A Fast and Efficient Semi-guided Algorithm for Flat Coloring Line-arts

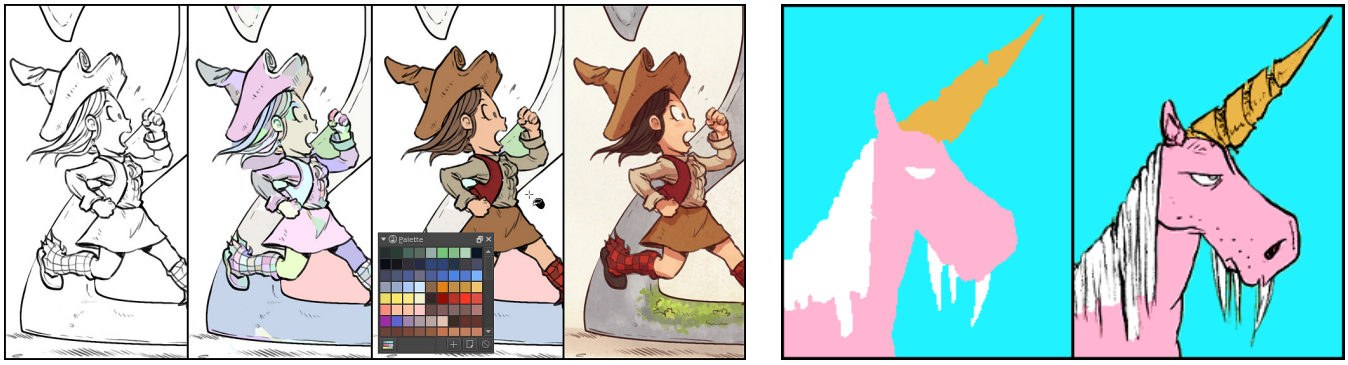
Sébastien Fourey, David Tschumperlé, David Revoy

Accepted in EUROGRAPHICS International Symposium on Vision, Modeling, and Visualization (2018)

A Fast and Efficient Semi-guided Algorithm for Flat Coloring Line-arts

S. Fourey[1], D. Tschumperlé[1], and D. Revoy[2]

[1]Normandie Univ., UNICAEN, ENSICAEN, CNRS, Caen, France
[2]Independent illustrator, Montauban, France



(a) (b) (c) (d) (e) (f)

Figure 1: *Colorizing a line-art image usually involves the creation of a* flat-color *layer. (a) Input line-art image. (b) Random colorization of connected white regions. (c) Image pre-colorized by the artist with* flat colors. *(d) Final colorization by the artist. (e) Layer with constant color areas produced by our algorithm. (f) Flat-colored drawing obtained by merging the line-art with the color layer.*

Abstract

We present a fast and efficient algorithm for the semi-supervised colorization of line-art images (e.g. hand-made cartoons), based on two successive steps: 1. A geometric analysis of the stroke contours, and their closing by splines/segments, and 2. A colorization step based on the filling of the corresponding connected components, either with random colors or by extrapolating user-defined color scribbles. Our processing technique performs image colorization with a similar quality as previous state of the arts algorithms, while having a lower algorithmic complexity, leading to more possible user interactivity.

CCS Concepts

• ***Computing methodologies*** → ***Image processing;*** *Non-photorealistic rendering; Shape analysis;*

1. Introduction

In the world of illustration, the task of colorizing a black and white line-art image (Fig. 1(a)) is usually done in two successive steps. First, the drawing is pre-colorized with *flat colors* only (Fig. 1(c)), i.e. by assigning a unique color to pixels of each distinct image region or object (the person being in charge of this specific task is a *flat color artist*). Second, the *colorist* improves this pre-coloring, adding shadows, lights and colorimetric ambience until the final colorization is achieved (Fig. 1(d)). In practice, the flat coloring step results in the creation of a new layer that contains only constant color areas, hence forming a colored partition of the plane (Fig. 1(e)). This color layer is then merged with the original *line-art* to render the flat-colored drawing (Fig. 1(f)).

Artists admit it: flat-coloring is a long and tedious process. "Classical" tools in digital painting software do not make this task easy: most of the region filling tools do not handle stroke discontinuities very well, neither they do with *anti-aliased* contours (Fig. 2(a)). It is then usual for artists to manually paint the flat colors with a brush on a separate layer, with all the precision problems that come along (particularly contour overflows, Fig. 2(b,c)).

It may even happen that the artist decides to explicitly constrain his drawing style, using for instance, aliased strokes in higher reso-

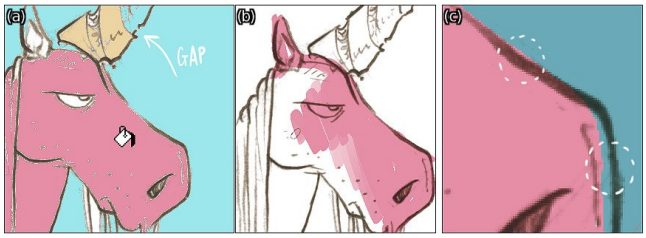


Figure 2: *Problems encountered for flat coloring using tools provided by classical digital drawing/painting.*

lution (rather than *anti-aliased*) and/or forcing himself to draw lines without "holes", just to ease the flat-colorization work afterwards.

In this paper, we propose an original technique for colorizing flat regions, producing results with a quality similar to optimization-based techniques (see related work, section 2), but with a much lower algorithmic complexity, enabling a possibly better interactivity and comfortable use for the user. The proposed algorithm is capable of handling drawing strokes with discontinuities, as well as *anti-aliased* contours (Section 3). It is based mainly on a fine analysis of the local geometry (specifically normal and curvatures) of the drawn lines (Section 4), followed by the closing of regions using splines curves or segments (Section 5). As a final step (Section 6), we make it possible to choose between three colorization modes: random coloring (Fig. 1(b)), colorization guided by markers placed by the user, or auto-cleaning of a roughly made color layer. These steps (described in the sequel) involve only low-complexity algorithms (linear time, or less) which makes the whole colorization procedure fast to render. In addition to its speed, we found out that the proposed algorithm is really robust regarding the choice of its different parameters: most experiments presented here were successfully conducted with the same working set of default parameters, despite the different drawing styles and resolutions of the considered input line-arts.

