Internet-Data-Driven Image Creation and Enhancement

Yiming Liu

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Professor Szymon M. Rusinkiewicz

January 2016

Abstract

Traditional systems for image creation and enhancement are completely reliant on user input, either to generate content directly or to evaluate and tune algorithms. The input must be generated by talented artists, and it is applicable only to the image for which it was originally intended. This poses scalability challenges for such traditional image generation and editing systems.

An alternative paradigm, which is being researched in a variety of content-generation domains, is to exploit the large amount of data made available by users over the Internet. This data may take many forms: images, image keywords, and even judgments about image quality. Although the data is noisy and was not originally intended for the task at hand, it nevertheless serves as a useful source of "prior information." These priors make it possible to build image creation and enhancement systems that require only a small amount of user input, and do not require the users to be professional artists or designers.

This thesis describes three such systems, including low-level image-processing (motion deblurring), high-level editing (keyword-based image stylization), and image creation (sketch-driven iconification). First, by learning based on a massive online user study, we develop a metric for automatically predicting the perceptual quality of images produced by motion deblurring algorithms, without access to the original images. Second, by leveraging online image search engines, we investigate an approach to photo filtering that only requires the user to provide one or more keywords. Third, we develop a system that synthesize a variety of pictograms by remixing portions of icons retrieved from a large online repository.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Szymon Rusinkiewicz, for his continuous support and mentoring of my research. His motivation, patience, optimistic attitude, and immense knowledge on research and engineering make this five-year journey pleasant and invaluable.

It is my honor to work with many great researchers, including Aseem Agarwala, Sunghyun Cho, Michael Cohen, Adam Finkelstein, Jingwan Lu, Matt Uyttendaele, Jue Wang, and my advisor, Szymon Rusinkiewicz. I truly appreciate them, for teaching me to identify and solve great research problems, sharing expertise in their research fields, and guiding me to explore effective ways of collaboration. Collaboration with them is such a fruitful and enjoyable learning experience.

I was fortunate to receive lots of help throughout my PhD career. I am indebted to the members of Princeton Media Group, Adobe Research, and Microsoft Research, and also anonymous reviewers of SIGGRAPH, SIGGRAPH Asia, and EGSR, for their selflessly sharing great thoughts on my research projects. Special thanks to Adam Finkelstein, Thomas Funkhouser, and Jianxiong Xiao, for suggestions on all my projects. My data-driven projects were impossible without the data on the Internet. I would like to thank Flickr users for their photos, TheNounProject artists for their icons, Mechanical Turkers for user studies, MIT and Adobe for their FiveK dataset, and Google and Microsoft image search engines for enabling the possibilities of my stylization project. My PhD projects and this thesis were supported by Princeton Fellowship, NSF grants CCF-1012147, CCF-1027962, and gifts from Adobe Research.

Finally, I would like to dedicate this thesis to my parents and my friends, for their continuous love, care, support, and encouragement through my PhD years.

To my parents and friends.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Everybody loves beautiful images. We wish we could take clean and sharp photos, and make them as artistic as the photos taken by famous photographers. We wish we could make concise yet intriguing posters and icons as great designers do. In computer science, image creation and enhancement is the research topic in which we develop systems to help us realize these dreams.

Traditional systems such as Adobe Photoshop, Lightroom, and Illustrator completely rely on user input. We have to draw with digital brushes and pencils, drag sliders to adjust parameters of tools, all by ourselves. For professional users, these systems are powerful and handy, which enable them to create and enhance images of equally high quality yet with less cost on computers. Novice users, however, would be lost in the systems due to lack of talents and experiences.

In recent years, software such as Instagram is invented to resolve the embarrassment of novice users. They come with several *preset* filters to stylize users' images without any additional input. The design procedure behind these preset filters, however, is still completely reliant on the input from filter designers. The effort in designing each filter is tremendous, and unfortunately is only applicable to that single filter. This poses serious challenges on the scalability of filter design, which in turn, narrows the end user's choices.

In this thesis, we explore an alternative paradigm, which exploits the large-scale and continuously-updated data made available by users on the Internet. This paradigm has been successfully deployed in machine translation, high-level computer vision, and shape analysis. We demonstrate that it also applies to image creation and enhancement. The data on the Internet, in our three systems, serves as "prior information," which enables our system to require only a minimal amount of user input, from non-expert users who may have limited talent in this domain.

First, we develop a metric for automatically predicting the perceptual quality of images produced by state-of-the-art deblurring algorithms. The metric is learned based on a massive user study, incorporates features that capture common deblurring artifacts, and does not require access to the original images (i.e., "no-reference"). It better matches user-supplied rankings than previous approaches to measuring quality, and in most cases it outperforms conventional full-reference image-similarity measures.

Second, we investigate an approach to photo filtering in which the user provides one or more keywords, and the desired style is defined by the set of images returned by searching the web for those keywords. Our method clusters the returned images, allows the user to select a cluster, then stylizes the user's photos by transferring vignetting, color, and local contrast from that cluster. This approach vastly expands the range of available styles, and gives each filter a meaningful name by default.

Third, we describe a system that synthesizes novel pictograms by remixing portions of icons retrieved from a large online repository. Depending on the user's needs, the synthesis can be controlled by a number of interfaces ranging from sketch-based modeling and editing to fully-automatic hybrid generation and scribble-guided montage. Our system combines icon-specific algorithms for salient-region detection, shape matching, and multi-label graph-cut stitching to produce results in styles ranging from line drawings to solid shapes with interior structure.

1.1 Key Problems in Designing Data-Driven Systems

The development of a data-driven image creation and enhancement system starts by choosing a data source with relevant data. Mechanisms are designed to retrieve relevant data from the source. Then, algorithms are developed to extract the essence from the relevant data. Finally, the extracted essence is used to facilitate image creation or image enhancement.

We certainly expect the system to be cost effective. In other words, we expect it to require minimal user input, and generate results of high quality. To this end, we have to resolve three key problems.

Data retrieval. Relevant data are critical for data-driven algorithms to generate results of high quality. It is important to choose a good data source with high-quality relevant data, and design an efficient method to retrieve them. In Ch. 2, 3, and 4, we will show how we retrieve data from the Internet, and how we benefit from it.

Noise. Relevant data usually contain noise. For instance, crowd-sourcing user studies often have noise introduced by malicious users, who just randomly answer questions. The tags of photos indexed by the image search engines might be incorrect due to defects of automatic tag extraction algorithms. Without removing the noise, it is impossible to retrieve high-quality data. To extract the essence, we also have to filter out noise.

Disagreement in data. We often observe that using a large amount of data in a naive way leads to results of low quality. This is due to the disagreement in the

Internet data, caused by difference in contributors' opinions, display settings, camera settings, etc. For example, some photos captured in sunset take on an orange hue, while some others might be in purple. We have to deal with it carefully.

1.2 Online Services for Data Retrieval

In recent years, online services for data retrieval were built to facilitate users to contribute, share, and search for images, and to conduct subjective evaluation experiments. These services make the projects in this thesis possible. Here we introduce several typical examples.

Amazon Mechanical Turk and Crowd-Sourced Studies. Amazon Mechanical Turk (MTurk) `<https://www.mturk.com>` is a marketplace for crowd-sourcing tasks on the Internet. There are two roles on the market: a *requester* is able to post tasks such as 'choosing the better ones of pairs of images' on the market, while a *worker* (also named as *turker*) can choose and work on tasks, and earn money from the requesters. Amazon provides a complete set of API for customizing task interface, placing tasks, and retrieving turkers' submissions, which makes it a cost-effective platform for crowd-sourced studies.

The development of MTurk enables a number of large-scale user studies in recent years. For example, Cole et al. [18] study human perception of line drawings depicting 3D shapes based on a gauge-figure study. Chen et al. [14] collect 3D segmentation results from a user study, and analyze evaluation criteria based on them. Secord et al. [86] conduct a large study of viewpoint preferences and develop a model for evaluating views of 3D models. Xue et al. [106] learn criteria of good composite images from user data. These approaches adopt the strategy of learning perceptual models or criteria from a large-scale user study, then using them for producing or evaluating visual results in a way that tries to match human judgments. In Ch. 2 we apply this strategy to a new problem: motion deblurring.

Keyword-Indexed Online Image Repository. Online image repositories accommodate a huge number of images indexed by keywords. For example, Flickr `<https://www.flickr.com>` hosts billions of photos contributed by more than 90 million users worldwide. TheNounProject `<http://thenounproject.com>` is a database of nearly 100,000 curated pictograms contributed by a wide array of designers. On both websites, it is easy to use one or more keywords to search for photos or pictograms associated with the keywords. In this sense, online image search engines such as Google Image Search `<https://images.google.com>` and Bing Image Search `<http://www.bing.com/images>` can also be considered as keyword-indexed image repositories. All these four websites provide well-designed APIs to enable data-driven applications to build upon them.

In recent years, researchers combine the power of crowd-sourced studies and online image repositories to build highly-accurate large-scale databases. For example, ImageNet [20] is a database with more than 10 million images retrieved from Flickr and image search engines, indexed by more than 20,000 nouns. Turkers are hired to verify candidate images collected for each noun synset. Microsoft COCO [59], a large-scale data set with about 0.3 million images with about 2.5 million object annotations, retrieves images from multiple sources including Flickr, and collects about 10,000 work hours from MTurk to annotate objects on images.

The availability of Internet image repositories and databases also empowers large-scale data-driven analysis. Deselaers and Ferrari [21] have analyzed the relation between semantic and visual similarity of images on the ImageNet. Lindner et al. [60] analyze a million Flickr photos by comparing various features of the photos annotated with a specific keyword. Doersch et al. [22] specifically study similar local structure of images in a collection associated with a common geographical keyword *paris*. It is commonly observed that images with higher semantic similarity are also more visually similar. This inspires us to exploit the common visual properties to develop our keyword-based stylization system and iconification system in Ch. 3 and Ch. 4.

1.3 Contributions

We summarize the main contributions of this thesis as:

- A data-driven perceptually-validated metric for measuring the quality of image deblurring.
- Applications that utilizes our metric to automatically produced an improved deblurring result.
- A data-driven system that performs photo stylization based on arbitrary keywords.
- Algorithms for transferring vignetting, color, and contrast from an image *collection* to the user's photos.
- A unified data-driven framework which enables four different workflows to produce a variety of icons.

Chapter 2

Data-Driven No-Reference Quality Metric for Motion Deblurring[1]

2.1 Introduction

The wide availability and ever-increasing sophistication of modern image processing and computational photography algorithms has brought about a need to evaluate their results. For instance, for a task such as image deblurring, a realistic characterization of image quality and the presence or absence of artifacts is necessary to select between different methods, as well as to choose parameters for each algorithm. Lacking an automated method for image quality assessment, many systems resort to asking the user. This, however, becomes increasingly impractical if dozens of algorithms and hundreds of parameter settings must be compared. While a large-scale user study (using, for example, the Amazon Mechanical Turk) might be able to compare many combinations of algorithms and parameters, it would be unrealistic to use this methodology for *every* image that is processed. Finally, the lack of a high-quality "ground truth" image in most applications precludes the use of traditional metrics

[1]Results from this chapter were previously presented in SIGGRAPH Asia 2013 [63].

for image comparison, such as peak signal to noise ratio (PSNR), structural similarity index (SSIM), or visual information fidelity (VIF).

We explore a methodology in which image quality and artifacts are scored according to some function of features that are computed over the image. The function is learned based on thousands of training examples provided in a massive online user study. By picking the right features, and collecting enough user input, we are able to obtain a metric that generalizes over images and over the algorithms used to process them. We call this learned function a *perceptually-validated metric*, as it is not built upon the underlying psycho-physical mechanisms of the human visual system, but rather can score image quality and artifacts consistently with human ratings.

In this chapter, we address the problem of motion deblurring or blind deconvolution: undoing the (unknown) motion blur introduced by camera shake. This is a problem that has been studied intensively over the past several years [24, 108, 87, 15, 50, 103, 105, 16, 56, 112, 33], but even state-of-the-art methods may produce significant artifacts such as noise and ringing. In surprisingly many cases, these algorithms may produce even worse results than the input blurry image. This makes it crucial to identify which methods are performing well on which images.

The main contribution of this chapter is a perceptually-validated metric for evaluating the output of deblurring algorithms. Our key hypothesis is that an image quality metric specialized to this problem will outperform general metrics, or ones specialized to different problems. We therefore conduct a crowd-sourced user study (Ch. 2.3), incorporating five deblurring algorithms applied to hundreds of images, to learn the relative importance of the principal artifacts of blind image deconvolution: ringing, noise, and residual blur. We design features (Ch. 2.4) to measure these artifacts, and use them to obtain a mapping from a feature vector to image quality (Ch. 2.5) that matches the users' rankings as closely as possible. We find that in most cases this *no-reference* metric (i.e., not having access to the ground-truth image, which is important for real-world applications) outperforms existing no-reference image quality metrics, as well as standard *full-reference* image comparison algorithms.

Our user study yields interesting conclusions about the relative importance of different artifacts. For example, we find that large-scale "ringing" is overwhelmingly harmful to perceived image quality, to a much greater extent than noise and blur. These kinds of conclusions may have influence on the design of future algorithms for deconvolution, or even other image processing tasks. The resulting metric also enables applications (Ch. 2.6) including automatic selection of the best deblurring algorithm for a given image, automatic parameter selection, and fusion of the highest-quality regions of different deblurring results.

We summarize our contributions as:

- We learn a perceptually-validated metric for measuring the quality of image deblurring.
- We provide to the community a data set with specially designed input images, current state-of-the-art deblurring algorithms' results, and users' feedback about their quality.
- We show applications that utilize our metric to automatically produce an improved deblurring result.
- We analyze the impact of each artifact on perceptual quality, which can guide the direction of future work and development of deblurring algorithms.

2.2 Background and Related Work

In addition to the related work in Ch. 1.2, we introduce some background knowledge and other related work here.

Image deblurring. Motion blur caused by camera shake is a common problem observed in photos captured by hand-held cameras. A blurred image b may be modeled as

$$b = l * k + n, \tag{2.1}$$

where l is a sharp latent image, k is a point spread function (PSF), or a blur kernel, reflecting the trajectory of the camera shake, and n is noise. Deblurring is the problem of solving for a sharp latent image l, given a blurred image b. If k is known, the problem is called *non-blind deconvolution*; otherwise it is called *blind deconvolution*.

Even non-blind deconvolution is an ill-posed problem, since the PSF usually contains null frequencies and the noise level is unknown. To resolve the ambiguity, different types of prior knowledge are used. Levin et al. [57] propose a natural image prior, which approximates the heavy-tailed distribution of image gradients. Joshi et al. [43] assume that within each small patch, colors should be linear combinations of two colors. Zoran and Weiss [112] exploit patch priors trained on natural images.

Blind deconvolution (with unknown PSF) is even more ill-posed, requiring applying strong priors on both the latent image and the PSF. Fergus et al. [24] assume that the gradients of natural images follow a heavy-tailed distribution and the PSF has a sparse support, and adopt a variational Bayesian approach to estimate a PSF. Levin et al. [56] assume similar properties on the gradients of natural images, but use an expectation-maximization (EM) method for optimization. Several recent approaches [42, 15, 105, 16] rely on extracting edge profiles for PSF estimation, under the assumption that edges are sharp in natural images. Shan et al. [87] exploit the sparsity on both the latent image and the PSF, and also a local smoothness prior to reduce ringing artifacts. Goldstein and Fattal [33] exploit the power-law dependence of the power spectra of natural images for PSF estimation.

Despite the tremendous progress that has been made in recent years, state-of-the-art deblurring methods still tend to generate noticeable visual artifacts on real-world data, such as ringing, noise, residual blur, etc. These artifacts may arise because: (1) the priors used in existing methods are simplified approximations to natural image statistics, thus may lead to estimation errors in the PSF and the recovered image; (2) non-linear response curves and non-additive noise in real-world images violate the linear blur model in Eqn. 2.1; and (3) real-world images often contain spatially-varying blur that may not be well approximated by a static PSF. There has been work on estimating spatially-varying PSFs [103, 34, 41]; however, as the dimensionality of the PSF increases, these methods are often less reliable and may produce more severe artifacts, as shown in a recent study [48]. Schuler et al. [85] detect spatially-varying PSFs for images with optical aberrations. However, their method strongly depends on the symmetry properties of optical aberrations, thus is not general enough for handling motion blur.

Image quality metrics. Image quality metrics can be classified into two categories: full-reference and no-reference, depending on whether the ground truth image is needed. Commonly used full-reference metrics include PSNR [96], multi-scale SSIM [101], VIF [90], HDR-VDP-2 [66], and a linear combination of them [68]. Although these metrics have been widely used for evaluating subtle image corruption, they are not able to measure the perceptual impact of the significant artifacts introduced by a variety of graphics and vision algorithms, such as image deblurring, as we will demonstrate later, and photo-realistic rendering, as demonstrated by Cadik et al. [10].

On the other hand, many no-reference metrics, such as BIQI [72], BLIINDS [83], CORNIA [107], LBIQ [95], and BRISQUE [71], use either supervised or unsupervised learning on a collection of images and their subjective scores to produce a general image quality metric. NIQE [70] uses unsupervised learning without subjective scores. However, unlike our metric, these metrics are not designed for the specific problem of image deblurring, which generates unique artifacts such as strong ringing.

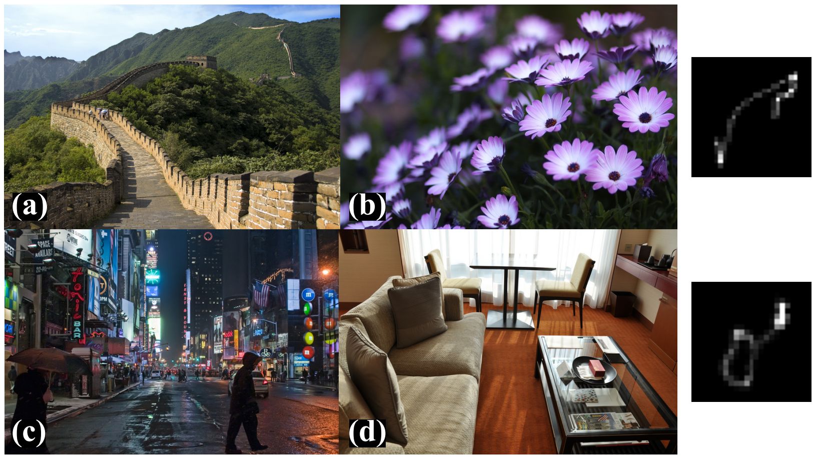


Figure 2.1: Left: four ground truth images used in our user study, which contains 40 images in total. Right: two PSFs used in the study. Image courtesy (a) Craig Maccubbin, (b) Jun Seita, (c) Bob Jagendorf, (d) uggboy@Flickr.

Image quality metrics have been successfully applied in a variety of applications, such as automatic parameter selection [110, 69], blur-aware and noise-aware downsizing [84, 97], and coded aperture design [68]. We will discuss the difference between these methods and our approach in more detail in Ch. 2.4.2 and 2.6.1.

2.3 Data Sets and User Studies

To understand how human beings perceive different deblurring artifacts, we first conduct a massive crowd-sourced user study to evaluate the perceptual quality of deblurring results. Here we describe the data set and the user study in more detail, then discuss some major observations we draw from the user study results, which serve as guidelines for our feature design in Ch. 2.4.

2.3.1 The Data Set

Our data set consists of synthetically motion-blurred images and the deblurring results generated by different algorithms. The deblurring results contain a large variation in quality: we intend to include good results as well as ones that have significant artifacts of various kinds.

Specifically, we begin with 40 sharp, high quality images of various scenes as the ground truth, which cover a wide variety of common scenes, such as landscape, cityscape, portrait, and indoor scenes. As several deblurring algorithms rely on extracting edges in the images, we also include images with different amounts of structural edges to have various levels of deblurring difficulty. Fig. 2.1 shows four of them. We downsample them so that their longest edges are 768 pixels. We then synthetically blur them with two different PSFs shown in Fig. 2.1, whose sizes are 27×27 and 23×23. In practice, the first PSF is more difficult to deal with and is more likely to cause ringing artifacts. The second PSF usually leads to relatively good results with more subtle artifacts. They lead existing methods to produce results with different levels of artifacts. Finally, we add Gaussian noise of three different levels ($\sigma = 0.0, 0.01, 0.02$) to each blurred image, resulting in $40 \times 2 \times 3 = 240$ blurred examples in total.

We run five recent algorithms [24, 87, 15, 56, 33] with publicly-available executables on all the blurred images to generate deblurring results. In doing so, we encountered some program failures (roughly 0.08% of all test cases) due to the instability of the research prototypes. For each sharp image, we obtain at most 30 deblurring results in this way. We call the collection of each sharp image and its deblurring results a *data group*.

2.3.2 The User Study

We employ the Amazon Mechanical Turk (MTurk) for the crowd-sourced user study, in which we ask users to compare and rank the quality of deblurring results.

There are a few options for constructing such a study. The most straightforward approach would be to ask each user to give a score for each deblurring result. However, absolute scores are subjective and inconsistent across users. Another choice would be to show all results at the same time and ask the user to rank all of them. This would solve the inconsistency issue, but the large number of images in each data group would make this task tedious for users.

To avoid these problems, in our study we ask users to compare the quality of deblurring results *pairwise*. For each comparison a pair of images is shown side-by-side, and the user is requested to choose the one that has better visual quality (thus using a forced-choice methodology). Each user session consists of 40 pairs of images in total. Once all the pairwise comparison results are obtained, we use them to fit a global ranking, as described in detail in Ch. 2.3.3.

To ensure the quality of the user study results, we only allow the users who have worked on more than 100 tasks with higher than 95% acceptance rate to participate in our study. Our user study also includes a mechanism to detect and reject "bad" results produced by malicious or careless users. For each data group, we include the ground-truth sharp image in the user study. Specifically, among the 40 comparisons a user performs in one session, 13 of them include ground-truth images, which are considerably better than most deblurring results. We reject results from users who ranked the ground truth images lower than the deblurring results more than twice. Under this design, the probability that randomly selecting an image in each pair will pass the test is 1.12%, which means that we can effectively reject outliers in the data.

In our study we collected $13,592$ user sessions from $1,041$ users, and 4% of them were rejected by the sanity check. In the remaining data, each pair was ranked by at least 20 different users.

Reasoning for crowd-sourced study instead of lab study. In contrast to controlled lab user studies, our crowd-sourced study has variation in display settings, which may de-emphasize certain image differences. If our goal were to study the fundamental properties of the human visual system, an experiment under controlled lab conditions would be appropriate. However, our goal is to develop a practical metric, useful in the real world, for quickly assessing and improving the quality of digital photos. We therefore exploit the variation in display settings among MTurk users, which is more representative of the variation found in the real world, by repeating each pairwise comparison dozens of times. This helps our metric be more robust and avoids overfitting to "perfect" lab conditions that do not generalize to the real world.

2.3.3 Fitting a Global Ranking

The problem of fitting pairwise comparison results to a global ranking has been well studied. We adopt the Bradley-Terry model [8], which is widely used for this purpose. The Bradley-Terry model generates a global "score" for each data point from the pairwise comparison. Here we briefly review how this model works. For each pair of images A and B, assuming their scores are δ_A and δ_B, let $\delta_{AB} = \delta_A - \delta_B$. We can then define the relation between the probability p_{AB} that one user chooses A over B, and the score difference δ_{AB} as:

$$p_{AB} = \frac{e^{\delta_{AB}}}{1 + e^{\delta_{AB}}} = \text{logit}^{-1}(\delta_{AB}), \tag{2.2}$$

where $\text{logit}(p) = \log\big(p/(1-p)\big)$. Here we see that:

$$p_{AB} + p_{BA} = \frac{e^{\delta_{AB}}}{1 + e^{\delta_{AB}}} + \frac{e^{\delta_{BA}}}{1 + e^{\delta_{BA}}} = \frac{e^{\delta_{AB}}}{1 + e^{\delta_{AB}}} + \frac{1}{1 + e^{\delta_{AB}}} = 1. \tag{2.3}$$

In the user study, since multiple users have compared A and B, we denote the number of users who favor A as a, and the number of users who favor B as b. Assuming the decisions are independent, a should follow a binomial distribution $\text{Bin}(a+b, a, p_{AB})$. Therefore, the likelihood of (δ_A, δ_B) is:

$$L(\delta_A, \delta_B) = P(a, b \,|\, \delta_{AB}) = \binom{a+b}{a} {p_{AB}}^{a} \, (1 - p_{AB})^{b}. \tag{2.4}$$

The likelihood for all pairs (A, B) is:

$$L = \prod_{(A,B)} \binom{a+b}{a} {p_{AB}}^{a} (1 - p_{AB})^{b}, \tag{2.5}$$

which is a function of δ_{AB}. Note that L only encodes the *difference* between scores of images. In other words, L does not change if the same offset Δ is added to all δ. To resolve this ambiguity, we need a reference image R with $\delta_R = 0$. In our user study, we let all ground truth images be the reference images. We then solve for δ of all images by maximizing L.

2.3.4 Observations

Based on the user study results, we make the following qualitative observations that provide useful insights for developing our metric:

- The main artifacts that appear in deblurring results include noise, ringing, and blurriness, and these artifacts typically co-exist in a deblurred image.
- In general users are very sensitive to ringing artifacts, which unlike noise and blurriness do not exist in natural images. The lowest-ranked images often contain strong ringing artifacts.
- The characteristics of the noise introduced by different deblurring algorithms can be very different.

Fig. 2.2 shows an example of deblurring results from one data group.

We use these observations as general guidelines for designing low-level image features used to train the perceptually-validated metric, as detailed in the next section.

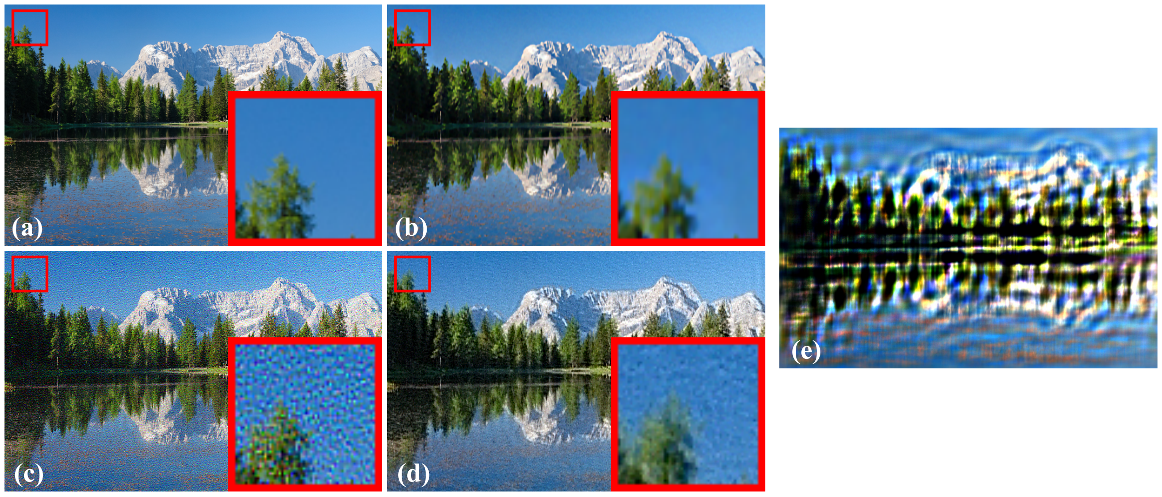


Figure 2.2: Sample deblurring results from one data group. (a) ground truth; (b) a deblurring result with blur; (c, d) two deblurring results with the same noisy input but different deblurring algorithms, yielding different types of noise in the results; (e) the lowest-ranked deblurring result, which contains strong ringing. Please refer to the supplementary materials for the ranking of the full data group. Original image courtesy Umberto Salvagnin.

2.4 The Collection of Features

We now derive a set of low-level image features for assessing the quality of deblurred images. These features serve as the basis for learning our metric. In general, our expectations for good deblurring results are twofold: (1) *naturalness*: the deblurred image should have a natural appearance; and (2) *sharpness*: the deblurred image should be sharp and have as little residual blur as possible. We design a collection of features to measure how well a deblurring result meets each constraint. Note that not all features will be eventually used in the perceptually-validated metric, and we will discuss how to select good features in Ch. 2.5.4.

2.4.1 Measuring Image Naturalness

The naturalness of a deblurring result describes the extent to which it looks like a natural image captured by a camera. While it is hard to measure naturalness directly, it is easier to describe artifacts that often appear in deblurring results making them look unnatural. We identify two dominant artifacts, which are common in deblurring results — *noise* and *ringing* — and design our feature set with a various methods to measure them quantitatively.

Noise

There exists a vast amount of previous work on estimating and removing image noise. Nevertheless, accurate noise estimation remains a challenging problem. Given that different noise estimation methods may work for different types of noise (e.g. chromatic noise vs. luminance noise), we use multiple measures to assess the amount of noise in a deblurred image. We briefly describe each measure below. We refer the readers to Ch. A.1.2 for details.

- **Two-color priors** [43] assume that, in natural images, colors within a local image neighborhood are linear combinations of a primary and secondary color. Following [43], we find the two prevalent colors for each local neighborhood, and measure the noise of each pixel based on the two-color model.

- **Sparsity priors (Sps)** [57] assume that gradient magnitudes of natural images follow a heavy-tailed distribution. We measure the norm of gradient magnitudes.

- **Gradients of small magnitudes (SmallGrad)** usually correspond to noise on flat regions. We use the variance of the top $m\%$ of smallest gradient magnitudes as a noise measure.

- **MetricQ** [109] is a no-reference metric sensitive to both noise and blur, which is based on the fact that noise and blur make an anisotropic patch more isotropic.

- **BM3D** [19] is one of the state-of-the-art denoising methods. It takes a parameter σ that specifies the noise level in the input image. We estimate the optimal

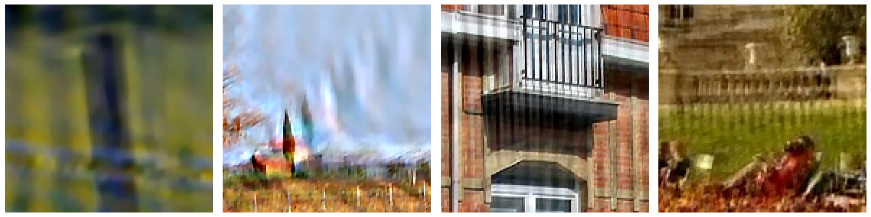


Figure 2.3: Typical ringing artifacts in deblurred images.

value of σ and include it in our noise feature set. Specifically, we apply BM3D to the image with different values of σ, and measure the errors of the results using a two-color prior. We choose the smallest σ, whose error is smaller than a certain threshold, as a noise measure.

