Color by Numbers: Interactive Structuring and Vectorization of Sketch Imagery

Amal Dev Parakkat
EEMCS - Dept. Intelligent Systems,
TU Delft, Netherlands

Marie-Paule Cani
École Polytechnique, CNRS (LIX),
IP Paris, France

Karan Singh
Computer Science,
University of Toronto, Canada

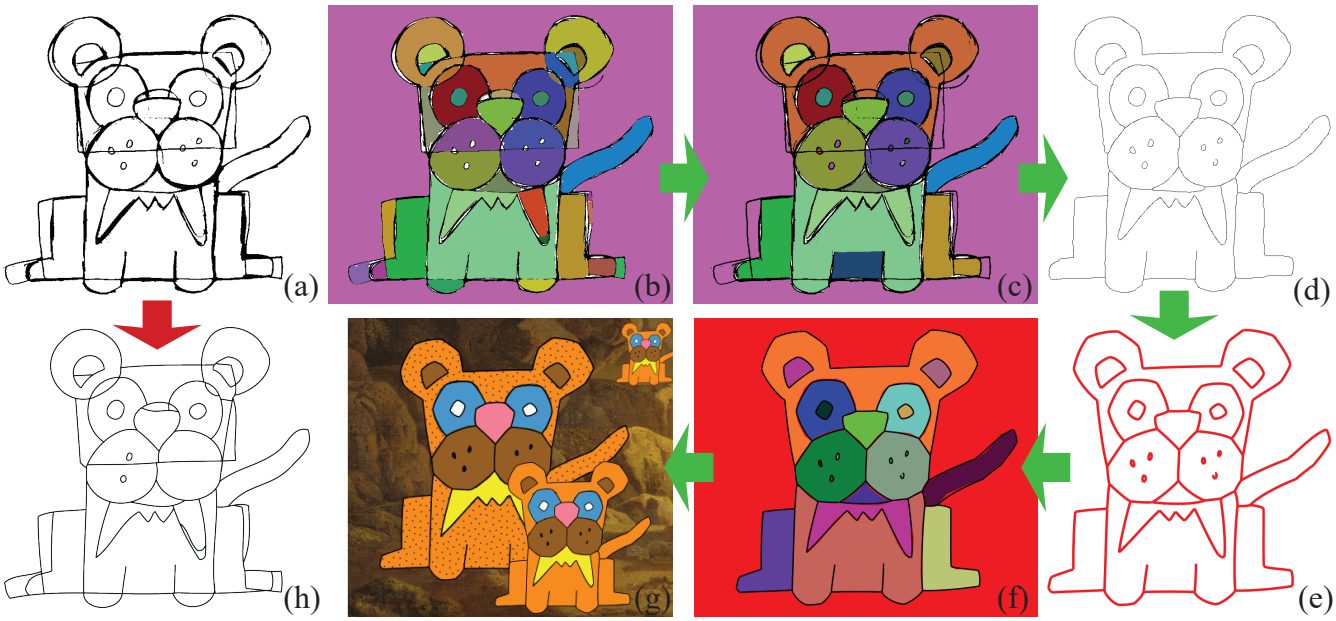

Figure 1: Color-by-Numbers takes an input raster sketch (a); and uses a Delaunay-based subdivision to color sketch regions (b); Users interactively recolor the sketch using a click, drag-drop interface, to quickly merge or segment regions as desired (c); The boundaries of these colored closed regions distinguish desired output curves from construction lines (d); The result is a neatened and structured vector drawing that meets user intent (e); that can be colored (f); textured and layered (g). In contrast, automatic vectorization approaches [8] are unable to vectorize messy input sketches as desired, without user input (h).

ABSTRACT

We present a novel, interactive interface for the integrated cleanup, neatening, structuring and vectorization of sketch imagery. Converting scanned raster drawings into vector illustrations is a well-researched set of problems. Our approach is based on a Delaunay subdivision of the raster drawing. We algorithmically generate a colored grouping of Delaunay regions that users interactively refine by dragging and dropping colors. Sketch strokes defined as marking boundaries of different colored regions are automatically neatened using Bézier curves, and turned into closed regions suitable for fills, textures, layering and animation. We show that minimal user interaction using our technique enables better sketch vectorization than state of art automated approaches. A user study, further shows our interface to be simple, fun and easy to use, yet effectively able to process messy images with a mix of construction lines, noisy and incomplete curves, sketched with arbitrary stroke style.

CCS CONCEPTS

• **Applied computing** → *Fine arts*; • **Human-centered computing** → *Systems and tools for interaction design*; • **Computing methodologies** → **Image manipulation**.

1 INTRODUCTION

Despite advances in vector drawing tools such as Adobe Illustrator, Inkscape [1, 13], digital raster sketching and traditional pen-and-paper drawing remain a visceral form of creative expression [6]. Children in particular, thrive on such a medium from an early age. In the current world context, with communication increasingly shifting online, such sketches are scanned, and sometimes redrawn using vectors, so they may be neatened, textured, replicated and even animated. We present a novel interactive approach to this problem of cleanup, neatening, structuring and vectorization of rough raster sketches [32] (Figure 1).

Increasingly, interfaces for creative expression are being designed to target simple, playful interaction, by non-experts, using

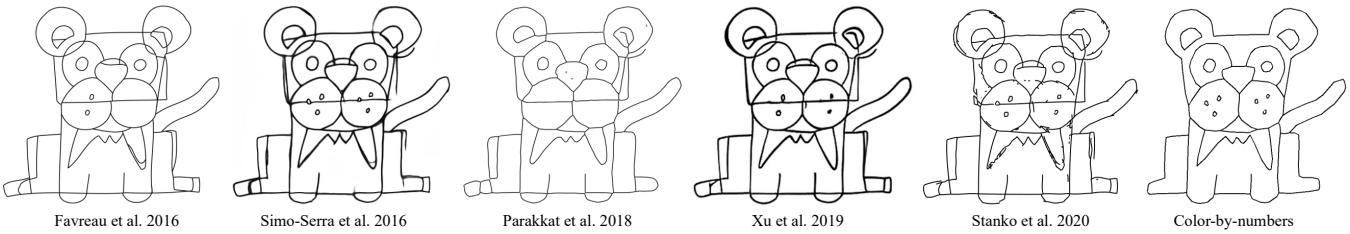

Figure 2: Comparison to prior-art [8], [28], [23], [31], [29], and our approach (left to right), given Figure 1(a) as input.

simple mouse/trackpad interaction [27]. We are driven by a similar objective to allow novice users to transform their rough raster sketches into neat, structured vector drawings (Figure 1).

