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

GREYC Laboratoire de recherche en sciences du numérique

Normandie Université

UNICAEN

ENSICAEN ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

cnrs

Context: Public Research Lab

- 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.

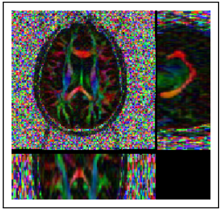
...

⇒ Various image data coming from very diverse sensors.

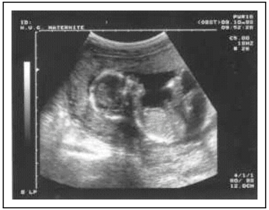
Context: Image Data

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

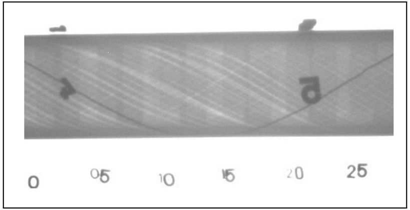
(a) $I_1 : W \times H \rightarrow [0, 255]^3$



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



(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).

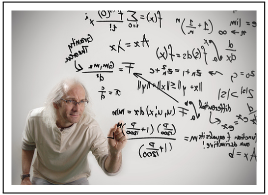
- We had developed generic libraries for image processing: **CImg** and **Pandore** (open-source, in C++):

http://cimg.eu

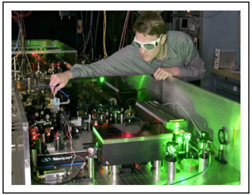
https://clouard.users.greyc.fr/Pandore/

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!



Mathematicians



Physicists

Programmers

Biologists ...

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

Proposal: The G'MIC Project (2008)

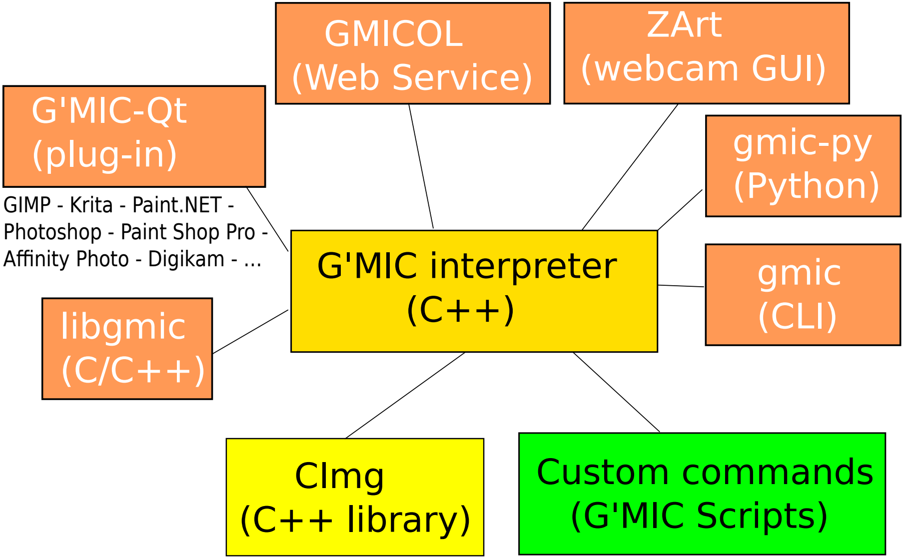
G'MIC
GREYC's Magic for Image Computing

`https://gmic.eu`

- **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



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

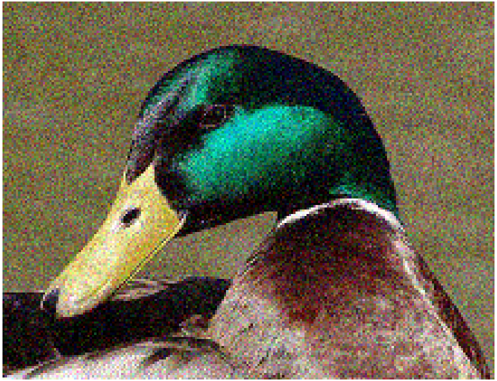
- G'MIC code that implements "Non Local-Means":

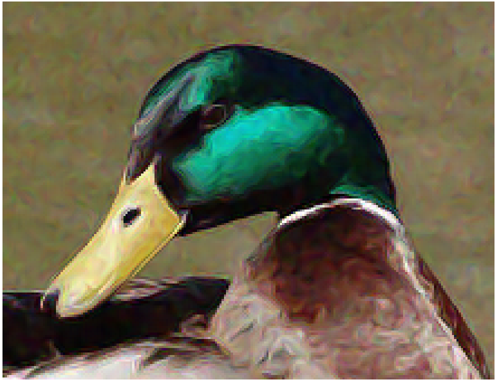
```
nlmeans_expr : -check "${1=10}>0 && isint(${2=3}) && $2>0 && isint(${3=1}) && $3>0"
  -fill "
    const sigma = $1;  # Denoising strength.
    const hl = $2;     # Lookup half-size.
    const hp = $3;     # Patch half-size.
    value = 0;
    sum_weights = 0;
    for (q = -hl, q<=hl, ++q,
      for (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`



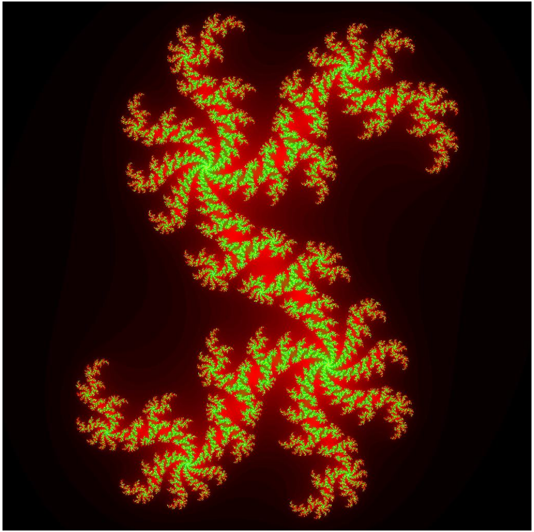


Left: Original color image, **Right**: Denoised with `nlmeans_expr`.

→ **G'MIC** is very convenient for quick algorithm prototyping.

Generating Image Data From Scratch

```
julia :
  2048,2048,1,1,"*
    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<256?iter:0"
  map 7
```

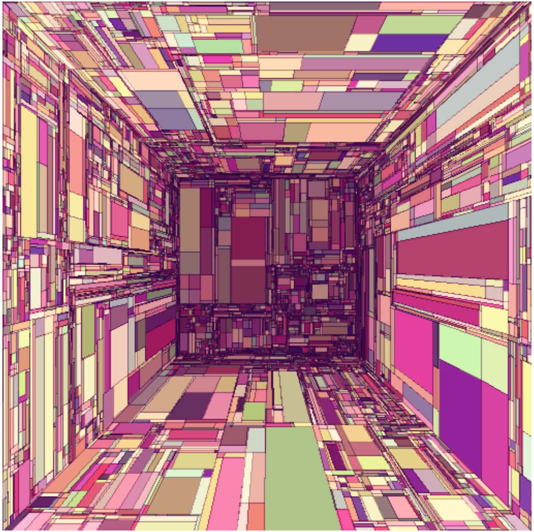


→ **G'MIC** is very convenient for generating synthetic image data

(**creative coding** as well!).

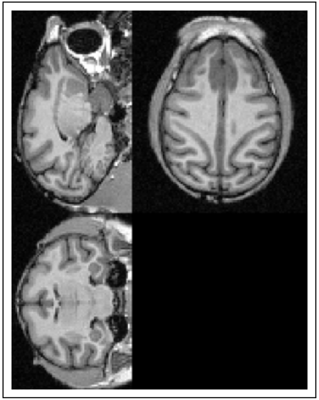
Also Good for Creative Coding

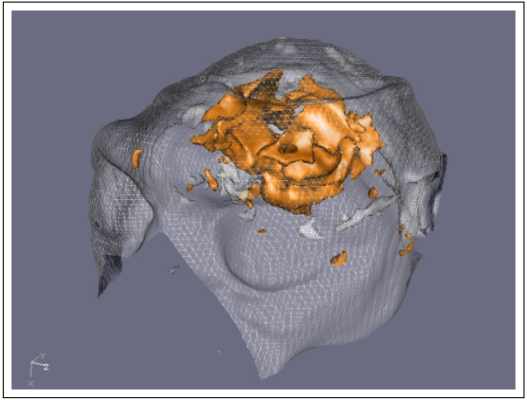
```
subdivide3d :
  512,512,1,3 1,1,1,9
  eval ${-math_lib}"
    dar_insert(#-1,[ -256,-256,128, 512,0,0, 0,512,0 ] );
    dar_insert(#-1,[ -256,-256,0, 0,0,128, 0,512,0 ] );
    dar_insert(#-1,[ 256,-256,0, 0,0,128, 0,512,0 ] );
    dar_insert(#-1,[ -256,-256,0, 512,0,0, 0,0,128 ] );
    dar_insert(#-1,[ -256,256,0, 512,0,0, 0,0,128 ] );

    repeat(8,r,
      N = dar_size(#-1);
      repeat(N,k,
        F = I[#-1,k]; P0 = F[0,3]; U0 = F[3,3]; U1 = F[6,3];
        (N<8 || u<0.75) && norm(U0 + U1)>10?(
          V0 = u*U0; W0 = U0 - V0;
          V1 = u*U1; W1 = U1 - V1;
          dar_insert(#-1,[ P0,V0,V1 ]);
          dar_insert(#-1,[ P0 + V0,W0,V1 ]);
          dar_insert(#-1,[ P0 + V1,V0,W1 ]);
          copy(I[#-1,k],[ P0 + V0 + V1,W0,W1 ]);
        );
      );
    );

    const focale = 64;
    proj2d(P) = [ focale*P[0]/P[2] + w#0/2, focale*P[1]/P[2] + h#0/2 ];
    repeat (dar_size(#-1),k,
      F = I[k]; U0 = F[3,3]; U1 = F[6,3];
      P0 = F[0,3] +  [ 0,0,focale ];
      P1 = P0 + U0; P2 = P1 + U1; P3 = P0 + U1;
      Z = 200 + focale - (P0[2]+P1[2]+P2[2]+P3[2])/4;
      coords = [ proj2d(P0),proj2d(P1),proj2d(P2),proj2d(P3) ];
      color = Z*u([20,20,20],[255,255,255]);
      polygon(#0,4,coords,1,color);
      polygon(#0,-4,coords,0.5,0xFFFFFFFF,0)
    )"
  rm. n 0,255 map_clut 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
  $\Longrightarrow$ 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.

