Image vectorization and editing via linear gradient layer decomposition

ZHENG-JUN DU, Qinghai University, China and BNRist, Department of CS&T, Tsinghua University, China
LIANG-FU KANG, BNRist, Department of CS&T, Tsinghua University, China
JIANCHAO TAN, Kuaishou Technology, China
YOTAM GINGOLD, George Mason University, USA
KUN XU*, BNRist, Department of CS&T, Tsinghua University, China

Fig. 1. Given the input image (1st column) and a segmentation mask (2nd column), we can decompose the image into several linear gradient layers to be saved in vector graphics (3rd column). Then we can perform recoloring (4th column) or object remove-insert-replace edits using these linear gradient layers in illustrator software (5th column). Image designed by Altagracia Art on Shutterstock.com.

A key advantage of vector graphics over raster graphics is their editability. For example, linear gradients define a spatially varying color fill with a few intuitive parameters, which are ubiquitously supported in standard vector graphics formats and libraries. By layering regions filled with linear gradients, complex appearances can be created. We propose an automatic method to convert a raster image into layered regions of linear gradients. Given an input raster image segmented into regions, our approach decomposes the resulting regions into opaque and semi-transparent linear gradient fills. Our approach is fully automatic (e.g., users do not identify a background as in previous approaches) and exhaustively considers all possible decompositions that satisfy perceptual cues. Experiments on a variety of images demonstrate that our method is robust and effective.

CCS Concepts: • Computing methodologies → Image manipulation; Image processing.

Additional Key Words and Phrases: images, gradient, layers, vectorization, RGB, color space, recoloring, compositing

*Kun Xu is the corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
0730-0301/2023/8-ART1 $15.00
https://doi.org/10.1145/3592128

1 INTRODUCTION

Image editing is a ubiquitous task in computer graphics. Raster images define each pixel’s color independently. Any edit is possible, yet every edit is tedious. For example, image scaling often introduces unexpected noise and is difficult to eliminate. In contrast, vector graphics image formats define images in a structured way, such as an arrangement of parametric strokes and regions filled with color-defining functions. For example, a linear gradient \( g(x, y) \) is defined by a 2D line and (potentially semi-transparent) linearly interpolated colors along the line. The color of any point \( x, y \) is defined as the color of the closest point on the line. Linear gradients are ubiquitously supported in 2D vector graphics standards and APIs (e.g., SVG, PostScript, CSS).

We propose an algorithm to decompose a segmented raster image into layered regions of linear gradients. Once decomposed, the linear gradient color-defining functions allow the image to be easily edited in a structured way. Our decomposition prefers semi-transparent upper layers, so complex effects like glossy highlights are preserved when editing the base color (Figure 1). Like other recent work on linear gradient decomposition [Favreau et al. 2017; Richardt et al. 2014], our approach also requires a segmentation of the input raster image into piecewise-continuous regions and does not consider the problem of converting the region boundaries into parametric curves. Unlike these approaches, our approach requires no manual interaction.
Our **contribution** is a fully automatic approach to gradient layer decomposition that finds globally optimal region groups and layers given a segmented input image. Expressed as a discrete optimization problem, the search space is intractably large. We drastically reduce the search space with perceptually-motivated constraints, resulting in a small space that can be exhaustively enumerated quickly.

## 2 RELATED WORK

### Image vectorization

Most image vectorization algorithms propose to represent images using various parametric representations, such as linear or quadratic gradients [Lecot and Levy 2006], diffusion curves and inverse diffusion curves [Orzan et al. 2008; Xie et al. 2014; Zhao et al. 2018], gradient meshes [Sun et al. 2007], subdivision surfaces [Liao et al. 2012] and so on. These methods provide specific global data structures with high complexities to represent the image content, rarely considering the image content’s semantic structure and inter-relationships between objects in the image. The editability enabled by these complicated parametric representations is limited. Several methods vectorize line drawings [Bessmeltsev and Solomon 2019; Favreau et al. 2016; Kim et al. 2018; Puhachov et al. 2021; Stanko et al. 2020] (see [Yan et al. 2020] for a recent survey), region boundaries [Alberto Dominici et al. 2020; Hoshyari et al. 2018], and textured materials [Lopez-Moreno et al. 2013; Song et al. 2015].

Recently, deep-learning-based image vectorization methods have been introduced, either via a supervised training approach using CNNs or RNNs [Huang et al. 2018; Reddy et al. 2021; Shen and Chen 2021] or unsupervised optimization with the help of a DNN [Sbai et al. 2020]. In our work, we propose to use a graph-based search to decompose the image contents into several editable layers whose colors and transparency are represented by linear gradient parameters.

### Layer decomposition

Decomposing the image into several editable layers containing meaningful content can enable efficient and intuitive editing. Commercial tools like Photoshop also manipulate images using layers as their data structure. Various layers extracted in [Aksoy et al. 2017, 2018; Du et al. 2021; Koyama and Goto 2018; Tan et al. 2018a, 2015, 2018b,c, 2016; Wang et al. 2019; Zhang et al. 2021, 2017] are in raster image format. These layers are either ordered RGB semi-transparent layers or unordered mixing weights maps, each of which corresponds to each palette color. Intrinsic layer decomposition [Meka et al. 2021; Shen et al. 2008] is suitable only for photographs and a particular kind of illumination-aware edit.

The most similar works to ours are [Richardt et al. 2014] and [Favreau et al. 2017]. The former work proposed to iteratively decompose the image region into vectorized layers (linear or radial gradient) by manually selecting the ordered regions. [Favreau et al. 2017] proposed a Monte Carlo Tree Search algorithm that can decompose and merge layers in a sampling manner, also with the help of both user hints for determining opaque layers and the X-junction hint [Singh and Huang 2003] for determining layer order. In contrast, we propose an efficient graph-based method to extract several linear gradient layers automatically, without any user intervention. We reduce the search space in a global manner using predefined rules, leading to a better chance of finding globally optimal solutions and more reasonable decomposition results.

