Pacific Graphics 2018
H. Fu, A. Ghosh, and J. Kopf
(Guest Editors)

Volume 37 (2018), Number 7

Decomposing Images into Layers with Advanced Color Blending

Yuki Koyama and Masataka Goto

National Institute of Advanced Industrial Science and Technology (AIST), Japan

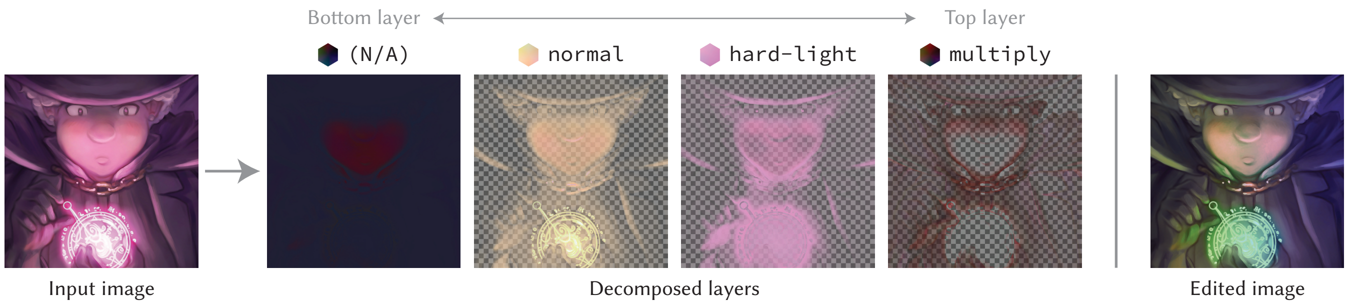

Figure 1: *Our method decomposes an input image (Left) into layers that can reproduce the image by composition with* advanced color-blend modes *such as* `hard-light` *and* `multiply` *(Middle). Users can specify the color-blend mode and desired color distribution for each layer, as well as the number of layers. Output layers are useful for non-trivial image manipulation such as lighting-aware color editing (Right). Input image courtesy of David Revoy.*

Abstract

Digital paintings are often created by compositing semi-transparent layers using various advanced color-blend modes, such as "color-burn," "multiply," and "screen," which can produce interesting non-linear color effects. We propose a method of decomposing an input image into layers with such advanced color blending. Unlike previous layer-decomposition methods, which typically support only linear color-blend modes, ours can handle any user-specified color-blend modes. To enable this, we generalize a previous color-unblending formulation, in which only a specific layering model was considered. We also introduce several techniques for adapting our generalized formulation to practical use, such as the post-processing for refining smoothness. Our method lets users explore possible decompositions to find the one that matches for their purposes by manipulating the target color-blend mode and desired color distribution for each layer, as well as the number of layers. Thus, the output of our method is a layered, easily editable image composition organized in a way that digital artists are familiar with. Our method is useful for remixing existing illustrations, flexibly editing single-layer paintings, and bringing physically painted media (e.g., oil paintings) into a digital workflow.

CCS Concepts

•***Computing methodologies*** $\rightarrow$ *Image processing;*

1. Introduction

Digital paintings are often created by compositing multiple layers using software such as Adobe Photoshop [Adob], GIMP [The], or Krita [KDE]. Layers are used mainly to manage different parts in an illustration (*e.g.*, separating a background layer and a main object layer) and create special effects on colors (*e.g.*, adding glow, shadow, and highlight). For the latter purpose, digital artists often use various *advanced color-blend modes* for blending colors between layers, such as `multiply`, `screen`, and `color-burn`, rather than just using the basic color-blend mode (*i.e.*, `normal`). For example, the `multiply` mode can create shading effects, and the `overlay` and `color-dodge` modes can create vivid contrast and lighting effects (*c.f.*, [Nie16, Jel17] and Figure 2). There are many combinations of color-blend modes and blended colors, which create various different effects, and there is no single correct way for such layering; every digital art work may use different creative layering styles. Once a painting is complete, it is exported as a bitmap image, which no longer has layer information. Once layer information is lost, it becomes much more difficult to manipulate the "baked" effects to revise the art work or create derivative works.

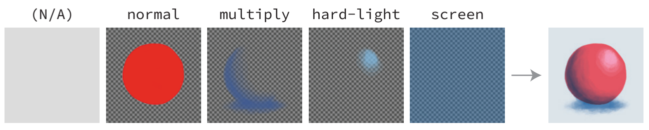

Figure 2: *Example of layer composition. Various advanced color-blend modes are combined to create effects of shadow, highlight, etc.*

Our goal is to provide digital artists with a means of decomposing a bitmap image into semi-transparent layers that can reproduce the original image by composition. This problem has been addressed by many researchers recently [CRA11, TLG16, AASP17, LFDH17, IRWM17, TEG18]. However, previous methods are formulated assuming specific layering models, for example, the *linear additive model* [AASP17, LFDH17, TEG18] (in which colors are blended simply by adding their RGB values). Although it is desirable for digital artists to be able to organize decomposed layers in a way that they are familiar with or in a way depending on inputs or situations (*i.e.*, taking manipulations after decomposition into account), previous methods do not satisfy this demand.

We propose a method for layer decomposition that can handle any user-specified advanced color-blend modes. To the best of our knowledge, this is the first method for accomplishing this. Our method enables digital artists to organize decomposed layers flexibly using advanced color blending (for example, separate layers based on semantics such as shading and lighting), and the output is easily editable by the user as the representation is the one that the user is familiar with. Figure 1 shows an example of our decomposition; the user specified desired color distributions for each of the layers and the `normal`, `hard-light` and `multiply` modes for the second, third, and fourth layers, respectively, so that the layers could extract color effects separately. The resulting layers are useful for non-trivial image manipulation; in this case, by filling the color channels of the `hard-light` layer by a gradation of different colors, the user could easily achieve lighting-aware color editing. In addition, animated effects (*e.g.*, blinking the glow) can be easily created based on the decomposed layers (see the accompanying video).

Our method is based on a generalized formulation of *color unblending* (or *color unmixing*; we favor using the former for consistency with other terms). Color unblending is a problem of finding each layer's RGB alpha (RGBA) value that can together reproduce the input color by composition and each layer color reasonably fits the desired layer-associated color distribution. We generalize the previous color-unblending formulation by Aksoy *et al.* [AAPS16, AASP17], which solved this problem by assuming a specific linear model. This generalization is achieved by using a general recursive composition rule. To efficiently solve the color-unblending problem, it is necessary that the partial derivatives of generalized constraints can be calculated; for this, we show that these can be calculated in a recursive manner using the *chain rule*. We also introduce several techniques for adapting our generalized formulation to practical use, including how to refine resulting layers by considering spatial smoothness, how to choose an initial solution for color unblending, and how to produce grayscaled layers by modifying constraints. Our method facilitates a variety of non-trivial image manipulations such as remixing existing illustrations in a complex way, applying layer-based effects to single-layer paintings, and bringing physical paintings into a digital workflow.

Contributions. Our contributions are summarized as follows.

- We present the first method of decomposing images into layers with advanced color blending. Our method allows users to flexibly organize decomposed layers by manipulating the number of layers and specifying any color-blend mode and desired color distribution for each layer.
- We present a generalized formulation of the color-unblending problem, which is derived from a general recursive composition rule. We show that necessary partial derivatives can be calculated in a recursive manner with this generalized formulation. We also introduce several techniques for adapting this formulation to practical use.
- We demonstrate a variety of applications of our method, in which users can carry out different decompositions depending on inputs or image-manipulation scenarios, and then edit decomposed layers using layer-wise operations.

