FROM THE DEVELOPMENT OF OPEN-SOURCE SOFTWARE TO THE DESIGN OF ALGORITHMS FOR ARTISTIC PRODUCTION

D. Tschumperlé

Joint work with P. David, S. Fourey, T. Keil, A. Mahboubi, C. Porquet, D. Revoy, ... and many others!

Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, Caen, FRANCE

https://tschumperle.users.greyc.fr
Research in the field of image processing at the GREYC lab of ENSICAEN / CNRS / University of Normandy (Caen) ⇒ IMAGE team.

⇒ We try to design innovative algorithms to solve generic image processing problems (denoising, enhancement, segmentation, feature detection,...).
Context: Collaborations

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

⇒ Various image data coming from very diverse sensors.
Context: Image Data

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

(a) $I_1 : W \times H \rightarrow [0, 255]^3$
(b) $I_2 : W \times H \times D \rightarrow [0, 65535]^3$
(c) $I_3 : W \times H \times T \rightarrow [0, 4095]$
(d) $I_4 : W \times H \times T \rightarrow [0, 4095]$
Context: End of the 90’s - Early 2000’s

⇒ Very few open-source tools existed then for these kind of tasks.

Existing software/libraries were

- Either easy to use, but not generic enough for our image data (GIMP, ImageMagick, GraphicsMagick, OpenCV, ...) → Mainly limited to RGB/RGBA 2D images.
- Or, targeted to experienced programmers (required the use of external libraries).

- We had developed generic libraries for image processing: CImg and Pandore (open-source, in C++):

  ![CImg and Pandore](http://cimg.eu)
  ![CImg and Pandore](https://clouard.users.greyc.fr/Pandore/)
Goal: Target a Broader Audience

- In practice, these libraries have been used “only” by a few hundred programmers.
  ⇒ **Cause**: High diversity of people in the image processing field. Not all C++ savvy!

⇒ Real need for **simpler interfaces** (than C++ libraries) to broaden our audience.

- **For the users**: Propose different user interfaces to apply image processing operators.
- **For the developers**: Ease algorithm prototyping and maintenance.

→ **Technical means**: 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. **All open-source**.
Global View of the G’MIC Framework

- **G'MIC interpreter** (C++)
- **GMICOL** (Web Service)
- **G'MIC-Qt** (plug-in)
- **libgmic** (C/C++)
- **ZArt** (webcam GUI)
- **gmic** (CLI)
- **gmic-py** (Python)
- **Custom commands** (G'MIC Scripts)
- **Climg** (C++ library)

Applications:
- GIMP - Krita - Paint.NET - Photoshop - Paint Shop Pro - Affinity Photo - Digikam - …
Using the CLI Tool “gmic”

$ gmic input.jpg -denoise_haar 1.4 -retinex 30 -sharpen 30

input.jpg  resulting image
Using the CLI Tool “gmic”

$ gmic cat.jpg fill
"J=[0,0,0];repeat(5,k,R=rot(k*72°);P=[w,h]/2+R*([x,y]−[w,h]/2);J+=I(P));J/5"

input.jpg

resulting image
Properties: Writing Custom Commands

- **G’MIC** code that implements “Non Local-Means”:

```c
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 duck.png nlmeans_expr 35,3,1
Properties: Writing Custom Commands

- $ gmic user.gmic duck.png nlmeans_expr 35,3,1

Left: Original color image, Right: Denoised with `nlmeans_expr`.

→ **G’MIC** is very convenient for quick algorithm prototyping.
→ **G’MIC** is very convenient for generating synthetic image data

*(creative coding as well!)*
Also Good for Creative Coding

