G'MIC

GREYC's Magic for Image Computing

A Full-Featured Open-Source Framework for Image Processing

http://gmic.eu

Présentation Equipe IMAGE / GREYC, Avril 2017

Context

- Research in the field of image processing at the **GREYC** lab of ENSICAEN / CNRS / University of Normandy (Caen).

⇒ Design of innovative algorithms to solve generic image processing problems (denoising, enhancement, segmentation, feature detection,...).

Context

- Frequent collaborations with companies / laboratories having specific images to process.

...

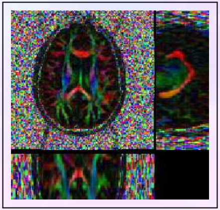
⇒ Various image data coming from very diverse sensors.

Context

- Image data are diverse: 2D, 2D+t, 3D, 3D+t, vector or matrix-valued pixels, float values, ...

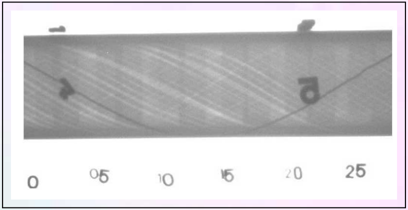
⇒ We stray far from usual 2D color pictures!

(a) $I_1 : W \times H \to [0, 255]^3$



(b) $I_2 : W \times H \times D \to [0, 65535]^{32}$

(c) $I_3 : W \times H \times T \to [0, 4095]$



(d) $I_4 : W \times H \times T \to [0, 4095]$

Motivations

- Needs for **tools** to visualize / explore data, convert image formats, apply classical IP operators (filtering, geometric transformations, frequential analysis, ...) for very generic image data, sometimes on thousands of images at the same time.

- Typical "technical" question we can ask for:

  "How to convolve 500 volumetric images having 32 channels each by 3d anisotropic gaussian kernels ?"

Motivations

⇒ Very few **open-source tools** exist for these tasks. They tend to be either:

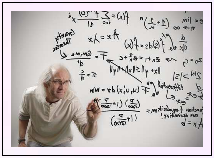
- Easy to use, but not generic enough for our image data (GIMP, ImageMagick, GraphicsMagick, ...).
- Or very flexible, but reserved for experienced programmers (require the writing of code, using specialized external libraries).

- We did like others: The team has developed generic libraries for image processing: **CImg** and **Pandore** (in C++):

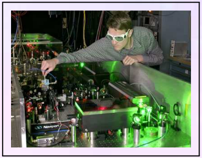

```
http://cimg.eu
https://clouard.users.greyc.fr/Pandore/
```

Motivations

- In practice, the libraries are used only by a few hundred “experimented” programmers.
  $\Rightarrow$ Cause: High diversity of people in the image processing field !



Mathematicians



Physicists

Programmers

Biologists ...

$\Rightarrow$ We clearly need more **simpler interfaces** (than C++ libraries) if we want to enlarge our audience.

Goals of the G'MIC project

GREYC's Magic for Image Computing

http://gmic.eu

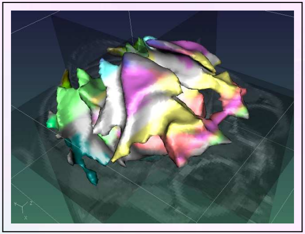
- **For the users :** Define different user interfaces to do image processing. (provided interfaces are more or less friendly - and powerful - depending on the user's skill.
- **For the developers :** Ease algorithm prototyping and maintenance.

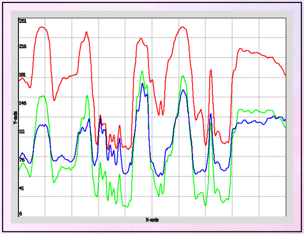
→ **Technical mean :** Definition of a **full-featured**, **concise** script language for **the processing of generic image data** (G'MIC language). Interpreter used as a base layer for all user interfaces.

Goals of the G'MIC project

- Definition of a **comprehensive** and **concise** script language for **the processing of generic image data** (**G'MIC** language).
  - **Full-featured:** More than 950 commands available for the visualization, filtering, geometric and colorimetric transformations, feature extractions, 3d rendering, matrix calculation, primitive drawing, ...

⟶ Documentation (.pdf) has more than 450 pages.
⟶ Reference documentation: `http://gmic.eu/reference`





- Distribution of a **open-source implementation** of the **G'MIC** language interpreter (as a C++ library).

  - **Integration:** Possible integration of **G'MIC** features in third-party software or plug-ins (photo retouching, digital painting, video editing software, ...).

  - **Free software license:** Distributed under the CeCILL license (GPL-compatible).

⟶ Some existing integrations of libgmic to date:

★ *Krita*, digital painting software.
★ *Photoflow*, non-destructive photo retouching software.
★ *EKD*, video post-production software.

Goals of the G'MIC project

- Provide **user interfaces for everyone**, embedding the **G'MIC** language interpreter (multi-platform).
  - **gmic:** Command-line tool to manipulate generic images.
    Complementary to the CLI tools from ImageMagick / GraphicsMagick.

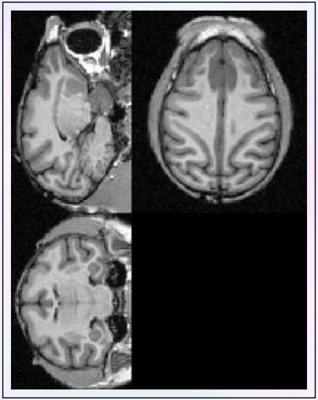
```
dtschump@[redacted]:~$ gmic ~/work/img/lena.bmp -blur 3 -mirror x
[gmic]-0./ Start G'MIC parser.
[gmic]-0./ Input custom commands file '/home/dtschump/work/src/resources.gmic' (added 14 commands, total 1121).
[gmic]-0./ Input custom commands file '/home/dtschump/work/src/gmic/src/gmic_def.gmic' (added 1107 commands, total 222
8).
[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 = (CImg<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 61.9959 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@[redacted]:~$
```

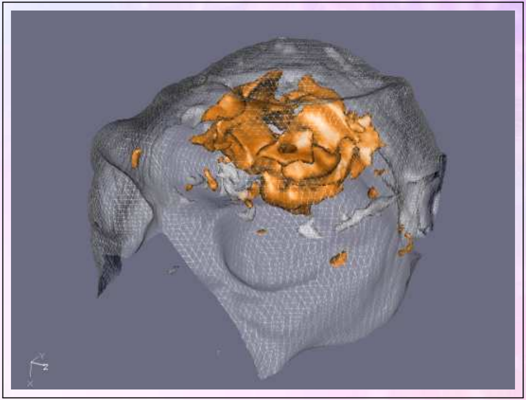
Example: using the CLI tool "gmic"

```
$ gmic lena.bmp -blur 3 -sharpen 1000 -noise 30 -+ "'cos(x/3)*30'"
```

Example: using the CLI tool “gmic”

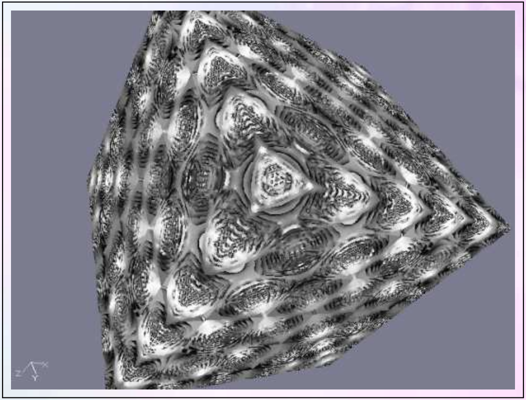
```
$ gmic reference.inr -flood 23,53,30,50,1,1,1000 -flood[-2] 0,0,0,30,1,1,1000
-blur 1 -isosurface3d 900 -opacity3d[-2] 0.2 -color3d[-1] 255,128,0 -+3d
```





Example: using the CLI tool “gmic”

```
$ gmic -isosurface3d "'sin(x*y*z)'",0,-10,-10,-10,10,10,10,128,128,64
```

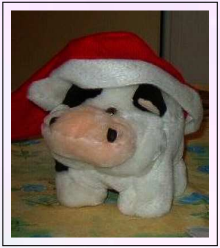


Example: using the CLI tool “gmic”

```
$ gmic milla.bmp -f '255*(i/255)^1.7' -histogram 128,0,255 -a c -plot
```

is the **G'MIC** equivalent to this C++ code (using CImg):

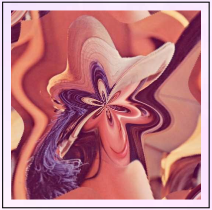
```
#include "CImg.h"
using namespace cimg_library;
int main(int argc,char **argv) {
const CImg<>
img("milla.bmp"),
hist = img.get_histogram(128,0,255),
img2 = img.get_fill("255*((i/255)^1.7)",true),
hist2 = img2.get_histogram(128,0,255);
(hist,hist2).get_append('c').display_graph("Histograms");
return 0;

}
```



First release: August 2008

- First release done in August 2008 $\Longrightarrow$ very few downloads (approx. 300/month).
- But... Writing image processing pipelines in **G'MIC** also allows to design and develop artistic filters and effects easily...

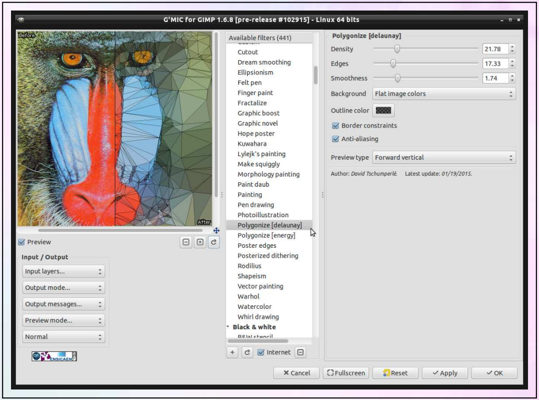
```
$ gmic lena.jpg -pencilbw 0.3 -o gmic_lena1.jpg
$ gmic lena.jpg -flower 10 -o gmic_lena4.jpg
```



$\longrightarrow$ Why not writing a **G'MIC** plug-in for **GIMP** ?

Goals of the G'MIC project

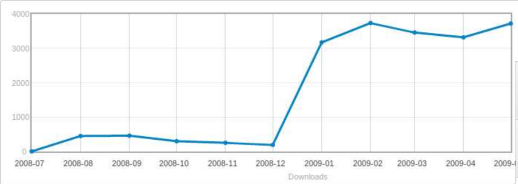
- Provide **user interfaces for everyone**, embedding the **G'MIC** language interpreter (multi-platform).
  - **gmic_gimp:** Plug-in for GIMP that provides hundred of **G'MIC** -based image filters for 2D RGB or RGBA images.



Before / after the GIMP plug-in

- First release of the **G'MIC** plug-in GIMP in January 2009.
  ⟶ Significant increase of the downloads and page views.

Home (Change File) Date Range: 2008-07-24 to 2009-05-06



DOWNLOADS
19,070
In the selected date range

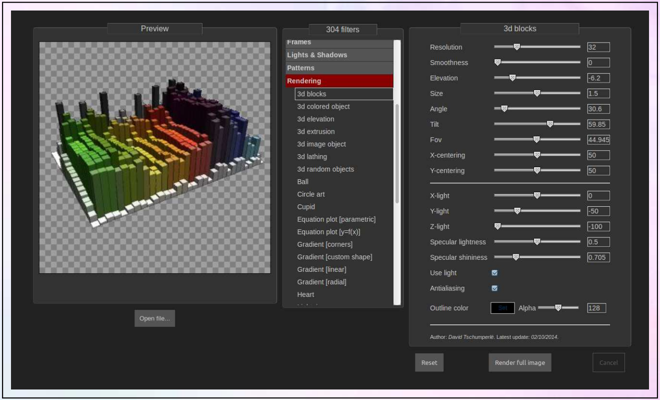
TOP COUNTRY
N/A

TOP OS
N/A

Date ▲	Downloads ⬍
2009-05	3,722
2009-04	3,320
2009-03	3,460
2009-02	3,736
2009-01	3,171
2008-12	193
2008-11	254
2008-10	300
2008-09	461
2008-08	451
2008-07	2

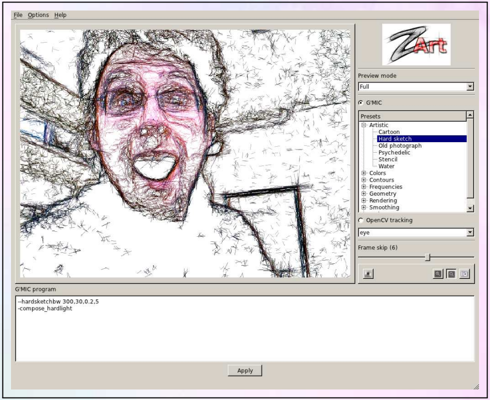
Goals of the G'MIC project

- Provide **user interfaces for everyone**, embedding the **G'MIC** language interpreter (multi-platform).
  - **G'MIC Online:** Web service for applying image filters and effects online (requires only a web browser).
    `https://gmicol.greyc.fr`

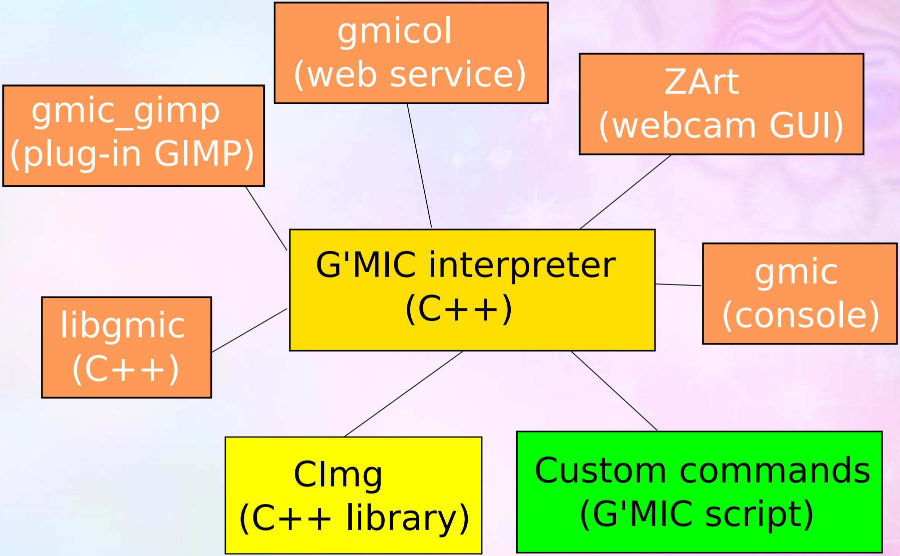


Goals of the G'MIC project

- Provide **user interfaces for everyone**, embedding the **G'MIC** language interpreter (multi-platform).
  - **ZArt:** Qt-based interface for the manipulation of video sequences (webcam or video file). Used as a demonstration platform.



