G'MIC: An Open-Source Self-Extending Framework for Image Processing

David Tschumperlé [1¶], **Sébastien Fourey** [1], **and Garry Osgood** [2]

1 GREYC Lab (IMAGE Team), CNRS, Normandie Univ, UNICAEN, ENSICAEN, F-14000 Caen, France **2** Independent researcher, New York City, USA ¶ Corresponding author

DOI: 10.21105/joss.06618

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @jamesrhester
- @Smattr
- @NicholasSynovic

Submitted: 13 November 2023
Published: 09 January 2025

Abstract

We present G'MIC, an open-source self-extending framework that defines an original concise scripting language for writing possibly complex image processing operators and pipelines. G'MIC provides several user interfaces allowing for the manipulation of digital images, adapted to different levels of user expertise, either from the command line, or as a C/C++ library, or as a user-friendly graphical plug-in that extends the capabilities of popular digital image retouching applications, such as *GIMP*, *Krita*, *Photoshop*, *Affinity Photo*, and others.

Keywords

Image Analysis and Processing, Scripting Language, User Interfaces.

Statement of Need

Context

Intrinsic to G'MIC's design are means to map image processing pipelines to commands, advancing the tool as a self-extending language. Primal command pipelines may be further assembled into those having wider remits, these suitably named to bespeak their extended purposes and available for succeeding command prototyping.

G'MIC is distributed under the CeCILL license. The core language projects several user interfaces to convert, process, or visualize *image datasets*. Allied with the pipeline toolset, G'MIC embodies a highly flexible image model, ranging from 1D signals to 3D+t sequences of multi-spectral volumetric images, hence including 2D color images. This makes it a versatile tool for image processing, with a wide range of applications in research, industrya and graphic design.

History and Motivation

The G'MIC project was initiated in 2008 by research scientists of the IMAGE team at the *GREYC* laboratory, a public research lab in France. Their area of research focuses on development of image processing algorithms.

To that end, they first began developing CImg (Tschumperle et al., 2023), beginning in 1999 and continuing to the present. CImg is an open-source C++ library for generic image processing, which means a library that is able to address structurally diverse imagery: photographs, multi-spectral images, medical images (MRI, X-ray, tomography, etc.), animations, among others.

That said, CImg exhibits certain limitations for everyday research work:

1. When one simply wants to apply a predefined `CImg` algorithm to an image, one needs to write a small, C++ program. It is only a few lines long, but still must be compiled and linked before it can be executed. In the context of research work, such mechanics are distractions. Being able to run those algorithms directly from the command line is tempting.
2. Over time, a large number of these purpose-specific programs have accumulated. They are not broadly useful for integration into `CImg` and have become an unruly "collection" of specialized algorithms. By design, they cannot be easily distributed and are difficult to maintain.

These limitations motivated `G'MIC`'s development in 2008. Two design objectives came to the fore:

1. Enable *pipelines of image processing algorithms* that may be directly invoked from the command line, without requiring compilation/linking steps.
2. Gather the implementation of specialized algorithms in a single location, facilitating their evolution, maintenance and distribution.

These objectives, in combination with a desire to write new image processing pipelines and algorithms in the most flexible and concise way possible, gave rise to the idea of *self-extension*. All these objectives led initially to the development of a specialized scripting language, the `G'MIC` language, and its associated interpreter.

Related Software

- **Command-line Interfaces:**

The CLI tool `gmic` was originally inspired by *ImageMagick* (ImageMagick Studio LLC, 2023) and *GraphicsMagick* (GraphicsMagick Group, 2023), particularly the idea of being able to manipulate digital images from a shell. The main differences between `G'MIC` and *ImageMagick/GraphicsMagick* are that:

1. The type of images processed is more diverse in `G'MIC`.
2. The possibilities offered by the scripting languages associated with each project are more extensive in `G'MIC`. This makes it possible to have conditions, loops, and multi-threaded pipelines, without having to resort to an external scripting tool, such as `sh`.

- **Image Filter Collections:**

There are also related software packages offering predefined filters to be applied to images. Popular examples are Mathmap (Probst, 2009), Filter Forge (Ashbrook, 2018), and Pixelitor (Balázs-Csíki, 2023). While these software packages allow the user to create their own image processing pipeline, their use case is restricted to the provided graphical user interfaces, with limited scripting possibilities.

