G'MIC 1.6.8 : c'est déjà Noël pour les traiteurs d'images !

Posté par **David Tschumperlé** (**site web personnel**) le 11/12/15 à 09:54. Édité par **6 contributeurs**. Modéré par **Florent Zara**. **Licence CC By-SA.**

Étiquettes : **gimp** , **krita** , **tricot** , **g'mic** , **traitement_d'images** **+ Étiqueter**

La version **1.6.8** *« X-Mas 2015 Edition »* de ***G'MIC*** (*GREYC's Magic for Image Computing*), infrastructure libre pour le traitement d'images, a été publiée lundi 7 décembre 2015. C'est l'occasion de vous présenter les avancées et les nouveautés introduites dans ce logiciel depuis **la dernière dépêche *LinuxFr.org*** sur ce sujet, rédigée pour la sortie de la version 1.6.2.0, il y a de cela huit mois environ. Sept versions se sont succédées depuis.

La deuxième partie de la dépêche détaille les quelques nouveautés introduites dans le **greffon *G'MIC*** pour **GIMP**, qui reste l'interface de *G'MIC* la plus utilisée aujourd'hui. Mais elle présente aussi quelques évolutions majeures plus techniques du *framework*, qui ont déjà permis d'élaborer de nouveaux effets intéressants, et qui promettent surtout de belles choses pour l'avenir.

Sommaire

1. Le projet *G'MIC*

Fig .1.1. Mascotte et logo du projet G'MIC, framework libre pour le traitement d'images.

G'MIC est un projet libre ayant vu le jour en août 2008, dans l'équipe ***IMAGE*** du laboratoire ***GREYC*** (Unité Mixte de Recherche du ***CNRS*** située à Caen). Cette équipe est composée essentiellement d'enseignants-chercheurs et de chercheurs travaillant dans le domaine du traitement d'images (ça c'est une sacrée coïncidence !). Le projet est distribué sous licence libre ***CeCILL***.
G'MIC fournit plusieurs interfaces utilisateur pour la manipulation de données images *génériques*, à savoir des images ou des séquences d'**images hyperspectrales** [W] 2D ou 3D à valeurs flottantes, ce qui inclut bien évidemment les images couleur classiques. Son interface la plus utilisée aujourd'hui est **un greffon** disponible pour le logiciel ***GIMP*** (qu'on peut qualifier de relativement « populaire », il comptabilise déjà plus de deux millions de téléchargements à ce jour).

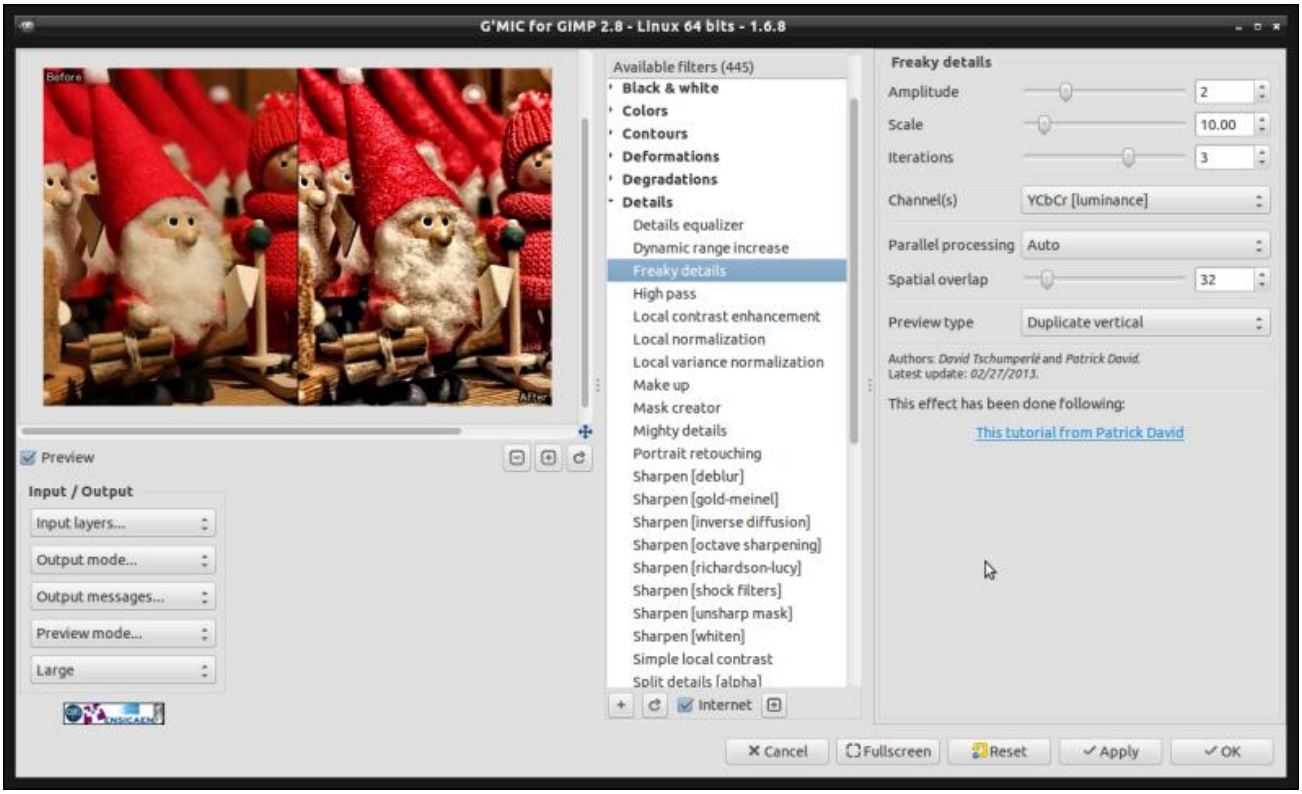

Fig. 1.2. Aperçu du greffon G'MIC pour GIMP.

G'MIC propose également une interface en **ligne de commande**, semblable aux outils *CLI* proposés par ***ImageMagick*** ou ***GraphicsMagick***, qui permet de bien profiter de toutes ses capacités. Il existe aussi un service web ***G'MIC Online*** pour appliquer des effets sur vos images directement à partir d'un navigateur (qui a été réactualisé d'ailleurs). D'autres interfaces basées sur *G'MIC* sont développées (***ZArt***, un greffon pour ***Krita***, des filtres pour ***Photoflow***…) mais celles-ci restent pour le moment plus confidentielles.
Toutes ces interfaces se basent sur la bibliothèque *C++* ***libgmic*** qui est portable, à multiples fils d'exécution — ***multi-thread*** [W] — et ***thread-safe*** [W], via notamment l'utilisation d'***OpenMP***. La bibliothèque *libgmic* est elle-même basée sur ***CImg***, une bibliothèque *C++* de traitement d'images générique plus « bas niveau », qui est développée dans la même équipe et qui existe depuis 1999 (et qui sort de façon synchronisée également en version 1.6.8). La bibliothèque *libgmic* implémente toutes les fonctions de calcul sur les images, et embarque son propre langage de script permettant aux utilisateurs avancés d'y ajouter leurs fonctions personnalisées de traitement d'images.

Aujourd'hui, *G'MIC* interprète plus de **900 commandes** de traitement différentes, toutes paramétrables, pour une bibliothèque d'environ 5,5 Mio correspondant à un peu plus de 100 000 lignes de code source. Ces commandes couvrent un large spectre du traitement d'images, en proposant des algorithmes pour la manipulation géométrique, les changements colorimétriques, le filtrage d'images (débruitage, rehaussement de détails par méthodes spectrales, variationnelles, non locales…), l'estimation de mouvement ou le recalage, l'affichage de primitives (y compris des objets 3D maillés), la détection de contours ou la segmentation, le rendu artistique, etc. C'est donc un outil très générique aux usages variés, très utile d'une part pour convertir, visualiser et explorer des données images, et d'autre part pour construire des *pipelines* personnalisés et élaborés de traitements d'images.

Pour en apprendre un peu plus sur les motivations et les buts du projet *G'MIC*, on ne peut que conseiller d'aller feuilleter **le diaporama de présentation du projet** qui ont été mis à jour récemment et qui contiennent de nombreux exemples de choses qu'il est possible de réaliser avec cette boîte à outils.

Les contributeurs principaux sont également présents pour discuter via le **nouveau forum** ou sur le canal IRC associé `#pixls.us` sur **Freenode**.

2. Améliorations propres au greffon *G'MIC* pour GIMP

Trois améliorations notables ont été apportées à l'interface du greffon *G'MIC* pour GIMP. Nous allons les détailler dans cette section.

2.1. Importation et exportation via les tampons *GEGL*

Le greffon est maintenant capable d'importer et exporter les données images de et vers *GIMP* en passant par les tampons ***GEGL*** qui forment la base de la version de développement 2.9 de *GIMP*. En pratique, cela signifie que le greffon peut travailler avec des images à grande profondeur de bits (par exemple avec des pixels stockés sous forme de flottants 32 bits par canal, à ne pas confondre avec des images au contenu pornographique !), et ceci sans aucune perte de précision.

En réalité, l'application d'un filtre seul sur une image va rarement avoir des conséquences visuelles graves dues au seul fait de la quantification de l'image traitée en 8 bits par canal. Mais tous ceux qui appliquent de nombreux filtres et effets, les uns à la suite des autres sur une même image, apprécieront que la précision numérique maximale soit conservée lors de leur flux de travail (*workflow*).

L'animation ci-dessous (*Fig. 2.1*) illustre le phénomène de quantification subtil qui apparaît lorsque l'on applique certains types de filtres sur une image (ici un filtre de lissage anisotrope).

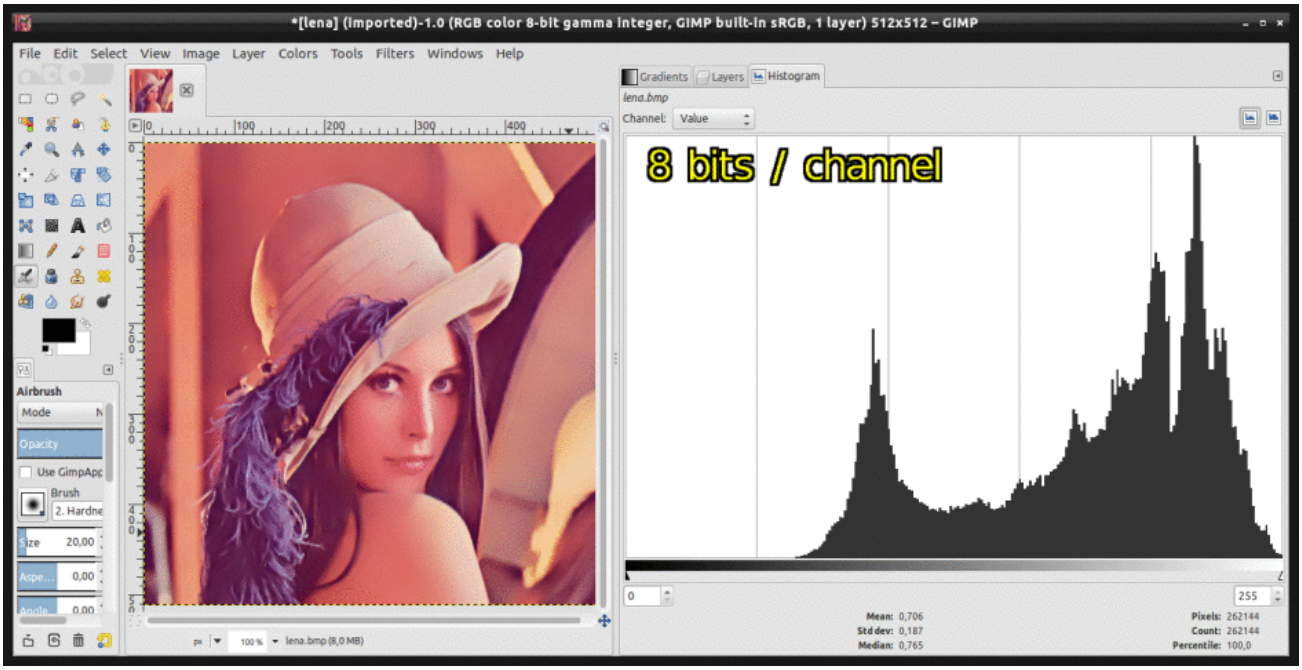

Fig. 2.1. Comparaison de l'application d'un même filtre G'MIC sur une image en 8 bits et en 32 bits par canal.