Global view of the G'MIC framework



- Script language (interpreted).
- Define a set of **native** "low-level" commands :
  e.g, `-convolve`, `-display`, `-if`, `-then`, `-else`, ...
  → **C++**, compiled, often multi-threaded (**OpenMP**).
- Define a set of **custom** commands :
  e.g, `-polygonize`, `-apply_gamma`, `-x_pacman`, ...
  → Grouped in the G'MIC **standard library**.
- Users can defined their own **custom libraries** of commands :
  e.g, `$ gmic user.gmic -my_command ...`
  → **Versatile and evolutive** framework (standard library updatable from network).
- Most G'MIC commands are actually **custom commands**.
  200 native, **+750** custom.
- Commands to manage also **display windows** and **user events**.

The G'MIC programming language

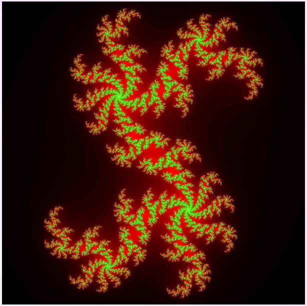
- Portion of the `-x_pacman` command:

```
score0="10" score1="100" score2="1000" score3="5000" score4="Argh!"
-repeat 5
  0 -t. ${score$>},0,0,13,1,255,255,255 -autocrop. 0 -expand_xy. 1,0 --dilate. 3
  -nm. scorem$> -nm.. score$>
-done
time4=255,255,255 time3=255,255,32 time2=255,128,32 time1=255,32,32
-repeat 11 0 -t. $<" s",0,0,23,1,${time{min(4,round(($<+1)/2))}} -nm. time$< -done
0 -t. "Get Ready!",0,0,32,1,255 -autocrop. 0 -expand_xy. 4,0 --dilate. 8 -r.. 100%,100%,1,3
-nm.. get_ready -nm. get_readym
0 -t. "Game\nOver!",0,0,53,1,255 -autocrop. 0 -expand_xy. 4,0 --dilate. 8 -r.. 100%,100%,1,3
-nm.. game_over -nm. game_overm

# Start game.
score=0 level=-1 lives=3 is_quit=0
-do

  # Build new level if necessary.
  -if {$level<0}
    _rlevel=33 _glevel=33 _blevel=255
    -_pacman_map_level{((-$level-1)%6)+1} mw={w} mh={h} mw2={int(w/2)} mh2={int(h/2)}
    -if {$level<-6} -replace. 3,2 -endif
    -nm. map0 -i[map] .

    # Precompute valid directions on each map point, and shortest path to the ghost's home.
    --shift[map] -1,0 --shift[map] 0,-1 --shift[map] 1,0 --shift[map] 0,1 -a[-4--1] z -!=. 1 -n
    --==[map] 1 100%,100% -=. 1,$mw2,$mh2 -distance. 1,.,.,3 -rm..
    -f. 'if(i==2,0,if(i==8,1,if(i==1,2,if(i==4,3,i))))' -nm. path
    --==[map] 2 pacdots={is} -rm.
    level={-$level}
  -endif
```

- Embedded math expression evaluator:

```
julia :
  1024,1024,1,1,"*
    z = 1.2*(2*[x/w,y/h] - 1);
    for (iter = 0, cabs(z)<=2 && iter<256, ++iter,
      z = z**z + [0.4,0.2]
    );
    iter
    "
  -n. 0,255 -map. 7
```

- **$ gmic user.gmic -julia** (0.631 seconds to run)



The G'MIC programming language

- Embedded math expression evaluator:

```
julia :
  1024,1024,1,1,"*
    z = 1.2*(2*[x/w,y/h] - 1);
    for (iter = 0, cabs(z)<=2 && iter<256, ++iter,
      z = z**z + [0.4,0.2]
    );
    iter
    "
  -n. 0,255 -map. 7
```

→ **JIT compiler :** expression is compiled by G'MIC into specific bytecode **for faster evaluation**.

→ **OpenMP :** expression is evaluated with **multiple threads** (when possible).

→ Manage usual calculations with **scalars, complexes, matrices** (`SVD, solve, ...`).

→ Expression may contain **variables, loops, conditions**, etc... (looks like C code).

- NL-means code :

```
nlmeans_expr : -check "${1=10}>0 && isint(${2=3}) && $2>0 && isint(${3=1}) && $3>0"
  -fill "
    const sigma = $1;  # Denoising strength.
    const hl = $2;     # Lookup half-size.
    const hp = $3;     # Patch half-size.
    value = 0;
    sum_weights = 0;
    for (q = -hl, q<=hl, ++q,
      for (p = -hl, p<=hl, ++p,
        diff = 0;
        for (s = -hp, s<=hp, ++s,
          for (r = -hp, r<=hp, ++r,
            diff += (i(x+p+r,y+q+s) - i(x+r,y+s))^2
          )
        );
        weight = exp(-diff/(2*sigma)^2);
        value += weight*i(x+p,y+q);
        sum_weights += weight
      )
    );
    value/(1e-5 + sum_weights)
  "
```

- Run with `$ gmic user.gmic leno.png -nlmeans_expr 35,3,1`

The G'MIC programming language

- `$ gmic user.gmic leno.png -nlmeans_expr 35,3,1`

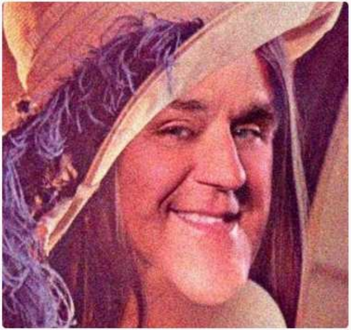


Fig.5.1. Crop of a noisy version of the Leno image, degraded with gaussian noise, `std=20`.

Fig.5.2. Denoised version using the NL-means algorithm (custom command `-nlmeans_expr`*).*

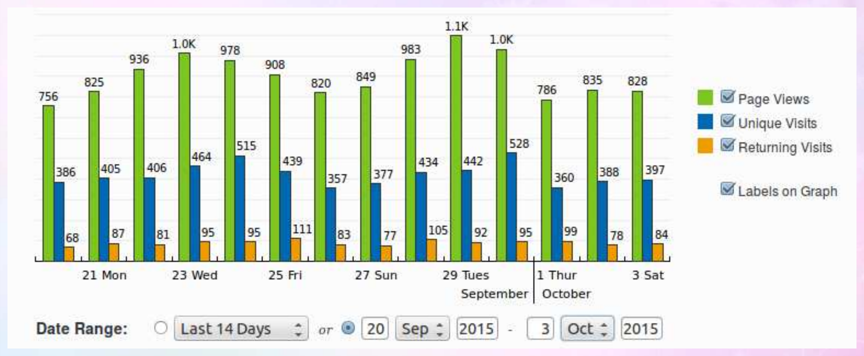
- Takes 3.156 seconds for a 512x512 RGB image, with 24 cores used.

→ Very convenient for quick algorithm prototyping.

Current project state

- Today, **G'MIC** is a project with:
  - Approx. 150.000 lines of code (without the code of interfaces) (in C++ and **G'MIC** language mainly).
  - 700+ downloads / day (more than. 3.000.000 since August 2008).
  - 400+ unique visitors / day on the project web pages.

  ⇒ Unexpected results considering the first targeted audience (researchers in image processing!).



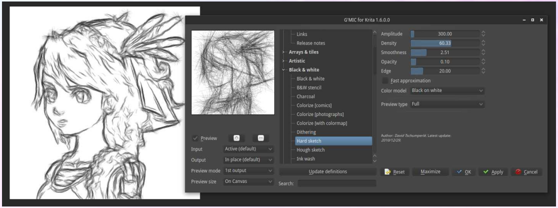
The open-source effect

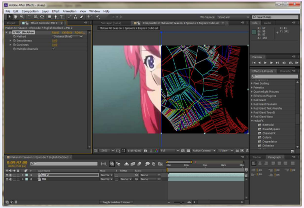
- **More users: G'MIC** becomes referenced on forums, blogs, news articles (about computer graphics or free software): `framasoft`, `linuxfr`, `webupd8`, `libregraphicsworld`, `pcastuces`, `gimpfr`, `linuxgraphics`, `gimpusers`, `...`).
- **More contributors:** Help from beta-testers, packagers (Debian, Ubuntu, Arch, Mageia, Gentoo, Windows, MacOSX,..), bug reports, language translations, design of mascots, new filters, ...

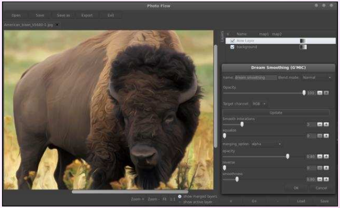
The open-source effect

- **G'MIC** has a great community of users: Flickr (+800 followers), Pixls.us, GimpChat, Twitter, Google+ (+2200 followers), ...

- Recent development of some **G'MIC** -based interfaces by external developers: Plug-ins for Krita, After Effects, Natron, PhotoFlow.







Why G'MIC does raise interest ?

Demonstration of some G'MIC features

(for artistic purposes)

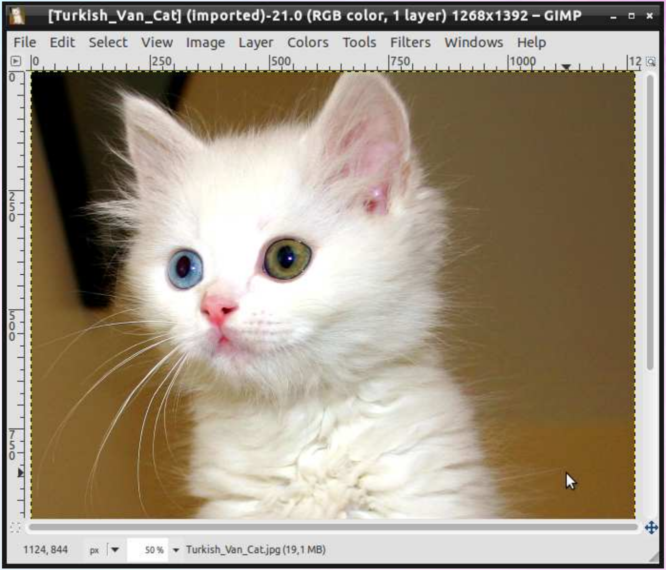
Filter Showcase:

Rodilius

GREYC cnrs

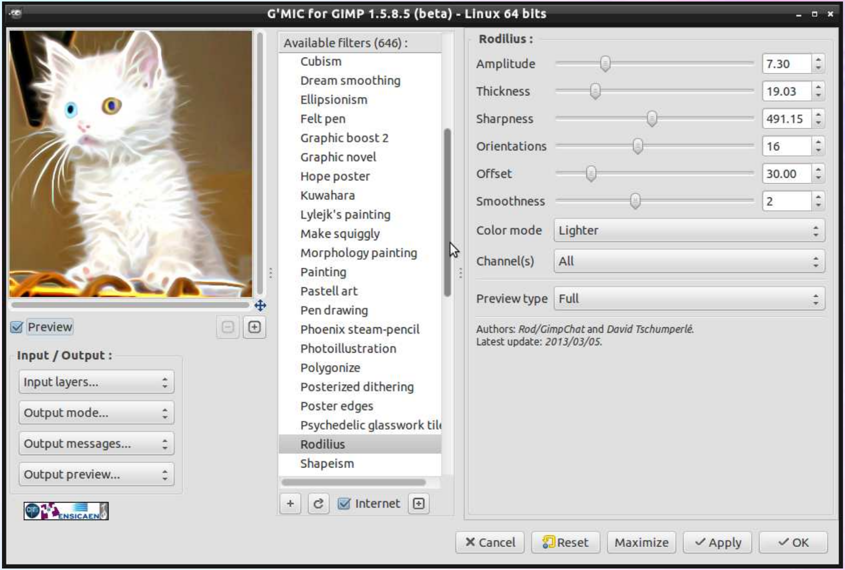
- **Goal:** Exaggerate the structure and length of the image contours to make them more visible.
- **Principle:** Several image convolution with oriented gaussian kernels are computed along different orientations of the plane. The resulting images are simply combined with layer blending modes Lighten only or Darken only. Finally, we smooth the blended image anisotropically, then sharpen its contours.
- **Similar to:** Filter banks for the geometric analysis of images (contourlets).

Artistic: **Rodilius**



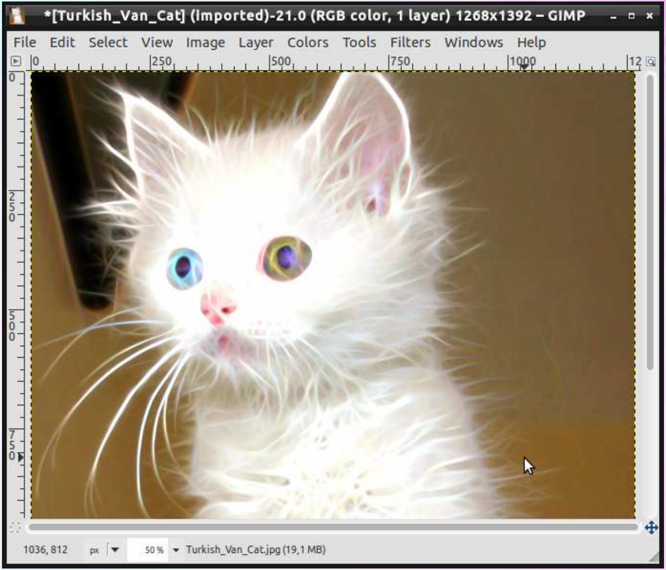
Open input image.

