Unmixing-Based Soft Color Segmentation for Image Manipulation

YAĞIZ AKSOY
ETH Zürich and Disney Research Zürich
TUNÇ OZAN AYDIN and ALJOŠA SMOLIĆ
Disney Research Zürich
and
MARC POLLEFEYS
ETH Zürich

We present a new method for decomposing an image into a set of soft color segments that are analogous to color layers with alpha channels that have been commonly utilized in modern image manipulation software. We show that the resulting decomposition serves as an effective intermediate image representation, which can be utilized for performing various, seemingly unrelated, image manipulation tasks. We identify a set of requirements that soft color segmentation methods have to fulfill, and present an in-depth theoretical analysis of prior work. We propose an energy formulation for producing compact layers of homogeneous colors and a color refinement procedure, as well as a method for automatically estimating a statistical color model from an image. This results in a novel framework for automatic and high-quality soft color segmentation that is efficient, parallelizable, and scalable. We show that our technique is superior in quality compared to previous methods through quantitative analysis as well as visually through an extensive set of examples. We demonstrate that our soft color segments can easily be exported to familiar image manipulation software packages and used to produce compelling results for numerous image manipulation applications without forcing the user to learn new tools and workflows.

Categories and Subject Descriptors: I.3.8 [**Computer Graphics**]: Applications; I.4.6 [**Image Processing and Computer Vision**]: Segmentation—*Pixel classification*; I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis—*Color*

General Terms: Image Processing

Additional Key Words and Phrases: Soft segmentation, digital compositing, image manipulation, layer decomposition, color manipulation, color unmixing, green-screen keying

ACM Reference Format:

Yağız Aksoy, Tunç Ozan Aydin, Aljoša Smolić, and Marc Pollefeys. 2017. Unmixing-based soft color segmentation for image manipulation. ACM Trans. Graph. 36, 2, Article 19 (March 2017), 19 pages.
DOI: http://dx.doi.org/10.1145/3002176

Authors' addresses: Y. Aksoy and M. Pollefeys, Department of Computer Science, ETH Zurich, Universitatstrasse 6, CH-8092, Zurich, Switzerland; emails: {yaksoy, marc.pollefeys}@inf.ethz.ch; T. Aydin, Disney Research Zurich, Stampfenbachstrasse 48, CH-8006, Zurich, Switzerland; email: tunc@disneyresearch.com; A. Smolic, Graphics Vision and Visualisation Group, School of Computer Science and Statistics, Trinity College Dublin, Dublin 2, Ireland; email: smolica@scss.tcd.ie.

DOI: http://dx.doi.org/10.1145/3002176

1. INTRODUCTION

The goal of soft color segmentation is to decompose an image into a set of layers with alpha channels, such as in Figure 1(b). These layers usually consist of fully opaque and fully transparent regions, as well as pixels with alpha values between the two extremes wherever multiple layers overlap. Ideally, the color content of a layer should be homogeneous, and its alpha channel should accurately reflect the color contribution of the layer to the input image. Equally important is to ensure that overlaying all layers yields the input image. If a soft color segmentation method satisfies these and a number of other well-defined criteria that we discuss in detail later, then the resulting layers can be used for manipulating the image content conveniently through applying per-layer modifications. These image manipulations can range from subtle edits to give the feeling that the image was shot in a different season of the year (Figure 1(c)) to more pronounced changes that involve dramatic hue shifts and replacing the image background (Figure 1(d)). In this article, we propose a novel soft color segmentation method that produces a powerful intermediate image representation, which in turn allows a rich set of image manipulations to be performed within a unified framework using standard image editing tools.

Obtaining layers that meet the demanding quality requirements of image manipulation applications is challenging, as even barely visible artifacts on individual layers can have a significant negative impact on quality when certain types of image edits are applied. That said, once we devise a soft color segmentation method that reliably produces high-quality layers, numerous image manipulation tasks can be performed with little extra effort by taking advantage of this image decomposition. Importantly, the resulting layers naturally integrate into the layer-based workflows of widely used image manipulation packages. By using soft color segmentation as a black box, and importing the resulting layers into their favorite image manipulation software, users can make use of their already-existing skills.

While the traditional *hard* segmentation is one of the most active fields of visual computing, soft color segmentation has received

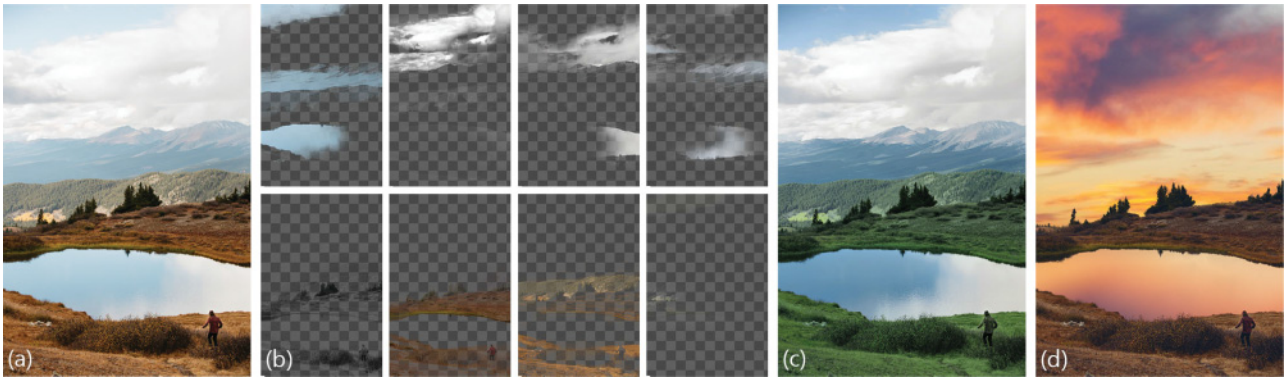

Fig. 1. Our method automatically decomposes an input image (a) into a set of soft segments (b). In practice, these soft segments can be treated as layers that are commonly utilized in image manipulation software. Using this relation, we achieve compelling results in color editing (c), compositing (d), and many other image manipulation applications conveniently under a unified framework.

surprisingly little attention so far. In addition to direct investigations of soft color segmentation [Tai et al. 2007; Tan et al. 2016], certain natural alpha matting and green-screen keying methods presented soft color segmentation methods—without necessarily calling them as such—as a component in their pipeline. While it may seem at a first glance that one can simply use any of these previous methods for practical and high-quality photo manipulation, a closer look reveals various shortcomings of the currently available soft color segmentation methods. To this end, in this article, we provide a theoretical analysis of the problems with previous work, which we also supplement with quantitative evaluation, as well as visual examples that support our arguments.

We address the two main challenges of soft color segmentation: devising a color unmixing scheme that results in high-quality soft color segments and automatically determining a content-adaptive color model from an input image. We propose a novel energy formulation that we call *sparse color unmixing* (SCU) that decomposes the image into layers of homogeneous colors. The main advantage of SCU when compared to similar formulations used in different applications such as green-screen keying [Aksoy et al. 2016] is that it produces compact color segments by favoring fully opaque or transparent pixels. We also enforce spatial coherency in opacity channels and accordingly propose a color refinement procedure that is required for preventing visual artifacts while applying image edits. We additionally propose a method for automatically estimating a *color model* corresponding to an input image, which comprises a set of distinct and representative color distributions. Our method determines the size of the color model automatically in a content adaptive manner. We show that the color model estimation can efficiently be performed using our novel *projected color unmixing* (PCU) formulation.

We present a comprehensive set of results in the article and supplementary material that show that our method consistently produces high-quality layers. Given such layers, we demonstrate that numerous common image manipulation applications can be reduced to trivial per-layer operations that can be performed conveniently through familiar software tools.

2. RELATED WORK

Previous work uses the term *soft segmentation* in various contexts, such as the probabilistic classification of computerized tomography (CT) scans [Posirca et al. 2011], computing per-pixel foreground/background probabilities [Yang et al. 2010b], and interactive image segmentation utilizing soft input constraints [Yang et al. 2010a]. In fact, generally speaking, even the traditional k-means clustering algorithm can be considered as a soft segmentation method, as it computes both a label as well as a confidence value for each point in the feature space [Tai et al. 2007]. In contrast to these approaches, we seek to compute proper alpha values rather than arbitrarily defined confidence values or probabilities.

The goals of our method are similar to those of the alternating optimization soft color segmentation technique [Tai et al. 2007] and the decomposition technique via RGB-space geometry [Tan et al. 2016], which decompose input images into a set of soft color segments and demonstrate their use in various image manipulation applications. A number of techniques proposed in natural matting and green-screen keying literature, such as the ones by Singaraju and Vidal [2011], Chen et al. [2013], and Aksoy et al. [2016], can also be regarded as soft color segmentation methods. While the goal of natural matting is to compute the alpha channel of the foreground in an image, the aforementioned methods can also be used to obtain soft color layers from the input image. Our method has several advantages when compared to these works in terms of the quality of the opacity channels, the color content of individual layers and the absence of need for user input. We present an in-depth discussion of our differences to and the shortcomings of these methods in Sections 4 and 6.

While we focus on soft *color* segmentation for the purpose of image manipulation, other methods in the literature have been proposed that utilize spatial information [Levin et al. 2008b; Tai et al. 2005], as well as texture [Yeung et al. 2008] for soft segmentation. The object motion deblurring method by Pan et al. [2016] performs soft segmentation simultaneously with blur kernel estimation to increase performance. There is also a substantial body of work on traditional, *hard* segmentation. For a discussion and evaluation of hard segmentation methods, we refer the reader to the recent survey by Pont-Tuset and Marques [2015]. Similarly, there are also examples of *hard color segmentation* such as the one by Omer and Werman [2004]. Decomposing an image into several layers with transparency also has distinct applications with specialized methods such as illumination decomposition for realistic material recoloring [Carroll et al. 2011], anti-aliasing recovery [Yang et al. 2011], vectorizing bitmaps [Richardt et al. 2014], and reflection removal [Shih et al. 2015].

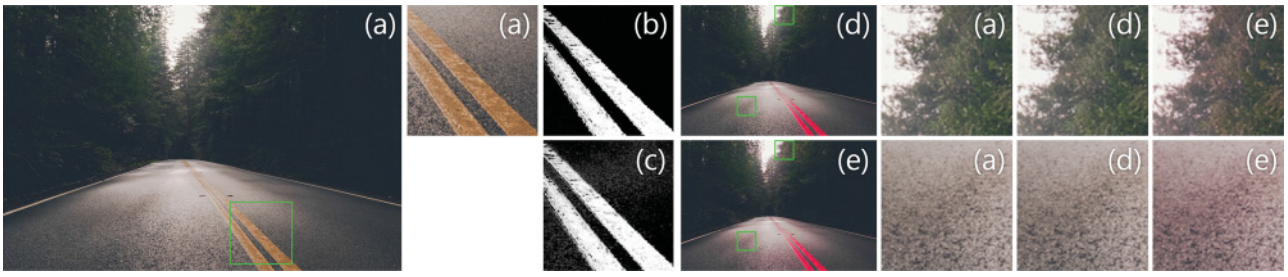

Fig. 2. The original image (a) is shown with the alpha channels of the layers corresponding to the yellow of the road lines estimated by the proposed sparse color unmixing (b) and by the color unmixing [Aksoy et al. 2016] (c). Notice the spurious alpha values on the road in (c). When our sparse color unmixing results are utilized for a color change (d) the regions that do not contain the yellow of the road are not affected. On the other hand, a color change using the color unmixing (e) effects unrelated regions as well. The color change was applied after matte regularization and color refinement stages for both (d) and (e).

3. SOFT COLOR SEGMENTATION

Our main objective in this article is to decompose an image into multiple partially transparent segments of homogeneous colors, that is *soft color segments*, which can then be used for various image manipulation tasks. Borrowing from image manipulation terminology, we will refer to such soft color segments simply as *layers* throughout this article.

A crucial property of soft color segmentation is that overlaying all layers obtained from an image yields the original image itself so editing individual layers is possible without degrading the original image. In mathematical terms, for a pixel p, we denote the opacity value as α_i^p and the layer color in RGB as $\boldsymbol{u}_i^p$ for the ith layer. and we want to satisfy the *color constraint*:

$$\sum_i \alpha_i^p \boldsymbol{u}_i^p = \boldsymbol{c}^p \quad \forall p, \tag{1}$$

where $\boldsymbol{c}^p$ denotes the original color of the pixel. The total number of layers will be denoted by N.

We assume that the original input image is fully opaque and thus requires the opacity values over all layers to add up to unity, which we express as the *alpha constraint*:

$$\sum_i \alpha_i^p = 1 \quad \forall p. \tag{2}$$

Finally, the permissible range for the alpha and color values are enforced by the *box constraint*:

$$\alpha_i^p, \boldsymbol{u}_i^p \in [0, 1] \quad \forall i, p. \tag{3}$$

For convenience, we will drop the superscript p in the remainder of the article and present our formulation at the pixel level, unless stated otherwise.

