Delaunay Painting: Perceptual image coloring from raster contours with gaps

Amal Dev Parakkat, Pooran Memari, Marie-Paule Cani

Delaunay Painting: Perceptual image coloring from raster contours with gaps

Amal Dev Parakkat[1,2], Pooran Memari[3] and Marie-Paule Cani[3]

[1]Delft University of Technology, Netherlands
[2]LTCI - Telecom Paris, Institut Polytechnique de Paris, France
[3]LIX, CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, France

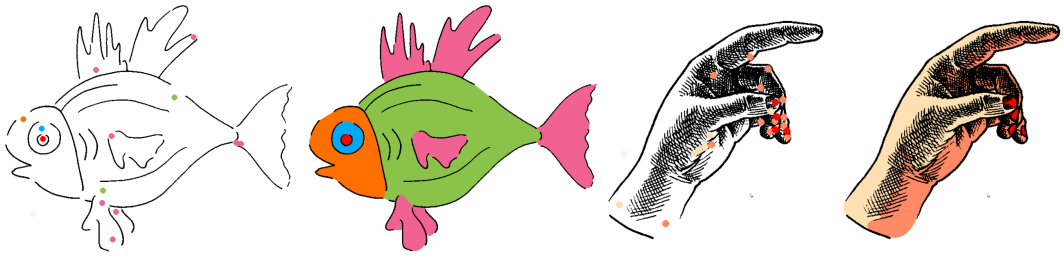


Figure 1: From left to right: A simple line-art and a more complex sketch along with user hints, and the corresponding results.

Abstract

We introduce Delaunay Painting, a novel and easy-to-use method to flat-color contour-sketches with gaps. Starting from a Delaunay triangulation of the input contours, triangles are iteratively filled with the appropriate colors, thanks to the dynamic update of flow values calculated from color hints. Aesthetic finish is then achieved, through energy minimisation of contour-curves and further heuristics enforcing the appropriate sharp corners. To be more efficient, the user can also make use of our color diffusion framework which automatically extends coloring to small, internal regions such as those delimited by hatches. The resulting method robustly handles input contours with strong gaps. As an interactive tool, it minimizes user's efforts and enables any coloring strategy, as the result does not depend on the order of interactions. We also provide an automatized version of the coloring strategy for quick segmentation of contours images, that we illustrate with an application to medical imaging.

CCS Concepts

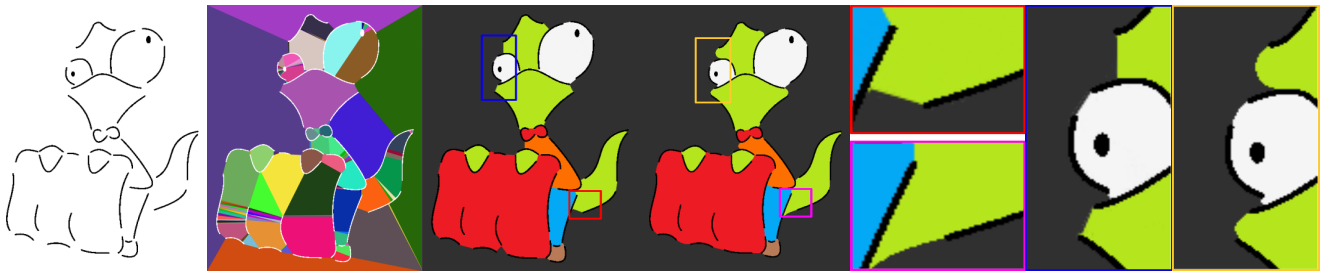
• Computing methodologies → Image manipulation; Shape analysis; • Applied computing → Fine arts; • Theory of computation → Computational geometry;

1. Introduction

Coloring raster contour images, such as line arts, is an essential step in many image manipulation tasks. Flood-fill, applied by the bucket filling tool present in most standard software such as Microsoft Paint or GIMP, is a simple solution for color filling. Unfortunately, it cannot handle contours with gaps, although gaps are a common error in hand-drawn line art. Professional artists may also voluntarily leave gaps (also called a "Professional Gaps") as part of their style while sketching line art. These gaps are small breaks in a line mimicking the light reflecting off an object. Gaps are also considered as an easy way to create transitions while drawing curves or long lines. Movies like "Ernest and Celestine" or "Le Grand Méchant Renard et Autres Contes" use such minimalist sketching style with gaps, enhanced with coloring. Though they make perfect sense from an artist's perspective, coloring line arts with gaps is a difficult, and time-consuming task for digital artists, which is more hectic when it needs to be done for a full animation movie.

We present a specialized algorithm which efficiently handles such coloring process, and is - contrary to flood-fill - robust to the absence of well-defined boundaries in the input. To achieve this, we rely on a novel, yet intuitive Delaunay-based framework. The main idea is to use Delaunay triangulation to create a graph between regions delimited by the contours, and then control a color flow through graph edges, from a few, input color hints.

We introduce two versions of our coloring procedure, depending on how the color hints are obtained: an interactive version, and



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

Figure 2: Comparison with [PMGC20] (a) Input sketch, (b) Segmentation and (c) Coloring result using [PMGC20], (d) Our coloring result. The main differences can be seen in the insets ((e) top: [PMGC20], bottom: Ours, (f) [PMGC20] and (g) Ours)

an automatic version. While the interactive coloring procedure is useful for artistic coloring of contour drawings (where the color hints fully depend on the artist's creativity, and cannot be predicted automatically), the automatic coloring approach can be used for automatically segmenting contour sketches into regions. We illustrate it by an application to medical image segmentation. While such automatic coloring is straightforward, making the interactive coloring approach usable for coloring line-arts brings additional challenges such as the response time, intuitiveness, and aesthetic beauty of the final results. Hence, the following objectives are taken into consideration while designing the interactive method:

- The tool should be as simple to use as Flood-fill: Everyone is familiar with the bucket-fill tool. It is easy to use and straightforward to understand. Such intuitive interaction is required.
- Visually pleasing results: Unless the gaps are correctly filled, the final colored results will not look complete. So the border of colored regions delimited by disconnected contours should reconnect gaps in an aesthetically pleasing way.
- Minimum user-interaction: The user should be able to color visually salient regions in the contour with minimum interaction. Hence, an appropriate color flow mechanism should be provided to spread colors.

To achieve the first point, our method uses an intuitive flow-based color propagation process on the underlying Delaunay triangulation [PCS21], making it as simple to use as Flood-fill. Secondly, instead of reconnecting gaps in the contours with straight lines, which would affect the aesthetic beauty of the final result, we make use of local contour information around gaps for generating the border of colored regions. More precisely, we make use of the concept of Scale Invariant Minimum Variation Curves (SIMVC) and a sharp corner detection heuristic to delimit the colored region in a visually pleasing way (see Figure 1, left). Lastly, we introduce a simple color diffusion mechanism to achieve the objective of minimal user input, even for drawings with additional information such as texture or hatches. In existing methods contours sharply bounds color spreading, requiring the user to paint each region separately, whatever their size. This becomes a cumbersome task when there are many small regions, which commonly occurs with complex raster sketches (see Figure 1, right). In contrast, we use a simple diffusion method based on the perceptual distance between regions (applicable even without any gap between them), enabling colors to seamlessly spread from user colored areas to the uncolored parts of the image. This color spreading procedure is reversible and also can be easily updated.

Overall, the proposed interactive Delaunay coloring method is straightforward and saves the user from many hurdles when coloring raster contour images with gaps: As simple as flood-fill, it only requires a minimal number of color hints, the color filling is fast and easily predictable, and the user can easily correct any mistake, since our method is order-independent and reversible.

In summary, while our method reuses the flow mechanism from [PCS21] in the new context of coloring raster contour images with minimal user inputs, it also makes new technical contributions:

- A color diffusion mechanism to minimize user efforts by propagating colors to uncolored small regions (for examples hatches)
- An aesthetic curve completion technique to fill gaps in the input contours in a visually pleasing way
- An automatic coloring algorithm with possible applications in segmentation of medical images and sketches

2. Related Work

Related work can be classified into two categories, namely automatic versus user-guided colorization methods.

Automatic colorization methods make use of prior knowledge to color an input image without any user intervention [LWCO*07, ISSI16, YBC*19, SCH20]. Most recent works use machine learning techniques for colorizing grayscale photographs, where the color to be used is predicted through an analysis of the input [ISSI16, YBC*19, SCH20]. This kind of work is not applicable to our line art coloring problem, due to the absence of extra information such as scales of gray in our input. Moreover, results would then be limited to quite standard, learned coloring palettes, with no user control. Another set of similar works [HA17, FHOO17] make use of a single, reference colored image to identify the appropriate colors to be used, which again restricts the set of colors as well as the colouring style.

User-guided colorization methods were introduced to give more freedom to digital artists. The latter are asked to provide additional hints to guide colorization, such as color brush-strokes, or even text input such as specifying the region and the color that

should be used to fill it [ZMG*19, KJPY19, CSG*17]. Indeed, this last category of methods can only be used for coloring meaningfully, well identified shapes enhanced with semantic information, and hence cannot be used for generic inputs. It is worth noting that a few learning-based methods, such as [SZC*20] enabled to color line arts in videos as well. In this paper, we focus on coloring methods based on user-defined colored brush strokes, which were extensively developed.