Artistic: **Rodilius**



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

Artistic: **Rodilius**



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

Artistic: **Rodilius**

Two other examples, works quite well on fur.

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

Artistic: **Rodilius**

3. Reproduces the 'Fractalius' effect (49$ plug-in for Photoshop) but for 0$ and 10 lines of **G'MIC** code !):

Redfield Fractalius

G'MIC Rodilius

Rodilius code in G'MIC: 10 lines

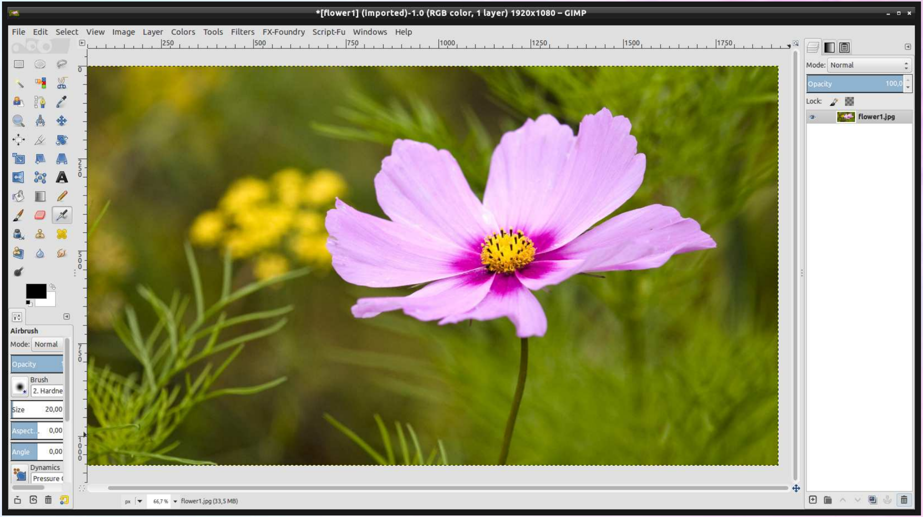
```
rodilius : -check "${1=10}>=0 && $1<=200 && ${2=10}>=0 && $2<=100 && ${3=400}>=0 && ${4=7}>0" -skip ${5=0},${6=1}
  -v - -repeat $! -l[$>] -split_opacity -rv
    -if {!$6} -negative. -endif
    --f. 0 -nm. {-2,n}
    -repeat {round($4)}
      angle={$5+$>*180/round($4)}
      --blur_linear.. $1%,{$1*$2/100}%,$angle,1 -b. 0.7 -sharpen. $3 -max[-2,-1]
    -done -rm..
    -if {!$6} -negative. -endif
  -rv -a c -endl -done -v +
```

Filter Showcase:

`Color transfer`

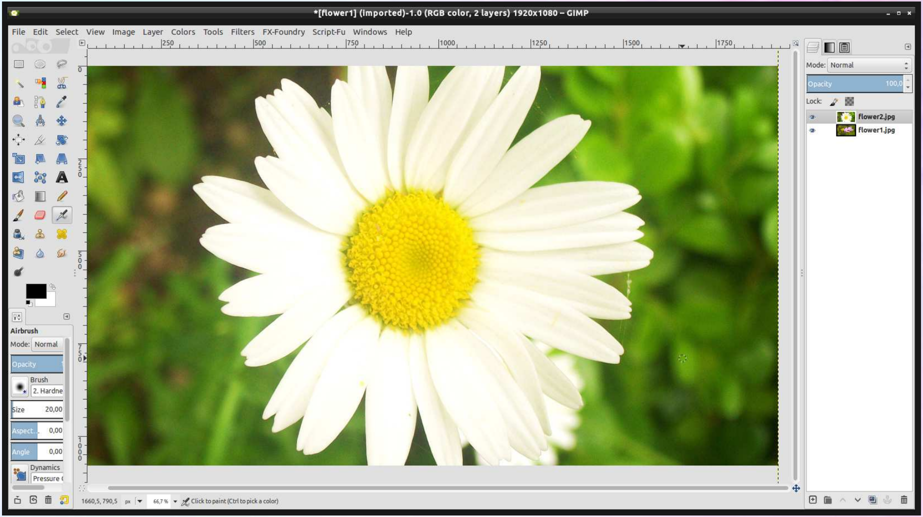
- **Goal:** Give a color ambiance to an image, from a reference image.
- **Principle:** We register two colorimetric functions in the RGB cube to determine a color correspondence map to apply to the input image.
- **Similar to:** Optical flow, image registration.

Colors: **Color transfer**



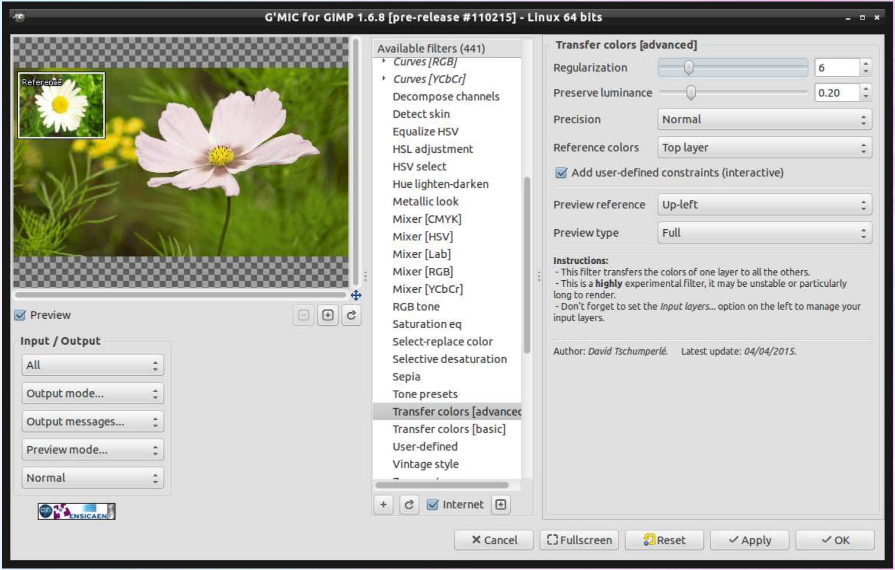
Open input image.

Colors: **Color transfer**



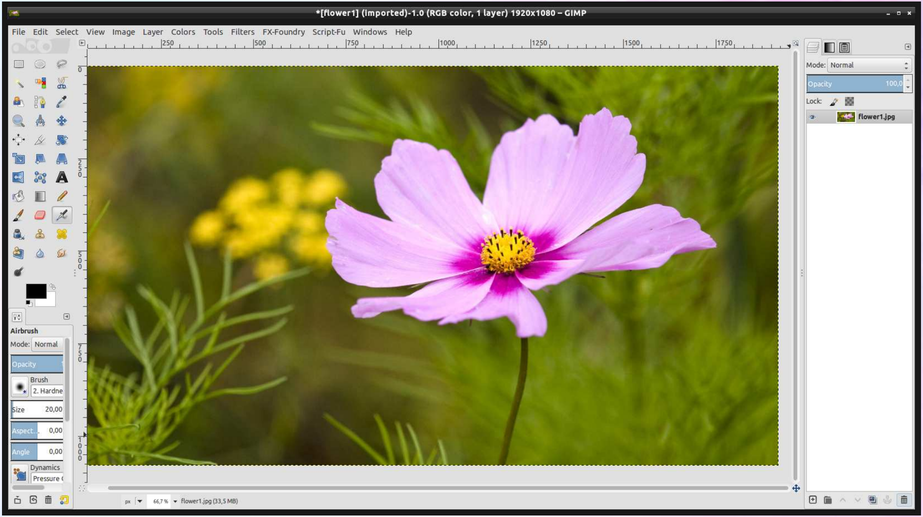
Open reference image (as a new layer).

Colors: **Color transfer**



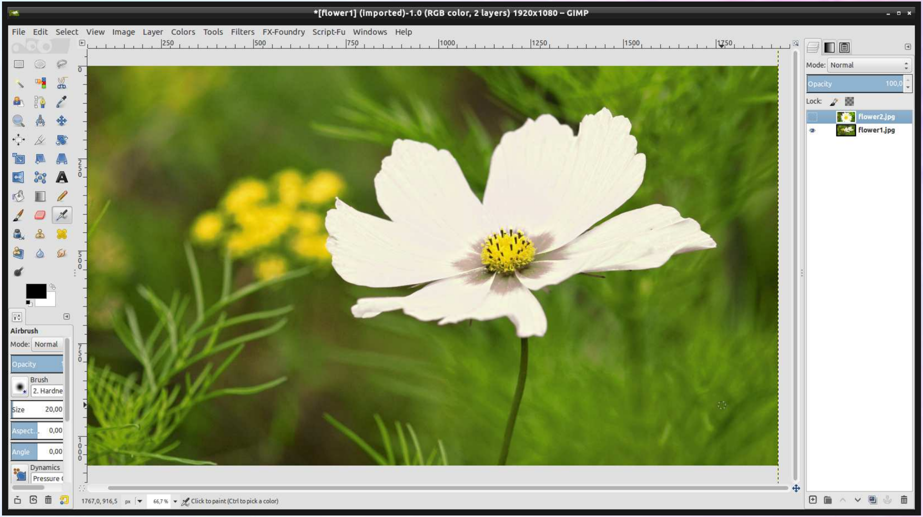
Invoke **G'MIC** plug-in and select **Colors** / **Transfer color [advanced]**.

Colors: **Color transfer**

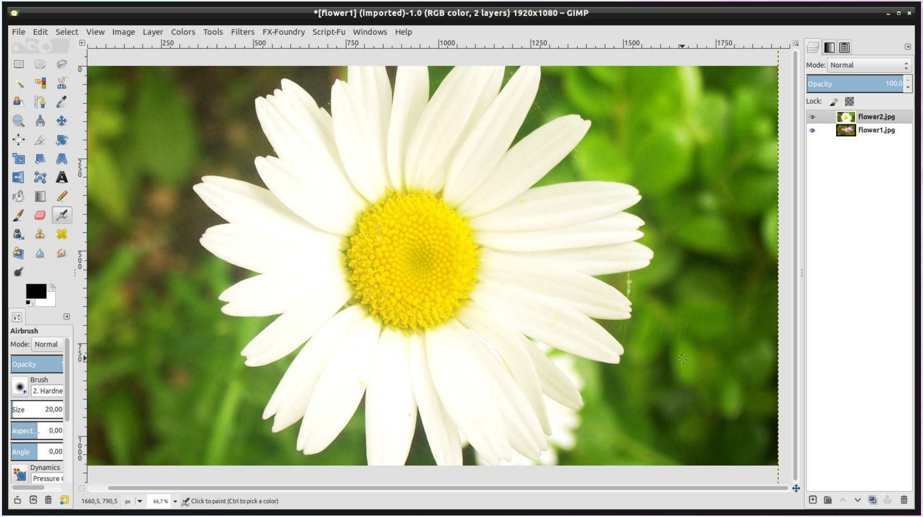


Original image.

Colors: **Color transfer**



Color-transferred result.



Reference image (reminder).

Colors: **Color transfer**

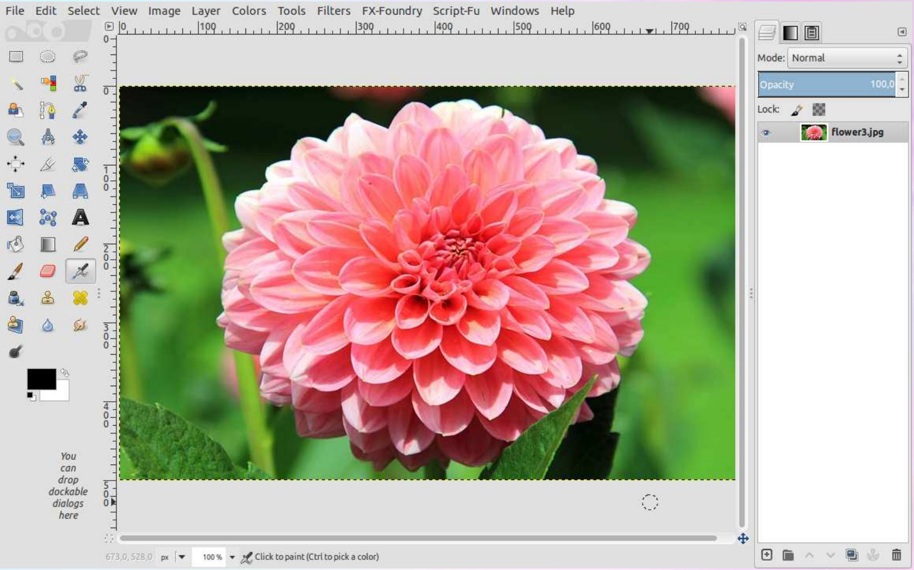
Other examples.

Filter Showcase:

`Extract foreground [interactive]`

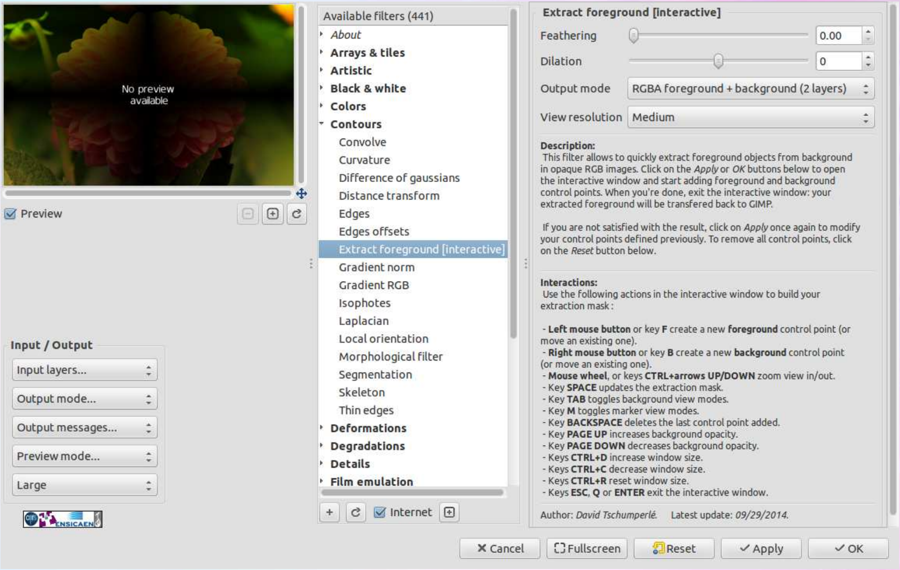
- **Goal:** Extract foreground objects from background in an image, and get the result as two distinct (complementary) layers.
- **Principle:** Same as before, but done only with key points having labels "foreground" or "background" instead of colors.