It should be noted that different representations for overlaying multiple layers exist. We use Equation (1), which we refer to as an *alpha-add* representation, in our formulation, which does not assume any particular ordering of the layers. This representation has also been used by Tai et al. [2007] and Chen et al. [2013] among others. In most commercial image editing software, however, the representation proposed by Porter and Duff [1984], referred to in this article as the *overlay* representation, is used. The difference and conversion between the two representations are presented in the appendix.

Our algorithm for computing high-quality soft color segments can be described by three stages: color unmixing, matte regularization, and color refinement, which will be discussed in the remainder of this section.

Color Unmixing: An important property we want to achieve within each layer is *color homogeneity*: The colors present in a layer should be sufficiently similar. To this end, we associate each layer with a 3D normal distribution representing the spread of the layer colors in RGB space, and we refer to the set of N distributions as the *color model*. Our novel technique for automatically extracting the color model for an image is discussed in detail in Section 5.

Given the color model, we propose the *sparse color unmixing* energy function in order to find a preliminary approximation to the layer colors and opacities:

$$\mathcal{F}_S = \sum_i \alpha_i \mathcal{D}_i(\boldsymbol{u}_i) + \sigma \left(\frac{\sum_i \alpha_i}{\sum_i \alpha_i^2} - 1 \right), \tag{4}$$

where the layer color cost $\mathcal{D}_i(\boldsymbol{u}_i)$ is defined as the squared Mahalanobis distance of the layer color $\boldsymbol{u}_i$ to the layer distribution $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$, and σ is the sparsity weight that is set to 10 in our implementation. The energy function in Equation (4) is minimized for all α_i and $\boldsymbol{u}_i$ simultaneously while satisfying the constraints defined in Equations (1)–(3) using *the original method of multipliers* [Bertsekas 1982]. For each pixel, for the layer with best fitting distribution, we initialize the alpha value to 1 and the layer color $\boldsymbol{u}_i$ to the pixel color. The rest of the layers are initialized to zero alpha value and the mean of their distributions as layer colors. The first term in Equation (4) favors layer colors that fit well with the corresponding distribution especially for layers with high alpha values, which is essential for getting homogeneous colors in each layer. The second term pushes the alpha values to be *sparse*, that is, favors 0 or 1 alpha values.

The first term in Equation (4) appears as the color unmixing energy proposed by Aksoy et al. [2016] as a part of their system for green-screen keying. They do not include a sparsity term in their formulation, and this inherently results in favoring small alpha values, which results in many layers appearing in regions that should actually be opaque in a single layer. The reason is that a better-fitting layer color for the layer with alpha close to 1 (hence a lower color unmixing energy) becomes favorable by leaking small contributions from others (assigning small alpha values to multiple layers) with virtually no additional cost as the sample costs are multiplied with alpha values in the color unmixing energy. This decreases the compactness of the segmentation and, as a result, potentially creates visual artifacts when the layers are edited independently. Figure 2 shows such an example obtained through minimizing the color unmixing energy, where the alpha channel of the layer that captures the yellow road line is noisy on the asphalt region, even though the yellow of the road is not a part of the color of the asphalt.

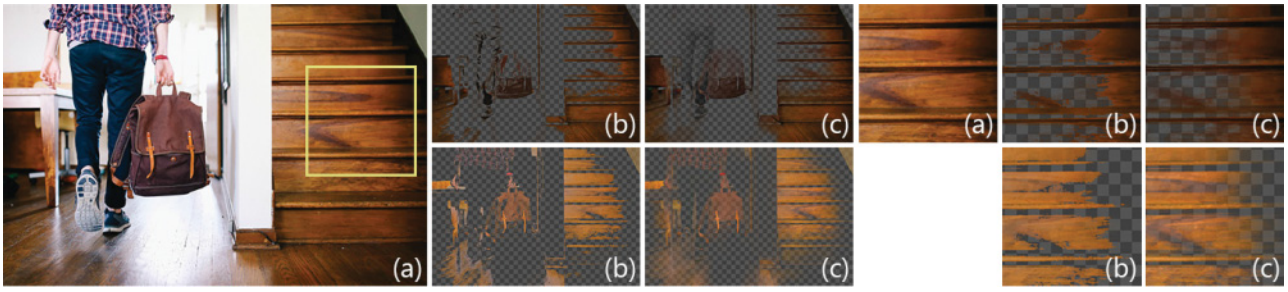

Fig. 3. Two layers corresponding to the dark (top) and light wood color in the original image (a) are shown before (b) and after (c) matte regularization and color refinement.

While these errors might seem insignificant at first, they result in unintended changes in the image when subjected to various layer manipulation operations such as contrast enhancement and color changes, as demonstrated in Figure 2.

The sparsity term in Equation (4) is zero when one of the layers is fully opaque (and thus all other layers are fully transparent) and increases as the alpha values move away from zero or 1. Another term for favoring matte sparsity has been proposed by Levin et al. [2008b]:

$$\sum_i |\alpha_i|^{0.9} + |1 - \alpha_i|^{0.9}. \tag{5}$$

This cost is infinitely differentiable in the interval [0, 1]. Its infinite derivatives at $\alpha_i = 0^+$ and $\alpha_i = 1^-$ causes the alpha values to stay at these extremes in the optimization process we employ. In spectral matting, the behavior of this function is used to keep alpha values from taking values outside [0, 1]. In our case, as the box constraints are enforced during the optimization of the sparse color unmixing energy, the negative values our sparsity cost takes outside the interval do not affect our results adversely.

Matte Regularization: Sparse color unmixing is done independently for each pixel, and there is no term ensuring spatial coherency. This may result in sudden changes in opacities that do not quite agree with the underlying image texture, as shown in Figure 3(b). Hence, spatial regularization of the opacity channels is necessary for ensuring smooth layers as in Figure 3(c). This issue also occurs frequently in sampling-based natural matting. The common practice for alpha regularization is using the matting Laplacian introduced by Levin et al. [2008a] as the smoothness term and solve a linear system that also includes the spatially non-coherent alpha values, as proposed by Gastal and Oliveira [2010]. While this method is very effective in regularizing mattes, on the downside, it is computationally expensive and consumes a high amount of memory especially as the image resolution increases.

The guided filter proposed by He et al. [2013] provides an efficient way to filter any image using the texture information from a particular image, referred to as the *guide* image. The guided filter is an edge-aware filtering method that can make use of an image, the guide image, to extract the edge characteristics and filter a second image using the edge information from the guide image efficiently. They discuss the theoretical similarity between their filter and the matting Laplacian and show that getting satisfactory alpha mattes is possible through guided filtering when the original image is used as the guide image. While filtering the mattes with the guided filter only approximates the behavior of the matting Laplacian, we observed that this approximation provides sufficient quality for the mattes obtained through sparse color unmixing. For a 1MP image, we use 60 as the filter radius and 10^{-4} as ϵ for the guided filter, as recommended by He et al. [2013] for matting, to regularize the alpha matte of each layer. As the resultant alpha values do not necessarily add up to 1, we normalize the sum of the alpha values for each pixel after filtering to get rid of small deviations from the alpha constraint. The filter radius is scaled according to the image resolution. Note that the layer colors are not affected by this filtering and they will be updated in the next step.

While enforcing spatial coherency on opacity channels is trivial using off-the-shelf filtering, dealing with its side effects is not straightforward. Obtaining spatially smooth results while avoiding disturbing color artifacts requires a second step that we discuss next.

Color Refinement: As the original alpha values are modified due to regularization, we can no longer guarantee that all pixels still satisfy the color constraint defined in Equation (1). Violating the color constraint in general severely limits the ability to use soft segments for image manipulation. For illustration, Figure 6 shows a pair of examples where KNN matting fails to satisfy the color constraint, which results in unintended color shifts in their results. To avoid such artifacts, we introduce a second energy minimization step, where we replace the alpha constraint defined in Equation (2) in the color unmixing formulation with the following term that forces the final alpha values to be as close as possible to the regularized alpha values:

$$\sum_i (\alpha_i - \hat{\alpha}_i)^2 = 0, \tag{6}$$

where $\hat{\alpha}_i$ represents the regularized alpha value of the ith layer. By running the energy minimization using this constraint, we recompute unmixed colors at all layers so they satisfy the color constraint while retaining spatial coherency of the alpha channel. Note that since the alpha values are determined prior to this second optimization, the sparsity term in Equation (4) becomes irrelevant. Hence, we only employ the unmixing term of the energy in this step. For the optimization, we initialize the layer colors as the values found in the previous energy minimization step.

Finally, to summarize our color unmixing process: We first minimize the sparse color unmixing energy in Equation (4) for every pixel independently. We then regularize the alpha channels of the soft layers using the guided filter and refine the colors by running the energy minimization once again, this time augmented with the new alpha constraint defined in Equation (6). This way we achieve soft segments that satisfy the fundamental color, alpha, and box constraints, as well as the matte sparsity and spatial coherency requirements for high-quality soft segmentation.

Note that the two energy minimization steps are computed independently for each pixel, and the guided filter can be implemented as a series of box filters. These properties make our algorithm easily parallelizable and highly scalable.

4. ANALYSIS OF STATE OF THE ART

While the particular deficiencies of current soft color segmentation methods will be apparent from the qualitative and quantitative evaluation results in Section 6 and the supplemental material, our goal in this section is to highlight the theoretical reasons behind some of those deficiencies. To this end, we take a closer look at the formulations of the current methods.

The soft color segmentation methods in the literature can be categorized into two classes: unmixing based and affinity based. Unmixing-based methods [Tai et al. 2007; Aksoy et al. 2016] including ours attempt to get unmixed colors and their corresponding alpha values by processing the observed color of each pixel with a statistical model for the layers. The method by Tan et al. [2016] can loosely be categorized as unmixing based as it computes the alpha values using a (non-statistical) color model using fixed layer colors. Affinity-based methods [Levin et al. 2008b; Singaraju and Vidal 2011; Chen et al. 2013], on the other hand, aim to use local or non-local pixel proximities to propagate a set of given labels to the rest of the image.

Alternating Optimization (AO): The alternating optimization algorithm proposed by Tai et al. [2007] is similar to ours in terms of the end goal. The authors define a Bayesian formulation that includes the alpha values, the layer colors, as well as the color model for the soft color segments and find the maximum *a priori* (MAP) solution to the problem by alternatingly optimizing for the alpha values, the layer colors, and the color model parameters. Here, we will only analyze AO's alpha and layer color estimation formulations and defer the discussion on its color model estimation to Section 5.

AO estimates the alpha values by defining a Markov random field that encodes the probability of the alpha values given the layer colors and the color model. After some algebraic manipulation, their maximization can be expressed as minimizing $\mathcal{E}_\alpha = \sum_p \mathcal{E}_\alpha^p$, with $\mathcal{E}_\alpha^p$ defined as

$$\mathcal{E}_\alpha^p = \frac{1}{2\sigma_c^2}\|\boldsymbol{c}^p - \sum_i \alpha_i^p \boldsymbol{u}_i^p\|^2 + \frac{1}{\sigma_c^2}\sum_i \alpha_i^p \mathcal{D}_i(\boldsymbol{u}_i) \tag{7a}$$

$$+ \sum_{q\in\mathcal{N}_p} \log\left(1 + \frac{1}{2\sigma_b}\left(\sqrt{\sum_i \left(\alpha_i^p - \alpha_i^q\right)^2}\right)\right), \tag{7b}$$

where $\mathcal{N}_p$ is the 8-neighborhood of the pixel p and σ_b and σ_c are algorithmic constants. The color unmixing energy actually appears as the second term in Equation (7a) together with the color constraint, whereas Equation (7b) shows the smoothness term. The MAP estimation in AO is done via loopy belief propagation. They assign the soft labels assigned by the belief propagation algorithm as the alpha values. This global optimization scheme becomes a limiting factor for AO in scaling to high-resolution images.

Due to the iterative nature of the algorithm, in the alpha estimation step, AO uses the color values from the previous iteration. Hence, it will find a compromise between the optimal alpha values and satisfying the color constraint, which is to be corrected in the color estimation step. Experimentally, we observed that this interdependency, when also coupled with the interdependency with the color model estimation, may result in layers oscillating between two local minima as the iterations progress. The smoothness term, on the other hand, depends on how close the estimated alpha values from the previous iteration are rather than using the image texture. Their results typically have unnatural gradients in layer alphas as it can be seen in Figure 4.

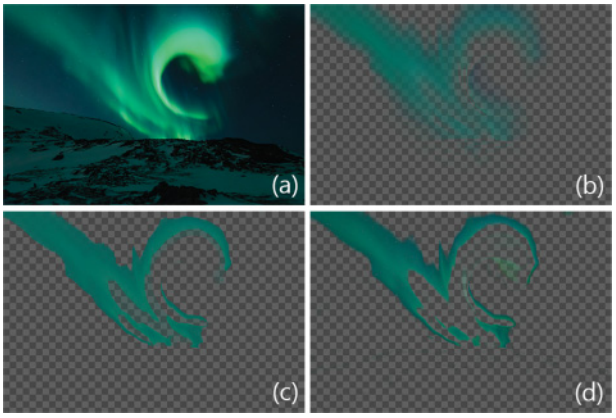

Fig. 4. The layers corresponding to the outer region of the northern lights in the input image (a) computed by the proposed algorithm (b), color unmixing (c) and alternating optimization (d). Notice that the layers in (c, d) have abrupt edges in opacity while our result has a smooth transition following the input image content. Note that AO's result already has smoothness enforced in their optimization procedure. A comparison of our results and that of CU after our matte regularization is presented in Figure 2.