2. Related work

Some research papers have been proposing algorithms dedicated to the colorization of images with flat colors. It started with methods designed for usual photographs [LLW04], then more specifically for drawings [QST*05, QWH06, SDC09]. These approaches are all based on similar steps: the user has to place colored markers at some key locations on his drawing. These markers are then extrapolated for all pixels of the image, thanks to the *minimization of a global energy* that takes into account the geometry of the drawing strokes. Unfortunately, the algorithmic complexity of these iterative minimization processes is really important, thus tends to decreases the user interactivity when running on modest computers. That's particularly true when dealing with high-resolution images (a typical example would be the colorization of a A4 paper, with resolution of 5000×7000).

More recently, Sato et al. [SMYA14] proposed a semiautomatic method that propagates the colors of a reference colored image to a target one. This method is based on a segmentation step of both images which are then represented as graphs, so that colors may be propagated between corresponding regions thanks to graph matching.

Because flat coloring somehow requires the segmentation of the line-art into regions, it shares with the problem of vectorization of line drawings [NHS*13, BS18] the important issue of "leakage" at small curve gaps. However, because these works are also mainly concerned with line junctions disambiguation (in order to extract topologically sound structures), few vectorization related papers address the problem of non-watertight contours. For example, Favreau et al. [FLB16] make use of the so called "Trapped-ball region segmentation" algorithm [ZCZ*09] based on floodfilling and morphological operators. For the same purpose, Qu et al.'s [QWH06] colorization method based on level sets uses a modified speed function that drops down to zero nearby small gaps identified thanks to the gradient of the blurred image. (We use a similar idea in the definition of a priority function in section 6.2.)

Eventually, it is worth mentioning related works based on Convolutional Neural Networks like [ISSI16], although dedicated (limited ?) to photographs. Closer to our target images, Frans proposes in [Fra17] an architecture based on a first network, used to "infer" a color scheme from the input line-art, so that a second network may produce a final image from the input outline and the obtained color scheme. Quoting the latter: "*Due to memory constraints, we train on 256×256 images [...]*", we may insist on one of our main goals in this work, which is developing a *lightweight* flat coloring method.

3. Pre-Processing of the anti-aliased image

We consider an input line-art image $I : \Omega \to [0,255]$, with dimensions $w \times h$, so that the image is defined on the discrete domain $\Omega = \{0,\dots,w-1\} \times \{0,\dots,h-1\}$. Generally, the drawing is made of *anti-aliased* strokes (Fig. 1(a)), as artists use "smooth" brushes. The algorithm we propose here relies on a crucial step that closes the drawing strokes between, or from, detected key-points. However, instead of developing a pixel-value based approach for this closing stage, we have chosen to use a straightforward method dealing with a binarized version ($I_b : \Omega \to \{0,1\}$) of the input image I. We may thus make use of efficient processings considering precisely defined object borders; namely, *bicurves* presented in subsection 4.1. Furthermore, it should be noticed that the simplification introduced by the binarization occurs without loss of generality, as discussed in Section 7.

The pre-processing stage consists in a binarization of the *anti-aliased* image, followed by an estimate of the typical stroke width.

First, for $\theta \in [0,255]$, we define the binarized image I_b by a simple thresholding operation of I; namely $I_{b(x,y)} = \mathbb{1}_{I_{(x,y)} \leq \theta}$. Note that line-arts are usually made of black strokes over a white background, hence the use of an upper limit, and not a lower one.

For the sake of independence with respect to the image resolution, a second step allows to reduce the width of the strokes to a few pixels, if necessary, using a morphological erosion. The radius to be used for this erosion is set automatically by estimating the width of the strokes found in the drawing. For that purpose, an euclidean distance transform [MRH00] is first used to compute

the maximum value of the distance to the background for every 8-connected component of pixels of strokes. The median value of all these maximums eventually provides a good approximation of the stroke half-width. One should note that disconnections that may result from the morphological erosion step applied here, which remain rare, will be corrected anyway by the closing step to be applied thereafter.

4. Characterization of key-points

The following steps of the method rely on the detection of key-points which are located at the extremities of the strokes and, at the same time, on the estimation of a valid direction to extend these strokes (Fig. 5(c)). In order to characterize these key-points, we propose to use an estimation of the normal field and curvature values along the border of non-zero areas of I_b. Key-points are then characterized as the loci of extreme curvatures.

4.1. Normal and curvature estimation along borders

The normal field estimation which we use here relies on a geometric local averaging of so called *canonical normals* (Fig. 5(a)). It is inspired by the *image-based contextual shading* for voxel based volumes [CHRU85], which we extend to a larger neighborhood to obtain a better approximation, as suggested by [FM09], while restricting ourserlves to the 2D case. However, unlike the latter, the method we use does not extend the averaging kernel by recursive convolutions with a small one, but merely applies a single-pass convolution with a proper Gaussian kernel. This method is now described in details.

We first define precisely the notion of border, before providing details about the estimation of a normal field along this border (green vectors in Fig. 5(b)). Our definition of a border follows the one of *bicurves* found in [Ros74, Section 4]. Let $X \subset \Omega$ be the set of pixels with non-zero value in I_b, and let us denote $\overline{X} = \mathbb{Z}^2 \setminus X$. Classically, two pixels $p = (x_p, y_p)$ and $q = (x_q, y_q)$ are said to be 4-*adjacent* if $|x_p - x_q| + |y_p - y_q| = 1$, in which case they share an *edge*. The *border* of X, denoted by $\delta(X)$, is the set of edges shared by two 4-adjacent pixels, one of which belongs to X and the other one belongs to $\overline{X}$. Since an edge can be identified to an ordered pair of pixels, we have $\delta(X) = \{(p,q) \in X \times \overline{X}, p \text{ and } q \text{ are 4-adjacents}\}$. Unless stated otherwise, in what follows all edges are supposed to be part of the border of X, in other words they are *border edges*.