Contours: **Extract foreground [interactive]**



Open input image (single-layer color photograph).

Contours: **Extract foreground [interactive]**

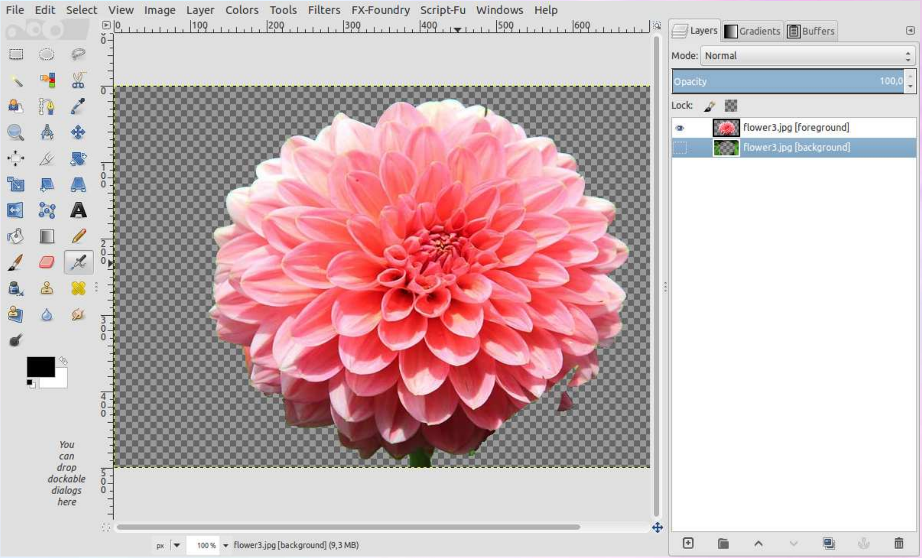


Invoke **G'MIC** plug-in and select **Contours** / **Extract foreground [interactive]**.

Contours: **Extract foreground [interactive]**

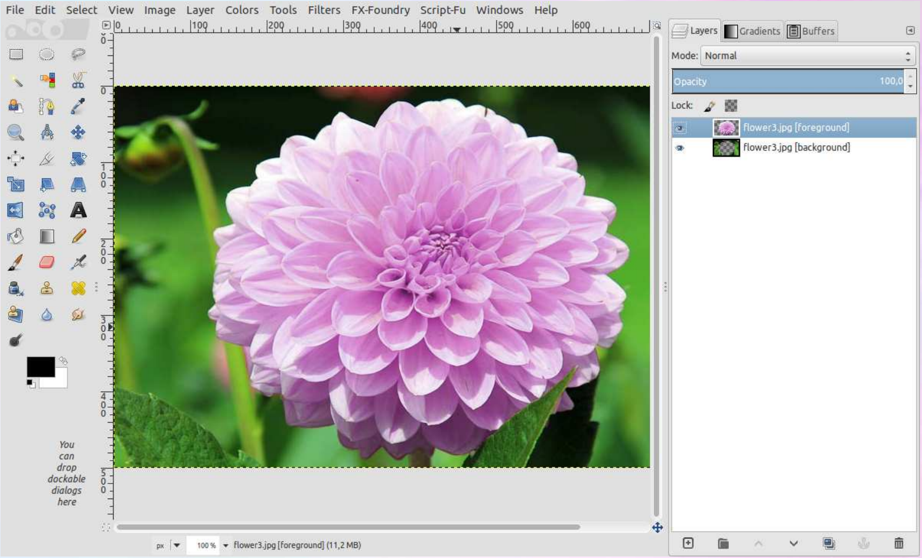
Place some “foreground” and “background” key points.

Contours: **Extract foreground [interactive]**



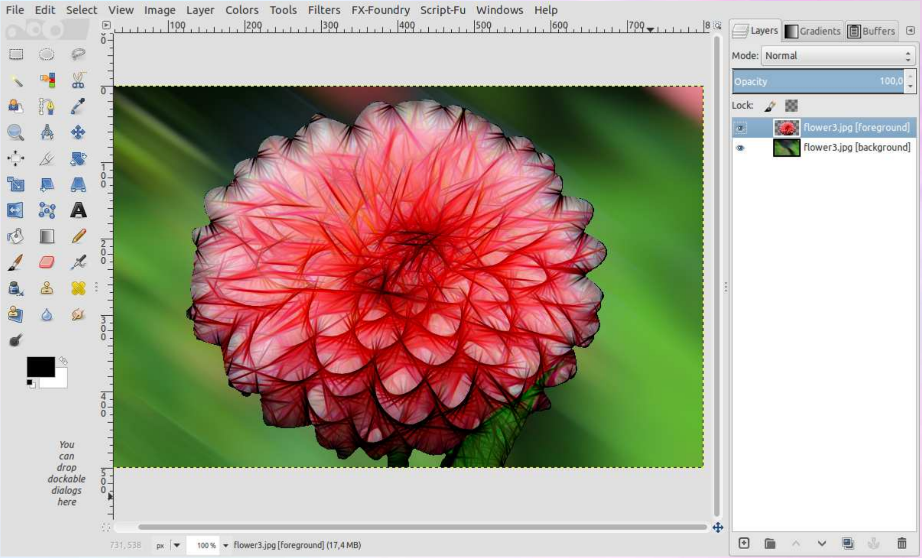
Result of the filter: 2 layers (foreground shown here).

Contours: **Extract foreground [interactive]**



Result of the filter: Image after modification of the color hue on foreground layer only.

Contours: **Extract foreground [interactive]**



Another example of result, processing background and foreground independently.

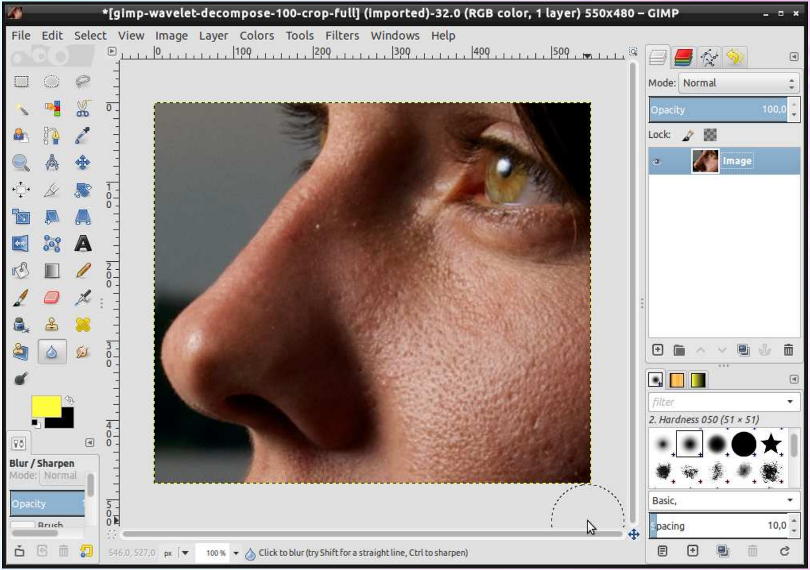
Filter Showcase:

Split details

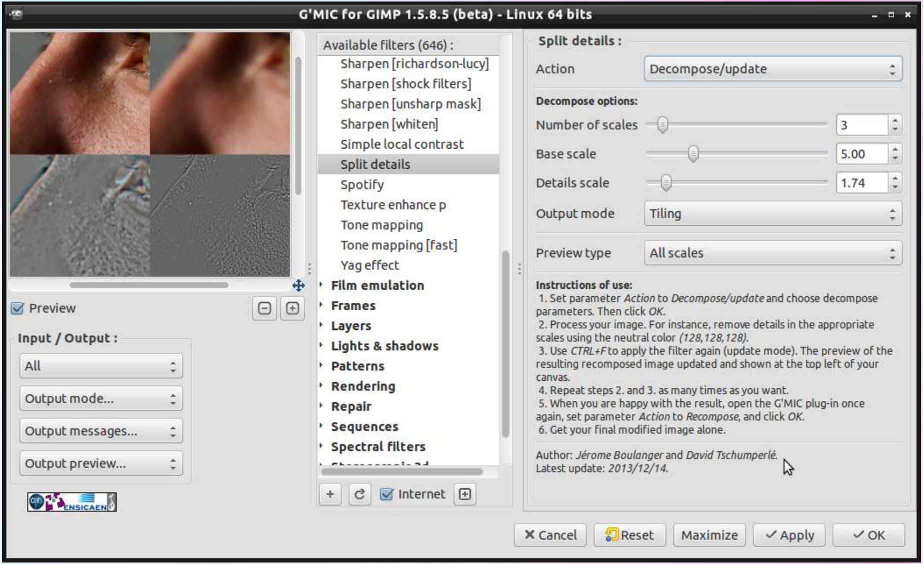
Details: **Split details**

- **Goal:** Decompose an image into several levels of details, in order to work separately on the different image scales before recomposing the image.
- **Principle:** The image is decomposed/recomposed using a pyramidal representation obtained by the iterative convolution by gaussians kernels + residues.
- **Similar to:** Scale space analysis.

Details: **Split details**

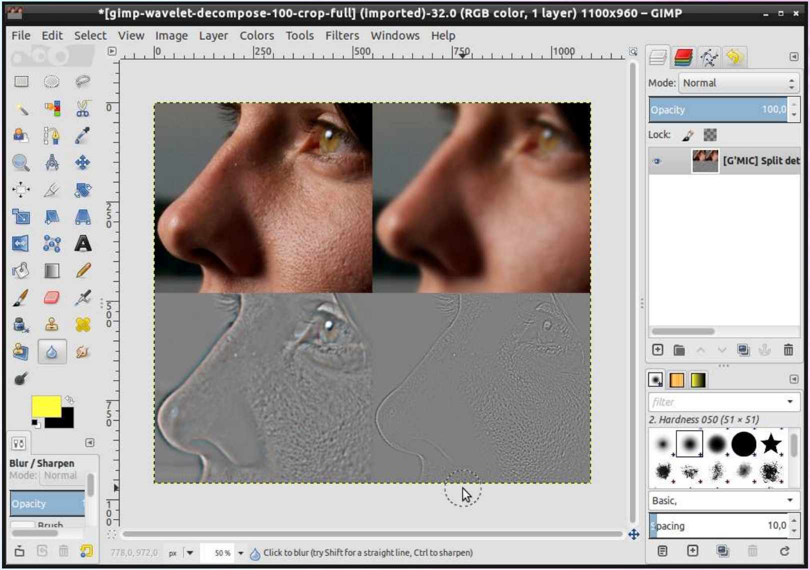


Open input image.



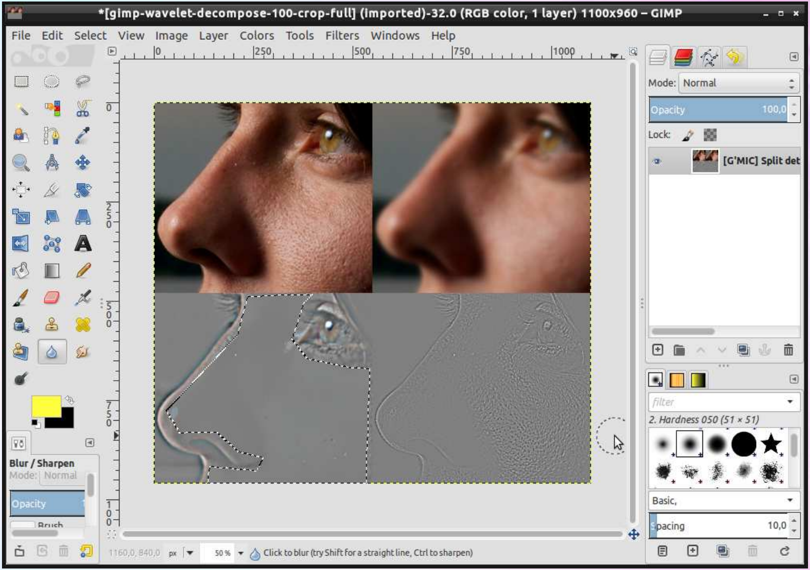
Invoke **G'MIC** plug-in and select **Details** / **Split details**.

Details: **Split details**



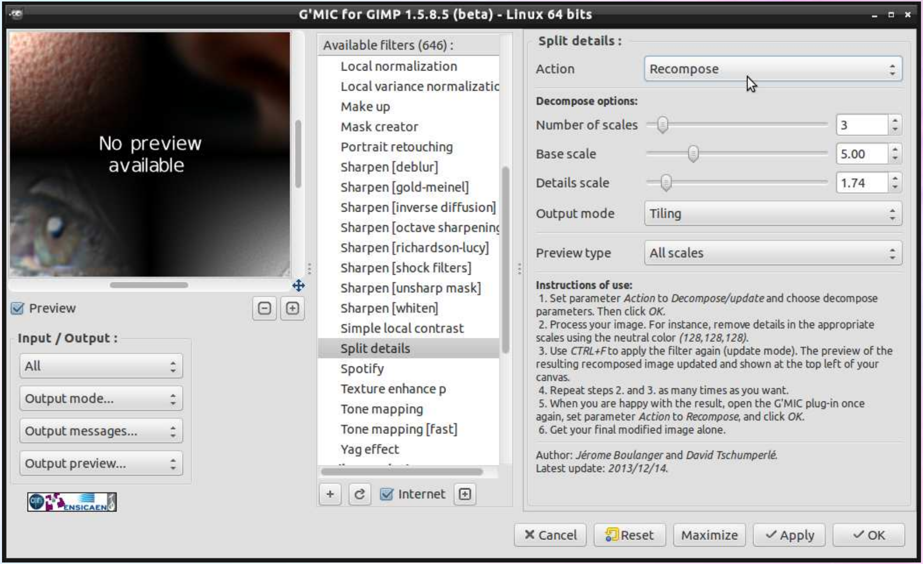
You get your input (top-left) + the decomposition into scales (here 3 scales).