Framework Environment

Core Components

The current `G'MIC` framework architecture is depicted below.

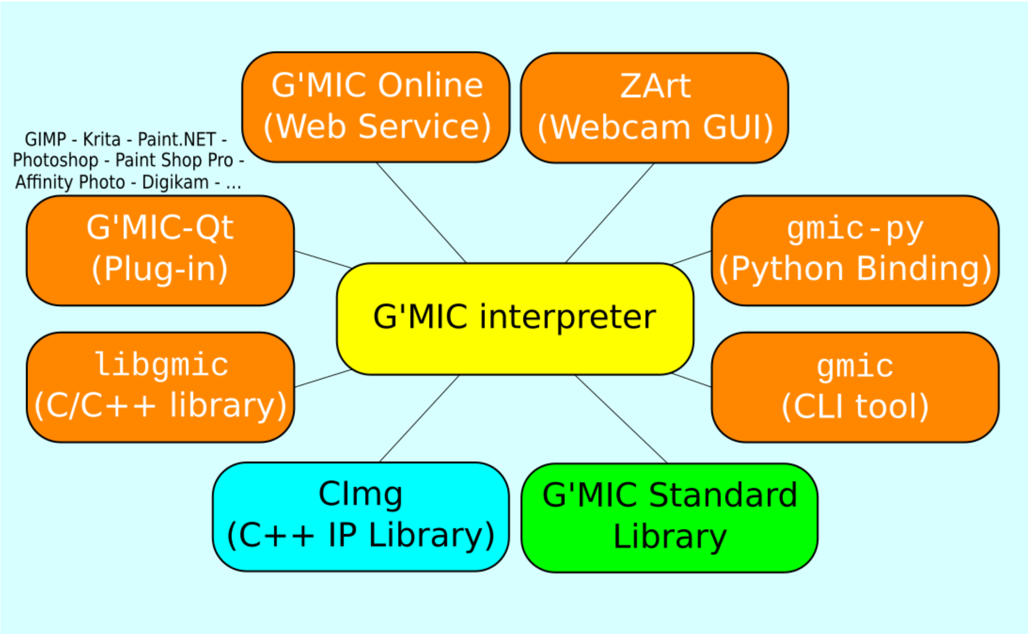


Figure 1: Framework architecture.

It revolves around a central component: the **`G'MIC` scripting language interpreter** (yellow), which uses the native functionalities of the **`CImg` library** (implemented in C++, blue), but relies also on a set of commands, written in the `G'MIC` language themselves, constituting a ***standard library* (`stdlib`)** for the framework (green). The other components (orange) stand for the various user interfaces provided by the framework. More than 1000 distinct commands are currently implemented, covering a large portion of image processing needs.

The interpreter lets the user implement their own scripts, for tasks as varied as writing image filters or generative algorithms, or creating user interfaces for image manipulation.

User Interfaces

On top of the interpreter are the user interfaces. Several types of UI are implemented, adapted to varying degrees of user expertise:

- **`gmic`**, a command-line tool to control the `G'MIC` interpreter from a terminal (Fig. 2).
- ***G'MIC-Qt***, a *Qt*-based (Qt, 2020) graphical interface intended to be used as a plug-in for digital image retouching software, such as *GIMP*, *Krita*, *DigiKam*, *Photoshop*, *Affinity Photo* and others (Fig. 3).
- **`G'MIC` Online**, a website where users can upload color images and apply *G'MIC-Qt* filters on them.
- **`libgmic`** and **`libcgmic`**, C++ and C libraries respectively, which basically provide simple C/C++ APIs to run `G'MIC` pipelines on a set of input images.
- ***ZArt***, a *Qt*-based interface used mainly for demonstration purposes, which applies `G'MIC` filters on streamed webcam images in real-time.