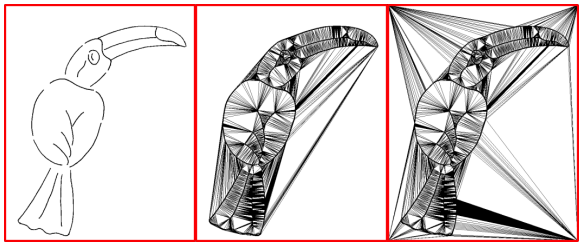
One of the prominent works in brush-based assisted coloring is Lazybrush [SDC09], where the authors express the problem of colouring based on user hints as an energy minimization problem. Various machine learning-based solutions were also introduced [LWCO*07, ZLW*18, HJRD19, Pai20]. As will be shown in Section 8.1, these methods do not necessarily generate a flat coloring result. Moreover, the context of [LWCO*07], in which users apply color-strokes to a grayscale photograph, is quite different from ours. Closer to our sketch-coloring goals, [ZLW*18], [HJRD19], [Pai20], and [ZLSS*21] address the lack of user control of pure, learning-based approaches by enabling the user to add multiple, additional color hints to influence the result. However, when not applied to a specific category of drawings on which the model could be trained (e.g., manga characters), such solutions typically require extensive user intervention, since the same region may need to be recolored multiple times until the desired output is achieved. When the goal is not to address a specific class of drawings, methods that do not use machine-learning may thus be more suitable.

Coloring contour sketches may also be considered as a variant of a **contour closing problem**, a problem tackled since 1994 [GTF94], and which was recently investigated using learning-based approaches [SISSI17, LDL*19]. In [PM16], the authors pose the classical curve reconstruction problem as a way to connect gaps in a contour sketch, enabling the newly created regions to be colored using the standard flood-fill tool. Though more generic curve reconstruction methods were later proposed (eg. [PMM18]), the effectiveness of these algorithms typically depend on sampling density and stroke positions. Moreover, using standard flood-fill after reconnecting contours reduces user's freedom, since only uniform colors can then be applied within the newly-closed regions. This prevents the user from giving multiple colors hints within a given region, as we did, for instance, on the wrist of the hand in Figure 1. Instead of creating connections as in curve reconstruction to fill gaps, Fourey et al. [FTR18] connected appropriate strokes to create a set of segments in the contours. The regions are then interactively updated and colored with the help of the user. The method, which was later embedded in Gimp as a line-art coloring tool, is easier to use than the previous ones. However, user freedom is still limited since it can color only a set of segments, and the boundaries of the resulting sketches are not necessarily well formed, as will be shown through comparison to our solution in Section 8.1.

In this work, our goal is to propose a simple and flexible coloring technique that can be used as a user-friendly toolbox in various sketch coloring scenarios. We therefore do not rely on any optimization or machine-learning-based approach, which avoids the need for extensive user input. Our solution belongs to geometry-based methods, but specifically addresses the issues of user freedom and quality of the resulting color sketches, by introducing an intuitive and light-weight framework for coloring of raster contour sketches with gaps.

Related Delaunay-based techniques: Note that a former image coloring from the same authors was published as a short paper [PMGC20], but the later used a time-consuming sketch segmentation algorithm which also brought restrictions on coloring freedom, since only pre-segmented regions could be colored (eg. regions shown in Figure 2(b)). Moreover, this early solution did not generate visually pleasing boundaries for colored regions, and did not include any color diffusion mechanism between neighboring regions. In contrast, user-freedom, visual quality, and efficiency are all improved by our new interactive method, which comes as well with an automatic companion method. The latter can be used as an auxiliary or corrective segmentation tool for medical images or freehand sketches, as shown by our preliminary results. In addition, we reused the first step shared by our two methods, the so-called Delaunay grouping technique detailed in Sec. 4, in a recent paper aimed at the structuring and vectorization of rough stroke sketches (a quite different purpose compared to coloring, see also [FLB16]), with a focus on user experience [PCS21]. Lastly, let us mention that the coloring flow we introduce is fundamentally different from the flow defined in [DGG03], which is based on Delaunay balls radii and designed for shape segmentation.

3. Overall framework



(a) (b) (c)

Figure 3: (a) Input sketch, (b) Delaunay triangulation of extracted sketch pixel positions, (c) Delaunay triangulation after padding points.

We aim to propose a flat coloring method for contour images, where contours may have gaps. We will first present an interactive method and then an automatic companion method that we illustrate with a few basic examples in the context of medical image and freehand sketch segmentation.

For the main interactive method, our goal is to make the interaction process as easy for the user as using the flood-fill tool, while robustly handling non-closed contours. In particular, the method should work for coloring minimalist sketches with open strokes, such as those used in several recent animation movies. In contrast with previous work, our solution relies on computational geometry.

The entire process starts with the user providing a raster, contour drawing with a set of unbounded regions that has to be bounded by coloring or segmenting it (throughout the remaining of this paper, we will be calling these regions *Bounded regions*). Though

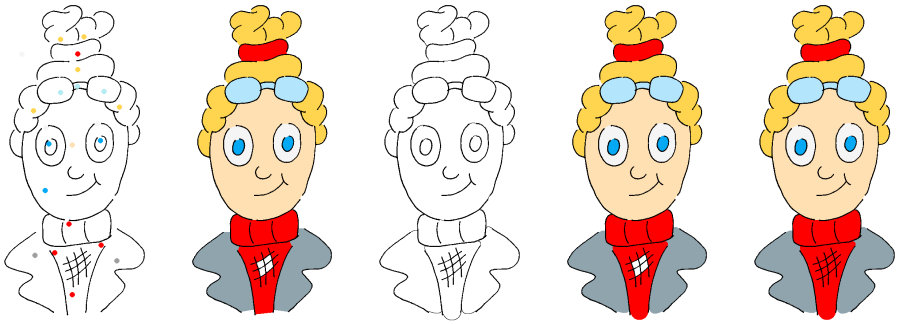


Figure 4: Overview of our method. (a) Input contours (sketch) with color hints, (b) Result of Delaunay grouping, (c) Aesthetically completed curves, (d) Result after aesthetic curve completion, (e) Result after color diffusion

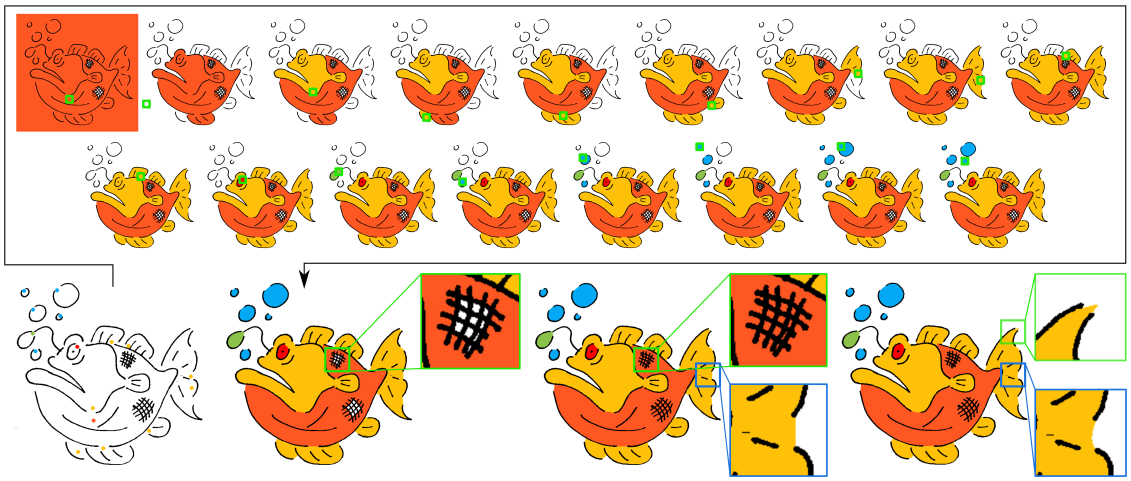


Figure 5: A complete pipeline of our coloring framework on a complex sketch. From left to right: Input sketch with color hints, Result of interactive Delaunay grouping (with steps shown inside the box), Result after color diffusion, Final result after adding aesthetic contours.

throughout this paper we used raster images (mostly hand-drawn) as inputs, one can also use vector images along with a curve sampler. Initially, the pixels that are part of the contours are extracted, and their positions are used to compute a Delaunay triangulation. Figure 3 shows a sample contour along with the computed Delaunay triangulation. As the name suggests, our method paints these Delaunay triangles as per the requirement. To facilitate coloring the background, there should be some triangles present in the background as well. To do that, we pad four points on the input image corners, which ensures that the entire image is triangulated. Figure 3(c) shows the updated Delaunay triangulation after inserting the padding points.

The initial Delaunay triangulation is further used for Delaunay grouping [PCS21] (Section 4). In Delaunay grouping, using a set of hints, the bounded regions are colored iteratively. The input to this step is a set of contours along with color hints (which are either interactively defined by the user or computed automatically) representing the bounded regions (as in Figure 4(a)), and the result is a set of flat colored bounded regions (as in Figure 4(b)).

