Interactive Flat Coloring of Minimalist Neat Sketches

Amal Dev Parakkat[1], Prudhviraj Madipally[2], Hari Hara Gowtham[2] and Marie-Paule Cani[1]

[1]École Polytechnique, CNRS (LIX), IP Paris, France
[2]Indian Institute of Technology Madras, India

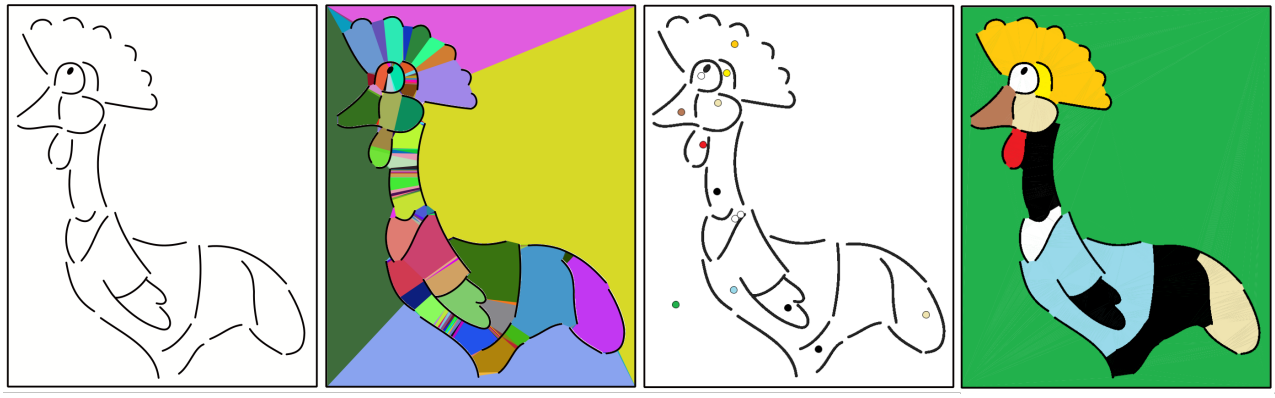

Figure 1: *Left to Right: Input sketch, initial automatic segmentation, user cues, result of iterative coloring*

Abstract

We introduce a simple Delaunay-triangulation based algorithm for the interactive coloring of neat line-art minimalist sketches, ie. vector sketches that may include open contours. The main objective is to minimize user intervention and make interaction as natural as with the flood-fill algorithm while extending coloring to regions with open contours. In particular, we want to save the user from worrying about parameters such as stroke weight and size. Our solution works in two steps, 1) a segmentation step in which the input sketch is automatically divided into regions based on the underlying Delaunay structure and 2) the interactive grouping of neighboring regions based on user input. More precisely, a region adjacency graph is computed from the segmentation result, and is interactively partitioned based on user input to generate the final colored sketch. Results show that our method is as natural as a bucket fill tool and powerful enough to color minimalist sketches.

CCS Concepts

• ***Theory of computation*** → *Computational geometry;* • ***Computing methodologies*** → *Image processing; Shape analysis;* • ***Applied computing*** → *Fine arts;*

1. Introduction

Colorizing line-art images is typically done in two steps [FTR18], namely flat coloring (assigning uniform colors to each region in the image) followed by the addition of extra information such as shading and lighting. While flat-coloring is easy for closed contours, many line sketches do include gaps which makes the task difficult. This is in particular the case for minimalist stylized sketches (see Figure 1, left), a style used in several recent movies such as "Ernest & Celestine" and "The Big Bad Fox and Other Tales". Though a few methods exist to assist users in coloring sketches with non-closed contours, most of them require heavy user interaction. Therefore, there is no viable solution for quickly coloring minimal sketches at a large scale, as for instance for movie production. Our goal is to provide an easy and robust way to perform flat-coloring, applicable to both closed and open contours. Since the "final good coloring" depends on the perspective of the artist, the system should provide the user with artistic freedom and enable to progressively edit and refine the result.

2. Related work

The first method used to make colorization of contour sketches more robust was trapped-ball segmentation [ZCZ*09], which

© 2020 The Author(s)
Eurographics Proceedings © 2020 The Eurographics Association.