Vectorization of raster sketches is an important ongoing research problem [32]. There are two aspects to vectorization: recovering vector curves [8]; and structuring these curves [5], that are typically handled in sequence. The majority of vectorization research is designed for automation and sketches that have a cleaner, professional quality than those seen in this paper. These approaches have algorithm parameters that can tune the vector output, such as an intensity value to suppress light construction lines. Despite such parameters, users typically have to fix the vector output post-hoc to match their expectations, deleting unwanted curves, fixing missing parts and other inaccuracies. Given the desired vector drawing, users can then define groupings of strokes and closed regions that can be filled and layered.

Region segmentation and grouping defines the core of our vectorization approach. We use points on the raster sketch to construct a Delaunay triangulation of space [23], whose groupings define coherent regions guided by the sketch strokes. These regions lend themselves to easy refinement by simply clicking on individual or groups of triangles. We use a playful region coloring metaphor, by which different colors define the salient regions of the sketch, and only portions of sketch strokes that separate two different colored regions are vectorized. We thus benefit from user's directly communicating their design expectations as part of our vectorization and structuring algorithm.

Our evaluation is threefold: we compare our workflow to automatic state-of-the-art vectorization algorithms, followed by vector drawing clean-up and region structuring using a commercial vector editing tool Inkscape [13]; we conduct an informal study of our approach with 12 novice users who were all able to use the tool effectively; and we show a diverse design gallery from a rough raster sketch benchmark [32], vectorized using Color-by-Numbers.

2 RELATED WORK

Our work is broadly related to research in sketch vectorization and playful novice interfaces.

Vectorization of sketches can be broadly classified into two categories: raster and digital, based on the input sketch representation.

Raster sketch vectorization algorithms [8, 16, 23], typically, process image pixels to produce pixel groupings representing sketch strokes from which a skeletal representation for the stroke is extracted. These stroke skeletons are then fit using smooth curves (typically cubic splines). The main difference between methods for raster sketch vectorization is in geometric or perceptual sketch priors used and the algorithm used to perform stroke grouping. Data-driven techniques, using CNNs to better group stroke pixels have also been explored [28, 31]. More recently, a grid-based approach to vectorizing raster input has been considered [29]. Designed for automation, these approaches have global parameters that can impact the overall output, but are generally ill-suited to messy sketches with local variation and inaccuracies (Figure 2). In contrast, our approach engages a user in a simple region coloring task, from which the user-intended stroke grouping is meaningfully inferred. Technically, our approach is founded on a Delaunay triangulation of space [4], constrained by image pixels, similar to [23]. We differ however, both in philosophy, using the triangulation as a sub-structure for easy interactive grouping of regions, as well as in the approach used to group, merge and represent the resulting vector curves.

Vectorization of clean raster sketch input, such as professionally created cartoon sketches is also an important problem. Defining individual strokes is less problematic in such drawings and research is focused on using topology-based [19] and frame-field based techniques [3], to produce coherent curve intersections and regions of the drawing. Such approaches however, are ill-suited to the rough drawings done by novices.

Image vectorization of photographs (instead of sparse line drawings) for the creation of posterized images, is also an important area of research [12, 14, 22]. We are inspired by interactive methods to find closed contours in images [30].

Complementary and concurrent to our research, Yan et al. [32] confirm the premise that rough sketch cleanup and vectorization is an ambiguous and difficult problem, motivating interaction to communicate user intent. While they do not present an interactive solution, they provide a benchmark of diverse raster images in 5 categories (art, logo, architecture, fashion, product), along with artist hand-created clean-up, structuring, and vectorization (Figures 15, 16). They further compared the performance of 7 state-of-the-art automatic vectorization techniques (5 of which we show in Figure 2) on their raster image benchmark. The comparison validates our choice of the Delaunay approach [23], with inherent region structuring, good vector quality and run-time performance, as a good algorithmic basis for an interactive solution.

Digital sketches drawn in rough free-hand fashion are also the subject of research that transforms them into neat and structured vector drawings. Unlike raster images, digitally sketched strokes already carry important information pertaining to stroke position and ordering, drawing speed and pressure. Research in this area

often uses geometric and perceptual principles to group, merge, neaten and structure the raw sketch strokes [2, 15, 20, 21].

Playful Interfaces aim at tools designed for a wide audience, that empower creativity by imbibing the playful exploratory aspects that come naturally to artists working in traditional media [24]. Such tools have a "low threshold" allowing even children to accomplish simple tasks quickly, and "high ceiling" providing enough control for professional quality work. Examples of such tools span color interaction [26, 27], shape modeling [10], animation [11], programming [17], and even tangible image-based drawing [25]. Our technique is similarly designed for the digital vector processing of casual pen-and-paper sketches in the wild.

3 ALGORITHM AND WORKFLOW

Vectorization and structuring of rough raster sketches is confounded by the varying presence of unwanted construction lines, hatching, extraneous noise, annotation and text. Color-by-Numbers addresses the vectorization of such sketches by automatically subdividing the sketch into uniquely colored closed regions, whose boundaries define the strokes to be vectorized. Users can visually inspect curves resulting from region coloring, and refine the result by interactively splitting, merging and recoloring regions. At any stage, the colored region induce a spatially structured vector output. Our workflow can be thus abstracted into four steps: automatically computing an initial region coloring based on a constrained Delaunay triangulation of the sketch; interactive subdivision, merging, and recoloring of regions; grouping and simplification of strokes based on the user-defined coloring; final creation of a structured vector drawing.

3.1 Computing an initial coloring

Rough sketches, commonly have overlapping, spurious and intermittent (with gaps) strokes. This is both due to the unstructured and free-form nature of sketching, and the casual scanning of paper sketches. Given a raster sketch as a binary image (grey-scale images may be thresholded into binary images [1]), we first construct a Delaunay triangulation for the point-set corresponding to the sketch (black) pixels. We observe that in general very small (in terms of maximum edge length) Delaunay triangles are created along sketch strokes, and in between small gaps in overlapping or intermittent strokes. Larger triangles define parts of regions bounded by these sketch strokes, and perceptually meaningful regions are connected clusters of these large Delaunay triangles.

Our initial coloring of the sketch thus follows a simple greedy approach: iteratively picking a large uncolored triangle, coloring it uniquely, and propagating the color across large shared edges into adjacent uncolored triangles.

Formally, we initialize all triangles as uncolored; we then iteratively pick and uniquely color C_T, the largest (in terms of circumradius) uncolored triangle T with max. edge length $> \delta$ (by default $\delta =$ 5 pixels for ≈ 1 mega-pixel images), and recursively flow color C_T, to uncolored adjacent triangles across large shared edges (length $> \delta$) to grow a region around it (Figure 3). Triangle T is referred to as the seed triangle for the region colored C_T (Figure 3).