After Delaunay grouping, the bounded regions' boundaries will be represented by a set of straight lines (Delaunay edges) - which are not convenient for some applications, such as coloring line-arts. The user can thus make use of an Aesthetic curve completion step (Section 5), which replace straight Delaunay edges (were they connecting the gaps between the contours or adjacent regions not separated by a contour) with visually pleasing curves, using the notion of Scale-Invariant Minimum Variation Curves (SIMVC) and sharp-corner detection heuristics. Figure 4(d) shows the effect of aesthetic curve completion. The input is a Delaunay grouping result, and the output is a set of bounded regions with visually pleasing finishing applied on the boundaries (the aesthetically completed curves can be seen in 4(c)).

To make interactive coloring easier, we also provide the user with an optional, color diffusion framework (Section 6). From only a few color hints, the latter helps to spread the colors into uncolored small regions delimited by hatches or textures, avoiding the burden of individually coloring them. The input is the result after Delaunay grouping (as in Fig. 4(d)), and the output is a full coloring, where

colors have been spread to all the uncolored regions (Fig. 4(e)). The complete pipeline of our coloring framework is shown in Figure 5.

Since the position of color hints directly impacts the grouping of Delaunay triangles (Section 4), such hints have to be automatically placed at relevant locations to achieve automatic segmentation of an input image. In Section 7, we discuss a strategy to generate such color hints and automatically generate a meaningful segmentation.

Given a set of color hints and the straightforward pre-process that we have explained, let us now describe two key steps of our method, namely Delaunay Grouping and Aesthetic Curve Completion.

4. Delaunay Grouping

The Delaunay grouping process described in this section is based on an intuitive flow mechanism that we have already introduced in the short version of this paper [PMGC20] and further used in [PCS21], as a part of a sketch simplification algorithm.

A region adjacency graph G is initially computed, in which each Delaunay triangle (seen as a region in the input) is denoted as a vertex, and an edge is created between two vertices if they share a Delaunay edge which is not part of the input contour (i.e. no edge is created between regions separated by a contour stroke). As we will see, this construction will prevent the merging of regions separated by a closed contour. A **weight** w is assigned to each edge of the graph G, and is set to its Euclidean length. In addition, a ($Color\_Strength()$) parameter is associated to each graph vertex to represent the amount of color already assigned to the corresponding region. This parameter is initially set to zero.

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

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. Note 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 the flow that can reach u when colored at v, denoted by $Edge\_Flow(u,v)$, constrained by the the length of the shortest transition edge along the way - ie. the smallest weight along a graph path, defined as:

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

The user then iteratively picks different colors and clicks on a chosen position in the contour (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 or equal to the $Edge\_Flow(vertex(R_i), vertex(R))$.

We use a priority queue to ensure that the color spreads through the largest gap first. For example, as shown in Figure 6, once a color is applied on a point p, it can spread to another point q through various paths (path1, path2, and path3). Flow-through each of these paths is restricted by the shortest edges in their path, in this case, $|\,ab\,|$, $|\,cd\,|$, and $|\,ef\,|$ for path1, path2, and path3 respectively. Usage of priority queue ensures that the color spreads from p to q through path2 (since that is the path having the maximum flow value). The priority queue is maintained as follows:

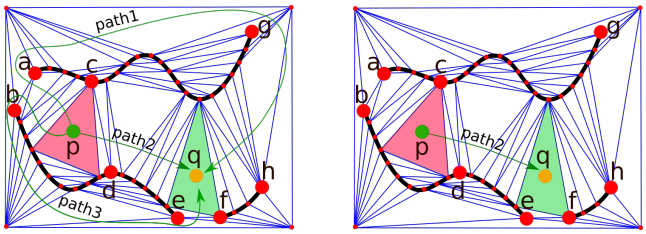


Figure 6: Priority based path selection

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 accordingly. In particular, 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. The overall procedure is summarized in Algorithm 1.

Figure 7 shows various steps in the iterative coloring procedure. In the first row, from left to right, the results of iterative Delaunay coloring are shown after giving each color hint. The color hints, along with the color maps showing the influence of each mouse click on the underlying Delaunay triangles, are shown in the second row. The impact at a triangle T_i after clicking at T_j is computed as $\frac{Color\_Strength(Region(T_i))}{||Longest\_Edge(T_j)||}$ (where $Longest\_Edge(T_j)$ is the longest edge in the triangle T_j). The largest to least influence is colored from a spectrum varying from red to yellow, and if there is no influence, then it is colored in white.

Since each triangle is given independent control, 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 contour is likely to be suppressed.

5. Aesthetic curve completion

Simply applying Delaunay grouping to the Delaunay triangulation of the input separates the resulting colored regions by straight edges. Using straight lines to connect gaps between curved contours is however not perceptually appropriate. Our aesthetic curve completion technique identifies the Delaunay edges connecting contour gaps and if needed, replaces them with appropriate curves, i.e. either sharp corners or smooth cubic curves (see Figures 8 and 9).

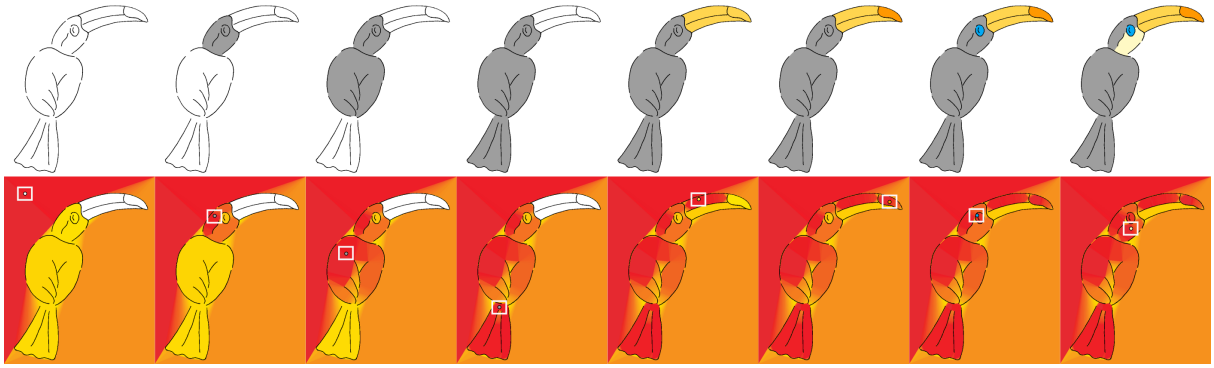


Figure 7: First row: Intermediate results of our Delaunay painting, Second row: Cumulative influence of mouse clicks (shown inside white squares.)

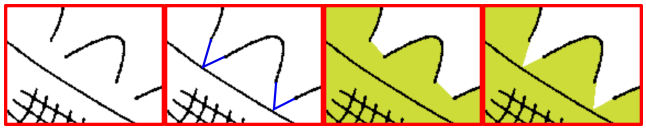
Algorithm 1 Filling algorithm

```
1:  procedure FillColor(G,R,C)
2:      Fill(R,C)
3:      Initialize currentflow of each vertex to zero
4:      for each adjacent vertex v of the (w ← Vertex(R)) do
5:          InsertPQ(v,Weight(v,w))
6:      while PQIsNotEmpty() do
7:          B ← RemovePQ()
8:          v ← B.element(), len ← B.priority()
9:          if currentflow(v)<=len then
10:             currentflow(v) ← len
11:             Fill(Region(v),C)
12:             for each adjacent vertex u of v do
13:                 if currentflow(u)<=Weight(u,v) then
14:                     if len>Weight(u,v) then
15:                         InsertPQ(u,Weight(u,v))
16:                     else InsertPQ(u,len)
```

Our algorithm starts by evaluating the tangents at the endpoints of each contour gap to be filled. To achieve a robust computation of tangents on raster data, use a one-pixel width skeleton of the input sketch, as follows:

- Once the Delaunay grouping is over, the interior and exterior of the required contour boundary get different colors. The Delaunay edges connecting endpoints of the gap therefore share two triangles of different colors. We call such edges "transition edges".
- For each endpoint of a transition edge, the nearest point p_1 in the one-pixel skeleton of the input contour sketch is identified.
- A flood-fill procedure is initiated from p_1 within the contour, and stopped when we reach a pixel p_2 lying at a given distance (set to 5 pixels in our experiments).
- The contour's tangent at p_1 is set to the vector joining p_2 to p_1.

Once tangents are identified, we conduct three tests (discussed next) to check whether the gap should be completed using a sharp corner, or a smooth curve. If any of the sharp corner test fails, then a smooth cubic curve is used to fill the gap.



(a) (b) (c) (d)

Figure 8: Sharp corners: (a) Input sketch, (b) Expected result, (c) Result of Delaunay coloring, (d) Result after gap filling procedure.