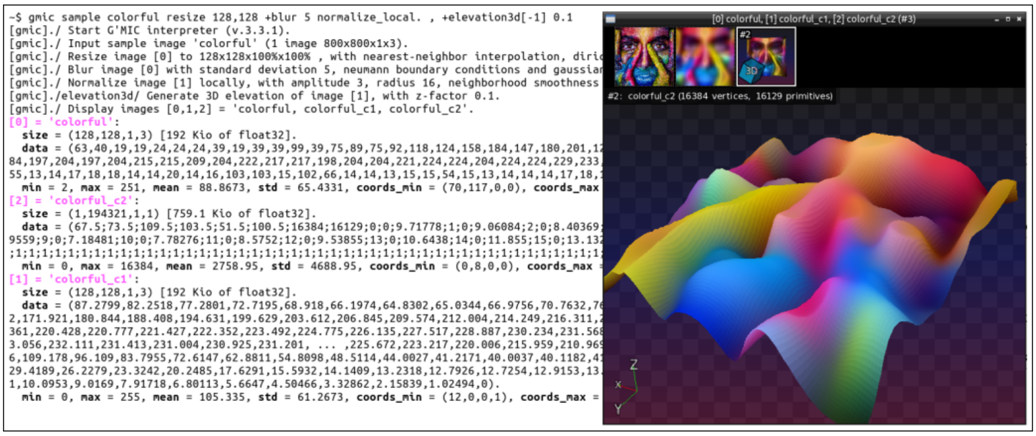


Figure 2: The command-line interface `gmic`.

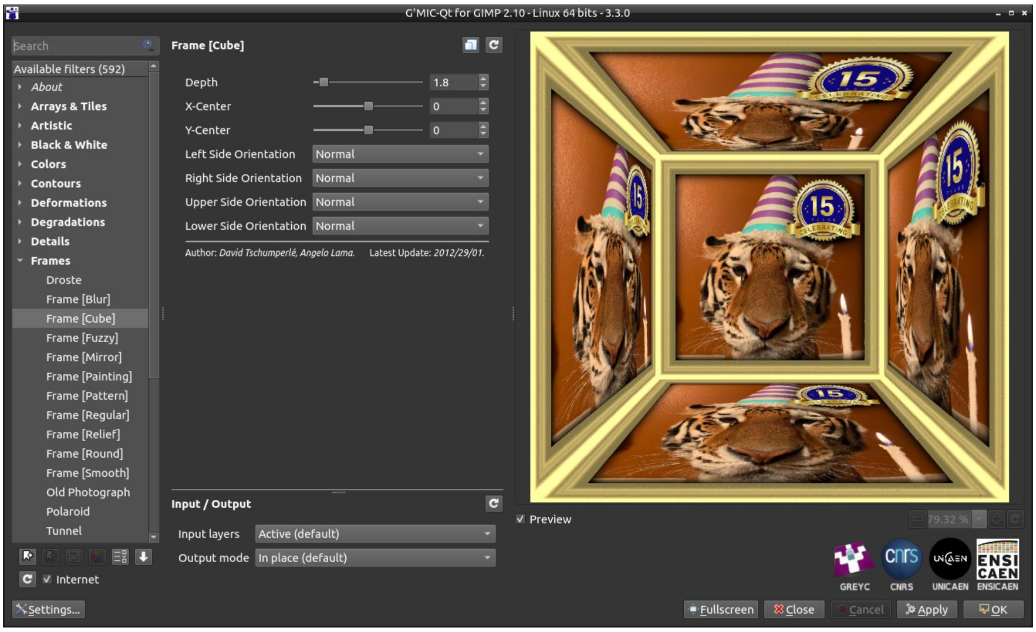


Figure 3: The *G'MIC-Qt* plug-in.

Visibility and Community

`G'MIC` has been developed since 2008, at the *GREYC* laboratory. The project web page is https://gmic.eu. This website brings together a range of resources, from software download links to documentation and tutorial pages.

The core features of the `G'MIC` interpreter are developed by David Tschumperlé and the *G'MIC-Qt* plug-in by Sébastien Fourey, both being permanent researchers at *GREYC*. The other contributors (for documentation, creation of new filters, or implementation of other user interfaces) can be found on the software's forum, hosted by *Pixls.Us*, an association that promotes the use of open-source software dedicated to photography and image creation.

The `G'MIC` source code is available on these various GitHub repositories: `gmic` (interpreter), `gmic-qt` (plug-in) and `gmic-community` (external contributions, documentation).

The Journal of Open Source Software

Examples of Research Work Conducted With `G'MIC`

To demonstrate the utility of `G'MIC` for research, we provide several examples of image processing tasks conducted using `G'MIC` for algorithm development and prototyping. For each example, we reference its associated research publication.

- **Patch-Based Image Inpainting**:

`G'MIC` has been used to design and implement an original patch-based image *inpainting* algorithm in (Buyssens et al., 2015) (Fig. 4).