## 3 BACKGROUND

Layers are indispensable components in digital image editing software to create images. Our paper focuses on vectorized images that are composed of one specific type of layer, named **linear gradient layers**. Below, we first review its concept, then describe how images can be composed with them.

### Linear gradient layers

Formally, a linear gradient layer is defined by a mask $L$ and several parameters. The color and opacity at a pixel position $p = (x, y)$ can be written as:

$$
C_{\text{RGBA}}(p) = \begin{cases} 
0 & \text{if } p \text{ is outside } L, \\
(p \cdot n) \cdot c_g + c_0 & \text{if } p \text{ is inside } L.
\end{cases}
$$

We use a 4-channel RGBA vector $C_{\text{RGBA}} = (C_{\text{RGB}}, C_A)$ to denote the RGB colors ($C_{\text{RGB}}$) and alpha-channel opacity ($C_A$) together. $n = (n_x, n_y)$ is the normalized gradient direction. $c_g$ and $c_0$ are both 4-channel color vectors. $c_g$ indicates the derivatives of colors and opacity along the gradient direction, and $c_0$ indicates color and opacity values at the origin. Inside its mask, both the color and opacity values change smoothly and linearly along the gradient direction $n$, while the values are zero outside the mask. In addition, we also restrict the 4-channel RGBA vector within the unit hypercube $[0, 1]^4$, to ensure valid RGB color and opacity values.

In summary, besides the mask $L$, the parameters of a linear gradient layer include the gradient direction $n$ and two 4-channel vectors $c_g$ and $c_0$. Since $n$ is a normalized direction, the total degrees of freedom of the parameters is 9. For simplicity, we refer to linear gradient layers as **layers**.

### Image compositing from linear gradient layers

Given a sequence of $n$ layers, the colors of image $I$ could be composited through iterative alpha blending from bottom to top:

$$
I_{k}\text{RGB} = C_{A}^{k}\cdot C_{\text{RGB}}^{k} + (1 - C_{A}^{k}) \cdot I_{k-1}\text{RGB},
$$

where $I_0\text{RGB}$ denotes the colors of the default opaque canvas, and $I_l = I_{l}\text{RGB}$ is the final composited colors. $C_{\text{RGB}}^{k}$ and $C_A^{k}$ denote the color and opacity values of the $k$-th layer, respectively. Note that the compositing is performed in a pixel-wise manner, but we omit the pixel position $p$ in the above equation for denotation simplicity.

Different orders of layers would result in different composition results. However, the order does not always matter. If two layers (i.e., their masks) do not overlap with each other, switching their order would not change the result.

## 4 OUR METHOD

### 4.1 Problem Formulation

The goal of our method is to solve the inverse problem of layer compositing described in Equation 2: finding compositing layers for a given input rasterized image. Specifically, two types of parameters need to be determined. One type is the discrete parameters: the number of layers, their masks, and stack order, which we refer to as the **layer configuration**. The other type is the continuous parameters:
the linear gradient direction and two 4-channel color vectors for each layer, which we refer to as the layer parameters.

Once all the layers are obtained, we could naturally reconstruct the original image and make edits to it by modifying the masks, color/opacity parameters, or adding/removing objects as shown in Fig. 1. However, the above problem is rather non-linear and very ill-posed. The number of layers, their masks, as well as layer parameters, are all unknown. To reduce the difficulty of the problem, we follow previous work [Favreau et al. 2017] and ask users to additionally provide a segmentation of the input image.

4.1.1 Input and outputs. Formally, given an input image $I$ and its segmentation which partitions the input image into a set of disjoint regions, i.e., $R = \{R_1, ..., R_m\}$, we seek a sequence of parameterized layers $L = \{L_1, ..., L_n\}$ in ascending order (i.e., $L_1$ is on the bottom while $L_n$ is on the top), which faithfully reconstruct the input image through alpha blending (Equation 2).

4.1.2 Assumption on layer masks. With the given segmented regions in hand, we impose an assumption that each layer (i.e., its mask) can only be composed of one or more adjacent regions. In other words, regions are the basic units forming layer masks.

An example is shown in Fig. 2, where the image is segmented into 6 regions. The bottom layer $L_1$ is composed of all 6 regions, i.e. $L_1 = \{R_1, R_2, R_3, R_4, R_5, R_6\}$, while the top layer $L_6$ is composed of 3 regions, i.e. $L_6 = \{R_4, R_5, R_6\}$. While regions are disjoint, layers are allowed to overlap with each other. A region can be covered by one or more layers.

Note that exchanging the order of two non-overlapping layers would not change the compositing result. As shown in Fig. 2 (b), layer configurations $\{L_1, L_2, L_3, L_4\}$ and $\{L_1, L_3, L_2, L_4\}$ are identical. In fact, only the topological order of layers matters.

4.1.3 Energy function. In order to find the best layer sequence, we define an energy function to assess its quality. The criteria of a good energy function would be considering both the fidelity of its reconstruction and the compactness of the layers. Specifically, our energy function is defined as the weighted sum of a reconstruction term $E_{\text{recon}}$, a gamut term $E_{\text{gamut}}$, and a compactness term $E_{\text{compact}}$:

$$E = w_r E_{\text{recon}} + w_g E_{\text{gamut}} + w_c E_{\text{compact}}.$$  \hfill (3)

We empirically set the weights to $w_r = 20$, $w_g = 10$, and $w_c = 0.02$ for all examples.

The reconstruction term $E_{\text{recon}}$ prefers the reconstructed image to be as close to the input image as possible. We follow Favreau et al. [2017] and define it as the average L2 difference between the reconstructed image $I$ and the input image $I^0$:

$$E_{\text{recon}} = \frac{1}{N} \sum_p \|\text{RGBA}(p) - \text{ori}_{\text{RGBA}}(p)\|^2,$$  \hfill (4)

where $p$ denotes a pixel, $N$ is the pixel count, and the summation enumerates over all pixels. Note that the channel format of the input image is allowed to be either 4-channel RGBA or 3-channel RGB. If it is the latter, we only compute the L2 difference on RGB colors.