Sur la droite de la figure 2.1, on aperçoit l'histogramme des luminances de l'image modifiée, qui contient une plus grande diversité de valeurs lorsque le filtre travaille sur une image stockée avec des flottants 32 bits plutôt qu'avec des entiers 8 bits. L'enchaînement de ce genre de traitements peut assez vite créer des effets indésirables de **quantification ou postérisation** [W] sur les images traitées (*banding effect*). En bref, le greffon *G'MIC* est déjà prêt pour la sortie de la prochaine version stable 2.10 de GIMP.

2.2. Prise en charge de l'UTF-8 dans les *widgets*

Le greffon gère maintenant les chaînes **UTF-8** [W] pour l'ensemble des *widgets* de l'interface, ce qui permet une meilleure internationalisation des filtres.
Par exemple, nous disposons maintenant d'une version japonaise de l'interface et de certains filtres, comme l'illustre la copie d'écran du greffon ci-dessous (*Fig. 2.2*).

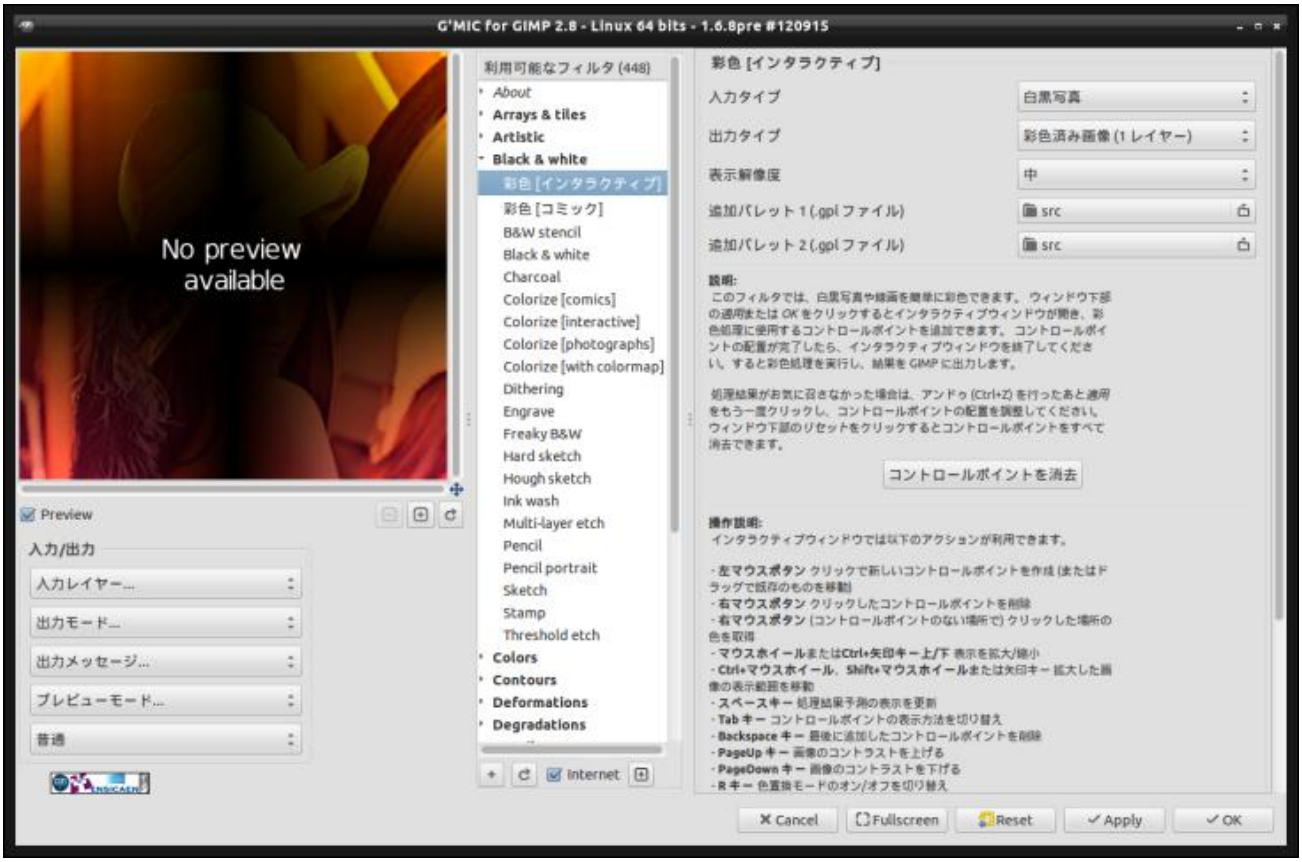

Fig. 2.2. Le greffon G'MIC pour GIMP partiellement traduit en japonais.

2.3. Une interface plus réactive

Le greffon réagit mieux aux changements de paramètres d'un filtre lorsque celui-ci est en train de calculer le résultat d'une prévisualisation. Un mécanisme d'annulation du calcul en cours a été ajouté pour ne plus attendre qu'un filtre ait fini de générer sa prévisualisation avant de pouvoir agir sur l'interface. C'est tout bête, mais ça améliore énormément l'expérience utilisateur.

Notons également l'initiative intéressante de Jean-Philippe Fleury, un aimable contributeur, qui propose sur **sa page web** une galerie recensant la quasi-totalité des effets disponibles dans le greffon. Voilà un très bon moyen d'avoir un aperçu rapide des filtres, et de pouvoir observer le résultat de chaque filtre sur des images différentes, parfois avec des paramètres différents.

3. Ajout de l'algorithme *Patchmatch*

PatchMatch [W] est un algorithme de **mise en correspondance de blocs d'images** [W] qui rencontre, depuis quelques années, un certain succès dans la communauté du traitement d'images, notamment grâce à sa rapidité d'exécution et au fait qu'il se parallélise relativement bien.
C'est devenu une technique de base utilisée dans de nombreux algorithmes récents nécessitant des comparaisons rapides de patchs, en particulier des algorithmes de **reconstruction de morceaux manquants dans des images** [W], de **synthèse de textures** [W], ou encore de **super-résolution** [W].
Il est donc satisfaisant d'annoncer que *G'MIC* intègre maintenant une implémentation parallélisée de cet algorithme (via la commande native `-patchmatch`) pour la mise en correspondance locale de patchs dans des images 2D et 3D. C'est sur la base de cet algorithme que deux filtres intéressants ont été ajoutés récemment.

3.1. *Inpainting* : reconstruction d'image

Un nouveau filtre de reconstruction d'image — ***inpainting*** [W] — utilisant un algorithme multi-résolution, permet de reconstruire des morceaux d'images *manquants* ou considérés comme *invalides*. Très pratique pour virer tata Renée en maillot de bain de vos photos de vacances à Ibiza !

À noter que ce n'est pas le premier algorithme de reconstruction « basé patch » disponible dans *G'MIC* (cette **dépêche de 2014** parlait déjà d'un filtre équivalent). C'est juste un algorithme différent, qui donne donc des résultats différents. Et, dans ce domaine, avoir le choix de pouvoir générer plusieurs types de résultats n'est vraiment pas du luxe, étant donné la nature particulièrement mal posée du problème de la reconstruction d'images.

Si vous avez comme moi une machine avec tout plein de cœurs qui ne demandent qu'à chauffer, alors cette nouvelle implémentation tirant parti du *multi-threading* risque de bien vous plaire ! Voilà deux exemples de résultats (*Fig. 3.1* et *Fig. 3.2*) que l'on peut obtenir très rapidement avec ce filtre :

Fig. 3.1. Application du nouveau filtre d'inpainting pour la suppression d'un ours.

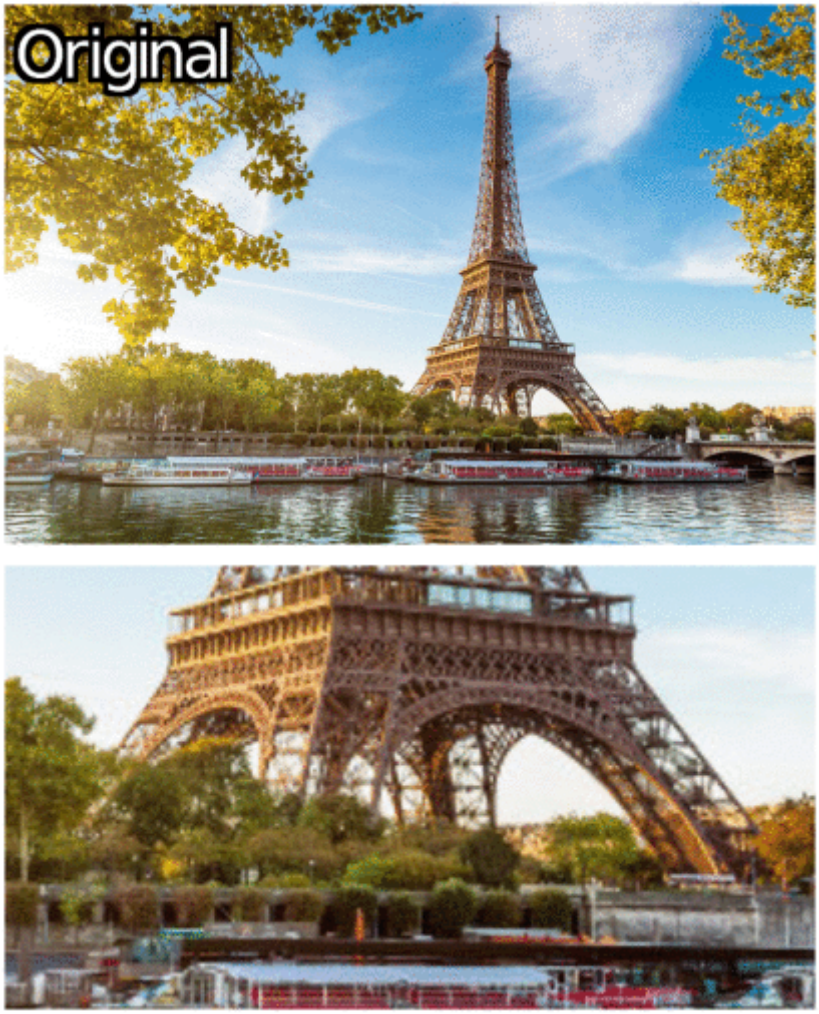

Fig. 3.2. Application du nouveau filtre d'inpainting pour la suppression de la tour Eiffel.

Une vidéo montrant comment ce dernier exemple a été réalisé (en 1 min 07 exactement) est **disponible sur YouTube** (vidéo en temps réel sans trucages, mais avec un PC à 24 cœurs_ quand même !). Ça paraît assez magique de voir que l'algorithme a été capable de reconstruire tout seul des bouts d'arbres entiers de manière assez cohérente à la place des pieds de la tour Eiffel (en effectuant en réalité un « bête » clonage d'un arbre existant ailleurs dans l'image !). Mais, je vous rassure, ça ne marche pas aussi bien avec toutes les images…

Cette technique de reconstruction multi-résolution basée sur *PatchMatch* est *grosso modo* la même que celle introduite dans ***PhotoShop CS5*** [W] en 2010, sous l'appellation (très marketing) « *Content-Aware Fill* ». La principale force de ce type d'algorithme est d'être capable de reconstruire de larges zones texturées à l'intérieur des masques définis par l'utilisateur. À noter que dans le cadre du doctorat de **Maxime Daisy** (qui a soutenu sa thèse avec succès la semaine dernière, bravo à lui !), nous avons également réalisé quelques extensions sympathiques de ce type d'algorithmes pour supprimer des objets en mouvement dans des séquences vidéos (voir la **page de démo** correspondante). Ces extensions ne sont pas encore disponibles dans *G'MIC* (et sont encore relativement coûteuses à calculer), mais cela arrivera peut-être un jour, qui sait ?