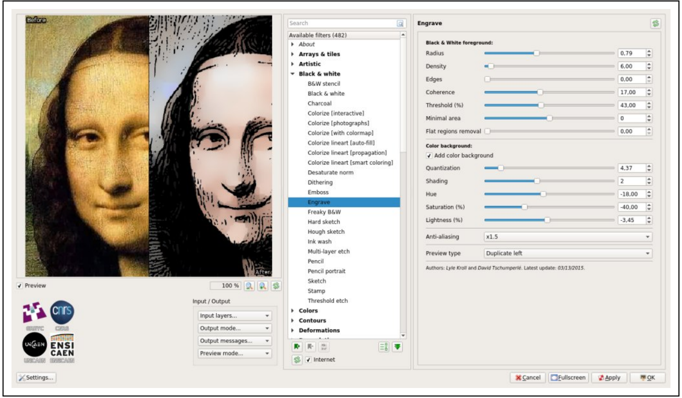
```
$ gmic colorful.jpg pencilbw 0.3        $ gmic colorful.jpg flower 10   ...
```




$\longrightarrow$ **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, . . .



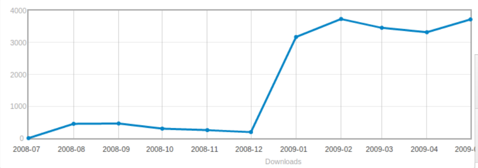
(The G’MIC-Qt plug-in is developed and maintained by Sébastien Fourey).

Before / After the GIMP Plug-In

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

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

Why did artists quickly show an interest for **G'MIC** ?

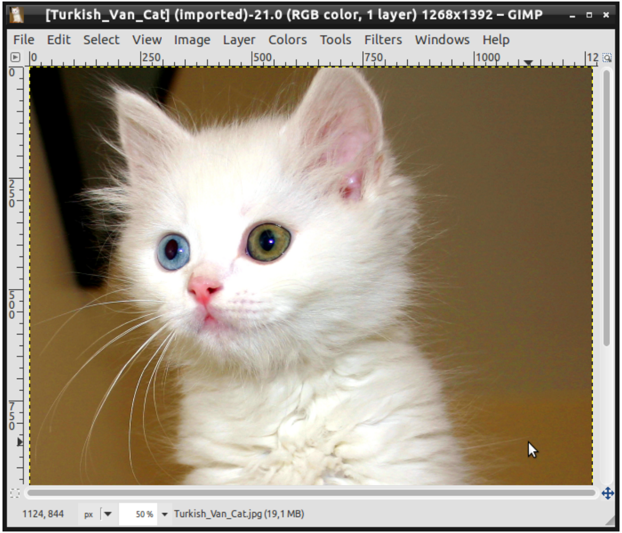
Demonstration of some of the G'MIC filters

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

Filter Showcase:

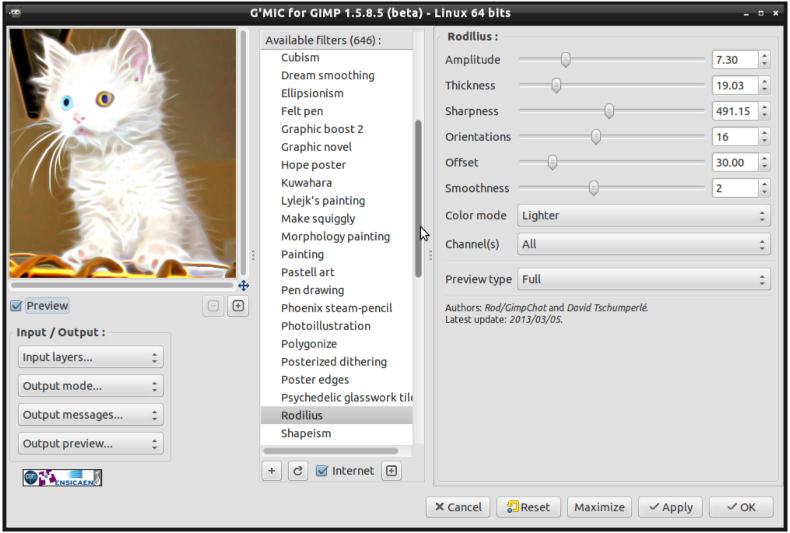
`Rodilius`

Artistic: Rodilius



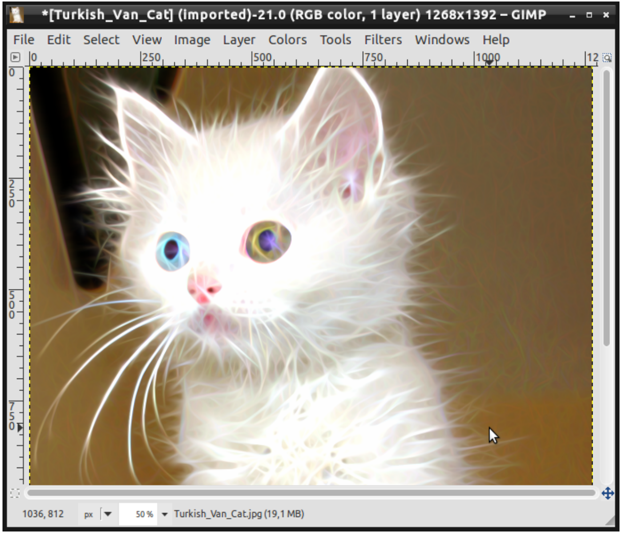
Open input image.

Artistic: Rodilius



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

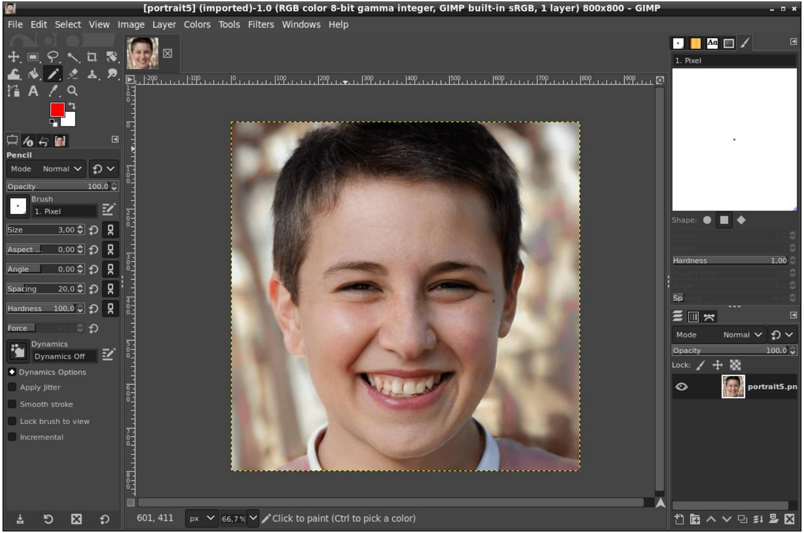
Rodilius Code in G'MIC: 12 Lines

```
rodilius : check "${1=10}>=0 && $1<=200 && ${2=10}>=0 && $2<=100 && ${3=400}>=0 && ${4=7}>0" skip ${5=0},${6=1}
  e[^-1] "Apply rodilius filter on image$? with amplitude $1, thickness $2, sharpness $3, $4 orientations,
          offset $5 and "${arg\ 1+!$6,brighter,darker}" color mode."
  repeat $! l[$>] split_opacity rv
    if !$6 negate. fi
    +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 negate. fi
  rv a c endl done
```

Filter Showcase:

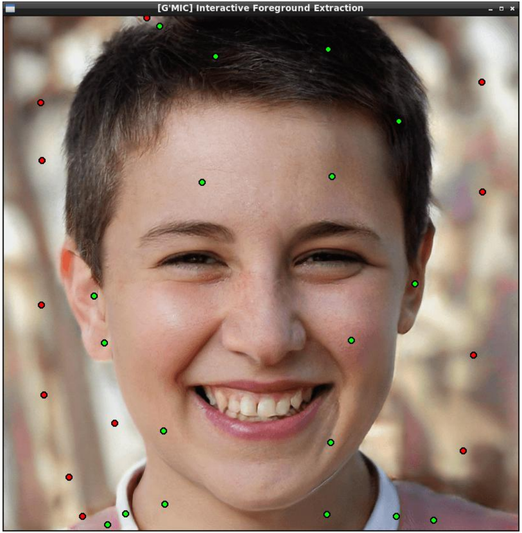
`Extract foreground [interactive]`

Contours: Extract Foreground [Interactive]



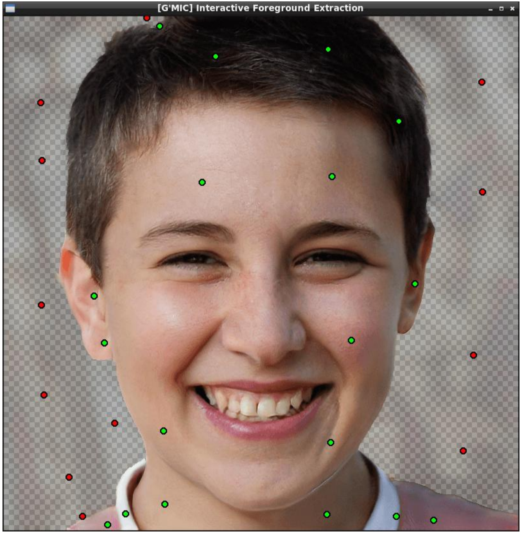
Open input image (single-layer color photograph).

Contours: Extract Foreground [Interactive]



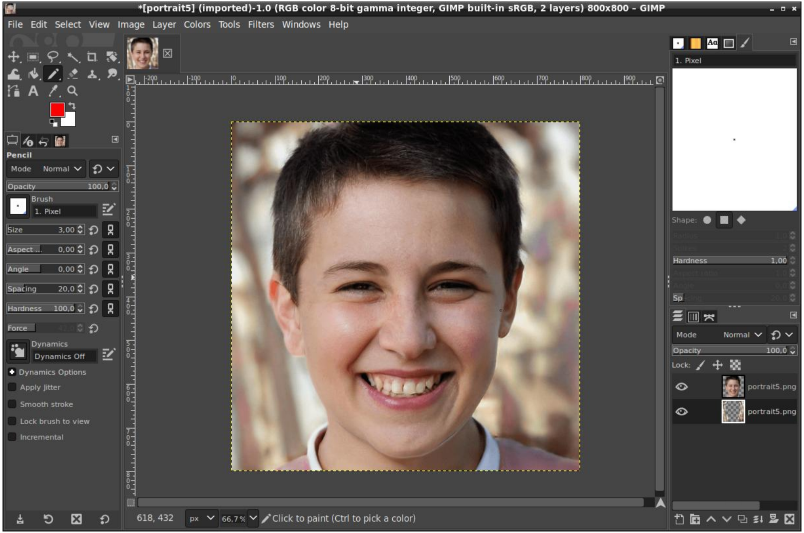
Place some “foreground” and “background” key points.