We define a gamut term to penalize layers with color outside the RGB cube or large opacity (i.e., larger than 0.8). Its purpose is to generate layers with valid color and opacity values, and encourage more semi-transparent layers to achieve good editability. It is defined as:

$$E_{\text{gamut}} = \frac{1}{M} \sum_{i=1}^{n} \sum_{p \in L_i} \|C_i^{\text{RGB}}(p) - \text{clip}(0,0.8,C_i^{\text{A}}(p))\|^2.$$

where $M$ denotes the total number of enumerated pixels. Note that if a pixel is covered by multiple layers, it would be considered multiple times. In the equation, $C_i^{\text{RGB}}(p)$ and $C_i^{\text{A}}(p)$ denote the RGB color and opacity of the $i$-th layer at pixel $p$, respectively. $\text{clip}(0,0.8,C_i^{\text{A}}(p))$ clamps the opacity value within $[0,0.8]$.

The compactness term $E_{\text{compact}}$ encourages smaller-sized layers to be on top of larger-size layers. It is defined as:

$$E_{\text{compact}} = \sum_{1 \leq i < j \leq n, L_i \cap L_j \neq \emptyset} \mathbb{I}(|L_i| < |L_j|),$$  \hfill (6)

where $|L_i|$ and $|L_j|$ denote the pixel number in $L_i$ and $L_j$, and $\mathbb{I}()$ is an indicator function with value 1 if the condition is satisfied and value 0 otherwise. In this term, we enumerate all overlapping layer pairs and penalize whenever the lower layer ($L_i$) has a smaller size than the upper layer ($L_j$).

4.2 Method Pipeline

To compute the layers of a given rasterized image, we need to first determine the layer configuration and then estimate the layer parameters. We solve this challenging problem with two key observations. First, we find that a layer configuration can be derived from a novel data structure we call the region supporting tree (Section 4.3). Second, we find that the space of region supporting trees can be effectively reduced with some perceptually-motivated rules, which makes the search process very efficient.

Based on these observations, we design our method as follows:

- **Construction of region adjacency graph.** First, based on the region segmentation of the input image, we build a region adjacency graph. It is a directed graph where a node corresponds to a region and an edge corresponds to a supporting relationship between adjacent regions. The graph essentially defines the space of all region supporting trees. See Sec. 4.4 for details.

- **Enumeration of region supporting trees.** Second, we perform a depth-first-search on the region adjacency graph to find all valid region supporting trees. To speed up this search, we further
We first introduce three concepts used throughout the paper: layer supporting relationship, top layers, and region supporting relationships.

**Layer and region supporting relationships.** In the example shown in Fig. 2, the image contains 6 regions \( \{R_1, \ldots, R_6\} \). Fig. 2 (b) shows a layer configuration with 4 layers (from bottom to top):

\[
L_1 = \{R_1, R_2, \ldots, R_6\}, L_2 = \{R_2, R_4\}, L_3 = \{R_3, R_6\}, L_4 = \{R_4, R_5, R_6\}.
\]

**Layer supporting relationship.** We say that a lower layer \( L_i \) supports a higher layer \( L_j \) if they satisfy two conditions: 1) the two layers overlap; 2) one or more overlapping regions are not covered by any in-between layer \( L_k \) (i.e., \( i < k < j \)). The overlapping regions satisfying condition 2) are referred to as the supporting region(s) from layer \( L_i \) to \( L_j \). For example, \( L_1 \) supports \( L_2 \) through regions \( R_2 \) and \( R_4 \), \( L_2 \) supports \( L_4 \) through region \( R_4 \), and \( L_3 \) supports \( L_4 \) through region \( R_6 \).

**Top layer.** One region might be covered by multiple layers. We denote the highest covering layer of a region as its top layer. For example, \( R_4 \) is covered by three layers \( L_1, L_2, \) and \( L_4 \), so its top layer is the highest layer \( L_4 \).

**Region supporting relationship.** Given two regions \( R_i \) and \( R_j \), suppose their top layers are \( L_i \) and \( L_j \), respectively. We say that \( R_i \) supports \( R_j \) (denoted as \( R_i \rightarrow R_j \)) if 1) they are adjacent regions in the segmentation; 2) layer \( L_i \) supports layer \( L_j \); and 3) region \( R_j \) is one of the supporting regions from \( L_i \) to \( L_j \). For example, \( R_1 \) supports \( R_2 \), and \( R_3 \) supports \( R_6 \). However, \( R_1 \) does not support \( R_4 \). While \( R_1 \)'s top layer \( (L_1) \) supports \( R_3 \)'s top layer \( (L_4) \), \( R_4 \) does not belong to their supporting regions. This is because \( R_4 \) is covered by an in-between layer, \( L_2 \), in addition to \( L_1 \) and \( L_4 \).

There are two properties for region supporting relationships: 1) Initially, a region can support multiple regions at the same time, but it can only be supported by at most one region. 2) We define \( R_i \rightarrow R_j \rightarrow \cdots \rightarrow R_n \) as a supporting path starting from any region \( R_i \). We claim that this path does not include cycles, because a supported region’s top layer is always higher than that of the supporting region. If there is a loop, a higher layer will in turn support a lower one, which is impossible.

**Region supporting trees.** From the above concepts and properties, we find that the region supporting relationship of a given layer configuration can be formulated as a DAG (directed acyclic graph), such as the one shown in Fig. 2 (c). In the DAG, each region corresponds to a node whose in-degree is \( \leq 1 \). A directed edge denotes a supporting relationship between two regions. Furthermore, for simplicity, we add a virtual node \( R_0 \) representing the canvas and also add virtual directed edges from \( R_0 \) to those regions without any support. Thanks to the virtual node \( R_0 \) serving as the unique root,
the DAG thus becomes a directed tree, which we call the region supporting tree.