Decomposition via RGB-Space Geometry (RGBSG): Tan et al. [2016] use a different approach to the soft color unmixing problem by fixing the layer colors beforehand and optimizing for the opacity values. Differing from all the approaches discussed in this section as well as ours, they use the overlay layers representation as defined in the appendix. This representation requires a pre-determined ordering of the layers, and RGBSG requires this ordering as input before the decomposition. The main advantage of using alpha-add representation over overlay representation can be said to be the indifference to layer order, which decreases the amount of user input needed. However, their end goal and application scenario is similar to ours.

They construct a color model that encompasses the hull of the RGB values in the image, which will be discussed in Section 5.2, and fix $\boldsymbol{u}_i$'s to these predetermined values. They define their energy function as

$$\begin{aligned} \mathcal{E}_{\text{RGBSG}} &= \omega_p\mathcal{E}_p + \omega_o\mathcal{E}_o + \omega_s\mathcal{E}_s \\ \mathcal{E}_p &= \frac{1}{K}\left\|\boldsymbol{u}_n - \boldsymbol{c} + \sum_i \left((\boldsymbol{u}_{i-1} - \boldsymbol{u}_i)\prod_{j=i}^{N}(1-\tilde{\alpha}_j)\right)\right\|^2 \\ \mathcal{E}_o &= \frac{1}{N}\sum_i -(1-\tilde{\alpha}_i)^2 \qquad \mathcal{E}_s = \frac{1}{N}\sum_i (\nabla\tilde{\alpha}_i)^2, \end{aligned} \tag{8}$$

where K is 3 or 4 depending on whether they use RGB or RGBA optimization as defined in their article, $\nabla\tilde{\alpha}_i$ is the opacity gradient, and $\omega_p = 375$, $\omega_o = 1$, and $\omega_s = 100$ are algorithmic constants. Notice the use of $\tilde{\alpha}$ as opposed to α due to their compositing formulation. As the layer colors are determined with the color model, this optimization only determines the opacity values, unlike the other unmixing-based approaches. The color constraint is satisfied with the $\mathcal{E}_p$ term while sparsity and smoothness is enforced via the $\mathcal{E}_o$ and $\mathcal{E}_s$ terms, respectively. Characteristically, their sparseness energy is somewhat similar to ours as it also depends on the sum of square of the alpha values. Their smoothness measure follows that of AO as it also depends on the smoothness of alpha values rather than the image texture. Their layers do not necessarily give the original image when overlayed due to the possibility of *imaginary* colors

in their color model, as will be discussed in Section 5.2. However, their layers have solid colors by definition.

One particular characteristic of RGBSG is that it requires a color model that encompasses all the colors that appear in the image from the outside. This requirement is sometimes limiting as the layers can not have colors that are close to the center of the RGB cube. A demonstration of this behavior is presented in Figure 9.

Green-Screen Keying via Color Unmixing (CU): The first step of our algorithm, sparse color unmixing, shares common elements with the color unmixing formulation proposed by Aksoy et al. [2016]. While CU does the energy minimization while satisfying the constraints in Equations (1)–(3) as we do, it lacks a sparsity measure in the energy formulation, as discussed in Section 3. The authors solve the matte sparsity problem by introducing a per-frame *local color model*, which locally determines subsets of their global color model. A first approximation of the local color model is computed via a Markov random field optimization, which later needs to be manually refined by the user in order to achieve high-quality keying results. Note that the user interaction for obtaining local color models is provided *not* for giving artistic freedom to the user, but it is *required* in order to fix the errors of the initial estimate.

CU also lacks a measure for spatial coherency. While they can achieve satisfactory results for green-screen keying with the help of local color models, when applied to soft color segmentation, their per-pixel formulation results in abrupt transitions between layers, as seen in Figure 4(c).

Multiple Image Layer Estimation (ML): Multiple image layer estimation method [Singaraju and Vidal 2011] makes use of the matting Laplacian proposed by Levin et al. [2008a]. The matting Laplacian encodes local affinities that effectively represent the alpha propagation between neighboring pixels. ML formulates the problem of estimating multiple soft layers into several sub-problems of two-layer estimation. Their formulation allows the estimation of N layers in closed form. However, they discuss in depth that it is not possible to solve for the layer alphas in closed-form while satisfying both non-negative alpha values and the alpha values summing up to 1 for $N > 2$.

Spectral Matting (SM): It is worth mentioning that the soft segmentation (as opposed to soft *color* segmentation) method spectral matting [Levin et al. 2008b] also extracts multiple layers by making use of the matting Laplacian. SM defines *matting components*, soft segments that can be determined as the eigenvectors of the matting Laplacian corresponding to its smallest eigenvalues. By making use of the sparsity prior shown in Equation (5), they find N plausible matting components, N being the number of layers specified by the user. The primary aim of SM is to extract spatially connected soft segments rather than layers of homogeneous colors. Their formulation fails to keep the alpha values in [0, 1].

Both ML and SM only estimate the alpha values. In order to get the layer colors, an additional step is needed for each of them. ML uses the layer color estimation method proposed by Levin et al. [2008a]. In this method, the authors define an energy for estimating the foreground and background colors (only for the $N = 2$ case) that make use of layer alpha and color derivatives:

$$\sum_p \| \boldsymbol{c}^p - \sum_i \alpha_i^p \boldsymbol{u}_i^p \|^2 + \begin{bmatrix} \nabla_x \alpha_0 \\ \nabla_y \alpha_0 \end{bmatrix}^T \sum_i \begin{bmatrix} \nabla_x \boldsymbol{u}_i^T \nabla_x \boldsymbol{u}_i \\ \nabla_y \boldsymbol{u}_i^T \nabla_y \boldsymbol{u}_i \end{bmatrix}. \quad (9)$$

This energy, which takes the alpha values as input, propagates the color values in the image to get plausible colors agreeing with the alpha values. As the color constraint is only one of the terms in the energy, the result does not necessarily satisfy the color constraint.

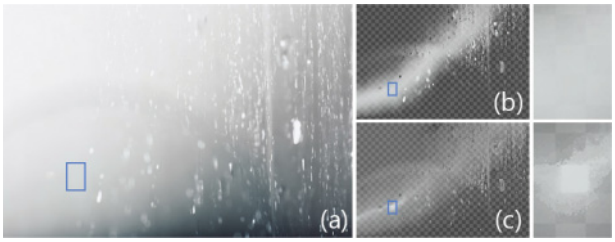

Fig. 5. A layer extracted by the proposed method (b) and KNN Matting (c). KNN's hard constraints on alpha values may cause artifacts as shown in the inset.

Fig. 6. Our layers, when overlaid (b), give us the original image (a), while layers extracted by KNN matting may result in erroneous reconstruction (c) by failing to satisfy the color constraint. The effect may be local color loss such as the lips in the top image or a global degradation of image quality as seen in the bottom image.

KNN Matting (KNN): KNN matting [Chen et al. 2013], in contrast to SM and ML, uses *non-local* affinities that are computed using the neighbors of each pixel in a feature space rather than only spatially close-by pixels. They also transform the layer estimation problem into a sparse linear system and solve for the alpha values of each layer separately. Unlike ML, they show that their algorithm naturally satisfies the constraint that the alpha values sum to 1. While the non-local approach, in fact, produces higher-quality soft layers when compared to its local counterparts, the sparse affinity matrix they construct ends up having many entries far from the diagonal. This significantly increases the runtime to be able to solve the linear system. KNN puts hard constraints around the seed pixels and this frequently results in disturbing blockiness artifacts due to their affinity definition as shown in Figure 5.

KNN matting proposes a layer color estimation algorithm that follows their non-local approach. By making a smoothness assumption on layer colors, they again propose a sparse linear system for estimating layer colors and solve for them for each layer independently. We observed that independently solving for each layer fails to satisfy the color constraint in the final result, as shown in Figure 6. This is highly undesirable, especially for image editing applications because of the information loss on the original image prior to editing.

To summarize, the main advantages of our method are satisfying all the constraints and requirements that we discussed in Section 3, not requiring a pre-determined layer ordering, as well as providing smooth transitions between layers and its per-pixel formulation that

Fig. 7. Color model estimation is done by placing seed pixels (marked green on the top row) one by one and estimating a normal distribution (visualized as the hexagons) from the local neighborhood of each of the seeds. To select the next seed pixel, we make use of the representation scores (bottom row, brighter means better-represented). The pixels that are already marked as well-represented are highlighted with blue on the top row. The hexagonal visualization shows the color variation along the three principal axes of the covariance matrices of the estimated 3D color distributions in their diagonals.

permits efficient implementation and scalability to high-resolution images. In contrast, any other soft color segmentation method we analyzed in this section fails in at least one aspect. Scalability is especially crucial for practical applications, as in terms of both computation time and memory consumption, current methods fail to scale to resolutions captured by consumer-grade cameras. We will further discuss the visual implications of these shortcomings and required computational resources for each algorithm in Section 6.

5. COLOR MODEL ESTIMATION

In Section 3, we defined the *color model* as the set of layer color distributions $\mathcal{N}_i$ that represents the color characteristics of layers, which we then used as a fundamental component of our soft color segmentation formulation. In this section, we discuss details of our automatic color model extraction process, which we treated as a black box until now.

One priority we have in the estimation of the color model is determining the number of prominent colors N automatically. As our layers are meant to be used by artists in image manipulation applications, it is important to keep N at a manageable number. At the same time, to enable meaningful edits, the color model should be able to represent the majority—if not all—pixels of the input image. In order to strike a balance between *color homogeneity* and the number of layers N, we would like to avoid including colors that can already be well represented as a mixture of other main colors.

Figure 7 illustrates our color model estimation process. In order to determine how well a color model represents the input image, we define a per-pixel *representation score* r^p, which is the color unmixing energy obtained by minimization using the current (possibly incomplete) color model. The color unmixing energy can be used to assess the representativeness of an intermediate color model, since, if the model fails to fully represent a pixel color in the input image, then the color unmixing energy will be high due to the $\mathcal{D}(\boldsymbol{u}_i)$ term. By using the color unmixing energy (Equation (4) without the sparsity term) instead of only $\mathcal{D}(\boldsymbol{u}_i)$ terms, we make sure that the colors that can already be represented as a mixture of several existing colors are not added to the color model. This increases the compactness of the estimated color model while preserving the color homogeneity of the to-be-estimated layers.

Our first goal when estimating the color model is to determine a set of *seed pixels* with distinct and representative colors. For estimating the color model, we rely on a greedy iterative scheme, where we keep selecting additional seed pixels until we determine that the current color model is sufficiently representative of the whole image. To select the next seed pixel, we rely on a voting scheme. To that end, we first divide the RGB color space into $10 \times 10 \times 10$ bins. Every pixel votes for its own bin, and each vote is weighted by how well represented the pixel already is such that the most underrepresented pixels get the highest voting right. We finally select the seed pixel from the bin with the most votes. If no bin has a significant number of votes, then the algorithm terminates.

Mathematically, the vote of each pixel is computed as

$$v^p = e^{-\|\nabla \boldsymbol{c}^p\|} \left(1 - e^{-r^p}\right), \tag{10}$$

where $\nabla \boldsymbol{c}^p$ represent the image gradient, which is often a good indicator of image regions with mixed colors. Since we would rather like to select seed pixels with pure colors, the above expression penalizes the votes coming from high-gradient image regions.

After selecting which color bin to add to the color model, we choose the next seed pixel as follows:

$$s_i = \arg\max_{p \in \text{bin}} \mathcal{S}^p e^{-\|\nabla \boldsymbol{c}^p\|}, \tag{11}$$

where $\mathcal{S}^p$ is the number of pixels in the same bin as pixel p in its 20×20 neighborhood. We then place a guided filter kernel around p to use as weights in estimating a normal distribution from the neighborhood of the seed.

We use a *representation threshold* to determine which pixels are sufficiently represented and remove them from the voting pool:

$$\text{Remove } p \text{ if } r^p < \tau^2. \tag{12}$$

The representation threshold τ roughly indicates the number of standard deviations the color of a pixel can be away from the mean of a layer distribution to be considered as *well represented*. Smaller values of τ would produce larger color models that may be cumbersome for users manipulating images, but the resulting layers would be more homogeneous in terms of their color content. On the contrary, a larger τ would result in compact color models but possibly cause the layers to be less homogeneous. We show in Section 6.1

that although our algorithm requires τ as a parameter, fixing τ instead of the number of layers N generalizes well over different types of images. Through experimentation, we found $\tau = 5$ to be a good compromise between color model size and layer homogeneity and use this value to produce all our results.