Contours: Extract Foreground [Interactive]



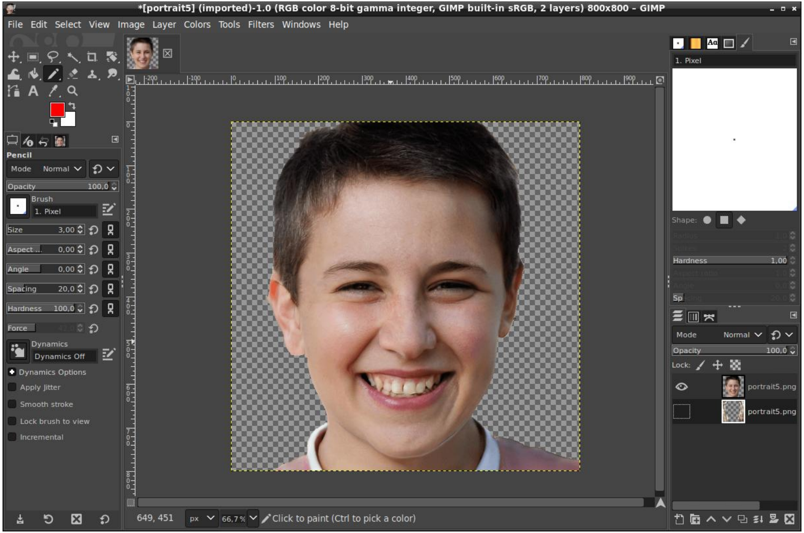
The algorithm segment the image into foreground and background regions.

Contours: Extract Foreground [Interactive]



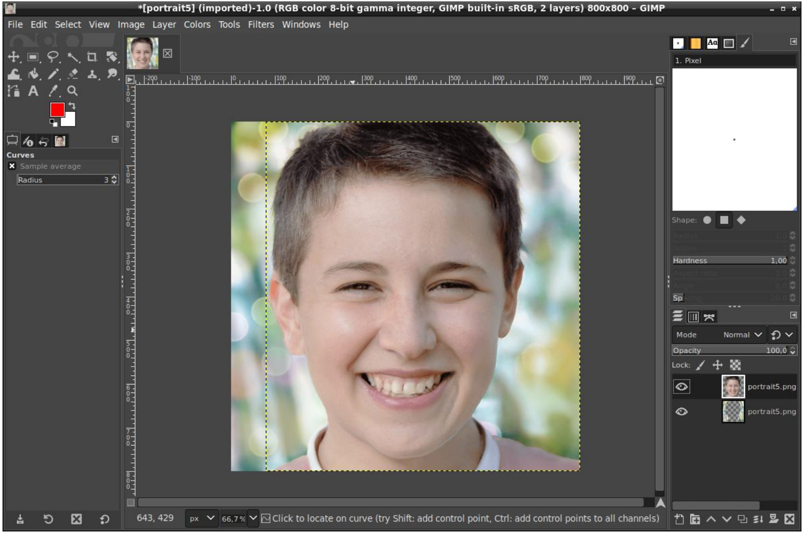
Result of the filter: 2 layers instead of a single one.

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]

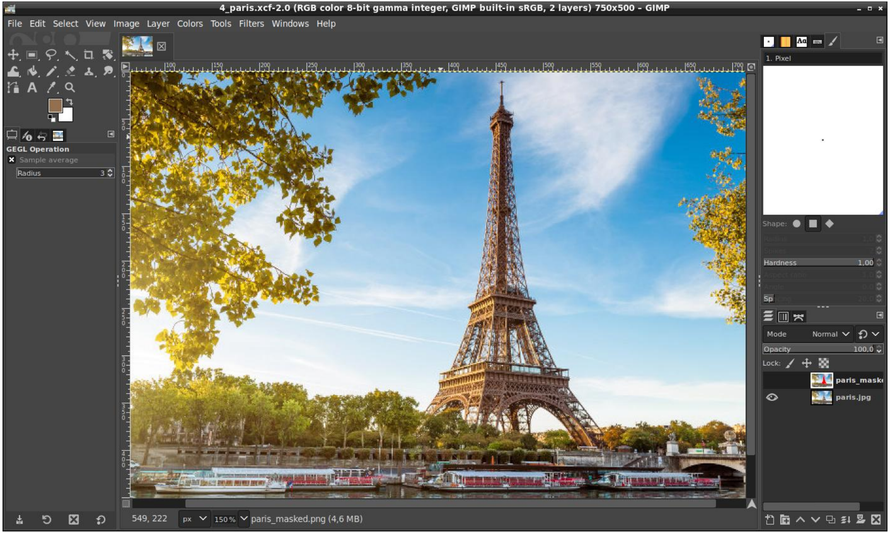
$\Longrightarrow$

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

Filter Showcase:

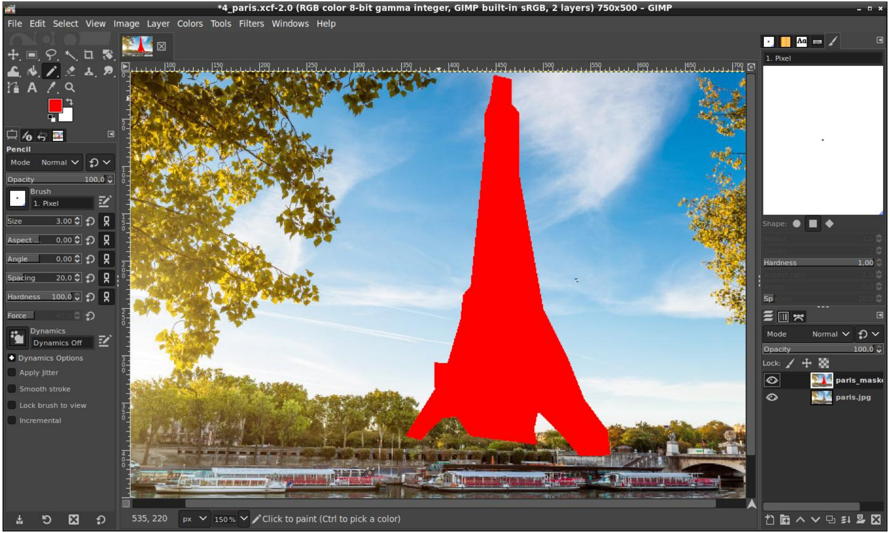
`Inpainting`

Repair: Inpainting



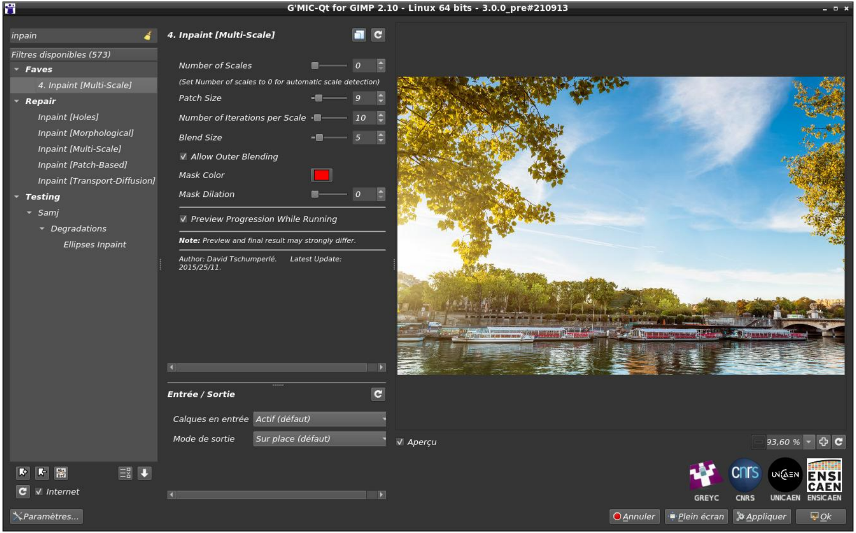
Open input image.

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

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:

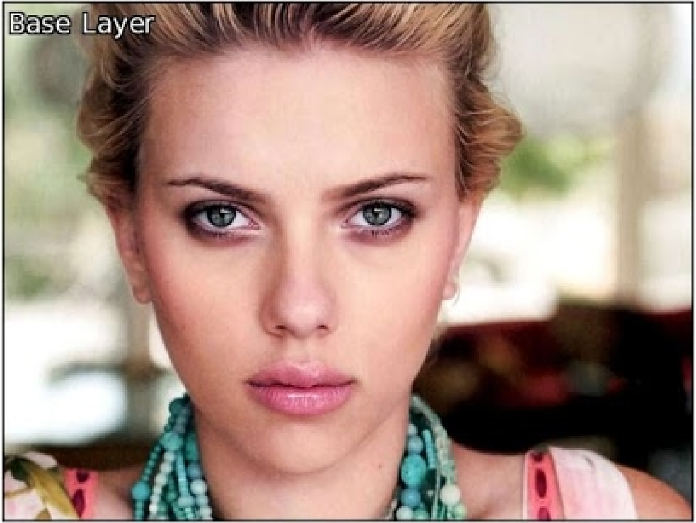
Repair: Inpainting

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

Filter Showcase:

`Poisson editing`

Poisson Editing

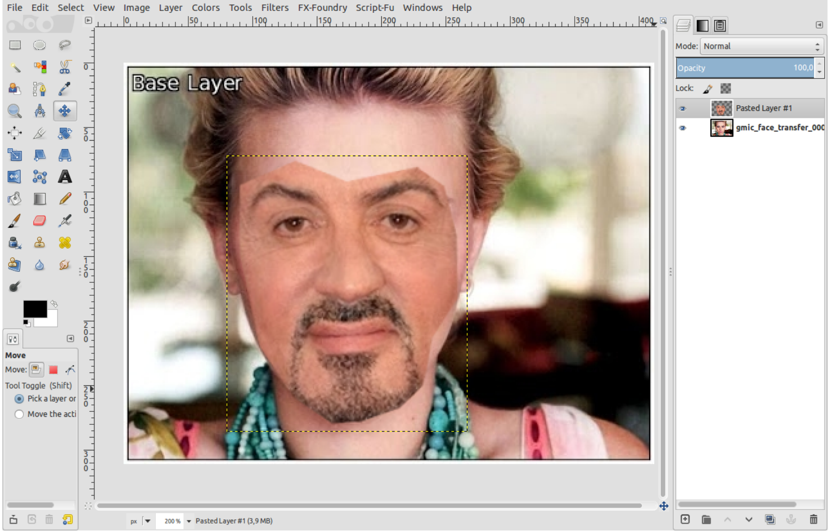


Open an image.

Poisson Editing

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.

Poisson Editing

Example of object insertion, using Poisson Editing.

Poisson Editing

Example of face transfer, using Poisson Editing.

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

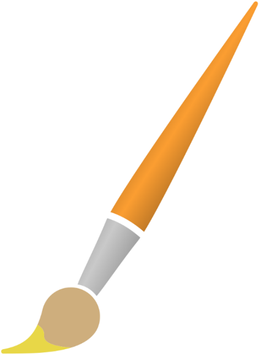
An Algorithm to Help Colorizing Comics

Joint work with Sébastien Fourey / GREYC.

David Revoy, author of “Pepper & Carrot”.