2. Related Work

2.1. Layer Decomposition

Layer decomposition is a problem of decomposing an input image into multiple layers. This is useful for non-trivial image manipulation, such as editing colors in an illumination-aware way and applying visual effects (*e.g.*, glow and blur) selectively.

Tan *et al.* [TLG16] presented a method of decomposing digital paintings into layers based on geometric operations in RGB space. This method was then extended in an arXiv paper [TEG18] to analyze the geometry in RGBXY space (the 5-dimensional space formed by the RGB color axes and XY position axes), which produces more spatially coherent results. These methods heavily rely on the linearity of the specific blending models (*i.e.*, the *simple alpha blending model* [TLG16] and the *linear additive model* [TEG18]); thus, layers with non-linear color-blend modes (*e.g.*, `color-burn`) cannot be produced. Lin *et al.* also presented a method based on the linear additive model in their arXiv paper [LFDH17].

Some methods were designed to decompose images into *vector graphics* layers [RLMB*14, FLB17]. These methods are based on the parametric nature of vector graphics; thus, take different approaches from our pixel-wise approach. Note that they only take into account the simple alpha blending model and do not support advanced color-blend modes. Some other methods were designed to support specific types of inputs, such as physical paintings [TDLG18, AMSL17] and time-lapse videos of paintings [TDSG15].

Intrinsic image decomposition is a special class of layer decomposition. The main goal is to decompose an input photograph into an albedo and irradiance layers [BBS14]. These two layers are usually composited using the `multiply` mode to reproduce the input photograph. Some methods use more complex layering models. For example, the method proposed by Innamorati *et al.* [IRWM17] incorporates a specular layer with the `add` (also known as `linear-dodge`) mode and occlusion layer with the `multiply` mode. The method proposed by Carroll *et al.* [CRA11] further decomposes an irradiance layer into direct and indirect illumination layers that are composited with the `add` mode. While their target images are photographs with consistent lighting, ours are artistic paintings, where every artist can use different layering models to create various effects. For example, the irradiance effect is not always expressed with the `multiply` mode in paintings. Thus, we need a general formulation that is capable of handling any layering model.

Aksoy *et al.* [AAPS16, AASP17] presented layer-decomposition methods based on color unblending, on which our method is built. While our purpose of layer decomposition is to separate colors, Aksoy *et al.*'s main purpose was to separate regions of an image (*i.e.*, soft segmentation) for region-based image manipulation such as green-screen keying. Their color unblending was formulated by assuming the linear additive model, and we introduce how it can be generalized for handling advanced color blending.

2.2. Extraction of Representative Colors

Palette extraction is a problem of extracting a small number of representative colors from an image. Chang *et al.* [CFL*15] proposed a palette-extraction method based on k-means clustering and used it for photo recoloring. Tan *et al.* [TLG16] proposed a palette-extraction method that is closely tied to their layer-decomposition method (and later a more efficient method was proposed in an arXiv paper [TEG18]). Instead of extracting colors, extracting color *distributions* is more useful in some scenarios. Aksoy *et al.* [AASP17] proposed a method of building a small number of representative color-distribution models from an image, in which each distribution is represented as a Gaussian kernel in RGB space (we also use this representation as explained later).

In our problem setting, none of the above automatic extraction methods is directly useful because they are not designed to be used with advanced color blending but designed to be used with the simple alpha blending or linear additive models. Our target users are likely to have their unique expectations about color distributions of resulting layers. Thus, instead of automatically estimating color distribution models, we enable users to interactively specify them.

2.3. Color-Blend Modes

Advanced color blending has been intensively used in software for digital painting (*e.g.*, [Adob, The, KDE]) and visual effects (*e.g.*, [Adoa]). In addition, it has recently been supported by web browsers and vector graphics software because it was added to CSS and SVG specifications [W3C11, W3C15]; thus, it has been more and more universal and its importance in visual design has been increasing. Advanced color blending is used not only for compositing layers but also for processing individual brush strokes. Some digital artists favor adding strokes in a single layer rather than in separated layers, but even in this case advanced color blending is often used to achieve various color effects (*c.f.*, [Rev15]).

Our method supports such advanced color blending. Specifically, we tested our method with 13 popular color-blend modes that appeared in a SVG specification [W3C11]. It is extendable for supporting other color-blend modes such as `linear-burn` and `pin-light`, as long as they are functions that take two RGB values as input and return an RGB value and are differentiable (at almost every point). Other color-blend strategies that do not fit this condition (*e.g.*, the Kubelka-Munk model [TDLG18, AMSL17] requires absorption and scattering coefficients of physical pigments and the `dissolve` mode relies on randomness) are out of the scope of this paper.

3. Basics of Color Blending and Composition

This section describes the basics of how multiple layers are composited into an image in general cases [W3C11, W3C15]. We include this section for readers who are not familiar with the general composition model.

Terminology. *Color-blend modes* (*e.g.*, `normal`, `multiply`, `screen`, and `add`) are for blending two input colors and specified by *blend functions*. *Composition operators* are the operators defining how two layers are composited together. These were defined by Porter and Duff [PD84], including 12 basic operators (*e.g.*, `source-over`, `destination-over`, and `xor`) and an additional operator called `plus`. The *layering model* refers to a specific combination of a blend mode and composition operator. For example, the *linear additive model* refers to the combination of the `add` mode and `plus` operator. (Note that this model is called by many different names; for example, "plus" [W3C11], "lighter" [W3C15], "alpha add" [AASP17, Adoa], and "additive color mixing model" [TEG18].) The *simple alpha blending model* refers to the combination of the `normal` mode and `source-over` operator.

Notations. Each pixel in a single layer has an RGBA value:

$$\mathbf{x} = \begin{bmatrix}\mathbf{c}^{\mathrm{T}} & \alpha\end{bmatrix}^{\mathrm{T}} = \begin{bmatrix}c^r & c^g & c^b & \alpha\end{bmatrix}^{\mathrm{T}} \in [0,1]^4. \tag{1}$$

In this paper, we do not use the *pre-multiplied* color-component notation [PD84] to avoid confusion. In what follows, s and d stand for *source* (corresponding to the top layer) and *destination* (corresponding to the bottom layer), respectively.

3.1. General Color-Blending and Composition Formulation

Concept. The composition and color blending can be considered as a two-step process as follows. Given source and destination RGBA values, a blended RGB value is first calculated from the source and destination RGB values using the blend function, and the blended pixel opacity is set to the source opacity. Then, it is composited with the destination RGBA value using Porter-Duff's composition operator. This process is performed independently for each pixel.

General Formulation. The two-step process can be performed simultaneously using a single *general* equation for any combination of the color-blend mode and composition operator [W3C11]:

$$\begin{aligned}&\mathtt{comp}_{\mathbf{c}}(\mathbf{x}_s, \mathbf{x}_d)\\ &\quad = \frac{f(\mathbf{c}_s, \mathbf{c}_d)\alpha_s\alpha_d + Y\alpha_s(1-\alpha_d)\mathbf{c}_s + Z\alpha_d(1-\alpha_s)\mathbf{c}_d}{\mathtt{comp}_{\alpha}(\alpha_s, \alpha_d)},\end{aligned} \tag{2}$$

$$\mathtt{comp}_{\alpha}(\alpha_s, \alpha_d) = X\alpha_s\alpha_d + Y\alpha_s(1-\alpha_d) + Z\alpha_d(1-\alpha_s), \tag{3}$$

where $\texttt{comp}_{\mathbf{c}}$ and $\texttt{comp}_{\alpha}$ correspond to RGB and opacity channels, respectively,

$$f : [0,1]^3 \times [0,1]^3 \to [0,1]^3 \tag{4}$$

is a *blend function* and defined for each color-blend mode (see [W3C15] for the definitions), and

$$X, Y, Z \in \{0, 1, 2\} \tag{5}$$

are defined for each Porter-Duff's operator [W3C11]. For convenience, we concisely represent the equations as

$$\texttt{comp}(\mathbf{x}_s, \mathbf{x}_d) = \begin{bmatrix} \texttt{comp}_{\mathbf{c}}(\mathbf{x}_s, \mathbf{x}_d) \\ \texttt{comp}_{\alpha}(\alpha_s, \alpha_d) \end{bmatrix}. \tag{6}$$

Discussion. This formulation can be found in an obsolete SVG specification [W3C11]. The latest specification [W3C15] has a different formulation with the same expressiveness. Instead of using three operator-specific constant values X, Y, Z, the latest one uses two operator-specific functions of source and destination alpha values. We favor the above formulation since it was useful in calculating its partial derivatives more concisely.