As stated earlier, the color model is computed as a pre-processing step to the soft color segmentation algorithm detailed in Section 3. A computational challenge here is estimating the representation scores efficiently. Instead of running the computationally demanding nonlinear optimization scheme of color unmixing every time we add a new seed pixel to the color model, we approximate the color unmixing cost using the novel projected color unmixing as described in Section 5.1.

5.1 Approximating the Representation Score

From an implementation point of view, the main challenge with the aforementioned color model estimation scheme is the expensive cost of recomputing the color unmixing energy every time we add a new entry to the color model. The key observation that enables us to circumvent this challenge is that in order to estimate the representation scores r^p, we only need to know the color unmixing energy $\mathcal{F}$, but do not necessarily need to obtain the correct alpha and color values as a result of the minimization process.

We reformulate the representation score computation as:

$$\hat{r}^p = \min(\{\mathcal{D}_i(\boldsymbol{c}^p), \forall i\} \cup \{\hat{\mathcal{F}}_{i,j}(\boldsymbol{c}^p), \forall i, \forall j \neq i\}), \tag{13}$$

where $\hat{\mathcal{F}}_{i,j}(\boldsymbol{c}^p)$ represents an approximation of the minimized color unmixing energy using the ith and jth color distributions as input. The major simplifying assumption we make here is that the mixed colors mainly constitute major contributions from at most two distributions in the model. The first term in Equation (13) corresponds to colors that fit well with a single color, while the second represents the case for two-color mixtures. We approximate the two-color minimum color unmixing energy using the method we call *projected color unmixing*.

5.1.1 *Projected Color Unmixing.* The color line assumption [Ruzon and Tomasi 2000], that is, the unmixed layer colors and the observed pixel color should form a line in the RGB space, is a useful tool especially for sampling-based natural matting methods in the literature, such as Gastal and Oliveira [2010]. While this assumption can be used for the two-layer case, this simplification does not generalize to $N > 2$. In this section, we utilize the color line assumption and provide an approximation to the color unmixing energy [Aksoy et al. 2016] for the $N = 2$ case, which we illustrate in Figure 8.

Color unmixing samples colors from 3D normal distributions in color space. In order to make use of the color line assumption, we restrict the possible space of samples taken from each distribution to a 2D plane. For a pair of normal distributions $\mathcal{N}_1(\boldsymbol{\mu}_1, \Sigma_1)$ and $\mathcal{N}_2(\boldsymbol{\mu}_2, \Sigma_2)$, the plane of possible unmixed colors for the first layer is then defined by the normal vector $\boldsymbol{n} = \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$ and the point $\boldsymbol{\mu}_1$.

We determine the approximations $\hat{\boldsymbol{u}}_{\{1,2\}}$ to the unmixed colors $\boldsymbol{u}_{\{1,2\}}$ as projections of the observed color $\boldsymbol{c}$ to the two planes:

$$\hat{\boldsymbol{u}}_{\{1,2\}} = \boldsymbol{c} - \frac{(\boldsymbol{c} - \boldsymbol{\mu}_{\{1,2\}}) \cdot \boldsymbol{n}}{\boldsymbol{n} \cdot \boldsymbol{n}} \boldsymbol{n}. \tag{14}$$

The alpha values are then determined such that they satisfy the color constraint:

$$\hat{\alpha}_1 = \frac{\|\boldsymbol{c} - \boldsymbol{u}_2\|}{\|\boldsymbol{u}_1 - \boldsymbol{u}_2\|}, \quad \hat{\alpha}_2 = 1 - \hat{\alpha}_1. \tag{15}$$

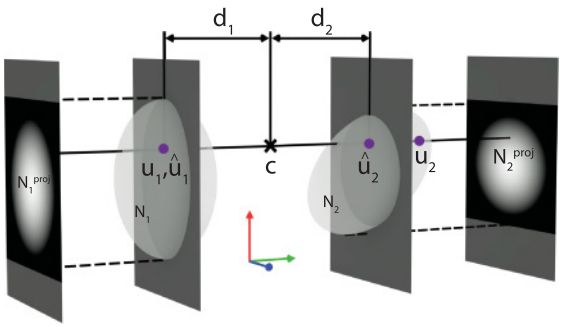

Fig. 8. An illustration of projected color unmixing. See text for discussion.

If $\boldsymbol{c}$ does not lie between the two planes, then we conclude that the pixel p can not be represented as a mixture of samples drawn from the two color distributions.

In order to compute the cost of this color mixture, we project the normal distributions onto the corresponding planes as well. We then apply the color unmixing energy formulation using these 2D distributions:

$$\hat{\mathcal{F}} = \hat{\alpha}_1 \mathcal{D}_1^{\text{proj}}(\hat{\boldsymbol{u}}_1) + \hat{\alpha}_2 \mathcal{D}_2^{\text{proj}}(\hat{\boldsymbol{u}}_2), \tag{16}$$

where we refer to $\hat{\mathcal{F}}$ as PCU energy.

If the two largest eigenvalues of the normal distribution are close to lying on the plane we defined above, then the estimated layer color is actually very close to the one found by color unmixing, as illustrated by distribution 1 in Figure 8. Otherwise, the approximation error is larger, as illustrated by distribution 2.

The approximation to the alpha values shown in Equation (15) is actually the same as the alpha estimation equation proposed by Chuang et al. [2001] and utilized by many sampling-based natural matting approaches,

$$\hat{\alpha}_1^{\mathcal{B}} = \frac{(\boldsymbol{c} - \boldsymbol{u}_2) \cdot (\boldsymbol{u}_1 - \boldsymbol{u}_2)}{\|\boldsymbol{u}_1 - \boldsymbol{u}_2\|^2}. \tag{17}$$

The cost of using a pair of samples is typically defined in relation to *chromatic distortion* [Gastal and Oliveira 2010],

$$\mathcal{C} = \|\boldsymbol{c} - \alpha_1 \boldsymbol{u}_1 - \alpha_2 \boldsymbol{u}_2\|, \tag{18}$$

which is basically the deviation from the color constraint. While both chromatic distortion and PCU measure the quality of the unmixing using the two samples/distributions, a significant difference between them is that PCU measures the cost when the color constraint is satisfied using the statistical models for the layers.

Experimentally, we found that the projected color unmixing energy gives us an approximation to the two-layer color unmixing energy by an error rate of 15% on average, while running approximately 3,000 times faster on a standard PC. By estimating the unmixing costs efficiently, our overall gain in total color model computation time is a 5-time improvement compared to using the color unmixing optimization procedure. On average, the time required to compute the color model for a 1MP image is 9s.

A step-by-step example of our color model computation procedure is presented in Figure 7. To summarize our color model computation, we add new layer color distributions to our color model by first computing the per-pixel representation scores efficiently using projected color unmixing (Equation (13)). We then group the image pixels into color space bins and execute a voting scheme where we give higher weights to pixels with lower representation scores. Within the bin with the highest votes (Equation (10)), we select

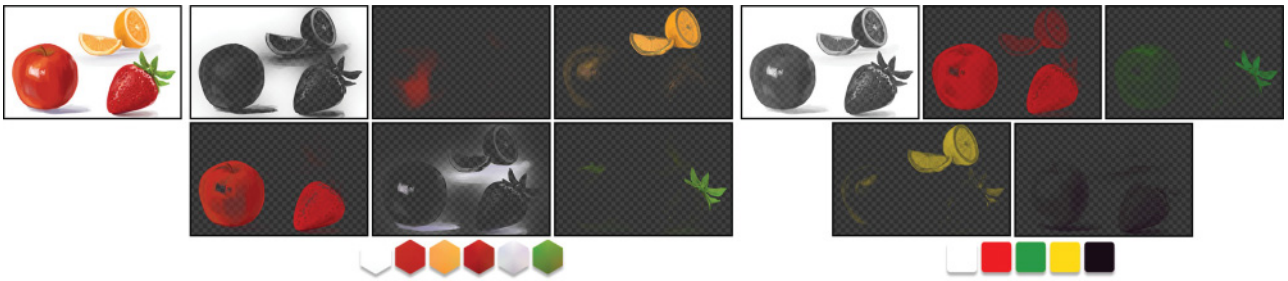

Fig. 9. The color model and corresponding layers computed by the proposed method (left) and by Tan et al. [2016]. The layers on the right have been converted to alpha-add representation from overlay representation (see the appendix) for a more meaningful comparison. The number of layers is different for the two algorithms because we are using the original result of Tan et al. [2016] which has 5 layers, and our automatic color model estimation method determined 6 dominant colors in the image. See text for discussion.

the seed pixel from an image region where the gradient magnitude is low (Equation (11)). We compute the parameters of a normal distribution from the neighborhood of the seed pixel and add this distribution to the color model. We remove the well-represented pixels (Equation (12)) and repeat the procedure until no bins have enough votes.

5.2 Color Model Estimation Methods in Literature

There are several methods to compute a color model given an image. We will discuss the common clustering methods such as k-means and Gaussian mixture models (GMM) as well as methods integrated with their soft color segmentation or color editing counterparts by Tai et al. [2007], Chang et al. [2015], and Tan et al. [2016].

Our method generalizes well over different types of images with a fixed parameter $\tau = 5$ as opposed to the rest of the algorithms, all of which need the number of layers N as the input parameter that change dramatically from image to image. In contrast, KNN [Chen et al. 2013], ML [Singaraju and Vidal 2011], and CU [Aksoy et al. 2016] rely on user input in forms of scribbles or seed pixels rather than a color model. RGBSG [Tan et al. 2016] requires the ordering of the layers as input in addition to N. It should be noted that there are generalized methods such as G-means [Hamerly and Elkan 2003] or PG-means [Feng and Hamerly 2006] for automatically estimating the number of clusters in arbitrary data sets. However, they do not take into account the specific characteristics of estimating the number of color layers, such as mixed-color or high-gradient regions.

The color model, referred to as the *palette* by Chang et al. [2015], is determined by a modified version of the k-means for the palette-based recoloring application [Chang et al. 2015]. They simplify the problem by using color bins rather than all of the pixel colors in the image and disregarding dark color entries.

Clustering methods such as k-means or expectation maximization for GMM, as well as the palette estimation by Chang et al. [2015], tend to produce layer color distributions with means far away from the edges of the RGB cube. This often results in under-representation of very bright or highly saturated colors in the color model.

On the other hand, the color model estimation of AO, as well as GMM, typically results in normal distributions with high covariances, which has an adverse effect on the color homogeneity of the resulting layers. This is due to the lower energy achieved with large covariances in the expectation maximization for GMM. In the case of AO, they estimate the parameters using the layer colors at a particular iteration of their algorithm. This causes their algorithm to begin with large covariances as in the first iteration they assume opaque pixels after an initial clustering, and this inclusion of unmixed colors in the model estimation causes large color variation in each layer. Large covariances promote non-uniform layers and as a result, further iterations do not tend to make color distributions more compact.

RGBSG begin their model estimation by assuming that the input image is formed by a limited set of colors. This assumption does not generalize well to natural images as they may include much more complex color schemes and mixtures. Their alpha estimation method requires the model colors to envelop the convex hull formed by the pixel colors in the image, and hence they identify the model colors by simplifying the wrapping of the hull. This results in selection of colors that are either on the edge or outside the convex hull. Hence, the colors picked by them do not necessarily exist in the image, and a dominant color that is in the middle of the RGB cube cannot be selected as a model color. The effect of this can be observed in Figure 9, where orange were not included by their color model and instead yellow, which does not appear anywhere in the image, is selected. The orange pixels are then represented by the mixture of red and yellow. Tan et al. [2016] points out that this behavior allows the algorithm to discover *hidden colors* in the image. However, it is suboptimal for image editing in various scenarios as when the edited color does not exist in the image, the effects of the edit becomes hard to predict. In addition, the simplified wrapping of the hull does not necessarily stay inside the valid color values. They map these imaginary colors onto the acceptable range and this may result in their alpha estimation not satisfying the color constraint.

Our method, in contrast to some of the competing approaches, exclusively selects colors that exist in the image. By constructing the model through selected seed pixels instead of clustering, we can include highly saturated or bright colors.

While each approach has advantages and disadvantages, an important thing that should be noted is that the integrated approaches in RGBSG, AO, and ours have characteristics coupled with their corresponding layer estimation methods. RGBSG requires colors that envelop the pixels, and AO requires an iterative approach to refine the layers together with the model. Our approach constructs the color model using the unmixing energy, which results in a model that is able to represent the pixel colors with low unmixing energy, which in turn makes our estimated layers have smaller color variation. We evaluate our automatic technique in Section 6.1.