Issues With Lineart Colorization

Issues With Lineart Colorization



Issues With Lineart Colorization

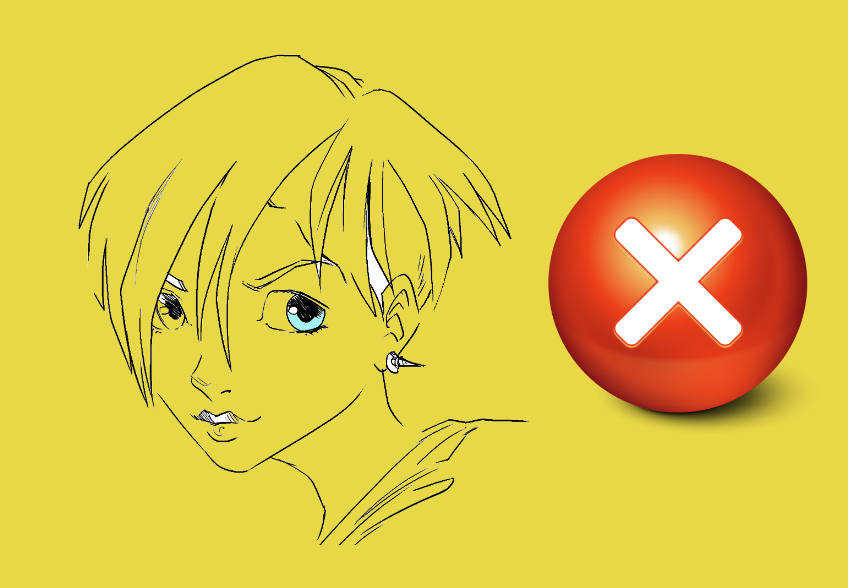
Issues With Lineart Colorization

Issues With Lineart Colorization

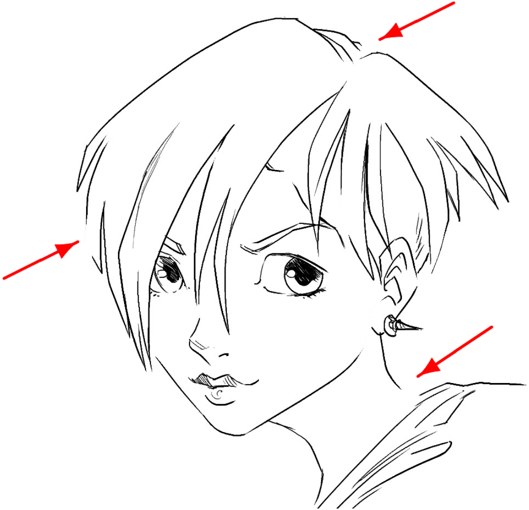
Issues With Lineart Colorization

Issues With Lineart Colorization

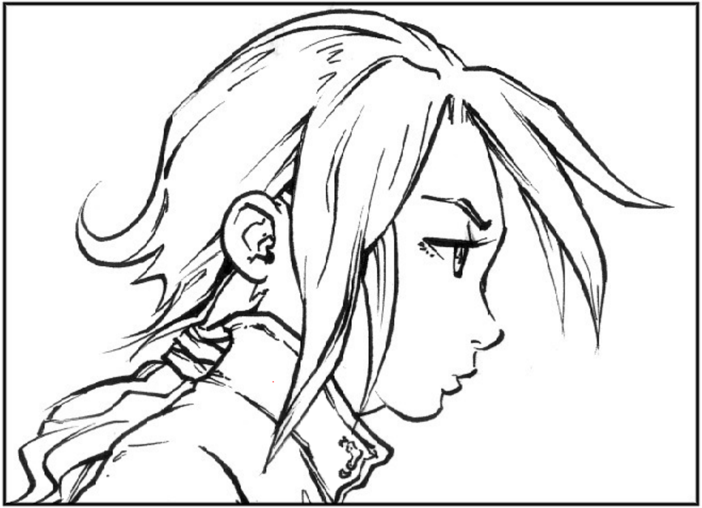
Issues With Lineart Colorization



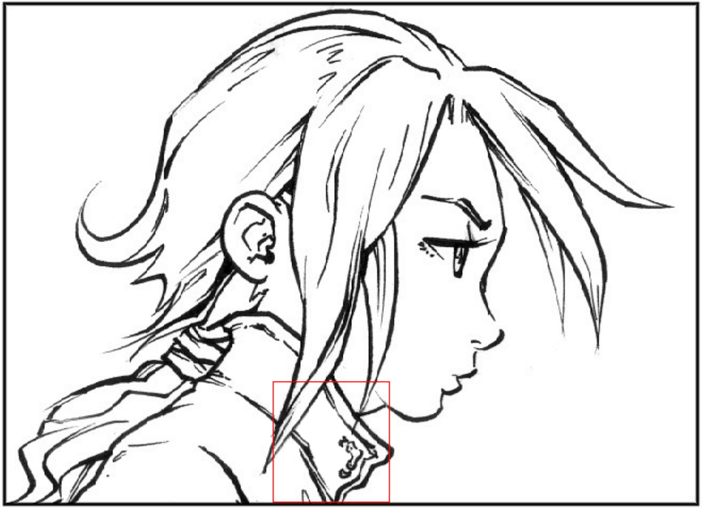
Issues With Lineart Colorization



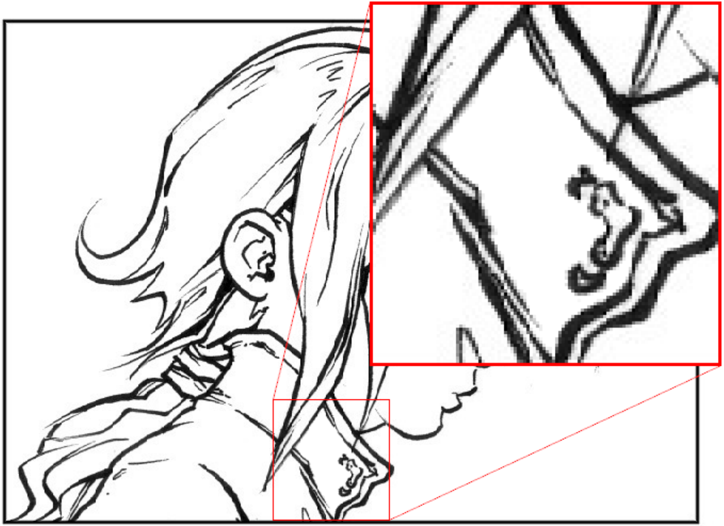
Issues With Lineart Colorization



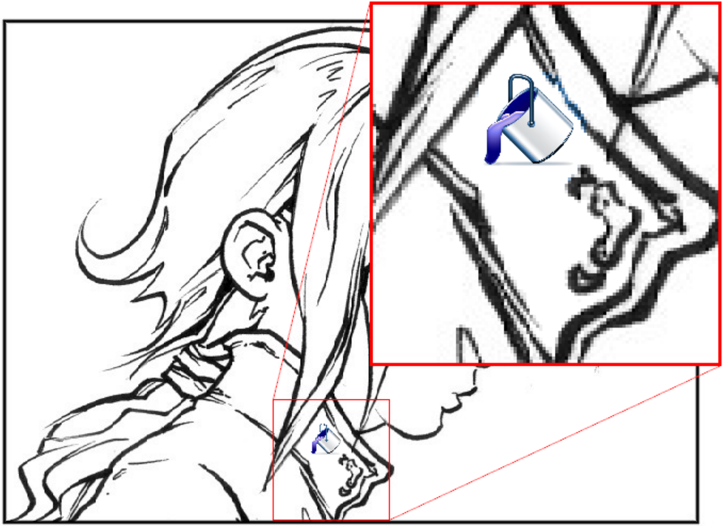
Issues With Lineart Colorization



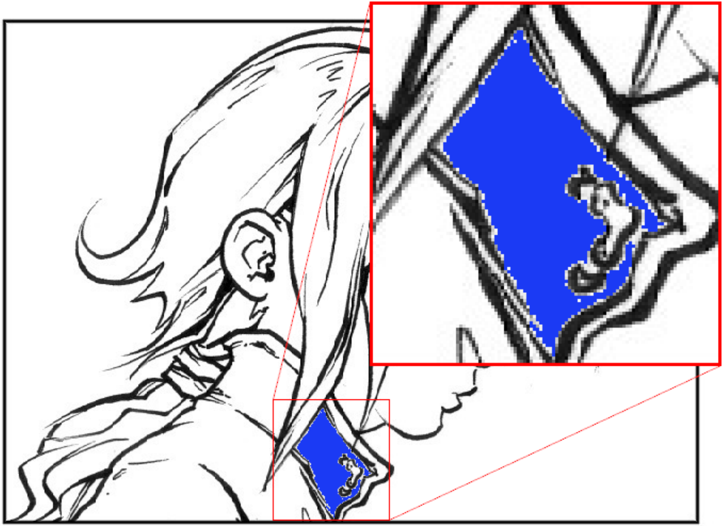
Issues With Lineart Colorization



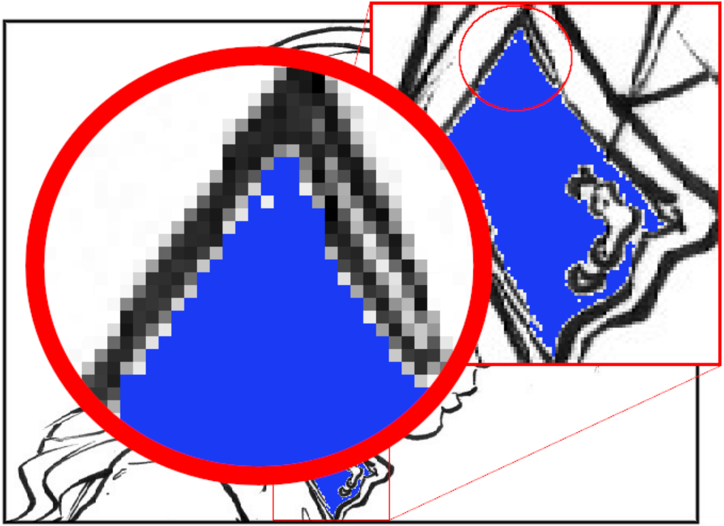
Issues With Lineart Colorization



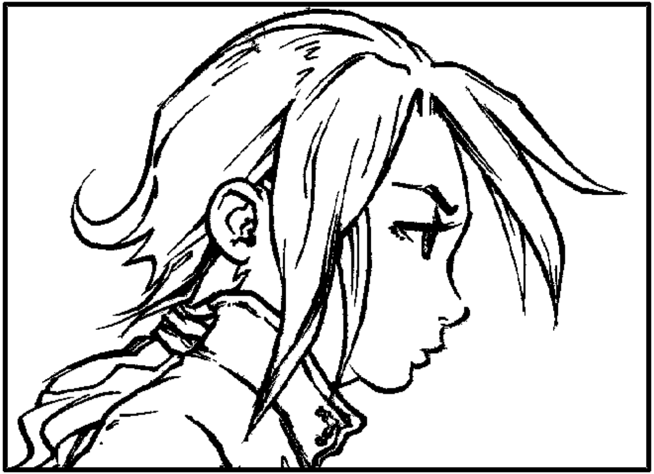
Issues With Lineart Colorization



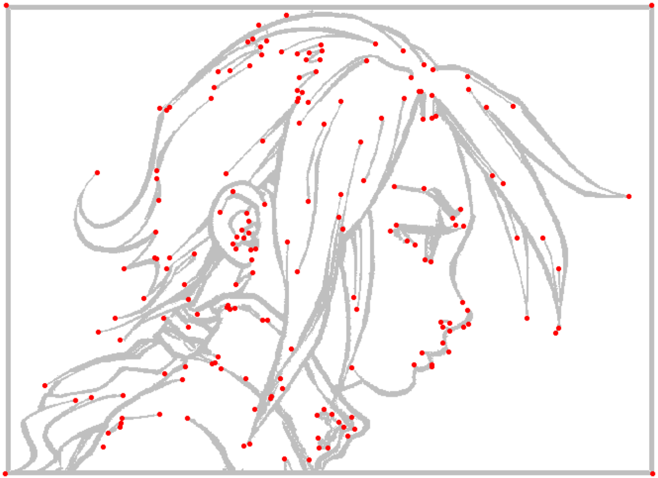
Issues With Lineart Colorization



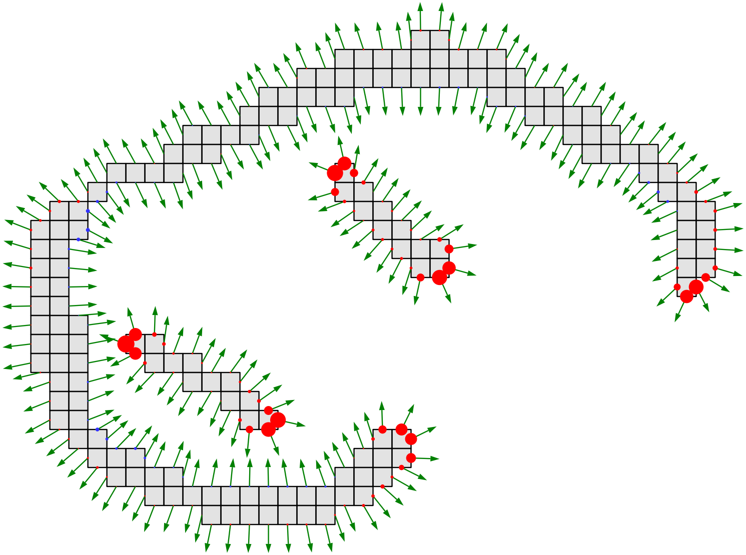
Step 1: Closing Strokes



