Diss. ETH No. 26080

Soft Segmentation of Images

A thesis submitted to attain the degree of
Doctor of Sciences of **ETH Zurich**
(Dr. sc. ETH Zurich)

presented by
Yağız Aksoy
MSc in Electrical and Electronics Engineering,
Middle East Technical University, Ankara
born on 01.02.1989
citizen of Turkey

accepted on the recommendation of
Prof. Dr. Marc Pollefeys, examiner
Prof. Dr. Konrad Schindler, co-examiner
Prof. Dr. Anat Levin, co-examiner
Dr. Sylvain Paris, co-examiner

2019

Abstract

Realistic editing of photographs requires careful treatment of color mixtures that commonly occur in natural scenes. These color mixtures are typically modeled using soft selection of objects or scene colors. Hence, accurate representation of these soft transitions between image regions is essential for high-quality image editing and compositing. Current techniques for generating such representations depend heavily on interaction by a skilled visual artist, as creating such accurate object selections is a tedious task.

In this thesis, we approach the soft segmentation problem from two complementary properties of a photograph. Our first focus is representing images as a mixture of main colors in the scene, by estimating soft segments of homogeneous colors. We present a robust per-pixel nonlinear optimization formulation while simultaneously targeting computational efficiency and high accuracy. We then turn our attention to semantics in a photograph and present our work on soft segmentation of particular objects in a given scene. This work features graph-based formulations that specifically target the accurate representation of soft transitions in linear systems. Each part first presents an interactive segmentation scheme that targets applications popular in professional compositing and movie post-production. The interactive formulations are then generalized to the automatic estimation of generic image representations that can be used to perform a number of otherwise complex image editing tasks effortlessly.

The first problem studied is green-screen keying, interactive estimation of a clean foreground layer with accurate opacities in a studio setup with a controlled background, typically set to be green. We present a simple two-step interaction scheme to determine the main scene colors and their locations. The soft segmentation of the foreground layer is done via the novel color unmixing formulation, which can effectively represent a pixel color as a mixture of many colors characterized by statistical distributions. We show our formulation is robust against many challenges in green-screen keying and can be used to achieve production-quality keying results at a fraction of the time compared to commercial software.

We then study soft color segmentation, estimation of layers with homogeneous colors and corresponding opacities. The soft color segments can be overlayed to give the original image, providing effective intermediate representation of an image. We decompose the global energy optimization formulation that typically models

the soft color segmentation task into three sub-problems that can be implemented with computational efficiency and scalability. Our formulation gets its strength from the color unmixing energy, which is essential in ensuring homogeneous layer colors and accurate opacities. We show that our method achieves a segmentation quality that allows realistic manipulation of colors in natural photographs.

Natural image matting is the generalized version of green-screen keying, where an accurate estimation of foreground opacities is targeted in an unconstrained setting. We approach this problem using a graph-based approach, where we model the connections in the graph as forms of information flow that distributes the information from the user input into the whole image. By carefully defining information flows to target challenging regions in complex foreground structures, we show that high-quality soft segmentation of objects can be estimated through a closed-form solution of a linear system. We extend our approach to related problems in natural image matting such as matte refinement and layer color estimation and demonstrate the effectiveness of our formulation through quantitative, qualitative and theoretical analysis.

Finally, we introduce semantic soft segments, a set of layers that correspond to semantically meaningful regions in an image with accurate soft transitions between different objects. We approach this problem from a spectral segmentation angle and propose a graph structure that embeds texture and color features from the image as well as higher-level semantic information generated by a neural network. The soft segments are generated via eigendecomposition of the carefully constructed Laplacian matrix fully automatically. We demonstrate that compositing and targeted image editing tasks can be done with little effort using semantic soft segments.

Zusammenfassung

Die realistische Bearbeitung von Fotos erfordert eine sorgfältige Behandlung von Farbmischungen, die häufig in natürlichen Szenen auftreten. Diese Farbmischungen werden typischerweise unter Verwendung einer weichen Auswahl von Objekten oder Szenefarben modelliert. Eine genaue Darstellung dieser weichen Übergänge zwischen Bildbereichen ist daher für eine qualitativ hochwertige Bildbearbeitung und -zusammenstellung wesentlich. Aktuelle Techniken zur Erzeugung solcher Darstellungen hängen stark von der Interaktion eines erfahrenen Grafikers ab, da das Erzeugen einer derart genauen Objektauswahl eine mühsame Aufgabe ist.

In dieser Arbeit nähern wir uns dem Problem der weichen Segmentierung von zwei komplementären Eigenschaften einer Fotografie aus. Unser erster Fokus liegt auf der Darstellung von Bildern als Mischung von Hauptfarben in der Szene, indem weiche Segmente homogener Farben geschätzt werden. Wir präsentieren eine robuste pixelweise nichtlineare Optimierung, die gleichzeitig effizient berechenbar und genau ist. Wir wenden uns dann der Semantik in einer Fotografie zu und präsentieren unsere Arbeit über die weiche Segmentierung bestimmter Objekte in einer Szene. Dieser Teil der Arbeit enthält graphbasierte Formulierungen, die speziell auf die genaue Darstellung von weichen Übergängen in linearen Systemen abzielen. In jedem Teil wird zunächst ein interaktives Segmentierungsschema vorgestellt, das auf Anwendungen abzielt, die im professionellen Compositing und in der Filmpostproduktion beliebt sind. Die interaktiven Formulierungen werden dann auf die automatische Schätzung generischer Bilddarstellungen verallgemeinert, mit denen mühelos eine Reihe ansonsten komplexer Bildbearbeitungsaufgaben durchgeführt werden können.

Das erste untersuchte Problem ist das Green-Screen-Keying, die interaktive Schätzung einer sauberen Vordergrundebene mit genauen Opazitäten in einem Studio-Setup mit einem kontrollierten Hintergrund, der normalerweise auf Grün eingestellt ist. Wir präsentieren ein einfaches, zweistufiges Interaktionsschema zur Bestimmung der Hauptszenenfarben und ihrer Positionen. Die weiche Segmentierung der Vordergrundschicht erfolgt über die neuartige Farbentmischungsformulierung, die effektiv eine Pixelfarbe als Mischung vieler Farben darstellen kann, die durch statistische Verteilungen gekennzeichnet sind. Wir zeigen, dass unsere Formulierung vielen Herausforderungen beim Green-Screen-Keying standhält und

im Vergleich zu kommerzieller Software in einem Bruchteil der Zeit zur Erzielung von Keying-Ergebnissen in Produktionsqualität verwendet werden kann.

Anschließend untersuchen wir die weiche Farbsegmentierung, die Schätzung von Schichten mit homogenen Farben und der entsprechenden Opazitäten. Die weichen Farbsegmente können überlagert werden, um das Originalbild zu erhalten und eine effektive Zwischendarstellung eines Bildes zu ermöglichen. Wir zerlegen die globale Energieoptimierungsformulierung, die typischerweise die Aufgabe der weichen Farbsegmentierung modelliert, in drei Unterprobleme, die effizient und skalierbar implementiert werden können. Die Effektivität unserer Formulierung basiert auf der Farbentmischungsenergie, die für homogene Schichtfarben und genaue Opazitäten unerlässlich ist. Wir zeigen, dass unsere Methode eine Segmentierungsqualität erzielt, die eine realistische Manipulation von Farben in natürlichen Fotografien ermöglicht.

Die natürliche Bildmattierung ist die verallgemeinerte Version der Green-Screen-Kodierung, bei der eine genaue Schätzung der Vordergrundopazitäten in einer uneingeschränkten Umgebung angestrebt wird. Wir nähern uns diesem Problem mit einem graphbasierten Ansatz, bei dem wir die Kanten des Graphen zur Modellierung eines Informationsflusses verwenden, der die Informationen aus der Benutzereingabe auf das gesamte Bild verteilt. Indem wir den Informationsfluss zu schwierigen Regionen in komplexen Vordergrundstrukturen sorgfältig definieren, zeigen wir, dass eine hochqualitative weiche Segmentierung von Objekten durch eine geschlossene Lösung eines linearen Systems geschätzt werden kann. Wir erweitern unseren Ansatz auf verwandte Probleme der natürlichen Bildmattierung wie Mattverfeinerung und Schätzung der Schichtfarbe und belegen die Effektivität unserer Formulierung durch quantitative, qualitative und theoretische Analysen.

Zum Schluss führen wir semantische weiche Segmente ein, eine Reihe von Ebenen, die semantisch bedeutsamen Bereichen in einem Bild mit genauen weichen Übergängen zwischen verschiedenen Objekten entsprechen. Wir betrachten dieses Problem aus der Sichtweise einer spektralen Segmentierung und schlagen eine Graphstruktur vor, die Textur- und Farbmerkmale aus dem Bild sowie semantische Informationen auf höherer Ebene, die von einem neuronalen Netzwerk generiert werden, einbettet. Die weichen Segmente werden durch Spektralzerlegung der speziell konstruierten Laplace-Matrix vollautomatisch erzeugt. Wir zeigen, dass Compositing und gezielte Bildbearbeitungsaufgaben mit semantischen weichen Segmenten mit geringem Aufwand erledigt werden können.

Acknowledgements

I was very lucky to have the chance to work with many great supervisors and mentors during my PhD. Firstly, I would like to thank my PhD advisor, Marc Pollefeys, for providing a rich, open and stable research environment. His continued support and dynamic mentorship created an excellent learning environment for me throughout my PhD. I would also like to thank my supervisor during my visit at MIT, Wojciech Matusik, for the dynamic collaborative research environment he has built and for his insightful academic mentorship. I thank my supervisor Aljoša Smolić for his support and encouragement that made my years at Disney very rewarding. I am very thankful to Sylvain Paris for his generous support and guidance as well as our continued collaboration. I thank Tunç Ozan Aydın for everything I learned from him over the past 6 years, his collaboration, and his support on many dimensions of life.

I would like to thank my committee members Konrad Schindler and Anat Levin for devoting their time to participate in my defense. Academia can be a refreshingly open and supportive environment for young researchers, and I am thankful for the support I received from Frédo Durand, Stelian Coros, Jana Gičeva, Tuğçe Yazıcıgil, and the SIGGRAPH community.

I received a lot of help from many amazing colleagues during my PhD. Here, I would like to mention Tae-Hyun Oh for his enthusiastic collaboration and friendship, Alessia Marra and Maurizio Nitti for their artistic advice as well as their help with many results shown in this thesis, and Alexandre Kaspar, Petr Kellnhofer, Michael Gharbi, Niko Stefanovski, and Oliver Wang for their help and our discussions.

Thanks to everyone with whom I had the chance to share my time at CVG, IVC, DRZ, and CFG. I can not think of CVG without Torsten who has been a friend and the go-to person for many questions for the whole time I was in Zurich, or Andrea after all the memories of Beckhammer 19. While the faces in the group kept changing, the friendly environment persisted thanks to Gim-Hee, Aparna, Christian, Bastian, Federico, Pablo, Johannes, Daniel, and Tara. I am glad I had the opportunity to delve into deep conversations with Olga, share many breaks and burger nights with Simone, Federico, Endri and Marios, and have Akın, Işık, Pelin, Amit, JC and many more I'm apologetically missing here as friends by my side in Zürich and Boston.

I was able to enjoy (and get through) these years thanks to many long-distance friends. Sırma, thanks for sharing many slices of life, struggles and joys alike, with your deep sincerity and generosity. Beril and Emin, thanks for being my support system for long rants on the phone, and for all the trips, Skype sessions, and perspectives on life. Ümit, thanks for everything I learned with and from you over the last decade, you taught me more than you realize. Kaan & Alex, it has been a pleasure to grow into our 30's together and share many struggles as well as many journeys in this mind-bending thing called life. Thanks to the lovely communities in the Sacred Valley and BRC. I am glad to have known you and that we have been able to continue our friendship across all the boundaries and time zones throughout all these years.

This thesis is dedicated to my parents Vicdan and Nadir, and my brother Yalın. Your unequivocal and immense support and understanding made everything I have possible and meaningful. I love you all.

Contents

List of Figures

List of Algorithms

List of Tables

CHAPTER 1

Introduction

Photographic expression of artistic intent and aesthetics has been a part of global culture with growing importance through easily accessible capture devices and sharing mediums. The widespread use of cheaper and higher-quality cameras as well as social content sharing websites brought in the democratization of photography and video production.

Despite the wide public interest, the image and video editing tools still have a steep learning curve. This brings an entry barrier for amateur content creators and casual photographers who wish to express themselves through high-quality image manipulation tools. In addition, creating production-quality content is still a bottleneck in terms of cost and time for professional movie studios and photographers. Hence, providing convenient tools for realistic image editing is of public and commercial 'interest.

One of the biggest challenges in realistic editing of imagery comes from the intricate mixtures of colors between distinct image regions in natural photographs. As Figure 1.1 demonstrates, some of the prominent reasons for these color mixtures can be listed as:

- Intricate structures such as hair, through which the background can be partially observed due to their small size,
- Multiple illuminations, that illuminate the scene with different colors from different directions,
- Fast moving objects, that create motion blur due to the finite exposure time of the camera,

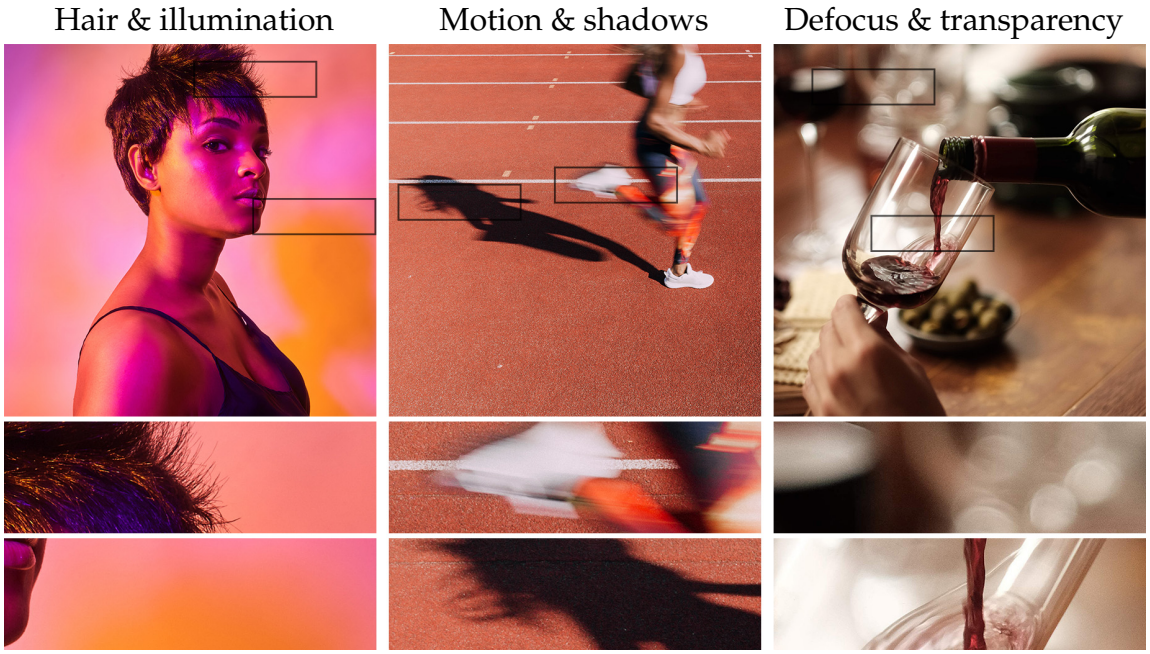

Figure 1.1.: *Soft transitions between regions may come from intricate regions such as hair, smooth illumination changes, motion blur, natural shadows, defocus blur or transparency.*

- Shadows, that has a distinctive penumbra around their edges when the light source has a finite size,
- Defocus blur, that occurs due to the aperture of the camera and is usually characterized by the lens, or
- Transparent objects, that partially transmits the light coming from behind the object.

In fact, color mixtures occur even around the hard edges between objects due to the finite size of the image sensors. These color mixtures occur in all natural photographs and they need to be handled carefully in image editing for a realistic end result.

These phenomenon can be modeled individually for targeted analysis. In the academic literature, there are many works that focuses on analyzing motion blur [Lin et al., 2011; Pan et al., 2016], illumination decomposition [Hui et al., 2019; Aksoy et al., 2018a], shadow analysis and removal [Chuang et al., 2003; Wu et al., 2007], or defocus blur analysis [Bae and Durand, 2007; Zhu et al., 2013], to list a few.

In this thesis, we will approach these color mixtures from a more generic perspective through *soft segmentation*. Image segmentation is the problem of partitioning the image into regions by assigning each pixel a particular label that corresponds to a segment. Soft segmentation aims to assign each

pixel *partial* labels, allowing each pixel to belong to multiple segments. This partial labeling can be used to represent the color mixtures that only depend on the appearance of the pixel in the digital representation of the photograph, without going into particulars of the physical phenomenon behind the mixture. That being said, our work on soft segmentation will include a focus on analyzing intricate structures and transparency that are of high interest in interactive compositing applications. We will demonstrate that carefully designed soft segmentation algorithms open up new realistic image editing capabilities that require minimal individual expertise from the artist.

1.1. Topics in this thesis

We approach the soft segmentation problem from two different directions: with respect to colors and with respect to objects in a given scene. In Part I, we will introduce *color unmixing*, a per-pixel nonlinear energy formulation that effectively represents the image as a mixture of scene colors. Part II focuses on graph-based representations of images that enable accurate soft segmentation of objects in the image through linear global energy formulations. Each part starts with the study of an interactive segmentation problem commonly used in image editing and movie post production. We then extend these formulations to propose fully automatic estimation of generic image representations that make complex image editing tasks trivial via per-layer operations.

1.1.1. Green-Screen Keying

As computer-generated imagery became convincingly realistic, compositing synthetic backgrounds and objects into live-action shots became a common practice in feature film production. The widespread use of composite shots over pure live-action is often motivated by the higher degree of artistic control over the final shot, as well as the potential to reduce production costs. Usually, the first step in a digital compositing workflow is the performance capture of the actors and various other live-action elements against a controlled — typically green — background. Then, in post-production, one needs to obtain RGBA foreground layers corresponding to the live-action elements that ideally carry no trace of the green-screen background. This process is often referred to as *keying*. Finally, one or more foreground layers are combined with the desired computer generated scene elements to obtain the composite shot.

Keying is a crucial intermediate step in any compositing workflow, as later

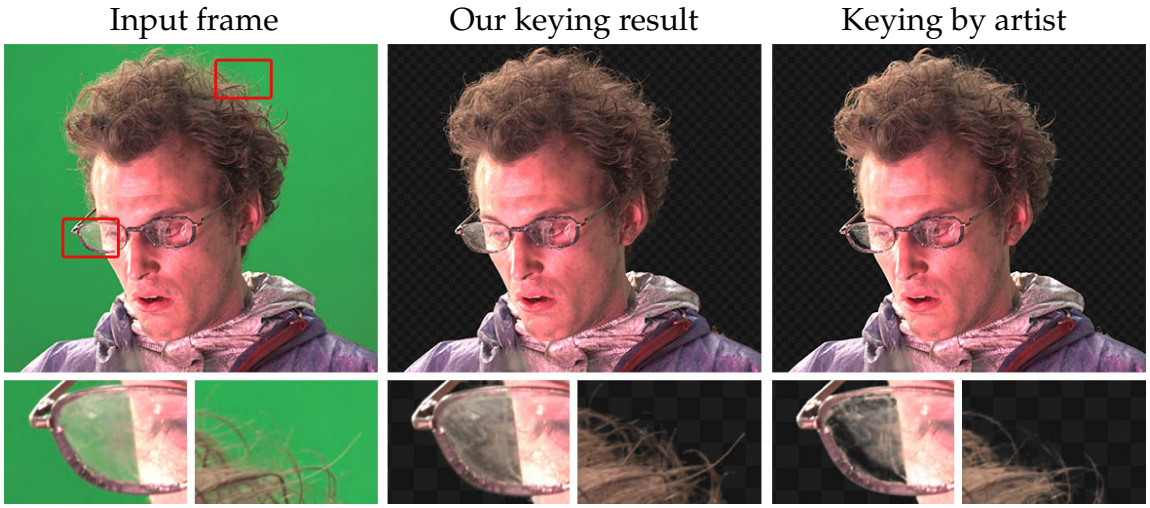

Figure 1.2.: *Green-screen keying is the extraction of the foreground object shot in front of a controlled background with accurate opacities and colors. We present an interactive approach that is robust around challenging regions such as translucency and intricate object boundaries like the curly hair as seen above. Our method achieves and usually surpasses the keying quality of a professional artist using commercial software at one-tenth of the interaction time.*

in the workflow seamless blending between the synthetic and live-action elements is highly dependent on obtaining high-quality keying results. The keying process usually starts with the compositing artist obtaining preliminary foreground layers by using multiple software tools in concert, some of the most popular ones being The Foundry's *Keylight*, Nuke's Image Based Keyer *(IBK)* and Red Giant's *Primatte*. Often, this first step already involves significant manual labor in the form of parameter tweaking or drawing roto-masks. Ideally, the preliminary foreground layers would already be sufficiently high quality so that one can move on to consecutive steps in the compositing pipeline. Unfortunately, this is rarely the case in practice and the imperfections in the foreground layer still have to be corrected by manual painting before moving forward. In professional circles, the combined manual work required for both obtaining preliminary keying results and later their refinement by manual painting is recognized as a significant bottleneck in post-production. We present an example process of keying by an artist using comercial tools in Section 3.3.

Our contributions

The feedback we collected from industry professionals as well as our own experience showed that commercial software tools have difficulties dealing

with image regions where the colors of multiple objects mix, either due to motion blur, intricate object boundaries (e.g. hair), or color spill, i.e. color cast due to indirect illumination from green-screen. Influenced by this observation, we propose a novel energy function for solving the fundamental problem of *unmixing* a color mixture, i.e. computing both the individual underlying colors as well as their mixing ratios, i.e. their opacities or alpha values. We efficiently minimize this energy function by utilizing priors for the underlying colors in the mixture, which are obtained and refined through a two-step user interaction designed specifically for green-screen keying. In a comprehensive set of quantitative and qualitative evaluations with the help of a specialized compositing artist, we show that our method consistently outperforms both the current commercial software tools and the state-of-the-art natural matting methods in the domain of green-screen keying. Importantly, the superior results of our technique can be obtained on average by using only *one-tenth* of the manual interaction time required by a trained artist for processing the same content with the current state-of-the-art tools. Figure 1.2 shows the keying results of the proposed method next to one by the professional artist.

1.1.2. Soft Color Segmentation

The goal of soft color segmentation is to decompose an image into a set of layers with alpha channels, such as in Figure 1.3. 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 Chapter 4, 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 at a different time of the day (Figure 1.3), to more pronounced changes that involve dramatic hue shifts or replacing the image background.

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

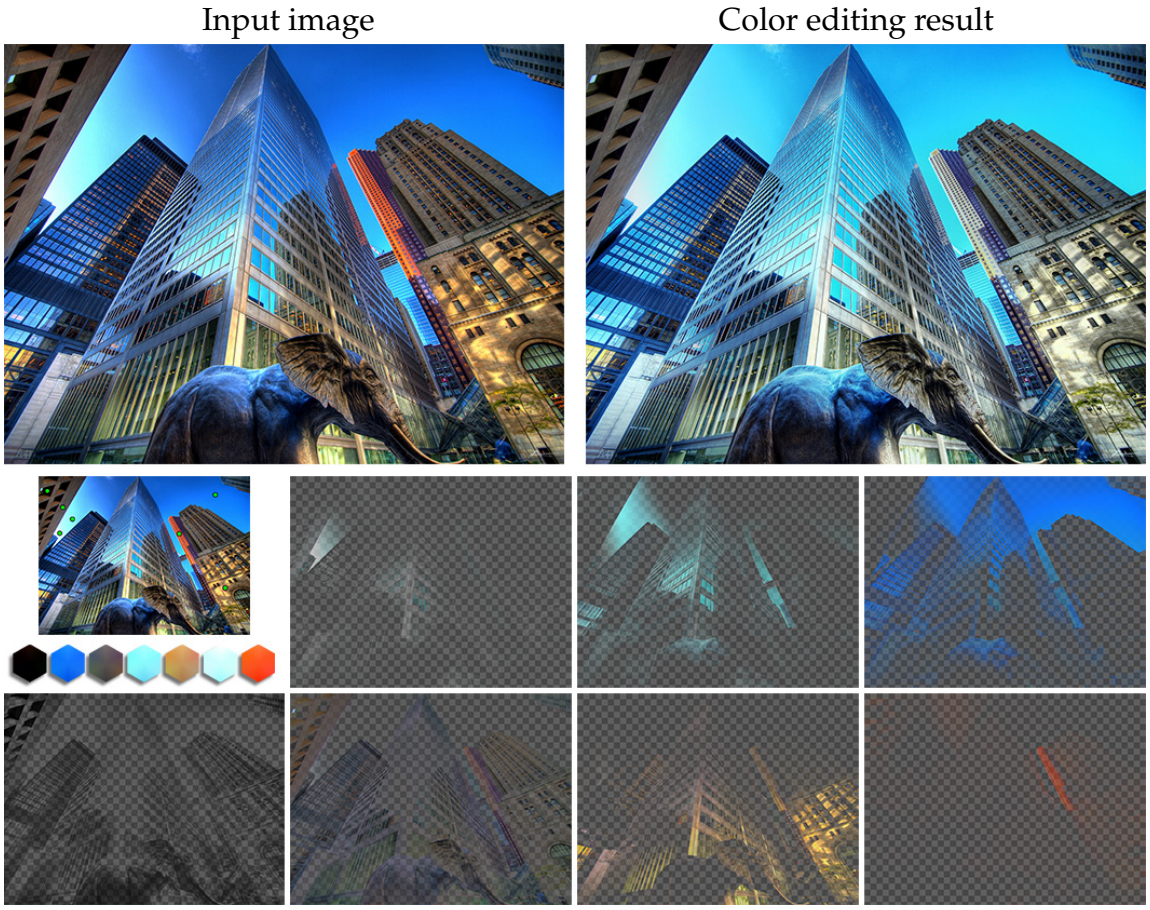

Figure 1.3.: *We propose a fully automatic soft color segmentation method that generates high-quality representations of photographs as mixtures of the main scene colors. These layers of homogeneous colors boils complex image editing tasks down to simple per-layer operations.*

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 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 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. We provide a theoretical analysis of the soft color segmentation methods in the literature in Section 4.2.

Our contributions

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 extend our color unmixing formulation to better fit the problem of a generic soft color segmentation that enforces *matte sparsity*, favoring fully opaque or transparent pixels. This extended formulation that we call *sparse color unmixing* (SCU) decomposes the image into layers of homogeneous colors. 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. By breaking the requirements of the soft color segmentation problem into these sub-problems, we require computational resources that are magnitudes less than the state-of-the-art. 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 show that our method consistently produces high-quality layers, and demonstrate numerous common image manipulation applications can be reduced to trivial per-layer operations that can be performed conveniently through familiar software tools.

1.1.3. Natural Image Matting

Extracting the opacity information of user-defined objects from an image is known as natural image matting. Natural image matting has received great interest from the research community in the last decade and can nowadays be considered as one of the classical research problems in visual computing. Mathematically, image matting requires expressing pixel colors in the transition regions from foreground to background as a convex combination of their underlying foreground and background colors. The weight, or the *opacity*, of the foreground color is referred to as the alpha value of that pixel. Since neither the foreground and background colors nor the opacities are known, estimating the opacity values is a highly ill-posed problem. To alleviate the

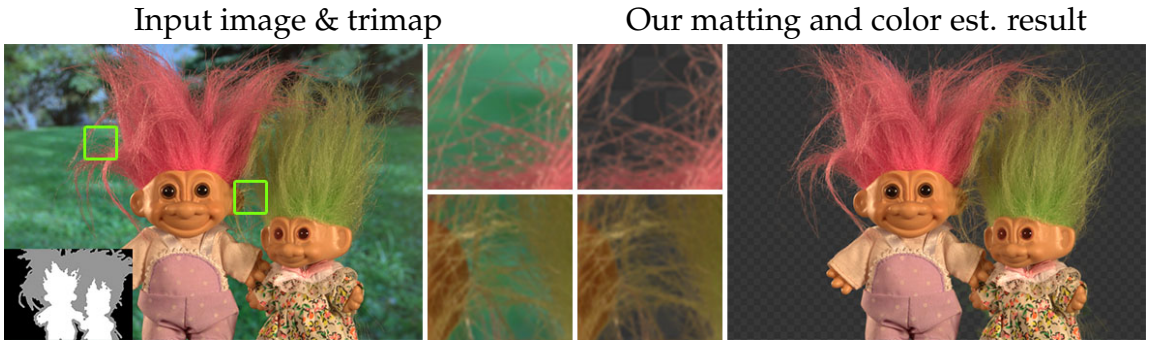

Figure 1.4.: *Given the input image and the corresponding trimap, our novel affinity-based matting formulation can estimate high-quality opacities and layer colors even around challenging scene structures and similar foreground and background colors.*

difficulty of this problem, typically a *trimap* is provided in addition to the original image. The trimap is a rough segmentation of the input image into foreground, background, and regions with unknown opacity.

The main application of natural image matting is compositing, i.e. combining different scenes together to generate a new image. Image matting methods aim to provide accurate opacities such that when the foreground is overlayed onto a novel background, the transitions between them look natural. However, together with the matte, compositing requires the actual, *unmixed* layer colors for realistic composites. The layer colors appear as a mixture of foreground and background colors in the input image, and they are underconstrained even with a given matte. Hence, accurate estimation of the layer colors is a critical component of a compositing pipeline and still an active research problem.

Affinity-based methods [Levin et al., 2008a; Chen et al., 2013a; Chen et al., 2012] constitute one of the prominent natural matting approaches in literature. These methods make use of pixel similarities to propagate the alpha values from the known-alpha regions to the unknown region. They provide a clear mathematical formulation, can be solved in closed-form and typically produce spatially consistent mattes. In addition, due to their formulation that can be modeled as a graph structure with each pixel as a node, affinity-based approaches can be generalized to related applications such as layer color estimation [Levin et al., 2008a], edit propagation [Chen et al., 2012], and soft segmentation [Levin et al., 2008b]. Studying affinity-based approaches for natural matting can open new directions for a larger set of applications in the image processing community.

Our contributions

In spite of these advantages, current affinity-based methods fail to effectively handle alpha gradients spanning large areas and spatially disconnected regions (i.e. *holes*) even in simple cases. This is because a straightforward formulation using the pixel-to-pixel affinity definitions can not effectively represent the complex structures that are commonly seen in real-life objects. We provide an analysis of different affinity-based methods through spectral decomposition in Section 5.5. In order to alleviate these shortcomings, we rely on a careful, case-by-case design of how alpha values should propagate inside the image. We conceptualize the affinities as *information flows* to help understanding and designing effective graph-based structures to propagate information in the image. We define several information flows, some of which target unknown-opacity regions that are remote and hence does not receive enough information in previous formulations. Other types of information flows address issues such as evenly distributing information inside the unknown region. Our final linear system can be solved in closed-form and results in a significant quality improvement over the state-of-the-art. In addition, we extend our graph-based formulation to matte refinement and layer color estimation. Figure 1.4 shows the result of our natural matting and layer color estimation methods.

1.1.4. Semantic Soft Segmentation

Soft selection of regions in the image is at the core of the image editing process. For instance, local adjustments often start with a selection, and combining elements from different images is a powerful way to produce new content. But creating an accurate selection is a tedious task especially when fuzzy boundaries and transparency are involved. Tools such as the *magnetic lasso* and the *magic wand* exist to assist users but they only exploit low-level cues and heavily rely on the users' skills and interpretation of the image content to produce good results. Furthermore, they only produce binary selections that need further refinement to account for soft boundaries like the silhouette of a furry dog. Natural matting tools also exist to help users with this task but especially for casual users, they add to the tedium of the entire editing process.

An accurate pre-segmentation of the image can speed up the editing process by providing an intermediate image representation if it satisfies several criteria. First of all, such a segmentation should provide distinct segments of the image, while also representing the soft transitions between them accurately. In order to allow targeted edits, each segment should be limited to the extent

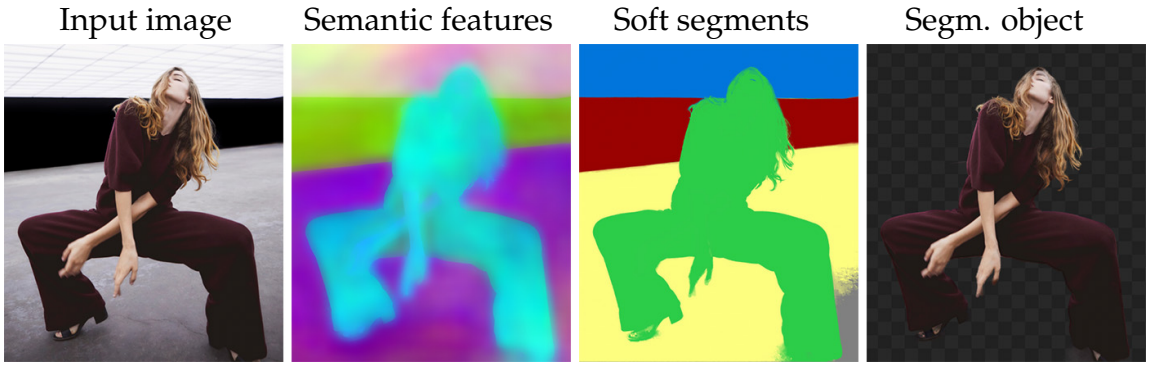

Figure 1.5.: *By fusing low-level features from the image with high-level information on objectness from a neural network in a single graph structure, we automatically extract mattes of the objects in the image in a process we call semantic soft segmentation. Semantic soft segments, visualized by assigning each a solid color, can be used for compositing or targeted image editing applications.*

of a semantically meaningful region in the image, e.g., it should not extend across the boundary between two objects. Finally, the segmentation should be done fully automatically not to add a point of interaction or require expertise from the artist. The previous approaches for semantic segmentation, image matting, or soft color segmentation fail to satisfy at least one of these qualities.

Our contributions

We introduce *semantic soft segmentation*, a fully automatic decomposition of an input image into a set of layers that cover scene objects, separated by soft transitions. We approach the semantic soft segmentation problem from a spectral decomposition angle. We combine the texture and color information from the input image together with high-level semantic cues that we generate using a convolutional neural network trained for scene analysis. We design a graph structure that reveals the semantic objects as well as the soft transitions between them in the eigenvectors of the corresponding Laplacian matrix. We introduce a spatially varying model of layer sparsity that generates high-quality layers from the eigenvectors that can be utilized for image editing.

We demonstrate that our algorithm successfully decomposes images into a small number of layers that compactly and accurately represent the scene objects. We also show that our algorithm can successfully process images that are challenging for other techniques and we provide examples of editing operations such as local color adjustment or background replacement that benefit from our layer representation. Figure 1.5 shows the high-level features that is used in the graph formulation, the semantic soft segments

and an isolated semantic region of the image that is ready for compositing applications.

1.2. Publications

The following peer-reviewed work has been published in the context of this thesis:

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

[Aksoy et al., 2017b] **Yağız Aksoy**, Tunç Ozan Aydın, Aljoša Smolić, and Marc Pollefeys. Unmixing-based soft color segmentation for image manipulation. *ACM Trans. Graph.*, 36(2):19:1–19:19, 2017.

[Aksoy et al., 2017a] **Yağız Aksoy**, Tunç Ozan Aydın, and Marc Pollefeys. Designing effective inter-pixel information flow for natural image matting. In *International Conference on Computer Vision and Pattern Recognition (Proc. CVPR)*, 2017.

[Aksoy et al., 2018b] **Yağız Aksoy**, Tae-Hyun Oh, Sylvain Paris, Marc Pollefeys, and Wojciech Matusik. Semantic soft segmentation. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 37(4):72:1–72:13, 2018.

The work below was also conducted during the time period of this doctoral study but was not included in this thesis:

[Angehrn et al., 2014] Florian Angehrn, Oliver Wang, **Yağız Aksoy**, Markus Gross, and Aljoša Smolić. MasterCam FVV: Robust registration of multiview sports video to a static high-resolution master camera for free viewpoint video. In *International Conference on Image Processing (Proc. ICIP)*, 2014.

[Ryffel et al., 2017] Mattia Ryffel, Fabio Zünd, **Yağız Aksoy**, Alessia Marra, Maurizio Nitti, Tunç Ozan Aydın, and Bob Sumner. AR museum: A mobile augmented reality application for interactive painting recoloring. In *International Conference on Game and Entertainment Technologies (Proc. GET)*, 2017.

[Aksoy et al., 2018a] **Yağız Aksoy**, Changil Kim, Petr Kellnhofer, Sylvain Paris, Mohamed Elgharib, Marc Pollefeys, and Wojciech Matusik. A dataset of flash and ambient illumination pairs from the crowd. In *European Conference on Computer Vision (Proc. ECCV)*, 2018.

[Kaspar et al., 2018] Alexandre Kaspar, Geneviève Patterson, Changil Kim,

Yağız Aksoy, Wojciech Matusik, and Mohamed Elgharib. Crowd-guided ensembles: How can we choreograph crowd workers for video segmentation? In *Conference on Human Factors in Computing Systems (Proc. ACM CHI)*, 2018.

[Tang et al., 2019] Jingwei Tang, **Yağız Aksoy**, Cengiz Öztireli, Markus Gross, and Tunç Ozan Aydın. Learning-based sampling for natural image matting. In *International Conference on Computer Vision and Pattern Recognition (Proc. CVPR)*, 2019.

CHAPTER 2

Related Work

Previous work uses the term *soft segmentation* in various contexts, such as the probabilistic classification of CT scans [Posirca et al., 2011], computing per-pixel foreground/background probabilities [Yang et al., 2010a], and interactive image segmentation utilizing soft input constraints [Yang et al., 2010b]. 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 will use soft segmentation to represent the color mixtures in images. In this context, the partial labels that get assigned to each pixel represent the weight of the color of each segment that forms the color mixture. In the case of assigning one of two labels to a pixel, i.e. foreground or background, these weights are called *opacities*, represented by α values per pixel. In this chapter, we will summarize work closely related to the soft segmentation problems studied in the rest of the thesis.

2.1. Interactive Soft Segmentation for Compositing

Interactive soft segmentation typically targets the extraction of the opacities of a user-defined foreground object. The main compositing model used in the literature for this two-layer decomposition is:

$$\boldsymbol{c}_p = \alpha_p \boldsymbol{f}_p + (1 - \alpha_p)\, \boldsymbol{b}_p, \tag{2.1}$$

where $\boldsymbol{c}_p$ is the observed image color at pixel p, and α_p, $\boldsymbol{f}_p$, and $\boldsymbol{b}_p$ are the opacity of the foreground, and the color of the foreground and background colors that got into the mixture at pixel p, respectively.

2.1.1. Green-Screen Keying

Keying is the process of extracting the foreground objects with corresponding opacities in a controlled capture setup that specifically targets compositing the foreground in a novel scene. It is widely used in movie post-production, television broadcasting, and amateur video making. The process is called *luma* keying when the background color $\boldsymbol{b}_p$ is constrained to be very bright or very dark, or *chroma* keying when the defining characteristic of the background is its color. The most popular background color in the digital age is green, and hence the process is commonly referred to as green-screen keying.

Commercial keying tools often use chroma-based or luma-based algorithms. In feature film post-production, these tools are operated by specialized compositing artists for obtaining a *preliminary* keying result. Preliminary results often require further manual processing, because, despite the parameter tweaking and the usage of roto-masks, they often fall short of the quality level demanded in professional productions. Since such keying results are unacceptable in professional production, the preliminary keying results undergo an extremely tedious manual painting process, where each pixel in the video is cleaned off of keying errors by hand. We provide a detailed step-by-step keying process by an independent keying artist to show the common practice in keying in Section 3.3.

Keying received relatively little attention in the literature. Smith and Blinn [1996] present an overview of chroma keying in the industry until 1996, typically disclosed in patents rather than academic manuscripts, and present the theoretical foundations of keying through the compositing equation (2.1) focusing on the color content of the foreground. While constraining the background color is helpful to get cleaner foreground results, there are still four unknowns ($\boldsymbol{f}$ and α) and three equations, one for each color channel, in (2.1). Grundhöfer et al [2010] proposes an overconstrained problem by shooting the same foreground with two different and controlled backgrounds, increasing the number of equations to six. In their recent work, LeGendre et al. [2017] uses oriented filters to increase keying performance around thin hair strands.

We present a keying approach that uses a statistical color model of the scene in Chapter 3. Unlike the commercial software that uses a color definition for the background and depends further on the skills of a digital artist, our color

model represents all the colors in the scene. Through the use of novel color unmixing, we show that very high-quality keying results can be achieved by an inexperienced user at one-tenth of the interaction time typically required by a professional artist using commercial tools.

2.1.2. Natural Image Matting

Natural image matting can be regarded as the generalized version of green-screen keying, as the background can take the form of a natural scene rather than a controlled studio environment. In this case, all the variables in (2.1) except for the observed image color are unknown, and hence natural image matting is a highly underconstrained problem.

This inherent ambiguity of the problem is typically alleviated through a user input called a *trimap*, which separates the image into three regions: fully foreground, i.e. opaque ($\alpha = 1$), fully background, i.e. transparent ($\alpha = 0$), and of unknown opacity, in which α values are to be estimated. With this additional user input, matte estimation becomes a color modeling problem, where the color and textural characteristics of the pure foreground and background regions are used to reason about the soft transitions between them.

Natural matting methods in the literature can be classified through how they approach this color modeling problem. Modern natural matting algorithms usually fall into three main categories:

- Affinity-based matting: using the similarity of colors between pixels to reason about the corresponding structure of the alpha matte
- Sampling-based matting: reasoning about $\boldsymbol{f}$ and $\boldsymbol{b}$ for the pixels in the unknown-opacity region using the foreground and background colors defined in the trimap to solve for α in (2.1)
- Data-driven matting: Using machine learning algorithms to directly reason about the matte

Affinity-based matting algorithms aim to propagate the opacities of the foreground and the background into the unknown-opacity region by defining pixel-to-pixel similarity metrics inside a graph structure. The use of color gradients as a clue for how opacities relate to each other in the work of Mitsunaga et al. [1995], or the use of a local color manifold in the soft transition regions in the work of Ruzon and Tomasi [2000] that defined the natural matting problem can be regarded as predecessors to affinity-based approaches. The modern formulations for affinity-based matting typically construct a

linear system that relates the opacities of local or non-local neighboring pixels through color-based affinity definitions.

One of the most fundamental approaches for relating local opacity changes with the observed colors in the original image is the work by Levin et al. [2008a]. Levin et al. derive the opacity of a pixel from the opacities of the neighboring pixels by assuming constant foreground and background layer colors in a small local window. These local relations can be formulated through a graph structure represented by *matting Laplacian*, a positive semi-definite matrix of size $N \times N$, where N is the number of pixels. They show that the trimap can be used as soft constraints in a linear system formulation together with the matting Laplacian to solve for the opacities in closed form. While the matting Laplacian is very effective in representing local soft transitions and widely used in related literature, its representative power does not generalize to complex scene structures for which the use of only local affinities is not enough.

The limitations of solely local connections can be overcome by introducing *nonlocal* affinities that relate the opacities of pixels that may be spatially far away from each other. Nonlocal matting by Lee and Wu [2011] modifies Levin et al.'s [2008a] in this spirit, by formulating an affinity matrix that connects nonlocal neighbors. KNN matting [Chen et al., 2013a] presents an approach where they solely depend on simple color and spatial proximities of pixels in varying color spaces to construct their graph structures and show that complex matte structures, such as ones that contain small holes of background in the foreground, can be represented in their model. Only using color similarity of pixels to relate their alpha values has its own shortcomings, such as a limited representational power of opacity gradients.

Chen et al. [2012] use nonlocal neighbors to represent the feature vector of a pixel, composed of its spatial coordinates and color, as a linear combination of its neighbors using locally linear embedding (LLE) [Roweis and Saul, 2000] and show that this linear relationship can be used to represent the opacity relationships by an empirical demonstration of matting results. They build upon this idea in their following work [Chen et al., 2013b] to propose a linear system better calibrated for image matting by combining the LLE-based graph with the matting Laplacian. While they demonstrate favorable results to that of KNN matting, they depend on a sampling-based method to provide an initial estimate, rather than representing the opacity transitions through a single graph structure.

The graph-based representations of opacity relationships between pixels provide flexible formulations that can be extended to related problems in natural matting such as matte refinement, layer color extraction, soft color

segmentation, and edit propagation. These extensions are covered later in this chapter in the corresponding sections.

In this thesis, we introduce an affinity-based matting algorithm that makes use of nonlocal neighbors, LLE, and matting Laplacian through careful neighborhood definitions that target challenging matte structures that appear in natural objects in Chapter 5. Further discussion of the state-of-the-art in affinity-based natural matting through spectral analysis is presented in Section 5.5.

Sampling-based approaches to matting aim to select a $\boldsymbol{f}$ and a $\boldsymbol{b}$ for each pixel in the unknown region by sampling from the known-opacity regions defined in the trimap so that α can be directly solved using (2.1). Early work on natural matting by Chuang et al [2001] called Bayesian matting defines local color distributions to represent these per-pixel samples, but more modern approaches tend to determine sets of many samples from foreground and background and select a particular pair for each pixel.

The methods in the literature focus on two main sub-problems separately: determining the sample sets and selecting samples from these sets for each pixel. Global matting [He et al., 2011] collects samples from the boundaries between the unknown and known regions in the trimap, while shared matting [Gastal and Oliveira, 2010] looks at different directions from each unknown pixel to diversify the possible sample colors. Karacan et al. [2015] and Feng et al. [2016] propose more sophisticated sampling approaches that depend on color clusters and sparsity of samples to create a more comprehensive set of samples that encompass the observed colors, while Shahrian et al. [2013] estimate local normal distributions of colors to collect their samples. For the selection of the samples, robust matting [Wang and Cohen, 2007] heavily favors spatial proximity to each unknown pixel, while Shahrian et al. [2013] propose a selection metric that combines the spatial proximity with compositing error of samples measured using (2.1) and the color similarity of the candidate samples from the foreground and the background. Once the samples are selected, the alpha values are estimated by solving (2.1) for α for each pixel. Due to the per-pixel decisions in the sample selection step and the sensitivity of the compositing equation to the accuracy and precision of the selected samples, the initial alpha estimates by the sampling-based methods suffer from spatial smoothness issues and are typically refined in an additional post-processing step.

We show that most sampling-based approaches can be outperformed by a simple k nearest neighbors search that determines many samples for each pixel in Section 5.6. This is mainly due to the sample selection process that does not generalize when there are similar colors in the foreground and

the background or give unreliable results around fully-opaque and fully-transparent regions in the state-of-the-art sampling-based methods. A recent approach by Tang et al. [2019] overcomes these issues by using deep neural networks for both the sample selection and matte estimation steps in a hybrid sampling- and learning-based approach.