6. EVALUATION

Evaluating soft color segmentation methods is challenging, since how the optimal set of layers resulting from decomposing a particular input image should exactly look like is not clear. In fact,

Table I. Quantitative Test Results for ML [Singaraju and Vidal 2011], SM [Levin et al. 2008b], KNN [Chen et al. 2013], AO [Tai et al. 2007], CU [Aksoy et al. 2016] and the Proposed Algorithm. *Italic* Font Indicates Values Below Quantization Limit

	SM	*ML*	*KNN*	*AO*	*CU*	*Ours*
$\alpha \notin [0, 1]$	34%	30%	0.0001%	0%	0%	0%
Rec. Error	0.0039	0.0138	0.0183	*0.00002*	*0.0003*	*0.0005*
Color Var.	0.050	0.044	0.006	0.051	0.001	0.005
Grad. Corr.	0.78	0.71	0.84	0.55	0.56	0.89

even the existence of such a ground-truth soft color segmentation is questionable at best. Accordingly, in this section, we present two types of evaluation. As qualitative evaluation, we present layers that are computed by our method in comparison with previous methods in Figure 12 and in the supplementary material for visual inspection by the reader. These results provide insights on the overall quality level of each algorithm and serve as a visual reference on the characteristic artifacts each method produces. For quantitative evaluation, we devise a set of blind metrics for assessing how well each method satisfies the constraints and requirements that we discussed in Section 3. These metrics are as follows:

—**Out-of-bounds alpha values:** Check if the box constraint (Equation (3)) is satisfied for alpha values. The metric returns the percentage of alpha values outside the permissible range.
—**Reconstruction error:** Check if the color constraint (Equation (1)) is satisfied. The metric computes the average squared distance between an input image and the alpha-weighted sum of all its layers.
—**Color variance:** Assess the color homogeneity of layers. The metric returns the sum of individual variances of RGB channels averaged over all layers of an input image.
—**Gradient correlation:** Assess whether the texture content of the individual layers agrees with that of the input image. Inconsistencies such as abrupt edges in a layer, which are not visibly identifiable in the original input image will result in a low score. The metric returns the correlation coefficient between the alpha channel gradients of the layers and the color gradients of the original image. We define the alpha channel gradient as the L2 norm of the vector that comprises the gradients of alpha values of every layer. Similarly, we compute the L2 norm of the gradient of each color channel as the color gradient.

Note that the above metrics only evaluate constraints and requirements that are not satisfied by at least one method. For example, we excluded the box constraint for color (Equation (3)), since the particular implementations of all the methods we consider in our evaluation produce color values within the permissible range. We executed the above metrics on a test set of 100 diverse images and report the average numbers in Table I.[1]

We also analyzed the runtime and memory requirements of each method both as a function of the color model size (Figure 10) and image resolution (Figure 11). For the former test, we selected eight images at 1MP resolution with varying color model sizes from 4 to 11. In the latter test, where we investigate the scalability of each method in terms of image size, we chose an image with seven layers,[2] which we resized from 5MP down to 40kP. It should be

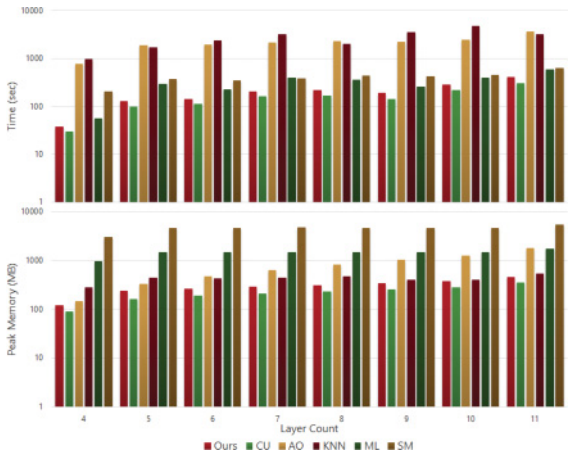

Fig. 10. Computational resources needed for soft color segmentation with respect to the number of layers. We selected example images corresponding layer count to see the trend in each algorithm. Note that the data axes are in logarithmic scale.

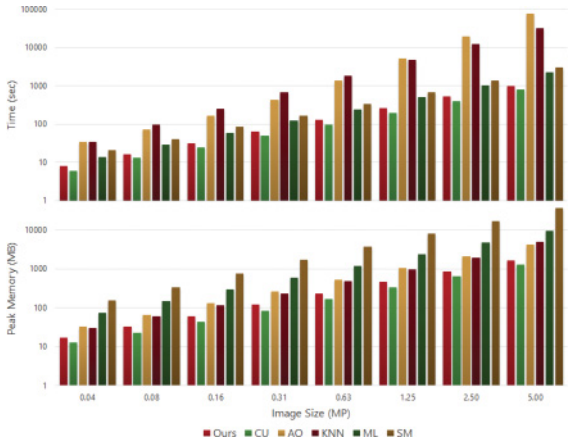

Fig. 11. Computational resources needed for soft color segmentation with respect to the image size. We scaled an image with 7 layers to see the trend in each algorithm. Note that the data axes are in logarithmic scale.

pointed out that our parallelized C++ application is compared to the MATLAB implementations of the competing methods except for CU. KNN, SM, and ML solve large linear systems which cannot be efficiently parallelized. However, these graphs still give us an idea of the scalability of each method.

In our evaluation, we used publicly available implementations of SM and KNN, whereas for CU, we use the original source code. We implemented ML using Levin et al.'s [2008a] publicly available matting Laplacian and foreground layer estimation implementation. We utilized the latter also for computing the layer colors of SM. As AO's implementation was not available,[3] we implemented the method ourselves in MATLAB, where we utilized the UGM toolbox for the loopy belief propagation [Schmidt 2007]. Our own results were generated using our research prototype written in C++ with parallelization using OpenMP.

[1] All images and the corresponding layers produced by six methods shown in the figure are presented in the supplementary material.
[2] The average size of the color model was 6.88 for the 100 images on which we do the evaluation.
[3] On correspondence with the first author, we learned that the original code is no longer available.

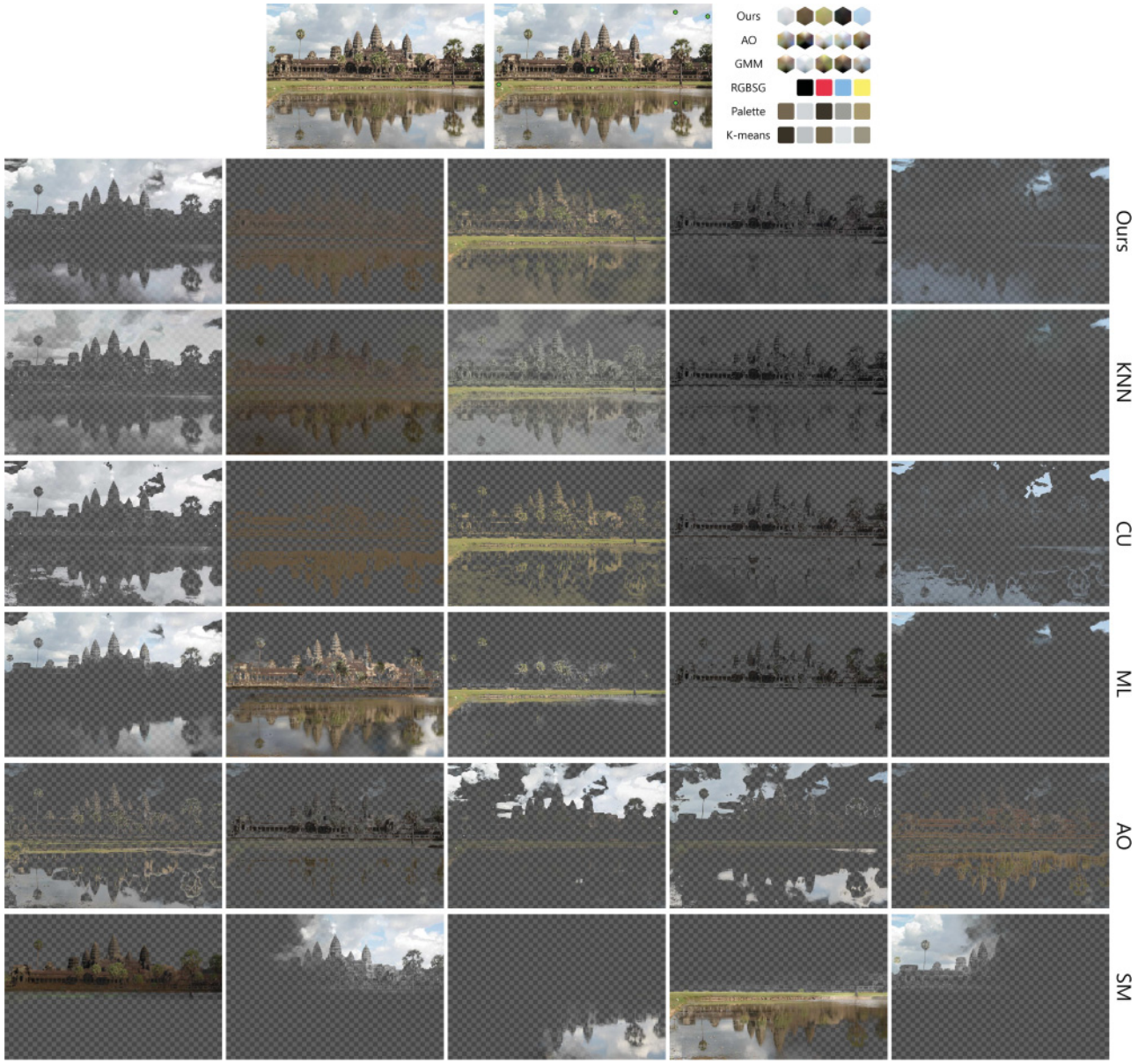

Fig. 12. Comparison of the soft color segments produced by various algorithms including ours. Results of all the algorithms shown here are provided for 100 different images in the supplementary material. See text for discussion.

SM and AO require the number of layers N *a priori* as input, whereas KNN obtain N seed pixels through user interaction, CU requires N scribbles, and ML similarly requires N user-specified regions. In order to enable a meaningful comparison among all competing methods, we utilized the number N, as well as the N seed points determined by our color model (Section 5). As the input to ML, we used N image regions, each comprising pixels with similar colors to a seed pixel. In order to avoid biasing the comparison by artistic talent, we excluded the local color models from CU. It should also be noted that SM, by design, produces spatially connected soft layers rather than emphasizing color homogeneity. Nevertheless, for completeness, we include SM in our evaluation.

A representative qualitative comparison is presented in Figure 12, where we show the layers produced by all competing methods.[4] The figure clearly illustrates the shortcomings of propagation based methods SM and ML. For example in ML's case, the sky and its reflection on the lake end up in two separate layers, despite having similar colors. The quantitative results for both methods, in agreement with our theoretical analysis (Section 4), indicate that they, in fact, suffer from alpha values outside the permissible range. Their

[4]See the supplementary material for many other examples and further discussion.

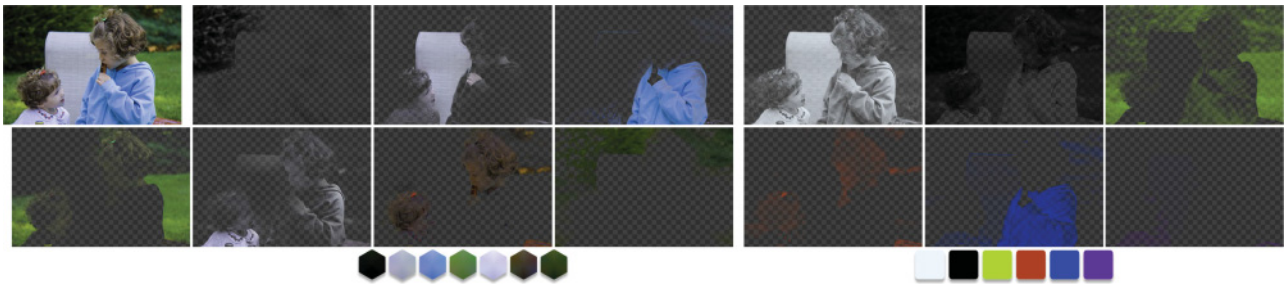

Fig. 13. The soft color segments produced by the proposed algorithm (left) and by the method by Tan et al. [2016] (right). The layers by Tan et al. [2016] have been converted to alpha-add representation from overlay representation (see the appendix) for a more meaningful comparison. An extension of this figure is available in the supplementary material. See text for discussion.

results on the remaining metrics reveal that they are generally worse than others for the task of soft color segmentation (Table I).

On the other hand, both Figure 12 and Table I confirm our claims from Sections 4 and 5 on AO's potential color homogeneity issues and the consequences of solving for alpha and color values separately. The figure clearly shows spurious hard edges on AO's layers, and the method performs badly on both color variance and gradient correlation metrics.

While KNN does not share AO's weaknesses, it instead tends to fail to satisfy the color constraint for reasons discussed in Section 4. Another significant drawback of KNN is its prohibitively long runtime, which prevents the method to be used on high-resolution images on current PC hardware.

The layers computed by CU in Figure 12 clearly show the visual effect of the absence of spatial coherency, which manifests itself as spurious hard edges throughout the layers. This problem is also evident from the gradient correlation metric outcomes in Table I, where CU gets the worst score among all the algorithms we evaluated.