Ringing

Ringing is perhaps the most common artifact one may observe in a deblurred image. It often appears as un-natural wavy structures parallel to sharp edges (Fig. 2.3). Strong ringing artifacts may span large image regions, or even the entire image. They are mainly caused by inaccurate PSF estimation, but even with an accurate PSF, they can still appear due to the Gibbs phenomenon [108].

Since we have observed that ringing is one of the most annoying artifacts in deblurring (Ch. 2.3.4), detecting and quantifying ringing is crucial for assessing the quality of a deblurred image. Unfortunately, detecting ringing artifacts is not easy, as ringing is a mid-frequency signal, which is mixed together with the image signal in the same frequency band. It is especially hard to detect ringing in textured regions where rich mid-frequency image signal exists. Methods have been proposed for measuring ringing artifacts [67], but most of them are only applicable to small scale ringing caused by lossy compression such as JPEG compression, and/or are full-reference metrics.

We introduce a new method to measure large-scale ringing caused by motion deblurring. We design our method to be conservative: it can reliably estimate ringing in flat regions, but is more conservative in textured regions to avoid misclassifying image structures as ringing, which will significantly damage the image quality assessment. It is also consistent with human perception, as ringing artifacts are more noticeable in smooth regions (Fig. 2.3).

Full-reference ringing detection. We first describe a full-reference method, which forms the basis for the no-reference method described later. The method begins with a deblurred image l' and the ground truth l, and applies the following steps:

1. We first align l and l' and compute the gradient maps of l and l' as $g(l)$ and $g(l')$.
2. We compute the difference map $\delta g = \max(g(l') - g(l) * k_g, 0)$. k_g is a scaled Gaussian function with a maximum value of 1, which is convolved with $g(l)$ to avoid misclassifying residual blur as ringing artifacts. We take $\max(\cdot)$ because most ringing artifacts in l' should have larger gradients than l;
3. Then, we compute the average of δg as a measure of ringing artifacts in l'.

No-reference ringing detection (PyrRing). For no-reference detection, we substitute l with the input blurry image b. As we apply a low-pass filter in step 1, the error caused by this substitution is not significant. To further mitigate the error, we create an image pyramid for both b and l', and measure the ringing artifacts on each level of the pyramid, then compute the average across all levels as the final indicator of ringing artifacts. At coarse levels, the downsized blurry input b will be closer to the downsized ground truth l, leading to more accurate ringing detection. Thus, the final ringing measurement computed from the pyramid is also more accurate than the one directly computed from the original l' and b. Fig. 2.4 shows an example of our ringing detection.

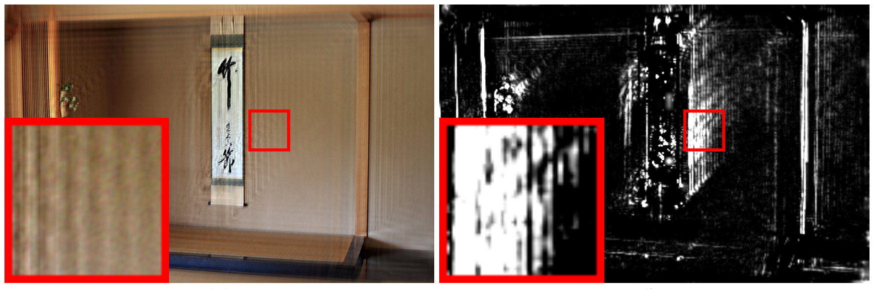


Figure 2.4: Measuring ringing artifacts. (a) Deblurred image. (b) Ringing feature response map. For visualization, we add up responses over all pyramid levels, and convert it into a grayscale image where pixel intensities indicate the strength of the response. Original image courtesy Tanaka Juuyoh.

Saturation. The ringing artifacts in images are usually accompanied with overshoot and undershoot artifacts. We measure the proportion of pixels with pixel value below 10 or above 245 (assuming all pixel values are between 0 and 255) as the saturation feature.

2.4.2 Sharpness

An ideal deblurred image should be sharp and contain no residual blur. On the other hand, an inaccurate PSF or too-strong regularization in deconvolution may cause remaining blur in a deblurring result. We measure the blurriness of a deblurring result with the following state-of-the-art sharpness measures from recent work:

- **Autocorrelation (AutoCorr)** of image derivatives can be an effective way to measure the remaining blur, since it is close to the autocorrelation of the blur kernel due to the power law on the power spectrum of natural images [25, 33]. Unfortunately, it is known to be vulnerable to long straight edges. We detect long straight edges using the Hough transform, and mask them out from image derivatives before computing autocorrelation. We measure how much the autocorrelation map is spread out and use it as a sharpness measure. Please refer to Ch. A.1.4 for details.

- **Cumulative Probability of Blur Detection (CPBD)** is a no-reference sharpness measure proposed by [74]. It defines sharp edges based on the notion of "just noticeable blur", and measures the proportion of sharp edges as a sharpness measure.

- **Local Phase Coherence (LPC)** [35] is a no-reference sharpness measure based on the coherence of local phase information across different scales.

- **Normalized Sparsity Measure (NormSps)** [49] is a measure that favors sharp images to blurry ones, and was originally proposed for blind deconvolution.

Note that some previous approaches [84, 97] also measure sharpness. However, Samadani et al. [84] use the shape of the PSF as its prior knowledge, and only Gaussian function is tested. Similarly, Trentacoste et al. [97] make a strong assumption that the PSF is a Gaussian function. In our application, we do not know the shape of the PSF, which could be arbitrary. The PSFs estimated by different delurring algorithms from the same input image are usually different, thus none of their shapes can be trusted. Therefore, previous methods assuming known (typically Gaussian) PSF shape are not suitable for our application.

2.5 The Perceptually-validated Metric

In this section, we provide performance analysis on the features defined in Ch.2.4, and show that any single feature cannot cover all kinds of artifacts that can appear in deblurring results. Then, we train a perceptually-validated metric based on the user study results (Ch.2.3) and a combination of features, and provide its performance analysis. For performance analysis on each feature and our trained metric, we first begin by describing our evaluation method.

2.5.1 Evaluation Method

To evaluate a feature and our metric, we adopt the idea of ranking comparisons to evaluate how well a feature or a metric can distinguish the relative quality difference between different deblurring results. Specifically, we compute the feature/metric scores f for all images in the data set, and then rank the images based on f. We then compare this ranking with the "ground truth" score δ. δ is generated by fitting a Bradley-Terry model to the user data (Ch.2.3.3) for all data sets in our experiment, except for the synthetic single-distortion image data sets (Ch. 2.5.2). If these two rankings correlate well, then the feature in question is a good stand-in for the perceived quality of a deblurred image.

Two widely used methods for comparing ranking results are Spearman's rank correlation [93] and Kendall τ distance [47]. However, these methods are not suitable in our application, because they do not consider two important factors. First, the distance between two adjacent images in the ground truth ranking is not uniform. Therefore, reversing a pair with a larger distance is worse than reversing a pair having a smaller distance. Second, accurate ranking among relatively good results is more important than ranking among bad ones for real applications, because bad deblurring results are typically so bad that the exact ranking among them is meaningless.

We thus propose a new ranking metric, named *weighted Kendall τ distance*, for performance evaluation. We first define a set $D_{(\delta,f)}$ of pairs (i,j), whose orders by δ and f do not agree, i.e., $(\delta_i - \delta_j)(f_i - f_j) < 0$. Kendall τ distance is defined as the cardinality of $D_{(\delta,f)}$. As this distance is solely defined based on the cardinality, it has the problems mentioned above. To overcome those problems, we define a weighted distance as:

$$M(\delta, f) = \sum_{(i,j)\in D_{(\delta,f)}} \left|\big(\max(\delta_i, \delta_j) - \delta_{\min}\big)(\delta_i - \delta_j)\right|, \tag{2.6}$$

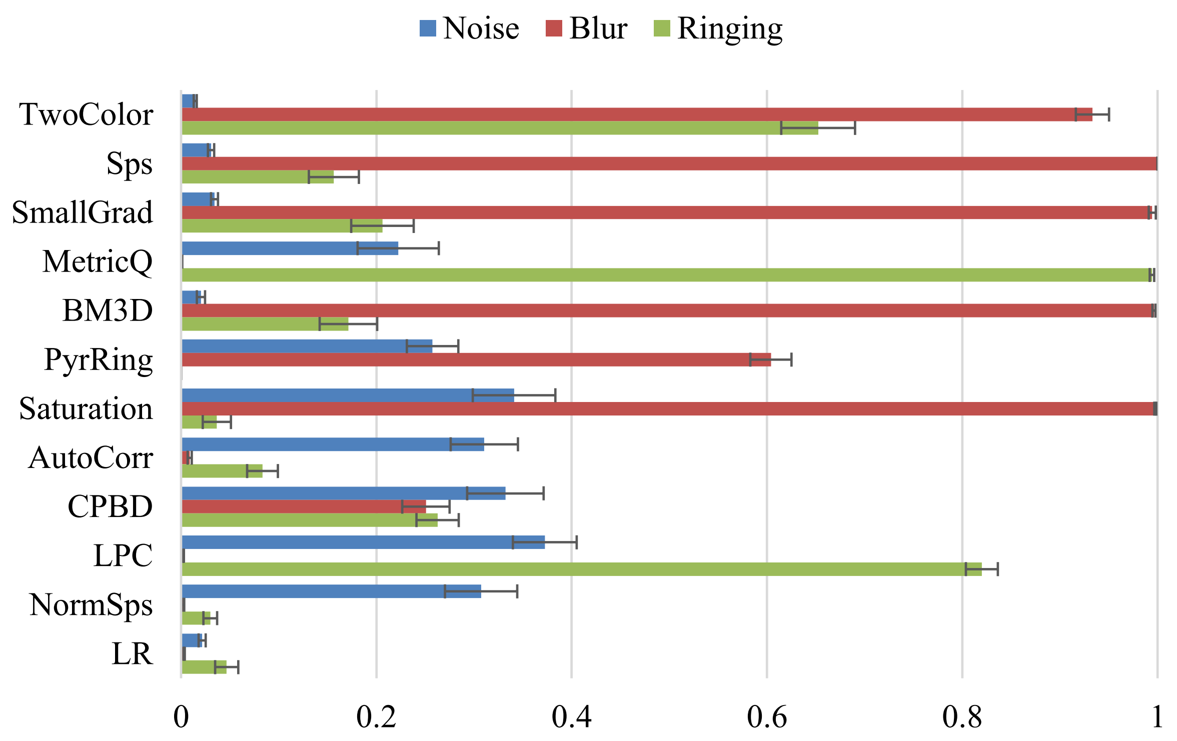


Figure 2.5: The mean weighted Kendall τ distance from each individual feature and our LR metric to the ground truth. Lower is better. The error bars indicate the standard error of the distance.

where $\delta_{\min}$ is the worst B-T score. For comparison with other features whose scores have different scales, we use a normalized version of Eqn. 2.6, which is defined as $\overline{M}(\delta, f) = M(\delta, f)/M(\delta, -\delta)$, where $M(\delta, -\delta)$ is the maximum mismatch generated by comparing against the exact opposite ranking.

The two factors in Eqn. 2.6 are the key difference between weighted Kendall τ distance and the conventional Kendall τ distance. The first factor $(\max(\delta_i, \delta_j) - \delta_{\min})$ in Eqn. 2.6 is larger for the highly-ranked images in the B-T model, placing an emphasis on them relative to images with worse rankings. The second factor $(\delta_i - \delta_j)$ measures mis-ranking between the target feature and the B-T model. Note that if we simply add up 1 for all pairs (i, j) in $D_{(\delta,f)}$ instead of these two factors, Eqn. 2.6 becomes the Kendall τ distance.

2.5.2 Evaluating Features on Single-Distortion Images

Based on the evaluation method proposed above, we first evaluate the usefulness of each feature on different kinds of artifacts. We design three new data sets. Each one contains only one artifact among noise, blur and ringing.

- **Noise:** The noise data set consists of deblurred images with different noise levels. To generate deblurred images, we generated synthetically blurred images with different amounts of Gaussian noise, and applied different deblurring algorithms. The standard deviation of Gaussian noise varies from 0 to 0.04 with the step 0.004 (assuming that image intensities are normalized into $[0, 1]$). We used 16 original sharp images and five different deblurring algorithms to build a data set consisting of $16 \times 5 = 80$ data groups.

- **Blur:** We add synthetic motion blur with PSFs with the same pattern but different sizes onto a sharp image to build a data group. Specifically, each PSF is resized with ratio from 0.25 to 2.25 with the step 0.25. The images blurred with larger PSFs are more blurry. We use 16 sharp images, and eight original PSFs to build a data set consisting of $16 \times 8 = 128$ data groups.

- **Ringing:** Inaccurate PSFs often cause ringing artifacts. Thus, we generate a series of inaccurate PSFs by upsampling the true PSF. We found higher upsampling ratio yields more severe the ringing artifacts. Therefore, in each data group, we use a non-blind deblurring algorithm with Gaussian derivative prior to deblur images using increasingly upsampled versions of the true PSF. We use 16 sharp images and eight original PSFs to build a data set consisting of $16 \times 8 = 128$ data groups.

Since we intentionally create images with increasing amounts of artifacts in each group, here the ground truth score of the i-th image δ_i is specially defined as $\delta_i = i$ instead of conducting a user study.

We measure the quality of our features on these three data sets. Fig. 2.5 shows the mean weighted Kendall τ distance of each individual feature. We make the following observations:

- All noise features except MetricQ mis-classify blurry images as good images.
- Sharpness features work moderately well on the noise data set. However, MetricQ and LPC mis-classify images with ringing artifacts as good images.
- PyrRing works well on images with ringing artifacts. Saturation, AutoCorr, and NormSps also have good performance on the ringing data set.

In summary, all of these features perform quite well on at least one data set. However, none of them are able to work on all the three data sets just by themselves, motivating our approach (described in the next section) of using learning algorithms to combine features. We refer the readers to Ch. A.2.2 for additional evaluation of each feature on our user study data set.

2.5.3 Learning a Metric

To derive a quality metric, we first define a metric as a mapping function from a feature space $\mathbf{X}$ to a scalar in $[0, \infty)$. The feature space $\mathbf{X}$ is defined as the concatenation of all features defined earlier. Statistical regression methods are used to fit the mapping function $f(\mathbf{x})$ based the user study results.

In this work we use a commonly-used regression method: Logistic Regression (LR) [38]. LR is a well-known generalized linear regression method, which is also used for deriving scores for the Bradley-Terry model. Similarly to Ch. 2.3.3, suppose we have a pair of images (A, B), and there are n user study submissions for this pair with a submissions favoring image A over B and b submissions favoring B over A.

As in Ch.2.3.3, we can define the following logistic regression model:

$$p_{AB} = \text{logit}^{-1}(\boldsymbol{\gamma} \cdot (\mathbf{x}_A - \mathbf{x}_B)), \tag{2.7}$$

where $\mathbf{x}_A$ and $\mathbf{x}_B$ are feature vectors of images A and B, respectively, and $\boldsymbol{\gamma}$ is the parameter vector for logistic regression. Then, following a similar approach to Ch.2.3.3, we can derive the likelihoods for all pairs (A, B), which have the same form as Eqn. 2.5. The derived likelihoods are functions of $\boldsymbol{\gamma}$, and we can solve for $\boldsymbol{\gamma}$ with Maximum Likelihood Estimation.

Once $\boldsymbol{\gamma}$ is obtained, a metric $f(\mathbf{x})$ can be derived as follows:

$$f(\mathbf{x}) = \boldsymbol{\gamma} \cdot (\mathbf{x} - \mathbf{x}_O), \tag{2.8}$$

where $\mathbf{x}_O$ is the feature vector of the 'ideal' sharp image. Since $\boldsymbol{\gamma} \cdot \mathbf{x}_O$ is a constant, it can be omitted and the final form of $f(\mathbf{x})$ becomes:

$$f(\mathbf{x}) = \boldsymbol{\gamma} \cdot \mathbf{x}. \tag{2.9}$$

2.5.4 Feature Selection

Our collection of features described in Ch. 2.4 contains 11 different features. To train a perceptually-validated metric, the straightforward solution is to use all of the features, but this increases the chance of overfitting. In addition, features have redundancy between each other. We therefore design the following four cross-validation tests in order to select the optimal subset of features:

1. Divide the 40 data groups into five sets, in which each has eight data groups. For each set, use all images in this set for testing, and all images in all other

data groups for training. By rotating the testing set we get five cross-validation errors. The mean error is then recorded as the average performance.

2. Similar as Test 1, but only use images blurred by the first PSF for training, and images blurred by the second PSF for testing, and measure the cross-validation error.

3. Similar as Test 1, but only use images with the noise levels 0 and 0.02 for training, and images with the noise level 0.01 for testing, and measure the cross-validation error.

4. Similar as Test 1, but only use images with the first PSF and noise levels 0 and 0.02 for training, and images with the second PSF and noise level 0.01 for testing, and measure the cross-validation error.

In this way, Tests 2–4 perform validation not only on image sets that were omitted from training, but also on PSFs (Tests 2 and 4) and noise levels (Tests 3 and 4) that were not used during training. This provides a better measure of generalization than cross-validation over images alone, as in Test 1.

Since we have 11 different features, there are only $2^{11}-1=2047$ possible combinations of features. Therefore, we exhaustively search through all possible combinations, and select the one that generates the best average performance of the four tests.

In the end, this method selects the following features: *Sps*, *SmallGrad*, *MetricQ*; *PyrRing*, *Saturation*; *AutoCorr*, *CPBD*, *NormSps*. Table 2.1 demonstrates the weights of these features in our metric. Here the weights are scaled with the standard deviation of the features to reveal the importance of each feature. We have the following observations: First, all three types of single-distortion artifacts are included with non-trivial weights, which confirms our observation that they are all relevant to deblurred image quality. Second, the sharpness features have the highest overall weight, which is reasonable because this metric is for deblurring. Third, ringing fea-

Features	Sps	SmallGrad	MetricQ
Scaled Weights	0.7344	0.1774	0.4106
Features	NormSps	AutoCorr	CPBD
Scaled Weights	0.7998	1.9179	0.4722
Features	PyrRing	Saturation	
Scaled Weights	1.7671	0.2283	

Table 2.1: Scaled weights of the selected features.

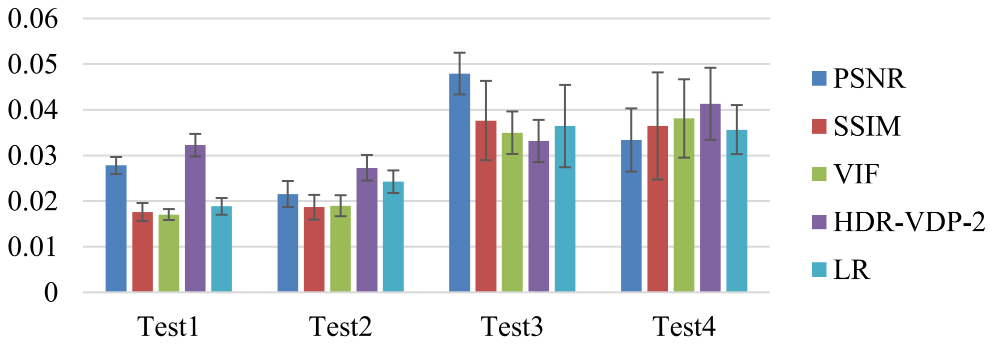


Figure 2.6: The weighted Kendall τ distance of different metrics on four different cross-validation test sets (see Ch. 2.5.4). The error bars indicate the standard error of the distance.

tures have a higher overall weight than noise features, which confirms our observation that users are sensitive to ringing artifacts.

Fig. 2.5 demonstrates that our LR metric performs very well on all three types of data sets. Fig. 2.6 compares the performance of our metrics trained with the optimal subset of features and three existing full-reference metrics (PSNR, multi-scale SSIM, VIF, and HDR-VDP-2), using the evaluation method described in Ch.2.5.1. Our LR metric is competitive with all full-reference metrics. Given that our metric is not only competitive with the existing ones but is also no-reference (i.e., does not require ground truth), we believe that its performance is promising for real-world applications, as shown in Ch.2.6.

2.5.5 Validation of User Study on Mechanical Turk

Because users on Amazon Mechanical Turk are unsupervised and come from all over the world, the noise in data can be considerably heavier than in controlled in-lab psychophysical experiments. To verify that our user study is robust to noise and variation of users, we analyze the consistency of our user study.

There are several ways to evaluate the consistency of a user study. A common approach might be to look at inter-subject variance. However, since paired comparison actually expects (and indeed relies on) disagreement on pairs of images that have similar quality, inter-observer variance is not applicable for this methodology.

To avoid this problem, we instead measure the inter-phase variance of our user study experiments. We repeat user study experiments of five out of the 40 data groups on Amazon Mechanical Turk, three months after the original experiments, and fit a new score δ' for each image in the data groups.

- Consider the original δ as the reference. The weighted Kendall τ distances of the new δ' of the five data groups are all below 1.21×10^{-3}.
- Consider the new δ' as the reference. The weighted Kendall τ distances of the original δ of five data groups are all below 1.35×10^{-3}.

These two results prove that our user study results are stable.

2.6 Applications and Results

We now demonstrate how our deblurring quality metric can be applied in different application scenarios.

2.6.1 Automatic Parameter Selection

In previous work image quality metrics have been applied to automatically select good parameter settings for image processing algorithms, such as image denoising [110, 69]. We demonstrate that our metric is helpful for parameter selection in a new problem: motion deblurring. We observe that there are two important parameters that almost all deblurring algorithms require: (1) the regularization strength for the final non-blind deconvolution step, which controls the trade-off between noise and residual blur in the final result; and (2) the maximum PSF size. We found that existing deblurring systems are sensitive to these two parameters, and there is no principled way to find good settings for them. Thus, previous deblurring methods often require extensive manual parameter tuning to generate good results.

To demonstrate that our metric can be used for automatic parameter selection, we conduct an experiment with six synthetic examples that are not included in our user study data set. The images are blurred with two PSFs that are also different from those used in the study. Gaussian noise with $\sigma = 0.01$ is added to each test image to form a parameter selection dataset. For this study we choose two algorithms:

1. The fast PSF estimation approach proposed by Cho and Lee [15] to estimate PSFs, and a non-blind deblurring method with a Gaussian prior for generating the final result, which is formulated as:

$$\underset{l}{\operatorname{argmin}} \left\{ \|b - l * k\|^2 + \lambda_C \left(\|f_x * l\|^2 + \|f_y * l\|^2 \right) \right\}, \tag{2.10}$$

   where the first term is a data fidelity term, and the second term is a regularization term. λ_C is a regularization weight, and f_x and f_y are first-order derivative filters along the x and y directions.

2. The PSF estimation approach proposed by Levin et al. [56], and with the non-blind deblurring method with a sparse derivative prior proposed by them:
$$\underset{l}{\operatorname{argmin}} \left\{ \|b - l * k\|^2 + \lambda_L \left(\sum_{i=1}^{2} |f_i * l|^{\alpha} + 0.25 \sum_{i=1}^{3} |g_i * l|^{\alpha} \right) \right\}, \tag{2.11}$$
where the f_1 and f_2 are first-order derivative filters, and g_1, g_2, and g_3 are second-order derivative filters. α is 0.8 to impose sparsity.

To select the PSF size, we try a sequence of different sizes from 11×11 to 61×61 with a step size of 5, while fixing the regularization strength $\lambda_C = 2^{-3}$ and $\lambda_L = 2^{-8}$, and use our metric to find the PSF size which results in the highest score. Secondly, we fix the selected PSF size, and apply a sequence of different λ_C values: $\{2^{-9}, 2^{-8}, \ldots, 2^{3}\}$ and λ_L values: $\{2^{-14}, 2^{-13}, \cdots, 2^{-2}\}$, and use our metric again to find the optimal value for λ.

To evaluate the parameter selection results, we conduct another pair-by-pair user study on the parameter selection data set, where each pair was ranked by at least 20 users. The Bradley-Terry model is then used to get a ground truth score for each image in this data set. We compare our Logistic Regression (LR) metrics with existing full-reference (PSNR, SSIM, VIF, and HDR-VDP-2), and a few state-of-the-art general-purpose no-reference image quality metrics, including CORNIA [107], BRISQUE [70] and NIQE [71]. Fig. 2.7 shows the mismatch scores of each method.

The following conclusions can be drawn from this experiment:

- For tuning λ, HDR-VDP-2 and our LR have the best performance. SSIM and the three no-reference metrics work reasonably well. PSNR and VIF perform worst.

- For tuning the maximum PSF size, all full-reference metrics and our LR have very good performance. However, the other no-reference metrics perform badly.

Overall, our LR achieves the best mean Kendall τ distance. We run a one-tailed paired t-test with confidence level 0.95 to compare our method against each other

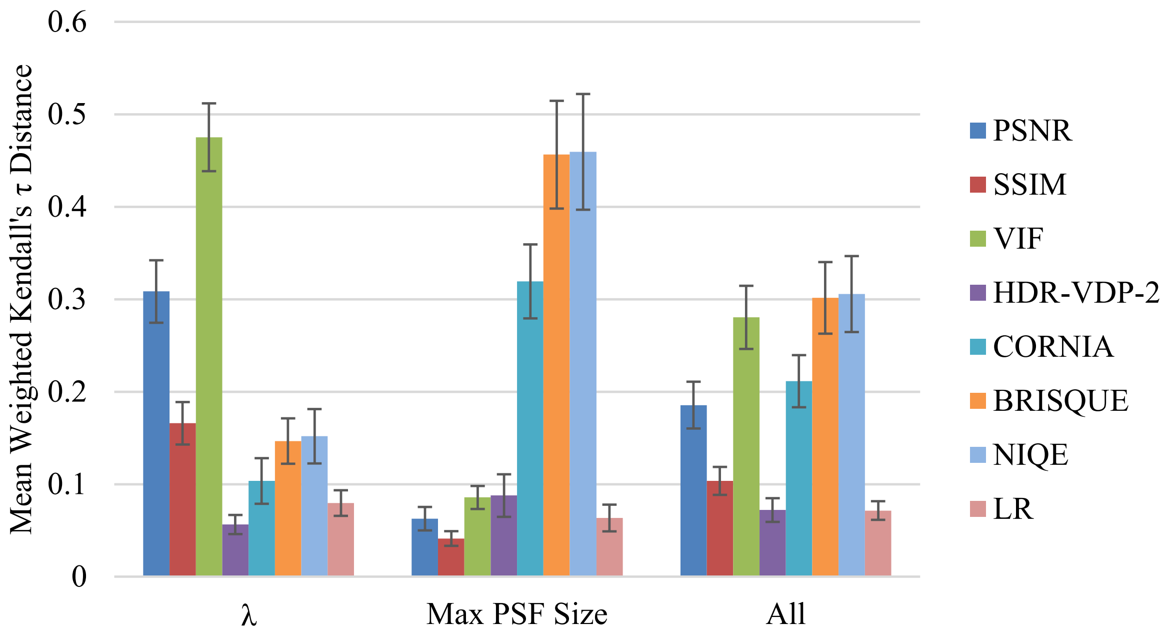


Figure 2.7: The mean weighted Kendall τ distance of full-reference (PSNR, SSIM, VIF, HDR-VDP-2) and no-reference (CORNIA, NIQE, BRISQUE, LR) metrics against the Bradley-Terry model on the data set of automatic parameter selection. Lower values indicate better performance.

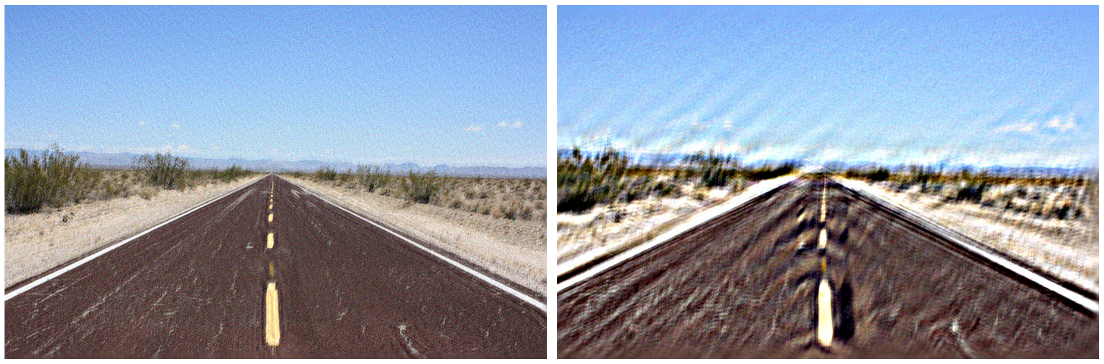


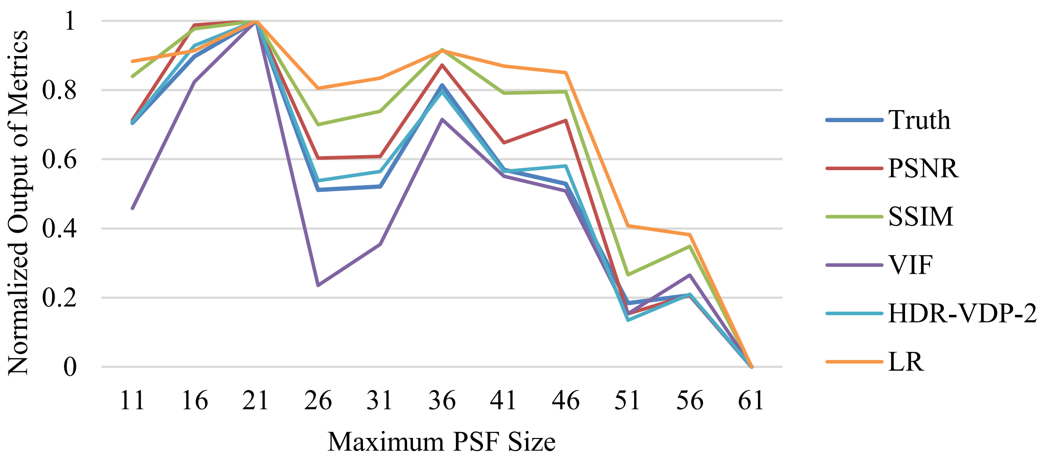
Figure 2.8: Deblurring results with max PSF size of 21 and 61. Original image courtesy Semio@Flickr.

method to check the statistical significance of its superiority. The p-value is below 0.05 for each metric except HDR-VDP-2, which suggests that our LR metric is statistically significantly better. The p-value for HDR-VDP-2 is 0.4864, which indicates that our LR metric is statistically comparative with it. Note that HDR-VDP-2 is a *full-reference* metric, while ours is *no-reference*.