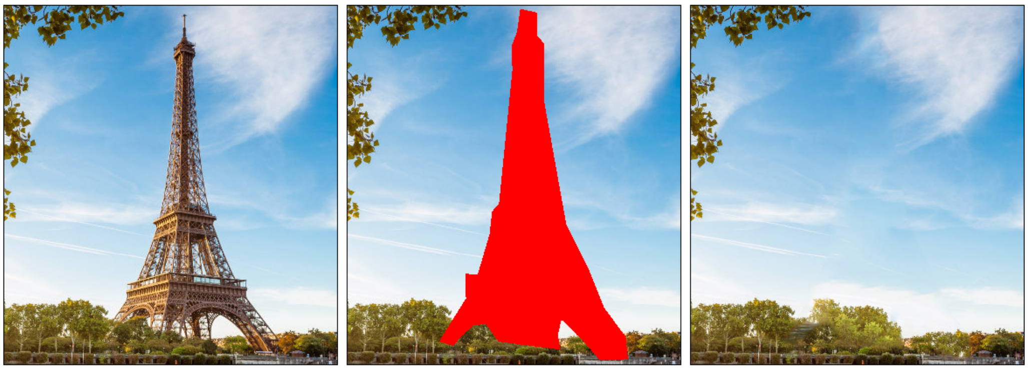
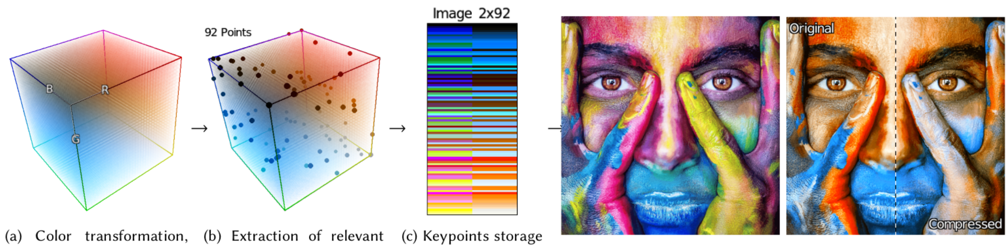


Figure 4: Left: input image. Middle: user-defined mask. Right: result of `G'MIC` inpainting.

- **Color LUT Compression**:

We used `G'MIC` to handle the problem of 3D *CLUTs* compression, for the efficient storage of generic color transformations (Tschumperlé et al., 2020). More than 1100 *CLUTs* are thus provided in `G'MIC`, requiring only 4 MB of data storage (Fig. 5).



(a) Color transformation, modeled as a 3D *CLUT* (b) Extraction of relevant color keypoints (c) Keypoints storage as a tiny color image (d) Original image (e) Image after transformation

Figure 5: `G'MIC` color *LUT* compression: a *CLUT* (a) is analyzed, relevant keypoints are deduced and stored (b,c). A perceptual metric ensure that the difference between original/compressed CLUTs are imperceptible.

- **Semi-automatic Colorization of Line Arts**:

Colorizing line art drawings is a problem that illustrators are familiar with, as traditional digital tools (e.g., *Bucket Fill*) do not always work well, e.g., when lines are anti-aliased or contain gaps in the drawing. In (Fourey et al., 2018), we describe a *"Smart coloring"* algorithm, implemented in `G'MIC`, that analyzes the geometry of the contours and automatically deduces a reasonable flat-colored layer from a user-defined set of colored strokes (Fig. 6).