The region supporting tree naturally inherits all the properties of a tree, such as that it is a connected graph and there are no cycles in it. Although a layer configuration corresponds to a unique region supporting tree, conversely, a region supporting tree may correspond to multiple layer configurations. Such an example is shown in Fig. 2 (d). The difference between this layer configuration and the one in Fig. 2 (b) lies in whether the top layers of 𝑅4, 𝑅5, and 𝑅6 should be different layers or merged into one layer. We discuss this further in Sec. 4.6.

![Region adjacency graph simplification](image)

Fig. 4. Region adjacency graph simplification. (a) The input region segmentation. (b) The initial region adjacency graph. (c) Edges removed with the size rule. (d) Edges removed with the surrounding rule. (e) Edges removed with the adjacent strength rule.

4.4 Construction of Region Adjacency Graph

Given an input image and its region segmentation, the first step of our method is to construct a region adjacency graph. We treat all regions as nodes and additionally add a virtual region node 𝑅0 to indicate the image canvas. A directed edge from region 𝑅𝑖 to region 𝑅𝑗 indicates that 𝑅𝑖 is allowed to support 𝑅𝑗. Initially, for all pairs of adjacent regions, we add bidirectional edges between them indicating that they are allowed to support each other. For the virtual region 𝑅0, we add directed edges from it to all other regions.

The region adjacency graph essentially defines the space of all possible region supporting trees, i.e., a spanning tree of the region adjacency graph is exactly a region supporting tree. However, directly enumerating over all spanning trees of the above initial densely-connected graph would be rather expensive. Motivated by figure-ground and part perception of shapes [Wagemans et al. 2012], we propose three rules to remove those very unlikely supporting edges from the graph to reduce the search space.

- **Surrounding rule.** Consider a region 𝑅1 which has holes in it. For any region 𝑅2 in its hole, we do not allow 𝑅2 to support 𝑅1, as this would violate figure-ground perception. Furthermore, we do not allow 𝑅2 to be supported by the virtual region 𝑅0. This is because 𝑅2 should be supported by its enclosing region (i.e., 𝑅1) or the descendant nodes of 𝑅1.

- **Size rule.** Consider two adjacent regions 𝑅1 and 𝑅2. If the size of 𝑅2 is much larger than the size of 𝑅1, i.e., \[ \frac{|𝑅2|}{|𝑅1|} > 2 \], we do not allow the small-sized region 𝑅1 to support the large-sized region 𝑅2. Violation of this rule will probably lead to larger layers covering smaller layers, which we would like to avoid, as this violates figure-ground perception.

- **Adjacent strength rule.** Consider a region 𝑅1 which is adjacent to multiple other regions. Let 𝑅2 be one of its adjacent regions and define the adjacency strength \( S(𝑅2, 𝑅1) \) of 𝑅2 to 𝑅1 as the length of their shared boundary. Let the largest adjacency strength from 𝑅1 to any of its adjacent regions 𝑅𝑖 be \( S_{max} = \max S(𝑅𝑖, 𝑅1) \). If the adjacency strength of 𝑅2 is extremely small, i.e., \( \frac{S(𝑅2, 𝑅1)}{S_{max}} < 0.4 \), we do not allow 𝑅2 to support 𝑅1. This rule removes edges between adjacent regions that are very weakly linked to each other. This is supported by the skeletal model of shape part perception.

After applying the above rules, the region adjacency graph is heavily simplified. Fig. 4 provides an example of the simplification process.

4.5 Enumeration of Region Supporting Trees

After obtaining the simplified region adjacency graph, we perform a depth-first search to find all spanning trees. Recall that each spanning tree is exactly a region supporting tree. The depth-first search starts from the virtual canvas node, which is always considered the root of the tree, and recursively grows the tree until all nodes have been visited. For acceleration, we further introduce two pruning rules to efficiently eliminate unnecessary search branches.

The first pruning rule is to restrict the tree’s depth. A region supporting tree with a large depth would lead to a relatively large number of layers and unnecessarily complex layer ordering. We set the maximum tree depth to 8.

The junction assumption is a powerful cue for early scene understanding [Waltz 1972] and recent transparency detection [Metelli 1974; Sayim and Cavanagh 2011]. The second pruning rule uses the X-junction assumption. An X-junction denotes 4 regions with a 2 × 2 grid-like layout, as shown in Fig. 5 (a). We assume that X-junctions appear when a higher semi-transparent layer covers two lower layers. According to the X-junction assumption, there are only 4 allowed configurations of supporting relationships (\( \rightarrow \)) denotes supporting):

1. \( 𝑅d \rightarrow 𝑅a \) and 𝑅c \( \rightarrow 𝑅b \); 𝑅a, 𝑅b, 𝑅c, 𝑅d are supported regions;
2. \( 𝑅a \rightarrow 𝑅b \) and 𝑅d \( \rightarrow 𝑅c \); 𝑅b, 𝑅c, 𝑅d are supported regions;
3. \( 𝑅b \rightarrow 𝑅c \) and 𝑅a \( \rightarrow 𝑅d \); 𝑅c, 𝑅d are supported regions;
4. 𝑅c \( \rightarrow 𝑅d \) and 𝑅b \( \rightarrow 𝑅a \); 𝑅d, 𝑅a are supported regions;

as shown in Fig. 5 (b-e), respectively. If we find an X-junction that violates the above assumption (e.g., when 𝑅a \( \rightarrow 𝑅b \) and 𝑅c \( \rightarrow 𝑅d \)), we prune the search branch. We adopt the same approach as [Favreau et al. 2017] to extract X-junctions, i.e., detecting the size-4 cliques in the region adjacency graph. Of course, some 4-connected regions may not meet the X-junction assumption in practice. Users can enable or disable this assumption according to the actual situation.

