'algorithme *PatchWorks*

1.1 Formalisme

1.1.1 Généralités

On dispose d'une image I dont une zone Ω a été clairement identifiée comme une région qu'il faut reconstruire, soit parce qu'elle est abîmée, soit par exemple parce qu'elle représente un objet ou une personne que l'on souhaiterait effacer de l'image.

Le but est de resynthétiser cette zone Ω, patch par patch, en copiant des morceaux du reste de l'image, de telle façon à ce que ceux-ci s'intègrent au mieux avec le voisinage déjà existant, sans qu'on voit les zones de recollement.

Pour ce faire, on établit un pavage de la région Ω, puis, pavé après pavé, et selon un ordre bien défini dont nous parlerons plus loin, nous remplaçons les patchs dont les couleurs sont partiellement ou complètement inconnues par des patchs choisis dans un dictionnaire qui a été construit préalablement en utilisant les zones connues de l'image.

1.1.2 Construction du dictionnaire

Le dictionnaire est simplement construit en choisissant des patchs de l'image I qui sont tels que les pixels qui les composent, ainsi que les pixels de leur voisinage, soient tous dans Ω^c. Le voisinage d'un pavé est tout simplement constitué des pixels situés dans une bande d'épaisseur fixée autour de ce pavé.

Plus précisément, on définit une « source » qui est une zone de l'image dans laquelle on peut aller chercher des mots du dictionnaire. Il est en effet généralement inutile de considérer l'intégralité de l'image, car cela aboutirait à un énorme dictionnaire et donc à des calculs très lents. Par défaut, on ne considère ainsi qu'une bande d'épaisseur fixée autour de la région Ω car la texture à synthétiser dans la zone abîmée est généralement un prolongement de la texture située à côté.

Néanmoins, dans certains cas, on peut souhaiter vouloir définir la source des mots du dictionnaire manuellement, par exemple pour empêcher qu'une certaine texture du bord de Ω ne se propage.

1.1.3 Distance entre patchs

Lorsqu'on veut remplacer un pavé inconnu i par un mot du dictionnaire, comme les couleurs des pixels du pavé i sont par définition inconnus, ce sont bien-sûr les pixels de son voisinage que l'on examine pour trouver le mot du dictionnaire $\alpha(i)$ qui s'intégrerait le mieux. On calcule une distance de ressemblance entre ce voisinage et les voisinages de tous les patchs qui sont

dans le dictionnaire, et on conserve le mot $\alpha(i)$ du dictionnaire qui minimise cette distance :

$$\alpha(i) = \underset{j \in \mathcal{D}}{\operatorname{argmin}}\, d(i,j)$$

où $d(i,j)$ se calcule en faisant la somme des distances entre les couleurs de chaque pixel connu du voisinage du pavé i et les pixels du voisinage du pavé j correspondants. La distance entre couleurs est simplement calculée en faisant la somme des valeurs absolues des différences des 3 composantes R, V et B. Cette distance est basique, mais donne des résultats aussi satisfaisants que d'autres distances qui demandent des temps de calcul plus longs.

1.1.4 Ordre de parcours pour le remplissage

Différents sens de parcours peuvent utilisés pour synthétiser la zone abîmée. Nous en avons notamment testé trois différents :

Lexicographique : on parcourt le pavage de la zone à remplir de gauche à droite et de haut en bas.

Concentrique : on part d'un pavé sur le bord de la zone à reconstruire (qui peut être choisi comme le pavé pour lequel on a le plus d'informations, c'est-à-dire celui dont on connaît le plus de voisins), puis on parcourt le bord en tournant, et en formant ainsi une sorte de spirale.

« Plus de voisins » : on considère à chaque fois le pavé dont le voisinage contient le plus de pixels connus.

1.2 L'algorithme

Finalement, l'algorithme de *PatchWorks* peut être résumé ainsi :

Pseudo-code

Construction du dictionnaire $\mathcal{D}$
Tant que tous les pavés ne sont pas remplis
- Choix, selon un sens de parcours prédéfini, du prochain pavé i à traiter
- Recherche dans le dictionnaire du patch $\alpha(i)$ qui s'intégrerait le mieux
- Copie du patch $\alpha(i)$ à la position du pavé i

Fin tant que

1.3 Les paramètres

Les principaux paramètres de l'algorithme sont les suivants :

Largeur des patchs : un des principaux apports de l'article de P. Pérez *et al.* et de synthétiser la région abîmée patch par patch et non pixel par pixel comme c'était généralement le cas auparavant. La taille des patchs est généralement comprise entre 4×4 et 20×20 pixels. Nous verrons dans la section suivante l'influence de ce paramètre.