The Journal of Open Source Software

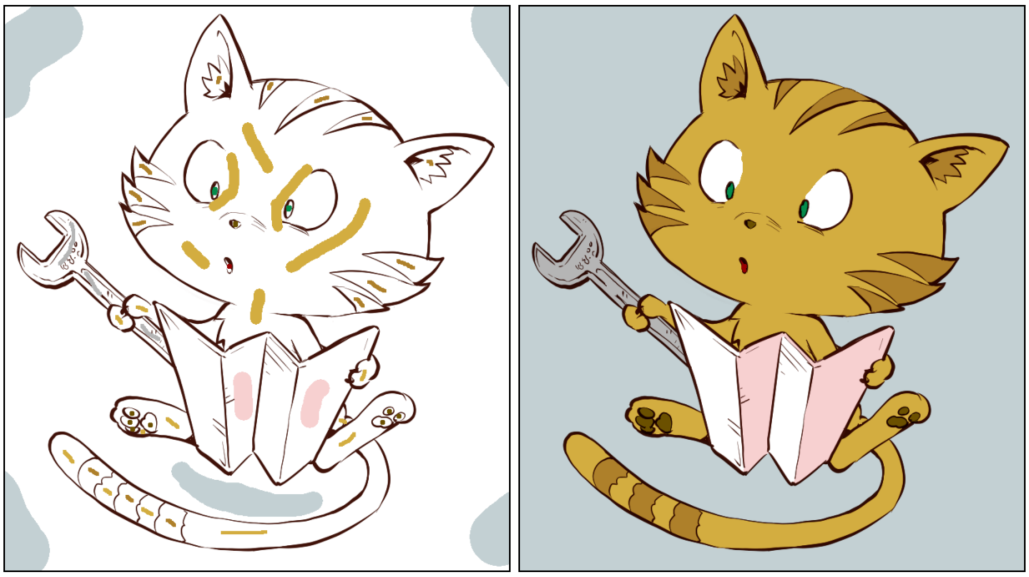


Figure 6: Color spot extrapolation for lineart colorization.

Note that this colorization algorithm has been subsequently implemented natively in *GIMP* (GIMP, 2018).

- **Automatic Illumination of Flat-colored Drawings**:

In a similar vein, we have designed an algorithm to automatically illuminate flat-colorized drawings by generating a light/shadow layer above a flat-colored layer (Tschumperlé et al., 2022) (Fig. 7).

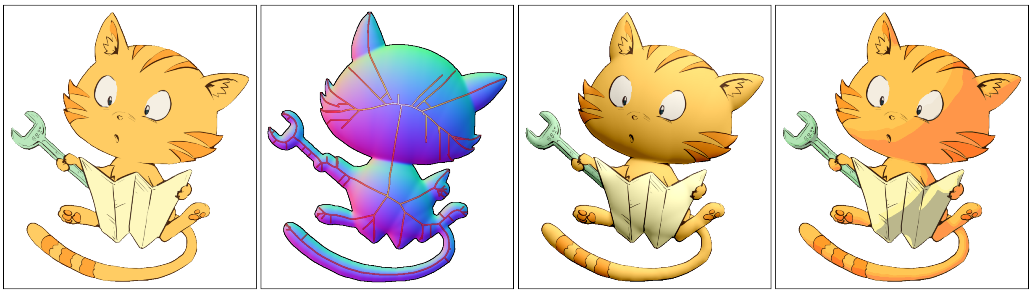


Figure 7: Left: input image, Middle-left: estimated 3D normals. Right: examples of automatic illuminations obtained with different parameters.

- **Patch-Based Image Style Transfer**:

Image stylization consists of transforming an input image to give it a pictorial style close to that of a second image (style image). In 2022, we successfully developed a patch-based multi-scale algorithm with low algorithmic cost (Samuth et al., 2022), which is now a part of `G'MIC` (Fig. 8).

Figure 8: `G'MIC` style transfer. Input image (top left) is stylized according to different style images (top row).

- **Debanding of Astronomical Images**:

`G'MIC` is used in the astronomy research community, in particular for processing images from the James Webb Space Telescope, which exhibits band frequency noise (efficiently mitigated with `G'MIC` filter **Banding Denoise**). `G'MIC` has been cited in (Ray et al., 2023), where images from protostar *HH211* were processed. One of those made the cover of *Nature* of October 2023 (Fig. 9).

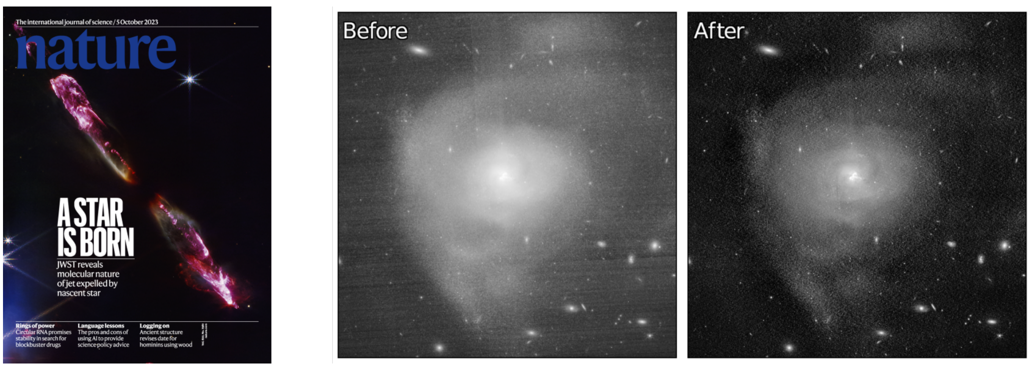


Figure 9: Left: image of protostar, processed with `G'MIC` (courtesy of Mark McCaughrean/ESA). Right: effect of the `G'MIC` **Banding Denoise** algorithm on a JWST image (courtesy of Judy Schmidt).