In order to better understand why our LR metric outperforms other metrics in general but not in λ tuning, we provide the following case studies. The full set of images and their rankings can be found in the supplementary materials.

Case study 1. The test image "road" is challenging for PSF estimation, because there are too few strong edges that PSF estimation methods typically rely on. Inaccurate PSF estimation will in turn introduce strong ringing artifacts into the deblurring results, as shown in Fig. 2.8. In particular, more severe ringing artifacts can be observed with larger PSF sizes. In Fig. 2.9 we compare the full-reference and no-reference metrics with the ground truth, which is obtained by the Bradley-Terry model. Here we normalize the outputs of all metrics to make them lie in $[0, 1]$. It shows that the full-reference metrics and our LR metric yield nearly the same trend as the ground truth, indicating a correct evaluation of the perceptual quality of the results. In contrast, the results of other no-reference metrics such as CORNIA, NIQE, and BRISQUE present large divergence from the ground truth, indicating poor performance. This is because the artifacts in deblurred images, particularly the residual motion blur and the very severe ringing artifacts, are quite different from artifacts in general-purpose image quality assessment data sets, thus the no-reference metrics trained from general data sets cannot work well on deblurred images.

Case study 2. On the other hand, when it comes to the trade-off between a little amount of residual blur and noise (i.e. the selection of λ), the problem is quite similar to general-purpose image quality assessment, as there are very few deblurring-specific artifacts involved. In this case, the general no-reference metrics perform much better, as shown in the comparison in Fig. 2.10. The peak of the truth curve indicates perceptually the best result. Images to its left are too noisy, and those to its right are too blurry. It shows that NIQE and our LR metric best match the left part of

(a)


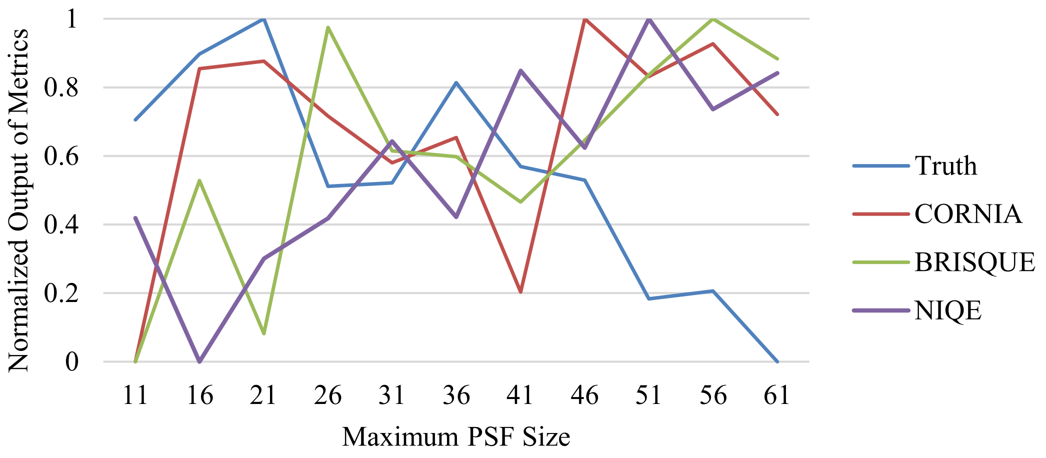
(b)


Figure 2.9: Comparing performance of different metrics in Case 1. Closer to the ground truth curve means better performance.

the truth curve; however, for the right part of the truth curve, we observe that all metrics favor blurry images slightly too much, including our metric.

2.6.2 Algorithm Selection

Given a blurry input image, we can apply different algorithms to deblur it, and then use our metric to automatically choose the best result. To demonstrate this we test our metric on the recently constructed motion blur data set proposed by Köhler et al. [48], which contains real motion-blurred images and their corresponding sharp latent images, as well as deblurring results from eight different algorithms. There are four scenes in this data set, and each is blurred with 12 camera motion trajectories

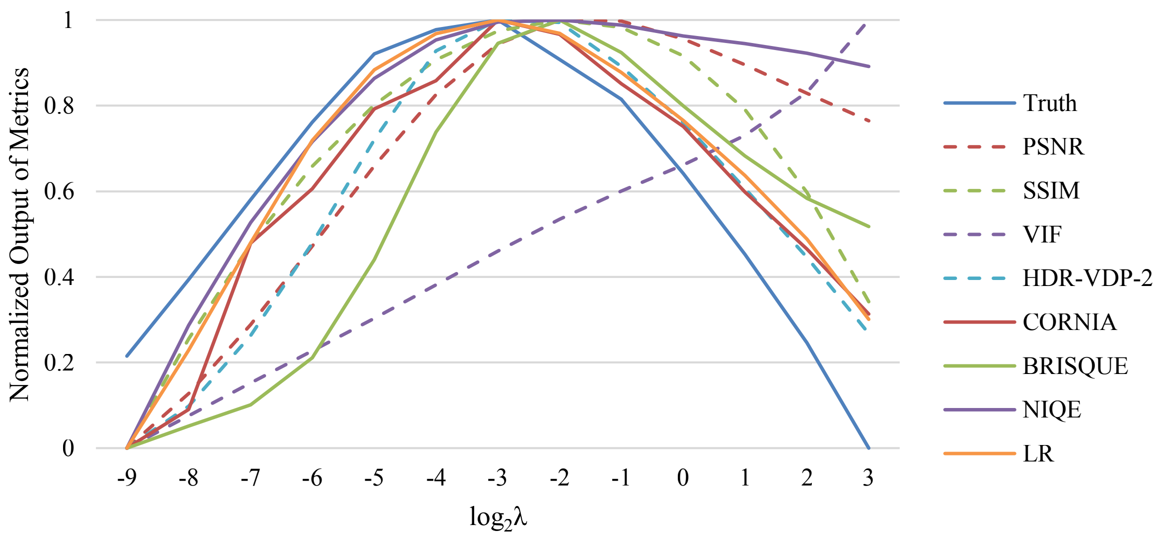


Figure 2.10: Comparing performance of different metrics in Case 2.

controlled by a robot. The 3rd trajectory is trivial, and all algorithms achieve good results. The 8th, 9th, and 10th trajectories are so large that no algorithm can generate reasonable results. We omit these four trajectories. We also omit the results of [24], because they consistently contain saturated pixel values in the blue channel, possibly due to an improper parameter setting. In total we thus have 32 data groups, each deblurred by seven different algorithms.

We applied our metric to rank the images in each data group. To evaluate the results, we conduct another pair-by-pair user study on this data set, where each pair was ranked by at least 20 users. The Bradley-Terry model is then used to get a ground truth score for each image in this data set. Fig. 2.11 shows the mean weighted Kendall τ distance from all metrics to the Bradley-Terry model. For this application, our LR metric performs slightly better than all full-reference metrics except SSIM, and significantly better than all the other no-reference metrics. We run a one-tailed paired t-test with confidence level 0.95 to compare our method against each other method. The p-values are 0.1159, 0.0340, and 0.1068 when we compare LR to the full-reference metrics PSNR, VIF, and HDR-VDP-2. The p-values are all below 10^{-4} when we compare LR to all other no-reference metrics. In Fig. 2.12 we show the

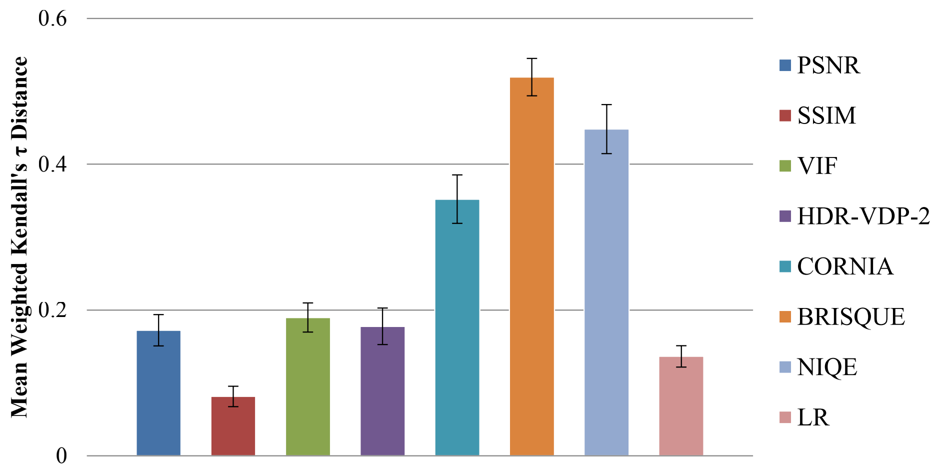


Figure 2.11: The mean weighted Kendall τ distance of full-reference (PSNR, SSIM, VIF, HDR-VDP-2) and no-reference (CORNIA, NIQE, BRISQUE, LR) metrics against the Bradley-Terry model on the data set of algorithm selection. Lower values indicate better performance.

result of one data group. The ranking generated by our LR metric matches well with the ranking given by the Bradley-Terry model, particularly for top three images.

2.6.3 Image Fusion

In addition to selecting the single best-performing algorithm for an input image, we can go further and use our metric to select the best local regions from multiple deblurred images and fuse them into a deblurred image that is better than any one result. This can also be used to handle spatially-varying blur. Specifically, we could estimate multiple PSFs from different local regions of an input image, and use them to generate multiple deblurring results. Our metric is then used to automatically find good regions in different versions and stitch them together to form the best possible result.

We conduct an experiment to validate this idea. First, we create several images with spatially-varying blur, using the tools provided by Whyte et al. [103], and add Gaussian noise with $\sigma = 0.001$ to them. Fig. 2.13(a) shows one of these examples. We design a simple algorithm to deblur these images. First, we divide the image

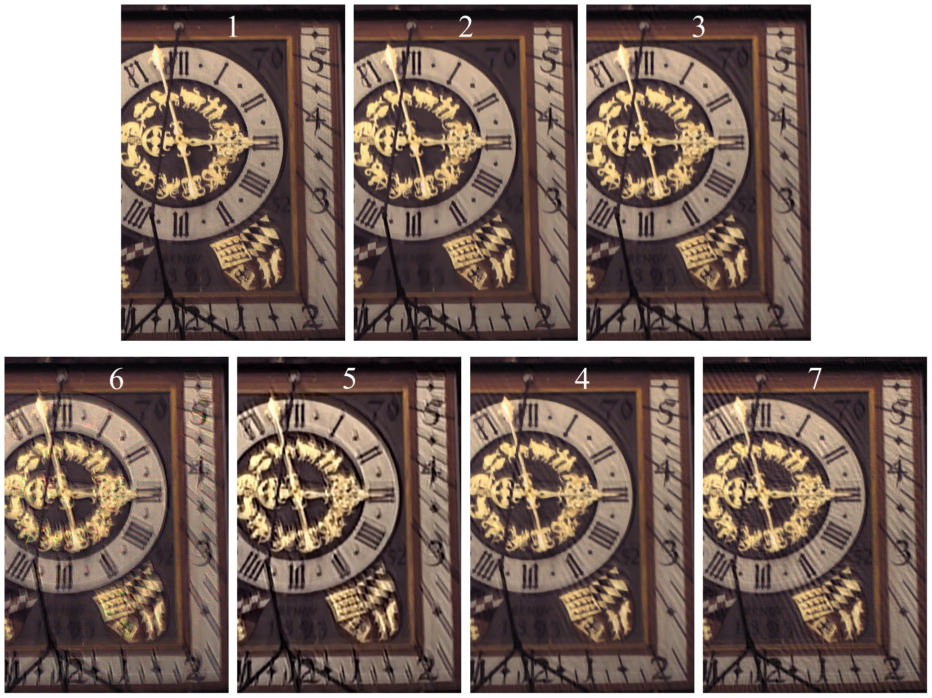


Figure 2.12: Using our metric for algorithm selection. The images are sorted by our LR metric with decreasing quality from left to right. The white numbers on top of images are the ranking given by the Bradley-Terry model.

into a 3×3 grid of regions, and estimate a blur kernel in each region using the kernel estimation method in [15]. We then use the non-blind deconvolution method from the same approach to recover a latent image from each kernel, resulting in nine deblurred images in total. Each deblurred image has some good regions, but also has severe artifacts in other areas.

To fuse these results, we first align all images using normalized cross correlation. We then apply our metric on small regions surrounding each pixel, yielding an evaluation score for each pixel in each image. We divide each image into overlapping patches with 11×11 pixels, and for each patch, we select the one from all results with the highest mean metric values. The selected patches are stitched together using graph cuts and Poisson blending [77] to generate the fused result.

Fig. 2.13 and Fig. 2.14 show two examples of fusing multiple deblurred images. We compare our results with two methods. In the first method, we compute the

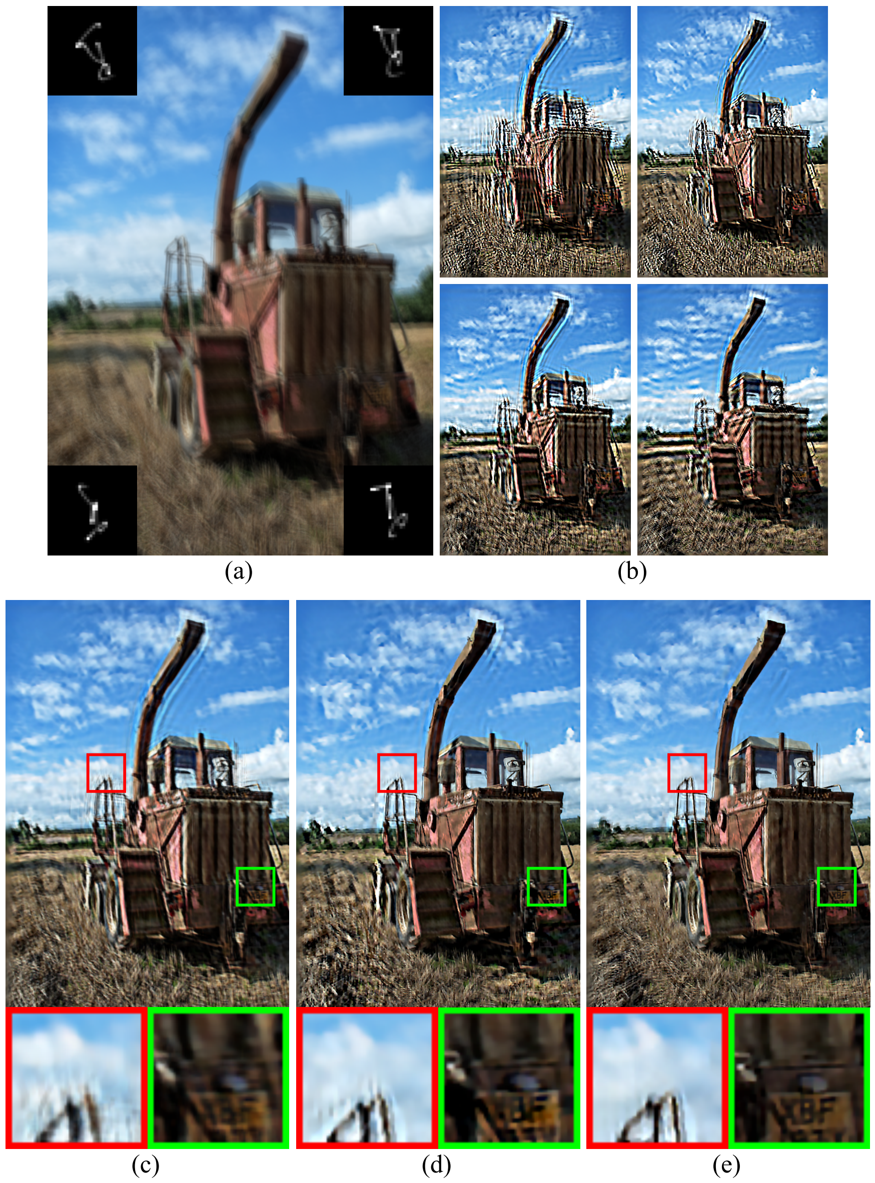
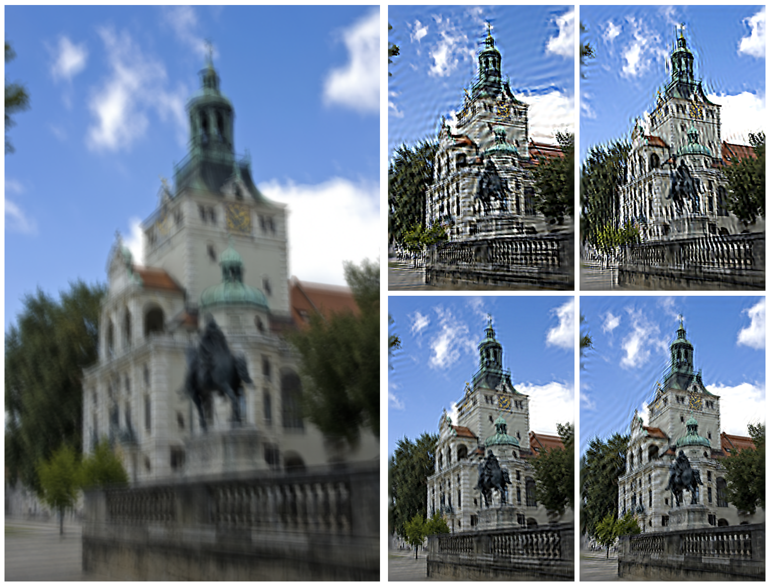
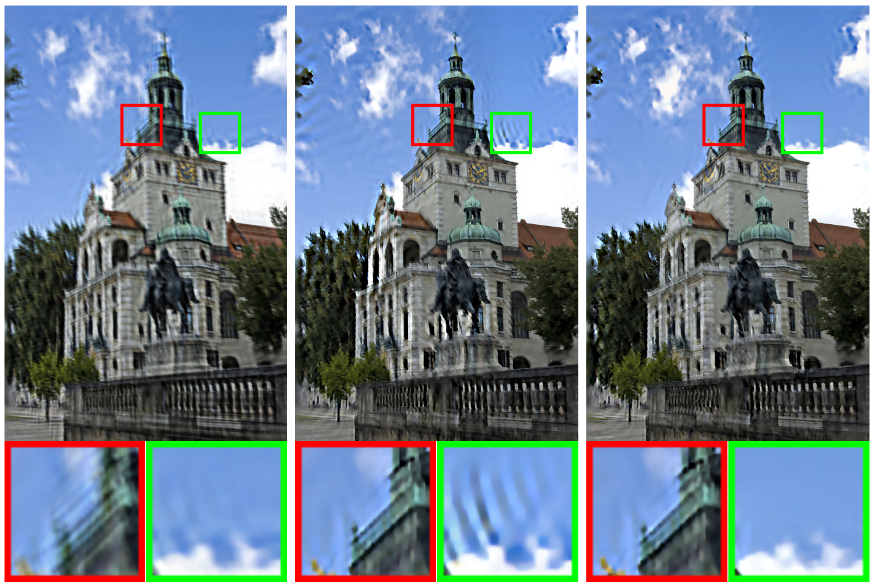


Figure 2.13: Using our metric to fuse multiple deblurring results from an image which contains spatially-varying blur. (a) The input image with spatially varying blur, as well as the PSFs at the four corners. (b) Four out of the nine deblurring results produced with the deblurring algorithm in [15] that has the assumption of spatially invariant blur. (c) the result by simply averaging all deblurring results. (d) result by the naive fusion method (see Ch. 2.6.3). (e) our fusion result. Original image courtesy Alex Brown.



(a) (b)



(c) (d) (e)

Figure 2.14: Another example of image fusion. Original image courtesy digital cat@Flickr.

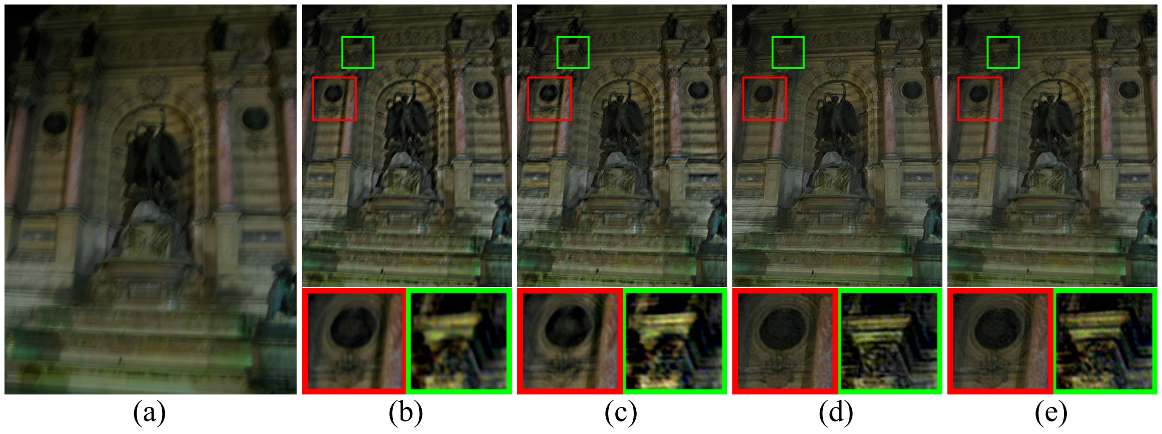


Figure 2.15: Using our metric to fuse multiple deblurring results of a real-captured spatially-varying blurry photo from [103]. (a) The input blurry image. (b) the result by simply averaging all deblurring results. (c) result by the naive fusion method. (d) result of [103]. (e) our fusion result.

average of all images, which is helpful for reducing artifacts that are uncorrelated in each deblurred image. For the second method, in each deblurred image we pick the same region that the algorithm uses to estimate the blur kernel, and combine them to obtain a naive fusion result. As shown in these examples, our results contain fewer artifacts and have better visual quality compared with results generated from these methods, as well as the individual images. Fig. 2.15 shows our fusion result on a real-world spatially-varying blurry photo from [103]. Our result is comparable to the result of the spatially varying deblurring algorithm [103]. Note that in this test case [103] uses an additional sharp noisy image as auxiliary information, while ours does not.

For a better comparison, we conduct a user study on a collection of 19 synthetic images with spatially-varying blur. Users were asked to do pairwise comparison as we described in Ch. 2.3. Each pair of images is compared at least 37 times. The proportion of users who favor our fusion results over that of the simple averaging method and the naive fusion method are 802/828 (96.86%) and 644/838 (76.85%), respectively. The proportion of users who favor the naive fusion results over the simple averaging results is 724/818 (88.51%). A randomized permutation test on the distributions of users' preferences shows the statistical significance ($p \ll 0.01$) of all these results. Please refer to the supplementary materials for the collection of images used in our user study.

2.7 Conclusion and Discussion

In this chapter, we have demonstrated a perceptually-validated no-reference metric for evaluating the quality of image deblurring results. To achieve this, we conduct a user study to collect users' evaluations on the visual quality of deblurred images. By studying user data, we identify the three most common deblurring artifacts, and design a collection of features for measuring them. We further show how to select an optimal subset of features and use them to train a metric. Extensive evaluation shows that this outperforms state-of-the-art no-reference metrics, and matches or outperforms full-reference metrics, for evaluating deblurred images. Finally, we demonstrate that our metric can be used for improving image deblurring by parameter selection, algorithm selection, and fusion of multiple results.

The quality metric in this chapter is developed for evaluating the quality of motion deblurring results. The quality scores generated by this metric are supposed to be comparable only across deblurring results of the same input motion blurred image. Therefore, subjective factors such as content, composition, and lighting are not supposed to be evaluated. Because of this nature, we choose to collect data by conducting a one-time user study, and train a metric which is able to generalize to all photos in real-world display settings. To ensure the generalizability in photos, we carefully design a data set with wide coverage on scenes. To ensure the generalizability in display settings, we conduct our crowd-sourced study on Amazon Mechanical Turk instead of in lab.

On Amazon Mechanical Turk, there are intentional noise introduced by malicious users, and unintentional noise caused by users' occasional careless clicks. In our design, we put substantial effort in dealing with noise. We filter out the intentional noise by 1) prefiltering users on their records, and 2) performing sanity checks on their submissions. We reduce the impact of the unintentional noise with obtaining multiple submissions for each pair of images. Users' opinions are obtained on every pair of images, in which the redundancy makes our metric robust to noise. We formulate our metric with a linear model, which also helps avoid overfitting.

Users' opinions often disagree, particularly for pairs of images of similar quality. Simply throwing them away or downweighting them yield a big loss of collected data. Instead, we choose to use the Bradley-Terry model, in which the disagreement in users' opinions are exploited to measure similarity between quality of images. The difference in display settings in our crowd-sourced study might also cause disagreement, but this type of disagreement in fact helps our metric generalize to the real-world display settings. In conclusion, our metric benefits from disagreement in the data.

Chapter 3

Data-Driven Keyword-Based Image Stylization[1]

3.1 Introduction

Stylizing images by manipulating color and contrast is a common task in digital post-processing of photos. In recent years, software packages such as Adobe Photoshop Lightroom [1] and Apple Aperture [3] have provided the ability to adjust color and contrast, but users still have to manually drag sliders to tune hue, tint, saturation, contrast, sharpness, vignetting, etc. An alternative approach is provided by popular online photo sharing services such as Instagram [23], which offer a set of pre-designed filters to automatically stylize photos. However, since their filters are all hand-crafted by artists, users have few choices. Also, most Instagram filters do not have intuitive names. (For instance, Instagram's *toaster* filter is actually named after the dog of the CEO's friend.)

In this chapter, we explore a data-driven methodology to make it much easier to stylize photos. Our insight is that a *collection* of photos with the same keyword often

[1]Results from this chapter were previously presented in EGSR 2014 [62].

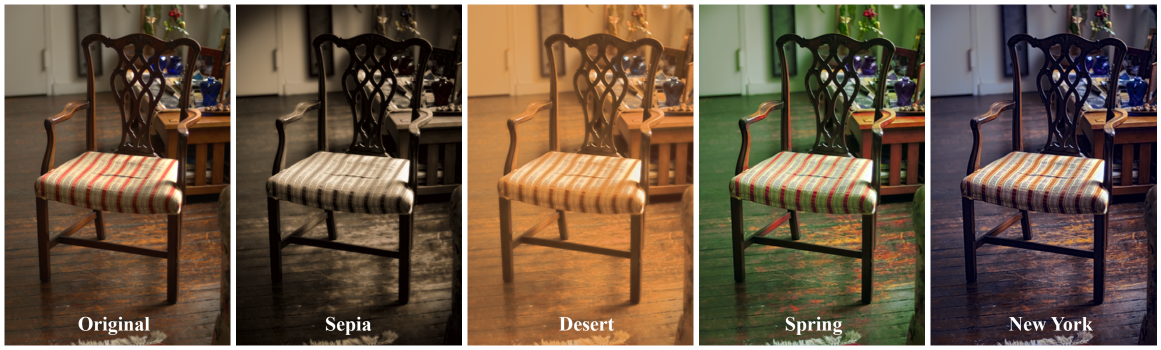


Figure 3.1: Our system stylizes a user's photo by transferring style from a collection of images returned by a web search for a particular keyword. Examples: *sepia* (tone of old photos, reduced local contrast), *desert* (enhanced orange color, reduced global and local contrast), *spring* (enhanced green color, enhanced saturation, enhanced local contrast), *New York* (enhanced blue/violet tone, enhanced local contrast).

share the same style. For instance, photos with the keyword *grass* are mostly green and have a lot of sharp edges, while photos with the keyword *night* usually are dark, and have a low local contrast. Therefore, in addition to the user's input photo I_s, we ask the user for a keyword K. The collection of images returned by searching the web for the keyword K serves as the data source for our style transfer algorithm.

The advantages of such an approach are four-fold. First, the ability to use arbitrary keywords to define styles provides users with a greater variety of possible stylizations than any pre-built set of filters. Second, a *collection* of images is robust to outlier photos, i.e., photos that are returned by the search but not really related to the keyword. Next, a collection of images instead of a single image has more chance to capture the *common* properties of a style, preventing the algorithm from over-fitting to a single image. Finally, the style generated by our system uses K as its name, which is intuitive for the user.

As Adobe Lightroom and Instagram filters do, we identify the *vignetting*, *color*, and *contrast* as the properties of a *style*. We have found that a single keyword often relates to multiple different settings of properties. For instance, *sunset* photos may have orange, blue, or violet hue. Therefore, we divide the fetched photo collection into different clusters (Ch. 3.3.1). Very small clusters are eliminated to make our system more robust to outliers. The final choice among the remaining clusters is left to the user.

Our style transfer algorithm begins with vignetting (Ch. 3.3.2). Instead of trying to find the physically correct vignetting layer, we focus on extracting a vignetting layer that captures the radially symmetric layout of lightness in the images. In the second stage, we transfer color from a cluster of images to the user's photo (Ch. 3.3.3). To do so we match the *3D* color histogram from a single image to a *collection* of images. We address this by modeling it as a transportation problem, and computing the color transform for each histogram bin by matching the lightness channel and a 2D elliptical distribution over the chrominance channels. Finally, we transfer local contrast by matching the fine-scale levels of Laplacian pyramids (Ch. 3.3.4). Some sample results of this process are shown in Fig. 3.1.

We summarize our contributions as follows:

- We design a system that performs photo stylization based on arbitrary keywords.
- We propose algorithms for transferring vignetting, color, and contrast from an image *collection* to the user's photos.