We employ the algorithm introduced by [Gabow and Myers 1978] to enumerate all spanning trees. The time complexity is \( O(V + E + EN) \), where \( V, E, N \) represent the number of vertices, edges, and spanning trees, respectively. The above two pruning rules greatly reduce the number of edges \( E \) and therefore reduce the number of spanning trees \( N \) significantly.
4.6 Layer Merging

For each obtained region supporting tree from the previous step, we first convert it into a layer representation and then try to merge some of them according to the X-junction assumption if possible.

Recall that the X-junction assumption supposes there is a higher semi-transparent layer on top, hence, we know that the two supported regions should be covered by the same top layer. For example, considering the configuration in Fig. 5 (b), which is $R_d \rightarrow R_a$ and $R_c \rightarrow R_b$, we trivially know that $R_a$ and $R_b$ have the same top layer. We refer to this as the same top layer property.

Suppose we have obtained the region supporting tree shown in Fig. 5 (g) from the input segmentation in Fig. 5 (f). Simply imagining that each region corresponds to a layer that covers it and all of its descendant regions (nodes), we could obtain an initial layer configuration with 6 layers:

$$L_1 = \{R_1, \ldots, R_6\}, L_2 = \{R_2\}, L_3 = \{R_3, R_4\}, L_4 = \{R_5\}, L_5 = \{R_6\}, L_6 = \{R_3\}.$$  

However, it is easy to find that this layer configuration is not optimal, e.g., $L_2$, $L_5$, and $L_6$ are obviously the same layer and should be merged. We will use the same top layer property of X-junction to help with layer merging.

This example has 3 X-junctions. Let’s first look at one X-junction $\{R_1, R_6, R_3, R_4\}$ (Fig. 5 (h)). According to the edge connections between them, we could derive that $R_3$ and $R_6$ are supported regions, hence their top layers (i.e., $L_3$ and $L_6$) should be merged. We refer to the merged layer containing $R_3$ and $R_6$ as $L_{6.5}$. This leads to a layer configuration of 5 layers:

$$L_1 = \{R_1, \ldots, R_6\}, L_2 = \{R_2\}, L_3 = \{R_3, R_4\}, L_4 = \{R_5\}, L_{6.5} = \{R_3, R_6\}.$$  

Then, let’s look at another X-junction $\{R_1, R_6, R_3, R_5\}$ (Fig. 5 (i)). Since $R_6$ and $R_3$ are covered by the same top layer (i.e., $L_{6.5}$), they are supported regions, hence, there should be a supporting edge from $R_2$ to $R_5$, which indicates that $L_4$ ($R_5$’s top layer) supports $L_{6.5}$ ($R_3$’s top layer). Since $L_{6.5}$ is already supported by $L_3$, we could merge $L_3$ and $L_4$ into a new layer $L_{3.4}$, which reduces to 4 layers overall:

$$L_1 = \{R_1, \ldots, R_6\}, L_2 = \{R_2\}, L_{3.4} = \{R_3, R_4, R_5\}, L_{6.5} = \{R_3, R_6\}.$$  

Finally, let’s look at the third X-junction $\{R_1, R_2, R_3, R_5\}$ (Fig. 5 (j)). We know that $R_2$ and $R_3$ are supported regions, so their top layers (i.e., $L_2$ and $L_{6.5}$) should be merged, we call the merged new layer $L_{6.5.2}$. Which finally results in a 3-layer configuration (bottom to top):

$$L_1 = \{R_1, \ldots, R_6\}, L_{3.4} = \{R_3, R_4, R_5\}, L_{6.5.2} = \{R_2, R_3, R_6\}.$$  

For layer merging with X-junctions, the time complexity is $O(KV + K\log(V))$, where $K$ is the number of X-junctions. It should be noted that the ordering in which X-junctions are processed will not affect the results. This is because an X-junction can be viewed as a constraint applied to the layer configurations. Multiple X-junctions can be viewed as an unordered set of constraints.

4.7 Layer Parameter Optimization

Once we have obtained a layer configuration from the previous step through merging, the next step is to solve for continuous layer parameters, including the gradient direction and two 4-channel color vectors of each layer. This is done by a direct non-linear optimization minimizing the energy function defined in Eq. 3. Specifically, we employ the L-BFGS-B algorithm [Liu and Nocedal 1989; Nocedal 1980] via the NLopt library [Johnson 2011] with automatically computed gradients via the autodiff library [Leal et al. 2018]. We randomly initialize all layer parameters. The optimization stops if the number of iterations reaches a predefined threshold (i.e., 1000) or the error does not change (i.e., $< 1.0e^{-3}$). Although the loss function is not convex, the optimization usually converges quickly.

Finally, when all these candidate layer configurations along with layer parameters are obtained, we select the one with minimal loss (Eq. 3) to generate the resulting vector graph with the Potrace library [Selinger 2003]. We are also aware that layers with different
stacking orders could also well reproduce the input, users can output the top k vectorization results for further selection if required.

5 EXPERIMENTS

We perform all experiments on a PC with an Intel i9-9900K 3.6 GHz CPU with 16 cores and 16 GB RAM. Our method is implemented in C++. We optimize for the layer parameters of each region supporting tree in parallel using multithreading. The rest of our algorithm is single-threaded.

5.1 Results

We generated 47 layer decomposition results. We show 17 of them in Figs. 1 and 6. More results can be found in the supplementary material.

We have consistently achieved high-quality decomposition results in all examples. Notice how the highlight regions in Teacup, Cone, Purple-circle and Telephone (Fig.6) are all successfully extracted as transparent layers. In examples Can (Fig. 1), Teapot, and Synt, TV and Bottle (Fig. 6), all of which have X-junctions, our method perfectly recovers the layer stack and achieves reconstruction results with high accuracy.

We also tested input images with segmentations automatically generated by the MeanShift algorithm [Comaniciu and Meer 2002; Demirović 2019]. Three such examples are provided in Fig. 7. In general, the decomposed layers reconstruct the input images rather well. Since automatic segmentation algorithms tend to generate more fragmented regions compared to manual segmentations, our method will produce more layers.