Following the description given in Fig. 3, it is then possible to define a symmetric and anti-reflexive binary relation $\mathscr{R}_e \subset \delta(X) \times \delta(X)$, such that every edge is in relation with exactly two other edges by $\mathscr{R}_e$: one for each of its extremities. The connected components of $\delta(X)$ following $\mathscr{R}_e$ (i.e. the equivalence classes of the transitive closure of $\mathscr{R}_e$) thus constitute the set of *border-components* of X. We have chosen the construction of $\mathscr{R}_e$ described by Fig. 3 in order to obtain border-components which separate an 8-connected component of X from one of the 4-connected components of $\overline{X}$. Here, "separate" means that any 4-connected path of pixels between two sets of pixels *crosses* the border. We indeed consider the strokes as a set of 8-connected components of non-zero pixels. Eventually, given the choice of an edge and one of its extremities, it is possible to provide every border-component with a complete parameterization $C = (e_0, \ldots, e_{n-1})$, $e_i \in \delta(X)$ where n is the number of edges of the component and $e_i \mathscr{R}_e e_{i+1 \bmod n}$ for all $i \in \{0, \ldots, n-1\}$ (see [Ros70, Section 6]).

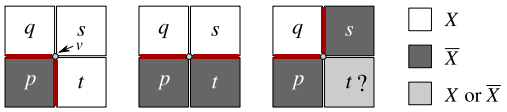


Figure 3: *Adjacency relation between edges. Given 4 pixels p, q, s and t such that $(p,q) \in \delta(X)$, a single edge incident to the point v is in relation with (p,q) by $\mathscr{R}_e$. Thus: $(p,q)\mathscr{R}_e(p,t)$ in case (a), $(p,q)\mathscr{R}_e(t,s)$ in case (b), and $(p,q)\mathscr{R}_e(s,q)$ in case (c) for any value of t.*

Definition 1 (Canonical normals) For any edge $e = (p,q)$ in the border of X, we define the *canonical normal of e*, denoted by $\mathbf{n}(e)$, as the unit vector $\mathbf{n}(e) = (x_q - x_p, y_q - y_p)$.

The set of all canonical vectors at the border of an object is depicted in Fig. 5(a).

Definition 2 (Estimated normal vector) Let $C = (e_0, \ldots, e_{n-1})$ be a parameterization of a border of X and $i \in \{0, \ldots, n-1\}$. We define the *estimated normal vector* at the edge e_i, denoted by $\tilde{\mathbf{n}}(e_i)$, as the vector $\tilde{\mathbf{n}}(e_i) = \frac{\mathbf{m}(e_i)}{\|\mathbf{m}(e_i)\|}$ where

$$\mathbf{m}(e_i) = \sum_{-L \le k \le L} e^{\frac{k^2}{L^2}} \mathbf{n}(e_{i+k \bmod n}) \qquad (1)$$

for an averaging kernel size $L \in \mathbb{N}^*$.

In our experiments, a kernel size of 11 edges ($L = 5$) proved to be sufficient for a good estimate. Note that if the border C contains less than $2L+1$ edges, the sum of equation (1) is restricted in order to take into account each edge only once. This estimation, based on a simple smoothing of the canonical normals, is precise enough to characterize afterwards the areas of high curvature in the border of strokes whose width is a few pixels.

Definition 3 (Estimated border curvature) Let $C = (e_0, \ldots, e_{n-1})$ be a parameterization of a border of X and $i \in \{0, \ldots, n-1\}$. We define the estimated signed curvature at the locus of the edge e_i, denoted by $\tilde{\kappa}(e_i)$, as follows:

$$\tilde{\kappa}(e_i) = \text{sign}(\det(\tilde{\mathbf{n}}(e_{i-1}), \tilde{\mathbf{n}}(e_{i+1}))) \frac{\|\tilde{\mathbf{n}}(e_{i+1}) - \tilde{\mathbf{n}}(e_{i-1})\|}{2} \qquad (2)$$

Property 1 For any border edge e, the estimated border curvature satisfies $|\tilde{\kappa}(e)| \le 1$.

From now on, we call *border pixel* a pixel of X which is 4-adjacent to a pixel of $\overline{X}$, i.e. a pixel in X which defines at least one border edge.

Definition 4 (Estimated normal at a border pixel) Let p be a border pixel, and $L(p)$ be the set of border edges of the form $(p,\cdot)$. We define $\tilde{\mathbf{n}}(p)$, the *normal vector associated with the pixel p*, as $\tilde{\mathbf{n}}(p) = \frac{\tilde{\mathbf{m}}(p)}{\|\tilde{\mathbf{m}}(p)\|}$, where

$$\tilde{\mathbf{m}}(p) = \sum_{e \in L(p)} \tilde{\kappa}(e)^2 \cdot \tilde{\mathbf{n}}(e) \qquad (3)$$

Definition 5 (Estimated curvature at a border pixel) Let p be a border pixel and $L(p)$ be the set of border edges of p. We define the *estimated curvature* at p, denoted by $\tilde{\kappa}(p)$ as follows:

$$\tilde{\kappa}(p) = \max_{e \in L(p)} \{\max(0, \tilde{\kappa}(e))\} \quad (4)$$

This curvature is thus defined as null at any border edge e such that $\tilde{\kappa}(e) < 0$. Indeed, the key-points to be detected afterwards being characterized by an extreme positive curvature, as depicted by red disks in Fig. 5(b), negative ones are not relevant. For example, in the case depicted in Fig. 4, the negative curvature at edge e in the inside part of the pixel curve should be ignored in the computation of the curvature at pixel p. Only the three other edges, with a positive curvature, should be taken into account.

4.2. Set of key-points

We have defined in the previous subsection the normal vector as well as the curvature associated with a border pixel of X. We may now define a first set of points $\mathcal{J}'$ containing the pixels that show an extreme positive curvature. Given a threshold θ_κ such that $0 < \theta_\kappa < 1$, we define

$$\mathcal{J}' = \{p \in X, \tilde{\kappa}(p) \geq \theta_\kappa\}. \quad (5)$$

However, the set $\mathcal{J}'$ is not necessarily made of isolated pixels but may contain several ones at a given actual extremity. Therefore, only one pixel is kept within each 8-connected components of $\mathcal{J}'$: the pixel with highest estimated curvature. We define $\mathcal{J}$, the remaining set of pixels (depicted in red in Fig. 5(c)), as the set of key-points to be considered in the stroke-closing stage. The result of this detection step is also depicted on a larger example, as enlarged red dots, in Fig. 10(e). The stroke closing stage will be described in the next subsection.

5. Closure of the strokes

Given the set of key-points $\mathcal{J}$, we propose an algorithm for closing the drawing strokes, as suggested by the Gestalt principle of good continuation, by combining two methods:

1. Link together some pairs of the key-points using digitized spline curves (Section 5.1). The latter being parameterized by the estimated normals at the two extremity points (see the blue curve in Fig. 5(c)).