Data-driven approaches to matting can vary from automatic estimation of the trimap to direct matte estimation to combine mattes estimated by various approaches. As the definition of the foreground in the trimap depends on the semantic content of the object to be matted, automatic trimap estimation methods target specific object categories, such as people. Shin et al. [2016] presents a portrait matting approach that determines the trimap through a deep neural network and obtains the matte using closed-form matting [Levin et al., 2008a]. With computational efficiency in mind, Zhu et al. [2017] replaces the matting step with a simple neural architecture that converts a hard segmentation into a soft one. Chen et al. [2018] uses two neural networks that can be trained in cascade for automatic matte estimation of people.

The conventional matting problem with the trimap given as input is addressed using deep neural networks through varying approaches. Cho et al. [2019] shows that mattes estimated by multiple methods can be combined to achieve better matting performance through the use of a deep neural network using a relatively small dataset. Xu et al. [2017] introduce a large dataset for image matting and achieve state-of-the-art matting performance through a network that takes the trimap and the image as input and produce an initial matte estimate. Their results, however, suffer from lack of sharpness in the matte despite their use of a second network targeted at this issue. Lutz et al. [2018] propose a generative adversarial network architecture that is able to produce sharp matting results. Tang et al. [2019] make use of a version of the network architecture by Lutz et al. in the final stage of their hybrid sampling- and learning-based approach, where they estimate the per-pixel color samples using network architectures that originally targeted image inpainting.

Matte refinement is usually required as post-processing after sampling-based matting that produce per-pixel initial matte estimations. The most popular refinement approach is the one proposed by Gastal and Oliveira [2010] that uses the initial estimates in a fidelity term in the linear system they construct using the matting Laplacian [Levin et al., 2008a] as the smoothing agent. We extend our matting framework for matte refinement using a linear system similar to that of Gastal and Oliveira's in Section 5.2.

Layer color estimation is required for the majority of the natural matting methods for the matte to be used in compositing applications. This is due to

the fact that natural matting typically aims only to estimate the opacities to ease the complexity of the problem. In the literature, layer color estimation methods are presented as the extensions of affinity-based methods that use similar graph structures. Levin et al. [2008a] construct their linear system to ensure spatial smoothness of layer colors around opacity transition regions. Chen et al. [2013a] proposes a graph structure that enforces similar layer colors to pixels with similar opacities and original pixel colors, following their matting formulation. We propose a layer color estimation method that uses multiple affinity definitions to better address the challenging foreground structures in Section 5.3.

2.2. Multi-Layer Soft Segmentation

The interactive approaches mentioned before is widely used in image and video editing in the industry. However, the concept of opacity estimation can also be used to represent images in terms of soft segments. These approaches focus on a specific characteristic of an image, such as the scene colors or the objects in the scene, and represent the extent of each through multiple mattes. The compositing equation shown in (2.1) can be generalized as:

$$\boldsymbol{c}_p = \sum_i \alpha_{i,p} \boldsymbol{u}_{i,p}, \tag{2.2}$$

where the index i corresponds to each layer, $\alpha_{i,p}$ and $\boldsymbol{u}_{i,p}$ are the opacity and the layer color of the i^{th} layer at pixel p, given that $\sum_i \alpha_{i,p} = 1$ representing the opaque input image. This compositing model is commonly referred to as *alpha add* representation, although other compositing models are also be used in the literature.

Although the target is the estimation of opacity channels, matting/keying and multi-layer soft segmentation have fundamental differences. Green-screen keying focuses on cleaning the foreground object off of user-defined background color, focusing on preserving the foreground details as much as possible through color modeling. Similarly, natural matting with a trimap as input becomes the problem of foreground and background color modeling, may it be through selection of color samples or propagation of color information. Meanwhile, soft segmentation focuses on identifying the soft transitions that best serve the target application, such as representing the color mixtures or semantically/spatially meaningful soft transitions in the image.

Several methods require seed pixels or regions as a starting point for soft segmentation. The multilayer matte estimation method by Singaraju and Vidal [2011] extends the closed-form matting [Levin et al., 2008a] to multiple

layers with an iterative formulation for spatial soft segmentation. The input they use can be characterized as a trimap with many different foreground regions. KNN matting [Chen et al., 2013a] can also be used for soft segmentation when multiple pixels are used as seeds for each segment. With their nonlocal affinity definition that relates pixels in terms of their spatial proximity as well as color similarity, the soft segmentation performed by KNN matting can be classified as a hybrid spatial- and color-based soft segmentation. A similar hybrid soft segmentation was done through an expectation maximization formulation for color editing by Tai et al. [2005].

Soft color segmentation, first proposed by Tai et al. [2007], aims to represent the input image in terms of color mixtures. In this approach, each layer is expected to have a homogeneous color, i.e. $\boldsymbol{u}_i$ should take similar values across the image for each layer i. The set of colors that define each layer's color content is usually called the *color model*. Tai et al [2007] propose an alternating optimization technique using a Markov random field formulation which estimates the opacities, the layer colors, and the color model alternatingly in an iterative scheme. The RGB-space geometry approach by Tan et al. [2016] first fixes the color content of each segment with a hull that encompasses the color values that appear in the input image and then estimates the opacities.

The aforementioned soft segmentation methods can generate representations that allow for easy image editing and compositing, but they have shortcomings in terms of spatial smoothness, computational complexity, color homogeneity, or the required user input. We provide an in-depth analysis of these methods in Section 4.2.

We extend the color unmixing formulation used in green-screen keying for generic soft color segmentation in Chapter 4. We demonstrate that the proposed method outperforms related work in terms of computational complexity, layer quality, and color homogeneity.

Levin et al. [2008b] introduced an automatic spatial soft segmentation formulation through the spectral analysis of the matting Laplacian. They show that the eigenvectors that correspond to the smallest eigenvalues of the matting Laplacian reveal the spatially coherent regions in the image together with the soft transitions between them. We augment the matting Laplacian with high-level information on *objectness* coming from a deep neural network and extend Levin et al.'s formulation to propose a novel soft segmentation paradigm, semantic soft segmentation, in Chapter 6.

Part I.

Color Unmixing

CHAPTER 3

Interactive High-Quality Green-Screen Keying via Color Unmixing

We study the problem of green-screen keying through the lens of color mixtures that represent many of the challenges in creating a foreground that is completely cleared from the controlled background color. We will first detail the *color unmixing* energy and its optimization which estimates the weight of each color that went into the color mixture at each pixel. The color unmixing formulation depends on a parametric representation of all the scene colors that we refer to as the *color model*. The color model is acquired via a two-step user interaction scheme, as detailed in Section 3.2. The main steps of our keying pipeline can be seen in Figure 3.1. We also give a detailed example on the common practice in green-screen keying in the industry to put the advantages of our approach in perspective in Section 3.3.

3.1. Color Unmixing

The central component of our method is an energy minimization framework, where the color $\boldsymbol{c}$ of a pixel is hypothesized to be a mixture of a number of *underlying colors* $\boldsymbol{u_i}$. The problem solved by our framework is the estimation of the underlying colors and their mixing ratios (α_i), such that the linear combination of the underlying colors weighted by corresponding mixing ratios gives the original pixel color $\boldsymbol{c}$. To that end, we build and utilize a parametric representation of all the colors present in the scene which we refer simply as the *color model*. The color model comprises N distributions in

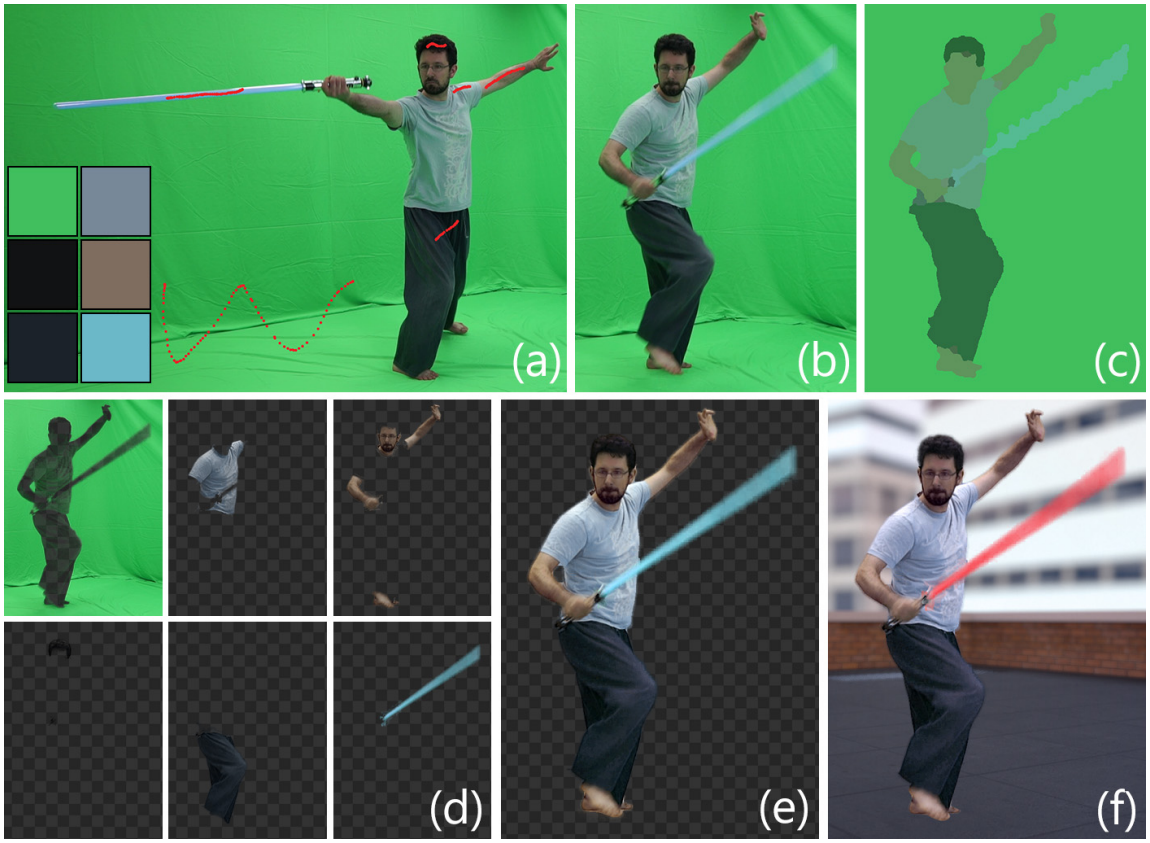

Figure 3.1.: *Major steps of our method. First, parameters of a global color model are obtained from a key frame via a simple scribble interface (a) (Section 3.2.1). For a different query frame (b), the global color model is refined into local color models (c) (Section 3.2.2) which are utilized for extracting multiple color layers via color unmixing (d) (Section 3.1). A subset of layers is then combined to get the final keying result (e). The layers can be used for compositing as well as color editing (f).*

RGB space. Both the number and the parameters of these distributions are obtained through user interaction. We assume that the color model for an input scene is already known to us throughout this section, and rather focus on the formulation and efficient solution of the color unmixing problem. A detailed discussion on building the color model of an input scene will follow in Section 3.2.

We start formulating our color unmixing framework by defining three basic constraints that each pixel should satisfy: (i) an *alpha constraint* which states that the alpha values α_i should sum up to unity, (ii) a *color constraint* which states that we should obtain the original color $\boldsymbol{c}$ of the pixel when we mix the underlying colors $\boldsymbol{u_i}$ using the corresponding alpha values, and (iii) a *box constraint* that limits the space of possible alpha and color values. Formally,

we express these constraints as follows:

$$\sum_i \alpha_i = 1, \quad \sum_i \alpha_i \boldsymbol{u}_i = \boldsymbol{c}, \quad \text{and} \quad \alpha_i, \boldsymbol{u}_i \in [0,1]. \tag{3.1}$$

The cost associated with the occurrence of an underlying color $\boldsymbol{u_i}$ in a mixture $\boldsymbol{c}$ is defined by how well it fits to the corresponding distribution $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$, where μ and Σ denote the mean vector and the covariance matrix, and $\mathcal{N}$ is the normal distribution. We use the squared Mahalanobis distance as our measure of goodness of fit:

$$\mathcal{D}_i(\boldsymbol{u}) = (\boldsymbol{u} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\boldsymbol{u} - \boldsymbol{\mu}_i), \tag{3.2}$$

and define our energy function $\mathcal{F}$ of selecting a particular mixture of N underlying colors accordingly as follows:

$$\mathcal{F} = \sum_i \alpha_i \mathcal{D}_i(\boldsymbol{u_i}). \tag{3.3}$$

This energy function favors layer colors that have the best likelihoods according to their corresponding color distributions, especially for the layers with higher alpha values. Minimization of this energy subjected to the color constraint makes sure that the resultant layers successfully represent the color mixture that formed the observed pixel color. We minimize the color unmixing energy for α_i and $\boldsymbol{u_i}$ simulatenously via the process detailed in Section 3.1.1.

If we visualize the i^{th} underlying colors for all pixels of the video frame with their alpha values, we obtain the RGBA layer corresponding to the distribution $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$. Green-screen keying can be seen as a special case where we remove the RGBA layer corresponding to the green-screen background.

In contrast to our color unmixing method, related works on parametric natural matting use Bayesian formulations with either local [Ruzon and Tomasi, 2000; Chuang et al., 2001] or global models [Tai et al., 2007]. Local methods solve for alpha values first and then estimate colors. On the other hand, Tai et al. [2007] iteratively estimates the alpha values and colors for all pixels of an input image, which makes it feasible for only low-resolution images. In contrast, our formulation is easily parallelizable as each pixel is treated independently, and thus, our method easily scales to HD resolutions and beyond.

Sampling-based natural matting methods such as comprehensive sampling [Shahrian et al., 2013] take alternate approaches to compute foreground layer colors where they try all the possible background-foreground color pairs to get the best match from a limited set of color samples. Certain priors

commonly utilized by these methods, such as matte sparsity [Wang and Cohen, 2007; Gastal and Oliveira, 2010], are often violated in green-screen keying due to color spill.

On the other hand, commercial chroma-based keying tools simply suppress the background green-screen color everywhere in the frame, which often distorts the colors of the foreground objects especially if they are similar to the color of the green-screen background. Around intricate object boundaries or motion blur, they extend the foreground region without actually unmixing the colors, and as a result they leave an unnatural halo around difficult regions.

To summarize, the natural matting methods in the literature, as well as commercial keying tools, fail to achieve production-level quality in green-screen keying due to their various shortcomings discussed above. The main advantages of our color unmixing formulation are the following:

- Our method does not enforce a matte-sparsity constraint, nor rely on the suppression of the color of the green-screen background.
- Our formulation is highly scalable and parallelizable as each pixel is processed independently.
- The proposed energy minimization successfully unmixes even mixtures of very similar colors (demonstrated later in Section 3.4.1) and is agnostic to the scene colors, i.e. we do not require a strong chroma or luma component as in commercial software.
- Similar to KNN Matting [Chen et al., 2013a], our method computes multiple RGBA layers as its output, which enables further interesting applications beyond green-screen matting, such as color editing.

3.1.1. Minimization of the Color Unmixing Energy

While the color unmixing energy $\mathcal{F}$ in (3.3) may seem straightforward, we found that its minimization is non-trivial. Since the energy function $\mathcal{F}$ and the color constraint defined in (3.1) are nonlinear, we are faced with a nonlinearly constrained nonlinear optimization problem. Specifically, the color constraint in (3.1) constrains a single alpha value for three unknown underlying color channels at once. This makes our energy function $\mathcal{F}$ prone to get stuck in local minima within the vicinity of the initial point if the constraints are enforced from the start. What we need instead is an algorithm that strictly enforces the constraints only after allowing to find some reasonable alpha and color values first. To that end, we utilize the method referred as *the original method of multipliers* [Bertsekas, 1982] as outlined in Algorithm 1.

Algorithm 1: The Original Method of Multipliers

Input: $\boldsymbol{x}_0$

Define: $k = 0$, $\rho_0 = 0.1$, $\boldsymbol{\lambda}_0 = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$, $\beta = 10$, $\gamma = 0.25$, $\epsilon > 0$

1: $\boldsymbol{x}_{k+1} = \arg\min_{\boldsymbol{x}} \left(\mathcal{F}(\boldsymbol{x}) + \boldsymbol{\lambda}_k^T \boldsymbol{\mathcal{G}}(\boldsymbol{x}) + \frac{1}{2} \rho_k \| \boldsymbol{\mathcal{G}}(\boldsymbol{x}) \|^2 \right)$

2: $\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \rho_k \boldsymbol{\mathcal{G}}(\boldsymbol{x}_{k+1})$

3: $\rho_{k+1} = \begin{cases} \beta \rho_k & \text{if } \| \boldsymbol{\mathcal{G}}(\boldsymbol{x}_{k+1}) \| > \gamma \| \boldsymbol{\mathcal{G}}(\boldsymbol{x}_k) \| \\ \rho_k & \text{otherwise} \end{cases}$

4: **if** $\| \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \| > \epsilon$ **then**

5: $\quad k \leftarrow k + 1$

6: $\quad$ **go to** Step 1

7: **else**

8: $\quad$ **return** $\boldsymbol{x}_{k+1}$

In order to pose the constraints as soft constraints at the beginning of the optimization process, we express the deviation from the constraints in (3.1) as:

$$\mathcal{G}_\alpha = \left(\sum_i \alpha_i - 1 \right)^2 \quad \text{and} \quad \boldsymbol{\mathcal{G}}_u = \left(\sum_i (\alpha_i \boldsymbol{u}_i) - \boldsymbol{c} \right)^{\bullet 2}, \tag{3.4}$$

where $(\cdot)^{\bullet 2}$ denotes the elementwise squaring operation. This leads to the constraint vector $\boldsymbol{\mathcal{G}} = \begin{bmatrix} \boldsymbol{\mathcal{G}}_u^T & \mathcal{G}_\alpha \end{bmatrix}^T$. The vector containing the variables $\boldsymbol{x}$ that are the arguments of the optimization is:

$$\boldsymbol{x} = \begin{bmatrix} \alpha_1 & \dots & \alpha_N & \boldsymbol{u}_1^T & \dots & \boldsymbol{u}_N^T \end{bmatrix}^T. \tag{3.5}$$

Note that $\boldsymbol{x}$ contains the variables for unmixing a single pixel. The optimization is performed independently for every pixel, where for each pixel we solve for both the alpha values and the underlying colors simultaneously.

The function minimized in Line 1 of Algorithm 1 is composed of the original energy function and the deviations from the constraints. Minimization at this step is done using the nonlinear conjugate gradient method that takes $\boldsymbol{x}_k$ as the initial value. The step size of the nonlinear conjugate gradient at each iteration is determined by a line search in the direction determined via the Polak–Ribière formula. The box constraints are enforced at each iteration of the nonlinear conjugate gradient method by clipping the elements to be in the range $[0, 1]$ and setting the gradients of the elements at the boundaries 0 and 1 to zero if they are positive or negative, respectively. As the parameters $\rho_{(\cdot)}$ and $\boldsymbol{\lambda}_{(\cdot)}$ increase at each iteration (Lines 2 and 3), the energy $\mathcal{F}(\boldsymbol{x})$ is minimized while allowing smaller and smaller deviations from the alpha

and color constraints in Line 1. $\lambda_{(\cdot)}$ punishes deviation from individual constraints, while $\rho_{(\cdot)}$ increases the constraint enforcement globally. The input to Algorithm 1, initial values for α_i and $\boldsymbol{u}_i$, are defined as:

$$\alpha_i = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \qquad \boldsymbol{u}_i = \begin{cases} \boldsymbol{c} & \text{if } i = j \\ \boldsymbol{\mu}_i & \text{otherwise} \end{cases},$$

where $j = \arg\min_i \mathcal{D}_i(\boldsymbol{c})$, *i.e.* only the alpha value corresponding to the most likely distribution in the color model is initialized to be 1. Note that the optimization procedure we described is independent for each pixel in an image.

3.2. Building the Color Model

The energy function $\mathcal{F}$ we defined in (3.3) requires a parametric representation of the colors that formed the color mixture which we refer as the color model. A set of distributions is obtained in the first step of the user interaction of our method. The resulting *global* color model (Section 3.2.1) is assumed to be able to represent the whole image. The global color model is locally overcomplete since very often each pixel color $\boldsymbol{c}$ is a mixture of only a subset of the scene colors. We call the subset of distributions that participate in the color mixture in a certain region of an image as the *active color distributions*. In Section 3.2.2, we refine the global color model such that each pixel is associated only with its active color distributions. This refinement process is performed automatically by utilizing a Markov Random Field optimization, but we also allow the user to edit the resulting *local* color models in a second user interaction step.

In comparison, commercial green-screen matting software packages offer a multitude of interaction modes ranging from background/foreground color selection to rotoscoping interfaces. They also typically offer user control over various parameters that control the amount of chroma suppression, matte blurring or matte bleed. Although this high level of control allows compositing artists to fine-tune keying results, it also makes the process highly time-consuming.

The goal of the user interaction in our method is to extract the information we need to build the color model as intuitively and efficiently as possible. Consecutively, instead of relying on complex user interactions like commercial keying tools, we utilize a two-step interaction that involves drawing a small number of scribbles (typically 7-8) and a pointing-and-clicking step.

Figure 3.2.: *The keying results generated using four different scribbles demonstrate the robustness of our algorithm against different user inputs.*

3.2.1. Global Color Model

The user interaction typically starts with the user loading the first frame of an input video using the interface of our method. The goal of the first user interaction step is building the global color model, which is achieved by the user drawing a scribble over each of the *dominant* scene colors. The number of the scribbles N, and hence the number of dominant scene colors, is determined by the user depending on the scene. For example, in Figure 3.9 (Our result, input) each different color on the person's wig is selected separately as a dominant color, whereas in Figure 3.8 (Our result, input), the actor's natural hair color is marked as a single dominant color.

Each scribble identifying a dominant color is used to extract the parameters of a distinct normal distribution. The mean and covariance of each distribution are computed simply from the pixels underneath the corresponding scribbles. Importantly, the results of our color unmixing method are not sensitive to the exact placement, size or shape of the scribbles, as demonstrated in Figure 3.2. This property is very useful in practice, as high-quality results can be obtained quickly from roughly drawn scribbles. Additionally, once the global color model is created for a single frame, it can typically be used for the remaining frames of the shot assuming the dominant colors do not change significantly.

The motivation behind this first user interaction step is utilizing the inherently good cognitive skills of the users for clustering colors. These cognitive skills are especially helpful in dealing with specific situations such as the presence

of strong color spill. Figure 3.14 (Original) shows an example where the color of the actor's robe is affected by the indirect illumination from the green-screen, except for only very few small regions. In this case, recognizing the color spill and selecting unaffected regions as a dominant color are trivial for a human user while the same tasks are extremely difficult for an automatic color clustering algorithm. In fact, although we experimented with methods for automatically building the global color model (see Section 3.4.3), we found that in most practical cases user interaction would be necessary, and therefore favored our current interactive approach. The ability to select the dominant colors also gives the user artistic control over the color composition of the resulting RGBA layers, which is especially useful for compositing artists.

3.2.2. Local Color Model

One shortcoming of the global color model is the assumption that the color of each pixel of the input video is a mixture of N underlying colors from the N distributions that make up the color model. However, in practice, this assumption is almost always incorrect. For example, in the original image in Figure 3.3, skin tones are only present in a small region near the actor's face and neck. If we solely rely on the global color model, we would have to use the distribution corresponding to the skin tones for unmixing pixels in completely unrelated image regions, such as the far edges of the green-screen background. This may cause the color unmixing to hallucinate non-existent colors with small alpha values in such regions. Thus, we perform a Markov Random Field (MRF)-based optimization procedure over superpixels to estimate the active subset of color distributions for different regions in an image.

The result of this optimization procedure can be edited through user interaction via a simple point-and-click interface. Since the automatic color activation is rather computationally costly, and it would be cumbersome to perform the local color model edits repeatedly for every frame, we propagate the local color model of an edited frame to the following frame through simple superpixel matching. For every superpixel in a new frame, we find a corresponding superpixel in the previous frame in a small spatial neighborhood with the closest mean color. The active distributions of a superpixel in the new frame is defined as the active distributions of its match in the previous frame. An example local color models, a typical user edit, and propagation to consecutive frames are illustrated in Figure 3.3.

The local color model computation step can loosely be related to the sample selection process employed by sampling-based natural matting methods

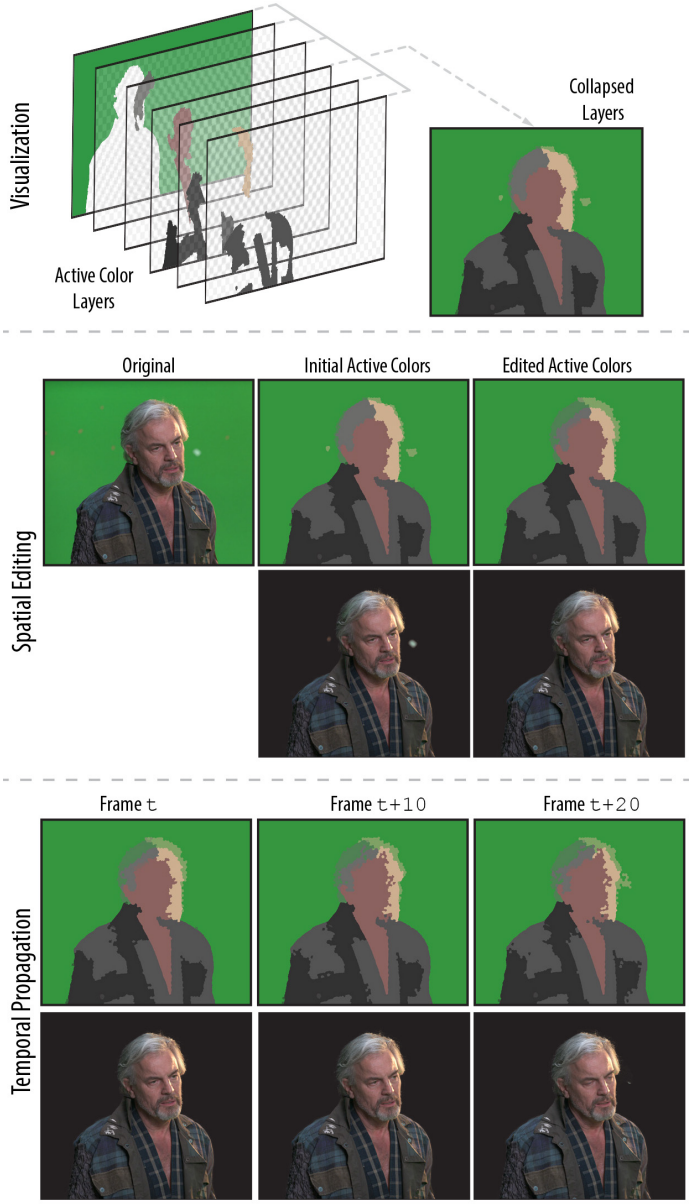

Figure 3.3.: ***Visualization:*** *The result of local color model estimation can be visualized as a cascade of layers that illustrate the active color distributions by their mean colors.* ***Editing:*** *The MRF optimization for the local color models may fail to distinguish between different objects with similar colors (such as the markers and the actor's face), or may give suboptimal results when one of the colors is present only faintly in a region (such as the color spill in the actor's hair from the green screen). Such situations can be alleviated by refining the local color model via a simple point-and-click user interface.* ***Propagation:*** *The user interaction can be streamlined by propagating the local color model to consecutive frames.*

such as shared sampling [Gastal and Oliveira, 2010], where the goal is also to find the best-fitting distributions for every pixel. However, their brute-force approach is fundamentally different from our MRF optimization process.

Several natural matting methods such as comprehensive sampling [Shahrian et al., 2013] utilize localized color models. While we select a subset of the global color model as the local color model, comprehensive sampling estimates a set of normal distributions from the closeby foreground and background regions for a mixed-color pixel. Although this approach provides some robustness against complex backgrounds, it has several shortcomings in the green-screen keying case. Under heavy color spill, estimating distributions locally is typically insufficient since the pure-color regions may occur in a very limited part of the image and can not be integrated into the local models. It also inherently increases the number of necessary distributions to represent the image, making the direct user-edits inconvenient if not impossible. The resulting localized layers then require additional temporal coherency steps to be applied to image sequences, since spatially they are expected to change from frame to frame. Hence, we found our definition of local color models as a subset of a global model to be practically well-fitting to our target application of green-screen keying.

Local color model estimation

We represent the active distributions of a pixel as a binary vector $\boldsymbol{A}$ of length N, and define the cost of activating a subset of distributions for a pixel as the sum of two terms. The first term is the minimum energy defined in (3.3) when the subset of distributions are fed to the energy minimization algorithm detailed in Section 3.1, denoted by $\mathcal{F}_{\boldsymbol{A}}$. The intuition here is that if the optimization is conducted with distributions that fail to effectively represent an observed color, the minimized energy will still be high. The second term $\mathcal{G}_{\boldsymbol{A}} = \|\boldsymbol{A}\|$, $\|\cdot\|$ representing the Euclidean norm, is added to this cost in order to favor fewer active colors for each pixel. Following these definitions, the unary potentials are defined as:

$$\mathcal{U}_{\boldsymbol{A}} = \mathcal{F}_{\boldsymbol{A}} + \delta\mathcal{G}_{\boldsymbol{A}}, \tag{3.7}$$

where δ is a user specified weight parameter typically in the range [5 10]. The binary potentials between neighboring pixels are defined as:

$$\mathcal{B}_{p,q} = \|\boldsymbol{A}_p - \boldsymbol{A}_q\| e^{-\|\boldsymbol{c}_p - \boldsymbol{c}_q\|}. \tag{3.8}$$

The energy function we want to minimize in order to determine active color distributions is:

$$\mathcal{E} = \arg\min_{A_{(\cdot)}} \sum_{p} \mathcal{U}_{A_p} + \sigma \sum_{(p,q)\in\Omega} \mathcal{B}_{p,q}, \tag{3.9}$$

where σ is the smoothness parameter, typically selected in the range $[0.01\ 0.05]$ and Ω is the set of 8-connected pixels.

The problem we defined in this section is analogous to multi-label segmentation if we treat each possible subset of active color distributions as a label. The minimization of the energy defined in (3.9) is NP-hard [Boykov et al., 2001]. We approximate the global solution of this energy minimization using $\alpha - \beta$ *swap* algorithm presented in [Boykov et al., 2001], using the publicly available implementation by the authors [Kolmogorov and Zabih, 2004; Boykov and Kolmogorov, 2004].

Although we presented our energy formulation in this section at the pixel level, computing $\mathcal{F}_A$ for every subset and every pixel can be time-consuming especially if N is high. In order to make the local color model estimation more efficient, we instead construct the random field using SLIC superpixels [Achanta et al., 2012] (typically 10k superpixels for a 1080p frame). This allows a user controllable trade-off between quality and computational efficiency.

3.3. Common Practice in Green-Screen Keying

The industrial application of green-screen keying is done by digital artists specialized in keying and compositing. The keying artists utilize a multitude of commercial tools in orchestration. While this process may change from artist to artist as well as from scene to scene, to see an keying with commercial software in action, we asked an independent keying artist to give step-by-step description of his work for an example image sequence.

Figure 3.4 shows several frames from the image sequence at the top, and several screen captures from each step the artist took in the process. The approximate time the artist had to spend at each step is also shown. The process starts with extracting a *clean plate*, a full frame of only the background without the actor, by combining different frames together (a). The clean plate is very helpful for the following steps, but its estimation is only possible thanks to the stationary camera setup. The artist then generates several initial alpha estimates using different tools, including IBK (b) and Primatte (d). Luma keying, a keying method that depends on the brightness of the pixels rather than their color, is utilized to capture the dark areas such as the actor's

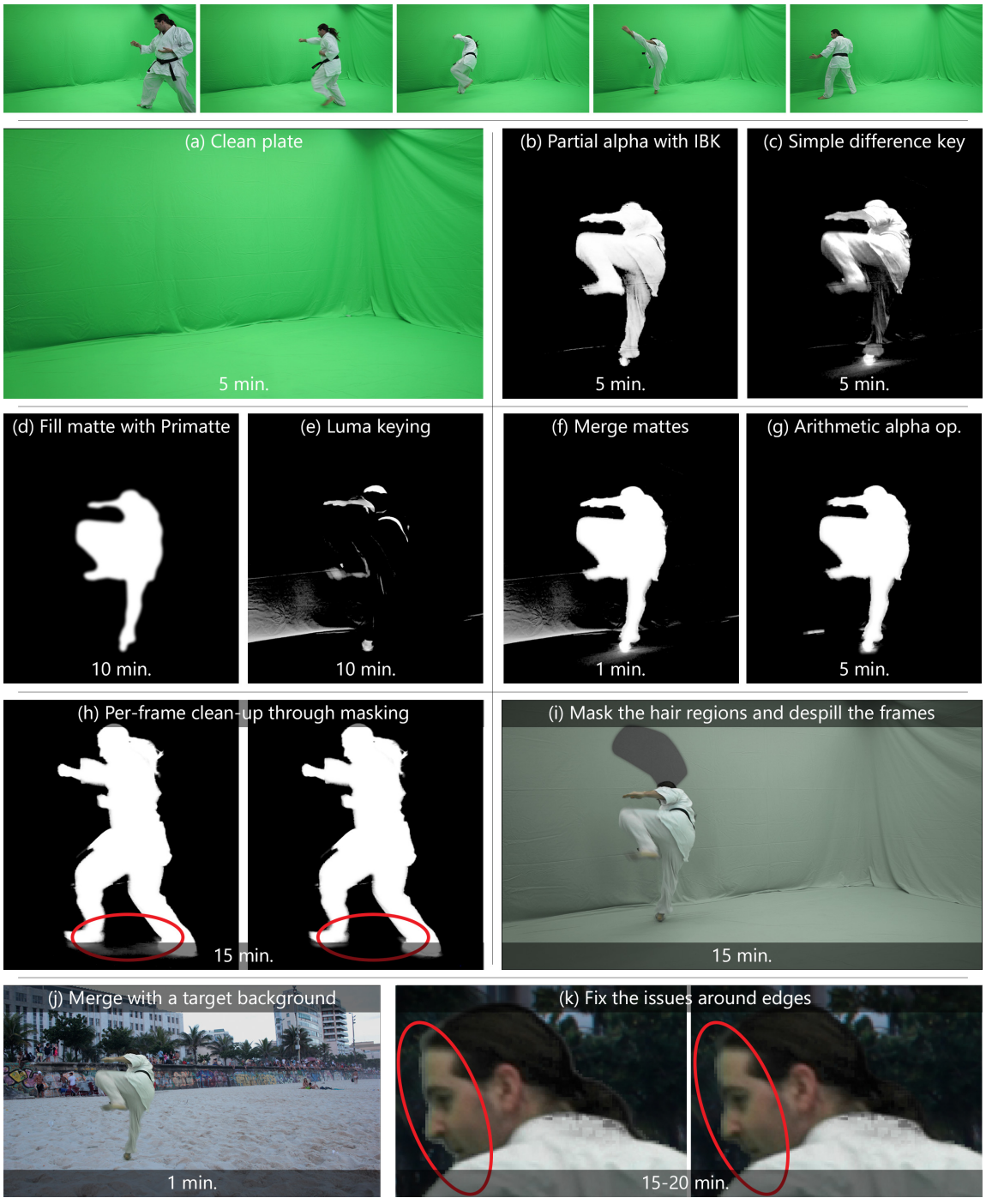

Figure 3.4.: *Step-by-step description of the keying process for the video shown at the top by an independent professional keying artist using multiple industrial keying software.*

hair and belt (e). These initial mattes are merged (f) and cleaned using simple tools (g).

The steps until now provide a very rough estimate of the foreground matte. The artist moves on to fix errors frame by frame by drawing masks by hand (h). Then the color spill on the actor is removed with a special attention to the actor's hair (i). Although we are interested in the keying result only, the artist informed us that the following corrections are typically done on a target background to identify the most visible issues with the keying result (j). In the final step, the overestimation of alpha values around the edges of the foreground is fixed frame by frame. While the matte is quite clean at this stage, the artist informed us that this result is only used as an initial point in a compositing framework, and further errors are fixed by *painting the matte* for the optimal result in a very time demanding procedure.

In contrast, our two-step interaction process determining the color models is intuitive for a non-expert, does not change from scene to scene, and takes one-tenth of the total interaction time when compared to the industrial keying procedure. In Section 3.4, we show that our keying results are on par with, if not better than, the use of commercial software.

3.4. Experimental Evaluation

Our method is suitable for parallel computation as discussed in Section 3.1. For a 1080p frame, our current C++/CUDA implementation typically requires 10 seconds for local color estimation (assuming 8 dominant colors), another second to propagate the local color model to the following frame, and approximately 3 seconds for color unmixing. Thus, at this resolution, the total computation time for a still image is 13 seconds, which drops to 4 seconds per frame for image sequences.

In this section, we evaluate our method and present results for various applications. In the absence of a comprehensive ground truth dataset of green-screen content, in our experiments, we utilize computer generated ground truth, as well as keying results generated by a paid independent professional compositing artist. In contrast, all user interaction with our method was performed by people with no prior experience in digital keying or compositing.

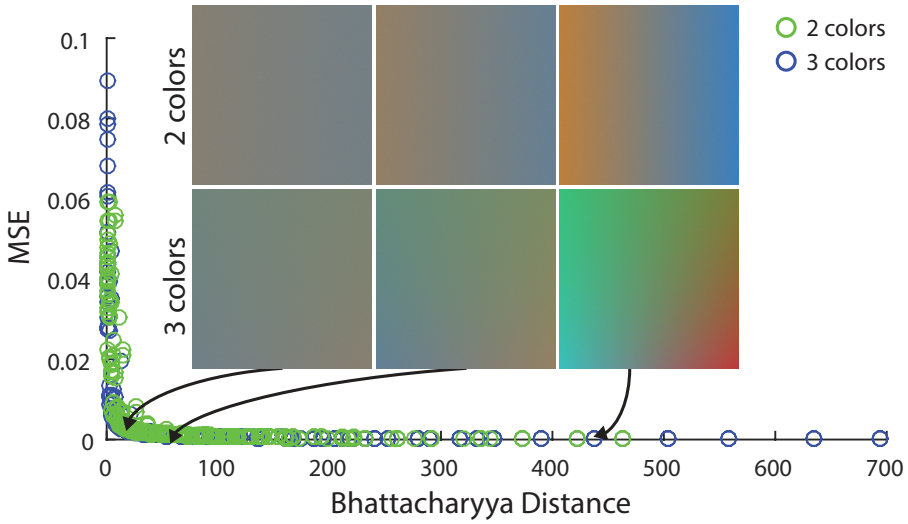

Figure 3.5.: *The mean squared error plotted with respect to the distance between the distributions in the color model. The rightmost images show two cases where distributions are very distinct. The leftmost images are at the point that our energy function starts to fail at discriminating between colors, effectively illustrating the limits of the proposed color unmixing.*

3.4.1. Statistical Validation

In this experiment, we test how distinct two colors have to be for our unmixing algorithm to work successfully. To that end, we generated a total of 480 images, each obtained overlaying 2 or 3 images created by randomly sampling from one of 720 different normal distributions with varying mean vectors and covariance matrices. The images were overlayed via a known alpha matte, which served also as the ground truth. Examples of these test images are shown in Figure 3.5. For the distinctiveness measure, we use Bhattacharyya distance that models the amount of overlap between two normal distributions. For two distributions $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$ and $\mathcal{N}(\boldsymbol{\mu}_j, \Sigma_j)$, Bhattacharyya distance is defined as

$$\frac{1}{8}(\mu_i - \mu_j)^T \Sigma^{-1} (\mu_i - \mu_j) + \frac{1}{2} \ln \frac{\det\left(0.5\left(\Sigma_i + \Sigma_j\right)\right)}{\sqrt{\det\left(\Sigma_i\right) \det\left(\Sigma_j\right)}} \tag{3.10}$$

Figure 3.5 shows that our method can successfully unmix colors up to a point when they become hard to distinguish by a human observer.

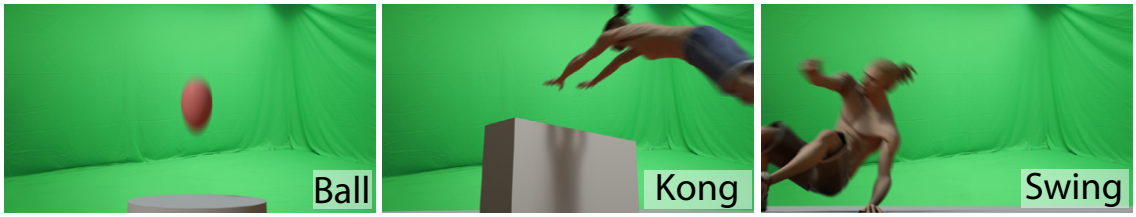

Figure 3.6.: *Three animated image sequences are overlayed onto a challenging green-screen in order to create data with ground truth.* Ball *represents a simple scene with high motion blur, while* Kong *and* Swing *represent live action scenes with fast motion.*

Table 3.1.: *Quantitative comparison of the proposed algorithm with industrial keying tools using minimal or optimal user interaction.*

	1000 × MSE Color			1000 × MSE Alpha			
	Ball	*Kong*	*Swing*	*Ball*	*Kong*	*Swing*	
IBK	0.0170	0.0553	0.1139	0.5353	0.5954	2.1232	**Minimal**
Keylight	1.6001	0.5247	0.4831	2.3645	1.3389	2.7036	
Primatte	5.8097	1.6830	27.0635	6.8404	2.6980	32.0337	
Ours	**0.0096**	**0.0250**	**0.0489**	**0.1286**	**0.4114**	**1.2722**	
IBK	0.0129	0.0504	0.1658	0.0583	0.1510	0.2291	**Optimal**
Keylight	0.0239	0.0842	0.1301	0.0200	0.4841	0.1583	
Primatte	0.0492	0.2348	0.2587	0.1391	0.5487	0.6166	
Ours	**0.0034**	**0.0189**	**0.0304**	**0.0089**	**0.0421**	**0.0585**	

3.4.2. Evaluation on Synthetic Video

Due to the absence of ground-truth data for green-screen keying, we prepared a test set of computer generated video sequences (Figure 3.6) rendered with a live-action green-screen in the background. We used this ground truth data to compare the performance of our method with three leading commercial keying tools (IBK, Keylight, and Primatte). In a first experiment, we compared the *out-of-the-box* performance by providing only minimal user input to all methods, i.e. by selecting a reasonable background color for the commercial tools and selecting 5-9 dominant colors for our method.

In a second experiment, we asked a paid compositing artist to generate the best possible result separately with each commercial tool. The artist reported spending 105-120 minutes with each tool. For comparison, we also processed the same sequences with our method to achieve the best possible keying result, for which we spent 10 minutes mostly refining the local color maps.

Table 3.1 shows that our keying results are objectively better than the three commercial tools for all test sequences, both with minimal and optimal level

Figure 3.7.: *The results of our algorithm when the color model is inferred from the scribbles (b) and when the color model is estimated by expectation maximization using different numbers of distributions (10 for (c), 6 for (d) and 4 for (e)). The EM algorithm is run using all the pixels in a small region of interest (a). The highlighted colors are the colors estimated by EM that are closest to our original four distributions.*

of user input. In some cases, such as the performance of Primatte in the Swing sequence, we observed that further processing by the artist is essential to get a more reasonable result, which means that for a novice user, it is harder to get a good initial estimate. Note also that the user interaction of our method is an order of magnitude more efficient when one seeks to obtain the best possible result.

3.4.3. Color Model Estimation using EM

As an alternative to scribble-based interaction to infer the global color model, we tried to estimate the distributions using expectation maximization.

The main problem with expectation maximization is that it is unable to separate the areas with color spill (indirect illumination from the green-screen material) from the clean areas. As Figure 3.7 shows, the distribution corresponding to the white robe of the actor appears greenish regardless of the number of distributions estimated by EM. This is expected since the pure white color appears in very limited regions while the greenish white is dominant due to the strong color spill. Using our scribble interface, the user can select regions without color spill and our color unmixing algorithm is able to separate the spill from the robe.