We could not include RGBSG [Tan et al. 2016] in these comparisons because it requires the user to define the ordering of the layers. The quality of its layers depend on this ordering and hence we could not use a random ordering to determine its true performance. This ordering is rather hard to define prior to soft color segmentation even by a user and the need for it is a significant shortcoming. Instead, we compare our layers against RGBSG using the layers provided by Tan et al. [2016] as supplementary material to their article in Figure 13.[5]

RGBSG provides opacity layers with smooth transitions as we do, as opposed to other algorithms we analyze in this section. Also, their layer colors are defined to be solid, which makes their color variation score 0. However, their formulation and model estimation requires the model colors to envelop the pixel colors of the original image from the outside in RGB space, which makes the color content of their layers differ from the dominant colors in the image. This behavior, when coupled with the solid color layers, results in reduced sparsity in their layers. Figure 13 demonstrates this particular shortcoming. While the girl's hoodie has a particular shade of the blue, it was not included in the color model of RGBSG. As a result, that region also has strong green and white components in addition to blue. Also, the purple color does not exist in the original image but is included in the model of RGBSG and computed as an additional layer. By containing a subset of actual image colors, our automatically computed layers provide an intermediate image representation that is more intuitive to use.

To summarize, all current soft color segmentation methods suffer from one or more significant drawback(s). Our method, on the other hand, successfully satisfies the alpha, box, and color constraints and produces layers with homogeneous colors. The transitions in between layers of our method are highly correlated with the texture of the original image. Importantly, due to its highly parallelizable per-pixel formulation, our method is more memory efficient and runs more than 20× faster than KNN and RGBSG, which are the closest competition in terms of the quality of results. The proposed algorithm can process a 100MP image in 4h using up to 25GB of memory. Note that KNN and AO can only process a 2.5MP image within the same time budget, and SM requires more than 25GB of memory for a 5MP image. Also, note that our modifications to the color unmixing formulation by Aksoy et al. [2016] cause only a modest performance hit, while significantly improving the quality of the layers.

6.1 Color Model Estimation

We also compare our color model estimation method (Section 5) with other methods, such as the specialized color model estimation schemes from AO, RGBSG, and palette-based recoloring [Chang et al. 2015], as well as general-purpose clustering methods k-means and expectation maximization for GMM.

Figure 14 shows exemplary results for discussion, whereas the full set color models for 100 images is presented in the supplemental material. These results demonstrate several advantages of our color model estimation method over others.

First, Figure 14 clearly demonstrates that the number of main colors N in an image (and thus the desired cardinality of the color model) is highly content dependent. However, all current methods require N to be specified *a priori*. Figure 14 shows that any fixed number will be overkill for certain images, whereas being too restrictive for others. Our method is highly advantageous in this regard, as it determines N automatically by analyzing the image content using a fixed parameter $\tau = 5$. Note that to enable a meaningful comparison, we set N for the competing methods to the value determined by our method.

One characteristic of clustering-based methods is the lack of highly saturated or bright colors in the estimated model. This is because, as they form clusters, the centers tend to be far from the edges of the RGB cube to get a lower clustering energy. By sampling colors directly from the image using our voting-based scheme, we are able to get the colors as they appear in the image. This behavior is especially apparent in the last image in Figure 14,

[5]Our layers for the 18 images used by Tan et al. [2016] are provided as supplementary material.

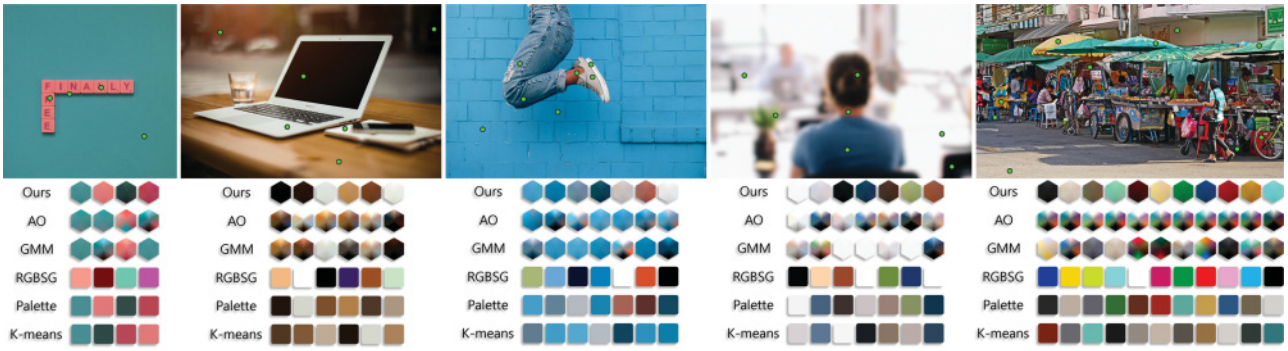

Fig. 14. Comparison of the color models estimated by our method, alternating optimization [Tai et al. 2007], expectation maximization using Gaussian mixture models (GMM), RGB-hull based method used in RGBSG [Tan et al. 2016], palette estimation used in palette-based recoloring [Chang et al. 2015], and K-means clustering. The green dots on the images denote seed points determined by the proposed algorithm. The hexagonal visualization shows the color variation along the three principal axes of the covariance matrices of the estimated 3D color distributions in their diagonals. The square visualization shows a single solid color, which is used for methods that only estimate colors rather than distributions. See text for discussion.

where k-means and palette-based recoloring fails to capture the vivid colors in the image.

RGBSG color model estimation, on the other hand, selects colors that lie outside the convex hull formed by the pixel colors in the image. This results in many colors appearing in the color model that do not exist anywhere in the image, such as the bright green entries in the second and third examples in Figure 14.

Another advantage of our color model estimation is its tendency to produce homogeneous color distributions, which directly influences the color homogeneity of the resulting layers computed by the soft color segmentation method. The hexagonal visualizations of each method's estimated color distributions in Figure 14 reveal that AO and GMM produce noticeably more color variation in each color model entry compared to our method.

Our method can also capture distinct colors confined in small image regions, such as the skin color in the middle image in Figure 14, or the green of the plant in the fourth image in the figure, which are missed by all other methods except palette-based recoloring. Since such regions usually form small clusters in color space, k-means and GMM tend to merge them into larger clusters.

While the competing methods have particular disadvantages, we refer the reader to Section 5.2 for the discussion on how and why the specialized model estimation methods in AO and RGBSG, as well as ours, fit well with their corresponding layer estimation counterparts.

7. APPLICATIONS

A key advantage of our framework is that it allows numerous, seemingly unrelated, image manipulation applications to be executed trivially, once an input image is decomposed into a set of layers. For example, color editing can simply be performed by translating the colors of one or more layers, green-screen keying amounts to removing the layer(s) corresponding to the background, texture overlay can be achieved by introducing new texture layers obtained from other images, and so on. In this section, we show high-quality results for different image manipulation applications that were produced by first letting our method do the heavy lifting and then applying a small set of basic per-layer operations.

Our method naturally integrates into the layer-based workflow adopted by the majority of the current image manipulation packages. In Figure 15 (bottom) we list some basic image manipulation operations that are implemented in image manipulation software packages. In practice, users can easily import the layers computed automatically using our method into their preferred software package and perform these edits through familiar tools and interfaces. This way, we prevent any unnecessary learning effort, as well as take full advantage of the powerful layer-based editing tools available.

We produced our application results by first exporting the layers computed automatically by our method to Adobe Photoshop and then performing a set of operations on individual layers.

We present our results in Figures 15 and 16, where input images and the corresponding automatically estimated color models are shown on the top, and edited images are shown at the bottom, along with the list of image manipulation operations applied to obtain the presented results. These image manipulation operations are denoted by small icons underneath the edited images, and the corresponding legend is presented at the bottom of Figure 15. For example, Figure 15 (Image (3)) is obtained by changing brightness, colors, and saturation on certain layers, whereas in Figure 15 (Image (8)) we only applied color changes to a subset of the layers. For brevity, in the remainder of this section we will refer to the specific results presented in Figures 15 and 16 solely by their designated indices.

In the following paragraphs, we discuss numerous examples of image manipulation applications and highlight our corresponding results. These applications can be categorized into layer adjustments, where we modify properties of existing layers, and compositing, where we add new layers to an image or remove existing ones. Note that some of our results contain specific manipulations that can be classified under more than one application, which can easily be performed under a single framework using our method.

7.1 Layer Adjustments

The users can easily enhance or even completely change image colors as well as adjusting brightness and exposure on a per-layer basis. Note that while performing such edits globally throughout the entire image is trivial, making local adjustments is often challenging and prone to producing visual artifacts.

Color enhancement/change: A number of effects can be achieved by simply using the hue/saturation tools available in most image manipulation software, as well as the more sophisticated color adjustment tools (such as the vibrance control in Photoshop)

Fig. 15. Example results that were generated using the layers provided by the proposed algorithm. Each set shows the input image (top), the color model, the edited image and the set of operations applied to individual layers. The set of operations is defined on the bottom. See text for discussion.

on selected layers of an input image. We showcase several examples, including changing colors of clothing (Images (1), (4), (6), (8), (13), and (18)), fauna (Image (3)), illumination (Images (2) and (7)), sky (Images (16) and (17)), fire (Image (17)), and metallic surfaces (Image (15)).

Brightness/exposure change: The brightness and exposure of specific layers can similarly be modified to make slight adjustments in the overall image appearance. Such subtle edits are performed in most of our results presented in Figures 15 and 16. Additionally, in Image (16) and Figure 1(c), we demonstrate a local enhancement of luminance contrast of the clouds, which results in certain details becoming visible and giving the impression of more volume compared to the input images. A particularly interesting image manipulation is showcased in Image (19), where we increase the intensity of shadows to facilitate establishing the skater as the center of attention in the composition.

Skin tone correction: Adjusting skin tones is one of the most common photo editing operations applied in practice. Since humans are usually the most salient scene elements and our visual system is highly tuned for detecting any imperfections, especially on faces, even the slightest visual artifacts caused by re-adjusting skin colors can be disturbing for the viewer. Our results (Images (1), (4), (6), and (18)) show examples of modified skin tones, which we achieved by making the mid-tones richer in the corresponding layer using the curve tool. The highlights on the faces can also be edited to be weaker (Images (1) and (4)).

7.2 Compositing

Our layers also serve as a useful intermediate image representation for general-purpose compositing applications.

Green-screen keying: Images (11), (12), (13), and (14) show our green-screen keying results obtained automatically by simply removing the layers corresponding to the green screen. Note that any other background object on the green-screen (such as the markers in Image (14)) can easily be removed using a standard garbage matte.

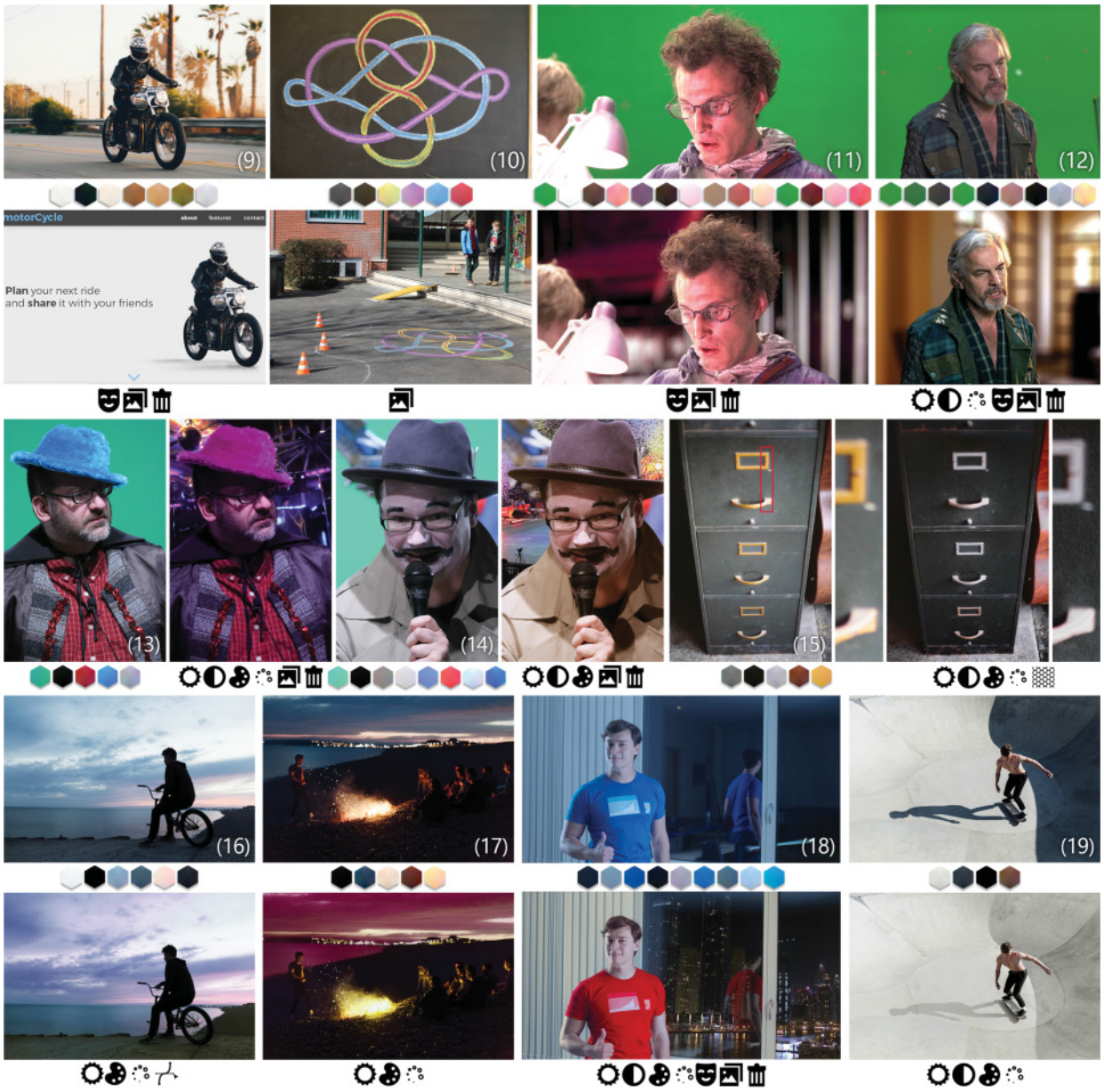

Fig. 16. Example results continued. See text for discussion.