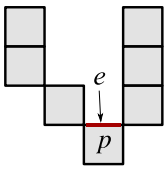
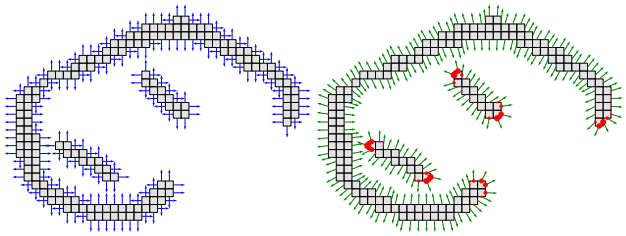
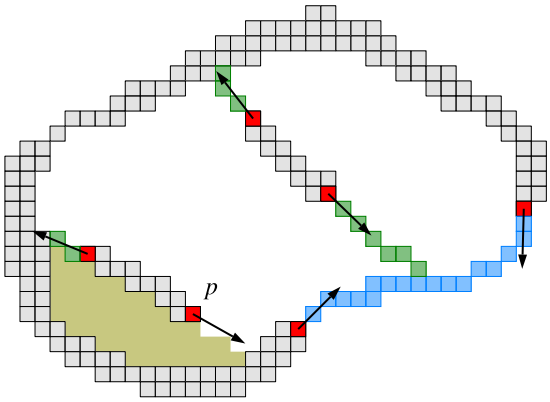


Figure 4: *At the locus of pixel p, the border of this pixel curve has a curvature which is both negative (in e) and positive (in other edges of p). We associate a positive curvature to the pixel p by ignoring the negative curvature of e (see equation (4)).*



(a) Canonical normals, in blue, at each border-edge.

(b) Normals (in green) and curvature (positive in red, negative in blue) estimated at each border-edge.



(c) Stroke closing using splines (in blue) and straight segments (in green). The detected key-points are depicted in red, together with their estimated normal vectors.

Figure 5: *Illustration of several steps of the stroke closing method.*

2. Extend some of the strokes by drawing a straight line segment from chosen key-points, in the direction of the estimated normal at these points (Section 5.2). Fig. 5(c) shows three such lines, in green.

From now on, we call *closing stroke* a sequence of pixels which is the digitization of a spline curve or a straight line segment. Such a stroke will be added to the binary line-art image I_b if some conditions are satisfied. It is indeed important to wisely choose the candidate keypoints for closing strokes, so that the number of closed regions is kept reasonable. The required conditions will be described in the next subsections. Furthermore, in what follows, we will denote by I_c the image obtained by drawing closing strokes in I_b.

5.1. Strokes closing using splines

First, we describe here the steps and notions used for the connection using splines, starting with the criterion used to decide whether or not two key-points should be linked.

5.1.1. Definition of a connection criterion

We define a *quality measure* associated with every pair of key-points which are candidates to be linked by a spline curve. This value will be used as a criterion to reject a pair when its associated

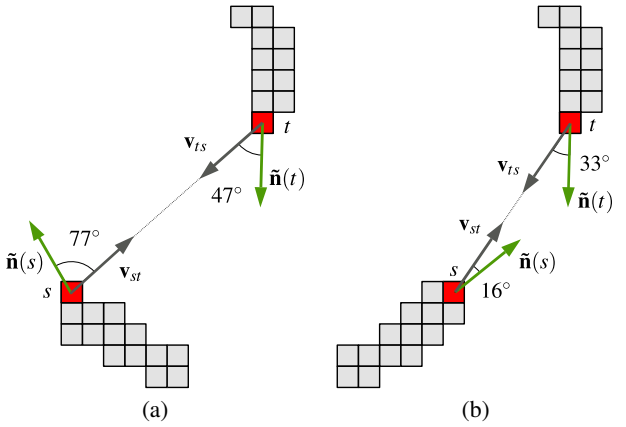


Figure 6: *Illustration of the second term of the quality measure* $\omega(s,t)$*. The configuration (b), where* $\tilde{\mathbf{n}}(s)\mathbf{v}_{st}+\tilde{\mathbf{n}}(t)\mathbf{v}_{ts}=1.8$*, is estimated to be a better candidate for closure using a spline curve than the configuration (a), for which* $\tilde{\mathbf{n}}(s)\mathbf{v}_{st}+\tilde{\mathbf{n}}(t)\mathbf{v}_{ts}=0.9$.

value is equal to zero, and as a mean to sort the remaining ones. Indeed, the proposed closing method makes it possible to allow, or avoid, intersections between the closing strokes. In the case when such intersections are not allowed, it is therefore important to draw first the strokes that are considered of higher relevance. Thus, if s and t are two pixels of $\mathcal{I}$, the connection of these key-points by a spline curve is parameterized by several values, among which one can find:

- $d_{\max}$, the maximum distance above which two key-points should not be linked together;
- α, the maximum admissible angle between the normals $\tilde{\mathbf{n}}(s)$ and $-\tilde{\mathbf{n}}(t)$, which should reasonably be in $[0°,90°]$.

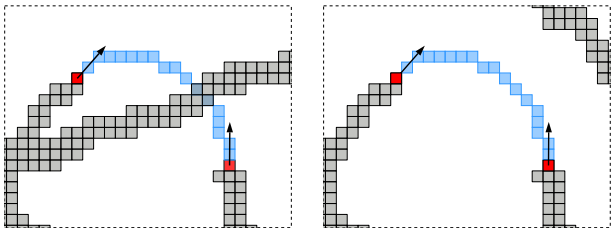
We define the *quality factor* associated with the pair of pixels s and t, denoted by $\omega(s,t)$, as follows:

$$\begin{aligned}\omega(s,t) \quad = \quad & \max(0,1-\tfrac{\|s-t\|}{d_{\max}}) \\ & \cdot\tfrac{1}{2}\max(0,\tilde{\mathbf{n}}(s)\mathbf{v}_{st}+\tilde{\mathbf{n}}(t)\mathbf{v}_{ts}) \\ & \cdot\max(0,\tilde{\mathbf{n}}(s)\cdot(-\tilde{\mathbf{n}}(t))-\cos(\alpha))\end{aligned} \tag{6}$$

where $\mathbf{v}_{st}=\frac{t-s}{\|t-s\|}$ and $\mathbf{v}_{ts}=-\mathbf{v}_{st}=\frac{s-t}{\|s-t\|}$.