DOI: 10.2312/egs.20201024

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

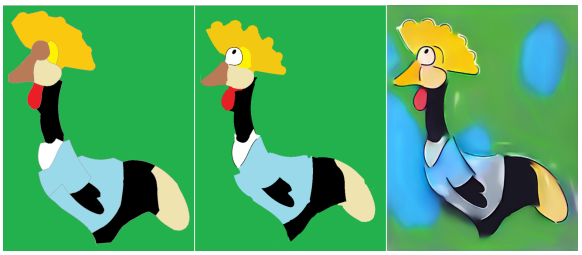

Figure 2: *Left to right: Coloring after running a curve reconstruction algorithm [PMM18]; Using GIMP's line art coloring tool [FTR18]; Results of PaintsChainer‡, a learning based semi-automatic colorization tool [ZLW*18].*

sweeps a ball of predefined radius inside the sketch to prevent paint from getting inside narrow pockets, or outside the outer contour. While this enables robust processing of small cracks in the contours, this method cannot be applied to arbitrary sketches, where the level of details and thus the size of gaps may locally vary.

Another solution is to use the curve reconstruction methods used for completing skip-stroke sketches [PMM18]: Curve reconstruction is applied on points sampled from the sketch, and used to generate a set of closed boundaries which can be filled using a simple flood-fill tool. Unfortunately, such methods only works for a predefined sampling rate which makes them inapplicable in case of arbitrary gap sizes.

A third approach used in the widely spread GIMP software relies on local sketch analysis [FTR18] instead of looking for global connectivity. A set of key points and the associated splines curves are computed, and used to create closed regions that can be filled using bucket fill. As curve-reconstruction-based methods, this method precomputes a set of closed boundaries, and hence iteratively refining the result becomes cumbersome. See Figure 2 (center).

Semi-automatic methods were used to enable interactive editing and refinement [SDC]: Based on user-specified color scribbles, some global energy is minimized while considering the geometry of the sketch. As mentioned in the paper, the main drawback is that the result highly depends on scribble sizes and weights - which the user needs to keep in mind while coloring. Figure 3 shows the effect of these cues while coloring in a sample sketch.

Recently, machine learning techniques were also used for coloring line art [ZLW*18]. These methods, however, do not utilize the geometric information that can be deduced from the sketch. Therefore, even with user interaction, the desired result may not be easily achieved. See Figure 2 (right).

3. Overview

Our goal is to propose a simple and natural method for the interactive flat-coloring of line arts. The main objective is to make the process as simple for the user as bucket fill, while robustly handling sketches with non-closed contours. In particular, the method

‡ https://paintschainer.preferred.tech

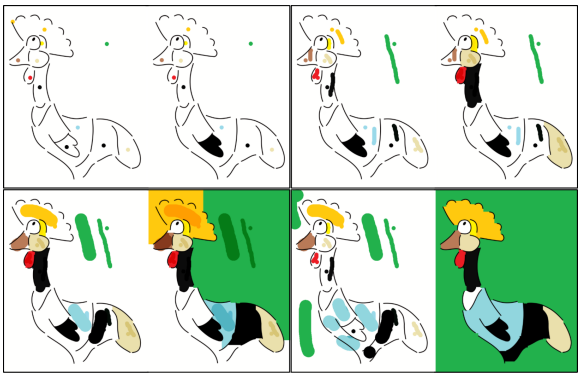

Figure 3: *Effect of scribble size and weight in LazyBrush [SDC] (each set represents input strokes and corresponding coloring)*

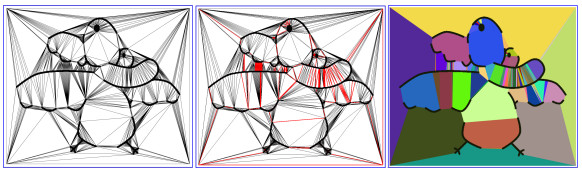

Figure 4: *Left to Right: Delaunay triangulation of a sample sketch, Transition edges (in red color), Result of segmentation*

should work for coloring minimalist sketches with open strokes, such as those used in several recent movies.