Épaisseur des voisinages : plus le voisinage considéré pour la recherche du patch du dictionnaire le plus ressemblant est grand, plus la quantité d'informations utilisée est importante, mais ce n'est pas nécessairement un avantage, comme nous le verrons plus loin. L'épaisseur du voisinage autour du patch est typiquement comprise entre 2 et 4 pixels.

Source du dictionnaire : si l'on décide de ne pas définir manuellement la zone qui pourra servir pour la source du dictionnaire, celle-ci est définie automatiquement en considérant le voisinage d'une épaisseur donnée autour de la zone à remplir.

Sens de parcours : comme nous l'avons évoqué plus haut, nous avons testé différents sens de parcours pour le remplissage de la zone abîmée et nous verrons que ceux-ci ont une importance non négligeable.

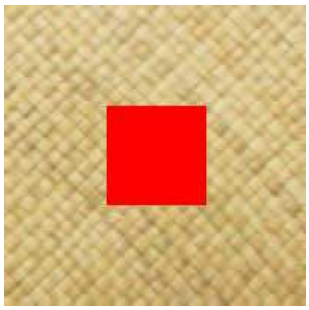
Dans la section suivante, nous allons voir l'influence de ces paramètres sur la reconstruction des images.

2 Exemples et considérations pratiques

2.1 Influence de la taille des patchs

La taille des patchs utilisée est certainement le paramètre le plus important : selon sa valeur, les résultats peuvent varier énormément, et il n'y a bien sûr pas de valeur optimale, car selon l'image considérée, il vaut parfois mieux utiliser des patchs de petite ou de grande taille.

Nous avons par exemple étudié l'influence de la taille du patch sur l'image texturée suivante (200×200 pixels) :



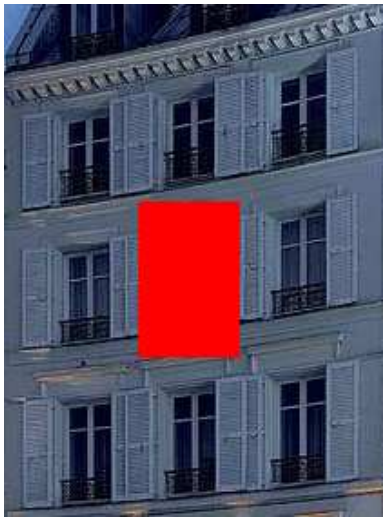
Dans chacun de nos exemples, la zone en rouge est reconnue par le programme comme étant la zone abîmée. Nous fixons les valeurs des autres paramètres (en l'occurrence, l'épaisseur des voisinages est fixée à 2 pixels, l'épaisseur de la source à 50 pixels et nous utilisons un remplissage concentrique).

Nous obtenons les résultats suivants :

FIG. 1 – Largeur des patchs, de gauche à droite : 1, 5 et 10 pixels

On constate que les résultats sont meilleurs lorsque le patch est plus grand car il permet de capturer la taille caractéristique de la texture. De plus, il faut noter qu'on améliore significativement la rapidité de l'algorithme en utilisant des patchs de grandes dimensions : en effet, même si la mesure de distance entre deux patchs est plus longue à calculer, il y a moins de pavés à remplir et il y a légèrement moins de mots dans le dictionnaire.

Considérons désormais une image où ce que l'on souhaite resynthétiser est non plus une texture mais plutôt de véritables motifs qui peuvent être de grandes tailles, comme pour la photo de la façade d'immeuble ci-dessous (200×280 pixels) :



Toujours pour un voisinage épais de 2 pixels, nous obtenons les résultats suivants :

FIG. 2 – Largeur des patchs, de gauche à droite : 5, 20 et 40 pixels

On remarque qu'il faut utiliser de très gros patchs (40×40 pixels) si l'on veut être capable d'obtenir un résultat satisfaisant. Bien-sûr, la synthèse est alors réalisée en quelques secondes malgré la taille importante de la zone abîmée.