We additionally present a video result for green-screen keying in the supplementary material.

Texture overlay: Our method also allows additional layers to be overlaid onto existing ones. Image (10) shows such an example where we extract the drawing from the blackboard on the source image and overlay it on a new image only by applying a perspective transform to the corresponding layers. Note that, due to their accurate opacity channels, the transferred layers properly mix with the texture of the concrete ground and shadows. In Image (15) we make a file cabinet appear more rugged by overlaying a texture pattern.

Layer replacement: Our method can also be used for removing existing layers and adding new ones, even if the content is not captured in a green-screen setting. Image (9) shows an example where we extract the object from the background with the help of roto masks and use it to create a web page. Note that the details, such as the shadow of the motorcycle, are retained in the composited result with proper opacity. Finally, in Image (18), we completely replace the original background with an external image while properly retaining the reflection on the window.

Our results demonstrate that state-of-the-art quality can be achieved using our soft color segmentation as an intermediate image representation, which in turn trivializes numerous image manipulation applications. This suggests that soft color segmentation is, in fact, a fundamental technical problem in image manipulation. By isolating this problem and solving it effectively and efficiently, our method serves as a unified framework for high-quality image

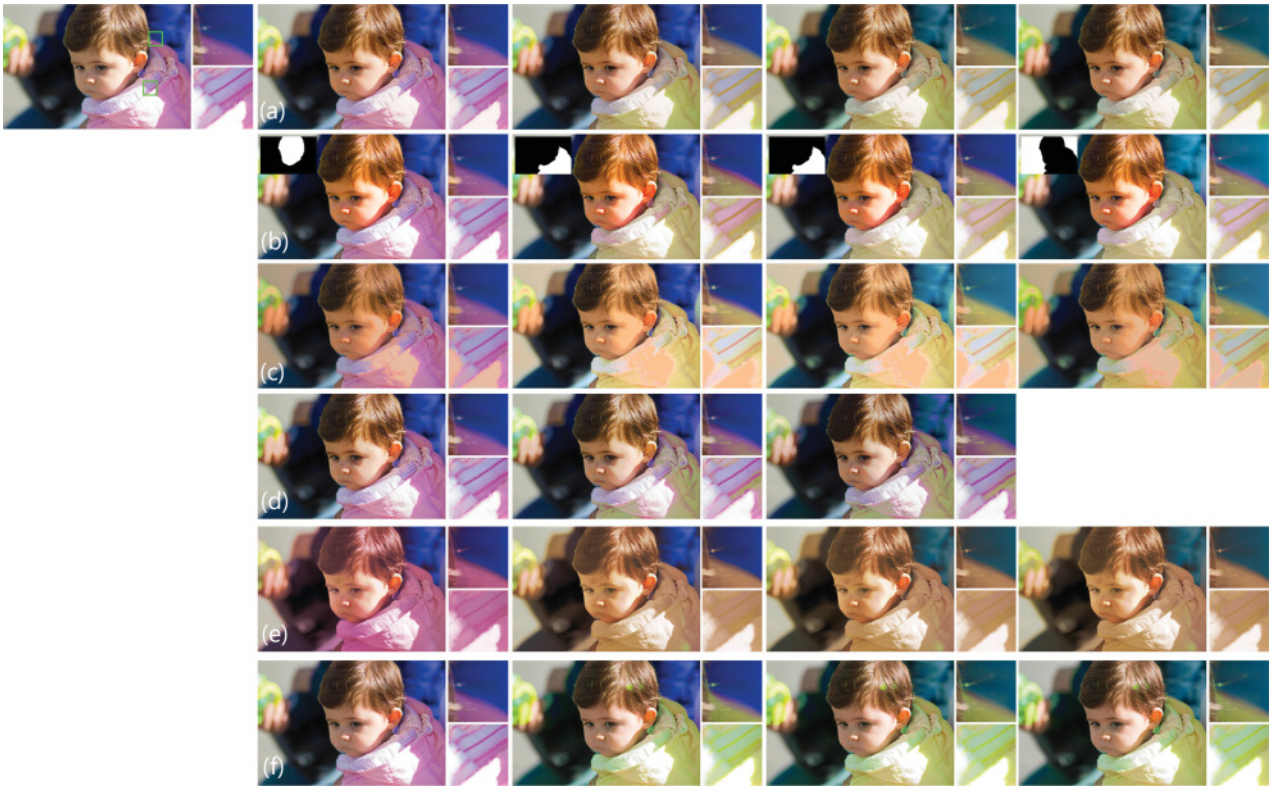

Fig. 17. Step-by-step editing of the image on the left in Adobe Photoshop using our layers (a), by a professional artist using only Adobe Photoshop (b), using the palette-based image editing tool by Chang et al. [2015] (c), and the layers computed by AO [Tai et al. 2007] (d), KNN [Chen et al. 2013] (e), and RGBSG [Tan et al. 2016] (f). AO and RGBSG rows lack one step each (the color of the sunset for AO and the color of the sea for RGBSG) because the layers corresponding to the intended edits did not exist in their color model. See text for discussion.

manipulation, which gives users significant flexibility for realizing their artistic vision.

8. COMPARISONS AT THE APPLICATION LEVEL

We demonstrated the use of our soft color segments in image editing in Section 7. Theoretically, any soft color segmentation method could produce layers that can be used for the same applications. We analyzed in Section 6 the differences between the current state of the art in soft color segmentation and the proposed method. In this section, we will demonstrate how these differences affect image editing results. We will also present the results of professional artists using commercially available tools for some of the demonstrated applications.

Figure 17 shows an image being edited by using our layers, layers by AO, KNN, and RGBSG, as well as those by a professional artist using Adobe Photoshop and using the palette-based recoloring application by Chang et al. [2015]. Creating a mask in Photoshop for targeted color edit results in color artifacts when the artist attempts to change the pink of the coat to yellow. The soft color transitions and fine structures are the main source of the problem since the layer masks do not include *unmixing* of the colors and the color changes result in suboptimal color transitions with the neighboring regions. The feedback we got from the artist was that very precise masks should be drawn at the pixel level manually and targeted color edits should be done to make the transitions look more natural, which is a very time-consuming and error-prone task. The fine structures and transitions are dealt with better by the palette-based recoloring. However, in this case, we see significant artifacts around bright regions. Using the layers by AO also creates significant visual artifacts, as the pink layer could not cover the full extent of the color. The layers by KNN do not give the original image and this results in a degraded version of the image even without editing. The pink layer covers areas in around the hair and in the background in RGBSG layers, which results in some artifacts in the edited result. It can also be observed that the transition from pink to blue of RGBSG layers is suboptimal, and some pinkish hue can be observed in this region after the color change.

We present color editing results using our layers in images used by Tan et al. [2016] (RGBSG) and Chang et al. [2015] (PBR) in Figure 18. We observed some matte smoothness issues in the results by RGBSG as apparent in the top two images. The color change from blue to pink in the bottom example results in artifacts around the soft transition from the chair to the background. The luminance constancy constraint of PBR, which states that the luminance of a particular palette color should always be lower than other palette colors that are originally brighter in the non-edited palette, results in overexposed regions when one attemps to increase the luminance of the sky in the top example. Also, the orange hue in the boat example still exists in the ground after the color change from orange

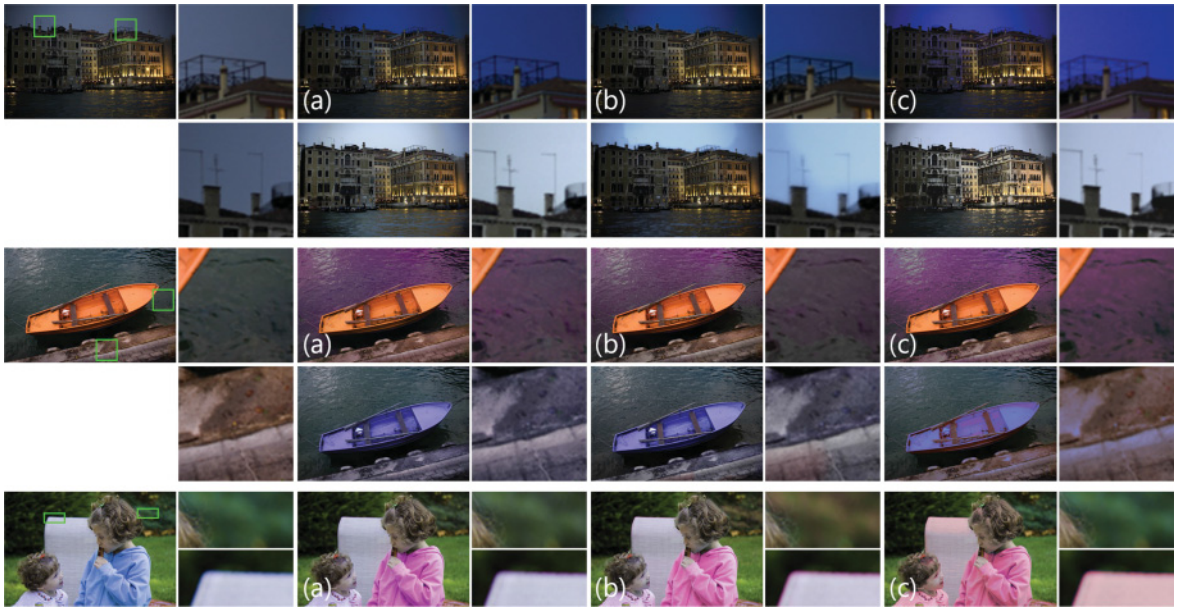

Fig. 18. Color editing results using our layers (a), layers by Tan et al. [2016] (b) and using the recoloring application by Chang et al. [2015] (c) on images used by Tan et al. [2016] and Chang et al. [2015] in their papers. An extension of this figure is available in the supplementary material. See text for discussion.

Fig. 19. Comparison to Aksoy et al. [2016] with manually edited active colors (a), and the same method with all colors activated at all pixels (b). Their results without the manual active color selection show severe visual artifacts. On the other hand, the proposed method (c) achieves comparable quality to (a) despite the lack of any user interaction. Three images on the right show work of an independent professional artist using multiple commercially available tools (d), only Keylight (e) and only IBK (f) as reported by Aksoy et al. [2016].

to purple, alongside with other artifacts around the boat itself. The color change in the bottom example erroneously affects the color of the chair completely when PBR is used.

In practice, green-screen keying depends heavily on user interaction. The common practice in industry requires a specialized compositing artist to combine multiple commercially available tools such as Keylight, IBK, or Primatte to get the best possible result. The state of the art in the academic literature [Aksoy et al. 2016] also requires multiple user interaction stages, although it decreases the total interaction time significantly. We compare our results with those of Aksoy et al. [2016] with user interaction, their results without user interaction using only the color unmixing proposed in the same article [Aksoy et al. 2016], as well as with the work of a professional compositing artist that were provided in their article in Figure 19. It can be observed in both examples in Figure 19 that we are able to conserve intricate details of the foreground object while requiring only a simple mask to get rid of the markers, and so on, in the background to clean the matte in a post-process. Note that this mask is only one of the steps of interaction that is required in commercially available tools. Unlike the proposed method, using color unmixing without the interaction steps described by Aksoy et al. [2016] results in disturbing artifacts.

In summary, while given enough time our results could potentially be reproduced by a skilled artist utilizing various specialized tools for each task, our method in general significantly reduces the manual labor required for achieving production-quality results, and provides artists a unified framework for performing various image manipulation tasks conveniently.