3.2 Related Work

Interactive Photo Enhancement. PixelTone [53] enables users to enhance photos via touch and voice. Shapira et al. [89] propose an interactive system for users to navigate the color transfer space. Cohen-Or et al. [17] develop a system to improve the color harmony in a photo. Though all of these systems are helpful for photo enhancement, they do not approach the problem as one of style *transfer*. Instead,

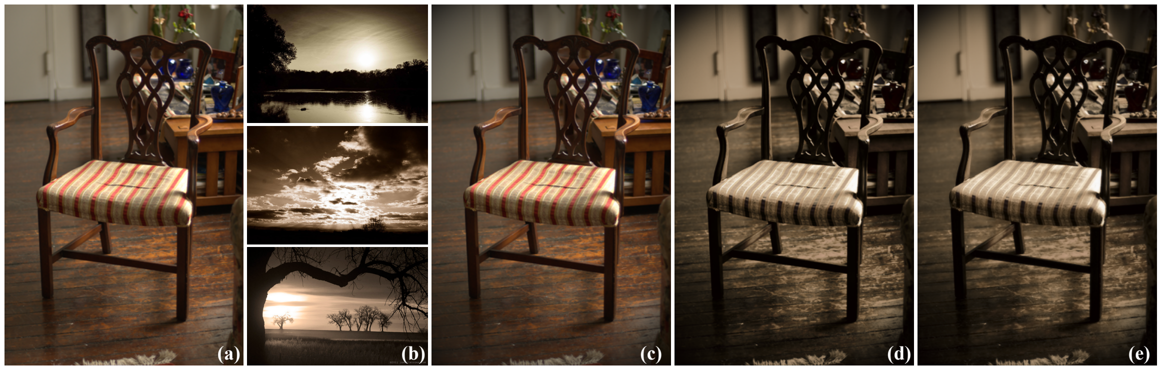


Figure 3.2: An overview of our system. (a) user's photo; (b) three representative images in one cluster of the web images with the keyword *sepia*; (c) after vignetting transfer; (d) after color transfer; (e) after local contrast transfer.

users must imagine a target style, and use the capabilities of the systems to achieve that style. In contrast, in our system, the user just has to provide a keyword.

Style Analysis for Image Collections. The work most closely related to ours is by Lindner et al. [60]. They also propose to analyze the style of photos by comparing various features of the photos annotated with a specific keyword. They analyze a million general photos using a Mann-Whitney-Wilcoxon test to assess features particularly associated with that keyword, and transfer the recovered "style". Our stylization system in Ch. 3 has four key differences from this work. First, we do not rely on a fixed corpus of photos, but rather leverage the full power of internet image search engines. Second, Lindner et al. assume a single style for each keyword, which we show to be invalid for many keywords. For these keywords, Lindner et al. either fails, or is only able to produce an oversmoothed "average" of all styles related to that keyword. Neither meets users' expectations. In contrast, our method identifies distinct styles for the same keyword through simple and effective clustering. Third, in addition to color and global contrast, our method transfers vignetting and local contrast, which are key components of a visual style. Fourth, we contribute a collection-based multi-dimensional histogram matching algorithm that unifies the color transfer and local contrast transfer under the same framework.

Color Transfer. There are two types of color transfer algorithms: color theme based and histogram based. The color theme based methods extract a small number of the most representative colors from the image as an abstraction of the color distribution in the image. Reinhard et al. [80] and Bonneel et al. [6] transfer color between images by matching the mean and variance of colors in the CIELAB color space. Murray et al. [73] group colors with a convex clustering algorithm, and compute the color transfer by solving a minimum-cost max-flow problem. Wang et al. [99] model the relationship between texture and color based on the images fetched from the Internet to avoid unnatural color transfer. These abstraction of images often result in just a very small number of colors, ignoring much of the information.

The histogram based methods describe color distribution with histograms, which are more precise when the number of bins is large enough. Pouli and Reinhard [79] progressively match histograms in a multi-scale manner. Pitie et al. [78] progresively match two 3D color histograms via matching 1D marginal distributions. When the source and target histograms are highly incompatible, histogram-based methods may overfit, which causes artifacts. To resolve this issue, Freedman and Kisilev [27] relax the conservation constraints for the count of pixels in each bin of histograms, and compute the transfer for each histogram bin with the mean and variance of pixel values in the bin. This is a compromise between the color theme based methods and the histogram based methods. Our method is similar to that of Freedman and Kisilev [27], but with one key difference: we take advantage of the range of counts in each histogram bin of the image collection to further avoid overfitting.

Shapira et al. [88] is also a collection-based histogram matching algorithm, but it is limited to 1D histograms, which are unable to model the correlation between color channels.

Laffont et al. [52] propose an intrinsic image decomposition algorithm that relies on a collection of photos *captured at the same place*, with an application on relighting.

Though relighting is able to mimic color transfer styles that is closely related to changes in lighting, it is difficult to be generalized to a wider spectrum of styles.

Contrast Transfer. Contrast transfer is mostly done in a multi-scale way. Bae et al. [4] decompose the image into a base layer and a detail layer with bilateral filtering, and transfer contrast via histogram matching on each layer. Sunkavalli et al. [94] decompose the image into a over-sampled Haar pyramid, and match the statistics via histogram matching with gain control [58]. Our method follows the same line, while we choose to decompose the image into a four-level over-sampled Laplacian pyramid because of its simplicity and high efficiency.

Example-Based Effect Transfer. Example-based effect transfer algorithms begin with original / enhanced image pairs, and transfer their effects to the users' photos. Image analogies [37] transfers the effects of filters with texture synthesis. Kang et al. [46] and Caicedo et al. [11] transfer parameters for generic image enhancement from a set of exemplars to users' photos. Wang et al. [100] and Bychkovsky et al. [9] transfer the color and tone style based on a set of before-and-after enhancement pairs. They all require pixel-aligned pairs of original (without the style) and enhanced (with the style) images, while our method only requires images with the desired style. Shih et al. [92] transfer the visual appearance at a different time of day to users' photo based on time-lapse videos, which essentially contain pixel-aligned image pairs.

3.3 Style Transfer

Fig. 3.2 illustrates how our system stylizes an input photo I_s (Fig. 3.2a). We begin by asking the user for a keyword K and passing K to Bing Image Search to retrieve a collection of 500 images. We group these photos into clusters containing images with similar contrast distribution and spatial layout of colors. The user selects one of these clusters to indicate the desired stylization. Fig. 3.2b shows three representative images in one cluster.

We transfer the style in the CIELab color space. The a and b channels encode the color. The L channel is decomposed into a four-level oversampled Laplacian pyramid L_1, L_2, L_3, L_4 (i.e., none of the levels are downsampled). The coarse-scale level L_4 encodes global contrast, while the finer-scale levels contains most of the high-frequency information in the image, thus encoding local contrast. For convenience, we let $L^{(L)}$ denote $L_1 + L_2 + L_3$.

To transfer style from an image cluster to the user's photo I_s, we extract a vignetting layer from the L_1 channels of the images in the cluster, and multiply I_s with it to obtain the vignetting transfer result I_v (Fig. 3.2c). We then match the statistics of the cluster's 3D color histogram to I_v, to obtain the color transfer result I_c (Fig. 3.2d). Finally, we transfer the local contrast by matching the statistics of L_1, L_2, and L_3 to obtain the final result I_l (Fig. 3.2e).

3.3.1 Image Collection

The image collection is built from the top search results on Bing Image Search. After excluding invalid images and duplicates, we take the top 500 images. The downloaded images are then resized to make their longest sides have the same size as the user's photo.

Although images with the same keyword tend to be visually similar, there are often multiple distributions of color and contrast. For instance, in Fig. 3.3a and 3.3b, we see two groups of sunset images with high similarity within each group, but quite different tones between the groups. In Fig. 3.3c and 3.3d we see two clusters of beach images, in which the first group does not have any green color, while the second has substantial green. Other image search results may display an even wider variation in styles: we observed that searching for "cloud" returns images of both clouds in the

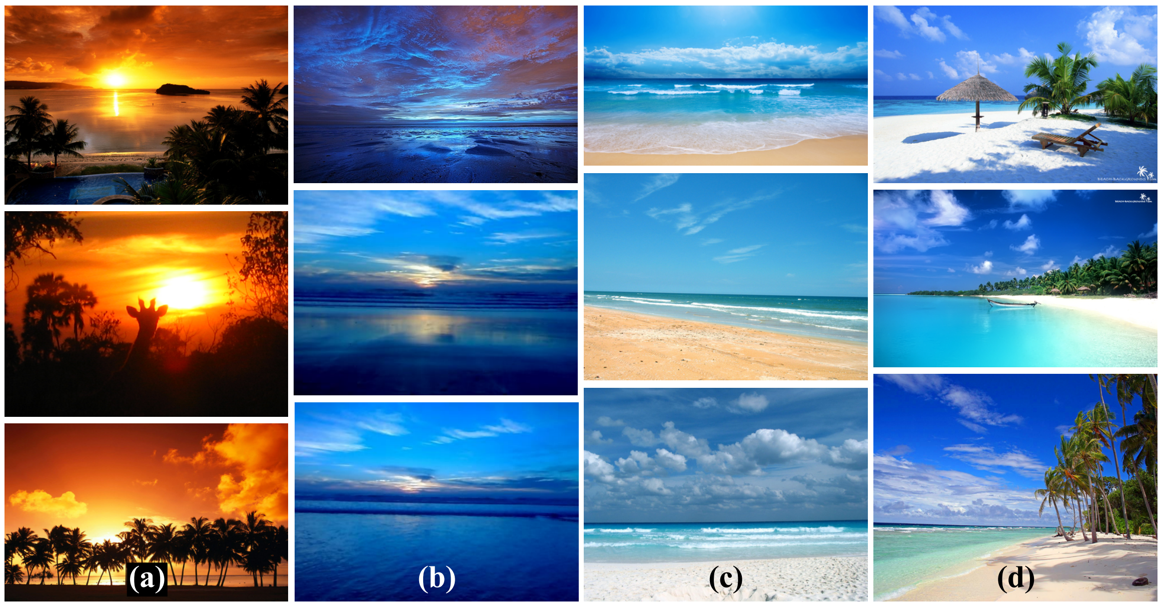


Figure 3.3: (a, b) Two groups of images with the keyword *sunset*. (c, d) Two groups of images with the keyword *beach*.

sky and a popular video game character. In any case, there is not a unique mapping from keyword to visual style, and we need additional information to determine *which* of the styles returned by the keyword the user wishes to use.

Therefore, we group the downloaded image collection via k-means clustering. We represent each image with a combination of a chrominance thumbnail and gist features [75]. The chrominance thumbnail is computed by averaging the CIELab a and b channels down to 8×8 spatial resolution, which captures the color information. The gist feature is built by aggregating 6 oriented edge responses at 4 scales at 4×4 spatial resolution, which captures the contrast information. The spatial information present in both features is helpful for the discovery of effective vignetting structure later. The two features are concatenated with the same weight. To be more robust to outliers, the k-means clustering is done with an L_1 distance metric. Small clusters (with fewer than five images) are removed, as they are likely to be outliers. In practice, we find that using $k = 20$ (before small-cluster removal) strikes a good balance between intra-class variation and the size of the clusters. The clusters are sorted by their average L_1 distance from each instance to the cluster centroid. As a result, clusters with small intra-cluster variation are ranked higher.

We have also experimented with mean-shift clustering. We find that it generally leads to results with similar quality while requiring significantly more computation.

In our system, each cluster yields a unique style transfer result. For instance, the *sunset* filter might produce results tinted orange, blue, or violet, and the user is able to choose among them. In comparison, the method of Lindner et al. [60] only produces one result.

3.3.2 Vignetting Transfer

Vignetting is the attenuation of light at the image periphery, which often has the effect of drawing attention to the objects in the image center. In addition to vignetting caused by the imaging system, which we observe for keywords such as *vintage*, we notice that images with keywords such as *sunset*, *horror*, and *flowers* often exhibit a similar effect: they have a visually outstanding object in the image center, with darker pixels in the periphery. To capture both real and "effective" vignetting, therefore, we do not focus on recovering a physically correct vignetting layer, but rather aim simply to characterize the radial distribution of luminance.

We make several assumptions about the vignetting layer. First, we assume that the layer is radially symmetric; i.e., it can be represented by a function $V(r)$, where r is the distance from a pixel to the center of the image and $r = 0$ and $r = 1$ stand for the center and the corners of the image, respectively. Second, we assume that vignetting darkens the image gradually from the center to the border of the image; i.e., $V(r)$ is a non-increasing function.

To find the vignetting function, we first find a function $V_i(r)$ for every image in the cluster. For each image i, we first normalize its L_1 by setting the 10th and 90th percentiles to 0 and 1, respectively. For each possible radius r, we collect all pixels with

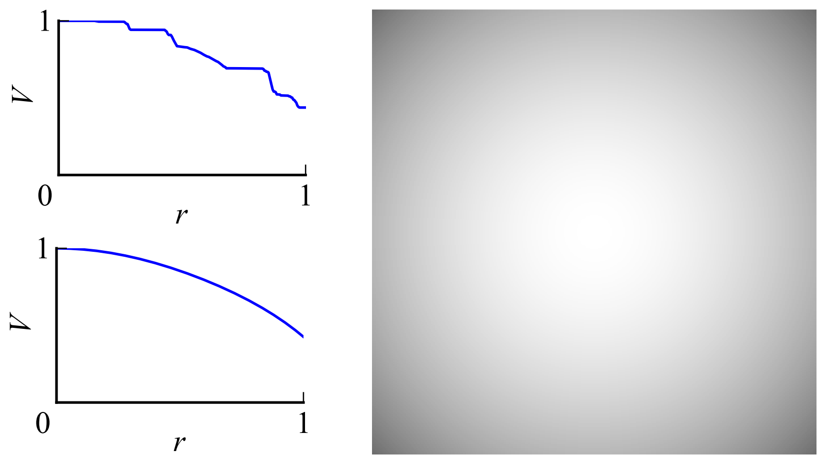


Figure 3.4: Vignetting layer extraction. Left: $V(r)$ before and after parameterization. Right: The vignetting layer created with the parameterized $V(r)$. The vignetting layer is extracted from the cluster shown in Fig. 3.3(a).

radius r, and robustly estimate the highest pixel value $M_i(r)$ by taking the 90th percentile of these pixels. To ensure that we have a non-increasing function, we compute the vignetting function $V_i(r) = \max_{x \geq r} M_i(x)$. The vignetting function $V(r)$ of the cluster is finally found by taking the median value at each r: $V(r) = \text{median}_i\, V_i(r)$. Similarly to [32], we parameterize $V(r)$ with a 6th-order even polynomial, holding its 0th-order coefficient at 1:

$$V(r) = 1 + k_2 r_2^2 + k_4 r_4^4 + k_6 r_6^6. \tag{3.1}$$

Fig. 3.4 demonstrates an example of extracting a vignetting layer from the image cluster shown in Fig. 3.3(a).

To apply the vignetting layer to the user's photo, we simply multiply each pixel by $V(r)$. Vignetting transfer changes color and contrast of the image, particularly for filters like *night* and *sunset*. Therefore, we apply vignetting transfer *before* color transfer and local contrast transfer. In addition, the spatially varying luminance caused by the vignetting transfer guides color transfer and local contrast transfer to mimic the spatial layout of global and local contrast.

3.3.3 Color Transfer

Transportation Problem

The color transfer phase essentially matches color distributions. In our system, the color distribution of the vignetting transfer result is represented by a histogram h. The histogram is normalized so that the sum of all bins is 1. In the image collection fetched from the Internet, each image cluster C_i has multiple images $I_1, \ldots, I_n$, with normalized histograms $h_1, \ldots, h_n$.

In contrast to traditional single-image-based color transfer algorithms, we face a new challenge: how to match h to the *collection* of 3D histograms? One naive method might be a "loose" matching that only requires that each bin of h lie somewhere within the range of the corresponding bins of the h_i. Alternatively, a "precise" matching might force each bin of h to be exactly equal to, say, the median of the corresponding bins of the h_i. We choose an intermediate strategy, by stating that we wish to have each bin of h lie between the ath percentile and $(100 - a)$th percentile of the corresponding bins in the cluster. Choosing $a = 0$ would thus give us the "loose" matching strategy described above, while $a = 50$ would correspond to "precise" matching. We discuss the tradeoffs in the choice of a later.

To implement this matching strategy, we define the desired lower and upper bounds on each histogram bin as

$$l(\mathbf{x}) = P_{a\%}\big(h_1(\mathbf{x}), \ldots, h_n(\mathbf{x})\big) \tag{3.2}$$

$$u(\mathbf{x}) = P_{(100-a)\%}\big(h_1(\mathbf{x}), \ldots, h_n(\mathbf{x})\big) \tag{3.3}$$

and formulate a transportation problem. We imagine each histogram bin as a warehouse, and consider how to move goods among the warehouses. Initially, warehouse $\mathbf{x}$ has $h(\mathbf{x})$ goods, and after transportation we would like it to have at least $l(\mathbf{x})$ and at most $u(\mathbf{x})$. We therefore solve for the amount of goods $f(\mathbf{x}, \mathbf{y})$ to be moved between each pair of warehouses $\mathbf{x}$ and $\mathbf{y}$.

This problem is a relaxation of the Earth Mover's distance [82], and can be solved by setting up a constrained optimization. First, the total amount of goods transported *from* each warehouse $\mathbf{x}$ should be equal to the amount it had originally, $h(\mathbf{x})$:

$$\sum_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}) = h(\mathbf{x}). \tag{3.4}$$

Second, at the end, the total amount transported *to* each warehouse $\mathbf{y}$ should lie in the desired interval $[l(\mathbf{y}), u(\mathbf{y})]$:

$$l(\mathbf{y}) \leq \sum_{\mathbf{x}} f(\mathbf{x}, \mathbf{y}) \leq u(\mathbf{y}). \tag{3.5}$$

Third, the amount of transported goods must be non-negative:

$$f(\mathbf{x}, \mathbf{y}) \geq 0. \tag{3.6}$$

Finally, our goal is to minimize cost:

$$\min_{f} \sum_{\mathbf{x}, \mathbf{y}} f(\mathbf{x}, \mathbf{y})\, c(\mathbf{x}, \mathbf{y}), \tag{3.7}$$

where in our system, $\mathbf{x}$ is a 3D color vector (L_x, a_x, b_x) that represents the mean pixel value in the histogram bin, and the cost function $c(\mathbf{x}, \mathbf{y})$ is defined as squared L_2 distance between colors:

$$c(\mathbf{x}, \mathbf{y}) = k_L |L_x - L_y|^2 + |a_x - a_y|^2 + |b_x - b_y|^2. \tag{3.8}$$

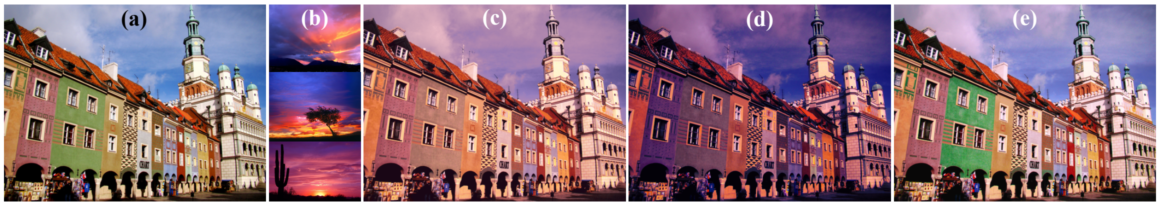


Figure 3.5: The impact of the parameter a. (a) input; (b) three images in the cluster; (c–e) results with $a = 40$, 20, and 1.

To prevent flipping of lightness channel, we assign it a higher weight $k_L = 9$. This is a linear programming problem (since it has linear objective and linear constraints), and can be solved quickly using the Simplex method.

Color Transform of Each Histogram Bin

The transportation plan f enables us to compute the color transform $T_b(\mathbf{x})$ for each source histogram bin $\mathbf{x}$. We transform the chrominance channels a, b and the lightness channel L separately.

Chrominance Channels. The distribution of a and b chrominance values in the histogram bin $\mathbf{x}$ of the vignetting transfer result can be represented by $(\boldsymbol{\mu}_v(\mathbf{x}), \boldsymbol{\Sigma}_v(\mathbf{x}))$, where $\boldsymbol{\mu}_v$ is the vector of mean chrominance values and $\boldsymbol{\Sigma}_v$ is the covariance matrix. We estimate the distribution of the transformed chrominance values $(\boldsymbol{\mu}_d(\mathbf{x}), \boldsymbol{\Sigma}_d(\mathbf{x}))$, supposing the distribution of chrominance values in the images in the cluster is represented by $(\boldsymbol{\mu}_c(\mathbf{x}), \boldsymbol{\Sigma}_c(\mathbf{x}))$:

$$\boldsymbol{\mu}_d(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})\, \boldsymbol{\mu}_c(\mathbf{y}) \tag{3.9}$$

$$\begin{aligned}\boldsymbol{\Sigma}_d(\mathbf{x}) = &\sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y}) \left(\boldsymbol{\Sigma}_c(\mathbf{y}) + \boldsymbol{\mu}_c(\mathbf{y})\boldsymbol{\mu}_c(\mathbf{y})^T\right) \\ &- \boldsymbol{\mu}_d(\mathbf{x})\boldsymbol{\mu}_d(\mathbf{x})^T.\end{aligned} \tag{3.10}$$

We seek a transform that minimizes the stretch of chrominance values $T_b(\mathbf{c}) = \mathbf{K}\mathbf{c} + \mathbf{\Delta}$, where $\mathbf{c}$ is the chrominance vector (a, b). We first apply SVD to $\mathbf{\Sigma}_v$ and $\mathbf{\Sigma}_d$:

$$\mathbf{\Sigma}_v = U_v D_v U_v^T, \quad \mathbf{\Sigma}_d = U_d D_d U_d^T, \tag{3.11}$$

where $D_v = \mathrm{diag}(d_{v,1}, d_{v,2})$ and $D_d = \mathrm{diag}(d_{d,1}, d_{d,2})$. As described by Freedman and Kisilev [27], if π is the permutation that minimizes $\sqrt{d_{d,1}/d_{v,\pi(1)}} + \sqrt{d_{d,2}/d_{v,\pi(2)}}$, then

$$\mathbf{K} = U_d S_\pi P_\pi U_v^T, \tag{3.12}$$

where $S_\pi = \mathrm{diag}(\sqrt{d_{d,1}/d_{v,\pi(1)}}, \sqrt{d_{d,2}/d_{v,\pi(2)}})$ and P_π is the permutation matrix of π. After $\mathbf{K}$ is solved, we can compute $\mathbf{\Delta}$ by $\mathbf{\Delta} = \boldsymbol{\mu}_d - \mathbf{K}\boldsymbol{\mu}_v$.

Lightness Channel The lightness channel is transformed in the same way as chrominance channels. The only difference is the lightness value is 1D instead of 2D, which makes it even more straightforward.

Color Transform of Each Pixel

We define the color transform $T_p(\mathbf{p})$ of each pixel $\mathbf{p}$ as a linear combination of the color transforms of all histogram bins: $T_p(\mathbf{p}) = \sum_{\mathbf{x}} \beta(\mathbf{p}, \mathbf{x})\, T_b(\mathbf{x})$, where

$$\beta(\mathbf{p}, \mathbf{x}) = \frac{s(\mathbf{p}, \mathbf{x})}{\sum_{\mathbf{x}} s(\mathbf{p}, \mathbf{x})}. \tag{3.13}$$

The similarity function $s(\mathbf{p}, \mathbf{x})$ is important to the quality of the color transfer result. We use a Gaussian as the similarity function:

$$s(\mathbf{p}, \mathbf{x}) = e^{-\frac{\|\mathbf{p}-\mathbf{x}\|^2}{2\sigma^2}}, \tag{3.14}$$

where $\sigma = 0.04$.

Parameter Settings

We normalize the three channels of the CIELab color space with the same scaling factor. After normalization, the ranges are $[0, 0.49]$, $[0, 0.84]$, and $[0, 1]$. Our color histograms have resolution $7 \times 12 \times 15$, and each histogram bin is a cube with edge length 0.07.

The value of the parameter a should be selected carefully. If a is close to 50, then it is equivalent to picking the median at each histogram bin. As a result, the variance between different images in the cluster is lost, which yields a color transfer result with a washed-out appearance. If a is close to 0, then each histogram bin is likely to be too flexible, which typically leads to the result that even before transferring color, the vignetting transfer result already satisfies the range at each histogram bin. Fig. 3.5 demonstrates an example. Notice that in (c) the variation between color on the buildings has been washed out, while in (e) the image has not been modified sufficiently to match the desired style. In our system, we choose $a = 20$.

3.3.4 Local Contrast Transfer

We represent the local contrast of an image with its high-frequency information, i.e., the finer-scale levels L_1, L_2, and L_3. Similarly to color transfer, we model these three levels with a 3D histogram over values in the Laplacian pyramid. Since local contrast is only meaningful for edges, we ignore all flat regions by excluding all pixels with $L_1 < 0.003$. The statistics of the 3D histogram are then transferred using the same algorithm as for color transfer, to obtain $L_l^{(L)} = L_{l1} + L_{l2} + L_{l3}$. Finally, we combine the local contrast transfer result with the coarse-scale level of the color transfer, to obtain the final result: $I_l = L_{v4} + L_l^{(L)}$.

Noise Reduction. Both color transfer and local contrast transfer potentially cause noise amplification. To prevent this, we convert L_{v1} into an alpha blending mask α

by clamping 0.003 to 0, 0.006 to 1, and linearly varying in between. We blend $L_l^{(L)}$ and $L_v^{(L)}$ with a blurred version of this mask $\alpha * G_\alpha$, where G_α is a Gaussian function with $\sigma = 2.4$.

3.4 Experimental Results

3.4.1 Comparison to Single-Image-Based Style Transfer

First, we validate our insight of using a *collection* of images to define a style. We take color transfer as an example. For each cluster of images, we compare our result (which uses the entire cluster) with the color transfer result based on just the top-ranked image in that cluster.

For this experiment, the original image is shown in Fig. 3.7(a), and due to space limitations we only demonstrate comparison results on two test cases here. We refer the readers to our supplementary materials for more comparisons.

For a style defined by a single image, each histogram bin has a fixed count value instead of a range. If the color distribution of the target image is highly incompatible with the user's photo, as in Fig. 3.6(b), we find that the result looks unnatural.

This issue can be partially resolved by introducing an *artificial* range $[h_1(\mathbf{x}) - \sigma, h_1(\mathbf{x}) + \sigma]$ for each histogram bin $\mathbf{x}$, where

$$\sigma = \text{median}_{\mathbf{x}} \Big(\text{stdev} \left(h_1(\mathbf{x}), \ldots, h_n(\mathbf{x}) \right) \Big). \tag{3.15}$$

This essentially assigns the same range to all histogram bins, resolving the problem with un-natural results. However, it causes a new problem: unrelated histogram bins may be incorrectly emphasized. As shown in Fig. 3.6(e–h), for this image cluster we would expect the blue and green tones to be emphasized while de-emphasizing other

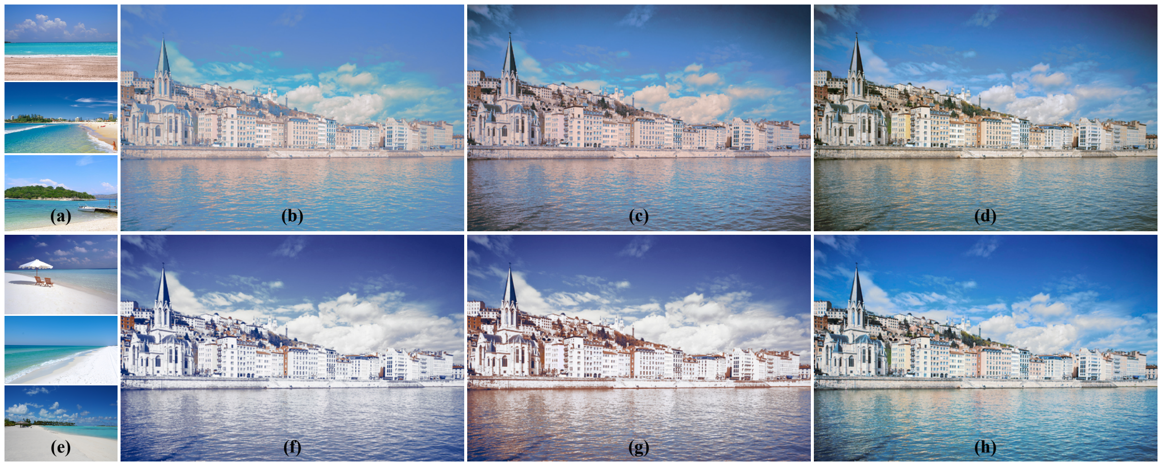


Figure 3.6: (a, e) Three images in each of two clusters for the keyword *beach*. (b, f) Single-image-based color transfer for the top image in each cluster, without artificial ranges. (c, g) Single-image-based color transfer, with artificial ranges. (d, h) Our collection-based color transfer.

tones. The single-image-based transfer result, however, incorrectly emphasizes the red tone.

3.4.2 User Studies

Methodology

We have also conducted two user studies to measure how our results match users' expectations of a filter associated with that style name. We used a pair of symmetric tests:

- **Study 1:** We showed users an image and its style transfer result. We prepared list of five style names, of which one was used to generate the style transfer result. Users were asked to select the correct name.

- **Study 2:** We showed users an image and a style name. We prepared a list of five style transfer results of the image, of which one was a result for that style name and the other 4 were from other style names. Users were asked to select the image resulting from the given style name transfer result.

For each study, we prepared 20 test cases with 20 different styles. The styles contained common objects and scenes such as *grass* and *desert*, time such as *night*, common styles of photos and movies such as *sepia* and *horror*, places such as *New York*, and abstract concepts such as *happy* and *sweet*.

How to select the *wrong* candidates is not straightforward. In Study 1, if the *wrong* style names are all quite different from the correct one in perception, then the correct one is too easy to select. If they are all highly correlated to the correct one, then it is too difficult. For instance, people may have similar expectations from *sepia* and *death*, because photos with the sepia style are often used to display people who have passed away. Therefore, we randomly selected four of nineteen styles as the wrong candidates. Similarly, in Study 2, we used style transfer results with four randomly selected styles as the wrong candidates. For all test cases, we used the style transfer results of the top-ranked cluster for each keyword. All the candidates are included in our supplementary materials.