Le cas extrême est quand la zone abîmée revient à l'identique, ou presque, une seule fois dans l'image. Il faut alors prendre une zone source très grande, et un patch qui correspond à peu près à la taille de l'objet. On a ici un exemple frappant avec la reconstruction d'un œil à partir de l'autre :

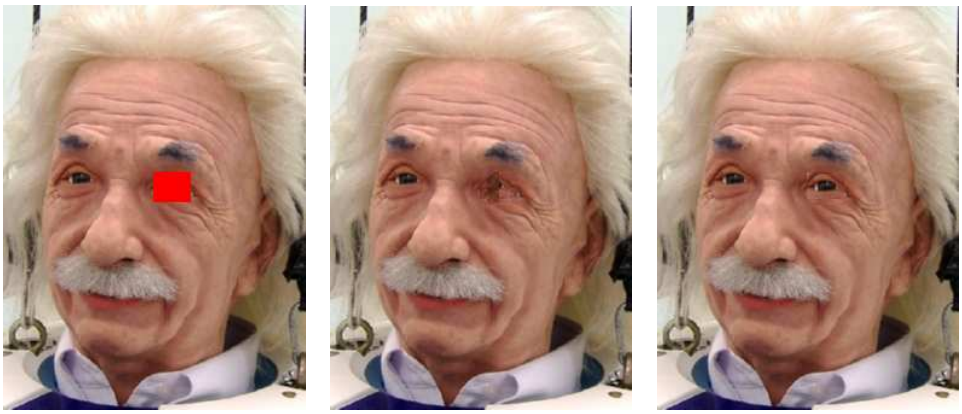


FIG. 3 – Image abîmée et restauration obtenue avec des patchs de 10×10 et de 40×40 pixels

Dans le cas d'images ayant une structure plus désordonnée, une petite taille de patch permet au contraire de mieux cerner l'élément structurant, comme ici une fourmi. En effet, on voit les bords des patchs lorsqu'ils sont trop larges (40 pixels), alors que la reconstruction est très correcte pour une largeur de 5 pixels :

FIG. 4 – Restauration obtenue avec des patchs de 5×5 et de 40×40 pixels

De même, et ceci nous permet de voir une autre application de l'algorithme, pour cet exemple où les structures sont de taille limitée, il vaut mieux de pas utiliser des patchs trop gros. En effet, si l'on utilise par exemple des patchs de 40×40, il est tout simplement souvent impossible de trouver dans la source un patch aussi gros qui s'intègre bien tout le long du voisinage, d'autant plus que la taille des éléments de la structure varie assez rapidement spatialement.

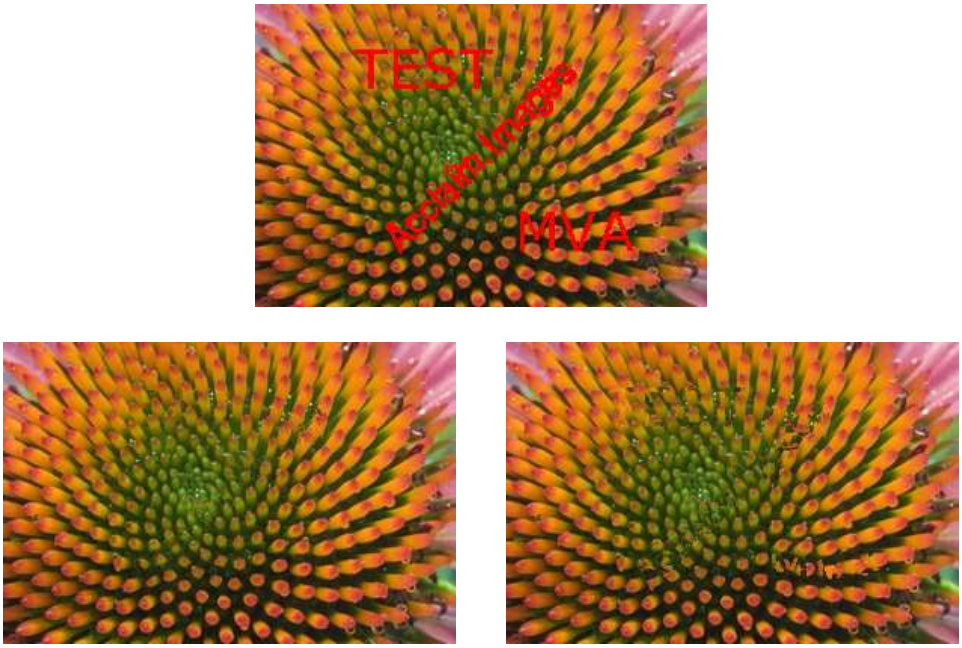


FIG. 5 – Restauration obtenue avec des patchs de 5×5 et de 40×40 pixels

2.2 Influence de l'épaisseur des voisinages

Le voisinage des patchs est la bande qui les entoure et qui est utilisée dans le calcul de distance. Il est donc utilisé pour choisir le patch de la source qui s'intégrerait au mieux avec l'environnement du patch considéré.

