



Current state and 2012-2013 retrospective

David Tschumperlé, Jérôme Boulanger and Patrick David

Libre Graphics Meeting, Madrid / Leipzig, April 2014

Project Overview

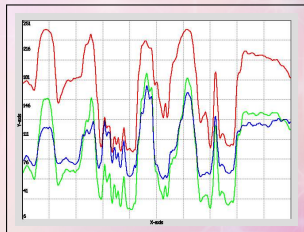
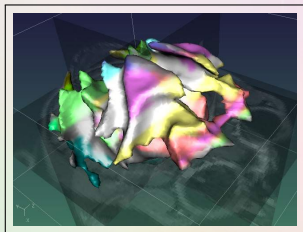
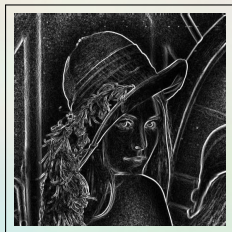


<http://gmic.sourceforge.net>

- A free software which aims at providing user interfaces to perform complex image processing operations.
- **Technical means :** G'MIC defines its own script language, specifically designed to build image processing pipelines. The G'MIC language interpreter is then embedded in all proposed user interfaces.

- **Full-featured:** More than 800 commands available for image visualization, filtering, geometry / color management, features extraction, 3d rendering, matrix computations, graphical plots, ...
- **Conciseness:** The G'MIC language has been designed specifically for being concise. This is an interpreted language, which can be extended by custom user-defined functions.

- Technical documentation (.pdf) has more than 400 pages.
- 83k lines for the whole source code (*CImg included*).

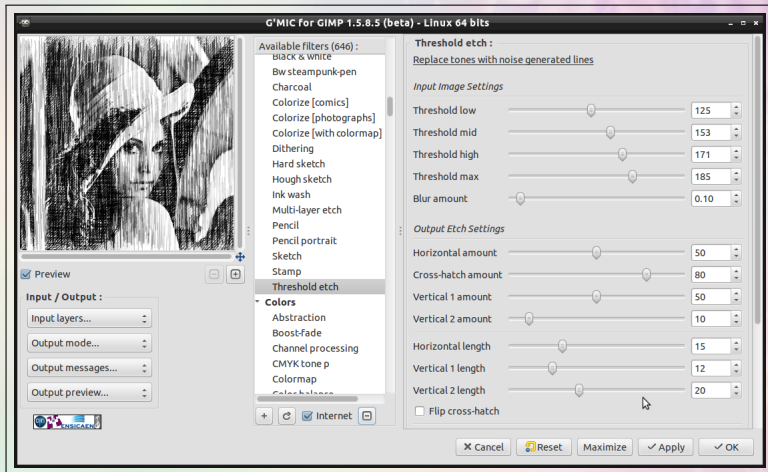


- G'MIC provides an open-source implementation of the **language interpreter** (as a C++ library).
 - ▶ **Integrations:** Third-party softwares can easily get all *G'MIC* features (interesting for image retouching or painting softwares, ...).
 - ▶ **Free software:** The G'MIC interpreter is distributed under the **CeCILL license** (GPL-compatible).
- **Examples of integrations:**
 - ★ *Krita* (plug-in), painting software, integration started in 2013.
 - ★ *EKD*, video editing software, integration started in 2010.
 - ★ *Planned: Delaboratory*, RAW photograph postprocessing application.

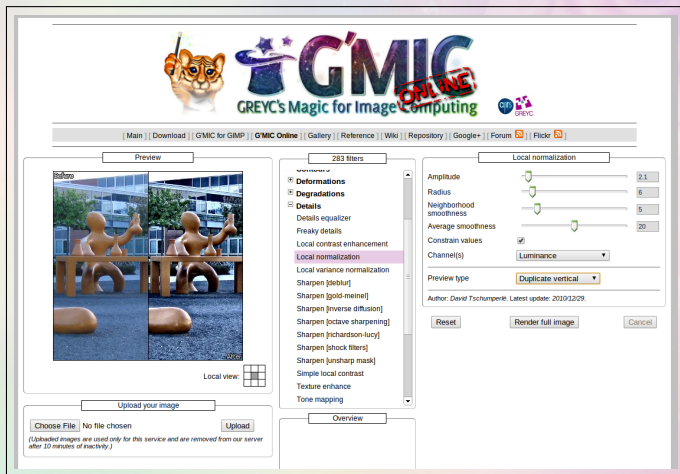
- **gmic**: Tool to manipulate the G'MIC interpreter from the command line (CLI). Competitor to the CLI tools of the ImageMagick / GraphicsMagick projects.

```
dtschump@ :"$ gmic ~/work/img/lena.bmp -blur 3 -mirror x
[gmic]-0./ Start G'MIC parser.
[gmic]-0./ Input custom commands file '/home/dtschump/work/src/resources.gmic' (added 14 commands, total 1121).
[gmic]-0./ Input custom commands file '/home/dtschump/work/src/gmic/src/gmic_def.gmic' (added 1107 commands, total 2228).
[gmic]-0./ Set dynamic 3d rendering mode to flat-shaded.
[gmic]-0./ Input file '/home/dtschump/work/img/lena.bmp' at position [0] (1 image 512x512x1x3).
[gmic]-1./ Blur image [0], with standard deviation 3 and neumann boundary.
[gmic]-1./ Mirror image [0] along the 'x'-axis.
[gmic]-1./ Display image [0] = 'lena.bmp*'.
lena.bmp* (512x512x1x3) : this = 0xbf9852e4, size = 1/16 [3072 Kb], data = (CIImg(float*))0xa027a44..0xa027a5b,
[0] : this = 0xa027a44, size = (512,512,1,3) [3072 Kb], data = (float*)0xb73ba008..0xb76ba007 (non-shared) = [ 203.2
86 207.053 210.367 212.452 212.914 211.713 209 205.179 ... 65.5009 63.9167 62.7955 61.9359 61.279 60.5754 59.9074 59.3
564 ], min = 9.43401, max = 250.19, mean = 128.229, std = 55.711, coords_min = (511,440,0,1), coords_max = (68,57,0,0)
+
[gmic]-1./ End G'MIC parser.
dtschump@ :"$ █
```

- **gmic_gimp**: Plug-in for GIMP, provides more than 600 image filters.



- **G'MIC Online:** Web service for manipulating images online (like the GIMP plug-in, with less filters and running on a web browser).
<https://gmicol.greyc.fr>

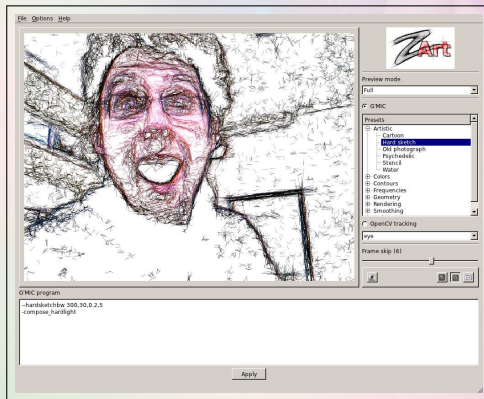


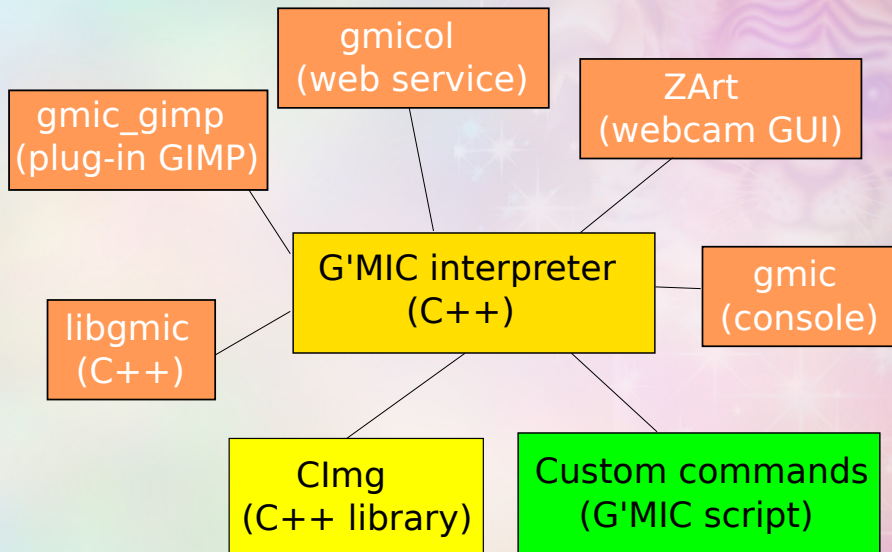
The screenshot displays the G'MIC Online web interface. At the top, there is a navigation bar with links: [Main] | [Download] | [G'MIC for GIMP] | **[G'MIC Online]** | [Gallery] | [Reference] | [Wiki] | [Repository] | [Google+] | [Forum] | [Flickr].

The main interface is divided into several sections:

- Preview:** Shows a side-by-side comparison of an image. The left image is the original, and the right image is the result of applying a filter. Below the images is a "Local view" grid.
- Filters:** A list of 283 filters is shown, with "Local normalization" selected and highlighted in pink. Other visible filters include "Deformations", "Degradations", "Details", "Details equalizer", "Freaky details", "Local contrast enhancement", "Local variance normalization", "Sharpen [seblur]", "Sharpen [gold-meinel]", "Sharpen [inverse diffusion]", "Sharpen [octave sharpening]", "Sharpen [richardson-lucy]", "Sharpen [shock filters]", "Sharpen [unsharp mask]", "Simple local contrast", "Texture enhance", and "Tone mapping".
- Local normalization:** A control panel for the selected filter with the following settings:
 - Amplitude: 2.1
 - Radius: 6
 - Neighborhood smoothness: 5
 - Average smoothness: 20
 - Constrain values:
 - Channel(s): Luminance
 - Preview type: Duplicate verticalButtons: Reset, Render full image, Cancel.
- Upload your image:** A section with a "Choose File" button, the text "No file chosen", and an "Upload" button. A note below reads: "(Uploaded images are used only for this service and are removed from our server after 10 minutes of inactivity.)"
- Overview:** A section at the bottom right, currently empty.

- **ZArt**: A QT-based interface for manipulating images acquired from the webcam (used as a demonstration platform).



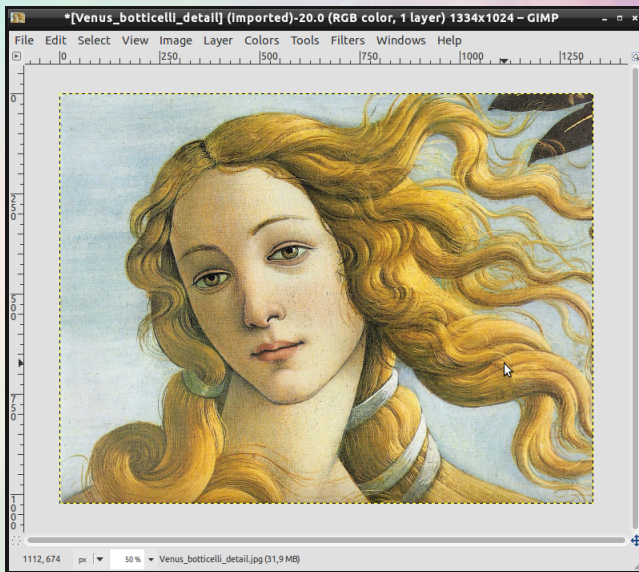


Filter Showcase:

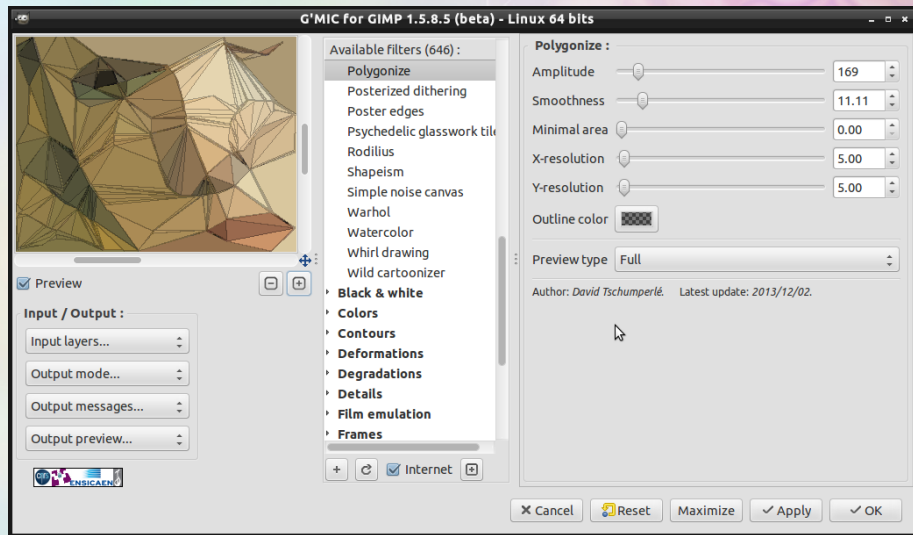
Polygonize

- **Goal:** Transform an image into a **polygonized** rendering with triangles having uniform flat colors.
- **Made by :** David, to test the stability of 3d flat objects rendering in G'MIC. Then, we realized it was a cool filter, so we kept it 😊.
- **How is this done?** Starting from an uniform grid, we move the grid points iteratively towards the nearest contour points.