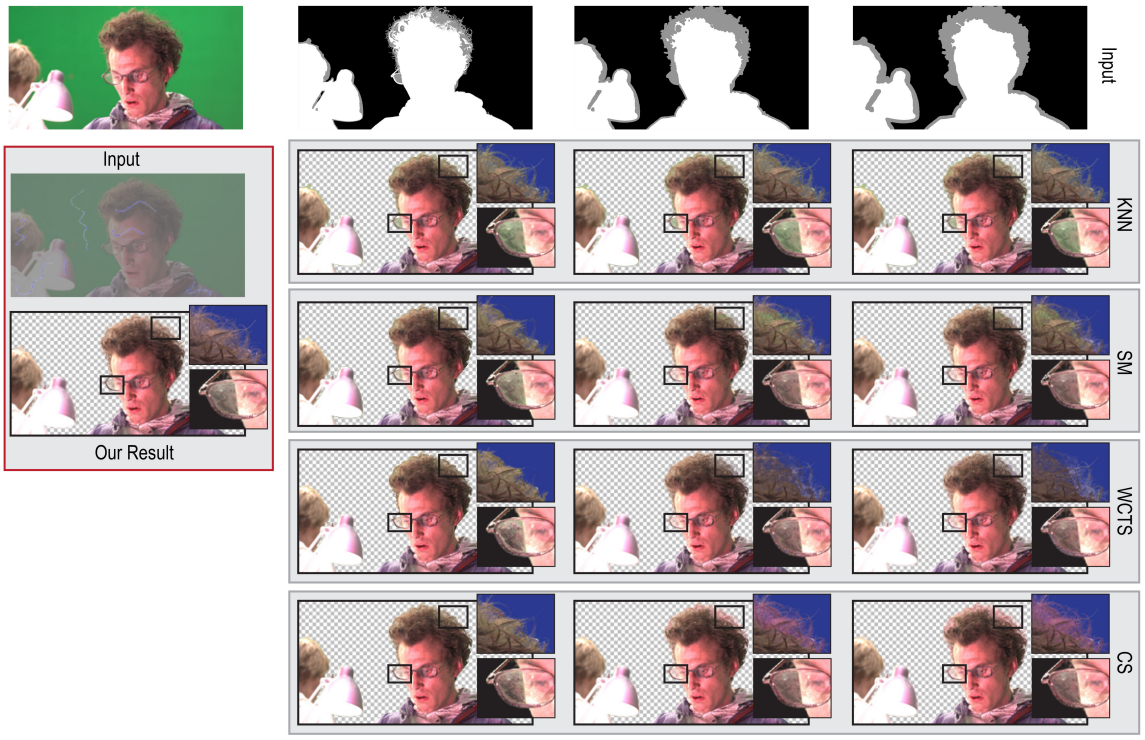

Figure 3.8.: *Results of KNN matting [Chen et al., 2013a] (KNN), shared matting [Gastal and Oliveira, 2010] (SM), weighted color and texture sampling [Shahrian and Rajan, 2012] (WCTS) and comprehensive sampling [Shahrian et al., 2013] (CS) are presented using different trimaps together with our input scribbles and keying results. Note that our scribbles are drawn on the first frames of the corresponding videos. Plate (a) shows an example with intricate object boundaries as well as translucent regions, and (b) shows another example with many foreground colors which also include a green tone close to the background color.*

3.4.4. Green-Screen Keying

Comparison with natural alpha matting methods

We compare our method to four natural matting methods with publicly available implementations. All four methods, namely KNN matting (KNN) [Chen et al., 2013a], shared matting (SM) [Gastal and Oliveira, 2010], weighted color and texture sampling (WCTS) [Shahrian and Rajan, 2012] and comprehensive sampling (CS) [Shahrian et al., 2013] compute not only alpha values but also the corresponding foreground colors. For this comparison, following the procedure from the alpha matting benchmark [Rhemann et al., 2009], we first prepared a very detailed and narrow trimap and dilated the unknown regions by 6 and 12 pixels to obtain two additional trimaps. For scenes with substantial color spill, we prepared two sets of trimaps, where one considers

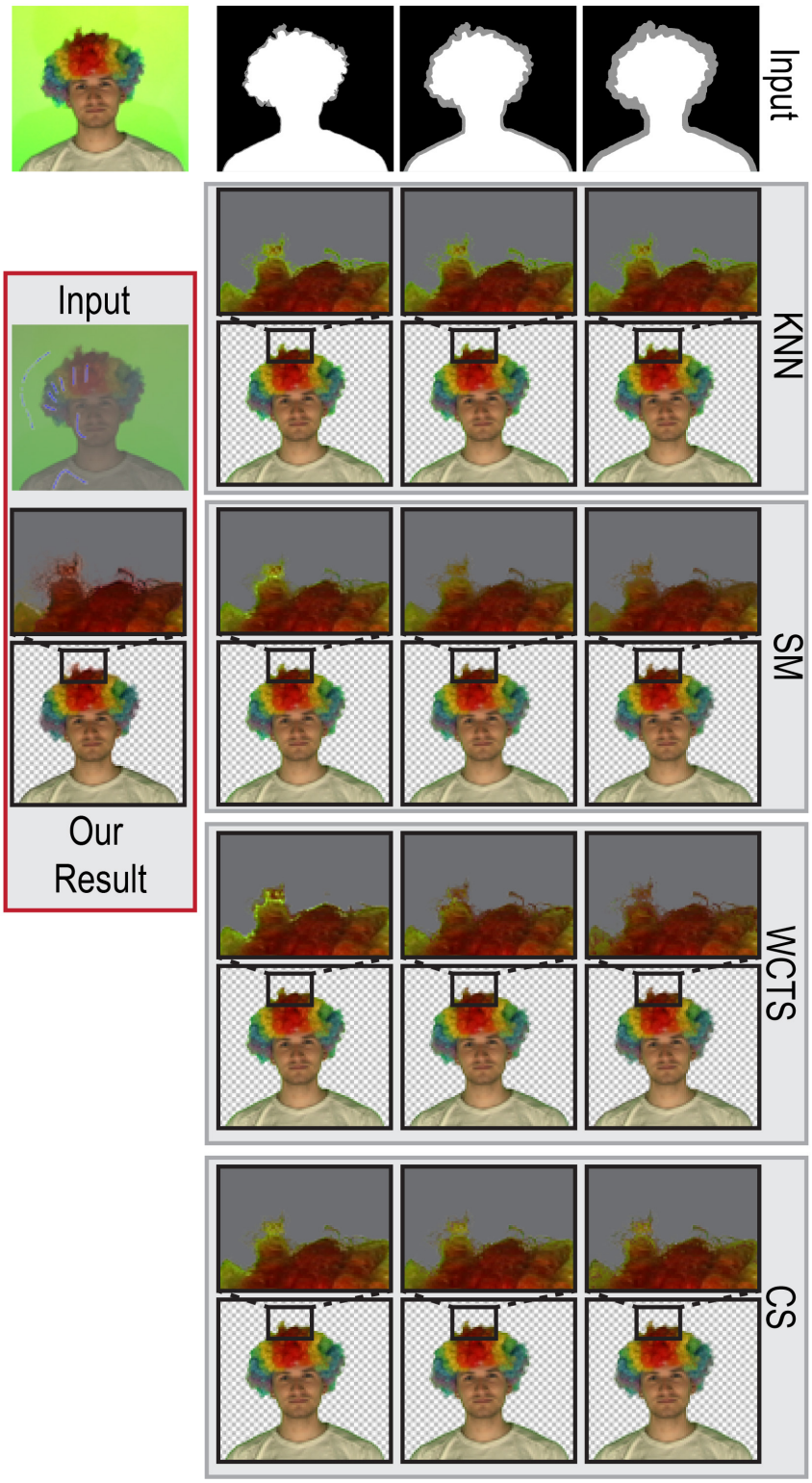

Figure 3.9.: *Figure 3.8 continued.*

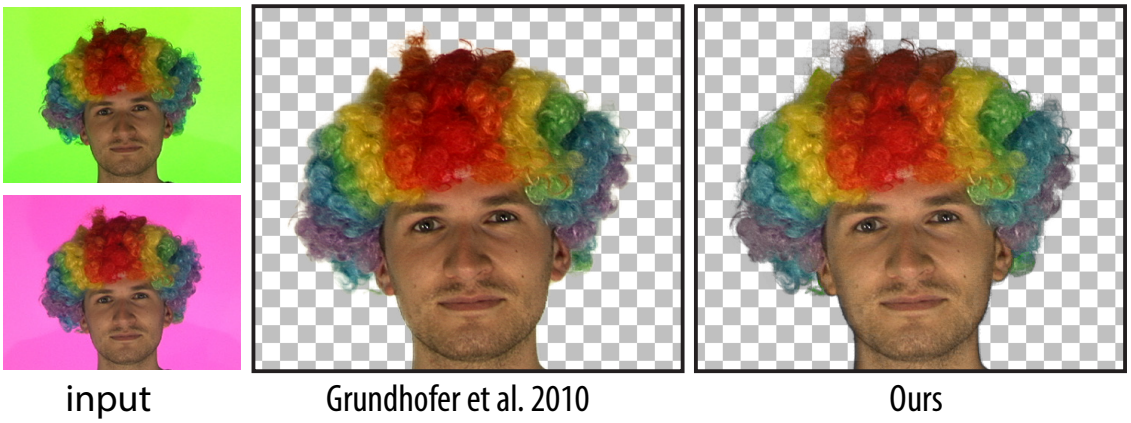

Figure 3.10.: *Our result obtained using only the image with the green background is comparable to Grundhöfer et al. [2010]'s result obtained with both input images.*

the regions with spill as unknown, and the other as foreground. The final trimaps and corresponding results can be seen in Figure 3.8.

The intricate object boundaries in Figure 3.8 demonstrate a fail case for sample selection strategies of WCTS and CS as they partly use samples from the actor's face rather than his hair, causing the hair to appear to have a red hue. SM gives the cleanest result in this case among the natural matting methods. Figure 3.9, shows that the presence of the color green on the actor's wig degrades the performance of KNN, WCTS, and CS while the local color model assumption of SM helps to extract a cleaner foreground. However, SM fails to extract the fine details as our method does, possibly due to the sparsity assumption of SM.

The scenes shown in Figures 3.8 and 3.9 are selected to highlight several challenges of green-screen keying. The results show that our method performs favorably against the state-of-the-art natural matting methods.

Comparison with commercial keying software

Several methods have been proposed to solve the keying problem by capturing the same foreground against different background colors. Figure 3.10 shows that our algorithm gives comparable results to such a method [Grundhöfer et al., 2010] using only a single background.

The keying tools that are widely used in production do not rely on any special setups. In this section, we compare our method with some of the leading commercial keying tools, namely Keylight, Primatte and IBK. To that end, we

Figure 3.11.: *Commercial keying tools, even when operated by a specialized compositing artist, may not be able to extract the fine details near intricate object boundaries while our algorithm is robust against such challenging regions.*

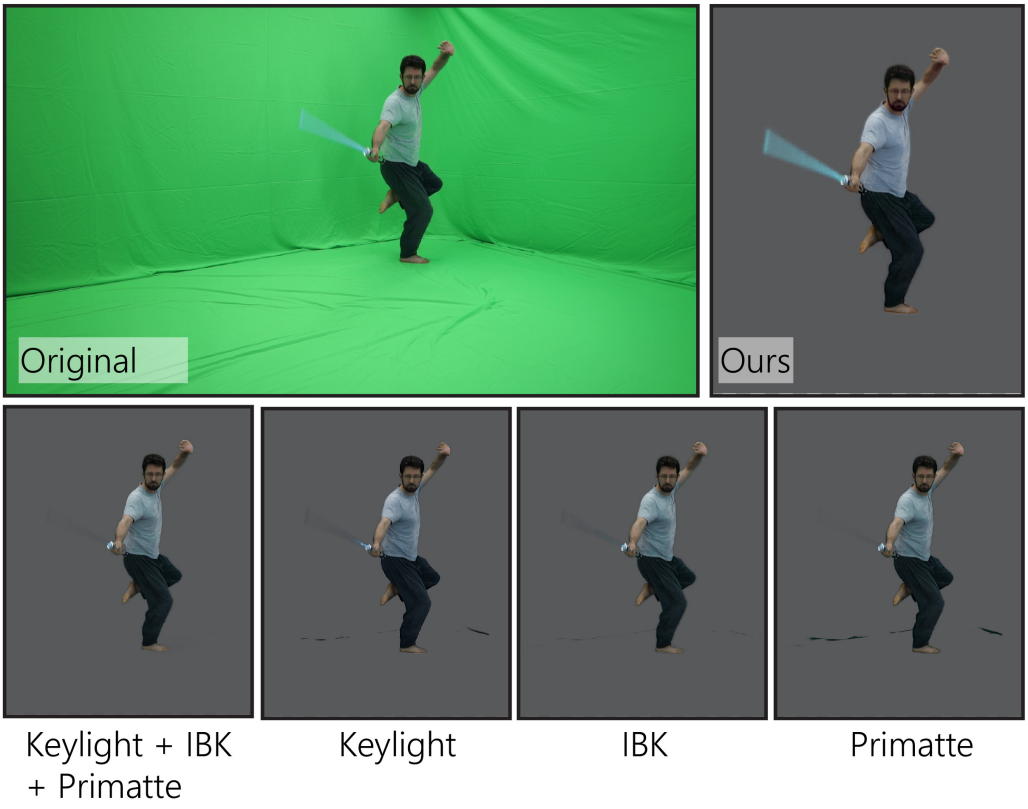

Figure 3.12.: *Commercial keying tools, even when operated by a specialized compositing artist, may fail to extract highly blurred objects while our algorithm is robust against such challenging regions.*

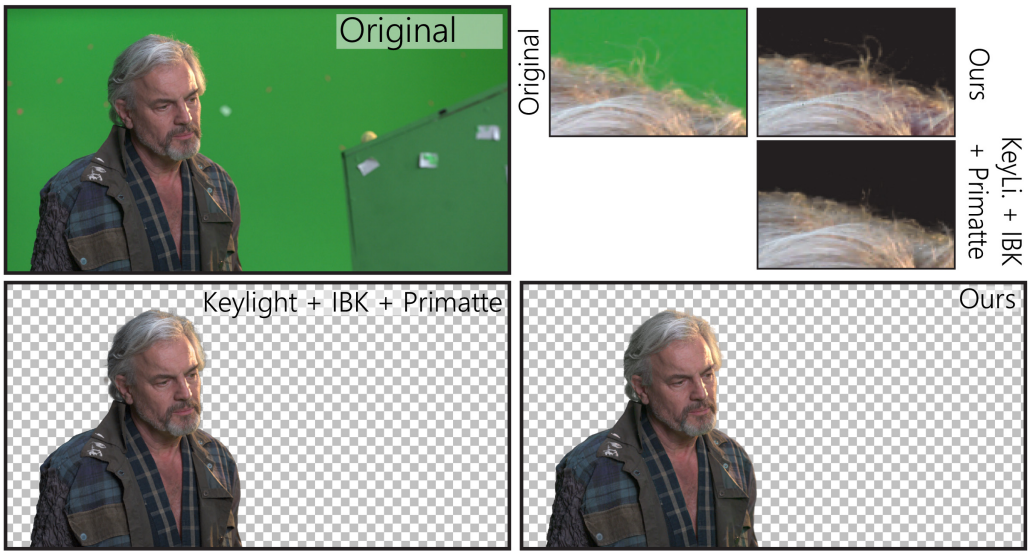

Figure 3.13.: *Commercial keying tools, even when operated by a specialized compositing artist, may distort the foreground color if it is similar to the background color while our algorithm is robust against such challenging regions.*

Figure 3.14.: *Commercial keying tools, even when operated by a specialized compositing artist, may create unnatural artifacts around blurred regions while our algorithm is robust against such challenging regions.*

used green-screen shots from the open source movie Tears of Steel[1] as well as some content that we shot with a Sony α7s camera.

In order to present a fair comparison, we asked a paid professional compositing artist to generate a separate result with each tool for each test scene. Based on the artist's feedback that in most real world scenarios all three tools would be used sequentially to take advantage of their individual strengths, we decided to ask the artist also to generate another set of result where he is allowed to use all of the three tools. We did not impose any constraints on the artist other than asking him to avoid manually painting pixels. A step-by-step description of the keying by the artist for one of our test sequences in presented in Section 3.3.

[1](CC) Blender Foundation — mango.blender.org

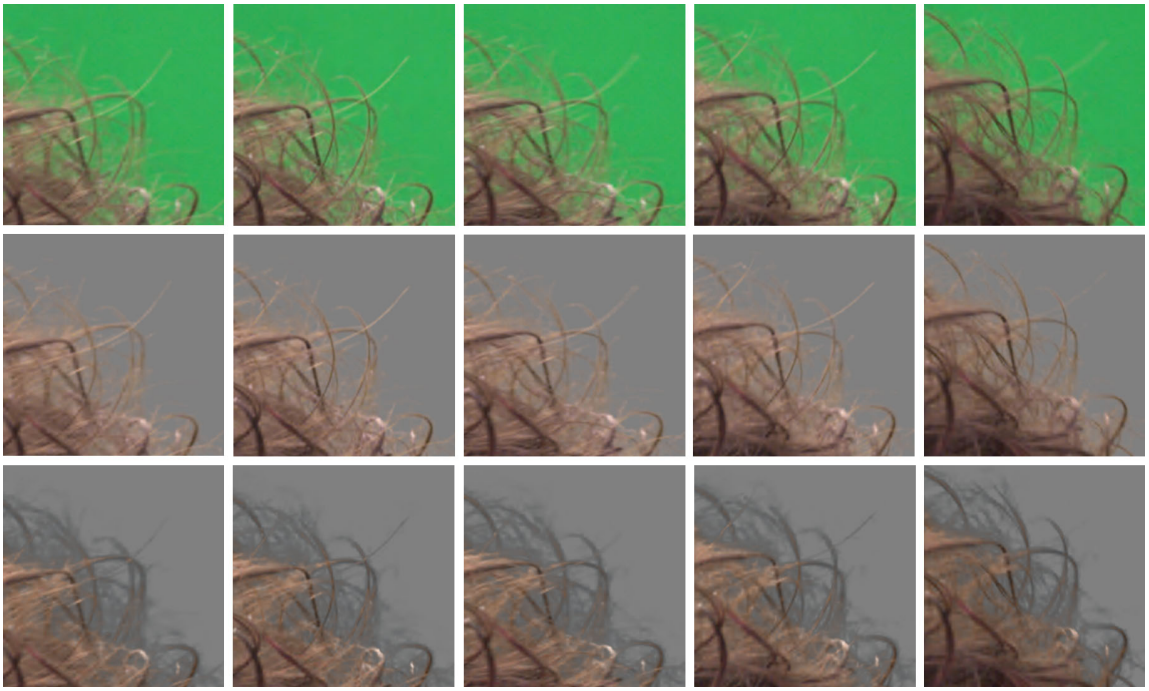

Figure 3.15.: *Close-up around the same single filament in several frames of the video. Notice that the filament that sticks out is captured using our method even when it is motion-blurred (middle), but the artist was only able to capture it in some of the frames (bottom).*

For the four sequences in our test set, the artist reported a total of 9 hours to get the results using multiple tools and reported an estimated 12 hours for fixing any remaining issues. Our results, on the other hand, were generated by ourselves using our tool in less than an hour. Almost the entire time was spent on refining the local color models using point and click interface of our method.

The results presented in Figures 3.11–3.14 show that our results compare favorably to the artist's results, even when the artist uses all the tools at his disposal and spends approximately an order of magnitude more time on manual editing. Additionally, the complex workflow and heavy local editing employed by the artist may result in temporal coherence artifacts. In contrast, our results for the same sequences do not suffer from such artifacts, as illustrated in Figure 3.15.

Because of the high amount of spill on the actor in scenes shown in Figures 3.12 and 3.14, actors appear transparent in the extracted foreground layer. Discriminating between transparency occurring from color spill or motion blur in a principled way is not a trivial problem. In order to account

Figure 3.16.: *The main real-world application of our method is digital compositing. The figure shows a number of toy examples that we generated using the foreground layers obtained with our prototype implementation. Background images courtesy of Flickr users* milanboers *(a) and* jeremylevinedesign *(c).*

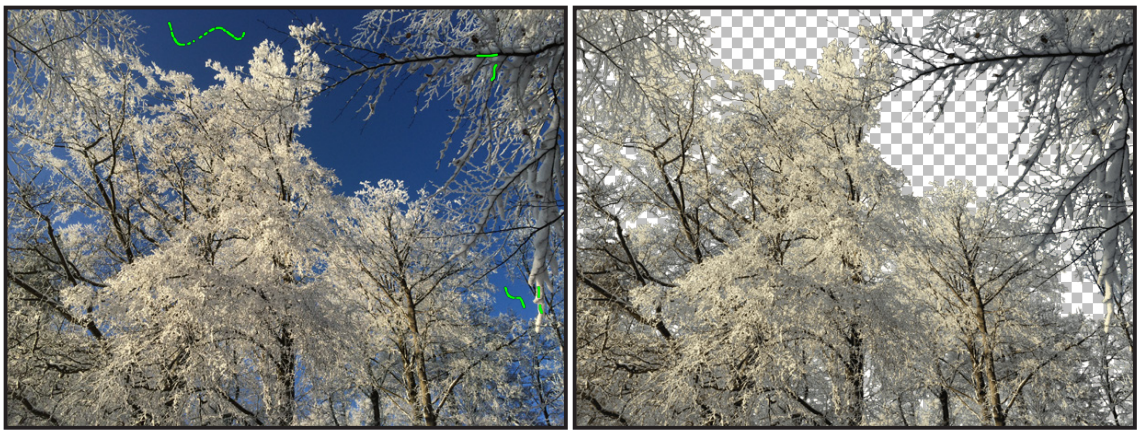

Figure 3.17.: *Despite the very complex scene structure, our algorithm successfully removes the sky in the background, demonstrating an advantage of our per-pixel approach to color unmixing that does not rely on spatial cues.*

Figure 3.18.: *The layers computed by our algorithm is used to replace the background (b) and change the color of the blurred object (c) while retaining the reflections. Note that the result images are color graded while compositing. Background image courtesy of Flickr user* davejdoe.

for this, we apply a simple post processing composed of boosting α values of the foreground layers with high spill to 1 except for the edges of the layers. For instance, the layer corresponding to the white robe in Figure 3.14 appears transparent after color unmixing. The robe layer is post-processed such that it has unity alpha values in regions that are not on the edges of the robe. The edges are left untouched to account for the smooth transition and the motion blur around the edges. While this post-processing is not completely fool-proof, i.e. its performance will degrade if there is strong color spill on layers with high transparency, we found it to be helpful for compositing and left the classification of non-unity alpha values to color-spill or transparency as a future work. Figure 3.16 shows examples compositing results generated using the foreground layers extracted by our method.

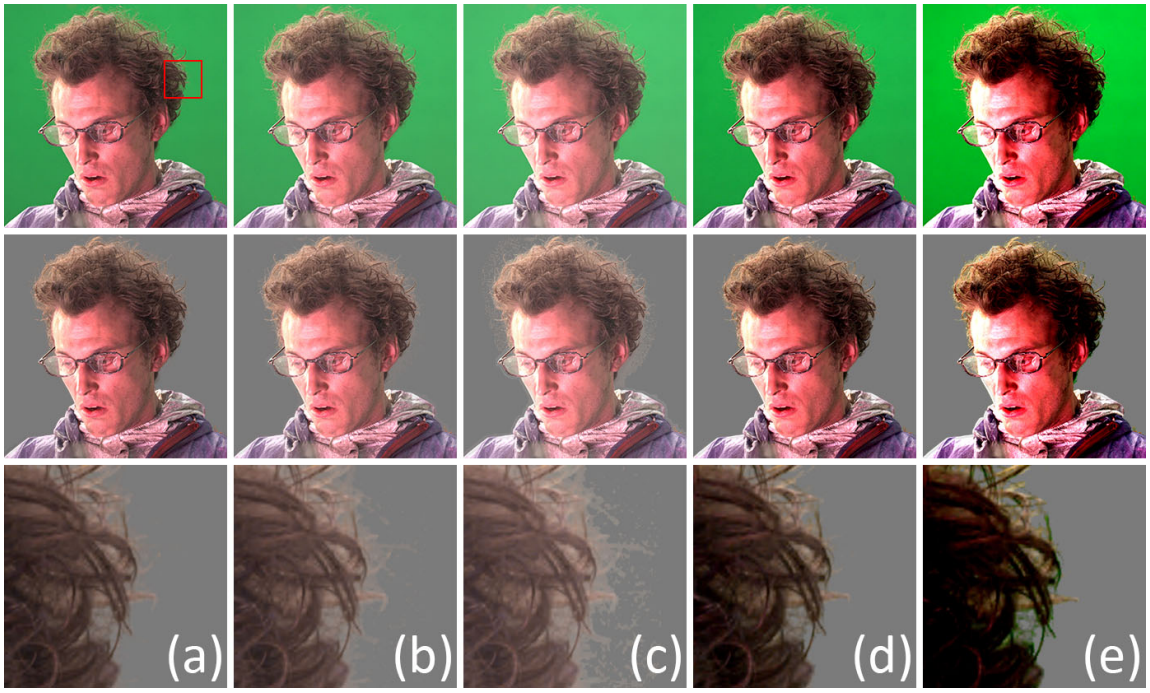

Figure 3.19.: *We changed the illumination or contrast of the input frame and extracted the foreground using the same color model constructed from the original image (a). With rather slight changes (b, d), our method is able to successfully extract the foreground. With a significant change in brightness (c), we observe a drop in the performance of our method, characterized by the halo around the actor. On the other hand, with very significant contrast change (e), some intricate details are missed and the background color remains in some small regions in the foreground.*

We also tested our method using scenes with non-green-screen backgrounds. Figure 3.17 shows an example in which our per-pixel color unmixing approach proves to be robust against complex foreground structures. Another example, one that includes reflections from a semi-transparent medium, is shown in Figure 3.18. While the backgrounds in these examples are admittedly simple, these results suggest that our method could be useful for an extended set of applications beyond green-screen keying. However, it is worth noting that our method is limited to simple backgrounds and is not suitable for general purpose natural matting.

3.5. Limitations

While in our experiments we have not noticed any significant temporal consistency issues, our test scenes had admittedly near constant illumination. In practice, keying may need to be performed in outdoor scenes (such as

Figure 3.20.: *When our assumption of a small number of scene colors is satisfied, we are able to get a successful foreground layer (left), but the quality drops significantly otherwise. Images courtesy of Rhemann et al.[2009].*

driving), where the illumination can change drastically from one frame to another. Due to the absence of any mechanism to enforce temporal coherence, we expect the performance of our method to decrease in such settings, as demonstrated in Figure 3.19.

The global color model as a small set of distributions may not be able to effectively represent non-green-screen backgrounds We tested our method on several images from the alpha matting benchmark [Rhemann et al., 2009]. Figure 3.20 shows typical natural matting results where our method works well when our main assumptions are satisfied, but fails when they are violated.

Our scribble interface for extracting the color model requires the unmixed colors to be present in at least one of the frames. For highly transparent media such as thin smoke, the pure color can not be determined via the proposed interaction and hence it is not possible for our keying system to extract the layer with only smoke. Devising an algorithm that can infer the colors that only appear mixed with others in a scene is an interesting direction for further research.

The proposed color unmixing algorithm may slightly overestimate the alpha values of some layers in some cases. Since the energy minimization favors underlying colors that are closer to the mean vector of the distributions, the foreground layer might get a small portion of the color mixture since matte sparsity is not enforced in the color unmixing energy minimization by design. This mainly occurs when the underlying color of one of the layers are not well-represented by the corresponding distribution. These artificially occurring alpha values being very small, we observed that this behavior does not result in any disturbing artifacts in green-screen keying.

CHAPTER 4

Unmixing-Based Soft Color Segmentation for Image Manipulation

Our main objective here is to decompose an image into multiple partially-transparent segments of homogeneous colors, i.e. *soft color segments*, which can then be used for various image manipulation tasks. At its core, the soft color segments represent the image in terms of mixtures of the segment colors at each pixel. Hence, the per-pixel per-layer opacities, or alpha values, can be seen as the mixing ratios that form the pixel color. Borrowing from image manipulation terminology, we will refer to such soft segments simply as *layers*. An example set of sofr color segments and their use in image manipulation can be seen in Figure 4.1.

A crucial property of soft color segmentation is that overlaying all layers obtained from an image yields the original image itself so that 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 i^{th} layer. and we want to satisfy the **color constraint**:

$$\sum_i \alpha_i^p \boldsymbol{u}_i^p = \boldsymbol{c}^p \quad \forall p, \tag{4.1}$$

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 require the opacity values over all layers to add up to unity, which we express as the

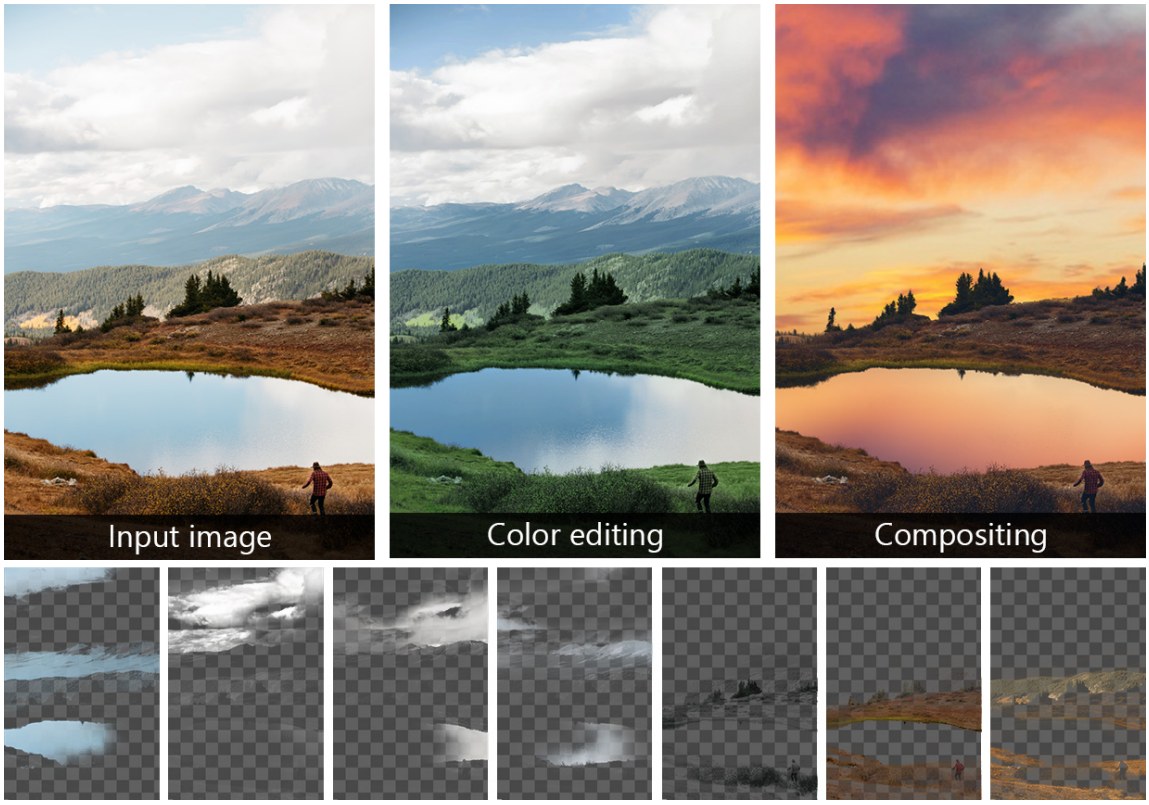

Figure 4.1.: *Our method automatically decomposes an input image (a) into a set of soft segments (b). In practice, these soft segments can be treated as layers that are commonly utilized in image manipulation software. Using this relation, we achieve compelling results in color editing (c), compositing (d), and many other image manipulation applications conveniently under a unified framework.*

alpha constraint:

$$\sum_i \alpha_i^p = 1 \quad \forall p. \tag{4.2}$$

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

$$\alpha_i^p, \boldsymbol{u}_i^p \in [0,1] \quad \forall i, p. \tag{4.3}$$

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

An important property of soft color segments that allows them to be used for image manipulation is their spatial coherency. In their estimation, as a result, the similarity of the layer colors and opacities in local neighborhoods has been heavily utilized in the literature. The resulting global optimization problems are typically very large, and suffer from high computation times and quality issues due to local minima. In this chapter, we break this global optimization

problem into three sub-problems each of which guarantee that one of the quality requirements of soft color segments is satisfied. The optimization procedures in these sub-problems can be implemented efficiently and with parallelization, making our method applicable to everyday image editing tasks.

It should be noted that different representations for overlaying multiple layers exist. We use (4.1), to which we refer as *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. [2013a] among others. In most commercial image editing software, however, the representation proposed by Porter and Duff [1984], referred to in this section as *overlay* representation, is used. The difference and conversion between the two representations are presented in Section 4.2.1.

4.1. Three-Stage Soft Color Segmentation

Our algorithm for computing high-quality soft color segments can be described by three stages: color unmixing, matte regularization, and color refinement.

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*. While we obtained the color model interactively for the problem of green-screen keying in Chapter 4, for the purpose of soft color segmentation, we will put forward a novel technique for its automatical extraction in Section 4.3.

We extend the color unmixing energy defined in (3.3) to better fit the soft color segmentation problem, and propose the *sparse color unmixing* energy function in order to find a preliminary approximation to the layer colors and opacities:

$$\mathcal{F}_{\mathcal{S}} = \sum_i \alpha_i \mathcal{D}_i(\boldsymbol{u}_i) + \sigma \left(\frac{\sum_i \alpha_i}{\sum_i \alpha_i^2} - 1 \right), \tag{4.4}$$

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 (4.4) is minimized for all α_i and $\boldsymbol{u}_i$ simultaneously while satisfying the constraints defined in (4.1)-(4.3) using the optimization scheme detailed in Section 3.1.1. For each pixel, for the layer with best fitting distribution, we

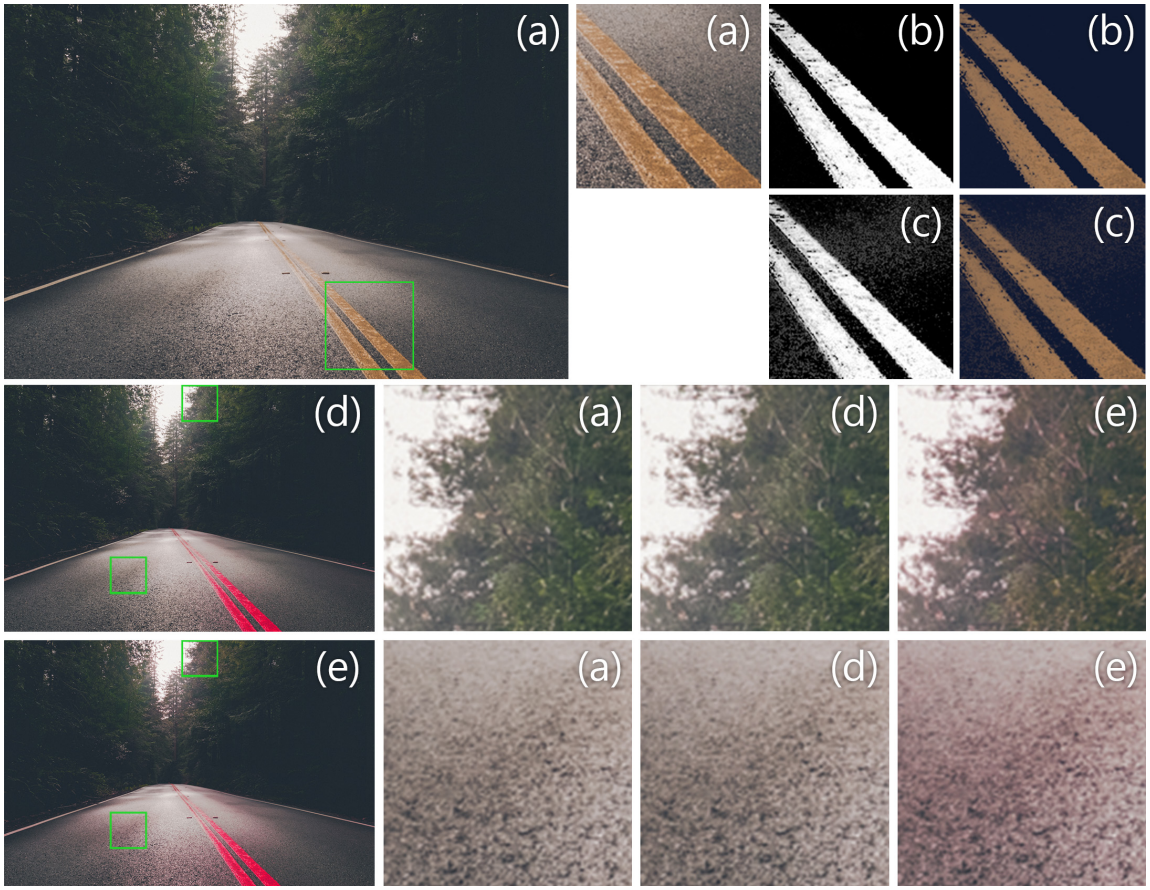

Figure 4.2.: *The original image (a) is shown with layers and their alpha channels corresponding to the yellow of the road lines estimated by the proposed sparse color unmixing (b) and by the color unmixing (c). Notice the spurious alpha values on the road in (c). When our sparse color unmixing results 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).*

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 (4.4), the color unmixing energy, 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*, i.e. favors 0 or 1 alpha values.

In the original color unmixing formulation, we do not include a sparsity term and this inherently results in favoring small alpha values as discussed in Chapter 3, 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. We used this property to our favor for green-screen keying, where enforcing matte sparsity would result in worse performance around regions with color spill. The use of local color models in our interactive keying scheme helped us circumvene the issues raised by the lack of a sparsity term. However, for a fully automatic soft color segmentation, we require a compact layer representation to avoid visual artifacts when the layers are edited independently. Figure 4.2 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. 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 4.2.

The sparsity term in (4.4) 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 one. Another term for favoring matte sparsity has been proposed by Levin et al. [2008b]:

$$\sum_i |\alpha_i|^{0.9} + |1 - \alpha_i|^{0.9}. \tag{4.5}$$

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 4.3(b). Hence, spatial regularization of the opacity channels is necessary for ensuring smooth layers as in Figure 4.3(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

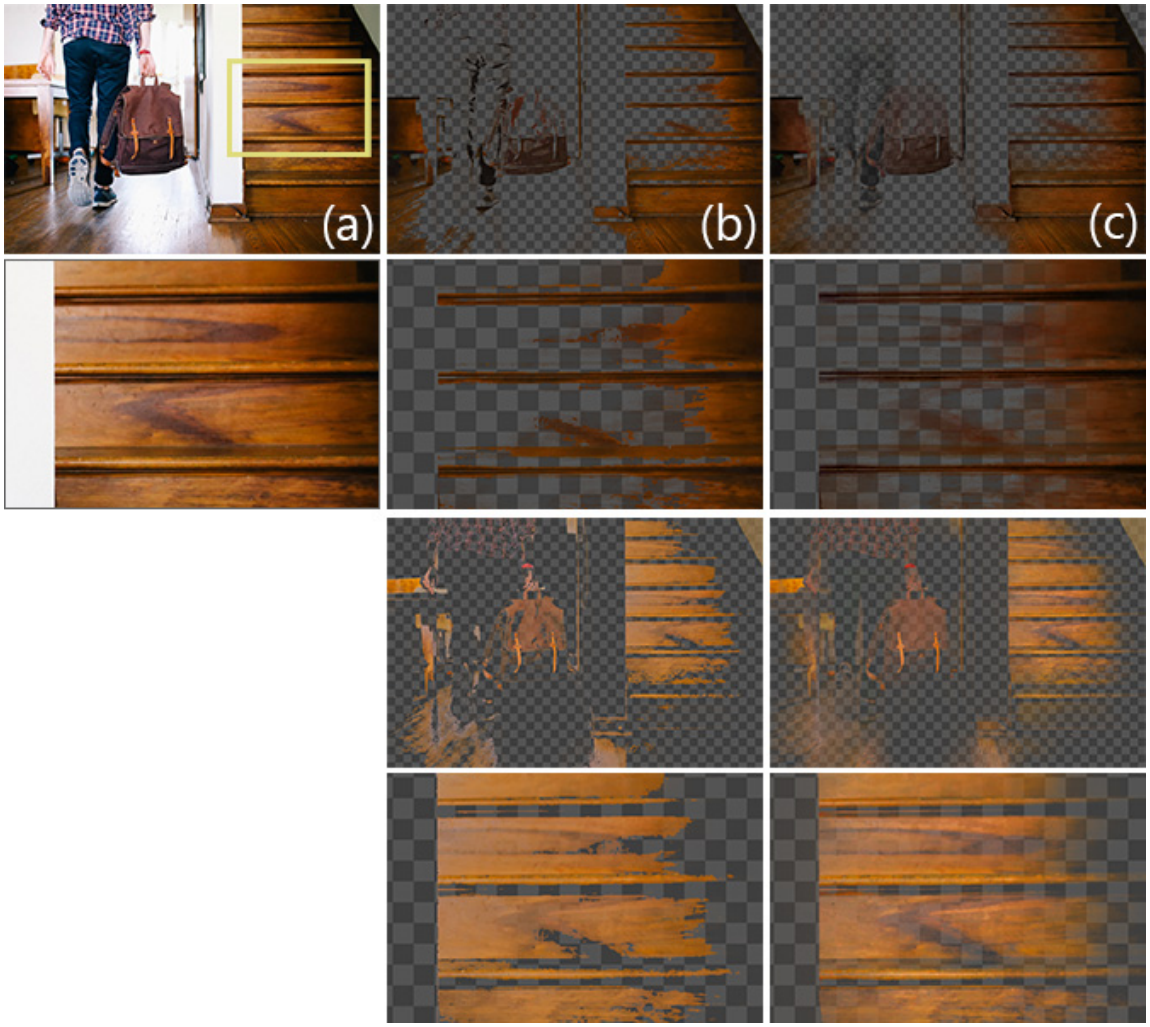

Figure 4.3.: *Two layers corresponding to the dark and light wood colors in the original image (a) are shown before (b) and after (c) matte regularization and color refinement.*

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 1 MP 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 (4.1). Violating the color constraint in general severely limits the ability to use soft segments for image manipulation. For illustration, Figure 4.6 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 (4.2) 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:

$$\sum_i (\alpha_i - \hat{\alpha}_i)^2 = 0, \tag{4.6}$$

where $\hat{\alpha}_i$ represents the regularized alpha value of the i^{th} layer. By running the energy minimization using this constraint, we recompute unmixed colors at all layers so that 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 (4.4) 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 (4.4) 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 (4.6). 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.

4.2. 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 4.4, 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 such as [Tai et al., 2007] and 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 [Singaraju and Vidal, 2011; Chen et al., 2013a], 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 4.3.

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

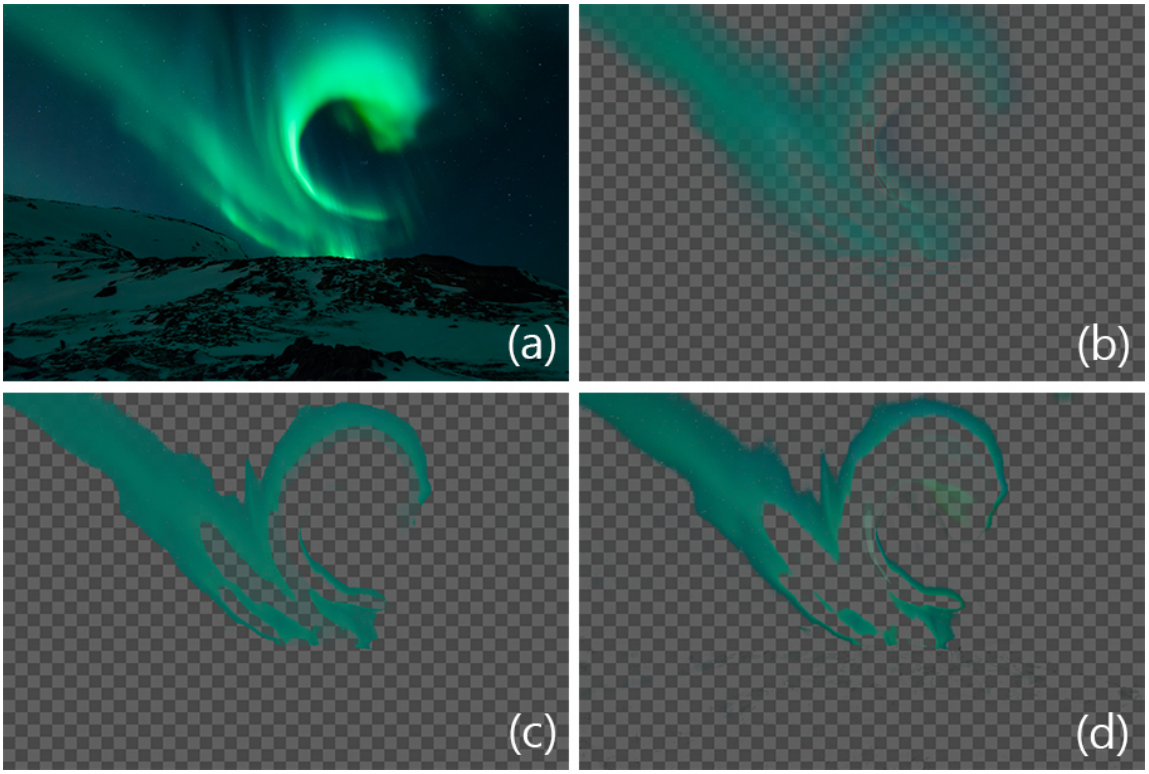

Figure 4.4.: *The layers corresponding to the outer region of the northern lights in the input image (a) computed by the proposed algorithm (b), color unmixing (c) and alternating optimization (d). Notice that the layers in (c, d) have abrupt edges in opacity while our result has a smooth transition 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 4.2.*

expressed as minimizing $\mathcal{E}_\alpha = \sum_p \mathcal{E}_\alpha^p$, with $\mathcal{E}_\alpha^p$ defined as:

$$\mathcal{E}_\alpha^p = \frac{1}{2\sigma_c^2} \| \boldsymbol{c}^p - \sum_i \alpha_i^p \boldsymbol{u}_i^p \|^2 + \frac{1}{\sigma_c^2} \sum_i \alpha_i^p \mathcal{D}_i(\boldsymbol{u}_i) + \tag{4.7a}$$

$$\sum_{q \in \mathcal{N}_p} \log \left(1 + \frac{1}{2\sigma_b} \left(\sqrt{\sum_i \left(\alpha_i^p - \alpha_i^q \right)^2} \right) \right), \tag{4.7b}$$

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 (4.7a) together with the color constraint, whereas (4.7b) 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 4.4.

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. Different than all the approaches discussed in this section as well as ours, they use the overlay layers representation as defined Section 4.2.1. 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 4.3.2, and fix $\boldsymbol{u}_i$'s to these predetermined values. They define their energy function as:

$$\begin{aligned} \mathcal{E}_{\text{RGBSG}} &= \omega_p \mathcal{E}_p + \omega_o \mathcal{E}_o + \omega_s \mathcal{E}_s \\ \mathcal{E}_p &= \frac{1}{K} \left\| \boldsymbol{u}_n - \boldsymbol{c} + \sum_i \left((\boldsymbol{u}_{i-1} - \boldsymbol{u}_i) \prod_{j=i}^{N} (1 - \tilde{\alpha}_j) \right) \right\|^2 \\ \mathcal{E}_o &= \frac{1}{N} \sum_i -(1 - \tilde{\alpha}_i)^2 \qquad \mathcal{E}_s = \frac{1}{N} \sum_i (\nabla \tilde{\alpha}_i)^2, \end{aligned} \tag{4.8}$$

where K is 3 or 4 depending on whether they use RGB or RGBA optimization as defined in their paper, $\nabla \tilde{\alpha}_i$ 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 4.3.2. 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 behaviour is presented in Figure 4.9.

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 2-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): While we will go into the details further in Chapter 6, it is worth mentioning here 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 (4.5), 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.

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:

$$\sum_p \| \boldsymbol{c}^p - \sum_i \alpha_i^p \boldsymbol{u}_i^p \|^2 + \begin{bmatrix} \nabla_x \alpha_0 \\ \nabla_y \alpha_0 \end{bmatrix}^T \sum_i \begin{bmatrix} \nabla_x \boldsymbol{u}_i^T \nabla_x \boldsymbol{u}_i \\ \nabla_y \boldsymbol{u}_i^T \nabla_y \boldsymbol{u}_i \end{bmatrix} \tag{4.9}$$

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

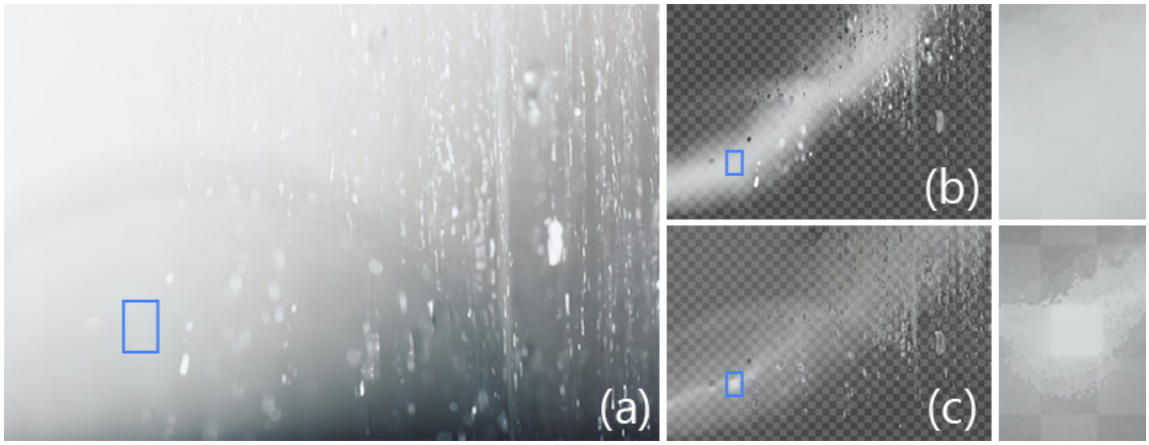

Figure 4.5.: *A layer extracted by the proposed method (b) and KNN Matting (c). KNN's hard constraints on alpha values may cause artifacts as shown in the inset.*

necessarily satisfy the color constraint. While this is not a big issue when the colors of a single layer, i.e. a predefined *foreground* layer, is estimated, this becomes problematic in terms of soft color segmentation.