Figure 1(b) shows an initial sketch coloring, where large bounded regions with small gaps $< delta$ do not spill into adjacent regions. Note also, by design, that small gaps between overlapping strokes

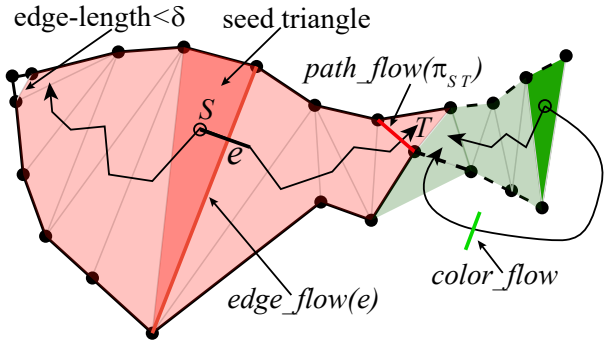

Figure 3: The largest uncolored Delaunay triangle is picked as a seed (red) and recursively propagated across Delaunay edges of length $> \tau$ to grow a region. The $path\_flow$ to a triangle from its seed is bounded by its smallest $edge\_flow$ and the $color\_flow$ of any triangle in a region is the maximum $path\_flow$ to it from its seed triangle (see green region).

are left uncolored. Also note that this simple coloring is not perfect: perceptually coherent regions may be subdivided into multi-colored parts; and open strokes may bound regions, or be embedded in a uni-colored region. Users now communicate design intent by interactively recoloring these Delaunay regions.

3.2 Interactive Delaunay recoloring

Theoretically, users can click and recolor each Delaunay triangle independently. We aim to reduce the tedium of individually clicking each desired Delaunay triangle by meaningfully propagating color out from a clicked triangle. Color is recursively propagated out from a clicked seed Delaunay triangle similar to the initial region coloring, albeit with different termination criteria. While the initially colored regions terminated at small Delaunay edges or already colored triangles, region recoloring potentially reassigns color to already colored triangles.

To this effect, We define a measure $color\_flow$, that captures the likelihood of a region's seed triangle(s) [1], to flow or assign its color to another triangle.

We use the dual of the Delaunay triangulation to define a region adjacency graph G, where each Delaunay triangle T is a vertex v_T, and an edge $e = (v_T, v_{T'})$ connects adjacent triangles T, T' (except triangles separated by a sketch stroke, i.e. edges $< \sqrt{2}px$ corresponding to adjacent pixels in the image). For each edge $e \in G$, we define a capacity for color to flow across the edge e, as $edge\_flow(e)$ set to the length of its dual Delaunay edge (Figure 3). Given any path $\pi(u, v)$ connecting two vertices $u, v \in G$, the capacity for color to flow between u and v along π is defined by its bottleneck edge, i.e. $path\_flow(\pi) = min_{e \in \pi}(edge\_flow(e))$. Now, the $color\_flow(u, v)$ between any two vertices $u, v \in G$, is the maximum $path\_flow$ of all paths connecting u and v in G (or 0 if u, v are not connected in G). In other words for connected vertices u, v, $color\_flow(u, v) = max_{\pi(u,v)}(path\_flow(\pi))$.

[1] Adjacent regions that are intentionally merged by a user assigning them the same color, would have multiple seed triangles

Figure 4: Stroke and uncolored pixels shown in black (left), bounded by different colored regions are retained and medially thinned (middle), and further turned into skeletal chains of pixels, suitable for vectorization (right).

When regions are colored, every Delaunay triangle T (a vertex v_T in G), stores its current flow $color\_flow(v_S, v_T)$, i.e. the $color\_flow$ capacity between the triangle T, and the seed triangle S for the region to which it belongs. Now, when a user clicks a new seed triangle S' (a vertex $v_{S'}$ in G) to reassign it a color $C_{S'}$, the color $C_{S'}$ is propagated recursively across adjacent triangles to a triangle T, as long as $color\_flow(v_{S'}, v_T)$ is greater than its current flow $color\_flow(v_S, v_T)$. Upon color reassignment of a triangle T to $C_{S'}$ the current flow is updated to $color\_flow(v_{S'}, v_T)$.

We ensure efficient recursive propagation across Delaunay edges along max flow paths (color flows across the largest unprocessed Delaunay edge first), by using a priority queue with priority based on length of the Delaunay edges. A sample interactive Delaunay recoloring of Figure 1(b) can be seen in Figure 1(c).

3.3 Sketch simplification and representation

Users specify desired strokes by coloring their two sides differently. In this step, these desired strokes are identified, grouped, and simplified. Initially, the the gaps between strokes are filled (identified by uncolored triangles). The filled sketch is further used for simplification and then replaced with Bézier curves. Finally, along with a set of Bézier curves, based on curve connectivity, our method also computes a set of layered regions.

Simplification:

From the colored sketch, we extract all the stroke pixels (black), and uncolored (white) pixels corresponding to small gaps in-between proximal and overlapping strokes (Figure 1(c)). Figure 4(left) shows a zoomed in sketch region with both the stroke and uncolored pixels colored black. A subset Q of these pixels is selected to represent the desired and simplified sketch strokes. A pixel $q \in Q$, only if there exist some colored pixels p_1 and p_2 such that:

- $\mid q - p_1 \mid - \mid q - p_2 \mid < \tau$ (default $\tau = \sqrt{2}$)
- $Color(p_1) \neq Color(p_2)$
- The line connecting q to p_1 and p_2 contains only stroke or uncolored pixels.

The above medial formulation with its tolerance τ for discrete pixels, does not guarantee that Q represents a single-pixel width stroke (Figure 4(middle)). We thus further simplify Q by extracting its skeleton [33] (Figure 4(right)). Figure 1(d) shows an example of user desired strokes after simplification.

Curve Network:

Once simplified pixels Q are generated, we create a network graph NG=(V_{NG}, E_{NG}) [8], with vertices representing the endpoints and branching points of the simplified shape. Start and end vertices on isolated closed loops are arbitrarily chosen. Edges are then created between vertices having a direct pixel path (a 2D poly-line) between them. We then attempt to replace each graph edge with a piece-wise smooth vector curve.

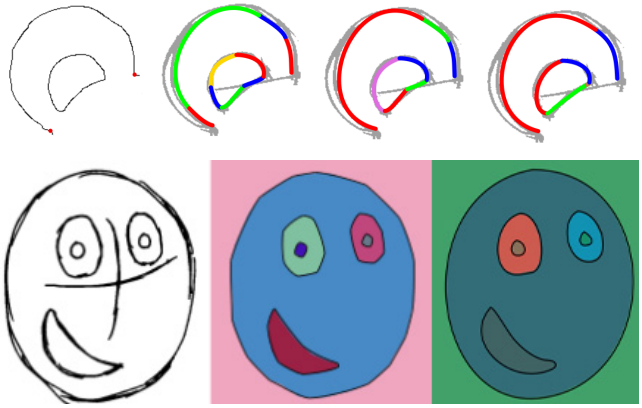

Figure 5: Impact of increasing tolerance ϵ on fitting cubic Bezier segments ($\epsilon/segment - length = 0.033, 0.05, 0.1px$ (top left-right); vector output with low and high ϵ (bottom).