Specialization. The linear additive model can be obtained by specifying the `add` blend function and the `plus` operator: $f(\mathbf{c}_s, \mathbf{c}_d) = \mathbf{c}_s + \mathbf{c}_d, X = 2, Y = Z = 1$. Similarly, the simple alpha blending model can be obtained by specifying the `normal` blend function and the `source-over` operator: $f(\mathbf{c}_s, \mathbf{c}_d) = \mathbf{c}_s, X = Y = Z = 1$.

3.2. Recursive Layering

When there are n layers ($n \geq 2$), the composited RGBA value $\hat{\mathbf{x}}_n = \begin{bmatrix} \hat{\mathbf{c}}_n^{\mathrm{T}} & \hat{\alpha}_n \end{bmatrix}^{\mathrm{T}}$ is recursively calculated by the rule:

$$\hat{\mathbf{x}}_k = \begin{cases} \mathbf{x}_1 & (k = 1) \\ \texttt{comp}^k(\mathbf{x}_k, \hat{\mathbf{x}}_{k-1}) & (\text{otherwise}) \end{cases}, \tag{7}$$

where $\texttt{comp}^k$ is the composition operation associated with the k-th layer, a variable with a "hat" indicates that it is the result of the composition of the layer itself and the layers below, and the subscript indicates the layer index (the 1st layer is the bottom and the n-th layer is the top).

4. Generalized Color Unblending Formulation

In this section, we present a color-unblending formulation by generalizing Aksoy *et al.*'s formulation [AAPS16, AASP17] using the general composition formulation. While their formulation is specific to the linear additive model, our formulation can also cover more general cases. At this moment, we focus on the principle of the color-unblending formulation and keep the discussion as mathematically simple as possible; we will present more specific techniques for adapting this formulation to practical usages in the next section (*e.g.*, how user control is offered).

Problem Setting. The input for this problem consists of a target image, target number of layers n, and configurations for each of the n layers. A layer configuration consists of a color-blend mode, composition operator, and color model (which is explained later). Note that a color-blend mode and composition operator are not necessary specified for the bottom layer since they do not affect composition. Given such an input, color unblending is carried out for each pixel independently. Let $\mathbf{x}^{\text{target}}$ be a pixel's RGBA value of the target image. The goal of color unblending is to find RGBA values $\mathbf{x}_1, \ldots, \mathbf{x}_n$ that can reproduce the target RGBA value $\mathbf{x}^{\text{target}}$ when they are composited and reasonably fit the associated color models. For convenience, we represent them as

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1^{\mathrm{T}} & \cdots & \mathbf{x}_n^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{4n}. \tag{8}$$

Overview. An optimization-based approach is taken to solve this problem. Specifically, a constrained optimization problem is solved; an objective function is minimized such that several (hard) equality constraints are satisfied.

4.1. Objective Function

An energy function $E(\mathbf{x})$ is defined in the same way as Aksoy *et al.*'s method [AAPS16]:

$$E(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i D_i(\mathbf{c}_i), \tag{9}$$

where $D_i(\mathbf{c}_i)$ is the distance (or deviation) from the i-th layer's RGB values to the i-th layer's color model. This energy function is used as the objective function to be minimized. Note that we do not include the *sparsity term* [AASP17] in the objective function since it is for improving soft segmentation but our interest is not segmentation.

A color model is defined as a Gaussian distribution in RGB space following [AAPS16]. The distance is calculated using the squared Mahalanobis distance. Following [AASP17], we visualize a color model as a hexagon whose center color represents the center of the Gaussian kernel and color variation is determined by putting the principal axes of its covariance matrix to the diagonals.

4.2. Hard Constraints

Special Case. Aksoy *et al.*'s formulation [AAPS16, AASP17] takes the following equality constraints:

$$\sum_{i=1}^{n} \alpha_i \mathbf{c}_i = \mathbf{c}_{\text{target}}, \quad \sum_{i=1}^{n} \alpha_i = \alpha_{\text{target}}, \tag{10}$$

along with box constraints. These equations are derived by specifically assuming the linear additive model.

It is worth noting that their actual equality constraints are described differently; they formulate the equality constraints such that the element-wise *squared* deviations of these equations equal to zeros. However, it is not necessary for the deviations to be squared; thus, we prefer to represent them in simpler (not squared) forms.

General Case. The above equality constraints are for ensuring that the composited RGBA value becomes equivalent to the target RGBA value but derived from a specific layering model. We generalize these constraints for various cases. Specifically, the equality constraints can be represented as

$$\hat{\mathbf{c}}_n = \mathbf{c}_{\text{target}}, \quad \hat{\alpha}_n = \alpha_{\text{target}}, \tag{11}$$