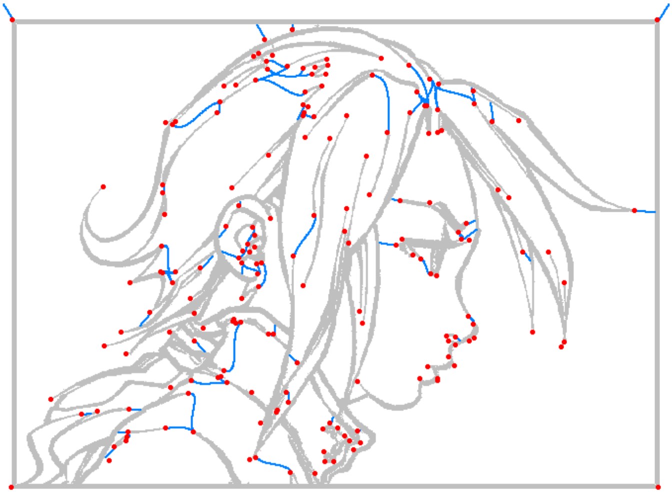
Step 1: Closing Strokes



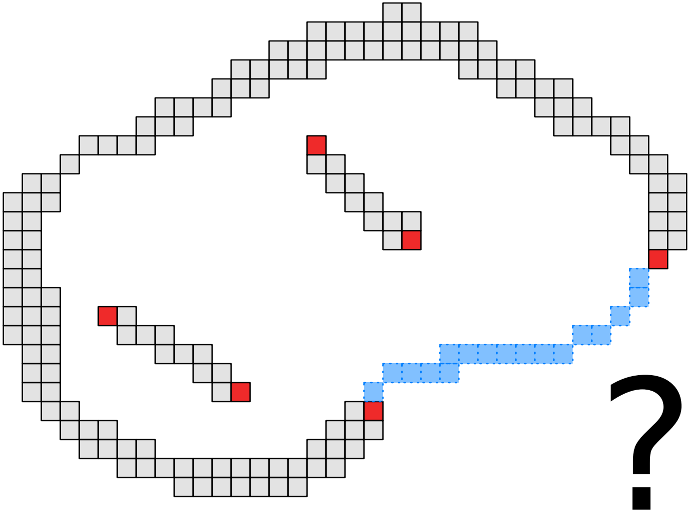
Step 1: Closing Strokes



Step 1: Closing Strokes



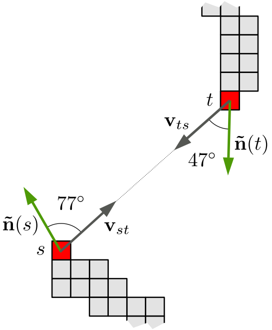
Step 1: Closing Strokes

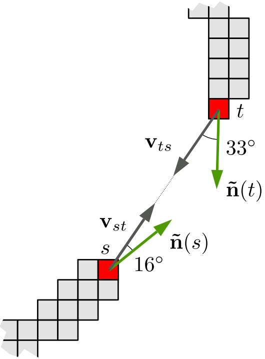


Step 1: Closing Strokes

Quality Measure:

$$\omega(s,t) \;:=\; \max\left(0, 1-\frac{\|s-t\|}{d_{\text{max}}}\right)$$
$$\cdot \max\left(0, \vec{\tilde{n}}(s)\cdot(-\vec{\tilde{n}}(t)) - \cos(\alpha)\right)$$
$$\cdot \max\left(0, \vec{\tilde{n}}(s)\vec{v}_{st} + \vec{\tilde{n}}(t)\vec{v}_{ts}\right)$$

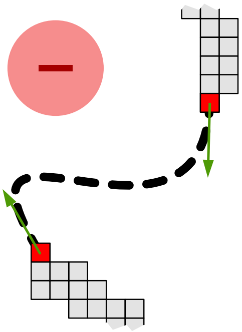


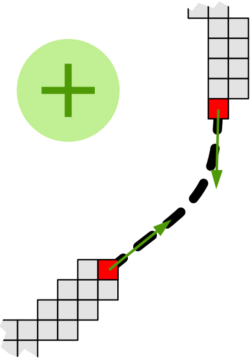


Step 1: Closing Strokes

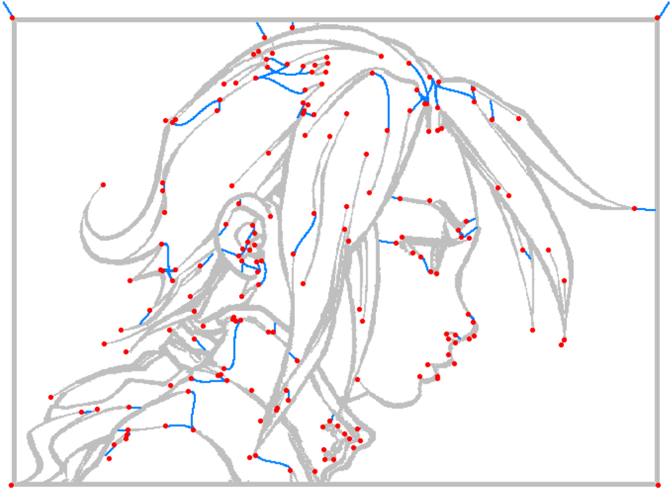
Quality Measure:

$$\begin{aligned}\omega(s,t) \quad := \quad & \max\left(0, 1 - \frac{\|s-t\|}{d_{\text{max}}}\right) \\ & \cdot \max(0, \vec{\tilde{n}}(s) \cdot (-\vec{\tilde{n}}(t)) - \cos(\alpha)) \\ & \cdot \color{red}{\max(0, \vec{\tilde{n}}(s)\vec{v}_{st} + \vec{\tilde{n}}(t)\vec{v}_{ts})}\end{aligned}$$