There are numerous approaches to curve fitting [18], any of which can be applied to our 2D pixel path poly-lines. We fit a sequence of cubic Bézier curves (with C^0 continuity)to each graph edge (a 2D poly-line) as follows:

We first segment the poly-line at sharp corners by finding points of large change in discrete curvature [18].

We then perform an end-point interpolating, iterative least-squares cubic Bezier fit to each segment S [29], minimizing error $\epsilon = \sum_{p \in S} \mid\mid B(t_p) - p \mid\mid^2$, where B is the cubic Bézier curve and $t_p \in [0, 1]$, the parameter corresponding to a pixel p in S.

If fit error ϵ is larger than a user specified accuracy, the segment S is split at a pixel p of largest error and the two sub-segments fit as above using cubic Bézier curves.

Figure 1(e) shows the set of Bézier curves generated to fit the image in Figure 1(d). Figure 5 shows the effect of varying the accuracy parameter ϵ on the resulting curves. The approach can be further improved by enforcing high-order continuity between curve segments [8].

Layering the regions:

The user intended Color-by-Number regions may over-segment some closed regions to capture open curves in the sketch, for example the mouth and front fin of the fish in Figure 9(i). In order to represent closed regions by vector polygons, we thus apply a recursive flood-fill algorithm on a raster version of the final vector sketch. The outer boundary of each such region is extracted and simplified using a recursive polygon edge snapping (such that the maximum distance between the pixels and the corresponding polygon edge is less than a threshold). The polygons resulting from this recursive flood-fill, extraction and simplification process, are ordered front to back based on increasing area, thus layering the sketch into vectorized regions (Figure 9(j)).

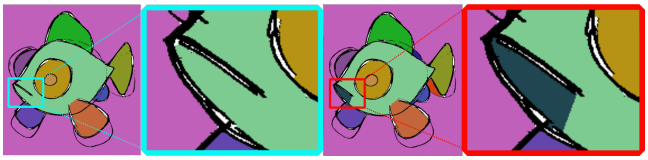

Figure 6: Fill color: the curve for the mouth of the fish (left) is retained by recoloring the region below it using a simple click interface (right).

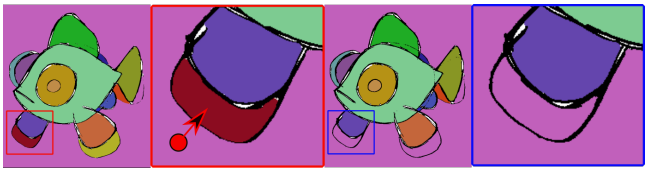

Figure 7: Pick and drop color: the background region (left) is grown by picking and dropping its color across an unwanted construction line into an adjacent region (right).

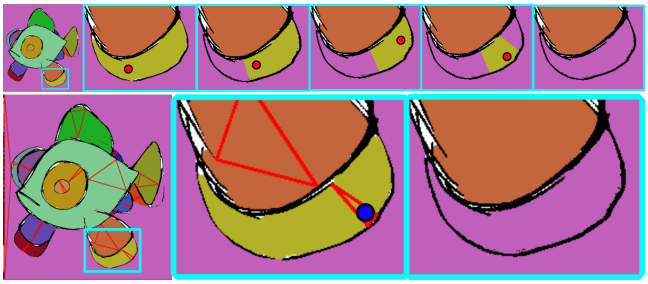

Figure 8: Recoloring without (top) and with (bottom) visualizing seed triangles.

4 USER INTERACTION

Our initial guess provides a starting point that users refine by recoloring to fix two possible problems: a relevant group of strokes has the same color on both sides; and undesired strokes have different colors on either side. We provide two simple options: Fill color, and Pick & drop color to rectify these mistakes.

Fill color: Especially in sketches with open curves, the initial coloring may apply the same color to both sides of an open stroke. This can be desirable to remove errant strokes lying within a region, but needs to be edited to retain such a curve. Users can simply click a new seed triangle to fill a region around it with a desired color (Figure 6). The Delaunay structure and *color_flow* based propagation automatically produce regions that are well aligned with stroke boundaries and shape features.

Pick & drop color: As shown in Figure 7, because of shading, construction lines or spurious strokes, the initial coloring may oversegment a single desired region. A "Pick & drop color" option allows users to drag and drop color from an existing region to another, typically adjacent region.

Visualizing seed triangles: As described in Section 3.2, the propagation extent during recoloring, depends on the *color_flow* out of the clicked seed triangles. Intuitively, small seed triangles will spread to a smaller region when adjacent to larger seed triangles. Often, a user may wish to entirely recolor an existing region and this is simply guaranteed by clicking on its existing seed triangle. We thus aid recoloring operations by highlighting all seed triangles in the current coloring. Note, however, that a user is free to interactively click anywhere and not just on seed triangles as part of the recoloring process. As shown in Figure 8, the user may have to click multiple times to recolor a region, alternately accomplished using a single click on its seed triangle.

The various steps involved in coloring a rough sketch using our interface is shown in Figure 9. As can be seen, a number of unnecessary pseudo-regions generated due to unwanted strokes are removed using the Pick & drop color tool (Figure 9(b)-(h)). Later, a new region is introduced using the Fill color tool (Figure 9(i)).

Vector Output: Color-by-Numbers produces vector output in the popular SVG (Scalable Vector Graphics) format. Our SVG files contain two sections: A set of layered regions and a set of cubic Bézier curves. This SVG file can further be manipulated using any vector editing software such as Adobe Illustrator or Inkscape.

5 EVALUATION

We evaluated Color-by-Numbers, with a comparison to automated vectorization (Figure 2), an informal 12 novice user study, feedback from 2 design professionals, and by using our approach to vectorize 40 sketches from the rough sketch benchmark [32].

5.1 User Study

We conducted an informal study with 12 novice participants (aged 12-60), 6 of who had some experience with interactive graphics. We introduced Color-by-Numbers as a game, where users were asked to identify relevant strokes and then color them appropriately. They were shown a demo of coloring a sketch (as shown in Figure 9 and the accompanying video). Based on that visual input and explanation, users then tried to vectorize a given sketch. All 12 participants were able to successfully recolor and vectorize the sketch. Three of them produced the desired vector result by parsimoniously recoloring only the necessary regions. The rest also produced the desired vector result, but with additional hit-and-try recoloring of regions, and observing their impact on the vector output. After playing with a few more sketches, the participants rated our system on a scale of 1-10 (bad-good) on the following questions: How easy was it to use the tool?; How fun was the coloring process?; How easy was it to understand the concept/idea?; How happy were you with the final results?
All questions scored high (7 or above).