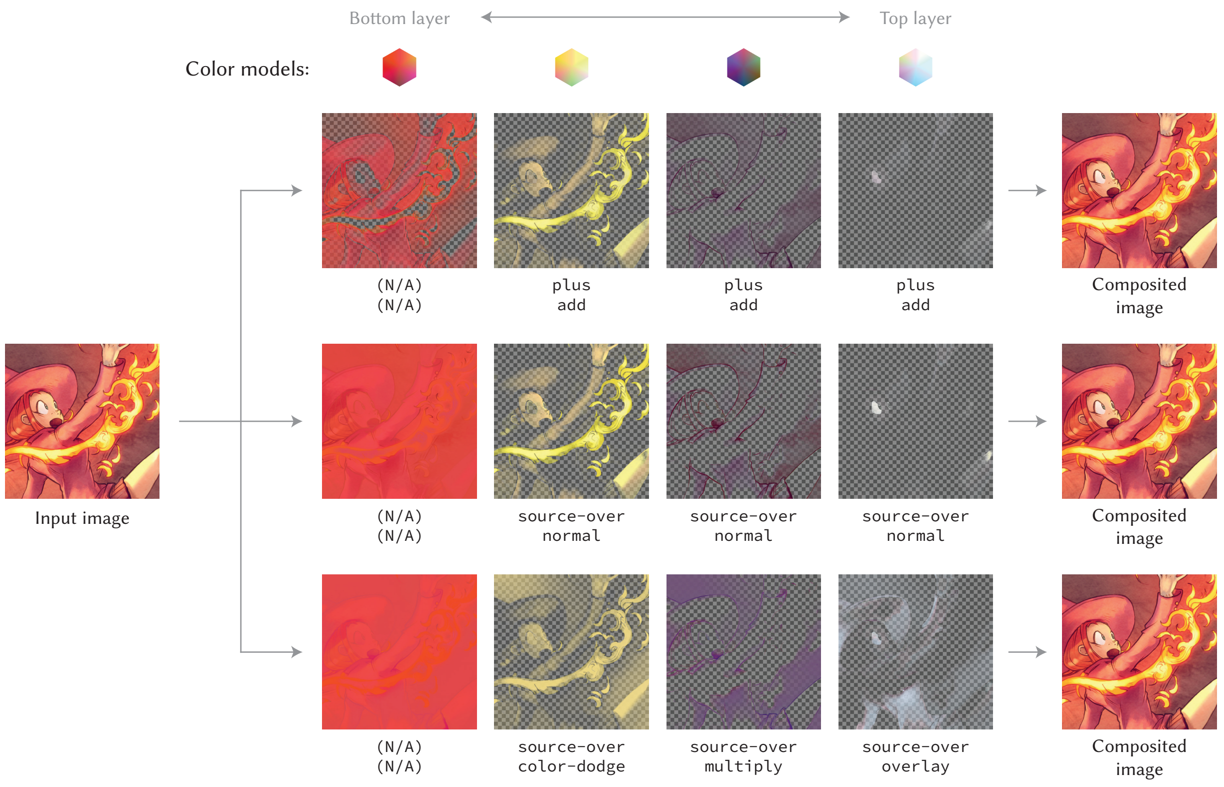

Figure 3: *Our generalized unblending formulation can produce various layer decompositions (even with same color models) by specifying different composition operators and color-blend modes. In the first row, the* `plus` *operator and* `add` *blend mode are specified to produce the (so-called) linear additive model; the result is identical to that generated using Aksoy et al.'s formulation [AAPS16, AASP17]. In the second and third rows, the* `source-over` *operator is specified. The second row uses the* `normal` *blend mode, so the results can be composited using the standard alpha blending. The third row uses advanced color-blend modes (i.e.,* `color-dodge`*,* `multiply` *and* `overlay`*) to extract semantically meaningful effects (i.e., shading). Input image courtesy of David Revoy.*

where $\hat{\alpha}_n$ and $\hat{\mathbf{c}}_n$ are obtained by the recursive application of general composite operations (Equation 2 and Equation 3). These constrains are simply written as

$$\mathbf{C}(\mathbf{x}) = \mathbf{0}, \quad \text{where } \mathbf{C}(\mathbf{x}) = \hat{\mathbf{x}}_n - \mathbf{x}_{\text{target}} \in \mathbb{R}^4. \tag{12}$$

4.3. Solver

Following [AAPS16], these equality constraints are handled by *the augmented Lagrangian method* [NW06] (also called *the original method of multipliers*). Specifically, to solve the constrained minimization problem, the augmented Lagrangian method iteratively solves a sequence of *unconstrained* minimization problems:

$$\min_{\mathbf{x} \in [0,1]^{4n}} \mathcal{L}(\mathbf{x}), \text{ where } \mathcal{L}(\mathbf{x}) = E(\mathbf{x}) - \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{C}(\mathbf{x}) + \frac{\rho}{2}\|\mathbf{C}(\mathbf{x})\|^2, \tag{13}$$

where the parameters $\boldsymbol{\lambda} \in \mathbb{R}^4$ and $\rho \in \mathbb{R}$ are updated after each unconstrained minimization. See [AAPS16] for the details of the parameter settings and stop criteria of this iteration. As the solver for the unconstrained minimization, we use L-BFGS method [LN89].

4.4. Derivatives

To solve the unconstrained minimization problem (Equation 13), it is necessary to calculate the partial derivatives:

$$\frac{\partial \mathcal{L}(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial E(\mathbf{x})}{\partial \mathbf{x}} + \frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}}(\rho \mathbf{C}(\mathbf{x}) - \boldsymbol{\lambda}). \tag{14}$$

The energy function is easily differentiated given that the distance function D is differentiable. Calculating the derivative of the equality constraints is less trivial. First, the derivative is a concatenation of the derivatives of the composited RGBA values with respect to $\mathbf{x}_i$:

$$\frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial \hat{\mathbf{x}}_n}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \hat{\mathbf{x}}_n}{\partial \mathbf{x}_1}^{\mathrm{T}} & \cdots & \frac{\partial \hat{\mathbf{x}}_n}{\partial \mathbf{x}_n}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{4n \times 4}. \tag{15}$$

Due to the recursive nature of composition, it is difficult to express the derivatives in a single equation. Yet, they can be calculated recursively using the *chain rule*; the derivative of the RGBA values obtained by compositing the bottom k layers with respect to the RGBA values of the i-th layer is calculated by recursively applying the following rule:

$$\frac{\partial \hat{\mathbf{x}}_k}{\partial \mathbf{x}_i} = \begin{cases} \mathbf{I} & (i = k = 1) \\ \frac{\partial}{\partial \mathbf{x}_k} \mathtt{comp}^k(\mathbf{x}_k, \hat{\mathbf{x}}_{k-1}) & (i = k \neq 1) \\ \frac{\partial \hat{\mathbf{x}}_{k-1}}{\partial \mathbf{x}_i} \cdot \frac{\partial}{\partial \hat{\mathbf{x}}_{k-1}} \mathtt{comp}^k(\mathbf{x}_k, \hat{\mathbf{x}}_{k-1}), & (\text{otherwise}) \end{cases} . \quad (16)$$

Note that the case of $i > k$ never appears in the recursive calculation. The general composition rules for compositing two layers (Equation 2 and Equation 3) are easily differentiated by the source RGBA values $\mathbf{x}_s$ or the destination RGBA values $\mathbf{x}_d$ given that the blend function f is differentiable.

4.5. Results and Reproduction of Previous Method

Our general formulation can be specialized by choosing a specific color-blend mode and composition operator to be used. By choosing the `add` mode and `plus` operator, we can derive equations identical to Aksoy *et al.*'s ones [AAPS16]; thus our method can reproduce results identical to theirs (Figure 3; Top). Using the same formulation but choosing the `source-over` operator and `normal` mode, we can generate different results even with the same color models (Figure 3; Middle). Finally, by specifying the `source-over` operator and advanced color-blend modes, it generates results with advanced blending (Figure 3; Bottom). These examples demonstrate the generality of our method. Note that these results were generated with the refinement techniques described in [AASP17] (Top) and the next section (Middle, Bottom), and the same color models were specified for every case only for explanatory purposes.

5. Techniques for Adapting Our Formulation to Practical Use

5.1. Additional Assumptions

Considering practical use of our method, we introduce the following additional assumptions.

Always Source-Over Advanced color-blend modes are virtually always used with the `source-over` operator. For example, the latest specification of CSS and SVG [W3C15] explains them with only the `source-over` operator. Furthermore, popular software [Adob, The, KDE] does not support other operators. Thus, it is reasonable to limit the available operator in this way from a practical viewpoint.

Opaque Background Like painting with physical media, digital painting is often created with a fully opaque layer at the bottom. We examined over 10 online tutorials by professionals and found that all follow this condition.

5.2. Modifying Formulation

Omitting Alpha Constraint. As a result of the above two assumptions, composited pixels are ensured to be fully opaque. Thus, we can remove the alpha component from the equality constraints (Equation 12) as it is always satisfied; thus redundant.

Resolving Ambiguity. When using the `normal` mode, the solution of the color-unblending optimization may be ambiguous (*i.e.*, there could be many solutions that are equally optimal). For example, when a top layer has a fully opaque region, the region in the layers below can have arbitrary RGBA values without changing the

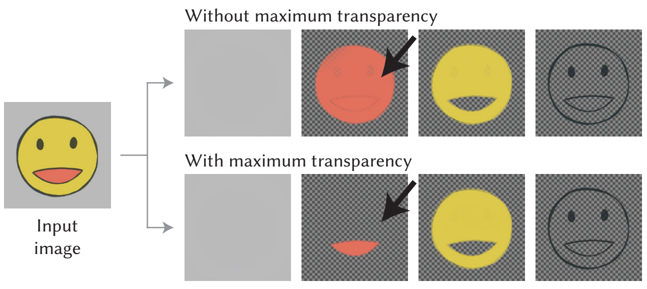

Figure 4: *Toy example showing that some pixels' alpha values can be arbitrary. The pointed region will be hidden by the above yellow layer when compositing, so the alpha values from optimization can be ambiguous. We solve this ambiguity by favoring transparency.*

composite RGBA value (see Figure 4; Top). We provide an option to solve this ambiguity by preferring these arbitrary pixels to be fully transparent (see Figure 4; Bottom) to reduce redundant paintings. This is achieved by adding the following term to the objective function: $\varepsilon \sum_{i=1}^{n} \alpha_i$, where ε can be small (we set $\varepsilon = 0.01$).