Nous avons étudié l'influence de la taille du voisinage sur l'exemple de texture suivant (image de 200×200 pixels) avec une largeur de patchs fixée à 10 pixels :

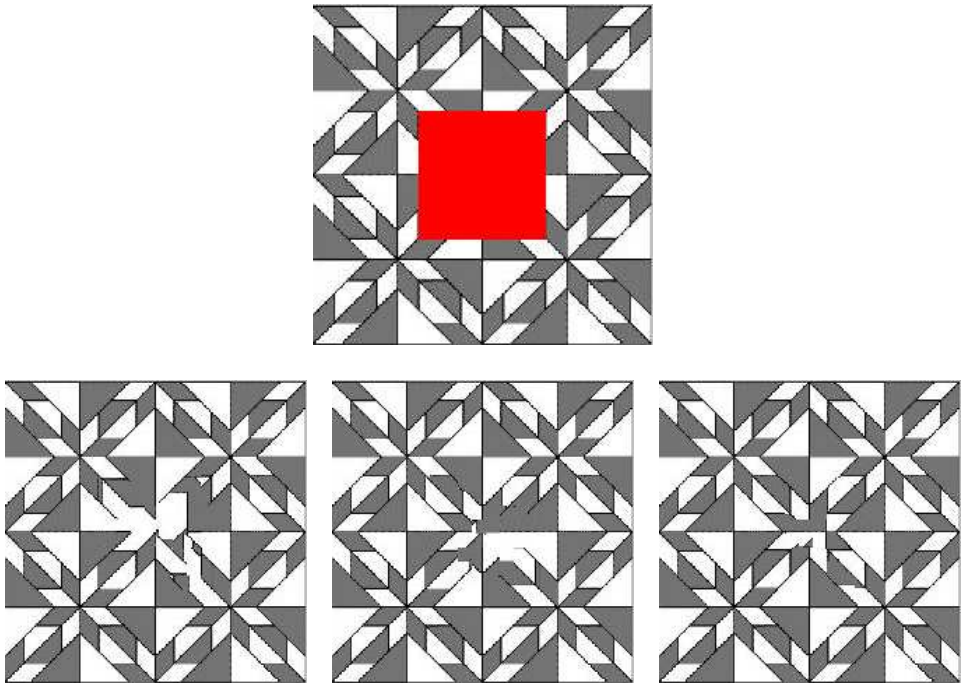


FIG. 6 – Épaisseur des voisinages de gauche à droite : 2, 5 et 10 pixels

Logiquement, on constate que la qualité de la reconstruction augmente avec l'épaisseur des voisinages considérés. Néanmoins, le temps de calcul devient aussi de plus en plus long.

On constate sur ces images que le centre n'est jamais vraiment bien reconstruit. Le fait que le problème ait lieu au centre est lié au sens de parcours de remplissage qui a été ici choisi concentrique, c'est-à-dire « en spirale » : une fois arrivé au centre, il faut choisir des patchs dont on connaît déjà quasiment tout le voisinage, ce qui est l'étape la plus difficile.

De plus, la reconstruction a ici tout simplement échoué car les patchs qu'il aurait fallu copier ne sont pas présents dans le reste de l'image et donc il est de toute façon impossible de les « inventer ».

L'épaisseur du voisinage est bien-sûr très liée avec la largeur des patchs. En effet, utiliser de plus gros patchs permet aussi d'une certaine façon d'utiliser un voisinage contenant plus de pixels.

Dans les exemples sur l'influence de la taille des patchs, nous avons vu qu'il convenait d'utiliser des patchs de grande taille lorsque l'on souhaitait resynthétiser des motifs eux-mêmes de grandes dimensions. Le plus souvent néanmoins, on peut obtenir des résultats semblables avec des patchs plus petits mais de très grands voisinages. Prenons le cas de cette image de taille 400×400 pixels :