Participants also commented (paraphrased):
It was effortless; I like how neat the outputs are; I like how it makes further animation easy; Coloring small, narrow triangles was difficult (commented on by 3 users); A good tool for all kinds of users since it is easy to understand and use.

5.2 Feedback from Design Professionals

We also sought feedback from two professionals working in illustration and animation. They appreciated the simplicity of the tool and affirmed the importance of the task. Their specific comments

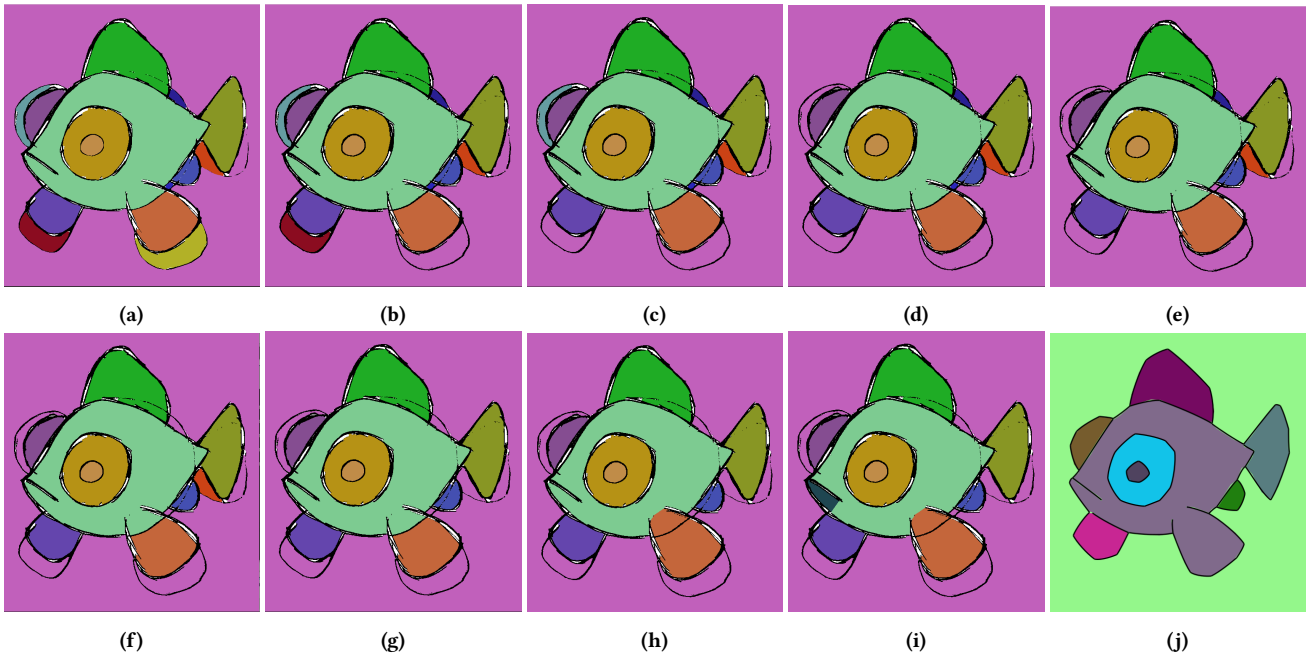

Figure 9: An interactive Color-by-Numbers session: Initial coloring (a); multiple Pick & drop steps (b)-(h); a Fill color step (i); and the vectorized result (j).

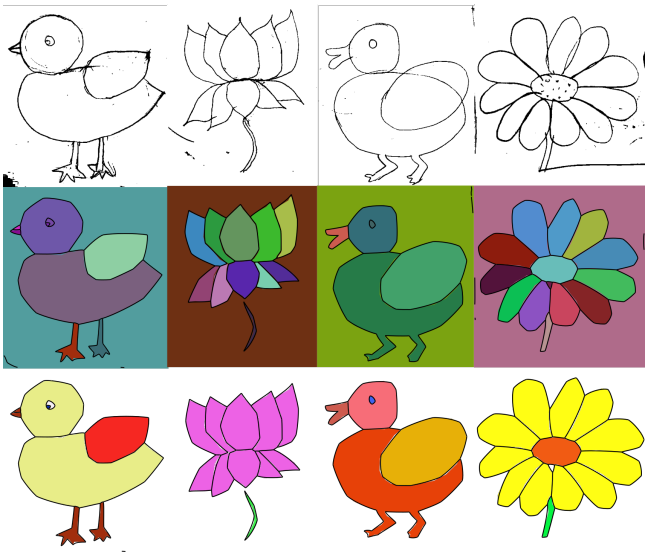

Figure 10: A 5-year old child's drawings vectorized.

(paraphrased) included:
The tool is easy to use; Initially, the clickable options were a little confusing, but after trying the tool, it became easy; The tool helps vectorize sketches across a range of artistic styles; Can see a lot of potential in this tool; A complete system can be created by incorporating animation features for the vector output; It can be a great tool for children since sketching on paper is common and will contain noise and unerased errors (Figure 10); the ability to work interchangeably with paper and digital media is useful even for a professional artist, especially when they want to quickly color and texture rough paper sketches.

5.3 Rough Sketch Benchmark

Fully automatic vectorization of rough sketches is challenging due varying degrees of sketch noise, construction lines, spurious strokes, texture and annotation (Figure 2). Yan et al. [32] present a diverse benchmark of nearly 101 raster sketches in 5 categories (art, logo, architecture, fashion, product), along with artist hand-created clean-up and structuring, as vector ground-truth. They also report that these hand-traced vector curves often lack precise curve connectivity and intersection needed to structure the sketch into regions, a desirable property that is inherent in our approach. Their baseline for interactive hand-created vectorization, was manual creation/editing of vector curves, by artists using Adobe Illustrator. These artists apparently took 30-90 mins. per sketch (communicated privately). In contrast, our interactive approach usually requires no more than a few minutes of interaction.

Figure 15 shows results of our approach producing vector output matching ground truth on 30 models spanning the 5 categories (all interactively processed in under 2 minutes). As shown by Yan et al. [32], and us in (Figure 2), fully automated approaches are unable to discern the user intent of sketch strokes and thus meet with mixed success when vectorizing rough raster sketches. Figure 16 shows models that proved challenging for our approach and were only partially successful despite more than a few minutes of user interaction (discussed under Limitations).

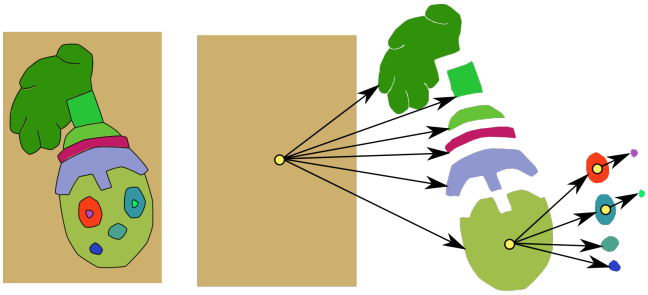

Figure 11: Structured layered tree vector output.