Handling Grayscale Layers. Digital artists sometimes use grayscale layers to draw shading or lighting effects. Our method is capable of handling grayscale layers if necessary. When the i-th layer is specified as grayscaled, the following additional equality constraint is added:

$$\sqrt{3}\mathbf{c}_i - \|\mathbf{c}_i\| \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^{\mathrm{T}} = \mathbf{0}. \quad (17)$$

5.3. User Control

Users specify the number of layers and then the color-blend mode, color model, and option of grayscaling for each layer. As for the color model specification, a Gaussian kernel in RGB space has nine degrees of freedom in total (three for the center and six for the covariance), but we expose only four sliders to users to control the center RGB value and single variance parameter σ. The covariance matrix is simply set as $\sigma\mathbf{I}$, and we found that this simplified input is expressive enough in practice.

Optionally, users can use automatic color-model extraction as a starting point to explore possible decompositions. In this case, color models are determined using Aksoy *et al.*'s method [AASP17], and all the layers are specified to use the `normal` mode.

Performing the color-unblending process to a full-resolution image takes a few minutes or more using a consumer-level laptop, which prohibits users from efficiently exploring possible decompositions. For this problem, we introduce a simple solution: we let users efficiently perform exploration with a down-sampled image (*e.g.*, 100×100 pixels), which runs less than ten seconds in most cases and is considered sufficiently fast for the trial-and-error purpose, and then runs our method with the full resolution.

5.4. Initial Solution

We choose the initial solution for optimization by the following rule. For color channels, we simply fill each layer using the center color of its corresponding color distribution. For alpha channels, we set all to 0.5 (except for the background layer; the background layer's alpha is set to one). The reason we do not use 0 or 1 for initial alphas is that the derivatives with respect to some layer colors could vanish with such alphas at the beginning of optimization, which is not desirable. Although this initialization rule is simple, it works well without any noticeable problem.

Note that Aksoy *et al.* [AAPS16, AASP17] use a different rule. For each pixel, their method "assigns" the target color to a layer that has the closest distance to its color distribution with full opacity, and the other layers are set to transparent. Thus, with the linear additive model, the initial solution is ensured to reproduce the target image. However, this rule is not reasonable in our case because this solution does not reproduce the target image with advanced color-blend modes. Note that the augmented Lagrangian method does not require the initial solution to satisfy the equality constraints.

5.5. Refinement for Spatial Smoothness

The techniques discussed thus far are performed for each pixel independently. Thus, generated layers may be noisy (although the composited image can be smooth). To ensure better spatial consistency, we use the *alpha matte refinement* procedure proposed by Aksoy *et al.* [AASP17]. Their original idea can be summarized as follows.

1. Perform color unblending for each pixel independently and obtain (possibly) noisy layers.
2. Apply the *guided filter* [HST13] to the alpha channel of each layer using the input image as guidance and obtain smoothed alpha channels.
3. Normalize the smoothed alpha values for each pixel such that the composited alpha value becomes 1.
4. Perform color unblending for each pixel again, but this time the smoothed and normalized alpha values are used as additional hard constraints.

To adapt this procedure to our problem setting, small modifications to the original method are made. First, thanks to the opaque-background assumption, we do not need to normalize alpha values after applying smoothing; regardless of the non-background layers' alpha values, the composited image always becomes fully opaque.

Second, we apply the guided filter not only to the alpha channels but also to the background RGB channels, and then use the smoothed channels as additional hard constraints in the second color unblending. The reason for this modification is that, unlike the original case, smoothing the alpha channel of the background layer does not contribute to spatial smoothness in the resulting background layer. Thus, to obtain a smooth background layer, its RGB channels need to be directly smoothed. Figure 5 shows the effect of this modification.

Note that there is a limitation in the second modification. Depending on the other layers' color-blend modes, there is no guarantee that the original image will be reproduced perfectly from the final (smoothed) results. For example, if an image is decomposed into two layers and the top layer has the `darken` blend mode, it is impossible to reproduce a color that is lighter than the background color

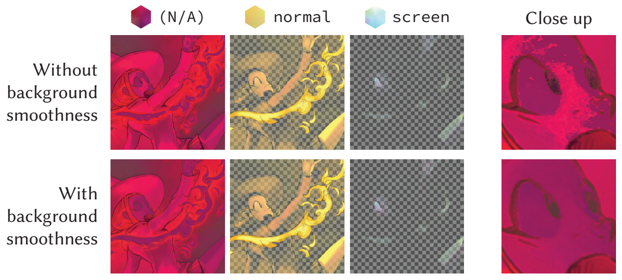

Figure 5: *Results of decomposition by not enforcing (Top) and enforcing (Bottom) background layer smoothness.*

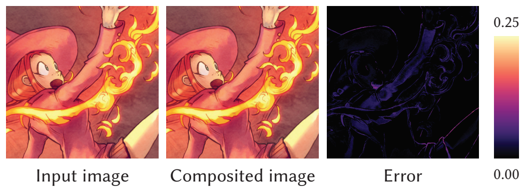

Figure 6: *Deviation caused by enforcing smoothness of background layer. (Left) Input image. (Middle) Composited image using the layers generated with background smoothness in Figure 5. (Right) Error between the two images in RGB-space Euclidean distance.*

regardless of the top color. However, we found that this modification produces only very small (unnoticeable in most cases) deviations from the input image in practical cases. Figure 6 shows an example of deviation caused by this modification.

6. Results and Applications

Editing Existing Illustrations. Figure 1 and Figure 7 illustrate applications of our method in which it is used for editing existing illustrations by taking into account lighting effects. For the case of Figure 7, suppose that the user wants to replace the blue sky background to a sunset one. Naïvely compositing the character region with a sunset background does not work well because the bluish lighting effect does not match to the sunset scene (Figure 7; Leftmost). Our method can extract the bluish lighting effect as, in this case, an `overlay` layer. By manipulating this layer's hue, the composited image becomes a better fit to the sunset scene.

Using Layers in Animations. Advanced color blending is used for creating not only static images but also animations [Adoa]; thus, decomposed layers by our method can be useful for animation authoring. See the accompanying video for examples of this scenario.

Decomposing Physical Paints. Our method is also useful for bringing physically painted media into digital workflows. Figure 8 shows an example of this scenario. In this case, a scanned oil painting was decomposed into layers and then layer-based digital effects were applied. Note that we do not intend to reproduce the underlying physical pigment distributions (*c.f.*, [TDLG18, AMSL17]) but intend to create layers that are useful in later digital workflows.