KNN Matting (KNN): KNN matting [Chen et al., 2013a], 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 one. 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 away from the diagonal. This significantly increases the run-time 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 4.5.

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 will discuss their layer color estimation further in Section 5.3. We observed that independently solving for each layer fails to satisfy the color constraint in the final result, as shown in Figure 4.6. 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 4.1, not requiring

Figure 4.6.: *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.*

a pre-determined layer ordering, as well as providing smooth transitions between layers and its per-pixel formulation that 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 4.4.

4.2.1. *Alpha-add* and *overlay* Layer Representations

For handling layers, our blending formulation in (4.1) corresponds to the *alpha add* mode present in Adobe After Effects. The *normal* blending option in Photoshop is slightly different than ours, which is defined for two layers as follows:

$$\boldsymbol{u}_o = \frac{\tilde{\alpha}_a \boldsymbol{u}_a + \tilde{\alpha}_b \boldsymbol{u}_b \left(1 - \tilde{\alpha}_a\right)}{\tilde{\alpha}_a + \tilde{\alpha}_b (1 - \tilde{\alpha}_a)}, \tag{4.10}$$

$$\tilde{\alpha}_o = \tilde{\alpha}_a + \tilde{\alpha}_b \left(1 - \tilde{\alpha}_a\right), \tag{4.11}$$

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 and $\boldsymbol{u}_b$ and $\tilde{\alpha}_b$ 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, N^{th} layer being at the top, one can convert alpha-add layers to overlay layers:

$$\tilde{\alpha}_n = \begin{cases} \frac{\alpha_n}{\sum_{i=1}^{n} \alpha_i} & \text{if } \sum_{i=1}^{n} \alpha_i > 0 \\ 0 & \text{if } \sum_{i=1}^{n} \alpha_i = 0 \end{cases}, \quad n \in \{1 \dots N\}. \tag{4.12}$$

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 4.5, 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:

$$\alpha_n = \tilde{\alpha}_n \left(1 - \sum_{i=n+1}^{N} \alpha_i\right), \quad n \in \{1 \dots N\}. \tag{4.13}$$

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

4.3. Color Model Estimation

In Section 4.1, 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

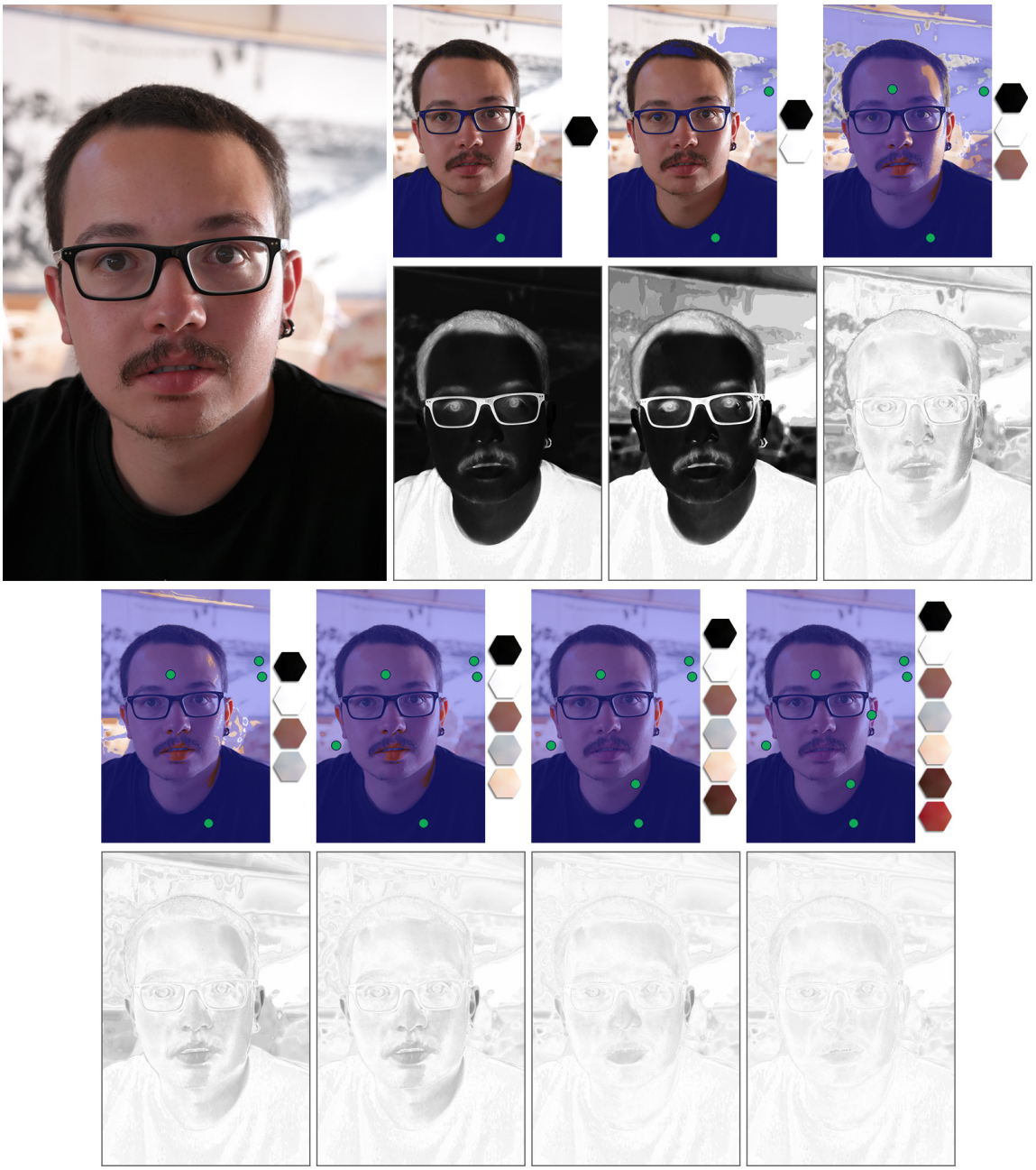

Figure 4.7.: *Color model estimation is done by placing seed pixels (marked green on the top row) one by one and estimating a normal distribution (visualized as the hexagons) from the local neighborhood of each of the seeds. To select the next seed pixel, we make use of the representation scores (bottom 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.*

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 which can already be well-represented as a mixture of other main colors.

Figure 4.7 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, the color unmixing energy will be high due to the $\mathcal{D}(\boldsymbol{u}_i)$ term. By using the color unmixing energy (3.3) 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, the algorithm terminates.

Mathematically, the vote of each pixel is computed as:

$$v^p = e^{-\|\nabla \boldsymbol{c}^p\|} \left(1 - e^{-r^p}\right), \tag{4.14}$$

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:

$$s_i = \underset{p \in \text{bin}}{\arg\max}\, \mathcal{S}^p e^{-\|\nabla \boldsymbol{c}^p\|} \tag{4.15}$$

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:

$$\text{Remove } p \text{ if } r^p < \tau^2. \tag{4.16}$$

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 4.4.1 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 4.1. 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 *projected color unmixing* that is discussed next.

4.3.1. 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:

$$\hat{r}^p = \min(\{\mathcal{D}_i(\boldsymbol{c}^p), \forall i\} \cup \{\hat{\mathcal{F}}_{i,j}(\boldsymbol{c}^p), \forall i, \forall j \neq i\}), \tag{4.17}$$

where $\hat{\mathcal{F}}_{i,j}(\boldsymbol{c}^p)$ represents an approximation of the minimized color unmixing energy using the i^{th} and j^{th} 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

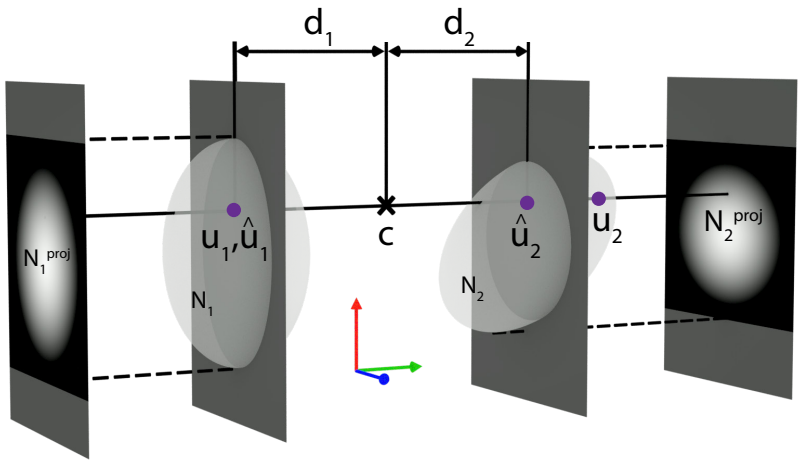

Figure 4.8.: *An illustration of projected color unmixing. See text for discussion.*

(4.17) 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*.

Projected color unmixing

The color line assumption [Ruzon and Tomasi, 2000], i.e. 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 2-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 for the $N = 2$ case, which we illustrate in Figure 4.8.

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:

$$\hat{\boldsymbol{u}}_{\{1,2\}} = \boldsymbol{c} - \frac{(\boldsymbol{c} - \boldsymbol{\mu}_{\{1,2\}}) \cdot \boldsymbol{n}}{\boldsymbol{n} \cdot \boldsymbol{n}} \boldsymbol{n}. \tag{4.18}$$

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

$$\hat{\alpha}_1 = \frac{\|\boldsymbol{c} - \boldsymbol{u}_2\|}{\|\boldsymbol{u}_1 - \boldsymbol{u}_2\|}, \quad \hat{\alpha}_2 = 1 - \hat{\alpha}_1. \tag{4.19}$$

If $\boldsymbol{c}$ does not lie between the two planes, 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:

$$\hat{\mathcal{F}} = \hat{\alpha}_1 \mathcal{D}_1^{\text{proj}}(\hat{\boldsymbol{u}}_1) + \hat{\alpha}_2 \mathcal{D}_2^{\text{proj}}(\hat{\boldsymbol{u}}_2), \tag{4.20}$$

where we refer to $\hat{\mathcal{F}}$ as projected color unmixing (PCU) energy.

If the two largest eigenvalues of the normal distribution are close to lying on the plane we defined above, the estimated layer color is actually very close to the one found by color unmixing, as illustrated by distribution 1 in Figure 4.8. Otherwise, the approximation error is larger, as illustrated by distribution 2.

The approximation to the alpha values shown in (4.19) is actually the same as the alpha estimation equation proposed by Chuang et al. [2001] and utilized by many sampling-based natural matting approaches:

$$\hat{\alpha}_1^{\mathcal{B}} = \frac{(\boldsymbol{c} - \boldsymbol{u}_2) \cdot (\boldsymbol{u}_1 - \boldsymbol{u}_2)}{\|\boldsymbol{u}_1 - \boldsymbol{u}_2\|^2}. \tag{4.21}$$

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

$$\mathcal{C} = \|\boldsymbol{c} - \alpha_1 \boldsymbol{u}_1 - \alpha_2 \boldsymbol{u}_2\|, \tag{4.22}$$

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 2-layer color unmixing energy by an error rate of 15% on average, while running approximately 3000 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 1 MP image is 9 seconds.

A step-by-step example of our color model computation procedure is presented in Figure 4.7. 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 ((4.17)). 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 ((4.14)), we select the seed pixel from an image region where the gradient magnitude is low ((4.15)). 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 ((4.16)) and repeat the procedure until no bins have enough votes.

4.3.2. 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., 2013a], ML [Singaraju and Vidal, 2011] and our green-screen keying approach in Chapter 3 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

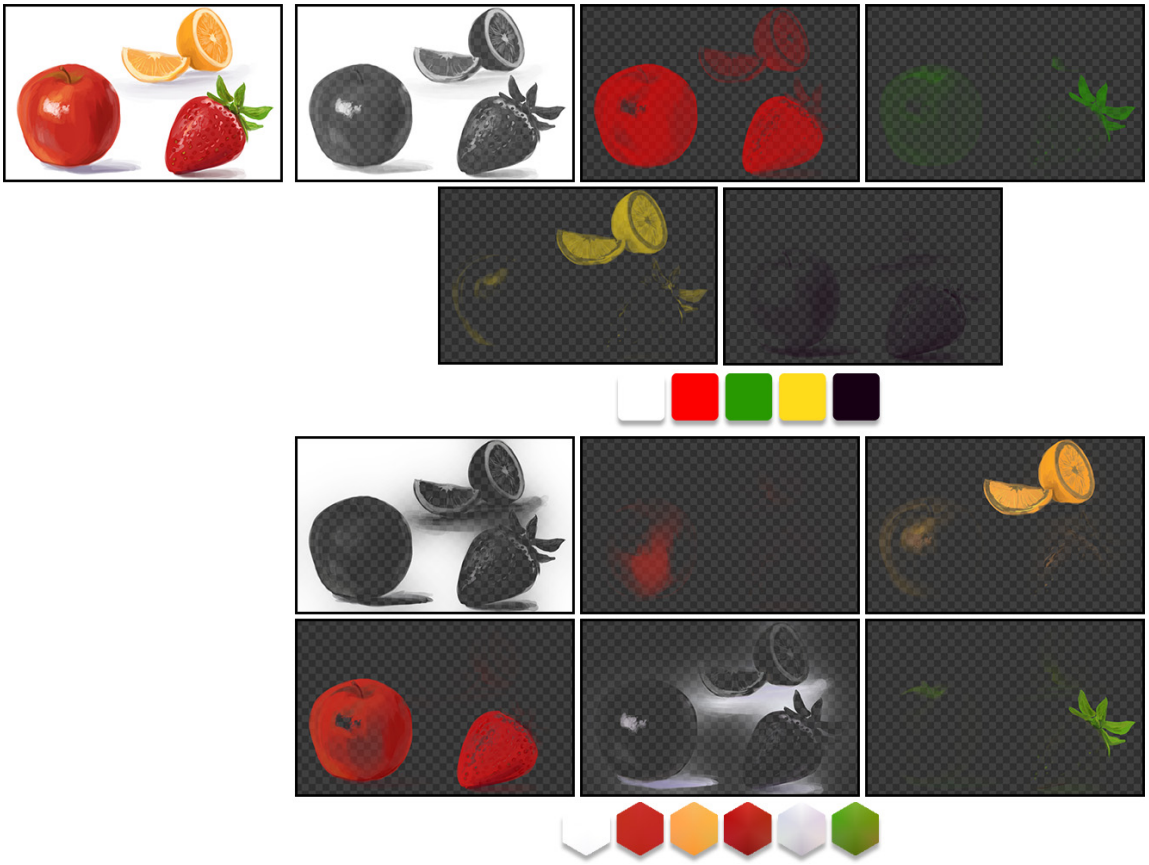

Figure 4.9.: *The color model and corresponding layers computed by the proposed method (bottom) and by Tan et al. [2016]. The layers on the right have been converted to alpha-add representation from overlay representation (Section 4.2.1) for a more meaningful comparison. The number of layers is different for the two algorithms because we are using the original result of Tan et al. which has 5 layers, and our automatic color model estimation method determined 6 dominant colors in the image. See text for discussion.*

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 can not be selected as a model color. The effect of this can be observed in Figure 4.9, 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 behaviour aloows 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 4.4.1.

4.4. Experimental Evaluation

Evaluating soft color segmentation methods is challenging, since how the optimal set of layers resulting from decomposing a particular input image

Table 4.1.: *Quantitative test results for ML [Singaraju and Vidal, 2011], SM [Levin et al., 2008b], KNN [Chen et al., 2013a], AO [Tai et al., 2007], CU (Chapter 3) 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

should exactly look like is not clear. In fact, 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 Figures 4.12–4.14 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 4.1. These metrics are:

- **Out-of-bounds alpha values:** Check if the box constraint ((4.3)) is satisfied for alpha values. The metric returns the percentage of alpha values outside the permissible range.
- **Reconstruction error:** Check if the color constraint ((4.1)) 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.

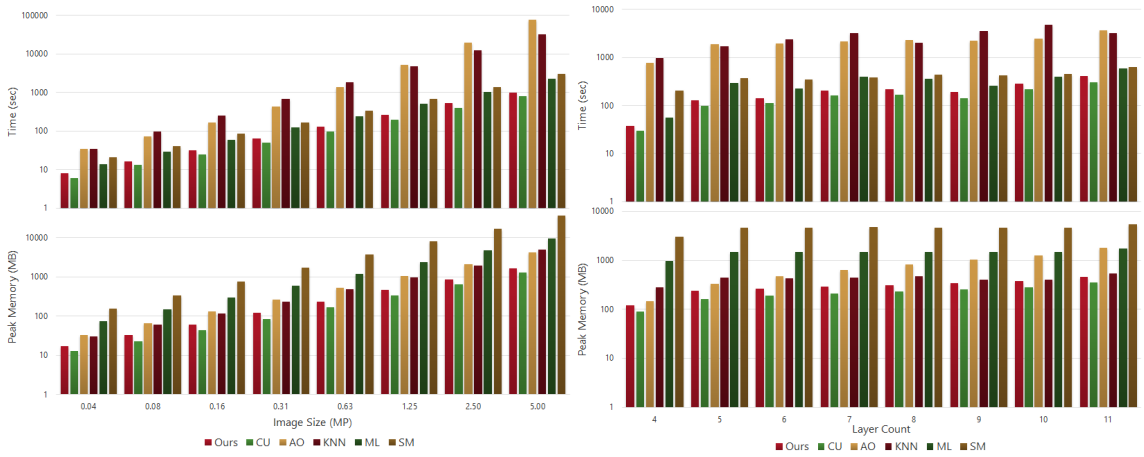

Figure 4.10.: *Computational resources needed for soft color segmentation with respect to the image size (left) and layer count (right). An image with 7 layers for and example images with corresponding layer counts were selected to see the trend in each algorithm. Note that the data axes are in logarithmic scale.*

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 in (4.3), 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 4.1.

We also analyzed the run-time and memory requirements of each method both as a function of the color model size and image resolution in Figure 4.10. For the former test, we selected 8 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 7 layers[1], which we resized from 5MP down to 40kP. It should be pointed out that our parallelized C++ application is compared to the MATLAB implementations of the competing methods. KNN, SM and ML solve large linear systems which can not be efficiently parallelized. However, these graphs still give us an idea of the scalability of each method. To demonstrate the scalability of our method, we show the soft color segments of a 100MP image in Figure 4.11

In our evaluation, we used publicly-available implementations of SM and KNN. We will compare against using only color unmixing (CU) as well. 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

[1]The average size of the color model was 6.88 for the 100 images on which we do the evaluation.

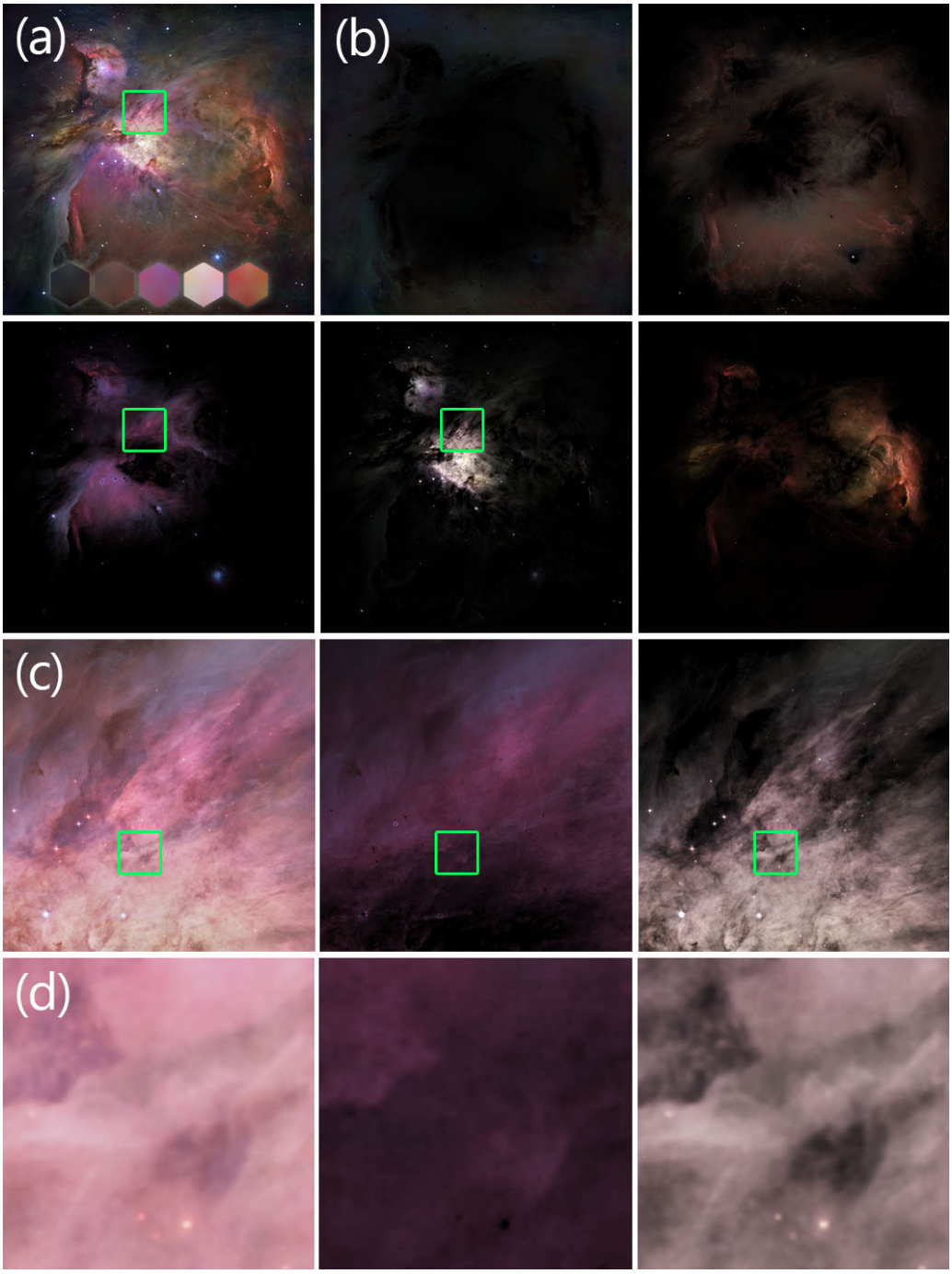

Figure 4.11.: *Our algorithm is able to process a 100MP image (a) to get corresponding soft layers (b) thanks to our per-pixel formulation and matte regularization by guided filtering. (c) and (d) show insets from the input layer and two of the layers at 10x and 100x magnification. This image was processed in 4 hours using up to 25 GB of memory. Note that KNN and AO can only process a 2.5MP image within the same time budget, and SM requires more than 25 GB of memory for a 5MP image.*

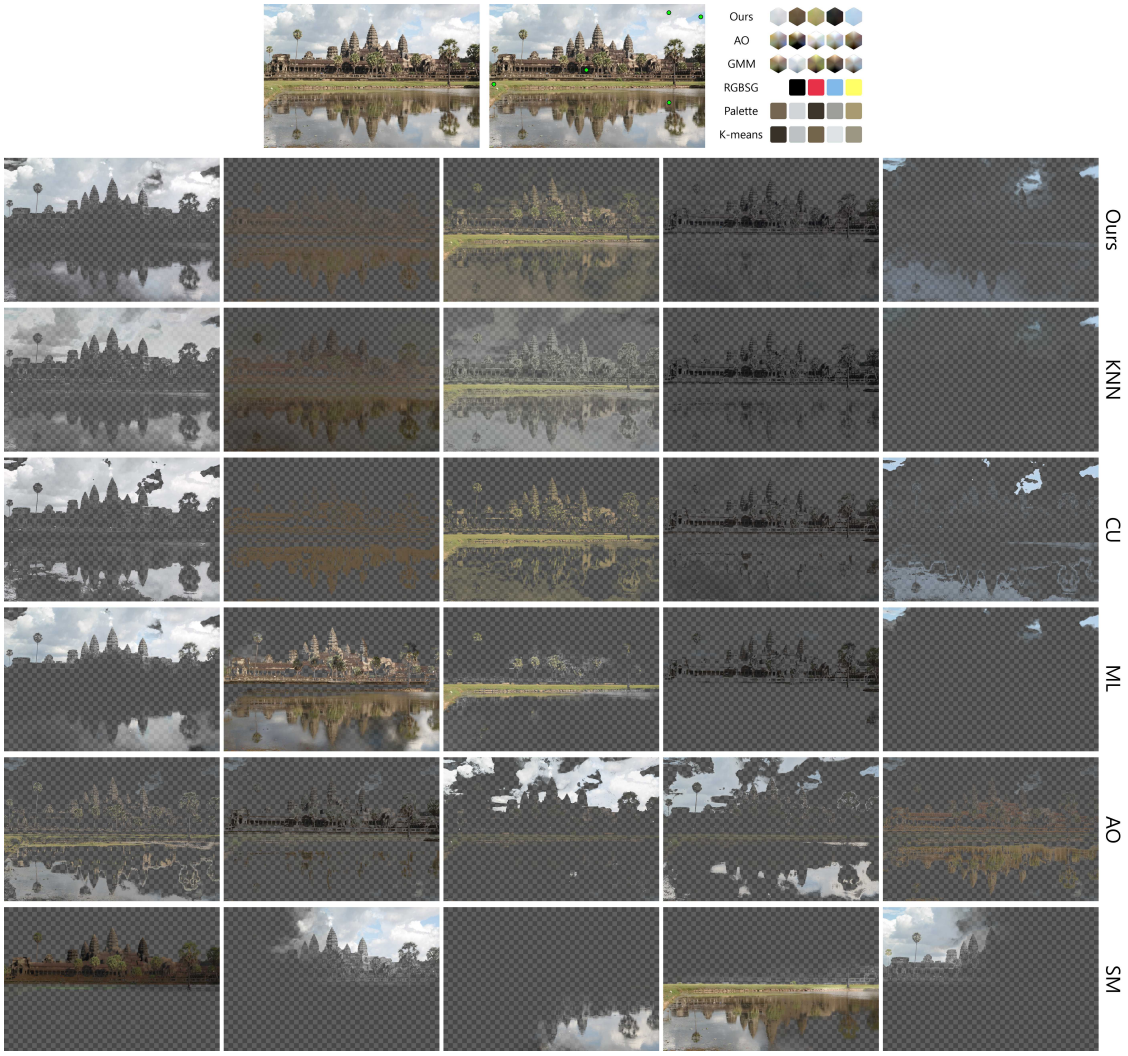

Figure 4.12.: *Comparison of the soft color segments produced by various algorithms including ours. See text for discussion.*

not available[2], we implemented the method ourselves in MATLAB, where we utilized the UGM toolbox for the loopy belief propagation [Schmidt, 2007]. Our own results for soft color segmentation and color unmixing (CU) were generated using our research prototype written in C++ with parallelization using OpenMP.

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

[2]Upon correspondence with the first author, we learned that the original code is no longer available.

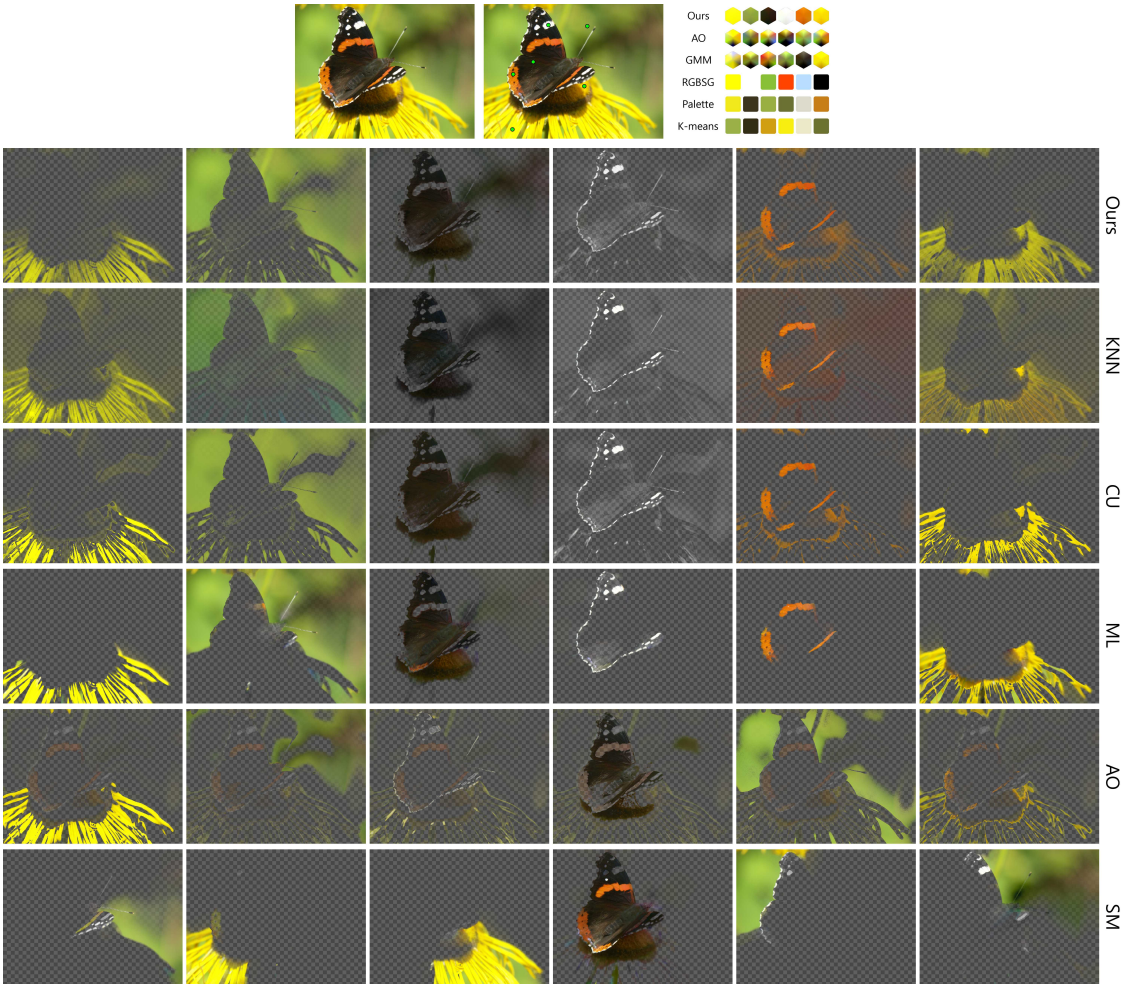

Figure 4.13.: *Comparison of the soft color segments produced by various algorithms including ours. See text for discussion.*

N, as well as the N seed points determined by our color model (Section 4.3). 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 Figures 4.12–4.14, where we show the layers produced by all competing methods. The figures clearly illustrates the shortcomings of propagation based methods SM and ML. For example in Figure 4.12, 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

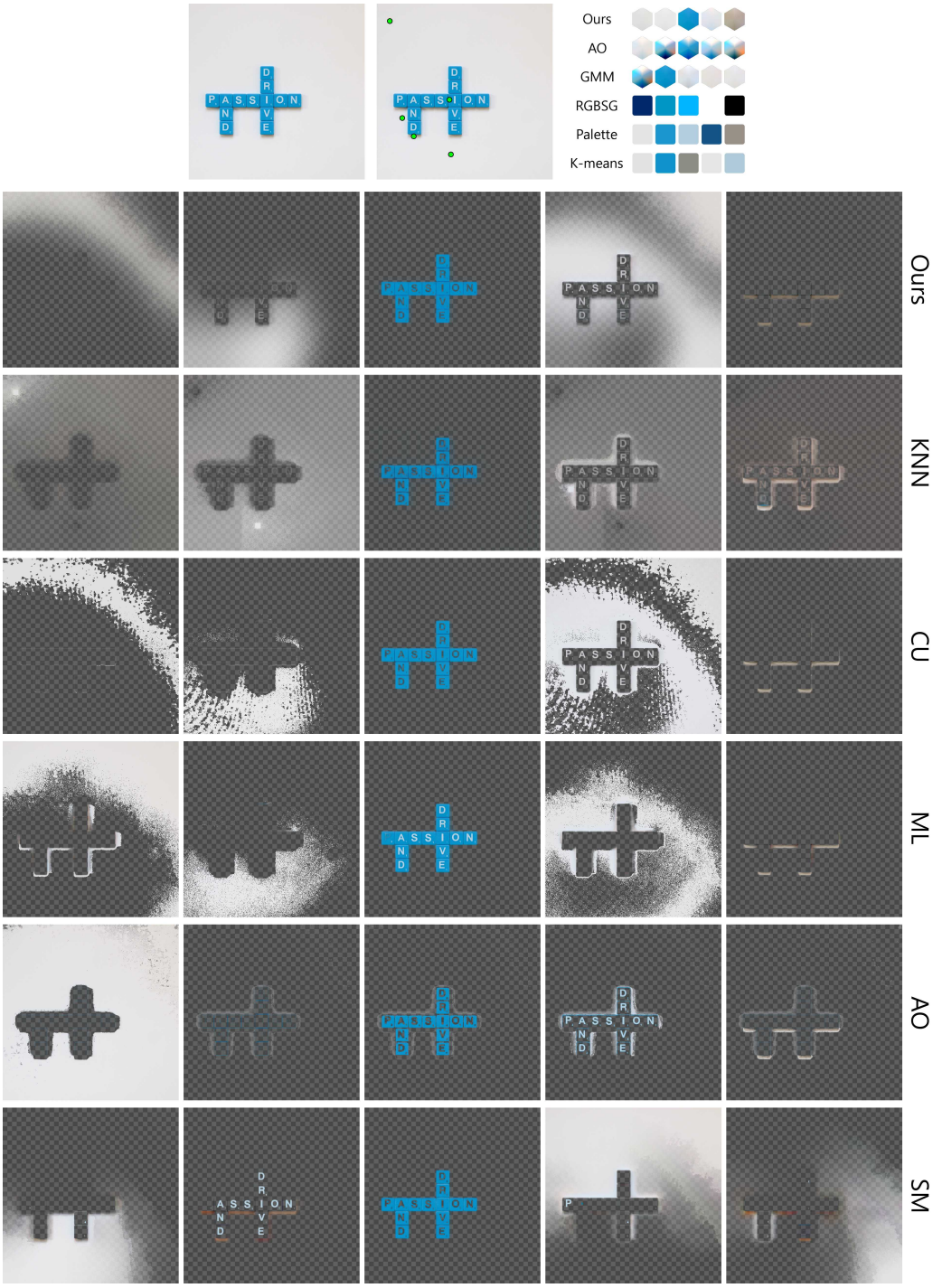

Figure 4.14.: *Comparison of the soft color segments produced by various algorithms including ours. See text for discussion.*

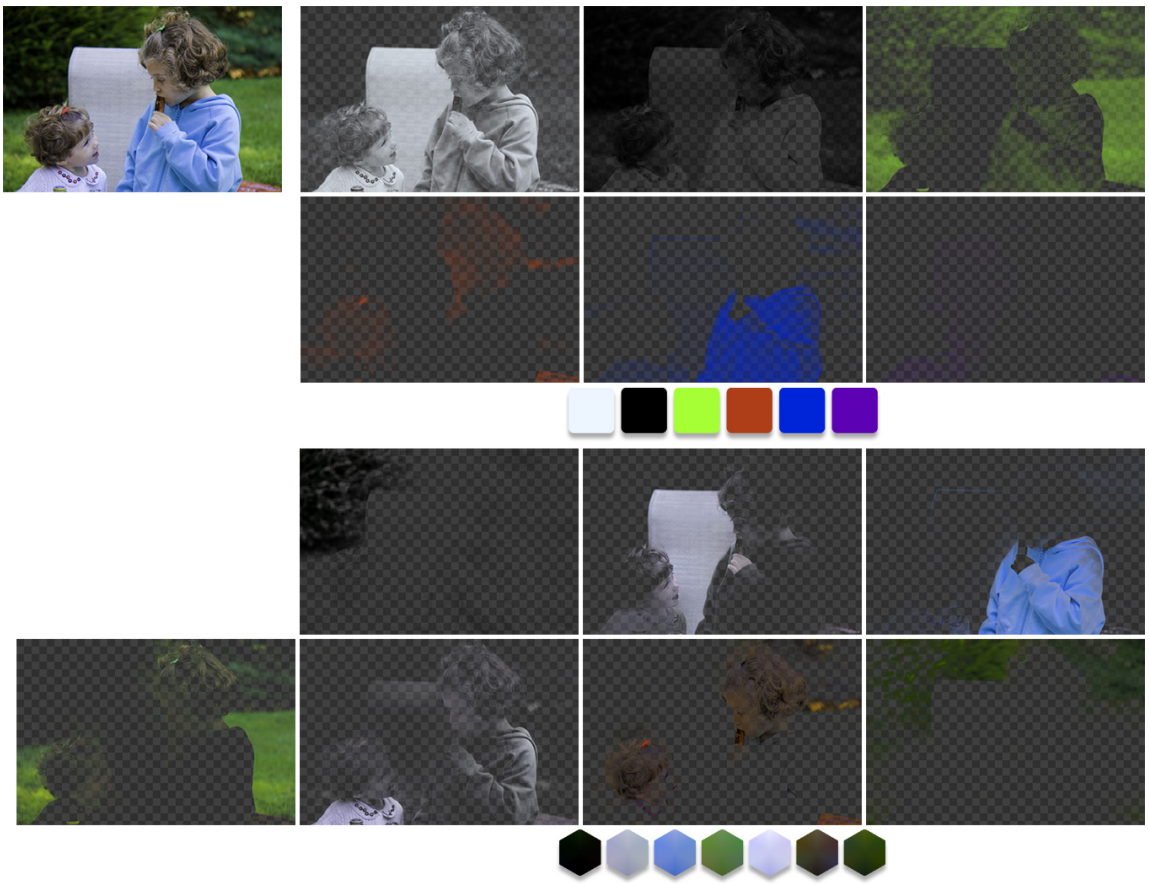

Figure 4.15.: *The soft color segments produced by the proposed algorithm (bottom) and by the method by Tan et al. [2016]. The layers by Tan et al. [2016] have been converted to alpha-add representation from overlay representation (Section 4.2.1) for a more meaningful comparison. See text for discussion.*

in Section 4.2, indicate that they, in fact, suffer from alpha values outside the permissible range. Their results on the remaining metrics reveal that they are generally worse than others for the task of soft color segmentation (Table 4.1).

On the other hand, both Figure 4.12 and Table 4.1 confirm our claims from Sections 4.2 and 4.3 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 4.2. Another significant drawback of KNN is its prohibitively long run-time, which prevents the method to be used on high-resolution images on current PC hardware.

The layers computed by CU in Figure 4.12 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 4.1.

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] in their paper in Figure 4.15.

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 different from the dominant colors in the image. This behavior, when coupled with the solid color-layers, results in reduced sparsity in their layers. Figure 4.15 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$\times$ 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 4 hours using up to 25 GB of memory. Note that KNN and AO can only process a 2.5MP image within the same time budget, and SM requires more than 25 GB of memory for a 5MP image. Also, note that our modifications to the color unmixing formulation cause only a modest performance hit, while significantly improving the quality of the layers.

4.4.1. Color Model Estimation

We compare our color model estimation method (Section 4.3) 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 Gaussian mixture models (GMM).

Figure 4.16 shows exemplary results for discussion. First of all, Figure 4.16 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 4.16 shows that any fixed number will be an 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 away 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 4.16, 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 4.16.

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 4.16 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 4.16, 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

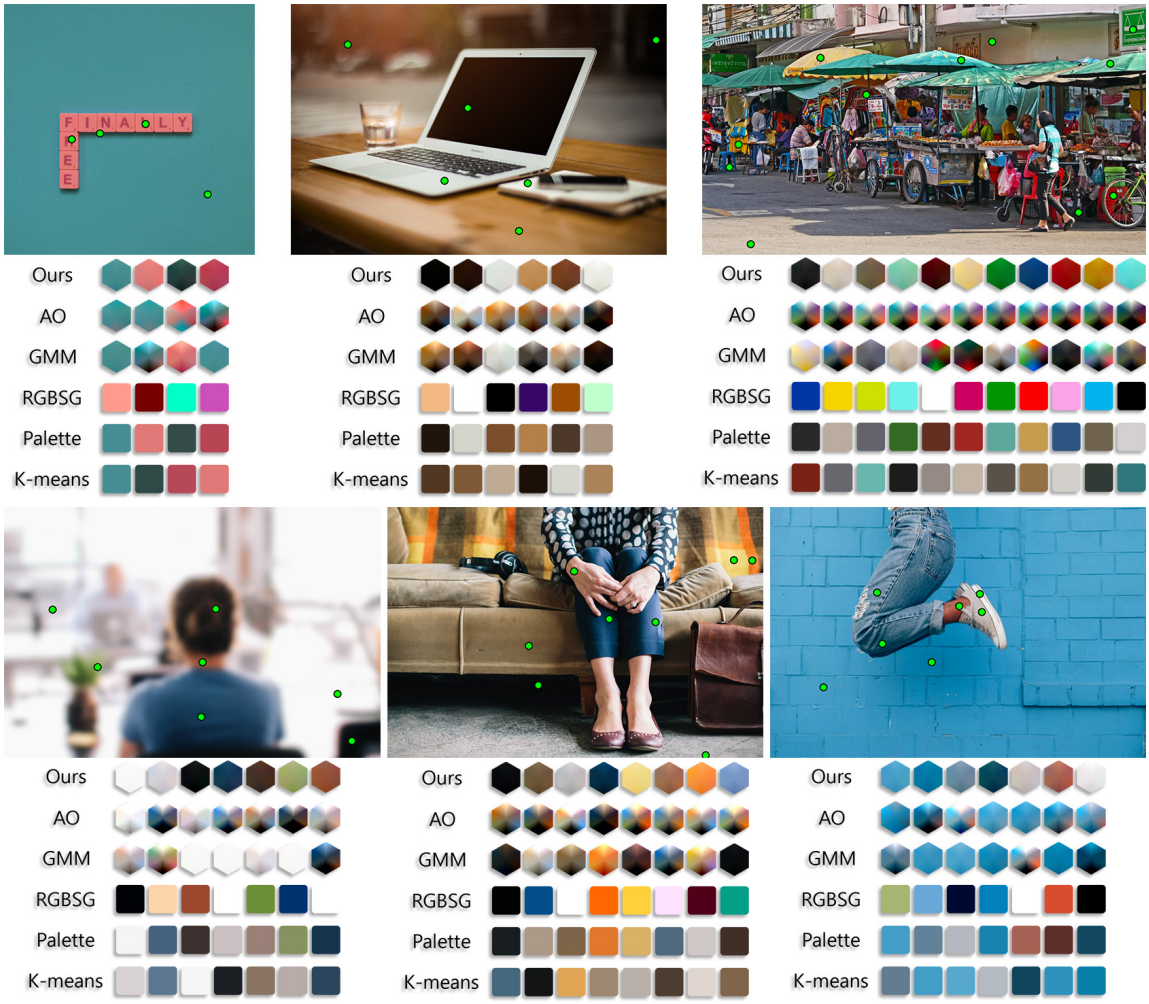

Figure 4.16.: *Comparison of the color models estimated by our method, alternating optimization [Tai et al., 2007], expectation maximization using Gaussian mixture models (GMM), RGB-hull based method used in RGBSG [Tan et al., 2016], palette estimation used in palette-based recoloring [Chang et al., 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.*

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 4.3.2 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.

4.5. 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, etc. 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 4.17 (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 allowing to 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 4.17 and 4.18, 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 4.17. For example, Figure 4.17 (3) is obtained by changing brightness, colors and saturation on certain layers, whereas in Figure 4.17 (8) we only

Figure 4.17.: *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.*

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 4.17 and 4.18 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.

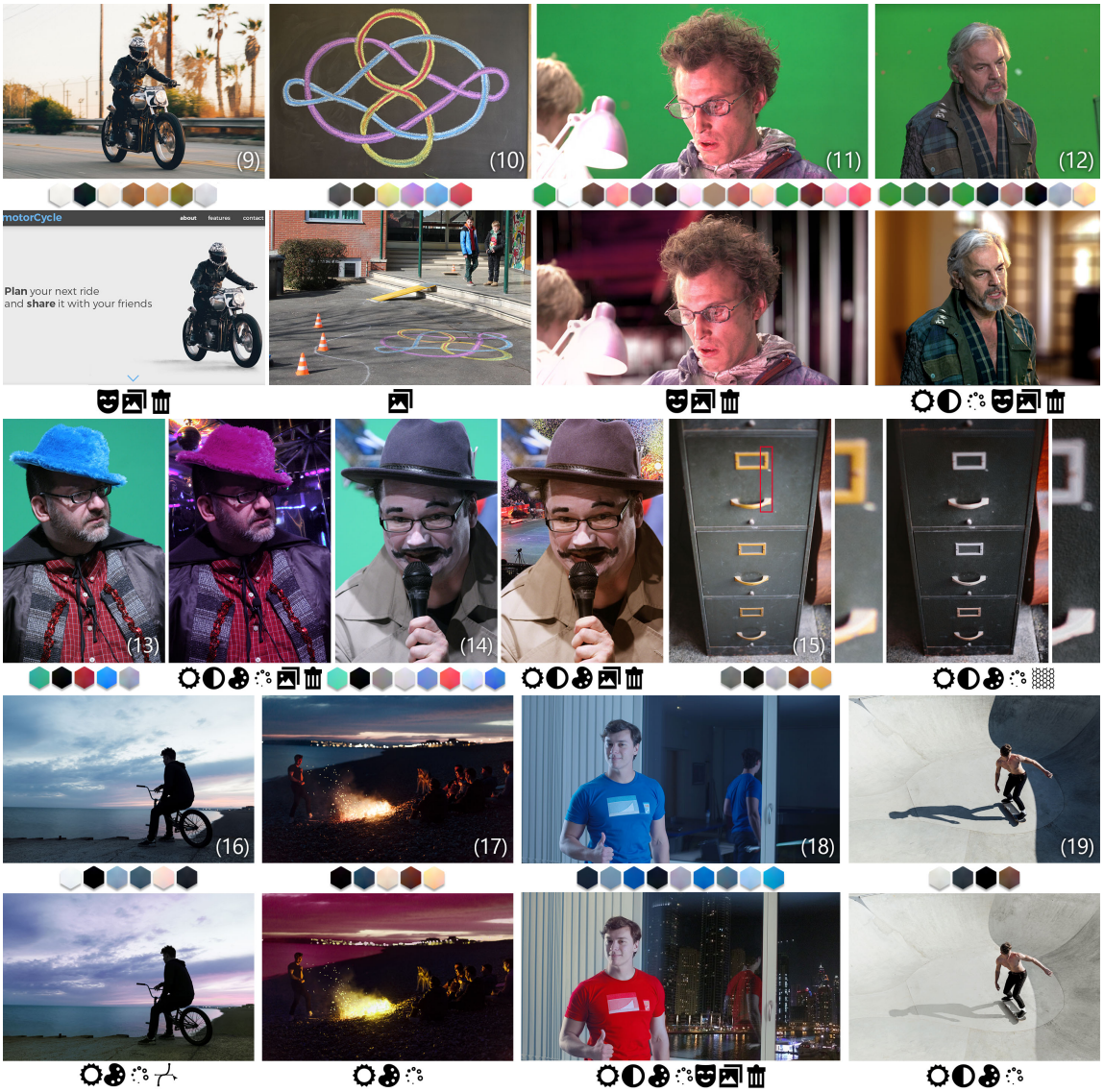

Figure 4.18.: *4.17 continued.*

4.5.1. 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) on selected layers of an input image. We showcase several examples including changing colors of clothing (1, 4, 6, 8, 13, 18), fauna (3), illumination (2, 7), sky (16, 17), fire (17) and metallic surfaces (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 4.17 and 4.18. Additionally, in (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 (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 (1, 4, 6, 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 (1, 4).