Timing and statistics for all examples shown in the paper are provided in Table 1. From the table, we see that the layer parameter optimization step is the bottleneck. In general, the processing time is proportional to the number of region supporting trees. Since only a few region supporting trees are retained after applying our pruning rules, the process is usually very fast.

While our method automatically decomposes the input image together with its segmentation into layers, to improve flexibility, we also allow users to manually specify opaque layers, especially for X-junctions that do not meet the transparency assumption. See the supplementary material for examples.

5.2 Parameter Evaluation

In this subsection, we evaluate the weighting parameters in our energy function (Eq. 3), including the weight for the reconstruction term \(w_r\), the weight for the gamut term \(w_g\), and the weight for the compactness term \(w_c\). The parameters are set as \(w_r = 20\), \(w_g = 10\), and \(w_c = 0.02\) by default. When evaluating one parameter, the values of the others are fixed to the default.

The decomposition results of the two tested examples are shown in Fig. 8. For \(w_r\), larger values lead to more accurate reconstructions but might result in more layers, i.e., small value (\(w_r = 1\)) leads to incorrect shading in the reconstructed image of Teapot, while a large value (\(w_r = 200\)) generates more complex layers for TV. For \(w_g\), a small value usually leads to layers with invalid colors out of the gamut, i.e., setting \(w_g = 0.1\) generates undesirable layer colors in both Teapot and TV. Setting \(w_g\) to a relatively larger value (i.e., \(w_g = 20 – 100\)) leads to acceptable results. For \(w_c\), we find that omitting the compactness term (i.e., setting \(w_c = 0\)) leads to larger layers on top of smaller layers, which is undesirable.

Overall, we find that our default parameter setting is generally very robust. We have used the same set of parameters for all 47 examples and consistently achieved high-quality decomposition results.

5.3 Ablation Study

We performed an ablation study to evaluate the effectiveness of the three rules in region adjacency graph simplification, including the surrounding rule, the size rule, and the adjacent strength rule. We measure how the number of edges, the number of region supporting trees, and the energy function (Eq. 3) change when disabling each rule. Statistics are given in Table 2.

The results demonstrate that these three rules all play important roles in region adjacency graph simplification. With these rules enabled, we greatly reduce the number of region supporting trees, and hence achieve a very large acceleration while maintaining a relatively low energy loss.

For example, in the Truck example (Fig. 6 (c)), there are several wheel regions which have holes in them. By enabling the surrounding rule (i.e., forcing hole-regions to be supported by their enclosing regions), we reduce the number of trees from 672 to 112. In the Teapot (Fig. 6 (b)) example, many adjacent regions have significant size differences. By enabling the size rule (i.e., forbidding smaller regions to support larger regions if their size ratio is larger than 2), we significantly reduce the number of trees from 80,689 to 960. In the Can (Fig. 1) example, many regions have multiple neighbors and share short boundaries with its neighboring regions. By enabling the adjacency strength rule, we successfully reduce the number of trees from more than 2304 to 64. Notice that in all these examples, the result quality (i.e., the energy loss \(E\)) is almost unchanged after enabling the rules.

5.4 Comparisons

The most closely related works to ours are by [Richardt et al. 2014] and Photo2ClipArt [Favreau et al. 2017]. The former completely relies on user interaction and does not provide source code or an executable, so we compared our method with the latter using 32 examples. The Photo2ClipArt results were generated by a Windows executable provided by its authors.

Photo2ClipArt adopts Monte Carlo Tree Search (MCTS) to explore the space of layer configurations. While MCTS is powerful in selecting the most promising search branches, it may become stuck in local minima due to its greedy behavior. Our method reduces the search space with perceptually-motivated rules and then enumerates all possible layer configurations to find the best result. It is more likely to find the optimal solution.

5.4.1 Visual comparisons. 6 of the 32 examples are shown in Fig. 9. The rest can be found in the supplementary material. As shown in the Battery example, our method successfully recovers the layer order at each X-junction, and the layers do not contain holes. In contrast, Photo2ClipArt fails to obtain complete semi-transparent
Fig. 6. Image decomposition results. In each example, we provide the input image and its segmentation on the left, the reconstructed images, and all decomposed layers on the right. © George Dolgikh on Shutterstock.com, macrovector on Freepik, Freepik, Zheng-Jun Du, Freepik, brgfx on Freepik, bijutoha on Pixabay, macrovector on Freepik, starline on Freepik, Alex Staroseltsev on Shutterstock.com, Ite, Chino, Sauey, Meicilin, bajinda on Shutterstock.com.
Image vectorization and editing via linear gradient layer decomposition

Fig. 7. Image decomposition results for automatically segmented images. For each example, we provide the input image, the region segmentation generated by the MeanShift algorithm [Comaniciu and Meer 2002; Demirović 2019], the reconstructed image, and all decomposed layers.

Table 1. Timing and statistics. For each example, we provide the region count, the number of region supporting trees, the layer count of the resulting vector graphic, and the processing time of region adjacency graph construction, region supporting tree enumeration, layer merging, and layer parameter optimization.