5.1. Sharp corners

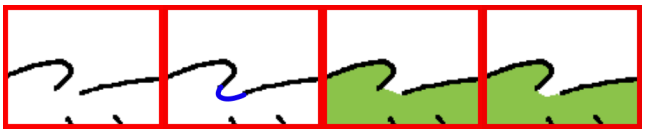
In this section, we propose a method to identify and handle the boundaries that need to be connected using sharp corners (see Figure 8). To do so, let e_{AB} be the edge to be replaced and T_A and T_B be the corresponding tangents, respectively. We check the following three constraints for a gap to be qualified for getting replaced by a sharp corner:

- Angle constraint, based on the angle between the tangent: The constraint is said to be qualified if the angle between the tangents is less then $\pi/3$;
- Perpendicular constraint, based on the distance from the edge to be replaced, to the point where the tangent directions intersect: The qualifier is satisfied if the perpendicular distance from the intersection point p_{ip} to the edge e_{AB} is less than $2* \mid\mid AB \mid\mid$ (which prevents adding long an thin spikes to the contours);
- Linearity constraint, based on how linear the contours are near the gap. The criterion is said to be qualified if the pixels lying close to the endpoints are linearly arranged (we check whether the maximum distance from the pixels lying near the contour endings to the line defined by the tangent is less than a given threshold).

If these three constraints are qualified (a criteria which can also be tuned), we identify it as a sharp corner and replace e_{AB} by $e_{Ap_{ip}}$ and $e_{p_{ip}B}$.

5.2. Minimum Variation Curves

If any of the constraints above is not satisfied, we use a smooth cubic curve to connect the gap (see Figure 9). To achieve a per-



(a) (b) (c) (d)

Figure 9: (a) Input sketch, (b) Expected result, (c) Result of Delaunay coloring, (d) Result after our gap filling procedure.

ceptually pleasing result, we use Scale-Invariant Minimum Variation Curves (SIMVC), which tend to minimize changes of curvature [Mor92]. As in [EPB*19], we define the SIMVC curve based on the tangents at the endpoints of the contours to be connected.

Let us T_A and T_B be the two tangents at the end-point of a (initially straight) transition edge. To achieve our objective of generating a perceptually pleasing curve between them, we connect A and B with a cubic Bezier curve defined by four control points (A, P_1, P_2, B), where $P_1 = A + c_1 T_A$ and $P_2 = B + c_2 T_B$ (P_1 and P_2 are the control points in the tangent directions from A and B respectively). We optimize the free parameters c_1 and c_2 for minimizing the SIMVC energy. The latter was initially introduced [Mor92] as:

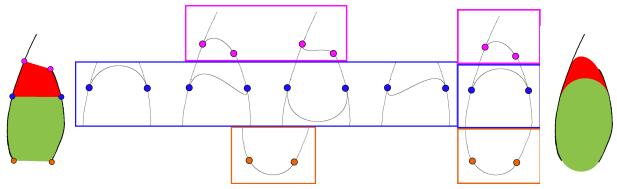
$$E_{SIMVC-Moreton} = (\int ds)^3 \int (\frac{d\kappa(s)}{ds})^2 ds \tag{2}$$

and later modified in [EPB*19] to create slightly shorter curves connecting the gaps as:

$$E_{SIMVC-Entem} = \frac{(\int ds)^5}{\|B-A\|^2} \int (\frac{d\kappa(s)}{ds})^2 ds \tag{3}$$

Where $C(s)$ is the parametric 2D curve with curvature $\kappa(s)$.

We made use of 7-15 Gauss-Kronrod quadrature rule [KMN89] for numerically solving the integration and used Gradient-descent for minimizing the energy function.



(a) (b) (c) (d)

Figure 10: (a) User input with endpoints circled over, (b) Possible configurations of curves connecting the gaps, (c) Selected curves, (d) Final result.

As shown in most of our examples (e.g. the wrist region of the hand at the right of Figure 1), the gaps that have to be filled along transition edges are not always between two ends of contour curves. Indeed, one (or both) of the contours may continue further, which gives two possible tangent orientations at the endpoint(s) of the transition edge (see Figure 10). For selecting this appropriate tangent, we initially identify all possible tangents at each endpoint for each transition edge. Then for each pair of tangents across the edge, we compute the MVC connecting them. Among all these pairs, the pair having minimum SIMVC energy is selected. Figure 10 shows a colored sketch with endpoints marked as circles, all possible configurations of SIMVC, the selected curves with least SIMVC, and the final result, respectively.

In practice, to achieve interactive results, we restrict the search of the free parameters by a scale of 5 of the tangent length. We also skipped the SIMVC computation if the gap is too small (we quantify too small as gaps smaller than 8 pixels) since the difference will not be visible. Figure 28 shows a few line-arts colored using our tool.

6. Color diffusion

The input contours sometimes have many small regions (for example, hatches). Although we could use our tool to fill each of these regions one by one, this would be tedious and time consuming. We thus introduce a color diffusion mechanism, enabling propagate color from colored areas to the uncolored areas.

As shown in Figure 1, while giving additional information like hatching or stippling, the artists usually make use of thin brush strokes (compared to the stroke size they used to draw the main sketch). The main intention behind this is to make the viewer clearly distinguish the sketched contours from the hatch strokes. Based on this observation, we introduce a simple, iterative color diffusion framework in which, the color from the best possible colored region is iteratively diffused to an adjacent, uncolored region. To quantify this "best possible colored region", we define a diffusion strength parameter between pairs of adjacent regions R_i and R_j, denoting how much strength a color needs, to diffuse between them. It is defined as the shortest possible distance between any two pixels (p_i, p_j) such that $p_i \in R_i$ and $p_j \in R_j$.

The color is then recursively spread from colored region to the uncolored regions in the order of increasing diffusion strength. Firstly, the regions are classified into two groups based on whether the user colored them or not. The diffusion strength from the uncolored regions to the colored regions are computed. The tuple having the least strength, is used to transfer the color. The regions (colored and uncolored) are updated, and the procedure is repeated until all regions are colored.

Though the procedure is simple and straightforward, identifying the regions with minimum diffusion strength in a brute force way is computationally expensive. To make it fast, we again make use of the underlying Delaunay structure. The complete color diffusion process, summarized in Algorithm 2, proceeds as follows:

We start with a random Flood-fill algorithm to decompose the contour into different regions. Since the pixels contributing to the minimal diffusion strength can only (trivially) lies on the boundaries, we extract the boundary of each decomposed region. Then we compute the Delaunay triangulation of the extracted boundary pixels. Based on the user coloring, the regions are classified into *ColRegions* (regions which user colored) and *UncolRegions* (where the color could not reach yet). Since the Delaunay edges connecting pixels inside a region are of no relevance and only results in

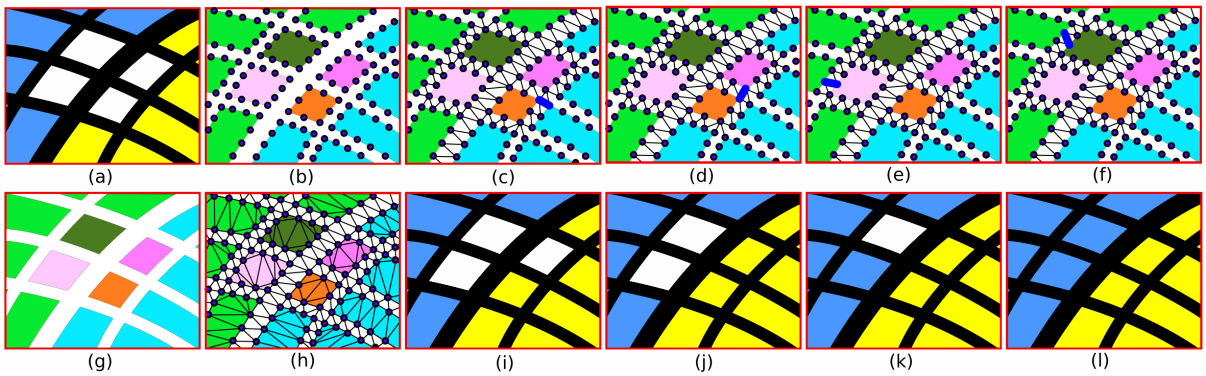


Figure 11: Representative workflow of color diffusion in a hashed region: Top row: (a) A result after Delaunay grouping with four uncolored regions (shown in white color), (b) Boundary samples (only a few of them are shown for a clearer illustration), (c, d, e,f) Filtered Delaunay edges with smallest valid edge shown in blue color. Bottom row: (g) Result of RandomFloodFill() on (a); (h) Delaunay triangulation of (b), (i,j,k,l) Respective updated colored regions from (c,d,e,f).

Algorithm 2 Color diffusion

```
1:  procedure DIFFUSECOLOR(IMAGE I)
2:      RFI = RandomFloodFill(I)
3:      BoundaryPixels = ExtractBoundaries(RFI)
4:      Compute Delaunay Triangulation DT(BoundaryPixels)
5:      (ColRegions, UncolRegions) = BipartateImage(RFI, I)
6:      DelaunayFilter(DT, RFI)
7:      while UncolRegions ≠ ϕ do
8:          e_ij = SmallestValid(Filtered DT(RFI))
9:          if v_i ∈ ColRegions and v_j ∈ UncolRegions then
10:             FloodFill(v_j), ColorAt(v_i))
11:             ColRegions = ColRegions ⋃ Region(v_j)
12:             UncolRegions = UncolRegions - Region(v_j)
13:         else
14:             if v_j ∈ ColRegions and v_i ∈ UncolRegions then
15:                 FloodFill(v_i), ColorAt(v_j))
16:                 ColRegions = ColRegions ⋃ Region(v_i)
17:                 UncolRegions = UncolRegions - Region(v_i)
```

more computation time, we apply a Delaunay filtering to filter out all such Delaunay edges (by removing Delaunay edges connecting pixels from the same region). Among all unfiltered Delaunay edges, the valid Delaunay edge with the smallest edge length is identified (where a Delaunay edge becomes valid only if it is between one colored and one uncolored region). Once a valid edge is identified, the color is diffused from the colored region to the uncolored region. Since the uncolored region under consideration is now colored, we update the regions appropriately. The procedure is continued until all the uncolored regions get a color through diffusion. Figure 11 shows a toy example demonstrating our color diffusion procedure. It has to be noted that the method is not limited to hatched regions, as illustrated on Figure 11.