4.5.2. Compositing

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

Green-screen keying: Images (11, 12, 13, 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 (14)) can easily be removed using a standard garbage matte.

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 (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. (9) show 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 (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 manipulation, which gives users significant flexibility for realizing their artistic vision.

4.6. Comparisons at the Application Level

We demonstrated the use of our soft color segments in image editing in Section 4.5. Theoretically, any soft color segmentation method could produce layers that can be used for the same applications. We analyzed in Section 4.4 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.

Figures 4.19 and 4.20 show two images being edited by using our layers, layers by AO, KNN and RGBSG, as well as by a professional artist using Adobe Photoshop and using the palette-based recoloring application by Chang et al. [2015]. The example in Figure 4.19 shows a simpler case in terms of the structure of the image. It can be observed that the artist can achieve a similar result to ours using masks generated by the artist, although drawing the masks can be time consuming, especially around fine structures such as the rear tire of the bicycle. The palette-based color editing results in unintended color changes for some edits, especially the effect of changing the color of the ground at the last step where a large region in the sea also gets affected. Using the layers by AO and KNN also gives similar results, although some smoothness issues can be observed. RGBSG layers did not allow the intended edits to be applied due to their color content in this example but we

Figure 4.19.: *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., 2013a] (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.*

Figure 4.20.: *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., 2013a] (e), and RGBSG [Tan et al., 2016] (f). AO rows lack one step (the color of the cup in the background in the bottom) because the layers corresponding to the intended edits did not exist in its color model. See text for discussion.*

included it in the figure for completeness. The blue layer of RGBSG covers the sky, sea and the ground and hence the targeted edits could not have been applied.

The example in Figure 4.20 is more challenging for the competing approaches. The simple masks that was useful in Figure 4.19 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 4.21. 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 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 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.

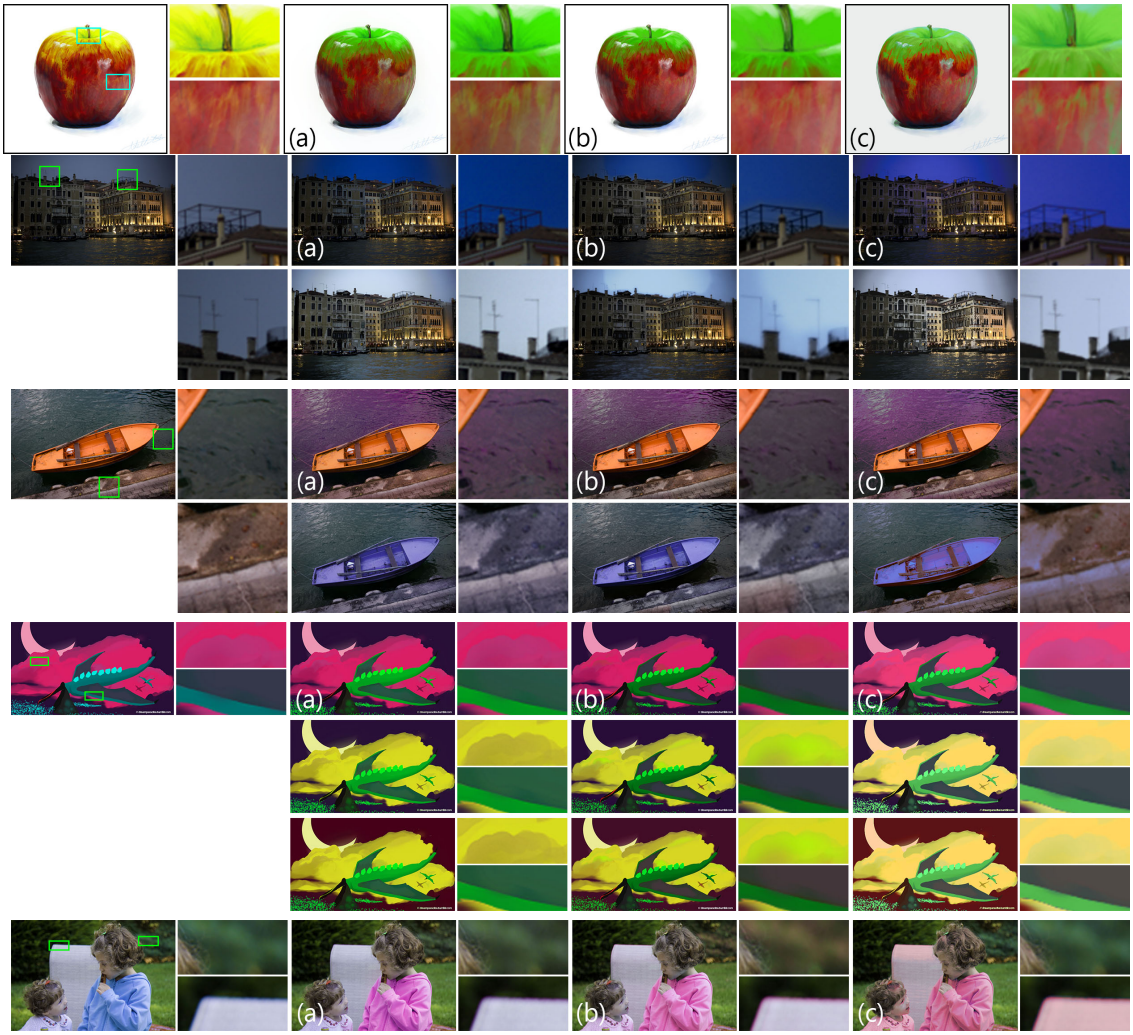

Figure 4.21.: *Color editing results using our layers (a), layers by Tan et al. [2016] (b) and using the recoloring application by Chang et al. [2015] (c) on images used by Tan et al. [2016] and Chang et al. [2015] in their papers. See text for discussion.*

4.7. 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 4.22(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 4.22(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.

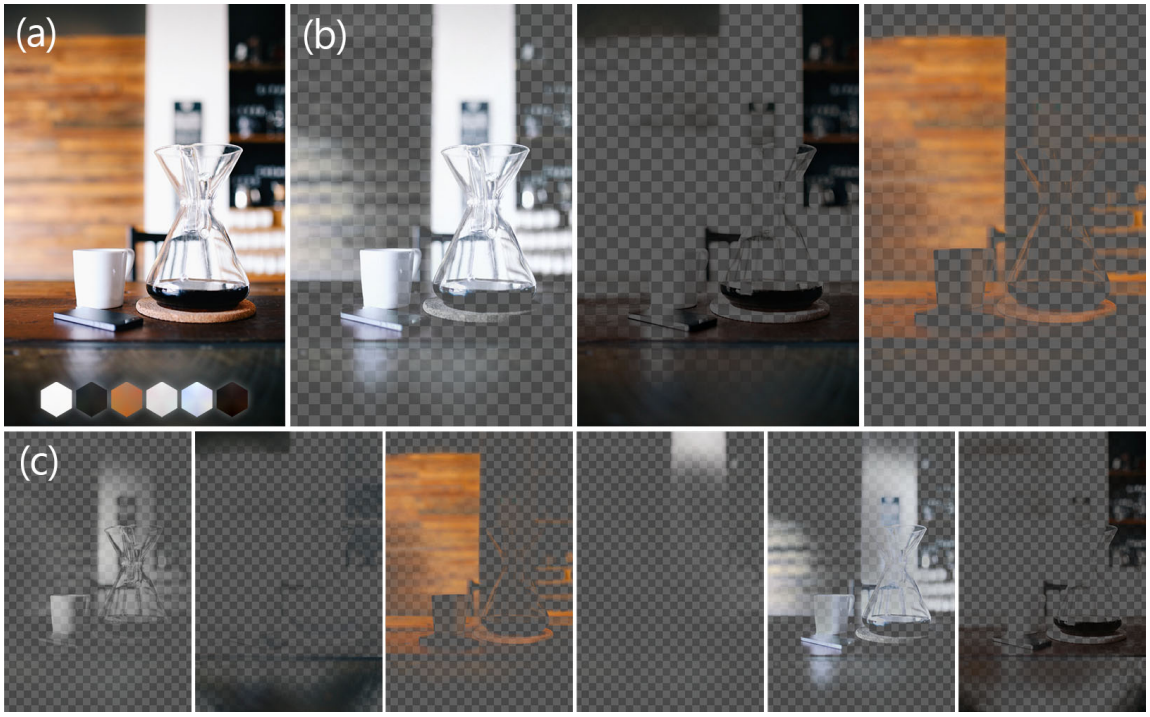

Figure 4.22.: *In some cases, the proposed color model estimation algorithm may give more layers than the user intends to make use of, as seen in the bottom row (c). However, these layers can easily be combined to be edited together, an example of which is shown in the top row (b) with the original image and the color model (a).*

We showed that our soft color segmentation method can also be used for green-screen keying without the two-step interaction process. That said, our sparse color unmixing energy formulation (Section 4.1), 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 our targeted pipeline in Chapter 3.

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 4.4.1. However, one limitation is that it can not 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 which 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

Figure 4.23.: *Although our automatic method is not able to deal with color spill (c) as well as our targeted keying method in Chapter 3 (b), it is possible to use commercial keying software such as Keylight to get rid of the spill as a post-processing step (d).*

.

information such as semantic segmentation or face detectors, the spatial extent of our soft segments do not necessarily overlap with semantically meaningful object boundaries. We will discuss a soft segmentation approach that targets detecting semantically meaningful soft boundaries in Chapter 6. It is fairly easy to limit the spatial extent of soft color layers by masking with semantic regions since our layers naturally integrate into current image manipulation software.

Part II.

Affinity-Based Matting

CHAPTER 5

Effective Inter-Pixel Information Flow for Natural Image Matting

Natural image matting can be seen as a generalization of the green-screen keying problem we studied in Chapter 3, where the background is not controlled and can contain any objects or image structures. This greatly increases the complexity of the problem, and a color-focused approach does not suffice to model the complex soft transitions between the foreground object and its surroundings. To address this complexity, we will rely on a graph-based approach where each pixel is modeled as a node on the graph, and the edges in the graph are inserted to best represent the soft transitions and color mixtures.

The under-constrained nature of this problem is typically alleviated by an additional input called *trimap*. Trimaps consist of three regions: fully opaque (foreground), fully transparent (background) and of unknown opacity. $\mathcal{F}$, $\mathcal{B}$ and $\mathcal{U}$ will respectively denote these regions, and $\mathcal{K}$ will represent the union of $\mathcal{F}$ and $\mathcal{B}$. Graph-based methods, commonly reffered to as affinity-based in the literature, operate by propagating opacity information from $\mathcal{K}$ into $\mathcal{U}$ using a variety of affinity definitions. The state-of-the-art in affinity-based matting usually rely on a single affinity definition, which results in limited generalizability to complex foreground structures. Figure 5.1 demonstrates the shortcomings of single-affinity approaches in two simple examples. We define this flow of information in multiple ways so that each pixel in $\mathcal{U}$ receives information effectively from different regions in the image. The quality increase in the matting results with each additional affinity definition can be seen in Figure 5.2.

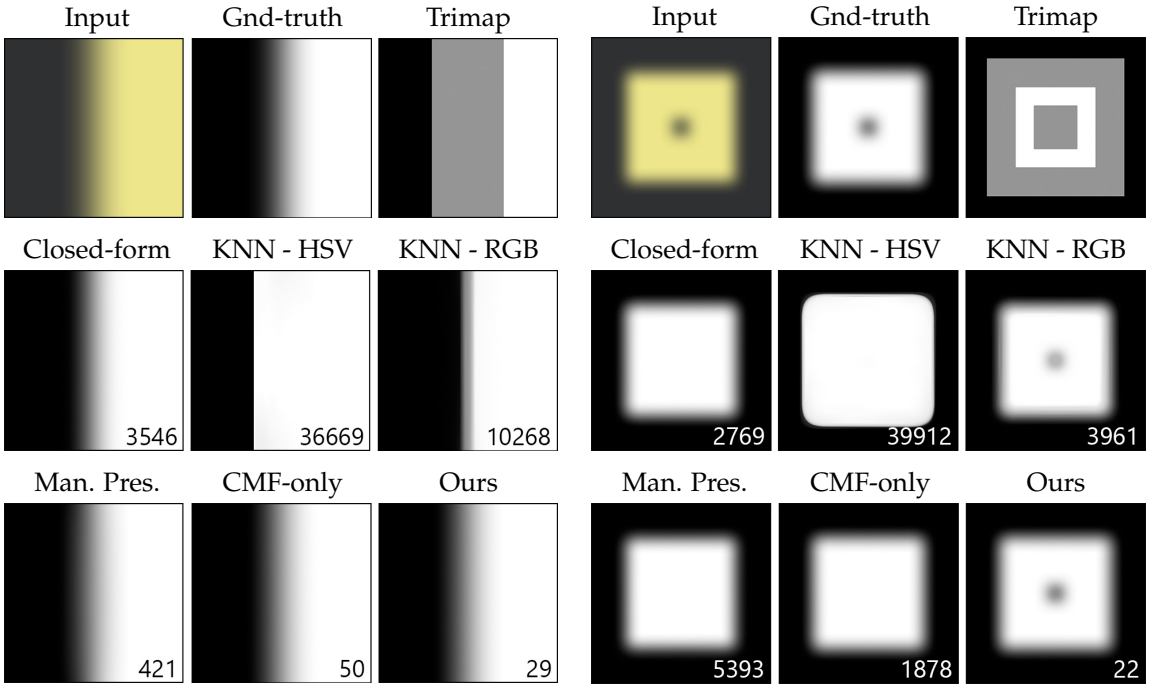

Figure 5.1.: *We created two duotone 500x500 images and blurred them to get soft transitions between regions. The numbers show the sum of absolute differences between the estimated alpha mattes and the ground truth. Closed-form matting [Levin et al., 2008a] uses local information flow, KNN Matting [Chen et al., 2013a] uses HSV- or RGB-based similarity measure, and manifold-preserving edit propagation [Chen et al., 2012] uses LLE weights [Roweis and Saul, 2000]. We observe a performance improvement in large opacity gradients even when only the color-mixture flow (CMF) is used (Section 5.1.1). Notice also that both large gradients and holes are recovered with high performance using our final formulation. See text for further discussion.*

We apply the multiple information-flow approach to related problems in natural matting, matte refinement and layer color estimation, in Section 5.2 and Section 5.3, respectively. We also provide a spectral analysis of different affinity choices as well as an analysis of state-of-the-art sampling-based matting techniques later in this chapter.

5.1. Information-Flow Matting

The opacity transitions in a matte occur as a result of the original colors in the image getting mixed with each other due to transparency or intricate parts of an object. We make use of this fact by representing each pixel in $\mathcal{U}$ as a mixture of similarly-colored pixels and defining a form of information flow that we call *color-mixture flow* (Section 5.1.1). We also add connections from

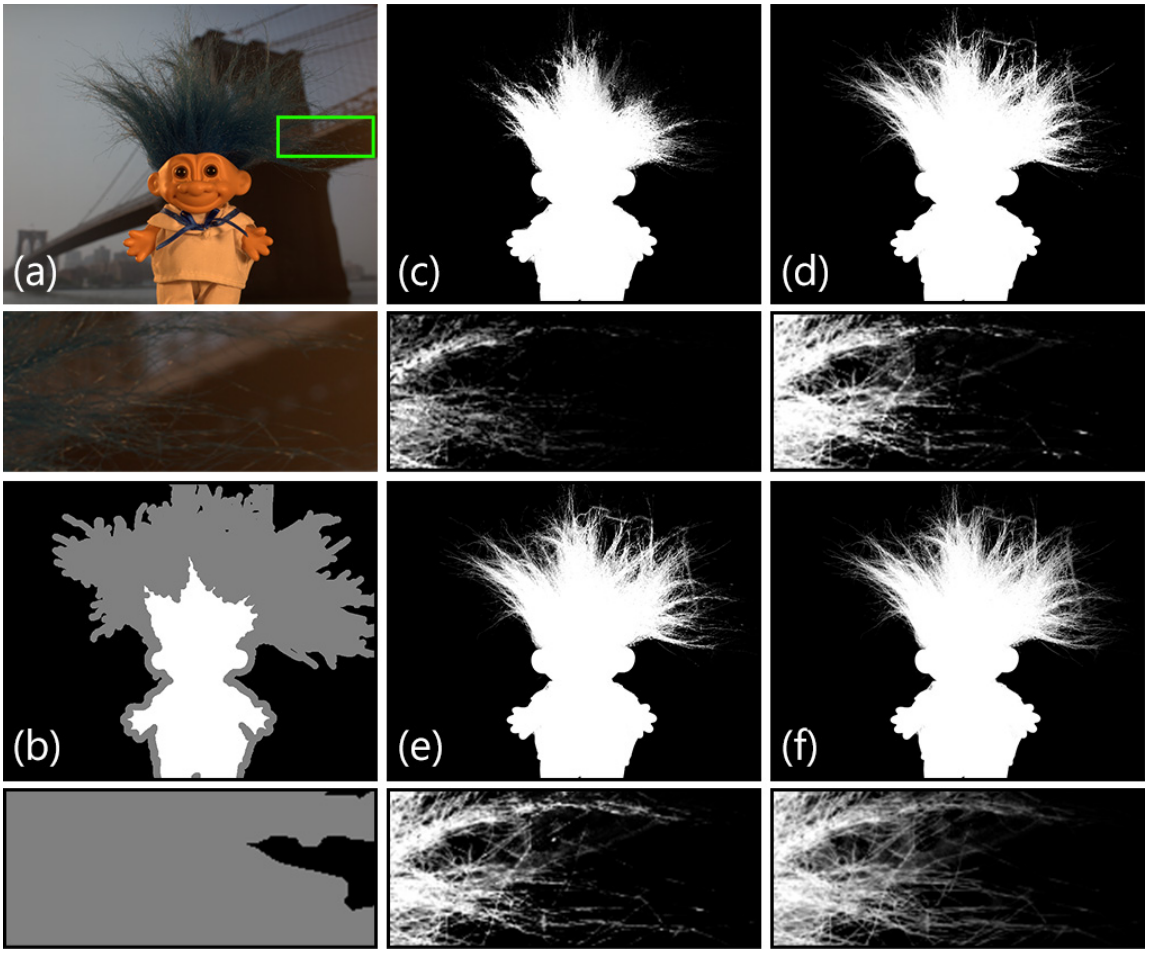

Figure 5.2.: *For an input image and a trimap (a), we construct our linear system by first using the color-mixture flow (b), then adding direct channels of information flow from known to unknown regions (c), and letting information be shared effectively inside the unknown region (d). We finally introduce local information flow to enforce spatial smoothness (e). Note that the intermediate results in this figure are solely for illustration. In practice, we construct a single energy function that accounts for all types of information flow and solve it once to obtain the end result.*

every pixel in $\mathcal{U}$ to both $\mathcal{F}$ and $\mathcal{B}$ to facilitate direct information flow from known-opacity regions to even the most remote opacity-transition regions in the image (Section 5.1.2). In order to distribute the information from the color-mixture and $\mathcal{K}$-to-$\mathcal{U}$ flows, we define intra-$\mathcal{U}$ flow of information, where pixels with similar colors inside $\mathcal{U}$ share information on their opacity with each other (Section 5.1.3). Finally, we add local information flow, a pixel affecting the opacity of its immediate spatial neighbors, which ensures spatially coherent end results (Section 5.1.4). We formulate the individual

forms of information flow as energy functions and aggregate them in a global optimization formulation (Section 5.1.5).

5.1.1. Color-Mixture Information Flow

Due to transparent objects as well as fine structures and sharp edges of an object that cannot be fully captured due to the finite-resolution of the imaging sensors, certain pixels of an image inevitably contain a mixture of corresponding foreground and background colors. By investigating these color mixtures, we can derive an important clue on how to propagate alpha values between pixels. The amount of the original foreground color in a particular mixture determines the opacity of the pixel. Following this fact, if we represent the color of a pixel as a weighted combination of the colors of several others, those weights should correspond to the opacity relation between the pixels.

In order to make use of this relation, for every pixel in $\mathcal{U}$, we find $K_{CM} = 20$ similar pixels in a feature space by an approximate K nearest neighbors search in the whole image. We define the feature vector for this search as $[r, g, b, \tilde{x}, \tilde{y}]^T$, where $\tilde{x}$ and $\tilde{y}$ are the image coordinates normalized by image width and height, and the rest are the RGB values of the pixel. This set of neighbors, selected as similar-colored pixels that are also close-by, is denoted by $\mathcal{N}_p^{CM}$.

We then find the weights of the combination $w_{p,q}^{CM}$ that will determine the amount of information flow between the pixels p and $q \in \mathcal{N}_p^{CM}$. The weight of each neighbor is defined such that the weighted combination of their colors yields the color of the original pixel:

$$\arg\min_{w_{p,q}^{CM}} \left\| \boldsymbol{c}_p - \sum_{q \in \mathcal{N}_p^{CM}} w_{p,q}^{CM} \boldsymbol{c}_q \right\|^2, \tag{5.1}$$

where $\boldsymbol{c}_p$ represents the 3x1 vector of RGB values. We minimize this energy using the method by Roweis and Saul [2000]. Note that since we are only using RGB values, the neighborhood correlation matrix computed during the minimization has a high chance of being singular as there could easily be two neighbors with identical colors. So, we condition the neighborhood correlation matrix by adding $10^{-3} I_{K_{CM} \times K_{CM}}$ to it before inversion, where $I_{K_{CM} \times K_{CM}}$ is the identity matrix.

Note that while we use the method by Roweis and Saul [2000] to minimize the energy in (5.1), we do not fully adopt their local linear embedding (LLE) method. LLE finds a set of neighbors in a feature space and uses all the

variables in the feature space to compute the weights in order to reduce the dimentionality of input data. Manifold-preserving edit propagation [Chen et al., 2012] and LNSP matting [Chen et al., 2013b] algorithms make use of the LLE weights directly in their formulation for image matting. However, since we are only interested in the weighted combination of colors and not the spatial coordinates, we exclude the spatial coordinates in the energy minimization step. This increases the validity of the estimated weights, effects of which can be observed even in the simplest cases such as in Figure 5.1, where manifold-preserving weight propagation and CMF-only results only differ in the weight computation step.

The energy term for the color-mixture flow is defined as:

$$E_{CM} = \sum_{p \in \mathcal{U}} \left(\alpha_p - \sum_{q \in \mathcal{N}_p^{CM}} w_{p,q}^{CM} \alpha_q \right)^2 . \tag{5.2}$$

5.1.2. $\mathcal{K}$-to-$\mathcal{U}$ Information Flow

The color-mixture flow already provides useful information on how the mixed-color pixels are formed. However, many pixels in $\mathcal{U}$ receive information present in the trimap indirectly through their neighbors, all of which can possibly be in $\mathcal{U}$. This indirect information flow might not be enough especially for remote regions that are far away from $\mathcal{K}$.

In order to facilitate the flow of information from both $\mathcal{F}$ and $\mathcal{B}$ directly into every region in $\mathcal{U}$, we add connections from every pixel in $\mathcal{U}$ to several pixels in $\mathcal{K}$. For each pixel in $\mathcal{U}$, we find $K_{\mathcal{KU}} = 7$ similar pixels in both $\mathcal{F}$ and $\mathcal{B}$ separately to form the sets of pixels $\mathcal{N}_p^{\mathcal{F}}$ and $\mathcal{N}_p^{\mathcal{B}}$ with K nearest neighbors search using the feature space $[r, g, b, 10 * \tilde{x}, 10 * \tilde{y}]^T$ to favor close-by pixels. We use the pixels in $\mathcal{N}_p^{\mathcal{F}}$ and $\mathcal{N}_p^{\mathcal{B}}$ together to represent the pixel color $\boldsymbol{c}_p$ by minimizing the energy in (5.1). Using the resulting weights $w_{p,q}^{\mathcal{F}}$ and $w_{p,q}^{\mathcal{B}}$, we define an energy function to represent the $\mathcal{K}$-to-$\mathcal{U}$ flow:

$$E_{\mathcal{KU}} = \sum_{p \in \mathcal{U}} \left(\alpha_p - \sum_{q \in \mathcal{N}_p^{\mathcal{F}}} w_{p,q}^{\mathcal{F}} \alpha_q - \sum_{q \in \mathcal{N}_p^{\mathcal{B}}} w_{p,q}^{\mathcal{B}} \alpha_q \right)^2 \tag{5.3}$$

Note that $\alpha_q = 1$ for $q \in \mathcal{F}$ and $\alpha_q = 0$ for $q \in \mathcal{B}$. This fact allows us to define two combined weights, one connecting a pixel to $\mathcal{F}$ and another to $\mathcal{B}$, as:

$$w_p^{\mathcal{F}} = \sum_{q \in \mathcal{N}_p^{\mathcal{F}}} w_{p,q}^{\mathcal{F}} \quad \text{and} \quad w_p^{\mathcal{B}} = \sum_{q \in \mathcal{N}_p^{\mathcal{B}}} w_{p,q}^{\mathcal{B}} \tag{5.4}$$

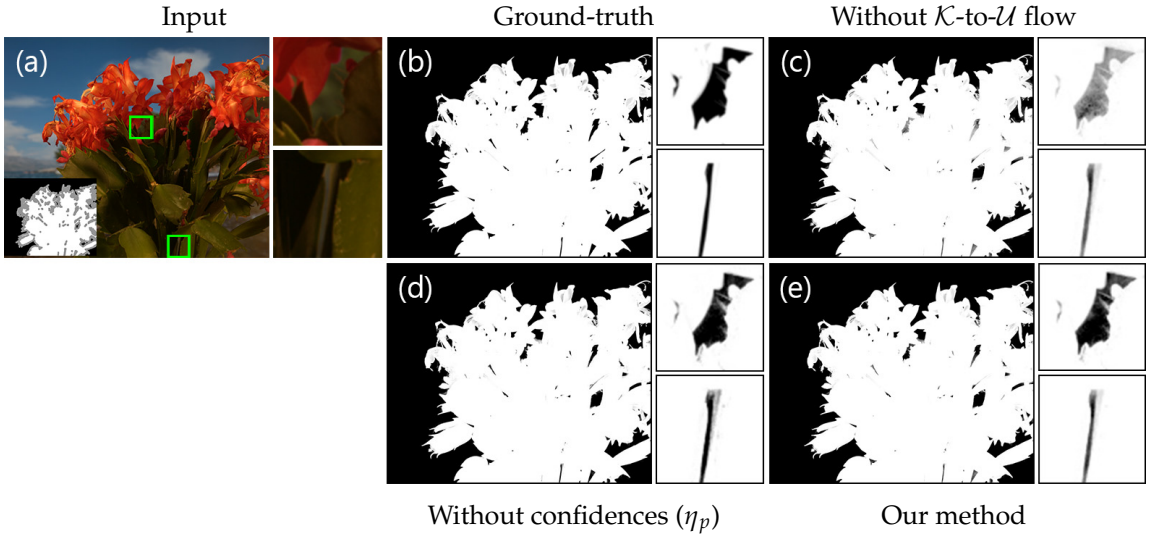

Figure 5.3.: *Direct information flow from both $\mathcal{F}$ and $\mathcal{B}$ to even the most remote regions in $\mathcal{U}$ increases our performance around holes significantly (top inset). Using confidences further increases the performance, especially around regions where FG and BG colors are similar (bottom inset).*

such that $w_p^{\mathcal{F}} + w_p^{\mathcal{B}} = 1$, and rewrite (5.3) as:

$$E_{\mathcal{KU}} = \sum_{p \in \mathcal{U}} \left(\alpha_p - w_p^{\mathcal{F}} \right)^2 . \tag{5.5}$$

The energy minimization in (5.1) gives us similar weights for all q when $\boldsymbol{c}_q$ are similar to each other. As a result, if $\mathcal{N}_p^{\mathcal{F}}$ and $\mathcal{N}_p^{\mathcal{B}}$ have pixels with similar colors, the estimated weights $w_p^{\mathcal{F}}$ and $w_p^{\mathcal{B}}$ become unreliable. We account for this fact by augmenting the energy function in (5.5) with confidence values.

We can determine the colors contributing to the mixture estimated by (5.1) using the weights $w_{p,q}^{\mathcal{F}}$ and $w_{p,q}^{\mathcal{B}}$:

$$\boldsymbol{c}_p^{\mathcal{F}} = \frac{\sum_{q \in \mathcal{N}_p^{\mathcal{F}}} w_{p,q}^{\mathcal{F}} \boldsymbol{c}_q}{w_p^{\mathcal{F}}}, \quad \boldsymbol{c}_p^{\mathcal{B}} = \frac{\sum_{q \in \mathcal{N}_p^{\mathcal{B}}} w_{p,q}^{\mathcal{B}} \boldsymbol{c}_q}{w_p^{\mathcal{B}}}, \tag{5.6}$$

and define a confidence metric according to how similar the estimated foreground color $\boldsymbol{c}_p^{\mathcal{F}}$ and background color $\boldsymbol{c}_p^{\mathcal{B}}$ are:

$$\eta_p = \left\| \boldsymbol{c}_p^{\mathcal{F}} - \boldsymbol{c}_p^{\mathcal{B}} \right\|^2 / 3. \tag{5.7}$$

The division by 3 is to get the confidence values between $[0, 1]$. We update

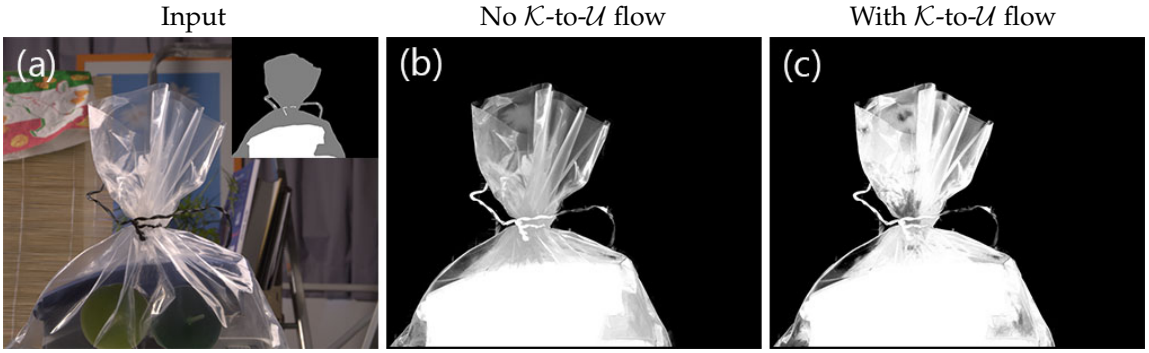

Figure 5.4.: *$\mathcal{K}$-to-$\mathcal{U}$ flow does not perform well when the foreground object is highly-transparent. See text for discussion.*

the new energy term to reflect our confidence in the estimation:

$$\tilde{E}_{\mathcal{KU}} = \sum_{p \in \mathcal{U}} \eta_p \left(\alpha_p - w_p^{\mathcal{F}} \right)^2 . \tag{5.8}$$

This update to the energy term increases the matting quality in regions with similar foreground and background colors, as seen in Figure 5.3.

It should be noted that the $\mathcal{K}$-to-$\mathcal{U}$ flow is not reliable when the foreground is highly transparent, as seen in Figure 5.4. This is mainly due to the low representational power of $\mathcal{N}_p^{\mathcal{F}}$ and $\mathcal{N}_p^{\mathcal{B}}$ for $\boldsymbol{c}_p$ around large highly-transparent regions as the nearest neighbors search does not give us well-fitting pixels for $w_{p,q}^{\mathcal{F}}$ estimation. We construct our final linear system accordingly in Section 5.1.5.

Pre-processing the trimap

Prior to determining $\mathcal{N}_p^{\mathcal{F}}$ and $\mathcal{N}_p^{\mathcal{B}}$, we pre-process the input trimap in order to facilitate finding more reliable neighbors, which in turn increases the effectiveness of the $\mathcal{K}$-to-$\mathcal{U}$ flow. Trimaps usually have regions marked as $\mathcal{U}$ despite being fully opaque or transparent, as drawing a very detailed trimap is both cumbersome and prone to errors.

Several methods [Feng et al., 2016; Karacan et al., 2015] refine the trimap as a pre-processing step by expanding $\mathcal{F}$ and $\mathcal{B}$ starting from their boundaries with $\mathcal{U}$ as proposed by Shahrian et al. [2013]. Incorporating this technique improves our results as shown in Figure 5.5(d). We also apply this extended $\mathcal{F}$ and $\mathcal{B}$ regions after the matte estimation as a post-processing. Since this trimap trimming method propagates known regions only to nearby pixels, in addition to this edge-based trimming, we also make use of a patch-based trimming step.

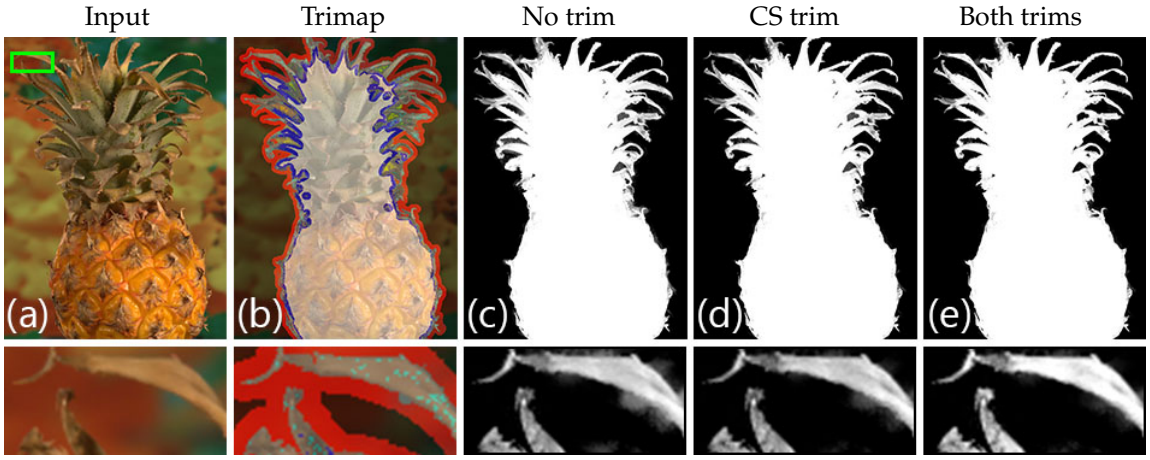

Figure 5.5.: *The trimap is shown overlayed on the original image (b) where the extended foreground regions are shown with blue (CS trimming [Shahrian et al., 2013]) and cyan (patch-search) and the extended background regions with red (CS trimming) and yellow (patch-search). CS trimming makes the fully opaque / transparent regions cleaner, while our trimming improves the results around remote structures.*

To this end, we extend the transparent and opaque regions by relying on patch statistics. We fit a 3D RGB normal distribution N_p to the 3×3 window around each pixel p. In order to determine the most similar distribution in $\mathcal{F}$ for a pixel $p \in \mathcal{U}$, we first find the 20 distributions with closest mean vectors. We define the foreground match score $b_p^{\mathcal{F}} = \min_{q \in \mathcal{F}} B(N_p, N_q)$, where $B(\cdot, \cdot)$ represents the Bhattacharyya distance between two distributions. We find the match score for background $b_p^{\mathcal{B}}$ the same way. We then select a region for pixel p according to the following rule:

$$p \in \begin{cases} \hat{\mathcal{F}} & \text{if } b_p^{\mathcal{F}} < \tau_c \text{ and } b_p^{\mathcal{B}} > \tau_f \\ \hat{\mathcal{B}} & \text{if } b_p^{\mathcal{B}} < \tau_c \text{ and } b_p^{\mathcal{F}} > \tau_f \\ \hat{\mathcal{U}} & \text{otherwise} \end{cases} \tag{5.9}$$

Simply put, an unknown pixel is marked as $\hat{\mathcal{F}}$, i.e. in foreground after trimming, if it has a strong match in $\mathcal{F}$ and no match in $\mathcal{B}$, which is determined by constants $\tau_c = 0.25$ and $\tau_f = 0.9$. By inserting known-alpha pixels in regions far away from $\mathcal{U}$-$\mathcal{K}$ boundaries, we further increase the matting performance in challenging remote regions (Figure 5.5(e)).

5.1.3. Intra-$\mathcal{U}$ Information Flow

Each individual pixel in $\mathcal{U}$ receives information through the color-mixture and $\mathcal{K}$-to-$\mathcal{U}$ flows. In addition to these, we would like to distribute the

information inside $\mathcal{U}$ effectively. We achieve this by encouraging pixels with similar colors inside $\mathcal{U}$ to have similar opacity.

For each pixel in $\mathcal{U}$, we find $K_{\mathcal{U}} = 5$ nearest neighbors only inside $\mathcal{U}$ to determine $\hat{\mathcal{N}}_p^{\mathcal{U}}$ using the feature vector defined as $\boldsymbol{v} = [r, g, b, \tilde{x}/20, \tilde{y}/20]^T$. Notice that we scale the coordinate members of the feature vector we used in Section 5.1.1 to decrease their effect on the nearest neighbor selection. This lets $\hat{\mathcal{N}}_p^{\mathcal{U}}$ have pixels inside $\mathcal{U}$ that are far away, so that the information moves more freely inside the unknown region. We use the neighborhood $\mathcal{N}_p^{\mathcal{U}} = \hat{\mathcal{N}}_p^{\mathcal{U}} \cup \{q \mid p \in \hat{\mathcal{N}}_q^{\mathcal{U}}\}$ to make sure that information flows both ways between p to $q \in \hat{\mathcal{N}}_p^{\mathcal{U}}$. We then determine the amount of information flow using the L^1 distance between feature vectors:

$$w_{p,q}^{\mathcal{U}} = \max\left(1 - \left\|\boldsymbol{v}_p - \boldsymbol{v}_q\right\|_1, 0\right) \quad \forall q \in \mathcal{N}_p^{\mathcal{U}}. \tag{5.10}$$

The energy term for intra-$\mathcal{U}$ flow then can be defined as:

$$E_{\mathcal{U}\mathcal{U}} = \sum_{p \in \mathcal{U}} \sum_{q \in \mathcal{N}_p^{\mathcal{U}}} w_{p,q}^{\mathcal{U}} \left(\alpha_p - \alpha_q\right)^2. \tag{5.11}$$

The information sharing between the unknown pixels increases the matte quality around intricate structures as demonstrated in Figure 5.2(d).

KNN matting [Chen et al., 2013a] uses a similar affinity definition to make similar-color pixels have similar opacities. However, relying only on this form of information flow for the whole image creates some typical artifacts in the matte. Depending on the feature vector definition and the image colors, the matte may erroneously underrepresent the smooth transitions (KNN - HSV case in Figure 5.1) when the neighbors of the pixels in $\mathcal{U}$ happen to be mostly in only $\mathcal{F}$ or $\mathcal{B}$, or create flat alpha regions instead of subtle gradients (KNN - RGB case in Figure 5.1). Restricting information flow to be solely based on color similarity fails to represent the complex alpha transitions or wide regions with an alpha gradient.

5.1.4. Local Information Flow

Spatial connectivity is one of the main cues for information flow. We connect each pixel in $\mathcal{U}$ to its 8 immediate neighbors denoted by $\mathcal{N}_p^L$ to ensure spatially smooth mattes. The amount of local information flow should also adapt to strong edges in the image.

To determine the amount of local flow, we rely on the matting affinity definition proposed by Levin et al. [2008a]. The matting affinity utilizes the local

patch statistics to determine the weights $w^L_{p,q}$, $q \in \mathcal{N}^L_p$. We define our related energy term as follows:

$$E_L = \sum_{p \in \mathcal{U}} \sum_{q \in \mathcal{N}^L_p} w^L_{p,q} \left(\alpha_p - \alpha_q\right)^2. \tag{5.12}$$

Despite representing local information flow well, matting affinity by itself fails to represent large transition regions (Figure 5.1 top), or isolated regions that have weak or no spatial connection to $\mathcal{F}$ or $\mathcal{B}$ (Figure 5.1 bottom).

5.1.5. Linear System and Energy Minimization

Our final energy function is a combination of the four energies representing the individual information flows:

$$E_1 = E_{CM} + \sigma_{\mathcal{KU}} E_{\mathcal{KU}} + \sigma_{\mathcal{UU}} E_{\mathcal{UU}} + \sigma_L E_L + \lambda E_{\mathcal{T}}, \tag{5.13}$$

where $\sigma_{\mathcal{KU}} = 0.05$, $\sigma_{\mathcal{UU}} = 0.01$, $\sigma_L = 1$ and $\lambda = 100$ are algorithmic constants determining the strength of corresponding information flows, and

$$E_{\mathcal{T}} = \sum_{p \in \mathcal{F}} \left(\alpha_p - 1\right)^2 + \sum_{p \in \mathcal{B}} (\alpha_p - 0)^2$$

is the energy term to keep the known opacity values constant. For an image with N pixels, by defining $N \times N$ sparse matrices W_{CM}, $W_{\mathcal{UU}}$ and W_L that have non-zero elements for the pixel pairs with corresponding information flows and the vector $\boldsymbol{w}^{\mathcal{F}}$ that has elements $w^{\mathcal{F}}_p$ for $p \in \mathcal{U}$, 1 for $p \in \mathcal{F}$ and 0 for $p \in \mathcal{B}$, we can write (5.13) in matrix form as:

$$\begin{aligned} E_1 =& \boldsymbol{\alpha}^T \mathcal{L}_{IFM} \boldsymbol{\alpha} + (\boldsymbol{\alpha} - \boldsymbol{w}^{\mathcal{F}})^T \sigma_{\mathcal{KU}} \mathcal{H} (\boldsymbol{\alpha} - \boldsymbol{w}^{\mathcal{F}}) + \\ & (\boldsymbol{\alpha} - \boldsymbol{\alpha}_{\mathcal{K}})^T \lambda \mathcal{T} (\boldsymbol{\alpha} - \boldsymbol{\alpha}_{\mathcal{K}}), \end{aligned} \tag{5.14}$$

where $\mathcal{T}$ is an $N \times N$ diagonal matrix with diagonal entry (p, p) 1 if $p \in \mathcal{K}$ and 0 otherwise, $\mathcal{H}$ is a sparse matrix with diagonal entries η_p as defined in (5.7), $\boldsymbol{\alpha}_{\mathcal{K}}$ is a row vector with p^{th} entry being 1 if $p \in \mathcal{F}$ and 0 otherwise, $\boldsymbol{\alpha}$ is a row-vector of the alpha values to be estimated, and $\mathcal{L}_{IFM}$ is defined as:

$$\begin{aligned} \mathcal{L}_{IFM} =& (D_{CM} - W_{CM})^T (D_{CM} - W_{CM}) + \\ & \sigma_{\mathcal{UU}} (D_{\mathcal{UU}} - W_{\mathcal{UU}}) + \sigma_L (D_L - W_L), \end{aligned} \tag{5.15}$$

where the diagonal matrix $D_{(\cdot)}(i,i) = \sum_j W_{(\cdot)}(i,j)$.

The energy in (5.14) can be minimized by solving

$$\left(\mathcal{L}_{IFM} + \lambda \mathcal{T} + \sigma_{\mathcal{KU}} \mathcal{H}\right) \boldsymbol{\alpha} = \left(\lambda \mathcal{T} + \sigma_{\mathcal{KU}} \mathcal{H}\right) \boldsymbol{w}^{\mathcal{F}}. \tag{5.16}$$

We define a second energy function that excludes the $\mathcal{K}$-to-$\mathcal{U}$ information flow:

$$E_2 = E_{CM} + \sigma_{\mathcal{U}\mathcal{U}} E_{\mathcal{U}\mathcal{U}} + \sigma_L E_L + \lambda E_{\mathcal{T}}, \tag{5.17}$$

which can be written in matrix form as:

$$E_2 = \boldsymbol{\alpha}^T \mathcal{L}_{IFM} \boldsymbol{\alpha} + (\boldsymbol{\alpha} - \boldsymbol{\alpha}_{\mathcal{K}})^T \lambda \mathcal{T} (\boldsymbol{\alpha} - \boldsymbol{\alpha}_{\mathcal{K}}), \tag{5.18}$$

and can be minimized by solving:

$$(\mathcal{L}_{IFM} + \lambda \mathcal{T})\, \boldsymbol{\alpha} = \lambda \mathcal{T} \boldsymbol{\alpha}_{\mathcal{K}}. \tag{5.19}$$

We solve the linear systems of equations in (5.16) and (5.19) using the preconditioned conjugate gradients method [Barrett et al., 1994].

As mentioned before, the $\mathcal{K}$-to-$\mathcal{U}$ information flow is not effective for highly transparent objects. To determine whether to include the $\mathcal{K}$-to-$\mathcal{U}$ information flow and solve for E_1, or to exclude it and solve for E_2 for a given image, we use a simple histogram-based classifier to determine if we expect a highly transparent result.

If the matte is highly transparent, the pixels in $\mathcal{U}$ are expected to mostly have colors that are a mixture of $\mathcal{F}$ and $\mathcal{B}$ colors. On the other hand, if the true alpha values are mostly 0 or 1 except for soft transitions, the histogram of $\mathcal{U}$ will likely be a linear combination of the histograms of $\mathcal{F}$ and $\mathcal{B}$ as $\mathcal{U}$ will mostly include very similar colors to that of $\mathcal{K}$. Following this observation, we attempt to express the histogram of the pixels in $\mathcal{U}$, $\mathcal{D}_{\mathcal{U}}$, as a linear combination of $\mathcal{D}_{\mathcal{F}}$ and $\mathcal{D}_{\mathcal{B}}$. The histograms are computed from the 20 pixel-wide region around $\mathcal{U}$ in $\mathcal{F}$ and $\mathcal{B}$, respectively. We define the error e, the metric of how well the linear combination represents the true histogram, as:

$$e = \min_{a,b} \|a\mathcal{D}_{\mathcal{F}} + b\mathcal{D}_{\mathcal{B}} - \mathcal{D}_{\mathcal{U}}\|^2. \tag{5.20}$$

Higher e values indicate a highly-transparent matte, in which case we prefer E_2 over E_1.