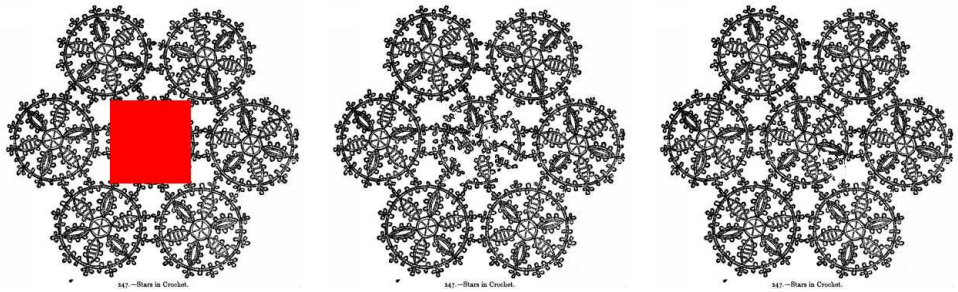


FIG. 7 – Reconstitution avec des voisinage d'épaisseur 3 puis 10 pixels

Pour obtenir ces résultats, nous avons utilisé des patchs d'une largeur fixée à 4 pixels. Ainsi, on voit qu'on est capable d'obtenir de très bons résultats, rien qu'en augmentant l'épaisseur du voisinage.

Cependant, pour obtenir la figure de droite, il a fallu laisser tourner l'algorithme une bonne dizaine de minutes, alors qu'en utilisant un voisinage d'une épaisseur de 3 pixels comme pour l'image du milieu, et en augmentant la largeur du patch de 4 à 20 pixels, on peut obtenir un résultat parfait en quelques secondes :

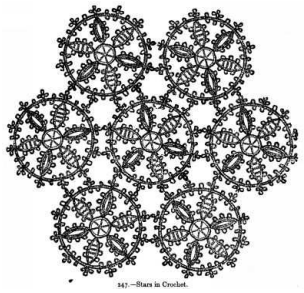


FIG. 8 – Reconstitution avec des voisinages d'une épaisseur de 3 pixels et des patchs d'une largeur de 20 pixels

Voici ci-dessous un autre exemple (image de 200×200 pixels) où nous nous contentons d'augmenter l'épaisseur des voisinages, en conservant une largeur de patchs fixée à 10 pixels :

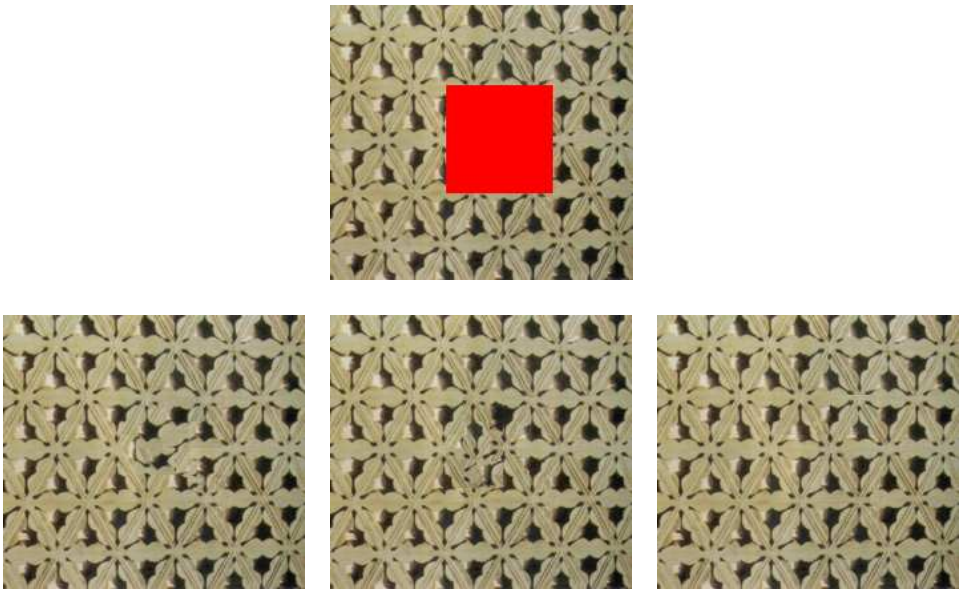


FIG. 9 – Épaisseur des voisinages de gauche à droite : 1, 2 et 5 pixels

Pour finir cette section sur l'influence commune de la taille des patchs et de l'épaisseur des voisinages, considérons cette image de 250×250 pixels que nous avons abîmée, en faisant attention de ne bien supprimer uniquement des zones qui sont présentes ailleurs dans l'image et qui donc pourront être recopiées :

FIG. 10 – Image originale et image abîmée

Nous savons qu'il est possible de récupérer l'image initiale vue que l'on a à faire à une image synthétique et que toutes les données manquantes sont présentes ailleurs dans l'image. Nous avons dans un premier temps fixé l'épaisseur du voisinage à 1 pixel et nous avons pu vérifier que l'on n'arrivait à retrouver l'image originale qu'avec une largeur de patchs égale au minimum à 16 pixels :

FIG. 11 – Largeur des patchs, de gauche à droite : 2, 6 et 18 pixels

Néanmoins, lorsqu'on augmente l'épaisseur du voisinage de 1 à 4 pixels, il suffit d'utiliser des patchs de largeur 8 pour retrouver l'image originale.