Remark 1 It is readily seen that $\omega(s,t)=\omega(t,s)$. Furthermore, we also have $0\leq\omega(s,t)\leq 1$ since each term of the product in equation 6 is non-negative and bounded by 1.

As previously mentionned, any pair of key-points $\{s,t\}$ will not be considered as soon as $\omega(s,t)=0$. The first term of the sum favors pairs of points that are close one to each other, and excludes pairs which satisfy $\|s-t\|\geq d_{\max}$. The second term promotes pairs for which the direction of the estimated normal at each point is close to the direction towards the other point (see Fig. 6). Finally, the third term allows to favor pairs whose estimated normal vectors have opposite directions but are close in orientation, excluding a difference of orientation greater of equal to the angle α.



(a) Invalid closing stroke. (b) Valid closing stroke.

Figure 7: *Configurations of stroke closing using a spline in a part of an image* I_b *(in gray). In (a) the spline curve* $\mathcal{S}$ *(in blue) between the two key-points (in red) is such that* $\tau(\mathcal{S},I_b)=4$*; it is therefore not added. Note however that the bottom-right stroke may be judiciously extended by a line segment (subsection 5.2). In (b) we have* $\tau(\mathcal{S},I_b)=2$*, the curve can therefore be drawn.*

5.1.2. Drawing digitized spline curves

The set of all pairs of pixels $\{s,t\}$ of $\mathcal{I}$ satisfying $\omega(s,t)>0$ is then traversed following the decreasing values of $\omega(s,t)$, and spline curves are drawn between these pairs as soon as two other conditions are satisfied (which are described in the remaining part of this subsection). Each curve is defined by its two extremities $\{s,t\}$ and respective tangent vectors. The distance between the two points is, in practice, a convenient factor that can be applied to the estimated normal vectors $\tilde{\mathbf{n}}(s)$ and $\tilde{\mathbf{n}}(t)$ to obtain the two tangent vectors. Still, it is possible in our proposed implementation to modulate the aspect of the splines by applying a further multiplication factor ρ, with $0\leq\rho\leq 2$.

5.1.3. Preventing an over-segmentation of the background

The proposed stroke closing method results in a segmentation of the image background into 4-connected areas which will be automatically colorized in a following step (Section 6). Generally, an over-segmentation of the background is not satisfactory: it would mean a tough job for the *colorist* who would then have to manually specify colors for many regions with small area. To overcome this problem, we propose to avoid intersections between closing strokes and the original ones (Fig. 7), and also to prevent the creation of regions with small areas.

5.1.4. Handling intersections

To check in a robust way whether or not a closing stroke intersects an input stroke, without taking into account its extremities, we rely on the notion of the number of *transitions* between a path of pixels and a binary image. This is necessary because of the existence of configurations for which several contiguous pixels, at the extremity of a spline, may belong to the input drawing.

Definition 6 Let J be a binary image defined over Ω and $\mathcal{C}=(p_0,\ldots,p_{n-1})$ be the parameterization of an 8-connected path of pixels connecting p_0 to p_{n-1}, $n\in\mathbb{N}^*$. We define the *number of transitions* associated to $\mathcal{C}$ and J, which we denote by $\tau(\mathcal{C},J)$, as follows:

$$\tau(\mathcal{C},J)=\left|\left\{i\in\{0,\ldots,n-2\}\text{ s.t. }(J_{(p_i)}=1)\veebar(J_{(p_{i+1})}=1)\right\}\right|$$

where $\veebar$ stands for "exclusive or".

Intuitively, if $p_0, p_{n-1} \in \Omega$ satisfy $J_{(p_0)} = J_{(p_{n-1})} = 1$, we have $\tau(\mathcal{C}, J) = 2$ if and only if the path $\mathcal{C}$ leaves exactly once the non-zero pixels of J, to enter them back a single time afterwards. Thus, a closing stroke $\mathcal{C}$ between two pixels will not be added to the binary image I_c (during the closing process) if $\tau(\mathcal{C}, I_c) \neq 2$ in the case when we want to avoid intersections between closing strokes, and if $\tau(\mathcal{C}, I_b) \neq 2$ otherwise. We have indeed chosen to introduce a parameter which allows or prevents intersections between closing strokes (called *self-intersections*). When such intersections are allowed, the previously described test is achieved using the input binary image I_b, otherwise it is done on the image I_c containing all the closing strokes which have already been added.

5.1.5. Creation of regions with suitable area

In order to avoid a possible over-segmentation, the proposed closing method also relies on a criterion which prevents the drawing of strokes when they would lead to the creation of at least one 4-connected background component whose number of pixels is lower than a given threshold $a_{\min}$. Nevertheless, a region with less than 5 pixels is not considered as being relevant from this point of view since regions with a very few pixels can appear. It may indeed happen for example in the neighborhood of a key-point from which several closing strokes are to be drawn. In this case, the apparition of such very small and therefore not significant regions should not prevent strokes from being drawn. Note that this value of 5, yet empirical, is actually effective to ignore a very local artifact which does not depend on the resolution of the image.

Eventually, for a candidate closing stroke $\mathcal{T}$ made of n pixels and a binary image J_b, we rely on an algorithm with complexity $O(n)$ allowing to check if any 4-connected background region $R \subset \overline{J_b \cup \mathcal{T}}$ adjacent to $\mathcal{T}$ satisfies $(|R| \geq a_{\min}) \vee (|R| < 5)$.

5.2. Closing with line segments

After the drawing of splines that have been considered valid during the previous step, some straight line segments may be added, starting at some of the key-points. These line segments should extend an existing stroke in the direction given by the estimated normal at the key-point, if another stroke can be found within a limited distance in that same direction. As for the spline curves, the drawing of such segments is subject to a few conditions:

- $s_{\max}$, the maximum length of the segment;
- $c_{\max}$, the maximum number of closing strokes originating from a given key-point;
- $a_{\min}$, the minimum number of pixels in the regions created by drawing the segment.

The straight segments which have been added by this step are depicted in green color in Fig. 5(c). Note on this figure the effect of the condition on the minimum area of the created regions. Indeed, the brown color region counts 50 pixels; which, for a parameter $a_{\min} > 50$ prevents the drawing of a segment starting at the pixel p.