5.2. Matte Regularization for Sampling-Based Matting Methods

Sampling-based natural matting methods usually select samples for each pixel in $\mathcal{U}$ either independently or by paying little attention to spatial coherency. In order to obtain a spatially coherent matte, the common practice is to combine their initial guesses for alpha values with a smoothness measure. Multiple methods [Feng et al., 2016; Gastal and Oliveira, 2010; Karacan et al., 2015; Shahrian et al., 2013] adopt the post-processing method

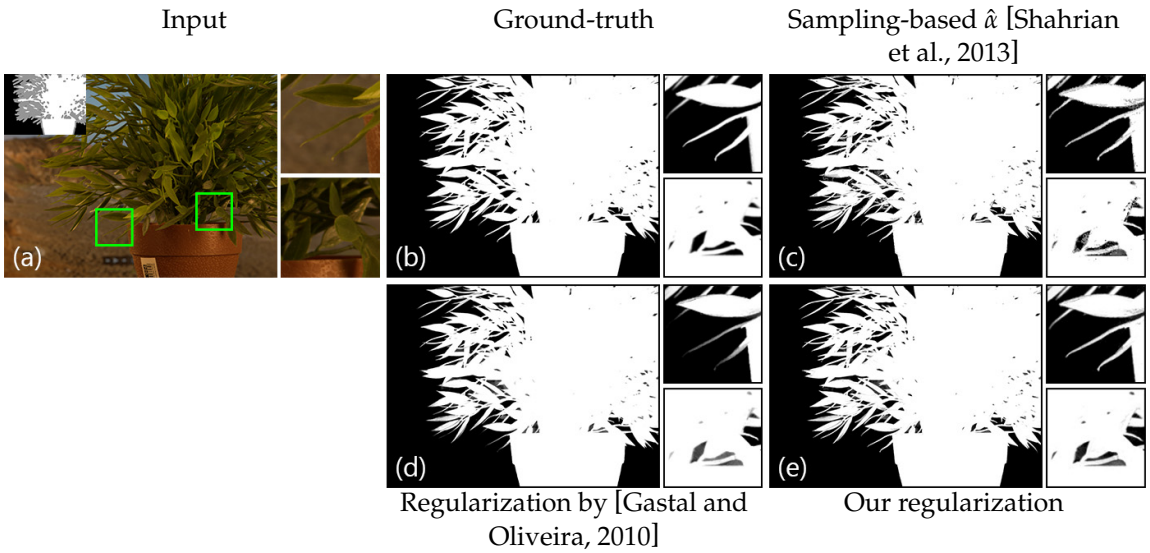

Figure 5.6.: *The matte regularization method by Gastal and Oliveira [2010] loses remote details (top inset) or fills in holes (bottom inset) while our regularization method is able to preserve these details caught by the sampling-based method.*

proposed by Gastal and Oliveira [2010] which combines the matting affinity [Levin et al., 2008a] with the sampling-based alpha values and corresponding confidences. This post-processing technique leads to improved mattes, but since it involves only local smoothness, the results can still be suboptimal as seen in Figure 5.6(d).

Our approach with multiple forms of information flow can also be used for post-processing in a way similar to that of Gastal and Oliveira [2010]. Given the initial alpha values $\hat{\alpha}_p$ and confidences $\hat{\eta}_p$ found by a sampling-based method, we define the matte regularization energy:

$$E_R = E_2 + \sigma_R \sum_{p \in \mathcal{U}} \hat{\eta}_p (\alpha_p - \hat{\alpha}_p)^2, \tag{5.21}$$

where $\sigma_R = 0.05$ determines how much loyalty should be given to the initial values. This energy can be written in matrix form and solved as a linear system in the same way we did in Section 5.1.5.

Figure 5.6 shows that this non-local regularization of mattes is more effective especially around challenging foreground structures such as long leaves or holes as seen in the insets. In Section 5.4.2, we will numerically explore the improvement we achieve by replacing the matte regularization step with ours in several sampling-based methods.

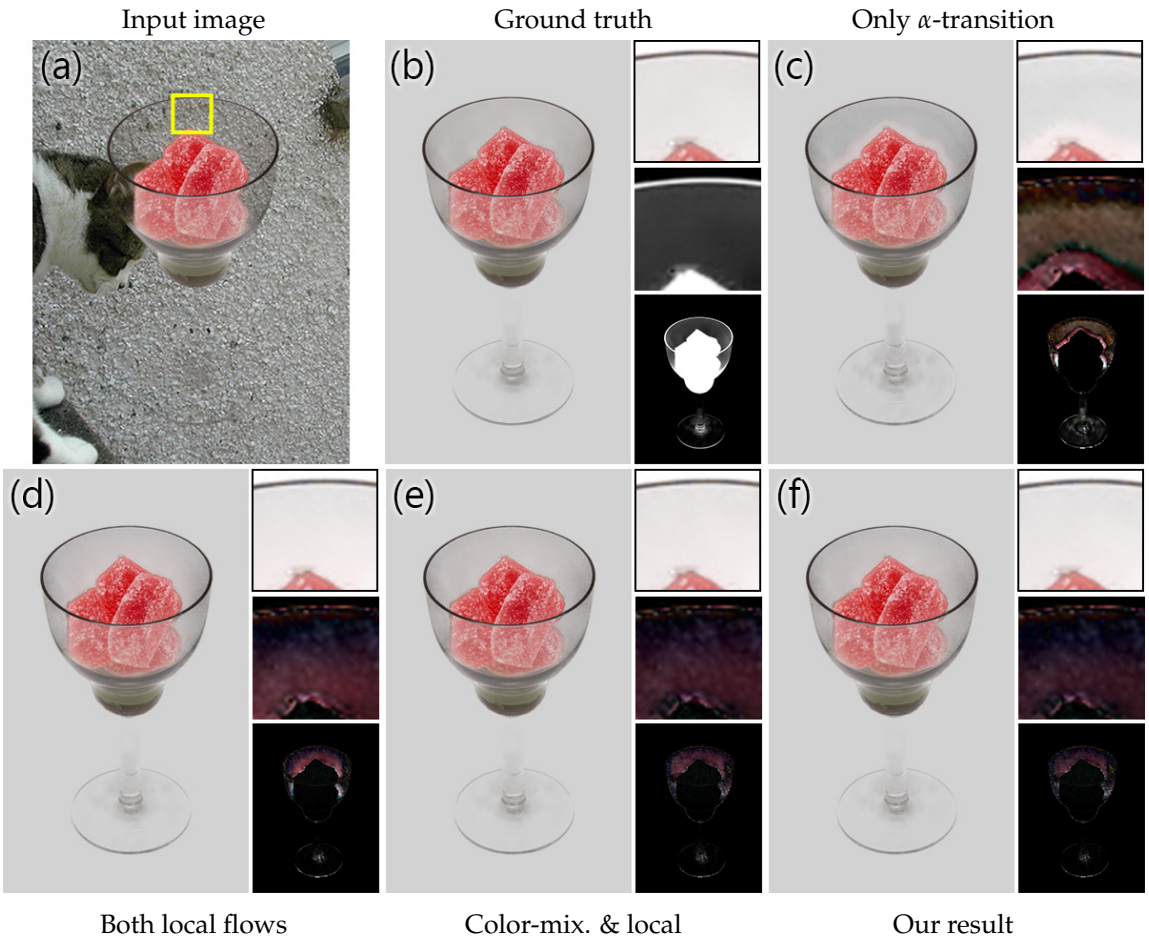

Figure 5.7.: *Color estimation results using a growing set of information flows using the ground truth matte. The bottom-right in each set shows per-pixel absolute difference between the estimation and ground truth multiplied by ten. See text for discussion.*

5.3. Foreground Color Estimation

In addition to the alpha matte, we need the *unmixed* foreground colors that got into the color mixture in transition pixels for seamlessly compositing the foreground onto a novel background. Similar to Levin et al. [2008a] and Chen et al. [2013a], we estimate the foreground colors for a given matte, after the matte estimation.

We propagate the layer colors from opaque and transparent regions in a similar way we propagate known alpha values in Section 5.1. We make use of the color-mixture and the intra-$\mathcal{U}$ information flows by extending the search space and affinity computation to include the given alpha values together with spatial coordinates and pixel colors. We also use the spatial smoothness

measure proposed by Levin et al. [2008a] in addition to a second spatial smoothness measure we introduce later in this section. Figure 5.7 shows how our color estimation result improves as we add more forms of information flow.

5.3.1. Information Flow Definitions

In the layer color estimation problem, the input is assumed to be the original image together with an alpha matte. This requires us to redefine the three regions using the matte instead of a trimap:

$$p \in \begin{cases} \tilde{\mathcal{F}} & \text{if } \tilde{\alpha}_p = 1 \\ \tilde{\mathcal{B}} & \text{if } \tilde{\alpha}_p = 0 \\ \tilde{\mathcal{U}} & \text{otherwise.} \end{cases} \tag{5.22}$$

$\tilde{\alpha}_p$ denote the alpha values that are given as input. The foreground and background colors to be estimated will be denoted by $\boldsymbol{f}$ and $\boldsymbol{b}$. For a pixel p, the compositing equation we would like to satisfy can be written as:

$$\boldsymbol{c}_p = \tilde{\alpha}_p \boldsymbol{f}_p + \left(1 - \tilde{\alpha}_p\right) \boldsymbol{b}_p \tag{5.23}$$

We will formulate the energy functions for a single color channel and solve for red, green and blue channels independently. The scalars f and b will denote the values for a single color channel.

Local information flows

Levin et al. [2008a] proposed the use of the gradient of the alpha channel as the amount of local information flow for the problem of layer color estimation. They solely rely on this form of information flow for propagating the colors. This local information flow basically enforces neighboring pixels to have similar colors if there is an alpha transition. This flow, which we refer to as *α-transition flow*, can be represented by the following energy:

$$E_{\nabla\tilde{\alpha}} = \sum_{\forall p} \sum_{q \in \mathcal{N}_p^L} \left|\nabla\tilde{\alpha}_{(p-q)}\right| \left(\left(f_p - f_q\right)^2 + \left(b_p - b_q\right)^2\right), \tag{5.24}$$

where $\nabla\tilde{\alpha}$ represents the alpha gradient. We compute the gradients in the image plane using the 3-tap separable filters of Farid and Simoncelli [2004]. Note that the neighborhood is defined as the local 3×3 neighborhood similar to the local information flow in Section 5.1.4.

The transition flow helps around small regions with alpha gradient but does not propagate information in flat-alpha regions, such as pure foreground or background regions or regions with flat opacity. We propose a new smoothness measure to address this issue, which we call *no-transition flow*. The no-transition flow enforces spatial smoothness in regions with small color and alpha gradients:

$$E_{\nabla \boldsymbol{c}\tilde{\alpha}} = \sum_{\forall p} \sum_{q \in \mathcal{N}_p^L} w_{p,q}^{\nabla \boldsymbol{c}\tilde{\alpha}} \left(\left(f_p - f_q\right)^2 + \left(b_p - b_q\right)^2 \right) \tag{5.25}$$

where $w_{p,q}^{\nabla \boldsymbol{c}\tilde{\alpha}} = \left(1 - |\nabla \tilde{\alpha}_{(p-q)}|\right) \left(1 - ||\nabla \boldsymbol{c}_{(p-q)}||\right)$ and $||\nabla \boldsymbol{c}_{(p-q)}||$ is the L_2 norm of the vector formed by gradients of the individual color channels. This term increases the performance around slow alpha transitions and flat-alpha regions, as well as around sharp color edges in the image.

No-transition flow already improves the performance quite noticably as seen in Figure 5.7(b). However, using only local information flows perform poorly in remote areas such as the end of long hair filaments (Figure 5.12(a)) or isolated areas (Figure 5.7, bottom inset). In order to increase the performance in these type of challenging areas, we make use of two types of non-local information flows.

Color-mixture information flow

The basic principle of color mixture as introduced in Section 5.1.1 also applies to the relationship between layer colors of pixels in the same neighborhood — if we represent the color and alpha of a pixel as a weighted combination of the colors and alpha of several others, those weights should also represent the layer color relation between the pixels. Since we have $\tilde{\alpha}$'s as additional information in the layer color estimation scenario, we extend the formulation of color-mixture flow to better fit the layer color estimation problem. Similar to its use in alpha estimation, it provides a well-connected graph and allows dense share of information. The performance improvement by the introduction of the color-mixture energy can be seen in Figure 5.7(c).

In the layer color estimation scenario, we optimize for both foreground and background colors in the same formulation. It should be emphasized that, as it is apparent from (5.23), the foreground and background colors are undefined for regions with $\tilde{\alpha} = 0$ and $\tilde{\alpha} = 1$, respectively. This requires us to avoid using color-mixture flow into $\tilde{\mathcal{U}}$ from $\tilde{\mathcal{B}}$ for $\boldsymbol{f}$ and from $\tilde{\mathcal{F}}$ for $\boldsymbol{b}$. We address this by defining two different neighborhoods and computing individual color-mixture flows for $\boldsymbol{f}$ and $\boldsymbol{b}$.

For f, we define the neighborhood $\mathcal{N}_p^{\tilde{\mathcal{U}}\tilde{\mathcal{F}}}$ by finding K_{CM} nearest neighbors in $(\tilde{\mathcal{U}} \cup \tilde{\mathcal{F}})$ using the feature vector $[r, g, b, \tilde{\alpha}, \tilde{x}, \tilde{y}]^T$. We then compute the weights $w_{p,q}^{C\tilde{\mathcal{F}}}$ as

$$\arg\min_{w_{p,q}^{C\tilde{\mathcal{F}}}} \left\| \begin{bmatrix} \boldsymbol{c}_p \\ \tilde{\alpha}_p \end{bmatrix} - \sum_{q \in \mathcal{N}_p^{CM}} w_{p,q}^{C\tilde{\mathcal{F}}} \begin{bmatrix} \boldsymbol{c}_q \\ \tilde{\alpha}_q \end{bmatrix} \right\|^2 . \tag{5.26}$$

Notice that the search space and the weight computation includes $\tilde{\alpha}$ in addition to the color and location of pixels.

We compute the background conjugates of the neighborhood and weights, $\mathcal{N}_p^{\tilde{\mathcal{U}}\tilde{\mathcal{B}}}$ and $w_{p,q}^{C\tilde{\mathcal{B}}}$, in the same way, and define our color-mixture energy for layer color estimation:

$$E_{CM}^{fb} = \sum_{p \in \tilde{\mathcal{U}}} ((f_p - \sum_{q \in \mathcal{N}_p^{\tilde{\mathcal{U}}\tilde{\mathcal{F}}}} w_{p,q}^{C\tilde{\mathcal{F}}} f_q)^2 + (b_p - \sum_{q \in \mathcal{N}_p^{\tilde{\mathcal{U}}\tilde{\mathcal{B}}}} w_{p,q}^{C\tilde{\mathcal{B}}} b_q)^2).$$

Intra-$\tilde{\mathcal{U}}$ information flow

Intra-$\mathcal{U}$ information flow, as detailed in Section 5.1.3, distributes the information between similar-colored pixels inside the unknown region without giving spatial proximity too much emphasis. Its behaviour is also very useful in the case of color estimation, as it makes the foreground colors more coherent throughout the image. For example, in Figure 5.7, bottom inset shows that the addition of intra-$\mathcal{U}$ flow helps in getting a more realistic color to the isolated plastic region between the two black lines.

We make modifications to intra-$\mathcal{U}$ flow similar to the modifications we made to color-mixture flow, in order to make use of the available information coming form $\tilde{\alpha}$'s.

We find $K_{\mathcal{U}}$ nearest neighbors only inside $\tilde{\mathcal{U}}$ to determine $\hat{\mathcal{N}}_p^{\tilde{\mathcal{U}}}$ using the feature vector defined as $\boldsymbol{v}^{\boldsymbol{c}} = [r, g, b, \tilde{\alpha}, \tilde{x}/20, \tilde{y}/20]^T$. We then determine the amount of information flow between two non-local neighbors as:

$$w_{p,q}^{\tilde{\mathcal{U}}} = \max\left(1 - \left\| \boldsymbol{v}_p^{\boldsymbol{c}} - \boldsymbol{v}_q^{\boldsymbol{c}} \right\|_1, 0\right) \quad \forall q \in \mathcal{N}_p^{\tilde{\mathcal{U}}}. \tag{5.27}$$

With the weights determined, we can define the energy function representing the intra-$\tilde{\mathcal{U}}$ flow:

$$E_{\tilde{\mathcal{U}}\tilde{\mathcal{U}}} = \sum_{p \in \tilde{\mathcal{U}}} \sum_{q \in \mathcal{N}_p^{\tilde{\mathcal{U}}}} w_{p,q}^{\tilde{\mathcal{U}}} \left(\left(f_p - f_q\right)^2 + \left(b_p - b_q\right)^2 \right). \tag{5.28}$$

Note that in the color estimation formulation, we exclude the $\mathcal{K}$-to-$\mathcal{U}$ information flow because we observed that the adaptation of the method in Section 5.1.2 to color estimation does not improve the quality of the final result.

5.3.2. Linear System and Energy Minimization

The final energy function for layer color estimation is the combination of the four types of information flow defined in Sections 5.3.1 to 5.3.1:

$$E_{\boldsymbol{c}} = \sigma_L E_{\nabla\alpha} + \sigma_L E_{\nabla \boldsymbol{c}\alpha} + E^{fb}_{CM} + \sigma_{\mathcal{U}\mathcal{U}} E_{\tilde{\mathcal{U}}\tilde{\mathcal{U}}} + \lambda E_{\text{COMP}}, \tag{5.29}$$

where σ_L, $\sigma_{\mathcal{U}\mathcal{U}}$ and λ are defined in Section 5.1.5 and E_{COMP} represents the deviation from the compositing equation constraint:

$$E_{\text{COMP}} = \sum_{\forall p} \Big(c_p - \alpha^I_p f - (1 - \alpha^I_p) b \Big)^2. \tag{5.30}$$

$E_{\boldsymbol{c}}$ is defined and minimized independently for each color channel.

Following the same strategy as we did in Section 5.1.5, we rewrite the energy function $E_{\boldsymbol{c}}$ in the matrix form, this time as a $2N \times 2N$ linear system, and solve it for foreground and background colors for 3 times, once for each color channel, using the preconditioned conjugate gradients method [Barrett et al., 1994].

5.4. Experimental Evaluation

We evaluate the proposed methods for matting, matte regularization, layer color estimation and green-screen keying with comparisons to the state-of-the-art of each application.

5.4.1. Matte Estimation

We quantitatively evaluate the proposed algorithm using the public alpha matting benchmark [Rhemann et al., 2009] in Table 5.1. At the time of initial publication, our method ranked in the first place according to the sum-of-absolute-differences (SAD) and mean-squared error (MSE) metrics. Our proof-of-concept implementation in Matlab requires on average 50 seconds to process a benchmark image.

Our performance in the test set by Xu et al. [2017] is shown in Table 5.2. This test set of 1000 images accompany a data-driven approach to matting.

Table 5.1.: *Our scores in the alpha matting benchmark [Rhemann et al., 2009] together with the top-performing published methods at the time of submission.* S, L *and* U *denote the three trimap types, small, large and user, included in the benchmark. Bold and blue numbers represent the best scores in the benchmark.*

	Average Rank				Troll			Doll			Donkey			Elephant			Plant			Pineapple			Plastic bag			Net		
	Overall	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*
Sum of Absolute Differences																												
Ours	2.7	3.3	2.3	2.6	**10.3**	**11.2**	12.5	5.6	7.3	7.3	3.8	4.1	3	1.4	2.3	2.0	5.9	7.1	**8.6**	3.6	5.7	4.6	18.3	19.3	**15.8**	20.2	22.2	22.3
DIM [Xu et al., 2017]	2.9	3.6	2.3	2.8	10.7	11.2	**11.0**	**4.8**	5.8	**5.6**	**2.8**	**2.9**	**2.9**	**1.1**	**1.1**	**2.0**	6.0	7.1	8.9	**2.7**	**3.2**	**3.9**	19.2	19.6	18.7	21.8	23.9	24.1
DCNN [Cho et al., 2019]	4.0	5.4	2.3	4.3	12.0	14.1	14.5	5.3	6.4	6.8	3.9	4.5	3.4	1.6	2.5	2.2	6.0	**6.9**	9.1	4.0	6.0	5.3	19.9	**19.2**	19.1	**19.4**	**20.0**	**21.2**
CSC [Feng et al., 2016]	11	14.4	7.4	11.3	13.6	15.6	14.5	6.2	7.5	8.1	4.6	4.8	4.2	1.8	2.7	2.5	5.5	7.3	9.7	4.6	7.6	6.9	23.7	23.0	21.0	26.3	27.2	25.2
Mean Squared Error																												
Ours	4.0	5.4	2.8	3.8	**0.3**	**0.4**	0.5	0.3	0.4	0.5	0.3	0.3	0.2	0.1	0.1	0.1	0.4	0.4	**0.6**	0.2	0.3	0.3	1.3	1.2	**0.8**	0.8	0.8	0.9
DCNN [Cho et al., 2019]	4.3	5.3	2.5	5.0	0.4	0.5	0.7	0.2	**0.3**	0.4	0.2	0.3	0.2	0.1	0.1	**0.1**	0.4	**0.4**	0.8	0.2	0.4	0.3	1.3	1.2	1.0	**0.7**	**0.7**	0.9
DIM [Xu et al., 2017]	4.6	3.5	4.0	6.3	0.4	0.4	**0.4**	**0.2**	0.3	**0.3**	**0.1**	**0.1**	**0.2**	**0**	**0**	0.2	0.5	0.6	1	**0.2**	**0.2**	0.4	**1.1**	**1.1**	1.1	0.8	0.9	1
LNSP [Chen et al., 2013b]	10.2	7.6	9.6	13.3	0.5	1.9	1.2	0.2	0.4	0.5	0.3	0.4	0.2	**0.0**	0.1	0.2	0.4	0.5	0.8	**0.2**	0.3	0.4	1.4	1.2	0.8	1.0	1.1	1.5

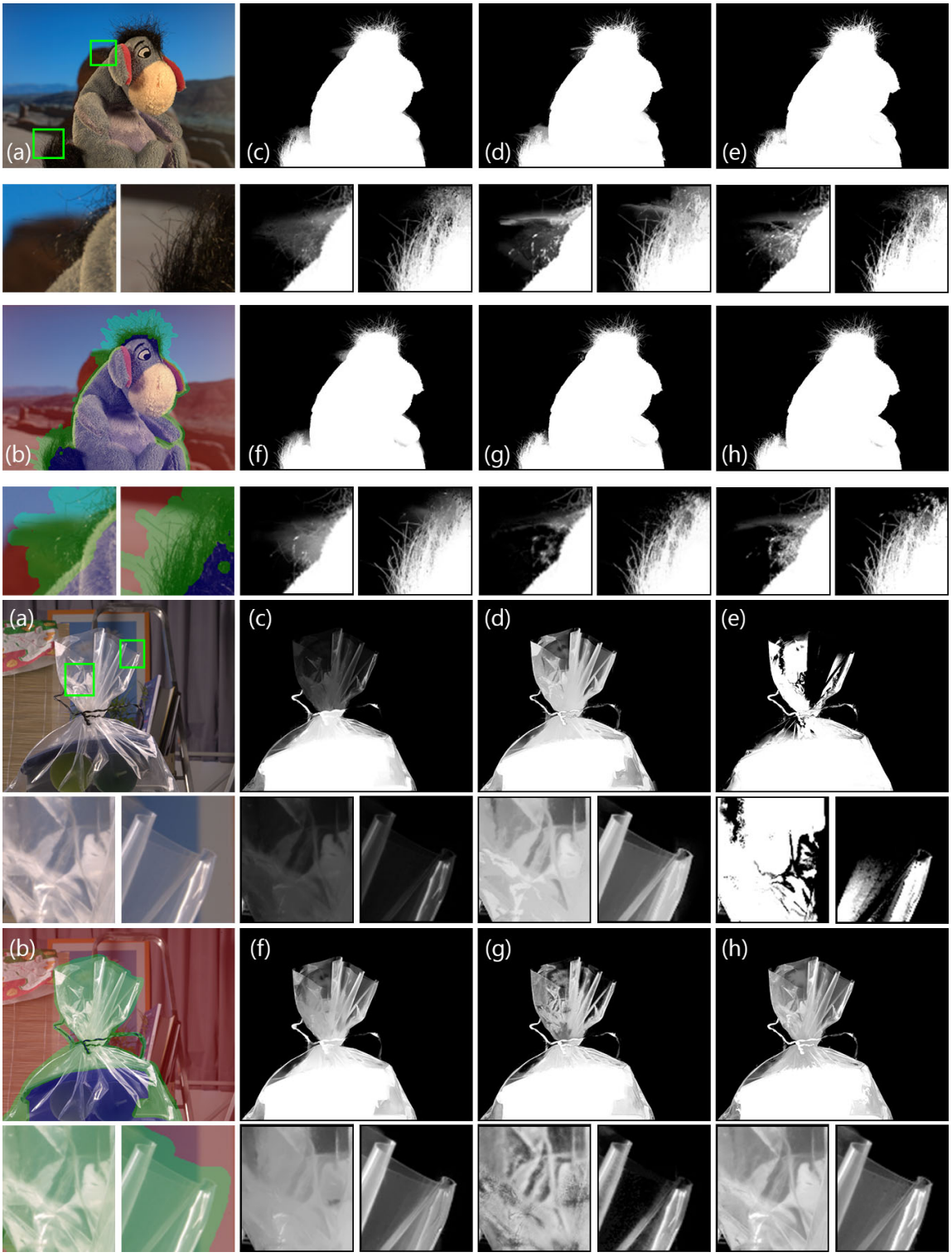

Figure 5.8.: *Several examples from the alpha matting benchmark [Rhemann et al., 2009] are shown (a) with trimaps overlayed onto the images (b). The mattes are computed by closed-form matting [Levin et al., 2008a] (c), KNN matting [Chen et al., 2013a] (d), manifold-preserving edit propagation [Chen et al., 2012] (e), LNSP matting [Chen et al., 2013b] (f), DCNN matting [Cho et al., 2019] (g) and the proposed method (h).*

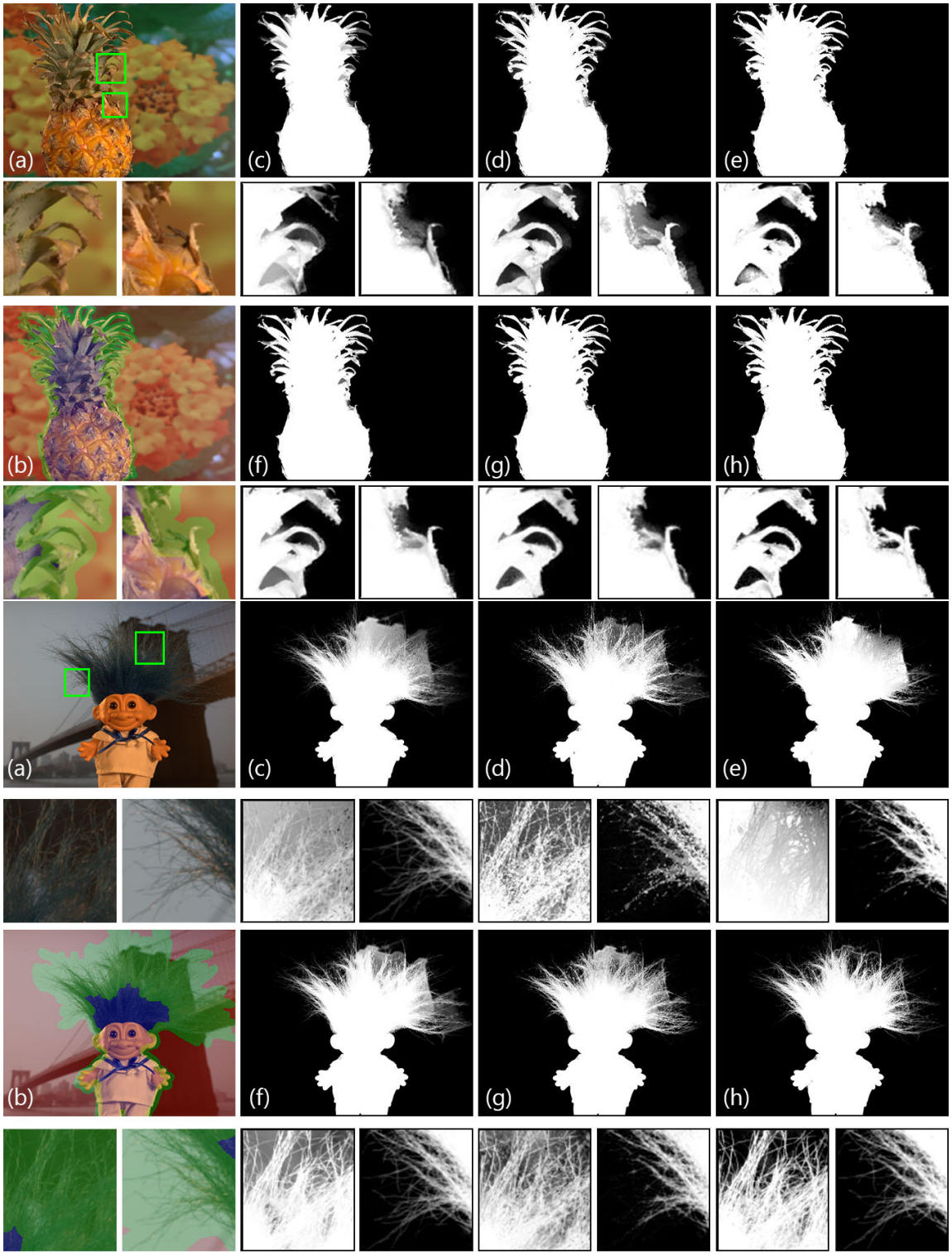

Figure 5.9.: *Several examples from the alpha matting benchmark [Rhemann et al., 2009] are shown (a) with trimaps overlayed onto the images (b). The mattes are computed by closed-form matting [Levin et al., 2008a] (c), KNN matting [Chen et al., 2013a] (d), manifold-preserving edit propagation [Chen et al., 2012] (e), LNSP matting [Chen et al., 2013b] (f), DCNN matting [Cho et al., 2019] (g) and the proposed method (h).*

Table 5.2.: *Matting performance on the test set of DIM [Xu et al., 2017].*

	SAD	MSE
DIM [Xu et al., 2017]	50.4	0.014
Ours	100.6	0.038
DCNN [Cho et al., 2019]	161.4	0.087
CF [Levin et al., 2008a]	168.1	0.091
KNN [Chen et al., 2013a]	175.4	0.103

Table 5.3.: *Average percentage performance change with changing parameters using 27 images and 2 trimaps from the benchmark.*

Param.	Def.	Val.	Perf.	Val.	Perf.	Val.	Perf.	Val.	Perf.
K_{CM}	20	10	1.07 %	15	0.44 %	25	-0.46 %	30	-0.62 %
$K_{\mathcal{K}-\mathcal{U}}$	7	1	-0.83 %	4	-0.41 %	10	0.12 %	13	0.22 %
$K_{\mathcal{U}-\mathcal{U}}$	5	1	-0.15 %	3	-0.1 %	7	0.08 %	9	0.11 %
$\sigma_{\mathcal{K}-\mathcal{U}}$	0.05	0.01	-6.44 %	0.025	-2.1 %	0.075	0.66 %	0.09	0.87 %
$\sigma_{\mathcal{U}-\mathcal{U}}$	0.01	0.001	-0.7 %	0.005	-0.1 %	0.02	-0.47 %	0.05	-3.12 %

One advantage of using a deep network for this problem, such as DIM [Xu et al., 2017], is that the network can infer the matte even when there is no foreground region defined in the trimap due to heavy transparency, and their test set includes several such examples. Affinity-based and sampling-based approaches, however, assume both known regions are present when they are modeling the color models of affinities. While this can be seen as a shortcoming, the images without well-defined regions inadvertently skew the scores in this dataset. We perform better than competing methods except for DIM in this dataset, and our scores improve to be 76.5 (SAD) and 0.021 (MSE) when the images that violate our assumptions are removed.

We also compare our results qualitatively with the closely related methods in Figure 5.8. We use the results that are available on the matting benchmark for all except manifold-preserving matting [Chen et al., 2012] which we implemented ourselves. Figure 5.8(c,d,e) show that using only one form of information flow is not effective in a number of scenarios such as wide unknown regions or holes in the foreground. The strategy DCNN matting [Cho et al., 2019] follows is using the results of closed-form and KNN matting directly rather than formulating a combined energy using their affinity definitions. When both methods fail, the resulting combination also suffers from the errors as it is apparent in the pineapple and troll examples. The neural network they propose also seems to produce mattes that appear slightly blurred.

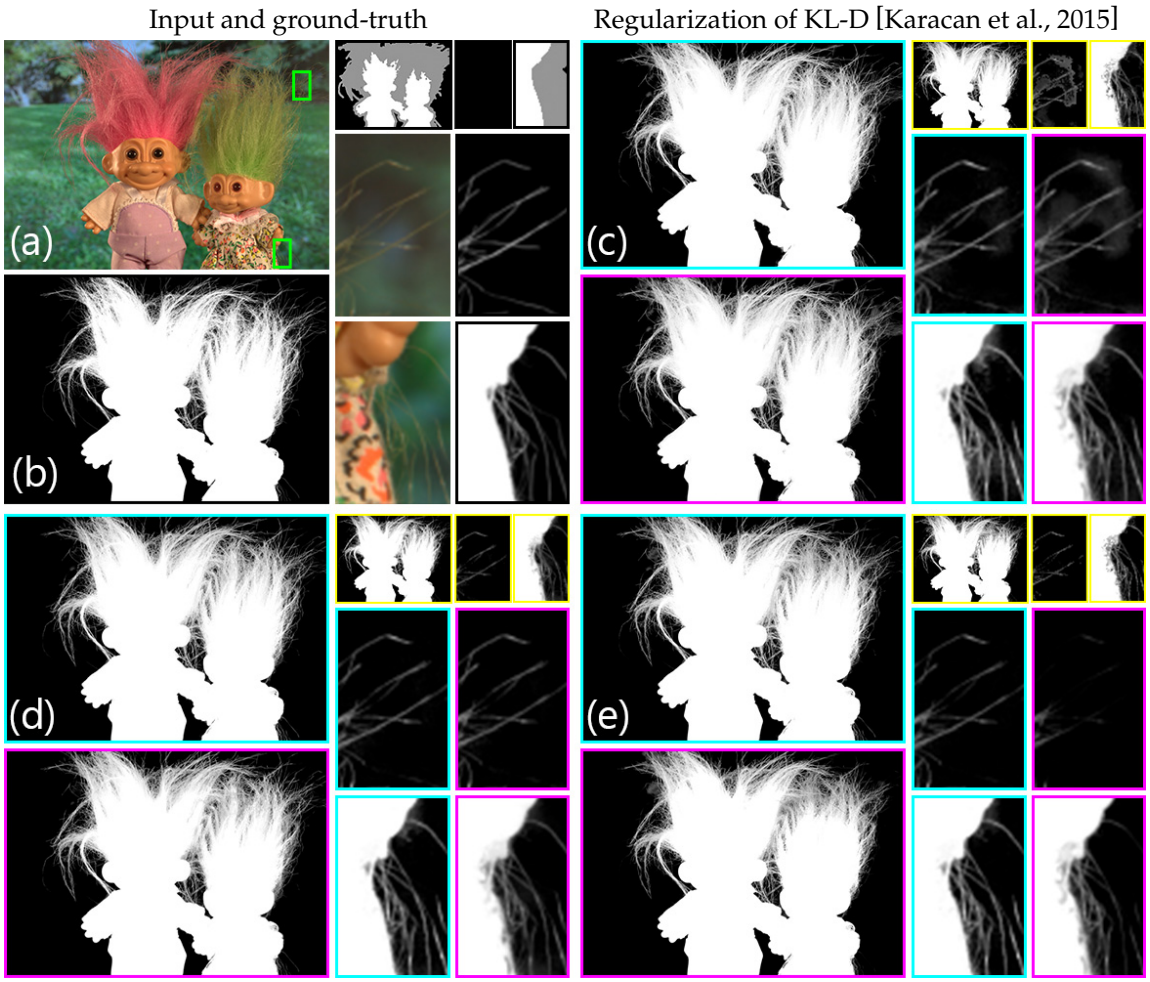

Figure 5.10.: *Matte regularization using the proposed method (cyan) or [Gastal and Oliveira, 2010] (magenta) for three sampling-based methods (yellow). Our method is able to preserve remote details while producing a clean matte (top inset) and preserve sharpness even around textured areas (bottom).*

LNSP matting [Chen et al., 2013b], on the other hand, has issues around regions with holes (pineapple example) or when the foreground and background colors are similar (donkey and troll examples). It can also oversmooth some regions if the true foreground colors are missing in the trimap (plastic bag example). Our method performs well in these challenging scenarios mostly because of the intra-unknown and unknown-to-known connections which results in a more robust linear system.

We evaluate the sensitivity of our method against different parameter values on the training dataset of the matting benchmark [Rhemann et al., 2009].

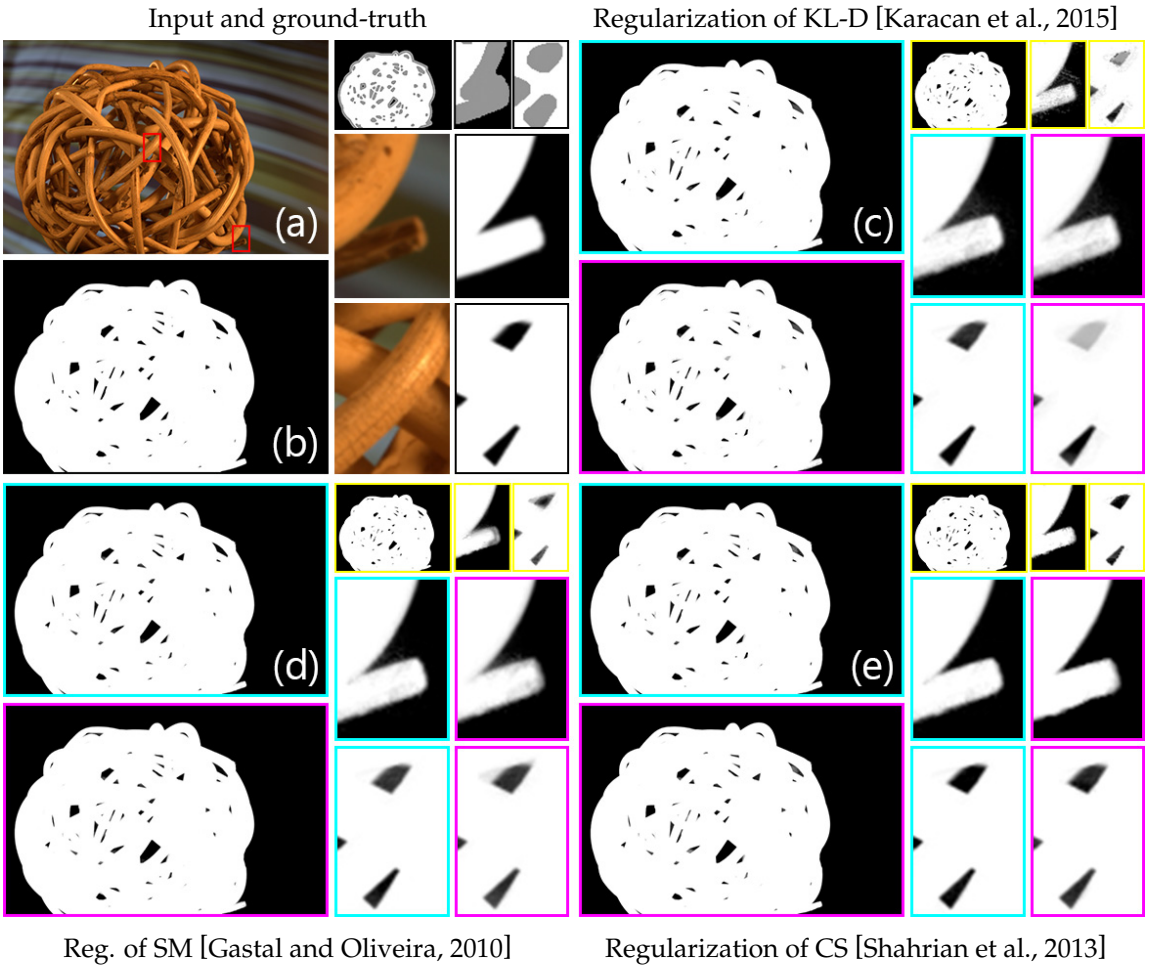

Figure 5.11.: *Matte regularization using the proposed method (cyan) or [Gastal and Oliveira, 2010] (magenta) for three sampling-based methods (yellow). Our method is able to correct errors in the initial mattes around remote holes in the foreground (bottom inset).*

Table 5.4.: *Performance improvement achieved when our matte regularization method replaces [Gastal and Oliveira, 2010] in the post-processing steps of 3 sampling-based methods. The training dataset in [Rhemann et al., 2009] was used for this experiment.*

	Sum of Absolute Differences			Mean Squared Error		
	Overall	S	L	Overall	S	L
KL-D [Karacan et al., 2015]	24.4 %	22.4 %	26.5 %	28.5 %	25.9 %	31.0 %
SM [Gastal and Oliveira, 2010]	6.0 %	3.7 %	8.4 %	13.6 %	8.5 %	18.8 %
CS [Shahrian et al., 2013]	4.9 %	10.0 %	-0.1 %	18.7 %	25.5 %	11.8 %

Table 5.3 shows that different values for the parameters generally have only a small effect on the performance on average.

5.4.2. Matte Regularization

We also compare the proposed post-processing method detailed in Section 5.2 with the state-of-the-art method by Gastal and Oliveira [2010] on the training dataset provided by Rhemann et al. [2009]. We computed the non-smooth alpha values and confidences using the publicly available source code for comprehensive sampling [Shahrian et al., 2013], KL-divergence sampling [Karacan et al., 2015] and shared matting [Gastal and Oliveira, 2010]. Table 5.4 shows the percentage improvement we achieve over Gastal and Oliveira [2010] for each algorithm using SAD and MSE as error measures. Figure 5.10 shows an example for regularizing all three sampling-based methods. As the information coming from alpha values and their confidences found by the sampling-based method is distributed more effectively by the proposed method, the challenging regions such as fine structures or holes detected by the sampling-based method are preserved when our method is used for post-processing.

5.4.3. Layer Color Estimation

We evaluate our layer color estimation method against the closed-form color estimation [Levin et al., 2008a] and KNN colors [Chen et al., 2013a], on the test set of deep image matting [Xu et al., 2017] using the ground-truth alphas as input. Closed-form colors only use a single local affinity to propagate the colors from the foreground, and this creates artifacts around holes in the foreground (Figure 5.12, top) or incorrect colors being propagated to nearby regions (bottom). KNN colors, on the other hand, uses only the similarity affinity and it typically generates flat-colored regions, which results in erroneous values especially around hair and fur. Our multi-affinity approach is able to correctly estimate the colors even in the isolated regions or intricate structures. These properties are also reflected in the quantitative comparison, as shown in Table 5.5.

Table 5.5.: *Layer color estimation performance on the test set of DIM [Xu et al., 2017].*

	SAD	MSE
Ours	3.8×10^3	6.9×10^{-4}
CF [Levin et al., 2008a]	4.3×10^3	9.2×10^{-4}
KNN [Chen et al., 2013a]	4.7×10^3	8.4×10^{-4}

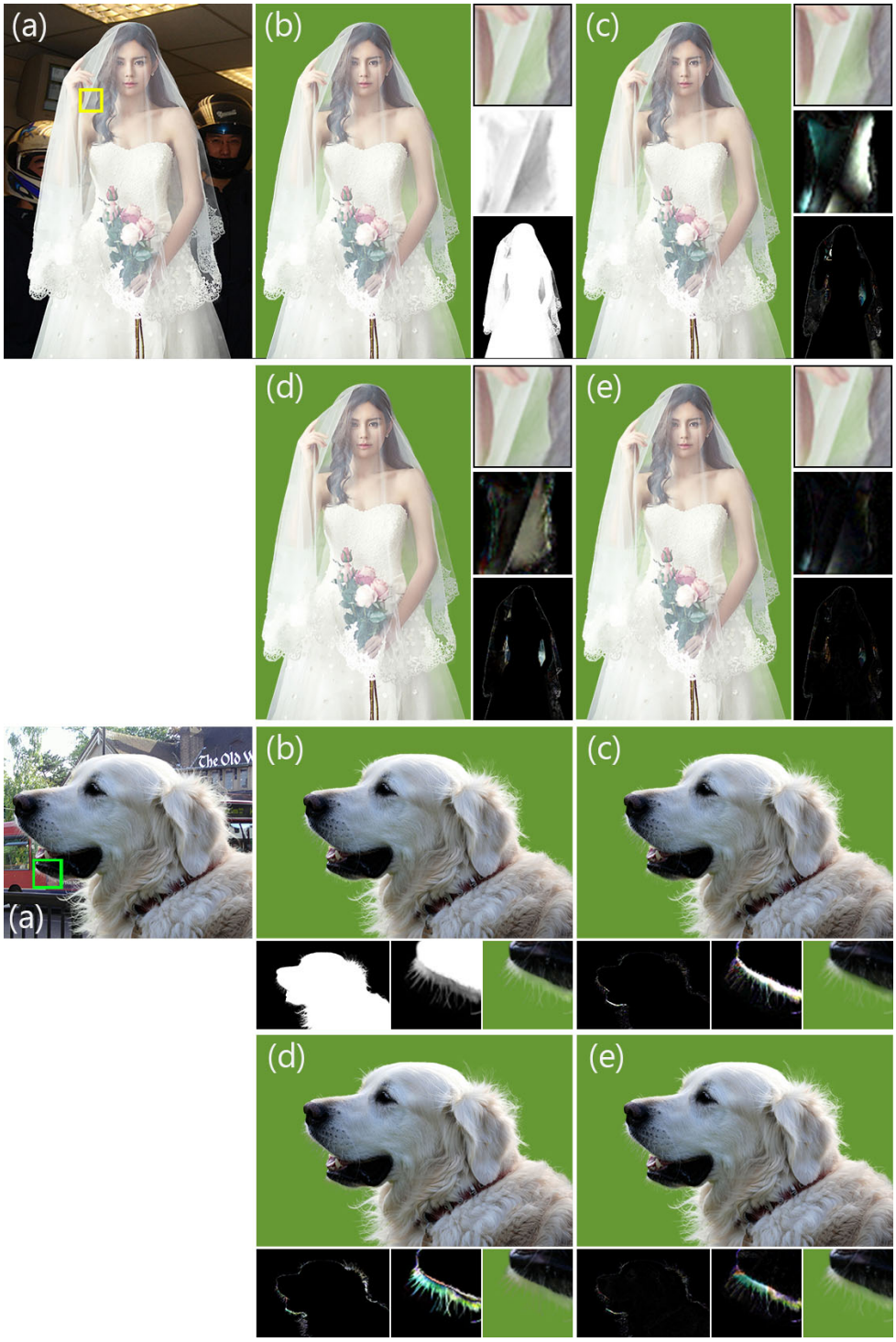

Figure 5.12.: *Color estimation results by closed-form matting [Levin et al., 2008a] (c), KNN matting [Chen et al., 2013a] (d) and the proposed method (e) together with the ground truth colors and matte (b). The bottom-right in each set shows per-pixel absolute difference between the estimation and ground truth multiplied by ten. See text for discussion.*

5.5. Spectral Analysis

The spectral clusters formed by Laplacians of affinity matrices can be effectively used to reveal characteristics of the constructed graph structure. For instance, Levin et al. [2008a] analyze the matting affinity by looking at eigenvectors corresponding to the smallest eigenvalues of the matting Laplacian. Spectral matting [Levin et al., 2008b] uses the eigenvectors together with a sparsity prior to create a set soft segments, or *alpha components*, that represent compact clusters of eigenvectors and add up to one for each pixel. The alpha components provide a more distilled and clear visualization to analyze the affinity matrix. In this section, we use the matting components computed using different subsets of information flows we defined for matte estimation to reveal the contribution of different flows at a higher level.