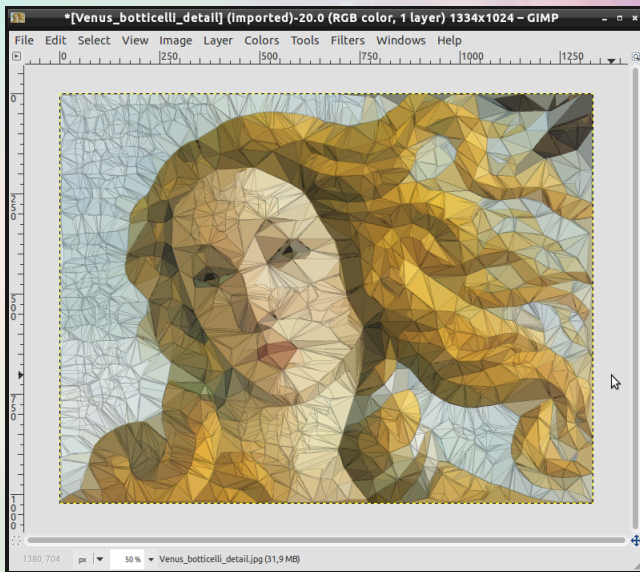
⇒ **35 lines of G'MIC code.**
(all included: GUI description + algorithm).



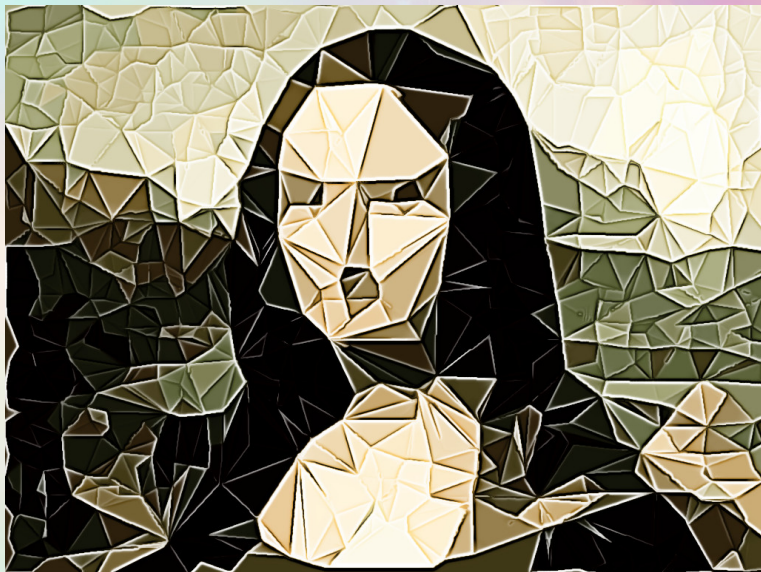
Open input image.



Invoke G'MIC plug-in and select **Artistic / Polygonize**.



Get your polygonized result.



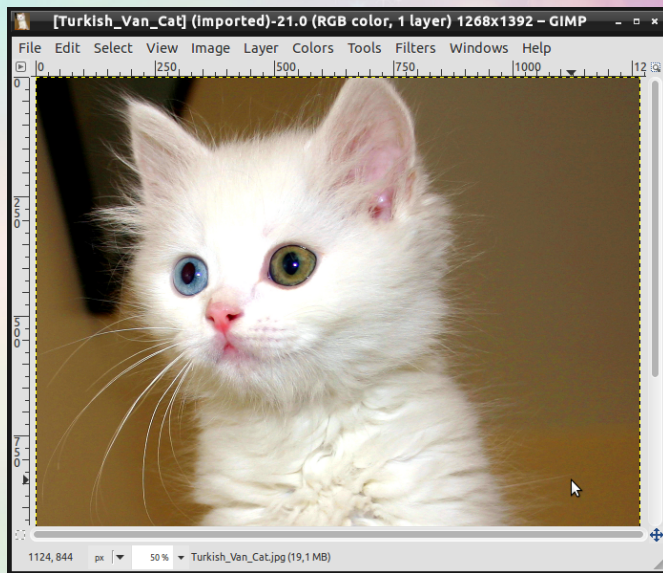
Another filter variation created by **Samj** on GimpChat (**folded paper?**).

Filter Showcase:

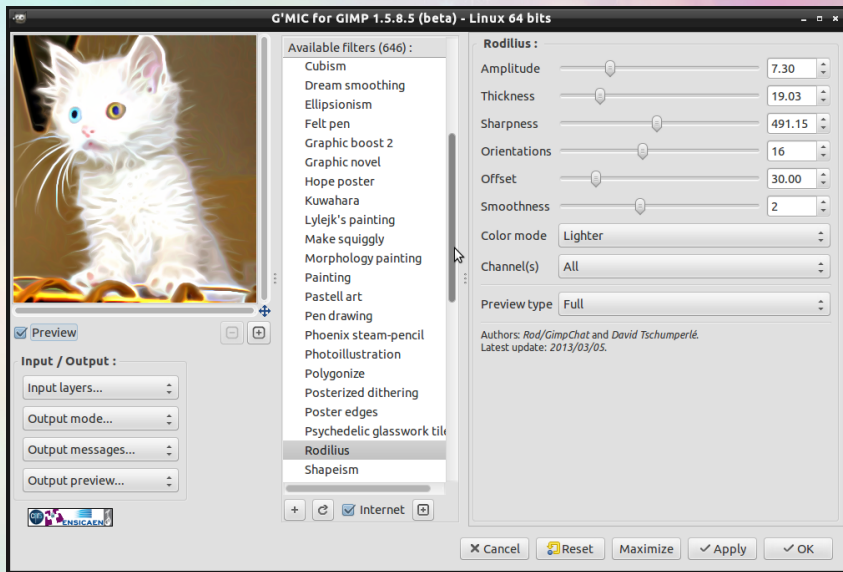
Rodilius

- **Goal:** Try to mimic the famous **Fractalius** effect from Redfield (39.90\$ plug-in for Photoshop).
- **Made for :** **Rod** on GimpChat has designed a first version of the filter as a **FilterForge** pipeline. **David** has translated it into G'MIC code.
- **How is this done?** We compute linear blurs with various orientations then mix them all using **Lighten only** or **Darken only** blending modes. Additional (aggressive) anisotropic smoothing and sharpening are added on each orientation layer.

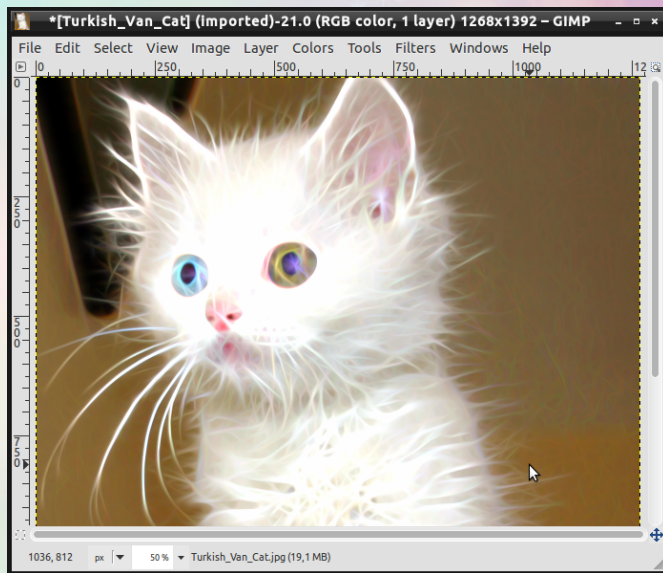
⇒ **28 lines of G'MIC code.**
(all included: GUI description + algorithm).



Open input image.



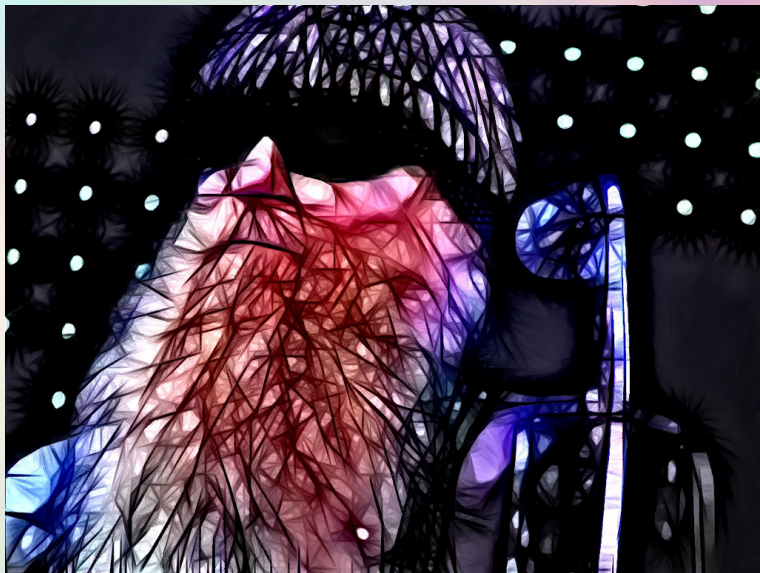
Invoke G'MIC plug-in and select **Artistic / Rodilius**.



Wait a little bit, then enjoy ! (recently **parallelized** for speeding up FFTs).



Two other examples, works quite well on fur.



Another example : with *Darken only* blending mode used.

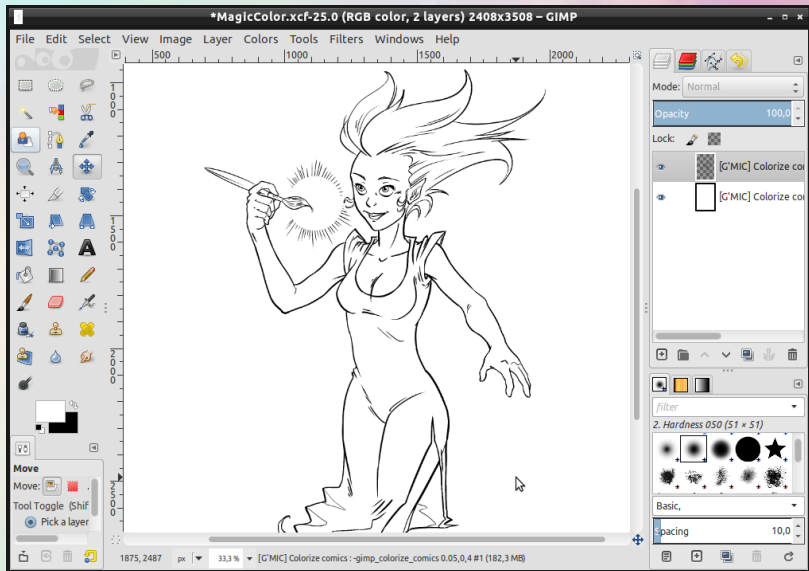
Filter Showcase:

Colorize [comics]

- **Goal:** Help coloring black and white sketches by allowing the artist to draw only small color spots inside the regions to fill-in.
- **Made by :** **David** for **David Revoy** and **Thimothé Giet**, two artists (famous Krita users), after they have seen the **Lazy Brush** plug-in for TVPaint.
- **How is this done?** Color spots are extrapolated considering edge-based priority maps, with a watershed-like algorithm.

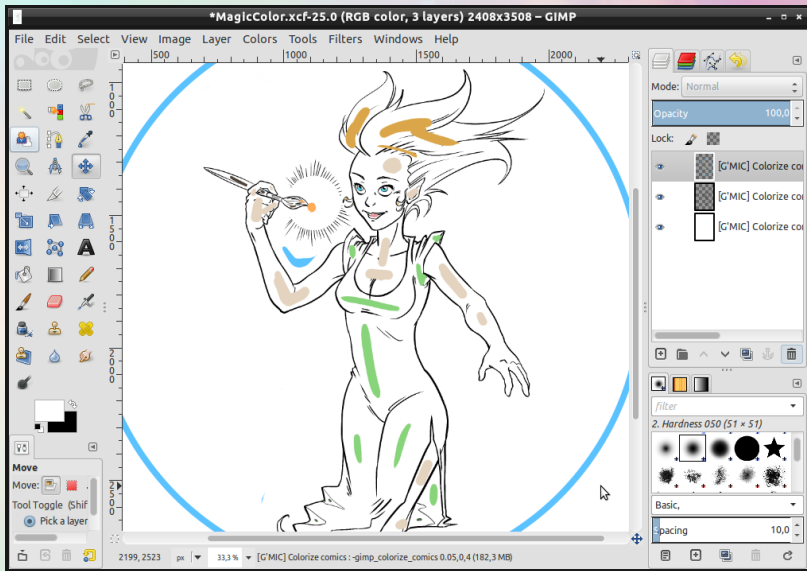
⇒ **44** lines of G'MIC code.
(all included: GUI description + algorithm).