To summarize, the proposed stroke closing method is composed of 3 main steps. First, a key-point detection is applied to the binarized image I_b. Second, key-points satisfying extra conditions are linked with splines to form a new image I_c. Last, additional gaps are filled in I_c by extending existing strokes with small line segments. These steps have been elaborated with the help of our collaborator, David Revoy, a professional digital illustrator. Analyzing the way he would have closed drawings by hand inspired the modelization of the proposed closing technique with splines and segments according to a quality factor.

6. Coloring flat regions

Obtaining the binary image I_c with the closed line-art contours is the most important part of our proposed colorization process. Following that step, we propose three distinct methods that generate a new color layer $I_{\text{col}} : \Omega \to [0,255]^3$ associated to a drawing I. Both correspond to valid workflows for the colorist.

6.1. Random colorization of connected regions

Here, the idea lies in generating a colorization layer I_{col} composed of regions filled with random and piecewise constant colors, so that the set of these regions form a slightly over-partitioning of the initial drawing (Fig. 10(b)). Then, the flat color artist will only have to assign a plausible color to each filled region of I_{col}, the same color being assigned to several neighboring regions if necessary (Fig. 10(c)). In practice, this can be done by applying the usual *bucket fill* tool found in the digital painting software, directly on the constant regions defined on I_{col}. With this mode, producing I_{col} is realized by a simple connected component labeling algorithm applied on the closed binary image I_c, e.g. with the fast algorithm [HMB01] (linear complexity), for all points $I_{f(x,y)} = 0$. Afterwards, those labels are propagated to non-zero pixels by a *watershed* algorithm [BM93] based on the following priority map P defined as:

$$\forall (x,y) \in \Omega, \quad P_{(x,y)} = - \min_{(p,q) \in \Omega} \Big[\|(x,y) - (p,q)\| \;\mid\; I_{f(p,q)} = 1 \Big]$$

This leads to an onion-peel reconstruction of the labels onto the residual contours. Note that the computation of P is also in linear time thanks to the fast algorithm of [MRH00] for the distance transform. Finally, a random color is assigned to each obtained label in I_{col}. The prior closing of the line-art contours is the key step to get many distinct regions that form a judicious partitioning of the input drawing I (e.g. see the forehead-hairs boundary in Fig. 10(e)).

6.2. Color scribbles-guided colorization

This second colorization technique takes up the idea of extrapolating color scribbles placed by the user (Fig. 10(f)), until one gets a full colored partition of the whole drawing (Fig. 10(g)). This idea is actually common to all state-of-the-art techniques [LLW04, QST*05, QWH06, SDC09]. The propagation of the known colored pixels is still done through a watershed algorithm [BM93], with a priority P defined from the closed binary image I_c, such as:

$$\forall (x,y) \in \Omega, \quad P_{(x,y)} = \min\big((I_c * G_{\sigma_1})_{(x,y)}, (I_c * G_{\sigma_2})_{(x,y)}\big)$$

where $I_c * G_\sigma$ denotes the convolution of I_c by a 2d isotropic Gaussian kernel with standard deviation σ. We consider indeed both local ($\sigma_1 = 1$) and global ($\sigma_2 = \max(l,h)/100$) scales for analysis

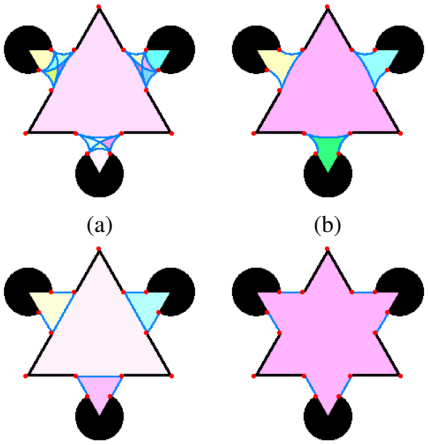


Figure 8: *Influence of the parameters for our region closing algorithm:* (a) Curved splines and low minimal area, (b) Curved splines and medium minimal area, (c) Straight splines and medium minimal area, (d) Straight splines and high minimal area.

in order to estimate a continuous potential map P from the image I_c. Here again, considering the closed binary contours for propagating the user-defined color scribbles plays a leading role in the efficiency of the colorization algorithm (as shown in Fig. 10(g), comparing the colorization results based on I against those based on I_c).

6.3. Automatic cleaning of a given color layer

We propose also a third original approach (which is basically a mix of the two former ones 6.1 and 6.2) that requires the user to provide an additional color image I_{col} (layer) corresponding to a very rough and quickly made colorization (Fig. 10(e)). It is not required that I_{col} has only piecewise constant regions of color. This layer I_{col} can be then *auto-cleaned* by combining its color data with the information got from the estimation of the closed binary line-art I_c: each connected component of I_c is simply colorized with the median color of the corresponding region the component overlaps in I_{col}. This median color is actually computed as the median of the three color components R, G and B taken separatly for each disctinct region of I_c, leading to a cleaned version of the input color layer I_{col} (Fig. 10(f)).

7. Results

Fig. 10 and 11 summarize the use of the two proposed flat colorization methods, and show very clearly that the closing of drawing strokes is indeed an essential step for getting relevant results. Fig. 8 illustrates the influence of the different algorithm parameters, for the colorization of the well-known *Kanizsa* triangle. By varying the maximum lengths of the spline curves and segments, and the maximum areas of the connected regions, we are capable of generating different (though all coherent) coloring results. This practically means we are able to adapt ourselves to many types and resolutions of input line-arts. Moreover, we avoid the computation of expensive minimization processes, and we use only quasi-linear complexity labeling/propagation techniques, so that our method becomes particularly efficient and simple to implement in comparison with state of the art algorithms doing a similar task (see *Benchmarks* below).

Image name	Resolution	Krita	Our	Gain
Mister	418 × 404	0.5 s	0.1 s	80%
Futaba	1119×720	2.1 s	0.42 s	80%
Zones	2210×1287	5.5 s	1.43 s	74%
Characters	5259×2400	28 s	8.5 s	70%

Table 1: *Comparison of computation times.*

Note also that our coloring technique works satisfactorily when the input line-art contains *anti-aliased* edges, although the proposed geometric analysis is done on a binarized image I_b. The reason is twofold: on the one hand, the binarization of I does not fundamentally deteriorate the geometric structure of the drawn lines, and even if some contours may be lost (light lines that could pass below the threshold for instance), they would be likely reconstructed by our closing algorithm. On the other hand, the propagation methods used when generating the color layer I_{col} ensure that colored pixels are effectively reconstructed "under the drawing lines". Merging I and I_{col}, typically by multiplication, makes it then possible to slightly color pixels that do not belong to anti-aliased part of the lines (hence not completely black).