Acknowledgments

We would like to express our deepest gratitude to the developers, contributors and donors, both regular and one-off, all over the world. We are very grateful to the "CNRS Sciences Informatiques" institute, which helped accelerate G'MIC's growth by funding a development engineer for two consecutive years, and to the heads of the GREYC laboratory, who have supported this project from the outset. We would also like to thank our amazing community of users, who regularly provide us with beautiful/fun/innovative creations in which G'MIC filters have been used.

In Memoriam

This article is dedicated to Sébastien Fourey, co-developer of the G'MIC project, who passed away in October 2024. He was the kindest, most caring person you'll ever meet, as well as being extremely competent and passionate about computing, algorithms, and software development as a whole. He always believed in the magic of free software. The world has lost a talented developer, a great researcher and teacher, but above all a person of great humanity. Rest in peace, Sébastien.

References

Ashbrook, B. (2018). Filter forge. *PSA Journal*, *84*(2), 8–10.

Balázs-Csíki, László. (2023). *Pixelitor* (Version 4.3.0). https://pixelitor.sourceforge.io/index.html

Buyssens, P., Daisy, M., Tschumperlé, D., & Lézoray, O. (2015). Exemplar-based inpainting: Technical review and new heuristics for better geometric reconstructions. *IEEE Transactions on Image Processing*, *24*(6), 1809–1824. https://doi.org/10.1109/tip.2015.2411437

Fourey, S., Tschumperlé, D., & Revoy, D. (2018). A fast and efficient semi-guided algorithm for flat coloring line-arts. *International Symposium on Vision, Modeling and Visualization*.

GIMP. (2018). *Lineart Bucket Fill*. https://developer.gimp.org/core/algorithm/line-art-bucket-fill/

GraphicsMagick Group. (2023). *GraphicsMagick* (Version 1.3.40). http://www.graphicsmagick.org/

ImageMagick Studio LLC. (2023). *ImageMagick* (Version 7.0.10). https://imagemagick.org

Probst, M. (2009). *The MathMap image processing application* (Version 1.3.5). https://www.complang.tuwien.ac.at/schani/mathmap/

Qt. (2020). *A cross-platform software for creating graphical user interfaces* (Version 5). https://www.qt.io

Ray, T. P., McCaughrean, M. J., Caratti o Garatti, A., Kavanagh, P., Justtanont, K., Dishoeck, E. F. van, Reitsma, M., Beuther, H., Francis, L., Gieser, C., & others. (2023). Outflows from the youngest stars are mostly molecular. *Nature*, *622*(7981), 48–52. https://doi.org/10.1038/s41586-023-06551-1

Samuth, B., Tschumperlé, D., & Rabin, J. (2022). A patch-based approach for artistic style transfer via constrained multi-scale image matching. *2022 IEEE International Conference on Image Processing (ICIP)*, 3490–3494. https://doi.org/10.1109/icip46576.2022.9897334

Tschumperle, D., Tilmant, C., & Barra, V. (2023). *Digital image processing with C++: Implementing reference algorithms with the Cimg Library (1st Ed.)*. CRC Press. https://doi.org/10.1201/9781003323693

Tschumperlé, D., Porquet, C., & Mahboubi, A. (2022). Automatic illumination of flat-colored drawings by 3D augmentation of 2D silhouettes. *2022 IEEE International Conference on Image Processing (ICIP)*, 371–375. https://doi.org/10.1109/icip46576.2022.9897386

Tschumperlé, D., Porquet, C., & Mahboubi, A. (2020). Reconstruction of smooth 3D color functions from keypoints: Application to lossy compression and exemplar-based generation of color LUTs. *SIAM Journal on Imaging Sciences*, *13*(3), 1511–1535. https://doi.org/10.1137/19m1306798