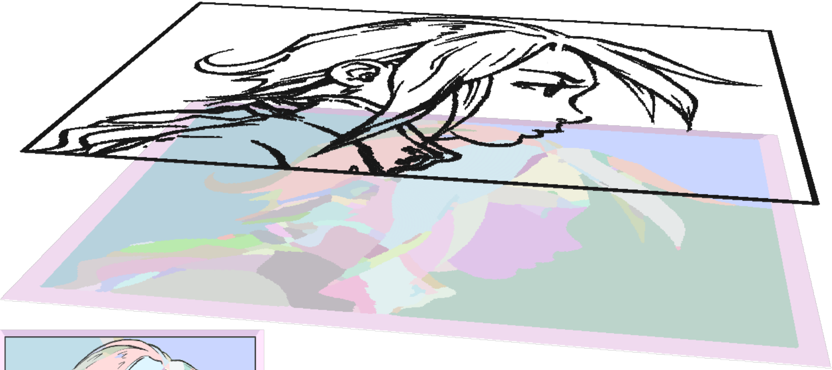


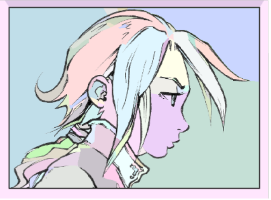
Step 1: Closing Strokes



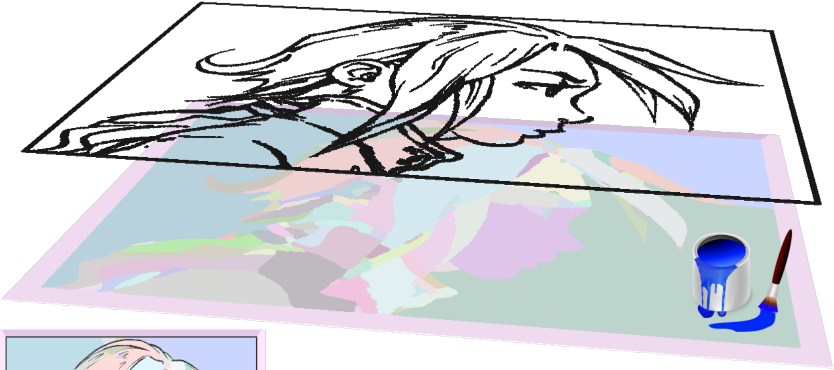
Step 2: Pre-colorization

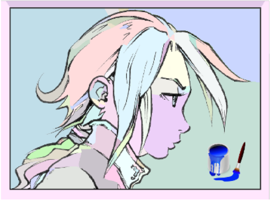
Artist’s Work Facilitated



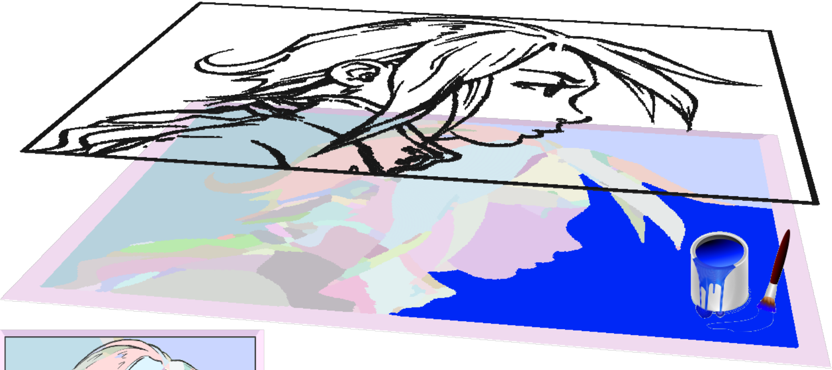


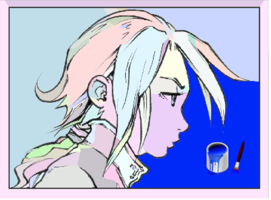
Artist’s Work Facilitated



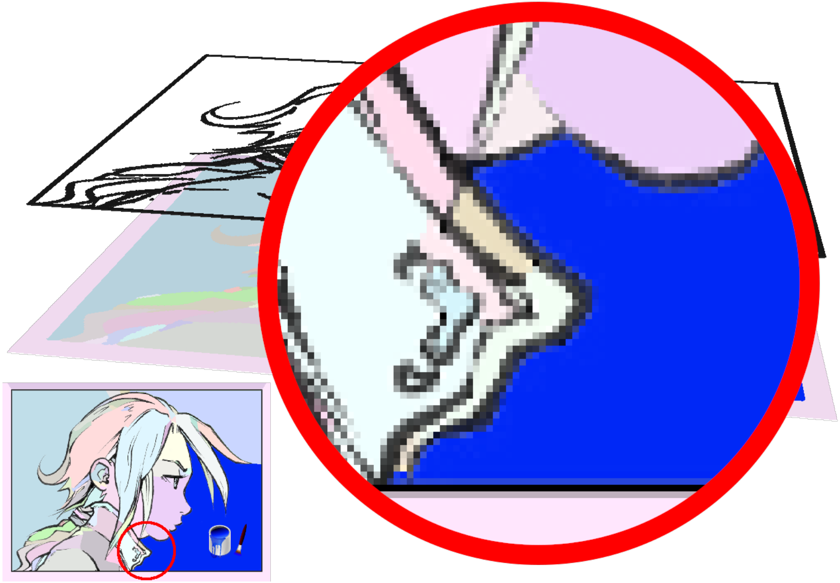


Artist’s Work Facilitated



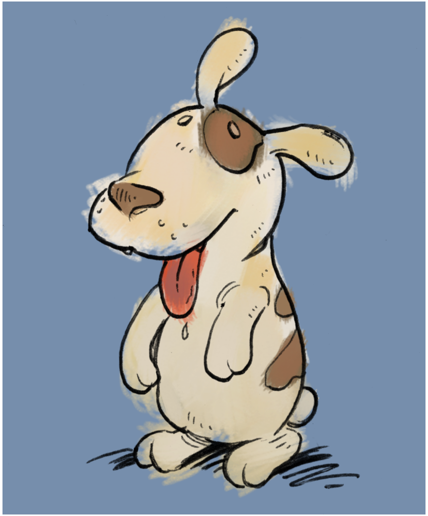


Artist's Work Facilitated

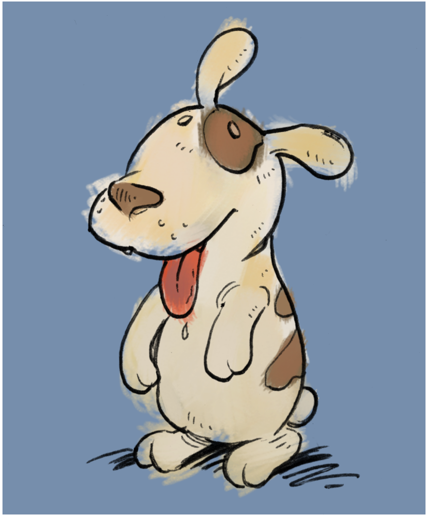


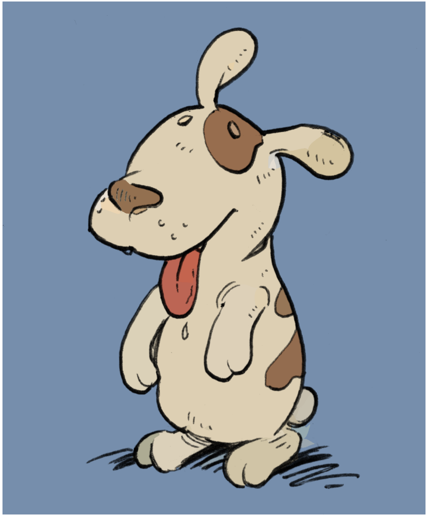
Auto-Clean Mode

Auto-Clean Mode

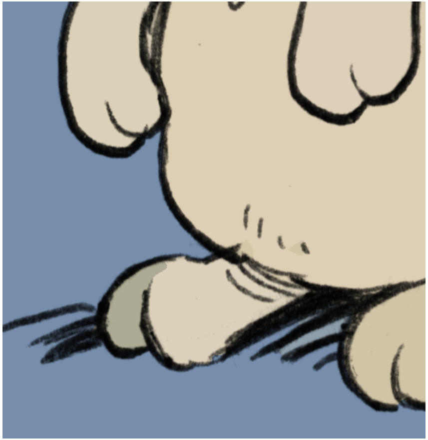


Auto-Clean Mode

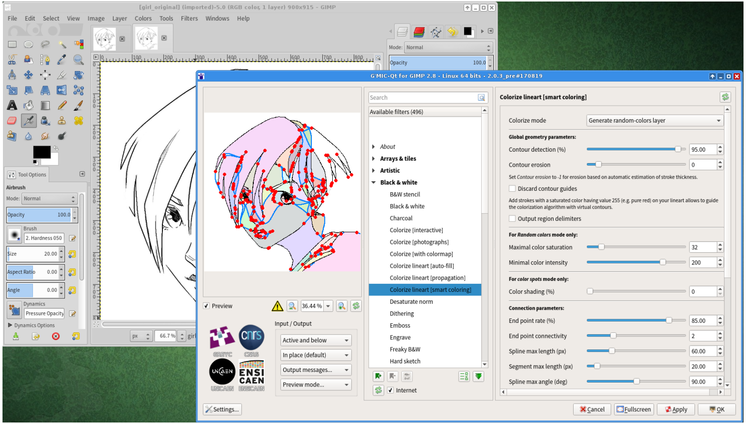




Auto-Clean Mode



Smart Colorize, a G'MIC-Qt Filter

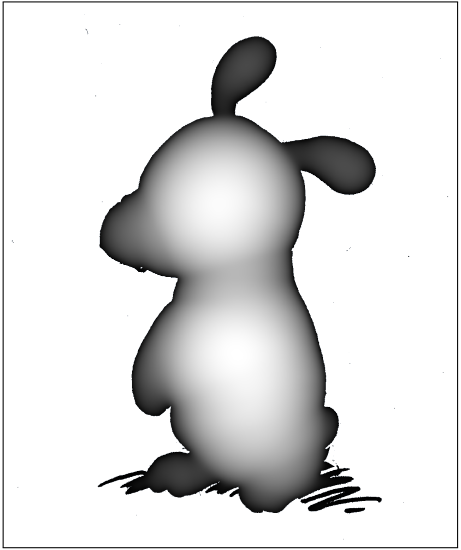


"A Fast and Efficient Semi-Guided Algorithm for Flat Coloring Line-Arts.",
S. Fourey, D. Tschumperlé, and D. Revoy.

EUROGRAPHICS International Symposium on Vision, Modeling and Visualization 2018, Stuttgart, October 2018.

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

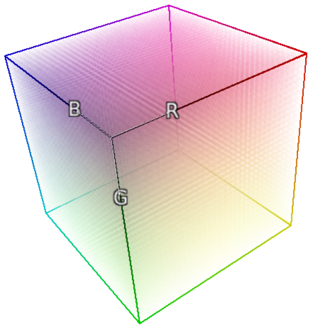
Collaboration with a photographer: Pat David

`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 $3D$

$$\boxed{\forall (x,y), \quad J_{(x,y)} = F(I_{(x,y)})}$$

(b) Original color image I

(c) Image J, after transformation $\mathbf{F}$

Examples of *CLUT*-Based Transformations

Original image

“60’s”

“Color Negative”

“Orange Tone”

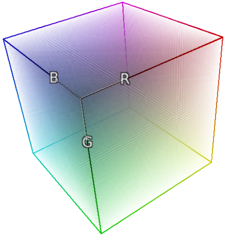
“Ilford Delta 3200”

“Backlight Filter”

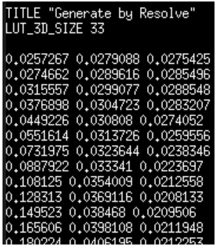
“Bleach Bypass”

“Late Sunset”

Standard Ways of Storing a *CLUT*

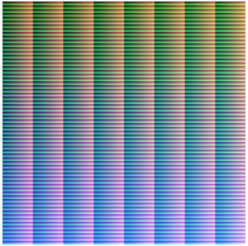


a) a *CLUT* is a $3D$ dense color volume



```
TITLE "Generate by Resolve"
LUT_3D_SIZE 33

0.0257267 0.0279088 0.0275425
0.0274662 0.0289616 0.0285496
0.0315557 0.0299077 0.0288548
0.0376898 0.0304723 0.0283207
0.0449226 0.030808 0.0274052
0.0551614 0.0313726 0.0259556
0.0731975 0.0323644 0.0238346
0.0887922 0.033341 0.0223697
0.108125 0.0354009 0.0212558
0.128313 0.0369116 0.0208133
0.149523 0.038468 0.0209506
0.165606 0.0398108 0.0211948
```

b) Storage as a `.cube`



c) Storage as a `.png`

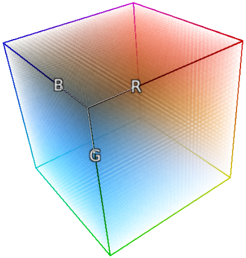
In both cases, lossless compression, but restricted to small sizes:

1. `.cube` file: *ASCII* zipped format (*CLUT* $64^3 \approx$ 1 Mo)
2. `.png` file: $2D$ image (*CLUT* $64^3 \approx$ 64 to 100 Ko)

⇒ **Issue:** Allow a large-scale distribution of *CLUTs* (**800+**), by limiting the *CLUT* storage as much as possible.