Black & white : Colorize [comics]

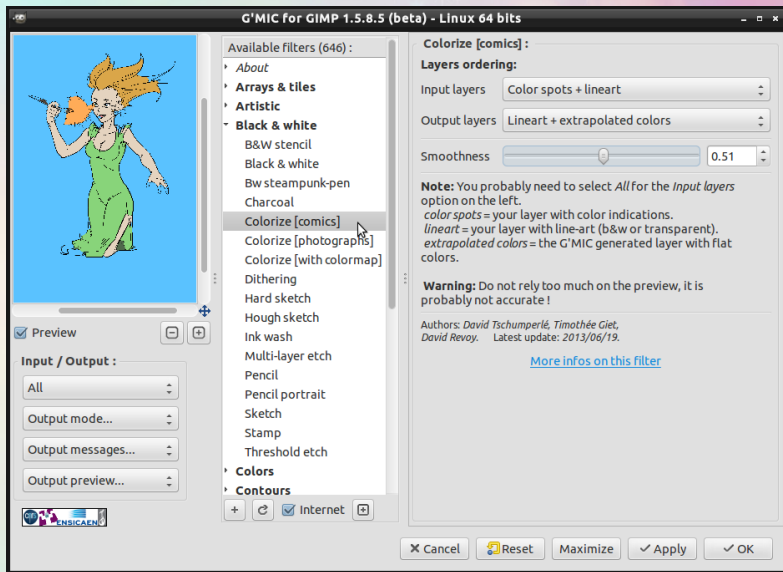


Open input image (here, two layers : dark lineart + white background).

Black & white : Colorize [comics]

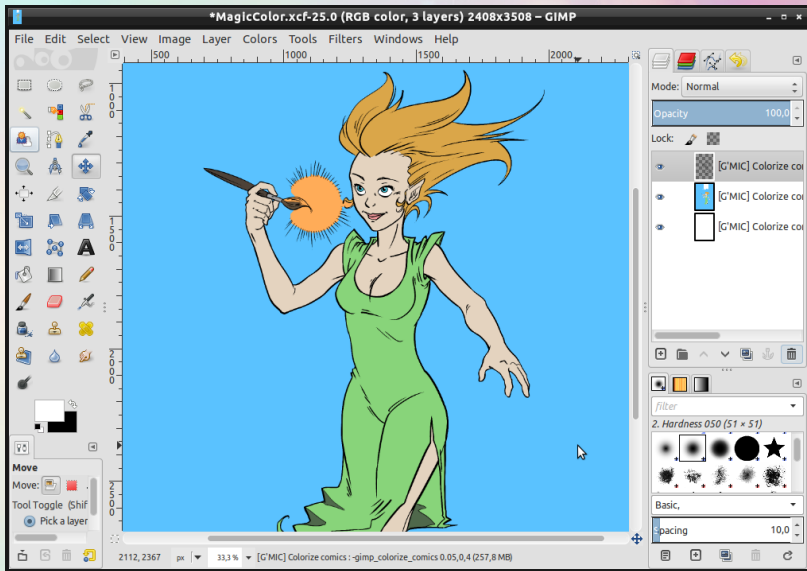


Add top layer with color spots on it.

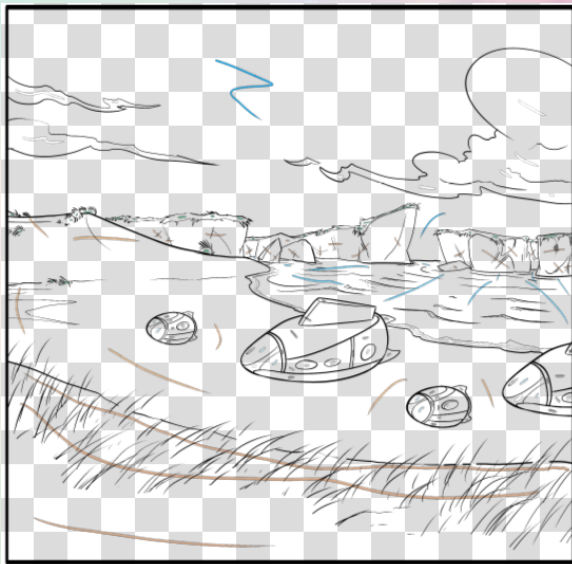


Invoke G'MIC plug-in and select **Black & White / Colorize [comics]**.

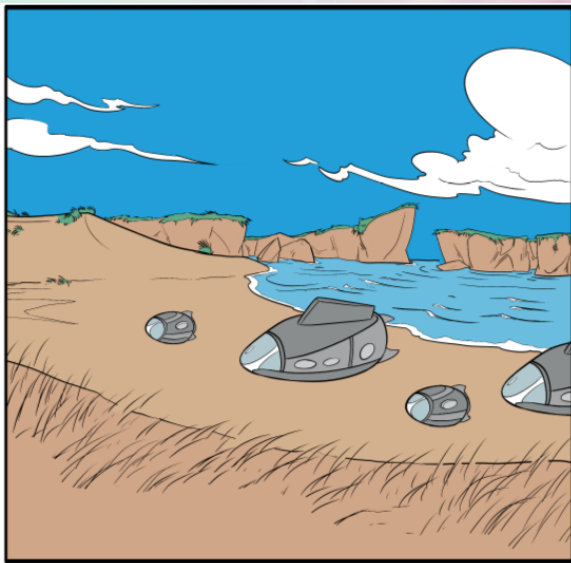
Black & white : Colorize [comics]



Wait a little bit, and enjoy !



Another example from **Thimothé Giet**: Original lineart + color strokes.



Result of the G'MIC **Colorize [comics]** filter.

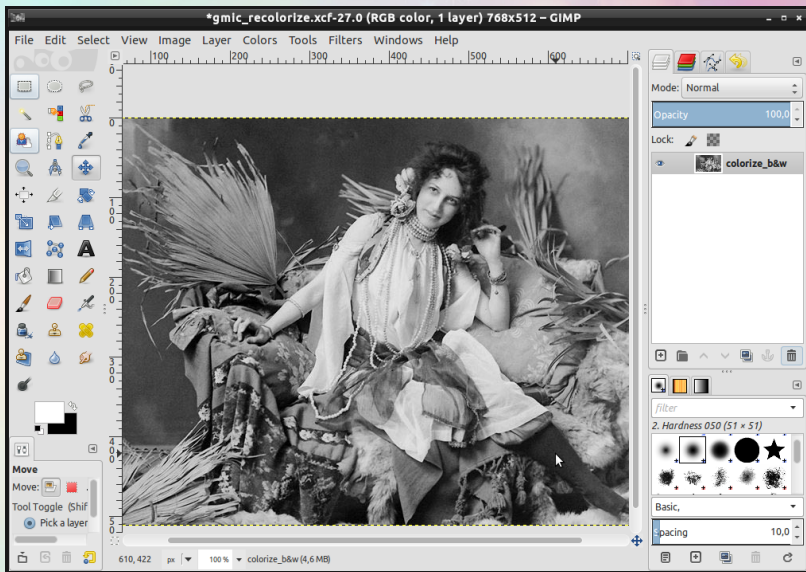
Filter Showcase:

Colorize [photographs]

- **Goal:** Same goal as before but for more classical photographs.
- **Made by :** David, to test the extension of the previous colorization algorithm to usual photographs.
- **How is this done?** Same kind of color extrapolation but **only on the chrominance channels CbCr** of the input image, so that luminance is preserved.

⇒ **24 lines of G'MIC code.**
(*all included: GUI description + algorithm*).

Black & white : Colorize [photographs] GREYC

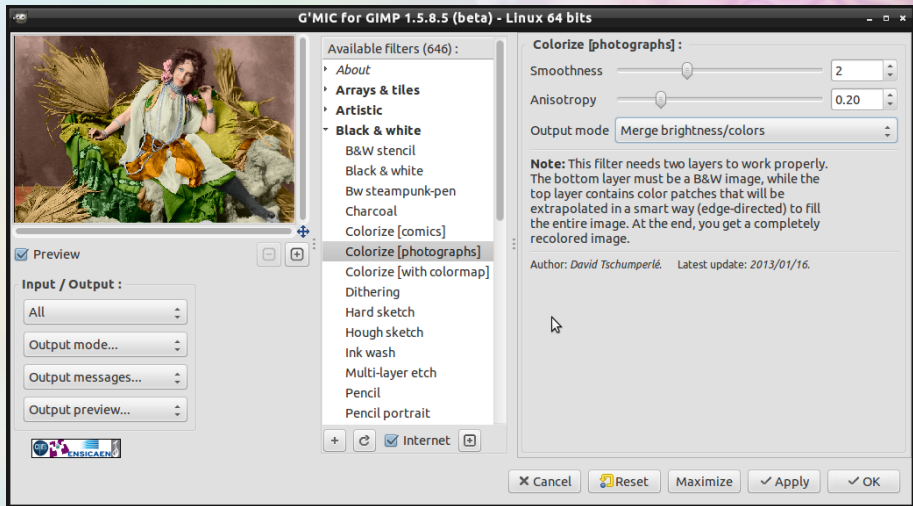


Open input image (single-layer B&W photograph).

Black & white : Colorize [photographs] GREYC

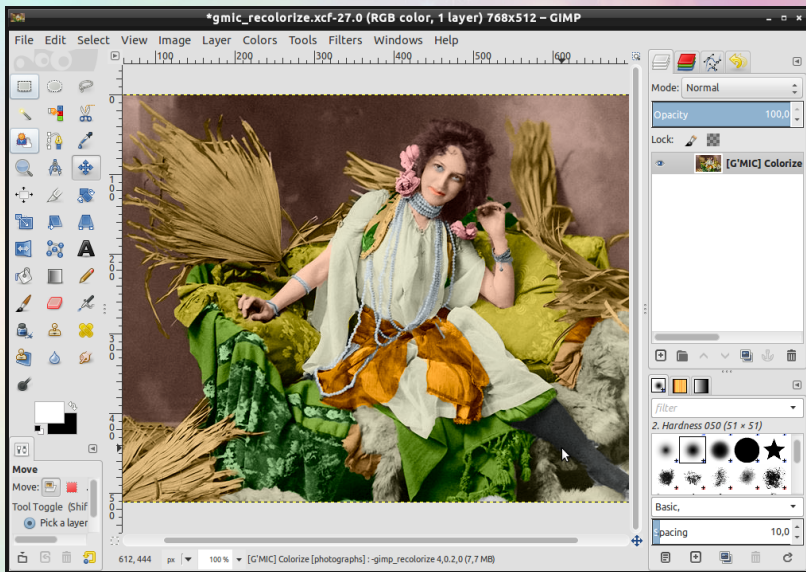


Add top layer with color strokes on it.



Invoke G'MIC plug-in and select **Black & White / Colorize [photographs]**.

Black & white : Colorize [photographs] GREYC



Result of the filter (courtesy of **pogogogo** / **GimpChat**).

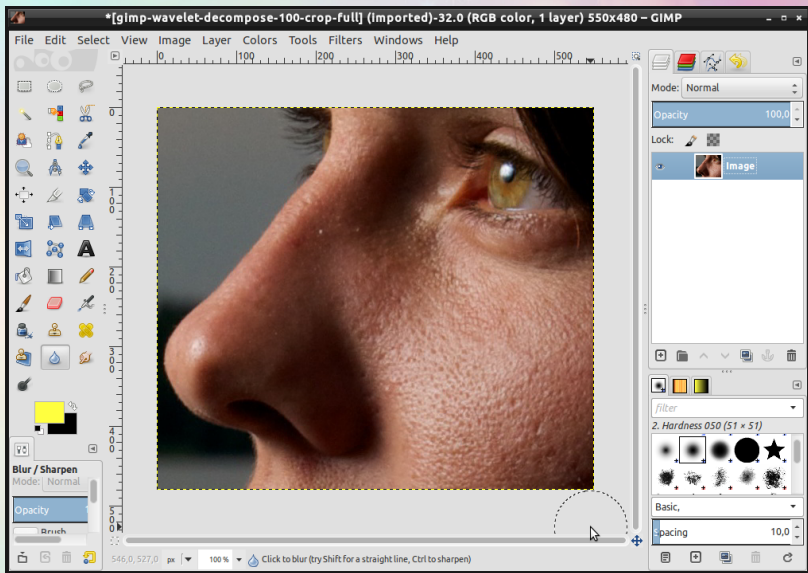
Filter Showcase:

Split details

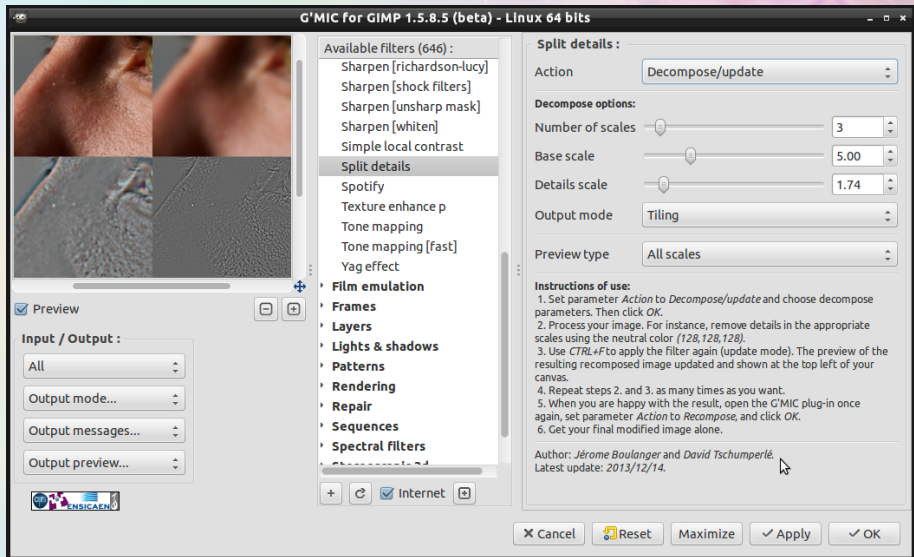
- **Goal:** Allow the decomposition of an image into several scales of details, so one can work on these different scales separately.
- **Made by :** Jérôme and David for having something similar to the **Wavelet decompose** feature in G'MIC.
- **How is this done?** Images are decomposed/recomposed using a stack of gaussian-filtered image pyramids + residuals.