Since "style" is a vague concept in people's minds, the feedback from individual users can vary considerably. Therefore, we conducted a relatively large user study on Amazon Mechanical Turk (MTurk) instead of a small-scale in-lab study. In each task on MTurk, we had 15 different test cases. To control the quality of the users, we repeated five of them to check the consistency of answers. In addition, we added five very easy tasks, i.e., tasks hand crafted with wrong candidates that are significantly different and easily distinguished from the correct one. To exclude respondents who simply chose randomly to complete the task, we rejected submissions in which fewer than three of the five repeated tests match, or fewer than three of the five easy tasks match the ground truth. We collected 56 and 60 submissions for Study 1 and Study 2, respectively. Each test case has answers from at least 40 different users.

Using only the top-ranked clusters can fail to produce results with distinctive visual appearance. This happened in our user study for three styles: *rust*, *sunset*,

	snow	**night**	**grass**	**desert**	**sepia**	**spring**
Study 1	0.927	0.929	0.905	0.800	0.786	0.810
Study 2	0.978	0.927	0.795	0.795	0.766	0.702
	beach	**horror**	**vintage**	**death**	**wasteland**	**avatar**
Study 1	0.634	1.000	0.860	0.595	0.643	0.372
Study 2	0.864	0.362	0.409	0.553	0.370	0.600
	rust	**new york**	**happy**	**sweet**	**sunset**	**candy**
Study 1	0.349	0.442	0.250	0.256	0.310	0.075
Study 2	0.500	0.318	0.356	0.349	0.217	0.409
	kyoto	**nightclub**	**rust***	**sunset***	**nightclub***	
Study 1	0.167	0.146	0.694	0.531	0.653	
Study 2	0.295	0.304	0.512	0.659	0.829	

Table 3.1: Proportion of correct answers in studies asking users to select the right style name for a stylized image ("Study 1") and the right stylized image for a keyword ("Study 2"). The three results marked '*' come from additional studies that used hand-selected clusters with distinctive appearance instead of the top-ranked cluster for each keyword.

and *nightclub*. For these three styles, we conducted an additional user study with lower-ranked clusters having more distinctive visual appearance.

Results

Table 3.1 shows the proportion of users that select the correct choice in both tests. We make the following observations:

- Users easily select style transfer results related to common objects and scenes, and common photo and movie styles, with the exception of *candy*.
- For styles related to time, the style transfer results match expectations, e.g., *night*, *sunset*, and *spring*.
- For abstract concepts strongly related to visual appearance, such as *death*, which often has dark tone, reduced local contrast, and reduced saturation, the style

transfer results match well. But for concepts closely related to image content, such as *happy* and *sweet*, the results do not match as well.

- Keywords based on places names usually do not result in distinctive styles, and users have difficulty with them. Since most MTurk users are English speakers, they have more difficulty in identifying the style of places outside English speaking countries, i.e., *kyoto*.

- In general, we find that users select the correct choice significantly more often when a distinctive cluster is selected, which validates our hypothesis that clustering helps create distinctive styles.

Case Studies

We examine several specific cases to better understand our user study results. All these cases demonstrate multiple distinctive yet reasonable styles, validating our motivation of clustering.

Desert. Our analysis starts with *desert*, which gets good results in our user study. Sand in deserts greatly reduces image contrast, and may or may not affect sky color. In Fig. 3.8, two styles (corresponding to different clusters) are faithfully transferred to the user's photo from Fig 3.7d: note the significant changes of tone and local contrast on the green bushes, making them look as if they are covered by sand and dust. Also note the distinct differences between the clusters validating the usefulness of clustering target images before transfer.

Sunset. Photos captured at sunset may appear red, orange, or violet, depending on the number of particles in the air and the white balance setting of the camera. Since light is scattered and absorbed over long paths through the air, local contrast is low.

In Fig. 3.9, the sunset style is transferred from three different sunset clusters to the user's photo in Fig. 3.7. Our system successfully transfers the hue of sunset photos to

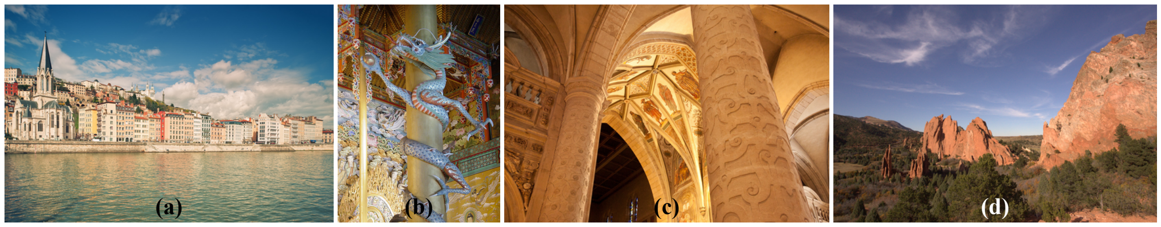


Figure 3.7: Photos used as inputs in the case studies.

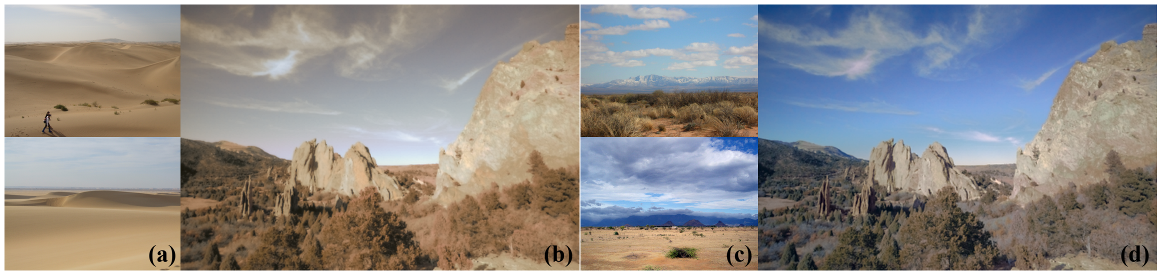


Figure 3.8: Case study: *Desert*. (a–b) two images from a cluster, and its style transfer result; (c–d) another result.

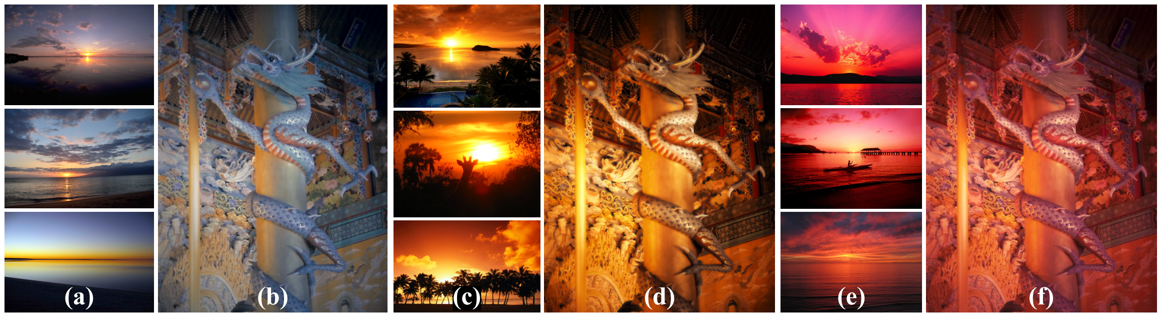


Figure 3.9: Case study: *sunset*. (a–b) Three images from the top-ranked cluster, and its style transfer result; (c–f) Images from two other clusters, and their transfer results.

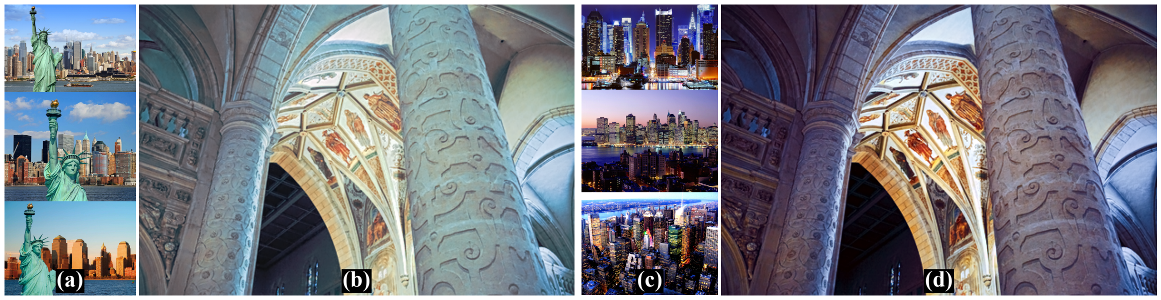


Figure 3.10: Case study: *New York*. (a–b) three images from a cluster, and its style transfer result; (c–d) another result.

the user's photo, and reduces its local contrast. The vignetting layer extracted from the sunset photos helps make the dragon stand out by de-emphasizing the details in the surrounding region.

Fig. 3.9b is based on the top-ranked *sunset* cluster, and hence was the image used in our user studies. However, this resulted in generally poor performance, so we chose to analyze this test case in more depth. In Study 1, more users choose *vintage* rather than sunset. We believe that this is a reasonable result, because 1) both vintage and sunset styles reduce local contrast; and 2) the tone of sunsets in this cluster is not very distinctive. To verify our hypothesis, we use the result in Fig. 3.9d, which is based on a more distinctive *sunset* cluster, to conduct an additional user study. The proportion of users selecting *sunset* increases from 31% to 53%. We find that for Fig. 3.9d, some users select *rust* and *autumn*, because these two styles feature with red and yellow colors, which is emphasized by this style transfer result. We think the reason why there are still users not selecting sunset for this test case is the semantics. Most sunset photos contain sky and sun, but these do not appear in the photos we use in our user studies.

New York. Photos of New York City often emphasize the high contrast of the buildings, so the *New York* style enhances local contrast. Fig. 3.10 shows two examples with enhanced local contrast and different colors. Fig. 3.10a shows the top-ranked cluster with blue sky, a green statue, and white, light yellow, and light red buildings. Fig. 3.10b shows a cluster with even more distinctive visual appearance, in which violet, blue, and yellow colors predominate. These colors show up in our results. The original image is in Fig 3.7c: note the substantial, yet plausible, shifts in color introduced by our method. Though the original image appears to have relatively uniform color, our color transport algorithm is able to use the subtle variations in hue present in the original to expand the range of colors, introducing different hues for the columns, the curved ceiling at center, and the alcove at right. In both Study

1 and Study 2, though the number of users that make the correct choice is less than half, it is still more than any other wrong style. Due to lack of semantics, it is difficult for users, particularly users outside US, to associate the visual appearance with New York. However, the style still looks visually pleasing.

3.5 Discussion

Our user studies seek to find out whether a style name produces expected results, and conversely if a result is produced by the expected style name. A more important question may be whether the filters produced by the method are desirable to users. Crafting filters by hand is tedious and requires expert knowledge. Allowing filters to be automatically produced by keywords may result in many beautiful results independent of their predictability. A further study would be required to assess this type of efficacy. An additional study should also assess how reliably users select keywords to generate expected results.

3.5.1 Limitations

Our system fails for styles not well defined by their color and contrast distributions. First, for styles strongly dependent on the content in the photos, such as *happy* and *sweet*, it is difficult to consistently produce style transfer results that most users agree on.

Second, some visual styles are defined not by their color and contrast distributions, but rather by local texture. Because we do not transfer the latter, we do not necessarily produce the expected results for keywords such as *Van Gogh* (see Fig. 3.11). Though a variety of systems for texture synthesis and "texture painting" have been explored in recent years, we believe that the investigation of *collection-based* texture transfer would make for interesting future work.

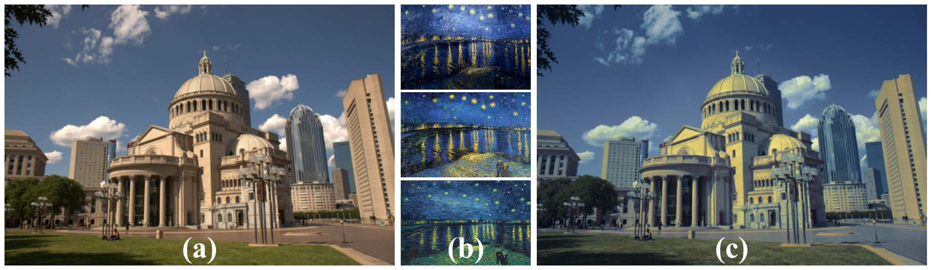


Figure 3.11: Failure case of our method. (a) The original image. (b) Three representative images in the cluster with the keyword *Van Gogh*. (c) Style transfer result — note that our method does not transfer local texture.

Third, our system relies on the power of image search engines. We find that typically many images retrieved are visually similar. However, due to the limitations of the filtering and ranking algorithms of search engines, a small portion of images may be irrelevant, which we call *noise*. Clustering helps group the relevant images into large clusters, and thus reduces the noise. However, some other clusters may be more random due to noise. Since we present results of all clusters to users, users may find that results for these clusters are not as good as others. In the future, we may adopt more sophisticated image understanding algorithms to filter out the noise, and thus further improve the quality of the results presented to users.

Fourth, our system does not consider the existing vignetting in the input image, thus yielding overly strong vignetting in some cases, e.g. Fig. 3.6(d). This problem may be resolved by estimating the *effective* vignetting in the inputs, and adding only enough additional vignetting to match it to the target. Though algorithms for detecting real vignetting have been explored, the investigation of *effective* vignetting would be interesting future work.

3.5.2 Running Time

Our system is implemented in Matlab. On a machine with a 2.2GHz CPU, using a single CPU thread, for photos with a 480-pixel longest edge, the time cost of k-means clustering is under 2 seconds. Times for applying the vignetting layer, color transfer, and contrast transfer were 0.02 seconds, 2 seconds, and 1.5 seconds, respectively.

Downloading, resizing, and extracting features and vignetting layers for hundreds of web images consumes tens of minutes in total, depending on the network condition and I/O speed. However, we envision that for common keywords the results could be pre-computed (at a number of resolutions) and distributed with the software. When the user wishes to filter a photo, the images with the closest resolution are used. For relatively unpopular keywords, after the first time downloading, resizing, and extracting features, the clustering result, vignetting layer, and histogram models can be cached and even shared with other users.

3.6 Conclusion and Discussion

We have demonstrated a data-driven system to automatically transfer style to a user's photos. To achieve this we download images related to a user-provided keyword, group the image collection into clusters based on their visual appearance, transfer vignetting based on the lightness distribution, and transfer color and contrast by matching 3D CIELab and Laplacian pyramid histograms. Our experiments suggest that the system is able to robustly transfer a variety of styles.

In contrast to quality metric for motion deblurring in Ch. 2, user intent for stylization is various across tasks. For this reason, we ask users to use a keyword to define the stylization filter in desire. Rather than crafting features and matching algorithms by ourselves, we adopt the well-engineered online image search engines, which enables our system to keep up-to-date on both image retrieval techniques and data. We then can focus on the algorithm components from where these filters end through the clustering and user interface.

Despite the continuous evolution of image search techniques, noise in image search results is almost inevitable. To make our system robust to noise, we fit our model to *ranges* of histogram bin counts instead of single counts. Since the noise image instances generally look different from relevant images, discarding tiny clusters also helps avoid overfitting to noise.

Images with the same keyword are often with *multiple* distinct distributions of color and contrast. We cluster the retrieved images, transfer the associated style of each cluster to the input photo, and ask the user to pick their favorite results. In this way, disagreement in retrieved images enhances the diversity of our stylization results.

Chapter 4

Data-Driven Iconification

4.1 Introduction

Pictograms, or *icons*, are abstracted pictorial representations of objects or concepts. Good pictograms offer an efficient, unambiguous, instantly recognizable visual lan-

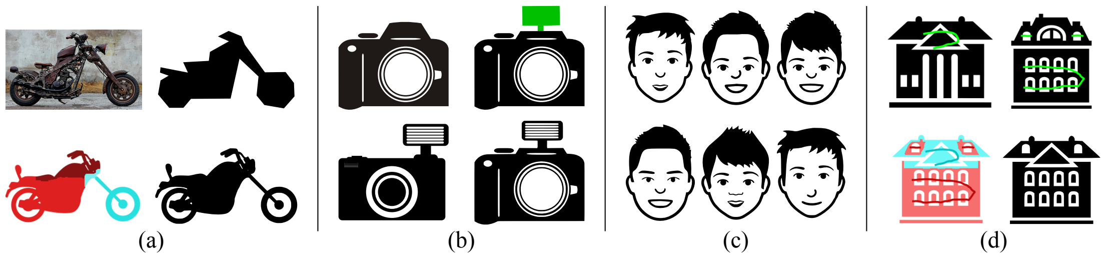


Figure 4.1: Four icon customization workflows supported by our algorithms. (a) **Sketch-based pictogram modeling:** given an input photo (top left) of a motorbike, the user sketches a polygon (top right) over the photo; this sketch, along with the keyword "motorbike," are the only user inputs (the photo is *not* used). Our method remixes partially similar pictograms (bottom left) to create a pictogram (bottom right) that matches the user's sketch. (b) **Sketch-based pictogram editing:** starting with an existing pictogram of a camera (top left), the user sketches a flash (top right, green blob) on top of the camera, and our method remixes the partially similar pictogram (bottom left) to create a pictogram of camera equipped with flash (bottom right). (c) **Pictogram hybrids:** Starting with several stock pictograms of boy faces (shown in Fig. 4.13a), our method creates random yet visually appealing hybrids. (d) **Pictogram montage:** guided by the user's scribbles (top, green), our method helps the user merge two pictograms while retaining the user-selected parts (bottom).

guage of concepts, from “global warming” to “cat,” that crosses language boundaries. They are commonly used in graphic designs, infographics, explainer videos, logos, and other forms of visual communication. Their ubiquity has given rise to collections such as The Noun Project `<http://thenounproject.com>`, a database of nearly 100,000 curated pictograms contributed by a wide array of designers. As such, The Noun Project provides a remarkable dataset of effective visual abstractions, covering an impressive variety of concepts.

While these hand-made icons are both beautiful and useful, they are a small subset of the much larger space of plausible icons. A long-standing challenge in computer graphics is to synthesize large varieties of plausible shapes from a few hand-created examples [45, 81]. For example, the existing “horse” icons in the dataset sample the larger space of all possible “horse” icons; is it possible to generate even more samples through interpolation of the existing ones? Can a user control this interpolation to create customized icons with particular properties?

This chapter describes a system for user-driven synthesis of customized icons. Our approach includes a number of ways to generate variations for a particular class of icons (e.g., “horse”), which all start by downloading a few exemplars fetched by keyword. *Pictogram Hybrids* (Ch. 4.4.3 and Figure 4.1c) generate icon variations completely automatically by randomly combining parts of the exemplars. The rest of our methods allow for user control. *Sketch-Based Pictogram Modeling* (Ch. 4.4.1 and Figure 4.1a) takes a rough sketch of an icon from the user (which may be traced over a photo), and assembles parts of exemplars to match the sketch. *Sketch-Based Pictogram Editing* (Ch. 4.4.2 and Figure 4.1b) allows a user to select a particular icon and then remove parts and/or add sketched regions; the algorithm finds parts of other exemplars to satisfy the user edits. Finally, *Pictogram Montage* (Ch. 4.4.4 and Figure 4.1d) allows a user to select multiple exemplars and combine parts of them into a new, plausible pictogram.

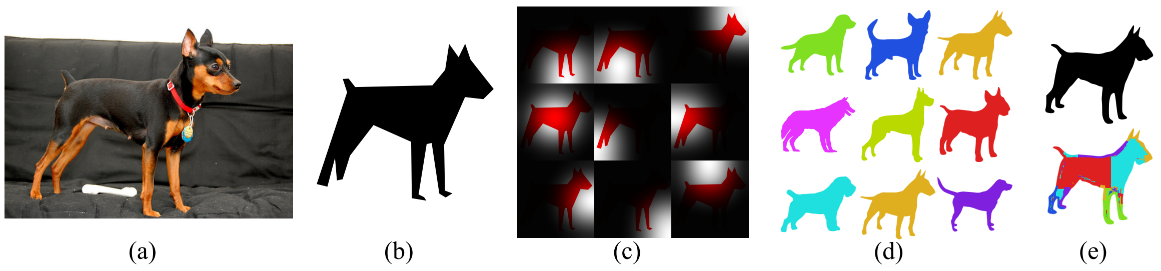


Figure 4.2: An overview of our sketch-based pictogram modeling algorithm for a *dog* query. The user first selects a photo (a) illustrating the desired concept, and sketches a rough polygon (b) over it. Our algorithm detects salient regions (c) in the polygon, then uses sliding-window matching to find the most similar exemplar pictograms (d) for each region (only one for each region shown here). These are remixed and blended to create a final pictogram (e, first row) that matches the user's intent. We color-code each icon pixel to indicate which icon the pixel comes from.

Our system allows multiple workflows because users have different reasons for customizing icons. Some users may simply want to be inspired by automatically generated variants; others might have a rough shape from a photograph they wish to mimic; and still others may like a particular icon but wish to change a local detail. All of these workflows are supported by our technical framework. We describe the rest of our algorithms in the context of our most challenging workflow, Sketch-Based Pictogram Modeling, and then show how the different workflows can be reduced to this variant.

We must solve two hard technical problems to accomplish the goal of customizing icons: search and remixing. First, we find parts of exemplar icons from a database (Ch. 4.3.1) that match salient parts of the user sketch, allowing for some geometric transformation. This step is similar to the problem of partial shape matching [98], though in our case we have the added complication that the user-drawn sketch will generally be much rougher and less detailed than the desired pictogram. We therefore decompose the sketch into a number of salient regions (Ch. 4.3.2), and then use approximate sliding-window matching (Ch. 4.3.3) to find candidate exemplar parts.

We then combine the candidate parts (Ch. 4.3.4) into a plausible, seamless whole that matches the user-drawn shape; for this step we use a Markov Random Field (MRF) optimization. We describe how to limit the space of matching pictograms to generate results in a variety of coherent styles (Ch. 4.3.5). Our results (Ch. 4.4.1) demonstrate many cases in which this algorithm is able to beautify user-provided pictograms while introducing detail, including interior detail that was not present at all in the input.

4.2 Related Work

There are a number of techniques for helping users create 2D sketches. One approach is to provide a content-sensitive visual reference [54, 44]; in this case, the user-drawn sketch is the final artifact. Another approach is to automatically beautify the sketch [111, 5]. Our approach is instead to replace a user-drawn shape with a combination of parts from professionally drawn art.

Combining parts of existing artwork has been explored in a number of domains. For example, combining parts of photographs [2, 36, 13] and manipulating photos [31] are common operations, though the domain of pictograms requires very different design choices than natural images. First, the lack of color and texture requires us to focus on shape alone. Imperfections in how contours match up cannot be masked by texture, and even modest deformations to the shape can ruin the beautiful forms designed by artists. Therefore, we must devote significant effort at both the retrieval and graph-cut stages to ensuring that the parts we remix match up seamlessly. Second, because our source icons are designed by artists, and not by the public at large, there will be a significantly smaller number of them available for any query: dozens or hundreds, as opposed to millions of amateur photographs. This means that we cannot rely on finding an almost-perfect match for a query, and our algorithm must focus on remixing exemplars with large variation, rather than on warping already-good retrieval results to match a query's details.

Combining parts of retrieved results has also been explored in data-driven 3D modeling systems [29]; however, significant user effort is typically required. More recent systems [12] reduce user effort, but require a pre-segmented and labeled 3D model database. Such model annotation would be difficult for pictograms, since they tend to be more abstract and less photorealistic than 3D shape databases. Most similar to our problem are photo-guided techniques that help users build a 3D model similar to a reference photograph using parts from a model database [104, 91]; again, these require aligned and segmented 3D models.

Another approach to remixing parts of objects into new ones is to generate large numbers of combinations without any user guidance. This idea has been explored for 3D models [76, 45, 40], but even more relevant to us are several 2D techniques [81, 39]. These 2D methods can also produce pictograms; however, they are not designed to allow user control.

Our technical approach requires a solution to a problem similar to partial 2D shape matching. Shape matching has a long history in computer vision [98], where it is used to find objects in photographs via their silhouettes. Shapes are often matched partially in order to handle occlusions or deformation, but even partial shape matching is typically evaluated for whole-object detection tasks [65]. In contrast, we are truly interested in finding partial shapes. There are two other significant differences from shape matching for object recognition. In our case, icons and sketches are typically in canonical positions, so we do not need to allow for arbitrary rotations (we do allow for reflections, scale and translation). Without rotation, much easier matching methods can be used, such as sliding windows. Second, the user-drawn query sketches typically contain much less detail than the database icons; we therefore first find the most salient regions of the sketch, and focus the matching on those.

Finally, once shape regions are matched, we use a graph-cut-based algorithm to seamlessly combine the pictograms, similar to algorithms used in image compositing [7, 51, 2]. However, our energy functions are customized to the pictogram context. Since the user-drawn sketch is only an outer contour, we use signed distance functions to propagate shape information globally. We also use larger windows (7×7) for the pairwise smoothness cost in graph cuts, since higher-order smoothness artifacts are more noticeable for vector art.

4.3 Algorithm

We first describe our full technical pipeline for the most challenging workflow, Sketch-Based Pictogram Modeling. We then show in Ch. 4.4 how the other workflows can be accomplished with the same system with minor technical modifications.

Fig. 4.2 illustrates the modeling workflow. We begin by asking the user to sketch a solid polygon P (Fig. 4.2b), which might be obtained by drawing the object of interest over a photo (Fig. 4.2a), or by drawing free-hand. For the user's convenience, we expect P to describe only the rough outline of the desired pictogram — our algorithm will add details to the contour and fill in the interior structure.

We then ask the user for a keyword K (*dog*, in this case) to describe the object class, and use K to retrieve a collection of at most 200 pictograms from The Noun Project (Ch. 4.3.1).

Since the user's sketch will be overly coarse compared to the desired result, we first detect the detailed, salient regions in P that should control the matching process (Ch. 4.3.2). We then find the best matches $\{I_i\}$ for each region (Ch. 4.3.3). These matches are combined using multi-label graph-cuts (Ch. 4.3.4) to produce a new pictogram (Fig. 4.2d) that mimics P. We restrict matches to come from exemplars of similar style (Ch. 4.3.5) in order to ensure consistency in our output.

Our remixing algorithm operates in the pixel domain, then re-vectorizes the final result. At first glance, this might seem wasteful, since the input SVG pictograms are vector, not raster, drawings. However, after some experimentation we chose pixel-domain processing for two reasons. First, it is difficult to remix interior structures consistently with the contours in the vector domain. Second, the deformations required to merge contours in the vector domain often lead to unnatural appearance or even break semantics. We note that recent work on stroke stylization [64] also made the same choice of raster-domain processing followed by vectorization, for many of the same reasons.

4.3.1 Fetching Data

Our system obtains source pictograms for remixing from The Noun Project, a large user-contributed but curated online pictogram repository. In order to restrict our remixing to models with detail appropriate for the given class, we query the repository with the user-provided keyword, and fetch pictograms in SVG format via its API. We only focus on the pictograms consisting of black and white solid shapes and lines, which covers the majority of the repository.

The SVG file of a pictogram fetched from The Noun Project represents a tree, with geometric primitives stored at its leaves and interior structure representing grouping information. However, the structure of the tree proves difficult to exploit for semantic grouping, for a few reasons. First, most artists do not segment the pictogram into semantically-independent parts. In other words, the tree structure in the SVG file is neither consistent nor reliable. Second, pictograms are often 2D projections of 3D objects; different pictograms, even with the same keyword, may represent projections from different views and with different occlusions. This makes it difficult to extract a *semantic* tree model to represent the hierarchical structure of parts in a pictogram, and hence to infer the semantic correspondence between pictograms. Therefore, we are only able to consider a pictogram as an unstructured collection of shapes.

4.3.2 Salient Region Detection

Given a sketch polygon P drawn by the user, we first find a number of overlapping salient regions. Intuitively, we assume that each noticeably convex or concave local structure on the polygon boundary reflects the user's intent — otherwise, the user would not have included that detail. We detect these local structures by first computing a salience map, and then selecting local regions from this map. The salience map is a sum of two terms that measure *turning angles* and *black/white balance*, respectively.

The turning-angle term measures geometric salience by aggregating the turning angles of polygon edges in local regions. In practice, we first rasterize P onto an $m \times m$ bitmap I_P, where $I_P(x, y) = 0$ (black) for all pixels inside of P, and $I_P(x, y) = 1$ otherwise. In our implementation, we set $m = 256$. We compute the (unsigned) exterior angle $\theta(x, y)$ for each pixel at a polygon vertex, and let $\theta(x, y) = 0$ for other pixels. Then, the aggregated exterior angle is the convolution of θ and a smoothing kernel M:

$$T = \frac{\theta * M}{\max(\theta * M)}, \tag{4.1}$$

normalized to ensure that $\max(T) = 1$. The mask M represents the extent of a region (e.g., the nine patterns shown in Figure 4.2c); the specific design of M is discussed below.

A good salient region should be neither mostly inside P nor mostly outside P, to prevent trivial matching results (all black/white pixels). The second term in the salience map therefore measures black/white balance — i.e., the balance between pixels inside and outside of P:

$$B = \left|(I_P - 0.5) * M\right|, \tag{4.2}$$

where both the subtract and absolute operations are element-wise. Thus, B will be near 0 when about half the pixels in the neighborhood are black.

We compute the final salience for each pixel by combining the turning-angle score and the black/white balance score:

$$S(x, y) = T(x, y) - \lambda_B \, B(x, y), \tag{4.3}$$

where $\lambda_B = 0.1$. This weighs $T(x, y)$ more than $B(x, y)$, because our ultimate goal is to detect geometric salience.

The design of our mask M has several motivations. First, to prevent false-positive matches due to overfitting local structures, M should be a rather large mask. In addition, we would like regions to have significant overlap with adjacent ones, so that selected pictograms will have similar geometry in overlapping regions. Finally, since a portion of boundary area will be cut in the remixing step, M should focus mostly on matching structures in its center. Therefore, M should be constant in the middle and fall off smoothly. In particular, we take M to be a 100×100 square smoothed by a Gaussian with $\sigma = 40$.