3.2. Re-synthèse de textures

En utilisant le même type de technique multi-résolution, un filtre de **synthèse de textures** [W] a également été ajouté. Il permet de synthétiser une texture de taille quelconque à partir d'une texture « modèle » donnée en entrée.

Bien sûr, il ne s'agit pas ici de faire une simple répétition en tuiles de la texture d'entrée, mais bien de régénérer une texture ayant les mêmes caractéristiques en copiant-collant des bouts de la texture modèle de telle manière que le résultat final ne contienne pas de discontinuités visibles.

La figure 3.3 illustre un exemple de synthèse d'une texture d'une taille de 512×512 pixels (image de droite) à partir d'une texture modèle plus petite, de taille de 280×512 pixels (image de gauche). Une comparaison du résultat obtenu avec l'application bête et méchante d'une simple répétition en tuiles de la texture modèle (image du milieu) est également visible.

Fig 3.3. Exemple de synthèse d'une texture complexe avec G'MIC.

Nous avions déjà un algorithme de re-synthèse de texture disponible dans *G'MIC*, mais celui-ci ne fonctionnait bien **qu'avec des *micro-textures***. L'exemple de la figure 3.3 montre que ce nouvel algorithme est quant à lui capable de régénérer des macro-textures plus complexes. On peut imaginer pas mal d'applications à ce genre de filtre, notamment dans le domaine du jeu vidéo (***rewind*** pourrait peut-être confirmer ?).

Bref, vous l'avez compris, avoir une implémentation de l'algorithme *PatchMatch* dans *G'MIC* va très certainement être bénéfique pour élaborer d'autres filtres d'image intéressants par la suite (on peut imaginer l'utiliser dans le cadre de la super-résolution ou du ***morphing***[W] à l'avenir, si le temps le permet).

4. Un évaluateur d'expressions plus performant

Je vais rentrer ici dans des considérations un peu plus techniques concernant l'évolution de *G'MIC*. En tant qu'utilisateur très régulier de *G'MIC* (pour mon travail de recherche), j'ai été confronté à un obstacle qui devenait récurrent lors de son utilisation. Mais, comme je suis aussi le développeur principal de *G'MIC*, cela m'a amené à repenser et améliorer une fonctionnalité clef du logiciel (et c'est probablement la contribution la plus significative de ces huit derniers mois, même si ce n'est pas encore très visible pour l'utilisateur). J'expose ici mon cheminement et la solution technique aboutissant à la contribution proposée.

Jusqu'à présent, *G'MIC* avait été pensé comme un moyen simple et rapide de construire des « *pipelines* d'opérateurs » de traitement d'image (pouvant intégrer éventuellement des boucles et des tests conditionnels), ces *pipelines* étant par la suite exécutés par un interpréteur. La majorité des filtres d'image présents dans *G'MIC* sont d'ailleurs élaborées comme ceci.

Or, il arrive fréquemment qu'on ait envie de « prototyper » des algorithmes plus « bas niveau », qui travaillent à l'échelle du pixel et qui ne peuvent pas s'exprimer efficacement comme une suite de macro-opérateurs (en supposant, bien sûr, qu'on ne dispose pas d'un macro-opérateur qui implémente déjà ledit algorithme !). C'est le cas par exemple pour des algorithmes qui parcourent tous les pixels *(x,y)* d'une image, et qui pour chaque pixel, effectuent une série d'opérations non triviales dépendant de la valeur d'autres pixels localisés plus ou moins loin du pixel courant (ou carrément sur une autre image) selon un schéma non ordonné par exemple (sans être complètement aléatoire non plus).

Dans ce genre de cas, généralement, le *pipeline G'MIC* correspondant devient un peu lourd à exécuter, notamment du fait de l'interprétation des boucles imbriquées en `(x,y)`. Notons que ce problème n'est pas propre à *G'MIC*. Demandez à un traiteur d'images utilisant **Matlab**[W] les trésors d'ingéniosités qu'il faut parfois déployer pour éviter d'écrire des scripts avec des boucles imbriquées explicites sur `(x,y)` pour parcourir les pixels d'une images, pour vous en convaincre. C'est un fait : en traitement d'images, on dispose très vite d'une quantité non négligeable de données à traiter. Et cela ne va pas en s'arrangeant, du fait de la progression constante des réso-

lutions des images, mais aussi du fait de la complexité croissante des algorithmes (il est courant d'avoir des algorithmes en $O(N^p)$ où $N = W \times H$ est le nombre de pixels de l'image et p supérieur à 2). Bref, faire du prototypage d'algorithmes « bas niveau » un peu lourds en traitement d'images avec des langages interprétés, ça peut vite devenir pénible en termes de temps de calcul.

La meilleure solution reste alors d'écrire ces algorithmes dans un langage compilé (au hasard, *C++*) pour pouvoir les tester sans attendre des plombes devant son écran que ça s'exécute. Mais on perd alors le confort et la rapidité de prototypage que procurent les langages interprétés. Ou alors on construit un module Matlab, Python ou autre à partir de l'algorithme compilé, et on se retrouve alors à faire du travail de liaison — ***binding*** [W] — plutôt pénible. Bref, on perd du temps pour un truc qui n'est même pas sûr de marcher au final.

Mon objectif était donc de pouvoir prototyper la plupart de mes algorithmes directement en *G'MIC*, en acceptant d'avoir une baisse de performances (comparativement à un langage compilé comme le *C++*), mais en évitant que ça devienne ridiculement lent. La solution classique, vous l'avez deviné, c'est d'inclure un mécanisme de **compilation à la volée** [W], ce que j'ai donc réalisé pour la sous-partie de l'interpréteur *G'MIC* qui s'occupe de l'évaluation des expressions mathématiques (et qui est centrale à l'infrastructure, comme on peut s'en douter).
Lorsque l'interpréteur rencontre une expression mathématique à évaluer plusieurs fois (pour chaque pixel d'une image par exemple), il la compile d'abord sous forme de code intermédiaire — ***bytecode*** [W] — qui devient ensuite très rapide à évaluer pour chaque pixel. En réalité, ce mécanisme est présent depuis des années dans *G'MIC*, mais il a été significativement amélioré ces derniers mois pour permettre l'évaluation d'expressions qui peuvent être considérées comme des petits programmes en eux-mêmes (contenant des boucles, des variables, des tests conditionnels, etc.), plutôt que comme de « bêtes » formules mathématiques.

L'exemple *jouet* qui illustre ces nouvelles possibilités est le rendu de **l'ensemble fractal de Julia** [W] qu'il est maintenant possible d'écrire en une ligne de commande (attention, ça peut piquer les yeux au premier abord) :

```
$ gmic 1024,1024,1,1,"zr = -1.2+2.4*x/w; zi = -1.2+2.4*y/h; for(iter = 0, zr^2+zi^2<=4 && iter<256, ++
```

Cet appel de *G'MIC* en ligne de commande génère cette image en résolution 1024 × 1024 pixels (l'image a été réduite ici par commodité de lecture) :

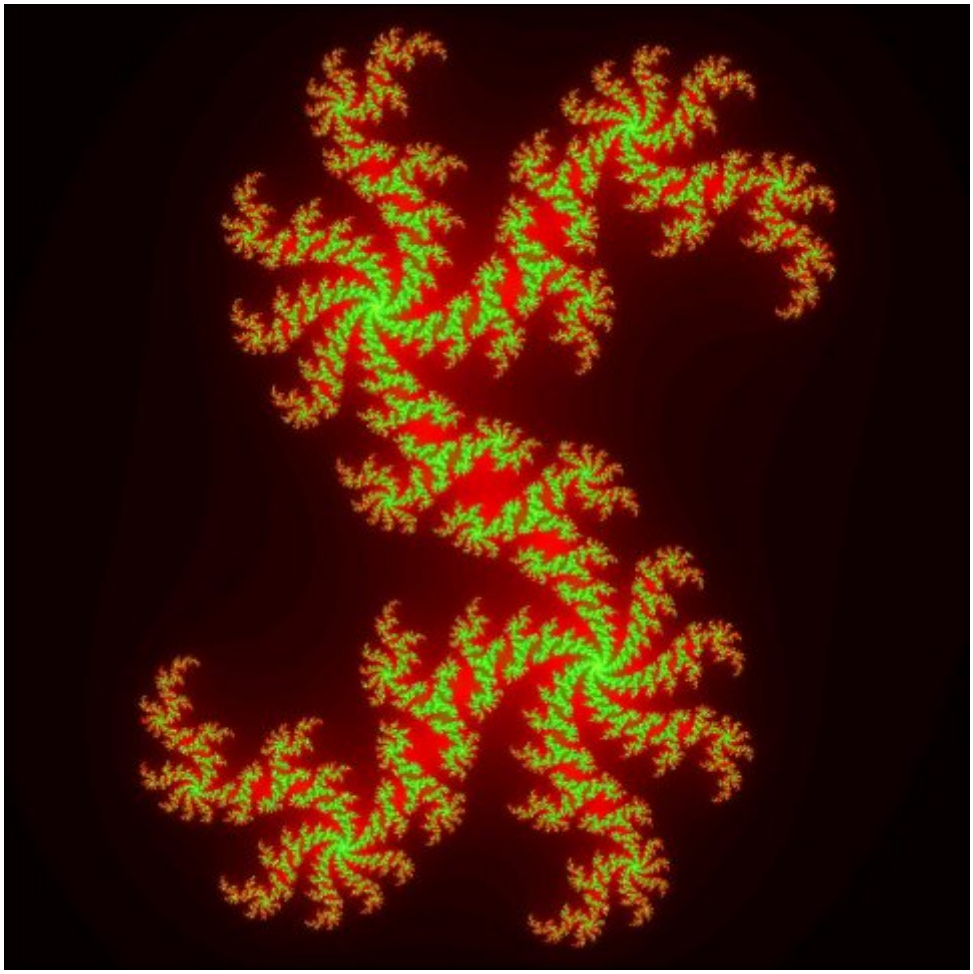

Fig. 4.1. Génération d'une fractale Julia utilisant le nouvel évaluateur d'expressions mathématiques de G'MIC.

Bien entendu, on peut aussi écrire ça plus posément, en utilisant des fichiers de commande *G'MIC* :

```
# Fichier 'julia.gmic'

julia_expr :
  -input $1,$1
  -fill "
    zr = -1.2 + 2.4*x/w;
    zi = -1.2 + 2.4*y/h;
    for (iter = 0, zr^2+zi^2<=4 && iter<256, ++iter,
      t = zr^2 - zi^2 + 0.4;
      (zi *= 2*zr) += 0.2;
      zr = t
    );
    iter"
  -map 7
```

On lance ensuite le rendu de l'image de la façon suivante :
`gmic julia.gmic -julia_expr 1024`

La complexité algorithmique de ce rendu fractal n'est pas démente, sans être négligeable non plus : pour chacun des 1024², soit 1 048 576 pixels de l'image, on va calculer au maximum 256 itérations pour évaluer la couleur d'un point (déterminé par le nombre d'itérations nécessaires pour vérifier le critère de divergence de la suite complexe calculée ici). Donc, ça fait quand même en moyenne plusieurs dizaines de millions d'itérations pour la génération de l'image complète. Là où c'est intéressant, c'est que *G'MIC* va automatiquement déterminer que l'expression mathématique associée peut s'évaluer en parallèle en divisant l'image en blocs de pixels évalués indépendamment sur chaque cœur disponible. Au final, le temps de calcul de l'image de la figure 4.1 (en résolution

1024 x 1024 pixels) se réalise en un temps plus que raisonnable : 0,176 s sur mon PC de bureau à 24 cœurs, et 0,631 s sur mon portable quadri-cœur. Évidemment, si l'on compare ce temps d'exécution avec la commande *native* équivalente `-mandelbrot` (donc compilée en *C++*) qui était déjà disponible dans *G'MIC*, on ne peut qu'être déçu : la commande *native* génère la même image en 0,055 s seulement (sur le quadri-cœur). C'est douze fois plus rapide !