⇒ **74 lines of G'MIC code.**
(all included: GUI description + algorithm).

Details : Split details



Open input image.



The screenshot shows the G'MIC for GIMP 1.5.8.5 (beta) - Linux 64 bits interface. The main window displays a preview of a skin image with a split view showing the original and the result of the 'Split details' filter. The 'Split details' dialog box is open, showing the following settings:

- Action: Decompose/update
- Decompose options:
 - Number of scales: 3
 - Base scale: 5.00
 - Details scale: 1.74
- Output mode: Tiling
- Preview type: All scales

Instructions of use:

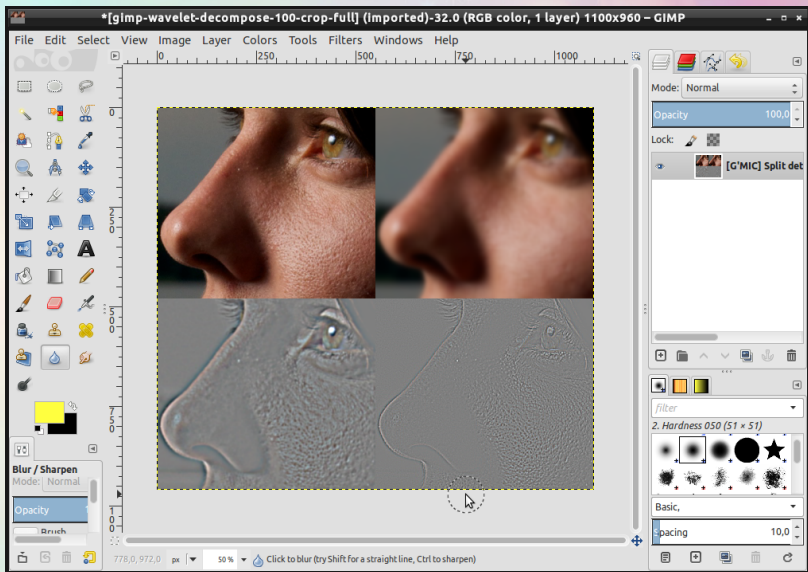
1. Set parameter *Action* to *Decompose/update* and choose decompose parameters. Then click *OK*.
2. Process your image. For instance, remove details in the appropriate scales using the neutral color (128,128,128).
3. Use *CTRL+F* to apply the filter again (update mode). The preview of the resulting recomposed image updated and shown at the top left of your canvas.
4. Repeat steps 2. and 3. as many times as you want.
5. When you are happy with the result, open the G'MIC plug-in once again, set parameter *Action* to *Recompose*, and click *OK*.
6. Get your final modified image alone.

Author: Jérôme Boulanger and David Tschumperlé.
Latest update: 2013/12/14.

Buttons at the bottom: Cancel, Reset, Maximize, Apply, OK.

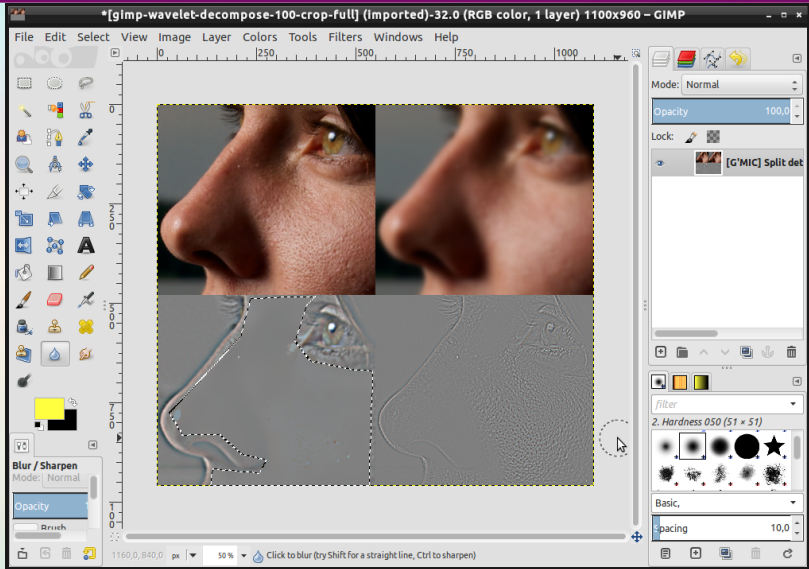
Invoke G'MIC plug-in and select **Details / Split details**.

Details : Split details

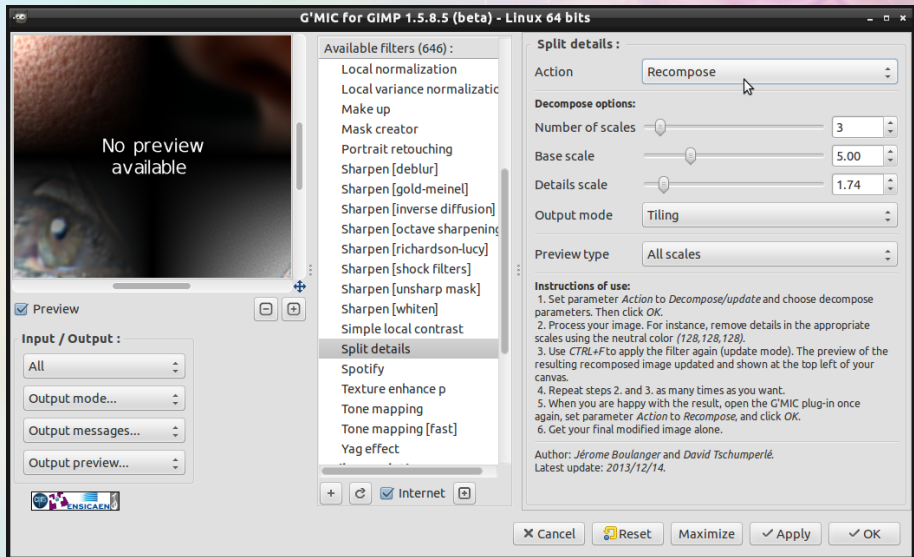


You get your input (top-left) + the decomposition into scales (here 3 scales).

Details : Split details



Do what you want on the scales (here, we simply erase the skin defects on the middle scale).



G'MIC for GIMP 1.5.8.5 (beta) - Linux 64 bits

Available filters (646) :

- Local normalization
- Local variance normalizati
- Make up
- Mask creator
- Portrait retouching
- Sharpen [deblur]
- Sharpen [gold-meinel]
- Sharpen [inverse diffusion]
- Sharpen [octave sharpening]
- Sharpen [richardson-lucy]
- Sharpen [shock filters]
- Sharpen [unsharp mask]
- Sharpen [whiten]
- Simple local contrast
- Split details**
- Spotify
- Texture enhance p
- Tone mapping
- Tone mapping [fast]
- Yag effect

Split details :

Action: Recompose

Decompose options:

Number of scales: 3

Base scale: 5.00

Details scale: 1.74

Output mode: Tiling

Preview type: All scales

Instructions of use:

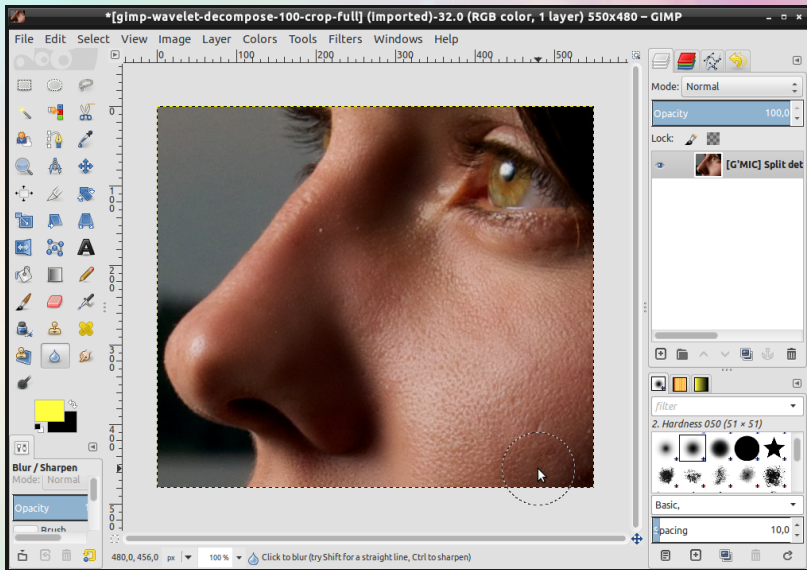
1. Set parameter *Action* to *Decompose/update* and choose decompose parameters. Then click OK.
2. Process your image. For instance, remove details in the appropriate scales using the neutral color (128,128,128).
3. Use **CTRL+F** to apply the filter again (update mode). The preview of the resulting recomposed image updated and shown at the top left of your canvas.
4. Repeat steps 2. and 3. as many times as you want.
5. When you are happy with the result, open the G'MIC plug-in once again, set parameter *Action* to *Recompose*, and click OK.
6. Get your final modified image alone.

Author: Jérôme Boulanger and David Tschumperlé.
Latest update: 2013/12/14.

Cancel Reset Maximize Apply OK

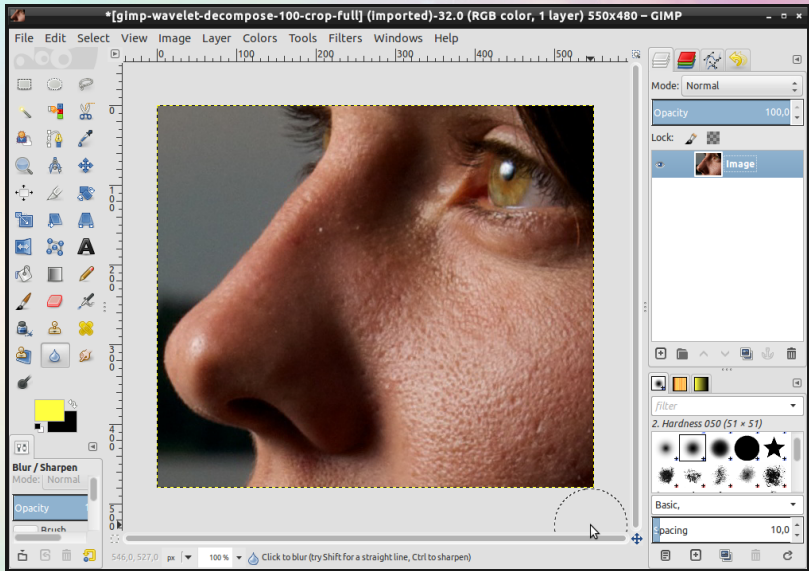
Invoke G'MIC plug-in again, to recompose the final image.

Details : Split details



Result of the recomposition, with cleaner skin (5mn work !).

Details : Split details



Comparison with initial image.

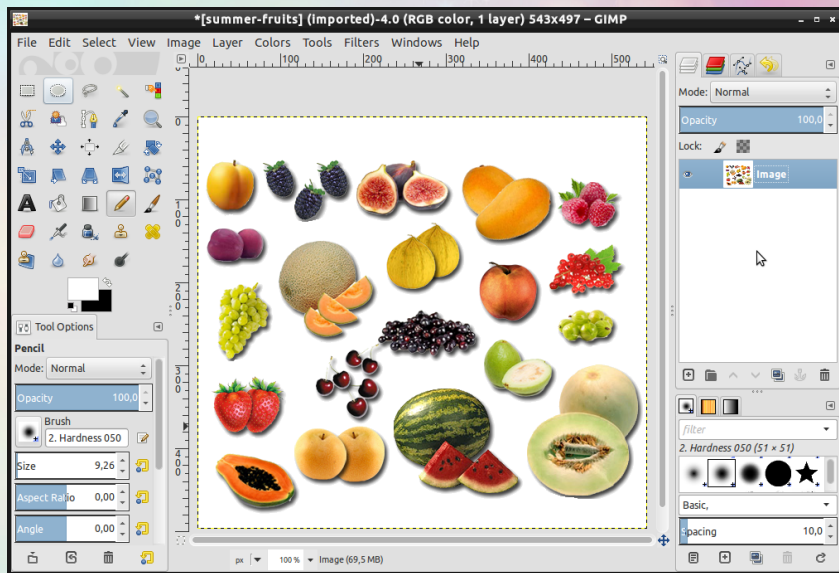
Filter Showcase:

Extract objects

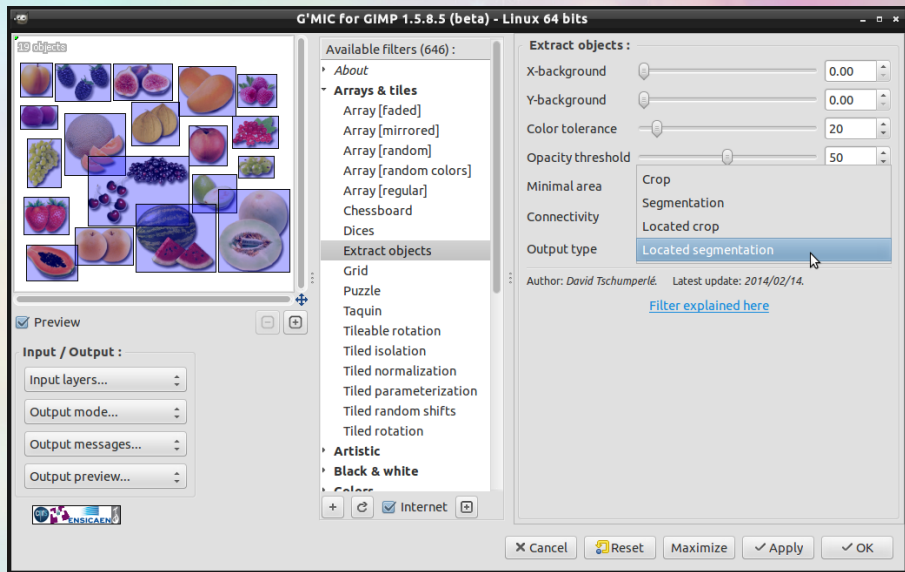
- **Goal:** Extract independent objects located on a flat colored background.
- **Made by :** David, to ease the use of the next filter **Pack Sprite**.
- **How is this done?** Background pixels are extracted (by their color), then the residual pixels are grouped into several connected regions corresponding to the objects to extract.