<table>
<thead>
<tr>
<th>Examples</th>
<th>#region</th>
<th>#tree</th>
<th>#layer</th>
<th>adj. graph construct.</th>
<th>tree enum.</th>
<th>layer merging</th>
<th>layer param opt.</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can (Fig.1)</td>
<td>25</td>
<td>64</td>
<td>20</td>
<td>0.1</td>
<td>0.5</td>
<td>0.1</td>
<td>65.8</td>
<td>66.5</td>
</tr>
<tr>
<td>Teacup (Fig.6 (a))</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>2.3</td>
<td>2.6</td>
</tr>
<tr>
<td>Teapot (Fig.6 (b))</td>
<td>34</td>
<td>960</td>
<td>30</td>
<td>0.1</td>
<td>45.6</td>
<td>0.2</td>
<td>849</td>
<td>894.9</td>
</tr>
<tr>
<td>Truck (Fig.6 (c))</td>
<td>45</td>
<td>112</td>
<td>43</td>
<td>0.1</td>
<td>0.9</td>
<td>0.1</td>
<td>204.3</td>
<td>205.4</td>
</tr>
<tr>
<td>Syn1 (Fig.6 (d))</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>3.9</td>
<td>4.2</td>
</tr>
<tr>
<td>Cone (Fig.6 (e))</td>
<td>13</td>
<td>10</td>
<td>9</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.4</td>
</tr>
<tr>
<td>Hammer (Fig.6 (f))</td>
<td>19</td>
<td>24</td>
<td>18</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>40.2</td>
<td>40.5</td>
</tr>
<tr>
<td>Lamp (Fig.6 (g))</td>
<td>20</td>
<td>56</td>
<td>20</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>32.8</td>
<td>33.1</td>
</tr>
<tr>
<td>Shoe (Fig.6 (h))</td>
<td>12</td>
<td>46</td>
<td>10</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>11.8</td>
<td>12.1</td>
</tr>
<tr>
<td>Purple-circle (Fig.6 (i))</td>
<td>12</td>
<td>2</td>
<td>11</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>4.9</td>
<td>4.7</td>
</tr>
<tr>
<td>Battery1 (Fig.6 (j))</td>
<td>20</td>
<td>7</td>
<td>20</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>24.4</td>
<td>24.7</td>
</tr>
<tr>
<td>Horn (Fig.6 (k))</td>
<td>22</td>
<td>24</td>
<td>22</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>28.5</td>
<td>28.8</td>
</tr>
<tr>
<td>Telephone (Fig.6 (m))</td>
<td>53</td>
<td>8</td>
<td>52</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>99.4</td>
<td>99.8</td>
</tr>
<tr>
<td>TV (Fig.6 (n))</td>
<td>28</td>
<td>8</td>
<td>27</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>17.9</td>
<td>18.3</td>
</tr>
<tr>
<td>Bottle (Fig.6 (o))</td>
<td>24</td>
<td>60</td>
<td>22</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>60.8</td>
<td>61.2</td>
</tr>
<tr>
<td>Tomato (Fig.6 (p))</td>
<td>16</td>
<td>23</td>
<td>16</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>11.2</td>
<td>11.5</td>
</tr>
<tr>
<td>Battery2 (Fig.9 (b))</td>
<td>18</td>
<td>92</td>
<td>14</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>48.7</td>
<td>49.0</td>
</tr>
<tr>
<td>Coffee (Fig.9 (d))</td>
<td>19</td>
<td>96</td>
<td>13</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>48.7</td>
<td>49.0</td>
</tr>
<tr>
<td>Syn2 (Fig.9 (f))</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>3.2</td>
<td>3.5</td>
</tr>
<tr>
<td>Tiger (Fig.9 (h))</td>
<td>32</td>
<td>16</td>
<td>28</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>23.6</td>
<td>23.9</td>
</tr>
<tr>
<td>Soda (Fig.9 (j))</td>
<td>26</td>
<td>8</td>
<td>23</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>31.9</td>
<td>32.2</td>
</tr>
<tr>
<td>Sound (Fig.9 (l))</td>
<td>32</td>
<td>256</td>
<td>32</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>931.8</td>
<td>932.2</td>
</tr>
</tbody>
</table>
layers at X-junctions and generates some unnatural layers (highlighted in the figure), such as the round layer and the rectangular layer on the right, both of which contain holes. In the Coffee example, Photo2ClipArt generates two broken curves on the coffee cup, while our method produces complete ones. In the Syn2 and Tiger examples, Photo2ClipArt generates incomplete layers with undesirable holes, while our method generates more complete layers. Similarly, in the Soda and Sound examples, our method generates more semi-transparent and complete layers than Photo2ClipArt.

5.4.2 Reconstruction errors. For the above 6 examples, we also computed the reconstruction error (Eq. 4) of the results generated by Photo2ClipArt and our method, respectively. Fig. 10 visualizes the difference between the input image and the reconstructed image generated by Photo2ClipArt (left) and our method (right), respectively. The reconstruction errors are given inside the figure. In general, the reconstruction quality of both methods is comparable.

5.4.3 Perceptual study. To further assess the effectiveness of our method, we conducted a perceptual study with 32 participants (aged 19 to 32). Each participant evaluated the 32 example images. Participants were shown the input images and two anonymous vectorization results in random order, one generated by Photo2ClipArt and one by our method. The visualization of vectorization results showed all semi-transparent layers and allowed participants to interactively choose subsets of the layers to composite. For each example, participants were asked three questions: Q1 (reconstruction quality): Which result do you think is closer to the input image? Q2 (shape consistency): Which decomposition better reflects the parts of the input shape? Q3 (editing convenience): Which decomposition is more...
### Table 2. Ablation study.

For each example, from left to right, we provide region count, the edge count in the initial adjacency graph, the final edge count, the number of region supporting trees, and the energy loss $E$ (Eq. 3) when the size rule, surrounding rule, or adjacent strength rule are disabled, respectively.