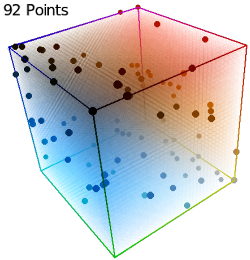
Our Approach: CLUT Compression

Compression: Let $\mathbf{F} : RGB \rightarrow RGB$ 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.



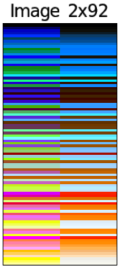
Original CLUT $\mathbf{F}$

$\longrightarrow$



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

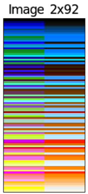
$\longrightarrow$



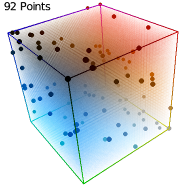
Storage of keypoints in compressed form

Our Approach: CLUT Decompression

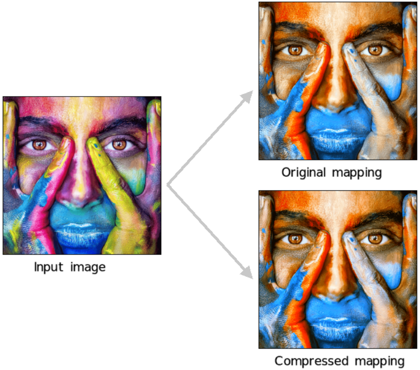
Decompression: A $3D$ interpolation based on anisotropic diffusion *PDEs* is applied to $\mathcal{K}$ in order to generate a reconstructed CLUT $\widetilde{\mathbf{F}}$ visually close to $\mathbf{F}$.



Storage of keypoints in compressed form

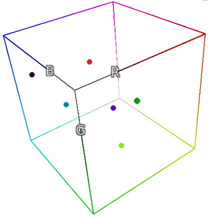


Decompressed CLUT $\widetilde{\mathbf{F}}$

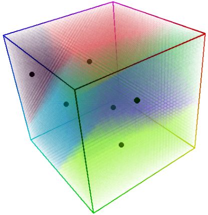


Goal: No perceptual differences between the two applied transformations (original/compressed)

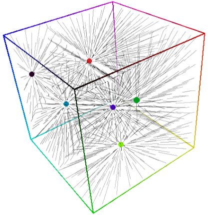
Reconstruction Principles



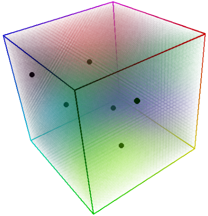
(a) Set $\mathcal{K}$ of known keypoints



(b) Initial state $\mathbf{F}_{\mathrm{t=0}}$
(*e.g., smoothed Voronoï* $3D$)



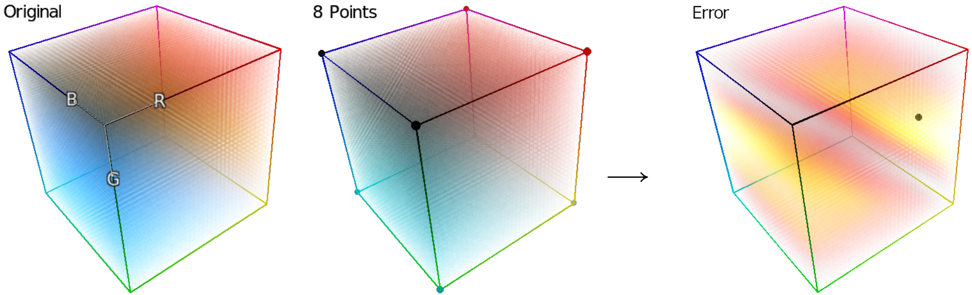
(c) Diffusion orientations η



(d) State at convergence (*PDE* Solution)

Compression: Generation of Keypoints

1. Initialization of $\mathcal{K} = \{(\mathbf{X}_k, \mathbf{F}_{(\mathbf{X}_k)} \mid k = 1 \ldots 8\}$ (the 8 vertices of the cube).

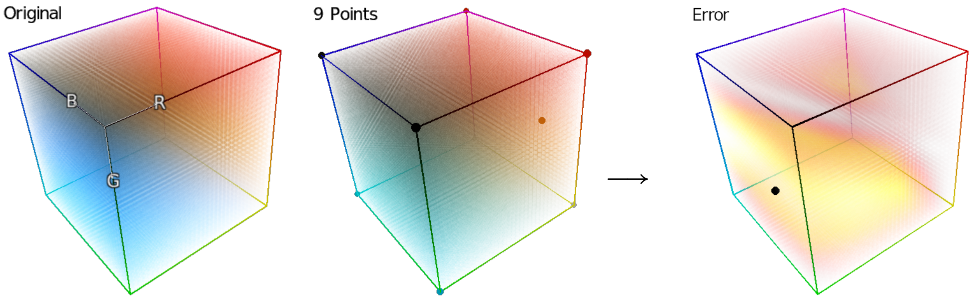


⇒ Calculation of the pointwise (3D) reconstruction error:

$\text{Err}_{(\mathbf{X})} = \Delta E(\mathbf{F}_{(\mathbf{X})}, \widetilde{\mathbf{F}}_{N(\mathbf{X})})$.