Details: **Split details**



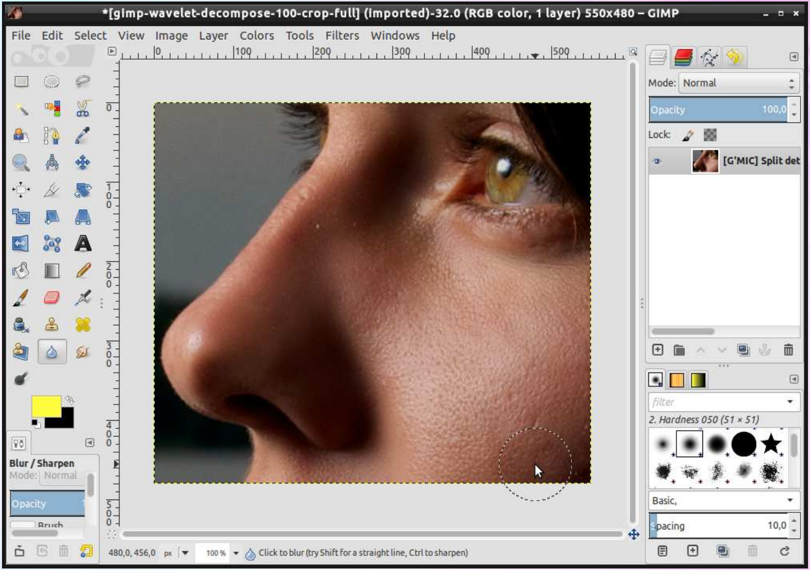
Do what you want on the scales (here, we simply erase the skin defects on the middle scale).

Details: **Split details**



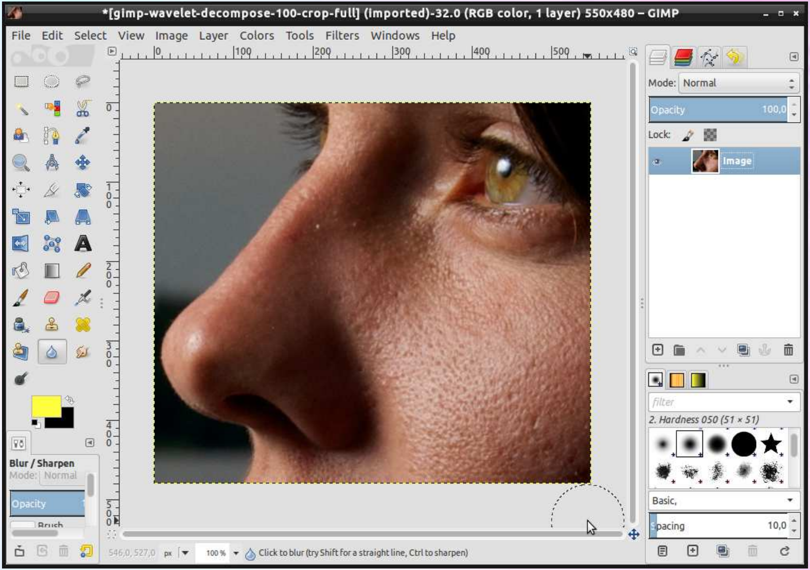
Invoke **G'MIC** plug-in again, to recompose the final image.

Details: **Split details**



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

Details: **Split details**



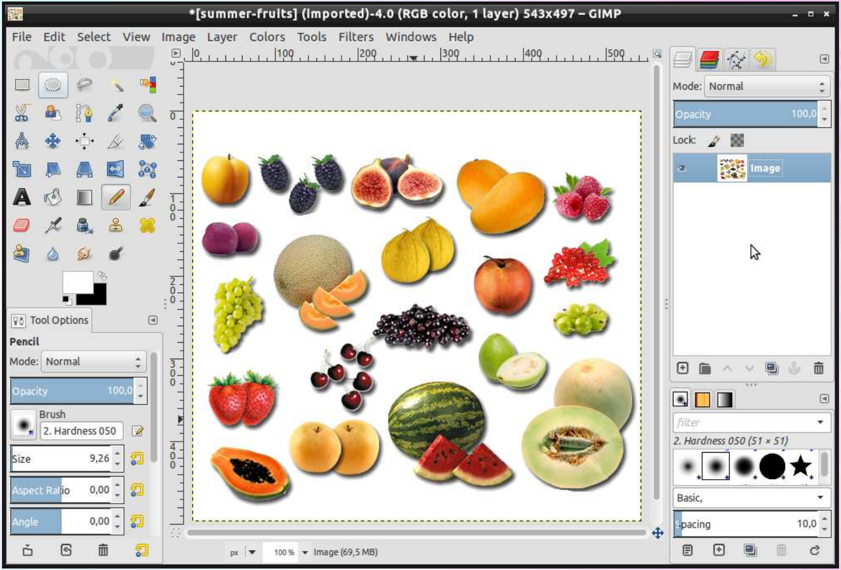
Comparison with initial image.

Filter Showcase:

`Extract objects`

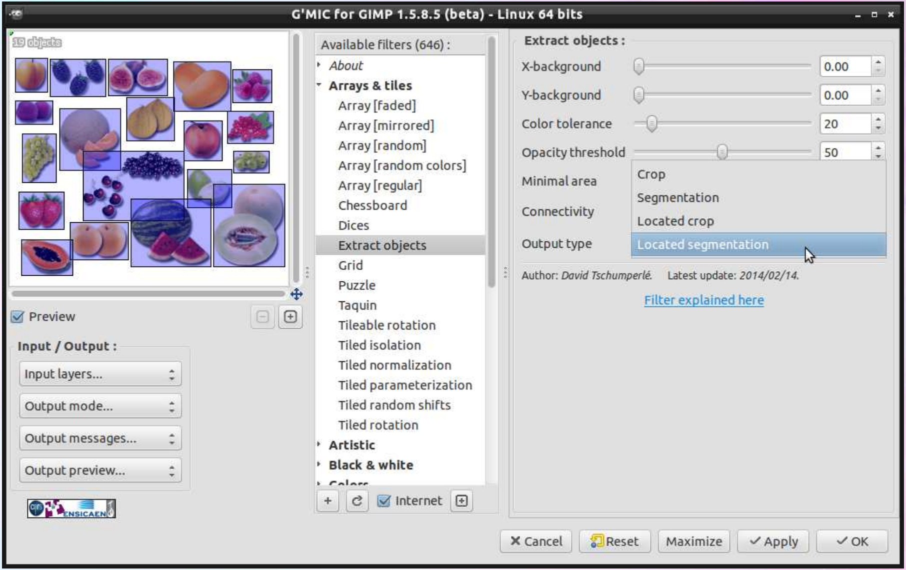
- **Goal:** Extract independent objects localized on a single image.
- **Principle:** Background detection is performed, then residual pixels are grouped as several connexe regions.
- **Similar to:** Object detection and segmentation on static background.

Arrays & Tiles: **Extract objects**



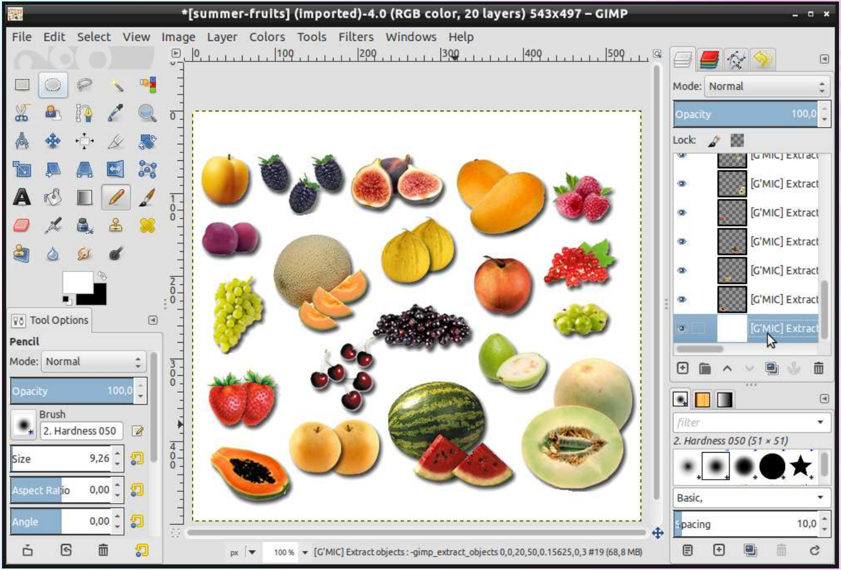
Open input image (single-layer).

Arrays & Tiles: **Extract objects**



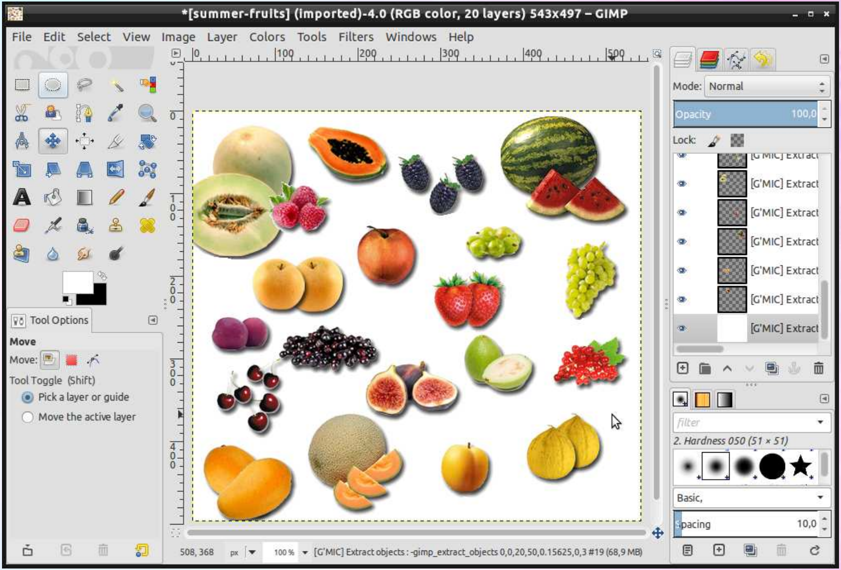
Invoke **G'MIC** plug-in and select **Arrays & tiles** / **Extract object**.

Arrays & Tiles: **Extract objects**



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

Arrays & Tiles: **Extract objects**



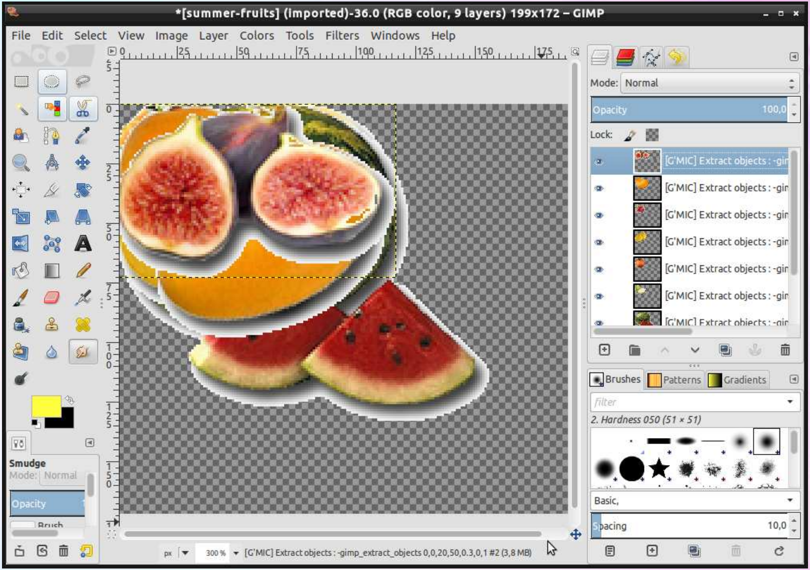
Managing each object independently is now possible (here, position change).

Filter Showcase:

`Pack sprites`

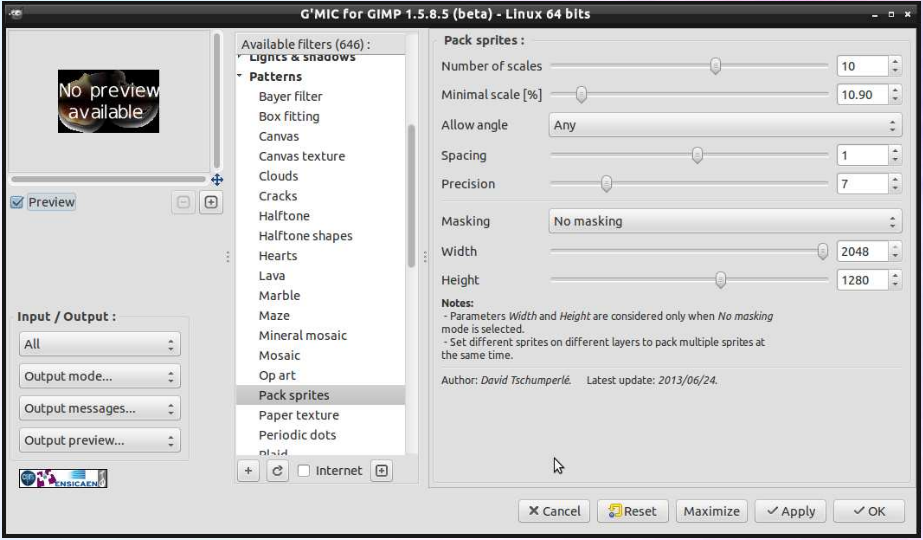
- **Goal:** Generate a synthetic image where multiple image thumbnails have been packed together without overlapping (resized and rotated).
- **Principle:** Valid pseudo-random positions are iteratively checked for the insertion of new objects, with decreasing dimensions.
- **Similar to:** Bin-packing problem (NP-hard).