<table>
<thead>
<tr>
<th>Examples</th>
<th>#region</th>
<th>#init_edge</th>
<th>#edge</th>
<th>#tree</th>
<th>$E$ with all rules enabled</th>
<th>#edge</th>
<th>#tree</th>
<th>$E$ w/o surrounding rule</th>
<th>#edge</th>
<th>#tree</th>
<th>$E$ w/o size rule</th>
<th>#edge</th>
<th>#tree</th>
<th>$E$ w/o adj. strength rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can (Fig.1)</td>
<td>25</td>
<td>109</td>
<td>39</td>
<td>64</td>
<td>1.339</td>
<td>41</td>
<td>64</td>
<td>1.339</td>
<td>69</td>
<td>120</td>
<td>1.339</td>
<td>50</td>
<td>2304</td>
<td>1.339</td>
</tr>
<tr>
<td>Teacup (Fig.6 (a))</td>
<td>9</td>
<td>35</td>
<td>14</td>
<td>10</td>
<td>1.855</td>
<td>14</td>
<td>10</td>
<td>1.855</td>
<td>17</td>
<td>18</td>
<td>1.853</td>
<td>19</td>
<td>194</td>
<td>1.853</td>
</tr>
<tr>
<td>Teapot (Fig.6 (b))</td>
<td>34</td>
<td>142</td>
<td>53</td>
<td>960</td>
<td>1.600</td>
<td>54</td>
<td>960</td>
<td>1.600</td>
<td>75</td>
<td>80,689</td>
<td>1.600</td>
<td>71</td>
<td>60,234</td>
<td>1.597</td>
</tr>
<tr>
<td>Truck (Fig.6 (c))</td>
<td>45</td>
<td>171</td>
<td>58</td>
<td>112</td>
<td>2.490</td>
<td>63</td>
<td>672</td>
<td>2.492</td>
<td>88</td>
<td>1,152</td>
<td>2.491</td>
<td>82</td>
<td>10,814</td>
<td>2.488</td>
</tr>
<tr>
<td>Syn1 (Fig.6 (d))</td>
<td>10</td>
<td>38</td>
<td>16</td>
<td>8</td>
<td>0.350</td>
<td>17</td>
<td>8</td>
<td>0.340</td>
<td>27</td>
<td>16</td>
<td>0.346</td>
<td>17</td>
<td>8</td>
<td>0.350</td>
</tr>
<tr>
<td>Cone (Fig.6 (e))</td>
<td>13</td>
<td>47</td>
<td>33</td>
<td>10</td>
<td>0.570</td>
<td>33</td>
<td>10</td>
<td>0.570</td>
<td>43</td>
<td>147</td>
<td>0.548</td>
<td>34</td>
<td>10</td>
<td>0.570</td>
</tr>
<tr>
<td>Hammer (Fig.6 (f))</td>
<td>19</td>
<td>72</td>
<td>26</td>
<td>6</td>
<td>1.245</td>
<td>31</td>
<td>12</td>
<td>1.245</td>
<td>37</td>
<td>12</td>
<td>1.244</td>
<td>36</td>
<td>200</td>
<td>1.241</td>
</tr>
<tr>
<td>Lamp (Fig.6 (g))</td>
<td>20</td>
<td>58</td>
<td>27</td>
<td>56</td>
<td>3.199</td>
<td>28</td>
<td>74</td>
<td>3.200</td>
<td>34</td>
<td>170</td>
<td>3.198</td>
<td>30</td>
<td>249</td>
<td>3.198</td>
</tr>
<tr>
<td>Shoe (Fig.6 (h))</td>
<td>12</td>
<td>50</td>
<td>21</td>
<td>46</td>
<td>2.179</td>
<td>21</td>
<td>46</td>
<td>2.179</td>
<td>33</td>
<td>120</td>
<td>2.183</td>
<td>27</td>
<td>182</td>
<td>2.178</td>
</tr>
<tr>
<td>Battery (Fig.9 (b))</td>
<td>18</td>
<td>68</td>
<td>29</td>
<td>92</td>
<td>0.929</td>
<td>29</td>
<td>92</td>
<td>0.929</td>
<td>50</td>
<td>328</td>
<td>0.925</td>
<td>32</td>
<td>150</td>
<td>0.927</td>
</tr>
<tr>
<td>Coffee (Fig.9 (d))</td>
<td>19</td>
<td>75</td>
<td>33</td>
<td>96</td>
<td>1.376</td>
<td>35</td>
<td>114</td>
<td>1.376</td>
<td>53</td>
<td>96</td>
<td>1.372</td>
<td>37</td>
<td>432</td>
<td>1.375</td>
</tr>
<tr>
<td>Syn2 (Fig.9 (f))</td>
<td>9</td>
<td>37</td>
<td>12</td>
<td>8</td>
<td>0.350</td>
<td>12</td>
<td>8</td>
<td>0.350</td>
<td>23</td>
<td>8</td>
<td>0.350</td>
<td>15</td>
<td>8</td>
<td>0.350</td>
</tr>
<tr>
<td>Tiger (Fig.9 (g))</td>
<td>32</td>
<td>114</td>
<td>40</td>
<td>16</td>
<td>2.448</td>
<td>40</td>
<td>16</td>
<td>2.448</td>
<td>52</td>
<td>64</td>
<td>2.448</td>
<td>49</td>
<td>512</td>
<td>2.448</td>
</tr>
</tbody>
</table>

Fig. 9. Visual comparisons with Photo2ClipArt [Favreau et al. 2017]. Please zoom in to see the differences. ©macrovector on Shutterstock.com, OpenClipart-Vectors on Pixabay, Zheng-Jun Du, callmetak on Freepik, bgrfx on Freepik, upklyak on Freepik.
Fig. 10. Reconstruction quality comparisons. For each example shown in Fig. 9, we visualize the difference between the input image and the reconstructed image generated by Photo2ClipArt [Favreau et al. 2017] (left) and our method (right). The reconstruction errors are provided (inset). Values are mean-squared error for RGB values lying in the range [0, 1].

It only considers decompositions that conform to our perception rules. In the future, we expect to make use of more perception rules to improve the output. Third, our method only supports linear gradient layers. In the future, we would like to consider other types of gradients, such as radial or quadratic, to better fit more complex color variations, especially for real images.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments and constructive suggestions. This work was supported by the National Natural Science Foundation of China (Project Number: 61932003), the Youth Program of Natural Science Foundation of Qinghai Province (Project Number: 2023-ZJ-951Q) and the Applied Basic Research Project of Qinghai Province (Project Number: 2021-ZJ-717). Yotam Gingold was supported in part by a gift from Adobe Inc.

REFERENCES