Since the diffusion process ultimately depends on the stroke size between the regions, sometimes, the result might not match the user expectation. Fortunately, thanks to our interactive color filling and independent color diffusion, the user can provide extra hints to recolor the regions. Figure 12 shows such an example, after giving a set of color hints (Figure 12(a)), using the Delaunay grouping, the sketch is colored. Since a few wool chunks (represented using circular strokes) are closed, Delaunay grouping will leave those regions uncolored (Figure 12(b)). To color these uncolored regions, we applied color diffusion. Unfortunately, as shown in Figure 12(c), a couple of chunks on the head mistakenly got the colors diffused from the background. The user then provides extra color hints (as in Figure 12(d)) to get a perfectly colored region. Figure 12(e) shows the final result of our framework.

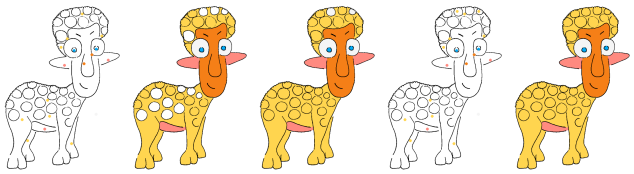


Figure 12: (a) Input sketch with color hints, (b) Result of Delaunay grouping, (c) Result after color diffusion, (d) Adding extra color hints to the existing hints, (e) Our final result.

A few results of our painting algorithm can be seen in Figure 29 and 30.

7. Automatic Coloring Algorithm

Motivated by applications such as automatic coloring and art line completion, we now introduce an automatic variant of the proposed coloring algorithm. Although the automatic prediction of colors that have to be filled inside a contour is nearly impossible since it heavily depends on the artistic imagination, our automatic coloring results can be further adjusted by the artist. Alternatively, the method can be used as a lightweight tool to segment contours of a raster image into plausible regions.

To simplify computations and avoid any unnecessary complica-

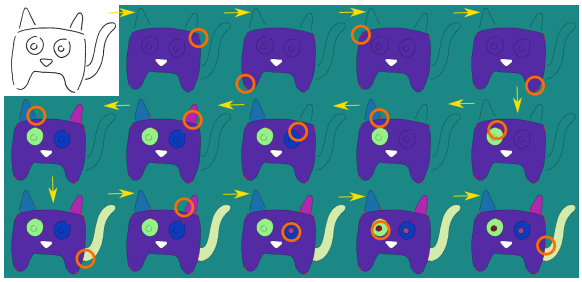


Figure 13: Various steps in the automatic segmentation algorithm. The order is shown using yellow color arrows, segmenting edges are shown in red color, the edge under consideration is marked using orange color circle.

tions due to stroke width, the automatic coloring starts by thinning the contours (i.e. the black regions in the input image) to make it single-pixel width. After thinning, potential endpoints that can contribute to gaps to be filled are found by identifying the open contours (using the neighborhood information available at each pixel). As in Delaunay grouping, our main objective for "Automatic coloring" is to segment the region by appropriately connecting these endpoints with proper Delaunay edges. So, next, we compute the Delaunay triangulation as in Section 4 but using the image with thinned contours as the input. Assuming the contour is Delaunay confirming, the next step is to pair the endpoints. To do so, we pick all Delaunay edges connecting two endpoints. If there are multiple such edges, the smallest edge connecting this pair of vertices is preserved. Depending on the contour, not all endpoints can be paired; and such endpoints might have to be connected to a non-endpoint. To address such cases, if an endpoint is not paired with other endpoints, adapting the idea from NN-Crust [DK99], we choose the shortest adjacent Delaunay edge, which makes an angle $> \pi/2$ with the contour.

Once the right Delaunay edges connecting the endpoints (let them be called segmenting edges) are identified, we use it to split the regions. In other words, we ensure that the triangles (Tri_A and Tri_B) sharing the segmenting edges are labelled differently in the automatic colouring procedure. To facilitate the labelling, depending on the current labels of Tri_A and Tri_B, two situations can arise: In the first one, both Tri_A and Tri_B can be unlabelled, in which we can initiate Delaunay grouping from Tri_A and then Tri_B, ensuring them to have different labels. Another possibility is for Tri_A and Tri_B to have the same labels. In this case, initiating Delaunay grouping from both the triangles will over segment the regions, and randomly initiating Delaunay grouping from one of the triangles might not always assign different labels to both the triangles. To tackle this, we keep track of the flow direction during Delaunay grouping, i.e. the triangle from which the current triangle is getting colored, and do the following:

- $FLOW(Tri_A) = Tri_B$ - means the color flows from Tri_B to Tri_A. In this case, we can safely initiate the Delaunay Coloring with new color from Tri_A.
- $FLOW(Tri_B) = Tri_A$ - case under which the color flows from Tri_A to Tri_B. In which, the coloring can be initiated from Tri_B.
- $FLOW(Tri_A) \neq Tri_B$ and $FLOW(Tri_A) \neq Tri_B$ - special case in which there is a larger path from Tri_A to Tri_B without crossing the segmenting edge under consideration. In this case, we can arbitrarily pick Tri_A or Tri_B and start coloring. The intermediate previously colored triangle in the path will prevent the color flowing from Tri_A to Tri_B and vice versa.

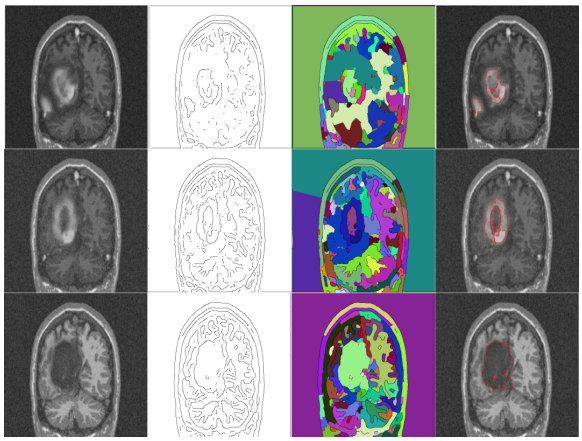


Figure 14: Tumor labelling (Left to Right: Input MRI, result after thresholding and inverting Canny edges, result of automatic coloring, labelled MRIs using our method). [Image source: [PBG09]]

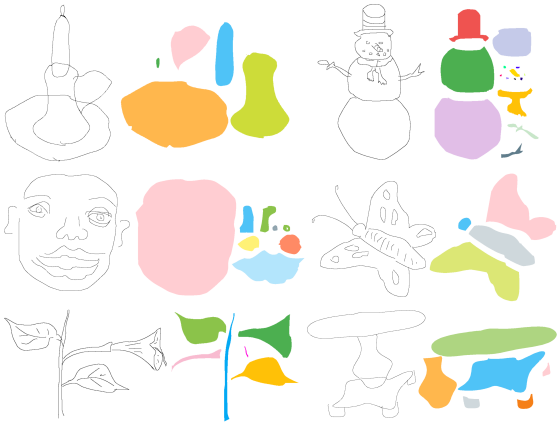


Figure 15: Few sketches (from [ST16]), segmented using our method and merged manually

The algorithm automatically picks and groups/labels the triangles appropriately in the decreasing order of the edge length to result in the required segmented contour. An input sample contour and the various steps in our automatic segmentation are shown in Figure 13.

Applications to automatic image segmentation

Our automatic coloring method can be particularly relevant for fast image segmentation. One such, exciting application is medical image segmentation which is a very well-studied and challenging area. While efficient, advanced tools have been specifically developed for segmentation tasks, the question of properly segmenting

pathological cases is still open. For instance, the medical imaging community has a special interest in identifying tumors in a medical image, especially from MRI scans [PBG09]. In particular to better guide the 3D segmentation input of many recent learning-based methods, manual labelling in 2D is still used for training data. We made a simple test for this application scenario, as a first attempt to alleviate this user-intensive task. Since our algorithm can only deal with contour images, we pre-processed the input MRI grey-scale images using a Canny edge detector, followed by binary thresholding and then image inversion. Figure 14 shows a few MRI scans and various steps in the automatic labeling procedure. These preliminary while promising results, show that our technique may potentially be used either for an initial segmentation process, or as a complementary tool to adjust and correct some resulting segmentation, for which the expert could click in the interest regions and highlight the desired regions appropriately. In summary, we hope that some ideas of our Delaunay painting method can inspire further work in automatic or interactive pathological 2D images labelling, and enrich or simplify this time-consuming process.