```cpp
3 subdivision:
4 S127,312,1,3 1,1,3,0
5 eval $(-math_lib)
6 dar_insert(#-1,[ -256.0,-256.0,256.0, 0, 512.0 ]);
7 dar_insert(#-1,[ -256.0,-256.0, 0, 512.0, 0 ]);
8 dar_insert(#-1,[ 256.0,-256.0, 0, 512.0, 0 ]);
9 dar_insert(#-1,[ 256.0, 0, 0, 512.0, 0 ]);
10 dar_insert(#-1,[ 0, 0, 0, 512.0, 0 ]);
11 repeat(4,7):
12 N = dar_size(F-1):
13 repeat[N,k,]
14 F = [(-1,k); P0 = F[0,0]; W0 = F[3,0]; U1 = F[6,0];
15 (W0 | u<0.75) & area(W0 + U1)>1000
16 V0 = u*U0; W0 = U0 - V0;
17 V1 = u*U1; W1 = U1 - V1;
18 dar_insert(#-1,[ P0,W0,V0 ];)
19 dar_insert(#-1,[ P0 + W0,W0,V0 ];)
20 copy(1[-1,k],[ P0 + W0,V0,W0 ];)
21 ];
22 
23 const focale = 64;
24 proj2D(P) = [ fociol*P[0]/P[2] + w0*2, fociol*P[1]/P[2] + w0*2 ];
25 repeat(3)[N,k,]
26 F = [(-1,k); U0 = F[0,0]; U1 = F[3,0];
27 P0 = P[0,0] + [ 0.0, fociol ];
28 P1 = P0 + U0; P2 = P1 + U1; P3 = P0 + U1;
29 Z = 256 = fociol - (P0[2]*P1[2]*P2[2]*P3[2])/4
30 coords = [ proj2D(P0),proj2D(P1),proj2D(P2),proj2D(P3) ];
31 color = Z[u(10,30,30),[255,255,255]]
32 polygon(*4,cos,1,color);
33 polygon(*6,4,coords,0.35,F0000000,0)
34 y}
35 rm n,0,255 rap clot golden_fade
```
Properties: Manipulation of 3D Volumetric Images

$ 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
G’MIC First Release: August 2008

- First release of the CLI tool in August 2008
  → very few downloads (approx. 300/month).
- But... Playing a bit with the G’MIC language to write image processing pipelines quickly showed interesting potential for artistic use.

$ gmic colorful.jpg pencilbw 0.3 $ gmic colorful.jpg flower 10 ...

→ **Idea:** Write a G’MIC plug-in for GIMP!
The G’MIC-Qt Plug-In

- **G’MIC-Qt**: Plug-in, originally written for GIMP, that provides hundred of G’MIC filters for RGB or RGBA images. Now available for Photoshop, Affinity Photo, Paint Shop Pro, Paint.NET, Krita, . . .

(The G’MIC-Qt plug-in is developed and maintained by Sébastien Fourey).
Before / After the GIMP Plug-In

- First release of the **G’MIC** plug-in for GIMP in **January 2009**.
  → Significant increase of the downloads.

![Graph showing increase in downloads](image_url)
Why did artists quickly show an interest for G’MIC?

Demonstration of some of the G’MIC filters

(classical image processing algorithms used for artistic/retouching purposes)
Filter Showcase:

Rodilius
Open input image.
Invoke **G'MIC** plug-in and select **Artistic / Rodilius**.
Artistic: Rodilius

Rodilius is based on multiple oriented filters applied on Fourier space.
Artistic: Rodilius

Two other examples, works quite well on images with fur.
Artistic: Rodilius

- Reproduces the 'Fractalius' effect (45$ plug-in for Photoshop) but for 0$ and 12 lines of G’MIC code.

Redfield Fractalius

G’MIC Rodilius
Rodilius Code in G’MIC: 12 Lines