Certes, mais supposons maintenant que je veuille visualiser cette fractale en colorant les points par une mesure autre que le nombre d'itération de la série complexe avant divergence. Par exemple, je souhaite visualiser la valeur de la dernière partie imaginaire calculée avant divergence. Il me suffit de modifier l'appel à *G'MIC* de la façon suivante (je vous la fais ici en version courte) :

```
$ gmic 1024,1024,1,1,"zr = -1.2+2.4*x/w; zi = -1.2+2.4*y/h; for(iter = 0, zr^2+zi^2<=4 && iter<256, ++
```

Et voilà ce que j'obtiens (toujours en moins de 0,7 s) :

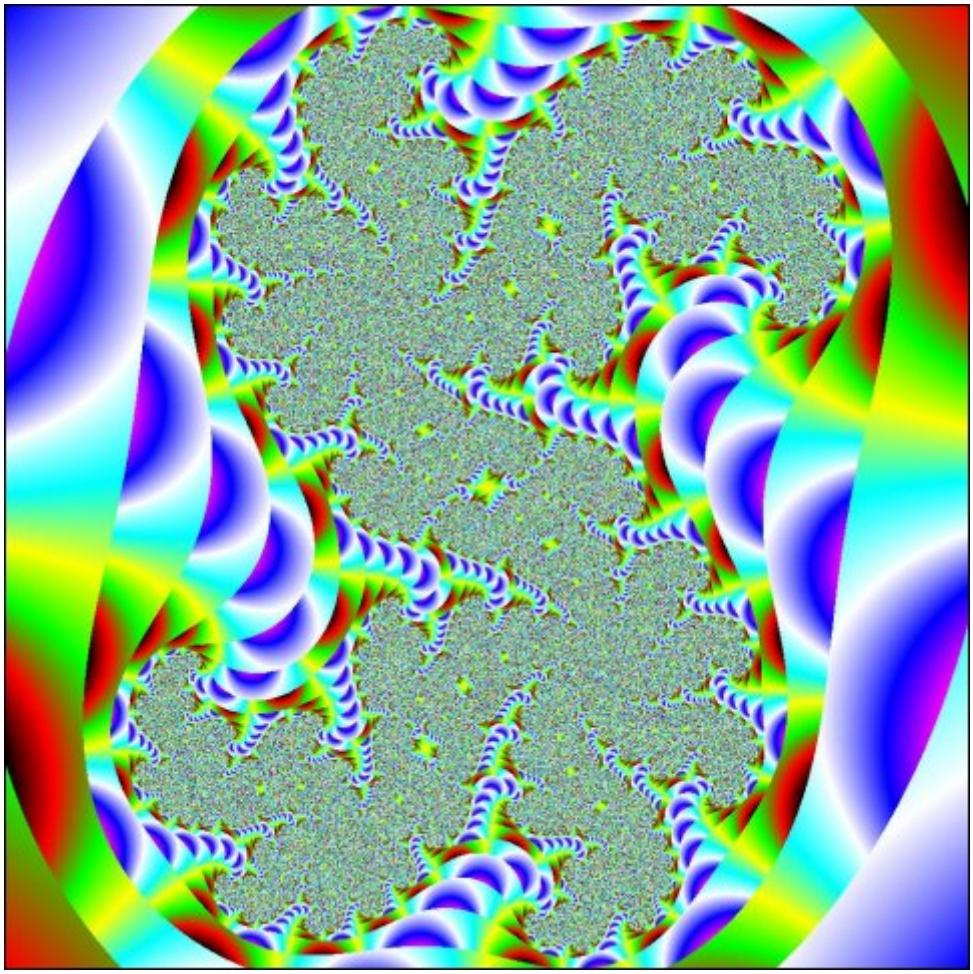

Fig. 4.2 : Autre type de visualisation lors de la génération d'une fractale Julia avec l'évaluateur d'expressions mathématiques de G'MIC.

Auparavant, si j'avais voulu réaliser la même chose avec *G'MIC*, j'aurais dû :

- Soit ajouter de nouvelles options à la commande *native* `-mandelbrot` pour permettre ce type de visualisation. Ce qui veut dire : écrire du code *C++*, compiler une version complète de *G'MIC* avec ces modifications incluses, l'empaqueter et sortir une nouvelle version. Ce ne serait pas vraiment une façon simple et rapide de faire profiter l'utilisateur de cette nouvelle possibilité de visualisation (si vous avez déjà utilisé le greffon *G'MIC* pour GIMP et son mécanisme de mise à jour des filtres par Internet, vous comprenez certainement ce que je veux dire par là).
  Sans compter qu'on ne peut décemment pas prévoir tous les types de visualisation qu'un utilisateur va vouloir générer !

- Soit écrire un script *G'MIC* qui aurait fait la même opération que la commande `julia_expr`. Mais comme l'algorithme ici est très spécifique et réalise des choses « sur mesure » au niveau pixel, cela aurait effectivement demandé l'écriture explicite de trois boucles `(x,y,iter)` imbriquées, qui aurait été interprétées, et qui auraient mis probablement plusieurs minutes à réaliser leur travail.

Nous avons maintenant un système de précompilation à la volée d'expressions mathématiques, dont l'évaluation devient relativement rapide pour chacun des pixels qui composent l'image à générer, d'autant plus que cette évaluation est réalisée en parallèle si votre machine possède plusieurs cœurs. Pour les algorithmes de traitement d'images qui nécessitent des opérations « sur mesure » (e.g. non linéaires) pixel par pixel, cette étape de précompilation rend le processus de prototypage incroyablement efficace. Ce système a été d'ailleurs abondamment utilisé pour le prototypage de deux algorithmes basés sur *PatchMatch* dont j'ai parlé en section précédente. Nul doute que beaucoup de futurs nouveaux filtres pourront bénéficier de cette nouvelle fonctionnalité présente dans *G'MIC*.
J'ai par ailleurs comparé la performance de l'évaluateur d'expressions de *G'MIC* avec la fonctionnalité identique présente dans *ImageMagick*, **via leur commande `-fx`**, qui exploite également le multi-cœur (mais qui n'utilise en revanche pas de précompilation à la volée des expressions) : *G'MIC* évalue des expressions même simples entre dix et vingt fois plus rapidement sur des images couleur de taille moyenne (3072×2048 pixels dans mes tests, à retrouver en **section 6 d'un article** publié sur le site *Open Source Graphics*).

5. *Vector Painting* : Un exemple de construction d'un filtre simple, en partant de zéro.

Un nouveau filtre d'abstraction d'image nommé *Vector Painting*, a été développé à partir de ces améliorations apportées à l'évaluateur d'expressions de *G'MIC*.
Ce n'est pas un filtre très impressionnant, mais j'en relate ici la conception, car ce filtre particulier a un fonctionnement suffisamment simple pour ne pas avoir à rentrer dans des détails techniques compliqués de traitement d'image, et ça donne une bonne idée de la façon dont un nouvel effet peut être construit rapidement avec *G'MIC* en partant de zéro, jusqu'à son intégration finale dans le greffon pour *GIMP*. Il est amusant de noter que ce filtre a été élaboré complètement par hasard, alors que je cherchais à corriger quelques erreurs dans le code de l'évaluateur d'expressions mathématiques de *G'MIC*.

Supposez que vous voulez déterminer pour chaque pixel d'une image, l'orientation discrète de la variation spatiale d'intensité lumineuse maximale du pixel (avec une précision d'angle de 45°). Pour chaque pixel centré dans un voisinage 3×3, on cherche à déterminer quel pixel du voisinage a la différence de valeur maximale avec le pixel du centre (cette différence étant mesurée en valeur absolue). Dans un premier temps, on calcule donc la luminance de l'image couleur d'entrée. Puis, on cherche à transformer chaque pixel de cette image de luminance en une étiquette (un entier entre 1 et 8) qui représente une des huit orientations possibles du plan (à 45° près). C'est typiquement le genre de problème qui nécessite l'application d'opérations au niveau pixel qui sont suffisamment « exotiques » pour ne pas disposer d'un macro-opérateur tout fait qui pourrait résoudre ce problème.

Avec le nouvel évaluateur d'expressions de *G'MIC*, la solution est étonnamment simple à mettre en œuvre :

```
# Fichier 'foo.gmic'
foo :
  -luminance
  -fill "dmax = -1;
         nmax = 0;
         for (n = 0, ++n<=8,
           p = arg(n,-1,0,1,-1,1,-1,0,1);
           q = arg(n,-1,-1,-1,0,0,1,1,1);
           d = (j(p,q,0,0,0,1)-i)^2;
           if(d>dmax,
              dmax = d; nmax = n,
              nmax)
         )"
```

En l'appliquant sur une image d'entrée nommée `leno.jpg` (à gauche sur la figure 5.1), nous obtenons l'image des orientations discrètes des variations maximales de luminance (à droite sur la figure 5.1) :

```
$ gmic foo.gmic leno.jpg -foo
```

Fig. 5.1. Calcul des orientations discrètes de variations maximales d'une image couleur.

*En aparté, laissez-moi vous raconter que j'ai récemment reçu plusieurs courriels et messages de gens qui prétendent que réutiliser l'image de Lena (bien connue des traiteurs d'images) est quelque chose de « sexiste » (quelqu'un a même utilisé le terme « pornographique ») car c'est un portrait provenant à l'origine d'un numéro du magazine **Playboy**[W] de 1972. Je vous invite à lire la page sur **l'histoire de Lena**, si vous ne savez pas pourquoi nous utilisons souvent cette image. L'occasion aussi de voir la photo originale dans son ensemble est de vous faire une idée sur le côté « pornographique » de la chose. Néanmoins, comme je ne souhaite pas blesser ces personnes à la sensibilité exacerbée, j'ai décidé d'utiliser une petite variation de l'image originale en la mixant avec un portrait de **Jay Leno**[W]. Appelons cette dernière image « Leno » plutôt que « Lena ». Et si une seule lettre peut tout arranger, alors qu'il en soit ainsi.*

L'image des orientations de la figure 5.1 peut paraître très moche (et elle l'est), mais ce n'est pas très étonnant : l'image originale contient un peu de bruit, ce qui se traduit par beaucoup de directions de variations d'intensité lumineuse incohérentes dans les régions de couleur quasi-constantes. Lissons l'image originale un petit peu avant de calculer l'image des orientations discrètes et voyons voir ce que ça donne (*Fig. 5.2*).

```
$ gmic foo.gmic leno.png --blur 1% -foo
```

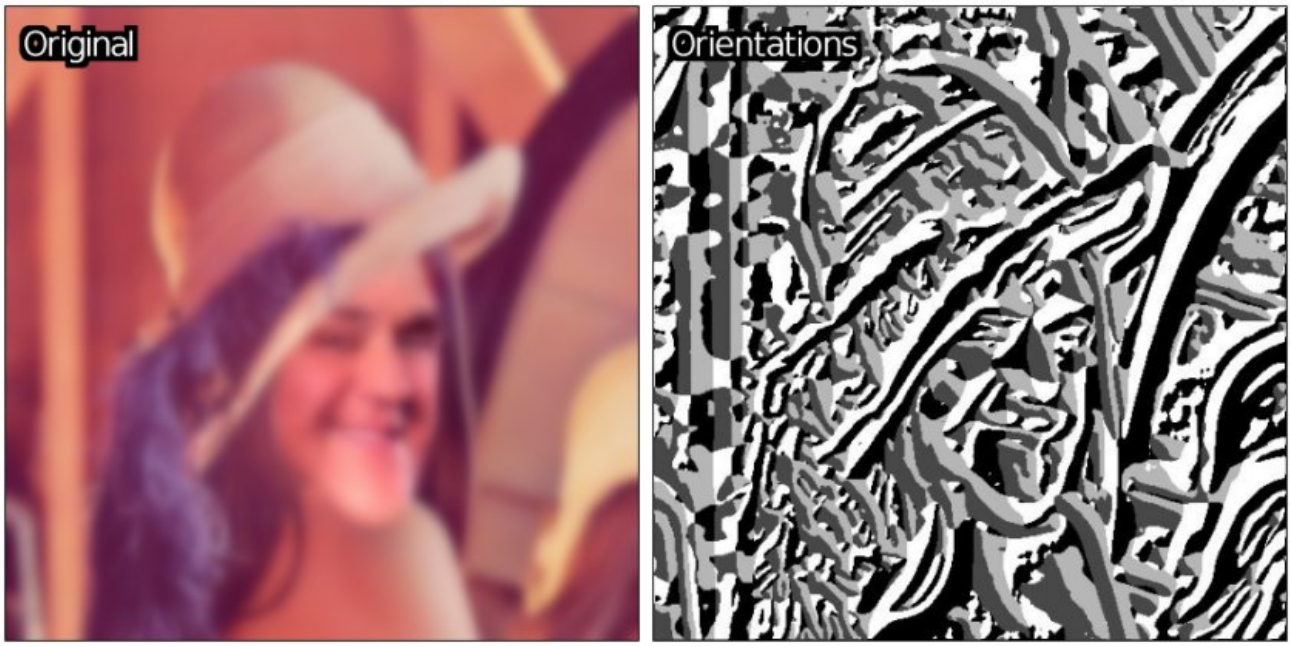

Fig 5.2. Calcul des orientations discrètes de variations de luminance sur une image lissée au préalable.