We compute the alpha components shown in Figure 5.13 using the public source code by Levin et al. [2008b]. We exclude the $\mathcal{K}$-to-$\mathcal{U}$ flow, which is only defined for the unknown regions as it requires explicitly defined known regions. The resulting Laplacian matrix does not give meaningful spectral clustering because of the pixels with missing connections. We overcome this issue for intra-$\mathcal{U}$ flow by defining it for the entire image instead of only the unknown region. In our matting formulation, we use the color-mixture flow to create the main source of information flow between close-by similarly-colored pixels. This approach creates densely connected graphs as both spatial and color distances are well accounted for in the neighborhood selection. We observed that spectral matting may fail to create as many components as requested (10 in our experiments) in some images, as many regions are heavily interconnected. Using the weighted average of neighboring colors for the flow creates soft transitions between regions.

The intra-$\mathcal{U}$ flow connects pixels that have similar colors, with very little emphasis on the spatial distance. This creates a color-based segmentation of the pixels, but as we compute the weights based on the feature distances, it is not typically able to create soft transitions between regions. Rather, it creates components with alpha values at zero or one, or flat alpha regions with alpha values near 0.5.

The local information flow, used as the only form of flow in the original spectral matting, creates locally connected components with soft transitions.

We observed a harmonious combination of positive aspects of these affinity matrices as they are put together to create our graph structure. This provides a neat confirmation of our findings in the evaluation of our algorithm. We analyze the characteristics of each flow more in detail through visual examples in the remainder of this section.

Figure 5.13.: *Selected matting components [Levin et al., 2008b] computed from Laplacian matrices constructed using different subsets of information flow. Two components are included in the bottom examples for* only CM *and* only local *cases as the included parts appeared in separate components.*

The top example in Figure 5.13 shows an input image with the matting components that include the green and the pink hair. Color-mixture affinities give components that demonstrate the color similarity and soft transitions, but they typically bleed out of the confined regions of specific colors due to the densely connected nature of the graph formed by corresponding neighborhoods. We clearly see the emphasis on color similarity for intra-$\mathcal{U}$ flow. While the color clusters are apparent, one can easily observe that unrelated pixels get mixed into the clusters especially around transition regions between other colors. We see a significant improvement already when these two flows are combined. When the local information flow is added, which gives spatially confined clusters of many colors when used individually, we see smooth clusters of homogeneous colors. The intricate transitions that were missed in the lack of the local flow are successfully captured when all three flows are included in the Laplacian definition.

The spatial connectivity versus color similarity characteristics are even more clearly observable in the bottom example of Figure 5.13. We see that bright and dark brown of the fur is clearly separated by intra-$\mathcal{U}$ flow in this example. In contrast, color-mixture and local flows separate the fur into three spatial clusters and the sweater into two separate clusters despite the uniform color. The combination, however, is able to successfully separate the dark and bright brown of the fur with smooth transitions.

The full Laplacian matrix we propose in this work blends the nonlocality of colors and spatial smoothness naturally. This is the key characteristic of the proposed matting method. When combined with $\mathcal{K}$-to-$\mathcal{U}$ flow which addresses remote regions and holes inside the foreground, the proposed algorithm is able to achieve high performance in a variety of images as analyzed in Section 5.4.

5.6. Sampling-Based Methods and $\mathcal{K}$-to-$\mathcal{U}$ Flow

The $\mathcal{K}$-to-$\mathcal{U}$ flow introduced in Section 5.1.2 connects every pixel in the unknown region directly to several pixels in both foreground and background. While the amount of flow from each neighbor is individually defined by the computed color-mixture weights, we simplify the formulation and increase the sparsity of our linear system using some algebraic manipulations. These manipulations, in the end, give us the weights $w_p^{\mathcal{F}}$ that go into the final energy formulation.

These weights, which show the connection of the unknown pixel to the foreground, are essentially an early estimation of the matte. This estimation is done by individually selecting a set of neighbors for each pixel and computing

Table 5.6.: *SAD scores of top sampling-based methods on the matting benchmark against the $\mathcal{K}$-to-$\mathcal{U}$ flow as a sampling based method, regularized by [Gastal and Oliveira, 2010]. Blue shows the best performance among the methods listed here for each image-trimap pair. Red marks the failure cases for the $\mathcal{K}$-to-$\mathcal{U}$ flow.*

	Troll			Doll			Donkey			Elephant			Plant			Pineapple			Plastic bag			Net		
	S	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*	*S*	*L*	*U*
CSC [Feng et al., 2016]	13.6	15.6	**14.5**	6.2	7.5	8.1	4.6	4.8	4.2	1.8	2.7	2.5	5.5	7.3	9.7	4.6	7.6	6.9	**23.7**	23.0	**21.0**	26.3	27.2	25.2
Sparse coding [Johnson et al., 2016]	12.6	20.5	14.8	5.7	**7.3**	**6.4**	4.5	5.3	**3.7**	1.4	3.3	2.3	6.3	7.9	11.1	4.2	8.3	6.4	28.7	31.3	27.1	23.6	25.1	27.3
KL-Div [Karacan et al., 2015]	11.6	17.5	14.7	**5.6**	8.5	8.0	4.9	5.3	3.7	1.5	3.5	2.1	5.8	8.3	14.1	5.6	9.3	8.0	24.6	27.7	28.9	**20.7**	**22.7**	**23.9**
$\mathcal{K}$-to-$\mathcal{U}$ inf. flow	12.0	**13.1**	14.6	7.5	9.1	8.9	**3.9**	**4.3**	3.8	**1.4**	**2.0**	**2.0**	**5.3**	**5.9**	**8.0**	**2.7**	**3.6**	**3.3**	**37.2**	**39.1**	**35.8**	**47.2**	**56.0**	**41.9**
Comp. Samp. [Shahrian et al., 2013]	**11.2**	18.5	14.8	6.5	9.5	8.9	4.5	4.9	4.1	1.7	3.1	2.3	5.4	9.8	13.4	5.5	11.5	7.4	23.9	**22.0**	22.8	23.8	28.0	28.1

an alpha based on the neighbor colors. While our approach is fundamentally defining affinities, it has parallels with sampling-based approaches in natural matting [Shahrian et al., 2013; Karacan et al., 2015; Feng et al., 2016; Johnson et al., 2016], which also select samples from foreground and background and estimates alpha values based on sample colors. We compute confidence values for $w_p^{\mathcal{F}}$ that depends on the similarity of colors of neighbors from the foreground and background. Sampling-based approaches also define confidence values for their initial estimation, typically defined by the *compositing error*, $\|\boldsymbol{c} - (\alpha \boldsymbol{f} - (1-\alpha)\boldsymbol{b})\|^2$.

Conceptually, there are several fundamental differences between our computation of $\mathcal{K}$-to-$\mathcal{U}$ flow and common strategy followed by sampling-based methods. The major difference is how the samples are collected. Sampling-based methods first determine a set of samples collected from known-alpha regions and do a selection for unknown pixels from this predetermined set using a set of heuristics. We, on the other hand, select neighbors for each unknown pixel individually via a k nearest neighbors search in the whole known region. Using the samples, state-of-the-art methods typically use the compositing equation to estimate the alpha value from only one sample pair (a notable exception is CSC matting [Feng et al., 2016]), while we use 14 samples in total to estimate the alpha by solving the overconstrained system using the method by Roweis and Saul [2000]. These differences also change the computation time. $\mathcal{K}$-to-$\mathcal{U}$ flow can be computed in several seconds, while sampling-based algorithms typically take several minutes per image due to sampling and sample pair selection steps.

In order to compare the performance of $\mathcal{K}$-to-$\mathcal{U}$ flow *as* a sampling-based method in a neutral setting, in this experiment, we post-process $w_p^{\mathcal{F}}$ and our confidence values using the common regularization step [Gastal and Oliveira, 2010] utilized by top-performing sampling-based methods in the benchmark. The quantitative results can be seen in Table 5.6.

As discussed in Section 5.1.2, $\mathcal{K}$-to-$\mathcal{U}$ flow fails in the case of a highly-transparent matte (net and plastic bag examples). This is due to the failure to find representative neighbors using the k nearest neighbor search. Sampling-based methods are more successful in these cases due to their use of compositing error in the sample selection. However, in the other examples, $\mathcal{K}$-to-$\mathcal{U}$ flow appears as the top-performing method among the sampling-based methods in 12 of 18 image-trimap pairs and gives comparable errors in the rest.

The performance of our affinity-inspired approach against the state-of-the-art [Shahrian et al., 2013; Karacan et al., 2015; Feng et al., 2016; Johnson et al., 2016] gives us some pointers for a next-generation sampling-based matting

Figure 5.14.: *Our method fails gracefully in the case of sparse trimaps.*

method. While one can argue that the sampling algorithms have reached enough sophistication, selection of a single pair of samples for each unknown pixel seems to be a limiting factor. Methods that address the successful and efficient selection of *many* samples for each unknown pixel will be more likely to surpass state-of-the-art performance. Furthermore, determining the alpha values using more robust weight estimation formulations such as (5.1) instead of the more simple compositing equation (5.23) will likely improve the result quality.

5.7. Limitations

As discussed in corresponding sections, the $\mathcal{K}$-to-$\mathcal{U}$ flow does not perform well in the case of highly-transparent mattes. We solve this issue via a simple classifier to detect highly-transparent mattes before alpha estimation. However, this does not solve the issue for foreground images that partially have transparent regions. For such cases, a locally changing set of parameters could be the solution.

The proposed matte estimation algorithm assumes dense trimaps as input. In the case of sparse trimaps, generally referred as scribble input, our method may fail to achieve its original performance, as seen in Figure 5.14. This performance drop is mainly due to the $\mathcal{K}$-to-$\mathcal{U}$ flow, which fails to find good neighbors in limited known regions, and intra-$\mathcal{U}$ flow which propagates alpha information based solely on color to spatially far away pixels inside the unknown region.

CHAPTER 6

Semantic Soft Segmentation

We seek to automatically generate a *soft segmentation* of the input image, that is, a decomposition into layers that represent *the objects* in the scene including transparency and soft transitions when they exist. Each pixel of each layer is augmented with an opacity value $\alpha \in [0,1]$ with $\alpha = 0$ meaning fully transparent, $\alpha = 1$ fully opaque, and in-between values indicating the degree of partial opacity. Similar to our study of soft color segmentation in Chapter 4, we use an additive image formation model:

$$(R,G,B)_{\text{input}} = \sum_i \alpha_i (R,G,B)_i \tag{6.1a}$$

$$\sum_i \alpha_i = 1, \tag{6.1b}$$

i.e., we express the input RGB pixels as the sum of the pixels in each layer i weighted by its corresponding α value. We also constrain the α values to sum up to 1 at each pixel, representing a fully opaque input image.

The core of our approach uses the same formalism as spectral matting in formulating the soft segmentation task as an eigenvector estimation problem [Levin et al., 2008b]. The core component of this approach is the creation of a *Laplacian* matrix L that represents how likely each pair of pixels in the image is to belong to the same segment. While spectral matting builds this matrix using only low-level local color distributions, we describe how to augment this approach with nonlocal cues and high-level semantic information. The original approach also describes how to create the layers from the eigenvectors of L using sparsification.

Forming the soft segments only from a limited set of eigenvectors usually results in matte sparsity issues around the object boundaries. We propose an

Figure 6.1.: *We propose a method that can generate soft segments, i.e. layers that represent the semantically meaningful regions as well as the soft transitions between them, automatically by fusing high-level and low-level image features in a single graph structure. The semantic soft segments, visualized by assigning each segment a solid color, can be used as masks for targeted image editing tasks, or selected layers can be used for compositing after layer color estimation.*

unconstrained matte sparsification formulation to address this issue, where the initial soft segment proposals are fed into a linear system that enforces matte sparsity at the pixel-level. Once sparsified using this method, the semantic soft segments can be effectively used in targeted image editing and compositing tasks. Our matte sparsification algorithm uses the same graph structure we use for the initial spectral segmentation, and can be used to post-process any matte that suffers from sparsity issues. Figure 6.1 shows several semantic soft segmentation examples, and Figure 6.2 shows an overview of our approach.

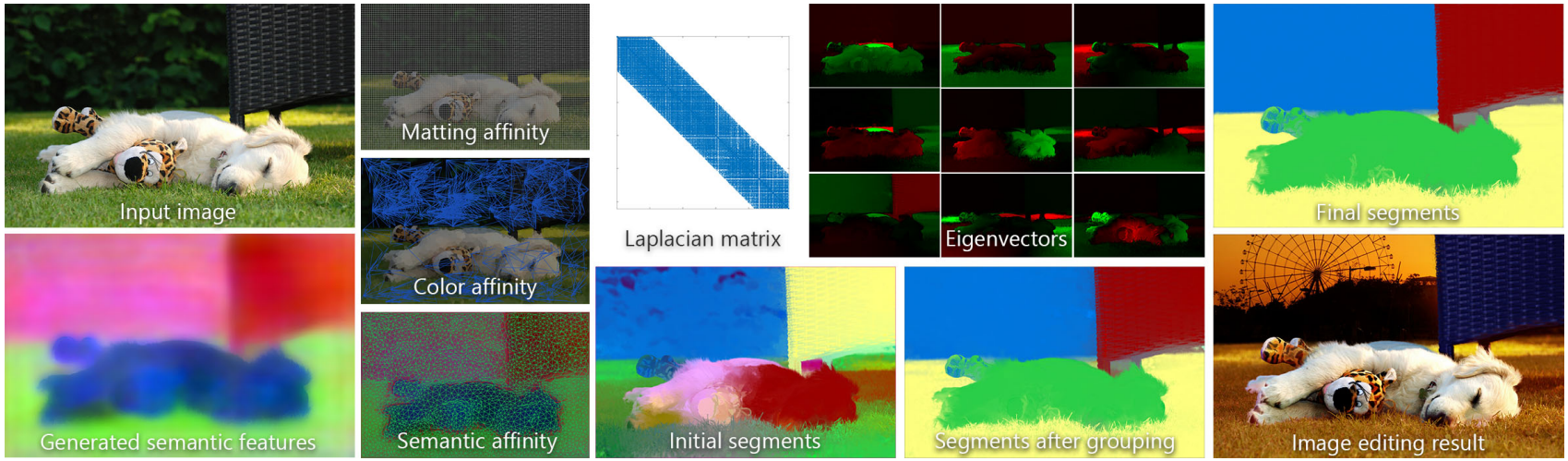

Figure 6.2.: *For an input image, we generate per-pixel hyperdimensional semantic feature vectors and define a graph using the texture and semantic information. The graph is constructed such that the corresponding Laplacian matrix and its eigenvectors reveal the semantic objects and the soft transitions between them. We use the eigenvectors to create a set of preliminary soft segments and combine them to get semantically meaningful segments. Finally, we refine the soft segments so that they can be used for targeted image editing tasks. Image from [Lin et al., 2014], background in the editing result by Flickr user* rumpleteaser.

6.1. Background on Spectral Matting

Our approach builds upon the work of Levin et al. [2008a; 2008b]. They first introduced the matting Laplacian that uses local color distributions to define a matrix L that captures the affinity between each pair of pixels in a local patch, typically 5×5 pixels. Using this matrix, they minimize the quadratic functional $\boldsymbol{\alpha}^T L \boldsymbol{\alpha}$ subject to user-provided constraints, with $\boldsymbol{\alpha}$ denoting a vector made of all the α values for a layer. This formulation shows that the eigenvectors associated to small eigenvalues of L play an important role in the creation of high-quality mattes. Motivated by this observation, their subsequent work on spectral matting used the eigenvectors of L to build a soft segmentation [Levin et al., 2008b]. Each soft segment is a linear combination of the K eigenvectors corresponding to the smallest eigenvalues of L that maximizes *matting sparsity*, i.e., minimizes the occurrence of partial opacity. The segments are created by minimizing an energy function that favors $\alpha = 0$ and $\alpha = 1$:

$$\arg\min_{\{\boldsymbol{y}_i\}} \sum_{i,p} |\alpha_{ip}|^\gamma + |1 - \alpha_{ip}|^\gamma \quad \text{with: } \boldsymbol{\alpha}_i = \mathsf{E}\boldsymbol{y}_i \tag{6.2a}$$

$$\text{subject to: } \sum_i \alpha_{ip} = 1, \tag{6.2b}$$

where α_{ip} is the α value of p^{th} pixel of the i^{th} segment, E is a matrix containing the K eigenvectors of L with smallest eigenvalues, $\boldsymbol{y}_i$ is the linear weights on the eigenvectors that define the soft segments, and $\gamma < 1$ is a parameter that controls the strength of the sparsity prior.

While spectral matting generates satisfying results when the image contains a single well-identified object with distinct colors, it struggles with more complex objects and scenes. Being based solely on the matting Laplacian that considers only low-level statistics of small patches, it is limited in its ability to identify objects. In our work, we extend this approach to fuse semantic features in the same Laplacian formulation and capture higher-level concepts like scene objects and to have a broader view of the image data.

6.1.1. Affinity and Laplacian Matrices

Levin et al. [2008a] formulate their approach as a least-squares optimization problem that directly leads to a Laplacian matrix. An alternative approach is to express the *affinity* between pairs of pixels, following our work on natural matting in Chapter 5. Pairs with a positive affinity are more likely to have similar values, zero-affinity pairs are independent, and pairs with a negative affinity are likely to have different values. In this work, we will use the

affinity approach and build the corresponding normalized Laplacian matrix using the well-known formula:

$$L = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}, \tag{6.3}$$

where W is a square matrix containing the affinity between all pairs of pixels and D is the corresponding degree matrix, i.e. a diagonal matrix with elements $W\mathbf{1}$, $\mathbf{1}$ being a row vector of ones. As noted by Levin et al., L may not always be a *true* graph Laplacian due to the presence negative affinities, but nonetheless shares similar properties such as being positive semidefinite.

6.2. Spectral Segmentation with Low- and High-Level Features

In addition to the matting affinity, we will introduce two additional affinity definitions, one based on colors and the other based on high-level semantic features, to build a graph structure that reveals the object boundaries readily in its eigenvectors.

6.2.1. Nonlocal Color Affinity

We define an additional low-level affinity term that represents color-based longer-range interactions. A naive approach would be to use larger patches in the definition of the matting Laplacian. However, this option quickly becomes impractical because it renders the Laplacian matrix denser. Another option is to sample pixels from a nonlocal neighborhood to insert connection while preserving some sparsity in the matrix. We have shown good results for medium-range interaction with such sampling with the intra-$\mathcal{U}$ flow in Chapter 5. However, this strategy faces a trade-off between sparsity and robustness: fewer samples may miss important image features and more samples make the computation less tractable.

We propose a guided sampling based on an oversegmentation of the image. We generate 2500 superpixels using SLIC [Achanta et al., 2012] and estimate the affinity between each superpixel and all the superpixels within a radius that corresponds to 20% of the image size. The advantage of this approach is that each feature large enough to be a superpixel is represented, sparsity remains high because we use a single sample per superpixel, and it links possibly disconnected regions by using a large radius, e.g. when the background is seen through a hole in an object. Formally, we define the color affinity $w^C_{s,t}$ between the centroids of two superpixels s and t separated by a distance less than 20% of the image size as:

$$w^C_{s,t} = \left(\operatorname{erf}\left(a_c\ \left(b_c - \|\boldsymbol{c}_s - \boldsymbol{c}_t\|\right)\right) + 1\right) / \ 2, \tag{6.4}$$

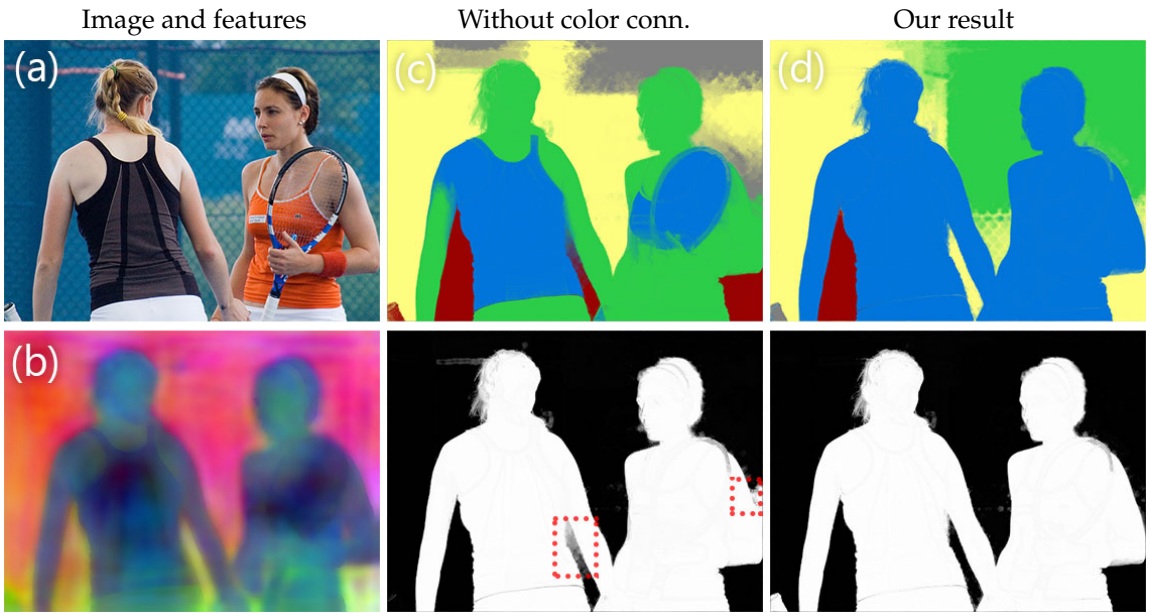

Figure 6.3.: *The color-based nonlocal affinity we include helps the decomposition recover isolated regions such as disconnected background regions or long extensions, as the highlights point out. Image from [Lin et al., 2014].*

where $\boldsymbol{c}_s$ and $\boldsymbol{c}_t$ are the average colors of the superpixels of s and t that lies in $[0, 1]$, erf is the Gauss error function, and a_c and b_c are parameters controlling how quickly the affinity degrades and the threshold where it becomes zero. erf takes values in $[-1, 1]$ and its use here is mainly motivated by its sigmoidal shape. We use $a_c = 50$ and $b_c = 0.05$ in all our results. This affinity essentially makes sure the regions with very similar colors stay connected in challenging scene structures, and its effect is demonstrated in Figure 6.3.

6.2.2. High-Level Semantic Affinity

While the nonlocal color affinity adds long-range interactions to the segmentation process, it remains a low-level feature. Our experiments show that, without additional information, the segmentation still often merges image regions of similar color that belong to different objects. To create segments that are confined in semantically similar regions, we add a *semantic affinity* term, that is, a term that encourages the grouping of pixels that belong to the same scene object and discourages that of pixels from different objects. We build upon prior work in the domain of object recognition to compute a feature vector at each pixel that correlates with the underlying object. We compute the feature vectors via a neural network, as described in Section 6.4. The feature vectors are generated such that for two pixels p and q that belong to

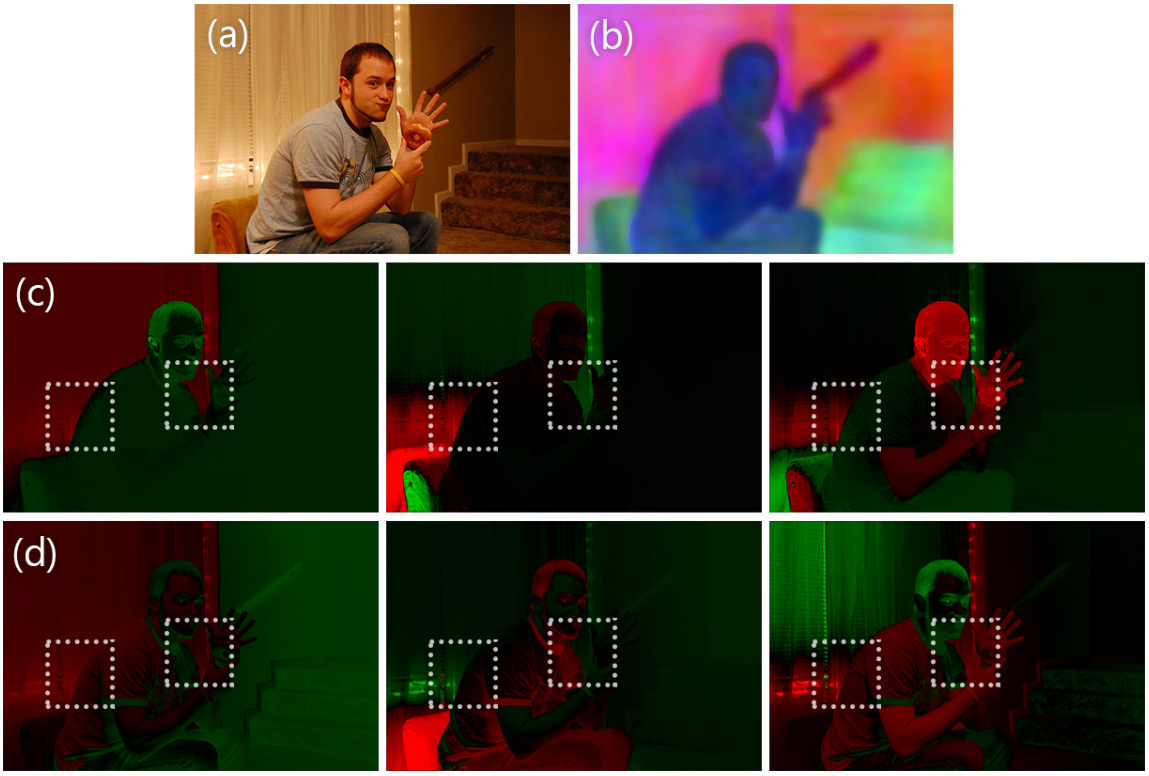

Figure 6.4.: *The image (a) and semantic features (b) are shown with several eigenvectors corresponding to some of the smallest eigenvalues[1] of the proposed Laplacian matrix (c, top row) and the matting Laplacian as used in spectral matting [Levin et al., 2008b] (d, bottom row). Green represents the positive values of an eigenvector while red shows negative. Our Laplacian matrix strongly reveals the semantic cuts in the eigenvectors while the matting Laplacian eigenvectors extend beyond the semantic edges, as the highlighted areas show. Image from [Lin et al., 2014].*

the same object $\boldsymbol{f}_p$ and $\boldsymbol{f}_q$ are similar, i.e. $\|\boldsymbol{f}_p - \boldsymbol{f}_q\| \equiv 0$, and for a third pixel r in a different semantic region, $\boldsymbol{f}_r$ is far away, i.e. $\|\boldsymbol{f}_p - \boldsymbol{f}_q\| \ll \|\boldsymbol{f}_p - \boldsymbol{f}_r\|$.

We define the semantic affinity also over superpixels. In addition to increasing the sparsity of the linear system, the use of superpixels also decrease the negative effect of the unreliable feature vectors in transition regions, as apparent from their blurred appearance in Figure 6.5. The superpixel edges are not directly used in the linear system, the connections in the graph are between superpixel centroids. This information from the centroids then spreads to nearby pixels while respecting the image edges with the matting affinity term. With these vectors and the same oversegmentation in the previous section (§ 6.2.1), for each superpixel s, we associate its average feature vector $\tilde{\boldsymbol{f}}_s$ to

[1] In fact, the eigenvector corresponding to the smallest eigenvalue is not shown here as it is a constant vector for both matrices.

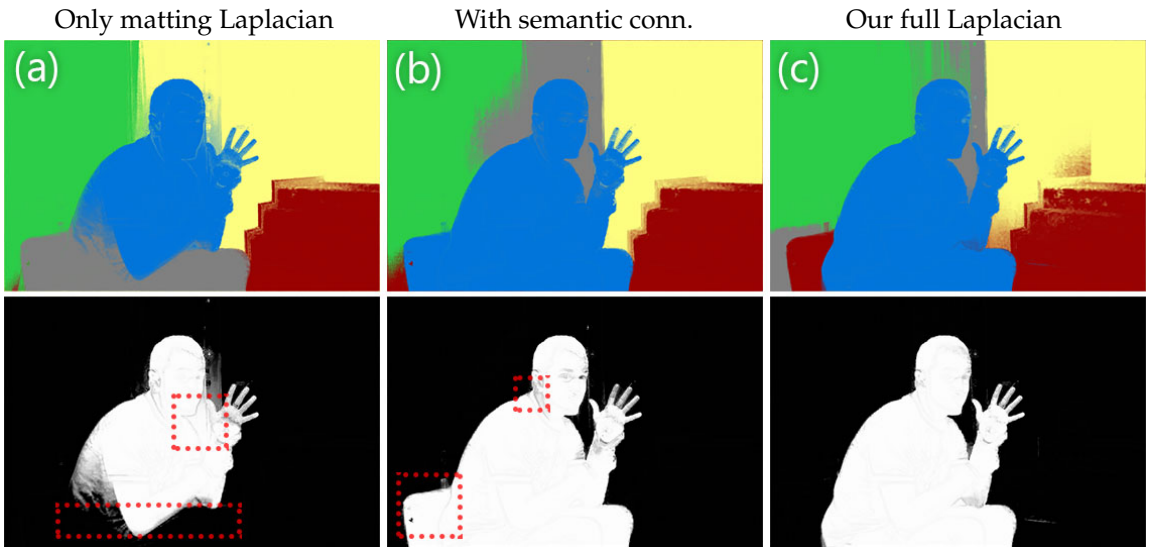

Figure 6.5.: *The results of our entire pipeline using only the matting Laplacian (a), matting and semantic Laplacians (b) and the two together with the sparse color connections (c), for the image shown in Figure 6.4. The top row shows a distinct color for each produced soft segment, and the bottom row shows the extracted matte corresponding to the person. Due to the eigenvectors that are unable to represent the semantic cut between the person and the background, using only matting Laplacian results in the person soft segment including large portions of the background, as highlighted. Adding the sparse color connections provides a cleaner foreground matte.*

its centroid $_s$. We use these vectors to define an affinity term between each adjacent superpixels s and t:

$$w^S_{s,t} = \text{erf}\left(a_s\left(b_s - \|\tilde{f}_s - \tilde{f}_t\|\right)\right), \tag{6.5}$$

with a_s and b_s parameters controlling the steepness of the affinity function and when it becomes negative. We discuss how to set them in Section 6.4. Defining negative affinities help the graph disconnect different objects while the positive values connect regions that belong to the same object.

Unlike the color affinity, the semantic affinity only relates nearby superpixels to favor the creation of connected objects. This choice of a *nonlocal* color affinity together with a local semantic affinity allows creating layers that can cover spatially disconnected regions of the same semantically coherent region. This often applies to elements like greenery and sky that often appear in the background, which makes them likely to be split into several disconnected components due to occlusions. As a result of including the local semantic affinity, the eigenvectors of L reveal object boundaries as demonstrated in Figure 6.5 and 6.4.

6.2.3. Creating the Layers

We create the layers by using the affinities described earlier in this section to form a Laplacian matrix L. We extract the eigenvectors from this matrix and use a two-step sparsification process to create the layers from these eigenvectors.

Forming the Laplacian matrix

We form a Laplacian matrix L by adding the affinity matrices together and using Equation 6.3:

$$L = D^{-\frac{1}{2}}\big(D - (W_L + \sigma_S W_S + \sigma_C W_C)\big)D^{-\frac{1}{2}} \tag{6.6}$$

where W_L is the matrix containing the matting affinities, W_C the matrix containing the nonlocal color affinities (§ 6.2.1), W_S the matrix with the semantic affinities (§ 6.2.2), and σ_S and σ_C parameters controlling the influence of each term, both set to be 0.01.

Constrained sparsification

We extract the eigenvectors corresponding to the 100 smallest eigenvalues of L. We form an intermediate set of layers using the optimization procedure by Levin et al. [2008b] on Eq. 6.2 with $\gamma = 0.8$. Unlike spectral matting that uses k-means clustering on the eigenvectors to initialize the optimization, we use k-means clustering on the pixels represented by their feature vectors $\boldsymbol{f}$. This initial guess is more consistent with the scene semantics and yields a better soft segmentation. We generate 40 layers with this approach and in practice, several of them are all zeros, leaving 15 to 25 nontrivial layers. We further reduce the number of layers by running the k-means algorithm with $k = 5$ on these nontrivial layers represented by their average feature vector. This approach works better than trying to directly sparsify the 100 eigenvectors into 5 layers, because such drastic reduction makes the problem overly constrained and does not produce good-enough results, especially in terms of matte sparsity. The initially estimated soft segments before and after grouping are shown in Figure 6.6. We have set the number of segments to 5 without loss of generalization; while this number could be set by the user depending on the scene structure, we have observed that it is a reasonable number for most images. Because these 5 layers are constrained to lie within the subspace of a limited number of eigenvectors, the achieved sparsity is suboptimal, leaving many semi-transparent regions in the layers, which is

Figure 6.6.: *The input image and computed semantic features are shown with the initially estimated soft segments with many layers (middle) and the intermediate soft segments after grouping (right). The soft segments are visualized by assigning each segment a solid color. Note that these results are refined further with relaxed sparsification. Images from [Lin et al., 2014].*

unlikely in common scenes. Next, we introduced a relaxed version of the sparsity procedure to address this issue.

6.3. Relaxed Sparsification of Soft Segments

To improve the sparsity of the layers, we relax the constraint that they are a linear combination of the eigenvectors. Instead of working with the coefficients $\boldsymbol{y}_i$ of the linear combination (Eq. 6.2), in this step, each individual α value is an unknown. We define an energy function that promotes matte sparsity on the pixel-level while respecting the initial soft segment estimates from the constrained sparsification and the image structure. We now define our energy term by term.

The first term relaxes the subspace constraint and only ensures that the

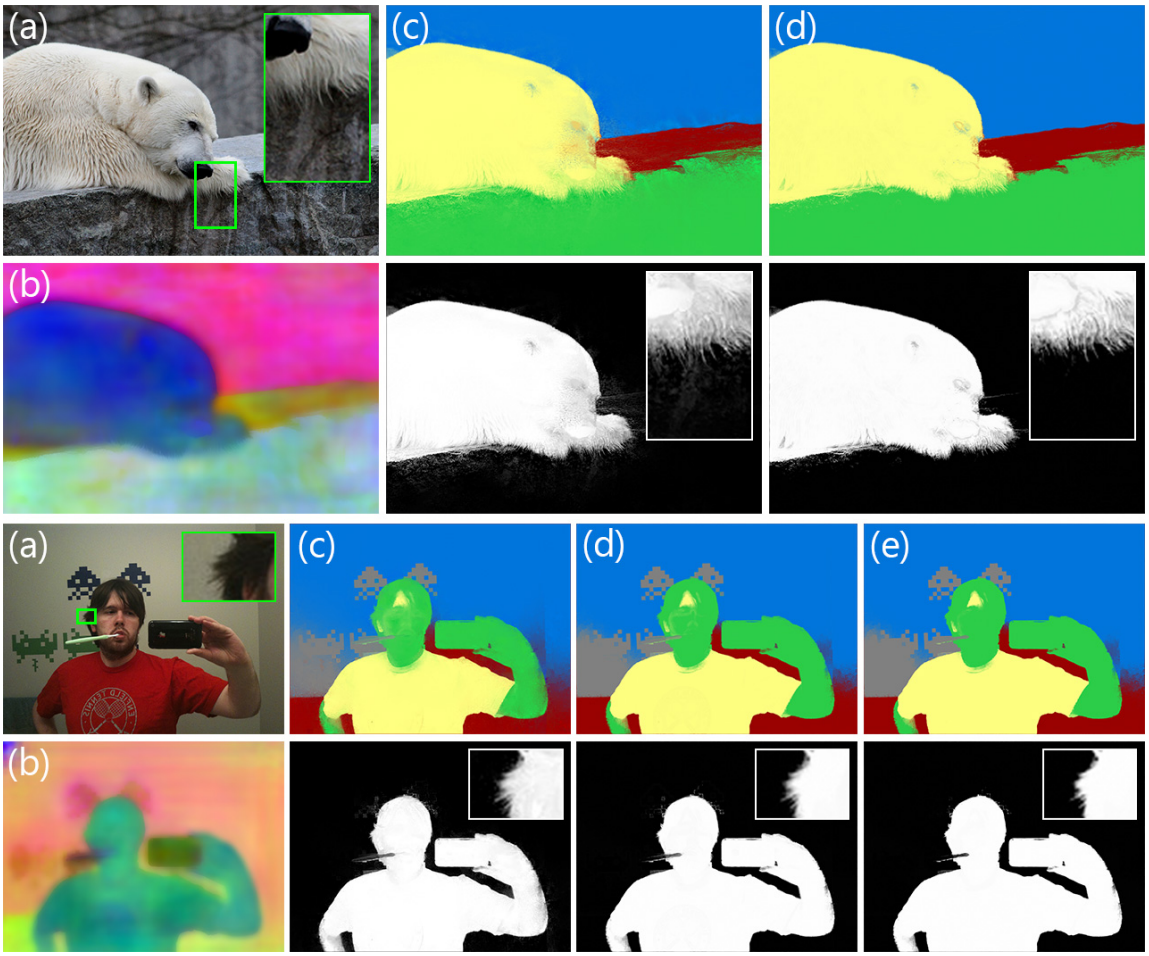

Figure 6.7.: *The images (a) and features (b) are shown with the results before pixel-level sparsification (c) and after (d). Color-coded segments are shown with a single alpha channel that corresponds to the foreground objects. This final step cleans spurious alpha values that occur due to the limited expressional power of the eigenvectors while preserving the soft transitions. The bottom example also features a sparsification result that uses a constant* 0.9 *as sparsity parameter* γ *(e), while we use spatially-varying* γ_p *which relaxes the sparsity constraint in transition regions. The effect of this can be seen in the inset, as our result (d) preserves the soft transitions around the hair while a constant parameter (e) results in an overly sparse result. Images from [Lin et al., 2014].*

generated layers remain close to the layers $\hat{\boldsymbol{\alpha}}$ created with the constrained sparsification procedure:

$$E_F = \sum_{ip} \left(\alpha_{ip} - \hat{\alpha}_{ip}\right)^2. \tag{6.7}$$

We also relax the sum-to-one requirement (Eq. 6.1b) to be integrated into the linear system as a soft constraint:

$$E_C = \sum_p \Big(1 - \sum_i \alpha_{ip}\Big)^2, \tag{6.8}$$

where α_{ip} is the α value of the p^{th} pixel in the i^{th} layer. The next term is the energy defined by the Laplacian L defining the spatial propagation of information defined in Eq.6.6:

$$E_L = \sum_i \boldsymbol{\alpha}_i^{\mathsf{T}} L \boldsymbol{\alpha}_i. \tag{6.9}$$

Finally, we formulate a sparsity term that adapts to the image content. Intuitively, partial opacities come from color transitions in the image because in many cases, it corresponds to a transition between two scene elements, e.g., the fuzzy transition between a teddy bear and the background. We use this observation to build a spatially varying sparsity energy:

$$E_S = \sum_{i,p} |\alpha_{ip}|^{\tilde{\gamma}_p} + |1 - \alpha_{ip}|^{\tilde{\gamma}_p} \tag{6.10a}$$

$$\text{with: } \tilde{\gamma}_p = \min(0.9 + \|\nabla \boldsymbol{c}_p\|, 1), \tag{6.10b}$$

where $\nabla \boldsymbol{c}_p$ is the color gradient in the image at pixel p computed using the separable kernels of Farid and Simoncelli [2004]. We design this term such that when $\tilde{\gamma}_p = 1$ on image regions where the gradient is large enough, the energy profile is flat for $\alpha_{ip} \in [0:1]$, i.e. the energy only acts as a penalty on values outside the valid range and lets α_{ip} take any value between 0 and 1. In comparison, in uniform regions where $\nabla \boldsymbol{c}_p \approx 0$, it encourages α_{ip} to be 0 or 1. These two effects combined favor a higher level of sparsity together with the softness of the opacity transitions. The effect of our spatially varying sparsity energy on preserving accurate soft transitions can be seen in Figure 6.7 (c,d).

Putting these terms together, we get the energy function

$$E = E_L + E_S + E_F + \lambda E_C. \tag{6.11}$$

A unit weight for each term works well except for the sum-to-one term E_C that represents the soft constraint with a higher weight $\lambda = 100$. Without the

sparsity term E_S, E would be a standard least-squares energy function that can be minimized by solving a linear system. To handle E_S, we resort to an iterative reweighted least-squares solver that estimates a solution by solving a series of linear systems. We describe the detail of this approach in the rest of this section.

We name $N_i = 5$ the number of layers, N_p the number of pixels, $\mathtt{a}$ the vector made of all $\boldsymbol{\alpha}_i$'s and $\hat{\mathtt{a}}$ the vector made of all $\hat{\boldsymbol{\alpha}}_i$'s. The dimensionality of $\mathtt{a}$ and $\hat{\mathtt{a}}$ is $N_{ip} = N_i N_p$. For clarity, we also introduce the $N_{ip} \times N_{ip}$ identity matrix $\mathcal{I}$. With this notation, we rewrite E_F (Eq. 6.7) in matrix form:

$$E_F = (\mathtt{a} - \hat{\mathtt{a}})^\mathsf{T} \mathcal{I} (\mathtt{a} - \hat{\mathtt{a}}) \tag{6.12}$$

We included the redundant $\mathcal{I}$ in this equation for a clearer transition when deriving Eq. 6.17. For rewriting E_C (Eq. 6.8), we introduce the $N_i \times N_{ip}$ matrix $\mathcal{C}$ made by concatenating N_i identity matrices horizontally, the vector $\mathbf{1}_i$ made of N_i ones, and the vector $\mathbf{1}_{ip}$ made of N_{ip} ones:

$$\begin{aligned} E_C &= (\mathbf{1}_i - \mathcal{C}\mathtt{a})^2 = \mathtt{a}^\mathsf{T}\mathcal{C}^\mathsf{T}\mathcal{C}\mathtt{a} - \mathtt{a}^\mathsf{T}\mathcal{C}^\mathsf{T}\mathbf{1}_i - \mathbf{1}_i^\mathsf{T}\mathcal{C}\mathtt{a} + \mathbf{1}_i^\mathsf{T}\mathbf{1}_i && \text{(6.13a)} \\ &= \mathtt{a}^\mathsf{T}\mathcal{C}^\mathsf{T}\mathcal{C}\mathtt{a} - 2\mathtt{a}^\mathsf{T}\mathbf{1}_{ip} + N_i, && \text{(6.13b)} \end{aligned}$$

where we used $\mathtt{a}^\mathsf{T}\mathcal{C}^\mathsf{T}\mathbf{1}_i = \mathbf{1}_i^\mathsf{T}\mathcal{C}\mathtt{a}$, $\mathcal{C}^\mathsf{T}\mathbf{1}_i = \mathbf{1}_{ip}$, and $\mathbf{1}_i^\mathsf{T}\mathbf{1}_i = N_i$. We then rewrite E_L:

$$E_L = \mathtt{a}^\mathsf{T}\boldsymbol{\mathcal{L}}\mathtt{a} \tag{6.14a}$$

$$\text{with: } \boldsymbol{\mathcal{L}} = \begin{bmatrix} L & 0 & \cdots & 0 \\ 0 & L & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & L \end{bmatrix}. \tag{6.14b}$$

For the sparsity term E_S, we introduce the approximate energy:

$$\tilde{E}_S = \sum_{i,p} u_{ip}(\alpha_{ip})^2 + v_{ip}(1 - \alpha_{ip})^2 \tag{6.15a}$$

$$\text{with: } u_{ip} = |\alpha'_{ip}|^{\tilde{\gamma}_p - 2} \quad \text{and} \quad v_{ip} = |1 - \alpha'_{ip}|^{\tilde{\gamma}_p - 2}, \tag{6.15b}$$

where α' is equal to the constrained sparsification result at the first iteration and to the solution of the previous iteration later. For the matrix reformulation, we use $\boldsymbol{\mathcal{D}}_u$ the diagonal matrix built with the u_{ip} values, and $\mathtt{v}$ and $\boldsymbol{\mathcal{D}}_v$ the vector and diagonal matrix built with the v_{ip} values:

$$\begin{aligned} \tilde{E}_S &= \mathtt{a}^\mathsf{T}\boldsymbol{\mathcal{D}}_u\mathtt{a} + (\mathbf{1}_{ip} - \mathtt{a})^\mathsf{T}\boldsymbol{\mathcal{D}}_v(\mathbf{1}_{ip} - \mathtt{a}) && \text{(6.16a)} \\ &= \mathtt{a}^\mathsf{T}(\boldsymbol{\mathcal{D}}_u + \boldsymbol{\mathcal{D}}_v)\mathtt{a} - 2\mathtt{a}^\mathsf{T}\mathtt{v} + \mathbf{1}_{ip}^\mathsf{T}\mathtt{v}, && \text{(6.16b)} \end{aligned}$$

where we used $\boldsymbol{\mathcal{D}}_v \mathbf{1}_{ip} = \mathtt{v}$ and $\mathtt{v}^\mathsf{T}\mathtt{a} = \mathtt{a}^\mathsf{T}\mathtt{v}$.

To derive a linear system, we sum all the energy terms in their matrix forms and write that the derivative with respect to $\mathtt{a}$ should be zero at a minimum. This leads to:

$$(\boldsymbol{\mathcal{L}} + \boldsymbol{\mathcal{D}}_u + \boldsymbol{\mathcal{D}}_v + \mathcal{I} + \lambda \mathcal{C}^\mathsf{T}\mathcal{C})\mathtt{a} = \mathtt{v} + \hat{\mathtt{a}} + \lambda \mathbf{1}_{ip} \tag{6.17}$$

We solve this equation using preconditioned conjugate gradient optimization [Barrett et al., 1994]. In our experiments, 20 iterations generate results with satisfactory sparsity. Figure 6.7 illustrates the benefits of our approach.

The size of the linear system is $N_i N_p$. While this is large, it remains tractable because the number of soft layers N_i is set to 5 and it is close to being block-diagonal, the only coefficients outside the diagonal coming from the sum-to-one term E_C that contributes $\mathcal{C}^\mathsf{T}\mathcal{C}$ to the system. Since $\mathcal{C}$ is made of 5 juxtaposed $N_p \times N_p$ identity matrices, $\mathcal{C}^\mathsf{T}\mathcal{C}$ is made of 25 $N_p \times N_p$ identity matrices in a 5×5 layout, i.e. it is very sparse and is easily handled by the solver.

6.4. Semantic Feature Vectors

We defined our semantic affinity term (§ 6.2.2) with feature vectors $\boldsymbol{f}$ that are similar for pixels on the same object and dissimilar for pixels on different objects. Such vectors can be generated using different network architectures trained for semantic segmentation. In our implementation, we have combined a semantic segmentation approach with a network for metric learning. It should be noted that we do not claim the feature generation as a contribution.

We begin by computing a set of per-pixel semantic features for each input image. In principle, the network generating the features can be easily replaced to improve the results in parallel to advances in semantic segmentation, or to change the definition of *semantic objects*, such as to serve fine-grained or instance-aware semantic segmentation scenarios.