Patterns: **Pack sprites**



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

Patterns: **Pack sprites**

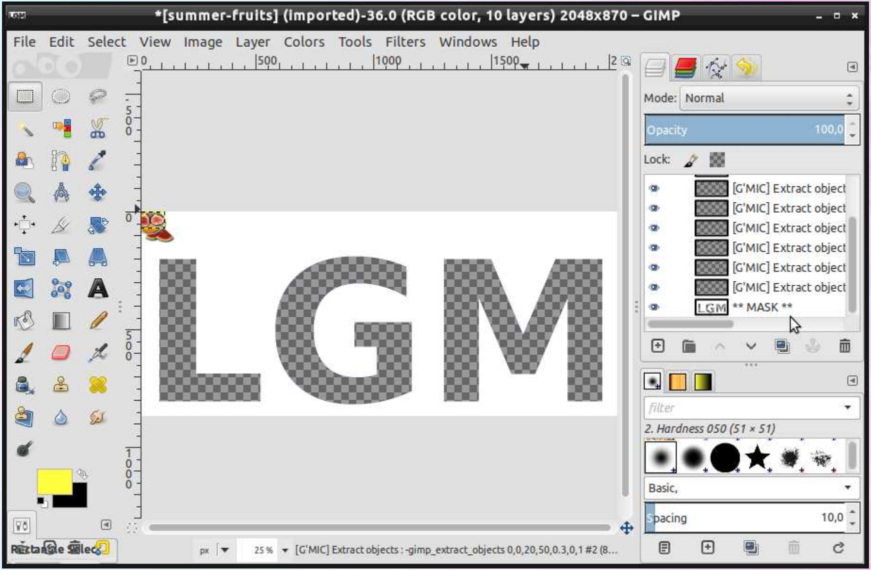


Invoke **G'MIC** plug-in and select **Patterns** / **Pack sprites**.

Patterns: **Pack sprites**

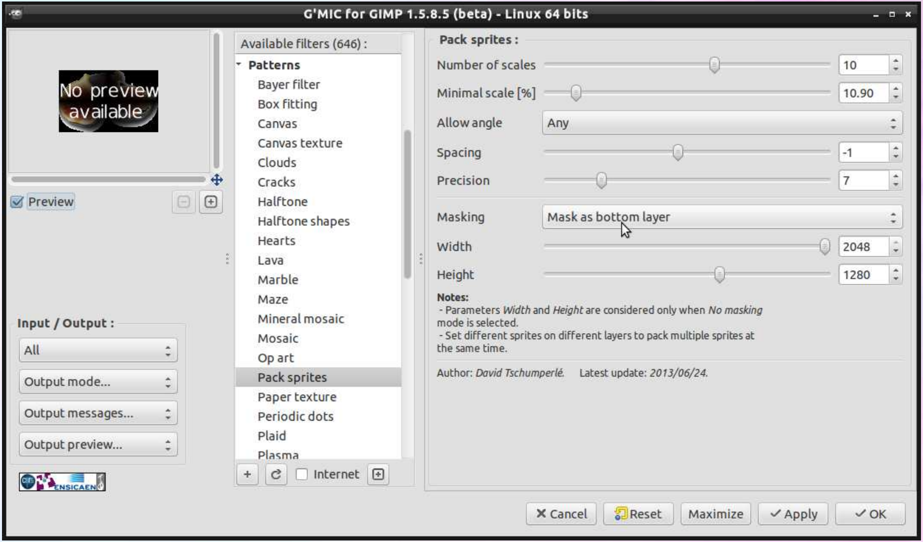
Get your image with randomly packed sprites (after a while).

Patterns: **Pack sprites**



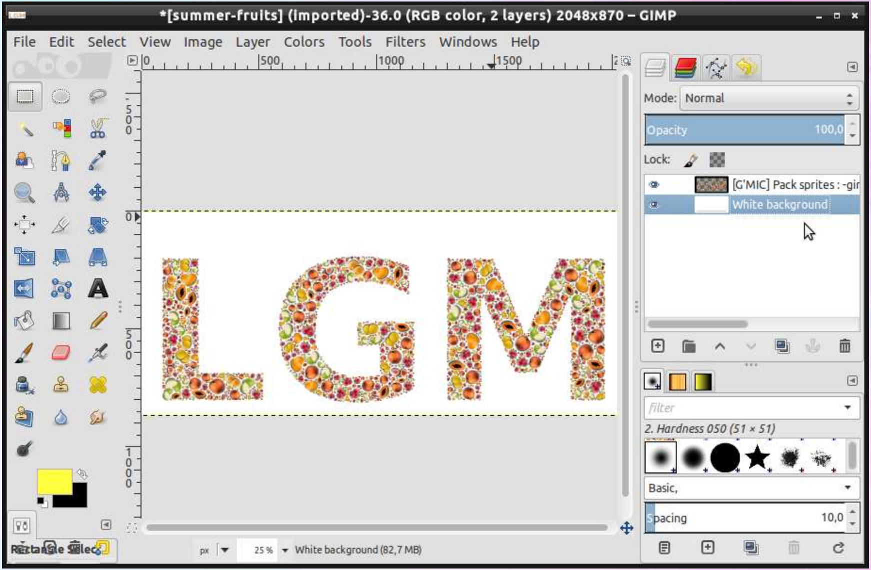
Now, you can add a bottom layer to restrict packing on transparent regions.

Patterns: **Pack sprites**



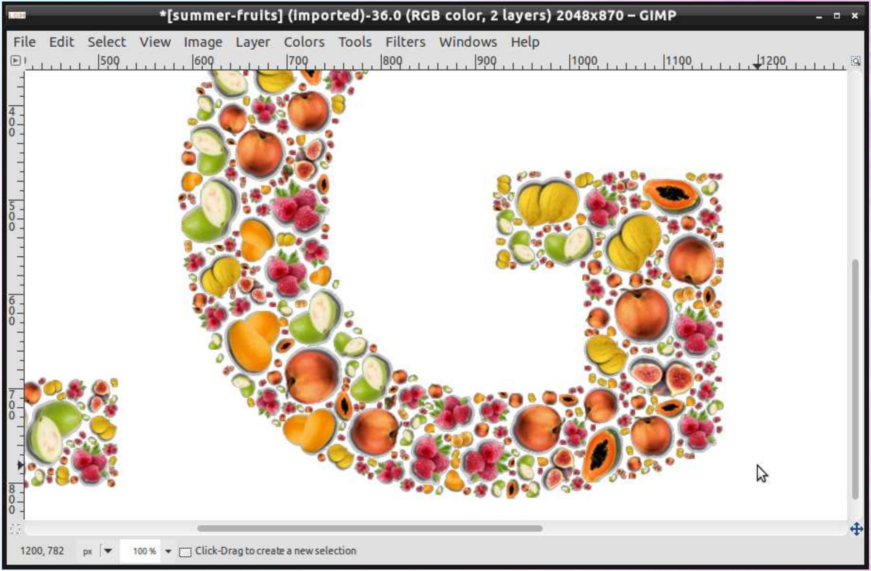
Invoke **G'MIC** again, and select **Mask: Mask as bottom layer**.

Patterns: **Pack sprites**



Go for a coffee, and you get this.

Patterns: **Pack sprites**



Detail of the result.

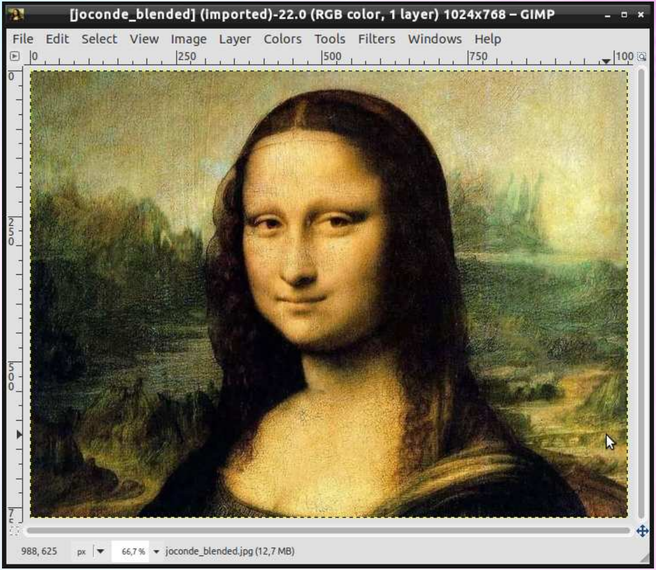
Example of rendering, by **Chris Fiedler**, on GimpChat.

Filter Showcase:

`Shapeism`

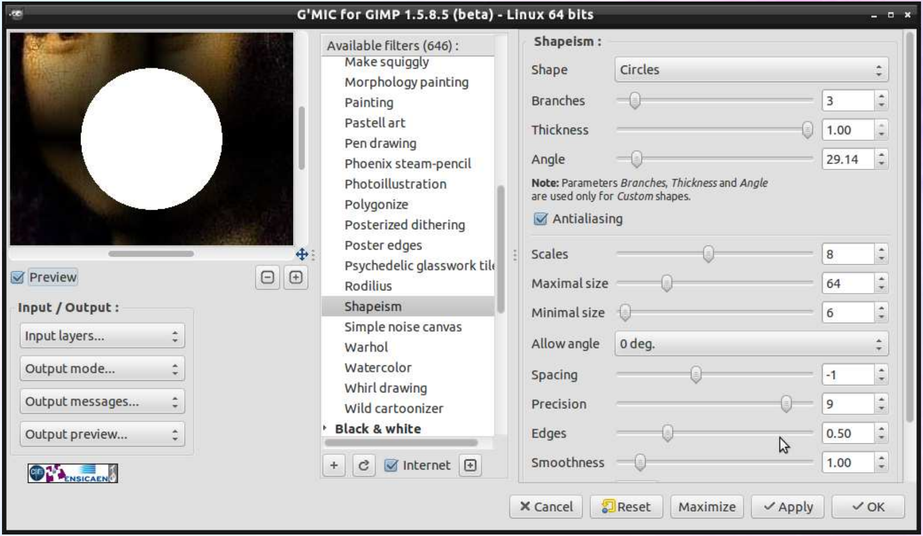
- **Goal:** Use the previous Sprite packing filter to create image abstraction (such as the Circlism from artist Ben Heine).
- **Principle:** Monochrome shapes are packed together at different scales, with constraints to put only small shapes on image contours. Shape colorization is performed afterwards by averaging the color pixels covered by each shape.

Artistic: **Shapeism**



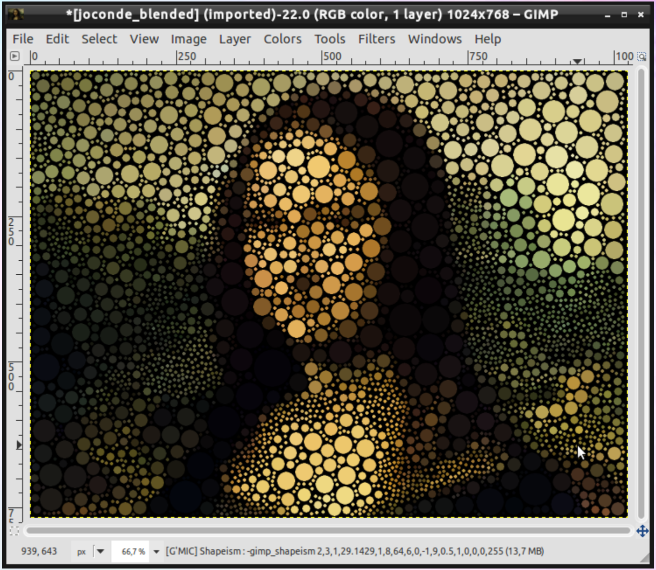
Open input image.

Artistic: **Shapeism**



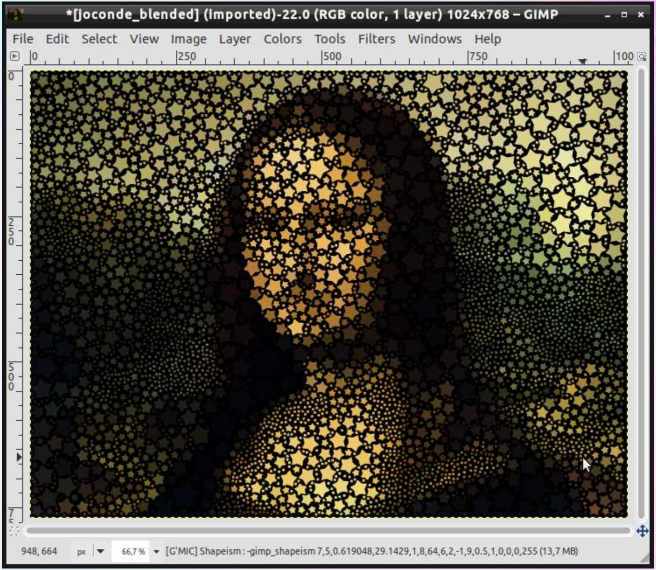
Invoke **G'MIC** plug-in and select **Artistic** / **Shapeism**.

Artistic: **Shapeism**



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

Artistic: **Shapeism**



Result with another shape selected (a star).