⇒ **72 lines of G'MIC code.**
(all included: GUI description + algorithm).

Arrays & Tiles : Extract objects



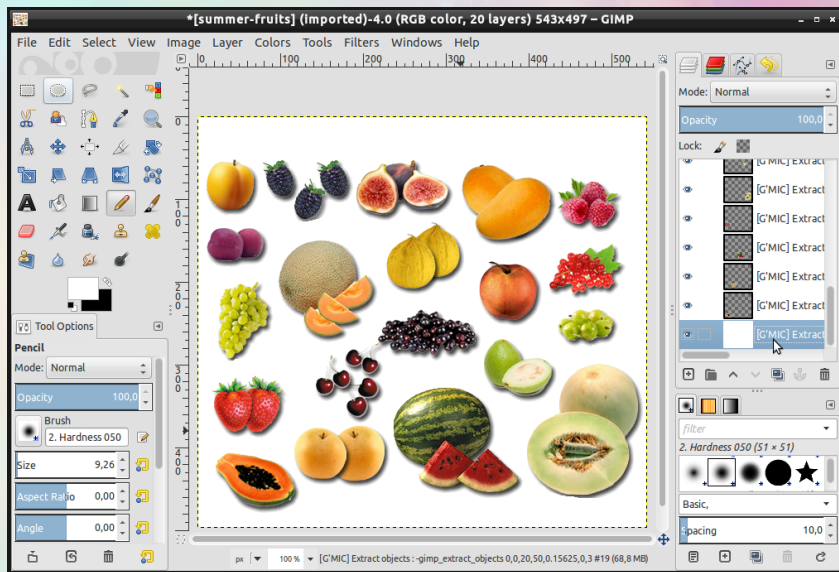
Open input image (single-layer).



The screenshot shows the G'MIC for GIMP 1.5.8.5 (beta) - Linux 64 bits interface. The main window displays a grid of 19 objects, including various fruits like oranges, grapes, strawberries, and watermelons. The 'Extract objects' filter is selected in the 'Arrays & tiles' category. The 'Output type' dropdown is set to 'Located segmentation'. The 'Available filters (646)' list includes categories like 'About', 'Arrays & tiles', 'Artistic', 'Black & white', and 'Colors'. The 'Extract objects' filter settings are visible on the right, including 'X-background', 'Y-background', 'Color tolerance', 'Opacity threshold', 'Minimal area', 'Connectivity', and 'Output type'. The 'Output type' dropdown is currently set to 'Located segmentation'. The 'Author' is David Tschumperlé, and the latest update is 2014/02/14. A link 'Filter explained here' is provided. The interface also includes a 'Preview' checkbox, 'Input / Output' settings, and a bottom bar with 'Cancel', 'Reset', 'Maximize', 'Apply', and 'OK' buttons.

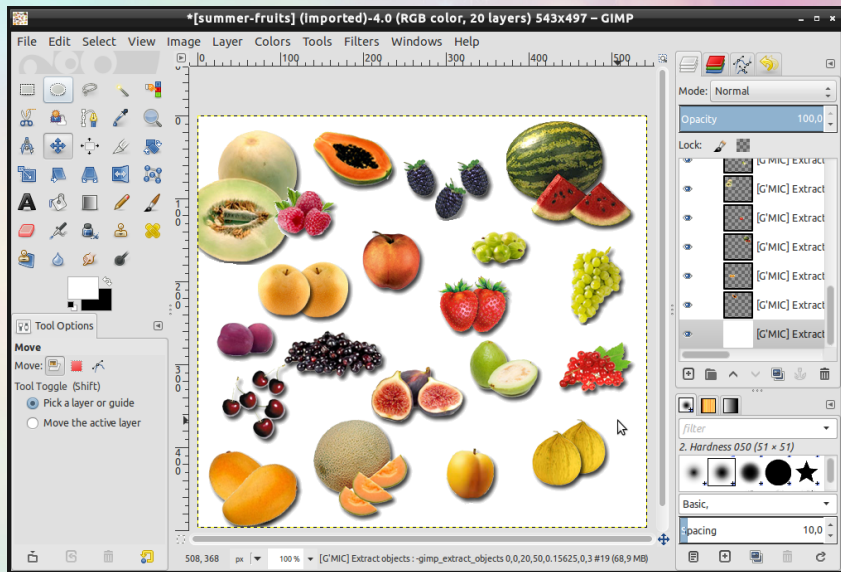
Invoke G'MIC plug-in and select **Arrays & tiles / Extract object**.

Arrays & Tiles : Extract objects



Output looks similar as input, but is divided into **several layers**.

Arrays & Tiles : Extract objects



Managing each object independently is now possible (here, position change).

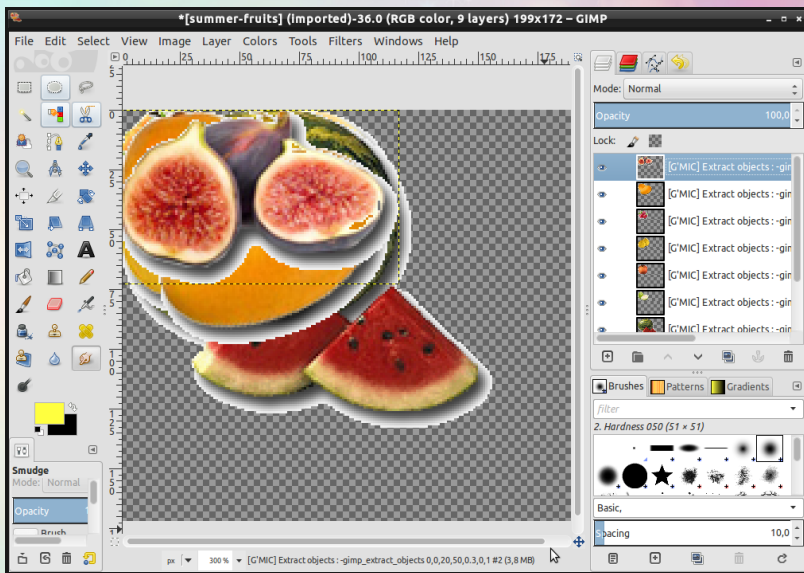
Filter Showcase:

Pack sprites

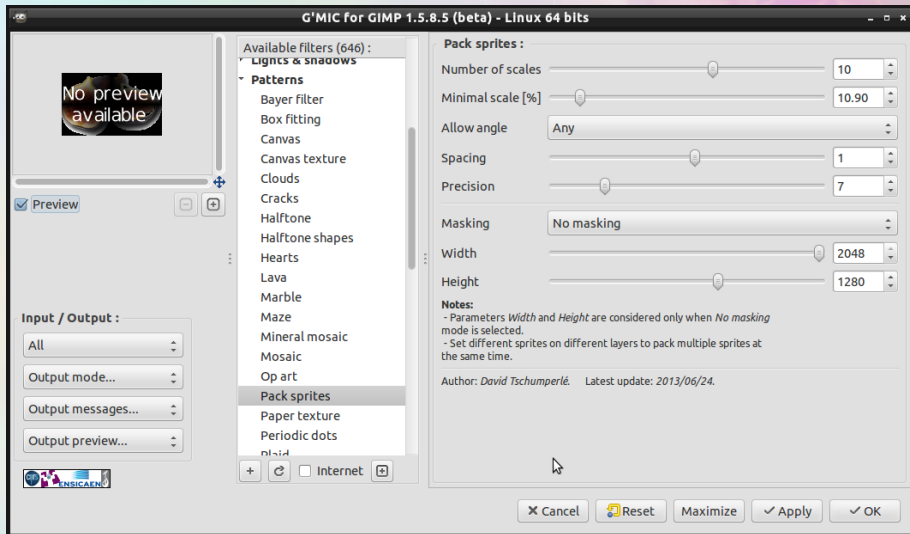
- **Goal:** Render an image where several small images have been packed together (scaled and rotated) without intersecting.
- **Made by :** David, for Lyle Kroll on GimpChat who has asked this for a long time.
- **How is this done?** Pseudo random positions (random + heuristic) are iteratively tried to pack images, with decreasing scales.

⇒ **122 lines of G'MIC code.**
(all included: GUI description + algorithm).

Patterns : Pack sprites

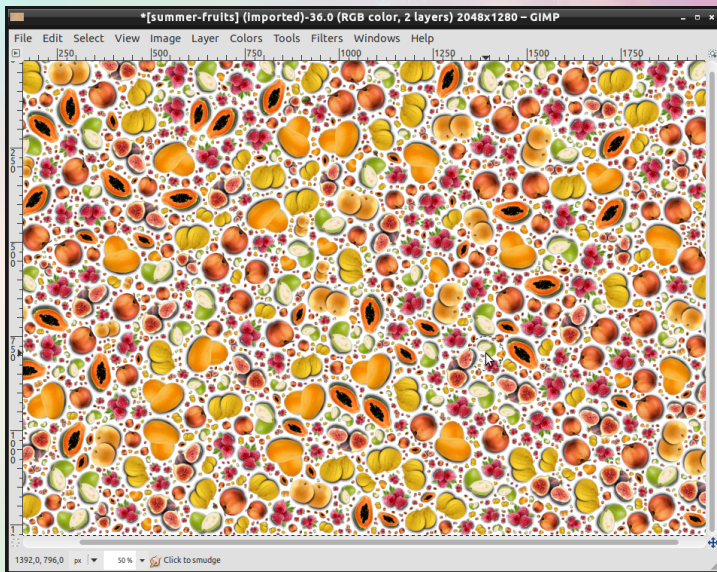


Select your objects to pack (multi-layer image).



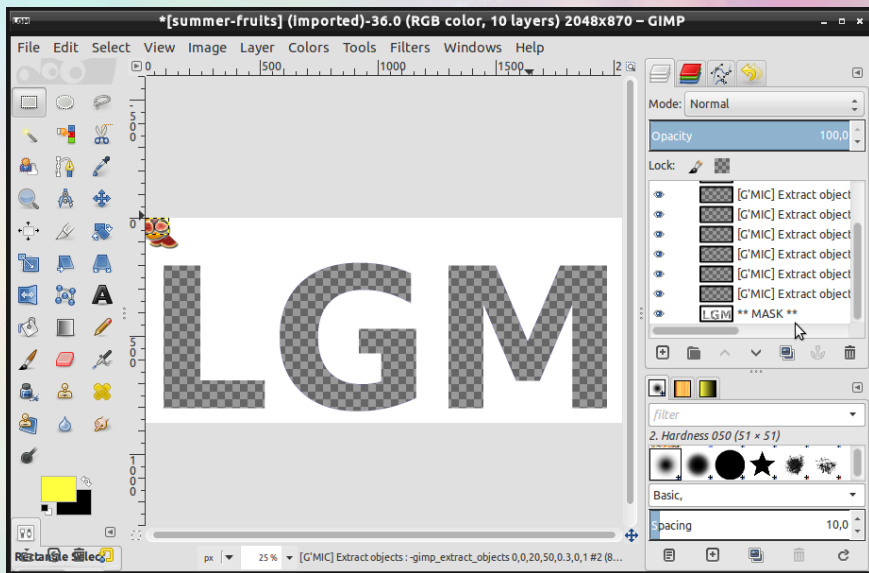
Invoke G'MIC plug-in and select **Patterns / Pack sprites**.

Patterns : Pack sprites

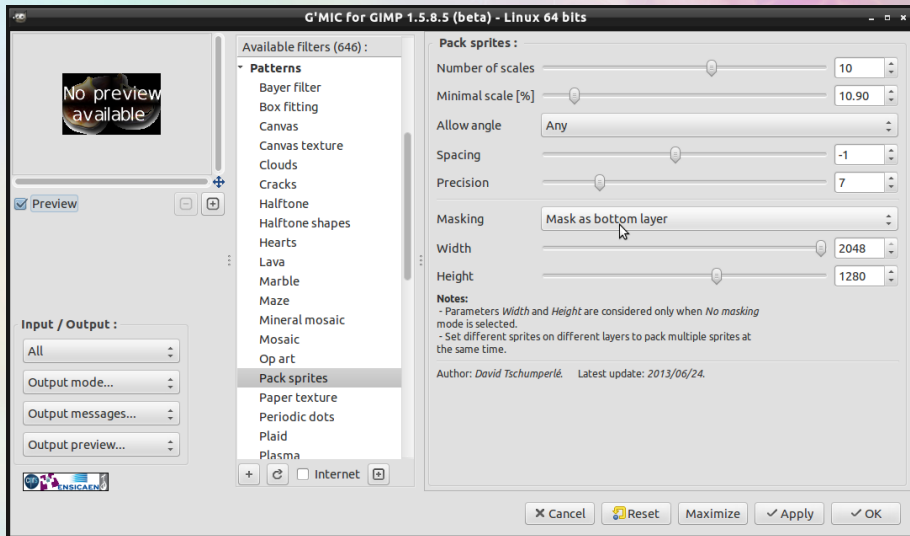


Get your image with randomly packed sprites (after a while).