On peut alors se demander ce qui est le plus intéressant : des patchs de taille moyenne avec des assez gros voisinages ou des tout petits voisinages mais avec des très gros patchs ? D'un point de vue rapidité d'exécution, utiliser de très gros patchs est bien-sûr le plus avantageux. Néanmoins, on peut imaginer que lorsqu'on travaille avec des pavés de grandes dimensions, il est plus difficile de les assembler sans qu'on puisse discerner leurs bordures.

2.3 Influence du sens de parcours pour le remplissage

Rappelons que nous pouvons utiliser trois types différents de sens de parcours pour remplir la zone abîmée : de façon lexicographique, de façon concentrique, ou en choisissant à chaque fois le pavé dont on connaît le plus de voisins.

Le mode lexicographique s'est souvent avéré dans nos expériences le moins performant de tous, notamment car il introduit une anisotropie puisqu'il se fait dans un sens particulier et donc favorise certaines portions de l'image. Pour constater ce phénomène, étudions la reconstruction de cette photo de montagnes :

FIG. 12 – Image originale

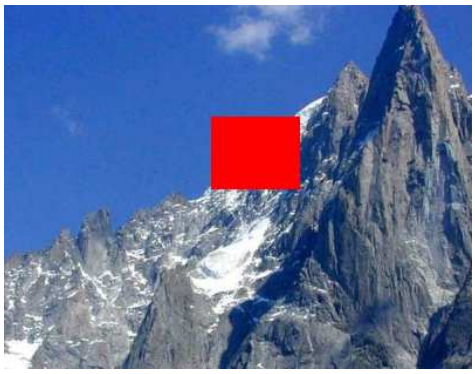


FIG. 13 – Zone à reconstruire

Voyons les résultats obtenus avec les trois sens de parcours, en conservant bien évidemment les autres paramètres égaux (à savoir notamment une largeur de patchs de 12 pixels et un voisinages épais de 4 pixels, en sachant que la taille de l'image est de 400×300 pixels) :

FIG. 14 – Reconstruction en utilisant, de gauche à droite, un sens de parcours lexicographique, concentrique et « par plus de voisins »

On constate, comme nous l'avions annoncé, que le sens de parcours lexicographique donnent les moins bons résultats car comme on se déplace de gauche à droite et de haut en bas pour reconstruire la scène, la texture située en haut à gauche et favorisée. En effet, pour cette image, l'algorithme

cherche typiquement un patch à copier à droite d'une partie de ciel, choisit naturellement un pavé contenant lui aussi uniquement du ciel, et ainsi de suite.

Le sens de parcours concentrique permet lui une bien meilleure reconstruction, alors que la méthode qui choisit à chaque fois le pavé qui a le voisinage contenant le plus de pixels connus a des résultats intermédiaires. Ceci s'explique par le fait qu'avec ce sens de parcours spécifique, on commence bien à traiter les pixels du bord de la zone à reconstruire, car le pavage est tel que les pavés des contours contiennent généralement une partie de pixels connus. Cependant, une fois que tout le contour a été traité, choisir les pixels qui ont le plus de voisins connus revient à suivre l'ordre lexicographique car on commence par un des 4 coins du rectangle, et une fois ce coin traité, les 3 autres coins ainsi que les 2 voisins du pavé qui vient d'être traité ont autant de pixels du voisinage connus, mais l'implémentation doit bien définir un choix par défaut et il se trouve que ce choix revient à parcourir les pavés dans l'ordre lexicographique.

La méthode consistant à choisir à chaque fois le pavé qui a le plus de voisins n'est en fait pas prévue pour de tels cas où la zone à remplir est un simple rectangle. Quand la zone est plus complexe, elle tire partie du fait qu'elle commence à remplir les zones pour lesquelles on a le plus d'informations et donc qu'on peut compléter avec le plus de sûreté.

D'une manière générale, nous avons néanmoins testé ces trois sens de parcours sur de nombreux exemples, et il en resort surtout qu'il est difficile d'aboutir à de véritables enseignements, si ce n'est le fait que l'ordre lexicographique semble être mis en difficulté dans des cas comme celui exposé ci-dessus. Selon l'image considérée, et même selon les autres paramètres comme la taille des patchs et des voisinages, le sens de parcours donnant les meilleurs résultats varie beaucoup et nous préférons ne pas tirer de conclusions trop hâtives.