For the number of salient regions, we choose 9; we find that this strikes a balance between matching details in P while still maintaining much of the global structure of the exemplars. In addition, this covers the whole image with considerable overlap between regions. These 9 regions should have the highest scores, but be located at least $\Delta_S = 75$ pixels apart. We find these regions in a greedy fashion: in each round, we pick a region centered at a valid pixel with the highest score, then label pixels within Δ_S as invalid. With this strategy, the first regions we obtain often contain one or more of the most-salient local structures, and also have the best balance between pixels inside and outside the polygon. These regions help us to capture the important large-scale structure of the user's intent. The last regions we obtain usually contain just a single small salient local structure, and are often less balanced between inside/outside pixels. This is helpful to capture local details.

For each salient region, we create a mask M_i ($1 \leq i \leq 9$) with $m \times m$ pixels to be used for sliding-window matching. Specifically, for each pixel (x, y) inside the region, we let $M_i(x, y)$ equal the corresponding value of M; for other pixels, we let $M_i(x, y) = 0$. Fig. 4.2c shows the masks we generate for the 9 salient regions.

4.3.3 Sliding-Window Matching

For each salient region, we now wish to find the exemplar pictograms that match the query most closely in that region (i.e., as weighted by the mask M_i). To formulate the matching problem, we must define both the matching function (i.e., when are two pictograms considered similar) and the space of transformations over which to search.

The most natural choice for a matching function would be simply the difference between (black/white) rasterized images, weighted by M_i. However, this choice of matching function introduces large penalties for even slight misalignment, reducing our ability to match roughly-drawn user sketches to detailed icons from the database. Instead, we would like to allow for small mismatches between the sketch and a database icon, while still penalizing large misalignment.

We therefore choose to match pictograms by comparing *truncated signed distance fields* (TSDF), which have also been used in previous texture synthesis approaches [55]. For each pixel of both the sketch P and each retrieved pictogram E_j, we find its signed distance (positive outside, negative inside) to the nearest point on the exterior contour. To avoid over-penalizing mismatches, we clamp the signed distance value to some range $[-d_{max}, d_{max}]$; we experimentally found that $d_{max} = 6$

worked well. Finally, we normalize the TSDF by dividing by m. Given a normalized TSDF image D_P for the query and D_j for a pictogram E_j, our matching function is formulated as the weighted squared distance between them:

$$\delta(\mathcal{T}) = \sum_{x,y} M_i(x,y) \Big[\mathcal{T}\big(D_j(x,y)\big) - D_P(x,y) \Big]^2, \tag{4.4}$$

where $\mathcal{T}$ is an appropriate transformation.

We next need to choose a class of transformations over which to search. Previous shape-matching systems are complicated by the need to consider the full set of rigid-body transformations; rotations in particular lead to more complex shape matching. In our application, however, we observe that most pictograms have a clearly defined upright orientation, which causes rotation to break semantics. Therefore, we restrict our transformations $\mathcal{T}$ to be translations, which greatly simplifies the matching problem. In fact, the sliding-window computation can be effectively accelerated via the Fast Fourier Transform (see Appendix B).

In practice, we allow two further classes of transformations, which we have found to improve matching without violating the semantics of pictograms. First, we allow *horizontal* (but not vertical) flipping. Second, we allow for modest *uniform* (but not anisotropic) scaling, restricted to factors of 0.9 and 1.1. In practice, we find this limited set of scales to be sufficient in most cases. Indeed, because artists tend to draw fewer details on smaller parts of objects, very aggressive scaling potentially leads to results that combine parts with different levels of details, which is what we want to avoid. The search over these additional transformations is implemented by simply creating additional flipped and/or scaled versions of each retrieved pictogram.

Thus, for each salient region we can form a ranked list of the exemplars, together with translation/flip/scale, that best match that region. For each region, we keep the

3 best matches. Fig. 4.2d shows one of them per region. The resulting $9 \times 3 = 27$ transformed exemplars serve as sources for our remixing stage.

4.3.4 Pictogram Remixing

The goal of the remixing stage is to produce an output that is as plausible as the 27 input pictograms, but respects the user sketch. Specifically, the remixing should (1) yield a contour similar to the user's query P; (2) synthesize interior structure present in the exemplars (but not the query); and (3) produce a seamless result, both on the outside contour and on inside structure.

We formulate remixing as a multi-label graph-cut problem. Specifically, the result is modeled as a 4-connected 2D lattice containing $m \times m$ sites, one per pixel. Each site has an indicator label $L(i)$ that can take on one of 27 values, corresponding to the exemplars. The energy function is modeled as the sum of a unary data cost and pairwise smoothness cost:

$$F(L) = \sum_{i \in V} F_D\big(i, L(i)\big) + \lambda_S \sum_{(i,j) \in E} F_S\big(i, j, L(i), L(j)\big), \tag{4.5}$$

where V is the set of sites on the lattice and E is the set of edges connecting adjacent sites on the lattice. In this formulation, the data cost F_D measures the similarity to P, and the smoothness cost F_S measures the smoothness of remixing.

Data Cost. We expect the contour of our remixing result to be similar to P. Since P only describes the rough shape of the user's desired result, we tolerate small offsets between the contour of our remixing result and P. For this reason, we choose to use the truncated signed distance field of the contour to measure data cost. Specifically, we compute the TSDF for P and *only the external contour* (i.e., ignoring internal structure) of $\{I_k\}$ to obtain D_P and $\{D_k\}$. At each site i, the data cost is then defined as the difference between P and the exemplar:

$$F_D\big(i, L(i)\big) = \big|D_P(i) - D_{L(i)}(i)\big|. \tag{4.6}$$

Smoothness Cost. Since we expect the remixing result to be seamless, we would like to avoid visible offsets when remixing pictograms. For this reason, we use the image pixel color instead of the signed distance field in the smoothness cost. Specifically, we collect a 7×7 neighborhood $N_k(i)$ for each pixel in each exemplar I_k. The smoothness cost is defined as

$$F_S\big(i, j, L(i), L(j)\big) = \begin{cases} 0 & \text{if } L(i) = L(j) \\ \max(\epsilon, f_n + f_{int}) & \text{otherwise,} \end{cases} \tag{4.7}$$

where f_n is the distance between neighborhoods:

$$f_n = \big\|N_{L(i)}(i) - N_{L(j)}(i)\big\|_2 + \big\|N_{L(i)}(j) - N_{L(j)}(j)\big\|_2, \tag{4.8}$$

and f_{int} is a penalty for cutting through interior structure. Specifically, we set $f_{int} = 0.5$ if either $I_{L(i)}$ or $I_{L(j)}(j)$ is interior structure (a white pixel inside the contour), and set $f_{int} = 0$ otherwise. Finally, if two different labels are remixed, even if the remixing is perfectly seamless and avoids cutting through the interior structure, we still consider it worse than not remixing. Therefore, we clip the lower bound of F_S to $\epsilon = 0.01$ when $L(i) \neq L(j)$.

Symmetry Constraint. For pictograms exhibiting reflectional symmetry, *e.g.* a lamp, face, airplane, or chess piece, we would like to ensure that each pixel is taken from the same exemplar as its symmetric counterpart. To this end, we connect each site i to r_i, where i and r_i are symmetric with respect to the symmetry axis. The smoothness cost on this edge is then defined as:

$$F_S\big(i, r_i, L(i), L(r_i)\big) = \begin{cases} 0 & \text{if } L(i) = L(r_i) \\ +\infty & \text{otherwise,} \end{cases} \tag{4.9}$$

In our experiments, we find it sufficient to restrict the symmetry axis to horizontal, vertical, or diagonal lines through the center. Since each pictogram is rescaled to its tight bounding box and centered with respect to the canvas, as long as its outer boundary is symmetric (which is true for most cases), its true symmetry axis will pass through the center. (The same assumption is also made by previous work such as Structured Image Hybrids [81].)

It is not difficult to prove that our smoothness cost is a metric in the space of labels. That enables us to use the α-expansion algorithm [7] to approximately minimize $F(L)$.

Impact of $\boldsymbol{\lambda_S}$. The parameter λ_S controls the balance between data and smoothness cost. Intuitively, a large value of λ_S encourages smoothness of remixing, at the expense of similarity to the user's sketch polygon. A small value of λ_S makes the remixing result more similar to the user's sketch polygon, but artifacts tend to emerge. In practice, we observe that when $\lambda_S \geq 3$, the remixing result is most likely a single pictogram that is most similar to the user's sketch polygon. When $\lambda_S \leq 0.01$, the remixing result has contributions from almost all exemplars, but has many artifacts. We observe that there are often multiple values of λ_S that yield plausible but different results, and the ideal value depends on how closely the user wishes to match the input sketch. Therefore, we sample a set of values between 0.01 and 3.0, and run the graph-cut algorithm with each sampled value. All of these are presented to the user.

The Benefit of Patch-Based Smoothness Cost. A key difference between our remixing algorithm and Photomontage [2] is that we use 7×7 patches instead of

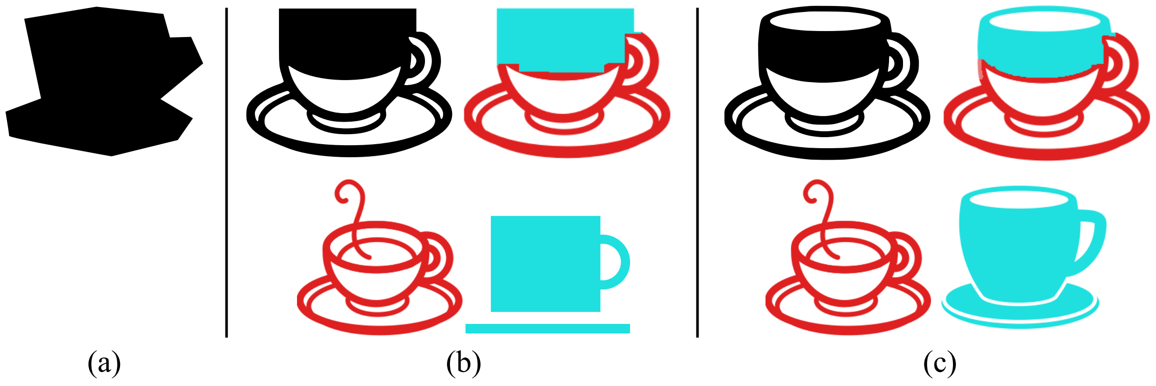


Figure 4.3: Patch-based vs. pixel-based smoothness cost. The user sketches a cup (a, top). We compare graphcut results with pixel-based (b) and patch-based (c) smoothness cost. Patch-based smoothness results in a more natural connection of the curves on the two exemplars. Note that the weight placed on smoothness, λ_S, is adjusted to 10 in (b) and 1 in (c) so that both results use exactly two exemplars.

pixels in our smoothness cost. Given the fact that the user's sketch is usually rough, this change is critical to avoid high-order artifacts, e.g. in the slope and curvature of strokes and contours. Fig. 4.3 demonstrates an example in which the user sketches a cup. Although we expect the remixing results to differ from the user's sketch, they should still have a natural appearance. However, we observe that the pixel-based smoothness cost results in an unnatural connection of a straight line and a curve on the edge of the cup. In contrast, our patch-based smoothness cost yields a natural remixing.

The neighborhood size in the smoothness term is a compromise among quality, speed, and memory consumption. Generally, a larger neighborhood is able to keep higher-order structures, but it makes graph-cuts slower and less memory efficient. In our experiments, we find that 7×7 neighborhoods work well enough in most cases, while also having acceptable speed.

Post-processing. As noted earlier, all of our matching and remixing operations are performed on rasterized versions of the pictograms. We use the open-source `potrace` package `<http://potrace.sourceforge.net>` to vectorize the result.

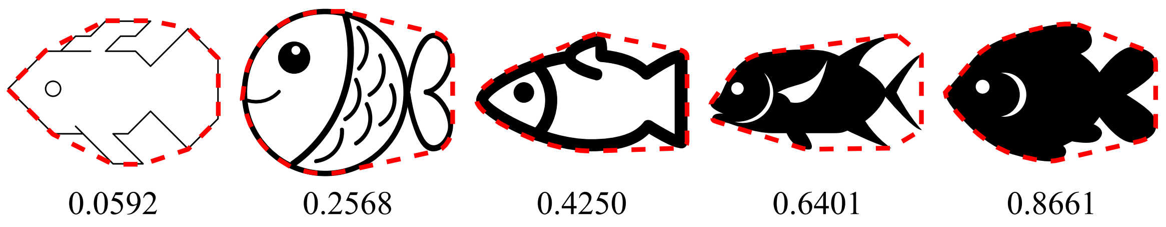


Figure 4.4: Five *fish* pictograms enclosed by their convex hulls (red dashed polylines), and the resulting blackness values.

4.3.5 Style Consistency

Pictograms from The Noun Project come in various styles. Some are solid shapes, some have rich interior structure, and some are line drawings. We would like our algorithm to only remix pictograms with similar styles, for two reasons. First, mixing multiple styles usually produces results that are implausible and not visually pleasing. Second, restricting the algorithm to one style at a time allows us to produce multiple results in various styles, offering the user a greater selection of results to match his or her intent.

There is an existing method to measure similarity of illustration style [30]; however, it is overkill for the limited range of "pictogram" or "icon" styles. We instead use a simple *blackness* feature to measure the style of a pictogram. We define blackness as the proportion of black pixels in the convex hull of the pictogram. Intuitively, this helps us distinguish among solid pictograms, those with rich interior structure, and line drawings. In addition, we observe that among line-drawing pictograms with the same keyword, blackness has a strong correlation with stroke width. In general, if we sort pictograms with the same keyword by their blackness values, the first ones are line drawings with very thin strokes. Strokes become thicker as blackness increases. Then pictograms with rich interior structure appear. Purely solid pictograms come last. Some examples are shown in Fig. 4.4.

Blackness values lie between 0 and 1, and we create four overlapping intervals to cover this range, as shown at right. We associate a "style" with each interval and generate separate results for each one by restricting matching to pictograms with blackness falling in that interval. In general, we find that each style for which there are sufficient pictograms in the database yields a plausible result.

style	blackness
lightest	[0.0 .. 0.4]
light	[0.2 .. 0.6]
medium	[0.4 .. 0.8]
dark	[0.6 .. 1.0]

4.4 Workflows

We first show results of the algorithm we just described for Sketch-Based Pictogram Modeling. Then, we describe minor modifications to this algorithm to support three other workflows, and provide results for them, as well.

4.4.1 Sketch-Based Pictogram Modeling

We now examine a set of test cases to help understand how our algorithm works. For each test case, we use each of the four intervals of blackness, and run our graph-cuts algorithm with a set of values for λ_S from 0.01 to 3.0. Ideally we would produce

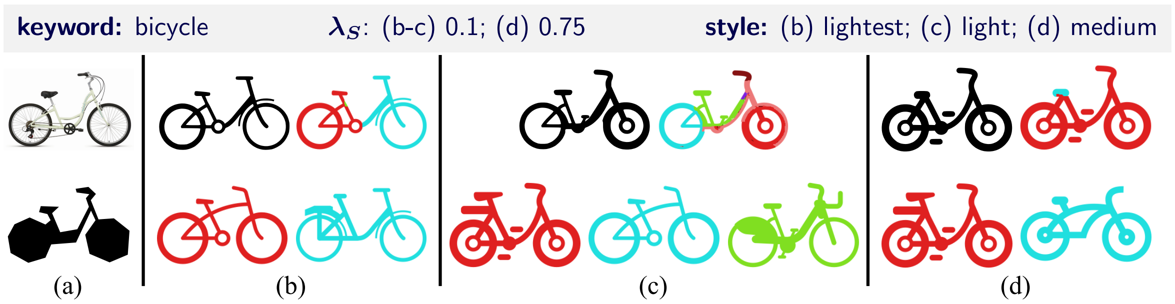

keyword: bicycle
λ_S: (b-c) 0.1; (d) 0.75
style: (b) lightest; (c) light; (d) medium
(a)
(b)
(c)
(d)

Figure 4.9: *Bicycle.* (a) The user sketches a polygon to roughly represent the shape of a bicycle in a photo. (b-d) Three remixing results in different styles (i.e., blackness values). The first row contains the result, and a color visualization of how the result is assembled. The second row contains the exemplars that contribute a significant portion of the result.

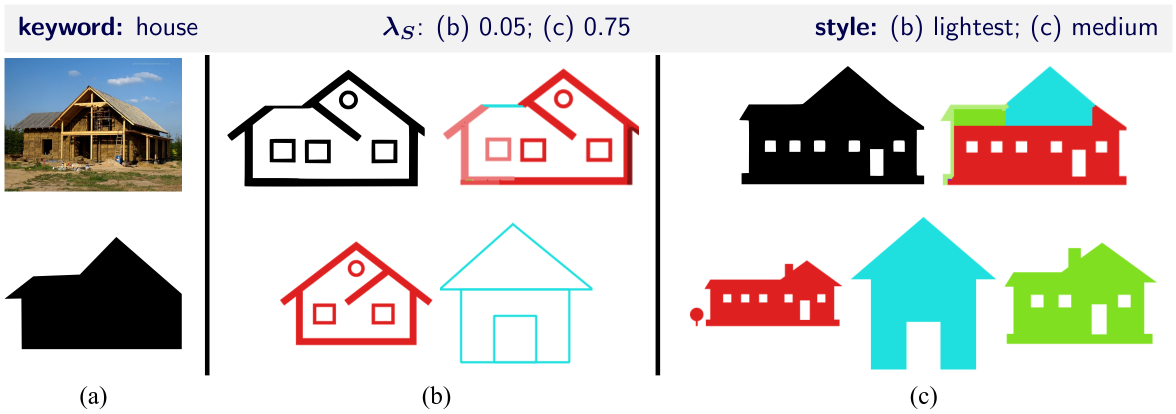

keyword: house
λ_S: (b) 0.05; (c) 0.75
style: (b) lightest; (c) medium
(a)
(b)
(c)

Figure 4.5: A *House* in three different styles.

four results, with four different styles, for each test case. However, there are often not enough pictograms for every style for each keyword, and our algorithm fails to produce good results for those styles. So, we only demonstrate styles that yield plausible results. Our results figures contain a mix of different λ_s and style parameter settings; we use a legend bar at the top of each figure to identify the parameters used.

The figures show how each result is assembled by assigning a distinct hue to each source pictogram. Often, several exemplars come from the same retrieved pictogram, but with different offset, flip, or scale. In this case, we use the same hue with different lightness (in HSL color space). Note that unless λ_S is large, our results may contain many tiny parts from different exemplars on the interior, which reduces data cost for pixels inside the contour. While this has no impact on the appearance of the remixing result, it makes the visualization of assembly more difficult to interpret. So, for clarity, we only include exemplars that contribute significantly to the result.

Motorbike. Fig. 4.1a demonstrates a test case of motorbikes. Note how our algorithm uses additional pictograms to make the motorcycle "chopped".

House. Fig. 4.5 demonstrates remixed house pictograms in three styles: line drawings, solid shapes with rich interior structure, and solid shapes. Note how our algorithm uses two identical line drawings of the house with offsets to mimic a wider house

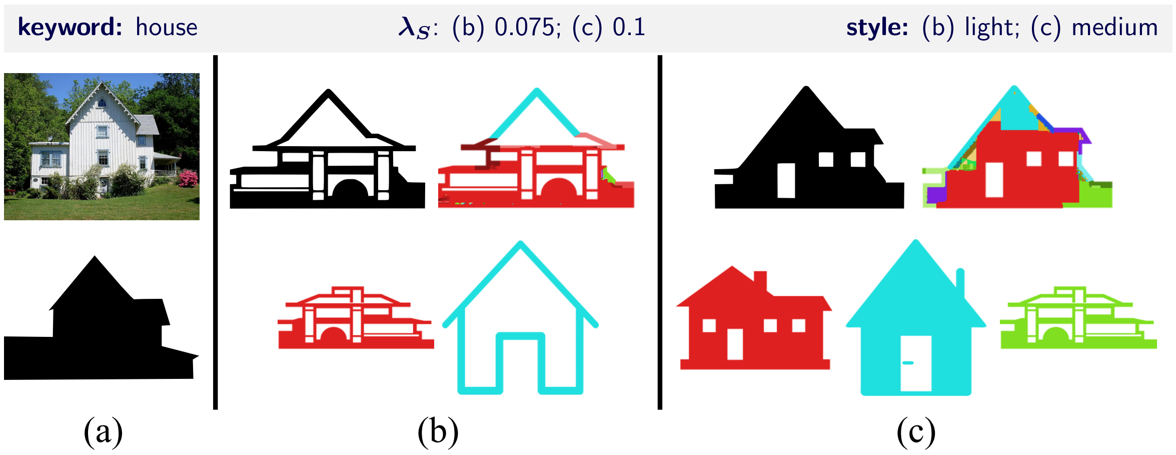


Figure 4.6: Another *House* in two different styles.

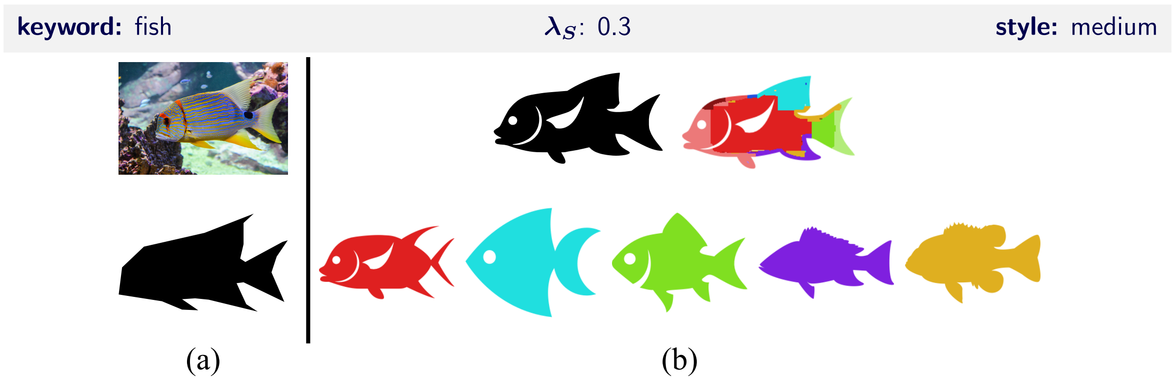


Figure 4.7: *Fish.* Parts of many exemplars are combined to match the sizes of the fins and tail.

in Fig. 4.5(b). Fig. 4.6 demonstrates a different house in two styles: line drawings and solid shapes with rich interior structure.

Fish. In Fig. 4.7a, the user sketches a fish with a large dorsal fin on top, two smaller fins on the bottom, and a fat tail. With $\lambda_S = 3$, our algorithm would just return the red pictogram in Fig. 4.7b as the result, matching the positions but not the sizes of the fins. With a smaller value of $\lambda_S = 0.3$, our algorithm uses several additional pictograms to produce a result with correctly-sized fins and tail (see Fig. 4.7b). Also, note how plausible internal details are included that the user does not need to sketch on her own.

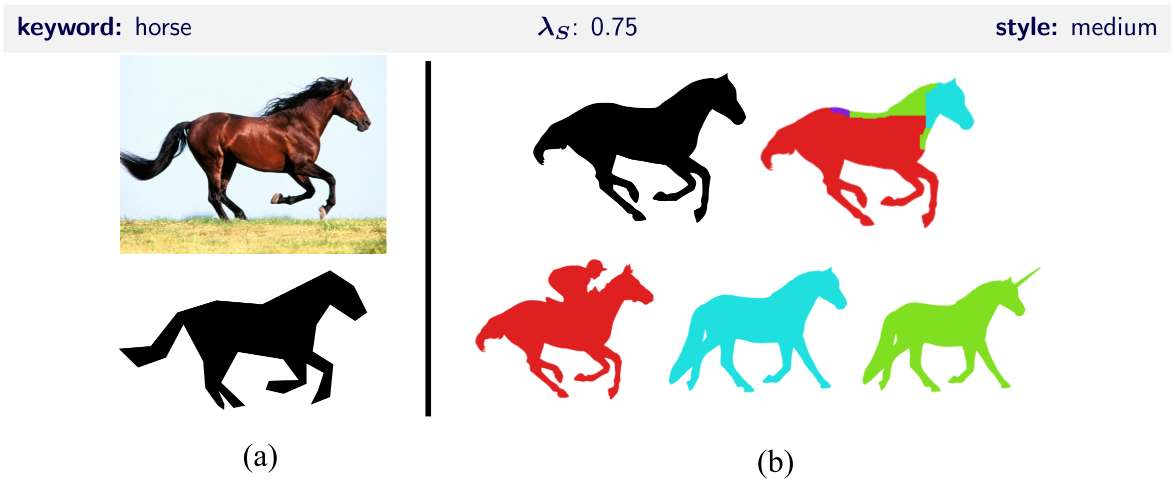


Figure 4.8: A *Horse* pictogram.

Horse. Fig. 4.8 demonstrates a test case of horses. Note how our algorithm uses additional pictograms to remove the rider from the back of the horse in Fig. 4.8.

Bicycle In Fig. 4.9a, the user sketches a polygon for a bicycle with a very low seat and without top tubes, and would like to remix pictograms from the Noun Project to mimic this appearance. Fig. 4.9(b-d) demonstrates three pictograms in different styles generated by our algorithm. In Fig. 4.9b, the first exemplar already has a low seat, but it has a top tube; the second exemplar is the opposite case. Our algorithm remixes them seamlessly, to produce a result that perfectly matches the user's intent. Fig. 4.9c collects parts from three pictograms, yet produces a seamless result with a low seat and no top tube. In Fig. 4.9d, our algorithm cuts a low seat from the second exemplar, and pastes it onto the first exemplar, to obtain a visually plausible result.

4.4.2 Pictogram Editing

Our system also enables editing existing pictograms rather than sketching from scratch. Fig. 4.10 demonstrates an example.

We enable editing by encouraging our remixing algorithm to choose the initial pictogram E_0 outside the changed area, i.e., the union of red and green parts, and

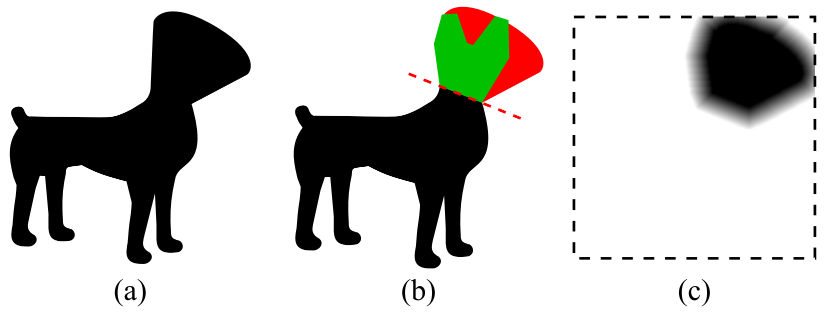
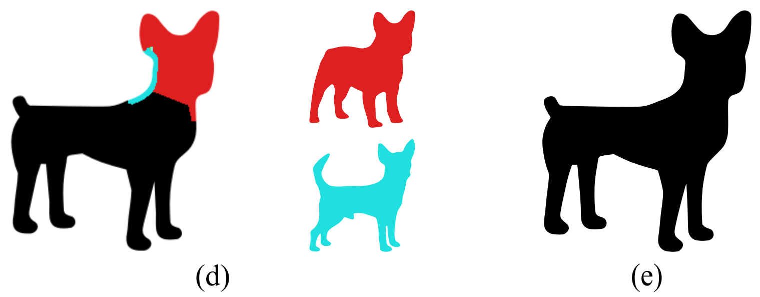



Figure 4.10: Pictogram editing. Starting from an existing pictogram (a), the user removes the red region using a red, dashed line and sketches the new, desired shape with a green polygon (partially occluding the red region). Our system generates a mask (c) for the changed region, and exemplars are retrieved and remixed (d) based on this mask to generate an edited pictogram (e).

penalizing E_0 inside that area. To this end, we create a mask D_E by computing a truncated distance field outside the union of the red and green parts (Fig. 4.10c). D_E is normalized by the truncation threshold $t = 25$ to ensure its maximum value equals 1. Our salient region search should focus on the changed area; we therefore add a term to S to emphasize the regions centered at the pixels inside the changed area, with a high weight $\lambda_D = 100$:

$$S'(x, y) = S(x, y) + \lambda_D \big[(1 - D_E) * M\big]. \tag{4.10}$$

We rule out pictogram E_0 when selecting exemplars during sliding-window matching, but add E_0 back in as label 0 during graph cuts. We then add a penalty P_E to

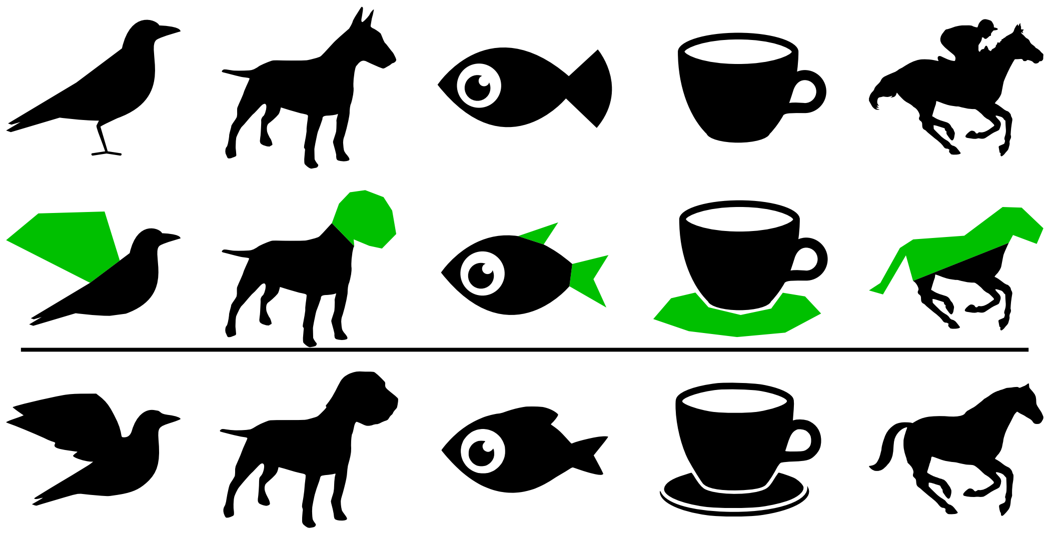


Figure 4.11: Pictogram editing results. Row 1: initial pictograms; Row 2: the user's edits; Row 3: edited results.

the data cost for graph-cuts:

$$P_E(i, L(i)) = \begin{cases} w_E\left[1 - D_E(i)\right] & \text{if } L(i) = 0 \\ w_E\, D_E(i) & \text{otherwise.} \end{cases} \tag{4.11}$$

For pixels inside the changed area, $P_E = w_E$ if $L(i) = 0$, so E_0 is penalized. The penalty is reduced as pixels get more distant from the changed area until the t threshold is crossed and the penalty becomes zero. Similarly, the other source pictograms are penalized outside the changed area, but are encouraged inside. Weight $w_E = 0.1$ controls the balance between this penalty and the original data term. The parameters w_E, t, and λ_D are set experimentally.