Voilà qui semble plus intéressant : le lissage permet de créer des portions larges d'étiquettes constantes, c'est-à-dire des régions où l'orientation des variations maximales de luminance est la même. Les contours du portrait original apparaissent comme des frontières naturelles dans l'image des orientations. Pourquoi alors ne pas remplacer les composantes connexes ainsi étiquetées par la couleur moyenne qu'elles recouvrent dans l'image originale ? Rien de plus facile avec *G'MIC* :

```
$ gmic user.gmic leno.png --blur 1% -foo[-1] -blend shapeaverage
```

Et nous obtenons alors une abstraction « constante par morceaux » de notre image d'entrée *Leno* (*Fig. 5.3*).

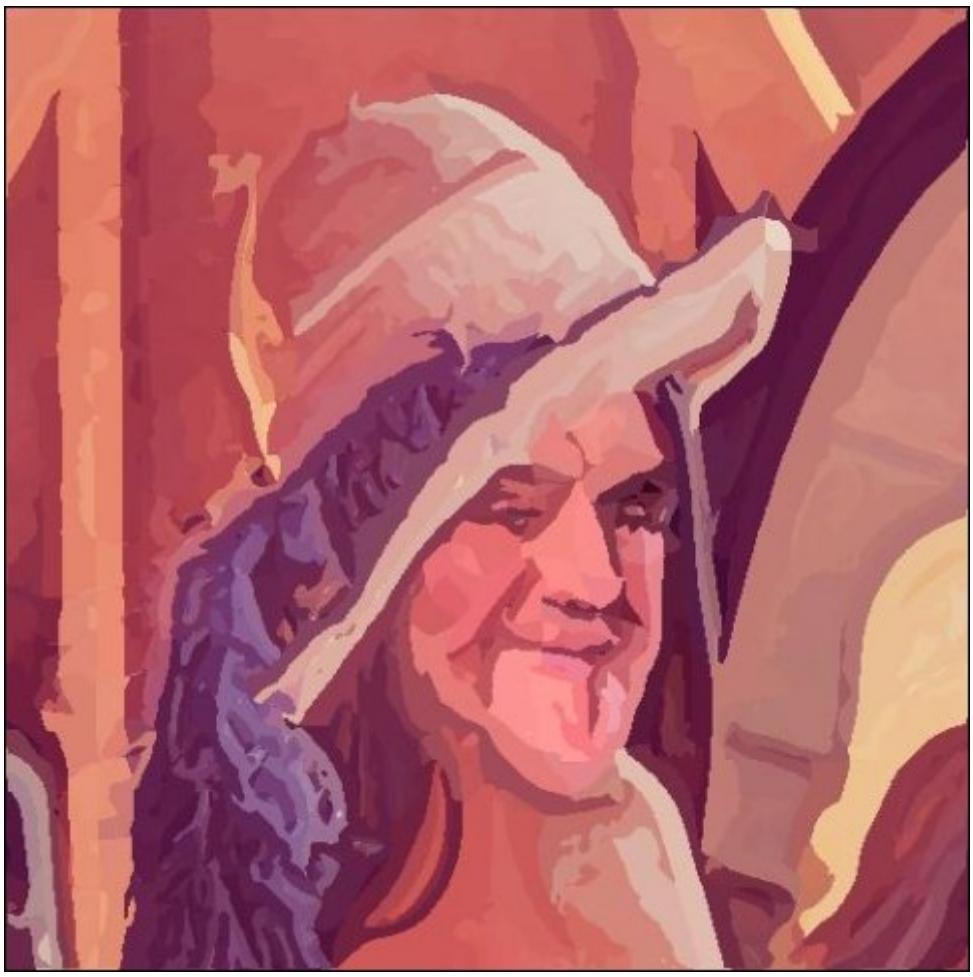

Fig. 5.3. Résultat après colorisation des zones d'orientations constantes.

Comme on peut l'imaginer, changer l'amplitude spatiale du lissage rend l'image résultante plus ou moins abstraite. À partir de là, il n'est pas bien compliqué de reprendre le code précédent, et de le transformer en filtre disponible immédiatement pour les utilisateurs du greffon *G'MIC* pour *GIMP* (le code complet de ce filtre, de seulement 19 lignes est visible ***ici***). Les utilisateurs n'ont plus qu'à appuyer sur le bouton « *Actualiser les filtres* » de l'interface du greffon afin d'immédiatement disposer de ce nouvel effet *Vector Painting*.

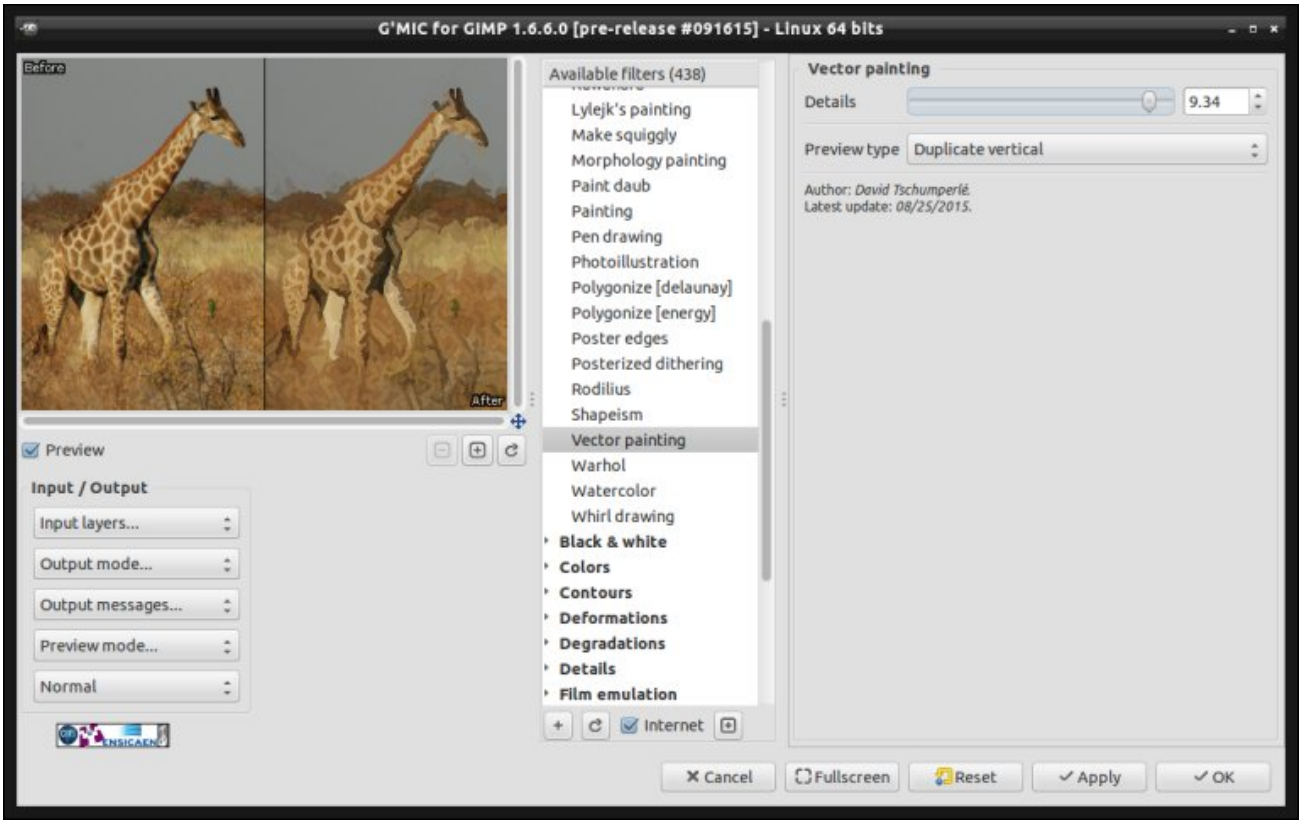

Fig. 5.3. Aperçu du filtre _Vector Painting dans le greffon G'MIC pour GIMP.

Voilà, cela vous donne une idée de la façon dont les filtres sont *prototypés* puis ajoutés dans *G'MIC*. Ce qui est intéressant, c'est que l'étape qui permet de passer d'un prototype d'algorithme à un filtre distribué et utilisable est au final très rapide à réaliser. Cela explique comment de nouveaux filtres sont ajoutés fréquemment dans *G'MIC*, et pourquoi le greffon possède aujourd'hui plus de **440 filtres** utilisables.

6. Des filtres et effets à foison !

Cette section illustre, en vrac, quelques autres fonctionnalités, effets et filtres qui ont été ajoutés depuis avril dernier. Les copies d'écran ci-dessous montrent certains filtres accessibles depuis le greffon *G'MIC* pour GIMP, mais ces filtres sont bien sûr applicables à partir de toutes les interfaces disponibles (en ligne de commande, notamment).

- **Moteur de recherche de filtres** : voilà une fonctionnalité qui nous a longtemps été demandée, et nous avons donc proposé une première ébauche d'un moteur de recherche de filtres par mots-clefs. En effet, le nombre de filtres du greffon n'allant pas en diminuant, ce n'est pas toujours facile de retrouver un filtre particulier, surtout si l'on a oublié de le mettre dans ses favoris. Bref, cet outil peut dépanner en cas de trous de mémoire (mais encore faut-il se rappeler d'un mot-clef pertinent pour la recherche).