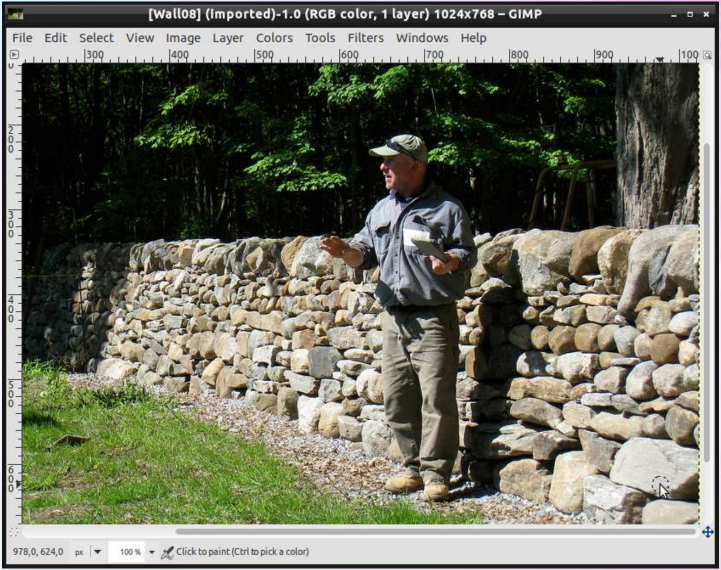
Filter Showcase:

`Inpainting [patch-based]`

Repair: **Inpaint [patch-based]**

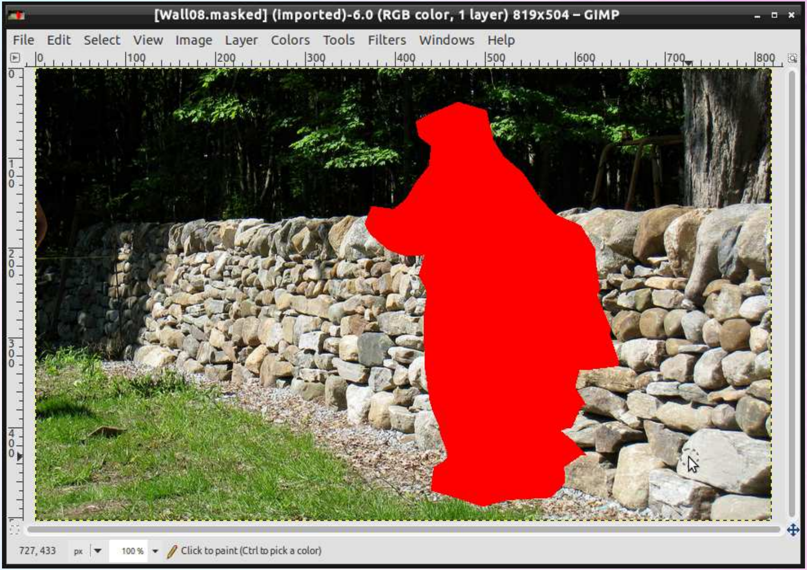
- **Goal:** Allow the reconstruction of "holes" in images (groups of pixels considered as missing or invalid).
- **Principle:** Implementation of an extension to the inpaiting algorithm of Criminisi-Perez-etal + patch blending technique.
- **Similar to:** Inpainting, "classical" (and hard-to-solve!) reconstruction problem in image processing.

Repair: **Inpainting**



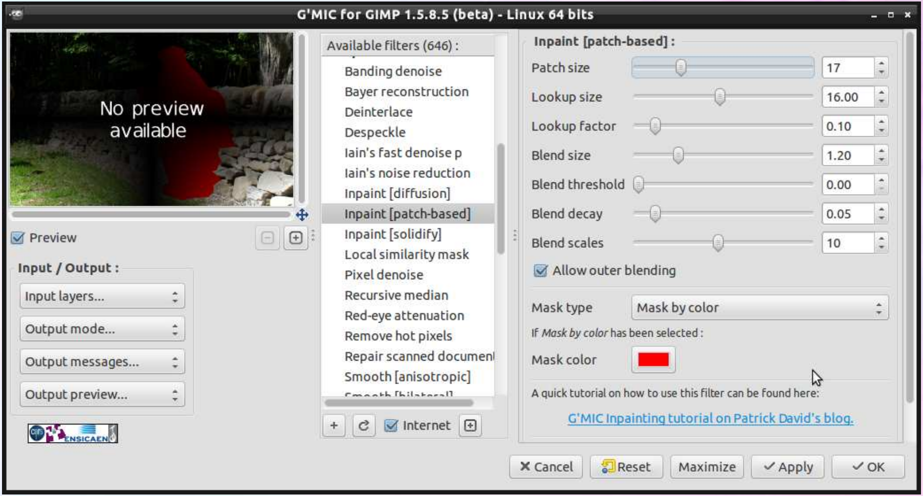
Open input image.

Repair: **Inpainting**



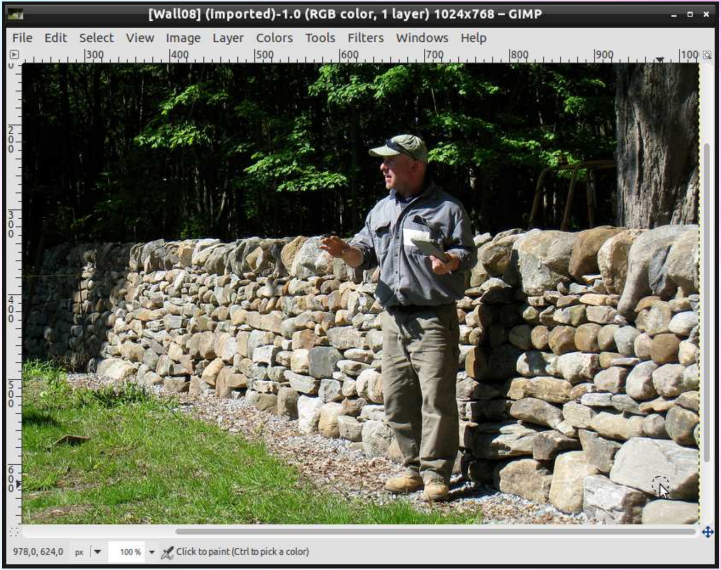
Draw an inpainting mask directly on it (with a constant known color).

Repair: **Inpainting**

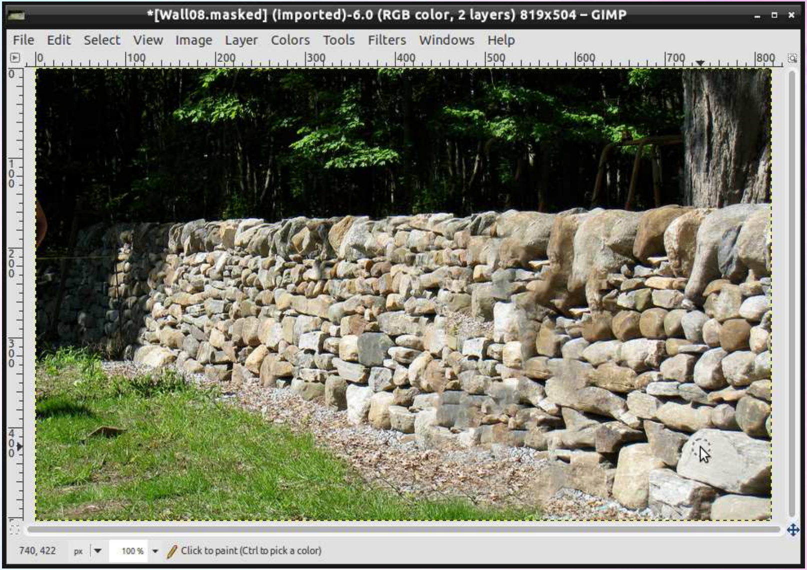


Invoke **G'MIC** plug-in and select **Repair** / **Inpaint [patch-based]**.

Repair: **Inpainting**



Input image.



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

- **G'MIC** is one of the few software to offer several "inpainting" algorithms:

- **G'MIC** is one of the few software to offer several "inpainting" algorithms:

- **G'MIC** is one of the few software to offer several "inpainting" algorithms:

- **G'MIC** is one of the few software to offer several "inpainting" algorithms:

Repair: **Inpainting**

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

Repair: **Inpainting**

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

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

Repair: **Inpainting**

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

Repair: **Inpainting**

(Extreme case!):
Input image (boat to be removed).

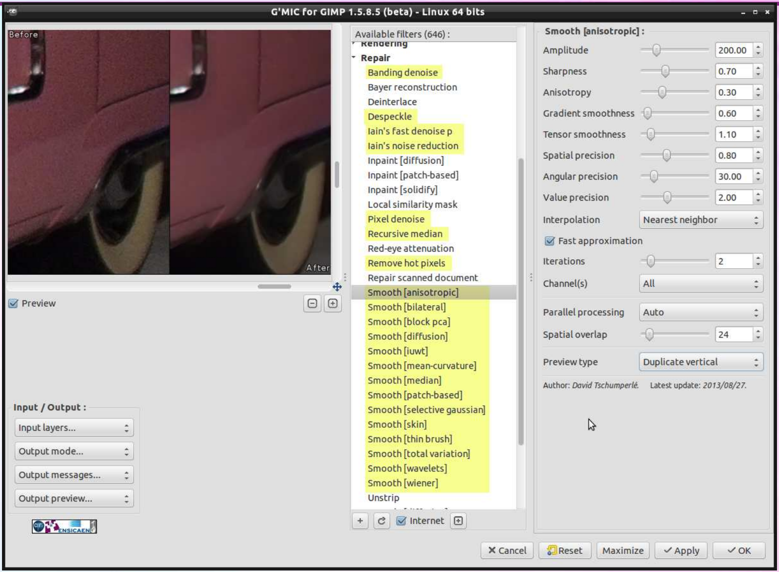
Result by the **G'MIC** inpainting algorithm.

Filter Showcase:

`Denoising filters`

- **Goal:** Algorithms to smooth an image while preserving the image details and textures.
- **Principle:** Reccuring issue in image processing, with a lot of algorithms existing (PDE's, Wavelets, Patch-based smoothing, etc...).

Repair: **Denoising filters**



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

Repair: **Denoising filters**

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

- **G'MIC** is one of the few software to offer efficient image denoising algorithms:

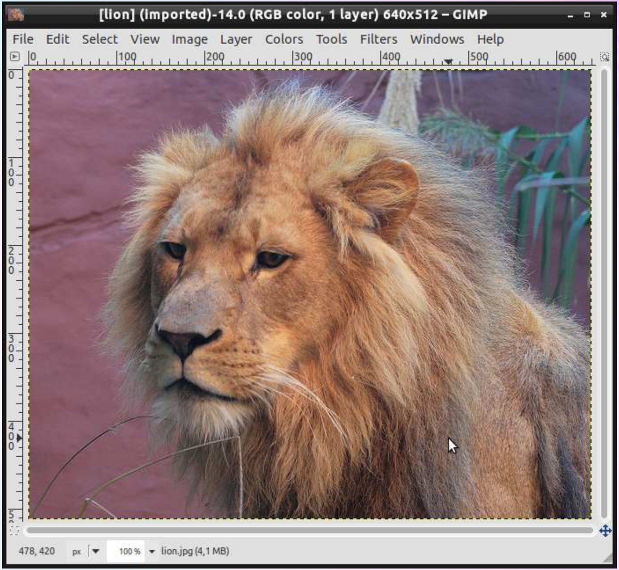
- **G'MIC** is one of the few software to offer efficient image denoising algorithms:

Filter Showcase:

`Dream smoothing`

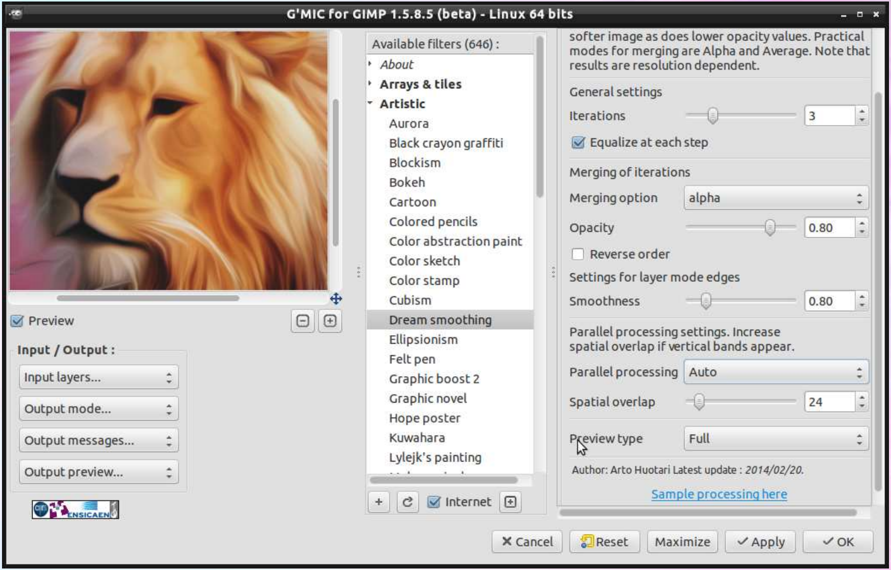
Artistic: **Dream smoothing**

- **Goal:** Apply one of the previous image smoothing technique, deliberatly exaggerated and make the colors more contrasted to create a painting effect.
- **Principle:** We apply multiple iterations of anisotropic smoothing with an "aggressive" color mix in the Lab color space.



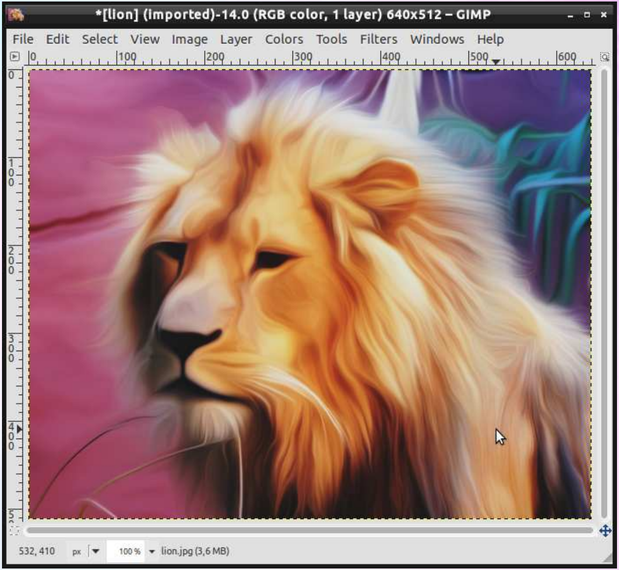
Open input image.