```
1. rodilius : check "$\{1=10\}\geq0 \&\& $\{1\leq200 \&\& $\{2=10\}\geq0 \&\& $\{2=100 \&\& $\{3=400\}\geq0 \&\& $\{4=7\}\geq0$" skip $\{5=0\},\{6=1\}
2. e[^-1] "Apply rodilius filter on image$\{1\}$ with amplitude $\{1\}$, thickness $\{2\}$, sharpness $\{3\}$, $\{4\}$ orientations,
3. offset $\{5\}$ and "$\{arg\ \{1+!\{6\}\},brighter,\{6\}\}\" color mode."
4. repeat $\{1\} \{!\{1\}\} \{split\_opacity\ \{rv\}\}
5. if $\{\{6\}\}$ negate. \{1\}
6. +f. 0 mm. [-2,\{n\}]
7. repeat round($\{4\}$)
8. angle=$\{\{5\}+\{6\}\}*180/\{\{round\(\{4\}\)\}
9. +\{blur\_linear.. $\{1\}$%,$\{1+\{2/100\}\}\%\}$,$\{\{single\ \{1\}\ b. 0.7\}$ sharpen. $\{3\}$ max[-2,-1]
10. done \{rn..\}
11. if $\{\{6\}\}$ negate. \{1\}
12. \{rv \{a \ c\} \{end\ \{1\}\} \{done\}
```
Filter Showcase:

Extract foreground [interactive]
Open input image (single-layer color photograph).
Place some “foreground” and “background” key points.
Contours: Extract Foreground [Interactive]

The algorithm segment the image into foreground and background regions.
Contours: Extract Foreground [Interactive]

Result of the filter: 2 layers instead of a single one.
Contours: Extract Foreground [Interactive]

Result of the filter: 2 layers (foreground layer shown here).
Contours: Extract Foreground [Interactive]

Each layer can be processed individually.
Contours: Extract Foreground [Interactive]

- **Background layer:** Change hue, add bokeh.
- **Foreground layer:** Apply color curves to adjust the contrast.
Filter Showcase:

Inpainting
Open input image.
Repair: Inpainting

Draw an inpainting mask directly on it (with a constant known color).
Invoke **G’MIC** plug-in and select **Repair / Inpaint [patch-based]**.
Inpainting result.
Repair: Inpainting

- **G’MIC** is one of the few software to offer several “inpainting” algorithms:
Repair: Inpainting

- G’MIC is one of the few software to offer several “inpainting” algorithms:
Repair: Inpainting

- **G’MIC** is one of the few software to offer several “inpainting” algorithms:
Filter Showcase:

Poisson editing
Poisson Editing

Open an image.
Open another image and select the inner face.
Poisson Editing

Scale and rotate selection to fit size of the first face.
Poisson Editing

Apply Poisson Editing filter “Seamless Blending” to merge selection and background.
Example of object insertion, using Poisson Editing.
Example of face transfer, using Poisson Editing.
Example of face transfer, using Poisson Editing (on the same input picture).
More Than 500 Filters Available in The G’MIC-Qt Plug-In

Example of another filter in G’MIC: Engrave

⇒ The possibilities for artistic creation are huge.
Collaboration with an illustrator: David Revoy

An Algorithm to Help Colorizing Comics

Joint work with Sébastien Fourey / GREYC.
David Revoy, author of “Pepper & Carrot”.
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Issues With Lineart Colorization
Step 1: Closing Strokes
Step 1: Closing Strokes
Step 1: Closing Strokes
Step 1: Closing Strokes
Step 1: Closing Strokes
Step 1: Closing Strokes

Quality Measure:

\[ \omega(s, t) := \max(0, 1 - \frac{\| s - t \|}{d_{\text{max}}}) \cdot \max(0, \tilde{n}(s) \cdot (-\tilde{n}(t)) - \cos(\alpha)) \cdot \max(0, \tilde{n}(s)\tilde{v}_{st} + \tilde{n}(t)\tilde{v}_{ts}) \]
Step 1: Closing Strokes

Quality Measure:

\[ \omega(s, t) := \max(0, 1 - \frac{\|s-t\|}{d_{\text{max}}}) \cdot \max(0, \tilde{n}(s) \cdot (-\tilde{n}(t)) - \cos(\alpha)) \cdot \max(0, \tilde{n}(s) \tilde{v}_st + \tilde{n}(t) \tilde{v}_ts) \]
Step 1: Closing Strokes
Step 2: Pre-colorization
Artist’s Work Facilitated
Artist’s Work Facilitated
Artist’s Work Facilitated
Artist’s Work Facilitated
Auto-Clean Mode
Auto-Clean Mode
Auto-Clean Mode
Auto-Clean Mode
Smart Colorize, a G’MIC-Qt Filter

“A Fast and Efficient Semi-Guided Algorithm for Flat Coloring Line-Arts.”,