Patterns : Pack sprites

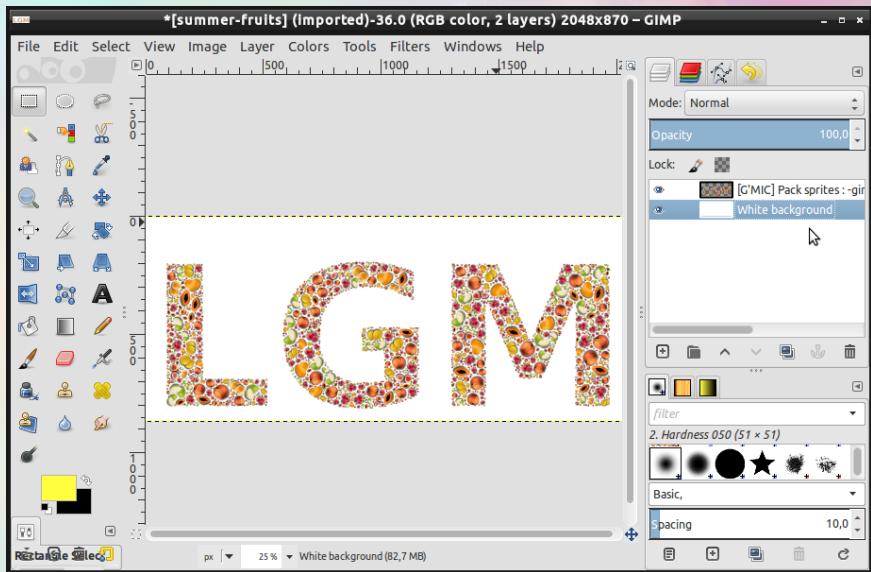


Now, you can add a bottom layer to restrict packing on transparent regions.



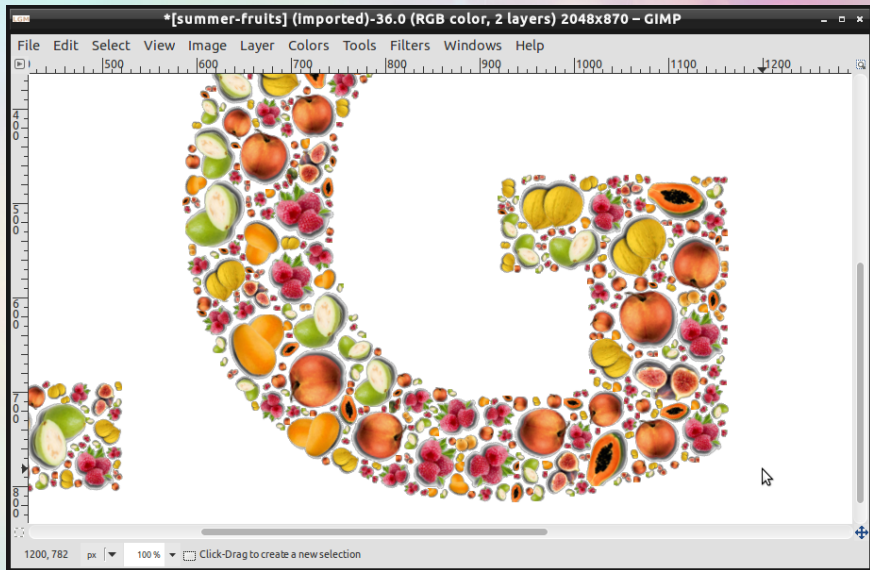
Invoke G'MIC again, and select **Mask : Mask as bottom layer**.

Patterns : Pack sprites



Go for a coffee, and you get this.

Patterns : Pack sprites



Detail of the result.



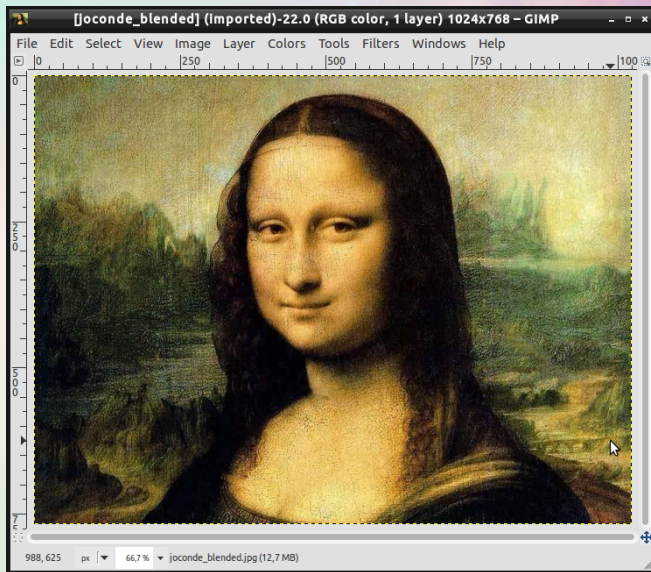
Example of rendering, by **Chris Fiedler**, on GimpChat.

Filter Showcase:

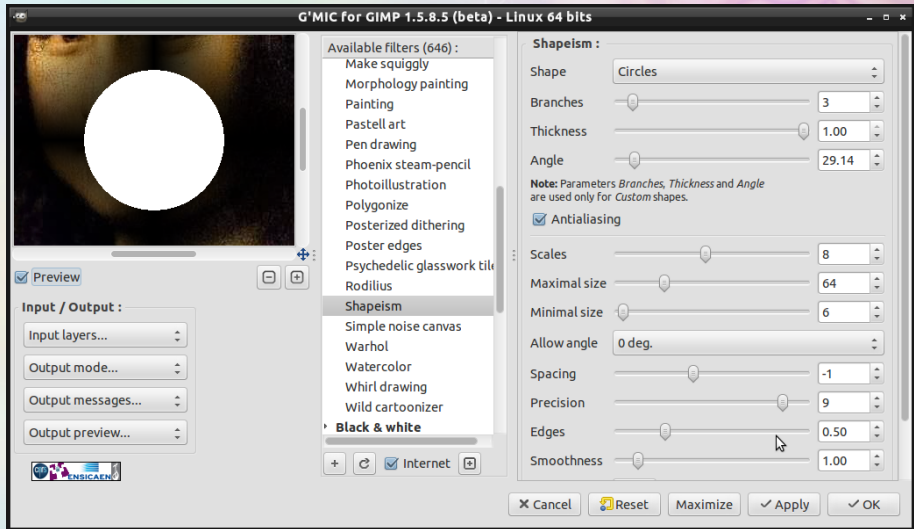
Shapeism

- **Goal:** Try to get close to the **Circlism** effect from artist **Ben Heine** (who do this manually, takes days), i.e. render an image with non-intersecting colored circles (or other shapes).
- **Made by :** **David**, for **Lyle Kroll** on GimpChat who has asked this for a long time.
- **How is this done?** Multi-scale monochrome shapes are packed together with a priority to put smaller shapes on image contours, then each shape is colored separately according to the corresponding image color behind.

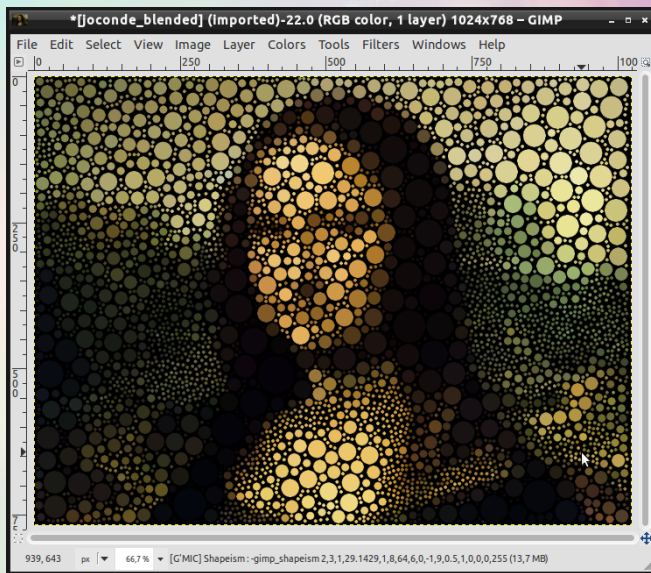
⇒ **75 lines of G'MIC code.**
(all included: GUI description + algorithm).



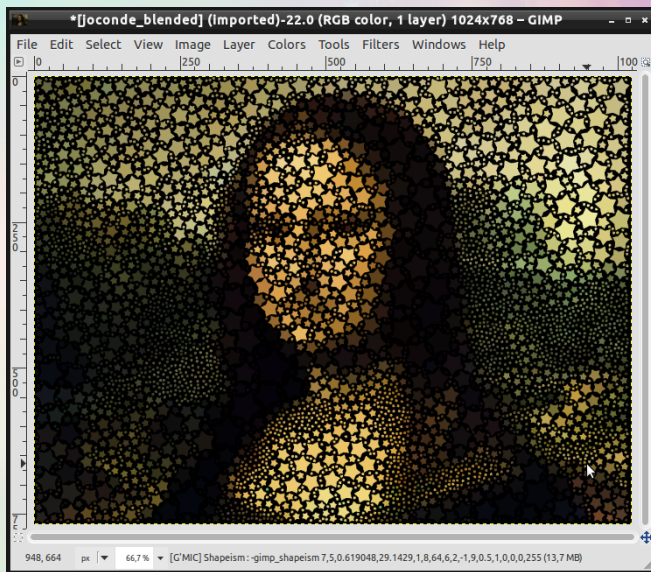
Open input image.



Invoke G'MIC plug-in and select **Artistic / Shapeism**.



Go drink a (big) coffee, and enjoy the result ! (can be slow to compute).



Result with another shape selected (a star).

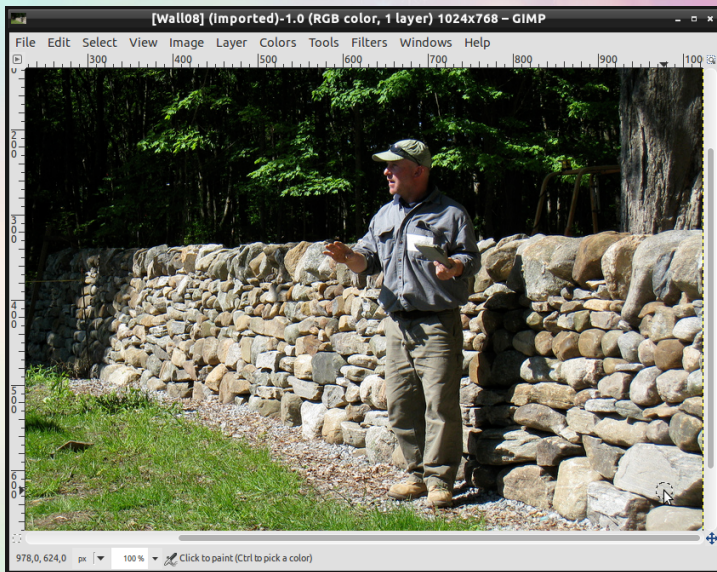
Filter Showcase:

Inpainting [patch-based]

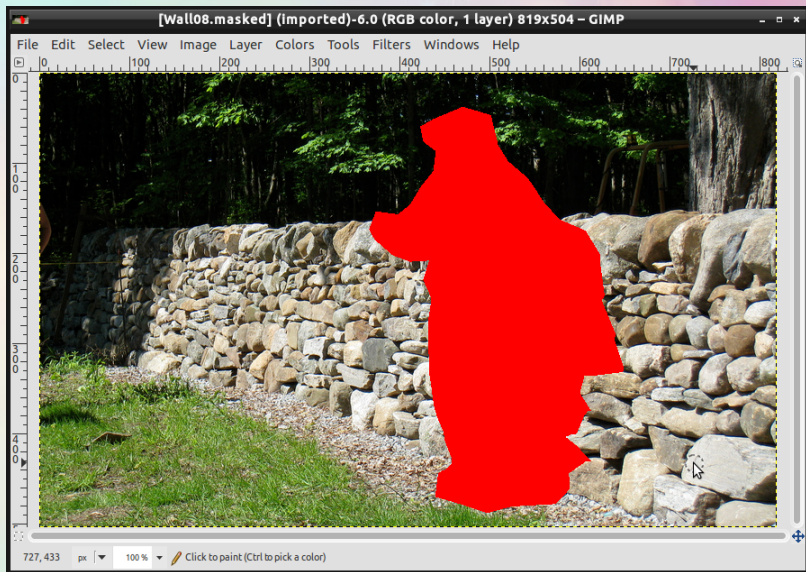
- **Goal:** Automatically heal missing image regions with a texture-aware algorithm. Similar to what the **Resynthesizer** plug-in does, but directly in G'MIC.
- **Made by :** **David** and **Maxime**, to have an alternate healing method in an active project (**Resynthesizer** looks stagnant).
- **How is this done?** It implements the **Criminisi-Perez-etal's** patch-based inpainting algorithm + a patch blending technique we've specifically designed.