Another interesting use-case for this automatic segmentation algorithm is the segmentation of freehand sketches into possibly overlapping regions. The problem itself is a non-trivial one [ST16] because of the presence of significant gaps, absence of strokes, and boundaries represented by multiple strokes. Luckily, our segmentation algorithm can fill large gaps, group regions bounded by various strokes, and group strokes shared between various regions. To get a meaningful result by appropriately resolving ambiguities and occlusion, the automatically generated segments are manually picked and merged. During this manual picking and merging, the user was also given the flexibility to split the regions (if needed). Figure 15 shows the segmentation generated in this way on different challenging cases.

8. Results and Discussion

Implementation: The method was implemented in C++, using the CGAL and OpenCV libraries: we used CGAL for Delaunay triangulation and its Delaunay data-structure for the efficient implementation of Delaunay painting. Whereas, image processing and interaction related functions were implemented using OpenCV. Zhang-Suen's algorithm [ZS84] was used for computing the skeleton of contour images.

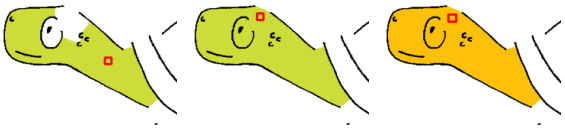


Figure 17: Influence of coloring near the prominent gap. Case 1: Giving color hints in two steps as in (a) and then (b), Case 2: Giving color hint near the prominent gap in the boundary required only one input as in (c).

Qualitative results: Though we explored a solution for automatic colorization, which has many applications on its own, the interactive version of our method is the most relevant for artistic colouring, where the artists needs to be in the loop to pick and drop the appropriate colors (irrespective of whether the underlying contour is segmented or not). Figures 28, 29 and 30 show that artists are able to generate visually pleasing results. Moreover, a demo of our system is given in the companion video.

Time: Our tool is light-weight and responds at an interactive rate. The experimentation was conducted on a Core i3 processor @ 1.70 GHz speed and 4GB RAM. In average, the Delaunay computation, Delaunay grouping, Color diffusion, and Aesthetic curve completion took around 0.1, 0.1, 0.5 and 88 seconds respectively. The time required for Color filling and color diffusion is negligible compared to the gap-filling procedure, which fortunately needs to be done only once at the end of the coloring procedure.

Coloring order: Since we give independent control to each triangle, we can attain the expected result irrespective of the coloring order. Figure 16 shows a sample sketch colored in several different orders. The user might need to give more hints depending on the order chosen, but thanks to our simple flow mechanism, the expected result can always be achieved.

Tips and tricks: The results and the time taken demonstrate that our tool is straightforward to use. Still, we can make coloring tasks easier by using a couple of tricks. The first one is to pre-fill the most

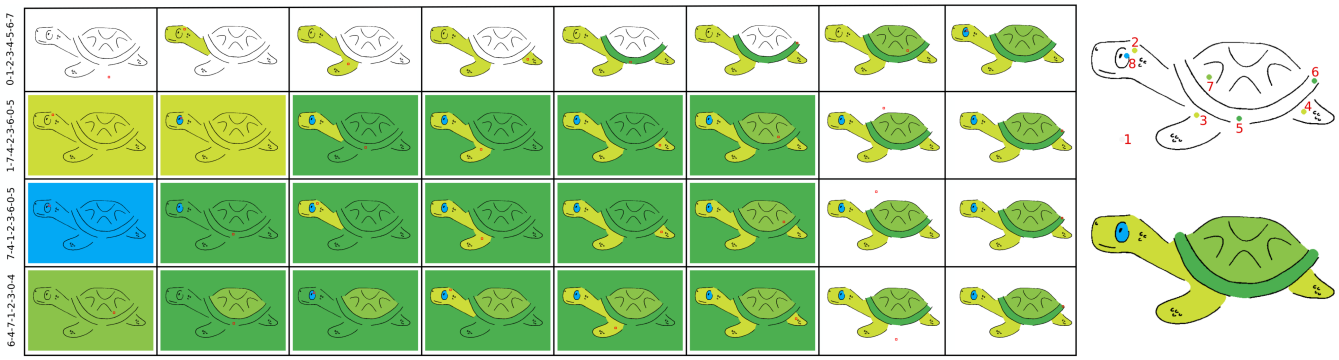


Figure 16: A sketch colored using same colors in different orders (the color hint number and the final result are shown in the right).

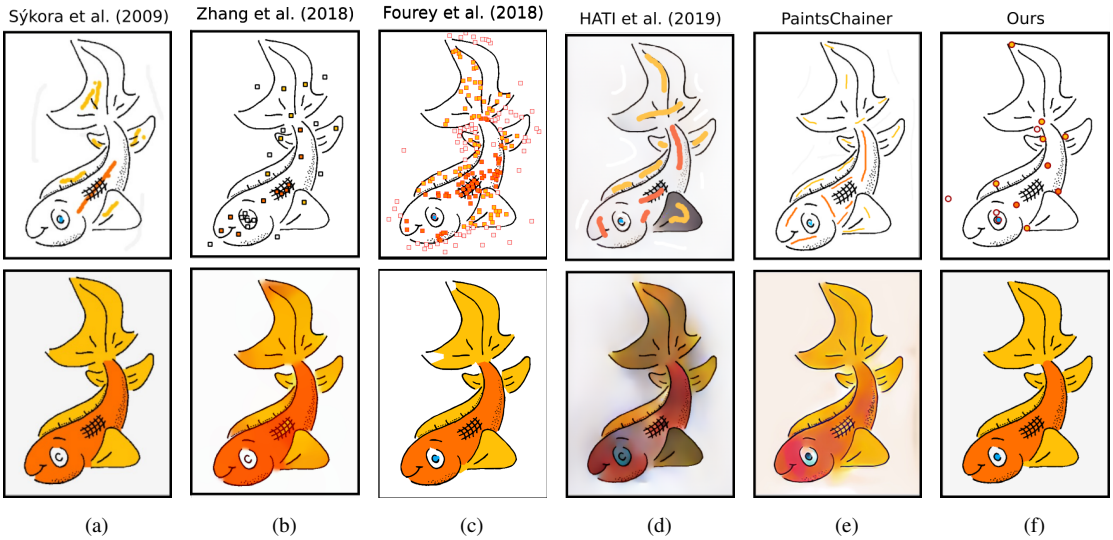


Figure 18: Delaunay painting vs related work. First row shows the input given to respective algorithms and second row shows the resulting colored sketches.

significant region at the border of the image by a specific background color.

This avoids the unwanted spreading of other colors to the background, hence reducing the need for recoloring and unnecessary confusions. Secondly, whenever filling a region with a color, we encourage the user to click near the largest, unwanted gap in the boundary.

This indeed will help the system to pick the locally largest Delaunay triangle and hence will prevent the need of clicking again on some other part to fill the same color. Figure 17 shows an example of how the coloring near a prominent gap influenced the final result (Note that, depending on the gap size and on the number of gaps around a region, the user sometimes has to give multiple hints to satisfactorily color a region).

8.1. Comparative Coloring Scenarios & Results Analysis

Let us now compare our coloring framework with related existing works through an example illustrated in Figure 18. Firstly, Figure 18(a) shows the result of Lazybrush [SDC09] for a sample set of contours. As can be seen in this result and as discussed in their paper, the boundaries are not good-looking in an aesthetic sense. Also, coloring a drawing using a LazyBrush is not necessarily straightforward for a novice user, since the coloring highly depends on the stroke size and strength (and hence is more appropriate for a professional user). The user may have to repaint the same region multiple times to get the desired result. Figures 18(b), 18(d), and 18(e) show the result of coloring using [ZLW*18], [Pai20] and [HJRD19] methods. As seen in the figures, these methods result in relevant and satisfactory coloring. However, learning-based approaches, following relatively time-consuming process, are highly dependent on the training data and cannot work for random artistic drawings (cannot be learned in advance since it depends on artist's imagination).

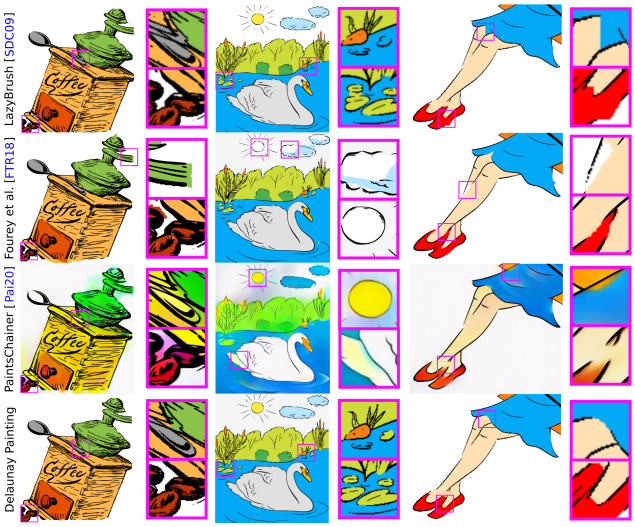


Figure 19: Visual comparisons (differences are highlighted)

Lastly, Figure 18(c) shows a sample contour colored using the line art coloring tool in GIMP (an implementation of [FTR18]). As can be seen, the contour is not colored properly and multiple user inputs are required to paint even a small area, which might perceptually look like a single region. Also note that the coloring of this sketch required 203 mouse clicks, which is a hectic task for the user. Figure 19 shows further visual comparisons between our solution and previous ones, on a few complex examples, showing that the quality of results can often be improved using our method.