2.4 Influence de la zone utilisée comme source du dictionnaire

Par défaut, le dictionnaire est construit à partir de patchs qui sont situés dans une bande autour de la zone abîmée. La largeur de cette bande doit être adaptée par l'utilisateur, de façon à ce que ce que la texture qu'il souhaite être synthétisée dans la zone à reconstruire y soit incluse. Ainsi, dans le cas d'une texture de petite taille caractéristique et à peu près constante sur toute l'image, une bande de petite largeur est suffisante.

Dans certains cas, l'utilisateur peut avoir intérêt à choisir manuellement la zone utilisée comme source du dictionnaire. Nous avons souhaité refaire une expérience présentée dans l'article car le résultat nous paraissait particulièrement impressionnant : nous partons de l'image suivante de taille 600×450 pixels et nous souhaitons supprimer la femme :

FIG. 15 – Image originale

Si nous effectuons les mêmes étapes que d'habitude, nous colorions en rouge la femme afin de l'effacer puis nous lançons l'algorithme. Nous obtenons, selon les paramètres choisis, des résultats comme celui-ci :

FIG. 16 – Zone à reconstruire définie manuellement

FIG. 17 – Résultat avec source définie autour de la zone abîmée

On s'aperçoit ainsi par exemple que plusieurs mains du bébé ont été synthétisées au milieu des fleurs. Ceci peut en effet se produire lorsque par exemple un patch p_1 contenant des roses a déjà été reconstruit et que l'algorithme cherche quel patch p_2 il pourrait copier à sa droite. Si le patch p_1 contient le même agencement de roses que la zone située à gauche de la main du bébé sur l'image originale, il n'y a rien pour empêcher le programme de copier le patch p_2 correspondant à la main du bébé.

La solution pour éviter ce genre de problèmes est d'effacer la femme en deux étapes. Dans un premier temps, on n'efface que la zone que l'on souhaite remplacer par des fleurs et l'on définit manuellement la source à utiliser par le dictionnaire comme ci-dessous :

FIG. 18 – Image avec la zone à reconstruire en rouge et la zone utilisée comme source du dictionnaire en vert

On lance alors l'algorithme pour qu'il synthétise uniquement des fleurs, puis on recommence l'opération en désignant cette fois le bras de la femme comme zone à effacer et le pantalon/pull du bébé comme source du dictionnaire. On arrive ainsi à obtenir le résultat suivant :

FIG. 19 – Image reconstruite en décomposant la synthèse de textures en deux étapes.

On peut noter que le contour inférieur du pantalon du bébé est un peu trop « net ». Il faut bien comprendre qu'il n'a pas vraiment été « synthétisé » en ce sens qu'aucun patch contenant de la bordure de pantalon n'a été utilisé. La bordure provient simplement du partage que l'on a fait manuellement entre la zone à synthétiser par des fleurs et la zone à synthétiser par du blanc. Ceci permet d'expliquer ce petit défaut de la reconstruction.

2.5 Autres exemples

Nous montrons dans cette section d'autres expériences que nous avons effectuées pour tester l'algorithme. Ce premier exemple montre qu'il est possible d'effacer de très grosses parties de l'image avec un résultat assez satisfaisant (la photo est de taille 400×300 pixels) :

FIG. 20 – Photo originale

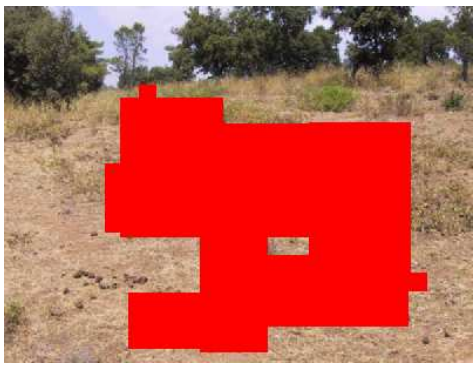


FIG. 21 – Partie à reconstruire

FIG. 22 – Deux exemples d'images reconstruites sans l'âne, avec différents paramètres

Lorsque l'image originale a déjà une texture qui n'est pas clairement définie, il paraît même difficile, pour un non initié, de savoir reconnaître la peinture originale de Cézanne de l'image reconstruite :