In contrast with previous work, our solution relies on computational geometry. It makes use of two steps: the automatic segmentation of the sketch into regions (Section 4), and then the iterative, on-the-fly grouping of these region based on interactive user input (Section 5). In the remainder of the paper, we assume our input sketch is "neat", ie. is a simple vector sketch, with contours stored as polylines and no extra information (no hatching, no shading). If this was not the case, some prepossessing would need to be applied before launching our method.

4. Segmenting a sketch with non-closed contours

The algorithm starts with an automatic segmentation of the input sketch into a set of potential regions. The foreground pixels are first mapped to Euclidean space, and a Delaunay triangulation (DT) is computed. The transition between regions is captured from this Delaunay structure by identifying "transition edges", ie. edges for which the circumcenters of the two associated triangles lie on opposite sides.

Neighboring triangles with no transition edge are then grouped using a triangle growing method similar to the one in [PPM18]: initially, a candidate triangle (an un-visited triangle which is not part of any already identified region) is selected. Starting from this candidate triangle, we recursively merge it to its neighbors until all boundary edges for the region are either transition edges or part of sketch contours.

To enable coloring of the background, points are added at the

© 2020 The Author(s)
Eurographics Proceedings © 2020 The Eurographics Association.

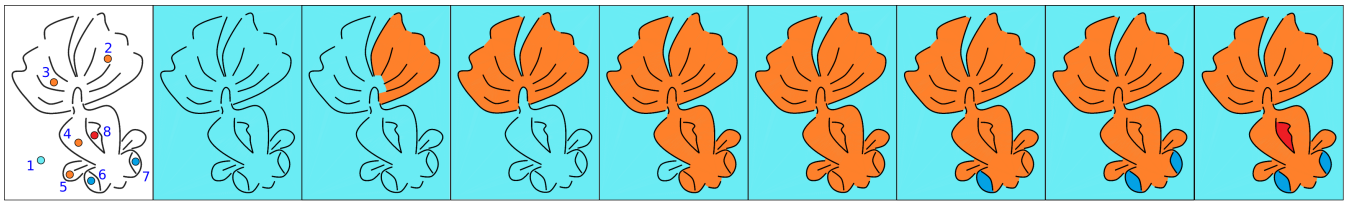

Figure 5: *Left to Right: A sketch with color cues (the order in which cues were given is indicated); Intermediate steps of the iterative coloring.*

four corners of the image before the process starts. Figure 4 shows the Delaunay triangulation of a sample sketch, the corresponding transition edges (in red color), and the resulting regions.

5. Interactive Grouping based on user input

As shown in Figure 4, the segmentation step often over-segments the input sketch. This is not a problem since in the next stage, we interactively and iteratively merge the regions based on the cues given by the user. To achieve this, a region adjacency graph G is initially computed, in which the initial regions output by the segmentation step are vertices, and an edge is created between two vertices if their corresponding region share a common transition edge (Note that we create no edge between regions separated by a contour stroke, in order to avoid merging between regions parted by a contour). A weight w is assigned to each edge of the graph, and set to the Euclidean edge length of the corresponding transition edge. The weight thus models the easiness for paint to flow between the two associated regions.

Once the graph is created, regions are iteratively merged based on the successive color cues given by the user in the form of mouse clicks, as follows:

The interactive merging starts with assigning initial strength to all vertices ($Color\_Strength()$) in the graph to -∞ (which represents the amount of color that has been already stored at this particular vertex). Once a user picks a color and clicks (filling color) at a specific position, the region R and its corresponding graph vertex v (denoted as $vertex(R)$) are identified from the location of the mouse click. All the regions in the graph that are reachable from region R are then filled with the selected color. It has to be noted that this process is initially similar to the *bucket fill* tool (since all vertices inside a closed boundary do have a path between them).