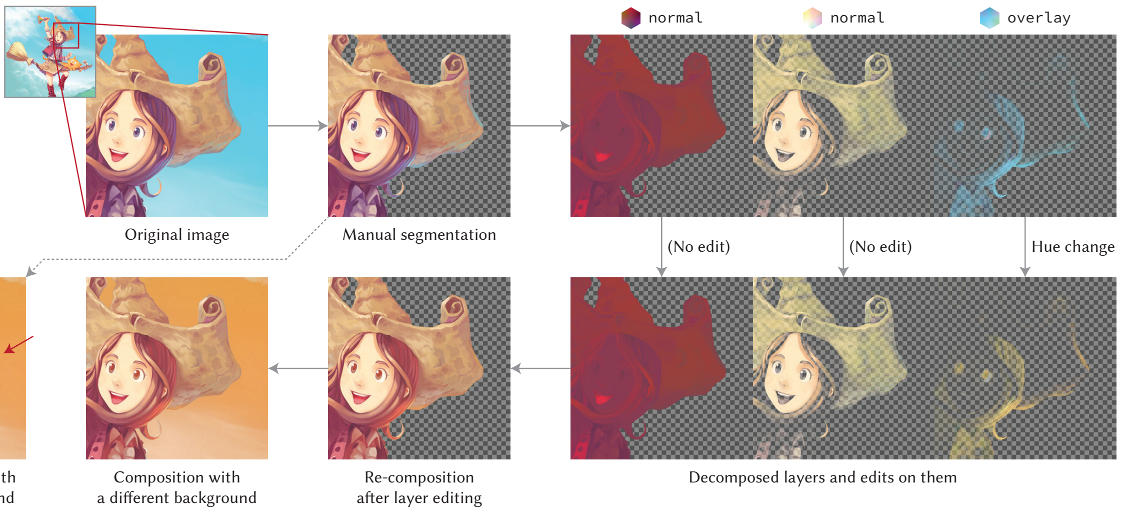
normal
normal
overlay
Original image
Manual segmentation
(No edit)
(No edit)
Hue change
Naïve composition with a different background
Composition with a different background
Re-composition after layer editing
Decomposed layers and edits on them

Figure 7: *Example use of our method, in which the blue sky background is replaced with a sunset one. With our method, the character region is decomposed into three layers: the bottom two layers roughly correspond to albedo, and the top layer roughly corresponds to lighting by the blue sky represented as an* `overlay` *layer. To match the lighting in sunset, the top layer's hue is modified. Finally, the layers are re-composited with the sunset background. Note that naïvely compositing the character region with the sunset background (the leftmost image) does not look good because lighting is mismatched. Input image courtesy of David Revoy.*

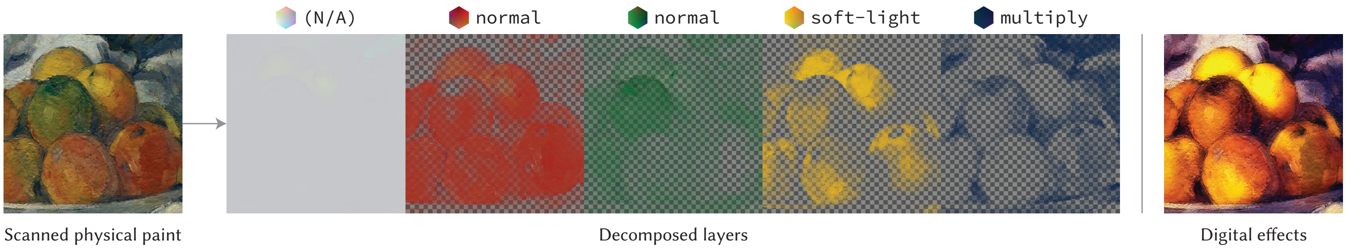

Figure 8: *Applying layer-based digital effects (e.g., adding glow to highlights, coloring shadows with gradation, etc.) to a physical painting.*

Fixing Deep Colorization. Automatic colorization of line sketches using deep neural networks has become popular [Pre, SLF*17]. A problem with this approach is that a generated image is not likely to be perfect. Thus, the user has to carry out a difficult manual fix or give up using it and start painting from scratch. Our method enables a new digital painting workflow by effectively using such a deep-colorized image as a good starting point. Figure 9 illustrates this workflow. Instead of directly editing the generated single-layered image, the user can decompose it into layers with familiar advanced color blending then individually edit each unsatisfactory layer by adding strokes, changing hue, *etc.*

Remixing Multiple Illustrations. Our method facilitates remixing of illustrations to create derivative works; users can easily harmonize colors of target illustrations via decomposed layers (see Figure 10).

Intrinsic Decomposition. Although our main goal is to decompose artistic digital paintings, it is interesting to see how this works for photographs. Figure 11 shows the results of decomposing photographs in the manner of intrinsic decomposition [BBS14, IRWM17]. Our method successfully produced high-quality decompositions. Note that our method requires manual specifications of color models unlike with automatic methods, and we do not intend to argue that our method is superior.

7. Evaluation

Computational Cost. We show results of performance tests in Figure 12. For this, we used the image and similar layer configurations in Figure 1. Like as the previous method [AAPS16, AASP17], our method basically runs for each pixel independently except for the guided filtering part; thus, it can gain benefits of parallelization (Figure 12; Left) and it linearly scales with respect to the number of pixels (Figure 12; Middle). We also evaluated how the number of layers affects its performance (Figure 12; Right). Note that our parallelization is based on CPU multi-threading; by implementing our method on GPU may improve its performance. It is also possible to improve the performance by reusing unblending results of some pixels as initial solutions for other pixels with similar colors, which we leave as a future work.

Compared to Aksoy *et al.*'s linear additive formulation [AAPS16], our formulation requires recursive calculations, which potentially

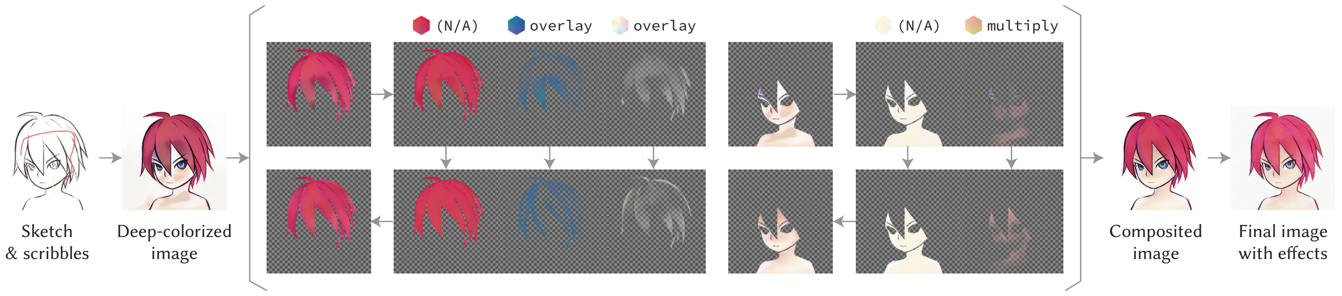

Figure 9: *New digital painting workflow possible with our method, where a deep-colorized image is used as a good starting point. First, the user generates a colorized image from a sketch using deep colorization (e.g., PaintsChainer [Pre]). To fix the generated image, which is unlikely to be perfect, the user decomposes each region (e.g., hair and skin) into layers with advanced blending. Then, she manually edits each layer individually if necessary (in this case, the user fixes shading by adding strokes). Finally, she re-composites the image and optionally adds post effects (e.g., overlaying textures).*

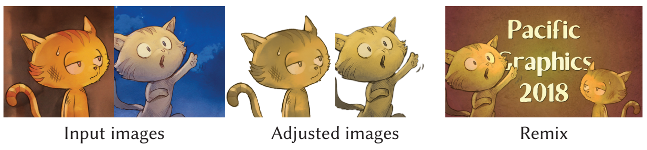

Figure 10: *Remixing illustrations to create a derivative work, in which our method is used to harmonize colors of target images. Input images courtesy of David Revoy.*