Artistic: **Dream smoothing**



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

Artistic: **Dream smoothing**



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

Artistic: **Dream smoothing**

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

Artistic: **Dream smoothing**

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

Filter Showcase:

`Poisson editing`

Poisson editing

- **Goal:** Copy/paste a piece of image into another one, without visible seams in the result.
- **Principle:** Solving the Poisson equation to reconstruct the final image from the gradient map where the paste has been done.

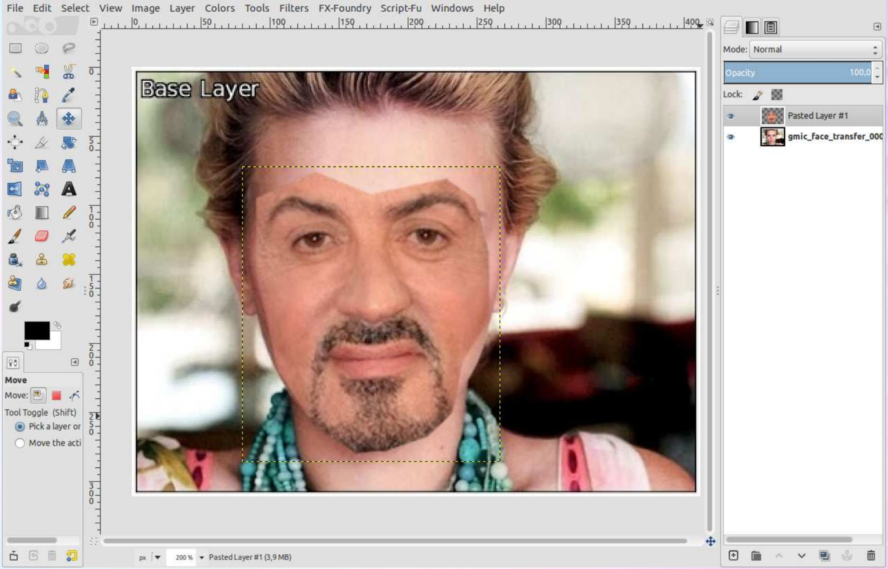
Poisson editing

Example of face swapping, using Poisson editing.

Poisson editing

Example of face swapping, using Poisson editing.

Poisson editing



Example of face swapping, using Poisson editing.

Example of face swapping, using Poisson editing.

Poisson editing

Example of object insertion, using Poisson editing.

Poisson editing

Example of face swapping, using Poisson editing.

Poisson editing

Example of face swapping, using Poisson editing (on the same input picture).

Poisson editing

Example of face swapping, using Poisson editing (on the same input picture).

Filter Showcase:

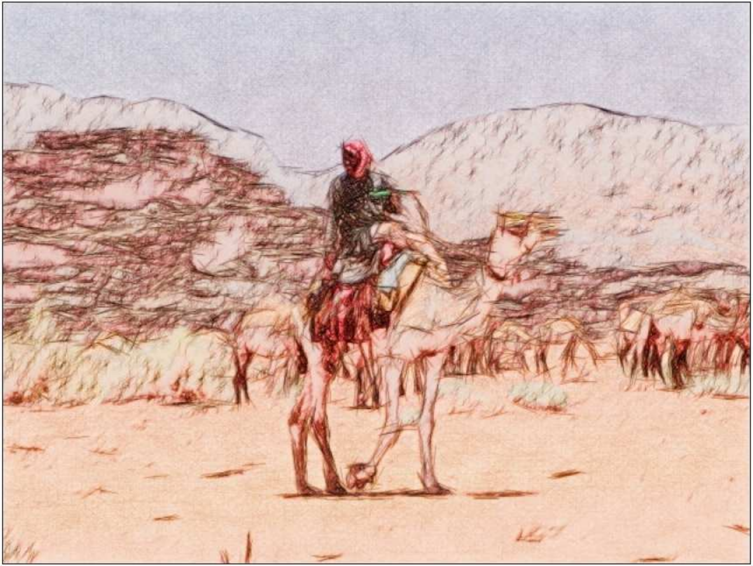
Sketch

- **Goal:** Algorithms to transform a picture into a sketch.
- **Principle:** Pencil strokes are iteratively simulated on a white canvas, by analyzing the contour geometry of the original picture.
- **Similar to:** Contour detection and extraction, texture analysis.

Black & White: **Sketch**

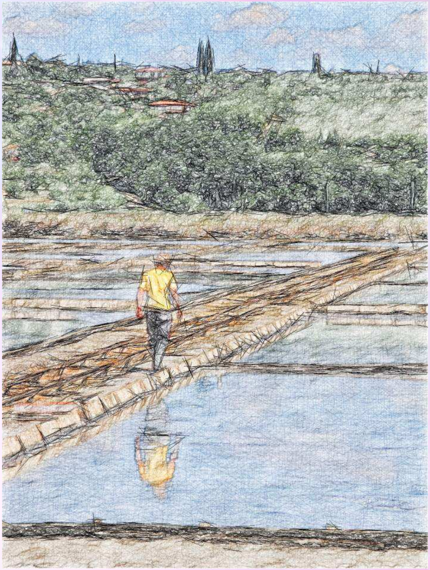
(Courtesy of Tom Keil)

Black & White: **Sketch**



(Courtesy of Tom Keil)

Black & White: **Sketch**



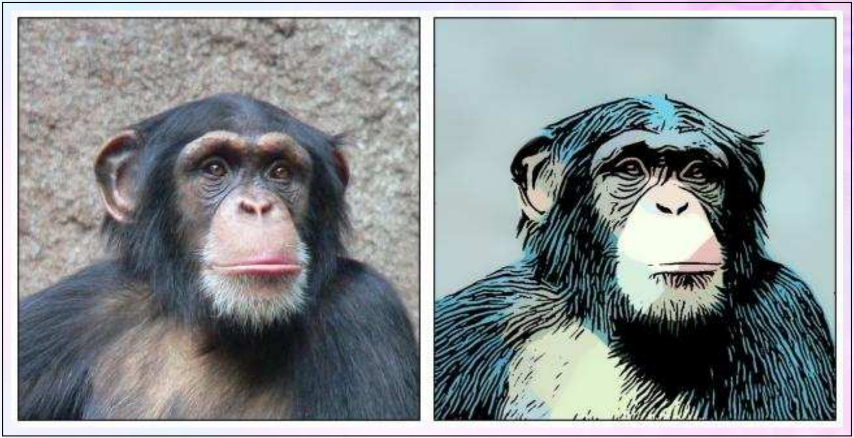
(Courtesy of Tom Keil)

Black & White: **Sketch**

Black & White: **Sketch**

(Courtesy of Tom Keil)

Black & White: **Engrave**



(Example of another similar filter in **G'MIC** : **Engrave**)

⇒ +440 filters like this available in the **G'MIC** plug-in for GIMP!

Filter Showcase:

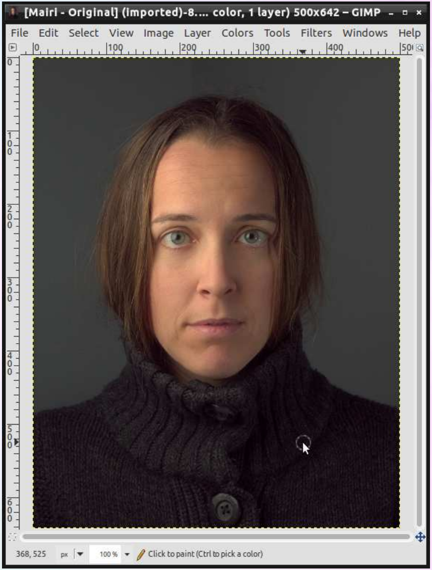
`Film emulation`

Repair: **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.

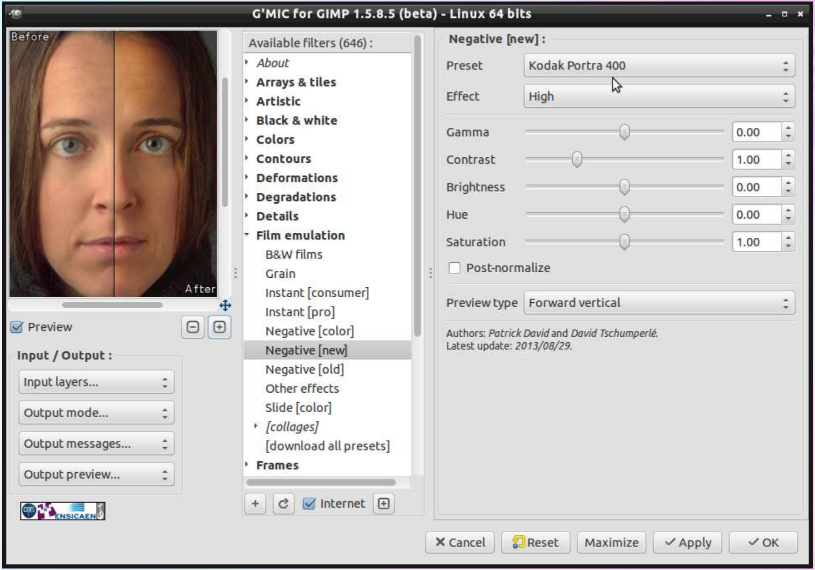
$\Longrightarrow$ **476** lines of **G'MIC** code (mostly for GUI).
(all included: GUI description + algorithm).

Film emulation



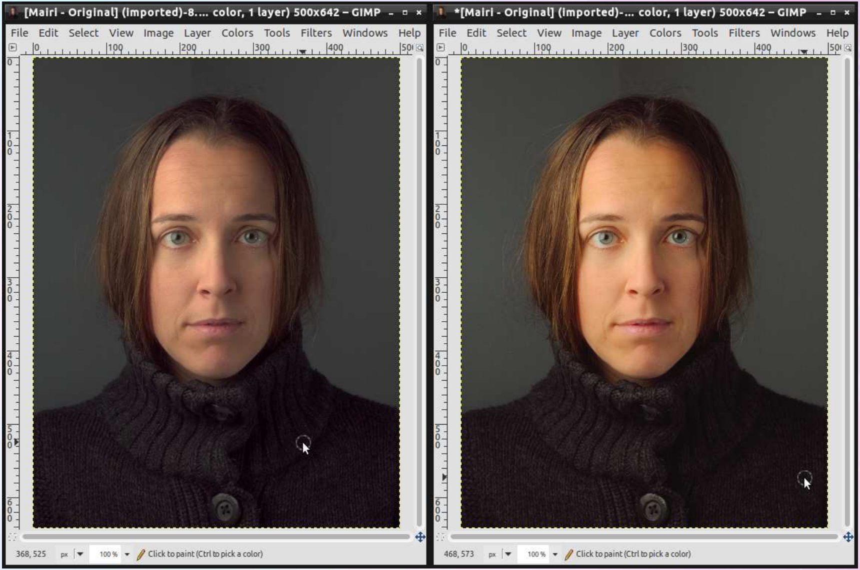
Open input image.

Film emulation



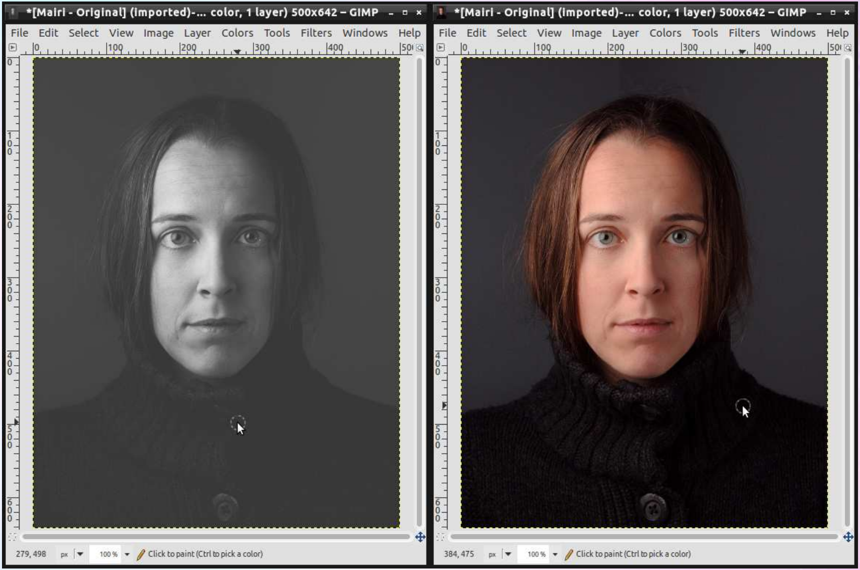
Invoke **G'MIC** plug-in, and choose one filter in folder **Film emulation**/.

Film emulation



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

Film emulation

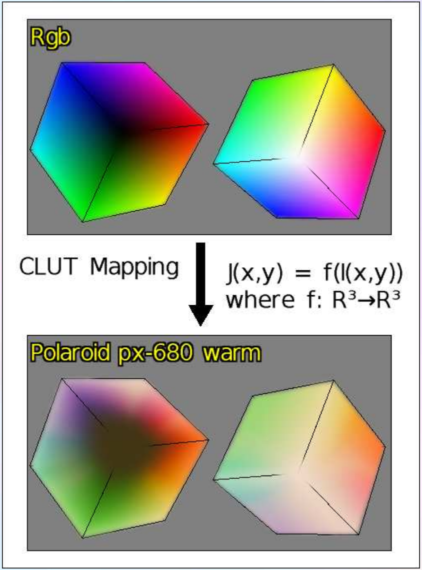


Two other examples: TMAX-3200 (*left*) and Kodak Kodachrome 64 (*right*).

Film emulation

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

Film emulation



Technically speaking:

- Each preset defines a mapping function from RGB to RGB (CLUT).
- The values of these functions are explicitely stored for all RGB colors.
- To avoid huge datasets, we consider 64*x*64*x*64 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.

G'MIC [`http://gmic.eu`]

- A full-featured open-source framework for image processing: Several user interfaces available, more to come.

☺ Positive for the GREYC: Evolving software.
"Image Processing" showcase for the general public.

☺ Positive for the IMAGE team: Useful software on a daily basis, for the analysis and exploration of image data, and the fast prototyping of new algorithms + derived publications.

☹ Time consuming: developing/maintenance, community animation, web pages, answering questions... (approx. 10-15h of work / week).