S. Fourey, D. Tschumperlé, and D. Revoy.

Work In Progress: Automatic Illumination of Comics
Work In Progress: Automatic Illumination of Comics

Automatic estimation of the bump map and the associated normal map
Work In Progress: Automatic Illumination of Comics

→ A single-click solution for object illumination
Work In Progress: Automatic Illumination of Comics

Results with different settings of light rendering
Collaboration with a photographer: Pat David

An Algorithm for Compressing Color LUTs

Joint work with Amal Mahboubi and Christine Porquet / GREYC.
Pat David, amateur photographer and free software enthusiast.
What is a **CLUT (Color Look-Up Table)**?

(a) **CLUT** $\mathbf{F} : RGB \rightarrow RGB$, visualized in 3D

$$\forall (x, y), \ J(x,y) = F(I(x,y))$$

(b) Original color image $I$

(c) Image $J$, after transformation $F$
Examples of *CLUT*-Based Transformations

Original image

“60’s”

“Color Negative”

“Orange Tone”

“Ilford Delta 3200”

“Backlight Filter”

“Bleach Bypass”

“Late Sunset”
Standard Ways of Storing a \textit{CLUT}

\begin{itemize}
\item \textbf{a)} a \textit{CLUT} is a 3D dense color volume
\item \textbf{b)} Storage as a \texttt{.cube}
\item \textbf{c)} Storage as a \texttt{.png}
\end{itemize}

In both cases, \textbf{lossless} compression, but restricted to small sizes:

1. \texttt{.cube file:} \textit{ASCII} zipped format \((\text{\textit{CLUT}} \ 64^3 \approx 1 \ \text{Mo})\)
2. \texttt{.png file:} 2D image \((\text{\textit{CLUT}} \ 64^3 \approx 64 \ \text{to} \ 100 \ \text{Ko})\)

\Rightarrow \textbf{Issue:} Allow a large-scale distribution of \textit{CLUT}s (800+), by limiting the \textit{CLUT} storage as much as possible.
Our Approach: CLUT Compression

**Compression**: Let $\mathbf{F} : \text{RGB} \to \text{RGB}$ be a 3D CLUT. We generate $\mathcal{K}$, a smaller representation of the CLUT, based on the storage of a set of **sparse** color keypoints.

Original CLUT $\mathbf{F}$  

Determination of 3D color keypoints $\mathcal{K}$  

Storage of keypoints in compressed form
Our Approach: CLUT Decompression

**Decompression:** A 3D interpolation based on anisotropic diffusion PDEs is applied to $\mathcal{K}$ in order to generate a reconstructed CLUT $\tilde{F}$ visually close to $F$.

Storage of keypoints in compressed form

Decompressed CLUT $\tilde{F}$

Goal: No perceptual differences between the two applied transformations (original/compressed)

D. Tschumperlé

Keynote CyberWorlds 2021
Reconstruction Principles

(a) Set $\mathcal{K}$ of known keypoints

(b) Initial state $F_{t=0}$
   *(e.g., smoothed Voronoï 3D)*

(c) Diffusion orientations $\eta$

(d) State at convergence *(PDE Solution)*
1. **Initialization** of $\mathcal{K} = \{(X_k, F(X_k)) \mid k = 1 \ldots 8\}$ (the 8 vertices of the cube).

⇒ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}(X) = \Delta E(F(X), \tilde{F}_N(X)).$$
Compression: Generation of Keypoints

2. **Iterative addition** into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(X)} = \Delta E(F_{(X)}, \tilde{F}_{N(X)})$$
Compression: Generation of Keypoints

2. **Iterative addition** into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$Err_{(X)} = \Delta E(F(X), \tilde{F}_{N(X)}).$$
Compression: Generation of Keypoints

2. Iterative addition into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

⇒ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(X)} = \Delta E(F_{(X)}, \tilde{F}_{N(X)}).$$
Compression: Generation of Keypoints

2. **Iterative addition** into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(X)} = \Delta E(F(X), \tilde{F}_N(X)).$$
Compression: Generation of Keypoints

2. **Iterative addition** into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}(x) = \Delta E(F(x), \tilde{F}_N(x)).$$
Compression: Generation of Keypoints