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

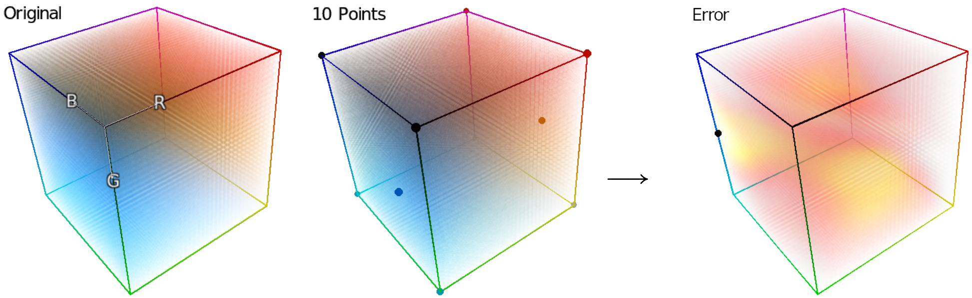


⇒ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

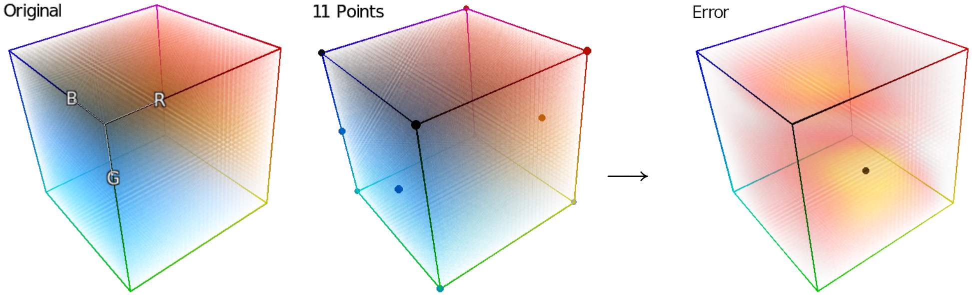


$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

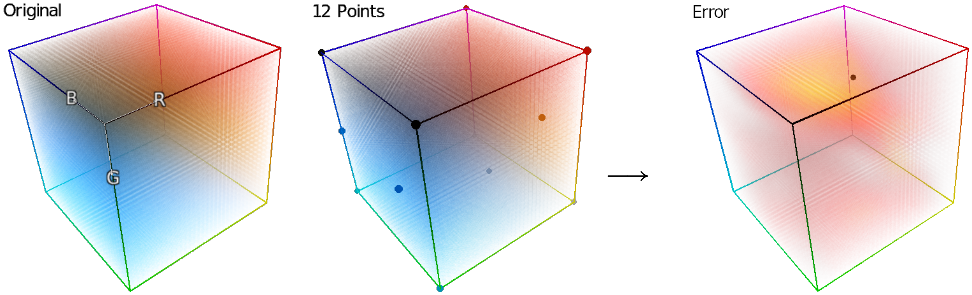


⇒ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

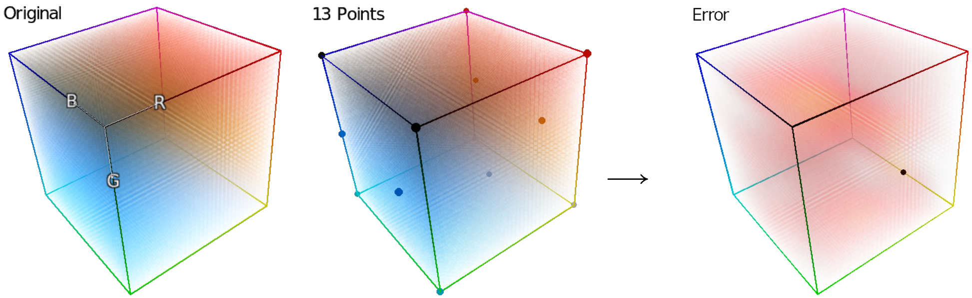


⇒ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

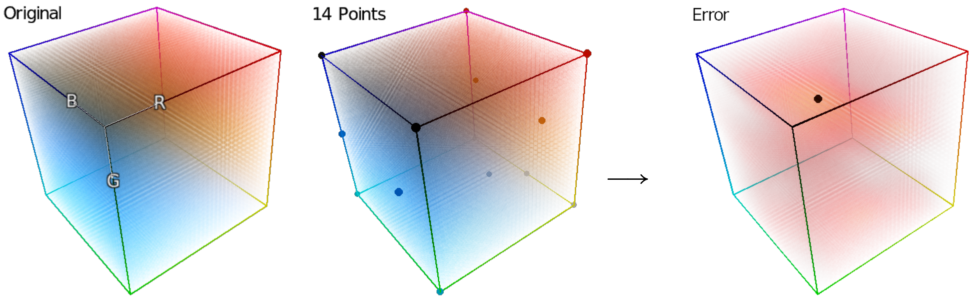


$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

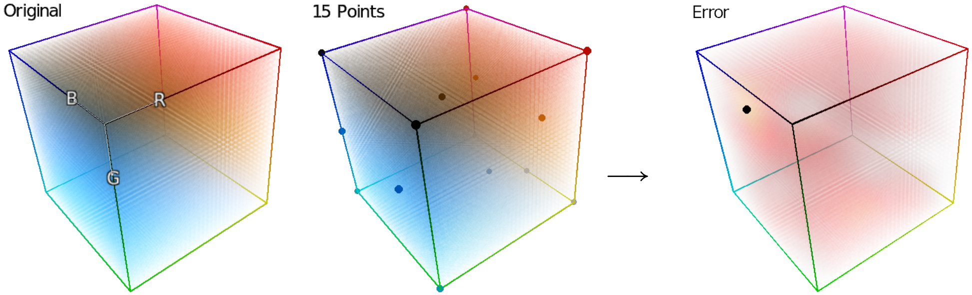


$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

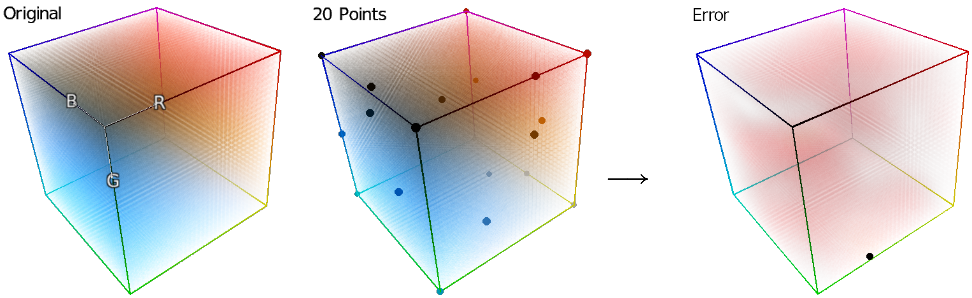


$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

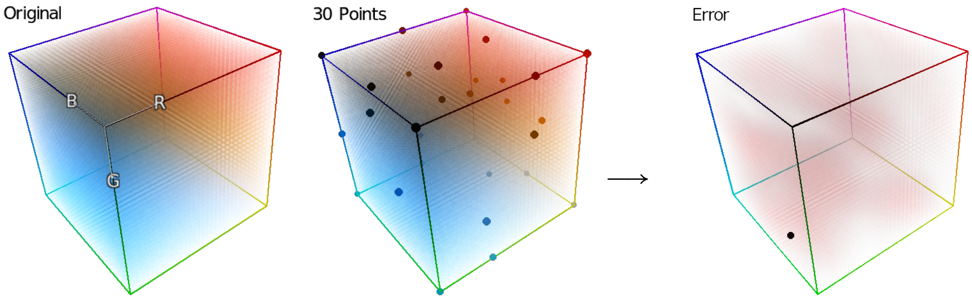


$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

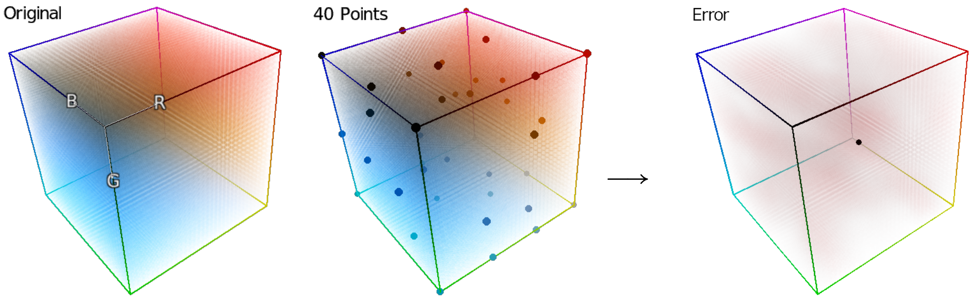


$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

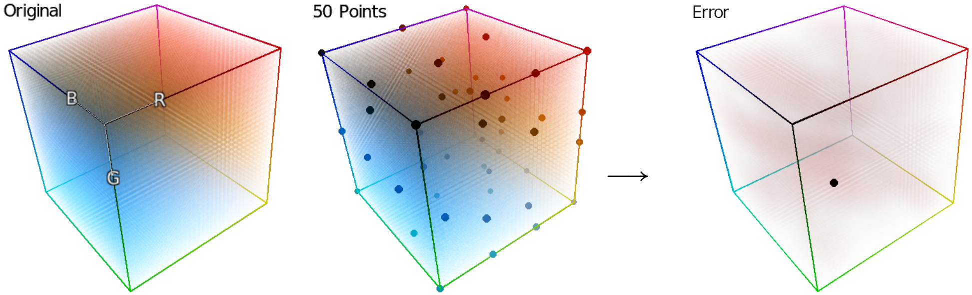


⇒ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

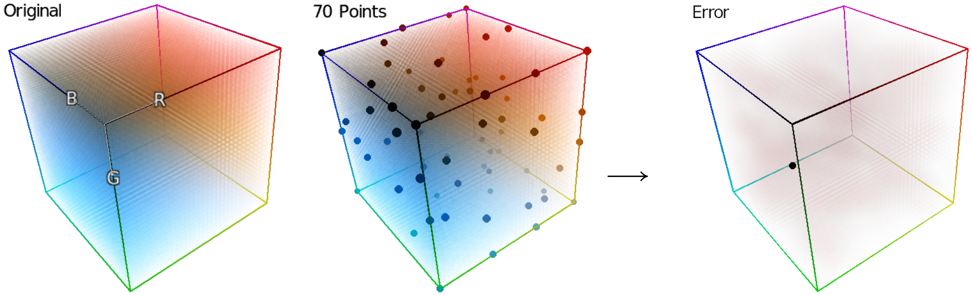


$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

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 $E_{\text{avg}} > \Delta E_{\text{avg}}$.

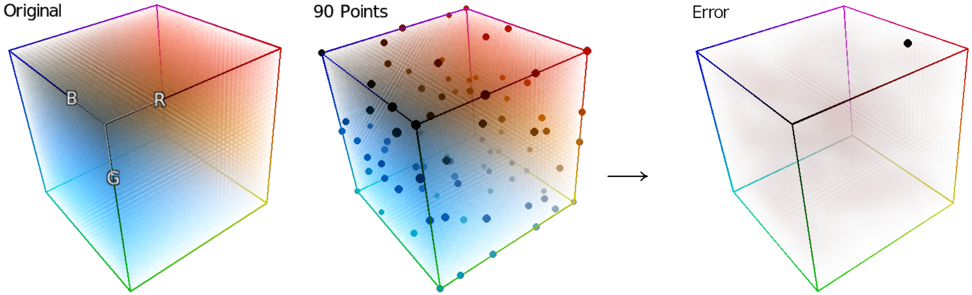


$\Rightarrow$ Calculation of the pointwise (3D) reconstruction error:

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

Compression: Generation of Keypoints

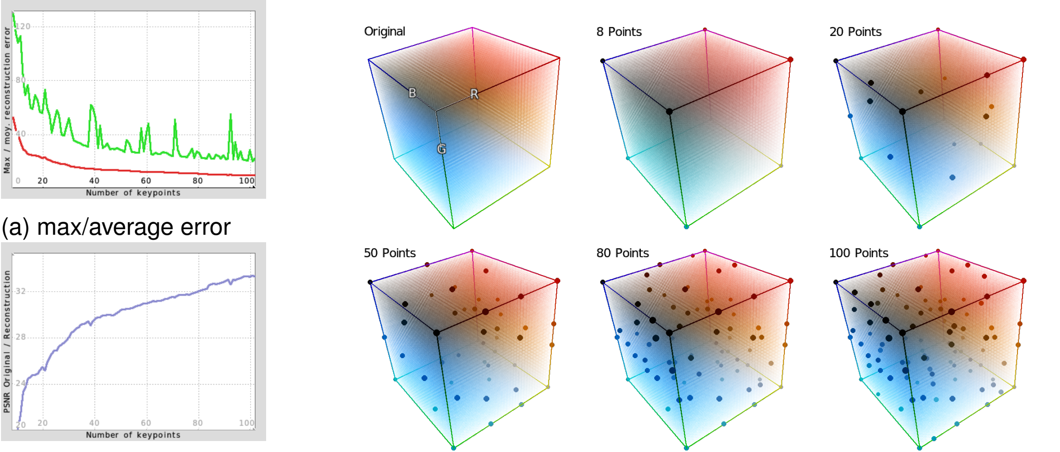
2. Iterative addition into $\mathcal{K}$ of the keypoints with maximum reconstruction error, while $E_{\text{max}} > \Delta E_{\text{max}}$ or $E_{\text{avg}} > \Delta E_{\text{avg}}$.



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

$$\text{Err}_{(\mathbf{x})} = \Delta E(\mathbf{F}_{(\mathbf{x})}, \widetilde{\mathbf{F}}_{N(\mathbf{x})}).$$

Adding Keypoints for *CLUT* Compression



(a) max/average error

(b) PSNR evolution

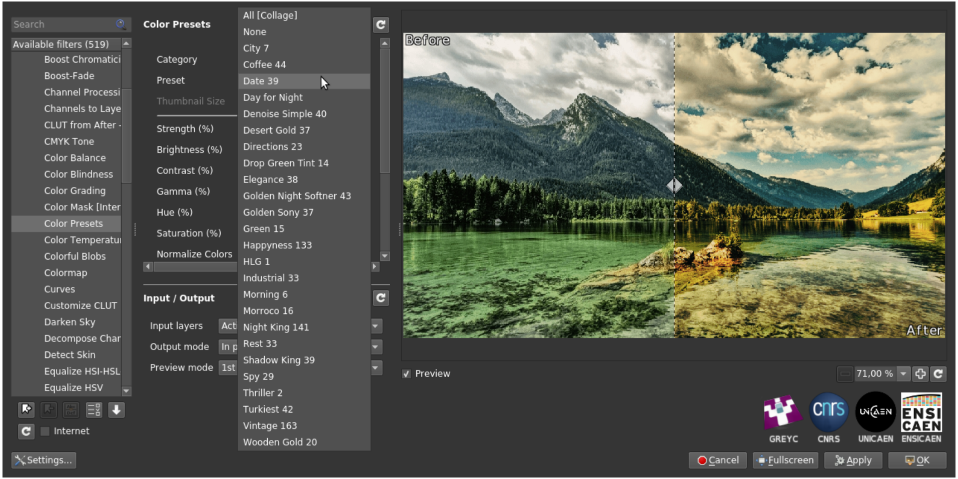
(c) Iterative addition of keypoints

CLUT Compression Results

CLUT name	`Bourbon 64`	`Faded 47`	`Milo 5`	`Cubicle 99`
Resolution	16^3	32^3	48^3	64^3
Size in `.cube.zip`	23.5 Kb	573 Kb	3 Mb	1.2 Mb
Size in `.png`	**3.7 Kb**	**22 Kb**	**72 Kb**	**92 Kb**
Number of Keypoints	562	294	894	394
PSNR	45.8 dB	45.6 dB	45 dB	45.2 dB
Compression time	28 s	92 s	1180 s	561 s
Decompression time	67 ms	157 ms	260 ms	437 ms
Keypoints in `.png`	**1.9 Kb**	**1.5 Kb**	**4.2 Kb**	**1.9 Kb**
%cRate / `.cube.zip`	92.1%	99.7%	99.8%	99.8%
%cRate / `.png`	49.5%	93.3%	94.2%	98%

- 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.
  SIAM Journal on Imaging Sciences (SIIMS), Vol. 13, Issue 3, 1511-1535, September 2020.

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** .

G'MIC
GREYC's Magic for Image Computing
A Full-Featured Open-Source Framework for Image Processing

https://gmic.eu

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