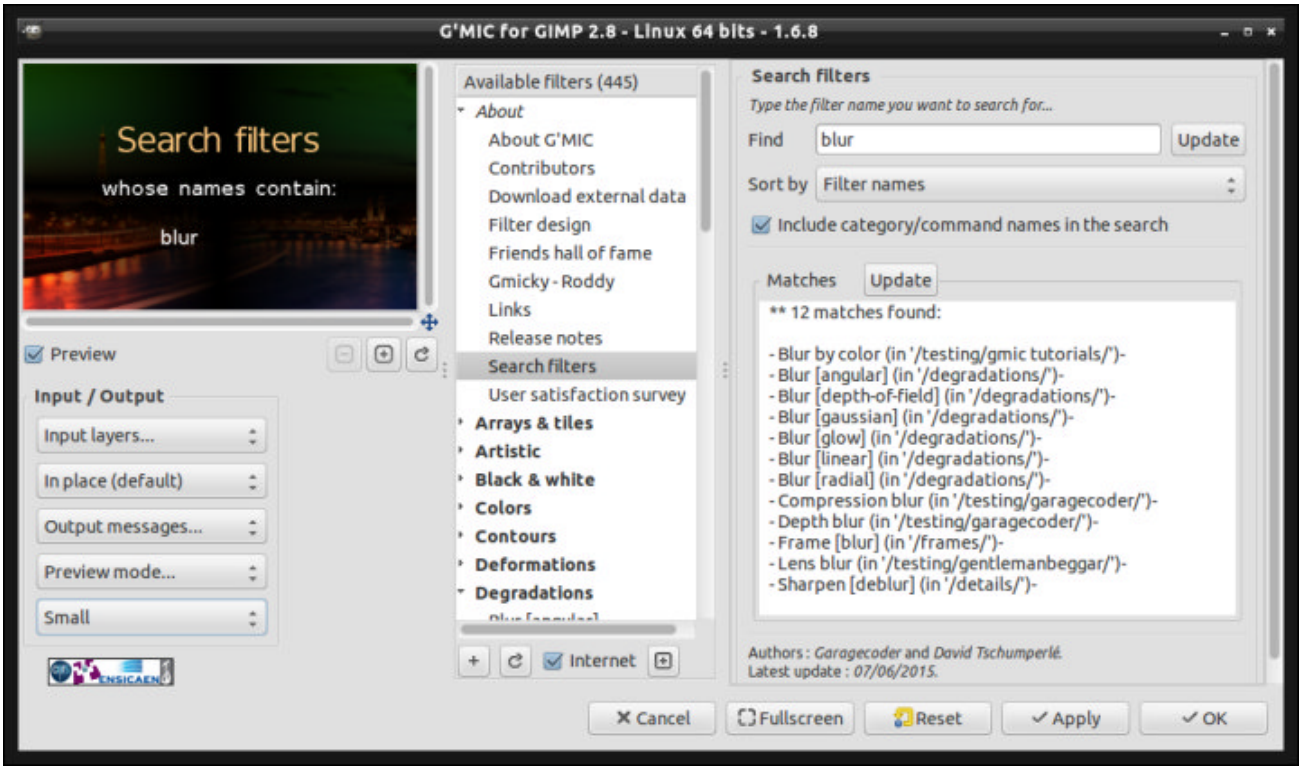

Fig.6.1. Nouveau moteur de recherche de filtres par mot clés dans le greffon GIMP.

- Filtre **Freaky B&W** : ce filtre propose de convertir une image couleur en image en noir et blanc (en niveaux de gris pour être plus précis), en résolvant **une équation de Poisson**[W] plutôt qu'en appliquant une simple formule linéaire de calcul de luminance. Le but est d'obtenir une image N&B qui contient les détails de contraste maximum présents dans chacun des canaux couleurs de l'image originale. Le filtre génère donc des images souvent très contrastées, à la façon des **images HDR**[W] (mais en niveaux de gris).

Fig. 6.2. Filtre Freaky B&W *pour la conversion d'image couleur en niveaux de gris.*

- Filtre **Bokeh** : ce filtre permet de générer des effets de type **Bokeh**[W] sur des images (flous artistiques). Il est très paramétrable et permet de générer des Bokeh avec des formes variées (cercles, pentagones,

octogones, étoiles…).

Fig. 6.3. Application du nouveau filtre Bokeh *sur une image couleur.*

- Filtre **Rain & snow** : comme son nom l'indique, ce filtre permet d'ajouter un effet de pluie ou de neige sur vos images (ici en prenant en exemple un zoom de l'image précédente).

Fig. 6.4. Ajout de pluie sur une image couleur avec le filtre Rain & snow.

- Filtre **Neon lightning** : pas grand chose à dire sur ce filtre, il génère des courbes partant d'une région *A* à une autre région *B* et stylise ces courbes comme des lumières à néon. Pratique pour faire des fonds

d'écran probablement. :)

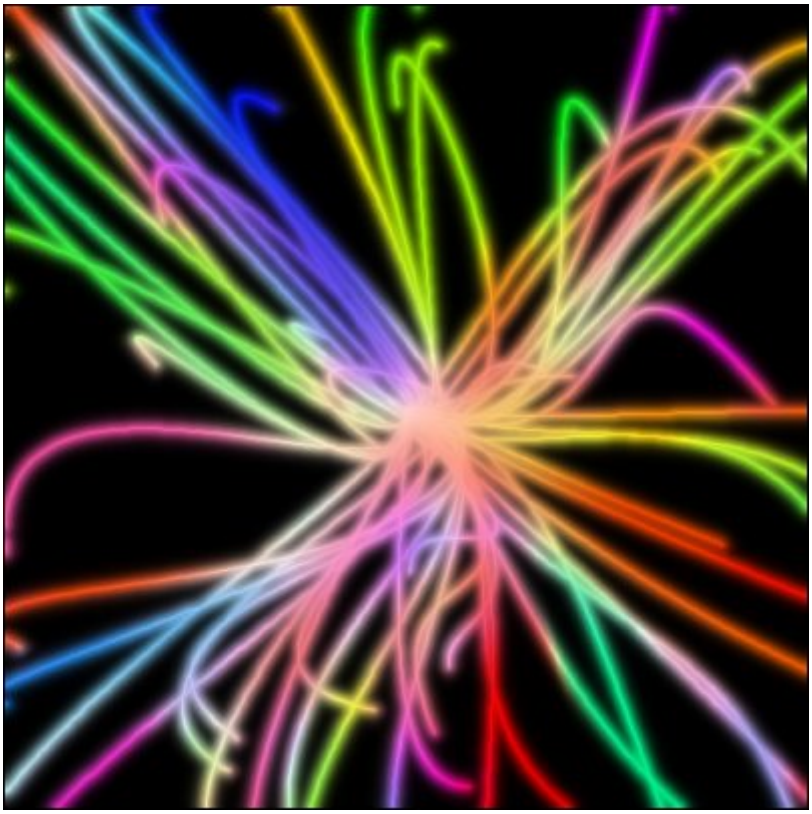

Fig. 6.5. Effet de néons courbes avec le filtre Neon Lightning.

- Filtre **Stroke** : ce filtre permet d'« habiller » des formes simples présentes sur un calque transparent, en les enrobant avec des dégradés de couleurs par exemple. La figure ci-dessous illustre la transformation d'un texte simple (monochrome) « *LinuxFr* » par ce filtre.

Fig. 6.6. Application du filtre Stroke *pour décorer un texte simple.*

- Filtre **Light leaks** : il s'agit ici de simuler des effets de lumière indésirable sur des photos. En général, c'est plutôt le genre d'effets qu'on souhaite au contraire enlever ! Mais la simulation de dégradations d'images

peut servir dans certains cas (pour ceux qui veulent rendre leur images de synthèse plus réalistes par exemple).

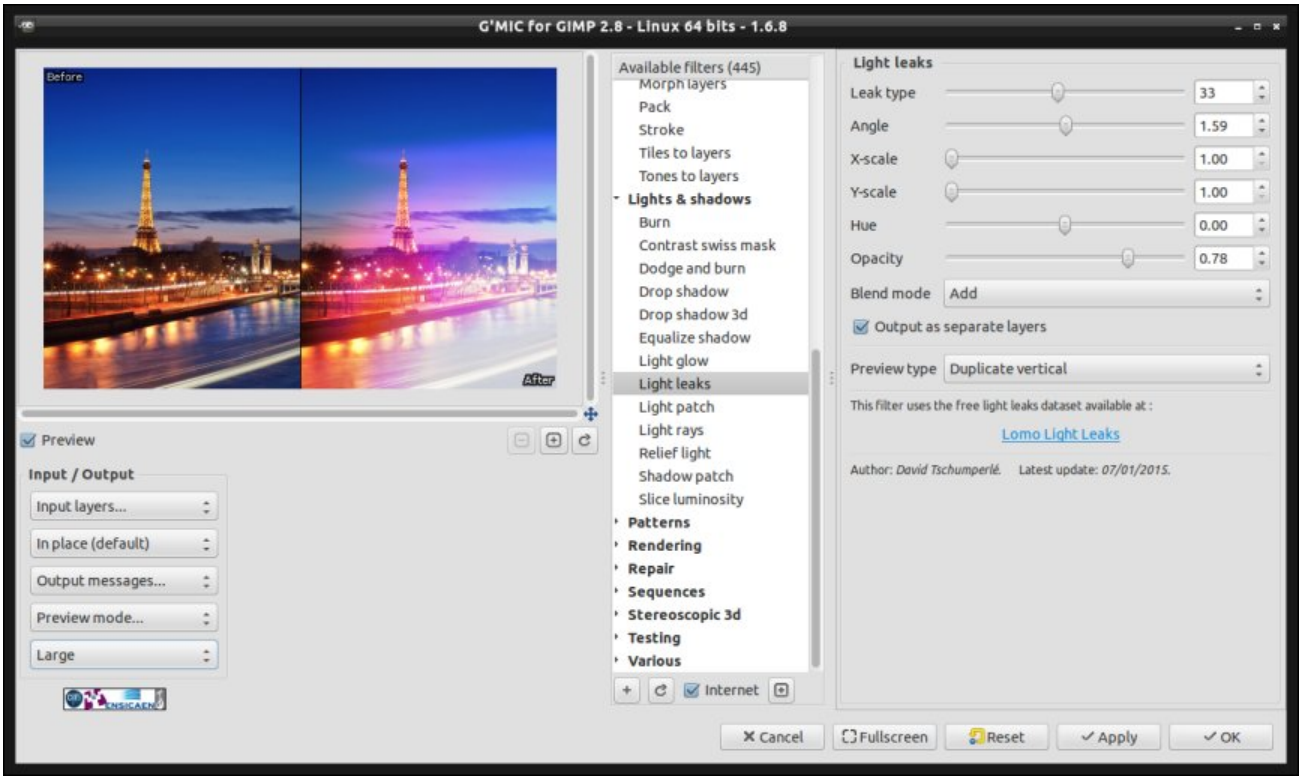

Fig. 6.7. Simulation d'effets de lumière indésirable avec le filtre Light leaks.

- Filtre **Grid [triangular]** : ce filtre transforme une image en grille composée de triangles, avec de nombreux choix de types de grilles différents.

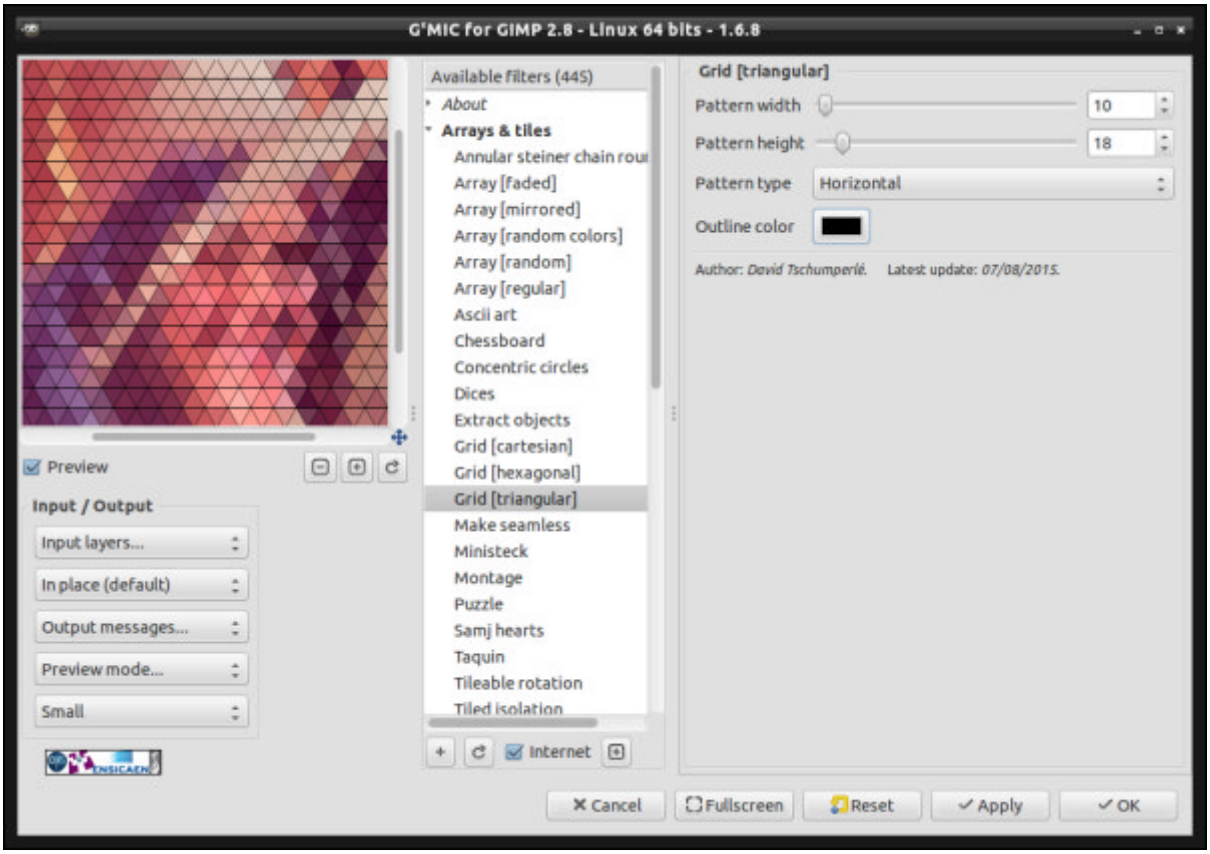

Fig. 6.8. Transformation d'une image en grille triangulaire.