We train a deep convolutional neural network cascaded with metric learning, to generate features that are similar if they belong to the same object class, and distant from each other otherwise. The network outputs per-pixel semantic features of $d = 128$ dimensions. For simplicity, we denote a semantic feature vector $\boldsymbol{f}_p \in \mathbb{R}^D$ for each pixel p.

The base network of our feature extractor is based on DeepLab-ResNet-101 [Chen et al., 2017]. The DeepLab model is built on a fully convolutional variant of ResNet-101 [He et al., 2015a] with atrous convolutions and atrous spatial pyramid pooling. In the DeepLab-ResNet-101, the `res4b22` layer is the

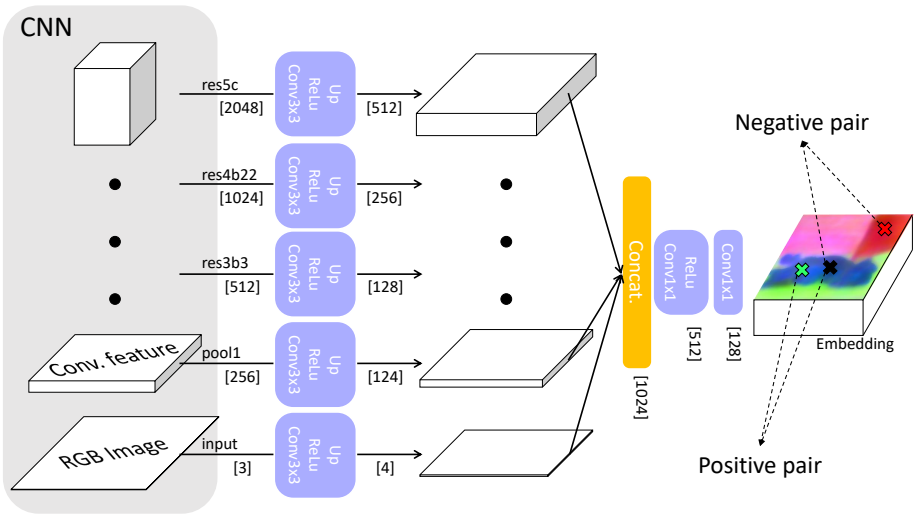

Figure 6.8.: *Network architecture. We extract the feature of intermediate representation of the base convolution neural network (we use the DeepLap variant of ResNet-101). The feature are compressed by* 3×3 *convolution followed by ReLu and bilinear up-sampling to have the same resolution with input. The concatenated features are fed to subsequent* 1×1 *convolution. On top of this feature, we apply sampling based metric learning in an end-to-end manner. We denote feature dimension as* $[\#]$.

most commonly used output as a generic feature, which is 2048 dimensional at one sixteenth of the original image resolution. Since our purpose is to extract per-pixel feature with plausible object contours and boundaries, directly leveraging multi-scale context information is favorable when compared to using a condensed feature at higher layer such as `res5b_relu`. We modify the architecture to take features from lower as well as higher-levels into account. We use the feature concatenation, motivated by [Hariharan et al., 2015; Bertasius et al., 2015], but we maintain a light representation to avoid large memory bottlenecks. We branch `input`, `pool1`, `res3b3`, `res4b22` and `res5c` layers to extract the features, followed by a 3×3 convolution with ReLu to compress the intermediate feature dimensions from $\{3, 256, 512, 1024, 2048\}$ to $\{4, 124, 128, 256, 512\}$, respectively, for a total of 1024 dimensions. We then upsample them via bilinear upsampling to the input image resolution, followed by two 1×1 convolution layers which gradually reduce 1024 feature dimension to 512 and then finally to $d = 128$. The final output of this process defines our per-pixel semantic features $\boldsymbol{f}_p$. Our architecture is visualized in Figure 6.8. It is worth pointing out that our architecture is fully convolutional, allowing it to be applied to inputs of arbitrary resolution. While Hariharan et al.; Bertasius et al. leverage the pre-trained network without re-training, we fine-tune the whole network for our purpose.

To train the whole network, we use L2 distance between pixel features as the metric to measure the semantic similarity. We will now describe our loss function on the pixel-level. Given a query vector $\boldsymbol{f}_p$ of a pixel p, we use positive vectors to pull the query towards positive one a and negative vectors to push to negative one for positive and negative examples from the query at a time [Hoffer and Ailon, 2015]. Since we work on input image resolution, to easily utilize more data and be computationally more efficient, we use N-pair loss [Sohn, 2016] with a slight modification. The N-pair loss benefits data efficiency by hard negative data-mining style formulation and cross-entropy style loss to alleviate the slow convergence by loss-balancing in triplet loss [Hoffer and Ailon, 2015]. While the N-pair loss is defined on an inner product-based metric, we replace it with L2 distance. Hence, our loss is defined by:

$$\begin{aligned} L_m = \frac{1}{|\mathcal{P}|} \sum_{p,q \in \mathcal{P}} & \mathbb{I}[l_p = l_q] \log \left(\left(1 + \exp \left(\| \boldsymbol{f}_p - \boldsymbol{f}_q \| \right) \right) / \, 2 \right) \\ & + \mathbb{I}[l_p \neq l_q] \log \left(1 + \exp \left(- \| \boldsymbol{f}_p - \boldsymbol{f}_q \| \right) / \, 2 \right), \end{aligned} \tag{6.18}$$

where $\mathcal{P}$ denotes the set of sampled pixels, $\| \cdot \|$ L2-norm (we divide it by d for normalization), $\mathbb{I}[\cdot]$ the indicator function that returns 1 if the statement is true and 0 otherwise, and l_p the semantic label of pixel p.

In (6.18), for positive pairs, i.e. $l_p = l_q$, the corresponding term $\log \left(\left(1 + \exp \left(\| \boldsymbol{f}_p - \boldsymbol{f}_q \| \right) \right) / \, 2 \right)$ approaches zero. The conjugate relation applies to the negative pairs in the second term in (6.18). Since we only use this cue, whether two pixels belong to the same category or not, specific object category information is not used during training. Hence, our method is a class agnostic approach. This fact does not harm our overall goal of semantic soft segmentation as we aim to create soft segments that cover semantic objects, rather than classification of the objects in an image. This also enables us to take into account diversity of semantics and not be limited to user-selected classes.

We construct the set of sampled pixels $\mathcal{P}$ as follows. During training, we feed a single image as a mini-batch to the network, and we get the features for all pixels. Given an input image and its corresponding semantic ground-truth labels, we first randomly sample P_{inst} number of instances, then for each instance, we randomly sample P_{pix} number of pixels within each instance label mask, so that the number of pixels in each group are balanced. We minimize (6.18) for the selected samples, and we repeat the sampling 10 times per image, accumulate gradients from them, and update at once. We set $P_{\text{inst}} = 3$ and $P_{\text{pix}} = 1000$. We can easily compute (6.18) in a matrix form by using $\mathbf{D} = \mathbf{F}\mathbf{1}_d\mathbf{1}_d{}^T + (\mathbf{V}\mathbf{1}_d\mathbf{1}_d{}^T)^T - 2\mathbf{V}\mathbf{V}^T$, where $\mathbf{D}$ is the matrix containing

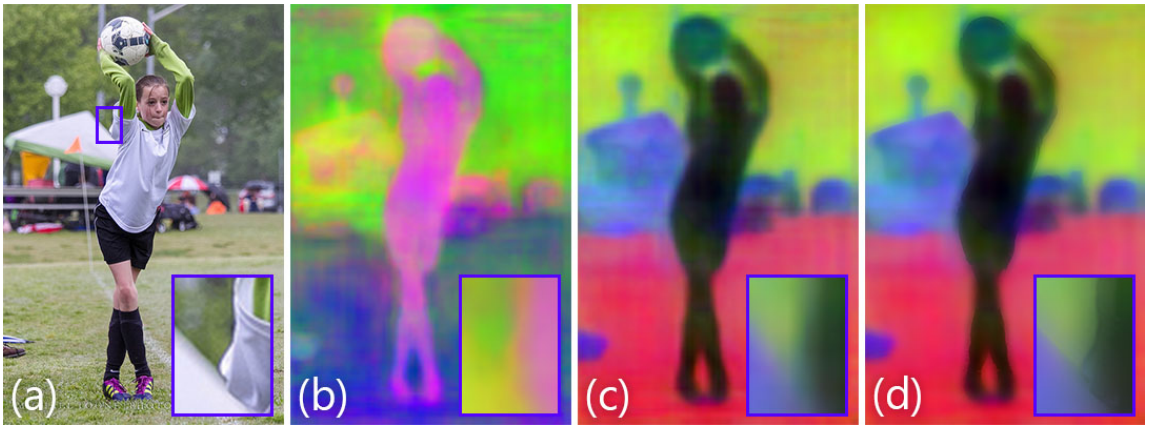

Figure 6.9.: *We first generate a 128-dimensional feature vector per pixel for a given image (a). A random projection of 128 dimensions to 3 is shown in (b). We reduce the dimensionality of the features to 3 using principle component analysis per image (c). Before the dimensionality reduction, we edge-align the features with guided filter [He et al., 2013] (d). Image from [Lin et al., 2014].*

L2 distances between the feature vectors, $\mathbf{1}_d$ is a row-vector of ones, and $\mathbf{F}$ contains the feature vectors of the samples pixels:

$$\mathbf{F} = \left[f_{1,1} \cdots f_{1,P_{\text{pix}}}, \mathbf{f}_{2,1} \cdots f_{2,P_{\text{pix}}}, \cdots f_{P_{\text{inst}},P_{\text{pix}}} \right]^T \tag{6.19}$$

We trained our network on the training split of COCO-Stuff [Caesar et al., 2016], which has 182 number of object and stuff categories with instance-level annotation. We initialized the base DeepLab part with the pretrained weights on the semantic segmentation task of MS-COCO [Lin et al., 2014] (80 categories), and the remaining parts with Xavier initialization [He et al., 2015b]. We set the learning rate to 5×10^{-4} for the base part and 5×10^{-3} for the rest to compensate for the random initialization. We use stochastic gradient descent with momentum 0.9, poly-learning rate decay of 0.9 as suggested by Chen et al. [2017], and weight decay of 5×10^{-4}. We also use drop-out with probability 0.5 for 1×1 convolutions at the two last stages. We train for $60k$ iterations and it roughly takes less than one day on an NVIDIA Titan X Pascal GPU.

Preprocessing

The 128-dimensional feature vectors f_p have enough capacity to represent a large diversity of real-world semantic categories. However, for a given image, as the number of object categories present in the scene is inherently limited, the effective dimensionality of the feature vector is much smaller.

Following this fact, in order to make the graph construction more tractable and less prone to parameter-tuning, we reduce the dimensionality of the feature vectors to three using per-image principle component analysis.

One of the major shortcomings of semantic *hard* segmentation is its inaccuracy around object boundaries [Bertasius et al., 2016]. This fact is well-reflected in the generated feature vectors as well, as shown in Figure 6.9. In order to compute more effective affinities when we are inserting the semantic information into the graph, we regularize the feature vectors using guided filtering [He et al., 2013] with the guidance of the input image. This makes the features to be more consistent with hard boundaries in the image, as shown in Figure 6.9. We do this filtering for all 128 dimensions prior to the dimensionality reduction. Finally, we normalize the lower-dimensional features to be in the range $[0, 1]$ to get the three dimensional feature vectors $\tilde{f}_p$ to be used for affinity computations.

6.5. Experimental Evaluation

Semantic soft segmentation, being at the intersection of semantic segmentation, natural image matting, and soft segmentation, is challenging to evaluate numerically. Semantic segmentation datasets provide binary labeling that is not always pixel-accurate, which makes them ill-suited for benchmarking semantic soft segmentation. Natural image matting methods are typically evaluated on dedicated benchmarks [Rhemann et al., 2009] and datasets [Xu et al., 2017]. These benchmarks are designed to evaluate methods that make use of a secondary input, called trimap, which defines the expected foreground and background, and an uncertain region. Further, the semantic aspect of our work is beyond the scope of these benchmarks. As a result, we resort to qualitative comparisons with related methods and discuss the characteristic differences between the various approaches.

6.5.1. Implementation Details

We use the sparse eigendecomposition and direct solver available in MATLAB for our proof-of-concept implementation for the constrained sparsification stage of our algorithm. This step takes around 3 minutes for a 640×480 image. The relaxed sparsification step uses the preconditioned conjugate gradient optimization implementation of MATLAB. Each iteration typically converges in 50 to 80 iterations and the process takes around 30 seconds. The run-time of our algorithm grows linearly with the number of pixels.

6.5.2. Spectral Matting and Semantic Segmentation

In Figures 6.10 and 6.11, we show our results together with that of spectral matting [Levin et al., 2008b] as the most related soft segmentation method to ours, and two state-of-the-art methods for semantic segmentation: the scene parsing method by Zhao et al. [2017] (PSPNet) and the instance segmentation method by He et al. [2017] (Mask R-CNN). Spectral matting generates around 20 soft segments per image, and provides several alternative foreground mattes by combining the soft segments to maximize an *objectness score*. These mattes are not definite results but are provided to the user as options, and showing all 20 segments would make the comparisons harder to evaluate. Instead, we apply our soft segment grouping method that uses the semantic features to the results of spectral matting.

The presented examples show that semantic segmentation methods, while being successful in recognizing and locating the objects in the image, suffer from low accuracy around the edges of the objects. While their accuracy is satisfactory for the task of the semantic segmentation, errors around object edges are problematic for image editing or compositing applications. On the other end of the spectrum, spectral matting is able to successfully capture most of the soft transitions around the objects. However, due to the lack of semantic information, their segments often cover multiple objects at once, and the alpha values are often not sparse for any given object. In comparison, our method captures objects in their entirety or subparts of them without grouping unrelated objects and achieves a high accuracy at edges, including soft transitions when appropriate.

It should be noted that it is not uncommon for our method to represent the same object in multiple segments such as the horse carriage in Figure 6.10 (2) or the background fence in Figure 6.10 (4). This is mainly due to the preset number of layers, five, sometimes exceeds the number of meaningful regions in the image. Some small objects may be missed in the final segments despite being detected by the semantic features, such the people in the background in Figure 6.11 (5). This is due to the fact that, especially when the color of the object is similar to the surroundings, the objects do not appear well-defined in the eigenvectors and they end up being merged into closeby segments. Our semantic features are not instance-aware, i.e. the features of two different objects of the same class are similar. This results in multiple objects being represented in the same layer such as the cows in Figure 6.10 (1), the people in Figure 6.10 (5) or the giraffes in Figure 6.11 (3). With instance-aware features, however, our method would be capable of generating separate soft segments for different instances of objects.

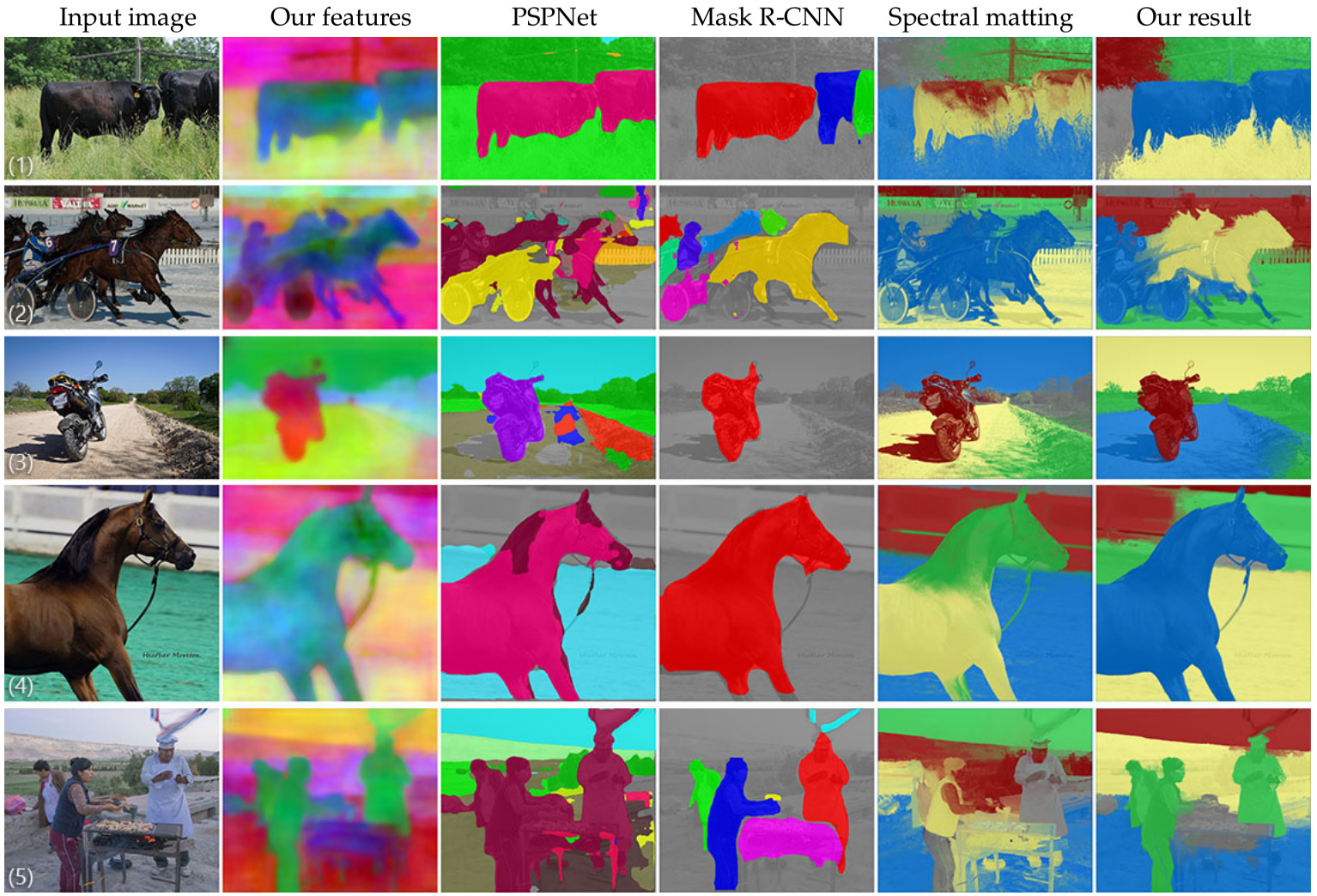

Figure 6.10.: *We show our results together with that of Zhao et al. [2017] (PSPNet), He et al. [2017] (Mask R-CNN), and spectral matting [Levin et al., 2008b]. The segmentations are overlayed onto the grayscale version of the image for a better evaluation around segment boundaries. Notice the inaccuracies of PSPNet and Mask R-CNN around object boundaries, and the soft segments of spectral matting extending beyond object boundaries. Images from [Lin et al., 2014].*

Figure 6.11.: *Figure 6.10 continued.*

Figure 6.12.: *From the input image (a), our method generates the matte shown in (b). We show that the trimaps with different unknown region widths, generated using the semantic segments by PSPNet [Zhao et al., 2017] (d) or Mask R-CNN [He et al., 2017] (e), fail to provide foreground and background regions reliably, which affects the matting result generated using information-flow matting Chapter 5 negatively. In the bottom example, PSPNet trimaps are generated by selecting a single class (left) or all the classes that correspond to the object. We also provide the matting result using a trimap generated by our result (c) which demonstrates the performance of the matting algorithm given an accurate trimap. Images from [Lin et al., 2014].*

Figure 6.13.: *Our soft segments and the corresponding mattes for the foreground objects. Note that trimaps usually provided for natural matting were not used to produce these results. Images from [Xu et al., 2017].*

Grayscale images are especially challenging for soft segmentation and image matting methods with the lack of color cues on which such methods typically rely. The performance of semantic segmentation methods, on the other hand, does not degrade substantially when processing a grayscale image. Figure 6.11 (5) demonstrates that our method can succesfully leverage the semantic information for soft segmentation of a grayscale image.

6.5.3. Natural Image Matting

In principle, semantic soft segments can be generated by cascading semantic segmentation and natural image matting. The *trimap*, defining the foreground,

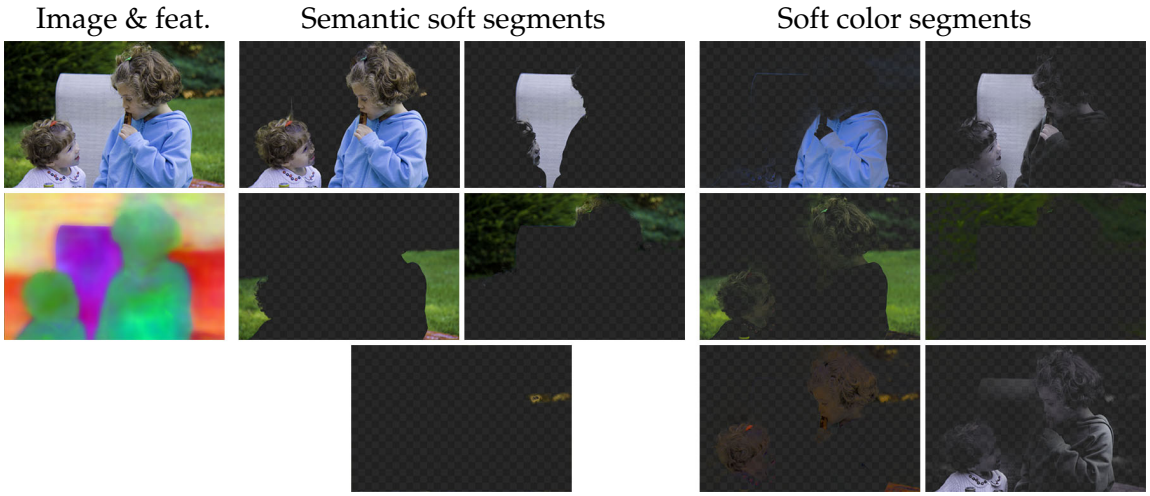

Figure 6.14.: *Semantic soft segments and soft color segments estimated by our method presented in Chapter 4 shown together for conceptual comparison. Both methods are fully automated and only require the input image for soft segmentation. Image from [Bychkovsky et al., 2011].*

background, and soft transition regions, can be generated from the semantic hard segments to be fed to the natural matting method. Shen et al. [2016] and Qin et al. [2017] use similar approaches for class-specific problems. We show two examples of such scenario in Figure 6.12 to demonstrate the shortcomings of this approach by generating trimaps using Mask R-CNN and PSPNet results and estimating the mattes using our natural matting method presented in Chapter 5. A strong assumption made by natural image matting methods is that the provided trimap is *correct*, i.e. the defined foreground and background regions are used as hard constraints to guide the methods in modeling the layer colors. Inaccuracies in the estimated semantic boundaries, however, often fails to provide reliable trimaps even with a large unknown-region width. This results in severe artifacts in the matting results, as highlighted in the figure. We show that the natural matting method succeeds given an accurate trimap, generated using our results for demonstration.

While general natural image matting is beyond the scope of our method, Figure 6.13 shows several examples where our method is able to generate satisfactory results on images from natural image matting datasets without requiring a trimap.

6.5.4. Soft Color Segmentation

Soft color segmentation, as discussed in detail in Chapter 4, decomposes the input image into soft layers of homogeneous colors and have been shown

to be useful for image editing and recoloring applications. As a conceptual comparison between semantic soft segments and soft color segments, Figure 6.14 shows our semantic soft segments with soft color segments. For a more convenient qualitative comparison, we estimated the layer colors for our soft segments using the closed-form color estimation method [Levin et al., 2008a].

It is immediately visible that the content of soft color segments extend beyond the object boundaries, while our results show semantically meaningful objects in the same segment, regardless of their color content. As these representations are orthogonal to each other, they can be used in orchestration to generate targeted recoloring results.

6.5.5. Using Semantic Soft Segments for Image Editing

We demonstrate several use cases of our soft segments for targeted image editing and compositing in Figure 6.15. Figure 6.15(1,3,4,7) show compositing results where we estimated the layer colors for our segments using closed-form layer color estimation [Levin et al., 2008a]. Notice the natural soft transitions between the selected foreground layers and the novel background. The soft segments can also be used for targeted image edits where they are used to define masks for specific adjustment layers such as adding motion blur to the train in (2), color grading the people and the backgrounds separately in (5,6) and separate stylization of the hot-air balloon, sky, terrain and the person in (8). While these edits can be done via user-drawn masks or natural matting algorithms, our representation provides a convenient intermediate image representation to make the targeted edits effortless for the artist.

6.6. Limitations

While we are able to generate accurate soft segmentations of images, in our prototype implementation our solvers are not optimized for speed. As a result, our runtime for a 640×480 image lies between 3 and 4 minutes. The efficiency of our method can be optimized in several ways, such as multi-scale solvers, but an efficient implementation of linear solvers and eigendecomposition lies beyond the scope of our study here.

In the constrained sparsification step, we generate around 15-25 segments, which are then grouped using the feature vectors into 5. The number of layers was set via empirical observations, and in some cases, an object may be divided into several layers. While this does not affect the applicability of

Figure 6.15.: *Semantic soft segments can be used for compositing tasks after layer color estimation.*

Figure 6.16.: *Semantic soft segments can be used as masks for targeted image editing tasks.*

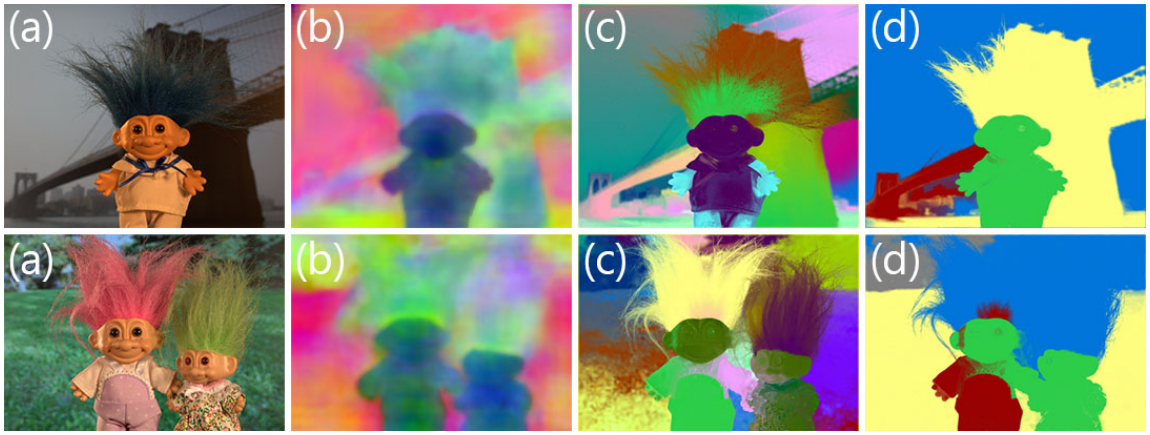

Figure 6.17.: *Two failure cases are shown. Top example: In case of large regions covering different objects with very similar colors (a) our feature vectors (b) and segments before grouping (c) fail to identify the separate objects in the image and result in inaccurate segmentation (d). Bottom example: When our feature vectors fail to represent the objects, even if when the initial layers are able to generate accurate soft transitions (c) the grouping of the soft segments (d) may fail. Images from [Rhemann et al., 2009].*

our method as combining those layers in editing is trivial, more sophisticated ways of grouping the layers such as through recognition and classification can be devised.

Our method does not generate separate layers for different instances of the same class of objects. This is due to our feature vectors, which does not provide instance-aware semantic information. Our soft segmentation formulation, however, is agnostic to the semantic features. Hence, a more advanced feature generator would make it possible to generate instance-level soft segmentation results when combined with a better-fitting segment-grouping strategy.

We have shown several results from natural matting datasets. However, it should be noted that we do not aim to solve the natural matting problem in general. Natural matting is a mature field with many specific challenges, such as generating accurate mattes around very similarly-colored foreground and background regions, and state-of-the-art methods depend on the color distributions of the two regions to increase performance around such areas. As Figure 6.17 demonstrates, our method may fail at the initial constrained sparsification step when the object colors are very similar, or the grouping of soft segments may fail due to unreliable semantic feature vectors around large transition regions.

CHAPTER 7

Conclusion

Soft segmentation is an essential part of realistic image editing and compositing. Accurate estimation of soft transitions between image regions and the underlying layer colors around these soft transitions allows applying complex color and scene changes effortlessly. In this thesis, we approached the soft segmentation problem from two complementary properties of a photograph. In Part I, we focused on analyzing the photograph in terms of color by representing the image as a mixture of a small set of dominant scene colors. We addressed a widely utilized industrial process of green-screen keying in Chapter 3 where the color-based analysis made a great fit. We then extended the core novelty of this approach, color unmixing, to the analysis of natural scenes in Chapter 4, which opens up numerous image editing possibilities. Our methodology shifted to graph-based representations of images for soft segmentation of objects in Part II. We introduced an affinity-based approach to the widely studied natural image matting problem. Finally, we studied a novel segmentation paradigm, semantic soft segmentation, that fully automatically generates soft segments for objects in an image by combining spectral segmentation and machine learning.

From professional movie post-production to amateur video editing, isolating the foreground in a scene such as the actors to replace or edit the background is an essential part of the pipeline. However, this task referred to as keying or matting is heavily expertise-driven and time-consuming. The difficulty of achieving production-quality results drives many to one popular setup: green-screens, shooting a scene in an environment where the background is controlled to have a solid and distinct color. Despite its common use, this

unnatural scene environment is not favored by many movie professionals and studios spend a lot of resources to be able to freely edit the background contents of natural scenes. We proposed novel approaches to both controlled and uncontrolled scenarios, where we aim to decrease the manual work that goes into the keying process while increasing the final quality.

Common commercial tools for green-screen keying typically makes use of the background color and relies on the compositing artist to refine the results. In Chapter 3, we introduced a keying pipeline that relies on our novel color unmixing formulation. Color unmixing is a per-pixel energy formulation which represents a pixel color as a mixture of the main scene colors. A two-step simplistic interaction procedure is utilized to effectively use color unmixing for green-screen keying, where the user is expected to identify the main colors in a scene and where they approximately appear. The heavy lifting, i.e. making sure the soft transitions well represent the foreground object, is then done by our energy formulation. We showed that the proposed technique substantially decreases the interaction time required for achieving production-ready keying quality when compared to commercial keying tools. Color unmixing shows favorable performance around challenging regions such as motion blur and is able to reliably extract the foreground even if the foreground colors appear similar to the background due to the extensive parametric representation of the scene colors.

We also provided a study of natural image matting that covers related problems such as matte refinement and layer color extraction in Chapter 5. Natural matting is only recently gaining popularity in professional movie production as new approaches are beginning to provide the matting quality necessary for production, although it has been studied in academia for two decades. Rather than relying on the background color and user expertise, commonly used interaction in natural matting is a coarse segmentation of foreground and background before the matte computation. We proposed an affinity-based approach to this problem, where we define inter-pixel connections to strategically target challenging foreground structures. The design of the underlying graph structure is conceptualized by *information flows*, modeling how the information provided by the user can be effectively distributed into the rest of the image. Our state-of-the-art matting results are computed with an elegant closed-form expression. Our design approach can be easily extended to problems that require propagation of sparse information to the whole image. We demonstrated this by providing another approach, this time targeting the computation of unmixed layer colors which are necessary for compositing together with the estimated mattes. We demonstrated the effectiveness of the proposed graph formulation further through spectral analysis and additional discussions.

While the interactive soft segmentation methods have wide use with targeted application scenarios, soft segmentation in general have many uses in common image editing tasks. Automatic soft segmentation, as a result, provides convenient intermediate representations of images that can be freely used even by an inexperienced user to perform a wide variety of image manipulation operations with very little effort. Although both interactive and automatic soft segmentation approaches aim to estimate soft transitions reliably, the existance of a user-defined foreground region greatly changes the theoretical approaches one should take in two scenarios. Keying and matting methods make heavy use of color characteristics of the foreground and background to reason about the soft transitions between them. In contrast, automatic soft segmentation approaches need to estimate the soft transitions *and* reason about which soft transitions are important or meaningful for the target segmentation scheme in the same formulation. In both parts of this thesis, we used the formulations and strategies we developed for interactive soft segmentation as a starting point to introduce color-based and object-based automatic soft segmentation methods.

Soft color segmentation is the estimation of multiple layers of homogeneous colors that represent each pixel of an image as a mixture of a small set of scene colors called color model. Chapter 4 describes a fully automatic pipeline for soft color segmentation that determines the number of layers, the color model itself and the corresponding layers with opacities. The fundamental component of our pipeline is an extended version of our color unmixing formulation. In the initial step, we determine a color model for the input photograph through a greedy scheme. The compactness of our color model is ensured by the use of color unmixing such the colors that can be well represented as a mixture of the existing entries in the color model are not considered as additional entries. By making sure that the optimized color unmixing energy is low for every pixel, we guarantee the color homogeneity in the final layers. Using the estimated color model, we formulated a three-step soft color segmentation formulation that can be implemented with parallelizability and scalability in mind. We showed that, by breaking this large global formulation into per-pixel sub-problems, high-quality segments can be estimated very efficiently when compared to state-of-the-art in the literature. Our extensive theoretical and experimental analysis shows that ours is the first soft color segmentation approach that can be effectively used in a wide range of realistic image editing scenarios.

We focused on the use of low-level features to formulate our interactive techniques as well as automatic soft color segmentation. In order to estimate a set of soft segments that correspond to *objects* in the image without any user input, we require a high-level sense of *objectness* in our formulation. By

merging the mathematical modeling of soft segmentation through spectral analysis and semantic segmentation through machine learning techniques, we studied a novel soft segmentation problem, semantic soft segmentation, in Chapter 6. The information on objectness comes from a deep neural network that is trained specifically to fit our graph-based soft segmentation formulation. We merged this high-level information with low-level information on local soft transitions and nonlocal color similarity in a single graph, following our information flow based design mechanism. We showed that this proposed graph reveals the objects with the soft transitions between them in the eigenvectors of the corresponding Laplacian matrix. We also proposed a matte sparsification approach using the same graph structure, that can remove spurious alpha values in multiple mattes together while keeping the meaningful soft transitions intact. We showed through many examples that the semantic soft segments can be used effectively for compositing and targeted image editing tasks.

7.1. Future directions

Joint illumination and color analysis. We based our color analysis in Part I solely on how the pixels appear in the final photograph. Although we showed many useful applications using color-based decomposition, the analysis can be extended to reflect how the observed colors were formed physically. The appearance of colors in photographs comes from the illumination color and direction in the scene, as well as the albedo of the object reflecting the light, in addition to the camera parameters such as the white balance. Our color-based decomposition gets affected by the illumination variations and shading as a result. A promising future direction is to integrate the color variations coming from illumination and albedo separately into the color unmixing formulation. Matte sparsity and representing the scene with a small set of colors, two assumptions we extensively make use of in soft color segmentation, apply better to the albedo. This can be combined with a smoothness assumption on shading for a joint illumination- and color-based decomposition.

Merging high-level and low-level features. We formulated an object-based soft segmentation approach without any user input by utilizing neural networks in a graph-based analysis. Our use of high-level features from a neural network in the same formulation as low-level features from the image texture represents an alternative use case for deep learning, where it was used to enrich our mathematical modeling of image representations. This way, we were able to study semantic soft segmentation which had been beyond the possible applications of deep learning because of limited and hard-to-

acquire ground truth data. This line of research will open up many new application scenarios with data-driven approaches enriching well-established mathematical models of image formation. Specifically for soft segmentation, the high-level properties of the final layers can be enriched with information on depth or material properties by extending our formulation.

Learning targeted affinity definitions. We showed in Chapter 5 that a carefully designed graph can represent complex foreground structures well for image matting. We relied on nonlocal connections between pixels when constructing our graph. The selection procedure of these nonlocal neighbors is admittedly simple, based on k nearest neighbors searches defined over variations of feature vectors comprised of colors and spatial coordinates of pixels. The demonstrated performance and generalizability of our empirically designed formulation show the potential of using targeted information flow definitions. The information propagation on the image space is useful for an extended set of applications in computer vision in addition to image editing and matting, such as refining depth maps that are typically inaccurate around edges between objects. With further research, this potential can be realized more fully by learning the best possible connections for the linear system formulation from large-scale data for the target application domain.

Efficiency and video. We focused specifically of soft segmentation of still images rather than videos. One reason behind this is the complexity of this problem even for a still image due to the expectation of high accuracy for realistic image editing. Applying our algorithm for green-screen keying for image sequences is able to give temporally stable results thanks to the stability of color unmixing in the presence of a constrained background. Soft color segmentation can be extended to video with relatively little effort that includes the estimation of a temporally evolving color model that accounts for colors that disappear or appear with the changing scene. Extending our graph-based formulation in Part II to video presents a more challenging research direction. Ensuring temporal stability in natural matting and semantic soft segmentation requires inter-frame connections and hence a larger linear system. Research towards more sparse and compact graph representations for video as well as faster eigensolvers will open up the application of automatic high-quality compositing for videos without a studio environment.

Bibliography

[Achanta et al., 2012] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2274–2282, 2012.

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

[Aksoy et al., 2017a] Yağız Aksoy, Tunç Ozan Aydın, and Marc Pollefeys. Designing effective inter-pixel information flow for natural image matting. In *Proc. CVPR*, 2017.

[Aksoy et al., 2017b] Yağız Aksoy, Tunç Ozan Aydın, Aljoša Smolić, and Marc Pollefeys. Unmixing-based soft color segmentation for image manipulation. *ACM Trans. Graph.*, 36(2):19:1–19:19, 2017.

[Aksoy et al., 2018a] Yağız Aksoy, Changil Kim, Petr Kellnhofer, Sylvain Paris, Mohamed Elgharib, Marc Pollefeys, and Wojciech Matusik. A dataset of flash and ambient illumination pairs from the crowd. In *Proc. ECCV*, 2018.

[Aksoy et al., 2018b] Yağız Aksoy, Tae-Hyun Oh, Sylvain Paris, Marc Pollefeys, and Wojciech Matusik. Semantic soft segmentation. *ACM Trans. Graph.*, 37(4):72:1–72:13, 2018.

[Angehrn et al., 2014] Florian Angehrn, Oliver Wang, Yağız Aksoy, Markus Gross, and Aljoša Smolić. MasterCam FVV: Robust registration of multiview sports video to a static high-resolution master camera for free viewpoint video. In *Proc. ICIP*, 2014.

[Bae and Durand, 2007] Soonmin Bae and Fredo Durand. Defocus magnification. *Comput. Graph. Forum*, 26(3):571–579, 2007.

[Barrett et al., 1994] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 1994.

[Bertasius et al., 2015] Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani. High-for-low and low-for-high: Efficient boundary detection from deep object features and its applications to high-level vision. In *Proc. ICCV*, 2015.

[Bertasius et al., 2016] Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani. Semantic segmentation with boundary neural fields. In *Proc. CVPR*, 2016.

[Bertsekas, 1982] Dimitri P. Bertsekas. The method of multipliers for equality constrained problems. In *Constrained optimization and Lagrange multiplier methods*, pages 96–157. Academic Press, New York, 1982.

[Boykov and Kolmogorov, 2004] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, 2004.

[Boykov et al., 2001] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, 2001.

[Bychkovsky et al., 2011] V. Bychkovsky, S. Paris, E. Chan, and F. Durand. Learning photographic global tonal adjustment with a database of input/output image pairs. In *Proc. CVPR*, 2011.

[Caesar et al., 2016] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context. `arXiv:1612.03716 [cs.CV]`, 2016.

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

[Chen et al., 2012] Xiaowu Chen, Dongqing Zou, Qinping Zhao, and Ping Tan. Manifold preserving edit propagation. *ACM Trans. Graph.*, 31(6):132:1–132:7, 2012.

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

[Chen et al., 2013b] Xiaowu Chen, Dongqing Zou, S.Z. Zhou, Qinping Zhao, and Ping Tan. Image matting with local and nonlocal smooth priors. In *Proc. CVPR*, 2013.

[Chen et al., 2017] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. DeepLab: Semantic image segmentation with

deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017.

[Chen et al., 2018] Quan Chen, Tiezheng Ge, Yanyu Xu, Zhiqiang Zhang, Xinxin Yang, and Kun Gai. Semantic human matting. In *Proc. ACM Multimedia*, 2018.

[Cho et al., 2019] D. Cho, Y. Tai, and I. S. Kweon. Deep convolutional neural network for natural image matting using initial alpha mattes. *IEEE Trans. Image Process.*, 28(3):1054–1067, 2019.

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

[Chuang et al., 2003] Yung-Yu Chuang, Dan B Goldman, Brian Curless, Brian Curless, David H. Salesin, and Richard Szeliski. Shadow matting and compositing. *ACM Trans. Graph.*, 22(3):494–500, 2003.

[Farid and Simoncelli, 2004] H. Farid and E. P. Simoncelli. Differentiation of discrete multidimensional signals. *IEEE Trans. Image Process.*, 13(4):496–508, 2004.

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

[Feng et al., 2016] Xiaoxue Feng, Xiaohui Liang, and Zili Zhang. A cluster sampling method for image matting via sparse coding. In *Proc. ECCV*, 2016.

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

[Grundhöfer et al., 2010] Anselm Grundhöfer, Daniel Kurz, Sebastian Thiele, and Oliver Bimber. Color invariant chroma keying and color spill neutralization for dynamic scenes and cameras. *The Visual Computer*, 26(9):1167–1176, 2010.

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

[Hariharan et al., 2015] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proc. CVPR*, 2015.

[He et al., 2011] Kaiming He, C. Rhemann, C. Rother, Xiaoou Tang, and Jian Sun. A global sampling method for alpha matting. In *Proc. CVPR*, 2011.

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

[He et al., 2015a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. `arXiv:1512.03385 [cs.CV]`, 2015.

[He et al., 2015b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet

classification. In *Proc. ICCV*, 2015.

[He et al., 2017] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. In *Proc. ICCV*, 2017.

[Hoffer and Ailon, 2015] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, 2015.

[Hui et al., 2019] Zhuo Hui, Ayan Chakrabarti, Kalyan Sunkavalli, and Aswin C. Sankaranarayanan. Learning to separate multiple illuminants in a single image. In *Proc. CVPR*, 2019.

[Johnson et al., 2016] J. Johnson, E. S. Varnousfaderani, H. Cholakkal, and D. Rajan. Sparse coding for alpha matting. *IEEE Trans. Image Process.*, 25(7):3032–3043, 2016.

[Karacan et al., 2015] Levent Karacan, Aykut Erdem, and Erkut Erdem. Image matting with KL-divergence based sparse sampling. In *Proc. ICCV*, 2015.

[Kaspar et al., 2018] Alexandre Kaspar, Geneviève Patterson, Changil Kim, Yağız Aksoy, Wojciech Matusik, and Mohamed Elgharib. Crowd-guided ensembles: How can we choreograph crowd workers for video segmentation? In *Proc. ACM CHI*, 2018.

[Kolmogorov and Zabih, 2004] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):147–159, 2004.

[Lee and Wu, 2011] P. Lee and Ying Wu. Nonlocal matting. In *Proc. CVPR*, 2011.

[LeGendre et al., 2017] Chloe LeGendre, David Krissman, and Paul Debevec. Improved chromakey of hair strands via orientation filter convolution. In *ACM SIGGRAPH 2017 Posters*, 2017.

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

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

[Lin et al., 2011] Hai Ting Lin, Yu-Wing Tai, and M.S. Brown. Motion regularization for matting motion blurred objects. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(11):2329–2336, 2011.

[Lin et al., 2014] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Proc. ECCV*, 2014.

[Lutz et al., 2018] Sebastian Lutz, Konstantinos Amplianitis, and Aljoša Smolić. AlphaGAN: Generative adversarial networks for natural image matting. In *Proc. BMVC*, 2018.

[Mitsunaga et al., 1995] Tomoo Mitsunaga, Taku Yokoyama, and Takashi Totsuka. Autokey: Human assisted key extraction. In *Proc. SIGGRAPH*, 1995.

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

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

[Posirca et al., 2011] Iulia Posirca, Yunmei Chen, and Celia Z. Barcelos. A new stochastic variational PDE model for soft Mumford-Shah segmentation. *Journal of Mathematical Analysis and Applications*, 384(1):104–114, 2011.

[Qin et al., 2017] S. Qin, S. Kim, and R. Manduchi. Automatic skin and hair masking using fully convolutional networks. In *Proc. ICME*, 2017.

[Rhemann et al., 2009] Christoph Rhemann, Carsten Rother, Jue Wang, Margrit Gelautz, Pushmeet Kohli, and Pamela Rott. A perceptually motivated online benchmark for image matting. In *Proc. CVPR*, 2009.

[Roweis and Saul, 2000] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

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

[Ryffel et al., 2017] Mattia Ryffel, Fabio Zünd, Yağız Aksoy, Alessia Marra, Maurizio Nitti, Tunç Ozan Aydın, and Bob Sumner. AR museum: A mobile augmented reality application for interactive painting recoloring. In *International Conference on Game and Entertainment Technologies*, 2017.

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

[Shahrian and Rajan, 2012] E. Shahrian and D. Rajan. Weighted color and texture sample selection for image matting. In *Proc. CVPR*, 2012.

[Shahrian et al., 2013] E. Shahrian, D. Rajan, B. Price, and S. Cohen. Improving image matting using comprehensive sampling sets. In *Proc. CVPR*, 2013.

[Shen et al., 2016] Xiaoyong Shen, Xin Tao, Hongyun Gao, Chao Zhou, and Jiaya Jia. Deep automatic portrait matting. In *Proc. ECCV*, 2016.

[Singaraju and Vidal, 2011] D. Singaraju and R. Vidal. Estimation of alpha mattes

for multiple image layers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(7):1295–1309, 2011.

[Smith and Blinn, 1996] Alvy Ray Smith and James F. Blinn. Blue screen matting. *ACM Trans. Graph.*, pages 259–268, 1996.

[Sohn, 2016] Kihyuk Sohn. Improved deep metric learning with multi-class N-pair loss objective. In *Proc. NIPS*, 2016.

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

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

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

[Tang et al., 2019] Jingwei Tang, Yağız Aksoy, Cengiz Öztireli, Markus Gross, and Tunç Ozan Aydın. Learning-based sampling for natural image matting. In *Proc. CVPR*, 2019.

[Wang and Cohen, 2007] J. Wang and M. F. Cohen. Optimized color sampling for robust matting. In *Proc. CVPR*, 2007.

[Wu et al., 2007] Tai-Pang Wu, Chi-Keung Tang, Michael S. Brown, and Heung-Yeung Shum. Natural shadow matting. *ACM Trans. Graph.*, 26(2), 2007.

[Xu et al., 2017] Ning Xu, Brian Price, Scott Cohen, and Thomas Huang. Deep image matting. In *Proc. CVPR*, 2017.

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

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

[Zhao et al., 2017] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proc. CVPR*, 2017.

[Zhu et al., 2013] X. Zhu, S. Cohen, S. Schiller, and P. Milanfar. Estimating spatially varying defocus blur from a single image. *IEEE Trans. Image Process.*, 22(12):4879–4891, 2013.

[Zhu et al., 2017] Bingke Zhu, Yingying Chen, Jinqiao Wang, Si Liu, Bo Zhang, and Ming Tang. Fast deep matting for portrait animation on mobile phone. In

Proc. ACM Multimedia, 2017.