Benchmarks

We have compared execution times of our method against a state of the art technique, *LazyBrush*, proposed in [SDC09]. It should be emphasized that GPU implementations of *LazyBrush* are available, which make it usable in interactive context. However, in order to conduct a fair comparison that reflects actual algorithmic complexities of both methods, we used CPU only implementations in our tests.

On the one hand, we used a *C++* implementation of *LazyBrush* shipped with version 4.0 of the digital painting software *Krita* (a functionality named "Colorize Mask"). It should be noticed that the Boykov-Kolmogorov graph-cut algorithm [BK04], as used by *LazyBrush*, showed to be a complexity bottleneck for a CPU only implementation. For that reason, developers of *Krita* replaced it with a custom watershed algorithm. On the other hand, we used a *C++* implementation of our method. Experiments were achieved on an Intel® Xeon® based computer, with 12 actual cores @2.6GHz and Hyper-Threading Technology. Computation times on four different images (Table 1) show a noticeable gain which, again, confirms the better algorithmic complexity of our method.

Note that we chose to compare our method to the only one we know with comparable workflow and with whom the comparison is fair with respect to memory complexity and, in turn, possible implementation environments. Indeed, some of the other methods mentioned in section 2, which are based on Deep Learning, have memory storage requirements which we see as a bottleneck.

What is more, we have found that our method provides comparable or better results than *LazyBrush*, especially in case of large gaps. As an illustration, Fig. 9 shows output of both methods on the "Girl" sample image.

(a) (b)

Figure 9: *Comparison with LazyBrush result on the Girl sample image with input scribbles of Fig. 10(f). (a) Output of the* LazyBrush *algorithm. (b) Output of our method where gaps tend to be, by design, closed in a more continuous way.*

8. Conclusion and future work

We have presented in this paper several methods intended to help artists in the tedious task of line-art colorization. Each of these methods relies on a common and straightforward step of curve closing, ending up with an algorithm of very reasonable space and time complexities. Thanks to this, implementation on systems with limited ressources (like inexpensive tablets) may be considered, offering a possible handled working environment for artists. It should be noted that our line closing method may (nicely) handle gaps with arbitrary size and not only small one.

For the sake of reproducible research and to ease the colorization task for artists, the proposed algorithm has been included as a user-friendly filter *Smart coloring* into G'MIC [TF18], an image processing framework freely available as a plugin for Gimp and Krita.

On the prospective side, it might be interesting to make use of deep learning methods to achieve the curve closing step, instead of using our heuristic criterions. However, space complexity would certainly suffer from this choice, for results that need to be compared.

References

[BK04] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence 26*, 9 (2004), 1124–1137. 7

[BM93] BEUCHER S., MEYER F.: *The morphological approach to segmentation: The watershed transformation*, vol. 34 of *Mathematical Morphology in Image Processing*. 1993, ch. 12, pp. 433–481. 6

[BS18] BESSMELTSEV M., SOLOMON J.: Vectorization of line drawings via polyvector fields. *CoRR abs/1801.01922* (2018). arXiv:1801.01922. 2

[CHRU85] CHEN L., HERMAN G. T., REYNOLDS R. A., UDUPA J. K.: Surface shading in the cuberille environment. *IEEE Computer Graphics and Applications 5*, 12 (1985), 33–43. 3

[FLB16] FAVREAU J.-D., LAFARGE F., BOUSSEAU A.: Fidelity vs. simplicity: A global approach to line drawing vectorization. *ACM Trans. Graph. 35*, 4 (2016), 120:1–120:10. 2

[FM09] FOUREY S., MALGOUYRES R.: Normals estimation for digital surfaces based on convolutions. *Computers & Graphics 33*, 1 (2009), 2–10. 3

[Fra17] FRANS K.: Outline colorization through tandem adversarial networks. *CoRR abs/1704.08834* (2017). arXiv:1704.08834. 2

[HMB01] HESSELINK W. H., MEIJSTER A., BRON C.: Concurrent determination of connected components. *Science of Computer Programming 41*, 2 (2001), 173 – 194. 6

[ISSI16] IIZUKA S., SIMO-SERRA E., ISHIKAWA H.: Let there be color!: Joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Trans. Graph. 35*, 4 (2016), 110:1–110:11. 2

[LLW04] LEVIN A., LISCHINSKI D., WEISS Y.: Colorization using optimization. *ACM Transactions on Graphics (TOG) 23*, 3 (2004), 689–694. 2, 6

[LWCO*07] LUAN Q., WEN F., COHEN-OR D., LIANG L., XU Y.-Q., SHUM H.-Y.: Natural image colorization. In *Proceedings of the 18th Eurographics conference on Rendering Techniques* (2007), pp. 309–320. 8

[MRH00] MEIJSTER A., ROERDINK J. B. T. M., HESSELINK W. H.: A general algorithm for computing distance transforms in linear time. In *Proceedings of the 5th International Symposium on Mathematical Morphology and its Applications to Image and Signal Processing, ISMM* (2000), pp. 331–340. 2, 6

[NHS*13] NORIS G., HORNUNG A., SUMNER R. W., SIMMONS M., GROSS M.: Topology-driven vectorization of clean line drawings. *ACM Trans. Graph. 32*, 1 (2013), 4:1–4:11. 2

[QST*05] QIU J., SEAH H. S., TIAN F., WU Z., CHEN Q.: Feature-and region-based auto painting for 2d animation. *The Visual Computer 21*, 11 (2005), 928–944. 2, 6

[QWH06] QU Y., WONG T.-T., HENG P.-A.: Manga colorization. *ACM Transactions on Graphics (TOG) 25*, 3 (2006), 1214–1220. 2, 6

[Ros70] ROSENFELD A.: Connectivity in digital pictures. *J. ACM 17*, 1 (1970), 146–160. 3

[Ros74] ROSENFELD A.: Adjacency in digital pictures. *Information and Control 26*, 1 (1974), 24–33. 3

[SBŽ05] SÝKORA D., BURIÁNEK J., ŽÁRA J.: Colorization of black-and-white cartoons. *Image and Vision Computing 23*, 9 (2005), 767–782. 8