A magnified visual comparison of the curve completion procedure for filling gaps used by various algorithms is provided in Figure 20 , while Figure 21 shows the full images. Note that except

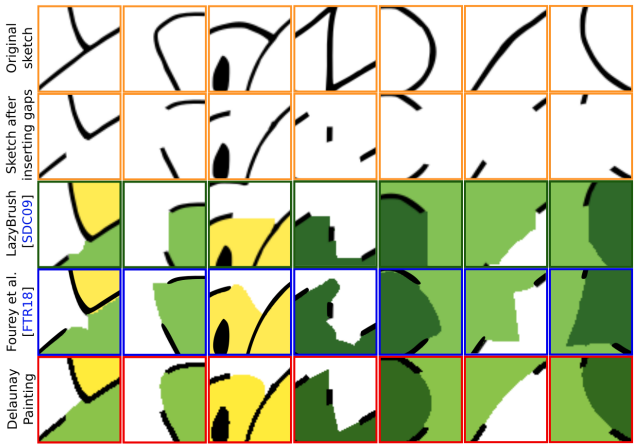


Figure 20: Curve completion by various algorithms (magnified version of Figure 21). From Left to Right: Increasing gap size.

Figure 21: Curve completion experimentation. From left to right: Input sketch, Altered sketch after manually adding gaps of various size, Results of coloring using [SDC09], [FTR18], and our tool.

for small gaps (on which contour beautification was ignored to save time), our method was able to generate visually pleasing gap-filling curves, close to the expected ones.

To still go further in this evaluation, we conducted a user study (with a style inspired from [ZLW*18]) that we report in Appendix.

8.2. Limitations

Though the proposed method is easy to use and has many advantages, it also has a few limitations. The first limitation is that, as the name suggests, our method can only be used for flat-coloring and is not designed for applying shading. Shading could, however, be easily applied on top of our flat-coloring results using existing methods such as [Joh02], [SKv*14], or [HGP*19].

Also, we expect a few strokes to be present to define the boundary that has to be colored. For example, for coloring a contour as in Figure 22(a), the input sketch shown in Figure 22(b) is not enough since the inner shape does not have any contours. But, once we add a few sketch strokes to define the inner border as in Figure 22(d), our method can achieve the expected result.

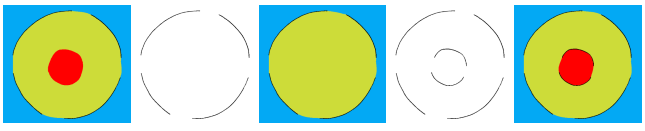


Figure 22: (a) Expected result, (b) Input sketch, (c) Result generated using our method, (d) Required input sample, (e) Result of coloring on required sketch.

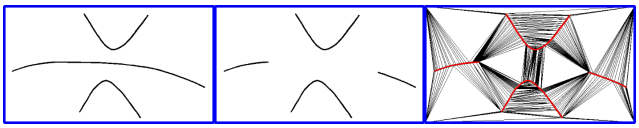


Figure 23: Limitation of Delaunay-based coloring: (a) Expected boundary, (b) Input sketch, (c) Underlying Delaunay triangulation.

Figure 24: [PCS21] analysis: from left to right: Initial Coloring, interactively generated segments, result after merging regions.

Failure case: Sometimes, the Delaunay edge connecting a gap in a contour may be missing, and thus this contour cannot be completed. The problem occurs when there is no empty circle containing both endpoints of a gap (see Figure 23). In other words, we expect the input contours to be "Delaunay confirming". Also, the heuristics used in our curve completion step are not guaranteed to always give a perceptually correct result, and are also prone to noise since they rely on pixel operations.

Lastly, our method only works for handling contour drawings and was not designed for coloring grayscale images, which are a failure case for us (contrary to the method in [ISSI16]).

9. Conclusion

In this paper, we introduced a simple framework for coloring raster sketches with gaps in the contours. We first presented a novel Delaunay-triangulation-based coloring method, using an intuitive flow mechanism introduced in [PCS21] (where there is no notion of colors, but two options as shown in Figure 24 to interactively segment or merge regions which are tailored for this specific application). Using the SIMVC concept and a few sharp corner heuristics, we introduced a method to connect gaps in the contour, enabling to give an aesthetic finish to the colored sketch. Finally, a color diffusion method was used to save user efforts, avoiding them to individually color each region in areas with dense patterns such as hatches. With minimum effort, the artist/user can modify or adjust the result. Our interactive method is an easy-to-use tool from different aspects, such as comparatively minimal input, independent color-ordering and quick response time. As our results show, our simple framework can be used to color a wide variety of hand-drawn sketches in a visually pleasing way.

We have also presented an automatic companion method and

applied it to efficient image segmentation. Our pipeline provides a handy tool for automatic pre-segmentation and fast interactive segmentation correction for 2D medical images, which is a well-studied while still challenging application.

In the future, we would like to strengthen the color diffusion process to appropriate regions by analyzing the contour globally, in combination with our current Delaunay-based local analysis. Lastly, we plan to investigate the temporal consistency of our method, by exploring the robust propagation of color hints through the sketches corresponding to the different frames of an animation.

References

[CSG*17] CHEN J., SHEN Y., GAO J., LIU J., LIU X.: Language-based image editing with recurrent attentive models. *CoRR abs/1711.06288* (2017). arXiv:1711.06288. 3

[DGG03] DEY T. K., GIESEN J., GOSWAMI S.: Shape segmentation and matching with flow discretization. In *Algorithms and Data Structures* (Berlin, Heidelberg, 2003), Dehne F., Sack J.-R., Smid M., (Eds.), Springer Berlin Heidelberg, pp. 25–36. 3

[DK99] DEY T. K., KUMAR P.: A simple provable algorithm for curve reconstruction. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms* (USA, 1999), SODA '99, Society for Industrial and Applied Mathematics, p. 893–894. 9

[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. 7

[FHOO17] FURUSAWA C., HIROSHIBA K., OGAKI K., ODAGIRI Y.: Comicolorization: Semi-automatic manga colorization. In *SIGGRAPH Asia 2017 Technical Briefs* (New York, NY, USA, 2017), SA '17, Association for Computing Machinery. 2

[FLB16] FAVREAU J.-D., LAFARGE F., BOUSSEAU A.: Fidelity vs. simplicity: A global approach to line drawing vectorization. *ACM Trans. Graph. 35*, 4 (jul 2016). 3

[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* (Goslar, DEU, 2018), EG VMV '18, Eurographics Association, p. 1–9. 3, 11, 12, 14

[GTF94] GANGNET M., THONG J.-M. V., FEKETE J.-D.: Automatic gap closing for freehand drawing. In *SIGGRAPH'94 Technical Sketch Submission* (12 1994). 3

[HA17] HENSMAN P., AIZAWA K.: cgan-based manga colorization using a single training image. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)* (Los Alamitos, CA, USA, nov 2017), vol. 3, IEEE Computer Society, pp. 72–77. 2

[HGP*19] HUDON M., GROGAN M., PAGÉS R., ONDŘEJ J., SMOLIĆ A.: 2dtoonshade: A stroke based toon shading system. *Computers Graphics: X 1* (2019), 100003. 12

[HJRD19] HATI Y., JOUET G., ROUSSEAUX F., DUHART C.: Paintstorch: A user-guided anime line art colorization tool with double generator conditional adversarial network. In *European Conference on Visual Media Production* (New York, 2019), CVMP '19, Association for Computing Machinery. 3, 11

[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 Transactions on Graphics (Proc. of SIGGRAPH 2016) 35*, 4 (2016). 2, 12

[Joh02] JOHNSTON S. F.: Lumo: Illumination for cel animation. In *Proceedings of the 2nd International Symposium on Non-Photorealistic Animation and Rendering* (New York, 2002), NPAR '02, Association for Computing Machinery, p. 45–ff. 12

[KJPY19] KIM H., JHOO H. Y., PARK E., YOO S.: Tag2pix: Line art colorization using text tag with secat and changing loss, 2019. arXiv:1908.05840. 3

[KMN89] KAHANER D., MOLER C., NASH S.: *Numerical Methods and Software*. Prentice-Hall, Inc., USA, 1989. 7

[LDL*19] LIU F., DENG X., LAI Y.-K., LIU Y.-J., MA C., WANG H.: Sketchgan: Joint sketch completion and recognition with generative adversarial network. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 5823–5832. 3

[LWCO*07] LUAN Q., WEN F., COHEN-OR D., LIANG L., XU Y.-Q., SHUM H.-Y.: Natural Image Colorization. In *Rendering Techniques* (2007), Kautz J., Pattanaik S., (Eds.), The Eurographics Association. 2, 3

[Mor92] MORETON H.: Minimum curvature variation curves, networks, and surfaces for fair free-form shape design. Ph.D. thesis; Berkeley, CA, USA. 7