6 DISCUSSION

Our vector results can be further textured, and processed for downstream applications like animation (Figure 14).

Layer Structure:

While drawing rough sketches of layered objects, artists commonly draw complete shapes albeit lightly, prior to hiding occluded parts. Color-by-Numbers can recover such layers (Figure 12), where varying the grouping of regions allows entire objects to extracted as layers. Figure 12 shows multiple, layered vector shapes that can be extracted from the same input sketch, useful for hand-drawn 2.5D animation. Our vector output further supports a layered tree representation. As shown in Figure 11, the root of the tree is the background, and each node represents a region, recursively built as children of the region that completely contains it.

Strengths and Limitations:

As shown by Figure 15, 13, our approach readily removes undesired construction lines, annotation and sketch noise, by fill coloring both sides of unwanted strokes with the same color. Conceptually, the raster image is structured into compact Delaunay regions bounded by sketch pixels. Users thus interact with coarser primitives (Delaunay triangles) than pixels, to convey the intended vector structure of the sketch. Our $color\_flow$ based region growing, further simplifies sketch recoloring.

The biggest limitation of Color-by-Numbers is that it is unable to handle solid filled sketch regions (thinned to their medial skeleton), for example the eyeballs and duck-bowtie in Figure 16. In terms of interaction, user effort can increase linearly with the number of coherent regions in the sketch. Hatching and shading in sketches can thus, result in a large number of small regions that are tedious to recolor. Such similar but fragmented regions are better served by a lasso or scribble select and recolor interface. Another limitation is that clicking on small or sliver Delaunay triangles (some seed triangles can be large but thin) can be challenging. Vectorizing only desired sub-segments of strokes can also be difficult as our $color\_flow$ based approach may propagate color in a region beyond the desired sub-segment. Additional tools to aid such precise selection and recoloring of Delaunay regions can help address these shortcomings. Finally, our current curve fitting in Section 3.3 is simple and local, and does not enforce any global neatening constraints on the vector output.

7 CONCLUSION

Color-by-Numbers is a simple but powerful interactive interface for structuring and vectorizing rough sketches. Users can typically produce a vectorized drawing to match the intent of a rough raster sketch using a few simple clicks. Along with a set of vector curves, our approach also generates a set of layered regions, making it easily usable for further texturing and animation.

There are a number of avenues for future work. Better region recoloring tools and global neatness constraints can improve the quality of our vector output. We can also exploit the potential of stroke thickness prior to skeletal thinning (Figure 4), to extract solid fills, define stroke appearance, aid in region layering, or directly produce animations from over-sketched drawings [7]. Design sketching is an evolving process [6], and it would be also be exciting to explore a system, where curve detail could be interactively added (or removed) to a sketch during clean-up and vectorization.

8 ACKNOWLEDGEMENTS

We thank the anonymous reviewers and our study participants for their time and helpful feedback. We also thank Gowtham (for the animations), Tibor Stanko and Xuemiao Xu (for helping us in generating results), and Simo-Serra and JD Favreau (for making their code public).

REFERENCES

[1] Adobe Inc. [n.d.]. *Adobe Illustrator.* https://adobe.com/products/illustrator
[2] Pascal Barla, Joelle Thollot, and François X. Sillion. 2005. Geometric Clustering for Line Drawing Simplification. In *Eurographics Symposium on Rendering (2005)*, Kavita Bala and Philip Dutre (Eds.). The Eurographics Association. https://doi.org/10.2312/EGWR/EGSR05/183-192
[3] Mikhail Bessmeltsev and Justin Solomon. 2019. Vectorization of Line Drawings via Polyvector Fields. *ACM Trans. Graph.* 38, 1, Article 9 (Jan. 2019), 12 pages. https://doi.org/10.1145/3202661
[4] Jean-Daniel Boissonnat. 1984. Geometric Structures for Three-Dimensional Shape Representation. *ACM Trans. Graph.* 3, 4 (Oct. 1984), 266–286. https://doi.org/10.1145/357346.357349
[5] Boris Dalstein, Rémi Ronfard, and Michiel van de Panne. 2014. Vector Graphics Complexes. *ACM Trans. Graph.* 33, 4 (July 2014).
[6] K Eissen and R Steur. 2009. *Sketching: Drawing techniques for product designers* (11 ed.). BIS Publishers, AMsterdam, The Netherlands.
[7] Even Entem, Amal Dev Parakkat, Loïc Barthe, Ramanathan Muthuganapathy, and Marie-Paule Cani. 2019. Automatic structuring of organic shapes from a single drawing. *Computers Graphics* 81 (2019), 125 – 139. https://doi.org/10.1016/j.cag.2019.04.006
[8] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. 2016. Fidelity vs. Simplicity: A Global Approach to Line Drawing Vectorization. *ACM Trans. Graph.* 35, 4, Article 120 (July 2016), 10 pages. https://doi.org/10.1145/2897824.2925946
[9] Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Frédo Durand, and Adrien Bousseau. 2019. OpenSketch: A Richly-Annotated Dataset of Product Design Sketches. *ACM Trans. Graph.* 38, 6, Article 232 (Nov. 2019), 16 pages. https://doi.org/10.1145/3355089.3356533
[10] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. 1999. Teddy: A Sketching Interface for 3D Freeform Design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 409–416. https://doi.org/10.1145/311535.311602
[11] Takeo Igarashi, Tomer Moscovich, and John Hughes. 2005. Spatial Keyframing for Performance-driven Animation. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. 107–116.
[12] Peter Ilbery, Luke Kendall, Cyril Concolato, and Michael McCosker. 2013. Biharmonic Diffusion Curve Images from Boundary Elements. *ACM Trans. Graph.* 32, 6, Article 219 (Nov. 2013), 12 pages. https://doi.org/10.1145/2508363.2508426
[13] Inkscape Project. [n.d.]. *Inkscape.* https://inkscape.org
[14] Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. 2009. Automatic and Topology-Preserving Gradient Mesh Generation for Image Vectorization. *ACM Trans. Graph.* 28, 3, Article 85 (July 2009), 8 pages. https://doi.org/10.1145/1531326.1531391
[15] Chenxi Liu, Enrique Rosales, and Alla Sheffer. 2018. StrokeAggregator: Consolidating Raw Sketches into Artist-Intended Curve Drawings. *ACM Trans. Graph.*