⇒ **427** lines of C++ code (native command) + **35** lines of G'MIC code.
(all included: GUI description + algorithm).

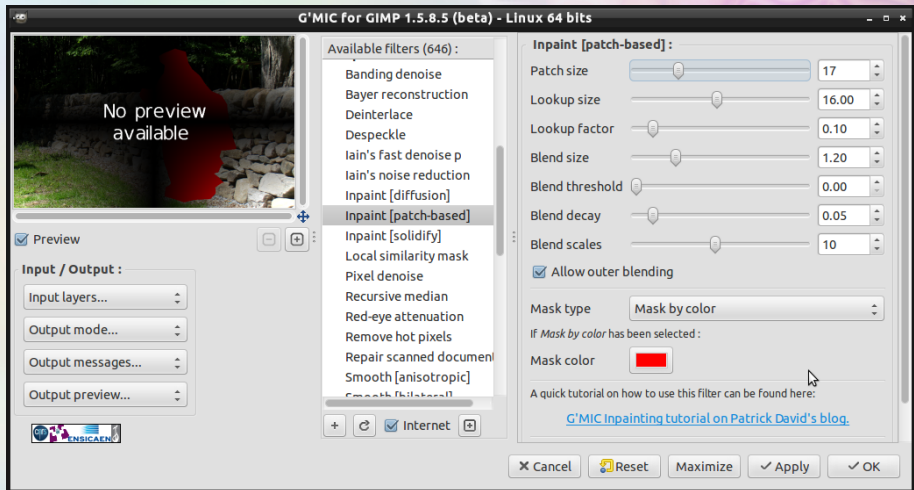
Repair : Inpainting



Open input image.

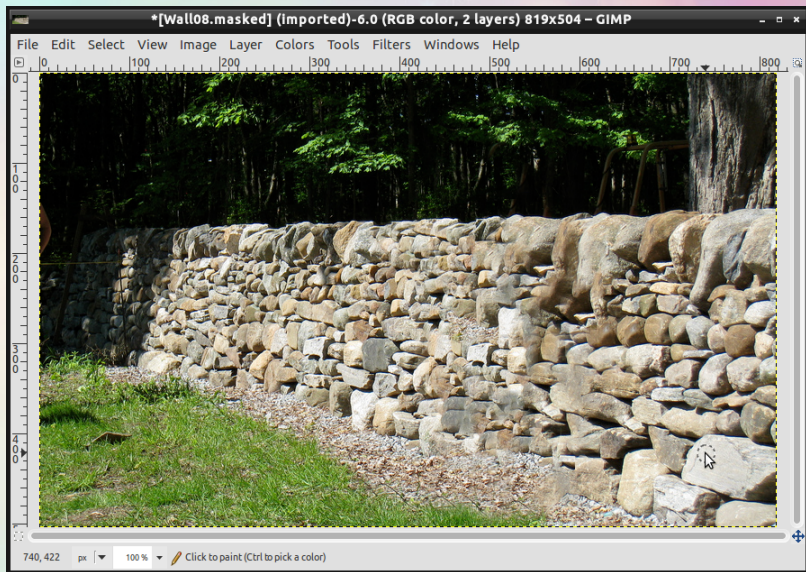


Draw an inpainting mask directly on it (with a constant known color).



Invoke G'MIC plug-in and select **Repair / Inpaint [patch-based]**.

Repair : Inpainting



If you choose carefully the parameters, this is what you get.



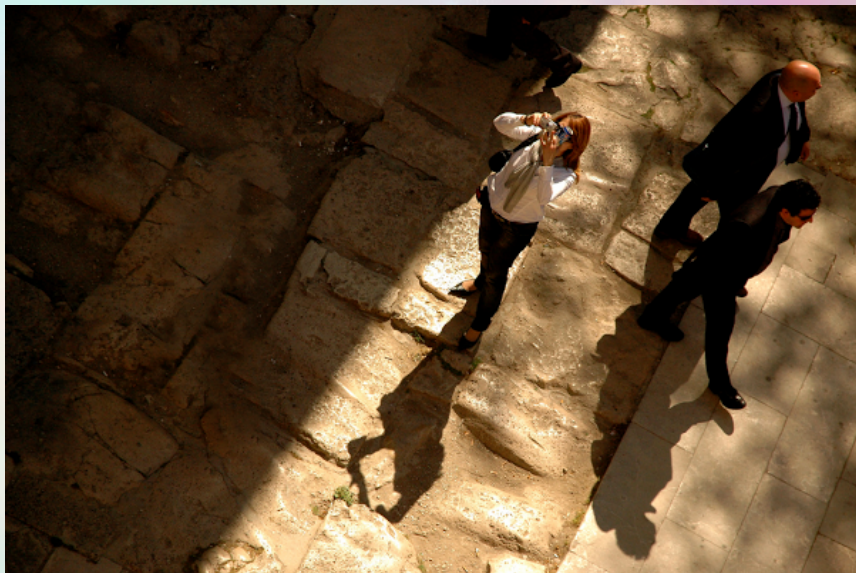
Example from **Patrick David**: Input image.



Example from **Patrick David**: Inpainted image.



Example from **Patrick David**: Input image.



Example from **Patrick David**: Inpainted image.

Input



Comparison with [Resynthesizer](#) (**Extreme case!**):
Input image (boat to be removed).

Resynthesizer



Result by the [Resynthesizer](#) heal selection algorithm.



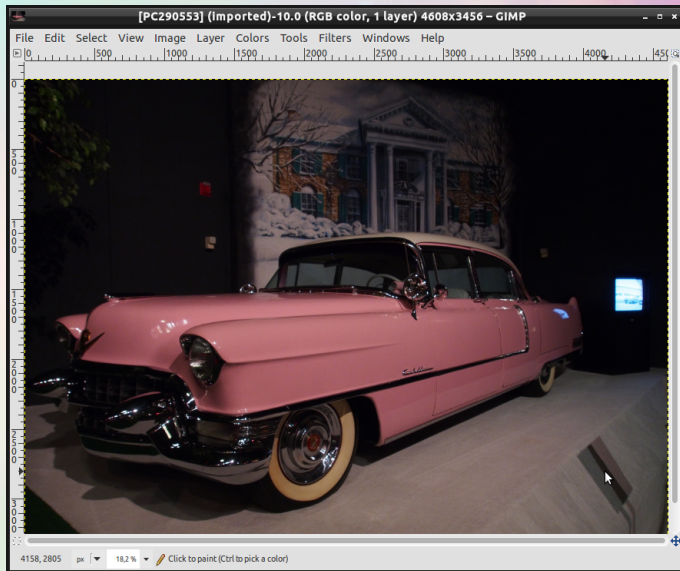
Result by the **G'MIC** inpainting algorithm.

Filter Showcase:

Denoising filters

- **Goal:** Provide a lot of algorithms to smooth images without losing (too much) the details and the textures, e.g. to remove shot noise.
- **Made by :** David, Jérôme, Iain and others.
- **Made for :** A lot of people need this. This is the logical sequel of our previous plug-in called GREYCstoration (*now discontinued*).
- **How is this done?** Lot of different smoothing algorithms have been implemented in G'MIC: Diffusion PDE's, NL-means, Wavelets-based, etc... In 2013, we have parallelized most of them.

Repair : Denoising filters



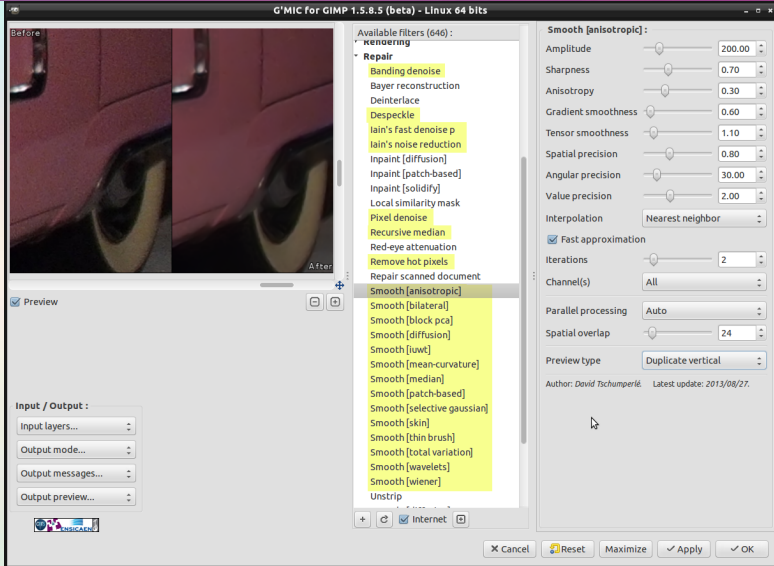
Open input (noisy) image.

Repair : Denoising filters



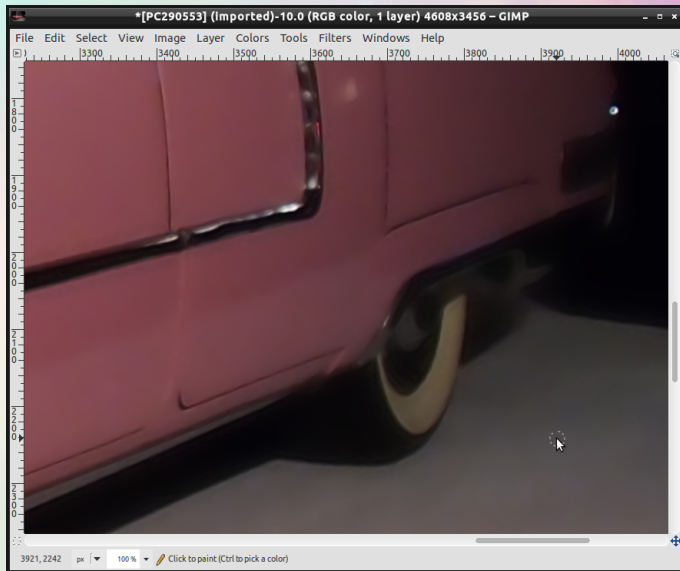
Open input (noisy) image (**detail**).

Repair : Denoising filters



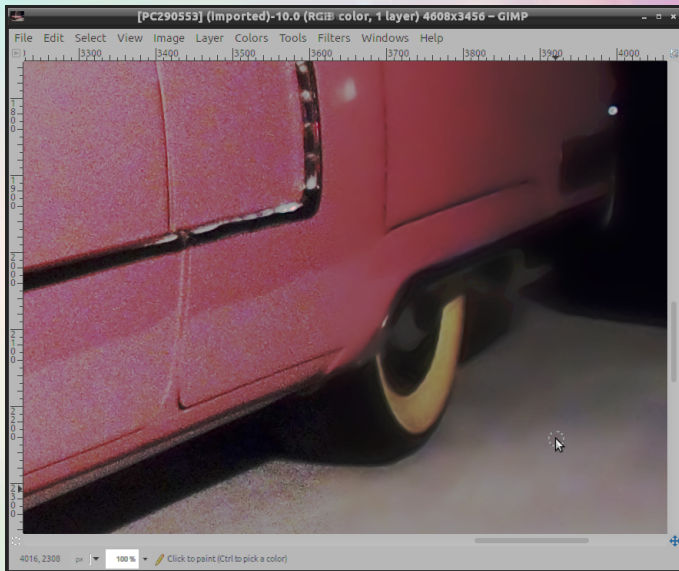
Invoke G'MIC plug-in, and select one of the denoising filters
(more than 20 methods available).

Repair : Denoising filters



Denoised result (with heavy parameters for making the effect more clear).

Repair : Denoising filters



Comparison between original / denoised image (equalized images for clarity).

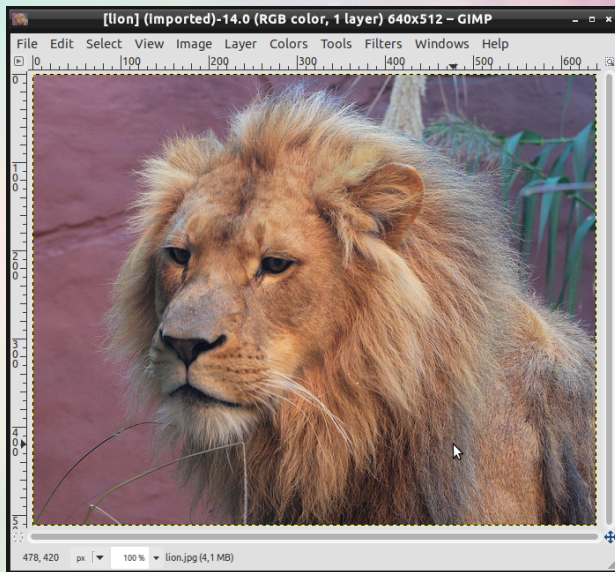
Filter Showcase:

Dream smoothing

- **Goal:** Apply exaggerated edge-directed smoothing and boost colors to create a kind of painting effect.
- **Made by :** Arto Huotari (aka Naggobot), artist and coder at the same time, who uses it on his own photographic workflow.
- **How is this done?** It intensively uses anisotropic smoothing (native G'MIC feature) as well as aggressive color mixing.