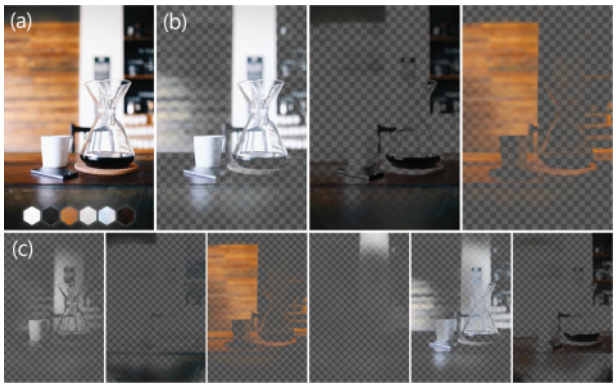

Fig. 20. In some cases, the proposed color model estimation algorithm may give more layers than the user intends to make use of, as seen in the bottom row (c). However, these layers can easily be combined to be edited together, an example of which is shown in the top row (b) with the original image and the color model (a).

Fig. 21. Although our automatic method is not able to deal with color spill (c) as well as [Aksoy et al. 2016] (b), it is possible to use commercial keying software such as Keylight to get rid of the spill as a post-processing step (d).

9. LIMITATIONS

In some cases, the automatically estimated color model comprises color distributions that are subjectively similar, such as the three separate white layers and the separate dark brown/black layers in Figure 20(c). While this level of granularity might be useful, a more concise color model would be more convenient for certain edits. Fortunately, combining multiple layers into one is trivial in our framework (Figure 20(b)). It is worth noting that the transitions between layers with perceptually similar color distributions are still smooth and have accurate alpha values. Therefore, they can still be edited separately if the user intends to do so.

Our method has a significant advantage over the state of the art in green-screen keying [Aksoy et al. 2016] in that it completely replaces the cumbersome two-step user interaction process with a fully automatic workflow without sacrificing the quality of the results in general. That said, our sparse color unmixing energy formulation (Section 3), which is designed for general-purpose soft color segmentation, is not as effective in dealing with color spill (indirect illumination from the green-screen) as the method by Aksoy et al. [2016]. Fortunately, we found that this problem can easily be alleviated in practice, as off-the-shelf tools can effectively remove spill even in challenging cases, such as Figure 21, after only a few mouse clicks.

The color model estimation method we propose selects seed pixels from the image and hence comprises only colors that exist in the image. This strategy is effective for including colors with high brightness or saturation when compared to the clustering-based methods as discussed in Section 6.1. However, one limitation is that it cannot identify colors that only appear as a mixture in the image, such as the original color of a colored glass or smoke that is not dense. For estimating such a color that does not appear opaque anywhere in the image, a different specialized approach is needed that would estimate the partial contributions from an existing incomplete color model in order to isolate the missing color.

Finally, as our soft color segmentation method does not utilize high-level information such as semantic segmentation or face detectors, the spatial extent of our soft segments do not necessarily overlap with semantically meaningful object boundaries. That said, it is also fairly easy to limit the spatial extent of layers by introducing masks since our layers naturally integrate into current image manipulation software.

10. CONCLUSION

We proposed a novel soft color segmentation method that is especially suitable for image manipulation applications. We introduced SCU, which gives us compact preliminary layers. We discussed the challenges of enforcing spatial coherence and proposed a color refinement step that prevents visual artifacts. We also proposed a method for automatically estimating the color model that is required for solving the color unmixing problem, which completely removes the need for user input. A key component in our color model estimation is projected color unmixing, which enables efficient computation of representation scores we need for compact color models. We presented a theoretical reasoning as well as quantitative and qualitative evaluations showing that our results are superior to previous work in soft color segmentation. We also demonstrated state-of-the-art results in various image manipulation applications. The future directions for our research include addressing the limitations of our method discussed in Section 9, as well as extending our per-frame method to exploit temporal information in order to natively handle image sequences.

APPENDIX

A. *ALPHA-ADD* AND *OVERLAY* LAYER REPRESENTATIONS

For handling layers, our blending formulation in Equation (1) corresponds to the *alpha add* mode present in Adobe After Effects. The *normal* blending option in Photoshop differs slightly from ours, which is defined for two layers as follows:

$$\boldsymbol{u}_o = \frac{\tilde{\alpha}_a \boldsymbol{u}_a + \tilde{\alpha}_b \boldsymbol{u}_b \left(1 - \tilde{\alpha}_a\right)}{\tilde{\alpha}_a + \tilde{\alpha}_b (1 - \tilde{\alpha}_a)}, \tag{19}$$

$$\tilde{\alpha}_o = \tilde{\alpha}_a + \tilde{\alpha}_b \left(1 - \tilde{\alpha}_a\right), \tag{20}$$

where $\boldsymbol{u}_o$ and $\tilde{\alpha}_o$ define the overlaid result with Photoshop-adjusted alpha value, $\boldsymbol{u}_a$ and $\tilde{\alpha}_a$ define the top layer, and $\boldsymbol{u}_b$ and $\tilde{\alpha}_b$ define the bottom layer. We refer to the layers following this representation as *overlay* layers with alpha values denoted by $\tilde{\alpha}$, in opposition to the representation used in our formulation, to which we refer as *alpha-add* layers with alpha values α. Unlike the alpha-add representation where the ordering of the layers is irrelevant, overlay layers depend on a pre-defined layer order.

Assuming that the layers form an opaque image when overlaid, given the layer order from 1 to N, with the Nth layer being at the

top, one can convert alpha-add layers to overlay layers as follows:

$$\tilde{\alpha}_n = \begin{cases} \frac{\alpha_n}{\sum_{i=1}^{n} \alpha_i} & \text{if } \sum_{i=1}^{n} \alpha_i > 0 \\ 0 & \text{if } \sum_{i=1}^{n} \alpha_i = 0 \end{cases}, \quad n \in \{1 \dots N\}. \tag{21}$$

Since some regions are completely occluded by the layers on top, the alpha values assigned to them are arbitrary, although we defined $\tilde{\alpha}_n = 0$ when $\sum_{i=1}^{n} \alpha_i = 0$. If the artist intends to remove some of the layers during editing for compositing applications as we demonstrate in Section 7, then those layers should be placed at the bottom before the conversion.

Similarly, the overlay layers can be converted to alpha-add layers using the following formulation:

$$\alpha_n = \tilde{\alpha}_n \left(1 - \sum_{i=n+1}^{N} \alpha_i\right), \quad n \in \{1 \dots N\}. \tag{22}$$

Note that the layer colors $\boldsymbol{u}_i$ are not affected by these conversions.

ACKNOWLEDGMENTS

We thank Alessia Marra for discussions on possible applications and her help in result generation; Jean-Charles Bazin, Henning Zimmer, Jordi Pont-Tuset, and the anonymous reviewers for their feedback on the text; Yu-Wing Tai and Jianchao Tan for our correspondence about their publications; and Andrew Speilberg for the voice-over of the accompanying video.

The original images in Figures 1, 2, 3, 5, 14 (Images (1), (3), and (4)), 16 (Images (9), (15), (16), (17), and (19)), and 20 are from Death to the Stock Photo; in Figure 4 from Flickr user Marcelo Quinan; in Figure 6 (top) from Flickr user Jason Bolonski; in Figure 15 (Image (2)) from Flickr user Nana B Agyei; in Figure 15 (3) from Flickr user taymtaym; in Figure 15 (Image (7)) from Flickr user Paul Bica; in Figure 16 (Images (10), (13), and (14)) by Flickr user Dave Pape; and in Figures 16 (Images (11) and (12)) and 19 are from the movie Tears of Steel by (CC) Blender Foundation (mango.blender.org). The result images in Figures 1(d) and 16 (Images (9)–(12)) were generated by Alessia Marra, and the ones in Figures 1(c), 15 (Images (1)–(8)), and 16 (Images (13)–(19)) by Yağız Aksoy.

REFERENCES

Yağız Aksoy, Tunç Ozan Aydın, Marc Pollefeys, and Aljoša Smolić. 2016. Interactive high-quality green-screen keying via color unmixing. *ACM Trans. Graph.* 35, 5 (2016), 152:1–152:12.

Dimitri P. Bertsekas. 1982. The method of multipliers for equality constrained problems. In *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, New York, NY, 96–157.

Robert Carroll, Ravi Ramamoorthi, and Maneesh Agrawala. 2011. Illumination decomposition for material recoloring with consistent interreflections. *ACM Trans. Graph.* 30, 4 (2011), 43:1–43:10.

Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. 2015. Palette-based photo recoloring. *ACM Trans. Graph.* 34, 4 (2015), 139:1–139:11.

Qifeng Chen, Dingzeyu Li, and Chi-Keung Tang. 2013. KNN matting. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 9 (2013), 2175–2188.

Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. 2001. A Bayesian approach to digital matting. In *Proc. CVPR*.

Yu Feng and Greg Hamerly. 2006. PG-means: Learning the number of clusters in data. In *Proc. NIPS*.

Eduardo S. L. Gastal and Manuel M. Oliveira. 2010. Shared sampling for real-time alpha matting. *Comput. Graph. Forum* 29, 2 (2010), 575–584.

Greg Hamerly and Charles Elkan. 2003. Learning the K in K-means. In *Proc. NIPS*.

Kaiming He, Jian Sun, and Xiaoou Tang. 2013. Guided image filtering. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 6 (2013), 1397–1409.

Anat Levin, Dani Lischinski, and Yair Weiss. 2008a. A closed-form solution to natural image matting. *IEEE Trans. Pattern Anal. Mach. Intell.* 30, 2 (2008), 228–242.

Anat Levin, Alex Rav-Acha, and Dani Lischinski. 2008b. Spectral matting. *IEEE Trans. Pattern Anal. Mach. Intell.* 30, 10 (2008), 1699–1712.

Ido Omer and Michael Werman. 2004. Color lines: Image specific color representation. In *Proc. CVPR*.

Jinshan Pan, Zhe Hu, Zhixun Su, Hsin-Ying Lee, and Min-Hsuan Yang. 2016. Soft-segmentation guided object motion deblurring. In *Proc. CVPR*.

J. Pont-Tuset and F. Marques. 2015. Supervised evaluation of image segmentation and object proposal techniques. *IEEE Trans. Pattern Anal. Mach. Intell.* 38, 7 (2015), 1465–1478.

Thomas Porter and Tom Duff. 1984. Compositing digital images. *SIGGRAPH Comput. Graph.* 18, 3 (1984), 253–259.

Iulia Posirca, Yunmei Chen, and Celia Z. Barcelos. 2011. A new stochastic variational PDE model for soft Mumford-Shah segmentation. *J. Math. Anal. Appl.* 384, 1 (2011), 104–114.

C. Richardt, J. Lopez-Moreno, A. Bousseau, M. Agrawala, and G. Drettakis. 2014. Vectorising bitmaps into semi-transparent gradient layers. *Comput. Graph. Forum* 33, 4 (2014), 11–19.

Mark A. Ruzon and Carlo Tomasi. 2000. Alpha estimation in natural images. In *Proc. CVPR*.

Mark Schmidt. 2007. UGM: A Matlab toolbox for probabilistic undirected graphical models. Retrieved from http://www.cs.ubc.ca/~schmidtm/Software/UGM.html.

YiChang Shih, Dilip Krishnan, Fredo Durand, and William T. Freeman. 2015. Reflection removal using ghosting cues. In *Proc. CVPR*.

D. Singaraju and R. Vidal. 2011. Estimation of alpha mattes for multiple image layers. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 7 (2011), 1295–1309.

Yu-Wing Tai, Jiaya Jia, and Chi-Keung Tang. 2005. Local color transfer via probabilistic segmentation by expectation-maximization. In *Proc. CVPR*.

Yu-Wing Tai, Jiaya Jia, and Chi-Keung Tang. 2007. Soft color segmentation and its applications. *IEEE Trans. Pattern Anal. Mach. Intell.* 29, 9 (2007), 1520–1537.

Jianchao Tan, Jyh-Ming Lien, and Yotam Gingold. 2016. Decomposing images into layers via RGB-space geometry. *ACM Trans. Graph.* 36, 1 (2016), 7:1–7:14.

F. Yang, H. Lu, and Y. W. Chen. 2010b. Robust tracking based on boosted color soft segmentation and ICA-R. In *Proc. ICIP*.

Lei Yang, Pedro V. Sander, Jason Lawrence, and Hugues Hoppe. 2011. Antialiasing recovery. *ACM Trans. Graph.* 30, 3 (2011), 22:1–22:9.

W. Yang, J. Cai, J. Zheng, and J. Luo. 2010a. User-friendly interactive image segmentation through unified combinatorial user inputs. *IEEE Trans. Image Process.* 19, 9 (2010), 2470–2479.

Sai-Kit Yeung, Tai-Pang Wu, and Chi-Keung Tang. 2008. Extracting smooth and transparent layers from a single image. In *Proc. CVPR*.

Received August 2016; revised December 2016; accepted December 2016