[Pai20] PAINTSCHAINER: Petalica paint, online demo (by preferred networks, inc.), accessed in june 2020. URL: https://petalica-paint.pixiv.dev/index_en.html. 3, 11, 14

[PBG09] PRASTAWA M., BULLITT E., GERIG G.: Simulation of brain tumors in mr images for evaluation of segmentation efficacy. *Medical Image Analysis 13*, 2 (2009), 297 – 311. Includes Special Section on Functional Imaging and Modelling of the Heart. 9, 10

[PCS21] PARAKKAT A. D., CANI M.-P., SINGH K.: Color by numbers: Interactive structuring and vectorization of sketch imagery. CHI '21, Association for Computing Machinery, p. 1–11. 2, 3, 4, 5, 12

[PM16] PARAKKAT A. D., MUTHUGANAPATHY R.: Crawl through neighbors: A simple curve reconstruction algorithm. *Computer Graphics Forum 35*, 5 (2016), 177–186. 3

[PMGC20] PARAKKAT A. D., MADIPALLY P., GOWTHAM H. H., CANI M.-P.: Interactive Flat Coloring of Minimalist Neat Sketches. In *Eurographics 2020 - Short Papers* (2020), Wilkie A., Banterle F., (Eds.), The Eurographics Association. 2, 3, 5

[PMM18] PARAKKAT A. D., METHIRUMANGALATH S., MUTHUGANAPATHY R.: Peeling the longest: A simple generalized curve reconstruction algorithm. *Computers Graphics 74* (2018), 191 – 201. 3

[SCH20] SU J.-W., CHU H.-K., HUANG J.-B.: Instance-aware image colorization, 2020. arXiv:2005.10825. 2

[SDC09] SÝKORA D., DINGLIANA J., COLLINS S.: Lazybrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum 28*, 2 (2009), 599–608. 3, 11, 12, 14

[SISSI17] SASAKI K., IIZUKA S., SIMO-SERRA E., ISHIKAWA H.: Joint gap detection and inpainting of line drawings. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 5768–5776. 3

[SKv*14] SÝKORA D., KAVAN L., ČADÍK M., JAMRIŠKA O., JACOBSON A., WHITED B., SIMMONS M., SORKINE-HORNUNG O.: Ink-and-ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters. *ACM Trans. Graph. 33*, 2 (Apr. 2014). 12

[ST16] SCHNEIDER R. G., TUYTELAARS T.: Example-based sketch segmentation and labeling using crfs. *ACM Trans. Graph. 35*, 5 (July 2016). 9, 10

[SZC*20] SHI M., ZHANG J.-Q., CHEN S.-Y., GAO L., LAI Y.-K., ZHANG F.-L.: Deep line art video colorization with a few references, 2020. arXiv:2003.10685. 3

[YBC*19] YOO S., BAHNG H., CHUNG S., LEE J., CHANG J., CHOO J.: Coloring with limited data: Few-shot colorization via memory-augmented networks, 2019. 2

[ZLSS*21] ZHANG L., LI C., SIMO-SERRA E., JI Y., WONG T.-T., LIU C.: User-guided line art flat filling with split filling mechanism. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021). 3

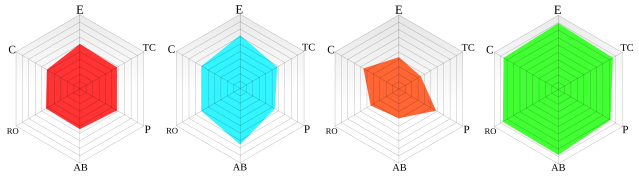


Figure 25: Visualization of our user study. From left to Right: LazyBrush, PaintsChainer, GIMP, and Ours.

[ZLW*18] ZHANG L., LI C., WONG T.-T., JI Y., LIU C.: Two-stage sketch colorization. *ACM Trans. Graph. 37*, 6 (Dec. 2018). 3, 11, 12, 14

[ZMG*19] ZOU C., MO H., GAO C., DU R., FU H.: Language-based colorization of scene sketches. *ACM Trans. Graph. 38*, 6 (Nov. 2019). 3

[ZS84] ZHANG T. Y., SUEN C. Y.: A fast parallel algorithm for thinning digital patterns. *Commun. ACM 27*, 3 (Mar. 1984), 236–239. 10

User Study

Inspired by [ZLW*18], we conducted a user study to evaluate the strengths and weaknesses of the proposed system. We invited eight non-experts users, aged between 18 to 45 to participate in the interactive coloring session remotely. The users were given an initial tutorial on how to use the interface and then asked to randomly pick and color one of the sketches shown in Figure 30 using our tool and the other three openly available coloring tools (LazyBrush [SDC09], PaintsChainer [Pai20], and GIMP [FTR18]). It has to be noted that the users were explicitly directed to define the background color first, to make the coloring process intuitive. At the end of the coloring process, the users were asked to participate in a multi-dimensional survey to grade each interface (on a scale of 0-10) based on the following aspects:

- Easiness (E) - How easy it is to give coloring
- Time consumption (TC) - How time-consuming the entire process is
- Predictability (P) - How easy it is to identify where colors should be given to get the expected results
- Aesthetic beauty (AB) - How good looking the final results are
- Region Obedience (RO) - Whether the user was able to capture the expected boundary
- Control (C) - How easy it is to control the color propagation

As it can be seen in the results of Figure 25, our proposed method has largely outperformed the competitors from all these aspects. Let us report some common observations made by various users.

- LazyBrush: Difficult to predict and control the effect of brush size. It was easy to color small regions and users did not have to worry about precise mouse location.
- PaintsChainer: The final result was having an artistic feeling (like in a painting). But color bleeding along with random colors made the final result look bad. Also, this method was not able to capture the boundaries correctly.
- Fourey et al. (GIMP): Coloring small pieces makes it time-consuming. Difficult to control the colors near borders. Small uncolored regions are left in some areas.

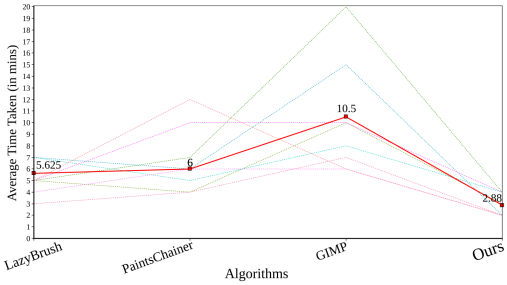


Figure 26: Time taken (average time is shown in red bold line) by various users to color the sketches using different tools

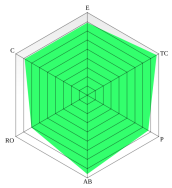


Figure 27: Second user study: Evaluation of our solution.

Users were also asked to comment on the advantages and disadvantages of our tool, and which resulted in he following remarks:

+ Easy to use without worrying about the gaps;
+ Less work since colors are spreading;
+ Captures the borders well;
- Editing/refining small regions or redoing colors is time consuming. A "lazy" paint brush as in LazyBrush would have been nice;
- Should have included a better color palette.

In general, though our tool currently does not support functionalities like undo/redo and does not have an advanced color palette, our simple interface allowed the users to color the sketches without worrying about contour gaps. Our method could not only capture the user's intended boundaries, but also simplify user interaction and provide aesthetically pleasing results. We also report the time taken by different users to color the sketch using various tools in Figure 26. It can be seen that users were able to quickly color complex sketches using our tool.

To get a feedback of how easy our interface is (without comparing to other methods), ten other users aged between 13 to 50 were asked to evaluate our interface, by grading each attribute on a scale of 0 to 10. The results are shown in Fig. 27, with the following comments:

- Following the guideline, it is a convenient and light interface.
- Cannot add free-form shapes.
- Very practical as far as we start by background color toward more detailed ones.
- A pre-visualization of color change (before click) may be useful.

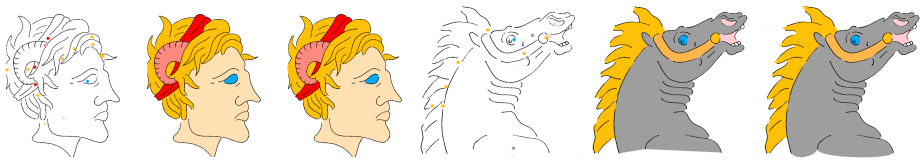


Figure 28: A few results of coloring given using our tool.

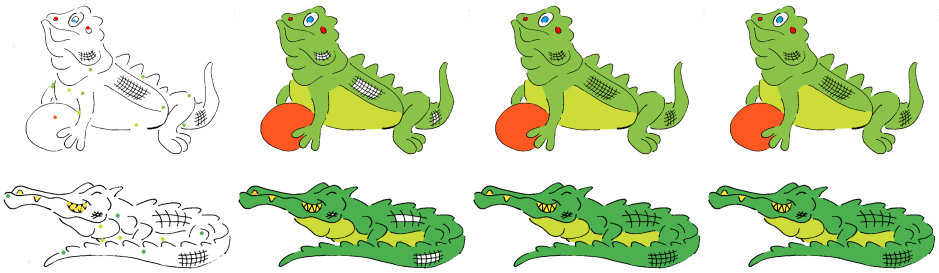


Figure 29: A few results of coloring given on simple sketches using our tool.

Figure 30: Various complex sketches colored using our method (original sketches taken from Pixabay).