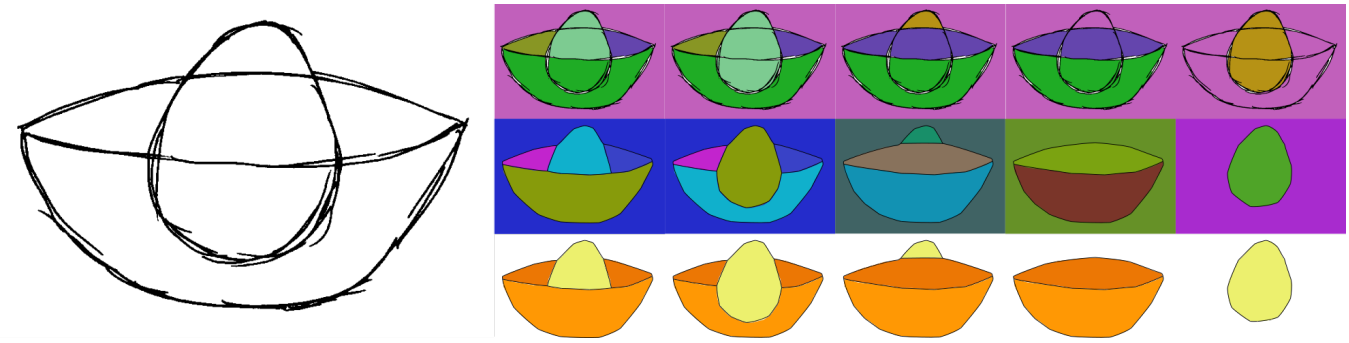

Figure 12: Input sketch (left); with multiple colorings (top right), their vector interpretation and rendering (mid-bottom right).

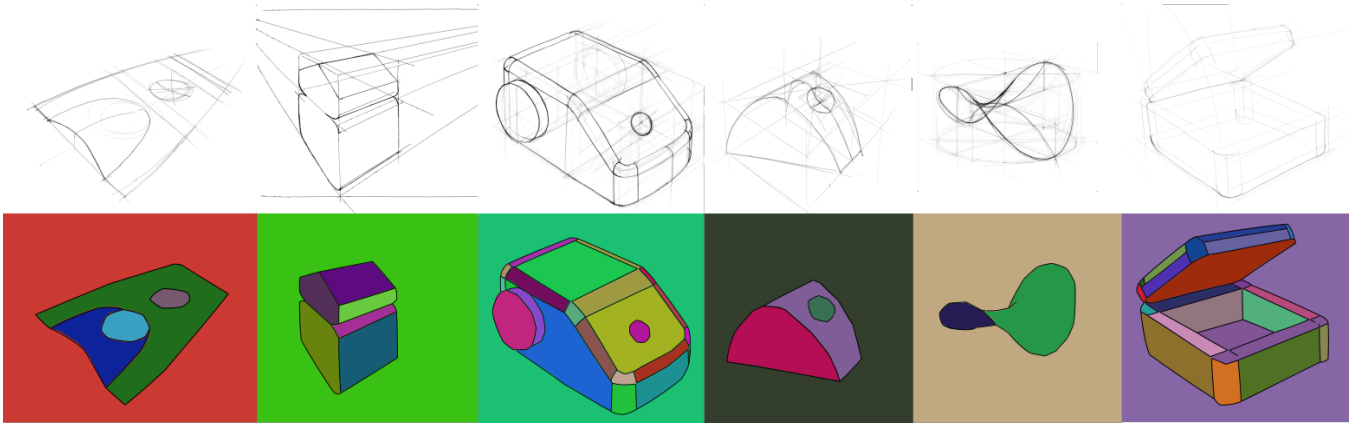

Figure 13: Design sketches (taken from [9]) and the results generated using our system.

37, 4, Article 97 (July 2018), 15 pages. https://doi.org/10.1145/3197517.3201314

[16] Xueting Liu, Tien-Tsin Wong, and Pheng-Ann Heng. 2015. Closure-Aware Sketch Simplification. *ACM Trans. Graph.* 34, 6, Article 168 (Oct. 2015), 10 pages. https://doi.org/10.1145/2816795.2818067

[17] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* 10, 4, Article 16 (Nov. 2010), 15 pages. https://doi.org/10.1145/1868358.1868363

[18] James McCrae and Karan Singh. 2008. Sketching Piecewise Clothoid Curves. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, Christine Alvarado and Marie-Paule Cani (Eds.). The Eurographics Association. https://doi.org/10.2312/SBM/SBM08/001-008

[19] Gioacchino Noris, Alexander Hornung, Robert W. Sumner, Maryann Simmons, and Markus Gross. 2013. Topology-Driven Vectorization of Clean Line Drawings. *ACM Trans. Graph.* 32, 1, Article 4 (Feb. 2013), 11 pages. https://doi.org/10.1145/2421636.2421640

[20] G. Noris, D. Sýkora, A. Shamir, S. Coros, B. Whited, M. Simmons, A. Hornung, M. Gross, and R. Sumner. 2012. Smart Scribbles for Sketch Segmentation. *Computer Graphics Forum* (2012). https://doi.org/10.1111/j.1467-8659.2012.03224.x

[21] G. Orbay and L. B. Kara. 2011. Beautification of Design Sketches Using Trainable Stroke Clustering and Curve Fitting. *IEEE Transactions on Visualization and Computer Graphics* 17, 5 (2011), 694–708.

[22] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion Curves: A Vector Representation for Smooth-Shaded Images. *ACM Trans. Graph.* 27, 3 (Aug. 2008), 1–8. https://doi.org/10.1145/1360612.1360691

[23] Amal Dev Parakkat, Uday Bondi Pundarikaksha, and Ramanathan Muthuganapathy. 2018. A Delaunay triangulation based approach for cleaning rough sketches. *Computers Graphics* 74 (2018), 171 – 181. https://doi.org/10.1016/j.cag.2018.05.011

[24] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. 2005. *Design Principles for Tools to Support Creative Thinking.* Technical Report.

[25] Kimiko Ryokai, Stefan Marti, and Hiroshi Ishii. 2004. I/O Brush: Drawing with Everyday Objects as Ink. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria) *(CHI '04)*. Association for Computing Machinery, New York, NY, USA, 303–310. https://doi.org/10.1145/985692.985731

[26] Maria Shugrina, Jingwan Lu, and Stephen Diverdi. 2017. Playful palette: an interactive parametric color mixer for artists. *ACM Trans. on Graphics (TOG)* 36, 4 (2017), 61.