[SC97] SOON S. H., CHUA B. C.: A skeletal line joining algorithm. *Insight Through Computer Graphics: Proceedings Of The Computer Graphics International 1994 (Cg194)* (1997), 62. 8

[SDC09] SÝKORA D., DINGLIANA J., COLLINS S.: Lazybrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum 28*, 2 (2009), 599–608. 2, 6, 7

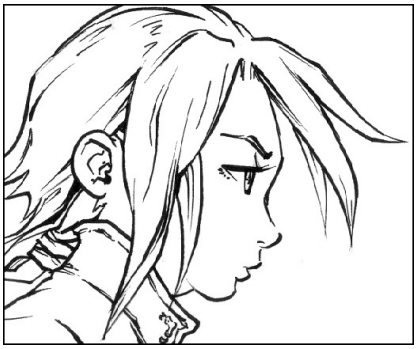
[SF00] SEAH H. S., FENG T.: Computer-assisted coloring by matching line drawings. *The Visual Computer 16*, 5 (2000), 289–304. 8

[SMYA14] SATO K., MATSUI Y., YAMASAKI T., AIZAWA K.: Reference-based manga colorization by graph correspondence using quadratic programming. In *SIGGRAPH Asia 2014 Technical Briefs, China* (2014), pp. 15:1–15:4. 2

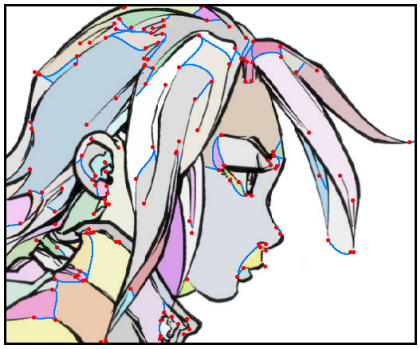
[TF18] TSCHUMPERLÉ D., FOUREY S.: G'MIC: GREYC's Magic for Image Computing: A full-featured open-source framework for image processing. http://gmic.eu/, 2008–2018. 8

[YS06] YATZIV L., SAPIRO G.: Fast image and video colorization using chrominance blending. *IEEE transactions on image processing 15*, 5 (2006), 1120–1129. 8

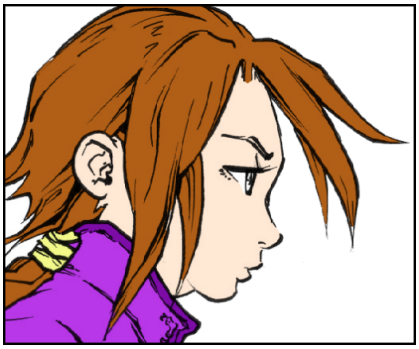
[ZCZ*09] ZHANG S., CHEN T., ZHANG Y., HU S., MARTIN R. R.: Vectorizing cartoon animations. *IEEE Transactions on Visualization and Computer Graphics 15*, 4 (2009), 618–629. 2



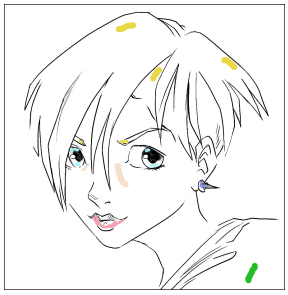
(a) Original line-art



(b) Detected keypoints (red), and curves (blue) added by our closing algorithm

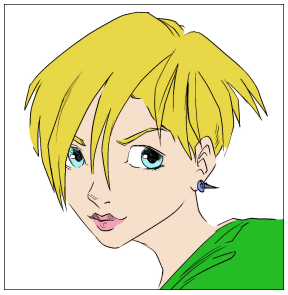


(d) Result after color assignment of each closed region, by the artist

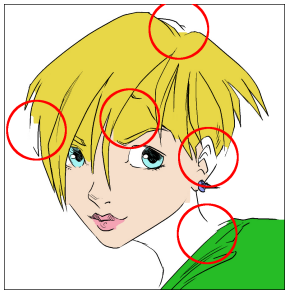


(f) Original line-art + user-defined color scribbles

(g) Color propagation after automatically closing the contours

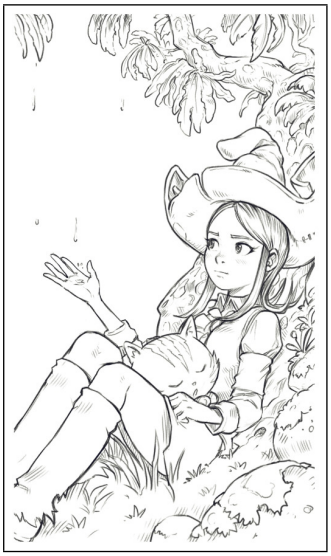


(h) Colorized result by our algorithm



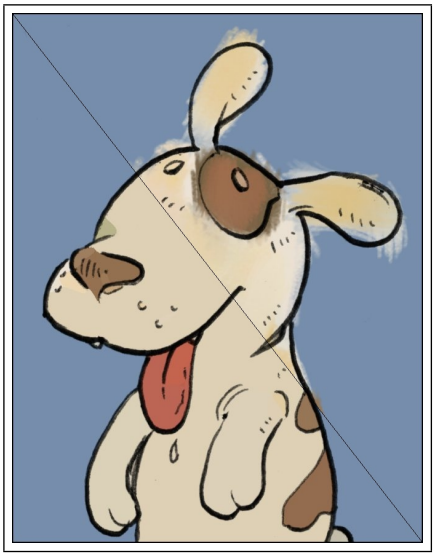
(i) Same, without closing the contours (flaws circled in red)

Figure 10: *Illustration of two of the proposed colorization methods.* **Method 1 (a,b,c,d,e):** Colorization using over-segmented partition with random colors. **Method 2 (f,g,h,i):** Colorization by the propagation of user-defined color scribbles.



(a) Input B&W lineart to be colorized.

(b) Result after our automatic colorization algorithm.



(c) Result of our automatic colorization algorithm, in auto-clean mode.

Figure 11: *Illustration of two different colorization techniques that our algorithm enables:* **(a)-(b)** *From the automatic random colorization* (b,top) *generated by our algorithm, the artist assign a plausible color to each generated region* (b,bottom). **(c)** *From an input line-art and a roughly made colorization layer* (c,top)*, our algorithm is able to "clean" the provided colorization to make it stick to the line-art contours* (c,bottom).