- Filtre **Intarsia** : ce filtre est relativement original, puisqu'il permet de transformer une image en un schéma de construction d'un tricot de type **Intarsia** [W].
  C'est un filtre qui a été suggéré par une utilisatrice du forum ***GimpChat*** pour lui permettre de distribuer des schémas de tricot personnalisés (apparemment il existe des sites qui en proposent, mais en échange d'espèces sonnantes et trébuchantes). Le filtre en lui-même ne modifie pas l'image, mais génère un schéma de création sous forme d'une page Web (dont **un exemple est visible sur *gmic.eu***).

- Filtre **Drop water** : alors, je dois avouer que je l'aime particulièrement celui-ci. Ce filtre permet de simuler l'apparition de gouttes d'eau sur une image. Dans sa version basique, il ressemble à ça :

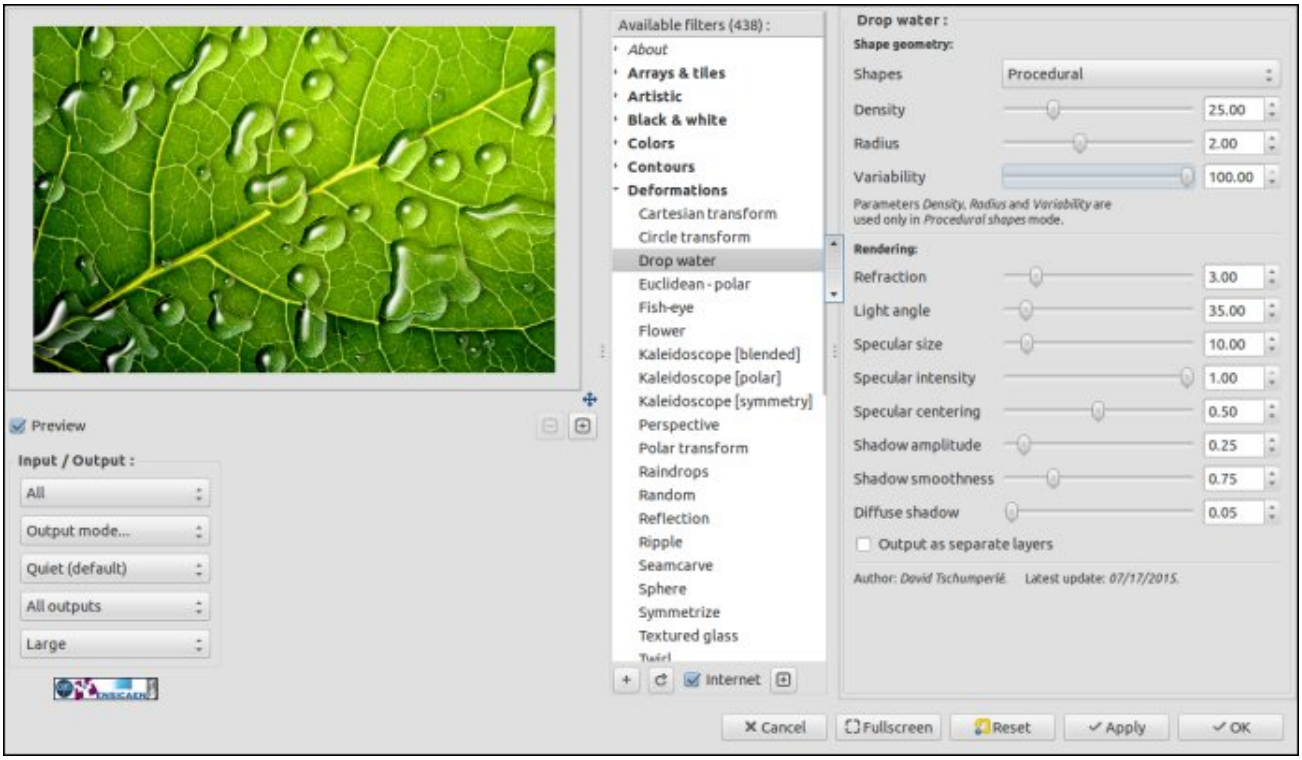

Fig. 6.9. Ajout de gouttes d'eau sur une image avec le filtre Drop water.

Mais là où ça devient vraiment intéressant, c'est que l'utilisateur peut définir ses propres formes de gouttes en ajoutant un calque transparent contenant quelques formes colorées. Par exemple, si l'on ajoute ce calque (ici, en rose) sur l'image précédente :

Fig. 6.10. Ajout d'un calque pour définir la forme des gouttes d'eau.

Alors, le filtre *Drop water* va vous générer cette image avec vos gouttes d'eau personnalisées :

Fig. 6.11. Synthèse de gouttes d'eau personnalisées avec le filtre Drop water.

Par ailleurs, le filtre a le bon goût de générer le résultat comme un empilement de plusieurs calques qui correspondent chacun au rendu des différents phénomènes physiques simulés pour la synthèse, à savoir : les tâches spéculaires, l'ombre portée et l'ombre propre, ainsi que l'effet de réfraction. On peut donc facilement manipuler tous ces calques par la suite, pour donner des effets supplémentaires à l'image générée, par exemple en appliquant un filtre monochrome sur l'image originale tout en gardant le calque de réfraction calculé sur l'image couleur originale, ce qui donne ceci :

Fig. 6.12. Filtre Drop water *suivi d'un changement colorimétrique de l'image originale uniquement.*

Une **petite vidéo tutoriel** a été réalisée pour expliquer comment réaliser cet effet pas à pas sous GIMP (ça prend pas plus de deux minutes, même pour un débutant).

Mieux encore, on peut mixer les calques générés par le filtre *Drop water* en les surimposant à d'autres images. La figure suivante montre un tel traitement à partir de deux images de portraits (qui ont été préalablement recalées). C'est très facile à réaliser, et le résultat est plutôt sympathique.

Fig. 6.13. Filtre Drop water *appliqué pour une fusion* liquide *de deux portraits distincts.*

Voilà qui termine cet aperçu non exhaustif de quelques filtres très « visuels » ajoutés dernièrement. À noter que chaque nouvel ajout fait l'objet d'une annonce sur le **flux Google+ de *G'MIC***, donc c'est assez facile de suivre l'évolution du projet au jour le jour, si ça vous intéresse de voir ce qu'on peut faire en traitement d'images libre.

7. Autres améliorations et faits notables

Ajoutons pour finir ces quelques informations en vrac, relatives au projet :

- Tout d'abord, signalons que nous sommes rentrés en contact avec **Tobias Fleischer**, un développeur professionnel de greffons pour ***Adobe After Effects*** [W] et ***Premiere Pro*** [W], entre autres logiciels de post-production vidéo. Il a déjà réalisé un gros travail de développement d'une ***DLL*** [W] encapsulant la bibliothèque *libgmic*, et a utilisé cette *DLL* pour implémenter des prototypes de greffons proposant les filtres *G'MIC* pour *After Effects*. Vous pouvez en voir un exemple sur la figure suivante (en l'occurence, un filtre de **« squelettisation » de formes** [W]) :

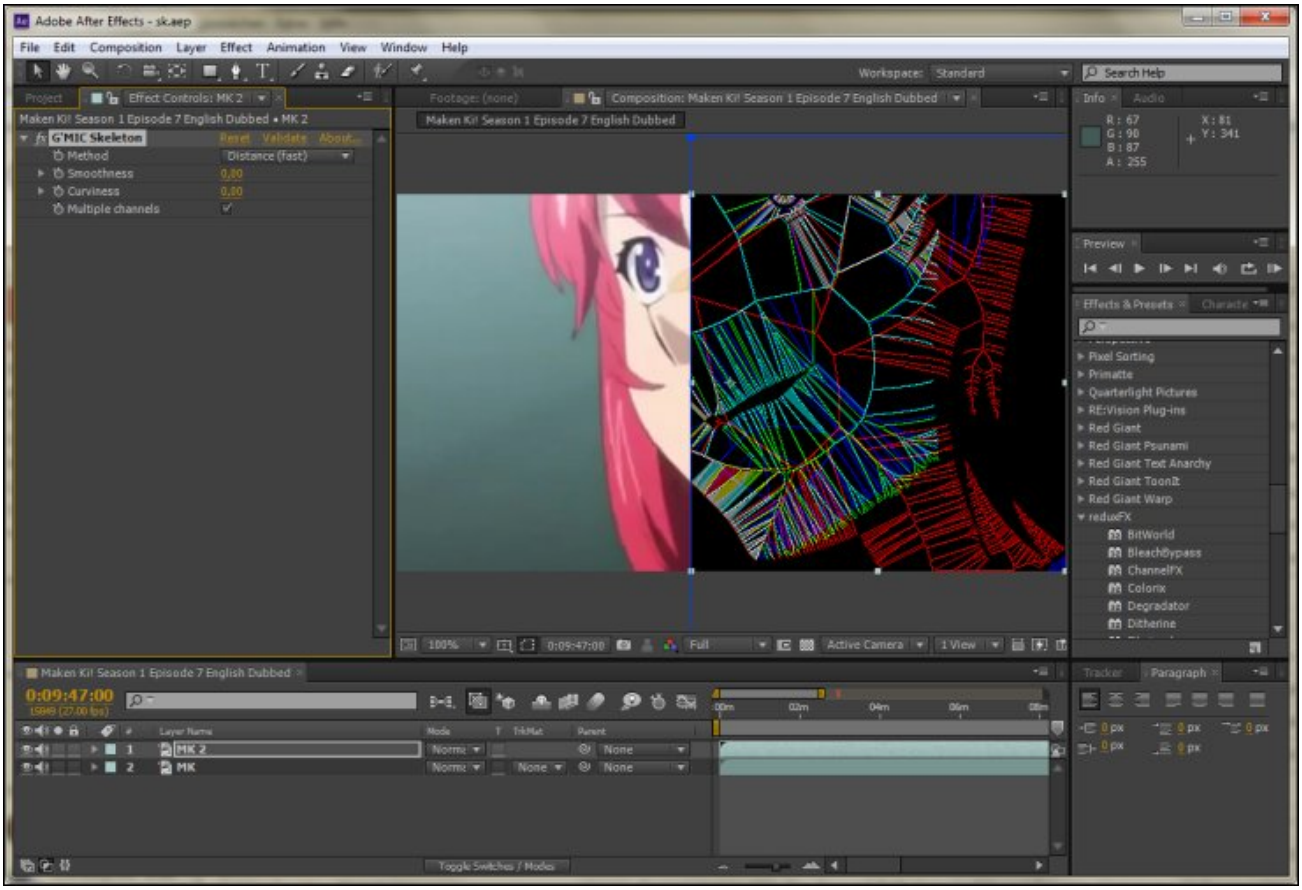

Fig. 7.1. Prototype de greffon G'MIC tournant sous Adobe After Effects.