2. Iterative addition into $K$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}(_{X}) = \Delta E(F(_{X}), F_{N}(X)).$$
Compression: Generation of Keypoints

2. **Iterative addition** into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

$\Rightarrow$ **Calculation of the pointwise (3D) reconstruction error:**

$$\text{Err}(\mathbf{x}) = \Delta E(\mathbf{F}(\mathbf{x}), \mathbf{F}_N(\mathbf{x})).$$
Compression: Generation of Keypoints

2. **Iterative addition** into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

⇒ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}(x) = \Delta E(F(x), \tilde{F}_N(x)).$$
Compression: Generation of Keypoints

2. **Iterative addition** into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$\text{Err}(\mathbf{x}) = \Delta E(\mathbf{F}(\mathbf{x}), \tilde{\mathbf{F}}_{N}(\mathbf{x})).$
Compression: Generation of Keypoints

2. **Iterative addition** into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}(X) = \Delta E(F(X), \tilde{F}_N(X)).$$
2. **Iterative addition** into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

⇒ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(x)} = \Delta E(F(x), \tilde{F}_N(x)).$$
Compression: Generation of Keypoints

2. Iterative addition into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(X)} = \Delta E(F_{(X)}, \tilde{F}_{N(X)})$$
Compression: Generation of Keypoints

2. Iterative addition into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.

⇒ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}(x) = \Delta E(F(x), \tilde{F}_N(x)).$$
Adding Keypoints for CLUT Compression

(a) max/average error

(b) PSNR evolution

(c) Iterative addition of keypoints
**CLUT Compression Results**

<table>
<thead>
<tr>
<th><strong>CLUT name</strong></th>
<th>Bourbon 64</th>
<th>Faded 47</th>
<th>Milo 5</th>
<th>Cubicle 99</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>16³</td>
<td>32³</td>
<td>48³</td>
<td>64³</td>
</tr>
<tr>
<td>Size in .cube.zip</td>
<td>23.5 Kb</td>
<td>573 Kb</td>
<td>3 Mb</td>
<td>1.2 Mb</td>
</tr>
<tr>
<td>Size in .png</td>
<td>3.7 Kb</td>
<td>22 Kb</td>
<td>72 Kb</td>
<td>92 Kb</td>
</tr>
<tr>
<td>Number of Keypoints</td>
<td>562</td>
<td>294</td>
<td>894</td>
<td>394</td>
</tr>
<tr>
<td>PSNR</td>
<td>45.8 dB</td>
<td>45.6 dB</td>
<td>45 dB</td>
<td>45.2 dB</td>
</tr>
<tr>
<td>Compression time</td>
<td>28 s</td>
<td>92 s</td>
<td>1180 s</td>
<td>561 s</td>
</tr>
<tr>
<td>Decompression time</td>
<td>67 ms</td>
<td>157 ms</td>
<td>260 ms</td>
<td>437 ms</td>
</tr>
<tr>
<td>Keypoints in .png</td>
<td>1.9 Kb</td>
<td>1.5 Kb</td>
<td>4.2 Kb</td>
<td>1.9 Kb</td>
</tr>
<tr>
<td>%cRate / .cube.zip</td>
<td>92.1%</td>
<td>99.7%</td>
<td>99.8%</td>
<td>99.8%</td>
</tr>
<tr>
<td>%cRate / .png</td>
<td>49.5%</td>
<td>93.3%</td>
<td>94.2%</td>
<td>98%</td>
</tr>
</tbody>
</table>

- **General measurement:** A set of 894 *CLUTs* (original size: around 500 Mo), mix of .cube.zip and .png files, compressed in 3.3 Mo.

- “Reconstruction of Smooth 3D Color Functions from Keypoints: Application to Lossy Compression and Exemplar-Based Generation of Color LUTs.”
  D. Tschumperlé, C. Porquet, and A. Mahboubi.
Filters for Color Transformations in G’MIC-Qt

- Filters “Color Presets” and “Simulate Film”:

  A lot of diverse color transformations (950+) are now available to our users, at no (memory) cost.
Conclusions

Artistic Imaging, an exciting field to explore
Conclusions/Future Prospects

- Open-source development ensures reproducible research (and happy users!).
- A lot of interesting image processing problems can be discovered in artistic imaging.

- Neural networks / Machine learning capabilities currently being implemented in G’MIC.

https://gmic.eu

- Neural networks are consumers of large resources. Light architectures and models still to be discovered.