After filling this color, the $Color\_Strength(u)$ of all vertices u which are reachable from vertex v is updated to the value of $Edge\_Flow(u,v)$, ie. the flow that can reach u when colored at v, constrained by the the length of the shortest transition edge along the way - ie. the smallest weight along a graph path, defined as:

$$\begin{aligned} Edge\_Flow(u,v) &= max(f(\mathcal{X}) : \forall\, paths\, \mathcal{X}\, from\, u\, to\, v) \\ f(\mathcal{X}) &= min(Weight(u,v) : \forall (u,v) \in \mathcal{X}) \end{aligned} \quad (1)$$

The user then iteratively picks different colors and clicks on a chosen position in the sketch (as in *bucket filling*). Based on the region R the user selected, the color is recursively spread to the neighboring regions R_i, but only if the $Color\_Strength(vertex(R_i))$ is smaller than the $Edge\_Flow(vertex(R_i), vertex(R))$.

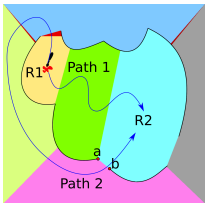

Figure 6: *Priority based path selection*

To ensure that the color spreads through the largest gap first (so that in Figure 6, a color applied on region $R1$ will reach region $R2$ through *Path 1*, instead of *Path 2* which priority is limited by the size of the gap between a and b), we make use of a priority queue, as follows: When a user clicks on a region R, the region is filled with the user-selected color, the $Color\_Strength(vertex(R))$ is updated to ∞ and each neighbor u of $vertex(R)$ is inserted into the priority queue with $Edge\_Flow(u, vertex(R))$ as priority. After that, vertices v are iteratively taken from the priority queue and if $Color\_Strength(v)$ is smaller than $Edge\_Flow(v, vertex(R))$, then the region corresponding to v is colored, the $Color\_Strength(v)$ is updated to $Edge\_Flow(v, vertex(R))$ and all neighbors of v are inserted to the queue. This procedure runs until the priority queue is empty.

Figure 5 shows various steps in the iterative coloring procedure. From left to right, the figures show the order in which the colors are given, and the result after each step. We believe that our algorithm is order-independent: indeed, whatever the order in which colors were applied, we end up in the same final result, thanks to the graph flow mechanism which models how much a transition edge in the sketch is likely to be suppressed.

6. Results & Discussion

Figure 7 shows interactive coloring results using our method. Each set shows the input sketch (with user click positions and corresponding colors in circles) along with the results. It can be observed that our approach took almost the same number of mouse clicks as that might have been required if the sketch was connected, and when a *bucket fill* tool is used. This shows the simplicity and power of the proposed algorithm.

Though our simple approach works quite well, it has few limitations. First of all, we assumed the input is neat; hence, the proposed approach is not applicable to sketches with shading information. A way to solve this issue would be to identify and filter out shading information and restore it back after coloring. Also, if the input is a rough sketch (drawn with multiple strokes - such as the pigface in Figure 7), though our approach helps in coloring it, the small gaps between strokes representing the same contour will be left unattended, unless the user manually colors those regions separately.

The main objective of the initial segmentation we used is to re-

© 2020 The Author(s)
Eurographics Proceedings © 2020 The Eurographics Association.