FIG. 23 – Tableau original

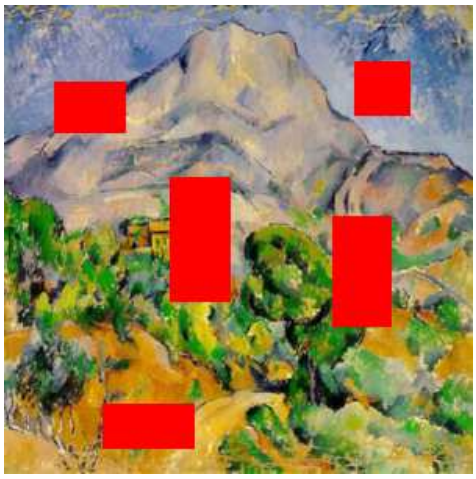


FIG. 24 – Tableau abîmé

FIG. 25 – Tableau reconstruit

L'algorithme de *PatchWorks* peut aussi permettre de synthétiser des nouvelles parties d'une image, en rajoutant par exemple une marge tout autour de l'image originale. Nous avons ainsi augmenté artificiellement la taille de l'image ci-dessous :

FIG. 26 – Image original

FIG. 27 – Image « agrandie »

On peut constater un défaut au niveau du prolongement de la colline sur la droite. Ce problème était prévisible, car il n'existe de toute façon nulle part ailleurs dans l'image une zone qui pourrait être copiée ici de façon adéquate, car le bas de la colline est située devant un ciel plus clair et le copier/coller générerait un problème au niveau du ciel...

3 Notre implémentation

Nous avons implémenté l'algorithme de *PatchWorks* et nous l'avons notamment testé avec la quasi-totalité des exemples qui figurent dans l'article en ayant généralement des résultats aussi bons en termes de qualité de reconstruction. Les performances de rapidité d'exécution pourraient elles être facilement améliorées car nous avons fait une implémentation assez basique, sans utiliser de structures de données particulières qui pourraient par exemple être exploitées pour la recherche du patch au voisinage le plus ressemblant parmi tous les mots du dictionnaire.

Les performances étant satisfaisantes, nous avons décidé de réaliser une petite interface basique qui permet plus ou moins à tout à chacun de « jouer » avec le programme pour retoucher et améliorer ses propres photos. Nous avons pour ce faire utilisé la bibliothèque graphique *WinLib* développé par le CERTIS, laboratoire de l'ENPC.

Voici les principales fonctionnalités de notre programme :

- Possibilité de charger une nouvelle image une fois le programme lancé, ou encore de sauvegarder l'image restaurée.
- Possibilité de définir dans le programme la zone à réparer grâce à un outil de peinture (avec possibilité par exemple de changer la taille du pinceau).
- Possibilité de définir la source du dictionnaire manuellement ou automatiquement.
- Une fois le programme lancé, il est possible de modifier tous les paramètres (largeur des patchs, épaisseur des voisinages, épaisseur de la source du dictionnaire et sens de parcours pour le remplissage).
- Une fenêtre avec différents onglets permet de jongler rapidement entre l'image originale et l'image reconstruite, ou encore de voir facilement quelle est la source utilisée pour le dictionnaire.
- Une animation basique permet de voir dans quel ordre précis les pavés ont été considérés, ou encore de voir quel patch du dictionnaire a été utilisé pour remplacer chaque pavé de la zone abîmée. Ceci est surtout utile pour les phases de débogage...)

Conclusion

Nous avons vu dans ce rapport que cette méthode permet d'obtenir des résultats parfois très spectaculaires. Pourtant, l'algorithme semble à première vue très basique et nous retenons par là qu'il ne faut pas toujours chercher à utiliser les théories mathématiques les plus complexes pour obtenir les meilleures résultats !

Par contre, nous avons aussi pu voir l'importance qu'ont les différents paramètres, la reconstruction pouvant être très médiocre s'ils ne sont pas choisis intelligemment. Ceci est bien évidemment un gros inconvénient quant à l'utilisation de cet algorithme par le grand public, même si l'on sait que la méthode est présente dans la dernière version de *Photoshop*. Ainsi, une piste pour l'améliorer pourrait être l'analyse de la texture environnant la zone abîmée, afin notamment de trouver ses dimensions caractéristiques, ce qui permettrait de dimensionner au mieux les paramètres de l'algorithme.

Enfin, nous sommes satisfaits d'avoir pu réaliser un programme qui pourra nous être utile dans la vie de tous les jours pour nos propres photos, ce n'est finalement pas si souvent le cas :-)