We show a number of results of editing existing pictograms to change one or more parts in Fig. 4.11. Notice how the algorithm uses parts of other icons to add detail to the user's rough sketches.

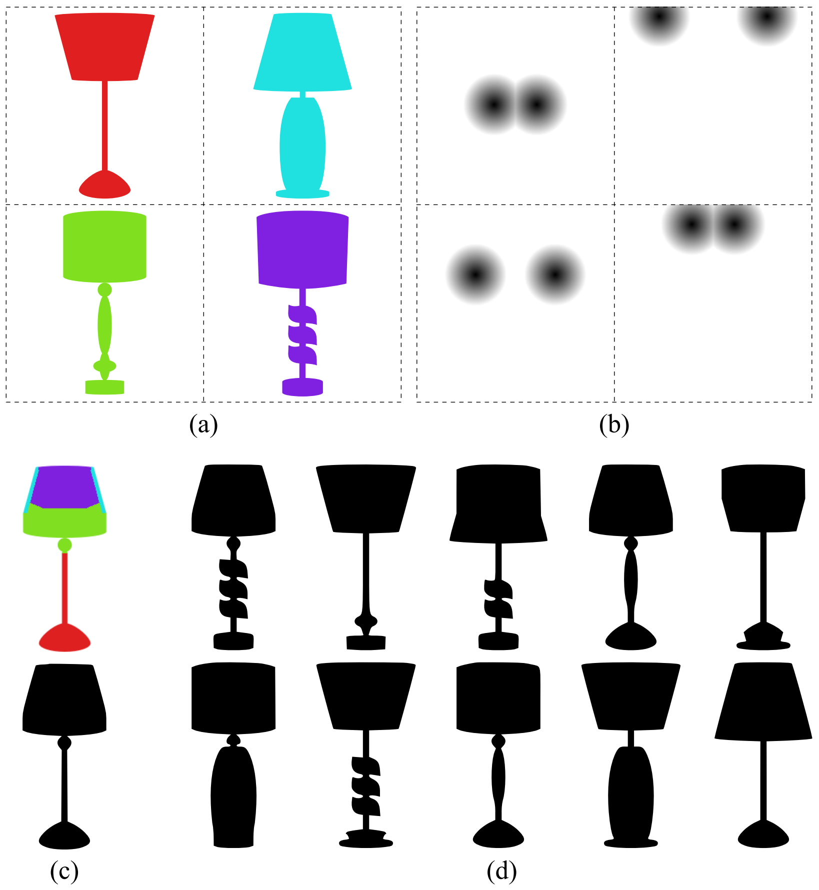


Figure 4.12: Pictogram hybrids. Starting from four existing pictograms (a), our system generates random data costs (b) as a guidance for graphcuts, to produce a hybrid pictogram (c). More hybrids are shown in (d). In this example, a horizontal symmetry constraint is enabled.

4.4.3 Pictogram Hybrids

Another way of using our system is to automatically synthesize hybrids of a collection of pictograms, inspired by Structured Image Hybrids [81]. Fig. 4.12 demonstrates an example. To synthesize hybrids, we randomly select a *seed position* from each exemplar in the collection to retain in the remixing result. In the case of symmetric shapes, we augment this to a pair of symmetric seed positions. We then compute truncated distance fields D_H from these seed positions (Fig. 4.12b), normalized by the truncation threshold $t = 40$ to ensure a maximum value equal to 1. We use the D_H as the data costs for graph-cuts. Intuitively, the pixels around seed positions have

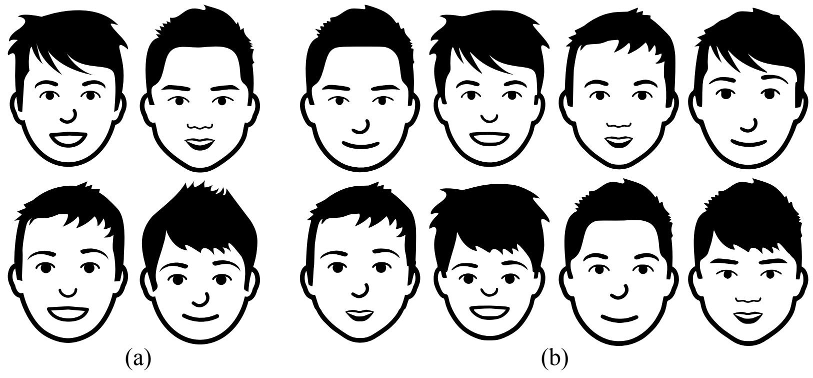


Figure 4.13: Pictogram hybrids: *boy*. (a) existing pictograms; (b) hybrids. A horizontal symmetry cost is enabled, but because the original exemplars have asymmetric hairstyles, the resulting hybrids are allowed to be asymmetric as well. In contrast, the eyes (which are symmetric in the input) are symmetric in the output.

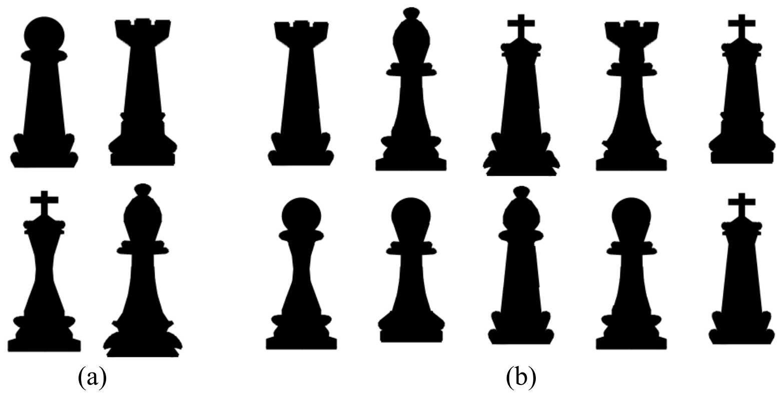


Figure 4.14: Pictogram hybrids: *chess* figures from [81]. (a) exemplar images; (b) hybrids. A horizontal symmetry constraint is enabled.

low data cost, and thus tend to be retained in the hybrid. The graph-cuts algorithm finds optimal cuts between these seed positions to remix exemplars.

Since the data cost is randomly generated, we need a strong smoothness constraint to ensure the quality of the remixing result. We find $\lambda_S = 3.0$ to be sufficient for all test cases. Needless to say, randomly generated data cost is not guaranteed to produce perfect results all the time. Also, different random seeds can also yield the same hybrids. Therefore, our results figures demonstrate 8 to 14 results from a larger set produced with 40 random seeds.

Fig. 4.13 shows a set of face hybrids (see also Fig. 4.1c). In order to maintain semantics, we enabled the horizontal symmetry constraint. However, in contrast to Structured Image Hybrids [81], we only require the *source positions of pixels* (as opposed to the pixels themselves) to be symmetric. Therefore, while our algorithm is able to maintain semantics, our system also retains the asymmetry of hair that is present in the original pictograms.

Fig. 4.14 demonstrates hybrids of the same chess exemplars used in the Structured Image Hybrids [81] paper. Our system achieves subjectively comparable quality.

4.4.4 Pictogram Montage

In some cases, users would like more direct control over how exemplars are aligned and remixed. In this scenario, our system resembles a pictogram-optimized instance of Photomontage [2], which we call *pictogram montage*. Fig. 4.15 demonstrates an example. We ask the user to select source pictograms, align them, then use strokes to select which parts to retain in the remixing results. We compute truncated distance fields D_M from the pixels covered by the strokes, truncated with threshold $t = 20$ and normalized to a maximum value of 1. We then use D_M as the data cost for graphcuts. We find that $\lambda_S = 3.0$ works well for all test cases. Fig. 4.16 demonstrates two more examples.

4.5 Evaluation

Our primary evaluation is qualitative: we ask the reader to refer to the figures in this chapter as well as supplemental materials for examples of all four workflows. In addition, we conducted two studies to evaluate several aspects of our system. First, we evaluate how often our algorithm is able to create a high-quality pictogram from a user-drawn sketch. Second, we compare the visual quality of algorithmically remixed

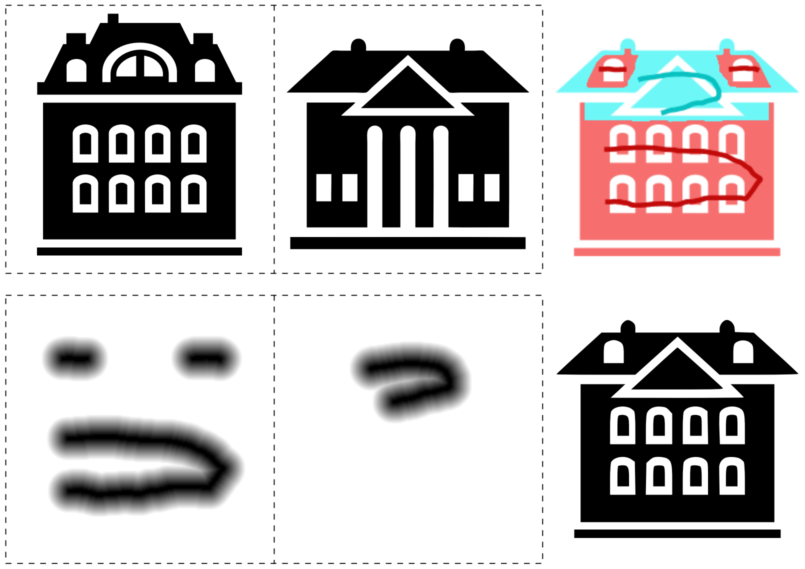


Figure 4.15: Pictogram montage. The user aligns pictograms of buildings and selects parts with strokes (top row). Our system generates data costs based on the strokes (bottom left), and generates a pictogram montage (bottom right).

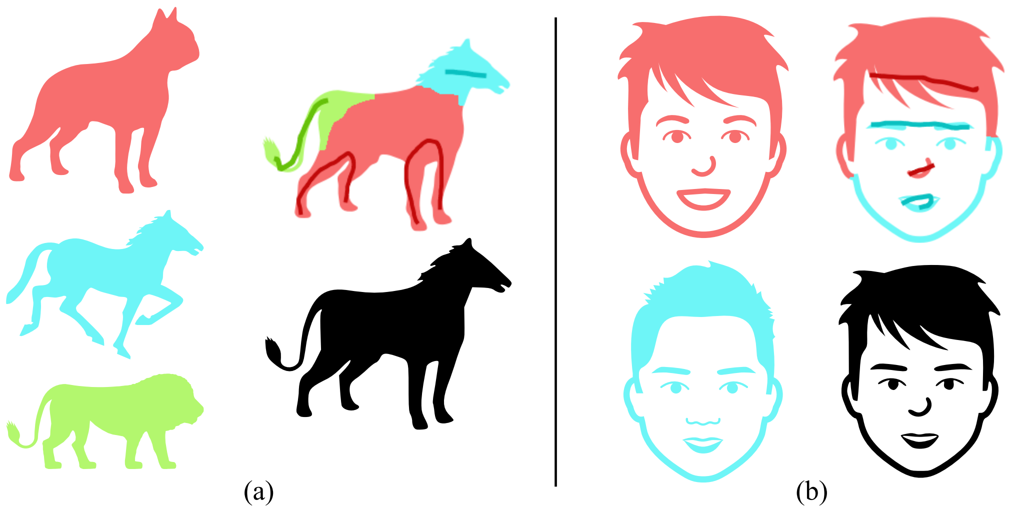


Figure 4.16: Two more examples of pictogram montage.

pictograms against original, hand-drawn icons, to see if humans prefer one over the other.

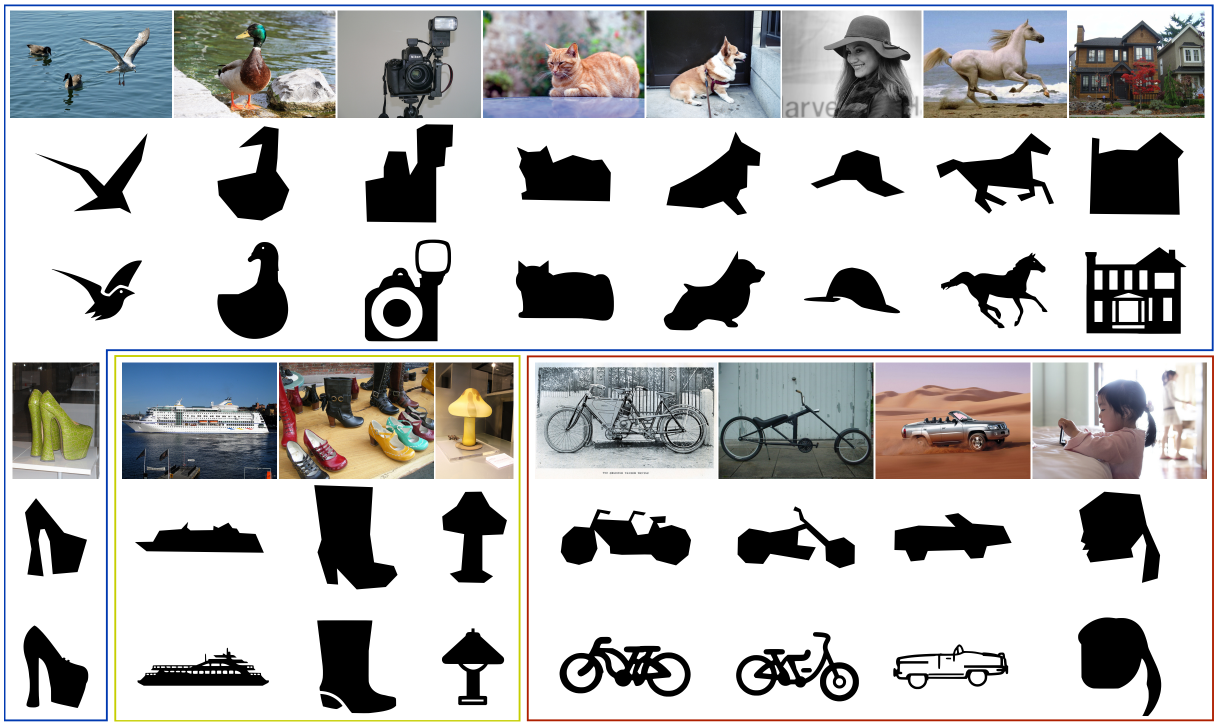


Figure 4.17: Semi-blind test results. Row 1: photo. Row 2: the user's sketch polygon. Row 3: the best pictogram generated by our algorithm. Blue box: test cases in which no existing pictograms match the user's intent, but our algorithm generates a matching pictogram by remixing. Yellow box: test cases in which our algorithm obtains a reasonable pictogram, but there are existing single pictograms that match the user's intent. Red box: test cases in which our algorithm fails.

4.5.1 A Semi-Blind Test

We first conducted a "semi-blind" test on our sketch-based modeling workflow, in which the authors produced the inputs to the pipeline. There are two reasons why we did this, instead of asking external users. First, we wished to avoid the difficult question of intent, i.e., whether the results match what the users had in mind. Instead, we only evaluate the quality of results. Second, as shown in Sec. 4.5.3, our system currently is not able to match and remix in real time, which makes it difficult to have users use our system.

For this test, we produced 16 input sketches, covering a diversity of subjects seen in reasonably canonical views. Fig. 4.17 shows the results of our algorithm on those sketches. Our algorithm produced plausible and recognizable pictograms 75% of the time, and in more than 50% of cases (in the blue box) no existing pictograms matched the sketch polygons (though this determination is necessarily subjective). We include the best match and other details in supplemental materials. Even when there was an existing matching pictogram, it can be valuable to see alternatives; in several cases (e.g., *boot* and *lamp*) our remixed pictograms have a different appearance or style that the user may prefer.

4.5.2 User Study on Visual Quality of Results

We also conducted a large-scale human study on Amazon Mechanical Turk to evaluate remixing quality. We choose to evaluate the *pictogram hybrids* workflow, as it is the only workflow without user control, and hence avoids the difficult-to-evaluate questions of user intent mentioned above. We asked each Turker to compare 30 pairs of pictograms for each of three test cases: *boy* (Fig. 4.1c and Fig. 4.13), *lamp* (Fig. 4.12), and *chess* (Fig. 4.14). We used a forced-choice methodology, in which Turkers had to select their preferred icon within each pair. Among these pairs, 24 of them compare the 4 exemplars against all the results shown in this chapter, while the remaining 6 pairs are sanity checks that compare poorly-synthesized pictograms against results shown in this chapter. We obtained the poorly-synthesized icons by using a very low weight (0.001) smoothness cost. Any submission that fails to give a correct answer for any sanity-check pair is discarded. This way, a randomly-guessing user has only a 1.6% chance of passing our sanity check.

Our study received 139 submissions from Turkers in the United States, of which 122 passed the sanity check. Each pair was compared by 20 to 24 users who passed our sanity check. Table 4.1 shows the results. We observe that about 45% of the time our synthesized icons were preferred, which is close to the 50% that we would expect to observe if the synthesized icons were of completely equivalent quality to the original hand-drawn ones. We thus conclude that, while there is a small difference, it is difficult for users to distinguish between our results and stock pictograms.

Table 4.1: Results of user study evaluating visual quality. We show the number of user selections preferring our results vs. stock pictograms, as well as the percentage preferring our results.

Test Case	**Ours**	**Stock**	**Ours%**
boy	512	659	43.7%
lamp	414	514	44.6%
chess	385	444	46.4%

4.5.3 Running Time

Our algorithm is implemented in Python, with the Fast Fourier Transform implemented via a Python binding for `FFTW` [28]. Graph-cuts are solved via the α-expansion algorithm in the `GCO` package [7]. For sketch-based workflows, the bottleneck of our algorithm is sliding-window matching. On a 3.6GHz CPU, with a single CPU thread, it takes about 13 msec to match a retrieved pictogram at one scale and reflection for each salient region. Therefore, for 200 retrieved pictograms, 3 different scales, and counting reflections, it takes about 140 seconds to perform sliding-window matching for all 9 salient regions. The multi-label graph-cuts algorithm takes 3 to 11 seconds with a single CPU thread, which means that even if we try 12 values for λ_S, the total running time for graph-cuts is roughly the same as for sliding-window matching. Therefore, in our implementation, after we perform sliding-window matching, we run the graph-cuts algorithm with a set of different values for λ_S, ranging from 0.01 to 3.0, and ask the user to choose the desired result. Both the matching and remixing stages of our algorithm are trivially parallelizable, and we would expect significant speedups with parallel or GPU implementations.

4.6 Limitations

Our algorithm has several failure modes, mainly in sketch-based workflows. First, as for almost all data-driven algorithms, our algorithm can fail due to insufficient data. In particular, most artists draw pictograms only in canonical views. If a novice user sketches a polygon over a photo with an arbitrary view of an object, our algorithm is unlikely to generate reasonable results. Even given a canonical view, if there are no existing pictograms which even partially match the user's sketch polygon, our algorithm will fail. The "chopper" bicycle in Fig. 4.17 is a typical case — it was designed and built by the person who took the photo, so none of the normal bicycles in the repository are similar to it. The algorithm will also fail for particular styles (i.e., blackness intervals) if the repository does not provide sufficient coverage for them. We expect that these problems will be gradually alleviated as more and more artists contribute to online repositories of icons.

Second, we expect users to describe the shape of their desired pictogram with a polygon. Because the polygon is rough, we detect salient regions by using large windows, and use TSDF to measure the similarity between the polygon and the exemplars. These technical decisions make our algorithm robust to small changes and noise on the polygon, but on the other hand make our algorithm insensitive to small details. The girl in Fig. 4.17 is a typical case: the polygon has details to represent the girl's nose and mouth, but our algorithm fails to retain these parts in the result. Giving even higher weight to fine details in the sketch, or allowing users to sketch interior detail, might help alleviate some of these problems.

Third, though modest scaling factors 0.9 and 1.1 are sufficient for sliding-window matching in most cases, there are exceptions. The tandem bicycle case in Fig. 4.17 is a typical case — it requires an aggressive scaling of wheels to fit two seats in between. We re-ran this example with a more aggressive set of scaling factors (0.8, 0.9, 1.1, and 1.25), giving the correctly remixed result in in Fig. 4.18. We do not see this as

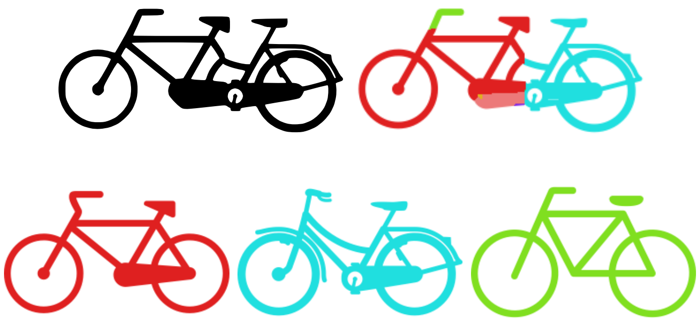


Figure 4.18: Remixing result for the tandem bicycle, if we allow a larger set of scaling factors for matching.

a general solution, though, since adding these additional scaling factors sometimes yields visually implausible results.

4.7 Conclusion and Discussion

In this chapter, we propose a system that enables users to generate a variety of pictograms. Our system is built on a large online pictogram repository. It combines icon-specific algorithms for salient-region detection, shape matching, and multi-label graph-cut stitching, which unify four different workflows under the same framework. We demonstrate that our system is able to produce results in various styles of quality similar to stock pictograms.

The system in this chapter asks users to use a keyword to define the desired pictogram, which is same as in Ch. 3. However, since this system is designed for data content *creation* rather than enhancement, a keyword is often not enough. Therefore, we design four workflows to satisfy users' needs in different scenarios. In sketch-based pictogram modeling and pictogram editing, we ask users to draw a sketch, which is exploited by our sliding-window search to find good candidates for remixing. In pictogram hybrids, we ask users to provide candidate pictograms by themselves. In pictogram montage, we further ask users to align the candidate pictograms. Generally, user intervention is increasing from the first to the last workflow, giving users more and more control on data retrieval.

As of today, noise is not a big problem on The Noun Project. Since the pictograms are tagged by the contributors, who earn money from the repository only when their pictograms get purchased, the tags are very clean.

Pictograms in different styles are very difficult to remix. We develop a simple yet effective metric, *blackness*, to group pictograms in similar styles. For each style in a specific task, as long as there are sufficient exemplar pictograms in the style, a reasonable remixing result can typically be generated. Similarly to our keyword-based image stylization system in Ch. 3, diversity of our results is greatly improved thanks to the disagreement in data.

Chapter 5

Conclusion, Discussion, and Future Work

In this thesis, we describe three data-driven image creation and enhancement systems.

First, based on our massive online user study, we train a no-reference metric to evaluate the perceptual quality of images produced by motion deblurring algorithms. Our metric exploits the disagreement in users' opinion to measure the similarity between quality of deblurring results. Extensive experiments demonstrate that our crowd-sourced user study is robust to noise, and our metric matches users' opinions. Three applications demonstrate that our metric helps users select deblurring algorithms, tune parameters, and improve quality of deblurring results.

Second, by leveraging the online image search engines, we develop a keyword-based system to automatically stylize photos. We model stylization as matching histogram of input image to range of counts in histogram bins of the retrieved image collection, which is simple yet robust to noise. We use clustering to filter out noise in data, and improve diversity of stylization results. Experiments and user studies demonstrate that our system is able to robustly transfer the user desired style to photos.

Third, we develop a system that synthesizes novel pictograms by remixing portions of icons retrieved from a large online repository. Our system combines icon-specific algorithms for salient-region detection, shape matching, and graph-cut stitching, and unifies four different workflows under the same framework. By clustering on our developed *blackness* values, our system is able to produce results in four styles.

5.1 Revisiting the Key Problems

Data retrieval. In all our projects, we retrieve data from the Internet. Based on the nature of the problem, we design appropriate strategies for data retrieval. Generally, for relatively objective problems (Ch. 2), most of the data are retrieved in the stage of system design, to reduce the end users' effort, and make our system fit most users' opinions. For subjective problems (Ch. 3 and Ch. 4), data are retrieved with a limited amount of input provided by the end user, to enable our system to generate the results that fit the user intent. We use keyword search to retrieve data, because a keyword serves as a concise semantic representation of user intent, and keyword-indexed data are widely available on the Internet. For very subjective problems (Ch. 4), we use multi-modal user input (such as combination of keywords and sketches), to facilitate users to fully describe their needs. We also use another strategy, which is clustering the retrieved the data into several groups, generating results for all groups, and asking users for relevance feedback, to effectively help users generate results that match their intent.

Noise. Our experiences in Ch. 2 and Ch. 3 demonstrate that data retrieved on the Internet are often noisy, but it is possible to reduce noise. First, we can explicitly filter out noise: in Ch. 2, we screen users on their records, and reject submissions from malicious users via sanity check; in Ch. 3, we eliminate very small clusters. Second, data redundancy helps us protect our systems from noise: in Ch. 2, we ensure each pair of images are compared by at least 20 users; in Ch. 3, we retrieve the first top *500* images for each keyword. Third, simple statistical models help us enhance robustness of our systems: in Ch. 2, we formulate our metric with a linear model; in Ch. 3, we model the color and contrast distribution of an image collection with a histogram with range of counts in bins. The simplicity of these models prevents our systems from overfitting to noise. We also observe that these simple models perform effectively in our systems, thanks to the big amount of data on the Internet.

Disagreement in data. We observe disagreement in data in all our projects. Simply eliminating data with disagreement would apparently lead to a big loss of data. In Ch. 3, we further demonstrate that using these data in a naive way would inevitably lead to unsatisfactory results. Therefore, we design several methods to handle this disagreement: in Ch. 2, we adopt the Bradley-Terry model, which exploits the disagreement in users' opinions as a measure of similarity in quality of deblurring results; in Ch. 3 and Ch. 4, the collected data are clustered to enhance the compatibility within clusters and improve the diversity of results. Our results demonstrate that with a careful algorithm design, our systems benefit from disagreement in data.

5.2 Discussion and Future Work

Internet Data vs. Data Collected in Controlled Settings. In this thesis, we collect data from the Internet. In comparison to approaches in which data are collected in controlled settings, it has several advantages to collect data from the Internet. First, it is clearly more scalable and more cost effective. Second, for crowdsourced user studies, as we discussed in Ch. 2, the variance in display settings makes the study result better generalize to real-world settings. Third, adoption of image search engine and indexed pictogram repository in Ch. 3 and Ch. 4 enables our system to keep up-to-date on retrieval algorithms and image data for free. These advantages, however, do not mean that controlled data collection is not useful any more. Data collected in the controlled setting reduces impact of factors such as display settings, which is important for fundamental research topics such as the human visual system. Data collected in controlled settings are also much cleaner, which is helpful for locating bugs in data-driven algorithms on early stages, before they are deployed on large-scale systems. Therefore, we think the Internet-based data collection and controlled data collection are complementary to each other: it is better to collect data in controlled settings for fundamental research; practical systems used in real world would prefer Internet-based data collection, while small-scale data collected in controlled settings can be used for pilot studies on early stage of development.

Data Retrieval: Keyword Search vs. Visual Search In Ch. 3 and Ch. 4, we retrieve data via keyword search. Keyword search has its strength in simplicity, computational efficiency, and ability to encode semantics, but also has its own problems. First, despite the wide availability of keyword-indexed data, there are far more non-indexed or poorly-indexed image data on the Internet. For example, when uploading a collection of vacation photos to Flickr, since it is tedious to carefully label each photo, we often instead label all the photos with the same set of keywords, or even just skip the keyword labeling step. These image data cannot be easily used by keyword search. Second, in some scenarios, keywords are not powerful enough to facilitate users to precisely express their intent. For example, every once in a while, we are attracted by the artistic style in a breathtaking photo, and really want to transfer it to our own photos, but often we have no idea what keyword describes the style. Visual search may be a good remedy for these two problems. To deploy visual search to retrieve the Internet image data in practical systems, there are two critical problems that have to be well solved. First, how to design a paradigm for user interaction in visual search, which can be easily learned by novice users, while also closing the semantic gap between image features and user intent as much as possible? Second, when keywords are absent, how to design algorithms and data structures to index a huge amount of visual data to enable fast retrieval? We think these two long-standing problems are interesting and challenging to explore in the future.

User-Guided Data Retrieval vs. Data-Guided User Intent. In Ch. 3 and Ch. 4, we exploit data to help users enhance and create images. Our systems assume users have clear intent in their mind, and generate results to match the user intent. This paradigm has two problems: first, instead of obtaining results to match their intent, novice users often would like to get inspired for intent; second, sometimes there are just not sufficient data to match the user intent. For instance, in Ch. 4, it might happen that none of the existing pictograms can generate remixing results that resemble the input sketch. It would be interesting and useful to develop systems that *guide* users based on the existing data, for instance, a system which is able to automatically suggest suitable stylization filters given an input photo, or a system guiding users to draw icons, which is similar to *ShadowDraw* [54].

Open-Loop vs. Closed-Loop. All our systems in this thesis are open-loop systems, as only data from the Internet are used. An interesting future direction is to close the loop, i.e., use both the data from the Internet and the feedback from users. For instance, it is useful to develop a system which allows users to make corrections in case they find two of deblurring results on their photo are ranked incorrectly, and updates the quality metric accordingly. It is also interesting to design a system that allows users to 'repair' the artifacts in iconification results, and exploits the user edits to detect and repair artifacts in the future results. It would be challenging yet interesting to explore how to design the user feedback, and how to compromise the Internet data and the user feedback.