[27] Maria Shugrina, Wenjia Zhang, Fanny Chevalier, Sanja Fidler, and Karan Singh. 2019. Color Builder: A Direct Manipulation Interface for Versatile Color Theme Authoring. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300686

[28] Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. 2016. Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup. *ACM Trans. Graph.* 35, 4, Article 121 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925972

[29] Tibor Stanko, Mikhail Bessmeltsev, David Bommes, and Adrien Bousseau. 2020. Integer-Grid Sketch Simplification and Vectorization. *Computer Graphics Forum* 39, 5 (2020), 149–161. https://doi.org/10.1111/cgf.14075 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14075

[30] Jun Xie, Holger Winnemöller, Wilmot Li, and Stephen Schiller. 2017. Interactive Vectorization *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 6695–6705. https://doi.org/10.1145/3025453.3025872

[31] X. Xu, M. Xie, P. Miao, W. Qu, W. Xiao, H. Zhang, X. Liu, and T. Wong. 2019. Perceptual-aware Sketch Simplification Based on Integrated VGG Layers. *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1.

[32] Chuan Yan, David Vanderhaeghe, and Yotam Gingold. 2020. A Benchmark for Rough Sketch Cleanup. *ACM Transactions on Graphics (TOG)* 39, 6 (2020).

[33] T. Y. Zhang and C. Y. Suen. 1984. A Fast Parallel Algorithm for Thinning Digital Patterns. *Commun. ACM* 27, 3 (March 1984), 236–239. https://doi.org/10.1145/357994.358023

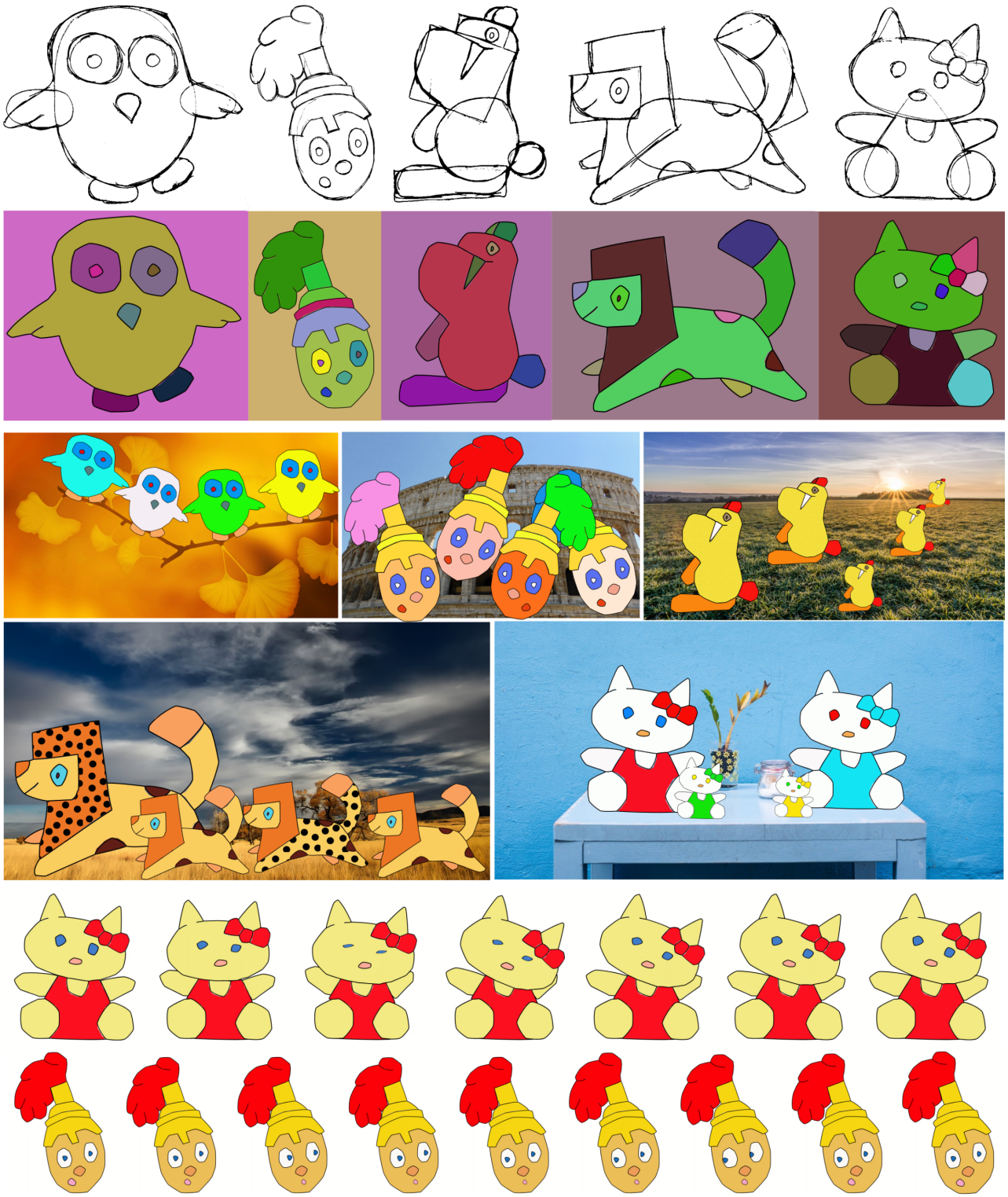

Figure 14: Rough sketches (top row); vectorized using our approach (second row); used to create textured scenes (next two rows); and key-frames from a vector animation (last two rows).

Figure 15: 30 benchmark sketches (raster input, interactively colored, vector output) across 5 categories [32] successfully vectorized using Color-by-Numbers. Sketch-Ids, top to bottom for the columns are (left) Ind_product_baseline_02, Ind_product_baseline_06, Ind_product_baseline_04, Art_freeform_GW_01, Ind_product_baseline_03, Ind_product_AS_18, Ind_product_baseline_12, Ind_product_baseline_10, Ind_fashion_RB_10, Ind_product_baseline_09, Art_freeform_baseline_02, (middle) Ind_architecture_baseline_02, Art_freeform_baseline_04, Art_freeform_baseline_05, Art_freeform_baseline_09, Ind_product_baseline_05, Ind_product_GW_09, Ind_product_GW_06, Ind_fashion_ML_10, Art_freeform_baseline_12, (right) Ind_product_baseline_02, Ind_architecture_baseline_03, Ind_architecture_baseline_05, Ind_architecture_baseline_06, Ind_product_GW_02, Art_logo_JST_01, Ind_product_baseline_09, Ind_architecture_baseline_01, Ind_fashion_ML_11, Art_freeform_baseline_18.

Figure 16: 10 benchmark sketches (raster input, interactively colored, vector output) across 5 categories [32] that were only partially successful due to shading and other raster sketch artifacts. Sketch-Ids, top to bottom for the columns are (left) Ind_fashion_HF_01, Ind_fashion_LB_01, Art_freeform_PB_06, (middle) Ind_fashion_RB_05, Ind_fashion_ML_05, Art_logo_BF_01, (right) Art_logo_VFS_16, Art_logo_JS_02, Art_logo_CA_01, Ind_architecture_JJ_03.