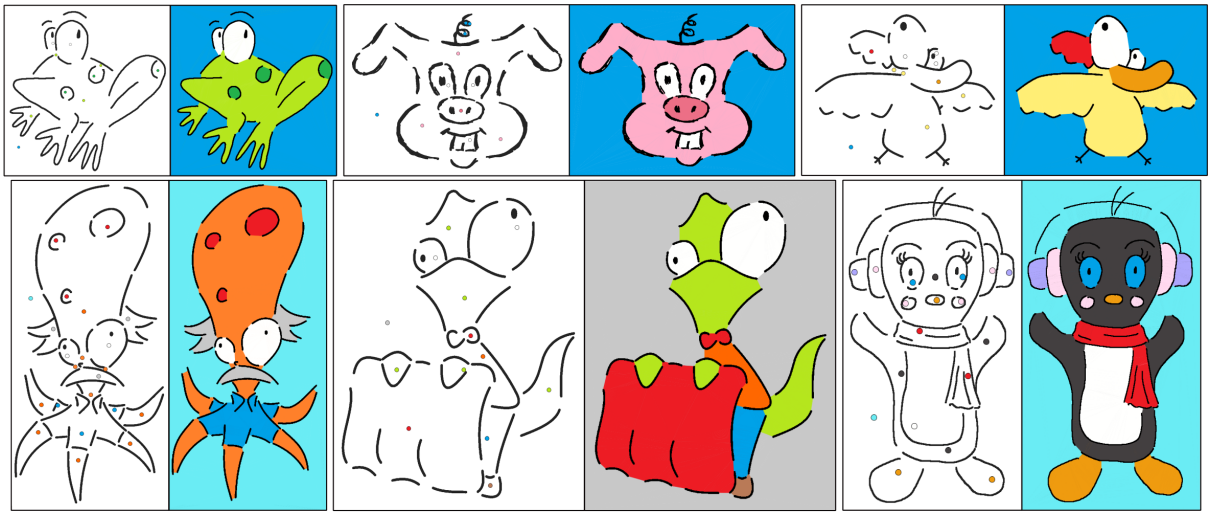

Figure 7: *Sketches and the color cues along with the result of our method*

Figure 8: *Left to Right: A sample sketch, Expected connection, Result of our method, Underlying region (with and without zooming)*

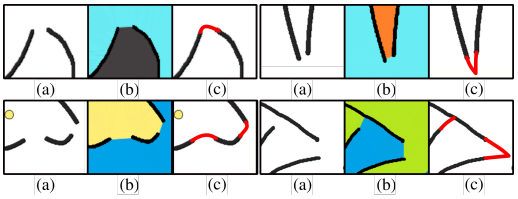

Figure 9: *(a) Parts of different sketches, (b) Results of our method, and (c) Expected boundaries.*

duce the complexity of the algorithm (triangles are merged to facilitate the easy spreading of colors while having a small region adjacency graph). This segmentation sometimes misses some of the features an human would perceive such as the connection between body and tail of the chameleon, as shown in Figure 8. One alternative would be to construct a region adjacency graph from vertices denoting each of the Delaunay triangles, instead to use presegmented, larger regions.

7. Conclusion & Future Work

In this paper, we proposed an easy to use approach for coloring minimalist sketches. The proposed Delaunay-triangulation-based algorithm is easy to implement and results shows that we are able to generate good results compared to the state of the art algorithms, and with less user effort.

In our current solution, gaps in the sketch's contours end up straightly connected by Delaunay edges, which might not be sufficient in some cases. For instance, Figure 9 shows a few perceptually expected connections and the results generated by our approach. An avenue for future research would be to make use of more sophisticated algorithms, such as the one introduced in Entem et al. [EPB*19] in the specific case of smooth shapes, for contour completion. A new alternative would need to be found for restoring contours with sharp corners. Another exciting direction of future work would be to extend the algorithm in order to color sketches with extra information, such as shading and hatching.

References

[EPB*19] ENTEM E., PARAKKAT A. D., BARTHE L., MUTHUGANAPATHY R., CANI M.-P.: Automatic structuring of organic shapes from a single drawing. *Computers & Graphics 81* (2019), 125 – 139. 4

[FTR18] FOUREY S., TSCHUMPERLÉ D., REVOY D.: A fast and efficient semi-guided algorithm for flat coloring line-arts. In *Proceedings of the Conference on Vision, Modeling, and Visualization* (2018), EG VMV '18, pp. 1–9. 1, 2

[PMM18] PARAKKAT A. D., METHIRUMANGALATH S., MUTHUGANAPATHY R.: Peeling the longest: A simple generalized curve reconstruction algorithm. *Computers & Graphics 74* (2018), 191 – 201. 2

[PPM18] PARAKKAT A. D., PUNDARIKAKSHA U. B., MUTHUGANAPATHY R.: A delaunay triangulation based approach for cleaning rough sketches. *Computers & Graphics 74* (2018), 171 – 181. 2

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

[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. 1

[ZLW*18] ZHANG L., LI C., WONG T.-T., JI Y., LIU C.: Two-stage sketch colorization. *ACM Trans. Graph. 37*, 6 (Dec. 2018), 261:1–261:14. 2

© 2020 The Author(s)
Eurographics Proceedings © 2020 The Eurographics Association.