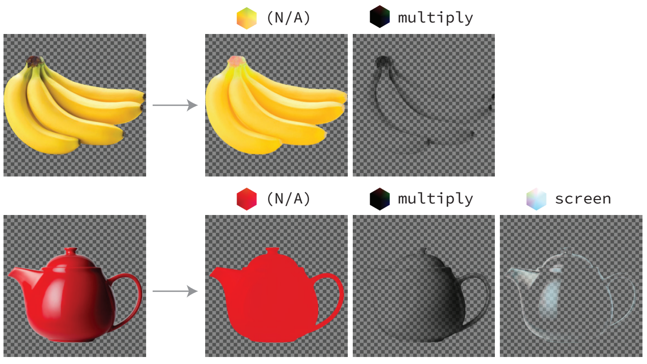

Figure 11: *Examples of applying our method to photographs. We intend to produce results similar to intrinsic decomposition. Grayscale constraints are used for the* `multiply` *layers for both cases. Input images are provided by [IRWM17].*

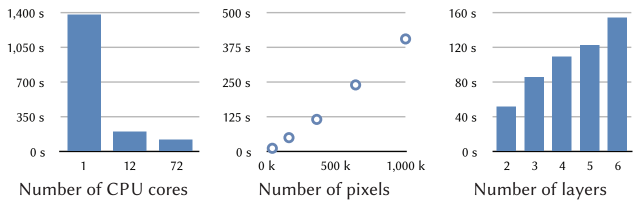

Figure 12: *Performance of our method. We used Intel Xeon Gold 6150 Processor 2.70 GHz. By default, we used 72 cores and specified an image with* 600 × 600 *resolution and 5 as the number of layers. We varied the number of cores (Left), the image size (Middle), and the number of layers (Right), respectively.*

makes our method slower even when our method handles linear additive cases. To evaluate this aspect, we implemented Aksoy *et al.*'s formulation by replacing our recursive calculation parts with Aksoy *et al.*'s (non-recursive) ones and then compared its performance with ours using the same image, color models, and layer configurations (*i.e.*, the `plus` operator and the `add` mode). While they produced identical results, our formulation was about 1.4x, 2.8x, and 3.8x slower than Aksoy *et al.*'s when specifying 2, 4, and 6 layers, respectively.

Advanced Blending vs. Linear Blending. Digital paintings created using advanced color blending are probably better decomposed by the originally used color-blend modes. To confirm this, we decomposed an image synthesized using the `multiply` and `color-dodge` modes (Figure 13; Top) using two different conditions: the originally used layering model and the linear additive model. In the latter condition, three color models were automatically selected by the method by Aksoy *et al.* [AASP17]. Figure 13 shows the results. Both models produced visually plausible decompositions, and especially the former one could produce the layers very similar to the original layers (though a slight difference is noticeable). We visualize maps of "color inhomogeneity" value (the alpha-weighted sum of the Euclidean distances between the pixel RGB value and the center of the desired color distribution for each layer). The choice of color-blend modes affected the color homogeneity of each layer; the former condition produced more homogeneous layers in terms of color distributions.

Random Test. To see how robustly our method can produce feasible results for possibly invalid inputs, we generated results with random settings. For each layer, we randomly assigned a color-blend mode from the 13 options and randomly generated a Gaussian color distribution. The number of layers was specified as either 2, 4, or 8. Figure 14 shows the results. Even when the input seemed unreasonable, our method could still robustly produce feasible results.

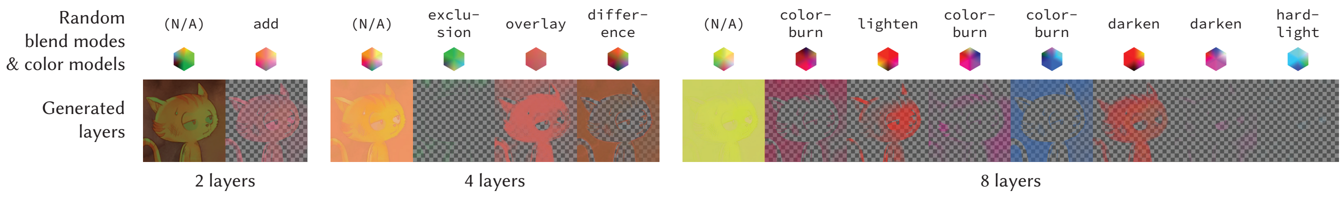

Figure 14: *Experimental results with randomly chosen color-blend modes and color models. Even with this challenging setting of unreasonable specifications, our method could still robustly produce feasible decompositions. Input image courtesy of David Revoy.*

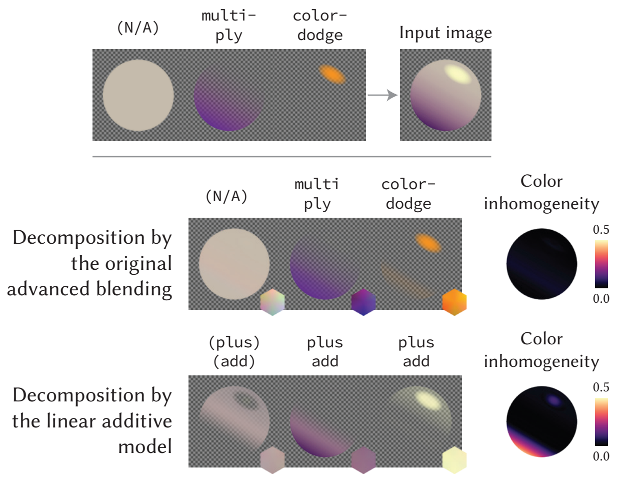
(N/A)
multi-ply
color-dodge
Input image
Decomposition by the original advanced blending
Color inhomogeneity
0.5
0.0
(plus) (add)
plus add
Decomposition by the linear additive model

Figure 13: *Decomposition of a synthetic image (Top) using two different conditions. (Middle) Using the originally used color-blend modes. (Bottom) Using the linear additive model with the same number of layers. The former produces layers that are similar to the original ones and are more homogeneous in terms color.*

Our method assigned zero alphas for the layers that were not useful for reasonable decomposition. This indicates that our method is not very sensitive to input configurations. More results can be found in the accompanying video.

8. Discussion and Future Work

Reproduction of Original Layers. Layer decomposition is inherently an ill-posed problem, where many feasible solutions can exist. Thus, there is no guarantee that our method can reproduce original layers from a composited image, even when appropriate color-blend modes and color distributions are specified. For example, Figure 13 (Middle) is an example of trying to reproduce original layers, which looks mostly successful, but there is some noticeable difference even for this simple case. Note that our goal is not to enable "reverse engineering" but to provide a flexible means of image manipulation that users want to perform.

Differentiability of Blend Functions. Some blend functions are not differentiable at every point. For example, the `overlay` function is defined in a per-channel manner as $f(c_s, c_d) = 2c_s c_d$ if $c_d < 0.5$ and $f(c_s, c_d) = 1 - 2(1 - c_s)(1 - c_d)$ otherwise [W3C15]; thus, it is not differentiable at $c_d = 0.5$. We have so far not observed any problem regarding this issue (*e.g.*, numerical instability).

Identical Colors. Our method is purely based on colors and does not take into account either semantics of decomposed regions (*c.f.*, [AOP*18]) or spatial distributions of pixels (*c.f.*, [TEG18]). Thus, identical color pixels cannot be decomposed in different ways even when they do not belong to the same region or are spatially distant.

Color-Blend Mode and Color Model Selection. We assume that users have their own expected decompositions in mind. Thus, we enable users to interactively specify both color-blend modes and color models directly via an interface. A few trials and errors of fine-tuning color models (especially adjusting RGB values; variances were not changed from default in most cases) were typically performed to obtain the results shown in this paper; we consider this is not a substantial burden for users in practice. Nevertheless, investigating (semi-)automatic approaches to determine reasonable color-blend modes and color models for initial suggestions, which would facilitate efficient specification, will be important. Also, it would be worth investigating to optimize the color models simultaneously during layer decomposition.

