Interactive Flat Coloring of Minimalist Neat Sketches
Amal Dev Parakkat, Prudhviraj Madipally, Hari Hara Gowtham, Marie-Paule Cani

To cite this version:
Amal Dev Parakkat, Prudhviraj Madipally, Hari Hara Gowtham, Marie-Paule Cani. Interactive Flat Coloring of Minimalist Neat Sketches. Eurographics 2020 short paper proceedings, May 2020, Norrköping, Sweden. hal-02495623

HAL Id: hal-02495623
https://hal.archives-ouvertes.fr/hal-02495623
Submitted on 2 Mar 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Interactive Flat Coloring of Minimalist Neat Sketches

Amal Dev Parakkat¹, Prudhviraj Madipally², Hari Hara Gowtham² and Marie-Paule Cani¹

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

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
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”, i.e. 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...
figure 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($v$)) in the graph to $-\infty$ (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:

$$\text{Edge Flow}(u, v) = \max(f(X) : \forall \text{ paths } X \text{ from } u \text{ to } v)$$

$$f(X) = \min(\text{Weight}(u, v) : \forall (u, v) \in X)$$

(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($R$)).

To ensure that the color spreads through the largest gap first (so that in Figure 6, a color applied on region $R_1$ will reach region $R_2$ 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 $\infty$ and each neighbor $u$ of vertex($R$) is inserted into the priority queue with $\text{Edge Flow}(u, \text{vertex}(R))$ as priority. After that, vertices $v$ are iteratively taken from the priority queue and if $\text{Color Strength}(v)$ is smaller than $\text{Edge Flow}(v, \text{vertex}(R))$, then the region corresponding to $v$ is colored, the Color Strength($v$) is updated to $\text{Edge Flow}(v, \text{vertex}(R))$ 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-
reduce 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 pre-segmented, 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