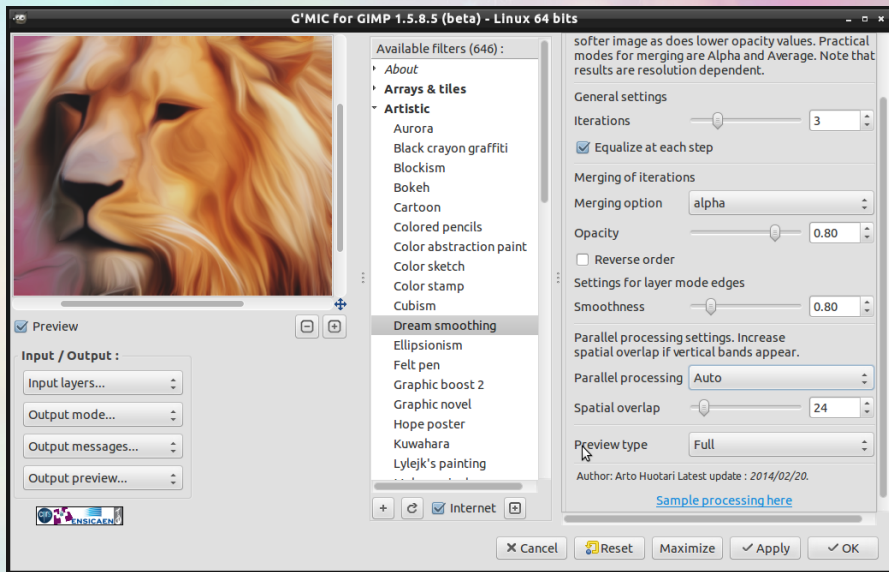
⇒ **76 lines of G'MIC code.**
(all included: GUI description + algorithm).

Artistic : Dream smoothing

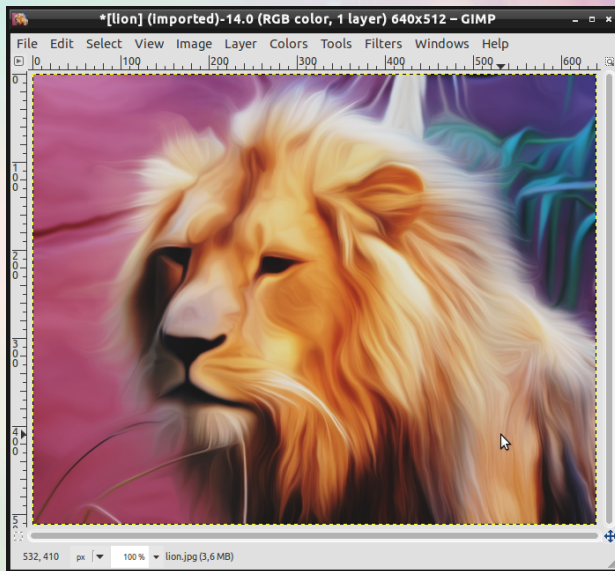


Open input image.

Artistic : Dream smoothing



Invoke G'MIC plug-in and select **Artistic / Dream Smoothing**.

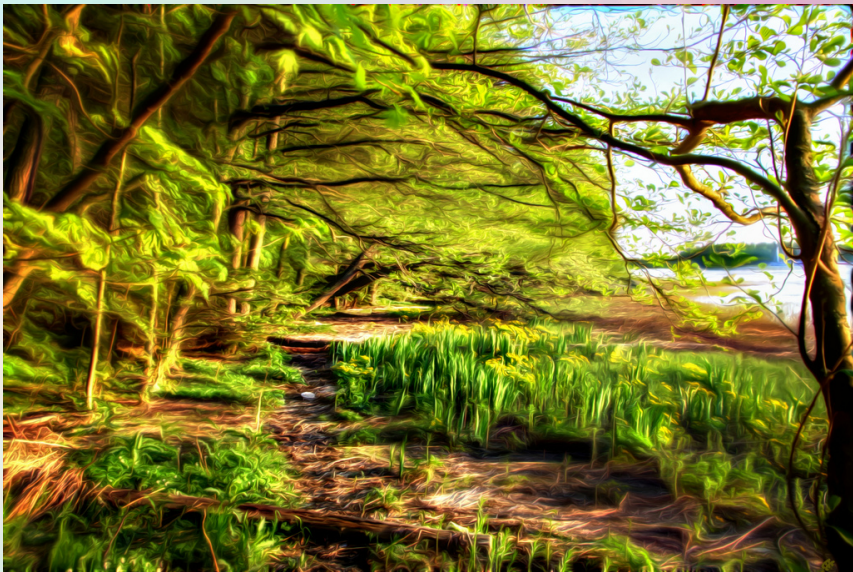


Enjoy your result ! (takes some time to render, recently parallelized).



Zarir Madon

How artists use it for real: Processing done by **Zarir Madon**.



How artists use it for real: Processing done by **Arto Huotari**.

Filter Showcase:
Film emulation

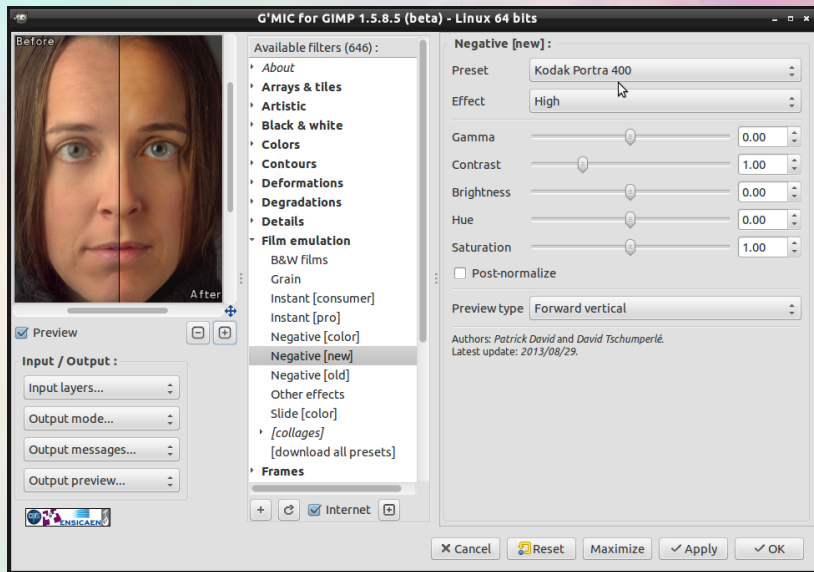
- **Goal:** Provide free film emulation filters, similar to what proprietary **DXO FilmPack** proposes.
- **Made by :** **Patrick** requested **David** to make his color profiles easily available for everyone.
- **How is this done?** Color transformations are encoded as RGB CLUT files, stored on the G'MIC server. Each color profile is downloaded on demand.

⇒ **476 lines of G'MIC code (mostly for GUI).**
(all included: GUI description + algorithm).

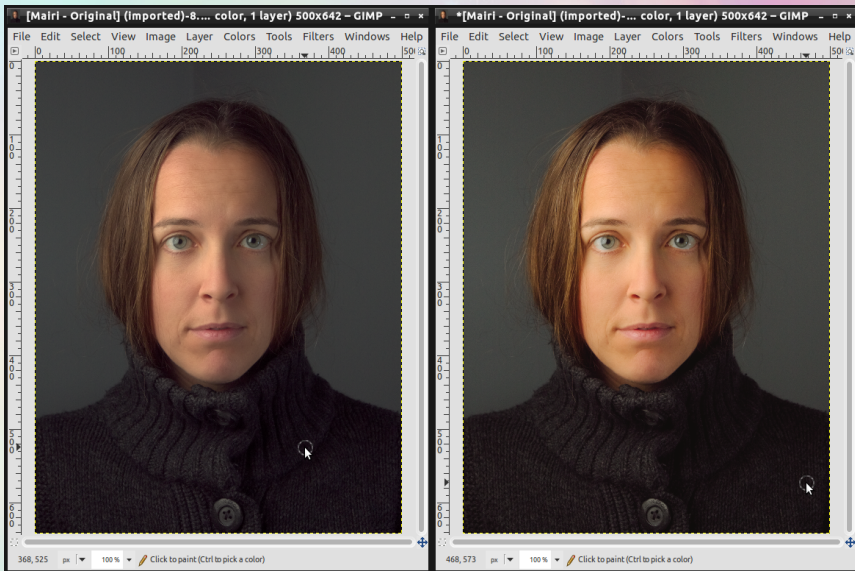


Open input image.

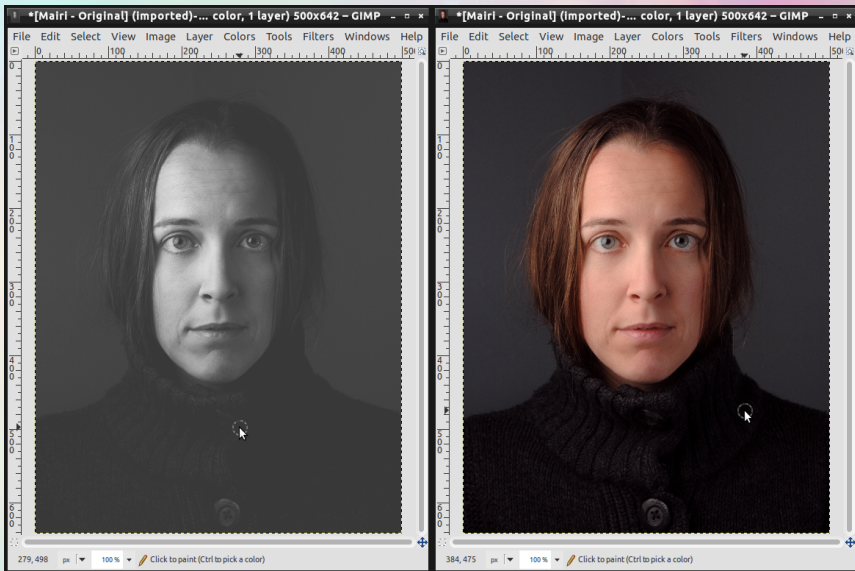




Invoke G'MIC plug-in, and choose one filter in folder **Film emulation/**.



Comparison: Before (*left*) / After (*right*).



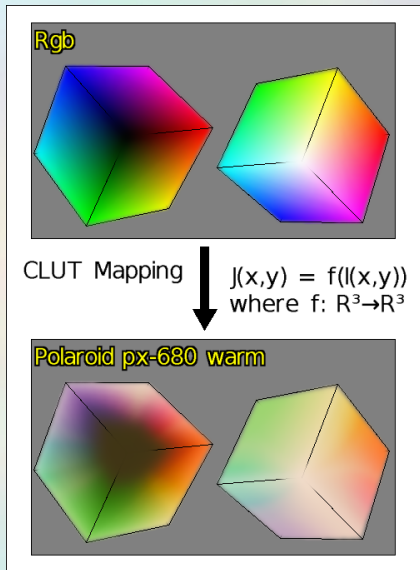
Two other examples: TMAX-3200 (*left*) and Kodak Kodachrome 64 (*right*).

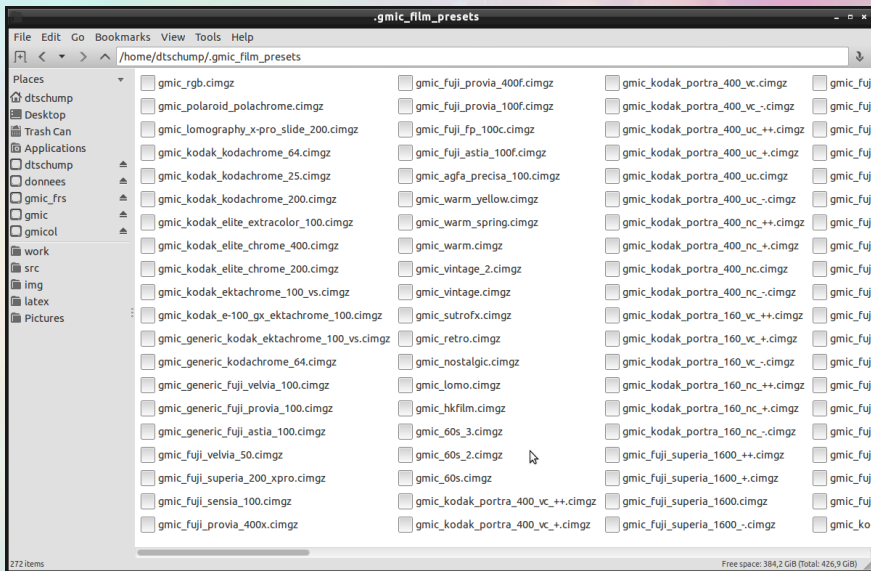


Patrick David has indeed done a lot of presets (here, a sample of them).

Technically speaking:

- Each preset defines a mapping function from RGB to RGB (CLUT).
 - The values of these functions are explicitly stored for all RGB colors.
 - To avoid huge datasets, we consider **64x64x64 downsampled versions** of the CLUTs and interpolate intermediate colors.
- **77Mb of data for 271 film emulation presets.**
- As the original color mappings are **smooth functions**, interpolation has almost no incidence on the quality.





Once downloaded, presets are stored locally on your drive for **off-line use**.

Conclusions

- G'MIC is really meant to be a **generic** image processing framework.
- All filters we regularly can be potentially available **for all interfaces or open-source projects** that integrates the G'MIC library.
- Since the beginning, lot of filters have been done **in collaboration with artists**. Lot of good ideas come from users.

Thanks for your attention!

Any questions ?