Color Model Representation. Our method uses a Gaussian model following the previous methods [AAPS16, AASP17]. Investigating more expressive models (*e.g.*, Gaussian mixture model) is probably useful for more controllable decompositions. In this case, user interfaces for controlling color models may need to be investigated.

Other Possible Applications. Typical intrinsic and illumination decomposition methods use relatively simple layering models. These methods may be improved by incorporating our generalized color unblending formulation since it supports non-linear color blending.

Data-Driven Layer Decomposition. To support those who are not familiar with advanced color blending, we believe that using data-driven approaches is a promising direction. For example, if a number of professional digital paintings with layer information are available, we might be able to extract templates of layer structures and learn typical color distributions [NRS15] while taking layer semantics into account. By investigating this direction, a future system could suggest an initial decomposition to the user rather than having her begin to specify layer configurations from scratch.

Acknowledgments

This work was supported in part by JST ACCEL Grant Number JPMJAC1602, Japan.

References

[AAPS16] Aksoy Y., Aydin T. O., Pollefeys M., Smolić A.: Interactive high-quality green-screen keying via color unmixing. *ACM Trans. Graph. 35*, 5 (Aug. 2016), 152:1–152:12. doi:10.1145/2907940. 2, 3, 4, 5, 6, 7, 8, 10

[AASP17] Aksoy Y., Aydin T. O., Smolić A., Pollefeys M.: Unmixing-based soft color segmentation for image manipulation. *ACM Trans. Graph. 36*, 2 (Mar. 2017), 19:1–19:19. doi:10.1145/3002176. 2, 3, 4, 5, 6, 7, 8, 9, 10

[Adoa] Adobe Systems Inc.: Adobe After Effects CC. http://www.adobe.com/products/aftereffects.html. 3, 7

[Adob] Adobe Systems Inc.: Adobe Photoshop CC. http://www.adobe.com/products/photoshop.html. 1, 3, 6

[AMSL17] Aharoni-Mack E., Shambik Y., Lischinski D.: Pigment-based Recoloring of Watercolor Paintings. In *Proc. NPAR '17* (2017), pp. 1:1–1:11. doi:10.1145/3092919.3092926. 2, 3, 7

[AOP*18] Aksoy Y., Oh T.-H., Paris S., Pollefeys M., Matusik W.: Semantic soft segmentation. *ACM Trans. Graph. 37*, 4 (July 2018), 72:1–72:13. doi:10.1145/3197517.3201275. 10

[BBS14] Bell S., Bala K., Snavely N.: Intrinsic images in the wild. *ACM Trans. Graph. 33*, 4 (July 2014), 159:1–159:12. doi:10.1145/2601097.2601206. 2, 8

[CFL*15] Chang H., Fried O., Liu Y., DiVerdi S., Finkelstein A.: Palette-based photo recoloring. *ACM Trans. Graph. 34*, 4 (July 2015), 139:1–139:11. doi:10.1145/2766978. 3

[CRA11] Carroll R., Ramamoorthi R., Agrawala M.: Illumination decomposition for material recoloring with consistent interreflections. *ACM Trans. Graph. 30*, 4 (July 2011), 43:1–43:10. doi:10.1145/2010324.1964938. 2, 3

[FLB17] Favreau J.-D., Lafarge F., Bousseau A.: Photo2clipart: Image abstraction and vectorization using layered linear gradients. *ACM Trans. Graph. 36*, 6 (Nov. 2017), 180:1–180:11. doi:10.1145/3130800.3130888. 2

[HST13] He K., Sun J., Tang X.: Guided image filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 6 (June 2013), 1397–1409. doi:10.1109/TPAMI.2012.213. 7

[IRWM17] Innamorati C., Ritschel T., Weyrich T., Mitra N. J.: Decomposing single images for layered photo retouching. *Comput. Graph. Forum 36*, 4 (July 2017), 15–25. doi:10.1111/cgf.13220. 2, 8, 9

[Jel17] JelArts: How to use layer modes in digital art // multiply, overlay, etc. https://youtu.be/OAv1CkQR4o4, 2017. 1

[KDE] KDE: Krita. https://krita.org/. 1, 3, 6

[LFDH17] Lin S., Fisher M., Dai A., Hanrahan P.: Layerbuilder: Layer decomposition for interactive image and video color editing, 2017. arXiv:1701.03754v2. 2

[LN89] Liu D. C., Nocedal J.: On the limited memory bfgs method for large scale optimization. *Math. Program. 45*, 3 (Dec. 1989), 503–528. doi:10.1007/BF01589116. 5

[Nie16] Nieves M.: How to shade easily with layer blend modes. https://design.tutsplus.com/articles/how-to-shade-easily-with-layer-blend-modes--cms-25895, 2016. 1

[NRS15] Nguyen C. H., Ritschel T., Seidel H.-P.: Data-driven color manifolds. *ACM Trans. Graph. 34*, 2 (Mar. 2015), 20:1–20:9. doi:10.1145/2699645. 10

[NW06] Nocedal J., Wright S. J.: *Numerical Optimization*. Springer, 2006. doi:10.1007/978-0-387-40065-5. 5

[PD84] Porter T., Duff T.: Compositing digital images. *SIGGRAPH Comput. Graph. 18*, 3 (Jan. 1984), 253–259. doi:10.1145/964965.808606. 3

[Pre] Preferred Networks: PaintsChainer. https://paintschainer.preferred.tech/. 8, 9

[Rev15] Revoy D.: Painting with blending-modes. https://youtu.be/AybFWViT-3Q, 2015. 3

[RLMB*14] Richardt C., Lopez-Moreno J., Bousseau A., Agrawala M., Drettakis G.: Vectorising bitmaps into semi-transparent gradient layers. *Comput. Graph. Forum 33*, 4 (2014), 11–19. doi:10.1111/cgf.12408. 2

[SLF*17] Sangkloy P., Lu J., Fang C., Yu F., Hays J.: Scribbler: Controlling deep image synthesis with sketch and color. In *Proc. CVPR '17* (2017), pp. 6836–6845. doi:10.1109/CVPR.2017.723. 8

[TDLG18] Tan J., DiVerdi S., Lu J., Gingold Y.: Pigmento: Pigment-based image analysis and editing. *IEEE Trans. Vis. Comput. Graph.* (2018). doi:10.1109/TVCG.2018.2858238. 2, 3, 7

[TDSG15] Tan J., Dvorožňák M., Sýkora D., Gingold Y.: Decomposing time-lapse paintings into layers. *ACM Trans. Graph. 34*, 4 (July 2015), 61:1–61:10. doi:10.1145/2766960. 2

[TEG18] Tan J., Echevarria J., Gingold Y.: Palette-based image decomposition, harmonization, and color transfer, 2018. arXiv:1804.01225. 2, 3, 10

[The] The GIMP Team: GNU Image Manipulation Program (GIMP). https://www.gimp.org/. 1, 3, 6

[TLG16] Tan J., Lien J.-M., Gingold Y.: Decomposing images into layers via rgb-space geometry. *ACM Trans. Graph. 36*, 1 (Nov. 2016), 7:1–7:14. doi:10.1145/2988229. 2, 3

[W3C11] W3C: SVG compositing specification, 2011. URL: https://www.w3.org/TR/2011/WD-SVGCompositing-20110315/. 3, 4

[W3C15] W3C: Compositing and blending level 1, 2015. URL: https://www.w3.org/TR/2015/CR-compositing-1-20150113/. 3, 4, 6, 10