Appendix A

Implementation Details Of Quality Metric

This chapter contains the implementation details of our no-reference quality metric for motion deblurring, described in Ch. 2.

A.1 Collection of Features

A.1.1 Prerequisite

First, we introduce a new vector norm called L_p *mean norm*, which will be extensively used to define features. Given a vector v of dimension n, the standard L_p norm of v is:

$$L_p(v) = \Big(\sum_i^n |v_i|^p \Big)^{\frac{1}{p}}, \tag{A.1}$$

whose scale depends on n. In our problem, however, we found that deblurring algorithms often cut off the image border after applying deconvolution to remove the well-known boundary artifacts [61], and the size of the cut-off region varies with the size of the PSF and the specific algorithm being applied. Therefore, we propose to use a modified L_p norm to remove the effect of n as:

$$L_{mp}(v) = \left(\frac{\sum_i^n |v_i|^p}{n} \right)^{\frac{1}{p}}. \tag{A.2}$$

Note that $p = 1$ induces the average of absolute values, and $p = 2$ induces the root mean square.

A.1.2 Noise

Two-color prior. We use k-means clustering ($k = 2$) to find the two most prevalent colors within a 5×5 neighborhood centered at each pixel, and measure the distance from each pixel's color to the line connecting the two colors as the fitting error. Finally, we compute the L_p mean norm of errors of all pixels as a noise feature. The parameter p is set to 1.3, which was chosen based on our user study data.

Sparsity prior. Sparsity priors [57, 50] assume that gradient magnitudes of natural images follow a heavy-tailed distribution. When heavy noise is present, the distribution changes significantly. We measure the L_p mean norm of the gradient magnitudes as another noise measure with with the same setting $p = 0.66$ as in the implementation from Krishnan et al. [50].

Gradients of small magnitudes. As the power-law of amplitude spectra [26] suggests, low-frequency components dominate natural images. Therefore, there should be a considerable amount of gradients in relative flat areas. Also, variations in flat areas are mostly caused by noise, thus we can measure the noise amount by measuring variation in flat regions. Specifically, we find the top $m = 30\%$ of smallest gradient magnitudes. The average variance of gradient magnitudes at these pixels is then used as a noise measure.

MetricQ. MetricQ [109] is a no-reference metric, which is sensitive to both noise and blur. This measure is based on the fact that blur and noise transform an anisotropic patch into a more isotropic one. The measure first finds anisotropic patches by inspecting the two eigenvalues s_1 and s_2 (where $s_1 \geq s_2$) of structure tensors. Then, $Q = s_1(s_1 - s_2)/(s_1 + s_2)$ measures the noise and blur in each anisotropic patch. The lower the mean value of Q over all anisotropic patches, the more blurry and noisy the whole image is considered to be. We use the publicly available Matlab code[1] from the authors, with their default setting PatchSize = 8.

BM3D. We use a threshold $T_c = 0.01$, which was chosen based on our user study data. We also utilize BM3D results as inputs for ringing and sharpness features to reduce the effects of noise.

A.1.3 Ringing

Detailed steps of our full-reference ringing detector. As in Ch. 2.4.1, we denote the ground truth and the deblurred image by l and l', respectively.

1. Align l' with l by seeking a translation of l' that maximizes the correlation between the two images. Then, apply a Gaussian filter with $\sigma = 2.5$ to both l and l' to remove noise and high-frequency details (note that ringing is preserved in l' as it is mid-frequency);

2. Compute the gradient magnitude of l and l' as $g(l)$ and $g(l')$, and compute the difference map $\delta g = g(l) - g(l')$;

3. For a pixel x, set $\delta g(x) \leftarrow \max(\delta g(x), 0)$;

4. Compute the mean value of δg as the indicator of the strength of the ringing artifacts in l'.

[1]`<http://users.soe.ucsc.edu/~xzhu/doc/metricq.html>`

Note that in step 1, we apply the same low-pass filter to both l and l' to remove high-frequency signals. If this step is eliminated, then the computed difference map δg in step 2 may contain high-frequency components caused by image noise and small misalignment between the two images, which makes extracting ringing artifacts harder. Applying a low-pass filter can effectively remove these high-frequency outliers from the difference map, resulting in a cleaner separation of ringing artifacts.

Step 3 removes all negative values from δg, motivated by the observation that a pixel corresponding to ringing artifacts should have a larger gradient magnitude in l' than in l.

A.1.4 Sharpness

An ideal deblurred image should be sharp and contain no residual blur. However, if a PSF is not estimated accurately, there is usually a fair amount of residual blur remaining in the deblurring result, as well as over-smoothing caused by regularization in the deconvolution process. We measure them with four sharpness features from recent publications.

Autocorrelation. To reliably measure the residual blur, we assume that it is roughly static across the whole image; i.e., all pixels have the same amount of residual blur. Under this assumption, we calculate the autocorrelation of the image gradients to measure the residual blur. For a sharp image, the autocorrelation map should have a shape similar to a delta function due to the power law [33]. If the image contains motion blur, then the center dot becomes an elongated shape that is similar to the autocorrelation of the motion PSF. However, this can be violated when there are long straight edges in the image, which may cause long straight lines along the same directions in the autocorrelation maps. To mitigate this problem, we use the Hough transform to detect them before computing the autocorrelation map.

Specifically, we first convert the image into grayscale, and then use the Hough transform to remove long straight lines in the gradients. Next, we compute two autocorrelation maps for its horizontal and vertical gradients. This results in two autocorrelation maps. Given that the residual motion blur should not be bigger than the minimum blur we observe, we compute a minimal autocorrelation map by choosing the minimal value at each position from the maps.

Next, we create a series of circles centered at the map center with radii of 1, . . . , r_{max}, dividing the map into r_{max} rings. In our system, r_{max} is set to 51, which is a safe upper bound for residual motion blur. We then find the maximum value in each ring, and compute the average of them as the sharpness measurement. Effectively, this feature measures how "spread out" the autocorrelation map is.

Cumulative Probability of Blur Detection (CPBD). CPBD [74] utilizes the theory that if the width of an edge is below a threshold named "just noticeable blur" (JNB) width, then the blur can be considered not detected. The more edges there are of this type, the sharper the image is. Specifically, this feature finds all patches containing edges, computes the JNB width based on the local contrast of each patch, and compares it with the width of edges in the patch. The proportion of the edges with width below the JNB width represents how sharp the image is.

Local Phase Coherence (LPC). It has been discovered that blur disrupts the coherence of local phase information across different scales [102]. Building upon this result, LPC [35] computes the local phase coherence of each pixel in a 3-scale 4-orientation complex wavelet pyramid. The local phase coherence then is weighted averaged to a single value. The weights are higher in sharper regions, to handle the possible shallow depth of field in natural images.

Normalized Sparsity. The normalized sparsity measure [49] is computed as the ratio between the L_1 norm and the L_2 norm of image gradients. Intuitively, when blurriness increases, L_2 norm increases faster than L_1 norm. Therefore, the lower the ratio is, the sharper the image is. In addition, since there is an L_1 norm in the nominator, this prior is also resistant to over-sharpening.

A.1.5 Parameter Selection for Individual Features

There are a number of parameters in the individual features proposed above. We use our collected user study data set to find a good parameter setting for these features. To do this for each individual feature, we compute its value on each training image, and rank all images based on the values of this feature. This ranking is then compared with the ground-truth ranking to compute a ranking difference. A smaller ranking difference indicates better performance of the feature. We then conduct an exhaustive search on the parameters of the feature to find the optimal setting yielding the minimal ranking difference.

A.2 The Perceptual Metric

A.2.1 Evaluating Features on Single-Distortion Images

Fig. A.1 shows the 8 PSFs used in our single-distortion data sets. Specifically, the noise data set only uses PSF (7), because the noise in the deblurring results usually does not depend on the shape of PSF. Both the blur and ringing data sets use all 8 PSFs. These 8 PSFs are all from Levin et al. [56].

A.2.2 Evaluating Features on User Study Data

In addition to the experiments shown in Ch. 2.5.2, we conducted an experiment to see the performance of each feature on the user study data set. We measured the performance of each feature using the ranking metric proposed in Ch. 2.5.1 on the

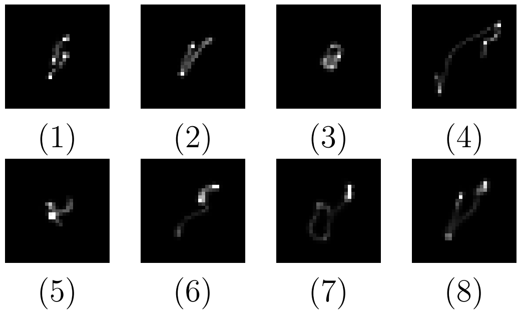


Figure A.1: The 8 PSFs used in our experiments.

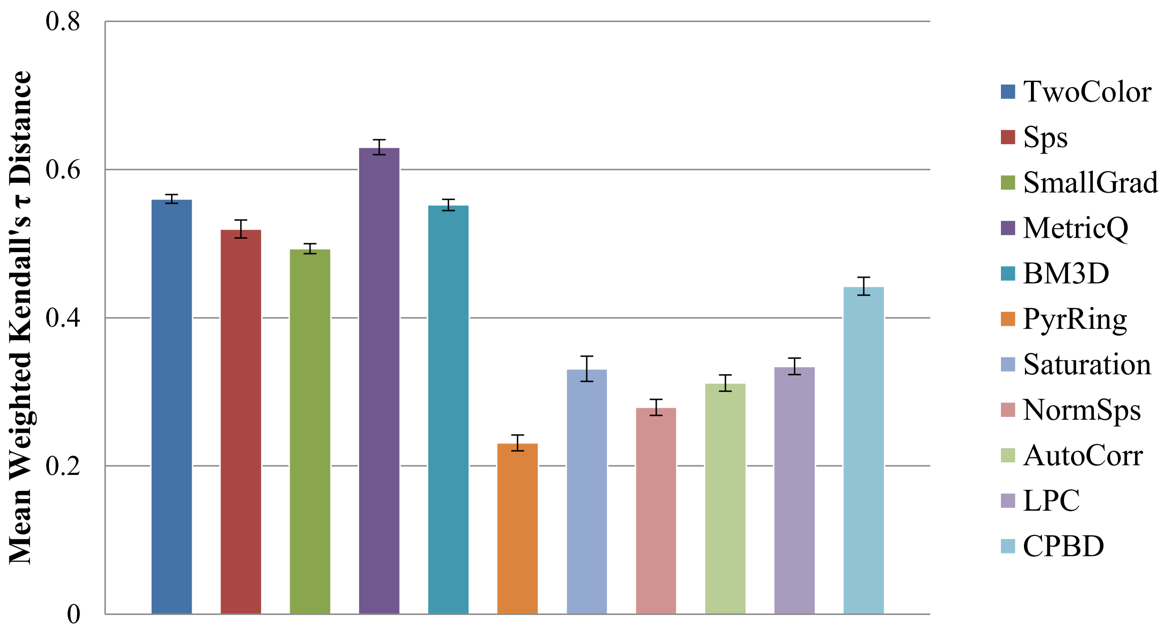


Figure A.2: The mean weighted Kendall's τ distance between each individual feature and the Bradley-Terry model (lower is better). The error bars indicate the standard error of the distance.

user study data set, and the results are shown in Fig. A.2. It shows that the PyrRing feature has the best performance. This is consistent with the qualitative observations on the user study results in Ch. 2.3.4, which indicate that human perception is most sensitive to ringing artifacts.

A.3 Details of Automatic Parameter Selection

The two PSFs used in our automatic parameter selection data set are PSF (1) and (8) in Fig. A.1. Note that they are different from the PSFs used in our user study data set (PSF (4) and (7)), for a fair comparison with other metrics.

Appendix B

Accelerating Matching with the FFT

We specialize the matching error (4.4) discussed in Ch. 4.3.3 to the case of pure translation, and split it into three terms:

$$\delta(u,v) = \sum_{x,y} M_i(x,y)\big(D_j(u+x,v+y) - D_P(x,y)\big)^2 \tag{B.1}$$

$$= \delta_1(u,v) - 2\,\delta_2(u,v) + \delta_3(u,v) \tag{B.2}$$

$$\delta_1(u,v) = \sum_{x,y} M_i(x,y)\, D_j(u+x,v+y)^2 \tag{B.3}$$

$$\delta_2(u,v) = \sum_{x,y} M_i(x,y)\, D_P(x,y)\, D_j(u+x,v+y) \tag{B.4}$$

$$\delta_3(u,v) = \sum_{x,y} M_i(x,y)\, D_P(x,y)^2 \tag{B.5}$$

We may think of δ_1 as filtering D_j^2 with M_i, and δ_2 as filtering D_j with $M_i D_P$. These operations can be accelerated with the Fast Fourier Transform. δ_3 is a constant that has no effect on the relative ordering of matching scores, and may be omitted.

Bibliography

[1] Adobe. Adobe Photoshop Lightroom, 2007.

[2] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 23(3):294–302, August 2004.

[3] Apple. Apple Aperture, 2005.

[4] Soonmin Bae, Sylvain Paris, and Frédo Durand. Two-scale tone management for photographic look. *Proc. SIGGRAPH 2006*, 2006.

[5] Pascal Barla, Joelle Thollot, and François X. Sillion. Geometric clustering for line drawing simplification. In *Proc. Eurographics Workshop on Rendering*, pages 183–192, June 2005.

[6] Nicolas Bonneel, Kalyan Sunkavalli, Sylvain Paris, and Hanspeter Pfister. Example-based video color grading. *Proc. SIGGRAPH 2013*, 2013.

[7] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. PAMI*, 23(11):1222–1239, November 2001.

[8] Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4), 1952.

[9] Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. In *Proc. CVPR 2011*, June 2011.

[10] Martin Cadik, Robert Herzog, RafałMantiuk, Karol Myszkowski, and Hans-Peter Seidel. New measurements reveal weaknesses of image quality metrics in evaluating graphics artifacts. *ACM Trans. Graphics*, 31(6), November 2012.

[11] J.C. Caicedo, A. Kapoor, and Sing Bing Kang. Collaborative personalization of image enhancement. In *Proc. CVPR 2011*, 2011.

[12] Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun. Probabilistic reasoning for assembly-based 3D modeling. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 30(4):35:1–35:10, July 2011.

[13] Tao Chen, Ming-Ming Cheng, Ping Tan, Ariel Shamir, and Shi-Min Hu. Sketch2Photo: Internet image montage. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 28(5):124:1–124:10, December 2009.

[14] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3D mesh segmentation. *ACM Trans. Graphics*, 28(3), 2009.

[15] Sunghyun Cho and Seungyong Lee. Fast motion deblurring. *ACM Trans. Graphics*, 28(5), 2009.

[16] Taeg Sang Cho, S. Paris, B.K.P. Horn, and W.T. Freeman. Blur kernel estimation using the Radon transform. In *Proc. CVPR 2011*, june 2011.

[17] Daniel Cohen-Or, Olga Sorkine, Ran Gal, Tommer Leyvand, and Ying-Qing Xu. Color harmonization. *Proc. SIGGRAPH 2006*, 2006.

[18] Forrester Cole, Kevin Sanik, Doug DeCarlo, Adam Finkelstein, Thomas Funkhouser, Szymon Rusinkiewicz, and Manish Singh. How well do line drawings depict shape? *ACM Trans. Graphics*, 28(3), 2009.

[19] K.. Dabov, A.. Foi, V.. Katkovnik, and K.. Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Trans. Image Processing*, 16(8), 2007.

[20] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR 2009*, 2009.

[21] T. Deselaers and V. Ferrari. Visual and semantic similarity in imagenet. In *Proc. CVPR 2011*, 2011.

[22] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei A. Efros. What makes paris look like paris? *ACM Transactions on Graphics (SIGGRAPH)*, 31(4):101:1–101:9, 2012.

[23] Facebook. Instagram, 2010.

[24] Rob Fergus, Barun Singh, Aaron Hertzmann, Sam T. Roweis, and William T. Freeman. Removing camera shake from a single photograph. *ACM Trans. Graphics*, 25(3), 2006.

[25] David J. Field and Nuala Brady. Visual sensitivity, blur and the sources of variability in the amplitude spectra of natural scenes. *Vision Research*, 37(23), 1997.

[26] David J Field et al. Relations between the statistics of natural images and the response properties of cortical cells. *J. Opt. Soc. Am. A*, 4(12), 1987.

[27] D. Freedman and P. Kisilev. Object-to-object color transfer: Optimal flows and SMSP transformations. In *Proc. CVPR 2010*, 2010.

[28] M. Frigo and S.G. Johnson. The design and implementation of FFTW3. *Proc. IEEE*, 93(2):216–231, February 2005.

[29] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 23(3):652–663, August 2004.

[30] Elena Garces, Aseem Agarwala, Diego Gutierrez, and Aaron Hertzmann. A similarity measure for illustration style. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 33(4):93:1–93:9, July 2014.

[31] Chen Goldberg, Tao Chen, Fang-Lue Zhang, Ariel Shamir, and Shi-Min Hu. Data-driven object manipulation in images. *Computer Graphics Forum (Proc. Eurographics)*, 31(2):265–274, May 2012.

[32] D.B. Goldman and Jiun-Hung Chen. Vignette and exposure calibration and compensation. In *Proc. ICCV 2005*, 2005.

[33] Amit Goldstein and Raanan Fattal. Blur-kernel estimation from spectral irregularities. In *Proc. ECCV 2012*, 2012.

[34] Ankit Gupta, Neel Joshi, Lary Zitnick, Michael Cohen, and Brian Curless. Single image deblurring using motion density functions. In *Proc. ECCV 2010*, 2010.

[35] R. Hassen, Zhou Wang, and Magdy Salama. No-reference image sharpness assessment based on local phase coherence measurement. In *Proc. ICASSP 2010*, 2010.

[36] James Hays and Alexei A. Efros. Scene completion using millions of photographs. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 26(3):4:1–4:7, July 2007.

[37] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proc. SIGGRAPH 2001*, 2001.

[38] Joseph M. Hilbe. *Logistic Regression Models*. Chapman & Hall/CRC Press, 2009.

[39] Thomas Hurtut and Pierre-Edouard Landes. Synthesizing structured doodle hybrids. In *SIGGRAPH Asia 2012 Posters*, page 43:1, 2012.

[40] Arjun Jain, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel. Exploring shape variations by 3D-model decomposition and part-based recombination. *Computer Graphics Forum (Proc. Eurographics)*, 31(2):631–640, May 2012.

[41] Hui Ji and Kang Wang. A two-stage approach to blind spatially-varying motion deblurring. In *Proc. CVPR 2012*, 2012.

[42] N. Joshi, R. Szeliski, and D.J. Kriegman. PSF estimation using sharp edge prediction. In *Proc. CVPR 2008*, 2008.

[43] N. Joshi, C.L. Zitnick, R. Szeliski, and D.J. Kriegman. Image deblurring and denoising using color priors. In *Proc. CVPR 2009*, 2009.

[44] Xie Jun, Hertzmann Aaron, Li Wilmot, and Winnemller Holger. PortraitSketch: Face sketching assistance for novices. In *Proc. UIST*, pages 407–417, 2014.

[45] Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):55:1–55:11, July 2012.

[46] Sing Bing Kang, A. Kapoor, and D. Lischinski. Personalization of image enhancement. In *Proc. CVPR 2010*, 2010.

[47] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2), 1938.

[48] Rolf Köhler, Michael Hirsch, Betty Mohler, Bernhard Schölkopf, and Stefan Harmeling. Recording and playback of camera shake: Benchmarking blind deconvolution with a real-world database. In *Proc. ECCV 2012*, 2012.

[49] D. Krishnan, T. Tay, and R. Fergus. Blind deconvolution using a normalized sparsity measure. In *Proc. CVPR 2011*, 2011.

[50] Dilip Krishnan and Rob Fergus. Fast image deconvolution using hyper-Laplacian priors. In *Proc. NIPS 2009*, 2009.

[51] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 22(3):277–286, July 2003.

[52] Pierre-Yves Laffont, Adrien Bousseau, Sylvain Paris, Frédo Durand, and George Drettakis. Coherent intrinsic images from photo collections. *Proc. SIGGRAPH 2012*, 2012.

[53] Gierad P. Laput, Mira Dontcheva, Gregg Wilensky, Walter Chang, Aseem Agarwala, Jason Linder, and Eytan Adar. PixelTone: A multimodal interface for image editing. In *Proc. SIGCHI 2013*, 2013.

[54] Yong Jae Lee, C. Lawrence Zitnick, and Michael F. Cohen. ShadowDraw: Real-time user guidance for freehand drawing. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 30(4):27:1–27:10, July 2011.

[55] Sylvain Lefebvre and Hugues Hoppe. Appearance-space texture synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 25(3):541–548, July 2006.

[56] A. Levin, Y. Weiss, F. Durand, and W.T. Freeman. Efficient marginal likelihood optimization in blind deconvolution. In *Proc. CVPR 2011*, 2011.

[57] Anat Levin, Rob Fergus, Frédo Durand, and William T. Freeman. Image and depth from a conventional camera with a coded aperture. *ACM Trans. Graphics*, 26(3), 2007.

[58] Yuanzhen Li, Lavanya Sharan, and Edward H. Adelson. Compressing and companding high dynamic range images with subband architectures. *Proc. SIGGRAPH 2005*, 2005.

[59] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[60] Albrecht Lindner, Appu Shaji, Nicolas Bonnier, and Sabine Süsstrunk. Joint statistical analysis of images and keywords with applications in semantic image enhancement. In *Proc. ACM Multimedia 2012*, 2012.

[61] Renting Liu and Jiaya Jia. Reducing boundary artifacts in image deconvolution. In *Proc. ICIP 2008*, 2008.

[62] Yiming Liu, Michael Cohen, Matt Uyttendaele, and Szymon Rusinkiewicz. Autostyle: Automatic style transfer from image collections to users' images. *Comput. Graph. Forum*, 33(4):21–31, July 2014.

[63] Yiming Liu, Jue Wang, Sunghyun Cho, Adam Finkelstein, and Szymon Rusinkiewicz. A no-reference metric for evaluating the quality of motion deblurring. *ACM Trans. Graph.*, 32(6):175:1–175:12, November 2013.

[64] Jingwan Lu, Connelly Barnes, Connie Wan, Paul Asente, Radomir Mech, and Adam Finkelstein. DecoBrush: Drawing structured decorative patterns by example. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 33(4):90:1–90:9, July 2014.

[65] Tianyang Ma and L. J. Latecki. From partial shape matching through local deformation to robust global shape similarity for object detection. In *Proc. CVPR*, pages 1441–1448, 2011.

[66] Rafat Mantiuk, Kil Joong Kim, Allan G. Rempel, and Wolfgang Heidrich. HDR-VDP-2: A calibrated visual metric for visibility and quality predictions in all luminance conditions. *ACM Trans. Graphics*, 30(4), 2011.

[67] Pina Marziliano, Frederic Dufaux, Stefan Winkler, and Touradj Ebrahimi. Perceptual blur and ringing metrics: Application to JPEG2000. *Signal Processing: Image Communication*, 19(2), 2004.

[68] Belen Masia, Lara Presa, Adrian Corrales, and Diego Gutierrez. Perceptually optimized coded apertures for defocus deblurring. *Computer Graphics Forum*, 31(6), 2012.

[69] A. Mittal, A. K. Moorthy, and A. C. Bovik. Automatic parameter prediction for image denoising algorithms using perceptual quality features. In *Proc. SPIE*, volume 8291, February 2012.

[70] A. Mittal, A.K. Moorthy, and A.C. Bovik. No-reference image quality assessment in the spatial domain. *IEEE Trans. Image Processing*, 21(12), 2012.

[71] A. Mittal, R. Soundararajan, and A. Bovik. Making a "completely blind" image quality analyzer. *IEEE Signal Processing Letters*, PP(99), 2012.

[72] A.K. Moorthy and A.C. Bovik. A two-step framework for constructing blind image quality indices. *IEEE Signal Processing Letters*, 17(5), 2010.

[73] Naila Murray, Sandra Skaff, Luca Marchesotti, and Florent Perronnin. Towards automatic concept transfer. In *Proc. NPAR 2011*, 2011.

[74] N.D. Narvekar and L.J. Karam. A no-reference image blur metric based on the cumulative probability of blur detection (CPBD). *IEEE Trans. Image Processing*, 20(9), 2011.

[75] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 2001.

[76] Maks Ovsjanikov, Wilmot Li, Leonidas Guibas, and Niloy J. Mitra. Exploration of continuous variability in collections of 3D shapes. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 30(4):33:1–33:10, July 2011.

[77] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graphics*, 22(3).

[78] François Pitié, Anil C. Kokaram, and Rozenn Dahyot. Automated colour grading using colour distribution transfer. *CVIU*, 2007.

[79] Tania Pouli and Erik Reinhard. Progressive histogram reshaping for creative color transfer and tone reproduction. In *Proc. NPAR 2010*, 2010.

[80] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE CG&A*, 2001.

[81] Eric Risser, Charles Han, Rozenn Dahyot, and Eitan Grinspun. Synthesizing structured image hybrids. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 29(4):85:1–85:6, July 2010.

[82] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *IJCV*, 2000.

[83] M.A. Saad, A.C. Bovik, and C. Charrier. A DCT statistics-based blind image quality index. *IEEE Signal Processing Letters*, 17(6), 2010.

[84] Ramin Samadani, Timothy A. Mauer, David M. Berfanger, and James H. Clark. Image thumbnails that represent blur and noise. *IEEE Trans. Image Processing*, 19(2), February 2010.

[85] Christian J. Schuler, Michael Hirsch, Stefan Harmeling, and Bernhard Schölkopf. Blind correction of optical aberrations. In *Proceedings of the 12th European conference on Computer Vision - Volume Part III*, Proc. ECCV 2012, pages 187–200, Berlin, Heidelberg, 2012. Springer-Verlag.

[86] Adrian Secord, Jingwan Lu, Adam Finkelstein, Manish Singh, and Andrew Nealen. Perceptual models of viewpoint preference. *ACM Trans. Graphics*, 30(5), 2011.

[87] Qi Shan, Jiaya Jia, and Aseem Agarwala. High-quality motion deblurring from a single image. *ACM Trans. Graphics*, 27(3), 2008.

[88] Dori Shapira, Shai Avidan, and Yacov Hel-Or. Multiple histogram matching. In *Proc. ICIP 2013*, 2013.

[89] L. Shapira, A. Shamir, and D. Cohen-Or. Image appearance exploration by model-based navigation. *Computer Graphics Forum*, 2009.

[90] H.R. Sheikh and A.C. Bovik. Image information and visual quality. *IEEE Trans. Image Processing*, 15(2), 2006.

[91] Chao-Hui Shen, Hongbo Fu, Kang Chen, and Shi-Min Hu. Structure recovery by part assembly. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 31(6):180:1–180:11, November 2012.

[92] Yichang Shih, Sylvain Paris, Frédo Durand, and William T. Freeman. Data-driven hallucination of different times of day from a single outdoor photo. *Proc. SIGGRAPH Asia 2013*, 2013.

[93] C. Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1), 1904.

[94] Kalyan Sunkavalli, Micah K. Johnson, Wojciech Matusik, and Hanspeter Pfister. Multi-scale image harmonization. *Proc. SIGGRAPH 2010*, 2010.

[95] Huixuan Tang, N. Joshi, and A. Kapoor. Learning a blind measure of perceptual image quality. In *Proc. CVPR 2011*, 2011.

[96] P.C. Teo and D.J. Heeger. Perceptual image distortion. In *Proc. ICIP 1994*, volume 2, 1994.

[97] M. Trentacoste, R. Mantiuk, and W. Heidrich. Blur-Aware Image Downsizing. In *Proc. Eurographics*, 2011.

[98] Remco C. Veltkamp and Michiel Hagedoorn. State of the art in shape matching. In Michael Lew, editor, *Principles of Visual Information Retrieval*, pages 87–119. Springer-Verlag, 2001.

[99] Baoyuan Wang, Yizhou Yu, Tien-Tsin Wong, Chun Chen, and Ying-Qing Xu. Data-driven image color theme enhancement. *Proc. SIGGRAPH 2010*, 2010.

[100] Baoyuan Wang, Yizhou Yu, and Ying-Qing Xu. Example-based image color and tone style enhancement. *Proc. SIGGRAPH 2011*, 2011.

[101] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Processing*, 13(4), 2004.

[102] Zhou Wang and Eero P Simoncelli. Local phase coherence and the perception of blur. *Adv. Neural Information Processing Systems (NIPS03)*, 16, 2003.

[103] O. Whyte, J. Sivic, A. Zisserman, and J. Ponce. Non-uniform deblurring for shaken images. In *Proc. CVPR 2010*, 2010.

[104] Kai Xu, Hanlin Zheng, Hao Zhang, Daniel Cohen-Or, Ligang Liu, and Yueshan Xiong. Photo-inspired model-driven 3D object modeling. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 30(4):80:1–80:10, July 2011.

[105] Li Xu and Jiaya Jia. Two-phase kernel estimation for robust motion deblurring. In *Proc. ECCV 2010*, 2010.

[106] Su Xue, Aseem Agarwala, Julie Dorsey, and Holly Rushmeier. Understanding and improving the realism of image composites. *ACM Trans. Graphics*, 31(4), 2012.

[107] Peng Ye, Jayant Kumar, Le Kang, and David Doermann. Unsupervised feature learning framework for no-reference image quality assessment. In *Proc. CVPR 2012*, 2012.

[108] Lu Yuan, Jian Sun, Long Quan, and Heung-Yeung Shum. Image deblurring with blurred/noisy image pairs. *ACM Trans. Graphics*, 26(3), 2007.

[109] Xiang Zhu and P. Milanfar. A no-reference sharpness metric sensitive to blur and noise. In *Quality of Multimedia Experience 2009*, 2009.

[110] Xiang Zhu and P. Milanfar. Automatic parameter selection for denoising algorithms using a no-reference measure of image content. *IEEE Trans. Image Processing*, 19(12), 2010.

[111] C. Lawrence Zitnick. Handwriting beautification using token means. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 32(4):53:1–53:8, July 2013.

[112] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *Proc. ICCV 2011*, 2011.