On ne peut qu'espérer que ceci arrive très bientôt. J'entends déjà râler certaines moules : « Mais pourquoi ne pas avoir fait ça pour un logiciel libre comme ***Natron*** plutôt que pour un logiciel 100 % propriétaire ? ». Le fait est qu'il est en train de le faire, justement ! Et pas seulement pour *Natron*, mais pour tout logiciel de traitement vidéo compatible avec l'***API*** [W] normalisée ***OpenFX*** [W] (dont *Natron* fait partie). La copie d'écran ci-dessous montre par exemple un greffon *G'MIC* tournant sous le logiciel **Sony Vegas Pro** [W] en utilisant l'API OpenFX. *A priori*, ça marcherait pareil pour *Natron*.

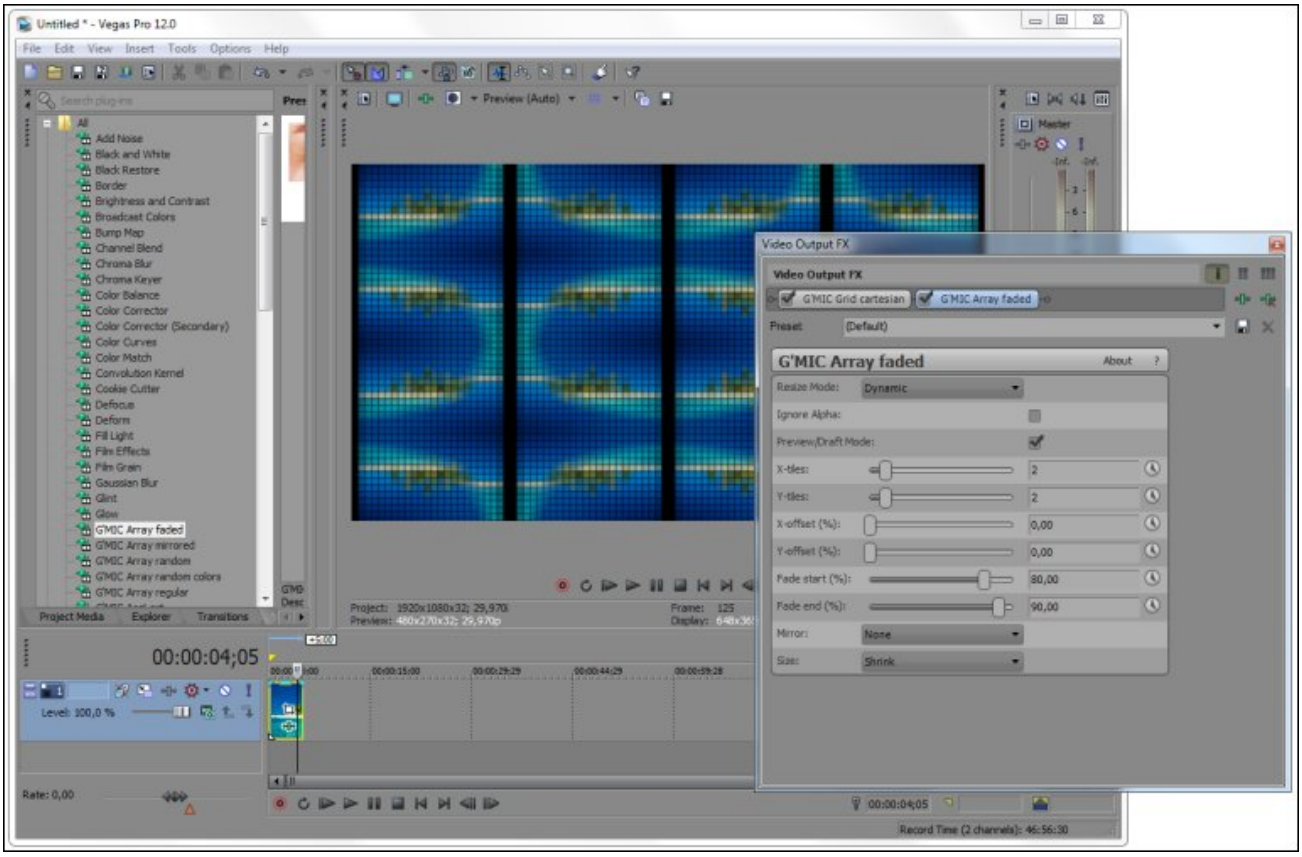

Fig. 7.2. Prototype de greffon G'MIC compatible OpenFX, tournant sous Sony Vegas Pro.

Tout ceci ne doit être encore considéré que comme du travail en cours, il reste des bogues à corriger et des améliorations à faire, mais c'est quand même prometteur.

- Continuons avec une autre bonne nouvelle : *Andrea*, un gentil contributeur (par ailleurs développeur du logiciel ***PhotoFlow***) a réussi à comprendre pourquoi *G'MIC* plantait fréquemment sous *Mac OS X* lorsqu'il effectuait ses calculs en parallèle, et a proposé un correctif permettant de résoudre ce problème (c'était un simple problème de pile allouée trop petite pour les fils d'exécution de calcul). *G'MIC* sous *Mac OS X* doit donc être pleinement fonctionnel à l'heure qu'il est.

- L'interface ***ZArt*** a également pas mal évolué, avec de nouveaux filtres ajoutés, une détection automatique des résolutions de *webcams*, ainsi que la possibilité d'avoir une double fenêtre de visualisation (une fenêtre de contrôle et une fenêtre de visualisation à mettre sur un deuxième écran pour faire des démos).

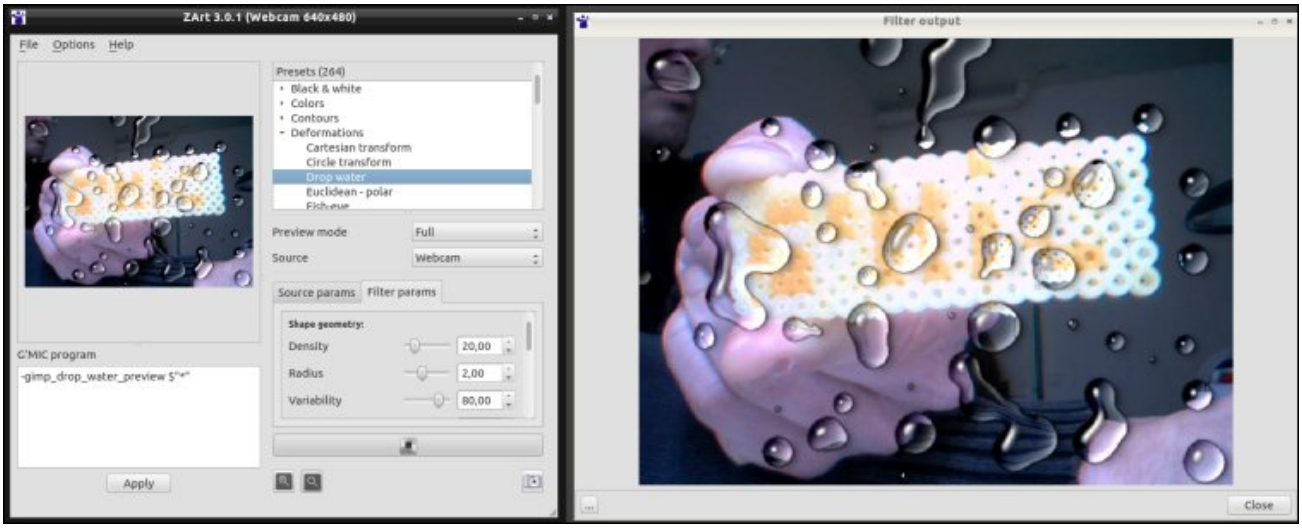

Fig 7.3. Le logiciel ZArt faisant tourner le filtre G'MIC Drop water *en double fenêtre en temps réel sur des images*

d'une webcam.

- Une nouvelle démo animée a également fait son apparition dans *G'MIC*, via la commande `-x_landscape`. Cela n'a en soi aucun intérêt concret (pour l'utilisateur) si ce n'est de tester la rapidité de l'interpréteur (et puis c'est marrant, non ?). Rappelons que l'ensemble des démos animées et interactives sont disponibles via la ligne de commande.

```
$ gmic -demo
```

Fig. 7.4. Paysage virtuel animé avec la commande « -x_landscape ».

Comme je commençais à avoir pas mal de petites animations marrantes intégrées à *G'MIC*, je les ai compilées sous forme d'une **petite intro** [W] unique, codée entièrement sous forme d'un script *G'MIC*, nommée la *BBQ intro 2016* et fleurant bon le style 8 bits/16 bits qui nous manque tant ! Vous pouvez apercevoir la **vidéo de cette intro sur YoutTube**. Ça n'a pas l'air forcément fameux, mais gardons à l'esprit que tout ça est généré à 100 % par l'interpréteur *G'MIC*, qui n'est pas forcément fait pour ça à la base. :)

8. Et ensuite ?

Cette dépêche a fait un (long) tour d'horizon des points les plus importants qui ont émergés après ces derniers mois passés à travailler sur le projet *G'MIC*. Nous n'avons pas forcément de plan bien défini des choses sur lesquelles nous voulons nous focaliser par la suite, nous ferons au gré de nos envies et de nos besoins (et des contributeurs qui se manifesteront). Dans tous les cas, il apparaît que le projet *G'MIC* est toujours bien dynamique, et peut potentiellement intéresser et toucher de plus en plus de gens dans le futur. Ceci, grâce aux divers contributeurs et aux utilisateurs qui font des retours réguliers sur le logiciel, encore un grand merci à eux pour leurs efforts ! On espère annoncer encore de belles choses sur *LinuxFr.org* autour du projet *G'MIC* dans les années à venir dans tous les cas.

Et pour finir, vive le traitement d'images libre !

Aller plus loin

- [Le projet G’MIC] (724 clics)
- [Le greffon G’MIC pour GIMP] (458 clics)
- [Initiation à G’MIC en ligne de commande] (142 clics)
- [Documentation technique de référence] (127 clics)
- [Forum de discussions autour de G’MIC] (121 clics)
- [Précédents articles LinuxFr.org sur G’MIC] (174 clics)

Une coquille ?

Posté par ǝpɐlƃuⱯ nǝıɥʇʇɐW-ǝɹɹǝıd (site web personnel) le 11/12/15 à 10:22. Évalué à 2.

> Merci pour ces nouvelles régulière sur G'Mic, et surtout pour cet outil. Il me semble avoir repéré une coquille : là où est mentionné le filtre « Bokeh » (un flou de délocalisation) l'image d'illustration semble plutôt montré un effet de « Flare » (des réflexions dans l'objectif de rayons de sources lumineuses, en l'occurence hors cadre) ?
>
> --
>
> « IRAFURORBREVISESTANIMUMREGEQUINISIPARETIMPERAT » — Odes — Horace

Re: Une coquille ?

Posté par whity le 11/12/15 à 11:22. Évalué à 2.

> Ce n’est pas un flare non plus, mais effectivement, le « bokeh » sur l’image d’exemple ne fait pas très naturel. On a plus l’impression d’un truc rajouté par-dessus l’image qu’un réel flou d’arrière plan (faut dire aussi qu’une image de paysage n’est pas le plus adapté pour illustrer le bokeh).
>
> --
>
> Mes commentaires sont en wtfpl. Une licence sur les commentaires, sérieux ? o_0

Re: Une coquille ?

Posté par David Tschumperlé (site web personnel) le 11/12/15 à 11:42. Évalué à 3.

> Oui c'est exact, en fait le filtre génére effectivement un nouveau calque qui s'ajoute à l'image.
> L'exemple ici n'est peut-être pas très bien choisi. En ne gardant que le calque généré avec des paramètres sympas, on peut quand même avoir un fond de type Bokeh (qui fait un peu synthétique mais sympa quand même).

Commentaire supprimé

Posté par Anonyme le 11/12/15 à 10:44. Évalué à 10.

> *Ce commentaire a été supprimé par l’équipe de modération.*

Merci

Posté par raum_schiff le 11/12/15 à 11:04. Évalué à 10.

> ++ pour la dépêche aussi touffue que le framework.
>
> Pour un GimpUser de base comme moi, mettre le nez dans G'MIC peut vous "manger" une semaine entière au bas mot …
> Et ce n'est pas un mal !